APPLICATION NOTE

AN707

Programmable peripherals using the PSD311 with the Philips XA

1996 Nov 04

Philips
Semiconductors

PHILIPS

Programmable peripherals using the PSD311 with the Philips XA

AN707

Author: Lane Hauck; with permission from WaferScale Incorporated.

Introduction

The Philips Semiconductors P51XA-G3 is the first of a new breed of fast, inexpensive 16-bit processors designed for high performance, high integration, and family growth. Although the P51XA (XA) family is promoted as a modern version of the venerable 8-bit 8051, it actually outperforms most of today's 16-bit embedded processors by a wide margin.

The XA is available in the usual array of OTP, ROMless and mask ROM versions so the cost/performance benefit that has made WSI PSD3XX chips attractive to embedded system designers applies the to XA. A typical system can be built using the ROMless version of the XA and a PSD311 for less cost than the OTP version of the XA.

Connection of a PSD3XX to the XA is not straightforward, due to the fact that the XA address and data lines are multiplexed in a manner unlike all other CPU chips that the PSD family is designed to support. This application note identifies the interface issues and solves them one by one to achieve an efficient XA–PSD interface.

The WSI PSD3XX devices can be used either with multiplexed address/data buses or with separate address and data buses. Multiplexed buses have the advantage that fewer PSD pins are required for the CPU interface, leaving more PSD pins available for general purpose system use. This application note addresses multiplexed bus connection of the XA and the PSD311.

The XA-PSD Marriage: Almost Perfect

The Philips XA designers took a radical departure from the 8051 bus architecture by bringing out the address lines A0–A2 on dedicated pins. These addresses are not multiplexed, which means that they do not require an ALE pulse to separate the address information from the data information. This allows up to 16 byte fetches on an 8-bit external bus with only one ALE pulse – the address is latched, the first byte is read or written, and then A0–A3 are incremented and the subsequent bytes are accessed.

This non-multiplexing of A0–A3 also allows very quick access of 16-bit operands on an 8-bit bus, because the time required to fetch the second byte can be as low as 20% of the normal ALE-R/W cycle time. This innovative timing allows external 8-bit bus systems to run nearly as fast as external 16-bit bus systems.

The XA gives very precise control (via internal programmable registers) of its bus timing. You can set the width of the ALE signal, and the positions and widths of the RD and WR signals. Given the inherent speed of the XA and the capability to fine-tune its bus timing, a word fetch using an 8-bit external bus can be significantly faster than other 16-bit CPUs that use a 16-bit external bus.

But...

For all the reasons it makes sense to buy the "ROMless" version of a CPU like the 8031 and attach a PSD chip for a lower system cost, it likewise makes sense to use a PSD chip with the ROMless XA. But there's a hitch. PSD3XX chips expect to see the low 8 bits of address and data multiplexed together, i.e., AD7–AD0. But the XA uses a different multiplexing arrangement, as shown in Table 1.

Table 1. Address-Data Multiplexing Schemes

CONVENTIONAL		XA	
A15		A15	
A14		A14	
A13		A13	
A12		A12	
A11		A11	D7
A10		A10	D6
A9		A9	D5
A8		A8	D4
A7	D7	A7	D3
A6	D6	A6	D2
A5	D5	A5	D1
A4	D4	A4	D0
A3	D3	A3	
A2	D2	A2	
A1	D1	A1	
A0	D0	A0	

As illustrated in Table 1, data lines D0 – D7 are multiplexed with A4 – A11 on the XA, not with A0–A7 as the PSD devices expect.

Basic Strategy

Given Table 1, how should the XA buses be connected to a PSD? In principle, it is possible to scramble address and data lines, as long as the scrambling is accounted for in the system design. For example, if you scramble address lines connected to a RAM, the scramble occurs for both writes and reads, so the effect is transparent to the system. However, address scrambling is not transparent in a device like a ROM that stores data at predetermined locations. When a CPU sends out an address to fetch an interrupt vector or execute one step of a program, it expects the data to be at that absolute address, not somewhere else due to scrambled address lines.

The first interface consideration is that the XA data lines must be connected to the corresponding PSD311 data lines. This dictates that XA A4/D0–A11/D7 must be connected to PSD311 AD0–AD7 as shown in the highlighted portion of Table 2.

Table 2. XA–PSD311 Bus Connection

XA	PSD311
A15	A15
A14	A14
A13	A13
A12	A12
A11/D7	AD7
A10/D6	AD6
A09/D5	AD5
A08/D4	AD4
A07/D3	AD3
A06/D2	AD2
A05/D1	AD1
A04/D0	AD0
A03	A11
A02	A10
A01	A09
A00	A08

As shown in Table 2, the upper four address lines A15–A12 are connected straight across. Because the data lines D7–D0 must line up, the CPU address lines A11–A4 must be connected to the PSD A7–A0. Then the remaining CPU lines A3–A0 connect to PSD A11–A8.

This address scramble must be accommodated for in the system design. There are three areas to consider: the EPROM, the IO port control registers, and the RAM.

EPROM

XA code is usually supplied to a device programmer using a file format called “Intel HEX”, and files of this type generally have the extension “HEX”. A HEX file is supplied to the WSI PSDsoft software, which combines it with PSD configuration information and writes out a new hex file with an “OBJ” extension.

A standard HEX file associates data with absolute addresses. Because of the address line scrambling shown in Table 2, a standard XA HEX file will not work. For example, if the XA sends out the address 0x1234, the EPROM location accessed within the PSD311 will actually be 0x1423. To account for this, we need a program that reads the XA HEX file, stores the data in memory in address–scrambled order, and then writes a new HEX file with the data residing at the scrambled addresses.

Appendix A is the source code for a C program to accomplish this address translation. It was compiled on the Borland C++ compiler Version 3.1 using the LARGE memory model. The SCRAMBLE.CPP and SCRAMBLE.EXE files are available on the WSI BBS. The source code is included in case you have any trouble running the program — you can freely adapt it to suit your purposes or cater to the whims of your particular C compiler.

To use the utility, place your XA HEX file and the SCRAMBLE.EXE file in the same directory, and type “SCRAMBLE myfile.hex” where myfile.hex is the HEX file to be scrambled. The SCRAMBLE program writes out a new file in address–scrambled order with the same filename and the “HX2” extension — in this example, myfile.HX2.

I/O Port Control Registers

The PSD311 port control registers appear at byte offset 2–7 from a programmable base address. The base address is set by the equation you write for the CSIOP output in the Programmable Address Decoder (PAD). If this base address is positioned at a 4 Kilobyte boundary, only the address lines A15–A12 participate in the decoding. These addresses are not scrambled, so there is a direct mapping of the equation you write for CSIOP and the memory space which the block of I/O Port Control Registers inhabit.

The address lines that participate in selection of the IO control registers, CPU A2–A0, **are** scrambled: CPU A0 is PSD A8, CPU A1 is PSD A9, and CPU A2 is PSD A10 (Table 2). Therefore the register offsets are translated as shown in Table 3.

Table 3. I/O Port Register Mapping

CPU REGISTER OFFSET	ACTUAL PSD ADDRESS
2 (Port A Pin Register)	0x20
3 (Port B Pin Register)	0x30
4 (Port A Direction Register)	0x40
5 (Port B Direction Register)	0x50
6 (Port A Data Register)	0x60
7 (Port B Data Register)	0x70

Table 3 indicates that to access the Port A direction register, for example, the byte at (BASE+0x40) must be accessed. This might be accomplished with the following XA code fragment, which sets PA0 and PA1 to outputs, and PA2–PA7 to inputs:

```
Apins   equ     $20
Bpins   equ     $30
Adir    equ     $40
Bdir    equ     $50
PortA   equ     $60
PortB   equ     $70
BASE    equ     $C000

mov     r0, #BASE
mov.b   [r0+Adir], #00000011b       ; 1=out, 0=in
```

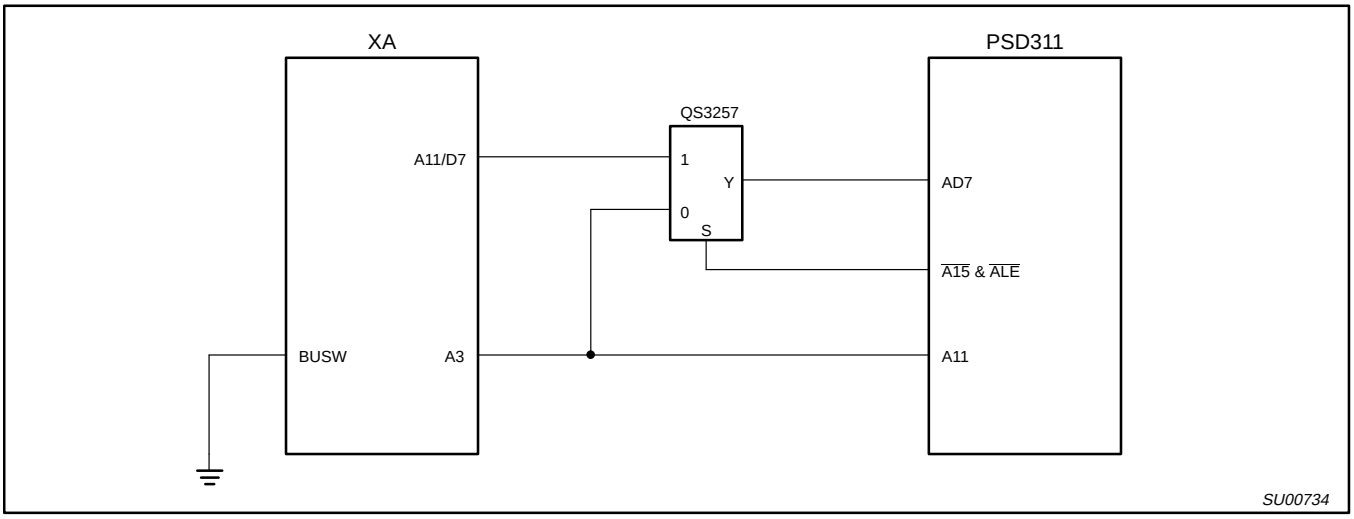

Figure 1. How to Convert A11D7 to A3D7

RAM

At first glance, it might appear that the PSD RAM is the easiest portion of the PSD to accommodate the scrambled address lines. After all, if the CPU writes to address XYZ, and unbeknownst to the CPU, it instead writes to address ABC, when the CPU tries to retrieve the data at XYZ it (again unknowingly) retrieves the data at ABC, which is the correct data. In other words, as long as the same scrambling occurs on a read-write device for both reads and writes, everything is copacetic.

A problem arises, however, because the connection shown in Table 2 connects CPU A3 to PSD A11, and CPU A11 (actually A11/D7) to PSD AD7. Why is this a problem? The RAM size in the PSD is 2 kilobytes, requiring eleven CPU address lines A0–A10. But look where CPU A03 is connected — it's to PSD A11, which is not used in the RAM addressing. Therefore, as far as the PSD311 RAM is concerned, it is missing CPU A03. Furthermore, the signal connected to the PSD311 A7 pin, CPU A11, is superfluous for RAM access.

The net result is that if the connections are made exactly as shown in Table 2, only half of the RAM would be addressable, and every eight bytes would repeat! This would tend to make the software people very unhappy, especially if they put data like the system stack in the PSD RAM.

The solution is to change the CPU "A11/D7" signal to "A3/D7". This change connects all eleven active CPU address lines A0–A10 to all eleven active PSD RAM address lines A0–A10, albeit in scrambled order (which is OK for a RAM). This is accomplished by the circuit shown in Figure 1. A15 is used as a RAM select signal to tell the circuit when to do the A11–A3 swap. The Address swap should be done for RAM accesses only, because A11 is required for EPROM addressing. In order to swap only the address and not the data portion of the multiplexed A11/D7 signal, the ALE signal is used as a qualifier.

The QS3257 is a quad bi-directional multiplexor made by Quality Semiconductor and others. In this circuit, A15 is used as the RAM chip select. When A15 goes HI to select the RAM, the MUX connects XA A3 to PSD311 AD7, but only for the ALE (address) portion of the cycle. When ALE de-asserts, the MUX re-connects XA A11/D7 to the PSD311 AD7 to connect the D7 signals together. The mux must be bi-directional to allow read-write access on D7. Note that the XA BUSW pin is tied low to support an 8-bit bus system at power-on.

How do we develop the logic for driving the MUX select (S) signal? Using the PSD311 PAD, of course. If the RAM is to be positioned within an 8K block, rather than the 32K block decoded by A15 alone, the other address lines A14–A12 may be used in the mux control equations. Appendix B is a PSDabel listing showing the mux select signal as 'mux', which uses PB0. Appendix C is the PSDsoft configuration file for the design.

Figure 2 is a scope photo of ALE, $\overline{\text{WRITE}}$, $\overline{\text{READ}}$ and address line A0. Figure 3 shows the timing for the MUX select signal. The measurements for Figures 2 and 3 were taken using a 30 MHz XA system, with the following bus timing parameters:

ALEW	1	[1.5 clock ALE pulse]
WM1	1	[long write pulse]
WM0	1	[1 clock data hold time for write]
DWA	3	[5 clock ALE–WR cycle]
DW	3	[4 clock WR cycle]
DRA	2	[4 clock ALE–RD cycle]
DR	3	[4 clock RD cycle]
CRA	2	[4 clock ALE–PSEN cycle]
CR	3	[4 clock PSEN cycle]

The XA listing in Appendix D gives the startup code that establishes the above bus timing plus other chip configuration data, and then runs a continuous loop to produce the waveforms shown in Figures 2 and 3.

Figure 2 illustrates two consecutive XA bus cycles. In the first cycle, the XA writes a 16-bit word by issuing two consecutive byte writes. Notice that address A0 changes from an odd address to an even address midway through the cycle (between write pulses) and a single ALE pulse is issued for both byte writes. The PSD3XX family devices work properly with the single ALE pulse because the addresses A8–A11, which are connected to XA addresses A0–A3, **are not latched in the PSD3XX.** PSD devices (PSD4XX/5XX) that latch all of the address lines would not work in this application, since they would not pick up the address change on A0 without a second ALE pulse.

Figure 3 shows the timing for the multiplexor select signal.

Because the first cycle writes data to memory outside the PSD311 RAM (A15=0), the mux select signal is high throughout the write cycle. The second ALE pulse corresponds to a read operation from RAM (A15=1). In this cycle the mux-S signal switches low, feeding A3 into the AD7 pin in place of A11. A3 is latched by the falling edge of ALE, the mux switches back to normal operation, and CPU D7 is connected to PSD AD7 for the remainder of the read operation.

The $\overline{RD}$ and A0 traces in Figure 2 illustrate the basic bus timing for the PSD311. The PSD311 access time can be determined by examining the read cycle which starts at the center division of the scope diagram. The XA reads the first byte by issuing the address of the first byte (A0=LO) and an ALE pulse. The RAM address is valid about 10 nanoseconds after the mux-S signal switches LO (to account for the 3257 mux switching time), and this address is latched inside the PSD311 by the falling edge of ALE. The XA reads the byte just before A0 switches from LO to HI, which starts the second RAM access cycle. (Remember that “A0” is actually A8 in the PSD311, which is not latched). The access time required for the first byte read (mux-S LO to A0 LO-HI transition) is about 100 nsec, and the access time required for the second byte read (A0 HI to $\overline{RD}$ going HI) is about 120 nsec. Thus a PSD311-90 is a good choice for this design.

Performance

As the bus timing waveforms of Figures 2 and 3 demonstrate, an 8-bit bus connection of the Philips Semiconductors XA CPU and the WSI PSD311 gives a very high performance system. Using fairly conservative timing, a word (double byte) read or write takes 400 nanoseconds using the PSD311-30.

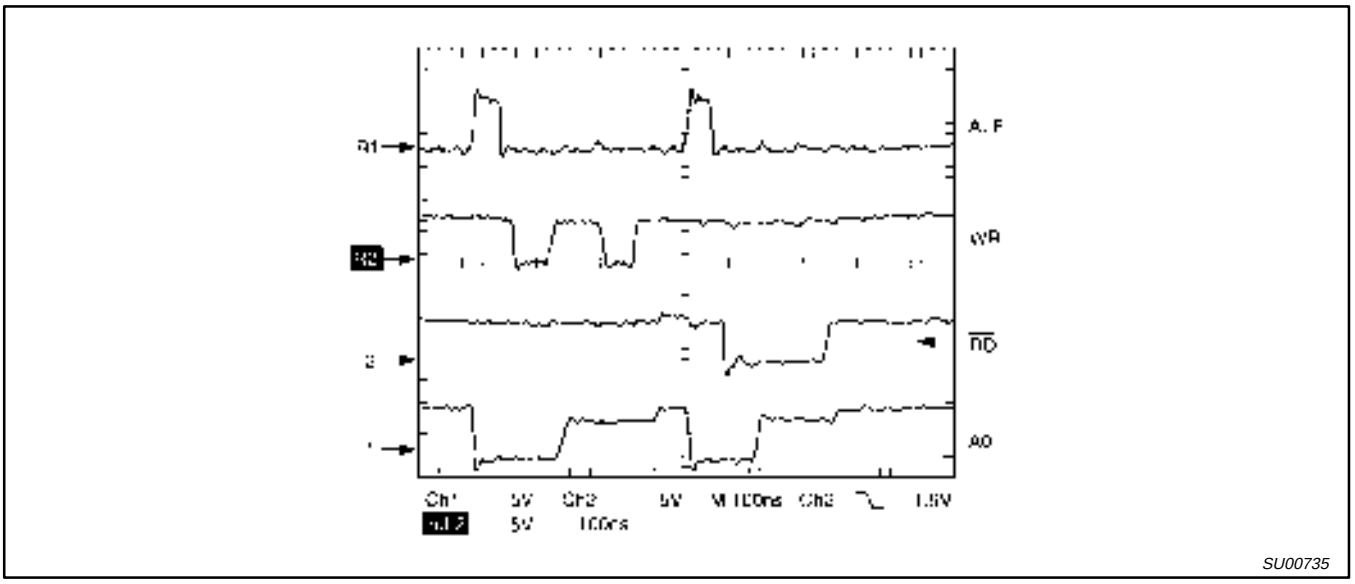

Figure 2. MUX Timing

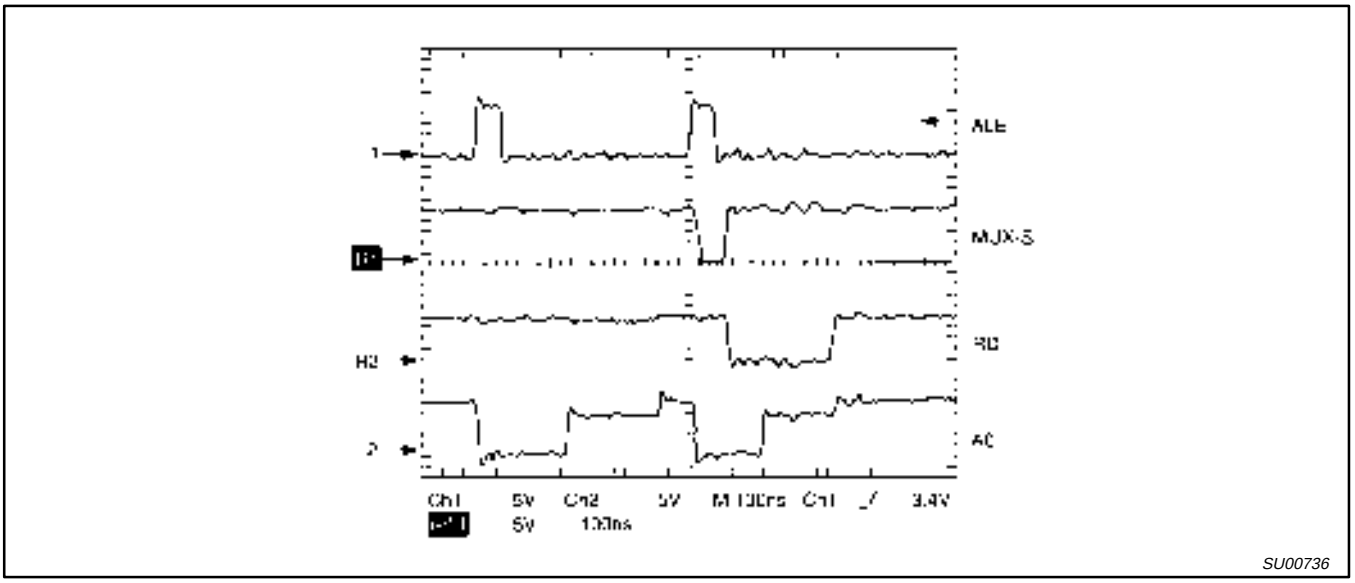

Figure 3. Multiplexor Select Signal (MUX-S)

Appendix A:
C Listing for SCRAMBLE Program

Scramble.cpp 9–12–95 Lane Hauck

This program is used to modify a standard Intel Hex file (.hex) so that it can be used to load a WaferScale PSD311 that is connected to a Philips Semiconductors XA microprocessor. Because the XA does not multiplex AD7–AD0, but instead multiplexes A11D7–A4D0, the addresses to the PSD311 must be scrambled for the data stored in the PSD311 ROM.

Typically the input hex file will be the output of a 51XA linker.

The program reads an Intel hex file, scrambles addresses, and writes a new Intel hex file with an “hx2” extension.

Invoke with: scram <infile.hex>.
Outputs file: “infile.hx2”.

Scramble order:

A15 A14 A13 A12 A11 A10 A09 A08 A07 A06 A05 A04 A03 A02 A01 A00
A15 A14 A13 A12 A07 A06 A05 A04 A03 A02 A01 A00 A11 A10 A09 A08

Hex ABCD becomes ADBC

Intel hex format:
(A) Data Record
: cc aaaa 00 [data] cs CR LF

: Colon
cc # data bytes (2 chars)
aaaa load addr (4 chars)
00 record type=data record
data 2 times cc chars
cs 2’s compl of checksum (binary values, not ASCII codes) includes cc,aaaa,00,data
CR carriage ret
LF line feed

(B) End Record
: 00 aaaa 01 cs CR LF

: Colon
00 no data bytes
aaaa program start address
01 indicates an END record
cs checksum of 00,aaaa,01

**********/

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <string.h>
#include <stdlib.h>
#include <dir.h>
#define ROMSIZE 32768L      //  PSD311 ROM size

// function prototypes
int         a2d(int  a);
int         scramble(int  inaddr) ;

// global variables
int huge   inarray[ROMSIZE];
FILE                *out;

int main(int argc, char *argv[])
{

FILE                *in;
unsigned int        pos,j,k,a,b,c,d,e,f,m,data;
char                outfilename[12];
char                *ptr;
int                 ch;
unsigned int        count,addr,scradd,csum;
char                string[16];
```

```
// check for two command line items: “scramble”, outfilename

if (argc ! = 2)
        {
        printf(“\nERROR: Usage: SCRAMBLE outfile\n”);
        sound(100);                     /* a little razz sound */
        delay(200);
        nosound();
        return 1;
        }

/* open the file given in the command line */

        if ((in=fopen(argv[1],”rt”)) == NULL)
        {
        printf(“Cannot open input file..%s\n”,argv[1]);
        return 1;
        }
printf(“File-%s-opened!\n”,argv[1]);

// open a file with input file name plus ‘.hx2’ extension

strcpy(outfilename,argv[1]);    // make a copy of filename
ptr=strchr(outfilename,’.’);    // ptr –> ‘.’
pos=ptr–outfilename;            // position of period
outfilename[++pos]=’h’;         // replace extension
outfilename[++pos]=’x’;
outfilename[++pos]=’2’;

if ((out=fopen(outfilename,”wt”)) == NULL)
        {
        printf(“Cannot open output file..%s\n”,outfilename);
        return 1;
        }
printf(“File-%s-opened!\n”,outfilename);

for (j=0; j<ROMSIZE; j++)
        {
        inarray[j]=0xFF;
        }

while (!feof(in))
        {
        ch=fgetc(in);
        if (ch==’:’)
                {
                csum=0;
                a=fgetc(in);
                b=fgetc(in);
                count=16*a2d(a)+a2d(b);
                if (count!=0)                           // ignore end record
                        {
                        csum+=count;
                        c=fgetc(in);
                        d=fgetc(in);
                        e=fgetc(in);
                        f=fgetc(in);
                        addr=4096*a2d(c)+256*a2d(d)+16*a2d(e)+a2d(f);
                        csum+=addr;
                        a=fgetc(in);            // should be two zero bytes
                        b=fgetc(in);
                        data=16*a2d(a)+a2d(b);
                        csum+=data;                     // (checks for 00 byte)
                        for (j=0; j<count; j++)
                                {
                                a=fgetc(in);            // data byte first digit
                                b=fgetc(in);            // data byte second digit
                                data=16*a2d(a)+a2d(b);
                                scradd=scramble(addr);
                                inarray[scradd]=data;
```

```
                                        csum+=data;         // NOTE: csum not checked
                                        addr++;             // here for debug/checkout only
                                        }
                                csum=255–(csum&0x00FF);   // 8–bit, 2's complement
                                }
                        }
                else;
                }
// Write the new hex file
addr=0;
for (j=0; j<=1023; j++)                 // 1024 lines of 32 bytes each
                {
                csum=0;
                fputs(":20",out);
                csum+=addr;
                sprintf(string,"%04X",addr);
                fputs(string,out);
                fputs("00",out);
                for (k=0; k<=15; k++)
                        {
                        for (m=0; m<=1; m++)
                                {
                                data=inarray[addr];
                                csum+=data;
                                sprintf(string,"%02X",data);
                                fputs(string,out);
                                addr++;
                                }
                        }
                csum=255–(csum&0x00FF);             // 8–bit, 2's complement
                sprintf(string,"%02X",csum);        // 2 chars in checksum
                fputs(string,out);
                fputs("\n",out);
                }
                fputs(":00000001FF\n",out);
sound(1000);                                        // a pleasant little sound...
delay(20);
sound(500);
delay(20);
nosound();
fclose(in);
fclose(out);
return 0;
}
// Scramble routine:    Change address ABCD to ADBC
int         scramble(int inaddr)
{
int         outaddr=0;
                outaddr =  inaddr           &       0xF000   // A
                        I   (inaddr <<8)    &       0x0F00   // D
                        I   (inaddr >>4)    &       0x00FF;  // BC
return(outaddr);
}
// ASCII to hex digit conversion
// converts ASCII char to integer 0–15
int a2d(int x)
{
if (x>=65 && x<=70)// A to F
                x–=55;     // –65 makes it 0–5, –55 makes it 10–15
else
                x–=48;     // "0" is ascii 48
return(x);
}
```

Appendix B:
PSDAbel File for MUX Control Signal

```
module xa311
title 'xa311';

mux                         pin 11;             " PB0
nA000                       pin 40;             " PC0–/CS8
ale,nRD,nWR                 pin 13,22,2;
a15,a14,a13,a12,a11         pin 39,38,37,36,35;
es0,es1,es2,es3             node 140,141,142,143;
es4,es5,es6,es7             node 144,145,146,147;
rs0,csiop           node 124,125;

equations

es0 =     !a15 & !a14 & !a13 & !a12 ;    " EPROM address map
es1 =     !a15 & !a14 & !a13 &  a12 ;
es2 =     !a15 & !a14 &  a13 & !a12 ;
es3 =     !a15 & !a14 &  a13 &  a12 ;
es4 =     !a15 &  a14 & !a13 & !a12 ;
es5 =     !a15 &  a14 & !a13 &  a12 ;
es6 =     !a15 &  a14 &  a13 & !a12 ;
es7 =     !a15 &  a14 &  a13 &  a12 ;
rs0 =      a15 & !a14 & !a13 & !a12 ;    " RAM select
csiop = !a15 & !a14 &  a13 & !a12 ;      " IOCTL select

mux    = !(a15 & ale);                          " a11–a3 mux control
!nA000 = a15 & !a14 & a13 & !a12;       " FPGA chip select

test_vectors

([a15,a14,a13,a12] –> [rs0,csiop,nA000])

 [ 0,  0,  0,  0 ] –> [ 0,   0,   1  ]; " nothing selected

 [ 0,  0,  1,  0 ] –> [ 0,   1,   1  ]; " IO  at 2000
 [ 1,  0,  0,  0 ] –> [ 1,   0,   1  ]; " RAM at 8000
 [ 1,  0,  1,  0 ] –> [ 0,   0,   0  ]; " FPGA at A000

test_vectors

([a15,ale] –> [mux])

 [ 0,  0 ] –> [ 1 ];
 [ 0,  1 ] –> [ 1 ];
 [ 1,  0 ] –> [ 1 ];
 [ 1,  1 ] –> [ 0 ];      " mux low only for address (ALE) time

end xa311
```

Appendix C:
PSDSoft Configuration File

****
WSI – PSDsoft Version 2.10
Output of PSD Configurations
****

PROJECT: xa311 DATE : 10/24/1995
DEVICE: PSD311 TIME : 18:31:39

****

==== **Bus Interface** ====

Data bus width	= 8–Bits
Address/Data Mode	= Multiplexed
ALE/AS signal	= Active High
Read/Write signals	= /WR,/RD,/PSEN
Memory space setting for EPROM	= Program space only (/PSEN)
Security bit	= OFF
Power–down capability	= OFF
EPROM low power mode	= OFF
Active–level of RESET signal	= LOW

==== **Other Configurations** ====

Port A : ADDRESS/IO Mode

Pin	IO/Address	CMOS/OD Output
PA0	IO	CMOS
PA1	IO	CMOS
PA2	IO	CMOS
PA3	IO	CMOS
PA4	IO	CMOS
PA5	IO	CMOS
PA6	IO	CMOS
PA7	IO	CMOS

Port B :

Pin	CMOS/OD Output
PB0	CMOS
PB1	CMOS
PB2	CMOS
PB3	CMOS
PB4	CMOS
PB5	CMOS
PB6	CMOS
PB7	CMOS

Appendix D:
XA Listing for Figures 2 and 3

```
; RAMTEST.ASM
$include xa–g3.equ
$pagewidth 132t
;
; PSD311 control registers
;
DDRA    equ             $40
DDRB    equ             $50
PortA   equ             $60
PortB   equ             $70
PinsA   equ             $20
PinsB   equ             $30
;
;
        org     0               ; System exceptions:
        dw      $8f00, Start    ; Reset PSW, Reset vector
; ============================
;  Begin initialization code.
; ============================
        org     100
;
Start:
        mov     R7, #$100       ; initialize stack pointer
;————————————————————————————
; SCR, System Configuration Register
;————————————————————————————
;               76543210
;               0000            ; reserved
;                   00          ; PT1:PT0 = 00 for periph osc/4
;                     0         ; XA mode
;                      1        ; Page 0 mode, uses 16–bit addresses
SCRval  equ     00000001q
;————————————————————————————
; WDCON, Watch Dog Timer Control Register
;————————————————————————————
;               76543210
;               000             ; Prescaler divisor is TCLK*32*2
;                  00           ; reserved
;                    0          ; WDRUN is OFF
;                     0         ; input bit WDTOF
;                      0        ; reserved
WDCONval  equ   00000000q
;————————————————————————————
; BCR, Bus Control Register
;————————————————————————————
;               76543210
;               000             ; reserved
;                  1            ; WAITD: disable EA/WAIT pin
;                   0           ; Bus Disable OFF (bus enabled)
;                    001        ; bc2:0 –> 8–bit data bus, 16–bit address bus
BCRval  equ     00010001q
;————————————————————————————
```

```
; Bus Timing Registers
;_____________________________________
; BTRH
;_____________________________________
;                       76543210
;                       11              ; DW=3  for 4 clock write–w/o–ALE cycle
;                         11            ; DWA=3 for 5 clock ALE–write cycle
;                           11          ; DR=3  for 4 clock read–w/o–ALE cycle
;                             10        ; DRA=2 for 4 clock ALE–read cycle
BTRHval  equ        11111110q
;_____________________________________
; BTRL
;_____________________________________
;                       76543210
;                       1               ; WM1=1  for 2 clock write pulse
;                        1              ; WM0=1  for 1 clock write hold time
;                         1             ; ALEW=1 for 1.5 clock ALE width
;                          0            ; (reserved)
;                           11          ; CR=3   for 4 clock PSEN cycle
;                             10        ; CRA=2  for 4 clock ALE–PSEN cycle
BTRLval  equ        11101110q
;
            mov.b   scr,#SCRval         ; (see above for bit assignments)
            mov.b   wdcon,#WDCONval     ; (see above for bit assignments)
;           mov.b   wfeed1,#$a5         ; Feed watchdog so new config takes effect.
;           mov.b   wfeed2,#$5a
            mov.b   bcr,#BCRval         ; (see above for bit assignments)
            mov.b   btrh,#BTRHval       ; (see above for bit assignments)
            mov.b   btrl,#BTRLval       ; (see above for bit assignments)
;_____________________________________
; Configure the IO port drivers
;_____________________________________
            mov.b   p0cfga,#11111111q   ; Configure port0 for bus(11)
            mov.b   p0cfgb,#11111111q
            mov.b   p1cfga,#11111111q   ; Configure p14–p17 for quasi–bidirec(10),
            mov.b   p1cfgb,#00001111q   ;    A3–A0 for push–pull (11).
            mov.b   p2cfga,#11111111q   ; Configure port2 for push–pull (11)
            mov.b   p2cfgb,#11111111q
            mov.b   p3cfga,#11111111q   ; Configure p35–p30 for quasi–bidirec(10),
            mov.b   p3cfgb,#11000000q   ;    WR(p36), RD(p37) for push–pull (11).
;
; End of initialization, begin user code.
;
                        mov     r1,#$8000; RAM
                        mov     r2,#$7000; not RAM
                        mov     r3,#$00FF
wr1:                    mov.w   [r2],r3 ; word write to outside RAM
                        mov.w   r3,[r1] ; word read from RAM
                        br      wr1

            END
```

NOTES

NOTES

NOTES

Programmable peripherals using the PSD311 with the Philips XA

AN707

DEFINITIONS

Data Sheet Identification	Product Status	Definition
Objective Specification	**Formative or in Design**	This data sheet contains the design target or goal specifications for product development. Specifications may change in any manner without notice.
Preliminary Specification	**Preproduction Product**	This data sheet contains preliminary data, and supplementary data will be published at a later date. Philips Semiconductors reserves the right to make changes at any time without notice in order to improve design and supply the best possible product.
Product Specification	**Full Production**	This data sheet contains Final Specifications. Philips Semiconductors reserves the right to make changes at any time without notice, in order to improve design and supply the best possible product.

Philips Semiconductors
811 East Arques Avenue
P.O. Box 3409
Sunnyvale, California 94088–3409
Telephone 800-234-7381

Let's make things better.

Philips Semiconductors

PHILIPS

Philips Semiconductors — a worldwide company

Argentina: see South America
Australia: 34 Waterloo Road, NORTH RYDE, NSW 2113,
Tel. +61 2 9805 4455, Fax. +61 2 9805 4466
Austria: Computerstr. 6, A-1101 WIEN, P.O. Box 213,
Tel. +43 1 60 101, Fax. +43 1 60 101 1210
Belarus: Hotel Minsk Business Center, Bld. 3, r. 1211, Volodarski Str. 6,
220050 MINSK, Tel. +375 172 200 733, Fax. +375 172 200 773
Belgium: see The Netherlands
Brazil: see South America
Bulgaria: Philips Bulgaria Ltd., Energoproject, 15th floor,
51 James Bourchier Blvd., 1407 SOFIA,
Tel. +359 2 689 211, Fax. +359 2 689 102
Canada: PHILIPS SEMICONDUCTORS/COMPONENTS,
Tel. +1 800 234 7381
China/Hong Kong: 501 Hong Kong Industrial Technology Centre,
72 Tat Chee Avenue, Kowloon Tong, HONG KONG,
Tel. +852 2319 7888, Fax. +852 2319 7700
Colombia: see South America
Czech Republic: see Austria
Denmark: Prags Boulevard 80, PB 1919, DK-2300 COPENHAGEN S,
Tel. +45 32 88 2636, Fax. +45 31 57 1949
Finland: Sinikalliontie 3, FIN-02630 ESPOO,
Tel. +358 615 800, Fax. +358 615 80920
France: 4 Rue du Port-aux-Vins, BP317, 92156 SURESNES Cedex,
Tel. +33 1 40 99 6161, Fax. +33 1 40 99 6427
Germany: Hammerbrookstraβe 69, D-20097 HAMBURG,
Tel. +49 40 23 53 60, Fax. +49 40 23 536 300
Greece: No. 15, 25th March Street, GR 17778 TAVROS,
Tel. +30 1 4894 339/911, Fax. +30 1 4814 240
Hungary: see Austria
India: Philips INDIA Ltd., Shivsagar Estate, A Block, Dr. Annie Besant Rd.,
Worli, MUMBAI 400 018, Tel. +91 22 4938 541, Fax. +91 22 4938 722
Indonesia: see Singapore
Ireland: Newstead, Clonskeagh, DUBLIN 14,
Tel. +353 1 7640 000, Fax. +353 1 7640 200
Israel: RAPAC Electronics, 7 Kehilat Saloniki St, TEL AVIV 61180,
Tel. +972 3 645 0444, Fax. +972 3 649 1007
Italy: PHILIPS SEMICONDUCTORS, Piazza IV Novembre 3,
20124 MILANO, Tel. +39 2 6752 2531, Fax. +39 2 6752 2557
Japan: Philips Bldg. 13-37, Kohnan 2-chome, Minato-ku, TOKYO 108,
Tel. +81 3 3740 5130, Fax. +81 3 3740 5077
Korea: Philips House, 260-199 Itaewon-dong, Yongsan-ku, SEOUL,
Tel. +82 2 709 1412, Fax. +82 2 709 1415
Malaysia: No. 76 Jalan Universiti, 46200 PETALING JAYA, SELANGOR,
Tel. +60 3 750 5214, Fax. +60 3 757 4880
Mexico: 5900 Gateway East, Suite 200, EL PASO, TEXAS 79905,
Tel. +9-5 800 234 7381
Middle East: see Italy
Netherlands: Postbus 90050, 5600 PB EINDHOVEN, Bldg. VB,
Tel. +31 40 27 82785, Fax. +31 40 27 88399
New Zealand: 2 Wagener Place, C.P.O. Box 1041, AUCKLAND,
Tel. +64 9 849 4160, Fax. +64 9 849 7811
Norway: Box 1, Manglerud 0612, OSLO,
Tel. +47 22 74 8000, Fax. +47 22 74 8341
Philippines: Philips Semiconductors Philippines Inc.,
106 Valero St. Salcedo Village, P.O. Box 2108 MCC, MAKATI,
Metro MANILA, Tel. +63 2 816 6380, Fax. +63 2 817 3474
Poland: Ul. Lukiska 10, PL 04-123 WARSZAWA,
Tel. +48 22 612 2831, Fax. +48 22 612 2327
Portugal: see Spain
Romania: see Italy
Russia: Philips Russia, Ul. Usatcheva 35A, 119048 MOSCOW,
Tel. +7 095 926 5361, Fax +7 095 564 8323
Singapore: Lorong 1, Toa Payoh, SINGAPORE 1231,
Tel. +65 350 2538, Fax. +65 251 6500
Slovakia: see Austria
Slovenia: see Italy
South Africa: S.A. PHILIPS Pty Ltd., 195-215 Main Road Martindale,
2092 JOHANNESBURG, P.O. Box 7430, Johannesburg 2000,
Tel. +27 11 470 5911, Fax. +27 11 470 5494
South America: Rua do Rocio 220, 5th Floor, Suite 51,
04552-903 São Paulo, SÃO PAULO-SP, Brazil,
Tel. +55 11 821 2333, Fax. +55 11 829 1849
Spain: Balmes 22, 08007 BARCELONA,
Tel. +34 3 301 6312, Fax. +34 3 301 4107
Sweden: Kottbygatan 7, Akalla. S-16485 STOCKHOLM,
Tel. +46 8 632 2000, Fax. +46 8 632 2745
Switzerland: Allmendstrasse 140, CH-8027 ZÜRICH,
Tel. +41 1 488 2686, Fax. +41 1 481 7730
Taiwan: PHILIPS TAIWAN Ltd., 23-30F, 66,
Chung Hsiao West Road, Sec. 1, P.O. Box 22978, TAIPEI 100,
Tel. +886 2 382 4443, Fax. +886 2 382 4444
Thailand: PHILIPS ELECTRONICS (THAILAND) Ltd.,
209/2 Sanpavuth-Bangna Road Prakanong, BANGKOK 10260,
Tel. +66 2 745 4090, Fax. +66 2 398 0793
Turkey: Talatpasa Cad. No. 5, 80640 GÜLTEPE/ISTANBUL,
Tel. +90 212 279 2770, Fax. +90 212 282 6707
Ukraine: PHILIPS UKRAINE, 4 Patrice Lumumba str., Building B, Floor 7,
252042 KIEV, Tel. +380 44 264 2776, Fax. +380 44 268 0461
United Kingdom: Philips Semiconductors Ltd., 276 Bath Road, Hayes,
MIDDLESEX UB3 5BX, Tel. +44 181 730 5000, Fax. +44 181 754 8421
United States: 811 East Arques Avenue, SUNNYVALE, CA 94088-3409,
Tel. +1 800 234 7381
Uruguay: see South America
Vietnam: see Singapore
Yugoslavia: PHILIPS, Trg N. Pasica 5/v, 11000 BEOGRAD,
Tel. +381 11 825 344, Fax. +381 11 635 777

For all other countries apply to: Philips Semiconductors, Marketing and Sales Communications,
Building BE-p, P.O. Box 218, 5600 MD EINDHOVEN, The Netherlands, Fax . +31 40 27 24825

Internet: http://www.semiconductors.philips.com

SCA51

Printed in the USA

Date of release: 11–96

Let's make things better.

Philips Semiconductors

PHILIPS