DALLAS SEMICONDUCTOR MAXIM

Microcontroller
ENGINEERING REVIEW

Volume 2

Dial-up networking with the DS80C400 microcontroller

The DS80C400 networked microcontroller provides a ready solution for sending information over the network to a central location for analysis and corrective action.

As technology advances, large network availability greatly simplifies the process of microcontrollers monitoring and controlling sensors/actuators. Information can now be sent over the network to a central location for analysis and corrective action. For such an application, the DS80C400 networked microcontroller provides a ready solution. Besides being loaded with extensive peripherals, the DS80C400 silicon software implements a TCP/IP stack[1]. The Tiny InterNet Interfaces (TINI®) platform[2], which includes a Java™ Virtual Machine (JVM), provides extensive support of IP networking. Although the DS80C400 includes an Ethernet interface, the TINI Runtime Environment (TRE) also supports dial-up networking using the point-to-point protocol (PPP). A compelling aspect of using PPP is that both endpoints of the connection can communicate over modems to leverage public communication networks and IP software infrastructure. This allows the deployment of remote embedded networking applications in outposts where an Ethernet network is not available, but the ubiquitous phone switch network is (**Figure 1**).

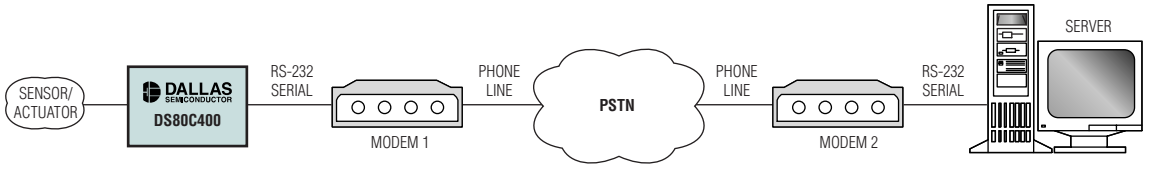

Figure 1. *A remote DS80C400 running the TINI Runtime Environment dials up a server to forward data.*

PPP overview

PPP is a general-purpose protocol that supports data transfer over many physical media, including (but not limited to) serial, parallel, Ethernet, and cellular phones, such as general packet-radio service (GPRS) devices. PPP is widely used in dial-up networking application because it requires little configuration and is easy to set up. The only requirement for the physical media is full-duplex capability. The communication can be either synchronous or asynchronous.

PPP is composed of three main components:

1) A method for encapsulating multiprotocol datagram over the same link. PPP encapsulation is based on the high-level data-link control (HDLC) format. Some encapsulation fields can be compressed if available bandwidth is limited.

2) A link control protocol (LCP) that establishes a connection, configures link options, detects errors, and terminates the link.

3) A family of network control protocols (NCP) that establish and configure corresponding network-layer protocols.

Table of Contents

PPP supports data transfer over many physical media, including serial, parallel, Ethernet, and cellular phones, such as general packet-radio service (GPRS) devices.

PPP operation

The Internet standard RFC 1661 describes PPP operation as a state machine going through different stages as the point-to-point link is configured, maintained, and terminated. **Figure 2** describes a simplified state diagram where PPP is divided into five distinct phases: dead, establish, authenticate, network, and terminate. PPP implements all phases except authenticate.

Link Dead Phase: Initial and ending phase of a link operation. The physical layer is not ready for packet transfer yet. When the physical layer is ready, an `UP` event is generated, and PPP proceeds to the link establishment phase.

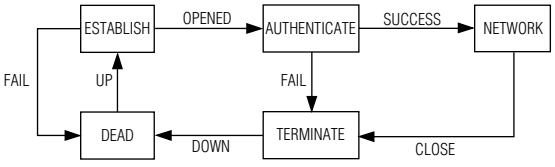

Figure 2. This simplified phase diagram illustrates PPP implementation described in RFC 1661.

Link Establishment Phase: The physical layer is up, and the link is negotiating transport options with its peer by exchanging LCP configuration packets. Only options independent of the network-layer protocol are configured at this stage. Once a configure-ACK packet has been sent and received, the PPP generates an `OPENED` event and proceeds to the next stage.

Authentication Phase: An optional phase that authenticates to the peer. On some links, such as dial-up networking, it is desirable for the link to authenticate before network-layer protocol packets can be exchanged. For such an implementation, a request for authentication must be sent during the link establishment phase.

Network-Layer Protocol Phase: After the link has been established and authentication has succeeded, the network-layer protocol is configured by exchanging NCP packets specific to the network layer supported. Each network-layer protocol has a unique NCP and must be negotiated separately.

Link Termination Phase: A `CLOSE` event is generated when the PPP link is terminated because of carrier loss, authentication failure, link-quality failure, or the administrative closing of the link. LCP terminate packets are exchanged between peers. The network-layer protocol is informed of the closing link and takes appropriate action. The physical layer is disabled after receiving a terminate-ACK, or timeout. A `DOWN` event is then generated, and the PPP returns to the link dead phase.

TINI PPP

TINI uses RFC 1661 as a framework for PPP implementation. PPP serves strictly as a transport mechanism for IP datagrams over a serial link. In the native network stack, PPP exists below the IP module and above the serial port drivers. To alleviate programming complexity, the PPP stages are simplified further (**Figure 3**).

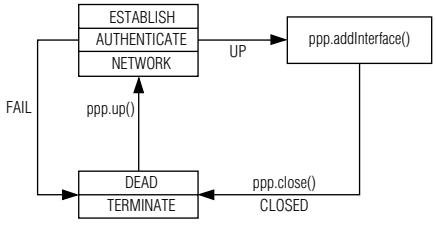

Figure 3. This TINI PPP phase diagram illustrates how PPP is implemented on TINI.

PPP is explained to an application developer through Java classes in the `com.dalsemi.tininet.ppp` package. The PPP states are event driven. `ppp.up()` establishes the link, authenticates, and sets up the network protocol. Password-authentication protocol (PAP) and challenge-handshake-authentication protocol (CHAP) are supported.[3] Once the link is configured, an `UP` event is generated and adds the PPP interface to the network stack so network traffic can be directed to that interface. `ppp.close()` issues a `CLOSE` event, brings the link down, and returns to the dead/terminate state.

Example 1 shows fragments of a `PPPClient` implemented on the DS80C400. (Go to www.ibutton.com and search for PPP for the latest example.) After a PPP object is created, the `PPPClient` is installed as the PPP object's `PPPEventListener`. PPP parameters are set, and the link is initiated with a series of `atCommand`. Once the link is established, `ppp.up()` is called to notify the network stack that PPP is now available for network traffic. The link is terminated with `ppp.close()`.

Example 1. PPPClient implementation

The TINI Runtime Environment provides user-friendly APIs that conceal details so developers can concentrate on their designs, using PPP as a utility. Even without a traditional phone network, the same applications can still run by replacing the modem with a GPRS wireless phone.

```
public class PPPClient extends Thread
  implements PPPEventListener, CommPortOwnershipListener{
  …
  public void run(){
    ppp = new PPP();
    openSerialPort(portNumber);
    // Add this object as a PPP event listener
    ppp.addEventListener(this);
    // Set the local and remote IP address
    ppp.setLocalAddress(localAddress);
    ppp.setRemoteAddress(remoteAddress);
    // Set client peer type options
    // Set the ACCM to escape all octets
    ppp.setRemoteAccm(0x00000000);
    ppp.setLocalAccm(0x00000000);
    ppp.setAuthenticate(false, true);
    // Set username and password
    ppp.setUsername(username);
    ppp.setPassword(password);
    // Initialize modem
    for (int i = 0; i < dialSequence.length; ++i)
      atCommand(dialSequence[i]);
    // Set connected flag
    connected = true;
    // Issue up command to PPP FSM
    ppp.up(serialPort);
   // PPP connection is now established, we can now
   // communicate with remote host
    sendData();
    ppp.close();
    closeSerialPort();
  }
  …
}
```

Example 2 demonstrates how a `PPPEvent` can be handled. Once the link is ready, and the `UP` event has been received, PPP is added as one of the network interfaces so IP packets can be forwarded to this interface. Whenever a `CLOSE` event is received, the network stack removes the PPP interface, terminating any network activity through PPP.

Example 2. PPPEvent method

```
/**
 * PPP event listener interface
 */
public void pppEvent(PPPEvent ev){

  switch (ev.getEventType()){
    case PPPEvent.UP:

      // PPP connection is up
      ppp.addInterface(interfaceName);
      interfaceActive = true;
      break;
    case PPPEvent.CLOSED:
      // PPP connection is closed
```

```
        if (interfaceActive){
          interfaceActive = false;
          ppp.removeInterface(interfaceName);
        }
        connected = false;
        break;
      default:
        break;
    }
}
```

Remote humidity data-logger example

In this article we write a powerful, networked application that takes full advantage of networking capabilities provided by a very economical, small form-factor computer. The example uses a TINI reference design called the TINIm400-030p, OEM Stamp+ Edition. This module provides a low-power, I/O-rich, low-part-count embedded controller and data collection module. When combined with the TRE, robust networked data-collection systems require minimal software effort. The Stamp+ module includes flash ROM, RAM, a real-time clock, 1-Wire® network, parallel I/O, and asynchronous serial ports. Intended for remote dial-up applications, this particular design does not use the Ethernet interface, although adding an Ethernet PHY and associated magnets would enable one.

This article presents a complete example[4] that captures and logs data, connects over the PSTN (public switched-telephone network) using PPP to manage dial-up connections, and makes the data available to a remote server. Dial-up networking support makes the data logger truly remote.

Application Note 702: Using TINI Point-To-Point Protocol (PPP) shows how TINI can be set up to use PPP, providing IP packet transport over the serial link. Go to www.maxim-ic.com/appnoteindex.

System overview

Figure 4 shows the setup for demonstrating dial-up networking capabilities using analog modems. If phone lines or their equivalent are not available, the hardwired serial-to-serial connection can also be used in a test setup.

The test configuration in Figure 4 includes the following equipment:

- A PPP module—running the `DataLogger` server
- A Windows® 2000 machine—running the `DataLoggerServer` software
- Two analog modems—one attached to the Win2K PC and the other attached to the serial port of the PPP module
- A humidity sensing circuit—collecting humidity data for logging

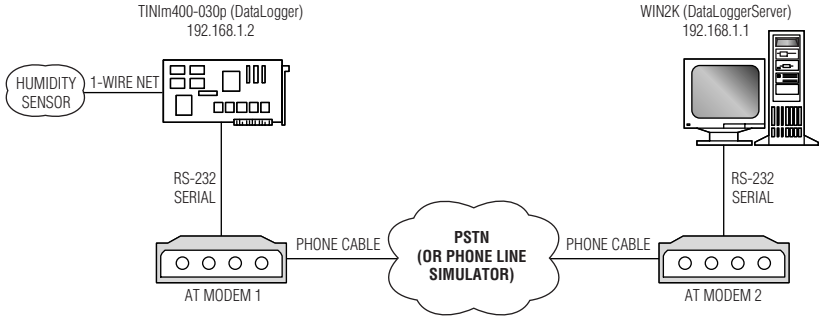

Figure 4. *A remote data-logging system uses generic modems to transfer data.*

The DS80C400 on the Stamp+ module has the TRE installed. The TRE platform supports ASM, C, and Java programming. The embedded firmware implements TCP/IP stack and provides framework for the PPP protocol. The term "TINI," which gives the Stamp+ module its remote data-logger capability, is used interchangeably with the term "Stamp+ module."

The PPP connection is made using two analog modems on either side of a phone line simulator. If two different phone lines are available, the public phone network can be used instead. To test

the PPP interface, a dial-up network connection should be created. Once the connection is initiated, the following sequence of events occurs:

1) TINI modem dials the server's modem.

2) The server's modem answers the incoming call.

3) PPP option negotiation begins.

4) Authentication information is transmitted from TINI to the remote server.

5) The server assigns an IP address to the TINI and notifies the TINI of the server's address.

Software and hardware overview

Complete source code can be downloaded from ftp://ftp.dalsemi.com/pub/tini/reference_designs/TINIm400-030p/DataLogger.zip. **Figure 5** shows humidity data is collected using a sensor circuit. The example sensor uses a DS1922H* 1-Wire temperature/humidity sensor packaged as an iButton®, although any 1-Wire device can be used with the DS80C400.

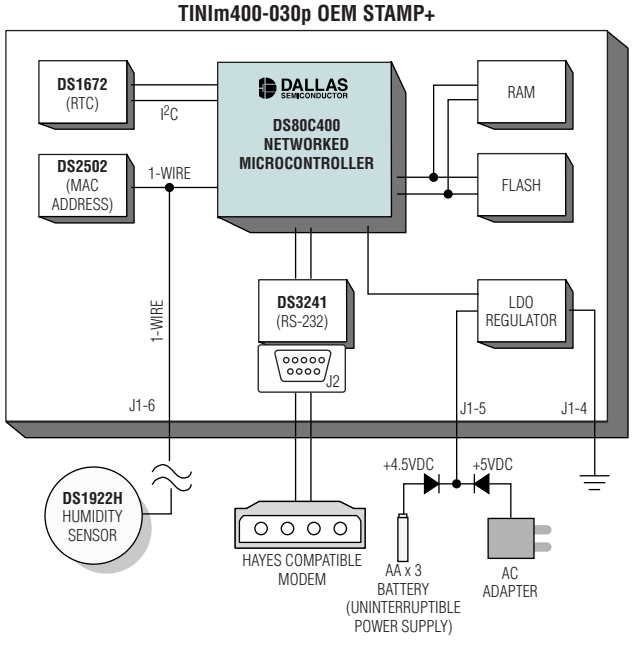

Figure 5. A TINIm400-030p data logger collects humidity data using a sensor.

TINI client software

The TINI `DataLogger` example demonstrates three concepts: 1-Wire networking, serial communications, and TCP/IP networking. Brief functionality descriptions of the most important classes are summarized below.

The `DataLogger` class

- Retrieves parameters from the configuration file `/etc/dataLogger.properties`
- Creates an instance of `HumidityLogger` to capture sample
- Creates an instance of `PPPDaemon` to manage PPP connection
- Initiates outbound connections to the remote server over the network interface such as Ethernet or PPP

The `HumiditySensor` class

- Handles communication with, and retrieves data from the DS1922H

The `HumidityLogger` class

- Initializes and manages humidity sensors
- Writes log data to the output stream to the server

The `PPPDaemon` class

- Dial-up client
- Establishes TCP/IP connections using a PPP interface
- Manages the physical data link
- Receives PPP event notification

For a detailed description of 1-Wire networks, refer to *Application Note 148: Guidelines for Reliable 1-Wire Networks* and *Technical Brief 1: 1-Wire Net Design Guide*, available at www.maxim-ic.com/appnoteindex.

*Future product—contact factory for availability.

• Notifies `DataLogger` of errors that occur in the physical data link

The `PPPSerialLink` class

• Implements the `PPPDataLink` interface

• Allows `PPPDaemon` to manage the data link

• Configures the serial port for the data link

The `PPPModemLink` class

• Subclass of `PPPSerialLink`

• Manages modem communications

• Monitors `SerialPortEvent.CD` (carrier detect) to detect if the modem hangs up

The `ModemCommand` class

• Handles serial communication with the modem

• Throws `DataLinkException` in case of a timeout while waiting for the desired response

Remote data-logging server

The `DataLoggerServer` is a simple GUI server application developed to accept connection from TINI and download its current log.

The DataLoggerServer class

• Displays log data

• Blocks on `accept` to wait for socket connection at PORT

• Builds logs and charts humidity and temperature changes over time

Running the example

Application files traditionally have been transferred to the TINI file system using the FTP protocol over an Ethernet interface. Because the Stamp+ module does not include an Ethernet interface, the ymodem file transfer protocol has been added to Slush and JavaKit. Ymodem allows files to be transferred to the TINI file system over the JavaKit serial link. For more detailed information about the Stamp+ module, refer to *Application Note 611: Dial-Up Networking with the TINIm400 Stamp*. Besides the application file, the `DataLogger.tini`, a `/etc/.startup` file containing the following text should be transferred to the TINI file system.

```
#
# Starting DataLogger application from .startup file
#
setenv FTPServer disable
setenv TelnetServer disable
setenv SerialServer disable
#
initializeNetwork
#
java /DataLogger.tini
```

This startup file disables the serial server and allows the data-logger application access to the serial port. Once the application and startup files have been transferred, resetting the TINI allows the new startup file to be processed and the data-logger application to be started.

The first thing TINI sends to `DataLoggerServer` is an integer value that tells the server the number of log entries to expect. After the server reads this value, it loops through all entries, reading each individual sample. The server displays each entry and logs this information to a file. The `DataLoggerServer` is written in Java and requires a Java Runtime Environment for execution, the same execution environment used to run Slush. See www.java.sun.com for installation instructions.

After running the `DataLogger` for several minutes to allow it to acquire a few samples, `DataLoggerServer` is run. In this example each sample was time-stamped one minute apart. If `DataLogger` runs more than one hour, it fills its sample vector, resulting in 60 (`MAX_SAMPLE`) data samples. If it runs for days, weeks, or even months, it still gets `MAX_SAMPLE` samples, but they always represent readings taken within the last hour.

Conclusion

Implementing a dial-up network connection on the DS80C400 is straightforward. The TINI Runtime Environment provides user-friendly APIs that conceal details so developers can concentrate on their designs, using PPP as a utility. Even without a traditional phone network, the same applications can still run by replacing the modem with a GPRS wireless phone.

The TINIm400 Stamp can be configured to initiate or receive dial-up connections. For data logging, a central server can dial into the TINIm400 Stamp and retrieve data at periodic intervals. In the event of a local fault, the Stamp module can initiate a PPP dial-up connection to the central server to notify the system of the error. By using the TINIm400 to detect local faults, the central server can be dedicated to analyzing the retrieved data.

As with any embedded module, the hardware and algorithms used depend on the specific application. The rich I/O capability of the DS80C400 and flexibility of the TINI Runtime Environment make adding remote sensors/actuators to networks quick.

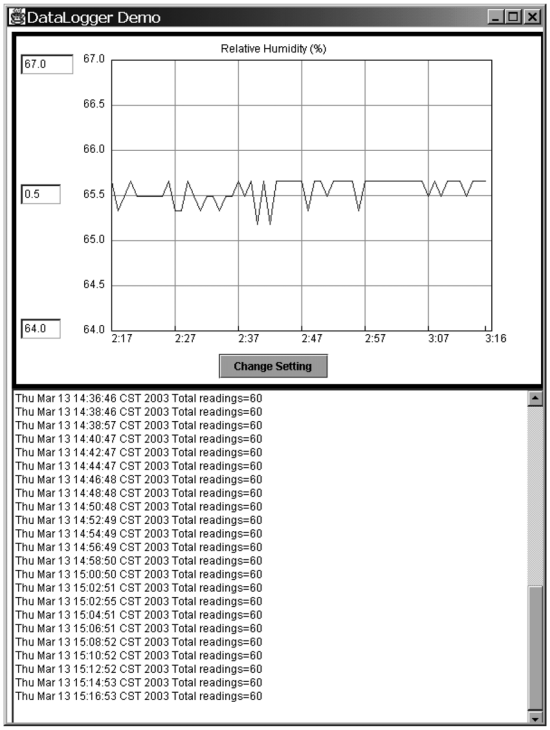

Figure 6. *The PC screen sample displays the DataLoggerServer operation.*

Footnotes:

[1]The silicon software supports IPv4/6 over Ethernet.
[2]*Application Note 708: Tiny InterNet Interfaces (TINI)*
[3]TRE Firmware Version 1.1 and later.
[4]This example is derived from Chapter 7 of *The TINI Specification and Developer's Guide*, available at www.maxim-ic.com/TINIguide.

For a comprehensive list of related application notes, go to www.maxim-ic.com/appnoteindex.

SRAM-based microcontroller optimizes security

Whether in an automated teller machine, passport/identity verification device, or a point-of-sale terminal at a convenience store, critical information such as passwords, personal identification numbers (PINs), encryption keys, and proprietary cryptographic algorithms must be protected against hackers. Elaborate policies and procedures are employed by financial services to protect both hardware and software. Consequently, designers of financial transaction systems face challenging trade-offs when developing equipment that processes billions of dollars every year.

The most powerful defense a secure microcontroller has is erasing memory contents quickly as a response to tampering.

To retain trust, a payment system must have end-to-end security. The server at the central bank is inside a restricted-access building with a fenced perimeter, but remote payment terminals in public places are easily susceptible to hacker invasion. Although it is possible to surround a microcontroller with a protective enclosure and wire an ancillary burglar alarm system, a determined assailant can still defeat the alarm system by turning the power off. Even though an enclosure can be opened, if the enclosure is coupled with the microcontroller's tamper-reactive cryptographic boundary, a safe cocoon for secure information is created. To be truly secure, the payment system architecture must have tamper-reactive technology built into the chip that employs the trusted computer. In this way the chip that does the computations defends its cryptographic boundary against intrusion by rapidly erasing the secret key, program, and data memory[1]. The most powerful defense a secure microcontroller has is erasing memory contents quickly when tampering is detected. The DS5250 secure high-speed microcontroller is an example that not only erases memory contents, but is also an inexpensive embedded system with SRAM for program and data storage.

Building trust with physical memory

Most embedded systems are developed using general-purpose computers chosen for flexibility and ease of debugging. But these benefits can become liabilities if they result in security breaches[2]. A hacker's first point of attack typically is the microcontroller's physical memory, so using optimum memory technologies for payment terminals is especially critical. Readily available logic analyzers, such as the Hewlett-Packard model HP16500B, can physically monitor the electrical signals of the address and data buses, which could reveal the contents of the memory and private data, such as secret keys. The two most important countermeasures to prevent this eavesdropping are to use strong cryptography on the memory bus and to choose memory technology for rapid erasure even in the absence of power. Some embedded systems attempt security by using microcontrollers with internal floating-gate memory, such as EPROM or flash memory. But the best memory technology erases its contents, not reveals it. While UV-erasable EPROM does not require electrical power for erasure, the awkwardness of supplying UV light for minutes increases its vulnerability. Flash or EEPROM memory requires that the processor remain operational and the supply voltage remain within the specified operating range to successfully accomplish erasure. These floating-gate-memory technologies are bad choices for secure applications because they hold their states indefinitely when power is removed, giving a hacker unlimited time to discover sensitive data. A better approach uses a memory technology like SRAM that reacts in one of the following ways if power is removed or the tamper detection circuitry is activated:

A hacker's first point of attack typically is the microcontroller's physical memory, so using optimum memory technologies for payment terminals is especially critical.

- The memory defaults to zeros when power is removed.
- The internal memory and encryption keys are erased in nanoseconds by the tamper detection circuitry.
- The external memory can be erased under application software control with write times of less than 100ns.

Some designers might be tempted to overcome the vulnerability of floating-gate memory by including the microcontroller on the same chip as the memory. This prevents unauthorized access to its memory contents. Some implementations use one or more internal lock bits, set as a final step at the

end of programming. When set, these bits prevent the microcontroller from revealing its contents if unsoldered from the PC board and placed in a device programmer, such as the widely used BP Microsystems Model BP-1700 Universal Engineering Programmer. In practice, the only way to erase the lock bits is by erasing all memory, which allows the device to be reprogrammed but destroys the program memory contents in the process. Further security attempts include adding an internal memory encryption array, which encrypts the output of the memory array when a device programmer attempts to verify or dump its contents. An example of this is found in the Intel MCS® 51 family of processors that use a 64-byte, user-programmed encryption array that XNORs the memory contents with the encryption array during verification. Unless the user knows the contents of the encryption array, any information extracted during a verification operation is meaningless. However, even the lock bit approach can be defeated. Techniques for hacking into floating-gate devices such as EPROMs, EEPROMs, and flash memories, and selectively erasing the security lock bits are easy to find in technical journals and Internet news groups[3]. Some device manufacturers suggest that one-time-programmable devices in a solid plastic package offer a degree of protection against lock-bit hacking. "Degree of protection" is a relative term, however. The application of hot acid can dissolve plastic encapsulation over the die without harming it. Then a careful study of the die layout using a tool such as a simple and inexpensive Karl Suss PM 8 Manual Probe Station can reveal the location of the security lock bits. This technique is often performed on UV-erasable EPROMs. After decapsulation, the die is painted with opaque paint or even electrical tape, and pinholes carefully made over the location of the lock bits. Exposing the device to a strong UV light then erases the security lock bits, yet leaves the main memory array unaffected. The device can then be read in a standard programmer as though the lock bits were never set (**Figures 1** and **2**). This simple procedure is routinely performed in semiconductor companies to analyze failures.

Floating-gate-memory technologies are bad choices for secure applications because they hold their states indefinitely when power is removed, giving a hacker unlimited time to discover sensitive data. A better approach uses a memory technology like SRAM.

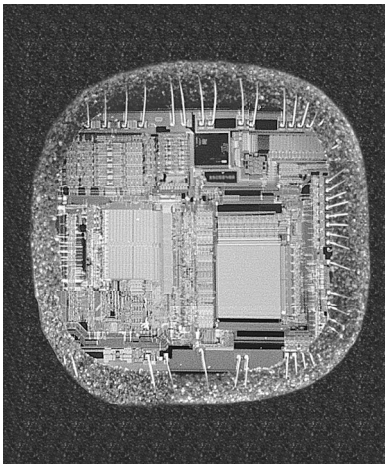

Figure 1. *Microcontroller die showing exposed EPROM after acid decapsulation.*

Another shortcoming of floating-gate memory technology is that the memory cells are intrinsically nonvolatile, maintaining their contents even if power is removed from the microcontroller. When power is removed from floating-gate devices, the data's decay time is rated in hundreds of years. This lapse in time is a problem when long-term protection of private keys is required for private-key-infrastructure (PKI)-based systems[4], as it gives a hacker unlimited time to breach physical defenses in the chip and access the memory before the device executes a tamper response.

SRAM and speed

All secure applications require fast read/write cycle times for the highest level of protection. SRAM is the fastest of all memory technologies. It can be instantaneously erased or "zeroed" as part of a tamper response. Additionally, SRAM is widely available, reasonable priced, and has unique features for secure data storage[4]. Although intrinsically volatile, it can easily be made nonvolatile for more than 10 years in the absence of V_{CC} using a lithium backup, which can also power a real-time clock for time stamping and dating transactions. These features are not possible with floating-gate-memory technologies.

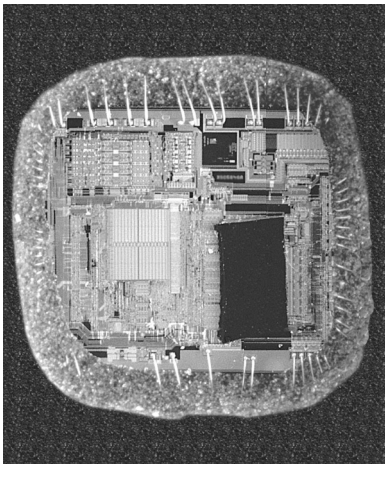

Figure 2. *The same microcontroller has its EPROM array covered, leaving security lock bits exposed for easy erasure.*

Authenticating the transaction

The PINpad module of today's payment terminals provides the core trust for financial payment systems. This module, regulated by banking authorities and credit card issuers, requires a secure microcontroller with resident software that includes device drivers for keypads, magnetic stripe card readers, smart card readers, and LCD displays. There must also be some method of high-speed serial communication to a general-purpose host (PC, 486, ARM) as well as PKI cryptography routines for secure end-to-end communication. The memory footprint of the PINpad module microcontroller can be hundreds of kBytes and exceed the economical size of a single chip, so external memories are needed. As previously mentioned, external memory is vulnerable to eavesdropping unless the communications between the microcontroller and external memory use strong cryptography. Such an encryption scheme has several requirements that build on each other:

MCS is a registered trademark of Intel Corporation.

For more information about EMV Integrated Circuit Card Specifications for Payment Systems, go to www.emvco.com.

- Encryption/decryption must occur at a rate comparable to instruction execution. Cryptographic operation must be performed on each program fetch or a small group of bytes if using block encryption such as data encryption standard (DES). The cryptographic algorithm must be strong, fast, and hardware-based. A superior solution is a triple DES (3DES) using dedicated on-chip 3DES hardware, which executes much faster than multiple passes through a single DES encryptor.
- External memory must be SRAM to support the high data transfer rates required by the cryptographic engine. Battery-backed SRAM is also required so that memory can be quickly erased when tampering is detected.
- Data crucial to the cryptographic operation such as encryption keys should never be seen outside the processor. The processor must generate and securely store at least some part of the encryption keys. These keys are erased as part of the tamper response, rendering the external memory unintelligible.

SRAM is the fastest of all memory technologies and can be instantaneously erased or "zeroed" as part of a tamper response.

- Initial loading and encryption of program and data must be done by a bootloader internal to the microprocessor. This prevents unauthorized viewing of the application code and hides the encryption method, making the bootloader a firewall. The bootloader must not only prevent access to information already loaded, but must also prevent a hostile agent from loading unauthorized rogue software. An example would be capturing a working PINpad or ATM, erasing its software, and loading new software designed to collect PIN numbers from unwitting users. Therefore, all communication between the bootloader and host system must be encrypted to prevent interception and decoding by hostile agents.

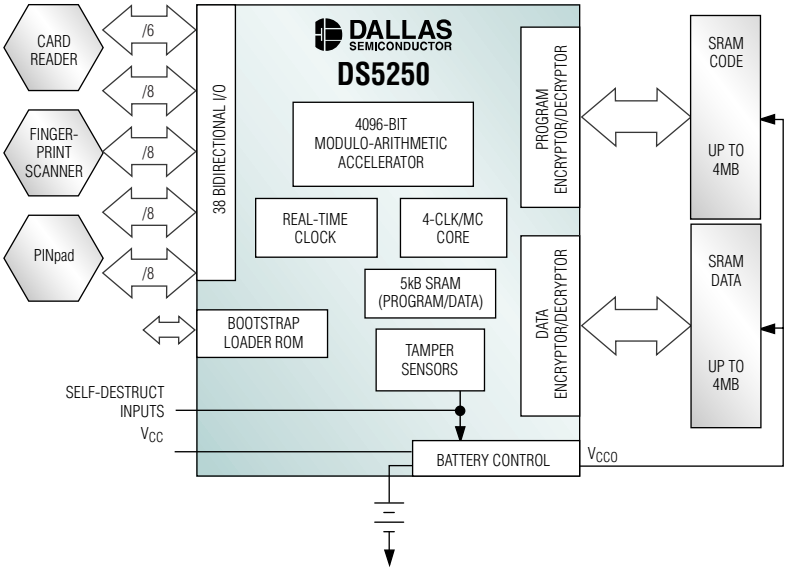

Figure 3. The DS5250 executes up to 6.25 MIPS and stores program and data memory in up to 8MB of external SRAM.

DS5250—putting it all together

It is possible to build an embedded system with SRAM for program and data storage by using encryption. The DS5250 secure microcontroller is an example of such a system (**Figure 3**). It executes up to 6.25 million 8051-based instructions per second, storing its program and data memory in up to 8MB of external SRAM. The most sensitive information can be stored in 5kB of internal data memory. Data retention of the SRAM is handled through dedicated battery switching hardware inside the microprocessor that supplies either V_{CC} or battery power to the external memory. Such a system can be attached to authentication peripherals such as ISO-7816-compliant smart-card readers, fingerprint scanners, and keypads.

Dedicated encryption/decryption engines on the program and data buses ensure the security of the external bus. The DS5250's program memory bus is 8-byte block-encrypted with either single or 3DES. The data memory bus is optionally encrypted in real-time with dedicated hardware. Key generation is aided by a true random-number generator that passes the statistical random-number generator tests as described in Section 4.11.1 of the Federal Information Processing Standards Publication 140-1 (FIPS PUB 140-1), *Security Requirements for Cryptographic Modules*. A program memory integrity-check feature further increases memory security by comparing the checksum of individual blocks against a previously calculated value. The failure of the block checksum to match the stored value evokes a user-programmable tamper response, preventing substitution attacks.

In addition to NV SRAM support, the DS5250 incorporates many system security features. A high-performance 4096-bit modulo-arithmetic accelerator (MAA) unit powers RSA calculations with a modulo exponentiation of 1024 bits in under 6ms. An additional 5kB of internal SRAM can be used for secret key storage, data memory, and/or program memory, and as scratchpad memory for the MAA. Application software is securely loaded through the serial port using a bootloader chal-

lenge/response protocol based on a chain-cipher, dual-key 3DES encryption algorithm. Alternately, the DS5250 allows the system designer to create application-specific bootloader software that takes advantage of all the microcontroller's security features.

Internal tamper sensors can detect physical attacks on the microprocessor die and initiate a tamper response, erasing the encryption keys used to decode external memory. User-defined sensors or switches can be connected to self-destruct input pins that have the same tamper response but also destroy any data stored in the internal code/data RAM memory. Additionally, the self-destruct input removes all power to the SRAMs, ensuring all program and data memory loss. A self-destruct interrupt source, wired to an external pin, allows the system software flexibility to create a custom tamper response based the needs of a particular implementation. The DS5250 secure microcontroller chip, however, has a self-contained cryptographic boundary that is tamper reactive, thus reducing system cost by eliminating the need for additional tamper response. **Figures 4A** and **4B** contrast a common security approach with the DS5250 approach.

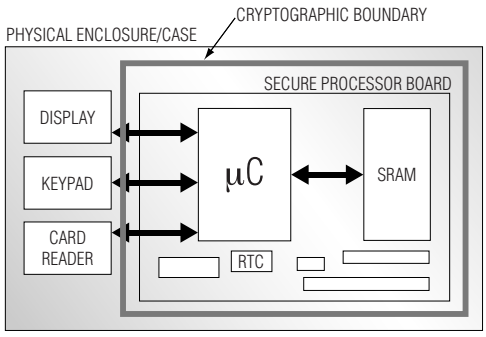

Figure 4A. *Embedded system designers are often tempted to build a secure computer using a general-purpose microcontroller with associated peripherals/memory, wrapping the PC board in multiple, expensive tamper sensors.*

Keeping it safe and secure

Every financial terminal, whether it is a PINpad, a POS terminal, or an ATM, processes confidential information that resides in its RAM and ROM. This makes the memory components critical in the security of the financial transaction. As hackers grow more sophisticated, so too must the security methods used to protect confidential information. Although there are many levels of protection, encrypted SRAM offers the best protection for embedded memory contents.

Perhaps most importantly, the DS5250 secure high-speed microcontroller protects sensitive information and maintains the trust of payment systems to meet financial industry regulations. When necessary, it will unconditionally erase private keys, programs, and data as a tamper response, keeping data safe and secure.

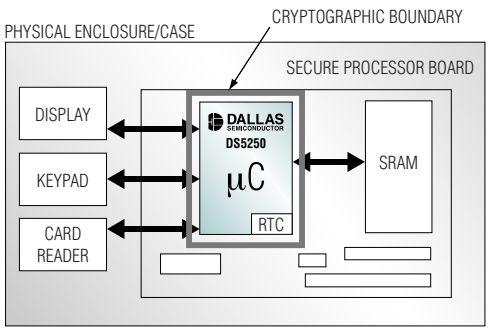

Figure 4B. *The DS5250 secure microcontroller has a self-contained cryptographic boundary that is tamper reactive, thus reducing system cost by eliminating the need for additional tamper response.*

References:

1. Smith, Sean; Palmer, Elaine; Weingart, Steve. *Using a High-Performance, Programmable Secure Coprocessor*. Proceedings of the Second International Conference on Financial Cryptography, Springer-Verlag Lecture Notes in Computer Science, 1998.

2. J. D. Tygar and B. S. Yee. *Dyad: A System for Using Physically Secure Coprocessors*. Proceedings of the joint Harvard-MIT Workshop on Technological Strategies for the Protection of Intellectual Property in the Network Multimedia Environment, April 1993.

3. A variety of sources discuss this subject, and can be found via a web search engine using the keywords "EPROM," "plastic," and "acid."

4. RSA Laboratories. *PKCS #1 v2.1: RSA Cryptography Standard*, Bedford, Massachusetts, 2002.

Creating networked multimedia applications with the DS80C400

...the DS80C400 can transmit four frames of raw black and white video (240 x 180) per second without any hardware-assisted image compression.

Impressive multimedia applications—including public address (PA) systems, networked doors, MP3 players, and security cameras—can be built using a low-cost, networked microcontroller. This article discusses how to use the DS80C400 networked microcontroller in example systems using audio and video.

Building a networked PA system

Picture a PA system (also called "overhead paging") where an operator announces messages such as "Attention all employees: The fire alarm is in test," or "Supervisor, please report to the chemical dock." This setup uses separate cabling and infrastructure, and is often built on technology that was available before the transistor. Imagine moving the system to the network. Not only can separate audio cabling be eliminated, but the system can also be made intelligent. For example, the paging system could interface to the building's access control system or a network server, which knows the most likely location of an employee. A computerized PA system can also automatically repeat a message, freeing an operator to take more calls. Additionally, the system could tap into the company-wide email system and allow for an email-to-voice service, or use a website where paging requests could be entered and then announced without further human intervention.

How do you build a networked PA system? First, start with at least one server that runs the web interface, email gateway, and has a microphone or something similar. We will call this server "master control." Next, you need a number of speaker modules. These modules are networked units with a digital-to-analog converter (DAC) to drive a speaker. The price for these speaker units must be kept low, and the units must be extremely easy to install in the field.

Figure 1 shows a network sketch with two buildings, seven speaker units, and one master control server. A router, not a bridge, connects the networks of the example buildings. (This has an important consequence for the software, described later in this article.) In our example, the DS80C400 networked microcontroller drives the speaker units. Even though a microcontroller does not have the processing power and memory resources of the latest PC systems, a PA system is neither bandwidth nor processing intensive. Uncompressed monaural audio sampled at 22.05kHz x 8 bits consumes less than 180kbps, and renders excellent voice quality. There is also no cost for hardware decompression.

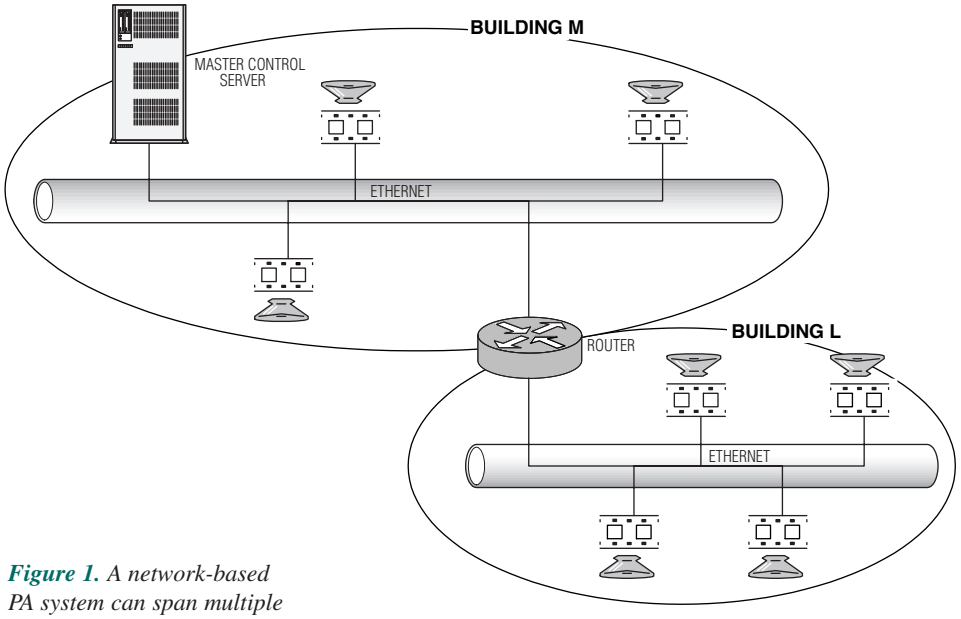

Figure 1. A network-based PA system can span multiple buildings.

Figure 2 illustrates the relative bandwidth requirements of a networked audio system. Even on an old (half duplex) 10Mb network with an effective 5Mbps bandwidth, the audio application uses less than 4% of that bandwidth. Most Ethernet networks today are at least 100Mbps.

Speaker hardware

Apart from the DS80C400, a speaker module requires some memory (512kB of SRAM is sufficient), a network PHY, DAC,

amplifier, and speaker. Refer to *Application Note 609: Internet Speaker with the DS80C400 Silicon Software* (available at www.maxim-ic.com/appnoteindex) for a complete working description.

The following technologies simplify installation and code distribution:

- DS80C400 NetBoot (see *Networked Application Update on the DS80C400*)
- DHCP to eliminate IP configuration, lowering installation and configuration cost
- Power-over-Ethernet (see *Power-over-Ethernet*) to simplify cabling and reduce material cost

Upon power-up, the DS80C400 ROM requests an IP address using DHCP, then queries the network for the latest version of the application. The application is then executed, and the system is ready to receive the audio data. A new speaker module can be installed in the field by locating an unused network port and connecting the cable.

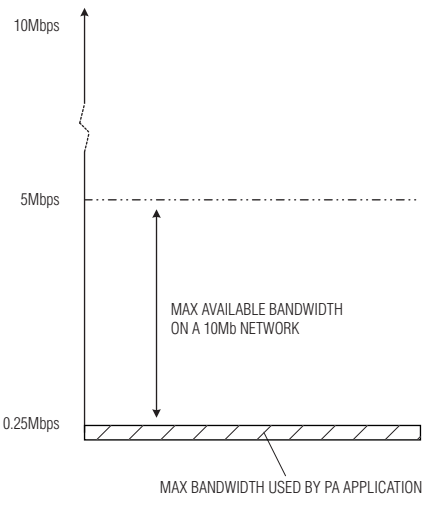

Figure 2. *The bandwidth requirements for networked audio are very low compared to the available bandwidth of Ethernet networks.*

Networked Application Update on the DS80C400

The DS80C400 is equipped with network boot capabilities, a convenient way to load code into a fresh part. "NetBoot" can be invoked using the `N` command in the serial loader. It supports SRAM and flash memories.

The use for network booting goes beyond manually loading new parts. On a design without nonvolatile memory, such as flash, it is the easiest way to install an application after a power loss. NetBoot can also automatically check the network for code updates and install them (proceeding with the previously loaded "old" code if the network is unavailable). NetBoot uses the DHCP and TFTP network protocols to acquire an IP address and load program data. The network configuration can also be statically set and stored in a 1-Wire device. Address support includes IPv6 addresses.

The following steps are required to set up automatic network code update for the TINI400 evaluation module/socket. (Refer to the *High-Speed Microcontroller User's Guide: DS80C400 Supplement* at www.maxim-ic.com/microcontrollers for additional details, including the types of supported 1-Wire devices.)

1) Make sure a DHCP server is available on the network, or allocate a static IP address and record it in the 1-Wire device.
2) Install a TFTP server (e.g., tftpd) and either publish its IP address using DHCP or record the address in the 1-Wire device.
3) Upload the code onto the TFTP server in *tbin2* format, under the file name `TINI400`.
4) Set the NetBoot jumper on the TINIs400.
5) Every time the system is reset, it will now automatically reload or update the code from the TFTP server.

Speaker software

The software does extra work for easy hardware installation. Because there is a router between buildings (Figure 1), broadcast messages cannot get from one building to the other. Therefore, simple broadcast messages cannot be used. A new speaker must send multicast messages (see *Multicasting*) until master control confirms the speaker's location and parameters. A new speaker system does not know the location of the master control beforehand, and consequently sends out multicast messages asking master control to identify itself. If security is an issue, this exchange

can be digitally signed to remove rogue systems claiming to be servers. Once configured using classic unicast messages, the speaker joins a multicast group and waits for audio packets. These audio packets are multicast from master control. The example software for the networked PA system was written in C (see *Developing for the DS80C400*). The following code can receive networked audio over a multicast socket.

```
#define MULTICAST_IP_MSB 239
#define MULTICAST_IP_2   192
#define MULTICAST_IP_3     0
#define MULTICAST_IP_LSB  22
#define MULTICAST_PORT  6789

int s;                       /* socket handle */
struct sockaddr address;  /* IP address */
unsigned char xdata buffer[1600];

/* Step 1: Create the socket */
s = socket(PF_INET, SOCK_DGRAM, 0);
/* Step 2: Join the multicast group */
memset(address, '\0', sizeof(address));
address.sin_addr[12] = MULTICAST_IP_MSB;
address.sin_addr[13] = MULTICAST_IP_2;
address.sin_addr[14] = MULTICAST_IP_3;
address.sin_addr[15] = MULTICAST_IP_LSB;
address.sin_port     = MULTICAST_PORT;
join(s, &address, sizeof(struct sockaddr));

/* Step 3: Listen for incoming packets */
memset(address, '\0', sizeof(address));
address.sin_port     = MULTICAST_PORT;
bind(socket_handle, &address, sizeof(struct sockaddr));

/* Step 4: Now we're ready to receive data */
recvfrom(s, buffer, 1600, 0, &address, sizeof(struct sockaddr));
/* We actually received data! We could play it. */
printf("Data received:\r\n");
...
```

A networked PA system could interface to a building's access control system or network server, know the location of an employee, automatically repeat a message, or use a website where paging requests are entered and announced without further human intervention.

With the exception of `join()`, these steps should be familiar to programmers who wrote code for TCP/IP networks. The function calls are very similar to Posix and Winsock. (Note: `xdata` is a Keil C keyword that tells the compiler where to allocate data.) On the DS80C400, `join()` and `leave()` initiate or terminate membership in a multicast group, respectively. All library calls—`bind()`, `recvfrom()`, etc.—in this and the following examples return status codes. Unlike the code in the abridged examples, it is preferable to check these return codes for errors and respond accordingly.

Power-over-Ethernet

Attaching a device to the network can have a downside—it adds an extra cable for the network. Fortunately, the power supply can be integrated into spare wires of the Ethernet cable. There are several solutions to the problem, most commonly the IEEE802.3af standard calling for a 48V supply on pins 7, 8 (+) and 4, 5 (GND) of the 8-pin Ethernet connector. A 48V supply is commonly used in telephony applications, and thus often readily available in cabling cabinets.

For use with microcontrollers, the supply has to be regulated down to suitable levels. Refer to www.maxim-ic.com/appnoteindex for application notes describing the use of the MAX5910 and MAX5014 to build an efficient IEEE802.3af-compliant circuit.

Text-to-speech

Using a microphone to capture live speech or playing canned audio stored on a network server is one way to use the PA system. Another use announces messages received in text form through email, from a web page, or by using the short message service on cell phones.

Retrofitting speech synthesis to the system is easy. The conversion can be done directly on the master control server, using a text-to-speech engine to generate waveform audio from the input text. The waveform can then be transmitted to the speakers like any other audio; changes to the speaker modules are not required.

Text-to-speech engines are widely available, and are an integral part of operating systems such as Mac OS X. A free Java speech engine can be found at freetts.sourceforget.net/. Commercial solutions sound more natural, so try the "Vocalizer" demo with American, British, and Australian accents on www.nuance.com.

Entertainment grade audio

CD audio poses a problem if the samples are transmitted in uncompressed form. Raw stereo samples at 44.1kHz x 16 bits would require 1.4Mbps of network bandwidth (or almost 30% of the 10Mb network), permanently exceeding the available bandwidth of many networks.

Compression algorithms such as MP3 can reduce the data rates, and therefore the network load tenfold and make the system feasible. Paired with a hardware decompression chip, the DS80C400 can easily manage the task. In fact, a clock rate of about 36MHz is fast enough to seamlessly play 192kb MP3s. The TINI MP3 project at www.mp3elf.net is a network MP3 design based on Dallas microcontrollers. Schematics and a full-blown multimedia application are also available.

Multicasting

The DS80C400 ROM and the TINI runtime support multicasting. Unlike unicast packets sent from one source to only one destination, multicasting allows several destination hosts to receive the same data, saving bandwidth by eliminating duplicate traffic. Multicast differs from simple broadcast. Using the IGMP protocol, a host can subscribe to one or more multicast groups, and multicast traffic gets routed only to those parts of the network with at least one recipient. Unlike broadcast messages that require the bridging of networks, routers forward multicast messages.

On the network, multicast packets use a special class of destination IP addresses, also called the multicast group. From an application perspective, adding multicast support is trivial. A call to `join()` adds the host to the multicast group, and packets are received like any other UDP traffic. Good manners dictate to `leave()` a group when reception is no longer desired. (A host sends periodic membership reports after joining at least one multicast group. If the host crashes without leaving the groups, the group memberships eventually expire.) When choosing a multicast group for your own applications, make sure to avoid duplicate group allocations by following the guidelines in RFC 2365.

Images

A video source can also be connected to the DS80C400. Such a system could be very useful as a security camera, where an inexpensive camera takes a snapshot every second and transmits it over the network for display and storage. Post-processing on the server side can perform motion detection and alert security personnel.

Good camera choices are those with modern cell phones—they are not only small, but also in expensive and widely available. Most use a serial protocol for communication, but details vary among manufacturers. Be sure you have all required technical information before committing to a particular camera make and model.

Network Camera System Example: An example network camera system, schematics, and code are available at ftp://ftp.dalsemi.com/pub/tini/ds80c400/reference_designs/netcam/.

Experimentation shows that the DS80C400 can transmit four frames of raw black and white video (240 x 180) per second without any hardware-assisted image compression and with some headroom that could be used for speech-quality audio.The following code sample opens a TCP connection to a network server to transmit a picture. (Even though we show the closing of the connection, the real application keeps the connection open to reduce overhead.)

```
int s;                          /* socket handle */
struct sockaddr address;  /* IP address */
unsigned char xdata buffer[1600];

/* Step 1: Create the socket */
s = socket(PF_INET, SOCK_STREAM, 0);

/* Step 2:  Fill in the target address 192.168.0.11 */
memset(address, '\0', sizeof(address));
address.sin_addr[12] = 192;
address.sin_addr[13] = 168;
address.sin_addr[14] =   0;
address.sin_addr[15] =  11;
address.sin_port = 8080; /* Target port */

/* Step 3: Connect to the server */
connect(s, &address, sizeof(address));

/* Step 4: Send the data in buffer */
send(s, buffer, sizeof(buffer), 0);

/* Step 5: Close the connection */
closesocket(s);
```

For those familiar with Unix network programming, `closesocket()` is `close()`. The DS80C400 version of the `close()` function is used by the file system. Like Windows systems, socket handles are not interchangeable with file handles on the DS80C400, and a separate function for sockets must be used.

The camera system clocks the DS80C400 at 73.7MHz, close to the 75MHz limit. The 73.7MHz frequency is generated using a fundamental mode crystal with an 18.432MHz frequency, and the PLL is integrated into the DS80C400 to multiply the frequency by four. This design reduces the overall system cost, while still allowing operation near the high end of the microcontroller's maximum frequency. In addition, 18.432MHz x 4 is also a good baud-rate generator for asynchronous serial communication.

Networked door

It is easy to combine the concepts of a security camera with bidirectional audio, a button, and a buzzer. This system allows us to build a networked door (**Figure 3**). The applications are endless, especially when combined with access control and security logs.

For the DS80C400 the button and buzzer are just peripherals that can be hooked up to a plain I/O port. In Keil C, I/O ports can be easily defined using `sfr` and `sbit`:

```
/* Define port 1 */
sfr p1 = 0x90;
/* Define P1.7 (port 1 is bit addressable) */
sbit p1_7 = p1^7;

/* Toggle P1.7 */
p1_7 = !p1_7;
```

Using iButtons and the 1-Wire master interface built into the DS80C400 makes adding authentication to the network door easy. (This interface is somewhat more complex to program, so Dallas Semiconductor provides libraries to simplify this task.) Refer to www.ibutton.com/TINI/applications/lock/ for an example showing how to connect a motorized door strike plate.

A final note: a system such as the networked door probably uses multiple processes (or tasks). The DS80C400 ROM contains a task scheduler. The following examples show how it can be used from C. Again, return codes should be checked in industrial-grade applications.

```
unsigned char pri, task;

/* Get the current task */
task = task_getcurrent();

/* The current task's priority */
pri = task_getpriority(0);

/* Decrease the priority */
task_setpriority(0, pri-1);

/* Sleep */
task_sleep(0, 0, 500);
```

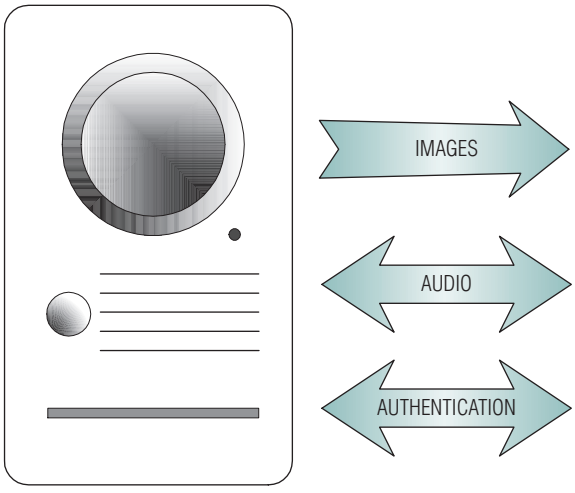

Figure 3. *A networked door interface combines video, audio, and access control.*

The programming model also contains useful functions such as `task_fork()`, which creates a new task by duplicating the current task. `task_kill()` destroys a task and `task_suspend()` puts a task on hold. These and other features are described in the *High-Speed Microcontroller User's Guide: DS80C400 Supplement* on the website.

Conclusion

A small and inexpensive networked microcontroller can be the powerful heart of interesting and useful multimedia applications. We encourage readers to use the DS80C400 to reproduce or refine the ideas presented in this article. For free samples, visit www.maxim-ic.com.

Developing for the DS80C400

Software applications for the DS80C400 can be developed in a variety of ways. For rapid prototyping, consider using Java and the TINI runtime environment. For applications where every cycle counts, hand-optimized assembly language is best.

For this article we use C. The Keil C compiler (www.keil.com) supports the 24-bit contiguous mode of the DS80C400 (refer to *Application Note 606: Configuring Keil PK51 Tools to Support 24-Bit Contiguous Addressing Mode*), allowing up to 16MB of code/data space. To use this mode, the eXtended versions of compiler and linker (CX51, LX51) are required. These tools are part of the Professional Developer's Kit (PK51).

Dallas Semiconductor provides C libraries that interface to the built-in DS80C400 network stack. The libraries and an illustrated step-by-step guide detailing how to create projects for the DS80C400 using the Keil development environment can be found on the Dallas Semiconductor ftp site, `ftp://ftp.dalsemi.com/pub/tini/ds80c400/`. These libraries greatly simplify network programming. For example, the creation of a TCP connection is reduced to the universally known sequence of `socket()` and `connect()`. For technical support, join the TINI mailing list at lists.dalsemi.com.

Using the DS5240/DS5250 as drop-in upgrades for the DS5002

Few or no changes are required to port DS5002FP software to the DS5240/DS5250.

The DS5240 and DS5250 high-speed secure microcontrollers in the 80-pin quad flatpack (QFP) package are pin-compatible, high-performance upgrades for the DS5002FP. Because the DS5240/DS5250 support the same feature set as the DS5002, they can be used as drop-in replacements for the DS5002FP in existing designs. Software written for the DS5002FP ports to the DS5240/DS5250 with few or no changes required, easily improving system performance and security while enabling the features available in the DS5240/DS5250 (**Table 1**).

Performance

The DS5240/DS5250 have higher maximum clock frequencies and require fewer clocks-per-machine cycle than the DS5002FP. Their streamlined cores execute single-byte instructions in only four clock cycles instead of the DS5002FP's 12 clock cycles.

A 1kB instruction cache reduces the effect of program memory encryption on execution speed, so that even with 3DES encryption active, the DS5240/DS5250 show an average 2.5X performance improvement over a DS5002FP operating at the same clock frequency.

Table 1. DS5002FP, DS5240, and DS5250 features

DS5002FP	DS5240	DS5250	FEATURES
PERFORMANCE			
16MHz	25MHz	33MHz	Maximum clock frequency
12	4		Clocks-per-machine cycle
25.2	8.4*		Average clocks per instruction
0.63	3.0*	3.9*	Average MIPS
SECURITY			
80-bit proprietary algorithm (single byte)	Single DES or 3DES (8-byte block)		Encryption of data in program memory (decrypted in parallel with program execution)
No	Yes		Separate encryption for program and MOVX memory
None	4096-bit MAA engine (1024-bit public key modular exponentiation in under 650ms)		Public key cryptography support
SDI pin	SDI pin; No battery/battery attach; Low temperature (<60°C)**		Destructive reset (DRS) triggers
No	Yes (optional)		Timed access on port write
48 bytes	1024 bytes		Vector RAM
No	Yes		Unique laser ID
No	No	Yes	Secure loader
FLEXIBILITY			
1x, idle, stop	1x, idle, stop; 2x/4x (crystal multiplier); Divide by 1024 (PMM); Internal ring oscillator		Oscillator clock modes
1	2		Serial ports
2	3		Timers
2	6		External interrupts
No	Hardware only	Hardware and loader ROM	Flash memory support
No	Yes**		Real-time clock

*Zero stretch cycles, 16-bit standard addressing mode.
**100-pin QFP only.

Security

DS5240/DS5250 external program memory is automatically secured using either single DES or 3DES encryption; data memory can optionally be encrypted. As with the DS5002FP, encryption keys are generated and loaded automatically from the on-board random-number generator. The DS5240/DS5250 improve security by using separate keys for program and data memory encryption, and by using an 8-byte block encryption for program memory instead of encrypting it byte-by-byte.

The DS5240/DS5250 provide 4kB of internal SRAM that can be used as program memory, data memory, or both; 1kB of SRAM can optionally be used as an extended stack.

In addition to DES and 3DES, the DS5240/DS5250 include a 4096-bit MAA engine to support public key encryption algorithms such as RSA. The expanded on-chip vector RAM (1024 bytes vs. 48 bytes for the DS5002FP) provides more space to store critical data and code, such as interrupt and reset handlers.

To protect the internal and external memory, the DS5240/DS5250 trigger a destructive reset (DRS) based on several conditions. As on the DS5002FP, a signal on the SDI pin clears the memory encryption keys, vector RAM, and any external battery-backed memory. Additionally, the DS5240/DS5250 trigger a DRS in response to a missing or newly attached battery.

The DS5240/DS5250 programmable evasion features protect against trial-and-error attacks that involve an abnormal number (user-selected) of resets within a short time. When enabled, this security mode responds to a repeated reset attack with increasing delays according to a user-selected time. Depending on the number of resets encountered, the delay time before execution resumes can increase from seconds to hours to weeks. At the highest security level, the DS5240/DS5250 can respond to a brute-force attack by locking execution permanently inside the ROM, rendering the part unusable until it is completely erased.

Another countermeasure against attacks is optional timed-access restriction on all output port writes. This increases the instruction bytes an attacker would have to encrypt correctly to write a recognizable pattern to an I/O port.

Flexibility

The DS5240/DS5250 provide an expanded set of on-chip resources over the DS5002FP, which increases flexibility for application and system design. In addition to the increased 1kB of vector RAM, the DS5240/DS5250 provide 4kB of internal SRAM that can be used as program memory, data memory, or both. Of this internal SRAM, 1kB can optionally be used as an extended stack.

For clock control and power management, the DS5240/DS5250 provide many features beyond the standard stop and idle modes supported by the DS5002FP. The clock multiplier allows the external crystal frequency to be doubled or quadrupled internally, expanding the range of crystals used to generate a given clock rate and allowing external EMI to be reduced. Power-management mode (PMM) divides the external crystal frequency by 1024 for reduced power consumption during periods when full-speed operation is not required. If rapid response to a serial input or interrupt is required during PMM, the switchback feature can be used to automatically revert to full-speed operation in response to these inputs. Instead of an external crystal input, an internal ring oscillator, which operates at approximately 12MHz, allows rapid exit from stop mode without waiting for the crystal oscillator to warm up.

Dual data pointers on the DS5240 improve performance of block copying operations in external memory. The INC DPTR instruction can be set to either increment or decrement the active data pointer, and the bit that determines which data pointer is active can be set to toggle automatically following certain data transfer operations.

The DS5240/DS5250 programmable evasion features protect against trial-and-error attacks that involve an abnormal number (user-selected) of resets within a short time.

Requirements for upgrading a design to the DS5240/DS5250

The DS5240/DS5250 provide all but two DS5002FP features. These features are:

- The DS5240/DS5250 perform all access to external memory (or memory-mapped I/O) using the dedicated data bus (BA14–BA0 and BD7–BD0). Accessing memory using a multiplexed address/data bus on ports 0 and 2 is not supported.

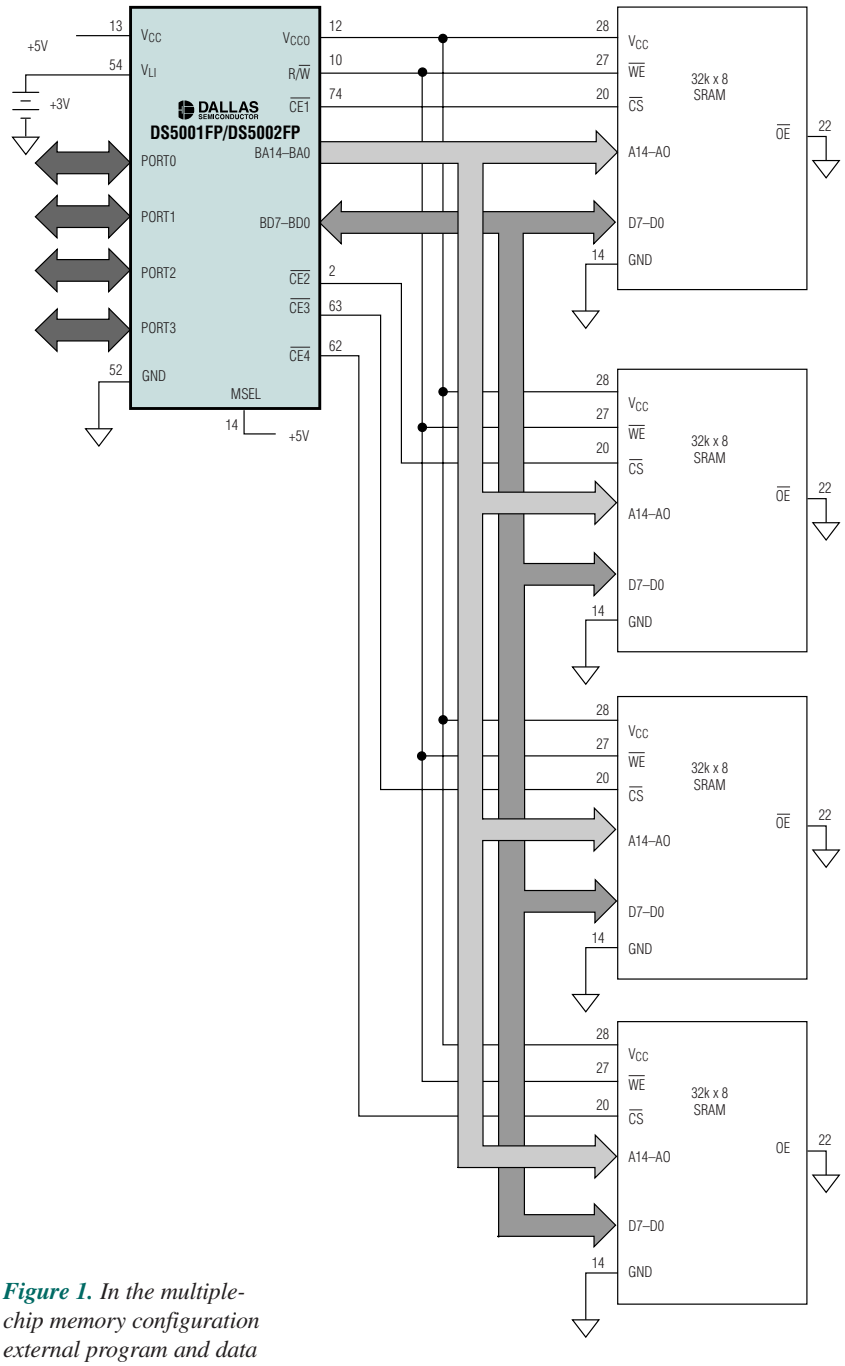

Figure 1. In the multiple-chip memory configuration external program and data memory are split across four 32kB x 8 SRAM devices.

- The DS5240/DS5250 do not support the reprogrammable peripheral controller (RPC) mode.

If the design relies on one of these features, it must be changed to use the DS5240/DS5250.

Additionally, the timing requirements for the external RAM(s) accessed through the dedicated data bus changed slightly from the DS5002FP to the DS5240/DS5250. An AC timing analysis will verify that the RAM and the DS5240/DS5250 are compatible, particularly if the DS5240/DS5250 run beyond the DS5002FP's maximum 16MHz.

Porting software from the DS5002FP to the DS5240/DS5250

The DS5002FP and the DS5240/DS5250 share the same instruction set, and most of the special function registers (SFRs) in the DS5002FP have the same location and function in the DS5240/DS5250. However, a few changes may be required when porting existing DS5002FP software to the DS5240/DS5250.

Memory configuration

The DS5002FP and the DS5240/DS5250 (80-pin QFP only) support two types of external memory configurations when accessing memory through the dedicated address bus.

Figure 1's multiple-chip configuration shows the MSEL pin connected to V_{CC} and four 32kB x 8 SRAM devices are connected to the dedicated bus. Two devices (enabled by $\overline{CE2}$ and $\overline{CE1}$) are mapped as 64kB of program memory, and the other two (enabled by $\overline{CE4}$ and $\overline{CE3}$) are mapped as 64kB of data memory. To select this configuration, use the following SFR settings:

- ACON.1 (AM1) and ACON.0 (AM0) should be set to 0 to select 16-bit addressing mode (for compatibility with existing software).
- MSIZE should be set to 00h to select the 32kB chip size. Note that this register can only be set in ROM-loader mode or user-loader mode.
- MCON.1 (PM) should be set to 1 to select nonpartitioned mode.
- Program memory is automatically encrypted; the PBCC.0 (TDESE) bit selects whether single DES (TDESE = 0) or 3DES (TDESE = 1) encryption is used for program memory. This bit can only be set in ROM-loader mode or user-loader mode.

- DMOS.1 (C3EE) and DMOS.2 (C4EE) should be set to 1 if data memory encryption is desired. Each of these bits controls encryption on one memory device.

In the single-chip configuration (**Figure 2**), the MSEL pin is connected to ground and a single 128kB x 8 SRAM device is connected to the dedicated bus. This device (enabled by $\overline{CE1}$) is used for program and data memory, and $\overline{CE2}$ and $\overline{CE3}$ are converted into two additional address lines. To select this configuration, the following SFR settings should be used:

- ACON.1 (AM1) and ACON.0 (AM0) should be set to 0 to select 16-bit addressing mode (for compatibility with existing software).
- MSIZE should be set to XX001001b to select the 128kB chip size. Note that this register can only be set in ROM-loader mode or user-loader mode.

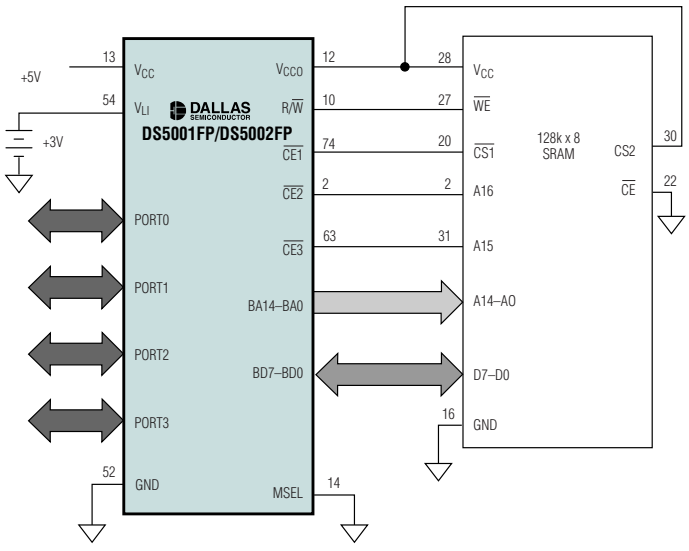

Figure 2. In the single-chip memory configuration, one 128kB x 8 SRAM device holds external program and data memory.

- Program memory is automatically encrypted; the PBCC.0 (TDESE) bit selects whether single DES (TDESE = 0) or 3DES (TDESE = 1) encryption is used for program memory. This bit can only be set in ROM-loader mode or user-loader mode.
- Data memory is automatically encrypted.

Handling interrupts

Five of the six interrupt sources supported by the DS5002FP are supported identically on the DS5240/DS5250. The power-fail interrupt is also supported on the DS5240/DS5250, but with the following differences:

- The power-fail interrupt vector is located at 33h instead of 2Bh; its priority level (0, the highest) remains unchanged.
- The enable bit for this interrupt is located at WDCON.5 (EPFI).
- The flag indicating that a power-fail event has occurred is located at WDCON.4 (PFI).

Determining reset sources

On the DS5002FP, the $\overline{POR}$ bit (PCON.6) is cleared when a power-on reset occurs, and this bit can be checked by software to determine the cause of a reset. On the DS5240/DS5250, the POR bit (WDCON.6) performs this function; however, this bit is set (not cleared) when a power-on reset occurs. Both bits require timed-access writes to be reset by software.

The DS5240/DS5250 streamlined cores execute single-byte instructions in only four clock cycles instead of the DS5002FP's 12 clock cycles.

Watchdog control

The watchdog functions the same on all parts. The following changes and additions apply, however, when controlling the watchdog on the DS5240/DS5250:

- The watchdog-enable bit is located at WDCON.1 (EWT).
- The watchdog-reset bit is located at WDCON.2 (RWT).
- The flag indicating that a watchdog timer reset has occurred is located at WDCON.2 (WTRF). Unlike the WTR bit on the DS5002FP, this bit must explicitly be written to 0 to be cleared.
- A watchdog interrupt (vector 63h) is available on the DS5240/DS5250. This interrupt, if enabled by the EWDI (EIE.4) bit, triggers before the watchdog reset occurs, allowing the watchdog

Power-management mode (PMM) divides the external crystal frequency by 1024 for reduced power consumption during periods when full-speed operation is not required.

timeout to be handled by software if desired. The WD1–WD0 (CKCON.7–6) bits on the DS5240/DS5250 control the time periods for the watchdog interrupt and reset, but the reset time defaults to roughly the same value as on the DS5002FP.

Random number generation

The random number register (RNR) functions the same in all three parts. However on the DS5240/DS5250, the bit indicating that a new random number is ready is located at RAMST.0 (RNRF). The time required to generate a new random number on the DS5240/DS5250 is approximately 30ms.

CRC operations

The DS5240/DS5250 support a new CRC-32 calculation function, and the process to calculate a CRC-16 value is different.

- The DS5240/DS5250 do not have CRC (C1h), CRCLow (C2h), and CRCHigh (C3h) registers.
- To perform a CRC-16 calculation, first select CRC-16 mode by clearing the CRCNT (RAMST.1) bit to 0. Then write the CRC data values to the CRC1 (B1h) register, allowing at least five machine cycles (or three NOPs between writes) to give the CRC registers time to settle. Once the last value is written, the CRC-16 value can be read from CRC2:CRC1.
- It is no longer necessary to write the CRC LSB byte back into the engine twice to clear the CRC registers. On the DS5240, writing any value to CRC2 automatically clears CRC1, CRC2, CRC3, and CRC4 to 0.

Table 2. DS5240/DS5250 new feature summary (80-pin QFP only)

FEATURE	SFRS
Timer 2	CKCON.5 (T2M)—Clock Mode Select P1.1 (T2EX), P1.0 (T2)—External Inputs T2CON—Flags and Mode Control T2MOD—Mode Control TH2, TL2—Timer Count MSB/LSB RCAP2H, RCAP2L—Timer Capture MSB/LSB
Data Pointers	DPH1, DPL1—Data Pointer 1 MSB/LSB DPS—Data Pointer Select
CRC-32	RAMST.1 (CRCNT)—CRC 16/32 Select CRC1, CRC2, CRC3, CRC4—I/O Registers
DES Engine	UDESC—DES Engine Control UDESD—DES Engine Data Input/Output
Modular Accelerator Engine	MAS0, MAS1—Operation Size Select MACT—Accelerator Control Register
User Loader Mode	ACON.6 (ULME)—User-Loader Mode Enable PEK1, PEK2, DEK, ROMST, PBCD, PBCC—Encryption/Loading Control
Laser ID	BP—Bootloader Password Register
Crystal Multiplier and Power Management	PMR—Power Management Register
Extended 1kB Stack	ESP—Extended Stack Pointer RAMST.5, RAMST.4—RAM1 Mode Select
Output Port Timed-Access Protection	RAR.3 (TAP)—Timed-Access Port Enable

Conclusion

Many new DS5240/DS5250 features can be used without any hardware changes. **Table 2** summarizes these new features and the SFRs that control them.

The DS5240/DS5250 feature sets increase security, performance, and flexibility over the DS5002FP. With only a few changes required to port software to the DS5240/DS5250, upgrading a DS5002FP design is easy.

Upgrading designs to the 100-pin QFP

The DS5240/DS5250 in the 100-pin QFP package provide all the features of the 80-pin package except for pin-for-pin compatibility with the DS5002FP and support for the DS5002FP modes of memory interfacing. They also provide the following features:

- Expanded memory-interfacing options, up to 8MB of program and data memory and 4MB of memory-mapped I/O.
- On-board RTC (driven by an external 32.768kHz crystal) useful for time stamping and self-imposed expiration dates.
- Temperature sensor that can trigger a DRS in response to a low-temperature attack.
- An additional SDI input pin that can be wired to an interrupt to allow a software-controlled tamper response.
- Hardware support for external flash memory (loader ROM support included on the DS5250).

For more information

An overview of the DS5240 and DS5250 high-speed secure microcontrollers is available online at www.maxim-ic.com. The confidential data sheets and user's guides require a non-disclosure agreement (NDA) prior to distribution. Contact Maxim/Dallas Semiconductor Customer Service for more information.