



AT45DB161D Atmel SPI FLASH Programmer



A Reference Design for the Spartan-3A Starter Kit
Includes a multi-rate UART

PicoBlaze™

Ken Chapman
Xilinx Ltd

Rev1 – 21st February 2007



Limitations

Limited Warranty and Disclaimer. These designs are provided to you “as is”. Xilinx and its licensors make and you receive no warranties or conditions, express, implied, statutory or otherwise, and Xilinx specifically disclaims any implied warranties of merchantability, non-infringement, or fitness for a particular purpose. Xilinx does not warrant that the functions contained in these designs will meet your requirements, or that the operation of these designs will be uninterrupted or error free, or that defects in the Designs will be corrected. Furthermore, Xilinx does not warrant or make any representations regarding use or the results of the use of the designs in terms of correctness, accuracy, reliability, or otherwise.

Limitation of Liability. In no event will Xilinx or its licensors be liable for any loss of data, lost profits, cost or procurement of substitute goods or services, or for any special, incidental, consequential, or indirect damages arising from the use or operation of the designs or accompanying documentation, however caused and on any theory of liability. This limitation will apply even if Xilinx has been advised of the possibility of such damage. This limitation shall apply notwithstanding the failure of the essential purpose of any limited remedies herein.

This design module is **not** supported by general Xilinx Technical support as an official Xilinx Product. Please refer any issues initially to the provider of the module.

Any problems or items felt of value in the continued improvement of KCPSM3 or this reference design would be gratefully received by the author.

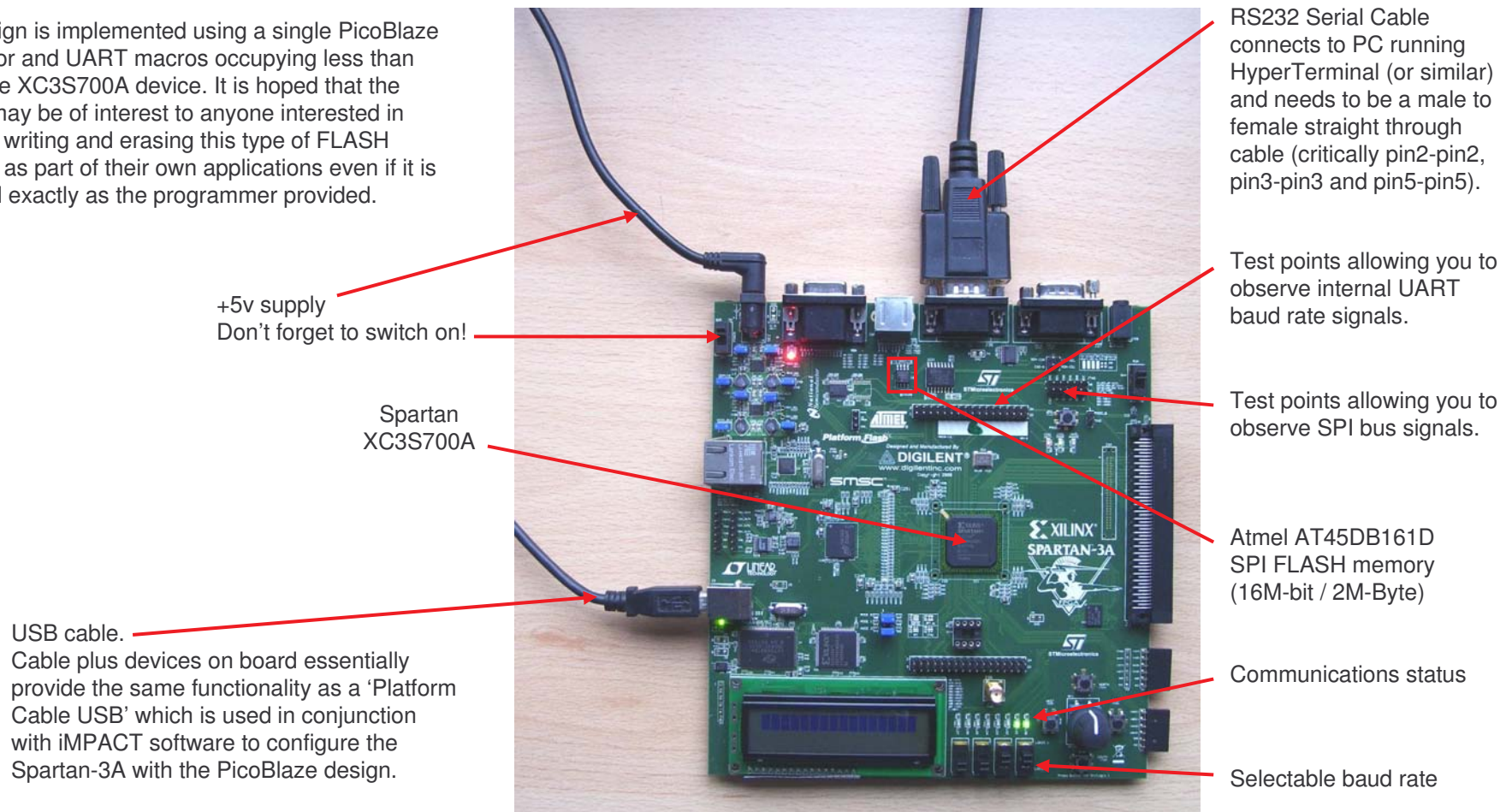
Ken Chapman
Senior Staff Engineer – Spartan Applications Specialist
email: chapman@xilinx.com

The author would also be pleased to hear from anyone using KCPSM3 or the UART macros with information about your application and how these macros have been useful.

Design Overview

This design will transform the Spartan-3A device on your Spartan-3A Starter Kit into a programmer for the 16M-bit (2M-byte) Atmel AT45DB161D DataFlash SPI memory which can be used to hold configuration images for the Spartan device as well provide general non-volatile storage for other applications implemented within the Spartan-3A. This design utilises the RS232 port to provide a connection to your PC. Using a simple terminal program such as HyperTerminal you can then use commands to manually program individual bytes or download complete configuration images for the Spartan-3A device using UFP files. The design also provides commands enabling you to erase 'pages of the memory, read the memory to verify contents and display the device identifier together with the 128-byte security register value.

The design is implemented using a single PicoBlaze processor and UART macros occupying less than 5% of the XC3S700A device. It is hoped that the design may be of interest to anyone interested in reading, writing and erasing this type of FLASH memory as part of their own applications even if it is not used exactly as the programmer provided.



Serial Terminal Setup

The design communicates with your PC using the RS232 serial link. Any simple terminal program can be used, but HyperTerminal is adequate for the task and available on most PCs.

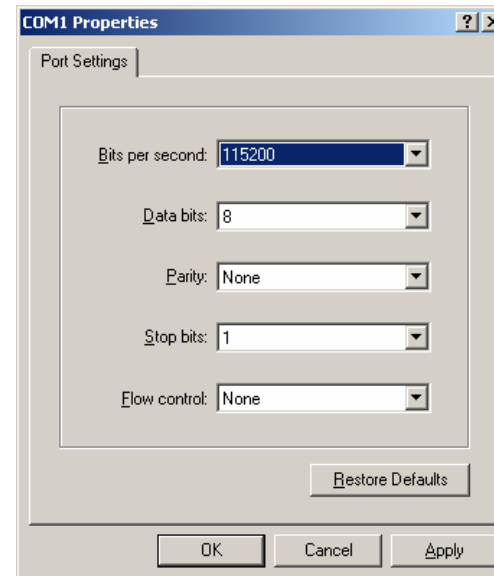
A new HyperTerminal session can be started and configured as shown in the following steps. These also indicate the communication settings and protocol required by an alternative terminal utility.

1) Begin a new session with a suitable name.

HyperTerminal can typically be located on your PC at
Programs -> Accessories -> Communications -> HyperTerminal.



2) Select the appropriate COM port (typically COM1 or COM2) from the list of options. Don't worry if you are not sure exactly which one is correct for your PC because you can change it later.



3) Set serial port settings.

Bits per second :

19200
or 38400
or 57600
or 115200

Match with setting of slide switches SW0 and SW1 (see table).

Hint – Try to be as fast as possible.

Data bits: 8

Parity: None

Stop bits: 1

Flow control: NONE

Baud Rate Setting of Design (also see page 6)

SW1	SW0	LD1	LD0	Baud Rate
Logic 0	Logic 0	Off	Off	19200
Logic 0	Logic 1	Off	On	38400
Logic 1	Logic 0	On	Off	57600
Logic 1	Logic 1	On	On	115200

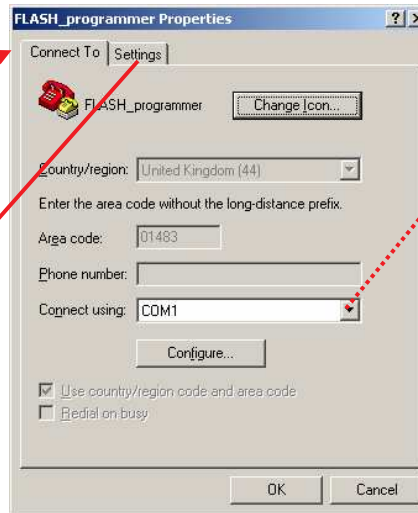
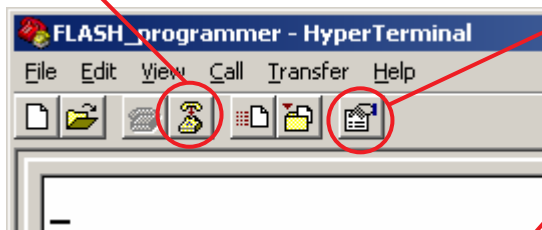
LED's help to confirm your switch selection.

HyperTerminal Setup

Although steps 1, 2 and 3 will actually create a Hyper terminal session, there are few other protocol settings which need to be set or verified for the PicoBlaze design.

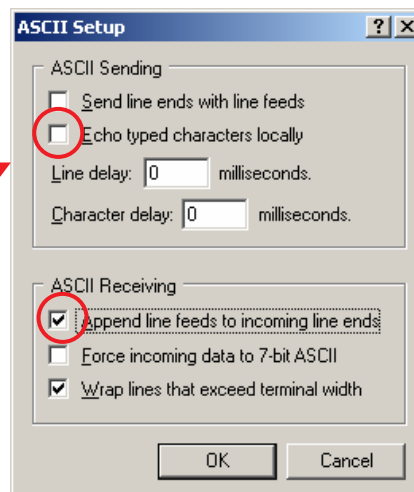
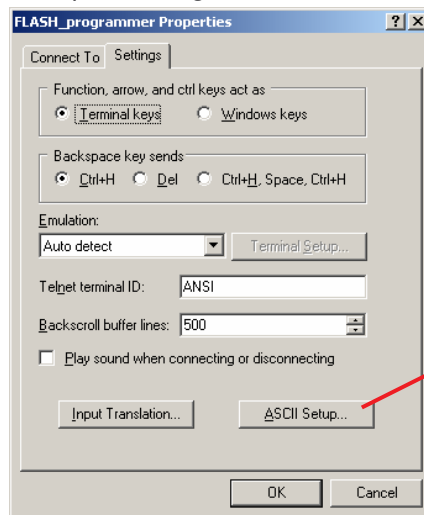
5 - Open the properties dialogue

4 - Disconnect



To select a different COM port and change settings (if not correct).

6 - Open Settings



7 - Open ASCII Setup

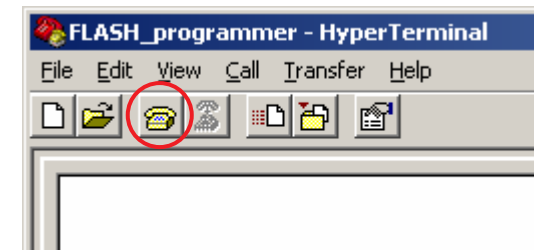
Ensure boxes are filled in as shown.

The design will echo characters that you type so you do not need the 'Echo typed characters locally' option.

The design transmits carriage return characters (OD_{HEX}) to indicate end of line so you do need the 'Append line feeds to incoming line ends' option to be enabled.

Click 'OK' two times to confirm and close dialogue boxes.

8 - Connect



Board Set Up

To make this easy to use the first time and for anyone using this design purely as a FLASH programmer, the compiled configuration BIT file is provided together with a rapid installation batch file. Follow these simple steps to get the design working on your board.

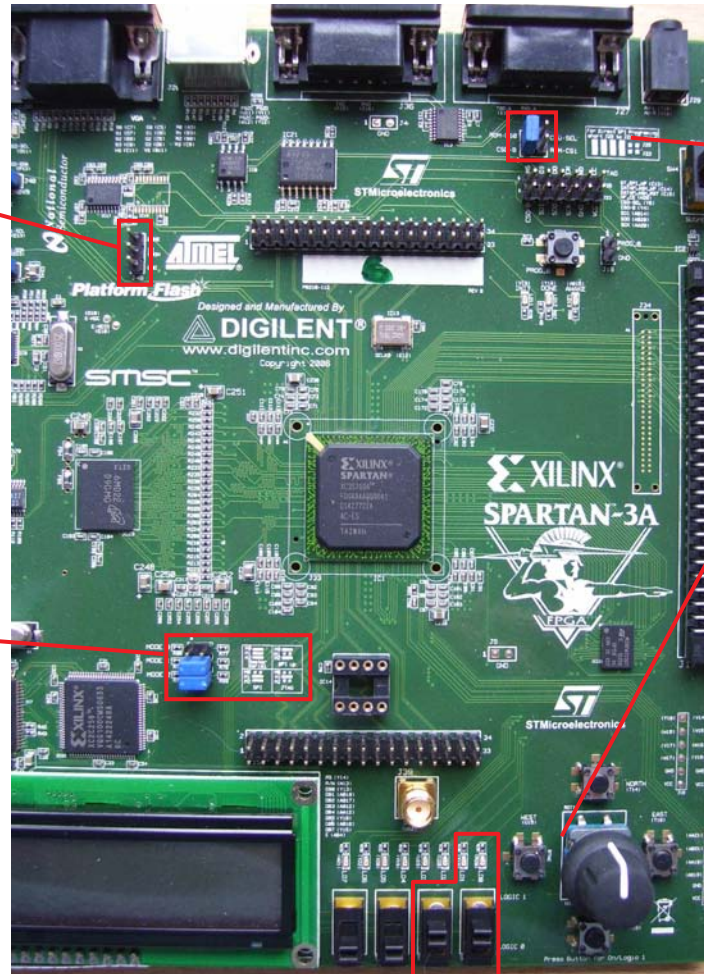
1) Connect the cables, turn on power and set up HyperTerminal all as shown on the previous pages.

2) Set up the board as shown here

Remove any jumpers installed in J46. These jumpers are used to allow selection of the Platform FLASH device whose output shares the same signal wire used by the data output from the Atmel SPI FLASH (miso).

Set Configuration Mode

Insert the lower two jumpers in J26 to select 'SPI' configuration mode. Although this is not a vital to use the programmer design it is highly likely that you will go on to configure the Spartan from the SPI FLASH and then these jumpers must be inserted as shown.



Install the left hand jumper in J1. This links CS0_B to ROM-CS0 enabling the Spartan to access the Atmel SPI FLASH.

Baud Rate Setting of Design

Set the switches SW0 and SW1 to define the baud rate to be used. The LEDs LD0 and LD1 confirm your selection once the design is configured. Start with the highest baud rate and then use the lower rates only if you encounter communication problems or want to experiment.

SW1	SW0	LD1	LD0	Baud Rate
Logic 0	Logic 0	Off	Off	19200
Logic 0	Logic 1	Off	On	38400
Logic 1	Logic 0	On	Off	57600
Logic 1	Logic 1	On	On	115200

Configure Spartan-3A

With your board and PC all ready to go it is time to configure the Spartan-3A with the programmer design.

3) Unzip all the files into a directory.

4) Double click on the file 'install_AT45DB161D_flash_programmer.bat'.

Note that you must have the Xilinx ISE tools installed on your PC

This batch file should open a DOS window and run iMPACT in batch mode to configure the Spartan XC3S700A device with the configuration BIT file provided.

Your terminal session should indicate the design is working with a version number and simple menu as shown here. If not, then check that the Spartan did actually configure (DONE LED on) and check your baud rate settings are correctly matching etc.

The 'H' command repeats the simple list of commands available

Commands can be entered at the > prompt using upper or lower case

```
FLASH_programmer - HyperTerminal
File Edit View Call Transfer Help

PicoBlaze AT45DB161D Programmer v1.21

S-Read Status
E-Erase Pages
P-Program UFP File
W-Write byte
R-Read Page
I-Device ID
T-Test RS232
H-Help

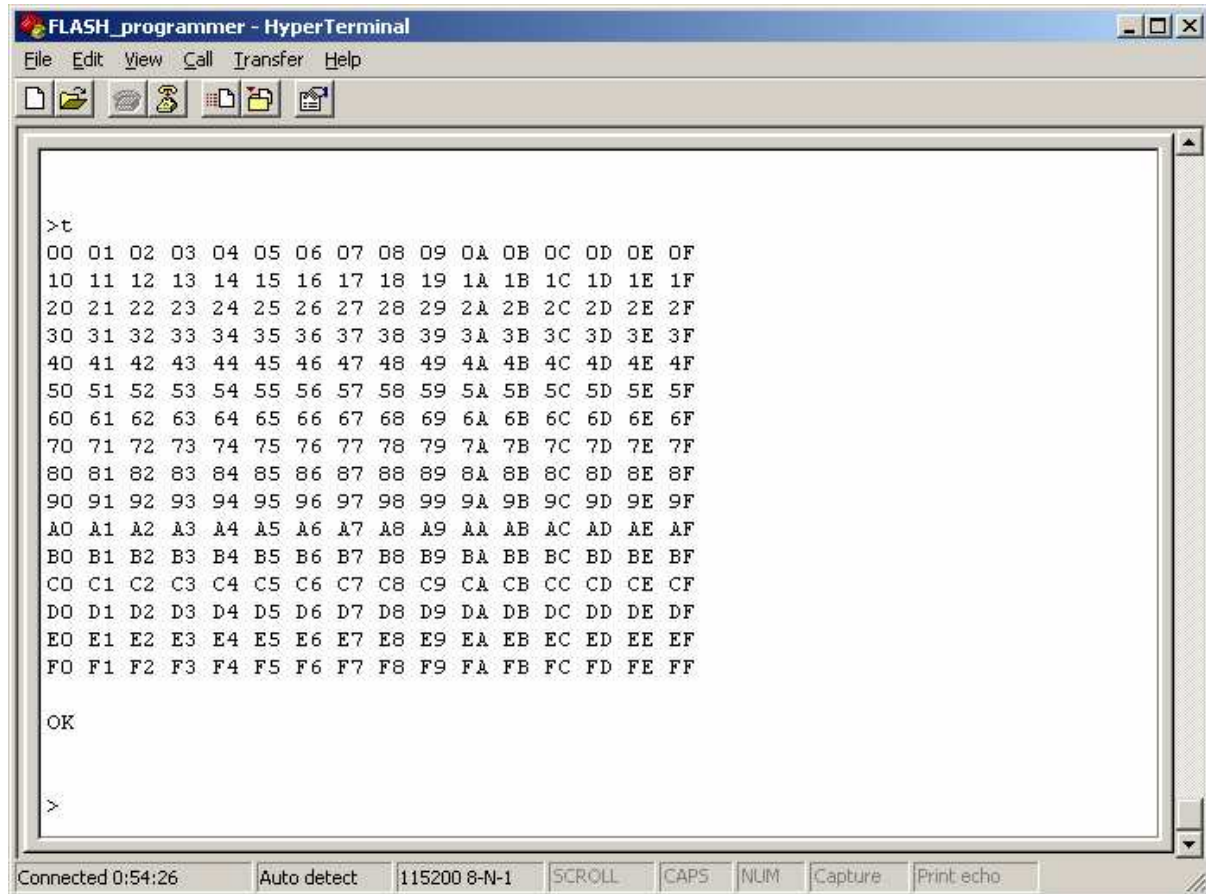
>
```

Test RS232 Command 'T'

The 'T' command is useful for testing that the RS232 communications are working correctly and reliably. The PicoBlaze design has been implemented in such a way that the communications should be reliable during any normal operation up to 115200 baud. However not all PCs are quite so well behaved and this test should make any issues apparent.

PicoBlaze responds to the 'T' command by returning the 256 hexadecimal values 00 to FF in a tabular format as shown here. This display should be 100% reliable.

It should be observed that when holding the 'T' key permanently down for some time that the LED's LD7, LD6, LD5 and LD4 slowly illuminate. This is an indication of the UART receiver FIFO buffer in the Spartan filling up (see FIFO details later). Providing the 'T' key is released as soon as LD4 illuminates then everything displayed in response to a 'T' command should be perfect. If you continue to hold the 'T' key down then eventually you should observe an 'RS232 Overflow' message although it will be hard to spot in given how many tables of values PicoBlaze continues to display.



```
>t
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F
60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F
70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F
80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F
90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF
C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF
D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF
E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF
F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF

OK

>
```

If corrupted values are displayed then try selecting a lower baud rate with the switches remembering that the HyperTerminal settings will need to be changed accordingly.

Status 'S' and ID 'I' Commands

```
>s
Status = AC
Ready
Page Size = 528 bytes
>
```

The 'S' command will display the Status information of the AT45DB161D FLASH memory. This is a single byte response from the FLASH with each bit having the meaning defined below. PicoBlaze is specifically decoding bits 0 and 7 and displaying their meaning in plain text. Note that a page size of 528 bytes is the default.

Bit 7 = RDY/BUSY ('1' = ready / '0' = busy)
Bit 6 = COMP
Bit 5 = '1'
Bit 4 = '0'
Bit 3 = '1'
Bit 2 = '0'
Bit 1 = PROTECT
Bit 0 = PAGE SIZE ('1' = 512 bytes / '0' = 528 bytes)

```
>i
ID = 1F 26 00
Security =
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
06 07 09 0B 38 3A 1F 26 00 00 14 0C FF FF 4F FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
>
```

The 'I' command initially reads the FLASH memory and displays Manufacturer Code (1F hex) and the Device Code for AT45DB161D which is 26 00 hex. This is then followed by the reading and display of the 128-byte security register.

The first 64 bytes of the security register are user one time programmable (OTP). As shown here the default value is 'FF' and it is left as an exercise to program these locations if desired but remember it is OTP memory!

The second 64 bytes of the security register are read only and contain a factory programmed (by Atmel) unique value which can be used to seed design security algorithms or provide registration and product tracking codes.

Hint – Combine the unique FLASH security value with the unique Spartan-3A device DNA to improve security algorithms.

Pages & Address Formats

Before using any of the commands that read, write and erase the AT45DB161D it is important to recognise how the FLASH memory is organised and appreciate the slightly unusual address format required in the default mode. Please take the time to study these two pages because it is critical to using the commands correctly as well as understanding how to work with AT45DB161D memory in your own designs.

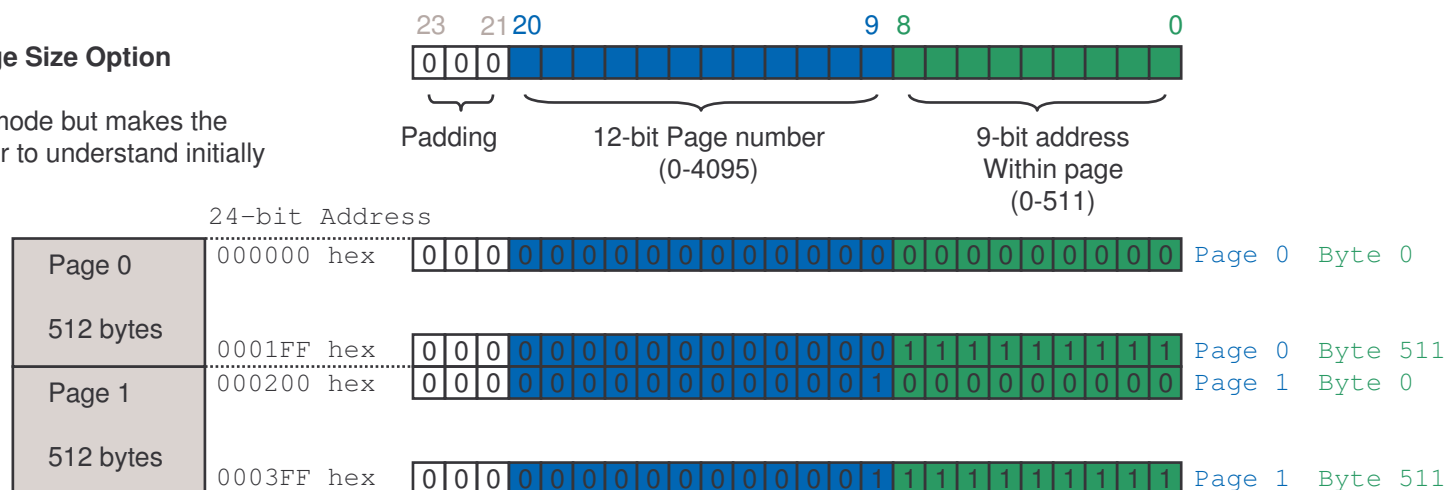
Pages and Bytes

Although an SPI FLASH memory employs a serial communication, all transfers are implemented as bytes (8-bits) and therefore above the very lowest level the whole memory is considered as byte oriented. Internally to the AT45DB161D groups of adjacent bytes are formed into 'pages' and the device has 4096 of these pages. The interesting part is that the default page size is 528 bytes which is not a power of 2 and doesn't feel particularly natural. Indeed the total size of the memory is $4096 \times 528 = 2,162,688$ bytes or 17,301,504 bits rather than 16,777,216 bits which is the normal size for a 16-Mbit memory ($16 \times 1024 \times 1024$).

To make sense of this, let's first look at what Atmel call the "Power of 2" Binary Page Size Option. The AT45DB161D contains a one time programmable (OTP) register that can be programmed to place the device into this mode. Programming of this register is not supported by this reference design but if it is programmed the reference design does know how to read, write and erase the memory correctly. In this mode a page is reduced to 512 bytes and a total of $4096 \times 16 = 65,536$ bytes (16K-bytes) are lost forever. However it is clear that a 9-bit address identifies any byte within a page. More significantly, it can be seen that the 24-bit address format used by the device is linear. As one page ends, then next begins and with it the address appears to roll over in a completely natural way. It is still important to recognize that the memory is organised in pages but from an address format perspective it is all very straightforward as shown below.

"Power of 2" Binary Page Size Option

This is not the default mode but makes the concept of pages easier to understand initially

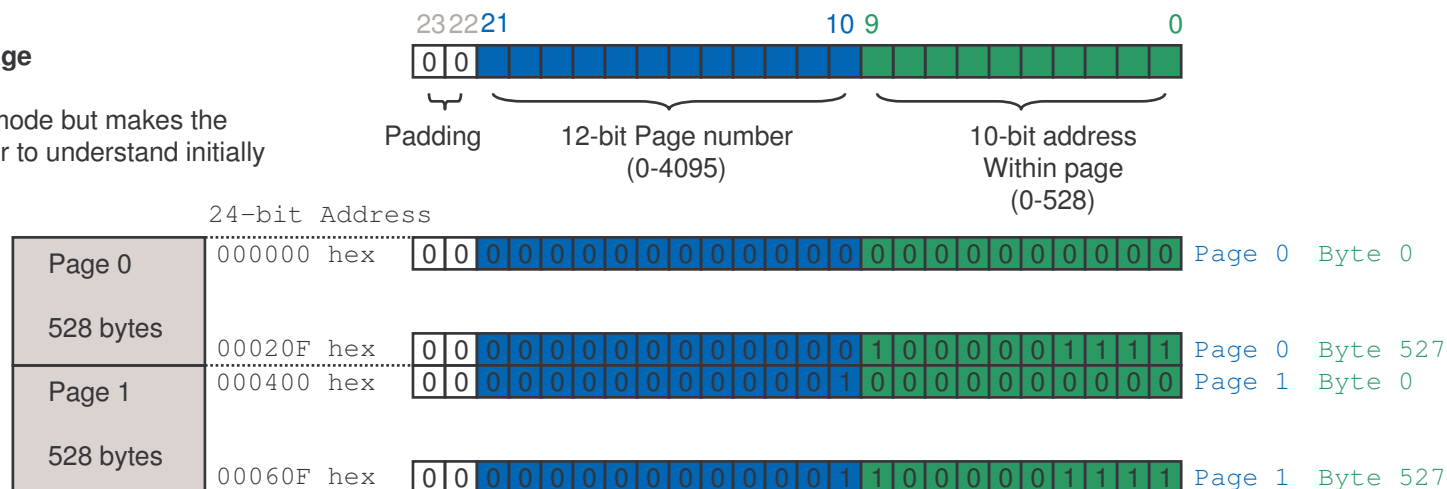


Pages & Address Formats

Returning to the default page size of 528 byte the issue is that it requires 10-bits to describe the address of any byte within a page. The range being 0 to 527 (000 to 20F hex). This also means that as one page ends and the next begins, there is apparently a gap of 496 bytes in the address space even though the pages are really adjacent in the FLASH array. Take time to study the diagram below to appreciate this situation and address format.

Default 528 bytes/page

This is not the default mode but makes the concept of pages easier to understand initially



This reference design has been presented in a way that asks you to specify the page number and byte address (location within a page) but then informs you of the actual 24-bit address values. As such the design is a useful aid to understanding the address format although inevitably you will find yourself performing manual conversions between 24-bit address and pages/byte address formats and the following example may be of value.

A single configuration image for the XC3S700A is 341,580 bytes. If this were to be stored in FLASH starting at page zero and byte address zero it will end at the following address....

"Power of 2" Locations 0 to $341579_{10} = 053664B_{16}$ hex 0000001010011011001001011 Page 667 Byte 75

Default 528 bytes/page $341579_{10} / 528 = 646.93 \Rightarrow$ Pages 0 to 645 completely full and then....
Page 646 is used for $(341579 - 646 \times 528) = 491$ bytes

$646_{10} = 286_{16}$ $491_{10} = 1EB_{16}$

0A19EB hex 00000010100001100111101011 Page 646 Byte 491

Erase Pages Command 'E'

The 'E' command will erase one or several consecutive pages of FLASH memory. PicoBlaze will prompt you to enter the page number of the first and last pages to be erased and then ask you to confirm your intentions to erase the specified memory.

```
>e
First
Page = 047
Last
Page = 105

Confirm Erase (Y/n) Y
Erase in Progress

.....
.....
.....

OK

>
```

Specify page numbers in hexadecimal (000 to FFF). PicoBlaze will reject a Last Page that is less than the First Page. Use the same number to erase a single page.

An upper case 'Y' is required to confirm erase operation otherwise the command will 'Abort'

A dot will be displayed each time a page is page erased to indicate progress.

As described previously, the AT45DB161D FLASH array is organised into 4096 pages of 528 bytes (default) or 512 bytes. A 'page' is the smallest amount of memory that can be erased and may take up to a maximum of 35ms to complete. If all 4096 pages are erased one page at a time it could take up to 144 seconds to erase the whole device although I found it to be just 58 seconds on my board which is consistent with the typical page erase time of 13ms stated in the Atmel data sheet.

Atmel have further organised their FLASH memory into 'blocks' and 'sectors'. A block is formed of 8 pages and a sector is formed of 32 blocks (256 pages). There are then device commands to erase blocks and sectors which reduce erase times when clearing large sections of memory. It should be possible to erase the whole device in 23 seconds (typical) to 51 seconds (worst case). Of course the coarse granularity may not always be as desirable.

Exercise – Implement a block erase command.

Write Byte Command 'W'

The 'W' command allows you to write an individual byte of data to any address in FLASH memory and is useful for setting up small data patterns or test values. PicoBlaze works with the AT45DB161D in a particular way that allows a single byte to be modified without erasing the memory first and therefore the memory can be locally modified as if it were an EEPROM rather than FLASH device.

```
>w
Page = 286
byte address = 1eb

Data = 42
Full address = 0A19EB

>
```

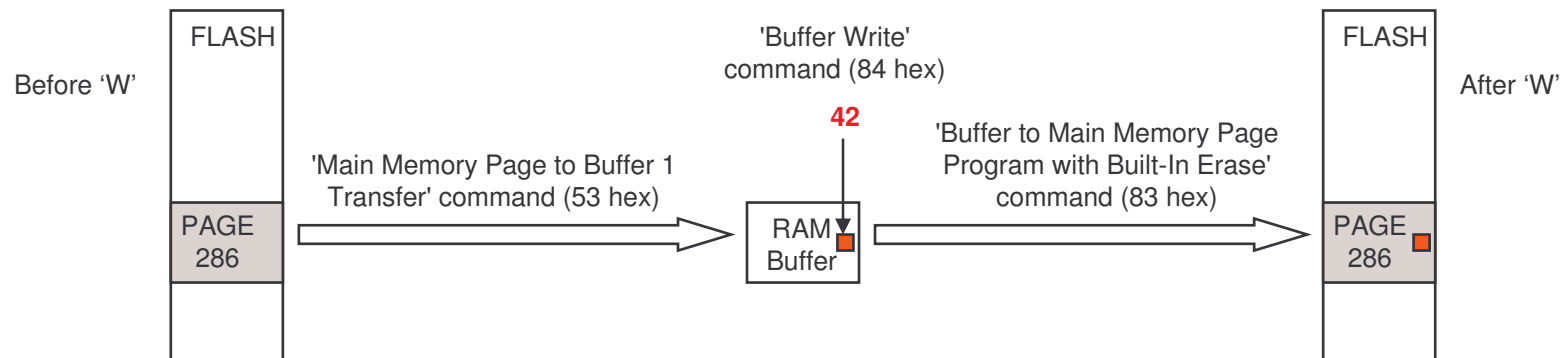
After entering the 'W' command you will be prompted to enter a hexadecimal page number in the range 000 to FFF and then the byte address within that page (range 000 to 20F for default page size or 000 to 1FF for power of 2 page size).

You will then be prompted to enter the data value and you should enter a 2 digit hexadecimal value 00 to FF.

PicoBlaze will display the 24-bit address associated with the page and byte address specified and will write the data. Hint – Look back at page 11 to see how this example works.

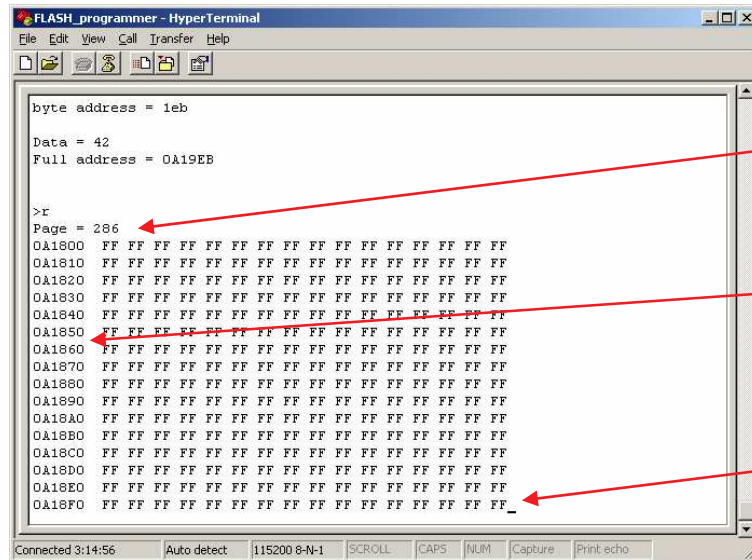
How does it work?

The AT45DB161D device contains two RAM buffers in addition to the FLASH array. Each buffer is sufficient to hold one complete page (528 bytes) of information. So to implement the write command above, PicoBlaze first commands the AT45BB161D to copy the contents of the target page from the FLASH array and into buffer 1. It then modifies the specific byte in the RAM buffer before writing the entire contents of the buffer back into the FLASH array using a built-in erase operation. In this way a whole page is erased but it has the overall effect of looking like a single byte has been modified.



Read Page Command 'R'

The 'R' command allows you to read and display the 528 (default) or 512 bytes contained in a page of the FLASH memory to verify contents.



```
FLASH_programmer - HyperTerminal
File Edit View Call Transfer Help

byte address = 1eb
Data = 42
Full address = 0A19EB

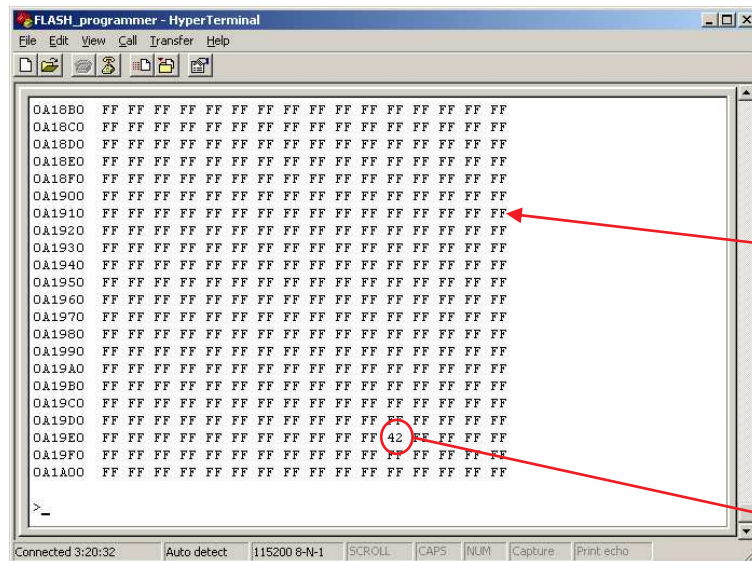
>r
Page = 286
0A1800 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1810 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1820 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1830 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1840 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1850 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1860 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1870 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1880 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1890 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A18A0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A18B0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A18C0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A18D0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A18E0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A18F0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
_
```

After entering the 'R' command you will be prompted to enter the page number in the range 000 to FFF hex. In this case I have entered page 286 to look for the data written on the previous page ('W' command).

The display indicates the 24-bit address of the first byte shown on each line followed by 16 successive bytes.

Hint – Use this to help you confirm address values for page/byte address values.

Because a page is so big, PicoBlaze will pause the display after the first 256 bytes have been displayed. Pressing any key at the PC keyboard will instruct PicoBlaze to continue displaying the remaining page contents.



```
FLASH_programmer - HyperTerminal
File Edit View Call Transfer Help

0A18B0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A18C0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A18D0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A18E0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A18F0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1900 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1910 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1920 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1930 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1940 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1950 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1960 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1970 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1980 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1990 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A19A0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A19B0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A19C0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A19D0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A19E0 FF FF FF FF FF FF FF FF FF 42 FF FF FF FF FF
0A19F0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1A00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

>_
```

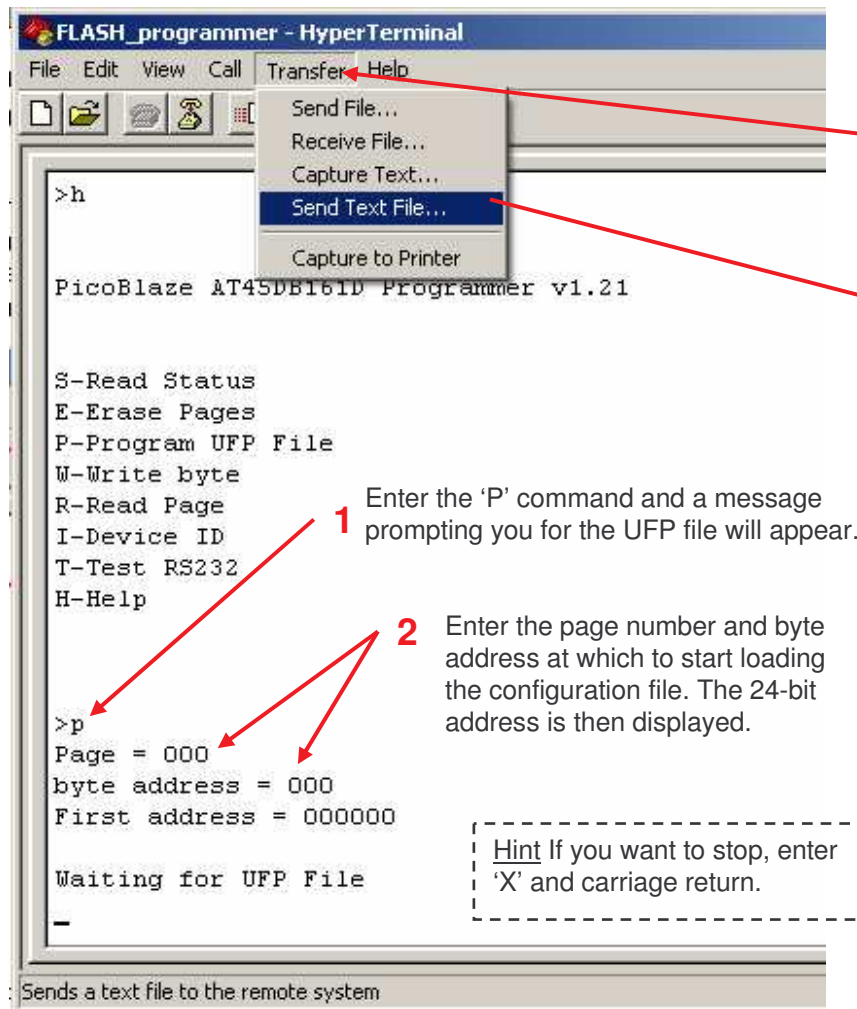
This screen shot shows the display after allowing PicoBlaze to continue.

Erased memory locations contain the value 'FF' and the 'P' command which is covered on the following pages will only work reliably if the memory is erased first.

Read display shows how address 0A19EB (page 286 and byte address 1EB) was modified to 42 hex by previous the 'W' command.

Program UFP File Command 'P'

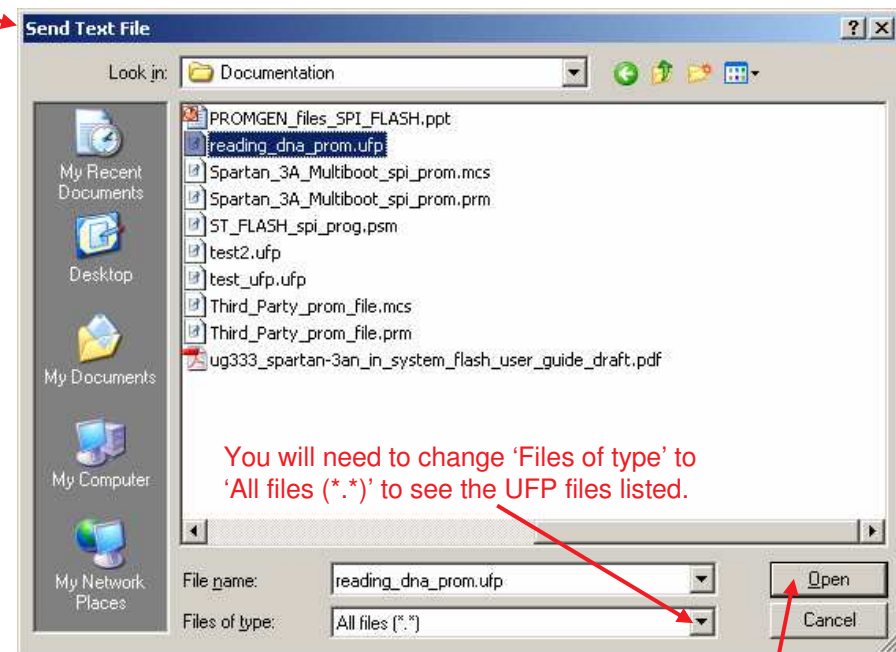
The 'P' command enables you to write an entire configuration image for the Spartan-3A device into the FLASH memory such that the Spartan device can then automatically configure from that image the next time power is applied to the board (or the PROG button is pressed). The following pages describe how to prepare an UFP file, but for now this page shows how to program the 'reading_dna_prom.ufp' file provided with this reference design and it is recommended that you try this particular file first.



IMPORTANT – Ensure you have erased the pages to be programmed (000 to 286 in this example) before using the 'P' command.

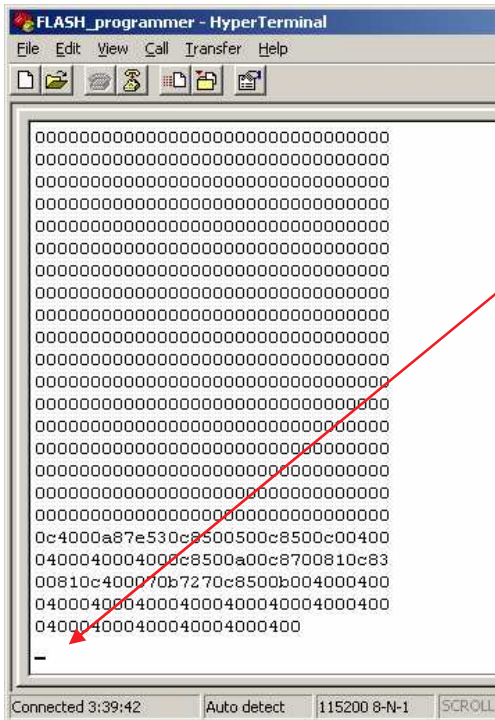
3 In HyperTerminal, select the 'Transfer' menu and then select the 'Send Text File' option (Note: Do not use the 'Send File' option).

4 Navigate to the appropriate directory and select the desired UFP file which in this case is 'reading_dna_prom.ufp'.



Once you are happy with your selection click on 'Open'. **5**

'P' Command continued



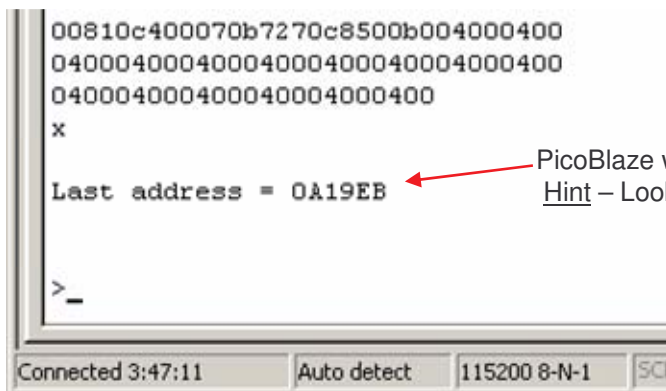
Programming of the FLASH memory will start immediately and PicoBlaze will indicate progress by echoing the UFP file back to the PC display. You will also see the LD7 on the board blinking as the UART receives data form the PC (see FIFO design later). Programming will typically take **82 seconds** to complete when using a baud rate of 115200.

The UFP file is a pure ASCII hexadecimal representation of the configuration data and there is nothing to indicate to PicoBlaze that the file transmission has completed. Therefore to finish enter 'X' followed by carriage return (in fact any no hexadecimal characters will also stop it).

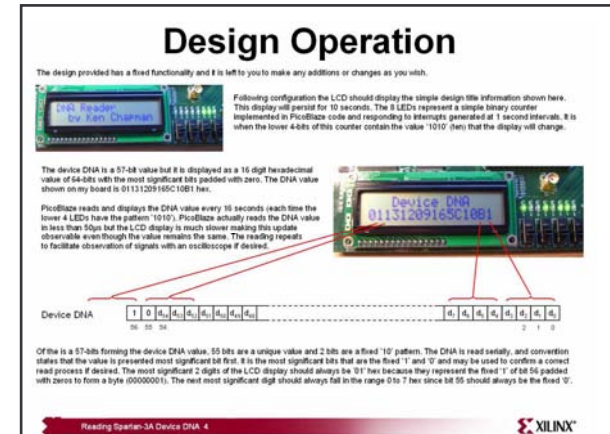
Exercise – Implement a ‘time out’ to automatically end programming.

It should now be possible to press the PROG button on the board to reconfigure the Spartan device directly from the new configuration image stored in the parallel FLASH memory.

If you used the supplied 'reading_dna_prom.ufp' file, then your board should now be displaying the unique DNA value of your Spartan-3A on your board. This reference design is also available from the Xilinx web site.



PicoBlaze will display the last 24-bit address programmed
Hint – Look back at page 11 to see how this example works.



Obviously once you have reconfigured the Spartan-3A using the image stored in FLASH memory the programmer design will have to be restored if you want to use it again.

```
'install AT45DB161D flash programmer.bat'.
```


UFP Files and SPI Configuration

If you use the 'R' command to check that the configuration image has been correctly programmed into the device then you may initially become alarmed to see that many values are different to those contained in the UFP file. However this is correct behavior and can be explained.

reading_dna_prom.ufp
First 256 bytes

[illegible]

Page 000 read back from FLASH
First 256 bytes

```


>r
Page = 000
000000  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000010  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000020  AA 99 30 A1 00 07 20 00 31 61 09 EE 33 21 3C 0F
000030  31 A1 00 C9 31 41 2F 00 31 C2 02 22 80 93 30 E1
000040  FF CF 30 C1 00 81 31 81 08 81 32 01 00 1F 32 C1
000050  00 05 32 E1 00 04 32 A1 00 0E 32 61 00 00 32 81
000060  00 00 33 41 18 F2 33 62 00 00 00 00 30 22 00 00
000070  00 00 30 A1 00 01 50 60 00 02 9A C2 00 00 00 00
000080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

A critical issue with configuration is that the Spartan must be able to read the data in the correct order. When configuring from Xilinx Platform FLASH serial memory the data is read least significant bit (LSB) first. However an SPI FLASH memory is read most significant bit (MSB) first and this requires the configuration data to be stored with the bits within each byte swapped (reversed or mirrored). In the 9.1i version of ISE the PROM Generator always writes UFP files with the data in normal direct format and therefore PicoBlaze swaps the bits before writing to the SPI FLASH.

4C Hex = 0100 1100₂

32 Hex = 0011 0010₂



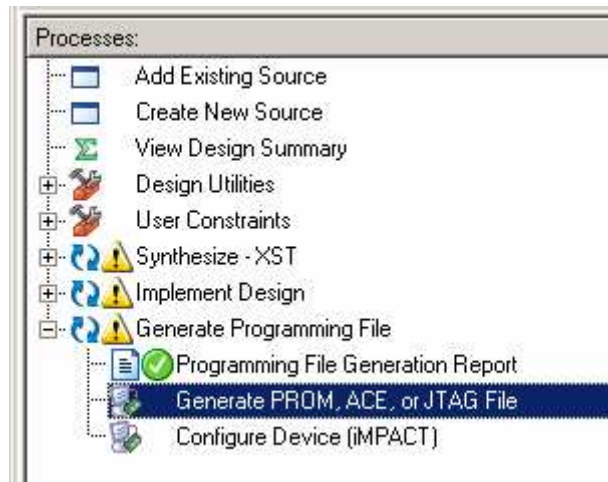
The diagram shows two lines of text. The first line is '4C Hex = 0100 1100₂' and the second line is '32 Hex = 0011 0010₂'. Two arrows originate from the first line: one from the '1' in '1100' pointing to the '1' in '0011' of the second line, and another from the '0' in '0100' pointing to the '0' in '0010' of the second line. This illustrates a bit swap between the two hex values.

This example shows how the byte 4C hex was written to FLASH memory as value 32 hex.

Preparing a UFP file

This reference design has been provided so that the contents of a UFP file can be programmed into the Atmel FLASH device. The following images indicate how a Spartan-3A configuration can be made into a UFP using the ISE tools but this is not intended to replace the existing documentation for PROM generation.

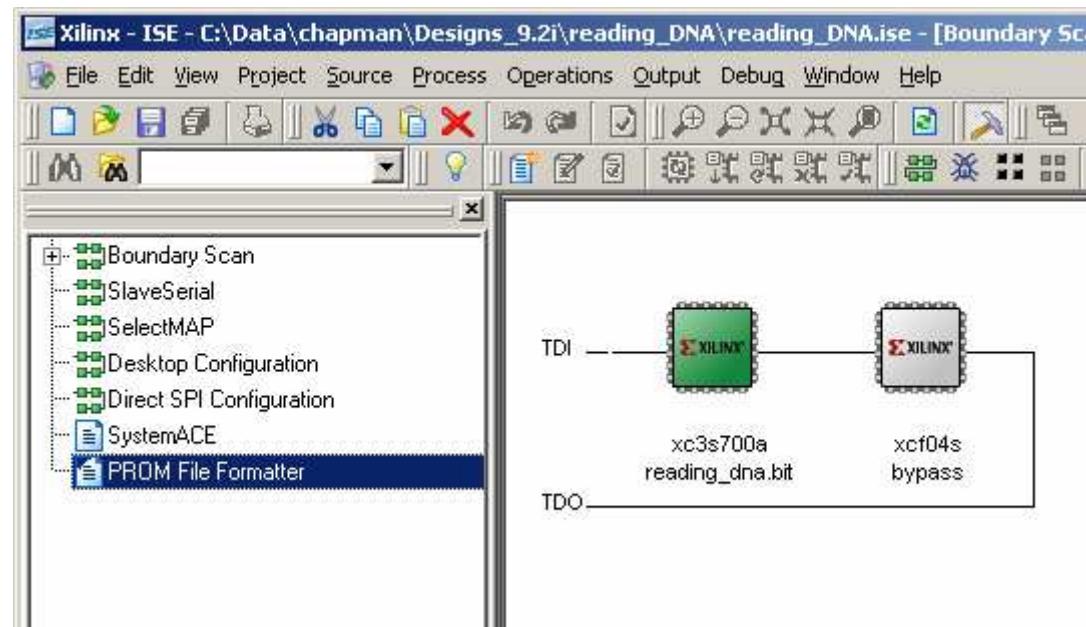
1) Select 'Generate PROM' in Project Manger



Note that a UFP file only contains data and the PicoBlaze design is used to specify the address at which the data should be programmed in the Atmel device.

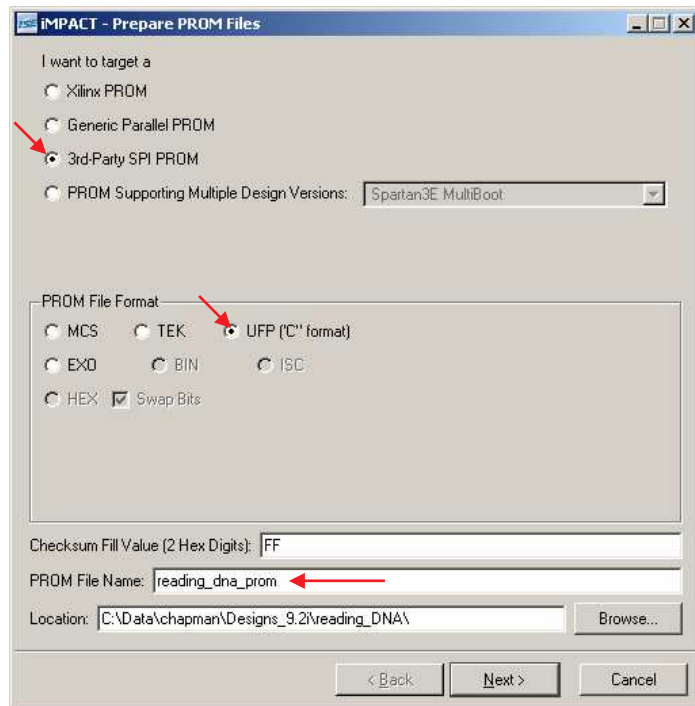
Hint – UFP files do not contain any address information allowing you to experiment with storing configuration images at any locations in the SPI FLASH. Spartan-3A supports multi-boot from SPI FLASH and the AT45DB161D is large enough to hold 6 configuration images for the XC3S700A device provided on the Starter Kit board.

2) This launches iMPACT in which you need to select (double click) the PROM File Formatter mode (You may need to expand the upper left window as shown here or pan down to see it)..

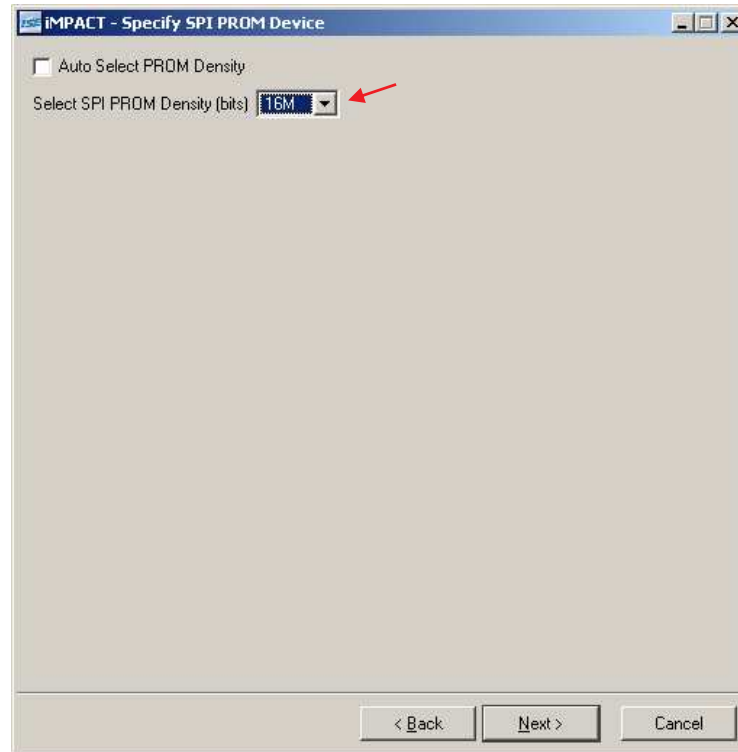


Preparing a UFP file

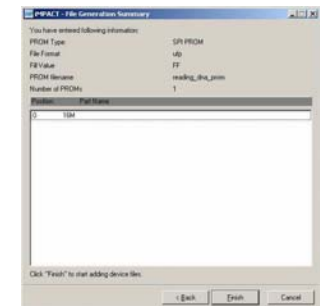
- 3) Select
 '3rd Party SPI PROM' ** to be consistent with the type of
 FLASH we are actually trying to program
 'UFP ("C" format)' for the PROM File Format.
 ...and provide a file name and location.



- 4) Select the density from the drop down list.
 The AT45DB161D is a 16M-bit FLASH.



- 5) Summary
 Page

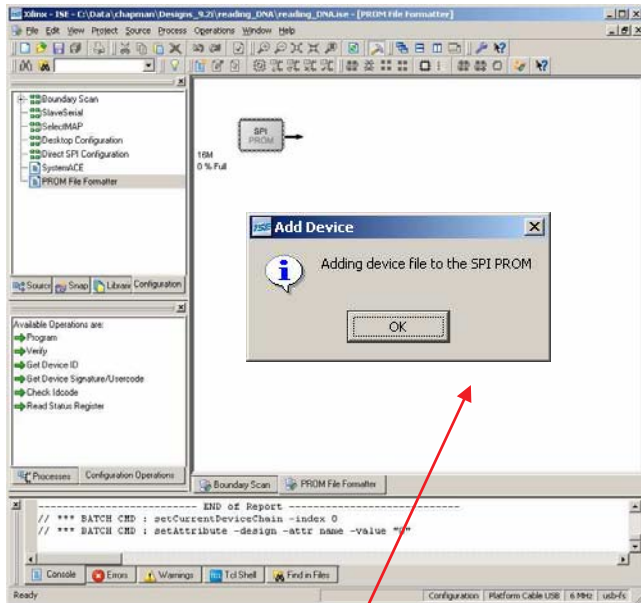


** Note for future reference

It is possible that a future release of ISE tools (after 9.1i) may write UFP files for SPI FLASH in a way that already has the bits of the bytes swapped (see page 17). If that does occur, simply use 'Generic Parallel PROM' to generate your UFP file or remove the bit swapping code from the PicoBlaze program.

Preparing a UFP file

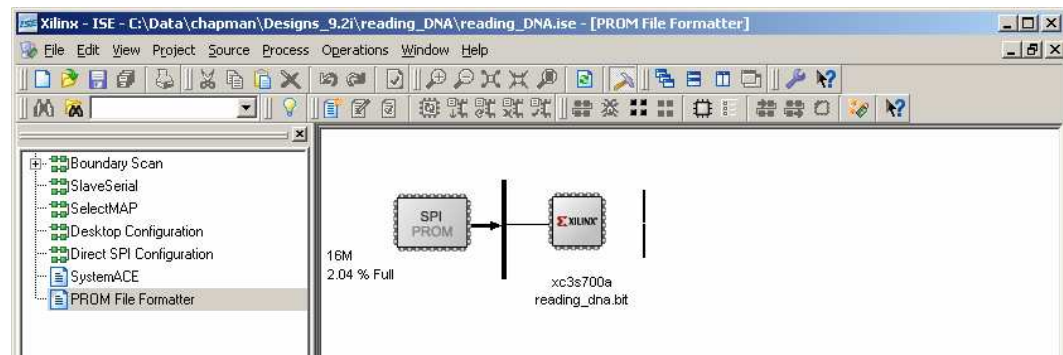
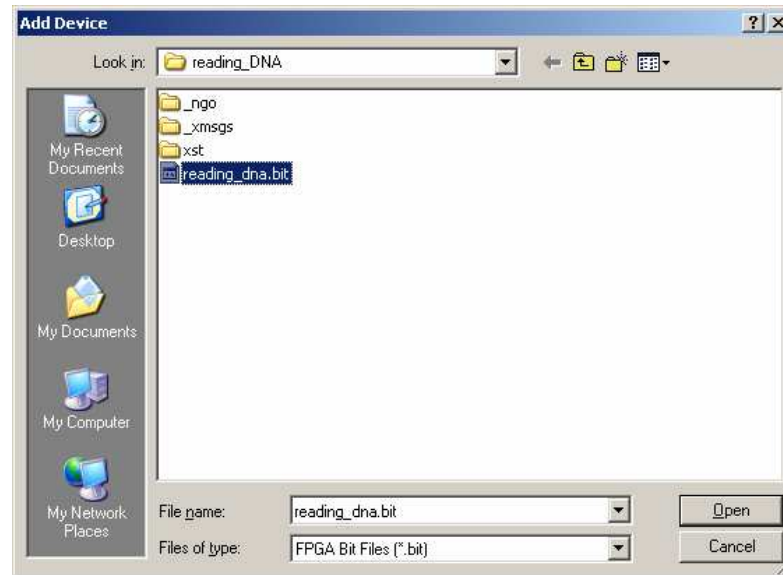
6) You are now presented with a picture of the PROM contents and an 'Add Device' box encouraging you to add your first device. Click 'OK' to continue.



Hint: If the 'Add Device' box does not appear, then right click in the white space and select 'Add Xilinx Device...'

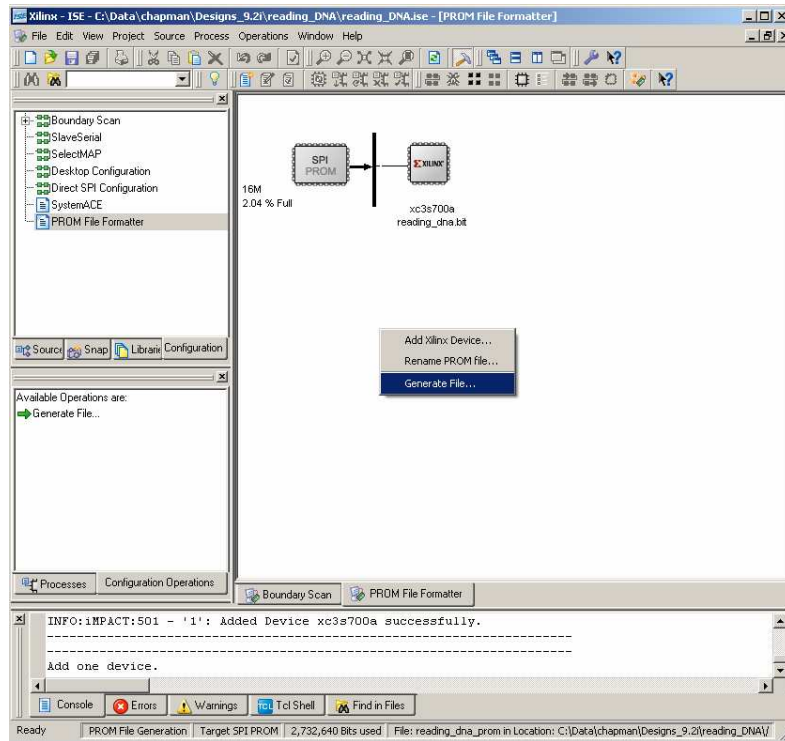
8) The main window updates to show the BIT file being located at the beginning of the PROM.

7) Navigate to the required configuration BIT file, select the file then click 'Open'.

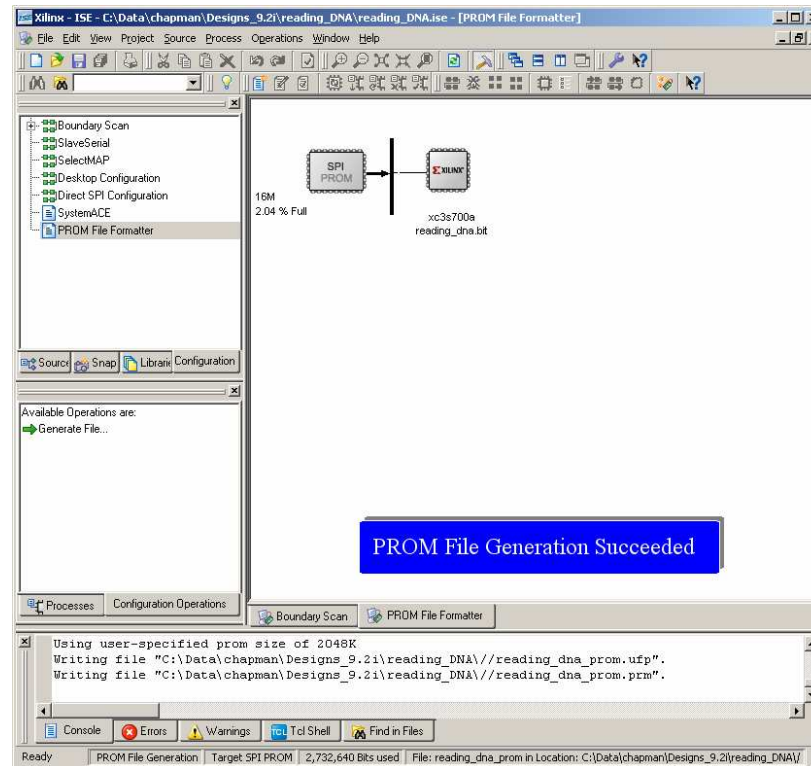


Preparing a UFP file

9) Right click in the white space of the main window and then select 'Generate File...' from the pop up box



10) The file is written to the directory specified in step 3 and the process is complete.

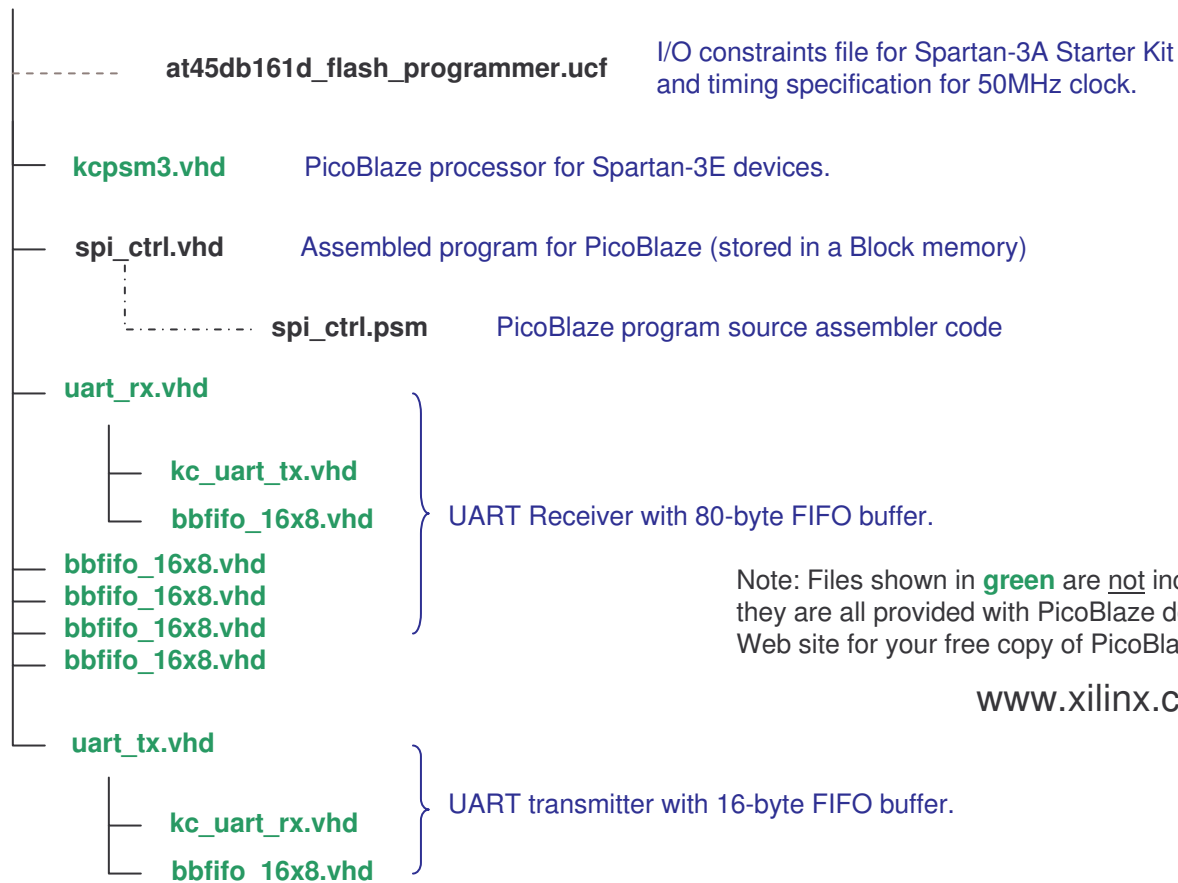


Design Files

For those interested in the actual design implementation, the following pages provide some details and an introduction to the source files provided. As well as these notes, the VHDL and PicoBlaze PSM files contain many comments and descriptions describing the functionality. Remember to have a copy of the Atmel AT45DB161D data sheet available when reading the various comments as this explains the commands of the FLASH memory in greater detail.

The source files provided for the reference design are.....

at45db161d_flash_programmer.vhd Top level file and main description of hardware.



www.xilinx.com/picoblaze

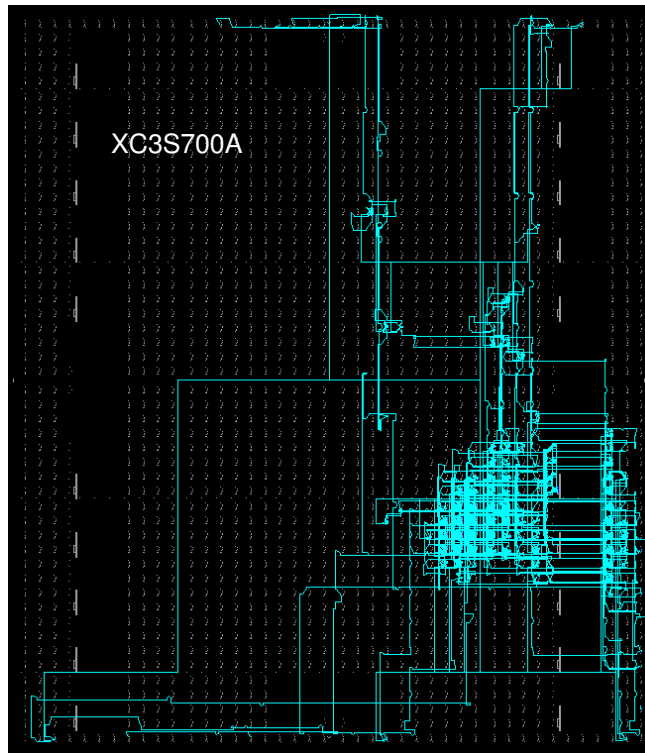
PicoBlaze Design Size

This reference design occupies less than 5% of the XC3S700A device. The PicoBlaze program uses the majority of the single Block RAM (RAMB16BWE) although in this case nearly 34% of the program is consumed by text strings used to guide the user of the programmer (e.g. command menu).

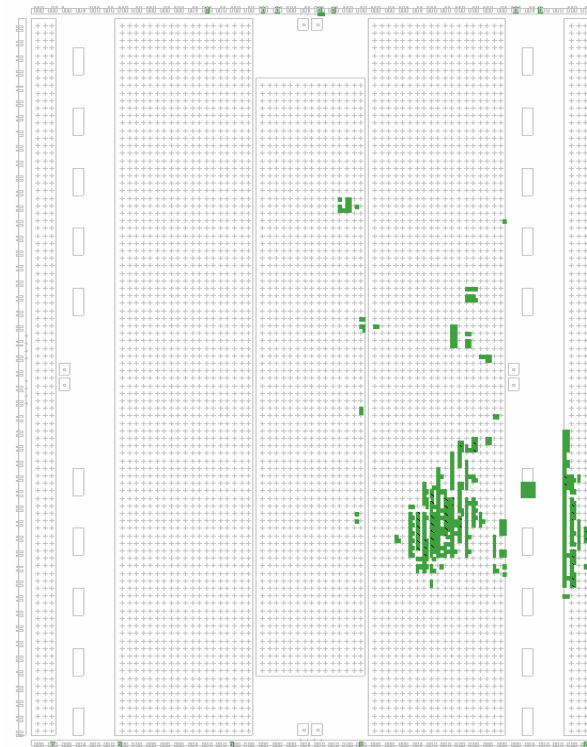
MAP report

Number of occupied Slices:	188 out of	5,888	3%
Number of RAMB16BWEs:	1 out of	20	5%

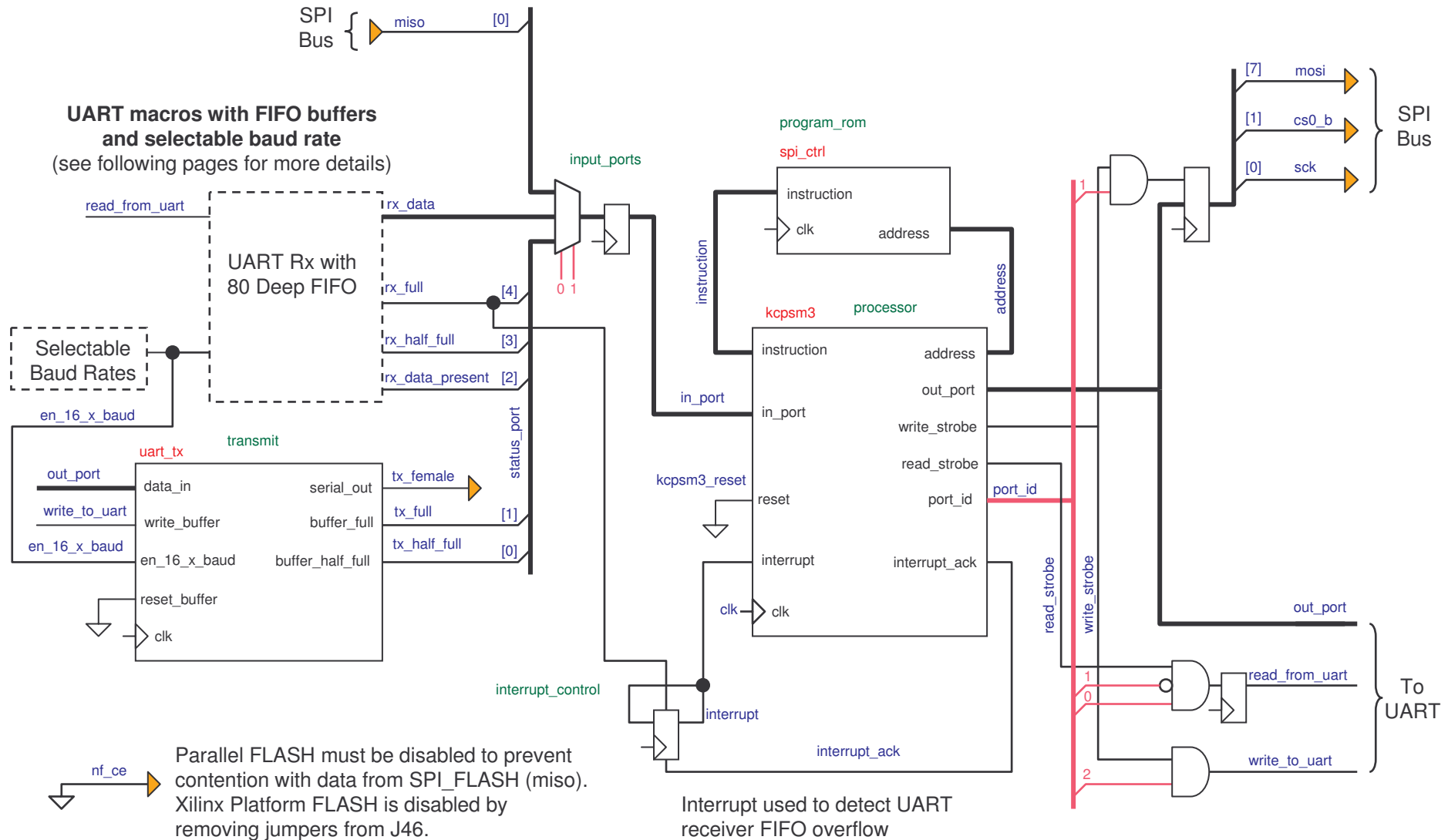
FPGA Editor view



Floorplanner view



PicoBlaze Circuit Diagram



Software Based SPI

At the heart of SPI communication with the AT45DB161D is the requirement to implement a full duplex operation in which byte data is simultaneously shifted to and from the FLASH memory most significant bit first. PicoBlaze is ideally suited to this task as it is able to be programmed at a low level and the timing is completely predictable. However it should be recognised that whilst using software to implement such signaling is cost efficient, flexible and easy to implement, it does not result in the fastest communication. In this reference design the 50MHz oscillator on the board is used and with this PicoBlaze is able to implement an SPI clock (SCK) rate of 1.786MHz. This is adequate for this application similar to this one but falls way short of the 66MHz maximum rate supported by the AT45DB161D device and a hardware based peripheral should be implemented if maximum communication rate is desirable.

The following PicoBlaze code shows the routine which transmits and receives a byte of information using the SCK (clock), MOSI (Master Out, Slave In) and MISO (Master In, Slave Out) signals of the SPI bus. This code, along with all the source code provided with this reference design is fully commented to help you understand and reuse in your own designs. In this case the register 's3' contains the byte value to be transmitted and on return it is 's3' that will contain the byte received from the AT45DB161D device.

```
SPI_tx_rx: LOAD s1, 08                ;8-bits to transmit and receive
           FETCH s0, SPI_bus_status    ;read current bus status
next_SPI_tx_rx_bit: LOAD s2, s3        ;determine next MOSI to be transmitted
           AND s2, 80                  ;isolate bit in transmit byte
           AND s0, 7F                  ;clear bit7 ready for MOSI
           OR s0, s2                   ;set bit7 to drive MOSI if data is High
           OUTPUT s0, SPI_out_port
           INPUT s2, SPI_in_port        ;read MISO
           TEST s2, SPI_miso            ;detect state of received bit
           SLA s3                      ;shift new data into result and move to next transmit bit
           XOR s0, SPI_sck              ;drive SCK clock High
           OUTPUT s0, SPI_out_port
           XOR s0, SPI_sck              ;drive SCK clock Low
           OUTPUT s0, SPI_out_port
           SUB s1, 01
           JUMP NZ, next_SPI_tx_rx_bit ;repeat until finished
           RETURN
```

The remaining signal used in SPI communication is the enable (CS0_B) which must be Low for communication to take place with the AT45DB161D device.

Drives CS0_B Low to start communication

```
SPI_flash_enable: CALL SPI_init
                  XOR s0, SPI_rom_cs
                  OUTPUT s0, SPI_out_port
                  STORE s0, SPI_bus_status
                  RETURN
```

Drives CS0_B High to end communication

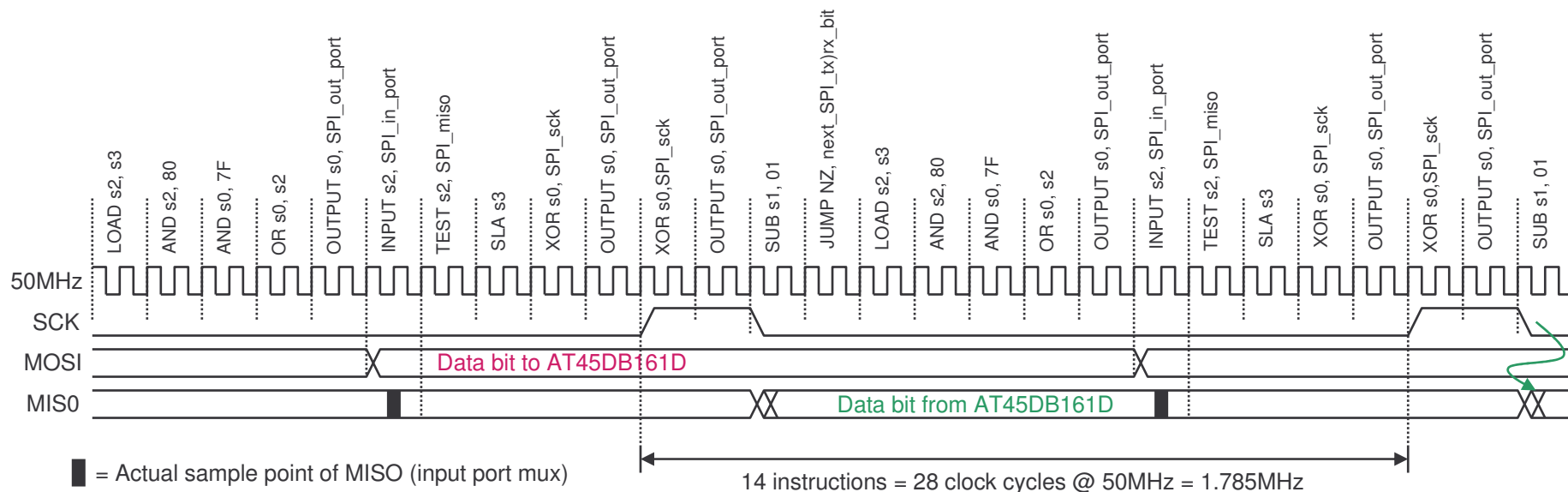
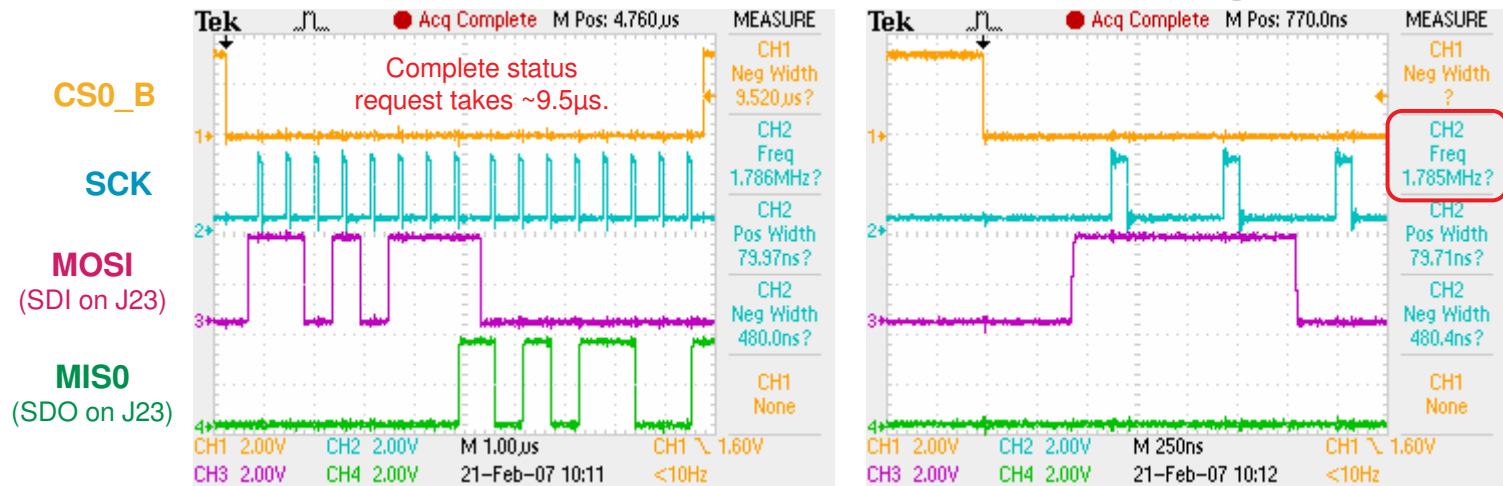
```
SPI_init: LOAD s0, 02
           OUTPUT s0, SPI_out_port
           STORE s0, SPI_bus_status
           RETURN
```

SPI Communication Timing

These oscilloscope traces show the SPI signals observed at the J23 show the timing of the SCK during a device status request (see 'S' command on page 8).

Status command
D7 hex = 11010111₂

Typical Response
AC hex = 10101100₂

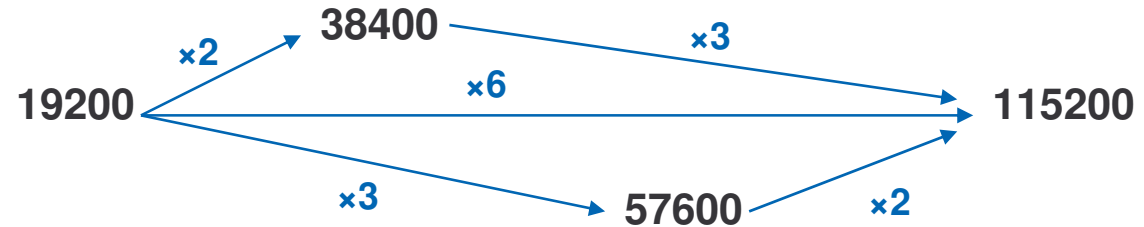


Since every PicoBlaze instruction executes in 2 clock cycles and the design uses the 50MHz clock source on the board, the actual SPI bit rate can be predicted and this is confirmed by the oscilloscope traces above.

Selectable Baud Rates

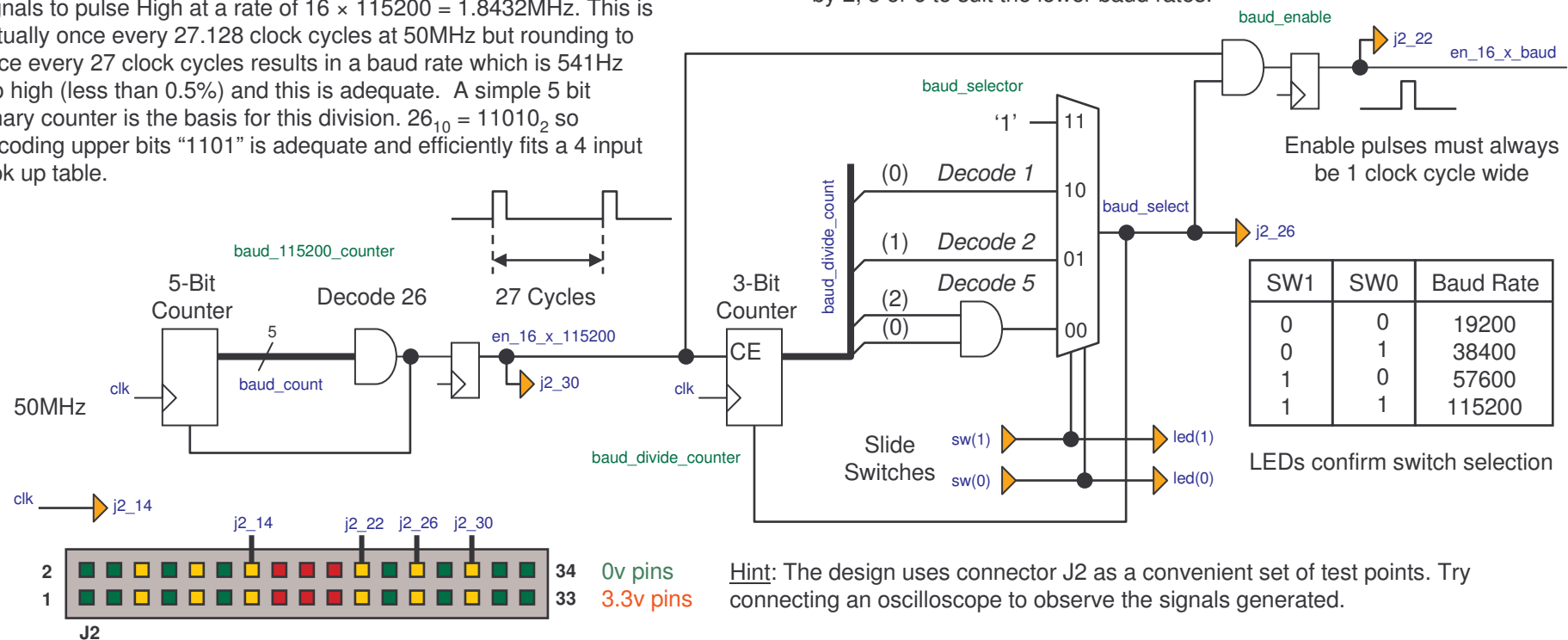
Baud rates are related by integer multiples and this simplifies the generation of the baud rate control signal used by the UART macros.

The UART_RX and UART_TX macros serially communicate at the rate defined by clock enable pulses applied to the 'EN_16_X_BAUD' input which, as the name suggests, is an input that should be enabled as 16 times the required baud rate. The Spartan-3A Starter kit is provided with a 50MHz clock from which this enable signal is defined.



The highest baud rate of 115200 requires the 'EN_16_X_BAUD' signals to pulse High at a rate of $16 \times 115200 = 1.8432\text{MHz}$. This is actually once every 27.128 clock cycles at 50MHz but rounding to once every 27 clock cycles results in a baud rate which is 541Hz too high (less than 0.5%) and this is adequate. A simple 5 bit binary counter is the basis for this division. $26_{10} = 11010_2$ so decoding upper bits "1101" is adequate and efficiently fits a 4 input look up table.

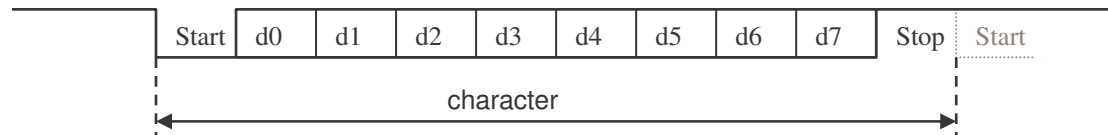
A second counter can then be used to divide the fast enable signal by 2, 3 or 6 to suit the lower baud rates.



Baud Rates vs FLASH Write Rates

In the majority of cases it is the RS232 (UART) communication rate with the PC which is the limiting factor of this design. However, when writing to the FLASH there is a potential for the performance of the FLASH memory itself to limit performance and this requires some analysis to ensure reliable operation.

A UART transmits or receives each ASCII character as a series of 10 bits communicated at the baud rate. This then defines the time taken to communicate each character.



Data to be stored in the FLASH memory is sent transmitted from the PC in the form of an MCS file which describes the data using ASCII characters. Therefore it requires 2 characters to describe each data byte to be written to the FLASH memory.

Baud Rate	Time for 1 Character	Time for 2 Characters
19200	520.8 μ s	1041.7.8 μ s
38400	260.4 μ s	520.8 μ s
57600	173.6 μ s	347.2 μ s
115200	86.8 μ s	173.6 μ s

From the AT45DB161D data sheet we find that a 'Buffer to Main Memory Page program without built-In Erase' will program the main FLASH array with the 512 or 528 bytes of data contained in one of the buffers in a worse case time of 6ms (t_p). At an RS232 baud rate of 115200 bits per second it requires at least 1024 ASCII characters to describe that much data and that communication will therefore last for at least 88ms. Therefore it is the bandwidth of the RS232 communications that limits the overall performance of this reference design even at the highest supported baud rate.

However at the actual time of programming the FLASH array the AT45DB161D is unable to receive any data and yet the PC will be continuing to transmit ASCII characters resulting in a temporary situation of up to 6ms where the RS232 communication is operating faster than the FLASH can be written and that would result in data being lost and the data written to FLASH memory would become corrupted. One way to prevent this is to perform RS232 flow control such as XON/XOFF but in practice this only slows down the average communication rate further. Therefore inserting a FIFO buffer of an adequate length to cope with this temporary situation between the RS232 interface and the FLASH enables the PC to continue transmitting continuously at 115200 baud because on average the FLASH memory will always be written faster than the PC transmits the UFP file characters to describe them.

Receiver FIFO Buffer Sizing

PicoBlaze programs the AT45DB161D by first writing 512 or 528 bytes of data to one of the buffers in the FLASH device and then issuing the 'Buffer to Main Memory Page program without built-In Erase' command which writes the buffer contents into the non-volatile array.

Most of the time PicoBlaze is reading ASCII characters from the UART receiver, converting pairs of these characters into true byte data values and then writing them to the buffer of the AT45DB161D device. Since the data conversion and writing to the buffer via SPI is significantly faster than the time taken for the PC to transmit one character over the RS232 link (even at 115200 baud) it is clear that PicoBlaze will spend most of its time waiting for the next ASCII character to be received and the receiver FIFO buffer will generally be empty.

However, during the actual FLASH array programming process, the AT45DB161D device is busy and can not accept any data being written to one of the buffers. Therefore PicoBlaze has to stop reading ASCII characters for up to 6ms (t_p) whilst the 'Buffer to Main Memory Page program without built-In Erase' process completes. This means that for a baud rate of 115200, the PC will continue to transmit approximately 70 characters which must be accommodated by the FIFO of the UART receiver so that they are not lost.

Once the AT45DB161D device has completed the program cycle, PicoBlaze can resume reading characters, converting them to data bytes and writing them to one of the buffers in the AT45DB161D and this will rapidly empty the FIFO again.

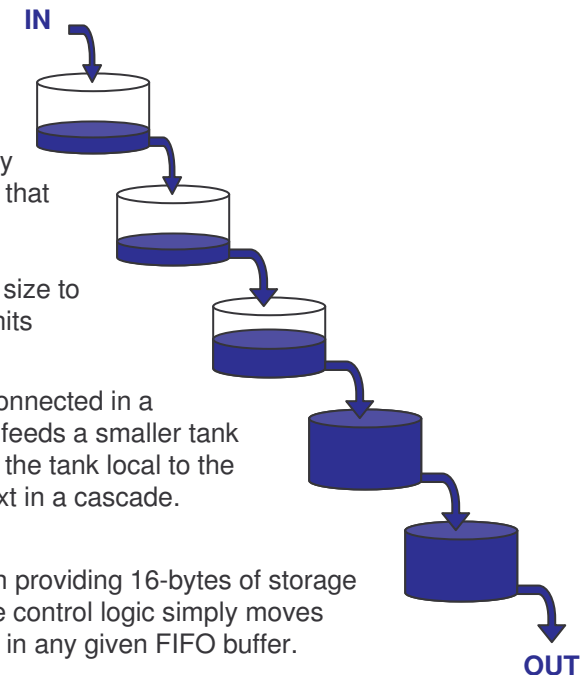
Bucket Brigade FIFO

The operation of a FIFO can be represented by a water tank. New water is added at the top, and the oldest water is drained from the bottom. The size of the tank (or bucket) must be large enough so that it does not overflow at times when more water is being added at the top than is being drained from the bottom. Obviously when a tank is empty nothing can be drained from the bottom. As soon as any water is added at the top then that water is available to be drained from the bottom.

It has become common practice for people to implement FIFO's as a single block of memory of an adequate size to prevent overflow. However, just as with water tanks, such a technique can often results in large (or heavy) units that are difficult to manage and connect up.

A 'Bucket Brigade' FIFO is constructed in a form that is similar to a series of smaller water tanks which are connected in a cascaded arrangement. This is similar to the way a house may contain a tank in the roof space which in turn feeds a smaller tank just above the toilet. There is a total amount of water in the system but it is distributed. The advantage is that the tank local to the toilet can react quickly (provide enough water fast) using a short but large diameter pipe each feeding the next in a cascade. Then the local tank is topped up from the larger tank in the roof using a smaller pipe.

In this reference design I have a total FIFO capacity of 80 characters by using five FIFO's (water tanks!) each providing 16-bytes of storage because this is the most natural size supported by the highly efficient SL16E form of distributed memory. The control logic simply moves any data along the cascade chain towards the final output just so long as there is spare capacity (i.e. not full) in any given FIFO buffer.



UART Rx with 80 Deep FIFO

