XC3S700AN
FLASH Programmer

A Reference Design for the Spartan™-3AN Starter Kit

Ken Chapman

Xilinx Ltd

Rev1 – 20th November 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 not-withstanding 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.

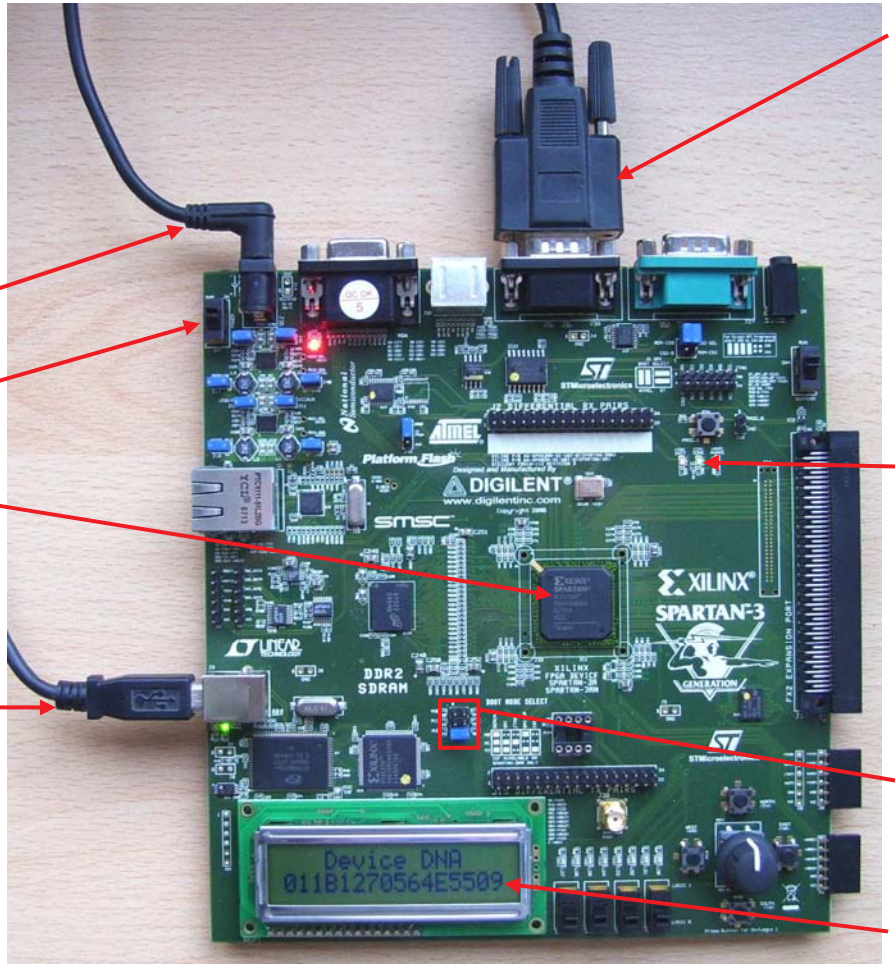
Any problems or items felt of value in the continued improvement of this reference design would be gratefully received by the author.

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

Design Overview

This design will enable you to experiment with the internal FLASH memory of the XC3S700AN device on your Spartan-3A Starter Kit. The 8M-bit (1M-byte) FLASH memory can be used to hold configuration images for the Spartan FPGA device as well provide general non-volatile storage for other applications that are then implemented by the device. 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-3AN 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 unique device DNA and 128-byte security register values.

The design is implemented using a single PicoBlaze processor and UART macros occupying less than 5% of the XC3S700AN device. It is hoped that the design may be of interest and a useful starting point for anyone interested in reading, writing and erasing the internal FLASH memory as part of their own applications.



RS232 Serial Cable connects to PC running HyperTerminal (or similar) and needs to be a male to female straight through cable (critically pin2-pin2, pin3-pin3 and pin5-pin5).

115200 Baud
8-bit
1 stop bit
No parity
Flow control: None

+5v supply
Don't forget to switch on!

Configuration 'DONE' LED

Spartan XC3S700AN

USB cable.
Cable plus devices on board provide essentially the same functionality as a 'Platform Cable USB' which is used in conjunction with iMPACT software to configure the Spartan-3AN with the PicoBlaze design. Note that iMPACT can program both the volatile FGPA or the non-volatile internal FLASH memory as you require.

IMPORTANT

Select Configuration Mode for internal FLASH memory.

Set to 'ISF'

LCD used in programming demonstration design

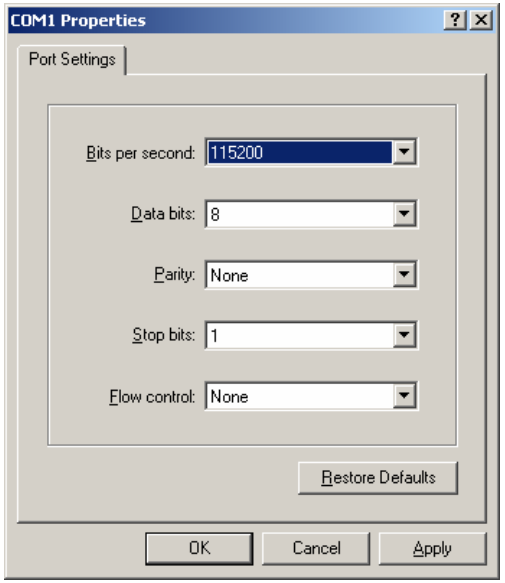
Serial Terminal Setup

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.



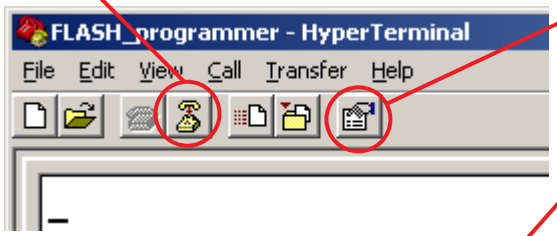
3) Set serial port settings.

Bits per second : 115200
Data bits: 8
Parity: None
Stop bits: 1
Flow control: NONE

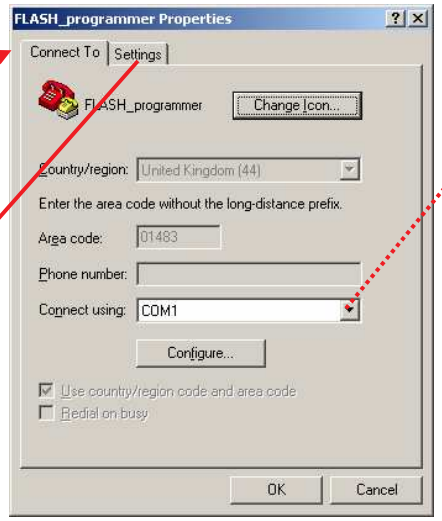
HyperTerminal Setup

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.

4 - Disconnect

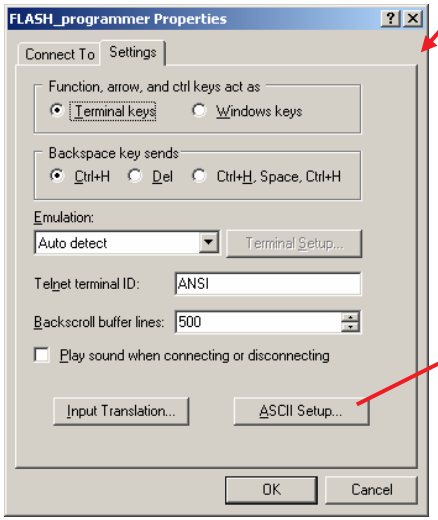


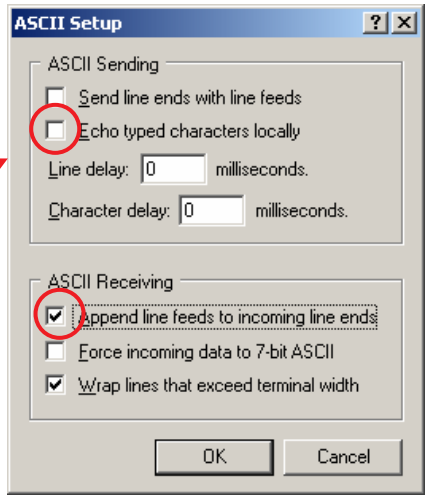
5 - Open the properties dialogue



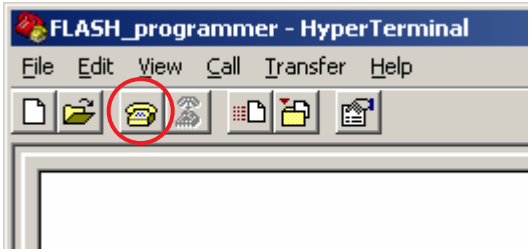
To select a different COM port and change settings (if not correct).

6 - Open Settings





8 - Connect



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.

Configure Spartan-3AN

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

1) Set up the board as shown on page 3 and open HyperTermnial set up as described on pages 4 & 5.

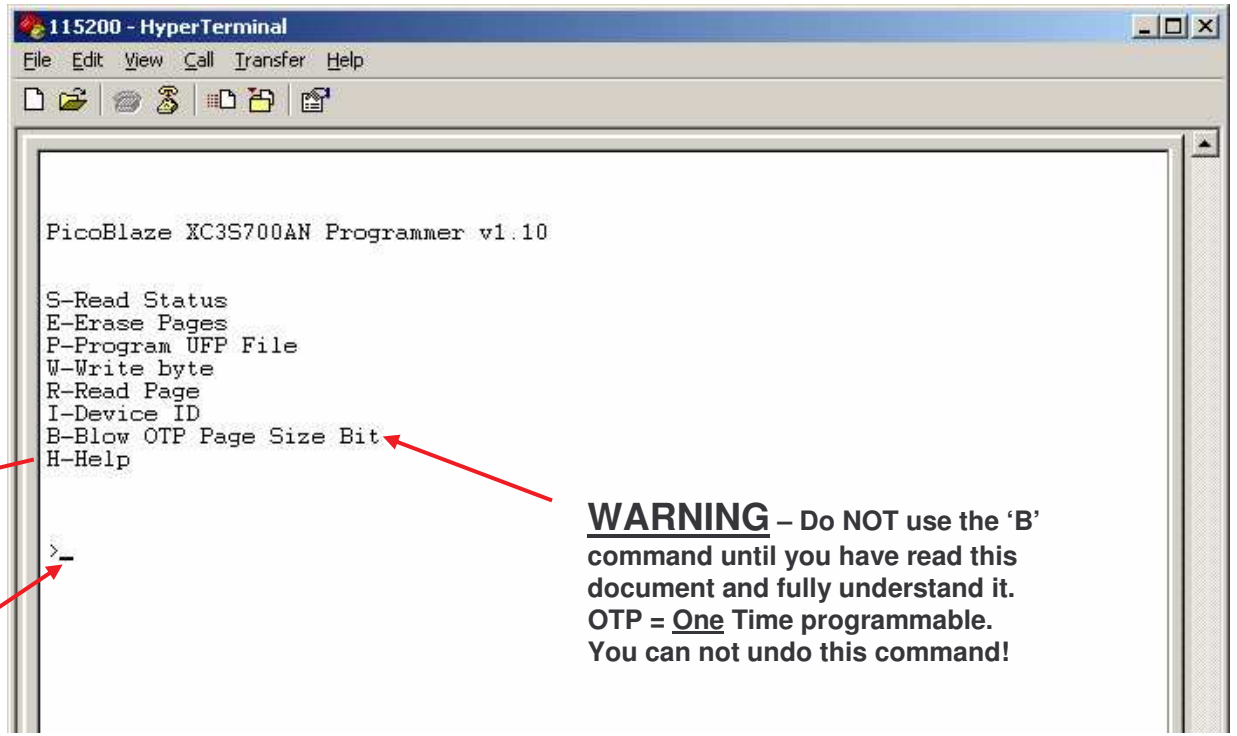
2) Unzip all the files provided into a working directory.

3) Double click on the file '**install_xc3s700an_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 volatile FPGA array of the Spartan XC3S700AN 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

WARNING – Do NOT use the 'B' command until you have read this document and fully understand it. OTP = One Time programmable. You can not undo this command!

Status ‘S’ and ID ‘I’ Commands

```
>s
Status = A4
Ready
Page Size = 264 bytes
>
```

The ‘S’ command will display the Status information from the internal 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 264 bytes is the default.

Bit 7 = RDY/BUSY ('1' = ready / '0' = busy)
Bit 6 = COMP
Bit 5 = '1'
Bit 4 = '0'
Bit 3 = ‘0'
Bit 2 = ‘1‘
Bit 1 = PROTECT
Bit 0 = PAGE SIZE (‘0' = 264 bytes / ‘1' = 256 bytes)

```
>i
ID = 1F 25 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 FF FF FF FF
 06 0C 08 08 23 2D 1F 25 00 00 01 29 FF FF 01 FF
 45 54 36 30 37 30 30 30 11 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

Device DNA = 011B1270564E5509
>
```

The ‘I’ command initially reads the FLASH memory and displays Manufacturer Code (1F hex) and the density Device Code which is 25 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 unique factory programmed value which can be used as a product serial number, provide registration and product tracking codes or as a seed in design authentication.

Finally the unique factory programmed device DNA value contained in the FPGA is also displayed. This can also be used as a product serial number, provide registration and product tracking codes or as a seed in design authentication.

Hint – Combine the unique FLASH security value with the unique device DNA to improve authentication security algorithms.

Pages & Address Formats

Before using any of the commands that read, write and erase the FLASH memory it is important to recognise how the FLASH memory is organised and appreciate the format required for the 24-bit address. Please take the time to study these two page because it is critical to using the commands correctly as well as understanding how to work with FLASH memory in your own designs.

Pages and Bytes

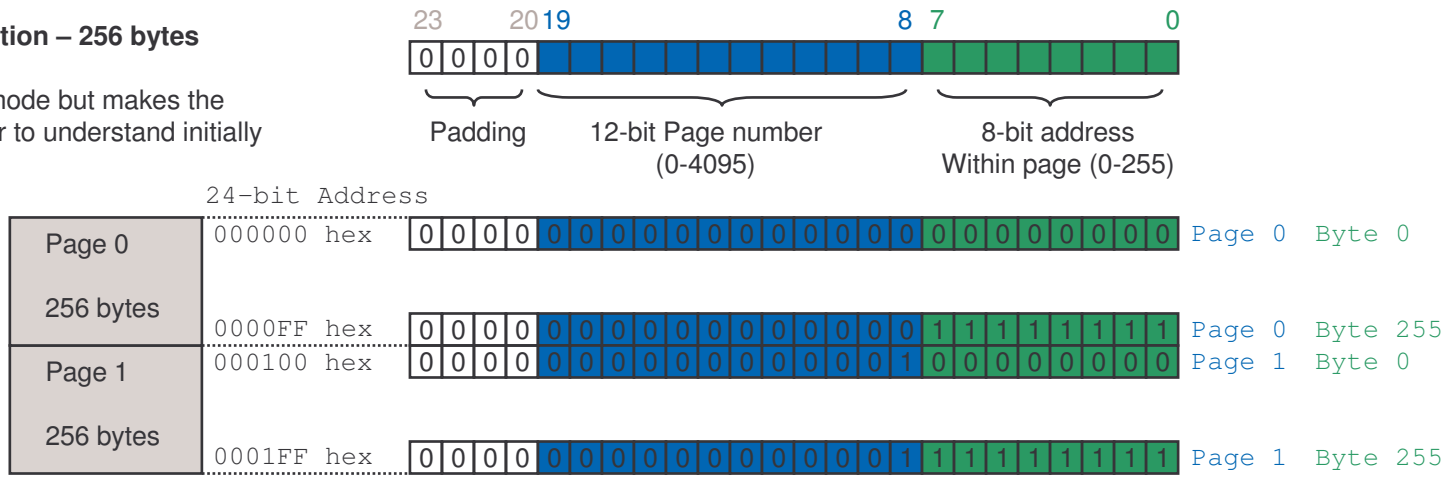
Although internally to the XC3S700AN device package the communication with the FLASH memory is a serial SPI bus, all transfers are implemented as bytes (8-bits). Therefore, it is possible to consider the memory as byte oriented except at the signally level. The FLASH is internally organised and presented as having 4096 'pages' which each consist of a number of adjacent bytes. The interesting part is that the default page size is 264 bytes which is not a power of 2 and doesn't feel particularly natural. Indeed the total size of the memory is 4096 × 264 = 1,081,344 bytes or 8,650,752 bits rather than 8,388,608 bits which is the normal size for an 8-Mbit memory (8×1024×1024). So it is also possible to program the memory such that the page size is a more natural 256 bytes but in doing so you really do lose 32,768 bytes forever.

To define the page size the FLASH memory contains a one time programmable (OTP) register that can be programmed to switch from the default size of 264 bytes to 256 bytes per page. Programming of this register is also supported by this reference design (see 'B Command') but remember that there is no going back if you use it! This reference design has the ability to perform all operations for both sizes of page but your own designs could probably be simplified.

To make sense of this, let's first look at the format of the 24-bit address when using what is called the "Power of 2" page size. In this mode, each page is reduced to 256 bytes and therefore any byte within a page can be identified with an 8-bit address. There are then 4096 pages which can be identified using a a further 12 address bits. The significant observation is that the end of one page (byte address FF) is adjacent to the start of the next page (byte address 00) and this leads to the natural linear address format we all tend to be familiar with. This is shown below and simply indicates which of the 24-bits are unused, which are used to identify a page and which are used to identify a byte within a page.

"Power of 2" Page Size Option – 256 bytes

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

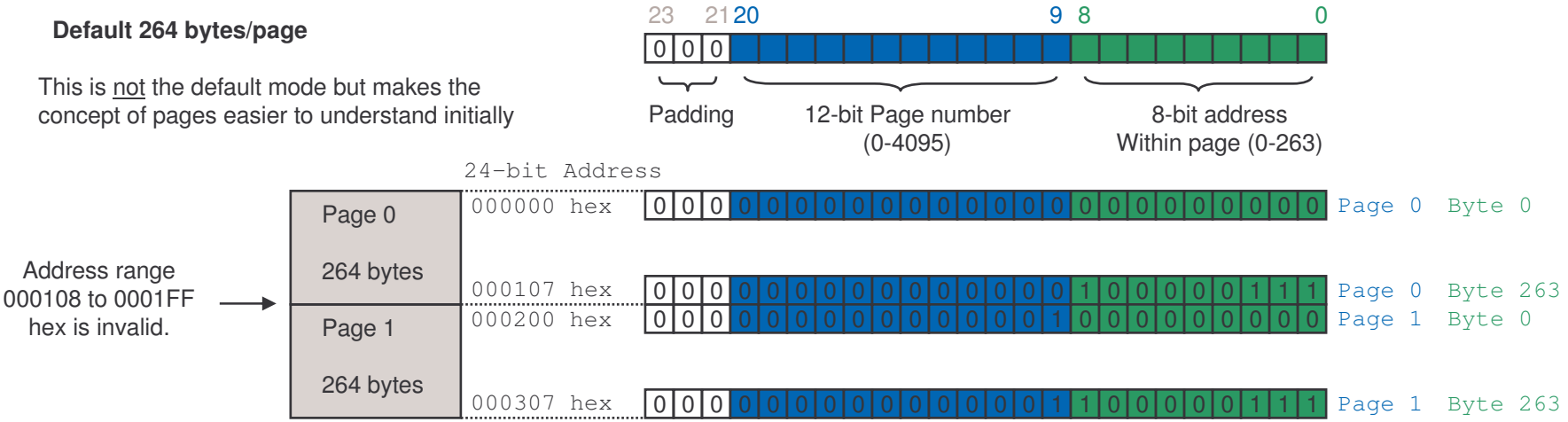


Pages & Address Formats

Returning to the default page size of 264 bytes, the issue is that it now requires a 9-bit address to identify a byte within a page. The range being 0 to 263 (000 to 107 hex). Although the end of one page is adjacent to the start of the next page in terms of the physical memory array, there appears to be a gap of 248 bytes in the address map. Please take the time to study the diagram below to appreciate this situation and its impact on the 24-bit address format.

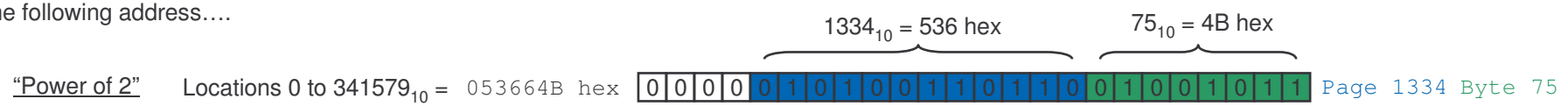
Default 264 bytes/page

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

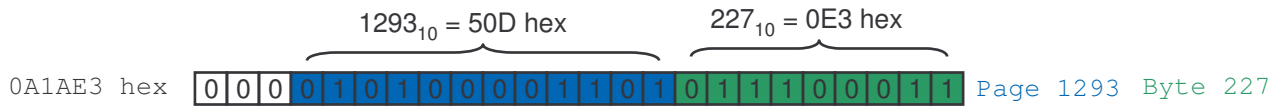


This reference design has been presented in a way that asks you to specify the page number and byte address (location within a page). PicoBlaze then formats the 24-bit address based on the page size. The 24-bit address is then displayed as a 6-digit hexadecimal value where appropriate. It is hoped that this feature of the reference design is a useful aid to understanding the address format as well as providing useful routines for your won designs.

A single configuration image for the XC3S700AN 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….



Default 264 bytes/page 341579_{10} / 264 = 1293.86 ⇨ Pages 0 to 1292 (1293 pages) completely full and then…..
Page 1293 is used for (341579 - 1293×264) = 227 bytes



Erase Pages Command ‘E’

The ‘E’ command will erase one or more 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 = 000
Last Page = 50d

Confirm (Y/n) Y
Erase in Progress

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

 OK
```

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

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

To indicate progress, a dot will be displayed each time a page is page erased. Erase can take up to 32ms per page.

This example shows the erase of pages 000 to 50D inclusive. This is consistent with erasing the primary configuration image for the XC3S700AN device when using the default page size.

A ‘page’ is the smallest amount of memory that can be erased but having 4096 pages this provides a great deal of flexibility in what is after all a FLASH memory. However, since each page may take up to 32ms to erase it may take up to 133 seconds to erase the whole device (but typically 53 seconds).

The FLASH memory is further organised into ‘blocks’ and ‘sectors’. A block is formed of 8 pages and a sector is formed of 32 blocks (256 pages). There are separate 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 16 seconds (typical) up to 39 seconds (worst case) using these course granularity commands.

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 FLASH memory using a particular sequence of commands that enables the FLASH device to emulate an EEPROM. This clearly has value when working with operational editing data which need to be non-volatile in a system.

```
>w
Page = 50d
byte address = 0e3

Data = 42
Full address = 0A1AE3


>
```

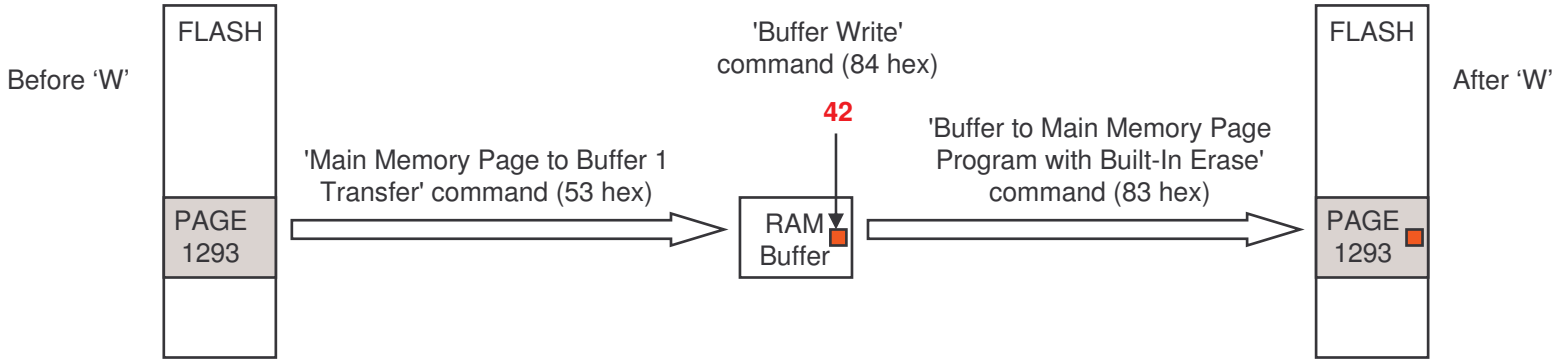
After entering the ‘W’ command you will be prompted to enter a hexadecimal page number in the range 000 to FFF and then the byte address within that page (range 000 to 107 for default page size or 00 to FF for power of 2 page size).

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 9 to see how the 24-bit address is formed in this example.

How does EEPROM emulation work?

The FLASH memory device contains two RAM buffers in addition to the FLASH array. Each buffer is sufficient to hold one complete page (264 bytes) of information. So to implement the ‘W’ command above, PicoBlaze first commands the FLASH memory 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 264 (default) or 256 bytes contained in a page of the FLASH memory in order to verify contents.

```
>r
Page = 50d

0A1A00  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1A10  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1A20  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1A30  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1A40  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1A50  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1A60  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1A70  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1A80  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1A90  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1AA0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1AB0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1AC0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1AD0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1AE0  FF FF FF 42 FF FF FF FF FF FF FF FF FF FF FF FF
0A1AF0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0A1B00  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 50D to look for the data written by the ‘W’ command on the previous page

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.

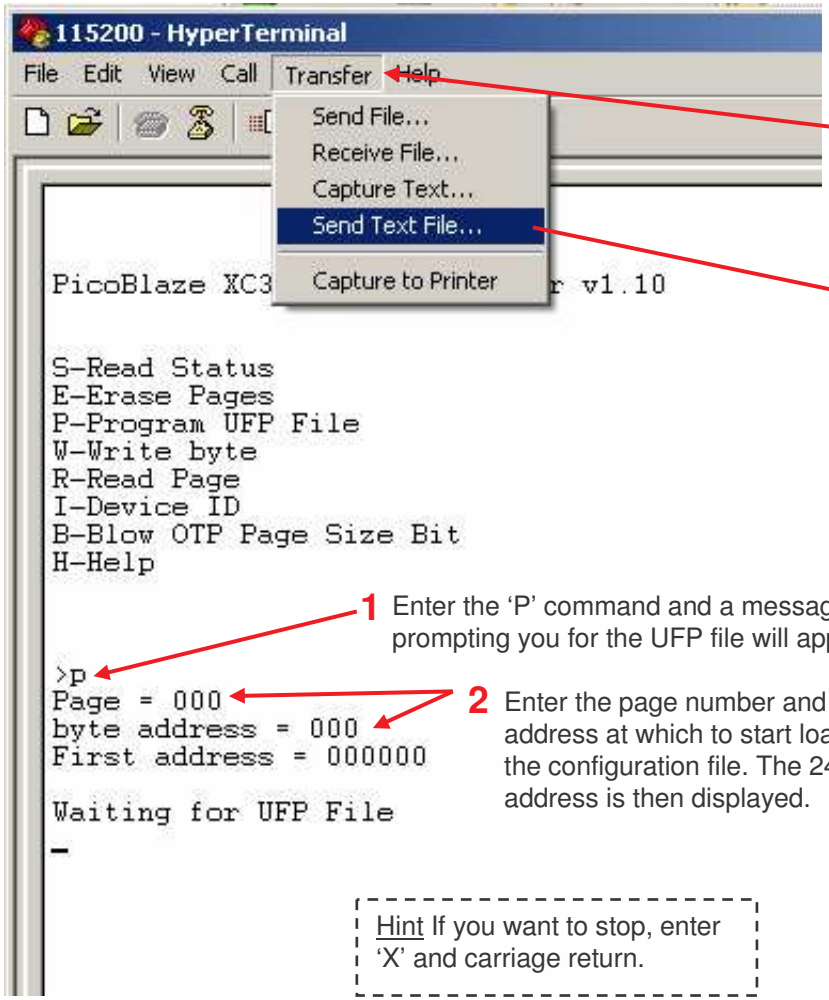
The last line is only 8 bytes and will only be displayed when the default page size of 264 bytes is being used.

Read display shows how address 0A1AE3 (page 50D and byte address 0E3) was modified to 42 hex by the ‘W’ command shown on the previous page.

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

Program UFP File Command ‘P’

The ‘P’ command enables you to write an entire configuration image for the Spartan-3AN 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 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.



1 Enter the ‘P’ command and a message prompting you for the UFP file will appear.

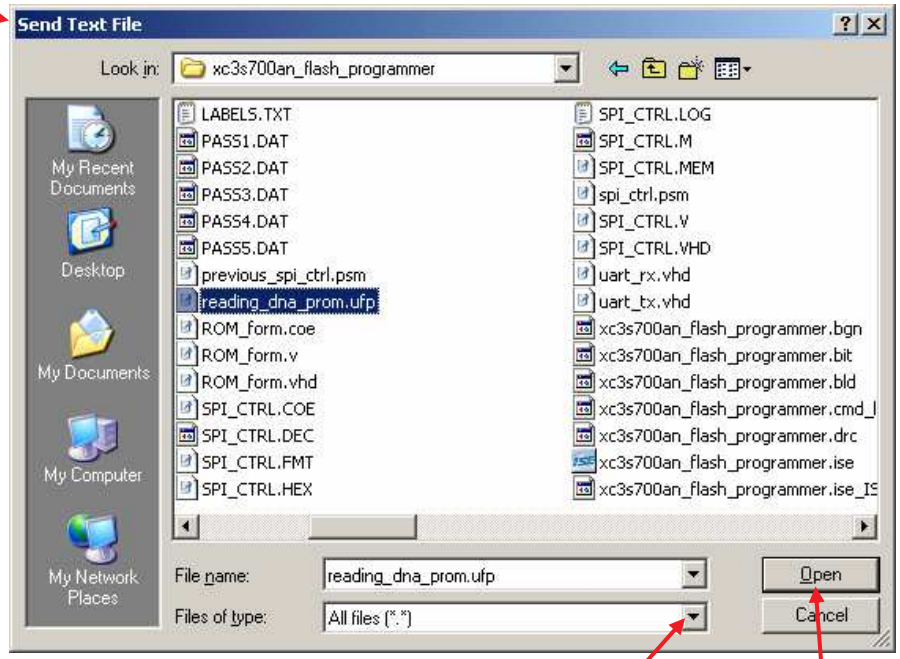
2 Enter the page number and byte address at which to start loading the configuration file. The 24-bit address is then displayed.

Hint If you want to stop, enter ‘X’ and carriage return.

IMPORTANT – Ensure you have erased the pages to be programmed (000 to 50D 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).

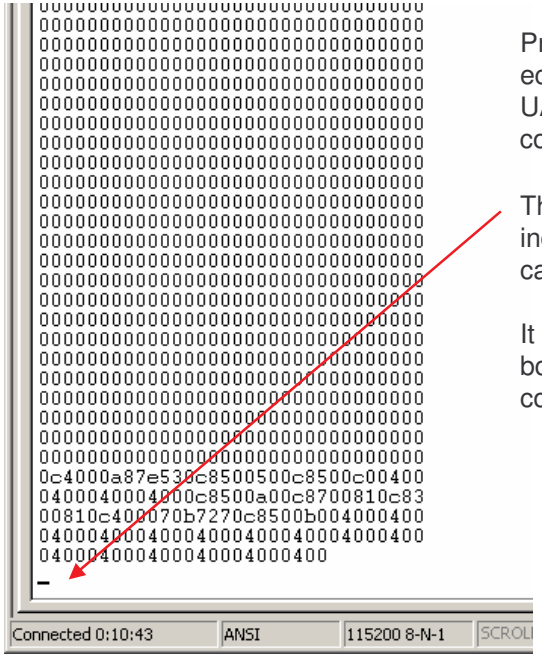
4 Navigate to the appropriate directory and select the desired UFP file which in this case is ‘reading_dna_prom.ufp’.



Hint - You will need to change ‘Files of type’ to ‘All files (*.*)’ to see the UFP files listed.

5 ‘Open’

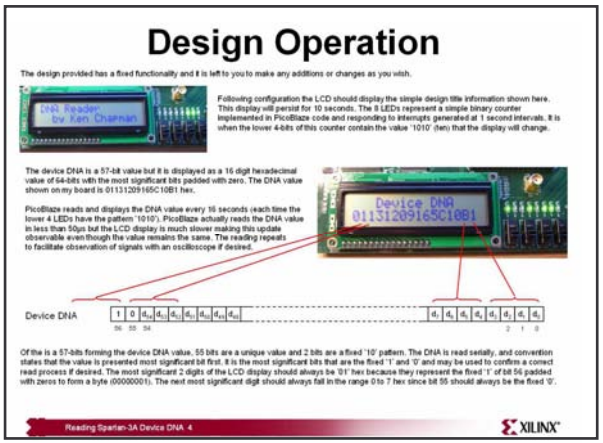
‘P’ Command continued



Programming of the FLASH memory will start immediately and PicoBlaze will indicate progress by echoing the UFP file back to the PC display. You may also see the LD7 LED on the board blinking as the UART receives data form the PC (see FIFO design later). Programming will typically take **68 seconds** to complete which is mainly due to the UFP file size and 115200 baud rate of the communication.

The UFP file is a pure ASCII hexadecimal representation of the configuration data and there is nothing to indicate to PicoBlaze when the file transmission has completed. Therefore to finish enter ‘X’ followed by carriage return (in fact any non-hexadecimal character will also stop it).

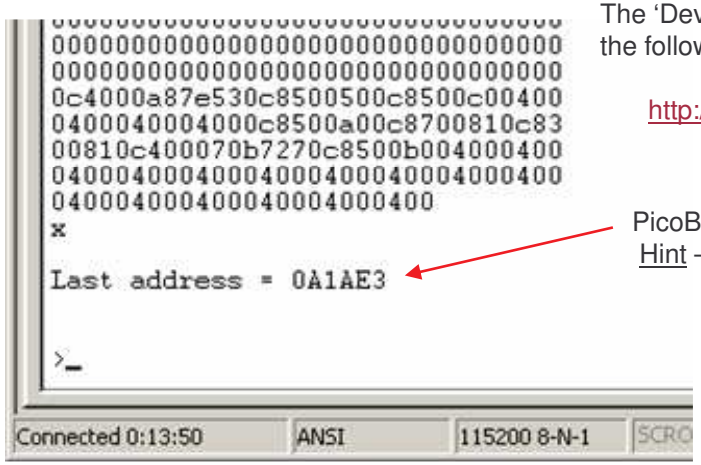
It should now be possible to press the PROG button on the board to reconfigure the Spartan device directly from the new configuration image stored in the internal FLASH memory.



If you used the supplied ‘reading_dna_prom.ufp’ file, then your board should now be displaying the unique DNA value of your Spartan-3AN on your board. The value should match that seen when using the ‘I’ command (see page 7).

The ‘Device DNA Reader’ design is also available from the following Xilinx web site.

http://www.xilinx.com/products/boards/s3astarter/reference_designs.htm



PicoBlaze will display the last 24-bit address programmed
Hint – Look back at page 9 to see how this example works.

Obviously once you have reconfigured the Spartan-3AN using the image stored in FLASH memory the programmer design will have to be restored if you want to use it again.
‘install_xc3s700an_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

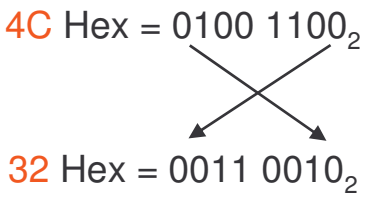
```
ffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffff
55990c8500e004008c869077cc843cf0
8c8500938c82f4008c43404401c90c87
fff30c8300818c8110814c8000f84c83
00a04c8700204c8500704c8600004c81
0000cc82184fcc46000000000c440000
00000c8500800a060040594300000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
```

Page 000 read back from FLASH (264 bytes)

```
>r
Page = 000

000000  FF 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 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 00
000090  00 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 00
0000B0  00 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 00
0000D0  00 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 00
0000F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000100  00 00 00 00 00 00 00 00
```

A critical issue with configuration is that the Spartan-3AN must be able to read the data in the correct order. When the SPI FLASH memory is read it provides each byte most significant bit (MSB) first. As described in the UFP file, the data needs to be read least significant bit (LSB) first. Therefore in this reference design, PicoBlaze is automatically swapping the order of the bits in each byte before writing them to the FLASH memory.

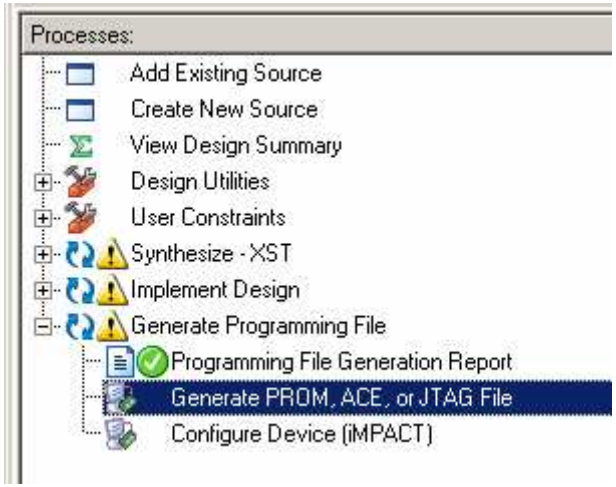


This example shows how the UFP byte 4C hex was written to FLASH memory as value 32 hex.

Preparing a UFP file

This reference design has been provided so that the contents of a UFP file can be programmed into the internal FLASH memory of the XC3S700AN. The following images indicate how a Spartan-3AN configuration can be made into a UFP file using the ISE tools but this is not intended to replace the existing documentation for PROM generation.

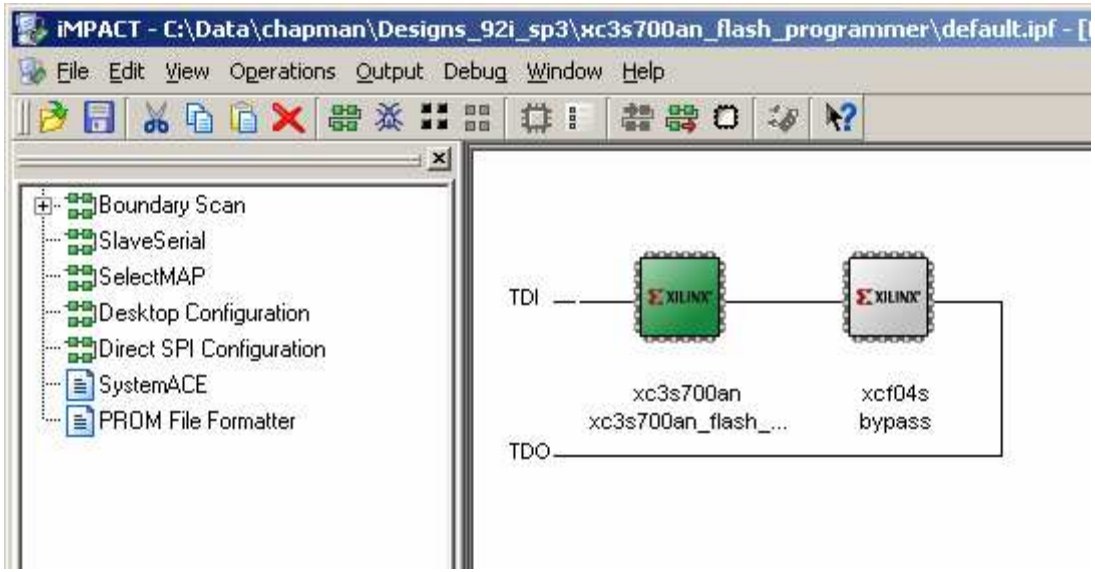
1) Select 'Generate PROM' in Project Manger



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.

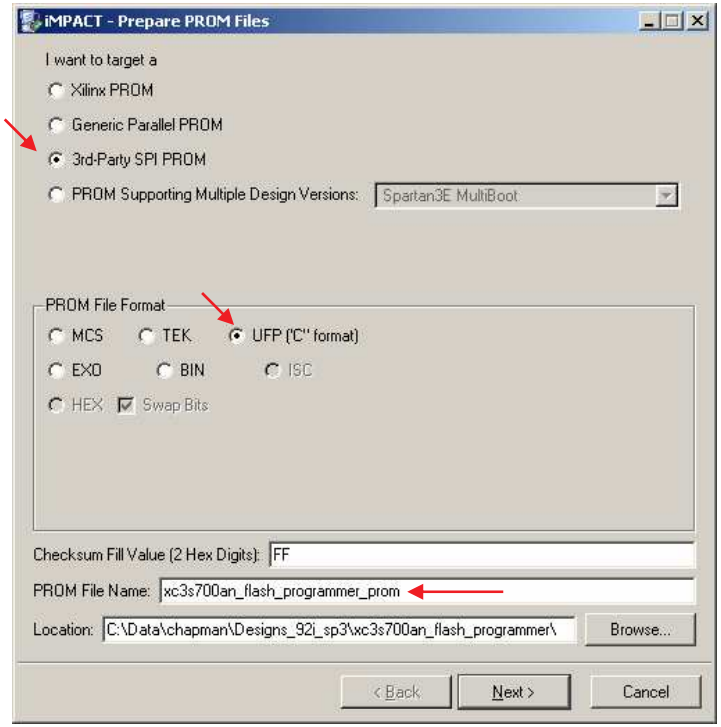
Hint – UFP files do not contain any address information allowing you to experiment with storing configuration images at any locations in the SPI FLASH. Spartan-3A supports multi-boot from SPI FLASH and the AT45DB161D is large enough to hold 6 configuration images for the XC3S700A device provided on the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

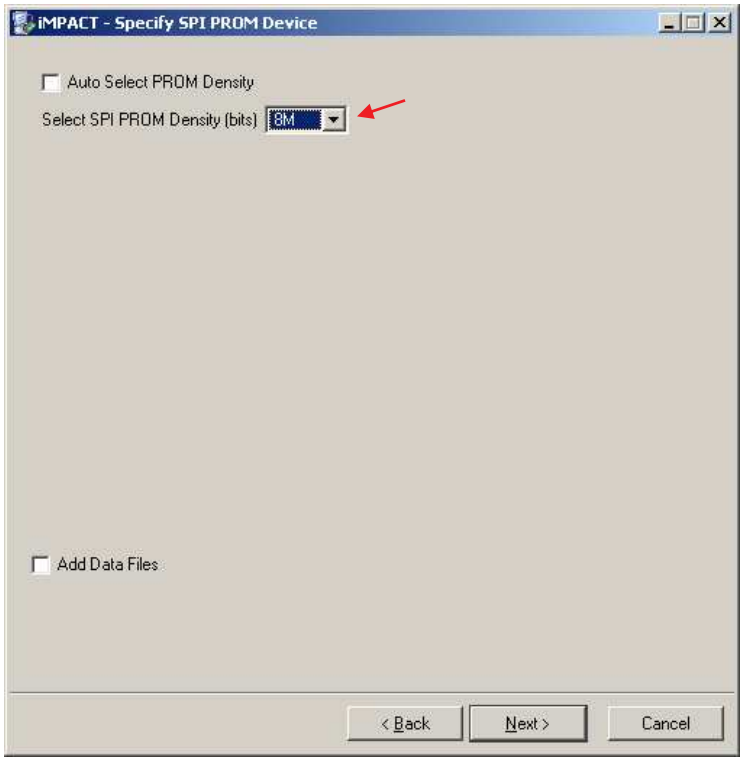


Preparing a UFP file

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 XC3S700AN contains an 8 M-bit FLASH memory.



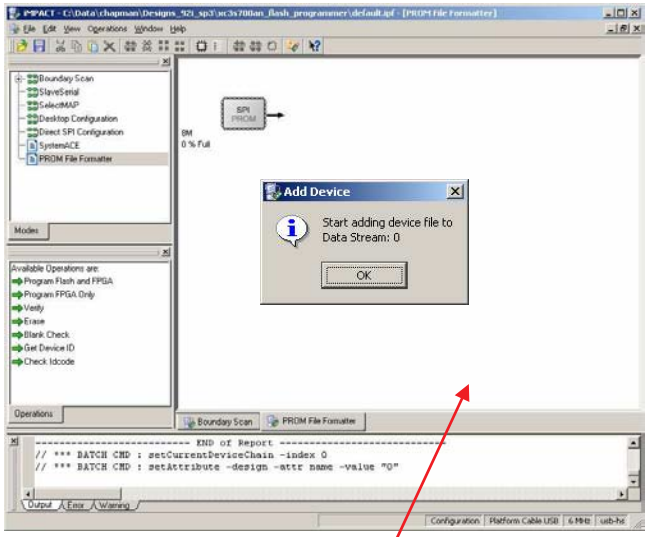
5) Summary Page

** Note for future reference

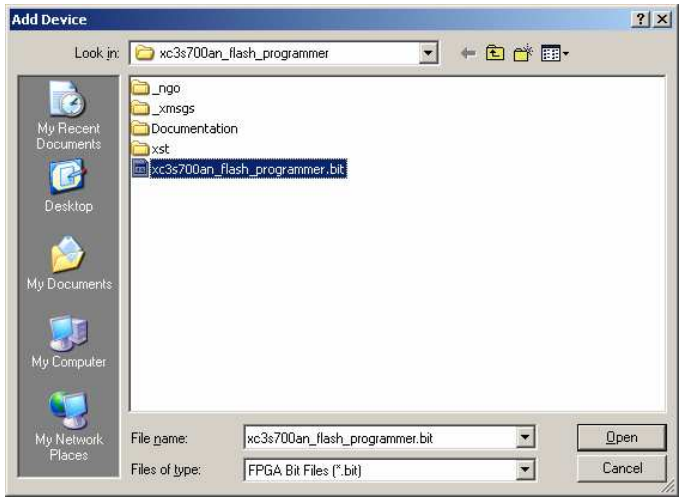
It is possible that a future release of ISE tools (after 9.2i) may write UFP files for SPI FLASH in a way that already has the bits of the bytes swapped (see page 15). If that does occur, simply use 'Generic Parallel PROM' to generate your UFP file or remove the bit swapping code from the PicoBlaze program.

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.

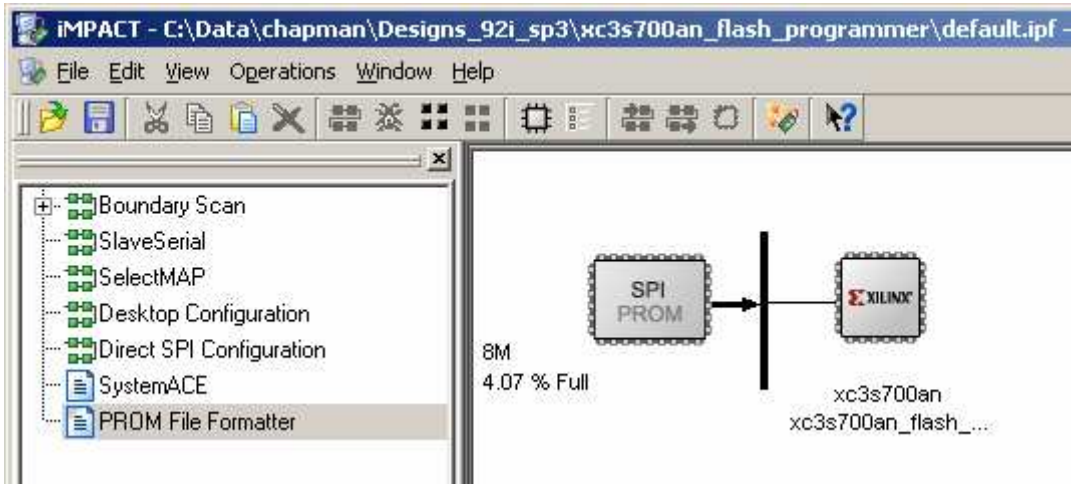


7) Navigate to the required configuration BIT file, select the file then click ‘Open’.



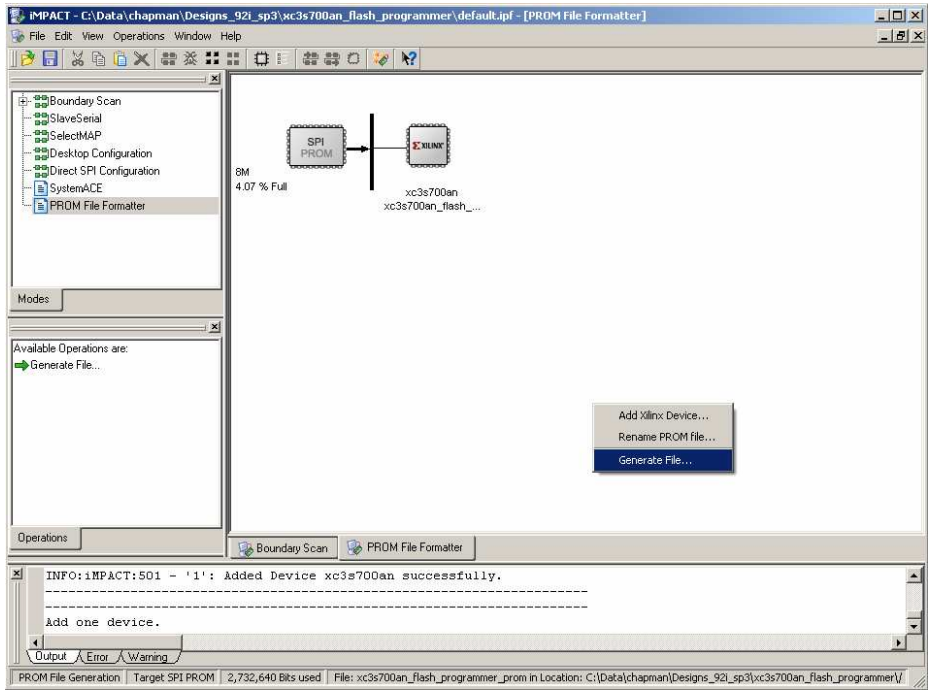
Hint: If the ‘Add Device’ box does not appear, then right click in the white space and select ‘Add Xilinx Device…’

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

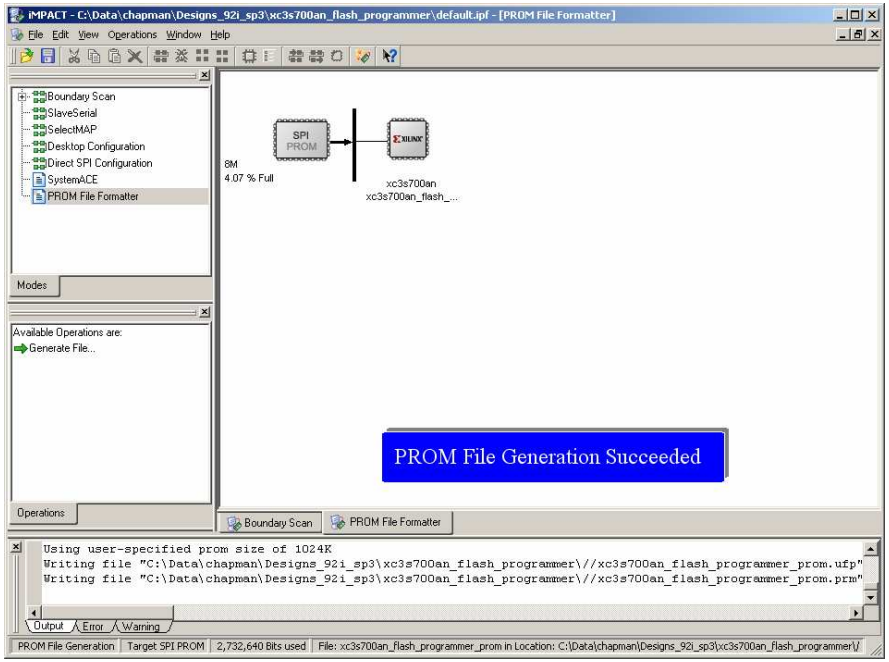


Preparing a UFP file

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.



Hint – The ‘xc3s700an_flash_programmer_prom.ufp’ generated in this example is also provided with the reference design.

Blow OTP Page Size Command ‘B’

This command programs the One Time Programmable (OTP) register which changes the page size from the default of 264 bytes to 256 bytes. Please see pages 8 and 9 of this document to understand the difference.

```
PicoBlaze XC3S700AN Programmer v1.10

S-Read Status
E-Erase Pages
P-Program UFP File
W-Write byte
R-Read Page
I-Device ID
B-Blow OTP Page Size Bit
H-Help

>b
Confirm (Y/n) _
```

Note that using this command will eliminate 8 bytes from the end of every page. Note only does this mean the loss of 32,768 bytes forever, but it also means that any data stored in those locations is also lost. For example, if the FLASH has been programmed with an FPGA configuration image under the default page size then that image will become invalid requiring the memory to be erased and reprogrammed before it will configure the device successfully again. As shown on page 9 of this document, configuration images will then occupy more pages due to the missing bytes.

WARNING – Only confirm with ‘Y’ if you are absolutely certain that you always want to use the smaller page size of 256 bytes in the future. There is absolutely no way to return to the default page size of 264 bytes after you have confirmed this command.

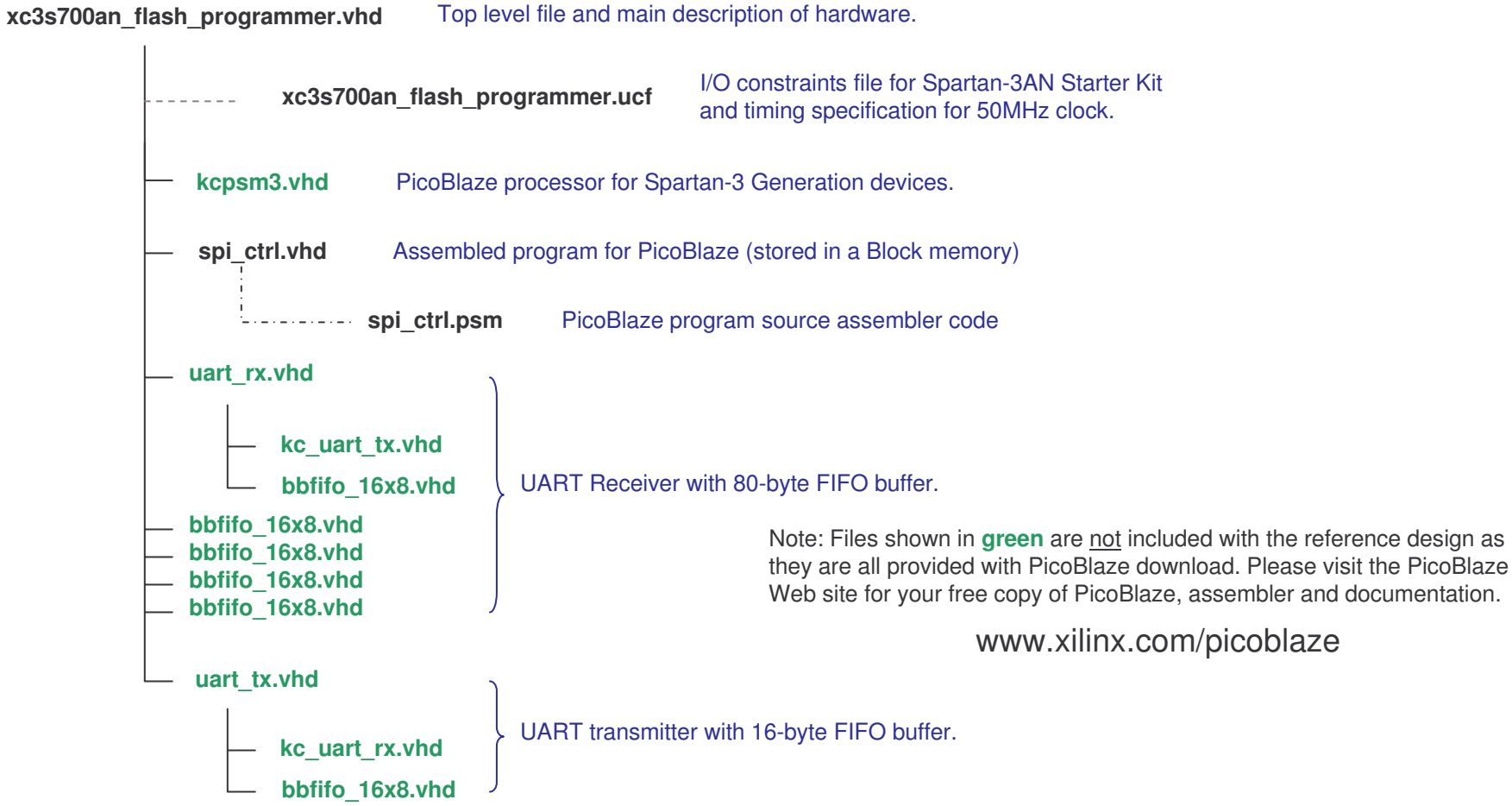
Power Cycle required - For the new page size to become active, the power to the XC3S700AN device must be cycled. So you will need turn the Starter Kit off and back on again after using this command.

Hint – Use the ‘S’ command to confirm the page size (see page 7).

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 additional descriptions. It is highly recommended that you have a copy of User Guide UG333 'Spartan-3AN In-System Flash User Guide' as reference. You may also find it useful to have a copy of the Atmel data sheet for the AT45DB081D device as this 8-Mbit SPI FLASH memory which is directly compatible with the internal FLASH of the XC3S700AN (see Table 6-9 of UG333).

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



Note: Files shown in **green** are not included with the reference design as they are all provided with PicoBlaze download. Please visit the PicoBlaze Web site for your free copy of PicoBlaze, assembler and documentation.

www.xilinx.com/picoblaze

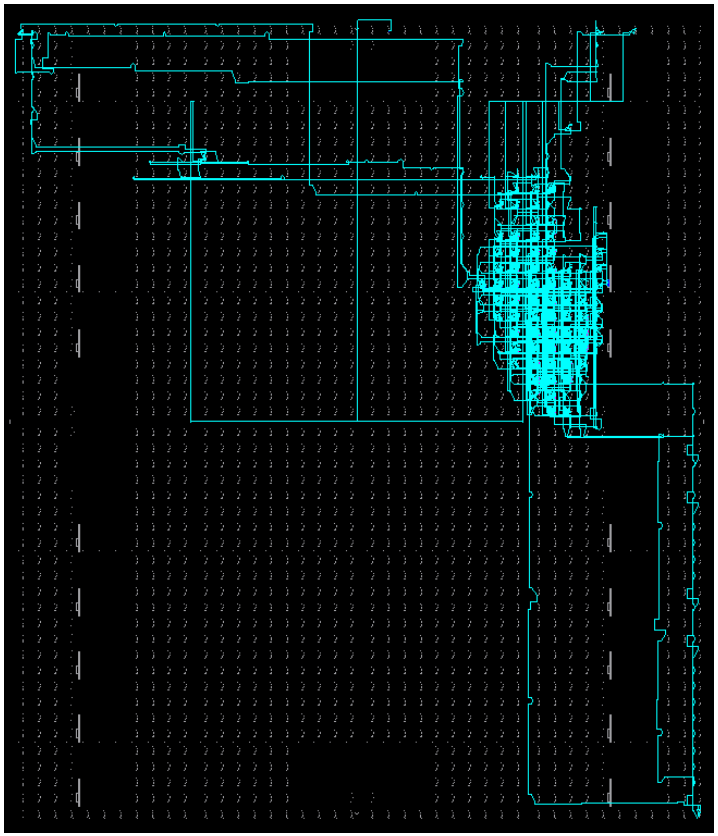
PicoBlaze Design Size

This reference design occupies less than 5% of the XC3S700AN 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:     187 out of    5,888     3%
Number of RAMB16BWEs:            1 out of       20     5%
```

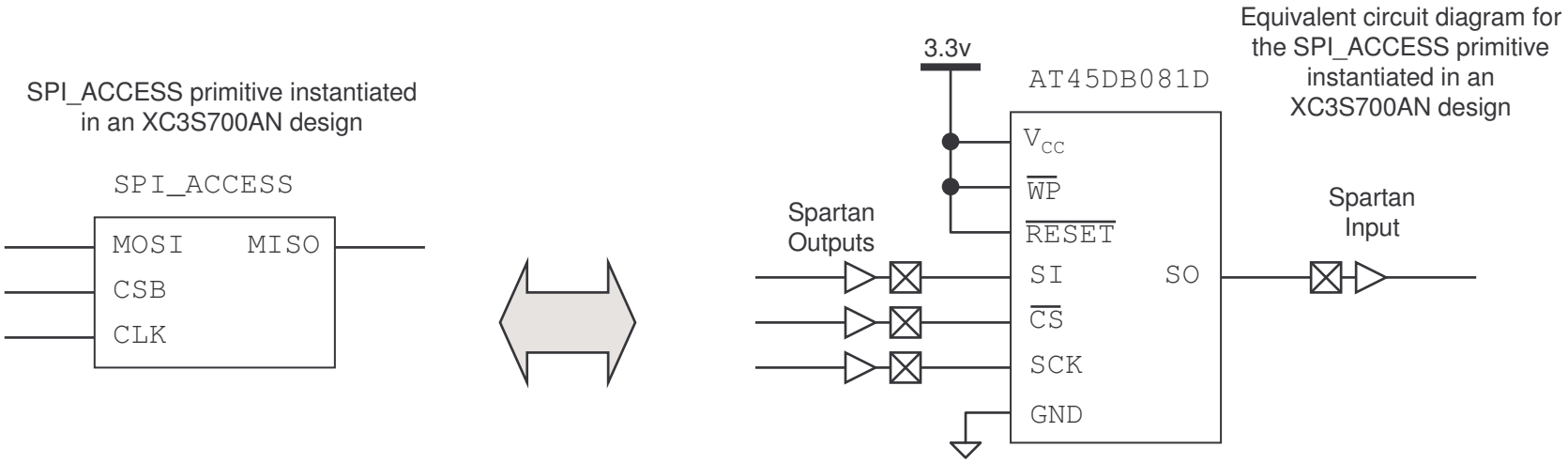
FPGA Editor view



Stacked FLASH Communication

The key to any design implemented in a Spartan-3AN device that needs to interact with internal stacked FLASH memory is how to establish communication. Although the majority of the detail is contained in the SPI signaling and the command protocol of the FLASH memory, it is of course vital to make an initial physical connection in the hardware design. This is achieved by using the 'SPI_ACCESS' primitive.

Inserting and connecting the 'SPI_ACCESS' primitive is the functional equivalent to defining 3 output pins and one input pin and connecting those to an external Atmel AT45DB081D FLASH device.

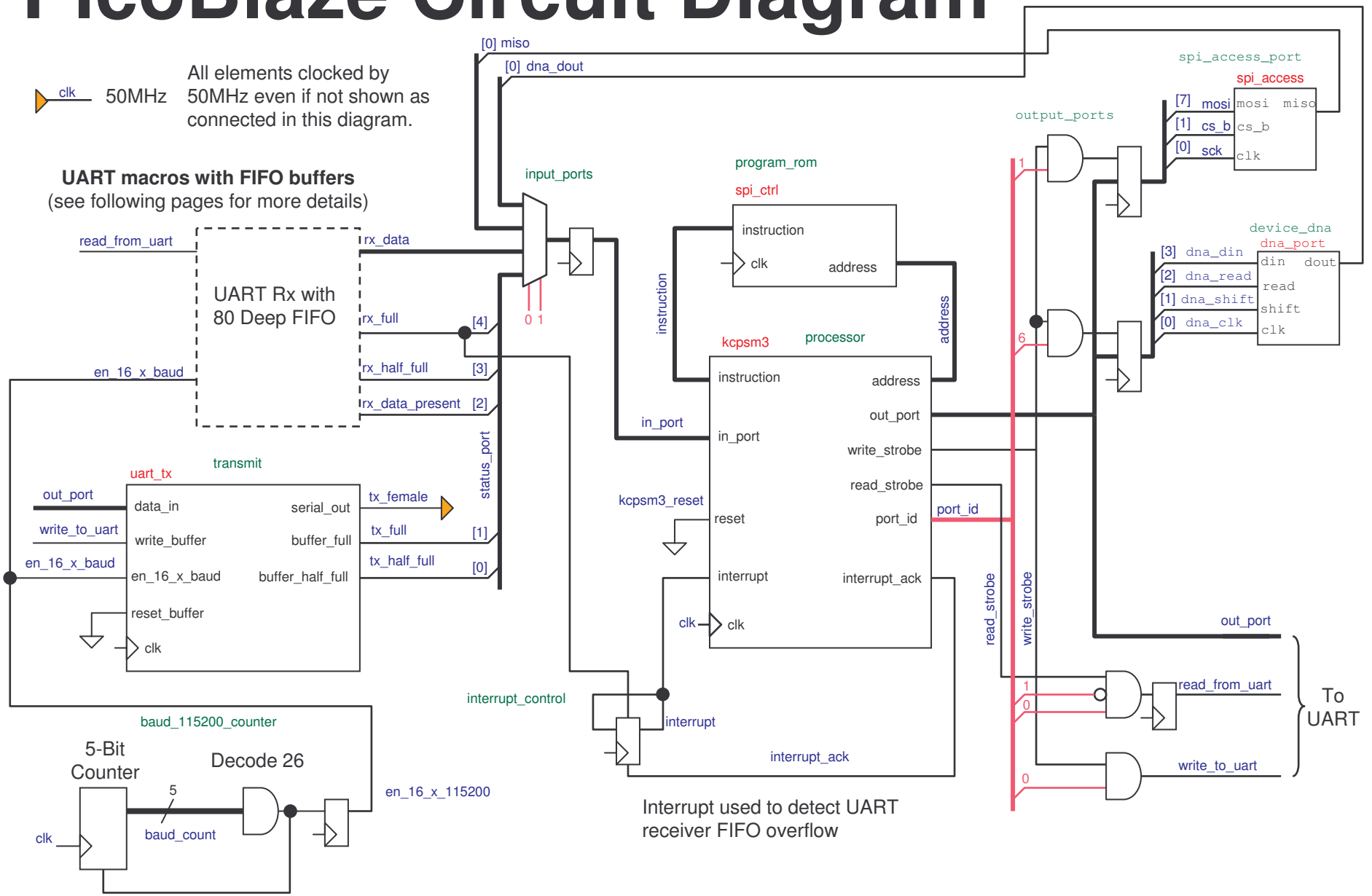


Since the SPI FLASH is bonded to the Spartan FPGA internally to the device package, the FLASH memory is the only SLAVE connected to this SPI 'bus' and the FPGA becomes the SPI bus master. In this particular reference design, a PicoBlaze processor within the FPGA implements the SPI master and is used to generate the SPI signals as well as define the content of the bus transactions using the protocol of the FLASH memory.

The VHDL hardware description uses the following signal names to define the bus. The names used are a helpful way to determine the signal direction.

'sck' - SPI clock from the master (PicoBlaze in the FPGA) to the slave (FLASH).
'cs_b' - Active Low device select generated by the master (PicoBlaze in the FPGA) to enable the slave (FLASH).
'mosi' - (Master Out, Slave In) Serial Data transmitted from the master (PicoBlaze in the FPGA) to the slave (FLASH).
'miso' - (Master In, Slave Out) Serial Data received by the master (PicoBlaze in the FPGA) from the slave (FLASH).

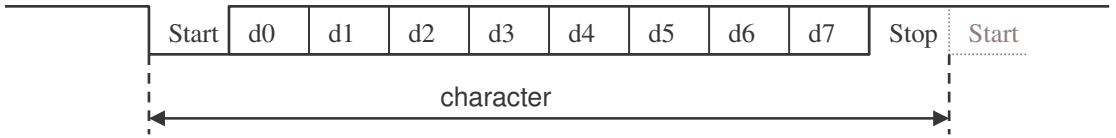
PicoBlaze Circuit Diagram



Baud Rate vs FLASH Write Rate

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.



Baud Rate	Time for 1 Character	Time for 2 Characters
115200	86.8µs	173.6µs

Data to be stored in the FLASH memory is sent transmitted from the PC in the form of a UFP 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 and that communication takes the times shown in this table.

From the AT45DB161D FLASH memory data sheet we find that a 'Buffer to Main Memory Page program without built-In Erase' will program the main FLASH array with the 264 or 256 bytes of data contained in one of the buffers. This programming cycle will take a worse case time of 4ms (t_P). During this time the PicoBlaze processor will wait for the memory to be ready and yet the PC will continue to transmit the UFP file. During that worse case 4ms the PC can transmit 47 ASCII characters and these must be buffered as part of the UART receiver. The standard UART receiver provided with PicoBlaze has a built-in 16-byte (character) buffer and therefore this reference required an extension to be made to the buffer (see next page).

During execution of the 'P' command, PicoBlaze reads the UFP file transmitted from the PC and segments the programming into pages in order to program the FLASH array. The smallest page size is 256 bytes (i.e. not the default size), and to program each page PicoBlaze needs to read 512 ASCII hexadecimal characters and 16 carriage return characters due to the format of the UFP file. The communication of those 528 ASCII characters will take 45.83ms which confirms that it is the RS232 communication rate which is the overall limiting factor in this reference design even though a FIFO buffer is required to cope with the actual programming cycles of the FLASH memory.

Hint – This worse case timing analysis for the XC3S700AN device has shown that the FIFO buffer needs to be at least 47 characters deep. If you should want to migrate this reference design to the XC3S1400AN then you would discover that the worse case page programming time (t_P) increases from 4ms to 6ms and that this would therefore require a FIFO able to buffer 69 characters. For this reason the design has been provided with a FIFO buffer of 8 characters in order that the design may be used with any Spartan-3AN devices with the minimum of changes.

Receiver FIFO Buffer Sizing

PicoBlaze programs the FLASH memory by first writing 264 or 256 bytes of data to one of the buffers in the stacked 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.

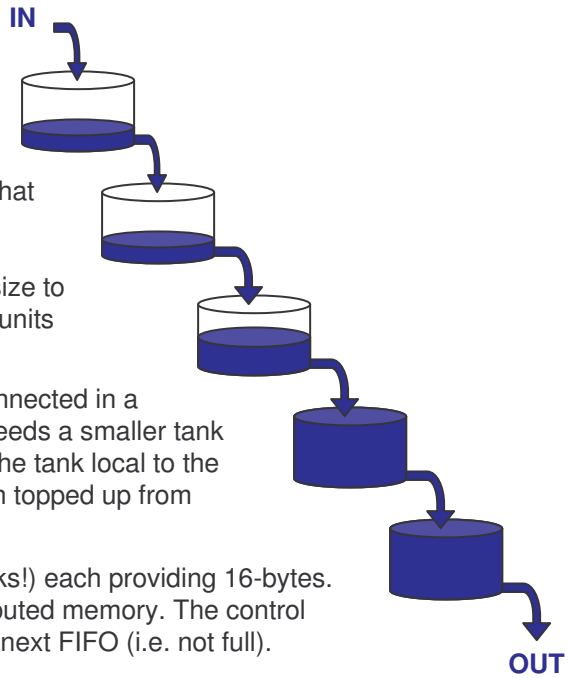
Most of the time PicoBlaze is reading ASCII characters from the UART receiver, converting pairs of these characters into true byte data values and then writing them to the buffer of the FLASH memory. Since the data conversion and writing to the buffer via SPI is significantly faster than the time taken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

However, during the actual FLASH array programming process, the FLASH memory is 'busy' and can not accept any data being written to its buffers. Therefore PicoBlaze has to stop reading ASCII characters for up to 4ms (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 47 characters which must be accommodated by the FIFO of the UART receiver so that they are not lost.

Once the FLASH memory 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 FLASH memory 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.

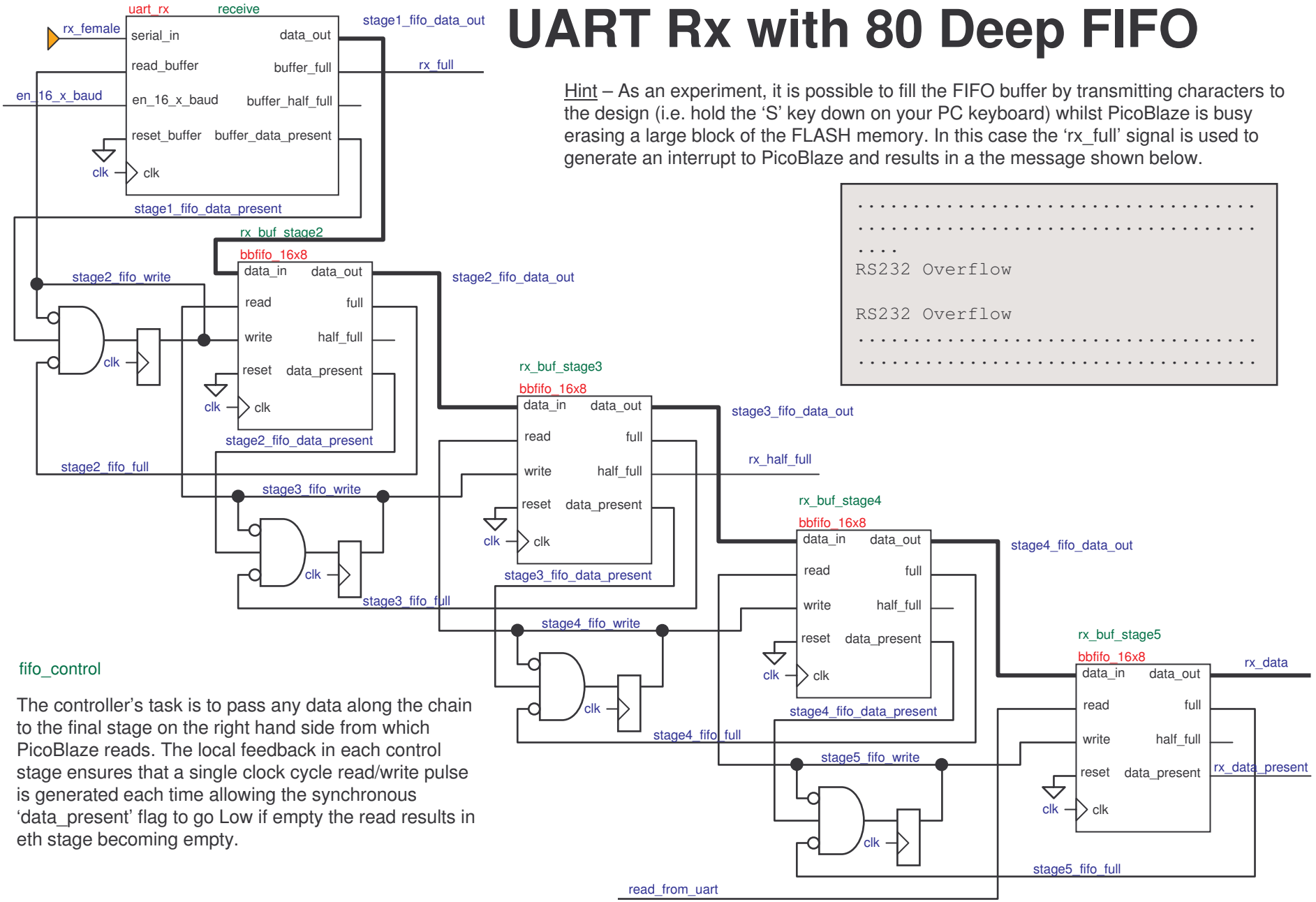


It has become common practice for people to implement FIFO's as a single block of memory of an adequate size to prevent an overflow. However, just as with water tanks, such a technique can often results in large (or heavy) units that are difficult to manage and connect up.

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. The local tank is then topped up from the larger tank in the roof using a smaller pipe.

In this reference design I have provided a total FIFO capacity of 80 characters by using five FIFO's (water tanks!) each providing 16-bytes. This means that each buffer is a natural size to be implemented using the highly efficient SL16E form of distributed memory. The control logic simply moves any data along the cascade chain towards the final output every time there is space in the next FIFO (i.e. not full).

UART Rx with 80 Deep FIFO



Hint – As an experiment, it is possible to fill the FIFO buffer by transmitting characters to the design (i.e. hold the 'S' key down on your PC keyboard) whilst PicoBlaze is busy erasing a large block of the FLASH memory. In this case the 'rx_full' signal is used to generate an interrupt to PicoBlaze and results in a the message shown below.

```
.....................................
.....................................
....
RS232 Overflow

RS232 Overflow
.....................................
.....................................
```

fifo_control

The controller's task is to pass any data along the chain to the final stage on the right hand side from which PicoBlaze reads. The local feedback in each control stage ensures that a single clock cycle read/write pulse is generated each time allowing the synchronous 'data_present' flag to go Low if empty the read results in eth stage becoming empty.

PSM Port Definitions

The PSM code defines constants corresponding to the connections made to the PicoBlaze I/O ports which make the code easier to write, understand and modify. As well as defining the port address themselves, constants are also used to define the use of the bits within a byte and enhance their meanings.

```
CONSTANT status_port, 00              ;UART and memory status input
CONSTANT tx_half_full, 01             ;  Transmitter     half full - bit0
CONSTANT tx_full, 02                  ;    FIFO               full - bit1
CONSTANT rx_data_present, 04          ;  Receiver     data present - bit2
CONSTANT rx_half_full, 08             ;    FIFO          half full - bit3
CONSTANT rx_full, 10                  ;                       full - bit4
CONSTANT spare1, 20                   ;                  spare '0' - bit5
CONSTANT spare2, 40                   ;                  spare '0' - bit6
CONSTANT spare3, 40                   ;                  spare '0' - bit6
;
CONSTANT UART_read_port, 01           ;UART Rx data input
;
CONSTANT UART_write_port, 01          ;UART Tx data output
;
;
CONSTANT SPI_in_port, 02              ;Read serial data from FLASH device
CONSTANT SPI_miso, 01                 ;             Master In Slave Out - bit0
;
CONSTANT SPI_out_port, 02             ;Data to write into FLASH device
CONSTANT SPI_sck, 01                  ;                              Clock - bit0
CONSTANT SPI_rom_cs, 02               ;  FLASH chip select (active Low) - bit1
CONSTANT SPI_mosi, 80                 ;             Master Out Slave In - bit7
;
; Device DNA access ports
;
CONSTANT DNA_control_port, 40         ; Input data and control to the DNA primitive
CONSTANT DNA_clk, 01                  ;      CLK - bit0
CONSTANT DNA_shift, 02                ;    SHIFT - bit1
CONSTANT DNA_read, 04                 ;     READ - bit2
CONSTANT DNA_din, 08                  ;      DIN - bit3
;
CONSTANT DNA_read_port, 03            ; Output data from the DNA primitive
CONSTANT DNA_dout, 01                 ;     DOUT - bit0
```

Each port and bit corresponds to the circuit diagram shown on page 24.

UART connections

SPI_ACCESS port.

Device DNA port.

Software Based SPI

At the heart of SPI communication with the FLASH memory is the requirement to implement a full duplex operation in which data bytes are 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 and using this clock PicoBlaze is able to implement an SPI clock (SCK) rate of 1.786MHz. This is adequate for this and similar applications but falls significantly short of the 66MHz maximum rate supported by the memory device. A hardware based SPI 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 FLASH memory.

```
         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 (CS_B) which must be Low for communication to take place with the FLASH memory.

Drives CS_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 CS_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 by routing them to header pins on J2 (a small addition to the supplied design).

CS0_B

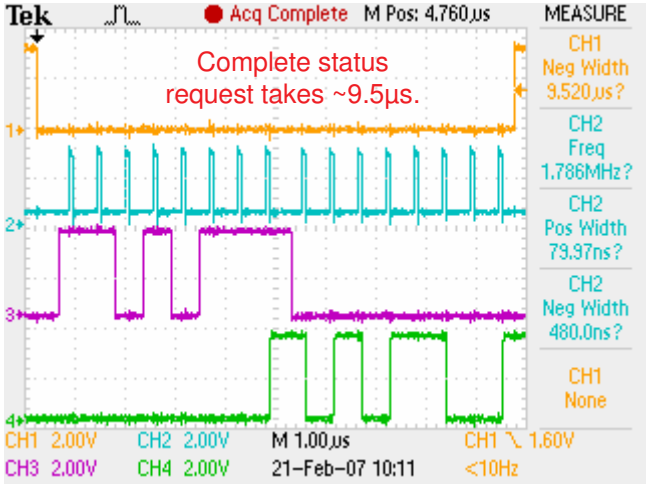
SCK

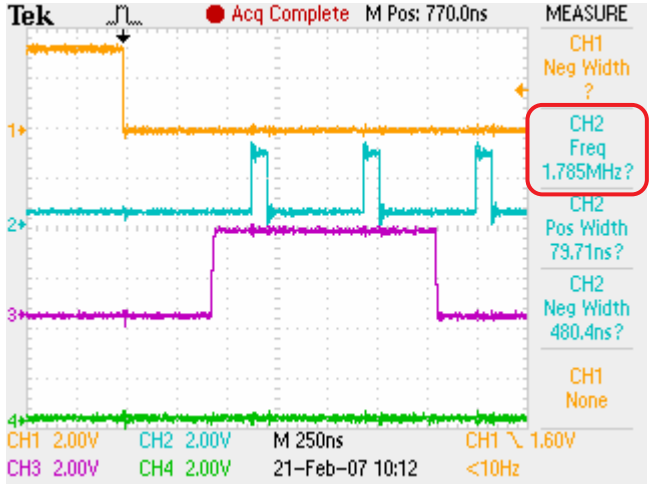
Status command
D7 hex = 11010111_2

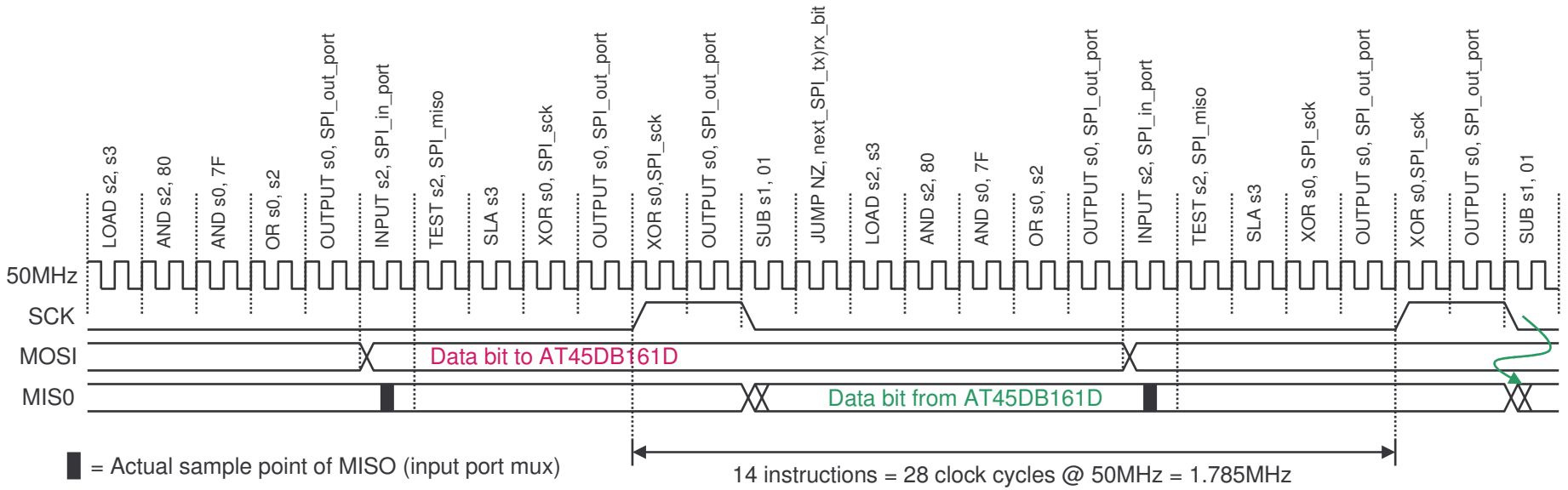
MOSI
(SDI on J23)

Typical Response
AC hex = 10101100_2

MISO
(SDO on J23)







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 shown above.

PSM Code to Confirm FLASH ID

All of the PSM code provided is fully commented. The following pages are intended to provide an introduction to the coding style and highlight the key points when starting to work with the internal FLASH memory.

Reading the device ID of the stacked FLASH ensures that reliable communication with the FLASH memory has been established and that the memory is in a known state ready to continue.

```
CALL SPI_init
```

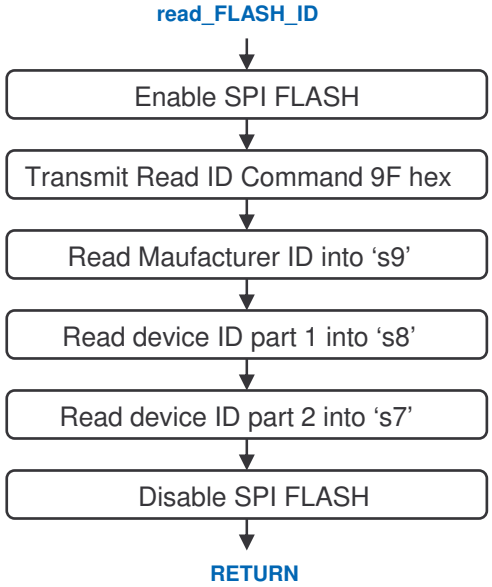
All signals used to control the stacked SPI FLASH via the 'SPI_ACCESS' primitive are generated by PicoBlaze and therefore the simple 'SPI_init' routine ensures all the control signals are in known states. Most importantly the enable signal is High to deselect the FLASH and hence place the FLASH in a known state before any other operations.

```
FLASH_ID_test: CALL read_FLASH_ID
               COMPARE s9, 1F
               JUMP NZ, FLASH_ID_test
```

The 'read_FLASH_ID' routine should read the device ID from the FLASH and return it in the registers 's9', 's8' and 's7' as shown below. The main program tests for the correct response to the first bye before continuing.

s9	s8	s7
1F hex	25 hex	00 hex
Manufacturer ID	Device ID part 1 (XC3S700AN)	Device ID part 2

The 'read_FLASH_ID' routine is also used by the 'I' command in this reference design.



Routine called 'SPI_flash_enable' implements this common signal control

```
CALL SPI_tx_rx
```

SPI is a full duplex serial bus and therefore transmitting and receiving is actually the same operation. The routine called 'SPI_tx_rx' is used to transmit the contents of register 's3' to the FLASH memory and will return with register 's3' containing the data received back from the FLASH memory (see page 29).

The 'SPI_init' routine is used to disable the FLASH.

PSM Code for FLASH Page Size & Ready/Busy

The XC3S700AN FLASH memory has a default page size of 264 bytes but this can be modified by writing a One Time Programmable (OTP) register within the FLASH memory to make the page size 256 bytes. It is important when working with the FLASH memory to know what page size is active. In most cases you will decide the page size to be used and it will remain fixed and in that case you can simplify the PSM code provided. However, this reference design enables you to experiment with both page sizes and hence it must determine the page size and adjust operations and address format appropriately.

The method for determining the page size also facilitates checking the status of the FLASH memory to determine if it is ready for another operation or busy completing a previous operation. This is vital when writing and erasing the memory.

```
CONSTANT AT45DB081D_page_mode, 02
```

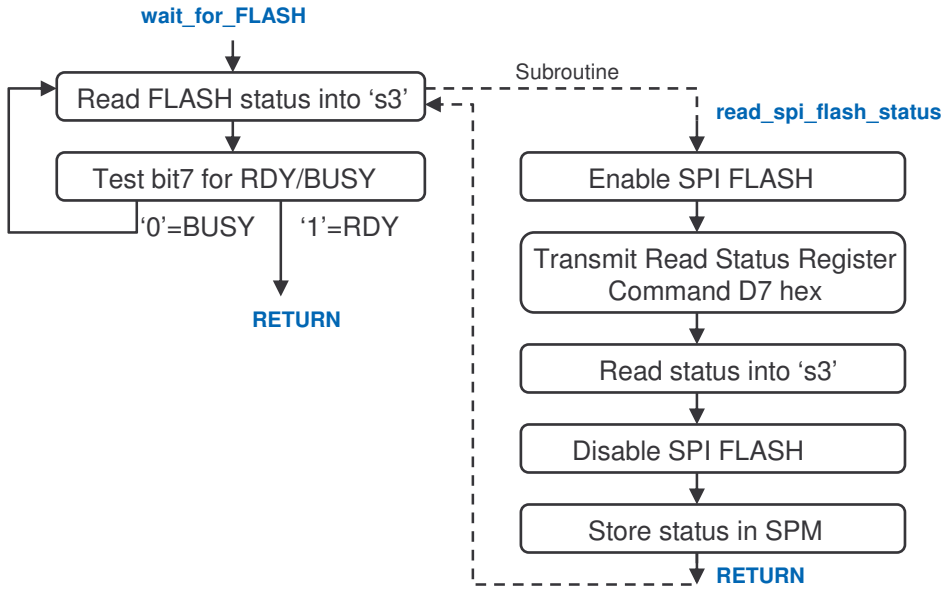
A single scratch pad memory (SPM) location is used to hold the page size information. Only the least significant bit (LSB) is really used with '0' indicating the default page size of 264 bytes and a '1' indicating a page size of 256 bytes.

```
CALL read_spi_flash_status
```

The 'read_spi_flash_status' routine is used to read the status register of the FLASH memory and update the page size indicator in the scratch pad memory. The status value is returned in register 's3'.

```
CALL wait_for_FLASH
```

The 'wait_for_FLASH' routine is used to confirm when the status of the FLASH memory and wait for the RDY/BUSY flag to indicate ready.



The bits of the status byte returned by the FLASH memory have the allocations shown below. The PSM code supplied only uses bit0 and bit7.

Bit7	RDY/BUSY	('1' = ready / '0' = busy)
Bit6	COMP	
Bit5	'1'	Hint – These bits are constant but are different depending on the density of the Spartan-3AN device being used.
Bit4	'0'	
Bit3	'0'	
Bit2	'1'	
Bit1	PROTECT	
Bit0	PAGE SIZE	('0' = 264 bytes / '1' = 256 bytes)

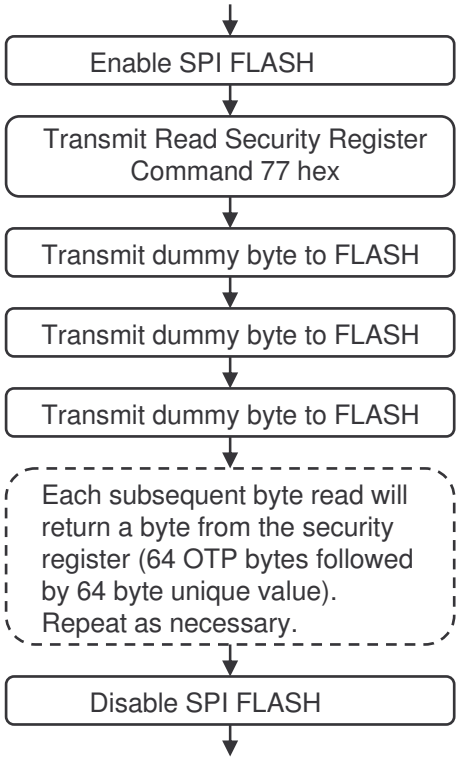
Registers used by this procedure

PSM Code for Reading FLASH Security Register

The SPI FLASH has a 128-byte security register. The first 64-bytes of the security register are allocated as a One Time Programmable (OTP) space which you may program if you wish but will otherwise be blank (byte values FF hex). The remaining 64-bytes of the security register contain a factory programmed unique value for each device which is a similar concept to the device DNA value provided within the FPGA. Therefore each Spartan-3AN device provided you with two unique values which can be exploited in an authentication algorithm.

The reference design provided reads all 128 bytes of the security register and displays them on the terminal (PC). Typically this information would be used internally to the device such as in an authentication algorithm. The flow chart next to the actual code from the reference design focuses only on the FLASH memory centric instructions highlighted in green.

```
                    CALL send_Security          ;display 'security = '
                    CALL SPI_flash_enable       ;enable FLASH memory
                    LOAD s3, 77                 ;Read Security Register command
                    CALL SPI_tx_rx              ;transmit command
                    CALL SPI_tx_rx              ;transmit dummy byte 1
                    CALL SPI_tx_rx              ;transmit dummy byte 2
                    CALL SPI_tx_rx              ;transmit dummy byte 3
                    LOAD s6, 08                 ;8 lines to display
send_security_line: CALL send_CR
                    LOAD s5, 10                 ;16 bytes to display on a line
send_security_byte: CALL send_space
                    CALL SPI_tx_rx              ;read byte from FLASH into s3
                    LOAD s0, s3                 ;display byte
                    CALL send_hex_byte
                    SUB s5, 01                  ;count bytes per line
                    JUMP NZ, send_security_byte
                    SUB s6, 01                  ;count lines
                    JUMP NZ, send_security_line
                    CALL SPI_init               ;FLASH disabled
```



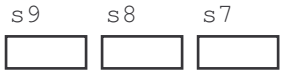
PSM Code for Formatting 24-Bit Addresses

As described on pages 8 and 9 of this document the format of the 24-bit address depends on the active page size. This reference design always prompts the user to enter page numbers and byte addresses as appropriate. PicoBlaze then formats the 24-bit address depending on the active page size determined from the FLASH status byte.

There are two routine used to read the page and address values…

```
CALL ask_page
```

The 'ask_page' routine prompts the user to enter a hexadecimal value in the range 000 to FFF. Any incorrect characters are rejected and the prompt repeated. A valid entry results in 12-bit page address which is shifted into the correct position of the 24-bit address corresponding to the page size (see page 9). The 24-bit value is then returned in the register set [s9,s8,s7] with the byte address forced to zero.




```
CALL ask_address
```

The 'ask_address' routine prompts the user to enter a hexadecimal value in the range 000 to 107 or 00 to FF depending on the active page size. Any incorrect characters or values outside the page range are rejected. A valid entry results in a 9-bit or 8-bit byte address which is superimposed on the existing 24-bit address contained in the register set [s9,s8,s7]. Hence the 'ask_address' routine is only used after the 'ask_page' routine.

Many operations with the FLASH memory require a 24-bit address to be specified. The following code indicates the PSM code required to read a page of memory. Because the SPI transmission of the 24-bit address is so common a separate routine has been provided.

```
CALL SPI_flash_enable
LOAD s3, D2
CALL SPI_tx_rx
CALL spi_tx_address
CALL SPI_tx_rx
CALL SPI_tx_rx
CALL SPI_tx_rx
CALL SPI_tx_rx
…
CALL SPI_tx_rx
…
CALL SPI_init
```

Enable the SPI port and transmit the 'Main Memory Page Read command' which is D2 hex.

Transmit the 24-bit address to identify page and first byte of page

```
spi_tx_address: LOAD s3, s9
                CALL SPI_tx_rx
                LOAD s3, s8
                CALL SPI_tx_rx
                LOAD s3, s7
                CALL SPI_tx_rx
                RETURN
```

Transmit 4 dummy bytes whilst FLASH prepares data to be read

Read data bytes from sequential byte addresses (repeat as desired)

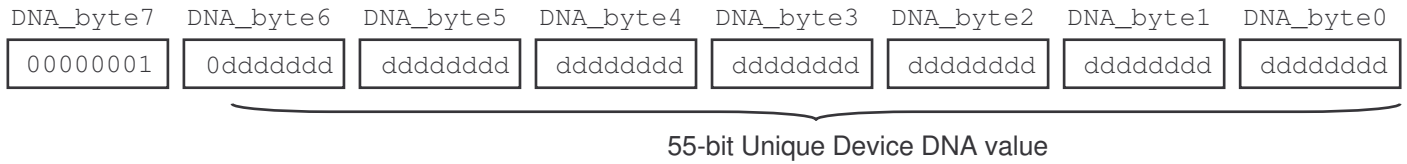
Disable the SPI port to complete the read command.

PSM Code for Reading Device DNA

The PSM code provided includes routines for the reading of the FPGA device DNA. This value is displayed by the 'I' command but would typically be used as a product serial number or as part of an authentication algorithm. Device DNA is described in more detail in the reference design called 'Device DNA Reader' for the Spartan-3A Starter Kit as indicated on page 14 of this document.

```
CONSTANT DNA_byte0, 10
CONSTANT DNA_byte1, 11
CONSTANT DNA_byte2, 12
CONSTANT DNA_byte3, 13
CONSTANT DNA_byte4, 14
CONSTANT DNA_byte5, 15
CONSTANT DNA_byte6, 16
CONSTANT DNA_byte7, 17
```

In this particular design, 8 bytes of scratch pad memory are defined to hold the device DNA value. The scratch pad memory contents are as indicated below following execution of the 'read_device_DNA' routine.



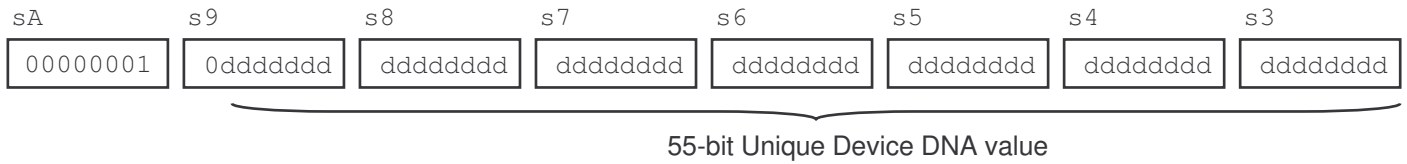

```
CALL DNA_init
```

All signals used to control the 'DNA_PORT' primitive are generated by PicoBlaze and therefore the simple 'DNA_init' routine ensures all the control signals are Low and places the 'DNA_PORT' in a known state before any other operations are commenced.

```
CALL read_device_DNA
```

The 'read_device_DNA' performs all the operations necessary to read the device DNA value from the 'DNA_PORT' and place the value in the scratch pad memory

The 'read_device_DNA' routine actually reads the device DNA a value into a set of 8 registers [sA,s9,s8,s7,s6,s5,s4,s3] and then stores this value into the scratch pad memory. Therefore this register set also holds the device DNA value on return. This indicates the potential for using the device DNA value in an algorithm without involving scratch pad memory if it is desirable to do so.



Registers 's0' and 's2' are also used by this routine so their contents should be unimportant or should be preserved before calling the routine.