MASTER THESIS

OPTICAL FAULT INJECTION ATTACKS IN SMART CARD CHIPS AND AN EVALUATION OF COUNTERMEASURES AGAINST THEM

Nikolaos Athanasios Anagnostopoulos

FACULTY OF ELECTRICAL ENGINEERING, MATHEMATICS AND COMPUTER SCIENCE
DEPARTMENT OF COMPUTER SCIENCE (EEMCS)

CHAIR OF SERVICES, CYBERSECURITY AND SAFETY (SCS)

prof. dr. F. E. Kargl
prof. dr. ir. R. N. J. Veldhuis
ir. J. T. M. H. Dielissen

DOCUMENT NUMBER
EEMCS - 66014

UNIVERSITY OF TWENTE.

25 September 2014

Nikolaos Athanasios Anagnostopoulos
Student number: 1318055

Optical fault injection attacks in smart card chips and an evaluation of countermeasures against them

Master Thesis

Master Computer Science – EIT

Security & Privacy – specialisation Network Security

University of Twente

EIT ICT Labs Master School

2013-14

Graduation advisors:

Prof. Dr. F. E. Kargl (University of Twente)
Prof. Dr. Ir. R. N. J. Veldhuis (University of Twente)
Dr. J. Y. Petit (University of Twente)
Drs. J. Schut (University of Twente)
Ir. C. Groot (NXP)
Ir. J. T. M. H. Dielissen (NXP)
Ir. H. de Jong (NXP)
Dr. A. Jansman (NXP)

Graduation Committee:

Prof. Dr. F. E. Kargl (University of Twente)
Prof. Dr. Ir. R. N. J. Veldhuis (University of Twente)
Ir. J. T. M. H. Dielissen (NXP)

Acknowledgements

I would like to thank all those people who accepted me, put up with me and helped me both in my life and, in finishing my master studies and this thesis, including my parents, my friends, as well as my professors, colleagues and supervisors.

"Life is unfair, but you gotta make the best of whatever you have..."

– from "Malcolm in the Middle"

Την φανέλα μου φορώ με τον κίτρινο θεό...

Table of Contents

Abstract

This document is a study of optical fault injection attacks and countermeasures against them, with particular emphasis on an evaluation and comprehensive comparison of the different methods, techniques, structures and implementations which can be used as countermeasures.

Initially, it is examined why this topic is significant, important and relevant for computer security, especially from the perspective of a relevant business firm, especially regarding financial issues, such as cost. Then we make a comprehensive and detailed introduction to optical fault injection attacks against smart card chips and the countermeasures employed against such attacks, including a review of the current solution of using light sensors as countermeasures and the potential shortcomings of this solution.

We continue by identifying and investigating other potential alternative countermeasures against optical fault induction attacks. We then proceed to make a full evaluation and comparison of them, including some of their different implementations, regarding the balance between the diverse costs required for their implementation in a smart card chip and the level of protection they would offer. Additionally, we also relate the results of this comparison to the results that the current solution provides.

We then go on to discuss what future work we have identified that it may be needed regarding this field. Different innovative approaches are discussed and the thesis concludes with some additional remarks regarding these proposals and some final conclusions drawn from the comparison between currently used and alternative countermeasures.

Keywords: optical, fault, injection, laser, countermeasures, evaluation, security, cost, risk

Chapter 1:
Introduction – The importance of security in a smart card chip

It is an undeniable fact that a large number of both online and offline transactions are conducted nowadays with the use of smart cards.[1] Most of these transactions make use of the security features found on the chips that these cards carry, such as encryption mechanisms and secret keys and codes. It is therefore crucial to protect these features from being compromised, in order to maintain the integrity, confidentiality and availability of the sensitive and/or confidential data stored in these smart cards.

Additionally, the increased usage of smart cards also introduces higher incentives for both culprits and researchers to discover and commence attacks against the integrated circuits of such cards, which also result in a raising demand for the integration of better and more efficient security features in such chips.

It is therefore, essential to assess how the addition of new security features to a chip may affect its development and production costs, and whether such costs may be justified by potential losses and damages that may otherwise occur. Moreover, apart from direct economic losses and costs, a hardware attack on the integrated circuits of smart cards can also cause serious harm to the relevant company's prestige and brand name.

Potential future clients will be quite unwilling to place their trust on a company that has failed to secure its current clients' private data and their online – or offline – transactions. Therefore, it seems extremely crucial for a company manufacturing chips for smart cards to place significant effort and resources on assuring their security, by strengthening any security measures that have already been implemented on them and developing further more adequate and efficient ways to reinforce their security.

It is also really important for manufacturers of chips for smart cards to continuously monitor the most recent developments in attacking such integrated circuits, because only in this way an adequate effort can be made towards providing sufficient countermeasures against the latest reported attacks. In general, because the development of a new chip usually takes at least a couple of years, it is inevitable that new ways of compromising the chip's security measures will be developed before the chip reaches the market.[2] As this new information cannot be integrated in a chip's architecture, design and features, while this is already being developed, without considerable new cost, the original design of such an integrated circuit must already implement a substantial degree of novelty regarding its security features, which may be able to counter future attacks.[2]

Categories of attacks and attackers

Attacks can be classified as invasive, semi-invasive or non-invasive depending on their degree of physical penetration of the chip.[3][4] Invasive attacks involve altering the chip's physical structure in some way, semi-invasive ones only entail removing the chip's packaging, while non-invasive ones do not physically alter the chip and its packaging.[4] Furthermore, attacks can be classified as active or passive.[5] Active attacks involve altering the chip's data, functions or structure, in order to cause leakage of data or damage the chip's availability or integrity, while passive attacks just observe the chip’s functions, data or structure, in order to identify possible leakages of secure or private data.[4] Finally, attacks may target not only the confidentiality of the data found in a secure chip, but also their integrity or availability.

Successful attacks on such chips may cause damages and losses of millions, or even billions, of euros.[1][6] It is, therefore, evident that there are high incentives both for culprits to develop such attacks and for secure chip manufacturers to try to counter them as effectively as possible. Apart from this, academic researchers also try to develop both theoretical and practical ways of attacking secure integrated circuits in an effort to push forward research on this field, while also gaining potential benefits regarding their careers.[3][2] Furthermore, intelligence agencies of different countries as well as competitors could have the motivation to stage an attack against integrated circuits that are used in secure transactions.[3] Finally, a certain category of people may try to successfully penetrate secure chips just to test their own skills and abilities.[2]

A novel attack: Optical fault injection

A large number of different ways to attack secure chips have been developed over the years, as well as adequate countermeasures against such known attacks.[3] However, constantly new more innovative ways of attacking such integrated circuits are being developed and, in such a fast pace, that current development must take into account potential future attacks and come up with novel ideas, ways and means of protecting future chips against not only currently known attacks, but also potential future ones.[2]

Such a recently identified category of attacks on secure chips is optical fault injection. Fault injection, or perturbation, is the process of altering the chip’s data, functions or structure in such a way as to cause faults, calculations being performed with false data values. Its aim is to circumvent, penetrate or disable the chip’s security.[4] Optical fault injection tries to achieve this through the use of optical means, light from lasers or other sources which will cause the injection of faults in integrated circuits. This attack was based on the previously observed effects of ionising radiation on semiconductors.[7] Ionising radiation can cause semiconductor transistors to be turned on, potentially causing faults and errors in the operation of the overall circuit.[8] This effect was replicated with the use of intense light and could potentially lead in the leakage of sensitive information.[7]

Current countermeasures

Several ways exist to deal with both ionising radiation and optical fault injection attacks.[8] However, the most commonly used countermeasures against optical fault injections are light detectors. Such light detectors can be light-sensitive semiconductor devices integrated in the rest of the circuit and developed on the surface of a silicon (Si) wafer along with the rest of the chip’s die. Nevertheless, this solution may not always provide an adequate level of security, because of the limitations of such devices regarding their area of detection and their detection thresholds. Moreover, these devices can be pretty large and this may add significant production costs, because more area, and thus more materials, will be needed if they are integrated in a chip.

Problem statement

In general, even though a novel attack method may currently be considered complex, expensive and time consuming, it will eventually become more accessible and less costly to perform. As this may slowly be happening with optical fault injection attacks as well, it is really important to examine the different ways of performing such attacks and whether there any potential shortcomings in the current countermeasure employed, photodetectors.

Even if the current countermeasure employed against optical fault injections is completely adequate against them, we should try to come up with additional countermeasures, in order to avoid future evolutions of optical fault injections attacks outperforming the current security incorporated in the final product, the smart card chip. Such solutions may include both physical and logical ways of increasing the robustness of the chip's circuit against optical fault injection attacks, as well as detecting such attack attempts. Consequently, we should, of course, compare the benefits and costs of the current solution employed, light detectors, to the ones related to the proposed new countermeasures, taking into account the level of security required.

Determining the level of security that is required for a smart card chip seems to be a complex process that is dependent on the manufacturing company, the client and, to a lesser extent, on the current certification and accreditation requirements. Therefore, it may not be possible to fully decide if a particular countermeasure provides enough protection at an efficient cost, and therefore may be considered an adequate one or not, as this decision has to be taken by the different stakeholders. Additionally, we cannot exactly determine the actions that should be programmed to be taken in the event of a successful detection of an attack, as these have to be determined between the client and the manufacturer.

Nevertheless, we can try to assess the relation between the costs and the benefits for the current solution and compare it to other proposed ways of detecting and/or protecting against optical fault injection attacks, in order to determine a relevant comparison and evaluation between the different options available. It would therefore be essential to define the different agents of cost identified in the production of a security component incorporated in a smart card chip, and, therefore, also in the production of the chip itself. In addition to this, we also need to examine the potential protection such a component may offer, in our case against optical fault injections.

Furthermore, we would, of course, need to take a look on how optical fault injection attacks may be enhanced in the future, in order to also be ready, in the future, to determine adequate countermeasures against such evolved attacks. However, as we cannot predict the future, we also cannot use the protection a countermeasure may be offering against a potential future evolved attack as an objective criterion regarding its merits. It would be up to the different stakeholders involved in the secure smart card market sector to determine the validity of such concerns over evolved attacks and thus, the need for adequate countermeasures against them.

Research questions

Having determined the problem that this thesis will address, we can now to define the following research questions based on this problem that we will try to answer:

- How do optical fault injections work? What is the physical mechanism behind them?

- What are the current countermeasures incorporated in smart card chips against optical fault injection attacks and how do they counter them?

- How adequate are such countermeasures and what are their potential shortcomings?

- Are there other countermeasures which could address these shortcomings?

- How do the different countermeasures compare to each other regarding cost and protection?

- Which factors may affect their effectiveness and cost efficiency?

- Can some countermeasures be considered optimal or potentially lead into (more) robust smart card chips against light attacks?

- Can we come up with a proposal for a better solution than the currently employed one?

Methodology

In order to answer these research questions, the following methodology was used. Initially, we conducted a literature study on both the relevant background of the topic and the topic itself, in order to gain knowledge on the different mechanisms, methods and facts concerning both optical faults injection attacks and countermeasures against them.

We then conducted a more in-depth study on the reported effectiveness of current countermeasures and on different proposed countermeasures, as well as coming up with a number of potential countermeasure ideas of our own. Our ideas mainly consisted of different photodetector structures and other mechanisms which would prevent light from reaching the sensitive parts of the chip or would at least detect either light itself or the faults induced by it.

Following this, we examined the proposed countermeasures for their effectiveness, probable cost and general feasibility to be implemented and integrated in secure smart card chips. We also designed and simulated some of the more efficient countermeasures proposed and examined in detail their potential protection level against optical fault injections in relation to their cost in terms of area and power consumed.

We then came up with a set of criteria in order to determine how general categories of countermeasures performed against each other and which of the designed solutions could be considered more optimal. Finally, we suggest what seems to be the most optimal solution and suggested different topics on which future research needs to be conducted.

The relation between cost and security

It is important to explain here why cost plays such a significant role in determining the efficiency of a solution in being actually implemented. It should be evident that any additional security feature integrated in a smart card chip will need to be extensively tested, as it regards applications which require a significantly high degree of security. Such a solution not only has to work efficiently, but also must be sufficiently robust itself against not only already known, but also potential future, attacks. Therefore, it requires an exhaustive amount of designing and testing both as a prototype, but also when manufactured in mass quantities.

Thus, although the integration of security measures in such chips is intentionally sought-after and inherently unavoidable,[9] it comes at a large cost. Inherent costs exist due to the research and development phase of such features, which include manpower and infrastructure costs, as well as costs for the design and implementation of their prototypes and, potentially, of the whole chip, as the chip may need to be significantly redesigned. Furthermore, significant costs may also occur during the mass production of the chip, as the new features may increase the chip's area and thus raise the cost of the materials required for its production.

Moreover, the addition of security features may also introduce performance costs, related to the chip's computation time, delay costs, as well as power costs, due to the increased complexity of the new integrated circuit and the need to also provide enough power for the additional features. Additionally, the integration of such novel security features can also imply a rise in packaging and marketing costs and surely involves a significant rise in costs for testing, as the new features require unit testing on their own, integration testing towards the rest of the chip and a full system and acceptance testing for the new chip as a whole.

All in all, the overall costs for the research and development phase may rise well into the region of hundreds of millions of euros.[10] And, although it has been noted that the cost of research and development for semiconductor products exceeds that of most other high technology industries,[11] the cost of a semiconductor fabrication plant is estimated to be at least a few billion euros,[12][13][14][15][16] which also makes the cost of using its services extremely high. It is for this reason that any changes required to be made in the manufacturing process also come at an extremely high cost.

Finally, it must be noted that the smart card security market sector is highly based on economies of scale and rather relies on incompatibilities to act as high barriers to enter it. Additionally, lock-in effects are being used by big companies in it in order to control a certain share of the market. Therefore, cost plays a significant role in every aspect of this market, as it can significantly affect the share of the market controlled.

Acceptable risk: a balance between cost and security

It is really critical to note here that a balance exists between the levels of cost and provided security, defined through a level of *acceptable* risk. As it was quite early recognised that hardware can never become invulnerable to each and every kind of attack,[17] a level of risk is now always considered acceptable regarding security. Therefore, given the relation between cost and security, by defining a level of acceptable risk, a level of adequate protection is also defined, which leads to trying to achieve the lowest level of cost possible while maintaining that set level of security. Additionally, the level of acceptable risk is also dependent on the costs of performing a successful attack. If such costs are high, then the level of acceptable risk can also be high.

Cost is a critical factor for attacks targeted at secure hardware, because they quite often require the use of advanced equipment in a very specific way and expert knowledge, not only in the general field, but also about the very specific hardware to come under attack. Also, such attacks can far less often be automated in comparison to software attacks, thus their yield over time is much smaller.
Moreover, as hardware attacks are more physical in their nature than software attacks, they often require a large number of integrated circuits of the same kind in order to be successful, which may generally be hard to obtain. In conclusion, this means that performing a hardware attack tends to have a high cost, which may not always be justified by its potential benefits and rewards for the attacker.

However, hardware attacks can cause damages or losses amounting to several millions of euros. Especially when chips integrated in bank cards are targeted, the losses may potentially be in the region of billions of euros.[1] It is therefore evident that depending on their uses and their potential clients, different smart card chips have very different levels of acceptable risk. Another important factor determining the level of acceptable risk is the number of smart cards that may be potentially compromised by a single attack. If a specific attack which is not uncommon can give the attacker access to a large number of different cards, then, the acceptable risk will obviously be particularly low.

For example, NXP's MIFARE brand of smart cards had recently come under increased attack in an effort to identify potential ways to overcome its security features and gain access on confidential information stored on the cards.[18][19][20][21][22][23][24][25][26] Researchers have quite often succeeded in compromising the security of these smart cards.[18][19][20][23][24][26] The MIFARE company claims to possess "a confirmed market share of 77% in public transport",[27] which is equivalent to at least 1 billion of its cards being used for secure online transactions related to transportation.[23] This means that if the techniques demonstrated by these researchers were to be employed by culprits, and assuming that around 1% of the cards used were to be exploited, this could potentially result in losses amounting to millions of euros in a very short period of time.

Additionally, as already mentioned before, as certain attacks become less expensive, easier to implement or more well-known, the risk associated with it rises significantly, thus causing the overall acceptable risk level to decrease, until adequate countermeasures against such an attack also become common and inexpensive. Nevertheless, a low level of acceptable risk can also act as an incentive for the research, development, implementation or integration of better countermeasures.

Essentially, the current aim of security is rather to make attacks economically infeasible than actually make them completely impossible, based on the level of acceptable risk.[3] In this way, a balance between cost and security is created. However, apart from the cost of actual damages or financial losses, security tries to also protect from damages the reputation, brand name and prestige related to the product and its manufacturer, owner or user. Therefore, the level of acceptable risk is also dependent on these matters.

Chapter 2: Background and related work – Optical fault injections and countermeasures against them

Optical fault injection

As already mentioned before, fault injection is the process of causing faults and errors in the operation of an integrated circuit. This can be achieved by manipulating data values, or by altering the chip's structure or causing it to generally malfunction, either in a particular point or in its overall function. Different ways and means of injecting faults in integrated circuits have been identified over the years. Some common ways include introducing voltage spikes and clock glitches, making the circuit operate under extreme temperatures, or using radiation, eddy currents or light.[28] However, it has been noted that with all of these ways, except for light, there is no way of controlling the location and the type of fault induced in the circuit.

Considering, in particular, fault injection with the use of light, this is usually performed by lasers, as their beam is quite focused and concentrated, while the spectrum of their wavelength is quite small and narrow. In this way, the effects of this method of fault injection can be isolated within a specified area and thus the manner in which faults are induced in the chip can be efficiently controlled.

Fault injection can be used for such purposes as retrieving secret information or bypassing secure execution.[29] For example, secret keys can be fetched or bypassed by (partial) key nulling, or instructions can be skipped in order to achieve the dumping of memory or to gain access to confidential data.[4] Furthermore, the results of induced faults can be used in differential fault analysis in order to extract secret or private data through mathematical analysis.

Optical fault injection is an active and semi-invasive attack on the integrated circuit,[4][7] as it actively intervenes in the chip's operation and the chip needs to be exposed for it to take place. Each chip has a front side, where transistors have been developed and metal layers placed, and a back side which consists of the chip's substrate. As the chip's front side is usually covered in epoxy, if this is not transparent, it has to be removed (a procedure known as decapsulation or decapping) for the attack to take place from that side.[4] Attacks performed from the back side of the chip usually involve depackaging and rebonding the chip, in such a way as to gain access on that side.[4][7]

Optical fault injection can be performed only with a strong source of light, such as a photo flash or a laser beam.[7] However, obviously, a photo flash would not be as accurate and strong as a laser beam, while it may even cost more to acquire an adequate photo flash than to buy a very cheap laser at retail, such as a laser pointer.[7] For these reasons, optical fault inductions tend to be carried out using lasers, while attacks with photo flashes and weak laser beams served mostly as initial proofs of concept. Furthermore, laser fault injection can be performed both from the front and the back side of the chip, but different light wavelengths are needed in order to penetrate each side.

Specifics of the physical mechanism behind optical fault injection attacks

As optical fault injection was based on the previously observed effects of ionising radiation on semiconductors,[7] it makes use of the fact that electromagnetic radiation can temporarily change the state or characteristics of silicon (Si) transistors, causing transient modifications of their functions, which can result in faults and errors in the operation of the overall circuit.[8]

Laser fault injection uses light produced by lasers to induce photo-currents[30] in the channel region of a transistor, trying to activate that region and cause current to flow through the transistor. Photons excite the electrons in that region of the transistor, therefore creating potential, which in the end results in an induced current between the source and the drain, and, thus, effectively turns the transistor on.[30][31]

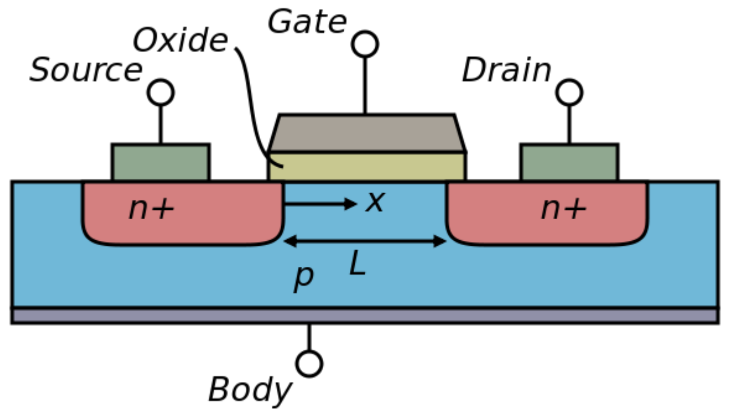


Image 1: Cross section of a lateral n-type MOSFET.[32] The channel is formed in the p-doped region of length L between the two n-doped regions, when current runs through it.

While n-type MOSFETs (Metal-Oxide-Semiconductor Field-Effect Transistors), also referred to as nMOS transistors (image 1), will be turned on, in p-type MOSFETs (pMOS transistors) usually only their threshold voltage will be lowered.[33] This happens because the activated region of nMOS transistors should have a majority of electrons, while the activated region of pMOS transistors a majority of holes.[34] Furthermore, high intensity light over a number of transistors may cause latch-ups; short circuits creating a low-impedance path inside the chip's substrate connecting the low and high power supply rails of a MOSFET circuit, thus trigerring a parasitic structure and disrupting the circuit's operation.[35]

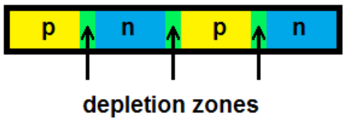


Image 2: Cross section of a lateral PNPN structure.

Such parasitic structures behave as PNPN structures (image 2), as they are essentially a pMOS and an nMOS transistor stacked together.[35] Their properties usually resemble those of thyristors, meaning that when one of the transistors is conducting, the other one also starts conducting.[35] The transistors both keep each other in saturation for as long as the structure is forward-biased and some current flows through it.[35]

Given that most chips are today implemented according to CMOS (Complementary Metal-Oxide-Semiconductor) technology, which basically uses complementary and symmetrical pairs of pMOS and nMOS transistors,[36] the above-mentioned difference in the effects of light on pMOS and nMOS transistors should not essentially make a big difference regarding the effects of light on contemporary integrated circuits, where whole specific areas made up by sets of multiple transistors, such as flip-flops, may be targeted.

Probable targets and technical characteristics of optical fault injection attacks

Flip-flops are probably the most targeted circuits of the whole chip, as they can store information by alternating (flipping) between two discrete stable states and are thus commonly used as memory cells.[37] Furthermore, their internal structure makes it quite easy to change the state of the overall circuit by changing the change of only one or two of their transistors.[7][29][37] Therefore, usually memory circuits and structures are most often targeted by optical fault injection, although other structures related, for example, to the logic of the chip could also be targeted.[38][39]

However, the design and structure of the basic logic circuits are more complex and differ significantly from those of the basic memory circuits. Therefore, it may be much more difficult to successfully make logic circuits perform a desired new operation, different than the one they are intended to perform. Instead, it is more probable that optical fault injection attacks will cause logic circuits to malfunction or cause permanent damage to them or the overall chip. Furthermore, it is improbable that this could be achieved by changing the state of only a few transistors. Nevertheless, faults could be induced optically in logic and, because of the overall structure of the chip, these may be propagated. In this way, after some experimentation, it may be possible to achieve the desired effect of disrupting the chip's operation in an intended and entirely predictable manner, without causing permanent damage.[3][38]

In general, silicon transistors are more vulnerable to optical fault injection with wavelengths between 400 and 700 nm. Wavelengths below 400 nm will be absorbed or reflected, before the photons reach the channel region of the semiconductor, while wavelengths above 700 nm are more likely to pass through the transistor without being absorbed.[40] Furthermore, silicon is transparent to (infrared) light with wavelengths above about 1100 nm.[41]

Technical characteristics of front side attacks

The front side of a chip usually consists of layers of metal and other shielding material, connecting and protecting the actual transistors which are buried below them and are mostly made of silicon with certain impurities, which enhance silicon as a semiconductor.[42] These intentionally implanted impurities (dopants) in the silicon forming the transistors may make them more vulnerable to light.[43] Depending on the number and kind of layers of metal and on whether the chip is shielded or not, different laser wavelengths may need to be used to successfully attack these transistors. Such laser wavelengths for front side fault injections may range from around 500 nm to around 800 nm, depending on the layers of metal and the shielding. For example, wavelengths of 523,[44][45] 532,[38][46][47][48] ~650,[7][39] 785,[49] 795[50] and ~800[4][51][52] nm have been used successfully in order to alter the operation of silicon transistors by being shed on their front sides.

Furthermore, unless the chip's covering and the epoxy adhesive used on its front cover are transparent, then these usually need to be removed before a successful front side attack.[4][49] Moreover, increases in the energy and the power of the laser being used will lead to further and/or more successful penetration of the front side of the chip. Depending on the materials of the front side of the side and the wavelength of the laser, some milliwatts may be sufficient for the emitted light to efficiently reach the transistors.[7][39] In other cases, hundreds of milliwatts, or even some watts, may be required.[45][49]

Technical characteristics of backside attacks

The back side of the chip is mainly composed of the substrate, the slice of silicon on which the transistors were built. In order to penetrate this material and reach the actual transistors, we have to use laser wavelengths for which silicon is semi-transparent, because otherwise most of the light will be absorb by the substrate before it reaches the transistors. These wavelengths are usually between 950[51] and 1100 nm,[41] while silicon is transparent for wavelengths above 1100 nm.[41][43][53]

However, not only wavelengths between 950 and 1000 nm have been successfully employed for backside fault injection,[51] but also lower wavelengths of around 800 nm have successfully caused fault injection in chips from their back side.[52] Additionally, after thinning of a chip's back side, laser fault injections with a wavelength of ~900 nm performed on its back side were successful,[54] while thinning the back side can generally provide better results for back side fault injection.[53][55] In general, however, wavelengths above 1000 nm are being used for backside fault injection attacks, with the wavelength of 1064 nm being the most commonly used,[4][53][56][57][58][59][60][61][62][63] along with some cases of a 1065 nm laser also having been successfully employed to induce faults through the back side of the chip.[39][64]

Furthermore, again, increasing the energy and the power of the laser being used leads to further and/or more successful penetration of the back side. Depending on the thickness of the back side and the wavelength of the laser, tens of milliwatts may be sufficient for the emitted light to efficiently reach the transistors.[39][53][57] In other cases, hundreds of milliwatts may be required.[64] Finally, accessing the back side of the chip may require its depackaging and rebonding.[4][7]

A comparison between frontside and backside attacks

Each side of the chip has different characteristics when a laser beam is shed upon it. The front side can provide a good visibility of the chip's layout, but accurate targeting from that side is difficult because of its multiple metal layers and potential shielding, which reflect most of the laser's light.[46][48] Additionally, as technology progresses, the number of metal interconnects on a chip grows, while its size reduces, which makes it even more difficult to reach the transistors of a chip from its front side.[46][48]

The back side of the chip does not provide as much visibility of the layout, so positioning is more difficult.[46][48] Furthermore, beams of higher wavelengths than the ones used for penetration of the front side are required in order for the laser light to successfully reach the chip's transistors through its back side.[46][48] However, because of the lack of metal layers and shielding on the back side, the laser beam is not significantly reflected or absorbed before it reaches the chip's transistors and, thus, this attack method is very efficient.[46][48]

Lasers as a means for optical fault injections

There is a number of reasons why light emitted from laser beams is quite an advantageous technique of inducing faults in an integrated circuit. Among other things, lasers are nowadays pretty cheap and easy to access compared with other material that could induce faults using radiation, such as for example X-ray or gamma-ray emitters. Furthermore, compared with other electromagnetic means, lasers are further more accurate in terms of location than the majority of them, being able to focus on very specific parts of the chip with an accuracy of at most some micrometers, while also being really accurate in terms of timing, as it is possible to select precisely the moment when a laser beam will hit a transistor and for how long this will happen, with an accuracy of nanoseconds.[29] Finally, the faults generated by lasers can be made to last only for the period of time during which the transistor is being hit by the laser beam, in order to cause only temporal faults.[29] In this way, the chip will be completely functional after this period, if some mechanism disabling it after a number of attacks has not been implemented into it, and therefore, the attack will be completely reproducible to its very detail.[29]

However, the laser beam has to be focused on the exact transistor or component that should be attacked, thus the layout of the specific integrated circuit or its relevant region must be known or found out. In addition to this, the time at which the laser beam should hit a transistor and the duration of this must also be specified quite precisely, sometimes with an accuracy of nanoseconds or picoseconds, depending on the chip's internal or external clock and the operation of the overall circuit.[4] Finally, other characteristics of the chip's material, such as silicon's refractive index, must be taken into account when planning an optical fault injection attack. Silicon has a refractive index between 5.6 and 3.5 for wavelengths between 400 and 1100 nm.[65]

Choosing a laser with appropriate wavelength and power to perform an attack is not always easy, as the characteristics of the chip's materials and its layout must all be taken into account. However, as we know the range of wavelengths that are suitable for performing both frontside and backside fault injection attacks with light, we can also pick a laser within those ranges. Additionally, as chips are getting smaller, less energy will be needed to achieve the same results.[7] So far, Nd:YAG (neodymium-doped yttrium aluminum garnet; Nd:$Y_3Al_5O_{12}$) lasers which produce a wavelength of 1064 nm[58][59][60][63][66] seem to be preferred, as their frequency can be doubled to produce a wavelength of 532 nm,[48][66] thus being able to perform both frontside and backside attacks. Other lasers with such characteristics would include Neodymium-doped yttrium orthovanadate (Nd:YVO_4) lasers which also emit a wavelength of 1064 nm[67] and Neodymium-doped yttrium lithium fluoride (Nd:YLF) lasers with a wavelength of 1047 nm,[68] which can provide a wavelength of ~523 nm with frequency doubling.[44][45][69] Furthermore, Ti:sapphire lasers (also known as Ti:Al2O3 lasers, titanium-sapphire lasers, or simply Ti:sapphs) are tunable lasers which emit in the range from 650 to 1100 nanometers[70] and have also been used for both frontside and backside laser fault injection attacks.[51] Finally, as already mentioned before, diode lasers of 1065 nm have also been successful in causing optical fault injection in transistors.[39][64]

Continuous wave and pulsed lasers as means for optical fault injections

Moreover, continuous wave or pulsed lasers can be used, i.e. lasers which emit a continuous light beam or lasers which emit a beam in the form of pulses of some duration at some repetition rate.[71] It has been identified, however, that the continuous wave laser operation is not really appropriate for precisely localised fault injections, as its collateral energy will effectively stimulate the whole region surrounding the spot that the beam hits, all the time that the laser is on, thus potentially creating unintended faults.[56] Therefore, it is far more effective to use pulsed lasers, as these can be switched on for a very short time period, when it is required to do so, and thus, minimise any collateral effects to negligible quantities and assuring resolution to the intended spot size.[56]

More specifically, pulsed lasers with a pulse duration of some nanoseconds have been widely employed for laser fault injection attacks.[38][45][54][61][62][72][73] However, also continuous wave lasers[53][56][58] or pulsed lasers with a much lower pulse duration of some picoseconds,[51][52][57][60][63] or even femtoseconds[50][51] can induce transient faults. The right pulse duration to be used strongly depends on the circuit's clock frequency, while the laser beam must be triggered at the right point of time and last the right amount of time, in order to induce a fault at the right point of time during the circuit's operation.[4] Therefore, some particular event in the operation of the circuit or some signal must be identified to serve as a trigger.[4]

Two-photon absorption in silicon and its significance as a future attack mechanism

Finally, silicon exhibits two-photon absorption, a process in which two photons are absorbed to excite a single electron.[74][75][76][77][78] The electron is excited from the valance band to the conduction band, resulting in the generation of an electron-hole pair.[74][77][78] However, it does not seem to exhibit higher multi-photon absorption.[79] This property of silicon has led into new forms of laser fault injection attacks, which employ lasers of higher wavelengths, which produce photons of lower energy, and which can be used to induce photo-currents through two-photon absorption in silicon transistors.

As the following formulas apply, regarding a laser beam's energy and wavelength:

$$f=\frac{v}{\lambda}$$

$$f=\frac{E}{h}$$

$$E=\frac{h*v}{\lambda}$$

$$h\approx 6.62606896*10^{-34} J*s\approx 4.13566733*10^{-15} eV*s,$$

where E is energy, f is frequency, v is velocity (speed) and h is Planck’s constant, it is easy to understand that 2 photons of 1340 nm will have the energy of a single 670 nm photon. Of course, this is dependent on the medium through which they move and other conditions, such as temperature, which may affect their speeds. However, in general they cause the same electron excitations on silicon that a photon of around 650-700 nm would.

So far, lasers with wavelengths of 1280[80][81] and 1340[53] nm have been successfully utilised to inject faults in transistors, taking advantage of the two-photon absorption property of silicon. Both lasers beams had an average power of tens of milliwatts, but while the 1340 nm laser was a continuous wave one,[53] the 1280 nm one was emitting pulses of light with a duration of 200 femtoseconds.[80][81] Such lasers exhibit an extremely precise location accuracy and may be harder to detect as silicon is transparent for these wavelengths.

The importance of the laser spot size in optical fault injections

In general, because the feature size of the transistors of chips used in smart cards is already quite small and is going to get smaller in the future, there is also an effort to keep the beam of any laser used in optical fault injection attacks as narrow and thin as possible, in order to be able to attack single transistors. For this reason, the laser beam is usually focused with the help of appropriate lenses in order to reach a spot size (beam width or diameter) of some micrometers.

More specifically, lasers used successfully for frontside or backside attacks tend to have a spot size of a few micrometers,[4][7][29][39][46][47][48][51][60][61][62][76][82] which means that for manufacturing processes with feature sizes smaller than 120 nm, the laser beam may unintentionally excite several adjacent transistors.[48][51] This happens because light cannot be focused to a perfect point due to the effects of diffraction.[48][83][84] However, depending on the circuit that is targeted even larger laser spot sizes of tens or hundreds of micrometers can be be employed successfully to induce faults to the chip, especially when logic circuits are targeted, since most logic gates take up an actual area of 1 μm or larger.[38][54][69][85]

Finally, lasers used in attacks taking advantage of two photon absorption can have significantly better spatial coherence,[53] causing them to exhibit a smaller spot size of a few hundreds of nanometers.[80][81] This could prove really important as current smart cards employ manufacturing processes of 90 nm or below. Furthermore, the laser spot size used in an attack is also related with the intended amplitude distribution and the sensitivity and structural topology of the circuits to be attacked.

Additionally, it has been noted that wavelengths of 1500 to 1600 nm,[78] or 1550 nm in particular,[74][86] may have the same effect on silicon and relevant transistors. However, apart from the particular light wavelengths that silicon may be able to absorb, other factors such as potentially overheating the chip’s transistors[53] or otherwise damaging them may also be limiting the light spectrum range that can be used in optical fault injection attacks. Nevertheless, using laser beams at an angle[3] or taking advantage of the diffraction of a laser or of the interference between two laser beams of the same or different wavelengths may provide ways for more novel successful attacks.

Light detectors as a countermeasure against optical fault injections

Optical fault injection attacks may be countered by technology hardening, or be addressed by countermeasure mechanisms and structures implemented either at the system level or at the circuit level.[82][87] Countermeasures may include the use of extra layers of protection such as various shields,[39][82][88] trying to reduce charge collection at sensitive nodes by using different materials and techniques for the fabrication of transistors, using extra circuits for error detection and correction (EDAC), or constructing specific structures, such as light sensors and photodetectors, which can detect and effectively counter optical fault injections.[3][4][39][82][87][89]

In general, there is a large number of different countermeasures that can be used against optical fault injection attacks and/or other types of fault injection attacks.[3][82][87][89] However, photodetectors, in particular, can play a significant role in detecting attempts to reach and tamper with the chip's transistors by using light, and can therefore be also employed to effectively prevent such attacks from being successful.[3] For this reason, chips manufactured for secure transactions, such as those found on secure smart cards, may often include photodetectors in their internal structure as a countermeasure against optical fault injection attacks.

Incompatibilities as a restraining factor

However, even though various different structures and mechanisms can be used to detect the presence of light,[90] many of these mechanisms are too large to be implemented within such integrated circuit structures as today's smart card chips[29] and/or can only detect a quite specific spectrum of light wavelengths and energies. Therefore, it is essential to examine which structures and mechanisms can act as photosensors in silicon integrated circuits currently produced for smart cards, which employ manufacturing processes of 90 nm or below and therefore adhere to their manufacturing and fabrication rules and limitations.

It is obvious that semiconductor structures which can act as photodetectors are inherently well-suited for this purpose, while mechanisms based on completely different materials cannot easily be incorporated in the manufacturing process of integrated circuits. Furthermore, it should be also evident that such structures as active-pixel sensors (APSs) and charge-coupled devices (CCDs), which are a whole integrated circuit, system or device on their own, cannot also be integrated in such other chips as integrated circuits for smart cards, due to their size.

Moreover, other devices such as particle detectors, or detectors which are based on chemical or thermal properties, or which operate in specific extreme conditions are effectively excluded from acting as photodetectors in integrated circuits, because they are either too big, cannot be used repeatedly, or their operation will be affected by the heat the integrated circuit emits on its own during its normal operation, or by other properties of the chip. However, devices which are completely vulnerable to light exposure can still be used as one-time detectors, in order to disable, or completely destroy, the chip after an attack has been detected. This way, although the chip will be rendered useless, it will at least be possible to protect confidential information from being compromised.

Compatible devices

Structures and mechanisms which can act as photodetectors and be successfully incorporated near vulnerable elements inside integrated circuits used for smart cards include photodiodes, phototransistors and photoresistors. These structures are basically common diodes and transistors built by such materials and in such ways that they are more sensitive to light, which causes current to flow through them, or in the case of photoresistors, the resistance of the semiconductor to be lowered. They make use of the photovoltaic, photoelectric and photoconductive effects and can be built in minuscule sizes, as quantum dots, thus also taking advantage of quantum mechanical properties.

Diodes

Furthermore, Light-Emitting Diodes (LEDs) can also be used as photodetectors, if reverse-biased to act as photodiodes, while the whole integrated circuit can be constructed in such a way as to act like a photovoltaic cell (a.k.a. a solar cell), especially as a quantum dot solar cell. Other structures, based on vacuum tubes, such as photomultipliers and phototubes, even though they would be really effective, are too big to be incorporated in a contemporary integrated circuit. Finally, photodiodes which are highly reverse-biased can take advantage of the avalanche breakdown effect, which allows the photocurrent to be significantly multiplied within the photodiode.

Photodiodes are diodes which can act as photodetectors and generate a current or voltage when their PN junction is illuminated.[91][92][93][94] If an external reverse bias is applied to them, they operate in a photoconductive mode, while if no bias is applied to them, they operate in a photovoltaic mode.[92][93][94][95] In the photovoltaic mode, a photodiode's dark current, the current flowing through the device when it is not illuminated, is kept at a minimum.[92][93][94] Even though the speed of response is much faster in the photoconductive mode, the dark current is also increased, without an equal increase in the produced photocurrent.[92][93][94][95]

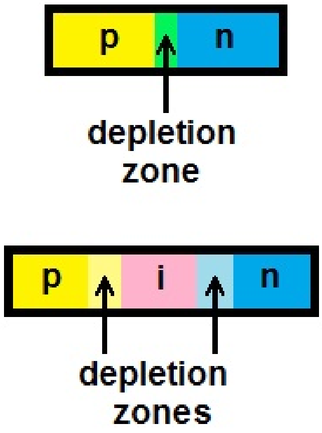


Image 3: Cross section of a lateral diode with a PN junction (p-n diode) and a lateral diode with a PIN junction (p-i-n diode).

Furthermore, diodes designed to act specifically as photodiodes may use a PIN junction, rather than a p-n junction (image 3), to increase the speed of response and also have a higher detection bandwidth.[93][95] A PIN (or p-i-n) diode contains an intrinsic (i.e. undoped) semiconductor region between the n-doped and p-doped regions of its junction,[96][97] thus having a thicker depletion region. Moreover, photodiodes can be built in such manner as to take advantage of the avalance breakdown effect by being highly reverse-biased.[91][93][97] This causes avalance photodiodes to have a significant internal gain and multiply the induced photocurrents. Avalance photodiodes combine high speed of response with high sensitivity and can be used as silicon photomultipliers.[91][98][99] However, they require a high reverse voltage and their gain (multiplication ratio) is dependent on temperature.[91][99]

Moreover, Single Photon Avalance (photo)Diodes (SPADs) can detect low-intensity light signals down to single photons.[91][99][100][101] They are avalance photodiodes that can provide a much higher gain than normal ones, by being reverse biased with a voltage set higher than their breakdown voltage,[91][99][100][101] the voltage at which the breakdown occurs and an avalance of electrons and/or holes takes place.[102] This regime of operation is called the “Geiger mode”[91][99][101] due to its analogy to the Geiger-Müller counter.[100] Silicon SPADs can be fabricated with standard CMOS technology, [101] but their dark current must be minimal.[100] However, due to the high voltage used in the reverse biasing of these diodes, their lifetime may decrease significantly, thus making them inappropriate for use in smart card chips that need to have an extended lifetime.

Material	Electromagnetic spectrum wavelength range (in nm)
Silicon	190 – 1100
Indium gallium arsenide	800 – 2600
Germanium	400 – 1700

Table 1: Relation between photodiode material and electromagnetic spectrum wavelength range of detection.[93]

The spectrum in which a photodiode can sufficiently detect photons is dependent on thematerial that is made of.[93][94][98] Obviously photodiodes made from silicon can detect light in the wavelengths that would excite silicon transistors and flip their state causing a fault, exactly because they are both made pretty much from the same materials (the dopants used in each of them may or may not differ). However, other materials, such as germanium (Ge), indium gallium arsenide (InGaAs), can also detect photons in the electromagnetic wavelength range used in optical fault injection attacks against silicon transistors (Table 1).[93][94][98]

Transistors

Phototransistors are transistors that are more sensitive to light than usual ones.[93] They are usually NPN bipolar transistors[103] with their reverse-biased base-collector junction being exposed to light, though transparent materials.[93][104][105][106] The photocurrent that is generated in the base-collector junction is injected into the base and amplified by the transistor’s current gain.[93] Phototransistors may or may not have a base lead, which would allow their light response to be biased.[103][106] In order to increase their gain even more, they can be more highly reverse-biased, like photodiodes.

Again, the same materials as those used for photodiodes can be used to create a phototransistor, having the same qualities of detection, while there might be a dark current flowing through phototransistors, too.[104][106] The difference between photo transistors and photodiodes is that phototransistors provide much higher gain than the photodiodes, allowing more current to flow when excited by light, and thus also having higher sensivity to light, although they have much slower response times than photodiodes.[93][103][104] If their emitter is left unconnected, they become photodiodes.[93]

Finally, field-effect phototransistors (photoFETs) also exist, being more light-sensitive field-effect transistors (FETs).[93] Unlike bipolar phototransistors, however, they control the drain-source current by creating a gate voltage.[93] Furthermore, there are Darlington phototransistors, which use the standard Darlington transistor configuration with the emitter of the input transistor being connected to the base of the output one and then both their collectors being connected together.[107] The gain of the Darlington transistor pair, which can be used as a single transistor, is the multiple of the gains of the two individual transistors.[107] In the Darling phototransistor (photoDarlington) configuration, the input transistor acts as a photodetector, being a phototransistor.[107] This provides a much higher gain, but the pair is also much slower than an ordinary phototransistor.[107][108]

One of the differences between the Darlington transistor and a normal one is that the Darling transistor has a higher voltage difference between its overall base and emitter, i.e. between the base of the input transistor and the emitter of the output one.[107] This is also true for the Darlington phototransistor when its base is biased.[107]

Resistors

Furthermore, photoresistors, also known as Light-Dependent Resistors (LDRs) or photocells, are light-controlled variable resistors, made of high resistance semiconductors, whose resistance depends on light.[109][110] Such materials have a high resistance in the dark, while this decreases in the presence of light.[109][110][111][112] If the incident light exceeds a certain frequency, photons are absorbed by the semiconductor, producing free electron-hole pairs, which conduct electricity and reduce its resistance.[109][111][112] The more intense the light falling on the photoresistor, the more its resistance will decrease.[110][111]

Photoresistors have a lower sensitivity and higher response times to light changes than phototransistors and photodiodes.[111] Photoresistors also lack a PN junction and act as passive components, in contrast to phototransistors and photodiodes.[111] Furthermore, they exhibit considerable time latency between changes in illumination and changes in their resistance and they may be also sensitive to temperature changes.[111] As they are based on their material’s photoconductivity, they may also be called photoconductors.[113]

The most common material used in photoresistors is cadmium sulfide (CdS),[109][111][112] with dopants which may lower the conduction band level.[112] Furthermore, cadmium selenide (CdSe), lead sulfide (PbS), lead selenide (PbSe) and indium antimonide (InSb) are currently used as photoresistors, while also silicon (Si) and germanium (Ge) can also be used as photoresistor materials, especially in components such as Germanium:Copper (Ge:Cu) photoresistors.[109][111][112] The resistance range and sensitivity differ between different materials, with cadmium sulfide and cadmium selenide photoresistors being sensitive in the visible spectral region and lead sulfide, lead selenide and indium antimonide in the infrared, while silicon and germanium are sensitive both in the visible and the infrared spectral regions.[109][111][112][114]

The use of cadmium and lead is severely restricted in the European Union (EU), based on the EU Restriction of Hazardous Substances Directive, and thus the use photoresistors made of lead or cadmium is really limited in the EU.[109][111] However, as cadmium sulfide photoresistors are inexpensive and their use has been widespread, the term ‘CdS cell’ is largely considered synonyms with the term ‘photoresistor’.[112]

A comparison of the compatible photodetectors and related potential improvements

It is obvious that different solutions and implementations exist regarding the categories of circuits that can be used as photodetectors in a smart card chip. It is also evident that photodiodes and phototransistors have a few advantages over photoresistors, as they are more sensitive and have a faster response time, with phototransistors being more sensitive, but also having a slower response time than common photodiodes. Furthermore, photodiodes can be reverse-biased by such a high voltage as to be able to detect single photons, while phototransistors in general have a much higher gain than photodiodes.

Moreover, different materials can be used to make a photodetector, such as silicon, germanium or others. It is also quite common for such photodetectors to be incorporated in other structures, including integrated circuits, therefore, their compatibility with chips has already been demonstrated in the past.[115][116] Furthermore, these photodetectors can even be fabricated in miniscule dimensions as nanostructures,[117][118] thus their size poses no challenge for integration in the CMOS manufacturing process used for smart card chips today.

Furthermore, it has been noted that common photodiodes have low noise, can be fairly inexpensive, are very efficient and can have a long lifetime.[93] In the future more efficient materials may be used as photodetectors, such as graphene, which has a very wide wavelength range of detection in both the visible and the infrared regions of the electromagnetic spectrum.[119][120] Although graphene, which is a single layer of carbon atoms assembled in a honeycomb lattice,[119] has a very fast response time, it has low sensitivity and no internal gain mechanism of its own.[119][120][121] However, this tends to be improved,[119][120][121][122] especially as graphene is being integrated in the structure of silicon phototransistors or photodiodes, which can provide a high gain mechanism and improved sensitivity.[120][121]

As it has already been demonstrated that graphene photodetectors can be successfully integrated in the CMOS manufacturing process,[120][121][122] it has been suggested that graphene-based integrated electronic-photonic circuits with a wavelength range from 300 nm to beyond 6 μm can be expected in the future.[119] Apart from being compatible with CMOS and other relevant technologies, other advantages of graphene are its low cost and the low level of energy required for it to be fully functional in any structure.[120][121]

The cost of photodetectors in relation to the protection they provide

Of course, there is a need not only to detect an optical fault injection attack, but also prevent such an attack from being successful. To this end, apart from photodetectors, there have to be some dedicated circuits which measure the electrical output of photodetectors and decide whether there has been an optical fault induction attack and which perform some preventive action when such an attack is detected.

It is evident therefore that these circuits need protection from being tampered with, while the whole integrated circuit that incorporates such security circuits needs to be optimised. For this reason, it may be efficient for several photodetectors to share their measurement and response circuits. As these circuits take up an additional area on the chip, they may significantly raise its production costs. However, these issues cannot be easily quantified, because they are strongly dependent on the exact implementation and placement choices made regarding the integration of photodetectors in the chip. Depending on how efficiently this integration is performed, it may or may not be possible to combine or share some of their auxiliary components, which could significantly affect their actual production cost.

Nevertheless, it is true that perfect security does not exist and a balance has to always be maintained between the level of security that a feature provides and its costs, while also taking into account the level of risks and deciding which level of security is really required for a specific chip.[4] Therefore, and as security is defined as a state which is not completely devoid of risk, but which is free of *unacceptable* risk,[123] smart card chips have not only to be protected against the risk of potential compromise, but also be cost-effective, perform well and fail rarely, i.e. be reliable and efficient.[4]

It has therefore been suggested that while photosensors may become really effective in detecting optical attacks and thus preventing them from being successful, it is impossible to really protect each and every transistor and thus an attacker who has found the right position on the chip to attack may still succeed if the laser spot size on the chip is small enough to not trigger the nearest sensor.[4]

For this reason, we will examine the disadvantages, shortcomings and problems of using photodetectors on an integrated circuit and try to come up with potential alternatives, which may not have to replace the use of photodetectors, but can be combined with them or act complimentary towards protecting the chip from optical fault injection attacks.

Potential problems regarding the operation of light detectors

Photodetectors may have certain limitations regarding their use as a countermeasure against optical fault injection attacks. As already mentioned before, they exhibit certain shortcomings even in their primary function of efficiently detecting the presence of light. In general, their characteristics must be optimised and balance their sensitivity, response time, internal gain and latency, taking into account the associated costs and intended security levels. Again, there is no perfect solution which would maximise their sensitivity and gain while minimising their response time and latency and a compromise must always be made among these characteristics.

As these structures and circuits need to be extremely small in order to fit in miniscule integrated circuits, they also suffer from this trait of theirs, because in order for them to be produced in a small scale, their detection area is also respectively reduced. Furthermore, there are other factors which may affect the effectiveness of using photosensors as a countermeasure against optical attacks such as the threshold voltage needed to signify an attack taking place, which if set too low will cause an unacceptable number of failures and if set too high will fail to detect successful optical injection attempts.

Moreover, photosensors are not always effective on detecting light attacks, as these may happen outside their effective area of detection, or hit the detection area from an inconvenient angle or side. It is important to note here that thinning the material of the chip that the laser beam has to go through before it reaches the chip’s transistors can significantly affect the operative area of detection of a photodetector, as the beam will be more focused and there will be less reflection, diffraction and scattering of the beam’s photons and thus the detector may not be triggered.

Additionally, refraction, diffraction and interference can be used to guide an attacking beam away from photodetectors. Furthermore, silicon’s refractive index increases with optical intensity, because of the electrons and holes that are excited and released, thus exhibiting a Kerr focusing non-linearity.[78][124] This can cause further issues regarding both the location where photodetectors should be placed to be effective and the characteristics that an attack should have to be successful, such as the beam’s angle, intensity and targeted location.

Another important issue regarding photodetectors is that their material(s) may not be equally sensitive along their surface and their reaction may differ depending on the wavelength and angle of the incident photons. Furthermore, design incompatibilities and costs related with the integration of photosensors in the integrated circuits may hinder further their adoption as efficient countermeasures against optical fault injection attacks.

Finally, it is important to note again that since no countermeasure can provide perfect security, it is more reasonable to use a combination of different methods, structures and techniques, both in hardware and software, to protect a chip against any particular kind of attack, than a single type of structure, such as photodetectors.[3] Additionally, despite their potential shortcomings, photosensors can always be improved and enhanced through further development and research, exploring novel structures, approaches and materials. It is therefore essential to examine which alternative solutions can adequately augment photodetectors in providing sufficient security against optical fault injection attacks in smart cards chips. Such protection must not only include the detection of such attacks, but also ways of neutralising such attempts, for example, by disabling the chip for some time period or locking it indefinitely.

Examining possible alternatives

As security is the essential characteristic of smart cards,[3] it is extremely important for smart card chips to be adequately protected against all kinds of compromising attacks, including, but not limited to, optical fault injection attacks. For this reason, an extensive number of countermeasures against potential attacks, such as photodetectors, has been developed and integrated in chips for secure transactions. However, since we have already identified that these countermeasures have certain shortcomings and their abilities are not unlimited, it is crucial not only to examine potential alternatives, but to also compare their characteristics, their benefits and costs and subsequently evaluate and classify them accordingly.

Optical fault injection was initially developed to simulate the results of ionising radiation on integrated circuits.[7] Therefore, optical fault injection attacks can, in general, be countered by known ways and means used for making computer chips resistant to damage, malfunctions or errors caused by ionising radiation. Radiation hardening, the act of making integrated circuits resistant to radiation,[8] can be used to successfully counter most types of radiation, i.e. most wavelengths in the electromagnetic spectrum, as well as different types of subatomic particles. While it is obvious that simple measures cannot be used to protect adequately against prolonged exposure or in extreme conditions, they may be able to ensure that the chip is rather destroyed than actually compromised.

Radiation-hardened chips are based on their non-hardened equivalents, with some design and manufacturing changes which make them less vulnerable to radiation.[8] However, radiation-hardened chips, like all secure chips,[2] may lag behind the most recent developments as they require a prolonged time for extensive development and testing.[8] Therefore, when they are designed they should incorporate novel approaches and countermeasures in order to try to compensate for this shortcoming.[2]

Radiation hardening techniques can be classified in two large groups according to the level on which they apply, physical or logical. Physical radiation hardening concerns the materials used to fabricate the chip, while logical hardening is about the logic employed in both hardware and software. In general, physical techniques focus on materials which make the chip impenetrable to radiation or which are less affected by radiation, while logical hardening focuses on duplicating structures or implementing accuracy controls and error detection and correction mechanisms.[8]

Physical alternative countermeasures

Insulators

More specifically, regarding physical radiation hardening, this could be achieved by using hardened chips manufactured on insulating substrates or by shielding the chip or its package against radiation.[8][39][72][82][88] The transistors of chips can be formed on top of a layered silicon-insulator-silicon substrate in place of conventional silicon substrates, in a manufacturing process referred to as “silicon on insulator” (SOI).[87][125] In this way, parasitic capacitance is reduced and performance is improved,[125][126] while also light attacks are made much more difficult.

Usually silicon dioxide (SiO_2), which has a much lower refractive index than silicon and is transparent at the visible wavelengths,[127] is used as the insulator. Specifically, if the silicon dioxide is amorphous, then its refractive index is between 1.47 and 1.45 for wavelengths between 400 and 1100 nm,[65][128] while if it is crystalline, its refractive index is between 1.56 and 1.53 for the same wavelengths.[65] Since silicon, as already mentioned before, has a refractive index between 5.6 and 3.5 for wavelengths between 400 and 1100 nm,[65] this can cause light coming through the back side even at low angles to undergo total internal reflection between the bottom layer of silicon and the layer of silicon dioxide and to remain on the bottom silicon layer of the wafer.[124][125]

Additionally, even if light does go through the silicon dioxide, this will be significantly refracted when it reaches the top layer of silicon. Furthermore, it will be far more difficult to induce photocurrents in the transistors of such a chip penetrating through its substrate and almost impossible to produce latch-ups.[125][126] However, for these exact reasons, if photons do reach the right transistors of a silicon-on-insulator chip and do induce photocurrents, the detection of such a successful attack may be almost impossible using only photodetectors. Finally, an attacker could remove the bottom layer of silicon and reach the transparent silicon dioxide by grinding the chip's backside, thus making attacks a lot easier.

However, other better insulators can also be used to produce a silicon on insulator chip, such as zirconium dioxide (ZrO_2) and sapphire (aluminium oxide, Al_2O_3).[129][130] The manufacturing process for sapphire substrates is called "silicon on sapphire" and a thin, usually thinner than 600 nm, layer of silicon is grown on a sapphire wafer.[129] The sapphire wafer is high-purity and artificially grown and it is an excellent electrical insulator which prevents stray currents from spreading to nearby circuit elements.[129]

Nevertheless, the synthetic sapphire wafer has no bottom silicon layer and is highly transparent to wavelengths between 150 and 5500 nm,[129][131][132] which means that the chip's transistors are fully exposed from the back side. However, since the sapphire's refractive index is between 1.79 and 1.74 for wavelengths between 400 and 1100 nm,[65] again even if light goes through it, this will be significantly refracted when it reaches the silicon layer. Furthermore, it will be impossible to produce latch-ups, as the sapphire is an even better insulator than silicon dioxide.

In spite of this, as this material is transparent and highly insulating, if photons do reach the right transistors of a silicon-on-insulator chip and do induce photocurrents, the detection of such a successful attack will again be almost impossible using only photodetectors. Moreover, as the sapphire substrate is transparent, it would be extremely easy for an attacker to locate and identify the chip's structure and circuits. Additionally, the sapphire demonstrates birefringence, which means that its refractive index depends on the polarisation and propagation direction of light through it.[133] Finally, the sapphire substrate is very hard, having a value of 9 on the Mohs scale of mineral hardness,[131] which means that it would be really hard to remove or thin it and physically access the chip's transistors from the back side.

Apart from the issues already mentioned, silicon on sapphire initially could not be manufactured using small-scale transistor manufacturing processes, because the silicon on sapphire process results in dislocations, twinning and stacking faults forming due to disparities between silicon and the sapphire substrate.[129] Moreover, the silicon closest to the interface is contaminated with aluminium, a p-type dopant used in the sapphire substrate.[129] Such issues together with the fact that this process is quite expensive and leads into bigger than normal chips may make using it quite a lot less attractive.[132] However, less fabrication mask sets need to be used during production and more dense chips can be produced, as transistors can be built closer, because there is no need for an additional insulating buffer material to be placed between two adjacent transistors.[132]

Zirconium dioxide is also, much like the sapphire, a hard,[134] transparent[135][136] material which again exhibits birefringence.[133] Again, using it as an insulating substrate could be expensive, while it suffers from the same vulnerabilities as the silicon on sapphire materials. Its hardness has a value of 8.7 on the Mohs scale and its refractive index is between 2.3 and 1.6 for wavelengths between 400 and 1100 nm, so, it can provide the same advantages as the sapphire.[65][136]

Furthermore, choosing other materials with wide band gaps, which require high energy levels for their electrons to move from the valence to the conduction band, may protect from ionising radiation effects,[8] such as latch-ups, developing in the substrate, but such materials as silicon carbide (SiC) and gallium nitride (GaN) are transparent[137][138][139] and their use rather makes optical fault injection attacks easier than actually hinders them. Furthermore, these materials again exhibit birefringence.[133][140][141] Of course, their low refractive indices[65][138][141][142] may mean that light will be significantly refracted before reaching the transistors and their hardness[138][142] may make it extremely difficult to thin or remove these substrates, but, these qualities do not significantly hinder the possibility of performing an optical fault injection attack against them. Finally, the usage of these materials adds significant production costs, as different processes and techniques have to be used than the ones currently employed in the mass production of smart card chips.

It is obvious that using germanium or indium gallium arsenide wafers will not significantly differ from using silicon as substrate, as their optical properties in the 400 to 1100 nm wavelength range are pretty similar. Furthermore, using materials that do not absorb photons in that range would mean that such photons would move through the material and reach the transistors. Even if some reflective material, such as aluminium (Al), silver (Ag) or gold (Au), could be effectively integrated with silicon and used to reflect light of such wavelengths, an attacker could completely remove it.

We could also try to shield the whole chip or its package by using dielectric mirrors, which are made from a transparent material on which one or more layers of an insulating material are deposited, or by using a material which would absorb light of the wavelength range which can cause optical fault injection in silicon transistors.[39][82][88] However, such materials can be removed, much like the opaque materials which are already used in the chip's packaging and/or encapsulation, or be cut and reconnected in order to be bypassed by an attacker.[3][4]

Changes in the basic components of a transistor

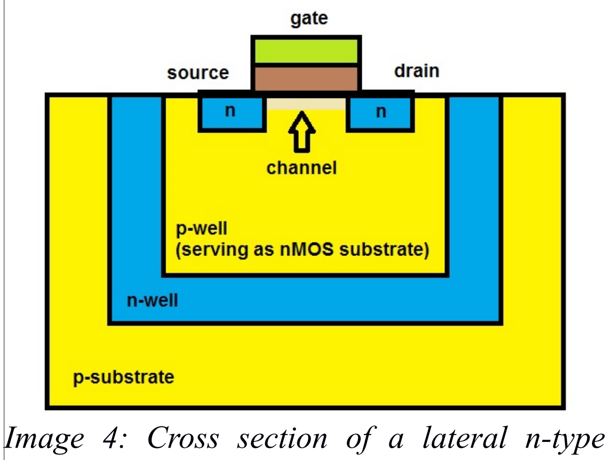


Image 4: Cross section of a lateral n-type triple-well MOSFET.

Furthermore, producing transistors which would have larger drain and source elements and the same size of channel and gate elements as normal ones could also make optical fault injection attacks more difficult.[62] Such transistors would, of course, require more space on the wafer (the disc of pure silicon on which a batch of chips is simultaneously fabricated), but would also counterbalance the effects of induced photocurrents,[62] as the enlarged drain and source elements would react more with the substrate and much less in the channel region. Therefore, more light intensity or longer exposure would be required to flip a transistor.

However, again, such enlarged drains would also require higher voltages to work efficiently and due to their size and shape, they may have more leakage compared to normal transistors. Nevertheless, it has been noted that more laser power is required to switch transistors operating at a higher voltage,[39] while such drains can be constructed through the merging of the drains of different adjacent transistors. This would not be as easily done for the drains of different transistors, as problems may occur in their operation, being related to the common drain's voltage and leakage levels, if their drains are combined.

Deep doped wells

Another approach, which can be combined with enlarging, or combining, the drains and, perhaps, also, the sources of different transistors, is to use a triple well manufacturing process.[62][87][143][144][145][146][147][148][149][150] For an nMOS built on a p-substrate, this involves fabricating a deep (and large) n-well and inside this n-well fabricating a p-well which will, in turn, serve as the substrate of the nMOS (image 4).[62][143][144][145][146][147][148][149][150]

This will result in backside light injections being effectively absorbed by the PN interfaces between the two wells and between the n-well and the p-substrate, inducing photocurrents.[62][143][144][145][146][147][148][149][150] Of course, these interfaces act as diodes[143][144][146][150] and thus the produced photocurrents will, in general, be collected by the n-well and will not propagate out of it. Furthermore, these diode interfaces could, under reverse bias, also act as photodiodes (and potentially also as a phototransistor, under the right biases). Moreover, in the case of frontside light injection attacks, some photons will possibly also reach these interfaces and produce photocurrents by being absorbed, which could be used to detect such attacks.

Furthermore, these PN interfaces may also act in an optical manner, causing a small proportion of the induced light to be reflected, thus allowing even less light to go through them.[62] However, these interfaces may also increase the leakage of the transistor, as they create additional depletion regions, which may interact with the transistor. A solution to this problem could be to fabricate an insulation layer between the two wells, made, for example, by silicon dioxide.

Another solution could be to change the shape of the n-well into forming only a thin layer close to the source and the drain of the transistor, which would expand into being thicker deeper inside the p-substrate, thus forming an n-basin rather than an n-well. Furthermore, we could completely eliminate the very top of the side structure of the n-well layer, which is close to the source and the drain of the transistor, and connect the p-substrate and the p-well with a p-doped silicon bridge, thus effectively burying the n-well inside the p-doped region. In this way, we could eliminate any interference caused by the n-well to the transistor.

Additionally, we could also bury any wires connected to the n-well, that are used to used to bias and/or detect its status, deep inside its body, while keeping them insulated and connecting them with the n-well structure at such a depth that their voltage does not interfere with the transistor's operation. Moreover, depending on its thickness, the n-well can also be fully transformed into a depletion region, having no actual pure n-region. However, this will significantly decrease the effect the n-well region has on the whole structure, regarding both the detection of optical fault injection attacks and the potential protection of the transistor against them.

Furthermore, we could form multiple layers of alternating n- and p-wells, which could, depending on their thickness, significantly reduce their own interaction with the transistor while also enhancing the detection capabilities of their PN interfaces. Nevertheless, such structures would also act as PNPN[P...] (photo)structures, which means that they would be more sensitive to light and much less to electricity, but would also keep conducting for an extended period of time between the transistor's substrate (top p-well) and the bottom p-substrate.[151] This means that while they would indeed detect more efficiently any light injection attacks, they would also potentially transfer the produced photocurrents inside the transistor's structure, through the top p-well which forms the transistor's substrate.

However, even if such structures cannot protect against optical fault injections attacks, they can serve as really efficient detectors for both frontside and backside optical injections. This holds true because the PN interfaces between the top p-well and the top n-well and between the p-substrate and the bottom n-well are able to effectively collect and thus detect both induced photons and the photocurrents these produce within an extended area around them in the front and the back side.[143][144][146][147][149][150] This, of course, is due to the size and location of their p-n interfaces, with the interface between the p-substrate and the bottom n-well covering a broad area on the back side of the transistor and its right and left sides, and the interface between the top p-well and the top n-well covering the main area of the transistor and its front, right and left sides. Any interfaces between the other wells could also be affected by a light injection targeted at the front size of this transistor, due to the spot size of the light beam being used.

Furthermore, to enhance the protection or detection that such a structure provides, we can use different dopant or semiconductor materials on each layer or per set of wells. However, in any case, a separate additional circuit will be required to monitor the wells and detect whether there has been an attack.[143][144][146][147][149][150] However, such a circuit could also implement the actions that need to be taken if an attack is detected.

Finally, if we build a non-depleted epitaxial p-layer between the p-substrate and the bottom n-well, we can enhance even more the ability of the interface between the bottom n-well and the p-layers below it to detect light injections and photocurrents and protect against them, because the added non-depleted epitaxial p-layer acts as a potential well for electrons moving by diffussion.[87][143][144][146][150] This means that the epitaxial p-layer will both act as a well for photocurrents, increasing their collection rate by the interface between the bottom n-well and itself, and also amplify the production of such photocurrents, due to the increased number of charged particles found in it.

This increased creation and propagation of photocurrents in the epitaxial p-layer will thus provide better detection of optical fault induction attacks in the p-n interface between the epitaxial p-layer and the bottom n-well.[143][144][146][150] It is easily understood that if on this whole structure, we also substitute the transistor fabricated on its top p-well for a phototransistor or a photodiode,[152] this structure constitutes an almost ideal photodetector. Furthermore, to avoid parasitic interferences, we can shield off any adjacent pMOS transistors within a deep p-well, covering the sides on which we expect interference from n-doped materials.[150][152]

DEPFETs, voltage and temperature monitors, and other ideas

Finally, a DEPFET (DEPleted Field Effect Transistor) with or without an internal gate can also be used for the detection of light injections (image 5).[150][153] In this case, photocurrents are trapped in the depleted layer and collected by the internal gate and eventually transferred to the clear (gate) when they collectively exceed a threshold voltage.[150][153] However, these transistors take up a larger area than normal transistors and also require an additional circuit to monitor their structure and detect whether there has been an attack.[150]

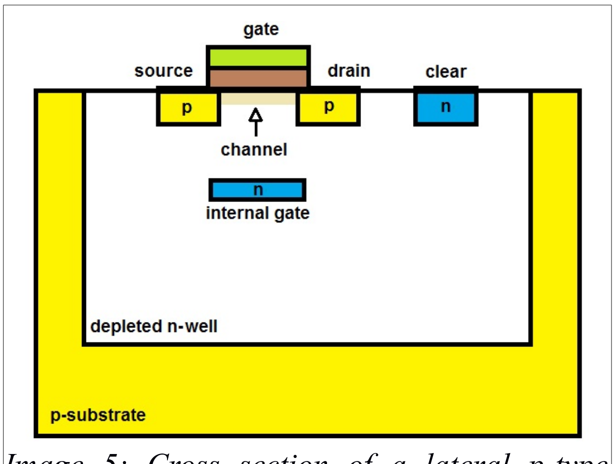


Image 5: Cross section of a lateral p-type DEPFET with an internal gate.

We could also try to indirectly detect optical fault injection attacks through supply voltage monitoring or temperature sensors,[3][4][72][88] but these means may prove quite unsuccessful due to the very short duration of the light pulses which may make voltage and temperature fluctuations caused by attacks undetectable. However, there are circuits that may be able to detect transient (photo)currents, but these require a disproportionate area compared to the transistors they may protect.[154][155]

Moreover, the supply and/or mesh shield wires on the front side of the transistor combined with the ground voltage of the substrate could act as a micro-channel plate detector and detect photons, but only if they have a high intensity, and thus collectively significant energy, enough to cause detectable fluctuations in the difference level between the supply and the ground voltage levels.

Furthermore, it has been suggested that bipolar integrated circuits generally have higher radiation tolerance than CMOS circuits,[8] and thus may also be less susceptible to optical fault injections. Additionally, Magnetoresistive Random Access Memory (MRAM) seems not to be significantly affected by ionising radiation, while offering rewritable, non-volatile memory.[8] MRAM will probably also not be significantly affected by optical fault injection attacks as its operation is based on magnetic storage and not on electric charges or currents.[156]

However, MRAM may take up a much larger area on the chip than conventional memory currently does.[156] Finally, capacitor-based Dynamic RAM (DRAM) and flip-flop-based Static RAM (SRAM) seem to be equally susceptible to optical fault injection attacks, as their stored value can be changed by flipping a single transistor. Nevertheless, adding capacitance or resistance in the feedback circuit of the SRAM could protect against transient light pulses,[87] although this will lead into slower responses and the need for wider signal pulses to change the state of the memory cells[87] and could be defeated by just using light pulses that last longer or have a higher intensity.

Enhancing pre-existing structures

Enhanced shielding

Moreover, although metallic shields and reflective coating do not seem to be adequately effective on their own, we could perhaps combine them with with multiple interconnected layers of fine wiring, in order to get better protection results. Both of these protection measures are ineffective on their own, with metallic shields and reflective coating being removed by attackers and wire meshes not providing any additional protection against optical fault induction attacks. However, by combining these two measures in this way, we can essentially use one (wire meshes protecting against tampering)[157] to protect the other (metallic shields and reflective materials used against laser light penetration).[39][88]

Of course, again, a determined attacker could put the effort and bypass both of these measures, but much more effort would be required to compromise a competent combination of them. Wire meshes could run through the shielding and/or reflective materials covering the whole chip and therefore be able to detect a potential damages in their structures and/or qualities. These meshes could be really dense and perhaps running between different levels of different shielding and reflective materials, with each level being interconnected with each other. Additionally, extra circuits in the internal chip structure could be handling the operation of these meshes.

In an extreme scenario, a battery powering the whole chip and its meshes could also be incorporated inside the whole “fortified” structure to provide internal power and assure that the meshes will keep being effective even if the chip is disconnected from external power sources. However, the cost of such a high level of protection, especially if it also required a power source to be integrated inside it during production, would be significantly high and most probably would prohibit it from being put into mass production.

3D chips and other enhancements

Furthermore, we can also assume that 3D chips may or may not be easier to protect against optical fault injections depending on their exact structure and characteristics. A three-dimensional chip may consist of two or more separately manufactured integrated circuits vertically stacked on top of each other, after their substrates have been aggressively thinned, with each of these integrated circuits referred to as a "tier".[158][159] Each tier is interconnected with the others using vertical metal pillars called "through-silicon vias" (TSVs).[158][159]

While the aggressive thinning of the substrates of integrated circuits combined in a single 3D chip could potentially make optical fault injections easier to achieve, if the chip has three or more tiers, then, we could gather all its security critical circuits and structures in its inner tiers and practically use the top and bottom tiers as protection layers. An attacker could not remove these tiers without effectively destroying the chip, while the transistors found on them would absorb most, if not all the light targeted at a more internal tier, thus shielding it from optical fault injection by essentially being attacked itself and having its operation and functionality sacrificed for this purpose during an attack targeting an element found in the same position on a more internal tier.

Therefore, if we could efficiently gather non-critical circuits and structures on the top and bottom tiers, we could avoid critical elements placed in more internal tiers being successfully attacked with light. Furthermore, we could, in this way, place photodetectors directly above or below such critical structures, which an attacker could, again, not remove without destroying the chip, as they would be integrated in the chip's structure.

However, there are conflicting opinions on whether 3D chips are really more efficient than normal chips or not regarding their speed.[158][159] It has been noted that they may require more power than normal chips,[158] but they have better power conservation.[159] They may require more area,[158] but could also have higher density.[159] It is nevertheless certain that they have much higher production costs, while only 2-tiered ones have ever entered mass production.[158]

Finally, thinning the substrate and shaping it into a convex or a concave mirror in order to deflect laser beams away from their targeted transistors will not be effective, as the attacker can thin the substrate further in order to flatten it out. However, perhaps, it is possible to integrate mirror-like structures or materials on the top side of transistors that will point light to nearby photodetectors, in order to prevent frontside optical fault injection attacks, while it may also be possible to incorporate small photodetectors, such as photodiodes, inside the structure of much larger transistors, in order to detect any attempt to optically induce faults in these transistors. Nevertheless, such solutions are still at an experimental research level and under development, not having yet been introduced in mass production.

Potential logical countermeasures

Therefore, since physical countermeasures cannot, in general, provide an adequate level of protection, it is crucial to also examine the other category of countermeasures, logical countermeasures, and try to compare their costs and benefits, in order to come up with an adequate classification and evaluation of them, in order to be able to draw well-founded conclusions regarding the protection they provide and how that relates to costs and the level of risk that a chip owner or user may be facing.

Logical countermeasures include a variety of different designs and implementations, such as redundant elements,[29][47][54][87][88] parity schemes[87][88] and time or signal checks and inspections,[29] in order to achieve error detection and correction.[8] These countermeasures can usually be implemented either at the level of single transistors or at the level of whole circuits or even in the entire chip.

Redundant elements

More particularly, redundant elements can be used both for error detection, by having a specific circuit or transistor duplicated, or even for error correction by using more than two identical circuits or transistors and comparing their results.[47][54][82][87][88] As the results are of a binary nature, when an odd number of identical devices, circuits or transistors is used for error correction, there will always be a majority of a particular result, while if an even number is used, there may be a majority or a tie between the two results, high or low current, logical one or zero.[88]

However, redundancy obviously leads into a multiplication of the needed area and power for the chip's transistors to be cloned.[47][54][82][87][88] Nevertheless, it does make optical fault injection attacks much more difficult, as a majority of, or even all, the clones of a targeted transistor need to be attacked at the exact same execution point for the attack to be successful.[87][88] Apart from these redundant elements, a comparison circuit will need to be added which will identify faults and either pick the majority result or implement some preventive action against the attack being successful, such as locking temporarily or permanently the chip, or even completely destroying it.[88]

Furthermore, in order to avoid constant faults from a particular instance, we could implement a system detecting whether an instance constantly provides minority results and turn this instance or the whole system off for some time or permanently. In this way, we can prevent a false majority being reached from faulty or targeted structures, while making it more difficult to synchronise attacks between the different instances, especially if the different instances have slightly different delay times. Finally, we could have results that are not reached by a supermajority of instances, for example four out of five, be recalculated,[72] in order to make successful attacks even more difficult due to timing constraints. Furthermore, we could be using different random instances every time, while also preferring instances which gave a majority result in the previous time or operation slot, so that attacks cannot be easily timed, replicated or be easily successful.

However, the implementation of all these ideas would require the addition of several more transistors and circuits in the integrated circuit, which would require more area, higher power consumption, and more design,[88] development and testing time, thus significantly raising the production costs. Nevertheless, at least these structures could easily be integrated in the manufacturing process of the chip, as they require only the addition of known structures that would be made out of silicon, thus not requiring any significant changes in the fabrication process of the chip.

Finally, it is also important to consider the potential changes that the implementation of redundancy and generally the addition of new elements on the chip may introduce on the EDA process and the software being used in smart cards. Especially for the design of such circuits certain steps need to be taken to supplement the EDA-related procedures considering the new circuit designs that may need to be added to them and the need for redundant elements not to be eliminated by them. In the same fashion, software designed to interact with the chip's structures must not only be designed as secured, but also needs to consider the changes that any added security measures may cause to the operation and functionality of the original chip.

Parity checks and algorithms

In general, different ways can be used to detect errors in data, such as repetitions, parity bits, checksums, cyclic redundancy checks and other forms of hash functions.[160] However, most of them face specific issues which may hinder their usage and make it easier for attackers to deceive error detection mechanisms based on them. For example, repetitions[88] can be identified by an attacker and be neutralised by just repeating the fault injection as many times as a pattern is repeated.[54] However, they would, of course, make fault injection attacks more difficult because of timing restraints, as the attack would have to be timed precisely for a number of times.

Moreover, error detection and correction can also be achieved by using parity checks.[8][82][87] This is usually implemented by including parity bits in the transmitted information, which will reveal a fault if a single bit is changed. Parity bits are calculated by parity functions, the most common of which is Boolean function whose value is 1 if and only if its input vector has an odd number of ones, while otherwise its value is 0.[161]

Due to this, problems may arise if an even number of bits is changed, because the parity check may not identify such a result as faulty. As the timing of an optical fault injection is not extremely precise in all cases, so as to always change an exact number of bits, it could easily be the case that there is a 50% chance for a successful attack, even with the introduction of parity bits.

However, if we introduce parity bits at extremely short intervals, this probability is bound to fall rapidly, as a lot of the series of changed bits are bound to be of an odd size and thus identify as faulty in parity checks. Nevertheless, this would introduce a significant account of latency overhead in the operation of such a circuit or of the whole chip.

Additionally, dynamic data, such as those stored in the memory, must be checked for their accuracy each time before they are further used, or preferably have their parity checked at random short intervals.[8] The parity of such data must be updated each time their values are changed, as their values don't remain the same, while each of their bits should participate in at least two different parity calculations and checks, in order to detect errors more effectively. Usually, this can be implemented by checking the parity of memory bits both per column and per row.

Moreover, hash functions can map data of arbitrary length to data of a fixed length and can therefore identify the correctness of data.[162] However, although the produced checksum will usually be completely different for similar input, it may also be the same for completely different sets of input data,[162] which means that a potential attacker may find a way to inject (multiple) faults that will not alter the produced checksum, thus compromising this protection mechanism. Furthermore, since these functions produce error detection data of a longer length than plain parity bits, they, of course, lead to even bigger delays and overheads, while also again requiring control circuits and structures.

Cyclic redundancy checks (CRCs) produce fixed-length check values based on the remainder of a polynomial division of the input data.[163] These short check values are calculated again when the data are accessed again and it is checked if the initial check value matches the new calculated result. [163] Although CRCs are easy to implement, they again produce delays and overheads and would require a check structure.

All these and other hashes can be used to detect errors and, other specific conditions, can also be used to correct these errors. However, they all introduce delays and overheads and could potentially be neutralised. Nevertheless, they do make optical fault injection attacks much harder, because in order to overcome these protection measures, multiple light injections need to be done in different spots on the chip at the same or different moments that need to be identified very precisely.[72]

Check schemes related to the transmission of data

Finally, in cases of data sets being transmitted, the data bus carrying this data must have its size adjusted in such a way that error detection or correction data bits can be transmitted along with the regular information data in an efficient manner. Error correction in transmitted data can generally be implemented in two different ways, either as automatic repeat requests (ARQs) or as forward error correction (FEC).[160]

Automatic repeat requests, which can also be referred as backward error correction, as their name implies are requests for data to be retransmitted each and every time they fail their error detection checks, until they can be successfully verified.[160] Forward error correction is an error detection and correction (EDAC) method in which the sent data also include an error-correcting code (ECC), which can be used to detect errors and also correct them accordingly, by using this redundant code.[160] Automatic transmission requests and forward error correction can be combined in a technique called hybrid automatic repeat request (HARQ), so that minor errors are corrected without retransmission, while major errors require the data to be retransmitted.[160]

Combining parity checks with redundancy

Even if a perfect code, one that achieves the highest possible rate for codes of its block length and minimum distance, such as a Hamming code, is used, still it will only be able to detect and correct a specific number of errors.[164] For example, common Hamming codes can usually detect up to two-bit errors or correct one-bit errors without detection of uncorrected errors.[164] Even improved versions such codes cannot usually detect more than 3 errors.[164] It is therefore evident that even though their usage will make fault injection quite difficult, it will again not be impossible to eventually compromise the circuit.

A combination of parity and redundancy could perhaps provide much more security than either of the two measures alone, as a potential attacker would have to overcome these two measures at the same time and this would create timing issues that would be extremely difficult to surmount.

Another way to increase the security of the chip would be to make each die (each functional unit on a wafer – image 6) completely unique by making slight changes in its design in comparison to every other, or at least slightly change the design per produced wafer. This would, of course, increase design and production costs and each wafer would require an additional amount of time to be fabricated, but could initially isolate the results of potential successful attacks to a small number of chips, by making such attacks quite more difficult to be replicated.

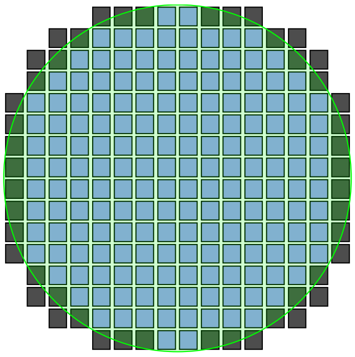


Image 6: Schematic representation of dice on a wafer; blue ones can be fully developed on the wafer, while green/black ones cannot and will be thus rejected, as their black area falls outside the wafer's surface.[165]

Repetition of software operations

Furthermore, even without redundant elements, the software itself could demand the repetition of sensitive and/or critical operations and the comparison of results, in order to determine whether there were errors or the initial result is correct.[4][54] Additionally, if the replicated results do not match the initial ones, the operations could be repeated an additional number of times, and then the correct result be determined by a majority or a supermajority of them.[44][166] Even though this technique would not introduce any particular costs in the development and production of the chip, it could bring about significant delays in the overall operation of such chips, which may still be highly undesirable. Again, this security measure could be neutralised by just repeating the fault injection the required amount of times to prevent the circuit from accurately deciding on the correct result.[4][72] However, this would again be difficult to achieve due to timing and other restraints.

Hardened elements and time or signal inspections and checks

Moreover, other measures exist that could further hinder fault injection attacks from being successful, such as redesigning the flip-flops in order to harden them against fault injections,[167] making the internal clock unstable and employing dummy clock cycles,[4][64][72] setting up timer structures which will perform hard system resets unless they receive a specific data sequence within some limited time period (watchdog timers) and whose operation will be dependent on the production of correct sensitive and/or critical results,[8][87] or detecting whether the input provided to each flip-flop, for example the reset signal voltage, or its output,[29] is the intended one.[168] However, these measures may also introduce significant overhead[29][87] and could eventually be compromised, albeit with much more difficulty.

Asynchronous logic and multiple-rail datapaths

Finally, using asynchronous logic could significantly improve protection against optical fault injection attacks, as it would make it almost impossible to accurately and properly time the attack.[88][89][169][170] Furthermore, we could also employ dual-rail logic, which uses two wires, being referred to as “rails”, to transfer data, implementing redundant datapaths.[29][87][169][170][171] One wire is used for the logic 0 and one for the logic 1.[7][89][170][171] Moreover, more than two rails can be used to encode each logic signal, and serving to detect errors, or there can implement even higher datapath redundancy by carrying the same signal, thus providing not only error detection, but also error correction.[170] However, an additional circuit would be needed to decide about the correctness and selection of the signals carried through them, by picking a majority or supermajority of them in case they are redundant to each other.

Additionally, asynchronous logic circuits have significant disadvantages, such as requiring much more area to be implemented,[89][172][173] being more difficult to design, test and debug than synchronous ones,[172][173] may not perform as well as synchronous ones and are incompatible with most commercial Electronic Design Automation (EDA) tools.[89][172][174] Dual-rail logic would also, obviously, require at least twice as many transistors as the commonly used single-rail logic to be implemented.[175]

Final remarks

As it has not been sufficiently proven that any of the previously described countermeasures can adequately protect against optical fault injection attacks while also being cost-efficient, it is evident that we should try to classify and compare them, in order to evaluate their merits. It is not difficult to classify the described countermeasures into categories according to the structural level of integration into physical-level, semiconductor-level, circuit-level, EDA-based and software-based ones. In this essay, we have mostly focused on the hardware-related categories, while, however, always aiming to identify the most optimal solution.

Therefore, we will try to evaluate these countermeasures regarding the level of protection they offer against optical fault injection attacks, the required costs for their implementation and integration and other potential shortcomings.

Chapter 3: The study and its results – A comparison and evaluation of countermeasures

In order to compare and evaluate the previously described countermeasures, it was considered necessary first to identify which of them seemed more optimal to be actually used as such and which ones cannot at present be used as countermeasures against optical fault injection attacks and for which reasons. A number of different physical structures and logical controls were considered as potential countermeasures, but especially the implementation of certain physical countermeasures proved to be significantly hindered by current technology, in terms of the manufacturing process being used at present. As it has already been discussed, this is an impenetrable barrier because changes on the manufacturing process have an enormous cost for the fabrication plant (which is commonly referred to as the *fab*) and can certainly not be imposed to it by the integrated circuit designing company or even be somehow implemented by this company on its own.

Therefore, it becomes evident that if a particular countermeasure requires changes in the manufacturing process or significant revisions of other processes related to its design, development and implementation, then this countermeasure probably cannot be utilized or even be considered for integration in an actual integrated circuit. Unfortunately, in such a case, we have to reject such a countermeasure and, at present, it would make little sense to examine its potential properties. However, such ideas could be used in the future and, perhaps, provide better results than the ones we have currently tried to fully evaluate regarding their cost and level of protection they provide.

We tried to evaluate a selected number of potential countermeasures either through trying to design them or by relevant simulations, in order to estimate their area cost and potential protection provided. Logical countermeasures were designed, simulated and converted into netlists, in order to both test their effectiveness and their space requirements, while physical countermeasures were simulated according to the design rules imposed by the current manufacturing process and then tested under a few different configurations to assess their effectiveness.

The primary function of these countermeasures is to improve the probability of detecting optical fault injection attacks, in order to make smart card chips more robust against them. In case faults and errors are detected, an adequate response step can be taken, such as locking or completely disabling the device, either temporarily or permanently.

Physical countermeasures

As most physical countermeasures introduce innovative structures that would need to be integrated in the produced integrated circuit, they tend to be highly incompatible with the current manufacturing process. In this thesis, the current manufacturing process under examination is the CMOS 90 nm low power one, because it is one of the most widely employed processes for the production of secure chips for smart cards. It has also reached a quite mature and stable stage as it has been used for at least 10 years now and therefore no significant changes are expected to be introduced in it, unless proven really essential for it.

Therefore, initially, we tried to identify the compatibility of the previously suggested physical countermeasures with it and thus establish whether further investigation over their qualities and merits was warranted, as in case they were totally incompatible with the current manufacturing process, there was no point to further examine them in much detail, as they cannot be efficiently implemented and/or integrated on a mass production end production. Therefore, the cost of their potential implementation is too high to even attempt their experimental integration and production, because there is an insignificant probability of them being introduced into a production phase working under economies of scale, and thus the cost of their research and development will never be regained through sales profits.

Selection of the design ideas to be explored in detail

For example, considering the idea of adding graphene on photodetectors to enhance their detection abilities, we can easily conclude that it may cost too much to be integrated in the current manufacturing process which requires clean and sterilized rooms, while graphene is a material which can easily spread around on unwanted places, due to its powdery texture. Therefore, it would require changes which are doable, but may not be financially feasible, and which may not have the desired results, due to several constraints regarding the handling of wafers, dicing and the other required processes to produce a chip.

Furthermore, using insulators as the chip's substrate would result in rather negative consequences regarding the level of protection against optical fault injection attacks, as already noted before, because these insulators tend to be transparent in the wavelength range between 400 and 1100 nm, thus rather enabling optical fault induction than actually preventing it. Additionally, significant changes would have to be done to the manufacturing process to incorporate such a change which would also come at a considerable cost, therefore making such a potential change completely undesirable.

Moreover, for the same reasons, any countermeasure based on the use of other semiconductors, such as germanium (Ge) and indium gallium arsenide (InGaAs) would also require excessive changes both to the manufacturing process and the electronic design automation (EDA) one. This would happen because these materials would require different constraints, rules and methods regarding their design, implementation, integration and actual production in order to provide the same or equivalent circuits and functionality, as they exhibit similar but not exactly the same physical characteristics and properties as silicon (Si). Furthermore, using transistors with larger drains and/or sourses would lead to significantly increased static power leakage and would also require more area and, of course, more dynamic power for their proper operation.

Additionally, while a deep n-well structure could significantly help in protecting transistors from backside light attacks, it would in practice do so only regarding an nMOS and not a pMOS. This would happen because the n-doped substrate of a pMOS is built to be connected to the deep n-well and therefore any photocurrent induced in the p-n junction between the overall p-substrate and the deep n-well will also be transferred to the n-substrate of the pMOS and thus will affect its operation, with a possibility of causing it to malfunction.

Apart from this, deep p-wells are not provided for in the current design and manufacturing process, therefore significantly hindering any attempts to shield a pMOS against the influence of an underlying deep n-well. Of course, this also means that any combination of deep n-wells and deep p-wells cannot be designed or implemented. Furthermore, both deep n-wells and deep p-wells require a significant amount of area, due to design constraints, while their combination may have also suffered from phenomena related to PNPN[…] structures and may have led to significant leakage.

DEPFETs would also require changes regarding both the current manufacturing process and the EDA process, as both of them do not currently provide for such designs. Additionally, DEPFETs require a larger area than usual transistors and are also bound to suffer from excessive leakage, as their detection operation is based on this very issue, with photocurrents being trapped in the depleted layer and collected by the internal gate, eventually leaking to the clear (gate) when they collectively exceed a threshold voltage. Thus, DEPFETs cannot really serve as regular transistors, as such leakage and the mere presence of the internal gate would interfere with any currents going through their channel area, and are designed to only operate as detectors.

Voltage and temperature monitoring would require a significant amount of area and designing in order to be implemented, as the current detection technology would face significant difficulties in identifying tiny voltage or temperature fluctuations caused by attacks with light pulses of very short duration. Especially as such attacks can be performed with laser beams made of pulses lasting as little as some nanoseconds or femtoseconds, it can prove very hard, if not impossible to adequately detect voltage or temperature disturbances caused by them.

However, while resistors are being used to detect temperature, photoresistors could be used to immediately detect light. Nevertheless, the current design rules provide only for resistors made from silicon (Si) and thus their usage as photoresistors is questionable. Moreover, they would require a significant amount of area in comparison to their detection area span, while photoresistors made from other materials are incompatible with the current manufacturing process and the EDA, or are even banned by legislation.

Mesh shields already exist on produced integrated circuits but have proven ineffective as they can mostly be removed. Furthermore, due to constraints in the design and in the manufacturing process, they cannot cover the whole surface of the integrated circuit, thus allowing for fault injection attacks to take place bypassing them. Additionally, the use of MRAMs instead of conventional memory structures would require a new EDA process to be developed to incorporate it and define its design, placement and constraints, therefore at present is not compatible with the current manufacturing process.

Finally, a 3D chip would obviously also require extensive changes in EDA regarding the introduction of new elements and different rules and constraints regarding its design and a completely different manufacturing process in order to be built, exactly because of its multi-tiered structure.

An idea which seemed more promising would be to use photodiodes as detection mechanisms instead of the current implementation of photodetectors made out of phototransistors. This is because photodiodes can be easily implemented based on the current EDA design rules and constraints and are completely compatible with the manufacturing process being used, while also being significantly smaller than phototransistors. Therefore, it would make sense to simulate a photodiode structure and measure its response to optical fault injection attacks and then compare it to that of a phototransistor.

Design and Simulation

We decided to explore two different scenarios regarding the integration of a photodiode in the chip; one where the photodiode is just built on the substrate of the chip and one where there is an underlying deep doped well. We decided to simulate the latter scenario not only because of the issues (photo)transistors may be facing in the presence of a deep doped well, but also because deep doped wells are common in areas of the chip that are targeted by optical fault injection attacks, such as areas where memory or other potentially vulnerable elements can be found. In order, however, to enhance the possible detection span of such devices, we provided for a small gap in the deep doped well exactly underneath the device's position. This would ensure that the deep doped well would allow for measurable detection results, while still providing protection to any nearby transistors above it.

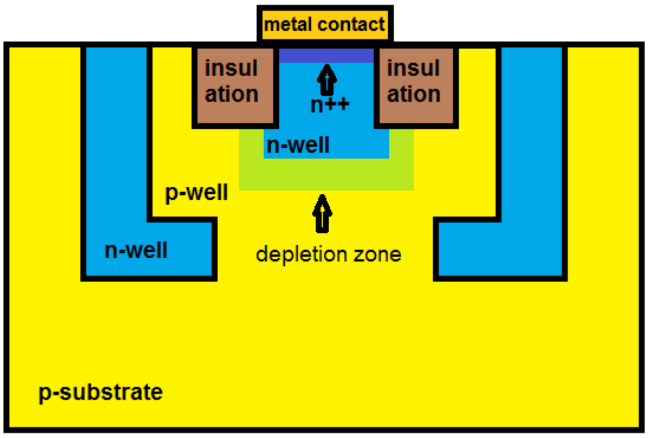


Image 7: Cross section of a vertical (photo)diode with a NP junction (n-p diode) above a deep n-well with a small hole exactly underneath it.

A series of simulated transistors were designed over a simulated area, as they would be found placed on a real chip, with a photodetector device, a phototransistor or a photodiode, simulated to be placed among them. The simulation was constructed using a custom-made company program in such a way as to be fully compatible with the CMOS 90 nm low power manufacturing process rules and constraints.

However, as only deep n-wells can be built in reality using this manufacturing process, we chose to simulate a deep n-well inside a p-substrate and a vertical n-p (photo)diode above it, with a small gap in the deep n-well exactly below the diode (image 7), as well as the same (photo)diode and substrate without the deep n-well structure (image 8). Of course, the same principles applied for an nMOS (photo)transistor, as well.

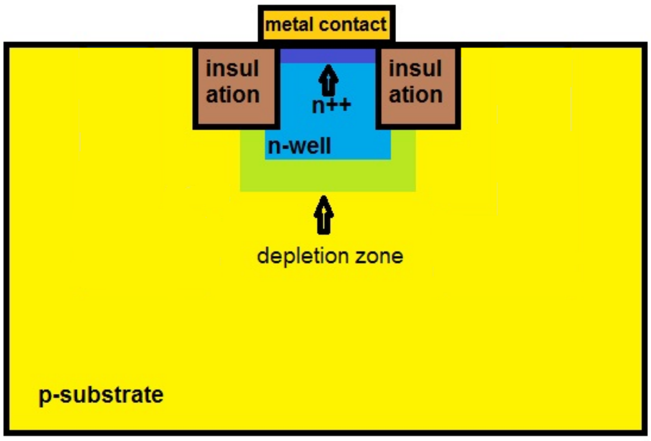


Image 8: Cross section of a vertical (photo)diode with a NP junction (n-p diode).

Then, each simulated chip was tested against a simulated optical fault injection attack and the detection rates of the photodiode and the phototransistor were compared. This was achieved by simulating the injection of light in the simulated structure with a known intensity and then dividing the resulting electric current response of each detector by that intensity.

The minimum size of the (photo)transistor was calculated to be around 2.96 μm^2,with an overall area needed for the whole structure being around 15 μm^2 due to design constraints and other factors. Therefore, we designed a (photo)diode of the same size, one of about half that size (1.53 μm^2) and one of about a quarter of the original size (0.78 μm^2), in order to see how scaling down would affect the detection rate of the (photo)diode and if the results would be comparable to that of the (photo)transistor. Additionally, the overall area was again simulated to be around 15 μm^2 due to design constraints.

Results

Unfortunately, the results proved that, in general, (photo)diodes are not as good photodetectors as (photo)transistors as they seem to perform ten times worse than (photo)transistors of the same size in detecting the presence of light of the same intensity. And, of course, as their size shrinks, their performance keeps deteriorating.

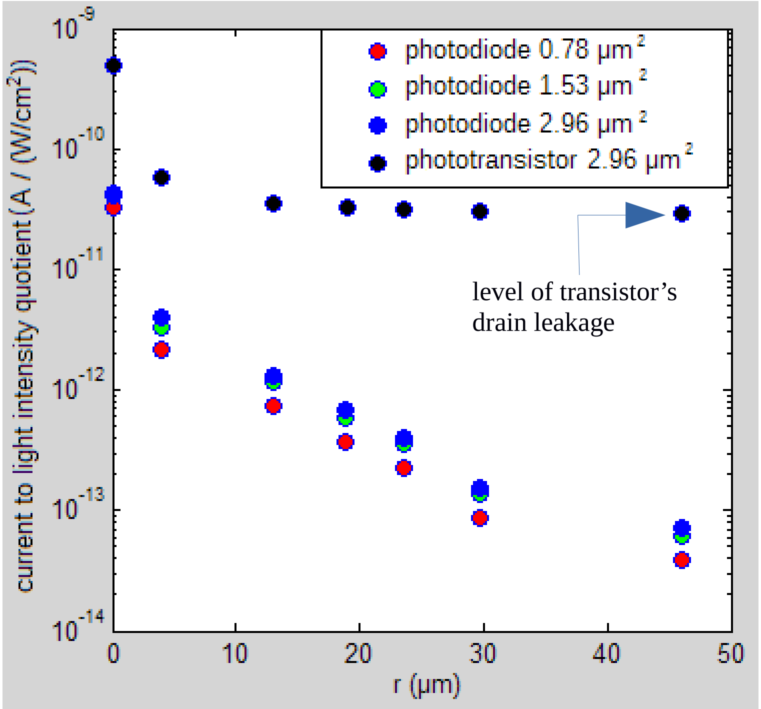


Image 9: The quotient of the current produced by the detector to the intensity of the light reaching it in relation to the distance (radius) away from the detector.[176]

We came to these conclusions by comparing the quotients produced by the division of the current produced by the device to the intensity of the light that reaches it for a phototransistor and a photodiode (image 9). In the case of a photodiode the generated current is equal to the light-generated charge, while in the case of a phototransistor it is equal to the light-generated charge multiplied by the phototransistor’s gain.

If less light reaches the photodetector, it will have less of an effect on it, thus more easily falling below the set detection threshold. We, of course, measured this quotient in different distances from the photodetector, in order to find the expected value of the effective detection area span.

Therefore, and as photodiodes having half, or a quarter of, the size of a phototransistor give comparable results to those of a photodiode of the same size as a phototransistor, to get the get same detection area span as a phototransistor, we would need to use a photodiode being way more than ten times the size of a phototransistor. Thus, we can easily conclude that using (photo)transistors as photodetectors is preferable to using (photo)diodes in order to detect optical fault injection attacks.

However, we also noted that diodes have less leakage in the simulation than phototransistors and, therefore, their threshold current could be set at a lower value, thus allowing for a bigger detection span. It is also important to note that the transistor’s drain leakage level is quickly approached by the current to light intensity quotient values, thus only allowing for a relatively low threshold and a small detection span. Nevertheless, current technology severely limits the ability to detect fluctuations of such small amplitudes as would be required to detect light-induced currents coming from light injections made some tens of micrometers away from the photodiode, because in this case they would be in the order of some picoamperes (10^{-12} A). Furthermore, the leakage of the photodiodes is dependent on the temperature and as temperature rises, it significantly increases, thus not allowing for a low enough threshold or producing a large number of false positive detections in the opposite case.

In case there is a deep n-well below the photodetector, then most of the light induced on the structure will be absorbed by the deep n-well. However, if we make a gap on the n-well exactly below the detector, then we can use it to detect potential optical fault injections. In the case of the photodiode, such a hole can be made on the overall deep well, while a phototransistor would require an isolated deep n-well of its own. This would have the effect that the overall area required for the integration of the phototransistor would be really large, while the phototransistor would remain quite smaller compared to it. Therefore, the actual area surrounding it that could be used for the placement of sensitive chip components would be quite smaller than the one in the case of a photodiode.

Additionally, in such a case, (photo)diodes may perform better than (photo)transistors, as their detection ability is not hindered by the presence or absence of a deep n-well surrounding the area below them. On the other hand, a (photo)transistor's detection ability significantly deteriorates in the same context, with the associated current to light intensity quotient taking values of the order of 10^{-15} A (some femtoamperes).

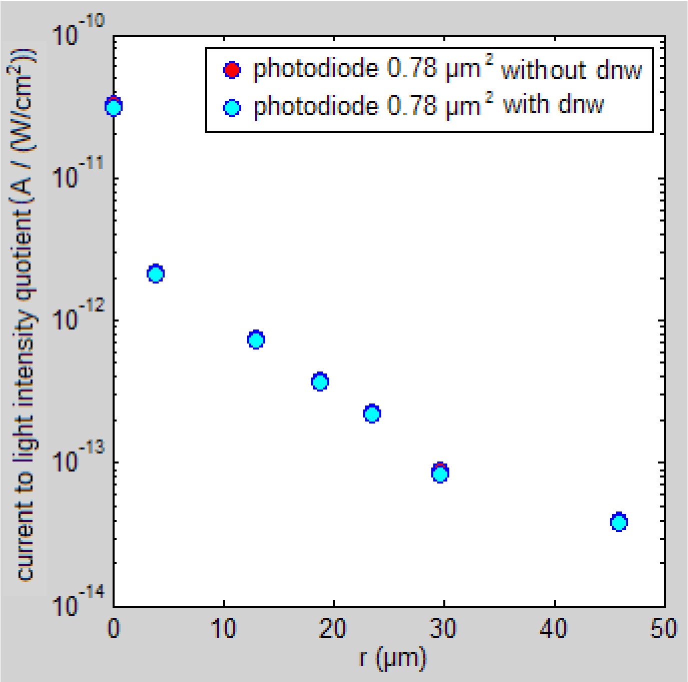


Image 10: The quotient of the intensity of the light induced on the chip to the intensity of the light reaching the detector in relation to the distance (radius) away from a photodiode in the presence and absence of a deep n-well.[176]

Therefore, and as a (photo)diode having ¼ the size of a (photo)transistor has 1000 times higher current to light intensity quotient (image 10), (photo)diodes in this case have a significant advantage over (photo)transistors, whose detection rates do not allow them to realistically compete to perform as photodetectors in such a case. It may however be good to note here that in the case a deep n-well is present in the substrate, more area will always be required to integrate any photodetectors above it, due to design rules and constraints.

However, as deep n-wells tend to be common in areas of the chip that are highly likely to be targeted by optical fault injection attacks such as areas incorporating components related to the chip's logic and other potentially vulnerable elements, the fact that photodiodes provide better results than phototransistors in the case a deep n-well is present is really important.

Logical countermeasures

Logical countermeasures were designed on VHDL and Verilog and subsequently simulated on Cadence SimVision in order to assess their ability to detect, or even correct, errors induced by faults injected on the chip. Logical countermeasures took the form of error detection (or correction) schemes and redundant elements used to detect (and possibly) correct errors occurring on data stored on a register file. The ability to write and read on the register file in real time was also coded, as well as flag bits to signify the detection of an error, as well as a clock and a reset signal. In some cases other flag bits were also coded in order to signify a correction of the previously corrupted data.

When the register file was simulated, data was written on it and read out of it, while errors were programmed to occur at specific points. Using SimVision and the error detection and signal bits or streams coded in the registers, we tested whether the error detection (or correction) schemes worked as originally planned. Single-bit errors were introduced on different words of the register simultaneously or at different points of time and were always detected. Additionally, depending on the properties of each exact algorithm, multiple-bit errors could or could not be detected, as it would be expected from a coded deterministic algorithm (a computer program).

Single-bit fault injections were simulated by forcing a single particular bit (cell) per word to take a specific value opposite to the one it should normally and correctly have, i.e. if the bit had the value of logical “0”, it was forced to become a logical “1” and vice versa. Then, as expected, when the check was performed in the next clock cycle, an error was reported, as an error detection bit or stream set to the value of logical “1”. If the algorithm was able to detect specific cases of multiple-bit errors, then these were indeed detected. Obviously, however, not all cases were tested, as exhaustive testing would take way too much time, and the algorithms used are not too difficult to test regarding their general functionality, due to their size and the operations they perform. In general, our focus was on how such algorithms may rank up depending on cost (in terms of area and power) and on the protection they offer.

Therefore, it was programmed that when an error was detected, all the register file data, as well as the output stream, would be erased. Of course, in case other actions should be deemed appropriate by the manufacturer or the client, such as locking the whole chip or erasing other data or using more flag bits or a counter for the amount of detected errors, these could easily be added to the existing code, using, in this way, the error signal in the context of the overall system. Additionally, it should be noted these logical countermeasures provide protection not only against optical fault injection attacks, but against any fault injection attacks, as they can be used to detect, and in some cases correct faults and errors in the chip’s data, in general.

Moreover, the design code for each logical countermeasure was converted into a netlist, in order to assess the area and power requirements. All designs were converted according to the CMOS 90 nm low power manufacturing process, because it is one of the most widely employed processes for the production of secure chips for smart cards and also it was the one used for the simulation of the physical countermeasures.

A simple register was selected as the basic reference design in order to assess the different logical countermeasures, because a registers is a particularly vulnerable and targeted area on a chip for optical fault injection attacks. The register was designed having 32 words (lines) of the same size. Three different register files each having a different word size were designed, in order to assess how the size of words in a register could affect the overall area required by each design. A register file with words of 8 bits, one with words of 16 bits and a 32-bit word one were designed.

Data could be written in the register by selecting one of its words and copying input data on it. Word lines can only be written while a check bit is set to logical “1”, in order to disallow illegal writes. In order to read data from the register, again, a single word had to be selected and its data was copied out of the register as output. The registers are also clocked and can be reset, i.e. have their data erased, when a reset signal is set to logical “1”. A series of different designs which were incorporating parity and redundancy as detection features (logical countermeasures) against fault injections were designed, simulated and transformed into netlists.

The detection schemes were designed in such a way that only errors consisting of a single bit fault can always be detected. Most errors happening in more than one bits will, in most cases, not be detected, due to the nature of the algorithms being used. This decision was taken in order to allow designs to be simple and not require an excessive amount of area for their implementation and integration on a chip. However a decision was made to check the integrity of each word of the register on every clock cycle for possible faults, in order to compensate for the inability of most algorithms being used to detect non-single bit faults. In this way, we could detect multiple-bit faults on a single register, only if they happened on different words, as, unfortunately, only one bit fault per each word of the register would always be detected.

The reason why single-bit errors are so important is that, as already stated before, attackers are aiming to be able to cause errors on specific bits in order to be able to precisely manipulate the stored data. In case more data bits are affected, the error may or may not be detected, depending on the exact data bits affected and the algorithm being used, but it is highly likely that the attacker will also not cause the desired result, but rather cause the chip to malfunction, especially if a large number of bits are affected. Additionally, since laser beams have a rather circular, or perhaps elliptical spot, and we are able to detect single-bit errors on each and every word of the register, it is highly unlikely that a laser beam that will cause multiple-bit errors on the word targeted, will not also cause a detectable error in one of the other nearby words of the register.

An obvious example would be using a laser beam with a large spot size that causes 3-bit faults due to its size on the targeted word, obviously 3 bits in a row. This would, however, mean that the data bits exactly below the central targeted bit, and which belong to the previous and following words following the targeted word, would also be affected and their values would change, due to the same light intensity and energy reaching them as the light intensity and energy reaching the bits nearby the central targeted bit of the targeted word.

In general, it would take very precise timing and placement in order to affect multiple data bits on only a particular word and not affect data bits in other words in such a way that an error would be detected. This would, of course, lead into raising quite significantly the cost of the attack, as potentially a large number of smart card chips would have to be used until the attack was performed successfully, which, in turn, would raise the question of availability of such a number of such chips, and, again, imply even higher costs.

Additionally, it must be mentioned that all the designed logical countermeasures are completely compatible both with the EDA and the manufacturing process, as they have been translated into netlists and thus can be implemented using the commonly used gates and components found on a chip. However, of course, further improvements on the produced netlist may result in less area and lower power being required for their implementation and operational functionality.

Furthermore, error correction mechanisms used in the transmission of data were not designed, because they are related to the specific design and implementation of a circuit and are strongly dependent on the integration of the circuit’s components with each other, therefore different designs would have to be made for each different chip design. Additionally, while general algorithmic schemes could be used to produce such designs and then integrate them in a particular chip, it is not yet sufficiently proven that faults can be optically induced on data while they are being transmitted.

Moreover, watchdog timers, specific flip-flop structures and other non-standard cells were not designed or otherwise tested because they require significant additional area for their implementation, introduce significant overhead in the integrated circuit’s operation and non-standard structures and cells are not compatible with the current Electronic Design Automation (EDA) process and, sometimes, not even compatible with the current manufacturing process.

Finally, the idea of using a completely unique design for each batch of produced chips or at least for each new chip design or chip family seems to also be thought of as too costly, because it would require additional research and development costs in order to produce each unique design, as well as additional productions costs due to its unique layout (requiring the designing and using of new production masks) and increased additional testing costs related with the integration of the different components with each other and the acceptance and functionality testing of the full system. In this case, a known tradeoff between cost and risk exists, as cases where chips were compromised based on their design being similar or identical to that of other already compromised chip families are well-known.

Parity checking designs

The first of the parity checking designs incorporates parity checks by including a single additional parity bit per word. This bit is calculated as the modulo of the division of the sum of each word’s bit cells by 2, which is equal to XORing these cells with each other. This bit is stored in an additional extra bit cell added on each word for this purpose and is calculated every time a word is being written. The parity bits are re-calculated for every word and checked against their stored values on each clock cycle. If the two values (stored and re-calculated ones) are not found to be equal, then an error bit is set to logical “1” and provided as output, while the register and all its other outputs are reset.

A more advanced design introduces a parity bit for every 4 bits of each word. This means that depending on the word size of the register, a different number of additional bit cells is required to store the produced parity bits. Again, each parity bit is produced by XORing each word’s cells with each other, but this time this happens for every 4 bits of the word. This means that for a register file with words of 8 bits, 2 parity bits will be produced, while for a register with words of 16 bits, 4 parity bits will be produced. Again, additional bit cells are added on each word of the register to store these bits. The parity bits are re-calculated for every word and checked against their stored values on each clock cycle. If the two values (stored and re-calculated ones) are not found to be equal, then an error bit is set to logical “1” and provided as output, while the register and all its other outputs are reset.

A design on the basis of a simple checksum of each word was also made. In this case, each word’s cells are again added to each other, but this time they are divided by a predefined divisor which is selected accordingly. Obviously, this divisor could be the number 2, resulting in the very first design, but could also be set to, for example, 3 or 4, in order to gain more variety in the produced checksum, which again corresponds to the modulo of this division. This is done in order to prevent the possible injection of a single bit fault injection on the parity bits resulting in not detecting a simultaneous fault injection of a single bit fault in the word's stored data (a false negative). As the produced modulo will almost always need more than a single bit in order to be stored, again, the number of additional bit cells required to be added on each word has to be set. Like before, the parity bits are re-calculated for every word and checked against their stored values on each clock cycle. If the two values (stored and re-calculated ones) are not found to be equal, then an error bit is set to logical "1" and provided as output, while the register and all its other outputs are reset.

Furthermore, CRC (Cyclic Redundancy Check) was also implemented in another design in the following way; this time, each whole word is divided with a predefined number, with the modulo of this division providing the parity bits of this design. In this way we can always detect a single-bit fault, if the predefined divisor is other than a power of 2. If a power of 2 is selected as the divisor, then any faults injected in the data bit cell corresponding to that power, for example the first bit cell for "1", the second bit cell for "2", the third one for "4" and so on, will not be detected. In case, however that the divisor is 3 or 5, then all single bit fault injections will be detected, but again multiple bit faults that result in a word changed by a multiple of 3 or 5 will not be detected. Again, as the produced modulo will need more than a single bit in order to be stored, the number of additional bit cells required to be added on each word has to also be set. The parity bits are re-calculated for every word and checked against their stored values on each clock cycle. If the two values (stored and re-calculated ones) are not found to be equal, then an error bit is set to logical "1" and provided as output, while the register and all its other outputs are reset.

Finally, Hamming code parity schemes were designed to calculate Hamming parity bits in different ways, in order to assess which code implementation was better. Hamming parity bits were calculated by XORing the relevant bit cells, for example odd position (0, 2, 4, …) bit cells for the first parity bit, bits in positions 1, 2, 5, 6, 9, 10, … for the second parity bit, bits in positions 3, 4, 5, 6, 11, 12, 13, 14, … for the third parity bit and so on. The number of additional bit cells needed could be easily calculated being equal to the base 2 logarithm of the word size of the register plus one more bit. Once again, the parity bits are re-calculated for every word and checked against their stored values on each clock cycle. If the two values (stored and re-calculated ones) are not found to be equal, then an error bit is set to logical "1" and provided as output, while the register and all its other outputs are reset. However, these designs did not, at this stage, include autocorrection features.

Modular redundancy elements

Physically redundant elements were also designed, with an initial design introducing a second register on which the same data is written at the same time as input data is written on the original register. The original register's word is checked against its copy, in order to detect faults injected in either register. If they are not found to be equal, then an error bit is set to logical "1" and provided as output, while both registers and all other outputs are reset. This design is based on dual modular redundancy and again can only detect faults, but not correct them.

A design based on triple modular redundancy introduces two redundant registers in addition to the original register for a total of three. In this case, the original register's word is checked against both its copies, in order to detect faults injected in any of the registers. If they are not found to be equal, then an error bit is set to logical "1" and provided as output. However, this time, it is examined whether any two copies of a particular word agree with each other. If this happens, then, the third copy is replaced by a copy of one the other two registers, which were found to have exactly the same data. If not, then, all the registers and all other outputs apart from the error one are reset. Therefore, this design can not only detect faults, but also correct them.

Dual modular redundancy was combined with all the previously mentioned parity checking designs in order to introduce error correction in them. The parity bits of both registers are re-calculated on each clock cycle. If based on the re-calculation of the parity bits of the original register an error is found, then, it is examined if the re-calculated parity bits of the other, redundant, register match the ones stored on it. If they do, then the relevant line is copied from the redundant register on the original one and the error is considered fixed. If not, then both registers and all other outputs apart from the error one are reset.

Autocorrection on a single register file

Hamming code can be used for autocorrection, as it can be calculated from its parity bits which exact data bit does not match them *in case only one bit is wrong*. Again, different ways of calculating the Hamming parity bits were designed in order to evaluate which one of them was better. Again, the number of additional bit cells needed is equal to the base 2 logarithm of the word size of the register plus one more bit. Once again, the parity bits are re-calculated for every word and checked against their stored values on each clock cycle. However, this time, if the two values (stored and re-calculated ones) are not found to be equal, then an error flag bit and the relevant bits on an error register are set to logical "1" and then used to identify the wrong bit's position in order to try to correct it. If the problem is considered fixed, then the bits of the error register which correspond to the faulty word are set to logical "0". If the fault is indeed fixed and no more detected, then the error flag is also set to logical "0". If not, the register file and all outputs apart from any error related ones are reset.

Additionally, one more parity bit can be added on each word of the previous design, calculated by XORing the Hamming parity bits. Then, in case the stored parity bits do not correspond to the re-calculated ones, again the error flag bit and error register are set accordingly, but, also, this additional bit is re-calculated and it is checked whether its re-calculated value corresponds to the stored one. If it does, then, the wrong data positions are calculated as previously described and the wrong data are corrected. If the re-calculated additional parity bit does not match its stored value, then, it is assumed that a fault has been injected in the parity bits while the data bits are correct, so the parity bits are re-calculated and their new values are stored in the register file. Again, if the problem is fixed, the error flag bit and the error register will be set to logical "0". However, if the fixing process fails to successfully address the error, the register file and all outputs apart from any error related ones are reset. In almost all designs, the error flag bit is used as the error output to signify a fault has been detected. The error register can also be used, but its size makes this choice not an optimal one.

Finally, another design which could support autocorrection using a single register would implement parity checks not only per word, but also per position in the word. In this way, we could add parity bits both for the rows and columns of the register, and, thus, identify faulty bits by row and column, in order to find their exact location and attempt to fix them. This design was initially coded using CRC in order to employ a parity scheme which is more robust and less prone to successful neutralization. The CRC of each row and column of the register is calculated and stored in parity bits on extra rows and columns. The number of extra rows and columns needed may vary, depending on which divisor is used to calculate the CRC of the rows and which for the calculation of the CRC of the columns. All the parity bits are re-calculated for every row and column, and, then, checked against their stored values on each clock cycle. If a mismatch is found between a recalculated and a stored value, then the relevant bit of the row and the column error registers, as well as an error flag bit, are set to logical "1". The faulty bits are thus identified by their position and an attempt is made to fix them. If the fixing process succeeds, no errors are found anymore and the error register bits are set to logical "0" again. If all the bits of both error registers are logical "0", then also the error flag bit is set to “0”, otherwise it stays “1”, until a reset signal is issued. The design can also be modified in order to reset the register and all outputs apart from any error related ones in case the error cannot be fixed.

The same design was later modified in order to use checksums per row and per column, or only single parity bits per column and per row, instead of CRC’s, in order to find out how using different parity checks would affect the area required for each circuit to be implemented. Additionally, more parity bits can be added in order to check whether the original parity bits are affected by faults, when an error is found. These are calculated by XORing the parity bits per column and per row and then XORing the two products of the two initial XOR operations. In this way, we get only one bit to store per parity row and parity column, which is stored in the relevant position on the register. We can use these additional parity bits to determine whether a fault has occurred in the initial parity bits and replace them on the register by recalculating them or if the actual data bits need to be fixed, when an error has been detected. Like before, if this process succeeds, no errors are found anymore and the error register bits are set to logical “0”. If all their bits are logical “0”, then also the error flag bit is set to “0”, otherwise it stays “1”, until the register is reset. Again, this design can be modified in order to reset the register and all outputs apart from any error related ones in case the error cannot be fixed.

General remarks on the designs

All the designs related to logical countermeasures were made, simulated and examined for three different register sizes; a 32x8 register, a 32x16 and a 32x32 one. This was done in order to assess how the word size of registers with the same number of words (32) affects the overall area required for a particular design. Additionally, all designs were coded dynamically and not statically, thus possibly resulting in more area being required for their implementation than otherwise. For example, register cells were added using a *for* statement in this manner:

```
sum = 0
for i = 1 to n
        //add reg[i] to sum
        sum = sum +reg[i],
```

rather than being added in the following manner:

```
sum =0
sum = reg1+reg2+reg3+...+regn,
```

where n is a known always static number. This was done in order to keep the program code dynamic in order to be able to scale the implementation easily. However, a *for* statement for a certain specific operation on a certain specific number of elements may require a more complex circuit to be implemented than a static operation on these elements.

Furthermore, the number of additional parity bits required by each design is given in the following tables. In the cases of redundancy, it is meant that one or more additional register(s) are designed as exact copies of the original register file, thus including the same number of additional parity bits as the original register file does. In each case, we chose to use as few parity bits as a functionally correct and complete design would allow, by selecting the most appropriate specific characteristics, such as divisors, in order to keep the designs as simple and cost efficient as possible.

Finally, the appendix of this thesis contains the Verilog code of the different designs of logical countermeasures for the 16-bit word registers.

Parity bits required per word			
Design	8-bit word size register	16-bit word size register	32-bit word size register
Checksum	$\log_2(max((\sum(word\,bits))mod\,3))=2$	$\log_2(max((\sum(word\,bits))mod\,3))=2$	$\log_2(max((\sum(word\,bits))mod\,3))=2$
CRC	$\log_2(max((word)mod\,3))=2$	$\log_2(max((word)mod\,3))=2$	$\log_2(max((word)mod\,3))=2$
Single parity bits	1	1	1
Parity bits per 4 data bits	$\frac{length(word)}{4}=\frac{8}{4}=2$	$\frac{length(word)}{4}=\frac{16}{4}=4$	$\frac{length(word)}{4}=\frac{32}{4}=8$
Hamming code (without redundancy and without autocorrection)	$\log_2(length(word))+1=\log_2(8)+1=4$	$\log_2(length(word))+1=\log_2(16)+1=5$	$\log_2(length(word))+1=\log_2(32)+1=6$

Parity bits required per word			
Design	8-bit word size register	16-bit word size register	32-bit word size register
Dual modular redundancy	$2\,x\,redundancy$	$2\,x\,redundancy$	$2\,x\,redundancy$
Triple modular redundancy	$3\,x\,redundancy$	$3\,x\,redundancy$	$3\,x\,redundancy$
Checksum with redundancy	$2\,x\,redundancy$	$2\,x\,redundancy$	$2\,x\,redundancy$
CRC with redundancy	$\log_2(max((word)mod\,3))=2$ $2\,x\,redundancy$	$\log_2(max((word)mod\,3))=2$ $2\,x\,redundancy$	$\log_2(max((word)mod\,3))=2$ $2\,x\,redundancy$
Single parity bits with redundancy	1 $2\,x\,redundancy$	1 $2\,x\,redundancy$	1 $2\,x\,redundancy$
Parity bits per 4 data bits with redundancy	$2\,x\,redundancy$	$2\,x\,redundancy$	$2\,x\,redundancy$
Hamming code (with redundancy)	$\log_2(length(word))+1=\log_2(8)+1=4$ $2\,x\,redundancy$	$\log_2(length(word))+1=\log_2(16)+1=5$ $2\,x\,redundancy$	$\log_2(length(word))+1=\log_2(32)+1=6$ $2\,x\,redundancy$

Parity bits required			
Design	8-bit word size register	16-bit word size register	32-bit word size register
Hamming code (with autocorrection)	per word: $\log_2(\mathit{length}(\mathit{word}))+1=\log_2(8)+1=4$	per word: $\log_2(\mathit{length}(\mathit{word}))+1=\log_2(16)+1=5$	per word: $\log_2(\mathit{length}(\mathit{word}))+1=\log_2(32)+1=6$
CRC per row and column	per row of the register file matrix: $\log_2(\mathit{max}((\mathit{word})\,\mathit{mod}\,3))=2$ per column of the register file matrix: $\log_2(\mathit{max}((\mathit{word})\,\mathit{mod}\,3))=2$	per row of the register file matrix: $\log_2(\mathit{max}((\mathit{word})\,\mathit{mod}\,3))=2$ per column of the register file matrix: $\log_2(\mathit{max}((\mathit{word})\,\mathit{mod}\,3))=2$	per row of the register file matrix: $\log_2(\mathit{max}((\mathit{word})\,\mathit{mod}\,3))=2$ per column of the register file matrix: $\log_2(\mathit{max}((\mathit{word})\,\mathit{mod}\,3))=2$
Checksum per row and column	per row of the register file matrix: $\log_2(\mathit{max}((\sum(\mathit{word\ bits}))\,\mathit{mod}\,3))=2$ per column of the register file matrix: $\log_2(\mathit{max}((\sum(\mathit{word\ bits}))\,\mathit{mod}\,3))=2$	per row of the register file matrix: $\log_2(\mathit{max}((\sum(\mathit{word\ bits}))\,\mathit{mod}\,3))=2$ per column of the register file matrix: $\log_2(\mathit{max}((\sum(\mathit{word\ bits}))\,\mathit{mod}\,3))=2$	per row of the register file matrix: $\log_2(\mathit{max}((\sum(\mathit{word\ bits}))\,\mathit{mod}\,3))=2$ per column of the register file matrix: $\log_2(\mathit{max}((\sum(\mathit{word\ bits}))\,\mathit{mod}\,3))=2$
Single parity bits per row and column	per row of the register file matrix: 1 per column of the register file matrix: 1	per row of the register file matrix: 1 per column of the register file matrix: 1	per row of the register file matrix: 1 per column of the register file matrix: 1

Results

A standalone register was also designed, its operation was simulated using SimVision and its design was converted into a netlist for each of the three word sizes designed (8-bit, 16-bit and 32-bit registers), in order to compare the area and power this standalone register would require with the area and power the designs of registers incorporating logical countermeasures need. The different designs are compared for the area and total power they require, as well as their leakage power. These values are provided for all the designs in the following tables. Area is given in nm^2 and power in nW and were calculated from the simulated designs and their corresponding netlists.

8-bit word registers			
Design	Calculated area required (nm^2)	Total power required (nW)	Leakage power (nW)
Standalone	11034.427	56455.076	371.040
Checksum	25720.065	77941.364	901.832
CRC	29579.768	79178.382	931.008
Single parity bits	18505.897	65153.759	658.399
Parity bits per 4 data bits	19896.699	74598.619	722.256
Hamming code (implementation A without autocorrection)	23405.463	86578.844	851.169
Hamming code (implementation B without autocorrection)	22590.917	84057.214	814.660
Dual modular redundancy	20289.291	97188.570	901.532
Triple modular redundancy	39408.240	146179.847	1429.015
Checksum with redundancy	50363.230	137719.726	1859.709
CRC with redundancy	57361.109	139878.298	1910.180
Single parity bits with redundancy	34878.749	117811.588	1235.940
Parity bits per 4 data bits with redundancy	40760.180	124803.832	1395.309
Hamming code (implementation A with redundancy)	52654.168	157819.940	1889.062
Hamming code (implementation B with redundancy)	48126.940	148306.334	1677.320
Hamming code (implementation A with autocorrection with a single error register)	50060.182	114477.340	1510.346
Hamming code (implementation A with autocorrection with multiple error registers)	54640.166	118192.169	1720.333
Hamming code (implementation B with autocorrection with multiple error registers)	46971.133	110454.218	1532.423

8-bit word registers			
Design	Calculated area required (nm^2)	Total power required (nW)	Leakage power (nW)
Hamming code (implementation A with autocorrection with multiple error registers and additional parity checks)	59813.577	116632.672	1749.808
Hamming code (implementation B with autocorrection with multiple error registers and additional parity checks)	51895.278	111452.690	1610.407
CRC per row and column	65354.980	128537.985	1939.574
Checksum per row and column	51131.210	101275.493	1762.310
Single parity bits per row and column	34540.448	91006.419	1158.243
CRC per row and column with additional parity checks	95216.890	154852.788	2848.715
Checksum per row and column with additional parity checks	70639.079	112412.762	2476.984
Single parity bits per row and column with additional parity checks	41027.099	99469.737	1440.915

16-bit word registers			
Design	Calculated area required (nm^2)	Total power required (nW)	Leakage power (nW)
Standalone	20486.535	107253.636	722.830
Checksum	48588.783	137216.917	1673.089
CRC	52273.544	147297.248	1727.161
Single parity bits	33898.485	130133.057	1231.775
Parity bits per 4 data bits	37577.702	136847.257	1407.841
Hamming code (implementation A without autocorrection)	42485.215	152474.130	1608.441
Hamming code (implementation B without autocorrection)	41799.786	143966.959	1546.148
Dual modular redundancy	39785.404	190226.773	1712.852
Triple modular redundancy	73703.621	293339.137	2813.953
Checksum with redundancy	92964.439	258371.689	3290.836
CRC with redundancy	99531.705	259658.584	3300.305
Single parity bits with redundancy	62842.780	217914.079	2316.196
Parity bits per 4 data bits with redundancy	86191.307	289536.930	2963.752
Hamming code (implementation A with redundancy)	97908.831	263197.071	3538.628

16-bit word registers			
Design	Calculated area required (nm^2)	Total power required (nW)	Leakage power (nW)
Hamming code (implementation B with redundancy)	88892.684	270435.747	3014.919
Hamming code (implementation A with autocorrection with a single error register)	82368.020	185316.047	2426.931
Hamming code (implementation A with autocorrection with multiple error registers)	91288.176	193958.846	2876.387
Hamming code (implementation B with autocorrection with multiple error registers)	81042.443	181564.618	2699.134
Hamming code (implementation A with autocorrection with multiple error registers and additional parity checks)	99757.293	187693.065	2953.549
Hamming code (implementation B with autocorrection with multiple error registers and additional parity checks)	89254.810	174694.578	2729.587
CRC per row and column	125566.718	253768.145	3665.551
Checksum per row and column	98101.656	184923.372	3330.666
Single parity bits per row and column	63471.738	164170.505	2219.904
CRC per row and column with additional parity checks	181100.292	273299.495	5431.319
Checksum per row and column with additional parity checks	137518.087	207459.336	4648.513
Single parity bits per row and column with additional parity checks	76091.522	177516.079	2743.609

32-bit word registers			
Design	Calculated area required (nm^2)	Total power required (nW)	Leakage power (nW)
Standalone	39347.239	212296.697	1434.144
Checksum	89328.458	266535.91	3183.279
CRC	100677.777	283058.418	3329.407
Single parity bits	64540.038	226974.017	2383.822
Parity bits per 4 data bits	72685.486	271577.575	2769.138
Hamming code (implementation A without autocorrection)	79270.144	290685.666	3075.767

32-bit word registers			
Design	Calculated area required (nm^2)	Total power required (nW)	Leakage power (nW)
Hamming code (implementation B without autocorrection)	108083.947	586053.446	3609.538
Dual modular redundancy	78423.834	376894.919	3487.897
Triple modular redundancy	158447.485	584489.063	5841.126
Checksum with redundancy	167521.823	470189.170	6069.810
CRC with redundancy	190754.034	468414.957	6329.054
Single parity bits with redundancy	119376.180	410984.785	4416.277
Parity bits per 4 data bits with redundancy	215968.501	638064.836	7051.320
Hamming code (implementation A with redundancy)	198449.309	486624.239	7189.487
Hamming code (implementation B with redundancy)	242845.564	1149957.098	7580.475
Hamming code (implementation A with autocorrection with a single error register)	142311.803	356193.806	4322.043
Hamming code (implementation A with autocorrection with multiple error registers)	165312.173	359962.894	5387.824
Hamming code (implementation B with autocorrection with multiple error registers)	154206.869	317280.001	4998.600
Hamming code (implementation A with autocorrection with multiple error registers and additional parity checks)	184693.331	372881.952	5543.351
Hamming code (implementation B with autocorrection with multiple error registers and additional parity checks)	229581.463	842459.877	6693.994
CRC per row and column	250683.288	517216.748	7352.287
Checksum per row and column	189813.413	361021.675	6395.311
Single parity bits per row and column	123102.802	319827.148	4298.594
CRC per row and column with additional parity checks	361263.443	539587.15	10868.867
Checksum per row and column with additional parity checks	259077.016	439114.957	8895.482
Single parity bits per row and column with additional parity checks	147033.705	330602.675	5240.641

Additionally, the following graphs show how the word size affects the area and power required by each design. The first graph (image 11) demonstrates how much area each design was calculated to require using columns in order to illustrate how the size of the register's word affects the area needed by each particular design. The second graph (image 12) serves the same function for the total power the implementation of each design would require, showing again how the register's word size would affect it, while the third graph (image 13) does the same thing for the leakage power of each design.

These three graphs can also help us visualise the amounts of area (image 11) and power (image 12) that each design would require and its leakage power (image 13) and, therefore, easily identify how these values could be subsequently classified in order to be able to evaluate the overall cost of each design.

The three graphs following them are line plots of the same data in order to more clearly demonstrate how the different designs compare with each other over different register word sizes. The fourth graph (image 14) demonstrates the difference in the area required by each design for each register word size with an emphasis on how that value increases in comparison to the other designs, while the fifth graph (image 15) serves the same purpose for the estimated total power consumption. Finally, the sixth graph (image 16) shows the difference in the power leakage each design would have in comparison to the others as the register's word size increases.

These three additional graphs provide a visualisation of how much effect differences in the size of the register's word can have on the overall area (image 14) and power (image 15) required by a particular design and its leakage power (image 16) and, therefore, allow us to easily deduce in which exact class a particular design should really be placed regarding these classifiers when we subsequently evaluate its overall cost.

Calculated area required (nm²) for each design

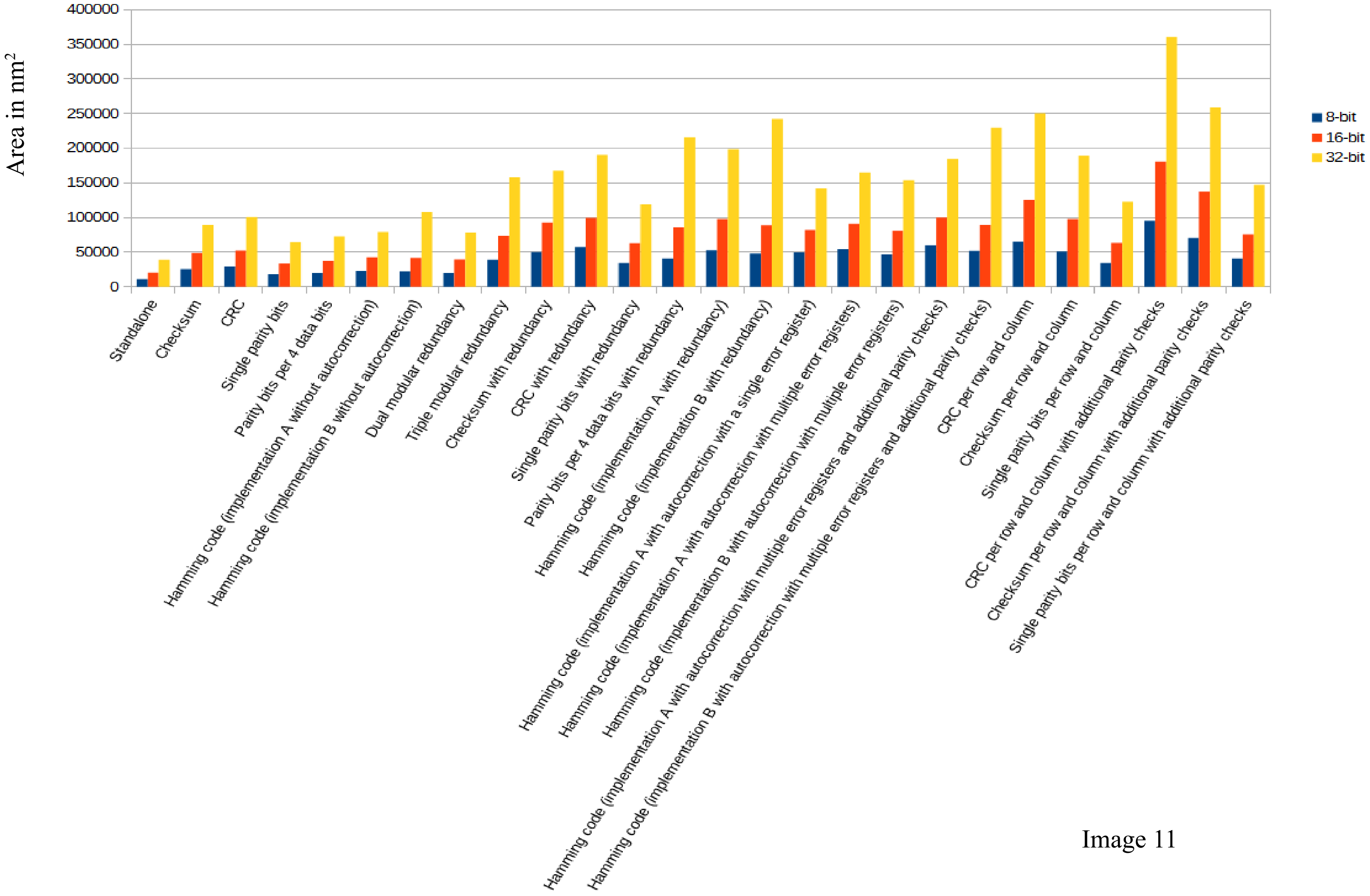


Image 11

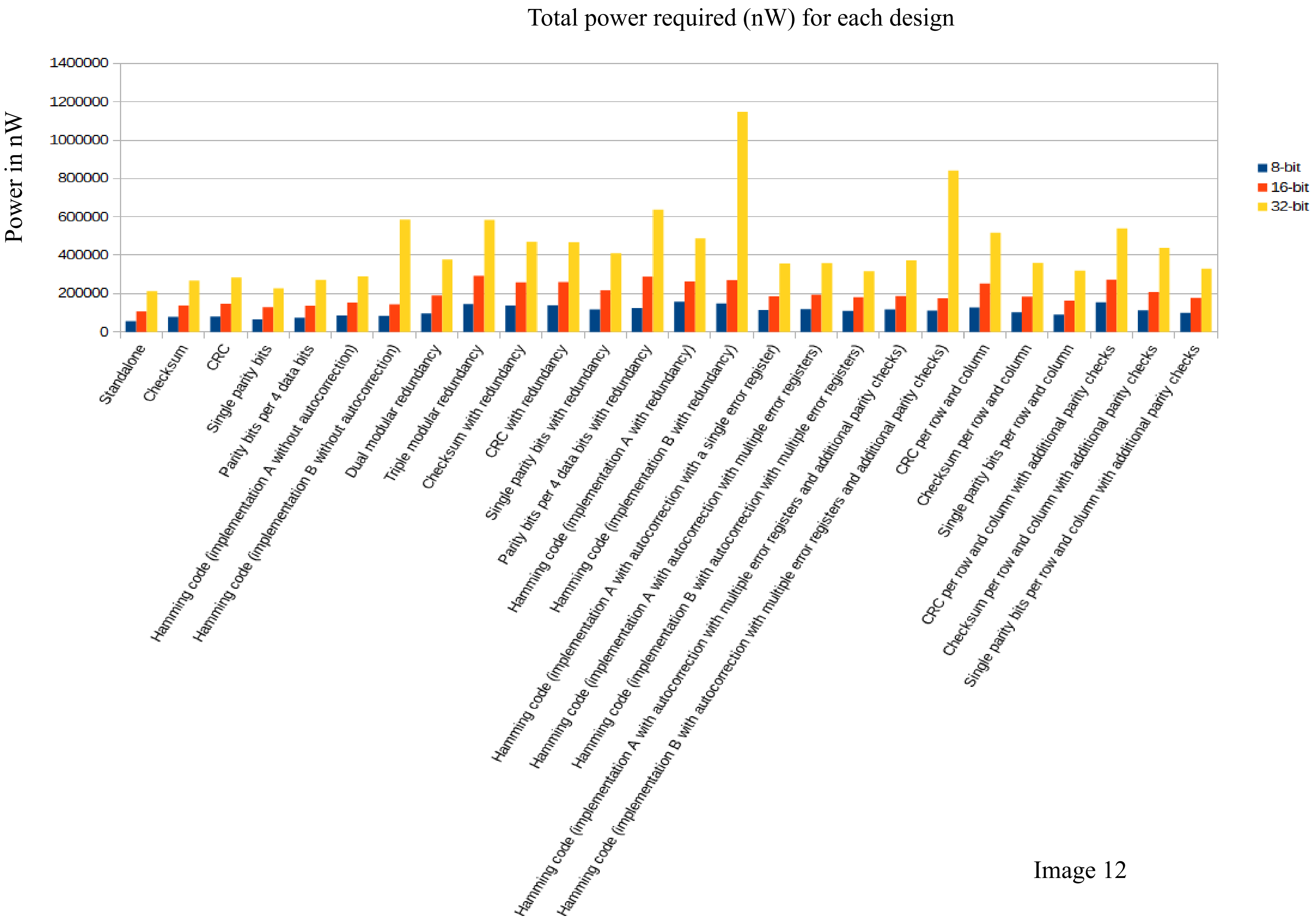

Total power required (nW) for each design
Power in nW
1400000
1200000
1000000
800000
600000
400000
200000
0
8-bit
16-bit
32-bit
Standalone
Checksum
CRC
Single parity bits
Parity bits per 4 data bits
Hamming code (implementation A without autocorrection)
Hamming code (implementation B without autocorrection)
Dual modular redundancy
Triple modular redundancy
Checksum with redundancy
CRC with redundancy
Single parity bits with redundancy
Parity bits per 4 data bits with redundancy
Hamming code (implementation A with redundancy)
Hamming code (implementation B with redundancy)
Hamming code (implementation A with autocorrection with a single error register)
Hamming code (implementation A with autocorrection with multiple error registers)
Hamming code (implementation B with autocorrection with multiple error registers)
Hamming code (implementation A with autocorrection with multiple error registers and additional parity checks)
Hamming code (implementation B with autocorrection with multiple error registers and additional parity checks)
CRC per row and column
Checksum per row and column
Single parity bits per row and column
CRC per row and column with additional parity checks
Checksum per row and column with additional parity checks
Single parity bits per row and column with additional parity checks

Image 12

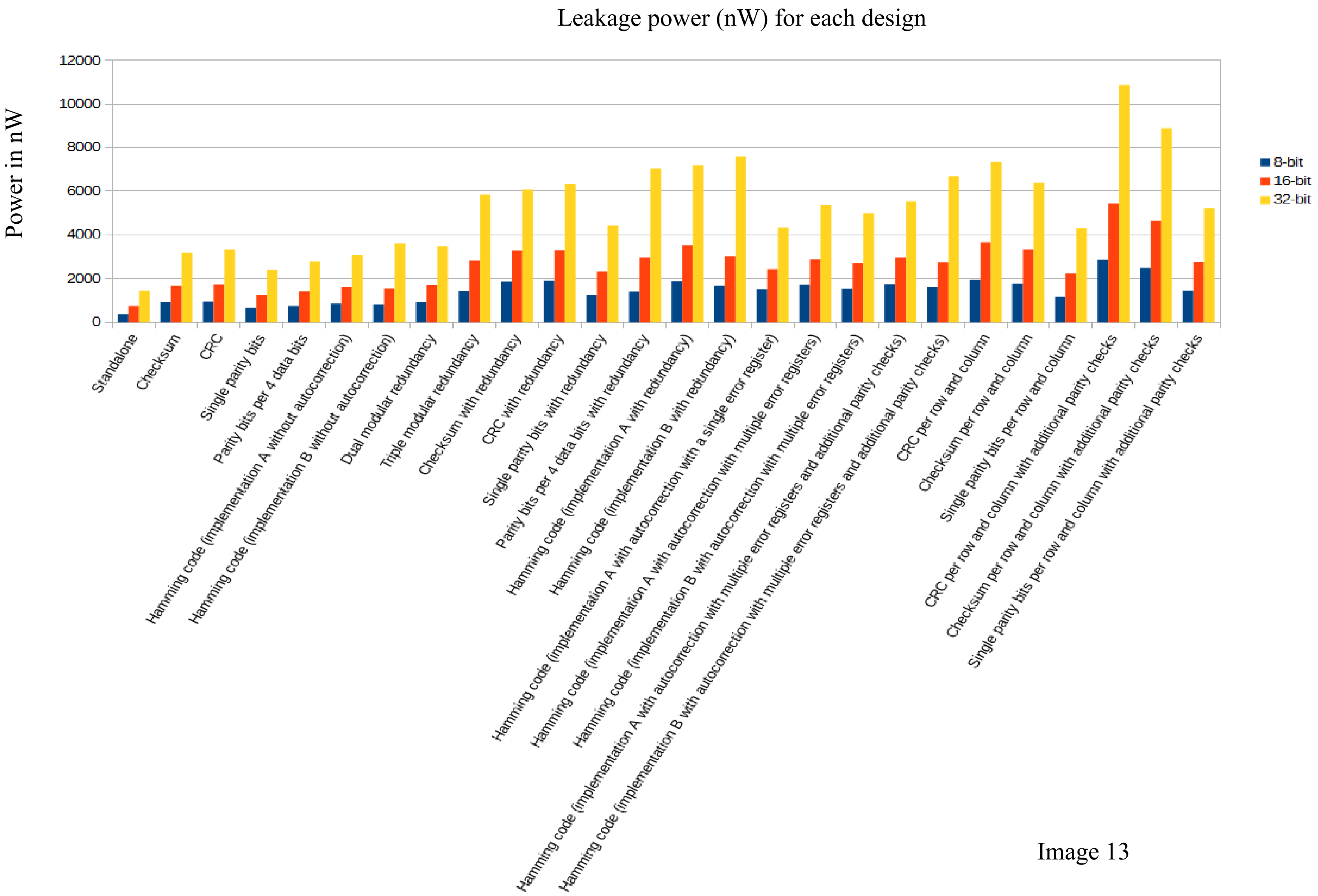

Leakage power (nW) for each design
Power in nW
12000
10000
8000
6000
4000
2000
0
8-bit
16-bit
32-bit
Standalone
Checksum
CRC
Single parity bits
Parity bits per 4 data bits
Hamming code (implementation A without autocorrection)
Hamming code (implementation B without autocorrection)
Dual modular redundancy
Triple modular redundancy
Checksum with redundancy
CRC with redundancy
Single parity bits with redundancy
Parity bits per 4 data bits with redundancy
Hamming code (implementation A with redundancy)
Hamming code (implementation B with redundancy)
Hamming code (implementation A with autocorrection with a single error register)
Hamming code (implementation A with autocorrection with multiple error registers)
Hamming code (implementation B with autocorrection with multiple error registers)
Hamming code (implementation A with autocorrection with multiple error registers and additional parity checks)
Hamming code (implementation B with autocorrection with multiple error registers and additional parity checks)
CRC per row and column
Checksum per row and column
Single parity bits per row and column
CRC per row and column with additional parity checks
Checksum per row and column with additional parity checks
Single parity bits per row and column with additional parity checks

Image 13

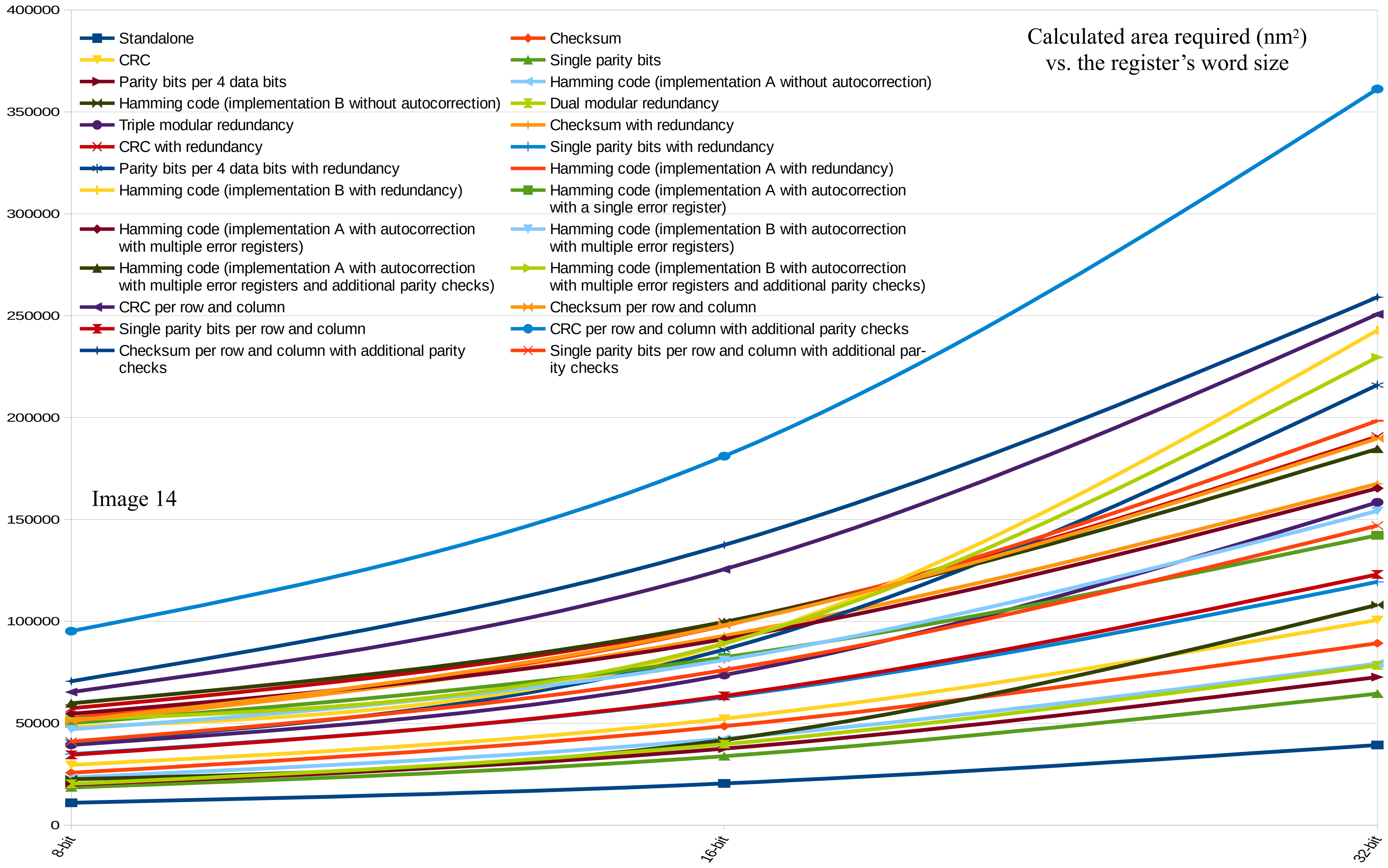
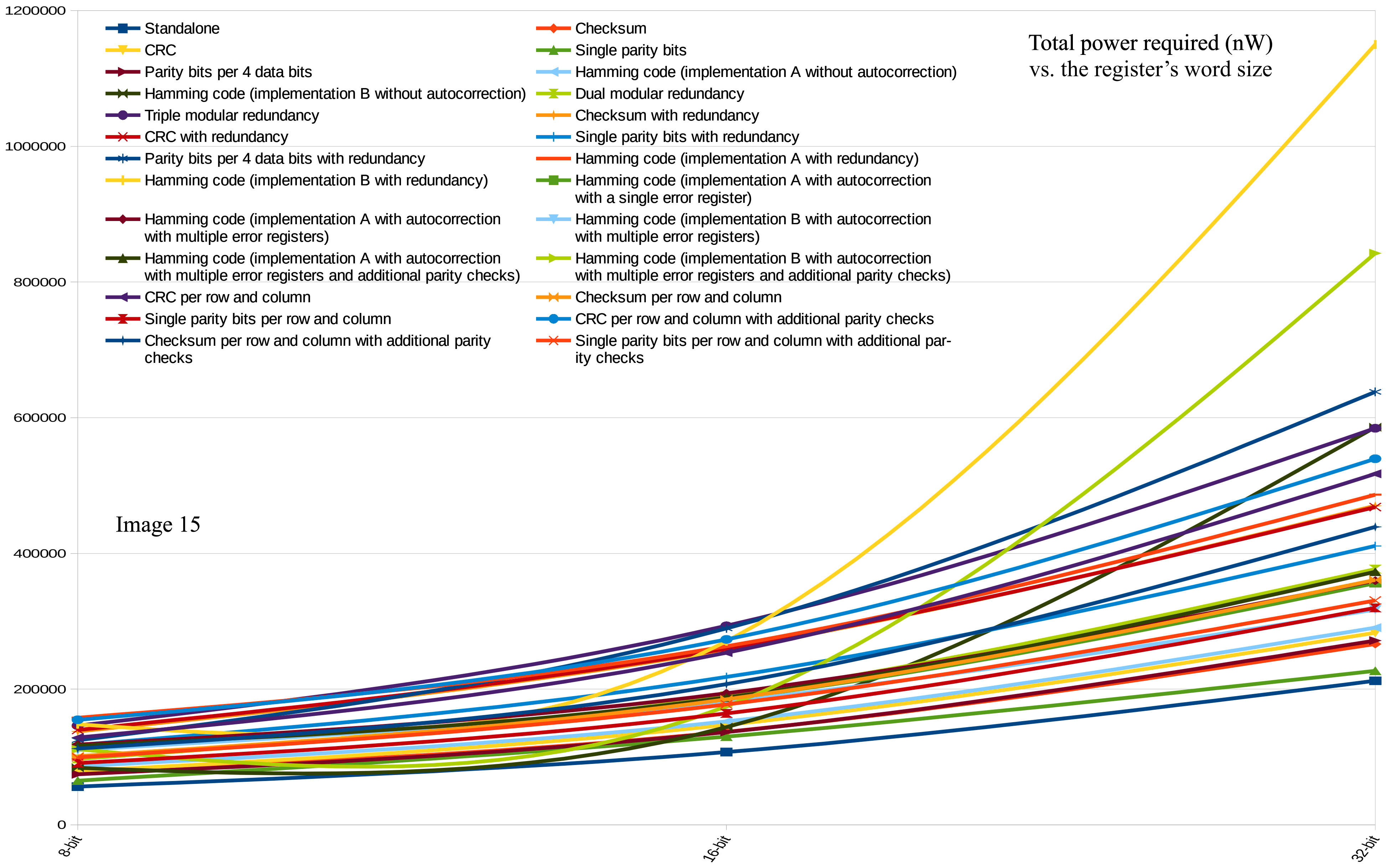

Calculated area required (nm²)
vs. the register's word size
400000
350000
300000
250000
200000
150000
100000
50000
0
8-bit
16-bit
32-bit
Standalone
Checksum
CRC
Single parity bits
Parity bits per 4 data bits
Hamming code (implementation A without autocorrection)
Hamming code (implementation B without autocorrection)
Dual modular redundancy
Triple modular redundancy
Checksum with redundancy
CRC with redundancy
Single parity bits with redundancy
Parity bits per 4 data bits with redundancy
Hamming code (implementation A with redundancy)
Hamming code (implementation B with redundancy)
Hamming code (implementation A with autocorrection with a single error register)
Hamming code (implementation A with autocorrection with multiple error registers)
Hamming code (implementation B with autocorrection with multiple error registers)
Hamming code (implementation A with autocorrection with multiple error registers and additional parity checks)
Hamming code (implementation B with autocorrection with multiple error registers and additional parity checks)
CRC per row and column
Checksum per row and column
Single parity bits per row and column
CRC per row and column with additional parity checks
Checksum per row and column with additional parity checks
Single parity bits per row and column with additional parity checks

Image 14



Total power required (nW)
vs. the register’s word size

Image 15

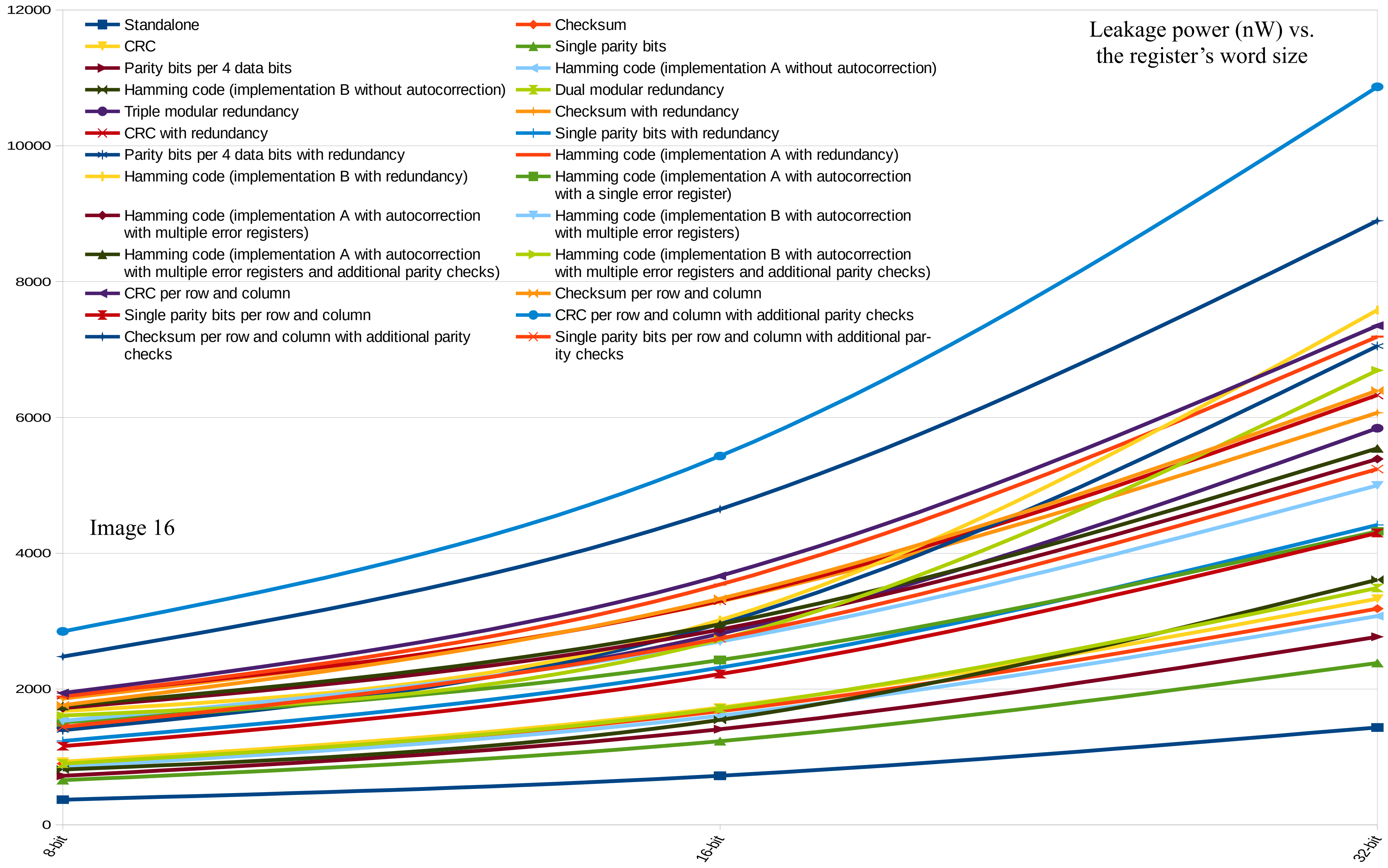


Image 16

Leakage power (nW) vs. the register’s word size

Since it can easily be deduced from the previous tables and graphs that some particular designs would require significantly more area and power, or have far more leakage, than others, it would make sense to make an initial assessment of them based on these criteria in relation to the perceived level of protection they may offer. Therefore, and taking the values of the standalone register as base values, we can come up with a table, in which we compare the additional area overhead, the total power overhead and the leakage each design would have if they were to be implemented, in comparison to the level of protection they provide against optical fault injection, in order to come up with some initial assessment of their benefits. Therefore, in order to construct such a table, we should first define its set of values, regarding these criteria, which is done on the following table.

Additional area overhead (in terms of the area of the standalone register)	Total power overhead (in comparison to the standalone register)	Leakage power (regarding the leakage of the standalone register as low and in comparison to it)	Level of protection	Assessment value
less than or around 1x	less than or around 1x	significantly less than 1 x	no protection	none
1x - 2x	1x - 2x	around 1x to 2x	can be neutralised by an attack on a single spot	low
2x - 3x	2x - 3x	2x - 3x	can be neutralised by attacks on several spots on the same register	medium
3x - 4x	3x - 4x	3x - 4x	can be neutralised by attacks on the same single spot on each of multiple registers	high
4x - 5x	4x - 5x	4x - 5x	can be neutralised by attacks on several same spots on each of multiple registers or on an amount of very specific single spots on the same register	very high
more than 5x	more than 5x	more than 5x	can be neutralised by attacks on an amount of very specific single spots on each of multiple registers or on an amount of several very specific spots on the same register	extremely high

Additionally, we award one point for each security level and deduct one for each additional overhead or leakage level. Obviously, the more (positive) points a design gets, the better it performs regarding the balance its implementation would offer between cost and security. However, as all the designs come at certain costs, they end up having a negative score, which is meant to exactly demonstrate the fact that the protection offered comes at an additional cost.

Therefore, the most optimal design could be considered to be the one that offers a certain chosen level of protection at the least cost possible, i.e. the one that provides at least the required level of protection, while also having the highest overall score. On the other hand, one could choose to decide the most optimal design based on a specific category of cost, or on cost in general, in which case the most optimal design would be the one having at most the particular level of cost chosen and the least negative overall score.

Furthermore, a decision to assume that the standalone register's leakage is low but not insignificant was made in order to signify that even a normal register that has to unavoidably be integrated in a chip comes with an additional cost, even if we consider its area and power requirements to be insignificantly small and totally justifiable. In this way, we can more evidently demonstrate that every additional element that is placed in an integrated circuit will always have an inherent cost. In general, and based on these facts, we cannot actually identify a most optimal solution in our assessment, but rather the least bad one.

Design	Additional area overhead	Total power overhead	Leakage power	Level of protection	Evaluation points
Standalone	none	none	low	none	-1
Checksum	medium	low	medium	low/medium	-3.5
CRC	medium	low	medium	medium	-3
Single parity bits	low	low	low	low	-2
Parity bits per 4 data bits	low	low	low	low	-2
Hamming code (implementation A without autocorrection)	low	low	medium	medium	-2
Hamming code (implementation B without autocorrection)	medium	medium	medium	medium	-4
Dual modular redundancy	low	low	medium	medium/high	-1.5
Triple modular redundancy	high	medium	high	high/very high	-4.5
Checksum with redundancy	very high	medium	very high	very high	-6
CRC with redundancy	very high	medium	very high	very high	-6
Single parity bits with redundancy	medium	low	medium	high	-2
Parity bits per 4 data bits with redundancy	extremely high	high	very high	high	-9
Hamming code (implementation A with redundancy)	very high	medium	very high	very high	-6
Hamming code (implementation B with redundancy)	extremely high	extremely high	extremely high	very high	-11

Design	Additional area overhead	Total power overhead	Leakage power	Level of protection	Evaluation points
Hamming code (implementation A with autocorrection with a single error register)	high	low	medium	high	-3
Hamming code (implementation A with autocorrection with multiple error registers)	very high	low	high	high	-5
Hamming code (implementation B with autocorrection with multiple error registers)	high	low	high	high	-4
Hamming code (implementation A with autocorrection with multiple error registers and additional parity checks)	very high	low	high	high	-5
Hamming code (implementation B with autocorrection with multiple error registers and additional parity checks)	extremely high	very high	very high	high	-10
CRC per row and column	extremely high	medium	very high	extremely high	-6
Checksum per row and column	very high	low	very high	extremely high	-4
Single parity bits per row and column	high	low	medium	very high	-2
CRC per row and column with additional parity checks	extremely high	medium	extremely high	extremely high	-7
Checksum per row and column with additional parity checks	extremely high	medium	extremely high	extremely high	-7
Single parity bits per row and column with additional parity checks	high	low	high	very high	-3

As already mentioned above, we see that we end up with negative assessment points, exactly because of the inherent costs that any design would introduce, especially when taking into account the overheads that can be caused by security features. Additionally, it must also be mentioned that one cannot base the selection of a security element to be integrated in a chip only on the last column of this table, as every chip has different requirements and specifications. Rather, the last column serves as an indicator of which design would be the least costly among an initial selection made on the requirements and specifications of a particular chip, regarding its area, power consumption, leakage and security requirements. Furthermore, we can easily observe the tradeoff between the protection a design offers and its area and power overheads and costs.

Finally, it must also be noted that the performance of most designs is highly dependent on their specific characteristics, such as, for example, the divisors chosen, in the cases of CRC and checksum. Such specific characteristics may significantly influence the security-cost tradeoff balance, as they may offer more protection but require further additional register cells for more parity bits, resulting in higher area and power overheads and costs. In this document, as noted before, we selected the most economic case that would adequately work, in order to keep our designs as simple as possible. Additionally, the decision to check each word of the register for faults on every clock cycle is also costly, but provides better results and seems unavoidable in order to be able to provide a certain level of security, given the fact that a fault injection attack can last as much as a single clock cycle, or perhaps even less. This once again demonstrates how fragile the balance between cost and security is and how much it depends on the level of acceptable risk.

Additional remarks on the evaluated countermeasures

In general, it would not be easy to precisely assess the benefits and costs associated with each category of countermeasures, even if we were to make prototypes of their implementations integrated in a smart card chip. An obvious reason for this would be that certain specific characteristics would naturally depend on the integrated circuit model we chose to use. Different categories of chips allow for different EDA and manufacturing processes, or may use different basic components to build an element. This is why it was deemed necessary to keep the comparison between the countermeasures on a more abstract level, while trying to come with an evaluation scheme that would be broad enough to address a large number of the different characteristics related to costs and benefits associated with the proposed countermeasures.

The difficulties in comparing the various proposed countermeasures with each other become evident in the case of photodiodes, which initially seemed quite ineffective as a countermeasure against optical fault injections, but when compared under a different scenario, they seemed to perform better than the already employed solution of phototransistors. Therefore, and based on the performance of all the countermeasures examined, what seems to be a credible conclusion remark is that there is not one single countermeasure that always performs optimally and is the best one among the others. What has actually been observed is that different countermeasures perform better or worse under different scenarios. Furthermore, depending on the specific criteria used, such as the exact size of a register's word, different countermeasures may provide more cost-efficient protection than others, but this relation can change as the criteria change, for example, with the register's word size increasing.

However, it must be noted that we can come up with general conclusions regarding the performance of the examined countermeasures, based on the observed results and their mean performance. Nevertheless, again, these conclusions hold true only for the CMOS 90 nm low power manufacturing process. A clear indicator of this limitation is the fact that a lot of the physical countermeasures examined in this document are currently not compatible with this process, and therefore their mass production would have an enormously unbearable cost, but they may be compatible with future manufacturing processes. The same, of course, holds true for the EDA process and any other factors which may hinder the implementation and integration of a specific countermeasure in an already existing or future smart card chip.

Additionally, regarding the risk and the value of an investment on a particular countermeasure, one must note that by just comparing the cost and effectiveness of physical countermeasures to those of logical countermeasures, we cannot really find out which category of countermeasures would be better to invest in. In order to be able to address this issue in a thorough manner, we would need to know the weight different stakeholders related to secure smart card chips place on each phase of the development and production of the final product. For example, while a physical countermeasure may seem cheaper to implement and integrate in this process, it may actually mean a significant, or even unbearable, cost for the company designing and producing these chips, when taking in regard the current state of the market and the relevant competition. On the other hand, perhaps, using more secure software, which implements re-transmissions of data and additional security checks, could actually be affordable for the end customer, although it may seem more expensive than the integration of an additional physical countermeasure.

This could be caused by the fact that the end customer may be required to actually invest less money in the production of relevant code, because it is already in possession of the relevant resources, while the company that actually designs or produces the chips may not currently possess enough resources to implement the required additional physical countermeasure and keep the chip's price at a competitive price. In general, the decision of which entity will bear the additional costs related to security is not as easy as it may seem, because it is dependent not only to the cost of each additional security feature, but also to the current state of the relevant market, the level of competition and the actual needs of the customers.

In general, a company may choose not to move to a newer, more secure technology, even when the present technology scheme being employed is compromised, if it estimates that the additional costs related to this change would potentially be more than its financial benefits. Instead, it may try to use internal checks of the back-end system and its databases or real-life checks of the overall system's components in order to keep potential damages at an acceptable level and, thus, not bear the costs of purchasing smart card chips incorporating more secure technology at that time.

Therefore, although security may not be far more expensive, a customer may choose to invest into a more vulnerable chip and try to keep risks at an acceptable level through basic countermeasures of its own, as these may be cheaper in comparison to more high-end security. Especially if these countermeasures are inherent in the overall system, then the incentives of investing to more secure hardware implementations may be particularly low. In this case, the incentives to the producers to develop more secure solutions are also low and the level of security offered is bound to reach a certain level balancing between demand and offer. Only if an inexpensive way is found to completely expose and compromise the existing solutions, the incentives for innovation and more security will rise again, resulting in another, higher, level of optimal security.

Therefore, we can conclude that security, cost and risk are interdependent, and at the same time, the cost of attacks, their effectiveness and the security demanded by the market are also interdependent. Taking this into account, it is really difficult to assess whether an additional security feature which seems to require a moderately inexpensive development will actually be invested upon, or whether a really secure implementation will not find many willing investors, without actually knowing the current state of the market and its associated level of risk. Thus, it is important to try to further assess how security, cost and risk affect the previously examined countermeasures, in general. Such an effort to assess a number of different countermeasures in regards to their security, cost and risk culminated in the following tables.

Countermeasures	Physical countermeasures					
	Photodiodes	Phototransistors	Photoresistors	Shielding	Voltage / temperature sensors	Other structures & techniques
Cost	Relatively low, as they are based on already integrated components	None, as they are already being used (Relatively low, as they were based on pre-existing components)	Depends on exact implementation (Medium if using current technology and materials, high otherwise)	None, as it is already being used (Medium if changes need to be made in the implementation or placement)	High, due to incompatibility	High, due to incompatibilities
Research & Development	Relatively low	Relatively low (possible innovations)	Relatively high, due to technology constraints	None to low (possible innovations)	Medium, to fit into requirements	High, to fit into requirements
Production and testing	Relatively low	Relatively low (possible innovations)	Medium to high, due to incompatibilities	None to medium (if changes in placement or implementation need to be made)	High, due to probable incompatibility	High, due to incompatibilities
Level of protection provided	Low to medium, depending on the scenario	Medium to high, depending on the scenario	Low to medium, due to placement constraints (not yet fully certain)	High, due to its nature	Uncertain, due to precision	Low to high (depends on the design)
Can protect against						
Frontside attacks	Yes	Yes	Yes	Yes	Potentially yes	Depends on the design
Backside attacks	Yes	Yes	Uncertain	Potentially yes	Potentially yes	Depends on the design

Countermeasures	Physical countermeasures					
	Photodiodes	Phototransistors	Photoresistors	Shielding	Voltage / temperature sensors	Other structures & techniques
Range of wavelengths (nm)	400 – 1100	400 – 1100	Depends on the materials used	Depends on the materials used	Depends on the material used	Depends on the materials used and the design
Uniqueness of the proposal	Low	Low	Medium	Low	High	High
Reliability						
Expected rate of false positives	Low	Low	Low to medium	Does not apply	Medium to high	Depends on the design (Low to medium)
Affected by temperature changes	Yes	Yes	Yes	No	Yes	Depends on the design (Mostly yes)
Technical feasibility & impact						
Compatibility with current manufacturing processes	Yes	Yes	Depends on the implementation	Yes (Changes could lead to incompatibilities)	Depends on the implementation	Mostly no
Impact on design flow (EDA)	None	None	Low to medium	Insignificant	Medium	High
Impact on software requirements	None	None	None	None	Low	None or low
Risk of a successful attack						

Countermeasures	Physical countermeasures					
	Photodiodes	Phototransistors	Photoresistors	Shielding	Voltage / temperature sensors	Other structures & techniques
Expertise required for a successful attack	Relatively high	Relatively high	Relatively high	Medium	High	High
Equipment required for a successful attack	Expert	Expert	Expert	Depends on implementation (Standard or expert)	Expert	Expert

Countermeasures	Parity & Redundancy			Others			
	Parity checks	Redundancy	Supermajority required and/or repeated checks performed	Signal or operation repetitions	Dummy data (on dummy clock cycles)	Asynchronous logic with or without multiple data rails	Unique design per wafer
Cost	Very low (Additional area and power overhead and leakage)	Very low (Additional area and power overhead and leakage)	Low (Additional area and power overhead and leakage)	Low, already implemented in some cases	Low, already implemented in some cases	High, due to incompatibility	High, because of constant development, which prevents economies of scale to develop
Research & Development	Very low (based on algorithmic research)	Very low (based on systems research)	Very low (based on algorithmic research)	Very low (based on systems research)	Very low (based on systems research)	High, due to incompatibility	High, because of constant development
Production and testing	Very low	Very low	Low	Medium	Medium	Low to medium	High, because of constant development

Countermeasures	Parity & Redundancy			Others			
	Parity checks	Redundancy	Supermajority required and/or repeated checks performed	Signal or operation repetitions	Dummy data (on dummy clock cycles)	Asynchronous logic with or without multiple data rails	Unique design per wafer
Level of protection provided	Depends on the algorithm	Depends on number of copies	Depends on the algorithm	Depends on the implementation	Very low (can only make meaningful attacks more difficult)	Depends on the implementation	Very low (can only make meaningful attacks more difficult)
Can protect against							
Frontside attacks	Yes	Yes	Yes	Yes	No	Yes	No
Backside attacks	Yes	Yes	Yes	Yes	No	Yes	No
Range of wavelengths (nm)	Any (as long as the algorithm is still functional)	Any (as long as one or more copies are unaffected)	Any (as long as the algorithm is still functional)	Any (as long as the system is still functional)	None	Any (as long as the algorithm is functional)	None
Uniqueness of the proposal	Depends on the algorithm being used	Relatively low	Low	Medium	Medium	Medium	High
Reliability							
Expected rate of false positives	Very low	Very low	Extremely low	Low	Does not apply	Low	Does not apply
Affected by temperature changes	No	No	No	No	No	No, as long as the system remains functional	No

Countermeasures	Parity & Redundancy			Others			
	Parity checks	Redundancy	Supermajority required and/or repeated checks performed	Signal or operation repetitions	Dummy data (on dummy clock cycles)	Asynchronous logic with or without multiple data rails	Unique design per wafer
Technical feasibility & impact							
Compatibility with current manufacturing processes	Yes	Yes	Yes	Yes	Yes	No	No
Impact on design flow (EDA)	None	None	None	Low	Low	High	Medium to high
Impact on software requirements	Depends on implementation	Depends on implementation	Depends on implementation	Medium	Medium to high	High	Low
Risk of a successful attack							
Expertise required for a successful attack	Relatively low	Relatively low	Low	Medium	Medium to high	High	Medium to high
Equipment required for a successful attack	Standard	Standard	Standard	Standard to expert	Standard to expert	Standard to expert	Expert

From these tables, we can see that different countermeasures offer advantages in different areas while requiring different costs and posing different degrees of risk. Therefore, while some of them may easily be dismissed at present as too costly or not secure enough, it may difficult to select the most optimal among the others, and such a decision will heavily depend on which criteria each selector considers more important. Additionally, in the future, some of the countermeasures whose implementation today seems unfavourable, may become easier to implement or their development may be requiring a lot less money than now. Finally, it is obvious that incompatibilities strongly hinder the entry to the market in our case, which makes the market for secure chips a highly specialised sector with high barriers against new competitors.

As a final conclusion, it seems that, at present, photodetectors (and especially phototransistors), shielding, (repeated) parity checks, and redundant elements, as well as the repetition of logical signals, functions and operations, are among the best ways to prevent, detect and act against a successful optical fault injection attack. This clearly indicates, however, that only multi-level security can complete secure a system against fault injection attacks, as each one of these countermeasures is vulnerable on its own and can be successfully neutralised. However, the development, integration and production of so many countermeasures will lead into enormous costs. It is therefore upon the different stakeholders of the development, production and usage of a particular secure smart card chip to agree on a specific combination of countermeasures to be employed, in such a way that the balance between security, cost and risk is favourable to them.

Chapter 4: Future work proposals and final remarks and conclusions

As already mentioned before, optical fault injection attacks are almost always performed with a laser beam nowadays. Both the front and the back side of a chip may be targeted, but usually the back side is preferred, because usually the top side contains layers of metal and other shielding material which can block light, while the back side doesn't. Lasers producing different wavelengths have to be used for each side in order for the light to reach the transistors.

To this end, light with a wavelength between 400 and 800 nm is effective in causing fault injections through unshielded front sides and beams with a wavelength between 800 and 1100 are able to cause fault injection by penetrating through thinned or normal back sides. Wavelengths below 400 nm will be absorbed or reflected, while silicon is transparent to wavelengths above 1100. However, due to two-photon absorption, the ability of silicon to absorb two photons at the same time, it may also be vulnerable to attacks with light having a wavelength between 1200 and 1600 nm, or even more.

In order to protect against such an attack or at least detect it and take adequate action, different countermeasures can be employed such as light detectors and other physical and logical ways and means. However, even though light detectors have been widely employed for this purpose, they do face considerable issues regarding their effectiveness, cost and operation. Therefore, it made sense to consider the alternative and/or complementary solutions that can enhance the detection and protection capabilities of a smart card chip against optical fault injections.

For this reason, we made a distinction between the available solutions and classified them into logical and physical. Nevertheless, it is possible that the identified countermeasures be further classified into physical-level, transistor-level and circuit-level ones, with the addition of other relevant solutions concerning EDA and software. However, we mostly focused on hardware-related countermeasures rather than software-based ones in this essay, considering only relatively simple software-related techniques, which concern the logic of a circuit, such as parity checks, while not examining solutions related to the concept of secure code or related techniques.

As none of the identified countermeasures seemed to be completely devoid of hindrances and shortcomings, it was evident that we needed to comprehensibly compare and evaluate their characteristics, merits and costs in relation to protecting against optical fault injection attacks, in order to be able to decide the most optimal implementations and/or combinations of them. To sum up, all countermeasures had to somehow be assessed regarding the level of protection they offer against optical fault injection attacks and the required costs for their implementation and integration and other potential shortcomings.

We discussed and examined different physical solutions and how they could enhance the chip's protection against optical fault injection. However, we also quickly identified their potential shortcomings, with the most important of them being the severe incompatibility of most of them with the current EDA and/or manufacturing processes. This led into their potential implementation being extremely expensive and therefore totally infeasible. Nevertheless, we examined how different photodetectors (photodiodes) than the ones currently integrated in smart card chips (phototransistors) may provide better results than them under different scenarios regarding their placement in the chip.

Moreover, we also discussed ways to harden the logic of the chip by implementing parity checks and/or using redundant elements. We quickly identified that the implementation of such features and their integration to the chip's design could potentially require outstanding additional area and power and, thus, significant additional costs. However, by examining how these features could potentially increase the chip's protection against fault injections, we found out that may provide significant protection against fault injections, as they may only be neutralised by simultaneous and/or repeated injections of faults at different spots. This would significantly increase the difficulty and cost of such attacks, as very precise timing and equipment would be required.

Additionally, we also considered other logical countermeasures, but again their implementation was estimated to be infeasible, due to incompatibilities with the current EDA and/or manufacturing processes as well as significant additional area and power requirements. In some cases, their implementation would be feasible, but highly dependent on the exact implementation of the whole chip, especially in the case of data transmission checks, or are deeply related to the software designed to be used in the smart card, such as the repetition of software operations, and therefore could not be examined in detail.

Therefore, it was not possible to fully examine all potential countermeasures mentioned in this document, not only because some of them are completely incompatible with the current manufacturing and EDA processes, but also because we chose to focus on the hardware solutions, rather than on the potential software ones. This was a conscious decision, however, as the software of a smart card is customised to fit the needs of the client, rather than being the same for every smart card family. Therefore, security that is intended to be provided by the manufacturer of the card, or, at least, the designer of the smart card chip, has to be primarily based on the hardware components of the chip, and software may only be playing a secondary role on the protection that the manufacturer can guarantee.

Therefore, a study on how the client may choose to enhance security and how this may be efficiently reflected into the smart card certification and accreditation requirements would provide helpful insight on further ways to increase the level of protection a smart card can offer to its owner. Security is a very far-reaching and multi-level concept that should be approached from every possible side, starting from the security provided by a design and the secrecy maintained by the company producing such a design and reaching to the end user's ability to keep certain confidential information private. For example, it can be easily understood that even the most secure design ever possible to come up with will become a complete nuisance if it ever gets public or if the end user gives away his PIN (Personal Identification Number) or some other similar information.

Software

Thus, it is essential to study not only how software used in a smart card chip can increase the protection level such a chip has against optical fault injection attacks, but also what further customisations have to be done on such a chip in order not only not to expose any additional security features incorporated in it, but also to adequately employ them to their full extent. In this light, it is essential that the manufacturer reaches an agreement with the client not only on what security level should be provided on a specific chip but also how issues related to security should be handled through the software used in such a chip. Essentially, it should be investigated and agreed upon what should happen when an attack has been successfully detected and what the detection threshold should be. For example, in order to avoid destroying a smart card due to a single false positive, in some cases, it would make more sense to allow for a certain number of detections per a specific time period before taking such an action.

Although such issues may seem trivial, they play a really significant role over whether a smart card product will be successful and widely used. One has to only imagine a credit card that would destroy itself every time sunlight fell upon it, or a secure pass that could be blanked under the same circumstances, to understand the seriousness of such decisions as described before. Additionally, security checks can be performed through software in order to determine whether the chip functions correctly or there has been an attack, for example faults have been injected. Such a way to use software in order to detect fault injections would be having secure operations be repeated and then comparing their results.

Furthermore, the propagation of errors can be prevented by using the (super)majority result of such repeated operations. However, such repetitions obviously also result in the introduction of an overhead in terms of memory space, processing time and robustness,[3] as well as leading into more complex code. It is therefore necessary to assess the costs and benefits of such software, especially in comparison to the already examined hardware countermeasures. Moreover, as the broad spectrum of the notion of security has already been demonstrated, software countermeasures could potentially provide better results being combined with hardware countermeasures than either category of countermeasures alone.

Finally, it should be noted that the software being used must itself be secured against bugs and vulnerabilities which may render the other countermeasures useless or provide vulnerable points which may be used by potential attackers. For example, if, due to some unintentional software bug that was overlooked during testing and verification, secure data are also reaching registers that were not planned to be used for secure purposes or go through other elements of the chip that they are not supposed to, then attackers may successfully attack these structures and gain access on the data. In this way, although the secure part of the chip will be functioning as intended to, a software vulnerability may null the overall protection provided to secure data by the chip. This has already been observed,[3][4] and efforts have been made to address it by either using distributed storage in order to keep any breaches contained,[3] or used double verifications in the form of repeated operations.[4]

Hardware

Regarding the hardware countermeasures proposed and examined, it must be noted that a large number of them could not be implemented because of their incompatibility with the current manufacturing process used for smart card chips, the 90 nm low power manufacturing process. This, of course, means that, in the future, further research regarding these solutions needs to be conducted in order to establish the extent of their potential merits towards preventing and/or detecting optical fault injection attacks.

Furthermore, the fact that most of the data used in this document are based on expert opinions and related calculations, simulations and estimations must be taken into account, as the actual implementation of the examined countermeasures could help consolidate the results and conclusions of this study. For example, certain issues regarding the most effective implementation of a photodiode remain open, as the current manufacturing process supports both the regular n-p junction diode structure and a gated p-channel(substrate)-n diode one. The latter case could perhaps allow for more biasing, but this needs to be established through further research.

Additionally, although the current manufacturing process allows for silicon resistors to be fabricated, it has not been established whether such resistors could function as photoresistors given the physical properties of silicon to act as a resistor. Moreover, even if we could build adequate photoresistors using the current manufacturing process, their efficiency in detecting backside optical fault injection attacks through the underlying substrate is highly questionable and would need to be thoroughly examined.

One of the biggest hindrances related to research and development of efficient countermeasures in a chip designing/manufacturing company is that due to the enormous costs associated with any changes in the manufacturing process, any countermeasure that is incompatible with it is considered effectively unusable. However, in order to merit changing the manufacturing process, an incompatible component needs to be proven extremely useful for the company. Therefore, there is a vicious cycle as no incompatible component can ever get implement and tested in such a degree as to prove that it really is useful enough to merits making changing in the manufacturing process, exactly on the basis that an incompatible component won’t be worth the time and other resources spent on its research and development.

Therefore, any research on deep doped wells, the use of graphene and structures such as DEPFETs as countermeasures against optical fault injection attacks will have to be conducted as future academic research. The same seems to be true for any countermeasures that would be based on changing the process treatment of the substrate during production, such as shaping it into particular shapes or adding shielding layers on it. It goes without saying that such ideas as using unique designs per wafer or chip batch are rather rarely examined, let alone implemented, by manufacturers as they introduce significant costs both in the field of research and development and in the field of production and testing. Thus, again, any future research on such ideas will probably need to be conducted by academic researchers.

On the contrary, research on the interaction between photodiodes or phototransistors and their nearby regular transistors may also be conducted by manufacturers, although they seem to just prefer to use the standard solution of insulating one element from the other, thus being unable to study the interaction between a targetted transistor and a nearby photodetector in case there is no insulation separating them. Additional ideas which could merit from further research but have rather been overlooked by the industry due to their cost include using redundant flip-flops and/or light detectors made from different materials which may be able to detect a broader wavelength spectrum than silicon or have a larger detection span.

Structures such as deep doped wells require specific research in order to determine how their shape may affect the detection abilities of different photodetectors, as demonstrated in the previous chapter for photodiodes and phototransistors. This research would have to take into account the physics regarding the interaction between such structures, their charge and light. More interesting future research could be conducted regarding the properties of such structures made of different materials, such as germanium for example.

The interaction of multiple deep doped wells with each other and light shed on them needs to also be examined in the future, as well as different ways to integrate graphene in photodetectors or on the whole chip. Additionally, the use of reflective or absorbing materials on the chip's surface and substrate, which could be incorporated in its shielding should also be considered regarding the feasibility of its implementation and its potential benefits and costs. The same is true regarding the use of photosensitive materials in the shielding in order to detect potential attacks.

Another interesting idea that needs to be examined is whether the chip's components can be shaped in such a way as to drive a portion of the light shed onto them towards the detection area of any nearby photodetectors, or at least reflect most of it. Furthermore, the idea of using the outer tiers of 3D chips to protect sensitive components place on inner tiers really seems to warrant further examination and research, especially if photodetectors are to be placed in such a way as to surround these sensitive components.

Additional research is also needed in order to identify potentially more resistant components to optical fault injections or improve already existing ones, such as MRAMs, and make them more compatible to the current manufacturing process. Moreover, voltage and temperature sensors also need to be fine tuned in order to be able to detect extremely small voltage or temperature fluctuations that may be caused by optical fault injection attacks and distinguish them from normally occurring ones. Such a process distinguishing between normally occurring fluctuations and externally caused ones could be based on their frequency, as the circuit is clocked.

Finally, it seems that the most difficult obstacle to overcome is the incompatibility of the proposed countermeasures with the manufacturing process, so any future research should also really take this into account and focus specifically on addressing this issue, by studying how each suggested component can actually be implemented. An increased focus must be placed on which materials can be used and how they may interact or interfere with the rest of the chip's circuitry, and how these factors may affect the detection area span and threshold.

Additional care must be taken to ensure that any protection features are not easily identifiable and either do not depend on a continuous and/or constant power supply or that their power supply is very difficult to interfere with. In order to achieve this effect, it is suggested that non-standard chip designs be used in order to achieve some additional level of complexity and obscurity in the final design to be produced. Finally, if a significant amount of area or power is required for the integration of a design or if its integration leads to a notable increase in the power leakage of the chip, this could make this chip more susceptible to side channel attacks based on power analysis or examinations of its structure and layout.

Logic

Regarding the logic of the circuit, it is important to stress that again actual implementation of the designed countermeasures will help consolidate and validate the results and conclusions of this study regarding them. Additionally, the impact of using parity checks and redundancy on registers of different sizes than the ones already examined needs to be assessed, as well as the effects of optimising the EDA process for the integration of such features in the already existing chip designs.

Moreover, the use of static code elements rather than dynamic ones could significantly affect the implementation of the designed countermeasures and, therefore, its exact effects must be thoroughly examined. Furthermore, the use of a single parity check for the whole register file, rather than a separate one for each individual word of it could potentially provide more cost-efficient solutions and should therefore also be explored in the future. Likewise, the use of different divisors in such already designed or future countermeasures as the ones incorporating CRC, checksums or other similar schemes should also be considered and thoroughly examined in future research and development regarding them.

Furthermore, the exact relation between such security features and the customised software installed on smart card chips as well as the actions to be taken in case of a successful detection of a fault injection attack need to be more clearly defined, in order to be able to calculate the required costs more adequately. Moreover, a variety of other algorithms and schemes could be examined for their efficiency and overall suitability to serve as detectors for fault injections, in order to assess how these other categories of logic checks and redundant elements may perform, in comparison to the already examined ones.

Additionally, the exact probability of non-detection of successful attacks should be thoroughly calculated based on the exact design, code and implementation of each logical countermeasure, because it is heavily dependent not only on the general scheme and design being used, but also on such details as the exact characteristic of the algorithm being used, how often checks are programmed to be performed, how long a clock cycle may last, when and how often a reset signal is scheduled to be transmitted and other relevant factors.

The importance of adjustment and optimisation of each algorithm and its relevant circuit design to the exact characteristics and properties of the chip in which it is to be integrated and its related manufacturing process must also be explicitly stressed, because it can lead to significant cost reductions both in expenses related to research and development and in production and testing costs. Additionally, it can also affect the probability of failing to detect successful attacks depending on how well the security features are combined with the rest of the chip's circuitry and how efficiently the whole integrated circuit is coordinated.

Essentially, it must also be noted that the presented results on the benefits and costs of the examined parity checks and redundant elements are based on simulations, estimations and calculations and therefore depending on how they are implemented, they may lead into similar but not exactly same results. This is important considering that different designs for the overall chip can significantly affect the area and power overheads that new security features to be integrated in that chip may introduce. For example, the location of the chip's registers and their distance from the other components will inadvertently affect how the security checks can be performed and what components may need to be introduced to the chip for their implementation.

Additionally, there are different ways to implement the same parity and redundancy schemes and especially when parity checks are combined with redundancy, it is very important to adapt the related designed circuits in order to collaborate with the operation and functionality of the other elements and components and, thus, be better integrated in the chip. Moreover, customised flip-flop designs should be produced in order to create more robust and fault-resistant implementations of such elements.

Furthermore, because logical countermeasures can be defeated by being bypassed or blocked, their availability and integrity has to be ensured and protected. Therefore, further research is required on how secure data and signal transmissions can be achieved, by detecting (and possibly correcting) errors in communication. Of course, as such mechanisms would highly depend on the specific design and implementation of both the logical countermeasures and the circuit in which these are integrated, a thorough study must be performed on this topic.

Moreover, an effort must be also made to eliminate any possible ways to block the execution of logical checks based on parity or redundancy by exploiting software vulnerabilities which may bypass or hold the execution of such checks and the use of their results. To this end, using an unstable internal clock could perhaps also help against the exploitation of any software vulnerabilities by making it more difficult to distinguish between the different commands being executed.

In addition, an unstable clock will also make it difficult to precisely determine the time at which particular internal operations of the chip are expected to be performed and, thus, would significantly hinder the ability of adversaries to correctly determine the timing of their attacks on specific chip components. In turn, this would substantially decrease the probability of such attacks being successful.

Watchdog timers can also be used in order to control whether the chip is operating correctly and their possible integration in smart card chips should therefore be examined, as well as the benefits and costs of their use. Furthermore, checks can be performed in order to determine whether the input of certain elements, such as flip-flops is the intended one. For example, it can be checked whether the reset signal is really transmitted to all flip-flops when required. Thus, additional research could determine whether this could be done with the value stored in a flip-flop in order to find out whether it remains correct or a fault has been injected, or whether we can avoid the propagation of faults by performing checks on the output of a flip-flop.

Additionally, dummy clock cycles when operations on dummy data are performed can be used to prevent attackers from gaining access on real data, or in this case, have them inject faults in dummy data, whose results could also lead to quick detection of attacks. Thus, further research on the overhead such changes may be introducing and their effectiveness towards detecting fault injection attacks must be conducted.

Furthermore, as already noted before, customised flip-flops could lead in more fault-resistant designs, so, in general different non-standard elements should be examined on how resistant they may be to optical fault injection and what the costs of their potential replacing standard components of a chip would actually be. Moreover, asynchronous logic can be used to prevent successful fault injections and their propagation. Nevertheless, it is currently highly incompatible both with the EDA process and the present manufacturing process, and introduces significant overheads in terms of area and power requirements. Therefore, ways of making it more compatible and decreasing its area and power requirements should be examined in detail, in order to assess its potential benefits and considerably decrease the perceived costs that prevent its introduction at present.

Finally, further research needs to be conducted on ways to produce different unique designs for each batch of manufactured chips in a cost-efficient ways. It is really important to at least change the general design used for the placement of a chip's components for different families of integrated circuits, otherwise a successful optical injection attack on a single smart card chip may render whole families of different chips vulnerable to being compromised. Furthermore, this field of research is especially interesting as it can help study the balance between the levels of cost and (acceptable) risk as well as the tradeoff between security and cost.

Final remarks and conclusions

Indisputably, an intriguing relation exists between cost, security and risk. In accordance with this relation, the level of acceptable risk is highly dependent on the balance between security and cost. As costs increase, more risk can potentially be considered as acceptable, while the cost of security decreases, a lot less risk will need to be accepted. Therefore, ideally, we aim for minimum costs and maximum security, in order to minimise the risks of investment and the potential losses. However, with inadequate investment it is highly improbable that security costs can be reduced in the smart card market sector, as this sector is particularly dependent on economies of scale, high barriers to enter it and quite often time-related advantages.

In this context, attacks act as a necessary and sufficient drive for the development of the market, because they form a cycle in conjunction with security enhancements, effectively causing the development of each other. Although attacks are highly undesirable, the fact that they keep being improved and becoming more and more sophisticated leads, inevitably, to more research being conducted on countermeasures and new security features.

This, in turn, leads into time-dependent economies of scale, which drive high-end security costs down, both for the manufacturers and the customers, as long as the level of protection offered is high enough to secure the device (a smart card chip) from being compromised at a cost sufficiently low for an attacker. Due to the fact that the market sector of attacks can also be taking advantage of economies of scale, if the costs of performing a successful attack fall below a certain level, then the current security level will not be adequate anymore. In fact, the level of cost required to perform a successful attack highly determines the level of acceptable risk, which then, in succession, defines the balance between the level of desirable security and affordable cost.

Therefore, it is really important not only to analyse the mechanisms behind already existing attack schemes, but also to take notice of the recent developments of new attack vectors and techniques, in order to develop smart card chips that are resistant not only against current threats, but also against future ways and means of breaching a chip's security. It is also evident that new attacks may require the development of new countermeasures, which serves as an apparent incentive for further research and innovation in this field.

New methods of compromising a smart card chip challenge the current equilibrium in the relevant computer security sectors inevitably leading in continuous research and development of new defences against such attacks. Regarding optical fault injection, the phenomena of interference and two-photon absorption can undoubtedly serve as new attack vectors and, therefore, could play a major role in the development of new attack schemes. Therefore, it is really critical to develop relevant effective new countermeasures to prevent such attacks from being successful.

Nevertheless, current countermeasures may prove vulnerable to already existing attack schemes and methods and therefore other solutions need to be examined in order to come up with better-performing and more cost-efficient countermeasures against optical fault injection attacks. To this end, a large number of different countermeasures was examined in order to determine how these countermeasures would fare in comparison to security features already employed against optical fault injection attacks, such as phototransistors.

In general, although most physical countermeasures under consideration proved to be incompatible with the current manufacturing process, these ideas could lead into the emergence of more effective solutions in the future. Additionally, it was proven that under specific circumstances photodiodes can achieve better detection of optical fault injections than phototransistors, therefore signifying that particular areas of the chip would be better protected by them. In conclusion, different countermeasures may be more suitable for different areas of a smart card chip.

Furthermore, by designing and testing a wide variety of different logical countermeasures based on redundancy and parity checks, we reached the conclusion that their implementation may require an extensive range of different levels of cost, while providing a higher or lower level of detection (and protection) against fault injection attacks. Therefore, not only it was proven that logical countermeasures constitute a viable, and potentially less expensive, alternative to physical countermeasures, but also that they can fit the varying needs and demands of the different stakeholders involved in the smart card security market sector.

It is worth mentioning here that as attacks become more and more focused, targeting particular areas of the chip and very specific components in them, it appears that a need arises for proportionately focused countermeasures, appropriately customised to meet the needs for protection identified in their region of placement on the chip.

However, although a lot of different ideas, schemes, mechanisms and designs were examined, none of them proved to be able to guarantee perfect detection rates of optical fault injection attacks or provide full protection against them. This, of course, was only to be expected, as it has been noted and demonstrated that there will never exist a complete system, or even a smart card, that will provide perfect security that can defend against everything and everybody,[3] because if enough resources, time and effort is devoted to an attack, every system can be breached or manipulated in the end.[3]

Nevertheless, the designs and ideas discussed in this document provided encouraging results regarding both the level of protection they can offer and the relevant costs required for their implementation and integration in a smart card chip, as they can offer an adequate level of protection against optical fault injections at a fair level of additional cost. In particular it was shown that different physical countermeasures than the one currently employed may perform better under certain circumstances and that a wide range of logical countermeasures exists which can allow a certain degree of flexibility in the decision of choosing the most optimal one for integration based on criteria of cost, security and the relation between them, i.e. the level of risk considered acceptable.

However, they still would need to be implemented and tested further in order to consolidate these results and draw more accurate conclusions regarding their merits and costs. Nonetheless, as every attack also comes at a specific cost, if the costs of a probable enough attack are made high enough to surpass its benefits through the integration and use of such countermeasures as the ones examined, then their integration in the final product could be considered highly justified.

Of course, again, the costs of these countermeasures should, in turn, not exceed their merits and the final decision over whether their integration in the final product is justified is likely to be based on the level of acceptable risk set by the different stakeholders involved, the designer, the manufacturer and the client companies. Therefore, it is up to these companies to decide which countermeasures would be considered optimal for integration in the final product, the smart card chip. Nevertheless, we must once again stress that, in any case, a compatible combination of different categories of countermeasures seems bound to be the most optimal solution regarding cost and protection, because it will be able to offer different aspects of security by combining suboptimal low-cost countermeasures.

However, although a number of the innovations regarding some of the chip's physical components and their logic presented in this document seem to offer promising results, further research needs to be conducted in order to examine in detail whether they may offer significant advantages in comparison to the countermeasures currently employed against optical fault injection. Apart from the implementation and testing needed, further research is required in order to define how they may best interact with the software used in smart cards, or whether this software can be used as an adequate countermeasure.

Furthermore, software and generally all components that interact in a smart card system also need to be secured against containing potential vulnerabilities that may lead in the overall system being compromised. Finally, other new security mechanisms and solutions need to be developed in the future in order to counter future ways and means of performing optical fault injections, as attacks constantly evolve, and therefore, the only effective way of protection is constant innovation in the field of security.

It is also clearly important to determine what actions need to be taken in case an attack is detected, in order to fully estimate the required costs for each solution. In order to get correct results, we also need to set commonly accepted criteria for the assessment of the different countermeasures, based on security and cost requirements as they are established in the semiconductors industry and the related procedures of certification and accreditation. In this way, we could perhaps recognise how effective design-time security analysis may be against optical fault injection attacks.

Finally, another potential objective would be to identify and propose a novel and more effective countermeasure against optical fault injection attacks and compare its perceived characteristics against all the already mentioned ones in order to evaluate it.

References

[1] Europol, "Situation Report – Payment Card Fraud in the European Union: Perspective of Law Enforcement Agencies", 2012.
https://www.europol.europa.eu/sites/default/files/1public_full_20_sept.pdf

[2] E. Sperling & M. Wagner, "Expert Interview: NXP On Security", Semiconductor Engineering, 2014.
http://semiengineering.com/expert-interview-nxp-on-security/

[3] W. Rankl & W. Effing, "Smart Card Handbook", 4th edition, chap. 16: "Smart Card Security", Wiley, 2010.
http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470743670.html

[4] J. G. J. van Woudenberg, M. F. Witteman & F. Menarini, "Practical optical fault injection on secure microcontrollers", Proceedings of the 8th International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2011), pp. 91-99, 2011.
https://www.riscure.com/benzine/documents/fdtc2011vanwoudenberg.pdf

[5] S. Mangard, E. Oswald, T. Popp, "Power analysis attacks: Revealing the secrets of smart cards", Springer, 2007.
http://link.springer.com/book/10.1007%2F978-0-387-38162-6

[6] A. Iqbal, "Security Threats in Integrated Circuits", MIT System Design and Management blog, 2013.
http://sdm-blog.mit.edu/2013/10/security-threats-in-integrated-circuits.html

[7] S. P. Skorobogatov & R. J. Anderson, "Optical Fault Induction Attacks", Proceedings of the 4th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2002), Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 2523, pp. 2-12, Springer, 2003.
http://www.cl.cam.ac.uk/~sps32/ches02-optofault.pdf
http://link.springer.com/chapter/10.1007%2F3-540-36400-5_2

[8] Wikipedia, "Radiation hardening", 25 April 2014.
https://en.wikipedia.org/wiki/Radiation_hardening

[9] W. Rankl & W. Effing, "Smart Card Handbook", 4th edition, chap. 5: "Smart Card Microcontrollers", Wiley, 2010.
http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470743670.html

[10] S. Higginbotham, "Forget servers; One day Facebook, Google and other web giants will make their own custom chips", Gigaom, 2013.
http://gigaom.com/2013/07/06/forget-servers-one-day-facebook-google-and-other-web-giants-will-make-their-own-custom-chips/

[11] L. Peters, J. Griffin & R. Skinner, "Cost effective IC manufacturing", ch. 2: "*Cost per wafer*", Integrated Circuit Engineering Corp., 1995
http://smithsonianchips.si.edu/ice/cd/CEICM/SECTION2.pdf

[12] L. Martin, “Using Semiconductor Failure Analysis Tools for Security Analysis”, FIPS Physical Security Workshop, 2005.
http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-3/physec/papers/physecpaper11.pdf

[13] Wikipedia, “Semiconductor fabrication plant”, 24 February 2014.
https://en.wikipedia.org/wiki/Semiconductor_fabrication_plant

[14] D. Art, M. O'Halloran & B. Butler, “Wafer fab construction cost analysis & cost reduction strategies: Applications of SEMATECH's future factory analysis methodology”, Proceedings of the IEEE/SEMI 1994 Advanced Semiconductor Manufacturing Conference and Workshop, pp. 16-21, 1994.
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=588160

[15] H. Föll, “Electronic Materials”, additional illustration chapter: “Costs of Chip Production” in ch. 5.4.1 “Money, Time, and Management”, Kiel University, 2013.
http://www.tf.uni-kiel.de/matwis/amat/elmat_en/kap_5/illustr/i5_4_2.html

[16] W. Rankl & W. Effing, “Smart Card Handbook”, 4th edition, chap. 5: “Smart Card Production”, Wiley, 2010.
http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470743670.html

[17] R. Anderson & M. Kuhn, “Tamper Resistance – a Cautionary Note”, Proceedings of the 2nd USENIX Workshop on Electronic Commerce, pp. 1-11, 1996.
http://static.usenix.org/publications/library/proceedings/ec96/full_papers/kuhn/

[18] G. de Koning Gans, J.-H. Hoepman & F. D. Garcia, “A Practical Attack on the MIFARE Classic”, CARDIS '08 Proceedings of the 8th IFIP WG 8.8/11.2 international conference on Smart Card Research and Advanced Applications, pp. 267-282, 2008.
http://www.cs.ru.nl/~flaviog/publications/Attack.MIFARE.pdf

[19] F. D. Garcia, G. de Koning Gans, R. Muijrers, P. van Rossum, R. Verdult, R. Wichers Schreur & B. Jacobs, “Dismantling MIFARE Classic”, Proceedings of the 13th European Symposium on Research in Computer Security: Computer Security (ESORICS '08), pp. 97-114, 2008.
http://www.cs.ru.nl/~flaviog/publications/Dismantling.Mifare.pdf

[20] F. D. Garcia, P. van Rossum, R. Verdult & R. Wichers Schreur, “Wirelessly Pickpocketing a Mifare Classic Card”, SP '09 Proceedings of the 2009 30th IEEE Symposium on Security and Privacy, pp. 3-15, 2009.
http://www.cs.ru.nl/~flaviog/publications/Pickpocketing.Mifare.pdf

[21] K. Nohl & H. Plötz, “Mifare: Little Security, Despite Obscurity”, 24th Chaos Communication Congress, 2008.
http://events.ccc.de/congress/2007/Fahrplan/events/2378.en.html

[22] K. Nohl, D. Evans & H. Plötz, “Reverse-Engineering a Cryptographic RFID Tag”, USENIX Security '08 Proceedings, pp. 185–193, 2008.
https://www.usenix.org/legacy/events/sec08/tech/nohl.html

[23] N. T. Courtois, K. Nohl & S. O'Neil, “Algebraic Attacks on the Crypto-1 Stream Cipher in MiFare Classic and Oyster Cards”, International Association for Cryptologic Research, 2008.

http://eprint.iacr.org/2008/166

[24] N. T. Courtois, “Conditional Multiple Differential Attack on MiFare Classic”, Eurocrypt 2009 rump session, 2009.
http://eurocrypt2009rump.cr.yp.to/7870fc6d38647a661145594ef0c33015.pdf

[25] N. T. Courtois, “The Dark Side of Security by Obscurity and Cloning MiFare Classic Rail and Building Passes, Anywhere, Anytime”, Proceedings of SECRYPT 2009 – International Conference on Security and Cryptography, pp. 331-338, 2009.
http://eprint.iacr.org/2009/137

[26] D. Oswald & C. Paar, “Breaking Mifare DESFire MF3ICD40: Power analysis and templates in the real world”, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 6917, pp. 207-222, 2011.
http://www.iacr.org/workshops/ches/ches2011/presentations/Session%205/CHES2011_Session5_1.pdf

[27] NXP Semiconductors Austria GmbH, “NXP celebrates 20th anniversary of MIFARE products”, 2014.
http://www.mifare.net/en/aboutmifare/news/nxp-celebrates-20th-anniversary-mifare-products/

[28] M. Karpovsky, K. J. Kulikowski & A. Taubin, “Robust protection against fault-injection attacks on smart cards implementing the advanced encryption standard”, Proceedings of the International Conference on Dependable Systems and Networks, pp. 93-101, 2004.
http://ieeexplore.ieee.org/xpl/login.jsp?arnumber=1311880

[29] O. Derouet, “Secure Smartcard Design against Laser Fault Injection”, 4th Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2007), 2007
http://conferenze.dei.polimi.it/FDTC07/Derouet_remaster.pdf

[30] Wikipedia, “Photocurrent”, 25 April 2014.
https://en.wikipedia.org/wiki/Photocurrent

[31] Wikipedia, “Photovoltaic effect”, 25 April 2014.
https://en.wikipedia.org/wiki/Photovoltaic_effect

[32] Wikipedia, “Lateral MOSFET”, 25 April 2014.
Licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license.
https://en.wikipedia.org/wiki/File:Lateral_mosfet.svg

[33] Wikipedia, “Laser-assisted device alteration”, 25 April 2014.
https://en.wikipedia.org/wiki/Laser-assisted_device_alteration

[34] Wikipedia, “MOSFET”, 25 April 2014.
https://en.wikipedia.org/wiki/MOSFET

[35] Wikipedia, “Latchup”, 25 April 2014.
http://en.wikipedia.org/wiki/Latchup

[36] Wikipedia, “CMOS”, 25 April 2014.

https://en.wikipedia.org/wiki/CMOS

[37] Wikipedia, “Flip-flop (electronics)”, 25 April 2014.
https://en.wikipedia.org/wiki/Flip-flop_(electronics)

[38] C. Roscian, J.-M. Dutertre & A. Tria, “Frontside laser fault injection on cryptosystems - Application to the AES' last round”, Proceedings of the 2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST 2013), pp. 119-124, 2013.
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6581576

[39] S. Skorobogatov, “Optical fault masking attacks”, Proceedings of the 7th International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2010), 2010.
https://www.cl.cam.ac.uk/~sps32/fdtc2010-masking.pdf
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5577358

[40] Olympus Microscopy Resource Center, “Interaction of Photons with Silicon”, 2012.
http://www.olympusmicro.com/primer/java/photomicrography/ccd/quantum/

[41] A. Richards, “Infrared Spectral Selection – It Begins with the Detector”, edu.photonics.com, 2014.
http://www.photonics.com/EDU/Handbook.aspx?AID=25132

[42] Wikipedia, “Silicon”, sect. “Electronics”, 25 April 2014.
https://en.wikipedia.org/wiki/Silicon#Electronics

[43] E. I. Cole Jr. & D. L. Barton, “Failure Analysis of Integrated Circuits: Tools and Techniques”, chap. 6: “Failure Site Isolation: Photon Emission Microscopy Optical/Electron Beam Techniques”, edit. L. C. Wagner, Springer, 1999.
http://link.springer.com/book/10.1007%2F978-1-4615-4919-2

[44] W. A. Moreno, F. J. Falquez & N. Saini, “Fault tolerant design validation through laser fault injection”, Proceedings of the IEEE International Caracas Conference on Devices, Circuits and Systems (ICCDCS), pp. 132-137, 1998.
http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=705820

[45] W. A. Moreno, J. R. Samson Jr. & F. J. Falquez, “Laser injection of soft faults for the validation of dependability design”, Journal of Universal Computer Science, vol. 5, iss. 10, pp. 712-729, 1999.
http://www.jucs.org/jucs_5_10/laser_injection_of_soft/Moreno_W_A.pdf

[46] É. Brier, D. Naccache, P. Q. Nguyen & M. Tibouchi, “Modulus fault attacks against RSA–CRT signatures”, Proceedings of the 13th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2011), Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 6917 LNCS, pp. 192-206, Springer, 2011.
http://link.springer.com/chapter/10.1007%2F978-3-642-23951-9_13

[47] Y. Monnet, M. Renaudin, R. Leveugle, C. Clavier & P. Moitrel, “Case study of a fault attack on asynchronous des crypto-processors”, Proceedings of the 3rd International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2006), Lecture Notes in Computer Science

(including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 4236 LNCS, pp. 88-97, Springer, 2006.
http://link.springer.com/chapter/10.1007%2F11889700_9

[48] M. Agoyan, J.-M. Dutertret, A.-P. Mirbahat, D. Naccache, A.-L. Ribottat & A. Tria, "Single-bit DFA using multiple-byte laser fault injection", Proceedings of the 10th IEEE International Conference on Technologies for Homeland Security (HST 2010), pp. 113-119, 2010.
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5655079

[49] M. Hutter, J.-M. Schmidt & T. Plos, "RFID and its vulnerability to faults", Proceedings of the 10th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2008), Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 5154 LNCS, pp. 363-379, Springer, 2008.
http://link.springer.com/chapter/10.1007%2F978-3-540-85053-3_23

[50] F. R. Palomo, J. M. Mogollón, J. Nápoles, H. Guzmán-Miranda, A. P. Vega-Leal, M. A. Aguirre, P. Moreno, C. Méndez, J. R. Vázquez De Aldana, "Pulsed laser SEU cross section measurement using coincidence detectors", IEEE Transactions on Nuclear Science, vol. 56, iss. 4, pp. 2001-2007, 2009.
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5204741

[51] V. Pouget, D. Lewis & P. Fouillat, "Time-resolved scanning of integrated circuits with a pulsed laser: Application to transient fault injection in an ADC", IEEE Transactions on Instrumentation and Measurement, vol. 53, iss. 4, pp. 1227-1231, 2004.
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1316010

[52] V. Pouget, P. Fouillat, & D. Lewis, "Radiation Effects on Embedded Systems", chap. "Using the SEEM software for laser SET testing and analysis", edit. R. Velazco, P. Fouillat, R. Reis, Springer, 2007.
http://link.springer.com/chapter/10.1007%2F978-1-4020-5646-8_12

[53] B. Niu, P. Pardy, J. Fortier, M. Ortega & T. Eiles, "Two-photon absorption laser assisted device alteration using continuous wave 1,340 nm laser", Journal of Materials Science: Materials in Electronics, vol. 22, iss. 10, pp. 1542-1552, Springer, 2011.
http://link.springer.com/article/10.1007%2Fs10854-011-0458-y

[54] G. Canivet, P. Maistri, R. Leveugle, J. Clédière, F. Valette & M. Renaudin, "Glitch and laser fault attacks onto a secure AES implementation on a SRAM-Based FPGA", Journal of Cryptology, vol. 24, iss. 2, pp. 247-268, Springer, 2011.
http://link.springer.com/article/10.1007%2Fs00145-010-9083-9

[55] M. Soucarros, J. Clédière, C. Dumas & P. Elbaz-Vincent, "Fault analysis and evaluation of a true random number generator embedded in a processor", Journal of Electronic Testing: Theory and Applications (JETTA), vol. 29, iss. 3, pp. 367-381, Springer, 2013.
http://link.springer.com/article/10.1007%2Fs10836-013-5356-1

[56] T. Kiyan, C. Brillert & C. Boit, "Timing analysis of scan design integrated circuits using stimulation by an infrared diode laser in externally triggered pulsing condition", Microelectronics Reliability, vol. 48, iss. 8-9, pp. 1327-1332, 2008.
http://www.sciencedirect.com/science/article/pii/S0026271408002199

[57] S. Petit, S. Rezgui, C. Carmichael, P. Fouillat & D. Lewis, “Investigation of SEU sensitivity of Xilinx Virtex II FPGA by pulsed laser fault injections”, Microelectronics Reliability, vol. 44, iss. 9-11 (spec. iss.), pp. 1709-1714, 2004.
http://www.sciencedirect.com/science/article/pii/S0026271404003014

[58] J. A. Rowlette & T. M. Eiles, “Near-IR laser-based "tuning" of 50 nm transistors”, 2003 IEEE Lasers and Electro-Optics Society Annual Meeting (LEOS) Annual Meeting Conference Proceedings, vol. 2, pp. 664-665, 2003.
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1252975

[59] J. A. Rowlette & T. M. Eiles, “Critical Timing Analysis in Microprocessors Using Near-IR Laser Assisted Device Alteration (LADA)”, Proceedings of the IEEE International Test Conference (TC) 2003, pp. 264-273, 2003.
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1270848

[60] N. Guibbaud, F. Miller, F. Moliere & A. Bougerol, “New combined approach for the evaluation of the soft-errors of complex ICs”, IEEE Transactions on Nuclear Science, vol. 60, iss. 4, pp. 2704-2711, 2013.
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6464613

[61] C. Roscian, A. Sarafianos, J.-M. Dutertre & A. Tria, “Fault model analysis of laser-induced faults in SRAM memory cells”, Proceedings of the 10th Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2013), pp. 89-98, 2013.
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6623559

[62] A. Sarafianos, M. Lisart, O. Gagliano, V. Serradeil, C. Roscian, J.-M. Dutertre & A. Tria, “Robustness improvement of an SRAM cell against laser-induced fault injection”, Proceedings of the 26th IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS 2013), pp. 149-154, 2013.
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6653598

[63] A. Bocquillon, G. Foucard, F. Miller, N. Buard, R. Leveugle, C. Daniel, S. Rakers, T. Carriere, V. Pouget & R. Velazco, “Highlights of laser testing capabilities regarding the understanding of SEE in SRAM based FPGAs”, Proceedings of the 9th European Conference on Radiation and its Effects on Components and Systems (RADECS 2007), 2007.
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5205500

[64] S. Skorobogatov, “Flash memory 'bumping' attacks”, Proceedings of the 12th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2010), Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 6225 LNCS, pp. 158-172, Springer, 2010.
https://www.cl.cam.ac.uk/~sps32/ches2010-bumping.pdf
http://link.springer.com/chapter/10.1007%2F978-3-642-15031-9_11

[65] Refractiveindex.info, Refractive index database, 2014.
http://refractiveindex.info/

[66] Wikipedia, “Nd:YAG laser”, 25 April 2014.
https://en.wikipedia.org/wiki/Nd:YAG_laser

[67] Wikipedia, “Neodymium-doped yttrium orthovanadate”, 25 April 2014.
https://en.wikipedia.org/wiki/Neodymium-doped_yttrium_orthovanadate

[68] Wikipedia, “Neodymium-doped yttrium lithium fluoride”, 25 April 2014.
https://en.wikipedia.org/wiki/Neodymium-doped_yttrium_lithium_fluoride

[69] Lee Laser Inc., “Nd:YLF as an Alternative to Nd:YAG – Advantages and Disadvantages”, 2014.
http://www.leelaser.com/pdf/Nd-YLF%20vs%20Nd-YAG.pdf

[70] Wikipedia, “Ti-sapphire laser”, 25 April 2014.
https://en.wikipedia.org/wiki/Ti-sapphire_laser

[71] Wikipedia, “Laser”, sect. “Continuous and pulsed modes of operation”, 25 April 2014.
https://en.wikipedia.org/wiki/Laser#Continuous_and_pulsed_modes_of_operation

[72] J. G. J. van Woudenberg, “Fault injection and countermeasures”, ECRYPT Summer School: Challenges in Security Engineering 2012.
https://wiki.crypto.rub.de/summerschool/slides/1_vanwoudenberg.pdf
https://www.riscure.com/documents/ecrypt_bochum_2012.pdf?1378978027

[73] J.H. Park, S.J. Moon, D.H. Choi, Y.S. Kang & J.C. Ha, “Differential fault analysis for round-reduced AES by fault injection”, ETRI (Electronics and Telecommunications Research Institute) Journal, vol. 33, iss. 3, pp. 434-442, 2011.
http://etrij.etri.re.kr/etrij/journal/article/article.do?volume=33&issue=3&page=434
http://etrij.etri.re.kr/etrij/journal/getPublishedPaperFile.do?fileId=SPF-1306908920780

[74] Wikipedia, “Silicon”, sect. “Two-photon absorption”, 25 April 2014.
https://en.wikipedia.org/wiki/Silicon_photonics#Two-photon_absorption

[75] Wikipedia, “Two-photon absorption”, 25 April 2014.
https://en.wikipedia.org/wiki/Two-photon_absorption

[76] V. Pouget, “Laser Beam Testing and Analysis of Integrated Circuits”, Seminar at Pontificia Universidad Católica de Perú (PUCP) in the framework of ALFA NICRON Project, 2007.
http://tima.imag.fr/alfa-nicron/documents/seminar%20PUCP_pouget_2007.pdf

[77] B. Jalali, “Silicon photonics: Nonlinear optics in the mid-infrared”, Nature Photonics vol. 4, iss. 8, pp. 506-508, 2010.
http://www.nature.com/nphoton/journal/v4/n8/full/nphoton.2010.170.html

[78] A. R. Motamedi, A. H. Nejadmalayeri, A. Khilo, F. X. Kärtner & E. P. Ippen, “Ultrafast nonlinear optical studies of silicon nanowaveguides”, Optics Express, vol. 20, iss. 4, pp. 4085-4101, 2012.
http://www.opticsinfobase.org/oe/viewmedia.cfm?uri=oe-20-4-4085&seq=0

[79] T. Wang, N. Venkatram, J. Gosciniak, Y. Cui, G. Qian, W. Ji & D. T. H. Tan, “Multi-photon absorption and third-order nonlinearity in silicon at mid-infrared wavelengths”, Optics Express, vol. 21, iss. 26, pp. 32192-32198, 2013.
http://www.opticsinfobase.org/oe/viewmedia.cfm?uri=oe-21-26-32192&seq=0

[80] K. A. Serrels, K. Erington, D. Bodoh, C. Farrell, N. Leslie, Th. R. Lundquist, Pr. Vedagarbha & D. T. Reid, “Two-photon laser-assisted device alteration in silicon integrated-circuits”, Optics Express, vol. 21, iss. 24, pp. 29083-29089, 2013.
http://www.opticsinfobase.org/oe/viewmedia.cfm?uri=oe-21-24-29083&seq=0

[81] K. A. Serrels, N. Leslie, T. R. Lundquist, P. Vedagarbha, K. Erington, D. Bodoh, C. Farrell & D. T. Reid, “Two-Photon-Absorption-Enhanced Laser-Assisted Device Alteration and Single-Event Upsets in 28nm Silicon Integrated Circuits”, Conference Proceedings from the 39th International Symposium for Testing and Failure Analysis (ISTFA 2013), pp. 173-181, 2013.
http://dcgsystems.com/PDF/2pLADA_and_2pSEU_ISTFA%202013.pdf

[82] H. Li & S. Moore, “Security evaluation at design time against optical fault injection attacks”, IEE Proceedings: Information Security, vol. 153, iss. 1, pp. 3-11, 2006.
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1613722

[83] CalcTool, “Focused spot size”, 2014.
http://www.calctool.org/CALC/phys/optics/spot_size

[84] II-VI infrared, “Spot size”, 2014.
http://www.iiviinfrared.com/resources/spot_size.html

[85] G. Canivet, R. Leveugle, J. Clédière, F. Valette & M. Renaudin, “Characterization of Effective Laser Spots during Attacks in the Configuration of a Virtex-II FPGA”, Proceedings of the 27th IEEE VLSI Test Symposium (VTS 2009), pp. 327-332, 2009.
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5116655

[86] L. Yin & G. P. Agrawal, “Impact of two-photon absorption on self-phase modulation in silicon waveguides: Free-carrier effects”, Optics Letters, vol. 32, iss. 14, pp. 2031-2033, 2007.
http://www.opticsinfobase.org/ol/viewmedia.cfm?uri=ol-32-14-2031&seq=0

[87] P. E. Dodd & L. W. Massengill, “Basic mechanisms and modeling of single-event upset in digital microelectronics”, IEEE Transactions on Nuclear Science, vol. 50 III, iss. 3, pp. 583-602, 2003.
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1208578

[88] J.-M. Dutertre, J. J. A. Fournier, A.-P. Mirbaha, D. Naccache, J.-B. Rigaud, B. Robisson & A. Tria, “Review of fault injection mechanisms and consequences on countermeasures design”, Proceedings of the 6th International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS'11), pp. 1-6, 2011.
http://ieeexplore.ieee.org/xpls/icp.jsp?arnumber=5941421

[89] J. J. A. Fournier, S. Moore, H. Li, R. Mullins & G. Taylor, “Security evaluation of asynchronous circuits”, Proceedings of the 5th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2003), Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 2779, pp. 137-151, 2003.
http://link.springer.com/chapter/10.1007%2F978-3-540-45238-6_12

[90] Wikipedia, “Photodetector”, 25 April 2014.
https://en.wikipedia.org/wiki/Photodetector

[91] Hamamatsu Photonics K.K., “Opto-semiconductor Handbook”, chap. 2: “Si photodiodes”, 14 June 2013.
http://www.hamamatsu.com/resources/pdf/ssd/e02_handbook_si_photodiode.pdf

[92] AP Technologies Ltd., “Photodiode Theory of Operation”, 2014.
http://www.aptechnologies.co.uk/support/photodiodes/photodiode-theory-of-operation

[93] Wikipedia, “Photodiode”, 25 April 2014.
https://en.wikipedia.org/wiki/Photodiode

[94] Thorlabs, “Photodiode Tutorial – Theory of Operation”, 2013.
http://www.thorlabs.de/tutorials.cfm?tabID=31760

[95] R. Paschotta, RP Photonics Encyclopedia, “Photodiodes”, 2014.
http://www.rp-photonics.com/photodiodes.html

[96] Wikipedia, “PIN diode”, 25 April 2014.
https://en.wikipedia.org/wiki/PIN_diode

[97] R. Paschotta, RP Photonics Encyclopedia, “p–i–n Photodiodes”, 2014.
http://www.rp-photonics.com/p_i_n_photodiodes.html

[98] R. Paschotta, RP Photonics Encyclopedia, “Avalanche Photodiodes”, 2014.
http://www.rp-photonics.com/avalanche_photodiodes.html

[99] Wikipedia, “Avalance photodiode”, 25 April 2014.
https://en.wikipedia.org/wiki/Avalanche_photodiode

[100] Wikipedia, “Single-photon avalance diode”, 25 April 2014.
https://en.wikipedia.org/wiki/Single-photon_avalanche_diode

[101] T. Esbach, W. Fumf, O. Kulikovska, D. Merli, D. Schuster, F. Stumpf, “A new security architecture for smartcards utilizing PUFs”, ISSE 2012 Securing Electronic Business Processes: Highlights of the Information Security Solutions Europe 2012 Conference, pp. 180-194, Springer, 2012.
http://link.springer.com/chapter/10.1007%2F978-3-658-00333-3_18

[102] Wikipedia, “Avalanche breakdown”, 25 April 2014.
https://en.wikipedia.org/wiki/Avalanche_breakdown

[103] The EncycloBEAMia, “Phototransistors”, 11 July 2003.
http://encyclobeamia.solarbotics.net/articles/phototransistor.html

[104] I. Poole, “Electronics and Radio Components: Phototransistor Tutorial”, 2014.
http://www.radio-electronics.com/info/data/semicond/phototransistor/photo_transistor.php

[105] B. Hansen, “How Phototransistors Operate”, 6 March 2013.
http://www.coilgun.info/theory/phototransistors.htm

[106] wiseGEEK, “What is a phototransistor?”, 2014.

http://www.wisegeek.com/what-is-a-phototransistor.htm

[107] I. Poole, “Electronics and Radio Components: Photodarlington”, 2014.
http://www.radio-electronics.com/info/data/semicond/phototransistor/photodarlington.php

[108] Kodenshi Corp., “Opto-semiconductor elements: Photo-darlington”, 2014.
http://www.kodenshi.co.jp/english/products/photodarlington/photodarlington.html

[109] Wikipedia, “Photoresistor”, 25 April 2014.
https://en.wikipedia.org/wiki/Photoresistor

[110] P. Marian, “LDR = Light Dependent Resistor = Photoresistor”, 2014.
http://www.electroschematics.com/6355/ldr-light-dependent-resistor-photoresistor/

[111] Resistor Guide, “Photoresistor”, 2014.
http://www.resistorguide.com/photoresistor/

[112] wiseGEEK, “What is a photoresistor?”, 2014.
http://www.wisegeek.com/what-is-a-photoresistor.htm

[113] Tooling University LLC, “What is the definition of "photoresistor"?”, 2013.
http://www.toolingu.com/definition-460355-34962-photoresistor.html

[114] Adafruit Industries, “Photocell tutorial”, 2013.
http://www.instructables.com/id/Photocell-tutorial/

[115] J. M. Fedeli, L. di Cioccio, D. Marris-Morini, L. Vivien, R. Orobtchouk, P. Rojo-Romeo, C. Seassal & F. Mandorlo, “Development of Silicon Photonics Devices Using Microelectronic Tools for the Integration on Top of a CMOS Wafer”, Advances in Optical Technologies, vol. 2008, art. ID 412518, 15 pages, Hindawi, 2008.
http://www.hindawi.com/journals/aot/2008/412518/
http://downloads.hindawi.com/journals/aot/2008/412518.pdf

[116] G. Masini, S. Sahni, G. Capellini, J. Witzens & C. Gunn, “High-Speed Near Infrared Optical Receivers Based on Ge Waveguide Photodetectors Integrated in a CMOS Process”, Advances in Optical Technologies, vol. 2008, art. ID 196572, 5 pages, Hindawi, 2008.
http://www.hindawi.com/journals/aot/2008/196572/
http://downloads.hindawi.com/journals/aot/2008/196572.pdf

[117] Leonid Khriachtchev, “Silicon Nanophotonics: Basic Principles, Current Status and Perspectives”, Pan Stanford, 2008.
http://www.panstanford.com/books/9789814241113.html

[118] C. Yang, C. J. Barrelet, F. Capasso & C. M. Lieber, “Single p-Type/Intrinsic/n-Type Silicon Nanowires as Nanoscale Avalanche Photodetectors”, Nano Letters, vol. 6, iss. 12, pp. 2929-2934, 2006.
http://www.chem.purdue.edu/yang/pubs/images/NanoLett_6_2929.pdf

[119] T. Mueller, F. Xia & P. Avouris, “Graphene photodetectors for high-speed optical communications”, Nature Photonics, vol. 4, iss. 5, pp. 297-301, 2010.

http://www.nature.com/nphoton/journal/v4/n5/full/nphoton.2010.40.html

[120] X. Wang, Z. Cheng, K. Xu, H. K. Tsang & J.-B. Xu, "High-responsivity graphene/silicon-heterostructure waveguide photodetectors", Nature Photonics, vol. 7, iss. 11, pp. 888-891, 2013.
http://www.nature.com/nphoton/journal/v7/n11/full/nphoton.2013.241.html

[121] G. Konstantatos, M. Badioli, L. Gaudreau, J. Osmond, M. Bernechea, F. P. G. de Arquer, F. Gatti & F. H. L. Koppens, "Hybrid graphene–quantum dot phototransistors with ultrahigh gain", Nature Nanotechnology, vol. 7, iss. 6, pp. 363-368, 2012.
http://www.nature.com/nnano/journal/v7/n6/full/nnano.2012.60.html

[122] A. Pospischil, M. Humer, M. M. Furchi, M.M., D. Bachmann, R. Guider, T. Fromherz, T. Mueller, "CMOS-compatible graphene photodetector covering all optical communication bands", Nature Photonics, vol. 7, iss. 11, pp. 892-896, 2013.
http://www.nature.com/nphoton/journal/v7/n11/full/nphoton.2013.240.html

[123] M. Albers, "Contactless Payments Chip Design", Card Tech Secure Tech Conference (CTST09), 2009.
http://www.sourcemediaconferences.com/CTST09/PDF09/D/Wednesday/AlbersManuel.pdf

[124] Wikipedia, "Silicon photonics", 25 April 2014.
https://en.wikipedia.org/wiki/Silicon_photonics

[125] Wikipedia, "Silicon on insulator", 25 April 2014.
https://en.wikipedia.org/wiki/Silicon_on_insulator

[126] G. K. Celler & S. Cristoloveanu, "Frontiers of silicon-on-insulator", Journal of Applied Physics, vol. 93, iss. 9, pp. 4955-4978, 2003.
http://scitation.aip.org/content/aip/journal/jap/93/9/10.1063/1.1558223
http://scitation.aip.org/deliver/fulltext/aip/journal/jap/93/9/1.1558223.pdf?itemId=/content/aip/journal/jap/93/9/10.1063/1.1558223&mimeType=pdf&containerItemId=content/aip/journal/jap

[127] Y. Gong & J. Vučković, "Photonic crystal cavities in silicon dioxide", Applied Physics Letters, vol. 96, iss. 3, 2010.
http://scitation.aip.org/content/aip/journal/apl/96/3/10.1063/1.3297877
http://scitation.aip.org/deliver/fulltext/aip/journal/apl/96/3/1.3297877.pdf?itemId=/content/aip/journal/apl/96/3/10.1063/1.3297877&mimeType=pdf&containerItemId=content/aip/journal/apl

[128] I. H. Malitson, "Interspecimen Comparison of the Refractive Index of Fused Silica", Journal of the Optics Society of America (JOSA), vol. 55, iss. 10, pp. 1205-1208, 1965.
http://www.opticsinfobase.org/josa/viewmedia.cfm?uri=josa-55-10-1205&seq=0

[129] Wikipedia, "Silicon on sapphire", 25 April 2014.
https://en.wikipedia.org/wiki/Silicon_on_sapphire

[130] Wikipedia, "Intel Terahertz", 25 April 2014.
http://en.wikipedia.org/wiki/Intel_TeraHertz

[131] Wikipedia, “Sapphire”, sect. “Transparency and hardness”, 25 April 2014.
http://en.wikipedia.org/wiki/Sapphire#Transparency_and_hardness

[132] E. Culurciello, “Silicon-on-Sapphire Circuits and Systems: Sensor and Biosensor Interfaces”, McGraw-Hill, 2009.
http://www.mcgraw-hill.co.uk/html/0071608486.html

[133] Wikipedia, “Birefrigence”, 25 April 2014.
https://en.wikipedia.org/wiki/Birefringence

[134] SurfaceNet, “Zirconium Oxide (Y203) (Stabilized with 9.5 m / ol%)”, 2014.
http://www.surfacenet.de/html/zirconium_oxide.html

[135] Fraunhofer Institute for Ceramic Technologies and Systems (IKTS), “Oxide Ceramics – Polycrystalline ZrO2: new transparent ceramics with high refractive index”, 2014.
http://www.ikts.fraunhofer.de/en/research_fields/materials/oxide_ceramics/ceramicsonbasezirconiumoxide/polycristallinetrnsp.html

[136] L. Liang & C. Su, “Facile deposition of zirconia optical films based on the solvothermal treatment of zirconyl nitrate-water-methanol system”, Proceedings of the 2nd International Conference on Advanced Design and Manufacturing Engineering (ADME 2012), Applied Mechanics and Materials, vol. 217-219, pp. 1033-1037, 2012.
http://www.scientific.net/AMM.217-219.1033

[137] Semiconductor Wafer Inc., “SiC Wafer”, 2011.
http://www.semiwafer.com/products/sic.htm

[138] Wikipedia, “Silicon carbide”, 25 April 2014.
https://en.wikipedia.org/wiki/Silicon_carbide

[139] S. Pezzagna, J. Brault, M. de Micheli, P. Vennégués, A. D. Wieck & J. Massies, “GaN, a new material for integrated nonlinear optics”, Proceedings of the 13th European Conference on Integrated Optics (ECIO 2007), 2007.
http://ecio-conference.org/2007/paper/2007_ThB2.pdf

[140] M. Fukuzawa, R. Kashiwagi & M. Yamada, “Observation of birefringence distribution caused by residual strain in bulk c-plane GaN substrates”, Proceedings of the 22nd International Conference on Indium Phosphide and Related Materials (IPRM 2010), pp. 184-186, 2010.
http://ieeexplore.ieee.org/xpls/icp.jsp?arnumber=5516031

[141] P. T. B. Shaffer, “Refractive Index, Dispersion, and Birefringence of Silicon Carbide Polytypes”, Applied Optics, vol. 10, iss. 5, pp. 1034-1036, 1971.
http://www.opticsinfobase.org/ao/viewmedia.cfm?uri=ao-10-5-1034&seq=0

[142] Wikipedia, “Gallium nitride”, 25 April 2014.
https://en.wikipedia.org/wiki/Gallium_nitride

[143] S. Bettarini, G. Batignani, G. Calderini, M. Carpinelli, R. Cenci, F. Forti, M. A. Giorgi, A. Lusiani, G. Marchiori, F. Morsani, N. Neri, E. Paoloni, M. Rama, G. Rizzo, G. Simi, J. Walsh, L. Ratti, V. Speziali, M. Manghisoni, V. Re, G. Traversi, L. Bosisio,G. Giacomini, L. Lanceri, I.

Rachevskaia & L. Vitale, “A new approach to the design of monolithic active pixel detectors in 0.13 µm source triple well CMOS technology”, Proceedings of the 14th International Workshop on Vertex Detectors (VERTEX 2005), Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, vol. 569, iss. 1, pp. 61-64, 2006.
http://www.sciencedirect.com/science/article/pii/S0168900206015956

Monolithic Active Pixel detector in 0.13 µm triple well CMOS Technology (A new approach with a full analog signal processor at pixel level and an extended collecting electrode)
http://epp.fnal.gov/DocDB/0000/000074/001/bettarini.ppt

[144] F. Forti, on behalf of the SLIM5 (Silicon detectors with Low Interaction with Material) Collaboration, “Development of Monolithic Active Pixel Sensors in a 0.13 µm Triple Well CMOS Technology with In-Pixel Full Analog Signal Processor”, SuperB III: The 3rd Workshop on a Super Flavor Factory based on Linear Collider Technology, 2006.
http://www.slac.stanford.edu/econf/C0606141/talks/DAY2/207.PDF

SLIM5 Collaboration: S. Bettarini, A. Bardi, G. Batignani, F. Bosi, G. Calderini, R. Cenci, M. dell’Orso, F. Forti, P. Giannetti, M. A. Giorgi, A. Lusiani, G. Marchiori, F. Morsani, N. Neri, E. Paoloni, G. Rizzo, J. Walsh, C. Andreoli, E. Pozzati, L. Ratti, V. Speziali, M. Manghisoni, V. Re, G. Traversi, L. Bosisio, G. Giacomini, L. Lanceri, I. Rachevskaia, L. Vitale, M. Bruschi, B. Giacobbe, N. Semprini, R. Spighi, M. Villa, A. Zoccoli, D. Gamba, G. Giraudo, P. Mereu, G. F. dalla Betta, G. Soncini, G. Fontana, L. Pancheri & G. Verzellesi

[145] S. Bettarini, A. Bardi, G. Batignani, F. Bosi, G. Calderini, R. Cenci, M. Dell’Orso, F. Forti, P. Giannetti, M.A. Giorgi, A. Lusiani, G. Marchiori, F. Morsani, N. Neri, E. Paoloni, G. Rizzo, J. Walsh, C. Andreoli, E. Pozzati, L. Ratti, V. Speziali, M. Manghisoni, V. Re, G. Traversi, L. Bosisio, G. Giacomini, L. Lanceri, I. Rachevskaia, L. Vitale, M. Bruschi, B. Giacobbe, N. Semprini, R. Spighi, M. Villa, A. Zoccoli, D. Gamba, G. Giraudo, P. Mereu, G.F. dalla Betta, G. Soncini, G. Fontana, L. Pancheri & G. Verzellesi, “Development of deep N-well monolithic active pixel sensors in a 0.13 µm CMOS technology”, Proceedings of the 10th Pisa Meeting on Advanced Detectors: Frontier Detectors for Frontier Physics, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, vol. 572, iss. 1, pp. 277-280, 2007.
http://www.sciencedirect.com/science/article/pii/S0168900206020936

[146] G. Rizzo, S. Bettarini, G. Calderini, R. Cenci, F. Forti, M.A. Giorgi, F. Morsani, L. Ratti, V. Speziali, M. Manghisoni, V. Re, G. Traversi & L. Bosisio, “A novel monolithic active pixel detector in 0.13 µm source triple well CMOS technology with pixel level analog processing”, Proceedings of the 3rd International Workshop on Semiconductor Pixel Detectors for Particles and Imaging (PIXEL 2005), Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, vol. 565, iss. 1, pp. 195-201, 2006.
http://www.sciencedirect.com/science/article/pii/S0168900206007698

[147] A. Gabrielli, on behalf of the SLIM5 Collaboration, “Development of a triple well CMOS MAPS device with in-pixel signal processing and sparsified readout capabilities”, Proceedings of the 11th International Vienna Conference on Instrumentation (VCI 2007), Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, vol. 581, iss. 1–2, pp. 303-305, 2007.

SLIM5 Collaboration: G. Batignani, S. Bettarini, F. Bosi, G. Calderini, R. Cenci, M. dell'Orso, F. Forti, P. Giannetti, M.A. Giorgi, A. Lusiani, G. Marchiori, F. Morsani, N. Neri, E. Paoloni, G. Rizzo, J. Walsh, L. Gaioni, M. Manghisoni, V. Re, G. Traversi, M. Bruschi, A. Gabrielli, B. Giacobbe, N. Semprini, R. Spighi, M. Villa, A. Zoccoli, G. Verzellesi, C. Andreoli, E. Pozzati, L. Ratti, V. Speziali, D. Gamba, G. Giraudo, P. Mereu, L. Bosisio, G. Giacomini, L. Lanceri, I. Rachevskaia & L. Vitale

[148] Università degli Studi di Pavia (University of Pavia), Dipartimento di Elettronica (Department of Electronics), Electronic Instrumentation Laboratory, "CMOS monolithic active pixel sensors for charged particle tracking applications", 2014.
http://eil.unipv.it/eil/?q=node/24

[149] L. Ratti, M. Manghisoni, V. Re, V. Speziali, G. Traversi, S. Bettarini, G. Calderini, R. Cenci, M. Giorgi, F. Forti, F. Morsani & G. Rizzo, "Monolithic pixel detectors in a 0.13 µm CMOS technology with sensor level continuous time charge amplification and shaping", Proceedings of the 10th European Symposium on Semiconductor Detectors: New Developments in Radiation Detectors, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, vol. 568, iss. 1, pp. 159-166, 2006.
http://www.sciencedirect.com/science/article/pii/S0168900206011004

[150] N. Wermes, "Pixel Detectors for Charged Particles", Proceedings of the 8th International Conference on Position Sensitive Detectors (PSD8), Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, vol. 604, iss. 1–2, pp. 370-379, 2009.
http://www.sciencedirect.com/science/article/pii/S016890020900196X
http://www-ucjf.troja.mff.cuni.cz/dolezal/teach/semicon/talks/Wermes_PSD8_080905.pdf

[151] Wikipedia, "Thyristor", 25 April 2014.
https://en.wikipedia.org/wiki/Thyristor

[152] R. Turchetta, CMOS Image Sensors CMOS Image Sensors, aka Monolithic Active Pixel Monolithic Active Pixel Sensors (MAPS), Oxford Fast Imaging Workshop, 2010.
http://www.physics.ox.ac.uk/lcfi/FastImSem/RenatoTurchetta.pdf

[153] L. Andricek, "DEPFET Pixel Detectors for Particle and Astrophysics", Proceedings of the International Symposium On Detector Development For Particle, Astroparticle And Synchrotron Radiation Experiments (SNIC 2006), 2006.
http://www.slac.stanford.edu/econf/C0604032/talks/snic-andricek.pdf

[154] V. F. Cavrois, V. Pouget, D. McMorrow, J. R. Schwank, N. Fel, F. Essely, R. S. Flores, P. Paillet, M. Gaillardin, D. Kobayashi, J. S. Melinger, O. Duhamel, P. E. Dodd & M. R. Shaneyfelt, "Investigation of the Propagation Induced Pulse Broadening (PIPB) effect on single event transients in SOI and bulk inverter chains", IEEE Transactions on Nuclear Science, vol. 55, iss. 6, pp. 2842-2853, 2008.
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4723807

[155] S. Yellampalli, N. S. Korivi & J. Marulanda, "Built-in current sensor for high speed transient current testing in analog CMOS circuits", Proceedings of the 40th Southeastern Symposium on System Theory (SSST), pp. 230-234, 2008.
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4480227

[156] Wikipedia, “Magnetoresistive random-access memory”, 25 April 2014.
https://en.wikipedia.org/wiki/Magnetoresistive_random-access_memory

[157] S.H. Weingart. “Physical Security for the µABYSS System”, Proceedings of the 1987 IEEE Symposium on Security and Privacy, pp. 52–59, 1987.
http://www.cs.washington.edu/research/projects/poirot3/Oakland/sp/PAPERS/00044429.PDF

[158] F. Imeson, A. Emtenan, S. Garg & M. V. Tripunitara, “Securing Computer Hardware Using 3D Integrated Circuit (IC) Technology and Split Manufacturing for Obfuscation”, Proceedings of the 22nd USENIX Security Symposium (USENIX Security '13/SEC ‘13), pp. 495-510, 2013.
http://0b4af6cdc2f0c5998459-c0245c5c937c5dedcca3f1764ecc9b2f.r43.cf2.rackcdn.com/12322-sec13-paper_imeson.pdf

[159] Tezzaron Semiconductor Corp., “3D-ICs and Integrated Circuit Security”, 2008.
http://www.tezzaron.com/media/3D-ICs_and_Integrated_Circuit_Security.pdf

[160] Wikipedia, “Error detection and correction”, 25 April 2014.
https://en.wikipedia.org/wiki/Error_detection_and_correction

[161] Wikipedia, “Parity function”, 25 April 2014.
https://en.wikipedia.org/wiki/Parity_function

[162] Wikipedia, “Hash function”, 25 April 2014.
https://en.wikipedia.org/wiki/Hash_function

[163] Wikipedia, “Cyclic redundancy check”, 25 April 2014.
https://en.wikipedia.org/wiki/Cyclic_redundancy_check

[164] Wikipedia, “Hamming code”, 25 April 2014.
https://en.wikipedia.org/wiki/Hamming_code

[165] Wikipedia, “Wafermap showing fully and partially patterned dies”, 24 February 2014. Licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license.
https://en.wikipedia.org/wiki/File:Wafermap_showing_fully_and_partially_patterned_dies.svg

[166] J. R. Samson Jr., W. Moreno & F. Falquez, “Validating fault tolerant designs using Laser Fault Injection (LFI)”, Proceedings of the 1997 IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems, pp. 175-183, 1997.
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=628323

[167] S. Krishnamohan & N. R. Mahapatra, “Analysis and design of soft-error hardened latches”, Proceedings of the 15th ACM Great Lakes Symposium on VLSI (GLSVLSI ‘05), pp. 328-331, 2005.
http://dl.acm.org/citation.cfm?id=1057740

[168] D.-G. Lee, D. Choi, J. Seo & H. Kim, “Reset tree-based optical fault detection”, Sensors (Switzerland), vol. 13, iss. 5, pp. 6713-6729, 2013.
http://www.mdpi.com/1424-8220/13/5/6713
http://www.mdpi.com/1424-8220/13/5/6713/pdf

[169] Y. Monnet, M. Renaudin, R. Leveugle, N. Feyt, P. Moitrel & F. M'Buwa Nzenguet, “Practical evaluation of fault countermeasures on an asynchronous DES crypto processor”, Proceedings of the 12th IEEE International On-Line Testing Symposium 2006 (IOLTS 2006), pp. 125-130, 2006.
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1655531

[170] Y. Monnet, M. Renaudin & R. Leveugle, “Designing resistant circuits against malicious faults injection using asynchronous logic”, IEEE Transactions on Computers, vol. 55, iss. 9, pp. 1104-1115, 2006.
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1668038

[171] Wikipedia, “Asynchronous system”, 25 April 2014.
https://en.wikipedia.org/wiki/Asynchronous_system

[172] Wikipedia, “Asynchronous circuit”, 25 April 2014.
https://en.wikipedia.org/wiki/Asynchronous_circuit

[173] J. Sparsø & Steve Furber, “Principles of Asynchronous Circuit Design: A Systems Perspective”, Springer, 2001.
http://link.springer.com/book/10.1007%2F978-1-4757-3385-3

[174] T. M. van Leeuwen, “Implementation and automatic generation of asynchronous scheduled data flow graphs”, M. Sc. thesis, TU Delft, 2010
http://repository.tudelft.nl/view/ir/uuid:5d87b87f-e084-491f-a18a-9c83ac2c41e1/
http://repository.tudelft.nl/assets/uuid:5d87b87f-e084-491f-a18a-9c83ac2c41e1/thesis.pdf

[175] A. Moradi, M. T. M. Shalmani, M. Salmasizadeh, “Dual-rail transition logic: A logic style for counteracting power analysis attacks”, Computers and Electrical Engineering, vol. 35, iss. 2, pp. 359-369, 2009.
http://www.sciencedirect.com/science/article/pii/S0045790608000621

[176] Image generated and kindly provided by Dr. André Jansman, Senior Scientist at NXP Semiconductors B.V., July 2014.

Appendix

This appendix contains the Verilog code of the different designs of logical countermeasures for the 16-bit word registers. The designed registers and countermeasures are fully functional only a clock cycle has been started with a logical “1” signal after the registers and the whole circuit have been reset with a logical “1” reset signal.

▪ Verilog code for standalone 32x16 register

```
module rf_standalone(clock, reset, writeEnable, dest, source, dataIn, dataOut);

parameter WIDTH = 16;
parameter DEPTH = 32;
parameter ADDRESSWIDTH = 5;

integer i,j;

input clock, reset, writeEnable;
input [ADDRESSWIDTH-1 : 0] dest;
input [ADDRESSWIDTH-1 : 0] source;
input [WIDTH-1 : 0] dataIn;

output [WIDTH-1 : 0] dataOut;

reg [WIDTH-1 : 0] dataOut; // registered output
reg [WIDTH-1 : 0] rf [DEPTH-1 : 0];

//reset
always@(posedge reset)
        begin
        for(i = 0; i<(DEPTH); i=i+1)
                begin
                rf[i] = 0;
                end
        dataOut = 0;
        end

// flip-flop for data-out, data-in and checking
always@(posedge clock)
        begin
        if (!reset)
                begin
                if (writeEnable)
                        begin
                        for (j=0; j<WIDTH; j=j+1)
                                begin
                                rf[dest][j] = dataIn[j];
                                end
                        end
                dataOut = rf[source];
                end
        end
endmodule
```

▪ Verilog code for a 32x16 register incorporating checksum

```
module rf_checksum(clock, reset, writeEnable, dest, source, dataIn, dataOut, err);

parameter WIDTH = 16;
parameter DEPTH = 32;
parameter ADDRESSWIDTH = 5;
parameter DIVISOR = 3;
parameter DIVBITS = 2;

integer i,j,k,l;

input clock, reset, writeEnable;
input [ADDRESSWIDTH-1 : 0] dest;
input [ADDRESSWIDTH-1 : 0] source;
input [WIDTH-1 : 0] dataIn;

output [WIDTH-1 : 0] dataOut;
output err;

reg [WIDTH-1 : 0] dataOut; // registered output
reg err; // registered error
reg [(WIDTH+DIVBITS) : 0] rf [DEPTH-1 : 0];
//reset
always@(posedge reset)
 begin
 for(i = 0; i<(DEPTH); i=i+1)
```

```
                begin
                rf[i] = 0;
                end
        dataOut = 0;
        err = 0;
        end

// flip-flop for data-out, data-in and checking
always@(posedge clock)
        begin
        for (i=0; i<DEPTH; i=i+1)
                begin
                l=0;
                for (j=0; j<WIDTH; j=j+1)
                        begin
                        l = l+rf[i][j];
                        end
                if ((l%DIVISOR) != rf[i][(WIDTH+DIVBITS):WIDTH])
                        begin
                        err = 1;
                        end
                end
        if (err)
                begin
                for (j=0; j<DEPTH; j=j+1)
                        begin
                        rf[j] = 0;
                        end
                dataOut = 0;
                end
        if ((!err) && (!reset))
                begin
                if (writeEnable)
                        begin
                        k=0;
                        for (j=0; j<WIDTH; j=j+1)
                                begin
                                rf[dest][j] = dataIn[j];
                                k = k+rf[dest][j];
                                end
                        rf[dest][(WIDTH+DIVBITS):WIDTH]=(k%DIVISOR);
                        end
```

```
                    dataOut = rf[source];
                    end
          end
endmodule
```

▪ Verilog code for a 32x16 register integrating CRC

```
module rf_crc(clock, reset, writeEnable, dest, source, dataIn, dataOut, err);

parameter WIDTH = 16;
parameter DEPTH = 32;
parameter ADDRESSWIDTH = 5;
parameter DIVISOR = 3;
parameter DIVBITS = 2;

integer i,j;

input clock, reset, writeEnable;
input [ADDRESSWIDTH-1 : 0] dest;
input [ADDRESSWIDTH-1 : 0] source;
input [WIDTH-1 : 0] dataIn;

output [WIDTH-1 : 0] dataOut;
output err;

reg [WIDTH-1 : 0] dataOut; // registered output
reg err; // registered error
reg [(WIDTH+DIVBITS) : 0] rf [DEPTH-1 : 0];

//reset
always@(posedge reset)
 begin
 for(i = 0; i<(DEPTH); i=i+1)
          begin
          rf[i] = 0;
          end
 dataOut = 0;
 err = 0;
 end

// flip-flop for data-out, data-in and checking
```

```
always@(posedge clock)
 begin
        for (i=0; i<DEPTH; i=i+1)
                begin
                if ((rf[i][WIDTH-1:0])%DIVISOR != rf[i][(WIDTH+DIVBITS):WIDTH])
                        begin
                        err = 1;
                        end
                end
        if (err)
                begin
                for (j=0; j<DEPTH; j=j+1)
                        begin
                        rf[j] = 0;
                        end
                dataOut = 0;
                end
        if ((!err) && (!reset))
                begin
                if (writeEnable)
                        begin
                        for (j=0; j<WIDTH; j=j+1)
                                begin
                                rf[dest][j] = dataIn[j];
                                end
                        rf[dest][(WIDTH+DIVBITS):WIDTH]=((rf[dest][WIDTH-1:0])
%DIVISOR);
                        end
                dataOut = rf[source];
                end
        end
endmodule
```

▪ Verilog code for a 32x16 register containing single parity bits per word

```
module rf_paritybits(clock, reset, writeEnable, dest, source, dataIn, dataOut, err);

parameter WIDTH = 16;
parameter DEPTH = 32;
parameter ADDRESSWIDTH = 5;
```

```
integer i,j,k,l;

input clock, reset, writeEnable;
input [ADDRESSWIDTH-1 : 0] dest;
input [ADDRESSWIDTH-1 : 0] source;
input [WIDTH-1 : 0] dataIn;

output [WIDTH-1 : 0] dataOut;
output err;

reg [WIDTH-1 : 0] dataOut; // registered output
reg err; // registered error
reg [WIDTH : 0] rf [DEPTH-1 : 0];

//reset
always@(posedge reset)
        begin
        for (i=0; i<DEPTH; i=i+1)
                begin
                rf[i] = 0;
                end
        dataOut = 0;
        err = 0;
        end

// flip-flop for data-out, data-in and checking
always@(posedge clock)
        begin
        for (i=0; i<DEPTH; i=i+1)
                begin
                l=0;
                for (j=0; j<WIDTH; j=j+1)
                        begin
                        l = l+rf[i][j];
                        end
                if ((l%2) != rf[i][WIDTH])
                        begin
                        err = 1;
                        end
                end
        if (err)
```

```
            begin
            for (j=0; j<DEPTH; j=j+1)
                begin
                rf[j] = 0;
                end
            dataOut = 0;
            end
        if ((!err) && (!reset))
            begin
            if (writeEnable)
                begin
                k=0;
                for (j=0; j<WIDTH; j=j+1)
                    begin
                    rf[dest][j] = dataIn[j];
                    k = k+rf[dest][j];
                    end
                rf[dest][WIDTH]=k%2;
                end
            dataOut = rf[source];
            end
    end
endmodule
```

▪ Verilog code for a 32x16 register containing parity bits per 4 data bits

```
module rf_parity_4(clock, reset, writeEnable, dest, source, dataIn, dataOut, err);

parameter WIDTH = 16; //must be multiple of PARITYBITS
parameter DEPTH = 32;
parameter ADDRESSWIDTH = 5;
parameter PARITYBITS = 4;

integer i,j,k,l,m;

input clock, reset, writeEnable;
input [ADDRESSWIDTH-1 : 0] dest;
input [ADDRESSWIDTH-1 : 0] source;
input [WIDTH-1 : 0] dataIn;
```

```
output [WIDTH-1 : 0] dataOut;
output err;

reg [WIDTH-1 : 0] dataOut; // registered output
reg err; // registered error
reg [(WIDTH-1)+PARITYBITS : 0] rf [DEPTH-1 : 0];

//reset
always@(posedge reset)
        begin
        for(i = 0; i<(DEPTH); i=i+1)
                begin
                rf[i] = 0;
                end
        dataOut = 0;
        err = 0;
        end

// flip-flop for data-out, data-in and checking
always@(posedge clock)
        begin
        for (i=0; i<DEPTH; i=i+1)
                begin
                for (k=0; k<PARITYBITS; k=k+1)
                        begin
                        m = rf[i][k*(WIDTH/PARITYBITS)];
                        for (l=1; l<(WIDTH/PARITYBITS); l=l+1)
                                begin
                                m = m ^ rf[i][l+(k*(WIDTH/PARITYBITS))];
                                end
                        if (m != rf[i][WIDTH+k])
                                begin
                                err =1;
                                end
                        end
                end
        if (err)
                begin
                for (j=0; j<DEPTH; j=j+1)
                        begin
                        rf[j] = 0;
                        end
        dataOut = 0;
        end
    if ((!err) && (!reset))
        begin
        if (writeEnable)
            begin
            for (j=0; j<WIDTH; j=j+1)
                begin
                rf[dest][j] = dataIn[j];
                end
            for (k=0; k<PARITYBITS; k=k+1)
                begin
                rf[dest][WIDTH+k] = rf[dest][k*(WIDTH/PARITYBITS)];
                for (l=1; l<(WIDTH/PARITYBITS); l=l+1)
                    begin
                    rf[dest][WIDTH+k] = rf[dest][WIDTH+k] ^ rf[dest][l+
(k*(WIDTH/PARITYBITS))];
                    end
                end
            end
        dataOut = rf[source];
        end
    end
endmodule
```

▪ Verilog code for a 32x16 register incorporating Hamming code (implementation A without autocorrection)

```
module rf_hamming(clock, reset, writeEnable, dest, source, dataIn, dataOut, err);

parameter WIDTH = 16;
parameter DEPTH = 32;
parameter ADDRESSWIDTH = 5;
parameter MAXBITS = 5; //log(2)WIDTH + 1

integer i, j, iter, remainder, o, k, b, y, a, z, it;

input clock, reset, writeEnable;
input [ADDRESSWIDTH-1 : 0] dest;
input [ADDRESSWIDTH-1 : 0] source;
input [WIDTH-1 : 0] dataIn;
```

```
output [WIDTH-1 : 0] dataOut;
output err;

reg [WIDTH-1 : 0] dataOut; // registered output
reg err; // registered error
reg [(WIDTH+MAXBITS)-1 : 0] rf [DEPTH-1 : 0];

//reset
always@(posedge reset)
        begin
        for(i = 0; i<(DEPTH); i=i+1)
                begin
                rf[i] = 0;
                end
        dataOut = 0;
        err = 0;
        end

// flip-flop for data-out, data-in and checking
always@(posedge clock)
        begin
        for (i=0; i<DEPTH; i=i+1)
                begin
                o = 1;
                iter = 1;
                for (j=0; j<WIDTH; j=j+1)
                        begin
                        if (j == o-1)
                                begin
                                b=1;
                                z=0;
                                remainder = 0;
                                it = iter;
                                while (it > 0)
                                        begin
                                        b = b*2;
                                        it = it -1;
                                        end
                                y = (b/2)-1;
                                for (k=j; k<WIDTH; k=k+1)
                                        begin
                while (y < (b-1))
                    begin
                    if ((k%b) == y)
                        begin
                    z = z + rf[i][k];
                        remainder = k - b;
                        end
                    a = y;
                    y = y + 1;
                    if (remainder < 1)
                        begin
                        y = b;
                        end
                    end
                y = a + 1;
                    if (y == (b-1))
                    begin
                    y = (b/2)-1;
                    end
                end
            if ((z%2) != rf[i][(WIDTH+iter)-1])
                begin
                err = 1;
                end
            iter = iter + 1;
            o = o *2;
            end
        end
    end
if (err)
    begin
    for (j=0; j<DEPTH; j=j+1)
        begin
        rf[j] = 0;
        end
    dataOut = 0;
    end
if ((!err) && (!reset))
    begin
    if (writeEnable)
        begin
        o = 1;
```

```
iter = 1;
rf[dest] = dataIn;
for (j=0; j<WIDTH; j=j+1)
    begin
    if (j == o-1)
        begin
        b=1;
        z=0;
        remainder = 0;
        it = iter;
        while (it > 0)
            begin
            b = b*2;
            it = it -1;
            end
        y = (b/2)-1;
        for (k=j; k<WIDTH; k=k+1)
            begin
            while (y < (b-1))
                begin
                if ((k%b) == y)
                    begin
                    z = z + rf[dest][k];
                    remainder = k - b;
                    end
                a = y;
                y = y + 1;
                if (remainder < 1)
                    begin
                    y = b;
                    end
                end
            y = a + 1;
            if (y == (b-1))
                begin
                y = (b/2)-1;
                end
            end
        rf[dest][(WIDTH+iter)-1] = (z%2);
        iter = iter + 1;
        o = o *2;
        end
```

```
                                        end
                                end
                        dataOut = rf[source];
                        end
                end
endmodule
```

▪ Verilog code for a 32x16 register containing Hamming code (implementation B without autocorrection)

```
module rf_hamming_new(clock, reset, writeEnable, dest, source, dataIn, dataOut, err);

parameter WIDTH = 16;
parameter DEPTH = 32;
parameter ADDRESSWIDTH = 5;
parameter MAXBITS = 5; //log(2)WIDTH + 1

integer i, j, k, b, y, d, c, ka, ya, da, ca;

input clock, reset, writeEnable;
input [ADDRESSWIDTH-1 : 0] dest;
input [ADDRESSWIDTH-1 : 0] source;
input [WIDTH-1 : 0] dataIn;

output [WIDTH-1 : 0] dataOut;
output err;

reg [WIDTH-1 : 0] dataOut; // registered output
reg err; // registered error
reg [(WIDTH+MAXBITS)-1 : 0] rf [DEPTH-1 : 0];

//reset
always@(posedge reset)
 begin
 for(i = 0; i<(DEPTH); i=i+1)
        begin
        rf[i] = 0;
        end
 dataOut = 0;
 err = 0;
 end
```

```
// flip-flop for data-out, data-in and checking
always@(posedge clock)
 begin
        for (i=0; i<DEPTH; i=i+1)
                begin
                c = 1;
                k = 0;
                b = 0;
                d = 1;
                while (k<(WIDTH-1))
                        begin
                        for(y=0; y<c; y=y+1)
                                begin
                                if ((y+k)<WIDTH)
                                        begin
                                        b = b ^ rf[i][y+k];
                                        end
                                end
                        k=k+c*2;
                        if (k >= WIDTH)
                                begin
                                if (b != rf[i][(WIDTH+d)-1])
                                        begin
                                        err = 1;
                                        end
                                d = d + 1;
                                c = c*2;
                                k = c-1;
                                b = 0;
                                end
                        end
                end
        if (err)
                begin
                for (j=0; j<DEPTH; j=j+1)
                        begin
                        rf[j] = 0;
                        end
                dataOut = 0;
                end
        if ((!err) && (!reset))
```

```
				begin
				if (writeEnable)
					begin
					rf[dest] = dataIn;
					ca = 1;
					ka = 0;
					da = 1;
					rf[dest][(WIDTH+da)-1] = 0;
					while (ka<(WIDTH-1))
						begin
						for(ya=0; ya<ca; ya=ya+1)
							begin
							if ((ya+ka)<WIDTH)
								begin
								rf[dest][(WIDTH+da)-1] = rf[dest][(WIDTH+da)-1] ^ 
rf[dest][ya+ka];
								end
							end
						ka=ka+ca*2;
						if (ka >= WIDTH)
							begin
							da = da + 1;
							ca = ca*2;
							ka = ca-1;
							rf[dest][(WIDTH+da)-1] = 0;
							end
						end
					end
				dataOut = rf[source];
				end
		end
endmodule
```

▪ Verilog code for a 32x16 register with dual modular redundancy

```
module rf_2xredundancy(clock, reset, writeEnable, dest, source, dataIn, dataOut, err);

parameter WIDTH = 16;
parameter DEPTH = 32;
parameter ADDRESSWIDTH = 5;
```

```
integer i, j;

input clock, reset, writeEnable;
input [ADDRESSWIDTH-1 : 0] dest;
input [ADDRESSWIDTH-1 : 0] source;
input [WIDTH-1 : 0] dataIn;

output [WIDTH-1 : 0] dataOut;
output err;

reg [WIDTH-1 : 0] dataOut; // registered output
reg err; // registered error
reg [WIDTH-1 : 0] rf [DEPTH-1 : 0];
reg [WIDTH-1 : 0] rr [DEPTH-1 : 0];

//reset
always@(posedge reset)
        begin
        for(i = 0; i<(DEPTH); i=i+1)
                begin
                rf[i] <= 0;
                rr[i] <= 0;
                end
        dataOut <= 0;
        err = 0;
        end

// flip-flop for data-out, data-in and checking
always@(posedge clock)
        begin
        for (i=0; i<DEPTH; i=i+1)
                begin
                if (rf[i] != rr[i])
                         begin
                         err = 1;
                         end
                end

        if (err)
                begin
                for (j=0; j<DEPTH; j=j+1)
```

```
                        begin
                        rf[j] <= 0;
                        rr[j] <= 0;
                        end
                dataOut <= 0;
                end

        if ((!err) && (!reset))
                begin
                if (writeEnable)
                        begin
                        for (j=0; j<WIDTH; j=j+1)
                                begin
                                rf[dest][j] <= dataIn[j];
                                rr[dest][j] <= dataIn[j];
                                end
                        end
                dataOut <= rf[source];
                end
        end
endmodule
```

▪ Verilog code for a 32x16 register with triple modular redundancy

```
module rf_3xredundancy(clock, reset, writeEnable, dest, source, dataIn, dataOut, err);

parameter WIDTH = 16;
parameter DEPTH = 32;
parameter ADDRESSWIDTH = 5;

integer i, j, fix;

input clock, reset, writeEnable;
input [ADDRESSWIDTH-1 : 0] dest;
input [ADDRESSWIDTH-1 : 0] source;
input [WIDTH-1 : 0] dataIn;

output [WIDTH-1 : 0] dataOut;
output err;
```

```
reg [WIDTH-1 : 0] dataOut; // registered output
reg err; // registered error
reg [WIDTH-1 : 0] rf [DEPTH-1 : 0];
reg [WIDTH-1 : 0] rr [DEPTH-1 : 0];
reg [WIDTH-1 : 0] mr [DEPTH-1 : 0];

//reset
always@(posedge reset)
 begin
 for(i = 0; i<(DEPTH); i=i+1)
            begin
            rf[i] <= 0;
            rr[i] <= 0;
           mr[i] <= 0;
           end
 dataOut <= 0;
 err = 0;
  end

// flip-flop for data-out, data-in and checking
always@(posedge clock)
           begin
           for (i=0; i<DEPTH; i=i+1)
                      begin
                      if (err)
                                 begin
                                 if ((rf[i] != rr[i]) && (rr[i] == mr[i]))
                                            begin
                                            rf[i] <= rr[i];
                                            end
                                 if ((rf[i] == rr[i]) && (rr[i] != mr[i]))
                                            begin
                                            mr[i] <= rr[i];
                                            end
                                 fix = 1;
                                 end
                      if ((rf[i] != rr[i]) || (rr[i] != mr[i]))
                                  begin
                                  err = 1;
                                  fix = 0;
                                  end
                      end
```

```
        if ((err) && (!fix))
                begin
                for (j=0; j<DEPTH; j=j+1)
                        begin
                        rf[j] <= 0;
                        rr[j] <= 0;
                        mr[j] <= 0;
                        end
                dataOut <= 0;
                end

        if ((!err) && (!reset))
                begin
                if (writeEnable)
                        begin
                        for (j=0; j<WIDTH; j=j+1)
                                begin
                                rf[dest][j] <= dataIn[j];
                                rr[dest][j] <= dataIn[j];
                                mr[dest][j] <= dataIn[j];
                                end
                        end
                dataOut <= rf[source];
                end
        end
endmodule
```

▪ Verilog code for a 32x16 register incorporating checksum with dual modular redundancy

```
module rf_checksum_red(clock, reset, writeEnable, dest, source, dataIn, dataOut, err);

parameter WIDTH = 16;
parameter DEPTH = 32;
parameter ADDRESSWIDTH = 5;
parameter DIVISOR = 3;
parameter DIVBITS = 2;

integer i,j,k,l,m,n,fix;
```

```
input clock, reset, writeEnable;
input [ADDRESSWIDTH-1 : 0] dest;
input [ADDRESSWIDTH-1 : 0] source;
input [WIDTH-1 : 0] dataIn;

output [WIDTH-1 : 0] dataOut;
output err;

reg [WIDTH-1 : 0] dataOut; // registered output
reg err; // registered error
reg [(WIDTH+DIVBITS) : 0] rf [DEPTH-1 : 0];
reg [(WIDTH+DIVBITS) : 0] rr [DEPTH-1 : 0];

//reset
always@(posedge reset)
        begin
        for(i = 0; i<(DEPTH); i=i+1)
                begin
                rf[i] = 0;
                rr[i] = 0;
                end
        dataOut = 0;
        err = 0;
        end

// flip-flop for data-out, data-in and checking
always@(posedge clock)
        begin
        for (i=0; i<DEPTH; i=i+1)
                begin
                l=0;
                m=0;
                for (j=0; j<WIDTH; j=j+1)
                        begin
                        l = l+rf[i][j];
                        m = m+rr[i][j];
                        end
                if ((l%DIVISOR) != rf[i][(WIDTH+DIVBITS):WIDTH])
                        begin
                        if ((m%DIVISOR) != rr[i][(WIDTH+DIVBITS):WIDTH])
                                begin
                                err = 1;
                fix = 0;
                end
            else
                begin
                err = 1;
                rf[i] = rr[i];
                fix = 1;
                end
            end
        end
    if ((err) && (!fix))
        begin
        for (j=0; j<DEPTH; j=j+1)
            begin
            rf[j] = 0;
            rr[j] = 0;
            end
        dataOut = 0;
        end
    if ((!err) && (!reset))
        begin
        if (writeEnable)
            begin
            k=0;
            n=0;
            rf[dest] = dataIn;
            rr[dest] = dataIn;
            for (j=0; j<WIDTH; j=j+1)
                begin
                k = k+rf[dest][j];
                n = n+rr[dest][j];
                end
            rf[dest][(WIDTH+DIVBITS):WIDTH]=(k%DIVISOR);
            rr[dest][(WIDTH+DIVBITS):WIDTH]=(n%DIVISOR);
            end
        dataOut = rf[source];
        end
    end
endmodule
```

▪ Verilog code for a 32x16 register integrating CRC with dual modular redundancy

```
module rf_crc_red(clock, reset, writeEnable, dest, source, dataIn, dataOut, err);

parameter WIDTH = 16;
parameter DEPTH = 32;
parameter ADDRESSWIDTH = 5;
parameter DIVISOR = 3;
parameter DIVBITS = 2;

integer i,j,fix;

input clock, reset, writeEnable;
input [ADDRESSWIDTH-1 : 0] dest;
input [ADDRESSWIDTH-1 : 0] source;
input [WIDTH-1 : 0] dataIn;

output [WIDTH-1 : 0] dataOut;
output err;

reg [WIDTH-1 : 0] dataOut; // registered output
reg err; // registered error
reg [(WIDTH+DIVBITS) : 0] rf [DEPTH-1 : 0];
reg [(WIDTH+DIVBITS) : 0] rr [DEPTH-1 : 0];

//reset
always@(posedge reset)
        begin
        for(i = 0; i<(DEPTH); i=i+1)
                begin
                rf[i] = 0;
                rr[i] = 0;
                end
        dataOut = 0;
        err = 0;
        end

// flip-flop for data-out, data-in and checking
always@(posedge clock)
        begin
        for (i=0; i<DEPTH; i=i+1)
            begin
            if ((rf[i][WIDTH-1:0])%DIVISOR != rf[i][(WIDTH+DIVBITS):WIDTH])
                begin
                if ((rr[i][WIDTH-1:0])%DIVISOR != rr[i][(WIDTH+DIVBITS):WIDTH])
                    begin
                    err = 1;
                    fix = 0;
                    end
                else
                    begin
                    err = 1;
                    rf[i] = rr[i];
                    fix = 1;
                    end
                end
            end
        if ((err) && (!fix))
            begin
            for (j=0; j<DEPTH; j=j+1)
                begin
                rf[j] = 0;
                rr[j] = 0;
                end
            dataOut = 0;
            end
        if ((!err) && (!reset))
            begin
            if (writeEnable)
                begin
                rf[dest] = dataIn;
                rr[dest] = dataIn;
                rf[dest][(WIDTH+DIVBITS):WIDTH]=((rf[dest][WIDTH-1:0])
    %DIVISOR);
                rr[dest][(WIDTH+DIVBITS):WIDTH]=((rr[dest][WIDTH-1:0])
    %DIVISOR);
                end
            dataOut = rf[source];
            end
        end
    endmodule
```

▪ Verilog code for a 32x16 register containing single parity bits per word with dual modular redundancy

```
module rf_paritybits_red(clock, reset, writeEnable, dest, source, dataIn, dataOut, err);

parameter WIDTH = 16;
parameter DEPTH = 32;
parameter ADDRESSWIDTH = 5;

integer i,j,k,l,m,n,fix;

input clock, reset, writeEnable;
input [ADDRESSWIDTH-1 : 0] dest;
input [ADDRESSWIDTH-1 : 0] source;
input [WIDTH-1 : 0] dataIn;

output [WIDTH-1 : 0] dataOut;
output err;

reg [WIDTH-1 : 0] dataOut; // registered output
reg err; // registered error
reg [WIDTH : 0] rf [DEPTH-1 : 0];
reg [WIDTH : 0] rr [DEPTH-1 : 0];

//reset
always@(posedge reset)
        begin
        for (i=0; i<DEPTH; i=i+1)
                begin
                rf[i] = 0;
                rr[i] = 0;
                end
        dataOut = 0;
        err = 0;
        end

// flip-flop for data-out, data-in and checking
always@(posedge clock)
        begin
        for (i=0; i<DEPTH; i=i+1)
                begin
```

```
    l=0;
    m=0;
    for (j=0; j<WIDTH; j=j+1)
        begin
        l = l+rf[i][j];
        m = m+rr[i][j];
        end
    if ((l%2) != rf[i][WIDTH])
        begin
        if ((m%2) != rr[i][WIDTH])
            begin
            err = 1;
            fix = 0;
            end
        else
            begin
            err = 1;
            rf[i]=rr[i];
            fix = 1;
            end
        end
    end
if ((err) && (!fix))
    begin
    for (j=0; j<DEPTH; j=j+1)
        begin
        rf[j] = 0;
        rr[j] = 0;
        end
    dataOut = 0;
    end
if ((!err) && (!reset))
    begin
    if (writeEnable)
        begin
        k=0;
        n=0;
        rf[dest] = dataIn;
        rr[dest] = dataIn;
        for (j=0; j<WIDTH; j=j+1)
            begin
            k = k+rf[dest][j];
```

```
                        n = n+rr[dest][j];
                        end
                    rf[dest][WIDTH]=k%2;
                    rr[dest][WIDTH]=n%2;
                    end
                dataOut = rf[source];
                end
        end
endmodule
```

▪ Verilog code for a 32x16 register containing parity bits per 4 data bits with dual modular redundancy

```
module rf_parity_4_red(clock, reset, writeEnable, dest, source, dataIn, dataOut, err);

parameter WIDTH = 16; //must be multiple of PARITYBITS
parameter DEPTH = 32;
parameter ADDRESSWIDTH = 5;
parameter PARITYBITS = 4;

integer i,j,k,l,m,n,o,fix;

input clock, reset, writeEnable;
input [ADDRESSWIDTH-1 : 0] dest;
input [ADDRESSWIDTH-1 : 0] source;
input [WIDTH-1 : 0] dataIn;

output [WIDTH-1 : 0] dataOut;
output err;

reg [WIDTH-1 : 0] dataOut; // registered output
reg err; // registered error
reg [(WIDTH-1)+PARITYBITS : 0] rf [DEPTH-1 : 0];
reg [(WIDTH-1)+PARITYBITS : 0] rr [DEPTH-1 : 0];

//reset
always@(posedge reset)
        begin
        for(i = 0; i<(DEPTH); i=i+1)
                begin
                rf[i] = 0;
        rr[i] = 0;
        end
    dataOut = 0;
    err = 0;
    end

// flip-flop for data-out, data-in and checking
always@(posedge clock)
    begin
    for (i=0; i<DEPTH; i=i+1)
        begin
        k = 0;
        for (o=0; o<PARITYBITS; o=o+1)
            begin
            m = rf[i][o*(WIDTH/PARITYBITS)];
            n = rr[i][o*(WIDTH/PARITYBITS)];
            for (l=1; l<(WIDTH/PARITYBITS); l=l+1)
                begin
                m = m ^ rf[i][l+(o*(WIDTH/PARITYBITS))];
                n = n ^ rr[i][l+(o*(WIDTH/PARITYBITS))];
                end
            if (n != rr[i][WIDTH+o])
                begin
                k=k+1;
                end
            if (m != rf[i][WIDTH+o])
                begin
                if (k != 0)
                    begin
                    err = 1;
                    fix = 0;
                    end
                else
                    begin
                    err = 1;
                    rf[i] = rr[i];
                    fix = 1;
                    end
                end
            end
        end
    if ((err) && (!fix))
```

```
                begin
                for (j=0; j<DEPTH; j=j+1)
                        begin
                        rf[j] = 0;
                        rr[j] = 0;
                        end
                dataOut = 0;
                end
        if ((!err) && (!reset))
                begin
                if (writeEnable)
                        begin
                        rf[dest] = dataIn;
                        rr[dest] = dataIn;
                        for (o=0; o<PARITYBITS; o=o+1)
                                begin
                                rf[dest][WIDTH+o] = rf[dest][o*(WIDTH/PARITYBITS)];
                                rr[dest][WIDTH+o] = rr[dest][o*(WIDTH/PARITYBITS)];
                                for (l=1; l<(WIDTH/PARITYBITS); l=l+1)
                                        begin
                                        rf[dest][WIDTH+o] = rf[dest][WIDTH+o] ^ rf[dest][l+
(o*(WIDTH/PARITYBITS))];
                                        rr[dest][WIDTH+o] = rr[dest][WIDTH+o] ^ rr[dest][l+
(o*(WIDTH/PARITYBITS))];
                                        end
                                end
                        end
                dataOut = rf[source];
                end
        end
endmodule
```

▪ Verilog code for a 32x16 register containing Hamming code (implementation A) with dual modular redundancy

```
module rf_hamming_red(clock, reset, writeEnable, dest, source, dataIn, dataOut, err);

parameter WIDTH = 16;
parameter DEPTH = 32;
parameter ADDRESSWIDTH = 5;
parameter MAXBITS = 5; //log(2)WIDTH+ 1
```

```
integer i, j, iter, remainder, o, k, b, y, a, z, it,zz,fix;

input clock, reset, writeEnable;
input [ADDRESSWIDTH-1 : 0] dest;
input [ADDRESSWIDTH-1 : 0] source;
input [WIDTH-1 : 0] dataIn;

output [WIDTH-1 : 0] dataOut;
output err;

reg [WIDTH-1 : 0] dataOut; // registered output
reg err; // registered error
reg [(WIDTH+MAXBITS)-1 : 0] rf [DEPTH-1 : 0];
reg [(WIDTH+MAXBITS)-1 : 0] rr [DEPTH-1 : 0];

//reset
always@(posedge reset)
        begin
        for(i = 0; i<(DEPTH); i=i+1)
                begin
                rf[i] = 0;
                rr[i] = 0;
                end
        dataOut = 0;
        err = 0;
        end

// flip-flop for data-out, data-in and checking
always@(posedge clock)
        begin
        for (i=0; i<DEPTH; i=i+1)
                begin
                o = 1;
                iter = 1;
                for (j=0; j<WIDTH; j=j+1)
                        begin
                        if (j == o-1)
                                begin
                                b=1;
                                z=0;
                                zz = 0;
```

```
remainder = 0;
it = iter;
while (it > 0)
        begin
        b = b*2;
        it = it -1;
        end
y = (b/2)-1;
for (k=j; k<WIDTH; k=k+1)
        begin
        while (y < (b-1))
                begin
                if ((k%b) == y)
                        begin
                        z = z + rf[i][k];
                        zz = zz + rr[i][k];
                        remainder = k - b;
                        end
                a = y;
                y = y + 1;
                if (remainder < 1)
                        begin
                        y = b;
                        end
                end
        y = a + 1;
        if (y == (b-1))
                begin
                y = (b/2)-1;
                end
        end
if ((z%2) != rf[i][(WIDTH+iter)-1])
        begin
        if ((zz%2) != rr[i][(WIDTH+iter)-1])
                begin
                err = 1;
                fix = 0;
                end
        else
                begin
                err = 1;
                rf[i]=rr[i];
```

```
                    fix = 1;
                    end
                end
            iter = iter + 1;
            o = o *2;
            end
        end
    end
if ((err) && (!fix))
    begin
    for (j=0; j<DEPTH; j=j+1)
        begin
        rf[j] = 0;
        rr[j] = 0;
        end
    dataOut = 0;
    end
if ((!err) && (!reset))
    begin
    if (writeEnable)
        begin
        o = 1;
        iter = 1;
        rf[dest] = dataIn;
        rr[dest] = dataIn;
        for (j=0; j<WIDTH; j=j+1)
            begin
            if (j == o-1)
                begin
                b=1;
                z=0;
                zz=0;
                remainder = 0;
                it = iter;
                while (it > 0)
                    begin
                    b = b*2;
                    it = it -1;
                    end
                y = (b/2)-1;
                for (k=j; k<WIDTH; k=k+1)
                    begin
```

```
                        while (y < (b-1))
                            begin
                            if ((k%b) == y)
                                begin
                                z = z + rf[dest][k];
                                zz = zz + rr[dest][k];
                                remainder = k - b;
                                end
                            a = y;
                            y = y + 1;
                            if (remainder < 1)
                                begin
                                y = b;
                                end
                            end
                        y = a + 1;
                        if (y == (b-1))
                            begin
                            y = (b/2)-1;
                            end
                        end
                    rf[dest][(WIDTH+iter)-1] = (z%2);
                    rr[dest][(WIDTH+iter)-1] = (zz%2);
                    iter = iter + 1;
                    o = o *2;
                    end
                end
            end
        dataOut = rf[source];
        end
    end
endmodule
```

▪ Verilog code for a 32x16 register containing Hamming code (implementation B) with dual modular redundancy

```
module rf_hamming_new_red(clock, reset, writeEnable, dest, source, dataIn, dataOut, err);

parameter WIDTH = 16;
parameter DEPTH = 32;
parameter ADDRESSWIDTH = 5;
parameter MAXBITS = 5; //log(2)16 + 1

integer i, j, k, b, y, d, c, ka, ya, da, ca,bz,fix;

input clock, reset, writeEnable;
input [ADDRESSWIDTH-1 : 0] dest;
input [ADDRESSWIDTH-1 : 0] source;
input [WIDTH-1 : 0] dataIn;

output [WIDTH-1 : 0] dataOut;
output err;

reg [WIDTH-1 : 0] dataOut; // registered output
reg err; // registered error
reg [(WIDTH+MAXBITS)-1 : 0] rf [DEPTH-1 : 0];
reg [(WIDTH+MAXBITS)-1 : 0] rr [DEPTH-1 : 0];

//reset
always@(posedge reset)
        begin
        for(i = 0; i<(DEPTH); i=i+1)
                begin
                rf[i] = 0;
                rr[i] = 0;
                end
        dataOut = 0;
        err = 0;
        end

// flip-flop for data-out, data-in and checking
always@(posedge clock)
        begin
        for (i=0; i<DEPTH; i=i+1)
                begin
                c = 1;
                k = 0;
                b = 0;
                bz= 0;
                d = 1;
                while (k<(WIDTH-1))
                        begin
                        for(y=0; y<c; y=y+1)
```

```
            begin
            if ((y+k)<WIDTH)
                begin
                b = b ^ rf[i][y+k];
                bz = bz ^ rr[i][y+k];
                end
            end
        k=k+c*2;
        if (k >= WIDTH)
            begin
            if (b != rf[i][(WIDTH+d)-1])
                begin
                if (b != rr[i][(WIDTH+d)-1])
                begin
                    err = 1;
                    fix = 0;
                    end
                else
                    begin
                    err = 1;
                    rf[i]=rr[i];
                    fix = 1;
                    end
                end
            d = d + 1;
            c = c*2;
            k = c-1;
            b = 0;
            end
        end
    end
if ((err) && (!fix))
    begin
    for (j=0; j<DEPTH; j=j+1)
        begin
        rf[j] = 0;
        rr[j] = 0;
        end
    dataOut = 0;
    end
if ((!err) && (!reset))
    begin
```

```
        if (writeEnable)
            begin
            rf[dest] = dataIn;
            rr[dest] = dataIn;
            ca = 1;
            ka = 0;
            da = 1;
            rf[dest][(WIDTH+da)-1] = 0;
            rr[dest][(WIDTH+da)-1] = 0;
            while (ka<(WIDTH-1))
                begin
                for(ya=0; ya<ca; ya=ya+1)
                    begin
                    if ((ya+ka)<WIDTH)
                        begin
                        rf[dest][(WIDTH+da)-1] = rf[dest][(WIDTH+da)-1] ^
rf[dest][ya+ka];
                        rr[dest][(WIDTH+da)-1] = rr[dest][(WIDTH+da)-1] ^
rr[dest][ya+ka];
                        end
                    end
                ka=ka+ca*2;
                if (ka >= WIDTH)
                    begin
                    da = da + 1;
                    ca = ca*2;
                    ka = ca-1;
                    rf[dest][(WIDTH+da)-1] = 0;
                    rr[dest][(WIDTH+da)-1] = 0;
                    end
                end
            end
        dataOut = rf[source];
        end
    end
endmodule
```

▪ Verilog code for a 32x16 register containing Hamming code (implementation A with autocorrection with a single error register)

```
module rf_hamming_with_autocorrection(clock, reset, writeEnable, dest, source, dataIn, dataOut,
err);

parameter WIDTH = 16;
parameter DEPTH = 32;
parameter ADDRESSWIDTH = 5;
parameter MAXBITS = 5; //log(2)WIDTH + 1

integer i, j, n, ze, m, c, k, b, d, a, s, sa, ca, ce, ka, ke, ba, be, da, de, aa, y, ya, ye, l, fix;

input clock, reset, writeEnable;
input [ADDRESSWIDTH-1 : 0] dest;
input [ADDRESSWIDTH-1 : 0] source;
input [WIDTH-1 : 0] dataIn;

output [WIDTH-1 : 0] dataOut;
output [(MAXBITS*DEPTH)-1 : 0] err;

reg [WIDTH-1 : 0] dataOut; // registered output
reg [(MAXBITS*DEPTH)-1 : 0] err; // registered error
reg [(WIDTH+MAXBITS) : 0] rf [(DEPTH-1) : 0];

//reset
always@(posedge reset)
 begin
 for(i = 0; i<(DEPTH); i=i+1)
        begin
        rf[i] = 0;
        end
 dataOut = 0;
 err = 0;
 end

// flip-flop for data-out, data-in and checking
always@(posedge clock)
 begin
 //error?
```

```
for (i=0; i<DEPTH; i=i+1)
        begin
        //fix
        if (err)
                //fix hamming code?
                begin
                ze = 0;
                for (m=0; m<MAXBITS; m=m+1)
                        begin
                        ze = ze ^ rf[i][WIDTH+m];
                        end
                if (ze != rf[i][WIDTH+MAXBITS])
                        begin
                        c = 1;
                        k = 0;
                        b = 0;
                        d = 1;
                        a = 0;
                        while (k<(WIDTH-1))
                                begin
                                for(y=0; y<c; y=y+1)
                                        begin
                                        if ((y+k)<WIDTH)
                                                begin
                                                b = b ^ rf[i][y+k];
                                                end
                                        end
                                k=k+c*2;
                                if (k >= WIDTH)
                                        begin
                                        if (b != rf[i][(WIDTH+d)-1])
                                                begin
                                                err[(i*MAXBITS+d)-1] = 1;
                                                end
                                        a = a ^ rf[i][(WIDTH+d)-1];
                                        d = d + 1;
                                        c = c*2;
                                        k = c-1;
                                        b = 0;
                                        end
                                end
                        rf[i][WIDTH+MAXBITS] = a;
        end
    else //fix data
        begin
        s = 0;
        sa = 1;
        for (n=0; n<MAXBITS; n=n+1)
            begin
            s = s + (err[(i*MAXBITS)+n]*sa);
            sa = 2 * sa;
            end
        if (s)
            begin
            rf[i][s-1] = ~rf[i][s-1];
            end
        end
/*  for (l=0; l<MAXBITS; l=l+1)
        begin
        err[i*MAXBITS+l] = 0;
        end */
    fix = 1;
    end
//scan for errors
ce = 1;
ke = 0;
be = 0;
de = 1;
while (ke<(WIDTH-1))
    begin
    for(ye=0; ye<ce; ye=ye+1)
        begin
        if ((ye+ke)<WIDTH)
            begin
            be = be ^ rf[i][ye+ke];
            end
        end
    ke=ke+ce*2;
    if (ke >= WIDTH)
        begin
        if (be != rf[i][(WIDTH+de)-1])
            begin
            err[(i*MAXBITS+de)-1] = 1;
            fix = 0;
```

```
                    end
                de = de + 1;
                ce = ce*2;
                ke = ce-1;
                be = 0;
                end
            end
        end

/*      //error not fixed, reset...
    if ((err) && (!fix))
        begin
        for (n=0; n<DEPTH; n=n+1)
            begin
            rf[n] = 0;
            end
        dataOut = 0;
        end */
    //IO
    if ((!err) && (!reset))
        begin
        if (writeEnable)
            begin
            rf[dest] = dataIn;
            ca = 1;
            ka = 0;
            da = 1;
            aa = 0;
            rf[dest][(WIDTH+da)-1] = 0;
            while (ka<(WIDTH-1))
                begin
                for(ya=0; ya<ca; ya=ya+1)
                    begin
                    if ((ya+ka)<WIDTH)
                        begin
                        rf[dest][(WIDTH+da)-1] = rf[dest][(WIDTH+da)-1] ^
rf[dest][ya+ka];
                        end
                    end
                ka=ka+ca*2;
                if (ka >= WIDTH)
                    begin
```

```
                                        aa = aa ^ rf[dest][(WIDTH+da)-1];
                                        da = da + 1;
                                        ca = ca*2;
                                        ka = ca-1;
                                        rf[dest][(WIDTH+da)-1] = 0;
                                        end
                                end
                        rf[dest][WIDTH+MAXBITS] = aa;
                        end
                dataOut = rf[source];
                end
        end
endmodule
```

▪ Verilog code for a 32x16 register containing Hamming code (implementation A with autocorrection with multiple error registers)

```
module rf_hamming_with_autocorrection_a(clock, reset, writeEnable, dest, source, dataIn, dataOut,
err);

parameter WIDTH = 16;
parameter DEPTH = 32;
parameter ADDRESSWIDTH = 5;
parameter MAXBITS = 5; //log(2)WIDTH + 1


integer i, j, iter, remainder, o, k, b, y, a, z, it, fix, l, m, ze, zee, ae, oe, be, itere, remaindere, ye, ite, s,
sa, eo, eb, ey, ea, ez, eit, eiter, eremainder, n;

input clock, reset, writeEnable;
input [ADDRESSWIDTH-1 : 0] dest;
input [ADDRESSWIDTH-1 : 0] source;
input [WIDTH-1 : 0] dataIn;

output [WIDTH-1 : 0] dataOut;
output err;

reg [WIDTH-1 : 0] dataOut; // registered output
reg [MAXBITS-1 : 0] error [(DEPTH-1) : 0]; //error matrix
reg err; // registered error
reg [(WIDTH+MAXBITS) : 0] rf [(DEPTH-1) : 0];

//reset
always@(posedge reset)
        begin
        for(i = 0; i<(DEPTH); i=i+1)
                begin
                rf[i] = 0;
                error[i] = 0;
                end
        dataOut = 0;
        err = 0;
        end

// flip-flop for data-out, data-in and checking
always@(posedge clock)
        begin
        //error?
        for (i=0; i<DEPTH; i=i+1)
                begin
                //fix
                if (error[i])
                        //fix hamming code?
                        begin
                        ze = 0;
                        for (m=0; m<MAXBITS; m=m+1)
                                begin
                                ze = ze ^ rf[i][WIDTH+m];
                                end
                        if (ze != rf[i][WIDTH+MAXBITS])
                                begin
                                oe = 1;
                                itere = 1;
                                rf[i][WIDTH+MAXBITS]= 0;
                                for (j=0; j<WIDTH; j=j+1)
                                        begin
                                        if (j == oe-1)
                                                begin
                                                be=1;
                                                zee=0;
                                                remaindere = 0;
                                                ite = itere;
                        while (ite > 0)
                            begin
                            be = be*2;
                            ite = ite -1;
                            end
                        ye = (be/2)-1;
                        for (k=j; k<WIDTH; k=k+1)
                            begin
                            while (ye < (be-1))
                                begin
                                if ((k%be) == ye)
                                    begin
                                    zee = zee + rf[i][k];
                                    remaindere = k - be;
                                    end
                                ae = ye;
                                ye = ye + 1;
                                if (remaindere < 1)
                                    begin
                                    ye = be;
                                    end
                                end
                            ye = ae + 1;
                            if (ye == (be-1))
                                begin
                                ye = (be/2)-1;
                                end
                            end
                        rf[i][(WIDTH+itere)-1] = (zee%2);
                        rf[i][WIDTH+MAXBITS] = rf[i]
[WIDTH+MAXBITS] ^ rf[i][(WIDTH+itere)-1];
                        itere = itere + 1;
                        oe = oe * 2;
                        end
                    end
                end
            else //fix data
                begin
                s = 0;
                sa = 1;
                for (k=0; k<MAXBITS; k=k+1)
                    begin
                    s = s + ((error[i][k])*sa);
                    sa = 2 * sa;
                    end
                if (s)
                    begin
                    rf[i][s-1] = ~rf[i][s-1];
                    end
                end
            error[i] = 0;
        //  err = 0;
            fix = 1;
            end
        //scan for errors
        eo = 1;
        eiter = 1;
        for (j=0; j<WIDTH; j=j+1)
            begin
            if (j == eo-1)
                begin
                eb=1;
                ez=0;
                eremainder = 0;
                eit = eiter;
                while (eit > 0)
                    begin
                    eb = eb*2;
                    eit = eit -1;
                    end
                ey = (eb/2)-1;
                for (k=j; k<WIDTH; k=k+1)
                    begin
                    while (ey < (eb-1))
                        begin
                        if ((k%eb) == ey)
                            begin
                        ez = ez + rf[i][k];
                            eremainder = k - eb;
                            end
                        ea = ey;
                        ey = ey + 1;
                        if (eremainder < 1)
                            begin
```

```
                            ey = eb;
                            end
                        end
                    ey = ea + 1;
                    if (ey == (eb-1))
                        begin
                        ey = (eb/2)-1;
                        end
                    end
                if ((ez%2) != rf[i][(WIDTH+eiter)-1])
                    begin
                    error[i][(eiter-1)] = 1;
                    err = 1;
                    fix = 0;
                    end
                eiter = eiter + 1;
                eo = eo * 2;
                end
            end

        end
    //error not fixed, reset...
/*  if ((err) && (!fix))
        begin
        for (n=0; n<DEPTH; n=n+1)
            begin
            rf[n] = 0;
            end
        dataOut = 0;
        end */
    //IO
    if ((!err) && (!reset))
        begin
        if (writeEnable)
            begin
            o = 1;
            iter = 1;
            rf[dest] = dataIn;
            rf[dest][WIDTH+MAXBITS]= 0;
            for (j=0; j<WIDTH; j=j+1)
                begin
                if (j == o-1)
```

```
                    begin
                    b=1;
                    z=0;
                    remainder = 0;
                    it = iter;
                    while (it > 0)
                        begin
                        b = b*2;
                        it = it -1;
                        end
                    y = (b/2)-1;
                    for (k=j; k<WIDTH; k=k+1)
                        begin
                        while (y < (b-1))
                            begin
                            if ((k%b) == y)
                                begin
                                z = z + rf[dest][k];
                                remainder = k - b;
                                end
                            a = y;
                            y = y + 1;
                            if (remainder < 1)
                                begin
                                y = b;
                                end
                            end
                        y = a + 1;
                        if (y == (b-1))
                            begin
                            y = (b/2)-1;
                            end
                        end
                    rf[dest][(WIDTH+iter)-1] = (z%2);
                    rf[dest][WIDTH+MAXBITS] = rf[dest][WIDTH+MAXBITS]
^ rf[dest][(WIDTH+iter)-1];
                    iter = iter + 1;
                    o = o * 2;
                    end
                end
            end
        dataOut = rf[source];
```

```
                end
        end
endmodule
```

▪ Verilog code for a 32x16 register containing Hamming code (implementation B with autocorrection with multiple error registers)

```
module rf_hamming_with_autocorrection_a_new(clock, reset, writeEnable, dest, source, dataIn,
dataOut, err);

parameter WIDTH = 16;
parameter DEPTH = 32;
parameter ADDRESSWIDTH = 5;
parameter MAXBITS = 5; //log(2)WIDTH + 1

integer i, j, n, ze, m, c, k, b, d, a, s, sa, ca, ce, ka, ke, ba, be, da, de, aa, y, ya, ye, l, fix;

input clock, reset, writeEnable;
input [ADDRESSWIDTH-1 : 0] dest;
input [ADDRESSWIDTH-1 : 0] source;
input [WIDTH-1 : 0] dataIn;

output [WIDTH-1 : 0] dataOut;
output err;

reg [WIDTH-1 : 0] dataOut; // registered output
reg [MAXBITS-1 : 0] error [(DEPTH-1) : 0]; //error matrix
reg err; // registered error
reg [(WIDTH+MAXBITS) : 0] rf [(DEPTH-1) : 0];

//reset
always@(posedge reset)
 begin
 for(i = 0; i<(DEPTH); i=i+1)
        begin
        rf[i] = 0;
        error[i] = 0;
        end
 dataOut = 0;
 err = 0;
 end

// flip-flop for data-out, data-in and checking
always@(posedge clock)
 begin
 //error?
    for (i=0; i<DEPTH; i=i+1)
        begin
        //fix
        if (error[i])
            //fix hamming code?
            begin
            ze = 0;
            for (m=0; m<MAXBITS; m=m+1)
                begin
                ze = ze ^ rf[i][WIDTH+m];
                end
            if (ze != rf[i][WIDTH+MAXBITS])
                begin
                c = 1;
                k = 0;
                d = 1;
                a = 0;
                rf[i][(WIDTH+d)-1] = 0;
                while (k<(WIDTH-1))
                    begin
                    for(y=0; y<c; y=y+1)
                        begin
                        if ((y+k)<WIDTH)
                            begin
                            rf[i][(WIDTH+d)-1] = rf[i][(WIDTH+d)-1] ^
rf[i][y+k];

                            end
                        end
                    k=k+c*2;
                    if (k >= WIDTH)
                        begin
                        a = a ^ rf[i][(WIDTH+d)-1];
                        d = d + 1;
                        c = c*2;
                        k = c-1;
                        rf[i][(WIDTH+d)-1] = 0;
                        end
                    end
                rf[i][WIDTH+MAXBITS] = a;
                end
            else //fix data
                begin
                s = 0;
                sa = 1;
                for (k=0; k<MAXBITS; k=k+1)
                    begin
                    s = s + ((error[i][k])*sa);
                    sa = 2 * sa;
                    end
                if (s)
                    begin
                    rf[i][s-1] = ~rf[i][s-1];
                    end
                end
            error[i] = 0;
        //  err = 0;
            fix = 1;
            end
        //scan for errors
        ce = 1;
        ke = 0;
        be = 0;
        de = 1;
        while (ke<(WIDTH-1))
            begin
            for(ye=0; ye<ce; ye=ye+1)
                begin
                if ((ye+ke)<WIDTH)
                    begin
                    be = be ^ rf[i][ye+ke];
                    end
                end
            ke=ke+ce*2;
            if (ke >= WIDTH)
                begin
                if (be != rf[i][(WIDTH+de)-1])
                    begin
                    error[i][de-1] = 1;
```

```
                    err = 1;
                    fix = 0;
                    end
                de = de + 1;
                ce = ce*2;
                ke = ce-1;
                be = 0;
                end
            end
        end
    //error not fixed, reset...
/*  if ((err) && (!fix))
        begin
        for (n=0; n<DEPTH; n=n+1)
            begin
            rf[n] = 0;
            end
        dataOut = 0;
        end */
    //IO
    if ((!err) && (!reset))
        begin
        if (writeEnable)
            begin
            rf[dest] = dataIn;
            ca = 1;
            ka = 0;
            da = 1;
            aa = 0;
            rf[dest][(WIDTH+da)-1] = 0;
            while (ka<(WIDTH-1))
                begin
                for(ya=0; ya<ca; ya=ya+1)
                    begin
                    if ((ya+ka)<WIDTH)
                        begin
                        rf[dest][(WIDTH+da)-1] = rf[dest][(WIDTH+da)-1] ^
rf[dest][ya+ka];
                        end
                    end
                ka=ka+ca*2;
                if (ka >= WIDTH)
```

```
                    begin
                    aa = aa ^ rf[dest][(WIDTH+da)-1];
                    da = da + 1;
                    ca = ca*2;
                    ka = ca-1;
                    rf[dest][(WIDTH+da)-1] = 0;
                    end
                end
            rf[dest][WIDTH+MAXBITS] = aa;
            end
        dataOut = rf[source];
        end
    end
endmodule
```

▪ Verilog code for a 32x16 register containing Hamming code (implementation A with autocorrection with multiple error registers and additional parity checks)

```
module rf_hamming_with_autocorrection_a_additional_check(clock, reset, writeEnable, dest,
source, dataIn, dataOut, err);

parameter WIDTH = 16;
parameter DEPTH = 32;
parameter ADDRESSWIDTH = 5;
parameter MAXBITS = 5; //log(2)WIDTH + 1


integer i, j, iter, remainder, o, k, b, y, a, z, it, fix, l, m, ze, zee, ae, oe, be, itere, remaindere, ye, ite, s,
sa, eo, eb, ey, ea, ez, eit, eiter, eremainder, n;

input clock, reset, writeEnable;
input [ADDRESSWIDTH-1 : 0] dest;
input [ADDRESSWIDTH-1 : 0] source;
input [WIDTH-1 : 0] dataIn;

output [WIDTH-1 : 0] dataOut;
output err;

reg [WIDTH-1 : 0] dataOut; // registered output
reg [MAXBITS-1 : 0] error [(DEPTH-1) : 0]; //error matrix
reg err; // registered error
reg [(WIDTH+MAXBITS) : 0] rf [(DEPTH-1) : 0];

//reset
always@(posedge reset)
     begin
     for(i = 0; i<(DEPTH); i=i+1)
        begin
         rf[i] = 0;
        error[i] = 0;
        end
    dataOut = 0;
    err = 0;
    end

// flip-flop for data-out, data-in and checking
always@(posedge clock)
    begin
    //error?
    for (i=0; i<DEPTH; i=i+1)
        begin
        //fix
        if (error[i])
            //fix hamming code?
            begin
            ze = 0;
            for (m=0; m<MAXBITS; m=m+1)
                begin
                ze = ze ^ rf[i][WIDTH+m];
                end
            if (ze != rf[i][WIDTH+MAXBITS])
                begin
                oe = 1;
                itere = 1;
                rf[i][WIDTH+MAXBITS]= 0;
                for (j=0; j<WIDTH; j=j+1)
                    begin
                    if (j == oe-1)
                        begin
                        be=1;
                        zee=0;
                        remaindere = 0;
                        ite = itere;
                        while (ite > 0)
                            begin
                            be = be*2;
                            ite = ite -1;
                            end
                        ye = (be/2)-1;
                        for (k=j; k<WIDTH; k=k+1)
                            begin
                            while (ye < (be-1))
                                begin
                                if ((k%be) == ye)
                                    begin
                                    zee = zee + rf[i][k];
                                    remaindere = k - be;
                                    end
                                ae = ye;
                                ye = ye + 1;
                                if (remaindere < 1)
                                    begin
                                    ye = be;
                                    end
                                end
                            ye = ae + 1;
                            if (ye == (be-1))
                                begin
                                ye = (be/2)-1;
                                end
                            end
                        rf[i][(WIDTH+itere)-1] = (zee%2);
                        rf[i][WIDTH+MAXBITS] = rf[i]
[WIDTH+MAXBITS] ^ rf[i][(WIDTH+itere)-1];
                        itere = itere + 1;
                        oe = oe * 2;
                        end
                    end
                end
            else //fix data
                begin
                s = 0;
                sa = 1;
                for (k=0; k<MAXBITS; k=k+1)
                    begin
                    s = s + ((error[i][k])*sa);
                    sa = 2 * sa;
                    end
                if (s)
                    begin
                    rf[i][s-1] = ~rf[i][s-1];
                    end
                end
            error[i] = 0;
        //  err = 0;
            fix = 1;
            end
        //scan for errors
        eo = 1;
        eiter = 1;
        for (j=0; j<WIDTH; j=j+1)
            begin
            if (j == eo-1)
                begin
                eb=1;
                ez=0;
                eremainder = 0;
                eit = eiter;
                while (eit > 0)
                    begin
                    eb = eb*2;
                    eit = eit -1;
                    end
                ey = (eb/2)-1;
                for (k=j; k<WIDTH; k=k+1)
                    begin
                    while (ey < (eb-1))
                        begin
                        if ((k%eb) == ey)
                            begin
                        ez = ez + rf[i][k];
                            eremainder = k - eb;
                            end
                        ea = ey;
                        ey = ey + 1;
                        if (eremainder < 1)
```

```
                        begin
                        ey = eb;
                        end
                    end
                ey = ea + 1;
                if (ey == (eb-1))
                    begin
                    ey = (eb/2)-1;
                    end
                end
            if ((ez%2) != rf[i][(WIDTH+eiter)-1])
                begin
                error[i][(eiter-1)] = 1;
                err = 1;
                fix = 0;
                end
            eiter = eiter + 1;
            eo = eo * 2;
            end
        end

    end
//error not fixed, reset...
if ((err) && (!fix))
    begin
    for (n=0; n<DEPTH; n=n+1)
        begin
        rf[n] = 0;
        end
    dataOut = 0;
    end
//IO
if ((!err) && (!reset))
    begin
    if (writeEnable)
        begin
        o = 1;
        iter = 1;
        rf[dest] = dataIn;
        rf[dest][WIDTH+MAXBITS]= 0;
        for (j=0; j<WIDTH; j=j+1)
            begin
```

```
                if (j == o-1)
                    begin
                    b=1;
                    z=0;
                    remainder = 0;
                    it = iter;
                    while (it > 0)
                        begin
                        b = b*2;
                        it = it -1;
                        end
                    y = (b/2)-1;
                    for (k=j; k<WIDTH; k=k+1)
                        begin
                        while (y < (b-1))
                            begin
                            if ((k%b) == y)
                                begin
                                z = z + rf[dest][k];
                                remainder = k - b;
                                end
                            a = y;
                            y = y + 1;
                            if (remainder < 1)
                                begin
                                y = b;
                                end
                            end
                        y = a + 1;
                        if (y == (b-1))
                            begin
                            y = (b/2)-1;
                            end
                        end
                    rf[dest][(WIDTH+iter)-1] = (z%2);
                    rf[dest][WIDTH+MAXBITS] = rf[dest][WIDTH+MAXBITS]
^ rf[dest][(WIDTH+iter)-1];
                    iter = iter + 1;
                    o = o * 2;
                    end
                end
            end
```

```
                    dataOut = rf[source];
                    end
          end
endmodule
```

▪ Verilog code for a 32x16 register containing Hamming code (implementation B with autocorrection with multiple error registers and additional parity checks)

```
module rf_hamming_with_autocorrection_a_additional_check_new(clock, reset, writeEnable, dest,
source, dataIn, dataOut, err);

parameter WIDTH = 16;
parameter DEPTH = 32;
parameter ADDRESSWIDTH = 5;
parameter MAXBITS = 5; //log(2)WIDTH + 1

integer i, j, n, ze, m, c, k, b, d, a, s, sa, ca, ce, ka, ke, ba, be, da, de, aa, y, ya, ye, l, fix;

input clock, reset, writeEnable;
input [ADDRESSWIDTH-1 : 0] dest;
input [ADDRESSWIDTH-1 : 0] source;
input [WIDTH-1 : 0] dataIn;

output [WIDTH-1 : 0] dataOut;
output err;

reg [WIDTH-1 : 0] dataOut; // registered output
reg [MAXBITS-1 : 0] error [(DEPTH-1) : 0]; //error matrix
reg err; // registered error
reg [(WIDTH+MAXBITS) : 0] rf [(DEPTH-1) : 0];

//reset
always@(posedge reset)
 begin
 for(i = 0; i<(DEPTH); i=i+1)
           begin
           rf[i] = 0;
           error[i] = 0;
           end
 dataOut = 0;
 err = 0;
 end

// flip-flop for data-out, data-in and checking
always@(posedge clock)
 begin
 //error?
    for (i=0; i<DEPTH; i=i+1)
        begin
        //fix
        if (error[i])
            //fix hamming code?
            begin
            ze = 0;
            for (m=0; m<MAXBITS; m=m+1)
                begin
                ze = ze ^ rf[i][WIDTH+m];
                end
            if (ze != rf[i][WIDTH+MAXBITS])
                begin
                c = 1;
                k = 0;
                d = 1;
                a = 0;
                rf[i][(WIDTH+d)-1] = 0;
                while (k<(WIDTH-1))
                    begin
                    for(y=0; y<c; y=y+1)
                        begin
                        if ((y+k)<WIDTH)
                            begin
                            rf[i][(WIDTH+d)-1] = rf[i][(WIDTH+d)-1] ^
rf[i][y+k];
                            end
                        end
                    k=k+c*2;
                    if (k >= WIDTH)
                        begin
                        a = a ^ rf[i][(WIDTH+d)-1];
                        d = d + 1;
                        c = c*2;
                        k = c-1;
                        rf[i][(WIDTH+d)-1] = 0;
                        end
                    end
                rf[i][WIDTH+MAXBITS] = a;
                end
            else //fix data
                begin
                s = 0;
                sa = 1;
                for (n=0; n<MAXBITS; n=n+1)
                    begin
                    s = s + ((error[i][n])*sa);
                    sa = 2 * sa;
                    end
                if (s)
                    begin
                    rf[i][s-1] = ~rf[i][s-1];
                    end
                end
            error[i] = 0;
        //  err = 0;
            fix = 1;
            end
        //scan for errors
        ce = 1;
        ke = 0;
        be = 0;
        de = 1;
        while (ke<(WIDTH-1))
            begin
            for(ye=0; ye<ce; ye=ye+1)
                begin
                if ((ye+ke)<WIDTH)
                    begin
                    be = be ^ rf[i][ye+ke];
                    end
                end
            ke=ke+ce*2;
            if (ke >= WIDTH)
                begin
                if (be != rf[i][(WIDTH+de)-1])
                    begin
```

```
                    error[i][de-1] = 1;
                    err = 1;
                    fix = 0;
                    end
                de = de + 1;
                ce = ce*2;
                ke = ce-1;
                be = 0;
                end
            end
        end
    //error not fixed, reset...
    if ((err) && (!fix))
        begin
        for (n=0; n<DEPTH; n=n+1)
            begin
            rf[n] = 0;
            end
        dataOut = 0;
        end
    //IO
    if ((!err) && (!reset))
        begin
        if (writeEnable)
            begin
            rf[dest] = dataIn;
            ca = 1;
            ka = 0;
            da = 1;
            aa = 0;
            rf[dest][(WIDTH+da)-1] = 0;
            while (ka<(WIDTH-1))
                begin
                for(ya=0; ya<ca; ya=ya+1)
                    begin
                    if ((ya+ka)<WIDTH)
                        begin
                        rf[dest][(WIDTH+da)-1] = rf[dest][(WIDTH+da)-1] ^
rf[dest][ya+ka];
                        end
                    end
                ka=ka+ca*2;
```

```
                                if (ka >= WIDTH)
                                        begin
                                        aa = aa ^ rf[dest][(WIDTH+da)-1];
                                        da = da + 1;
                                        ca = ca*2;
                                        ka = ca-1;
                                        rf[dest][(WIDTH+da)-1] = 0;
                                        end
                                end
                        rf[dest][WIDTH+MAXBITS] = aa;
                        end
                dataOut = rf[source];
                end
        end
endmodule
```

▪ Verilog code for a 32x16 register integrating CRC per row and column of the register

```
module rf_crc_row_by_column(clock, reset, writeEnable, dest, source, dataIn, dataOut, err);

parameter WIDTH = 16;
parameter DEPTH = 32;
parameter ADDRESSWIDTH = 5;
parameter DIVISORROW = 3;
parameter DIVROWBITS = 2;
parameter DIVISORCOL = 3;
parameter DIVCOLBITS = 2;

integer i,j,k,l,fix;

input clock, reset, writeEnable;
input [ADDRESSWIDTH-1 : 0] dest;
input [ADDRESSWIDTH-1 : 0] source;
input [WIDTH-1 : 0] dataIn;

output [WIDTH-1 : 0] dataOut;
output err;

reg [WIDTH-1 : 0] dataOut; // registered output
reg err; // registered error
reg [DEPTH-1 : 0] errd;
reg [WIDTH-1 : 0] errw;
reg [(WIDTH+DIVROWBITS) : 0] rf [(DEPTH+DIVCOLBITS) : 0];
reg [DEPTH+DIVCOLBITS : 0] help;

//reset
always@(posedge reset)
         begin
         for(i = 0; i<(DEPTH+DIVCOLBITS)+1; i=i+1)
                 begin
                rf[i] = 0;
                end
        dataOut = 0;
        errd = 0;
        errw = 0;
        err = 0;
        end

// flip-flop for data-out, data-in and checking
always@(posedge clock)
        begin
        if (err) //fix?
                begin
                for (l=0; l<DEPTH; l=l+1)
                        begin
                        if (errd[l])
                                begin
                                for (k=0; k<WIDTH; k=k+1)
                                        begin
                                        if (errw[k])
                                                begin
                                                rf[l][k] = ~rf[l][k];
                                                end
                                        end
                                end
                        end
                fix = 1;
                end
        for (i=0; i<DEPTH; i=i+1) //error row?
                begin
                if ((rf[i][WIDTH-1:0])%DIVISORROW != rf[i]
[(WIDTH+DIVROWBITS):WIDTH])
```

```
            begin
            err = 1;
            errd[i] = 1;
            fix = 0;
            end
        else
            begin
            errd[i] = 0;
            end
        end
    for (k=0; k<WIDTH; k=k+1) //error column?
        begin
        help = 0;
        for (j=0; j<DEPTH+DIVCOLBITS+1; j=j+1)
            begin
            help[j] = rf[j][k];
            end
        if ((help[DEPTH-1:0])%DIVISORCOL != help[(DEPTH+DIVCOLBITS):DEPTH])
            begin
            err = 1;
            errw[k] = 1;
            fix = 0;
            end
        else
            begin
            errw[k] = 0;
            end
        end
    if ((!errw) && (!errd)) //no error?
        begin
        err = 0;
        end
/*  if ((err) && (!fix)) //not fixed error?
        begin
        for (j=0; j<(DEPTH+DIVCOLBITS)+1; j=j+1)
            begin
            rf[j] = 0;
            end
        dataOut = 0;
        end     */
    if ((!err) && (!reset)) //write & read
        begin
```

```
			if (writeEnable)
				begin
				for (j=0; j<WIDTH; j=j+1)
					begin
					rf[dest][j] = dataIn[j];
					help = 0;
					for (l=0; l<DEPTH-1; l=l+1)
						begin
						help[l] = rf[l][j];
						end
					help[(DEPTH+DIVCOLBITS):DEPTH] = (help[DEPTH-1:0])
%DIVISORCOL;
					for (l=DEPTH; l<DEPTH+DIVCOLBITS+1; l=l+1)
						begin
						rf[l][j] = help[l];
						end
					end
				rf[dest][(WIDTH+DIVROWBITS):WIDTH]=((rf[dest][WIDTH-1:0])
%DIVISORROW);
				end
			dataOut = rf[source];
			end
		end
endmodule
```

▪ Verilog code for a 32x16 register incorporating checksums per row and column of the register

```
module rf_cs_row_by_column(clock, reset, writeEnable, dest, source, dataIn, dataOut, err);

parameter WIDTH = 16;
parameter DEPTH = 32;
parameter ADDRESSWIDTH = 5;
parameter DIVISORROW = 3;
parameter DIVROWBITS = 2;
parameter DIVISORCOL = 3;
parameter DIVCOLBITS = 2;

integer i,j,k,l,m,n,b,a,fix;

input clock, reset, writeEnable;
input [ADDRESSWIDTH-1 : 0] dest;
input [ADDRESSWIDTH-1 : 0] source;
input [WIDTH-1 : 0] dataIn;

output [WIDTH-1 : 0] dataOut;
output err;

reg [WIDTH-1 : 0] dataOut; // registered output
reg err; // registered error
reg [DEPTH-1 : 0] errd;
reg [WIDTH-1 : 0] errw;
reg [(WIDTH+DIVROWBITS)-1 : 0] rf [(DEPTH+DIVCOLBITS)-1 : 0];
reg [DIVCOLBITS-1 : 0] help;

//reset
always@(posedge reset)
        begin
        for(i = 0; i<(DEPTH+DIVCOLBITS); i=i+1)
                begin
                rf[i] = 0;
                end
        dataOut = 0;
        errd = 0;
        errw = 0;
        err = 0;
        end

// flip-flop for data-out, data-in and checking
always@(posedge clock)
        begin
        if (err) //fix?
                begin
                for (l=0; l<DEPTH; l=l+1)
                        begin
                        if (errd[l])
                                begin
                                for (k=0; k<WIDTH; k=k+1)
                                        begin
                                        if (errw[k])
                                                begin
                                                rf[l][k] = ~rf[l][k];
                                                end
                end
            end
        end
    fix = 1;
    end
for (i=0; i<DEPTH; i=i+1) //error row?
    begin
    m=0;
    for (j=0; j<WIDTH; j=j+1)
        begin
        m = m+rf[i][j];
        end
    if ((m%DIVISORROW) != rf[i][(WIDTH+DIVROWBITS)-1:WIDTH])
        begin
        err = 1;
        errd[i] = 1;
        fix = 0;
        end
    else
        begin
        errd[i] = 0;
        end
    end
for (k=0; k<WIDTH; k=k+1) //error column?
    begin
    a = 0;
    help = 0;
    for (j=0; j<DEPTH; j=j+1)
        begin
        a = a + rf[j][k];
        end
    for (l=DEPTH; l<DEPTH+DIVCOLBITS; l=l+1)
        begin
        help[l-DEPTH] = rf[l][k];
        end
    if (help != (a%DIVISORCOL))
        begin
        err = 1;
        errw[k] = 1;
        fix = 0;
        end
    else
```

```
                        begin
                        errw[k] = 0;
                        end
                end
        if ((!errw) && (!errd)) //no error?
                begin
                err = 0;
                end
/*      if ((err) && (!fix)) //not fixed error?
                begin
                for (j=0; j<(DEPTH+DIVCOLBITS); j=j+1)
                        begin
                        rf[j] = 0;
                        end
                dataOut = 0;
                end     */
        if ((!err) && (!reset)) //write & read
                begin
                if (writeEnable)
                        begin
                        n = 0;
                        for (j=0; j<WIDTH; j=j+1)
                                begin
                                rf[dest][j] = dataIn[j];
                                n = n+rf[dest][j];
                                b = 0;
                                help = 0;
                                for (l=0; l<DEPTH; l=l+1)
                                        begin
                                        b = b + rf[l][j];
                                        end
                                help = (b%DIVISORCOL);
                                for (l=DEPTH; l<DEPTH+DIVCOLBITS; l=l+1)
                                        begin
                                        rf[l][j] = help[l-DEPTH];
                                        end
                                end
                        rf[dest][(WIDTH+DIVROWBITS)-1:WIDTH]=(n%DIVISORROW);
                        end
                dataOut = rf[source];
                end
        end
endmodule
```

▪ Verilog code for a 32x16 register containing single parity bits per row and column of the register

```
module rf_pb_row_by_column(clock, reset, writeEnable, dest, source, dataIn, dataOut, err);

parameter WIDTH = 16;
parameter DEPTH = 32;
parameter ADDRESSWIDTH = 5;

integer i,j,k,l,m,n,a,b,fix;

input clock, reset, writeEnable;
input [ADDRESSWIDTH-1 : 0] dest;
input [ADDRESSWIDTH-1 : 0] source;
input [WIDTH-1 : 0] dataIn;

output [WIDTH-1 : 0] dataOut;
output err;

reg [WIDTH-1 : 0] dataOut; // registered output
reg err; // registered error
reg [DEPTH-1 : 0] errd;
reg [WIDTH-1 : 0] errw;
reg [WIDTH : 0] rf [DEPTH : 0];

//reset
always@(posedge reset)
        begin
        for(i = 0; i<DEPTH+1; i=i+1)
                begin
                rf[i] = 0;
                end
        dataOut = 0;
        errd = 0;
        errw = 0;
        err = 0;
        end
```

```
// flip-flop for data-out, data-in and checking
always@(posedge clock)
        begin
        if (err) //fix?
                begin
                for (l=0; l<DEPTH; l=l+1)
                        begin
                        if (errd[l])
                                begin
                                for (k=0; k<WIDTH; k=k+1)
                                        begin
                                        if (errw[k])
                                                begin
                                                rf[l][k] = ~rf[l][k];
                                                end
                                        end
                                end
                        end
                fix = 1;
                end
        for (i=0; i<DEPTH; i=i+1) //error row?
                begin
                m=0;
                for (j=0; j<WIDTH; j=j+1)
                        begin
                        m = m+rf[i][j];
                        end
                if ((m%2) != rf[i][WIDTH])
                        begin
                        err = 1;
                        errd[i] = 1;
                        fix = 0;
                        end
                else
                        begin
                        errd[i] = 0;
                        end
                end
        for (k=0; k<WIDTH; k=k+1) //error column?
                begin
                a = 0;
                for (j=0; j<DEPTH; j=j+1)
```

```
            begin
            a = a + rf[j][k];
            end
        if (rf[DEPTH][k] != (a%2))
            begin
            err = 1;
            errw[k] = 1;
            fix = 0;
            end
        else
            begin
            errw[k] = 0;
            end
        end
    if ((!errw) && (!errd)) //no error?
        begin
        err = 0;
        end
/*  if ((err) && (!fix)) //not fixed error?
        begin
        for (j=0; j<DEPTH+1; j=j+1)
            begin
            rf[j] = 0;
            end
        dataOut = 0;
        end     */
    if ((!err) && (!reset)) //write & read
        begin
        if (writeEnable)
            begin
            n = 0;
            for (j=0; j<WIDTH; j=j+1)
                begin
                rf[dest][j] = dataIn[j];
                n = n+rf[dest][j];
                b = 0;
                for (l=0; l<DEPTH; l=l+1)
                    begin
                    b = b + rf[l][j];
                    end
                rf[DEPTH][j] = (b%2);
                end
```

```
                        rf[dest][WIDTH]=(n%2);
                        end
                dataOut = rf[source];
                end
        end
endmodule
```

▪ Verilog code for a 32x16 register integrating CRC per row and column of the register with additional parity checks

```
module rf_crc_row_by_column_addpar(clock, reset, writeEnable, dest, source, dataIn, dataOut,
err);

parameter WIDTH = 16;
parameter DEPTH = 32;
parameter ADDRESSWIDTH = 5;
parameter DIVISORROW = 3;
parameter DIVROWBITS = 2;
parameter DIVISORCOL = 3;
parameter DIVCOLBITS = 2;

integer i,j,k,l,fix,a,b,m,n,p;

input clock, reset, writeEnable;
input [ADDRESSWIDTH-1 : 0] dest;
input [ADDRESSWIDTH-1 : 0] source;
input [WIDTH-1 : 0] dataIn;

output [WIDTH-1 : 0] dataOut;
output err;

reg [WIDTH-1 : 0] dataOut; // registered output
reg err; // registered error
reg [DEPTH-1 : 0] errd;
reg [WIDTH-1 : 0] errw;
reg [(WIDTH+DIVROWBITS) : 0] rf [(DEPTH+DIVCOLBITS) : 0];
reg [DEPTH+DIVCOLBITS : 0] help;

//reset
always@(posedge reset)
        begin
        for(i = 0; i<(DEPTH+DIVCOLBITS)+1; i=i+1)
                begin
                rf[i] = 0;
                end
        dataOut = 0;
        errd = 0;
        errw = 0;
        err = 0;
        end

// flip-flop for data-out, data-in and checking
always@(posedge clock)
        begin
        if (err) //fix?
                begin
                p=0;
                for (j=WIDTH; j<WIDTH+DIVROWBITS+1; j=j+1) //error in control bits?
                        begin
                        m = 0;
                        for (l=0; l<DEPTH; l=l+1)
                                begin
                                m = m + rf[l][j];
                                end
                        for (i=DEPTH; i<DEPTH+DIVCOLBITS+1; i=i+1)
                                begin
                                n = 0;
                                for (k=0; k<WIDTH; k=k+1)
                                        begin
                                        n = n + rf[i][k];
                                        end
                                if (rf[i][j] != (m%2) ^ (n%2))
                                        begin
                                        p=1;
                                        end
                                end
                        end
                if (p)  //fix control bits
                        begin
                        for (k=0; k<DEPTH; k=k+1)
                                begin
                                rf[k][(WIDTH+DIVROWBITS):WIDTH]=((rf[k][WIDTH-1:0])
%DIVISORROW);
```

```
                    end
                for (j=0; j<WIDTH; j=j+1)
                    begin
                    help = 0;
                    for (l=0; l<DEPTH; l=l+1)
                        begin
                        help[l] = rf[l][j];
                        end
                    help[(DEPTH+DIVCOLBITS):DEPTH] = (help[DEPTH-1:0])
%DIVISORCOL;
                    for (l=DEPTH; l<DEPTH+DIVCOLBITS+1; l=l+1)
                        begin
                        rf[l][j] = help[l];
                        end
                    end
                end
            else    //fix data
                begin
                for (l=0; l<DEPTH; l=l+1)
                    begin
                    if (errd[l])
                        begin
                        for (k=0; k<WIDTH; k=k+1)
                            begin
                            if (errw[k])
                                begin
                                rf[l][k] = ~rf[l][k];
                                end
                            end
                        end
                    end
                end
            fix = 1;
            end

        for (i=0; i<DEPTH; i=i+1) //error row?
            begin
            if ((rf[i][WIDTH-1:0])%DIVISORROW != rf[i]
[(WIDTH+DIVROWBITS):WIDTH])
                begin
                err = 1;
                errd[i] = 1;
```

```
                        fix = 0;
                        end
                else
                        begin
                        errd[i] = 0;
                        end
                end
        for (k=0; k<WIDTH; k=k+1) //error column?
                begin
                help = 0;
                for (j=0; j<(DEPTH+DIVCOLBITS)+1; j=j+1)
                        begin
                        help[j] = rf[j][k];
                        end
                if ((help[DEPTH-1:0])%DIVISORCOL != help[(DEPTH+DIVCOLBITS):DEPTH])
                        begin
                        err = 1;
                        errw[k] = 1;
                        fix = 0;
                        end
                else
                        begin
                        errw[k] = 0;
                        end
                end
        if ((!errw) && (!errd)) //no error?
                begin
                err = 0;
                end
/*      if ((err) && (!fix)) //not fixed error?
                begin
                for (j=0; j<(DEPTH+DIVCOLBITS)+1; j=j+1)
                        begin
                        rf[j] = 0;
                        end
                dataOut = 0;
                end      */
        if ((!err) && (!reset)) //write & read
                begin
                if (writeEnable)
                        begin
                        for (j=0; j<WIDTH; j=j+1)
                begin
                rf[dest][j] = dataIn[j];
                help = 0;
                for (l=0; l<DEPTH; l=l+1)
                    begin
                    help[l] = rf[l][j];
                    end
                help[(DEPTH+DIVCOLBITS):DEPTH] = (help[DEPTH-1:0])
%DIVISORCOL;
                for (l=DEPTH; l<DEPTH+DIVCOLBITS+1; l=l+1)
                    begin
                    rf[l][j] = help[l];
                    end
                end
            rf[dest][(WIDTH+DIVROWBITS):WIDTH]=((rf[dest][WIDTH-1:0])
%DIVISORROW);
            for (k=WIDTH; k<WIDTH+DIVROWBITS+1; k=k+1)
                begin
                a = 0;
                for (l=0; l<DEPTH; l=l+1)
                    begin
                    a = a + rf[l][k];
                    end
                for (i=DEPTH; i<DEPTH+DIVCOLBITS+1; i=i+1)
                    begin
                    b = 0;
                    for (j=0; j<WIDTH; j=j+1)
                        begin
                        b = b + rf[i][j];
                        end
                    rf[i][k] = ((a%2) ^ (b%2));
                    end
                end
            end
        dataOut = rf[source];
        end
    end
endmodule
```

▪ Verilog code for a 32x16 register incorporating checksums per row and column of the register with additional parity checks

```
module rf_cs_row_by_column_addpar(clock, reset, writeEnable, dest, source, dataIn, dataOut, err);

parameter WIDTH = 16;
parameter DEPTH = 32;
parameter ADDRESSWIDTH = 5;
parameter DIVISORROW = 3;
parameter DIVROWBITS = 2;
parameter DIVISORCOL = 3;
parameter DIVCOLBITS = 2;

integer i,j,k,l,m,n,b,a,c,d,p,o,q,r,s,fix;

input clock, reset, writeEnable;
input [ADDRESSWIDTH-1 : 0] dest;
input [ADDRESSWIDTH-1 : 0] source;
input [WIDTH-1 : 0] dataIn;

output [WIDTH-1 : 0] dataOut;
output err;

reg [WIDTH-1 : 0] dataOut; // registered output
reg err; // registered error
reg [DEPTH-1 : 0] errd;
reg [WIDTH-1 : 0] errw;
reg [(WIDTH+DIVROWBITS)-1 : 0] rf [(DEPTH+DIVCOLBITS)-1 : 0];
reg [DIVCOLBITS-1 : 0] help;

//reset
always@(posedge reset)
        begin
        for(i = 0; i<(DEPTH+DIVCOLBITS); i=i+1)
                begin
                rf[i] = 0;
                end
        dataOut = 0;
        errd = 0;
        errw = 0;
        err = 0;
        end

// flip-flop for data-out, data-in and checking
always@(posedge clock)
        begin
        if (err) //fix?
                begin
                p=0;
                for (j=WIDTH; j<WIDTH+DIVROWBITS; j=j+1) //error in control bits?
                        begin
                        o = 0;
                        for (l=0; l<DEPTH; l=l+1)
                                begin
                                o = o + rf[l][j];
                                end
                        for (i=DEPTH; i<DEPTH+DIVCOLBITS; i=i+1)
                                begin
                                q = 0;
                                for (k=0; k<WIDTH; k=k+1)
                                        begin
                                        q = q + rf[i][k];
                                        end
                                if (rf[i][j] != (o%2) ^ (q%2))
                                        begin
                                        p=1;
                                        end
                                end
                        end
                if (p)  //fix control bits
                        begin
                        for (k=0; k<DEPTH; k=k+1)
                                begin
                                r = 0;
                                for (j=0; j<WIDTH; j=j+1)
                                        begin
                                        r = r+rf[k][j];
                                        end
                                rf[k][(WIDTH+DIVROWBITS)-1:WIDTH]=(r%DIVISORROW);
                                end
                        for (j=0; j<WIDTH; j=j+1)
                                begin
```

```
            s = 0;
            help = 0;
            for (l=0; l<DEPTH; l=l+1)
                begin
                s = s + rf[l][j];
                end
            help = (s%DIVISORCOL);
            for (l=DEPTH; l<DEPTH+DIVCOLBITS; l=l+1)
                begin
                rf[l][j] = help[l-DEPTH];
                end
            end
        end
    else    //fix data
        begin
        for (l=0; l<DEPTH; l=l+1)
            begin
            if (errd[l])
                begin
                for (k=0; k<WIDTH; k=k+1)
                    begin
                    if (errw[k])
                        begin
                        rf[l][k] = ~rf[l][k];
                        end
                    end
                end
            end
        end
    fix = 1;
    end
for (i=0; i<DEPTH; i=i+1) //error row?
    begin
    m=0;
    for (j=0; j<WIDTH; j=j+1)
        begin
        m = m+rf[i][j];
        end
    if ((m%DIVISORROW) != rf[i][(WIDTH+DIVROWBITS)-1:WIDTH])
        begin
        err = 1;
        errd[i] = 1;
```

```
                fix = 0;
                end
            else
                begin
                errd[i] = 0;
                end
            end
        for (k=0; k<WIDTH; k=k+1) //error column?
            begin
            a = 0;
            help = 0;
            for (j=0; j<DEPTH; j=j+1)
                begin
                a = a + rf[j][k];
                end
            for (l=DEPTH; l<DEPTH+DIVCOLBITS; l=l+1)
                begin
                help[l-DEPTH] = rf[l][k];
                end
            if (help != (a%DIVISORCOL))
                begin
                err = 1;
                errw[k] = 1;
                fix = 0;
                end
            else
                begin
                errw[k] = 0;
                end
            end
        if ((!errw) && (!errd)) //no error?
            begin
            err = 0;
            end
/*      if ((err) && (!fix)) //not fixed error?
            begin
            for (j=0; j<(DEPTH+DIVCOLBITS); j=j+1)
                begin
                rf[j] = 0;
                end
            dataOut = 0;
            end      */
```

```
if ((!err) && (!reset)) //write & read
        begin
        if (writeEnable)
                begin
                n = 0;
                for (j=0; j<WIDTH; j=j+1)
                        begin
                        rf[dest][j] = dataIn[j];
                        n = n+rf[dest][j];
                        b = 0;
                        help = 0;
                        for (l=0; l<DEPTH; l=l+1)
                                begin
                                b = b + rf[l][j];
                                end
                        help = (b%DIVISORCOL);
                        for (l=DEPTH; l<DEPTH+DIVCOLBITS; l=l+1)
                                begin
                                rf[l][j] = help[l-DEPTH];
                                end
                        end
                rf[dest][(WIDTH+DIVROWBITS)-1:WIDTH]=(n%DIVISORROW);
                for (k=WIDTH; k<WIDTH+DIVROWBITS; k=k+1)
                        begin
                        c = 0;
                        for (l=0; l<DEPTH; l=l+1)
                                begin
                                c = c + rf[l][k];
                                end
                        for (i=DEPTH; i<DEPTH+DIVCOLBITS; i=i+1)
                                begin
                                d = 0;
                                for (j=0; j<WIDTH; j=j+1)
                                        begin
                                        d = d + rf[i][j];
                                        end
                                rf[i][k] = ((c%2) ^ (d%2));
                                end
                        end
                end
        dataOut = rf[source];
        end
        end
endmodule
```

▪ Verilog code for a 32x16 register containing single parity bits per row and column of the register with additional parity checks

```
module rf_pb_row_by_column_addpar(clock, reset, writeEnable, dest, source, dataIn, dataOut, err);

parameter WIDTH = 16;
parameter DEPTH = 32;
parameter ADDRESSWIDTH = 5;

integer i,j,k,l,m,n,b,a,c,d,p,o,q,r,s,fix;

input clock, reset, writeEnable;
input [ADDRESSWIDTH-1 : 0] dest;
input [ADDRESSWIDTH-1 : 0] source;
input [WIDTH-1 : 0] dataIn;

output [WIDTH-1 : 0] dataOut;
output err;

reg [WIDTH-1 : 0] dataOut; // registered output
reg err; // registered error
reg [DEPTH-1 : 0] errd;
reg [WIDTH-1 : 0] errw;
reg [WIDTH : 0] rf [DEPTH : 0];

//reset
always@(posedge reset)
        begin
        for(i = 0; i<DEPTH+1; i=i+1)
                begin
                rf[i] = 0;
                end
        dataOut = 0;
        errd = 0;
        errw = 0;
        err = 0;
        end
```

```
// flip-flop for data-out, data-in and checking
always@(posedge clock)
        begin
        if (err) //fix?
                begin
                p=0;
                o = 0;
                for (i=0; i<DEPTH; i=i+1)
                        begin
                        o = o + rf[i][WIDTH];
                        end
                q = 0;
                for (j=0; j<WIDTH; j=j+1)
                        begin
                        q = q + rf[DEPTH][j];
                        end
                if (rf[DEPTH][WIDTH] != (o%2) ^ (q%2))//error in control bits?
                        begin
                        p=1;
                        end
                if (p)  //fix control bits
                        begin
                        for (k=0; k<DEPTH; k=k+1)
                                begin
                                r = 0;
                                for (j=0; j<WIDTH; j=j+1)
                                        begin
                                        r = r+rf[k][j];
                                        end
                                rf[k][WIDTH]=(r%2);
                                end
                        for (j=0; j<WIDTH; j=j+1)
                                begin
                                s = 0;
                                for (l=0; l<DEPTH; l=l+1)
                                        begin
                                        s = s + rf[l][j];
                                        end
                                rf[DEPTH][j] = (s%2);
                                end
                        end
```

```
    else    //fix data
        begin
        for (l=0; l<DEPTH; l=l+1)
            begin
            if (errd[l])
                begin
                for (k=0; k<WIDTH; k=k+1)
                    begin
                    if (errw[k])
                        begin
                        rf[l][k] = ~rf[l][k];
                        end
                    end
                end
            end
        end
    fix = 1;
    end
for (i=0; i<DEPTH; i=i+1) //error row?
    begin
    m=0;
    for (j=0; j<WIDTH; j=j+1)
        begin
        m = m+rf[i][j];
        end
    if ((m%2) != rf[i][WIDTH])
        begin
        err = 1;
        errd[i] = 1;
        fix = 0;
        end
    else
        begin
        errd[i] = 0;
        end
    end
for (k=0; k<WIDTH; k=k+1) //error column?
    begin
    a = 0;
    for (j=0; j<DEPTH; j=j+1)
        begin
        a = a + rf[j][k];
```

```
            end
        if (rf[DEPTH][k] != (a%2))
            begin
            err = 1;
            errw[k] = 1;
            fix = 0;
            end
        else
            begin
            errw[k] = 0;
            end
        end
    if ((!errw) && (!errd)) //no error?
        begin
        err = 0;
        end
/*  if ((err) && (!fix)) //not fixed error?
        begin
        for (j=0; j<DEPTH+1; j=j+1)
            begin
            rf[j] = 0;
            end
        dataOut = 0;
        end     */
    if ((!err) && (!reset)) //write & read
        begin
        if (writeEnable)
            begin
            n = 0;
            for (j=0; j<WIDTH; j=j+1)
                begin
                rf[dest][j] = dataIn[j];
                n = n+rf[dest][j];
                b = 0;
                for (l=0; l<DEPTH; l=l+1)
                    begin
                    b = b + rf[l][j];
                    end
                rf[DEPTH][j] = (b%2);
                end
            rf[dest][WIDTH]=(n%2);
            c = 0;
```

```
                        for (l=0; l<DEPTH; l=l+1)
                                begin
                                c = c + rf[l][WIDTH];
                                end
                        d = 0;
                        for (j=0; j<WIDTH; j=j+1)
                                begin
                                d = d + rf[DEPTH][j];
                                end
                        rf[i][k] = ((c%2) ^ (d%2));
                        end
                dataOut = rf[source];
                end
        end
endmodule
```