PicoBlaze™

NOR FLASH Programmer
for
Spartan-3E Starter Kit

Ken Chapman
Xilinx Ltd
March 2006

Rev.1

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 KCPSM3 or this reference design would be gratefully received by the author.

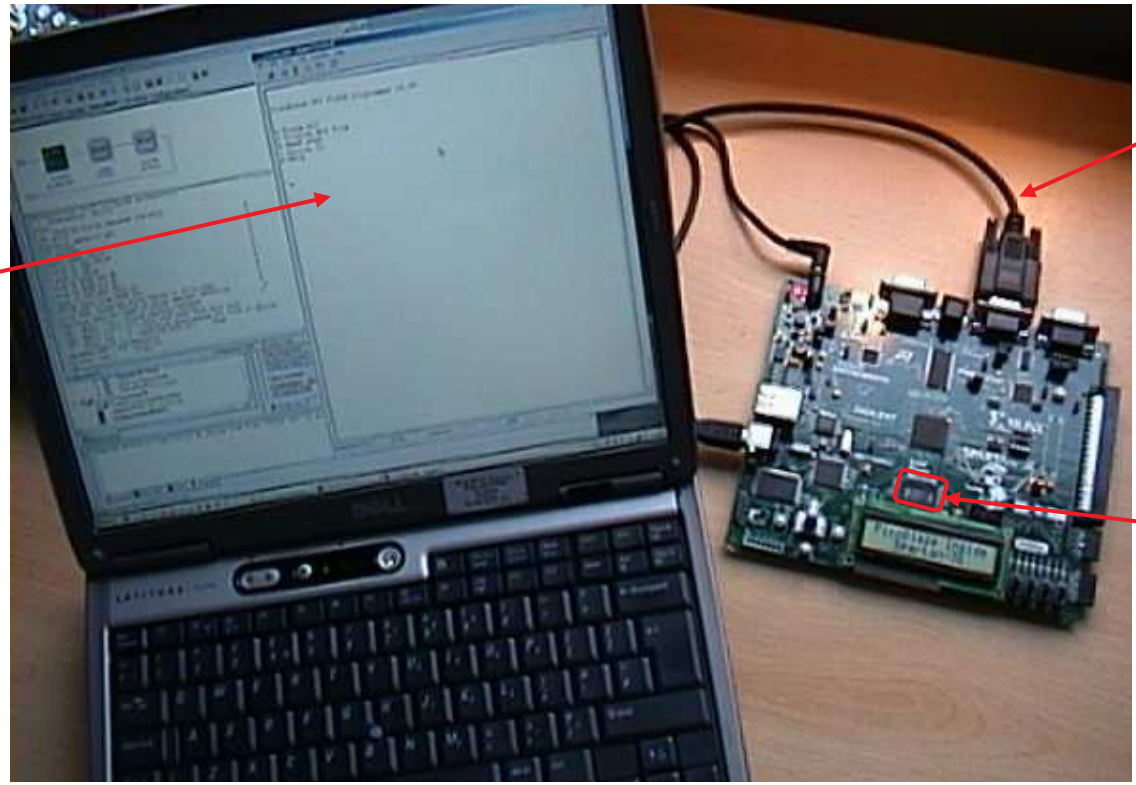
Ken Chapman
Senior Staff Engineer – Spartan Applications Specialist
email: chapman@xilinx.com

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-3E device on your Spartan-3E Starter Kit into a NOR FLASH programmer for the Intel StrataFlash memory (IC22). Using a simple terminal program on your PC such as HyperTerminal, you will be able to manually program individual bytes or download complete configuration images for the Spartan-3E device using standard MCS files. The design also allows you to read the memory to verify contents, perform memory ID check and erase operations.

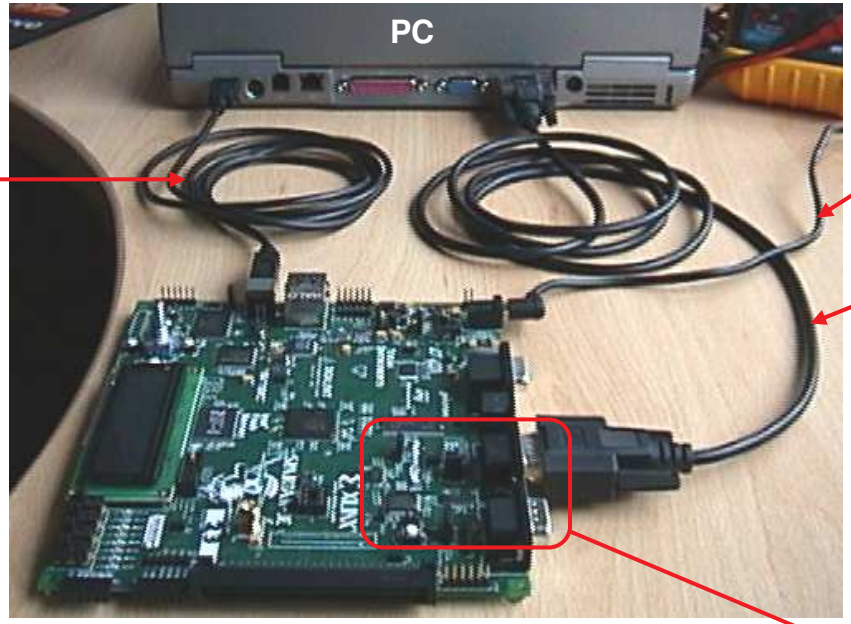
The design is implemented using a single PicoBlaze processor and UART macros occupying under 5% of the XC3S500E device. It is hoped that the design may be of interest to anyone interested in reading, writing and erasing NOR FLASH as part of their own applications even if it is not used exactly as provided.



Using the Design

The design is provided as a configuration BIT file for immediate programming of the Spartan XC3S500E provided on the Spartan-3E Starter Kit. Source design files are also provided for those more interested in the intricacies of the design itself. An example MCS programming file is also provided to enable you to verify that your set up is working.

Hardware Setup



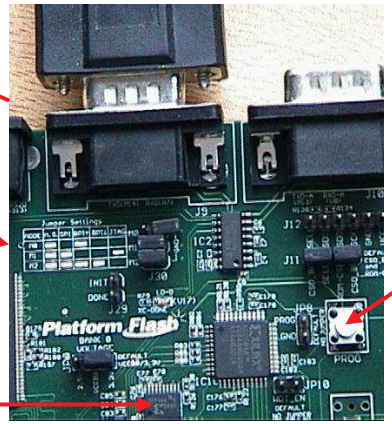
USB cable.
Used to configure the Spartan-3E with the PicoBlaze design.

Cable plus devices on board essentially provide the same functionality as a Platform Cable USB to be used in conjunction with iMPACT.

+5v supply
Don't forget to switch on the board too!
(SWP)

RS232 Serial Cable.
Used for programming of the SPI FLASH memory.

Cable connects J9 on the board to your PC serial port. For this you will need a male to female straight through cable (critically pin2-pin2, pin3-pin3 and pin5-pin5).



J30 configuration mode jumpers and selection chart.
It does not matter which settings you have during the JTAG programming of the XC3S500E from via the USB cable but remember to set correctly (M1=open, M0=M2=short) for BPI-UP configuration from the Parallel FLASH once it has been programmed (press PROG button or cycle power).

Note – This photograph shows the jumpers in SPI configuration mode

PROG button

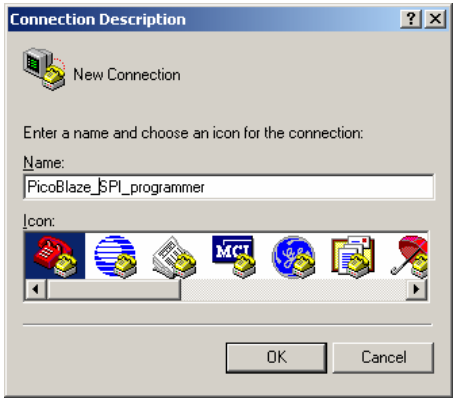
Idea – The PicoBlaze NOR programmer design could be programmed into the XCF04S Platform FLASH device so that it can be loaded directly on the board by changing the J30 jumpers.

Serial Terminal Setup

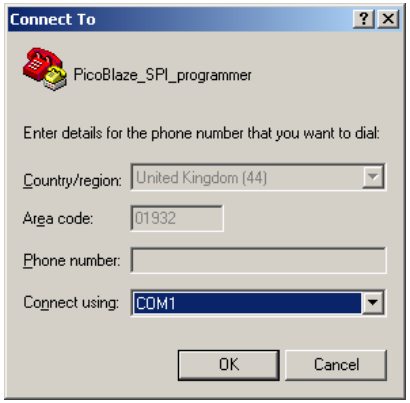
Once the design is loaded into the Spartan-3E, you will need to communicate using the RS232 serial link. Any simple terminal program can be used, but HyperTerminal is adequate for the task and available on most PCs. If you have already use the PicoBlaze SPI programmer reference design then this design uses exactly the same settings

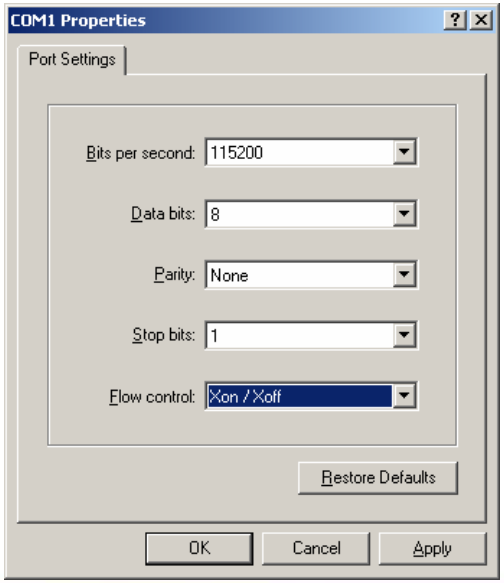
A new HyperTerminal session can be started and configured as shown in the following steps. These also indicate the communication settings and protocol required by an alternative terminal utility.

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.

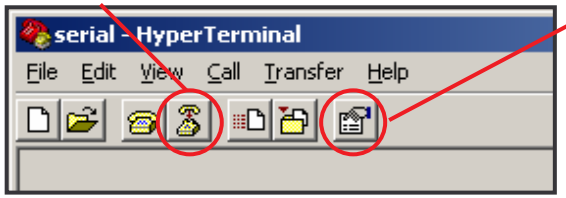
Bits per second : 115200
Data bits: 8
Parity: None
Stop bits: 1
Flow control: **XON/XOFF**

Hint – The design uses XON/XOFF flow control. It may be possible to modify the design and use higher baud rates to reduce SPI programming time .

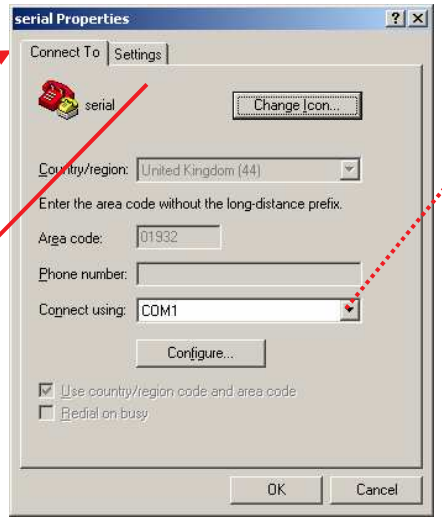
HyperTerminal Setup

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

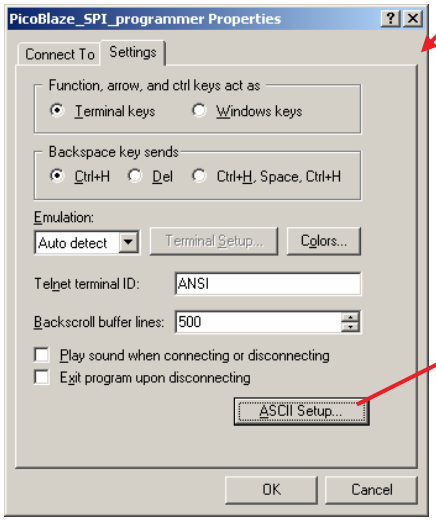


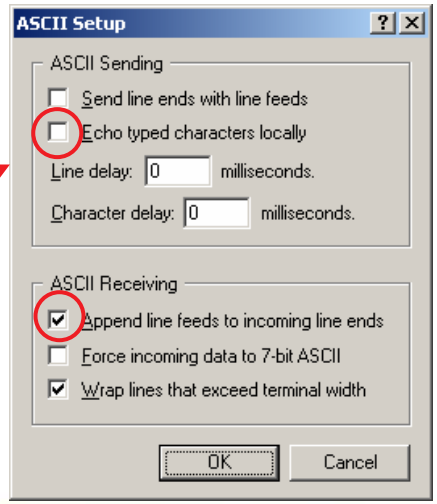
5 - Open the properties dialogue



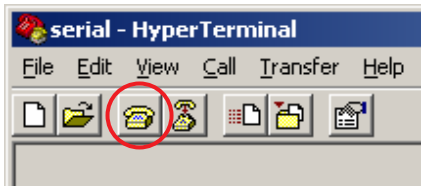
To select a different COM port and change settings (if not correct).

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.

Configure Spartan-3E

The Quick Way!

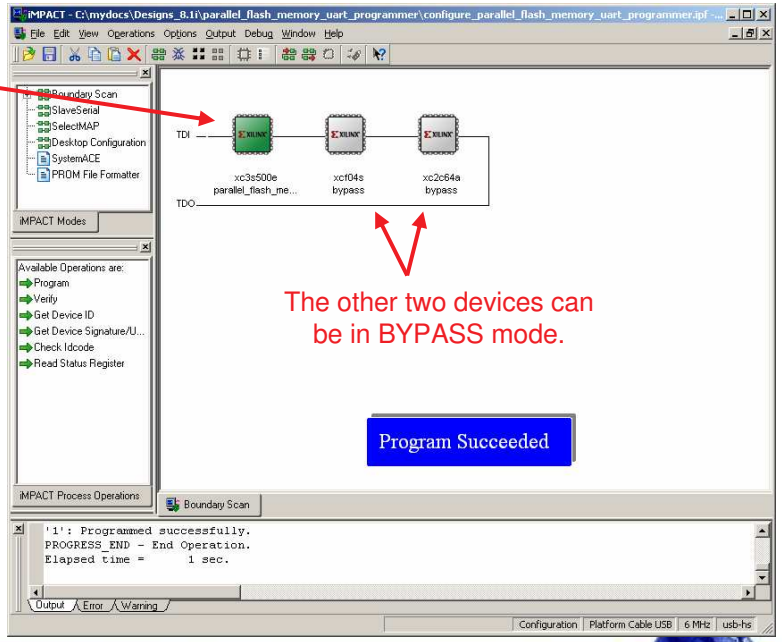
Unzip all the files into a directory.
Check you have the USB cable connected and the board is turned on.
Double click on the file **'install_parallel_flash_memory_uart_programmer.bat'**.
This should open a DOS window and run iMPACT in batch mode to configure the Spartan device.

Your terminal session should indicate the design is working with a version number and simple menu.

Alternatively use iMPACT manually to configure the XC3S500E device on the Spartan-3E Starter Kit via the USB cable.
An iMPACT project file is provided called 'configure_parallel_flash_memory_uart_programmer.ipf' or you can set up your own with the BIT file provided.

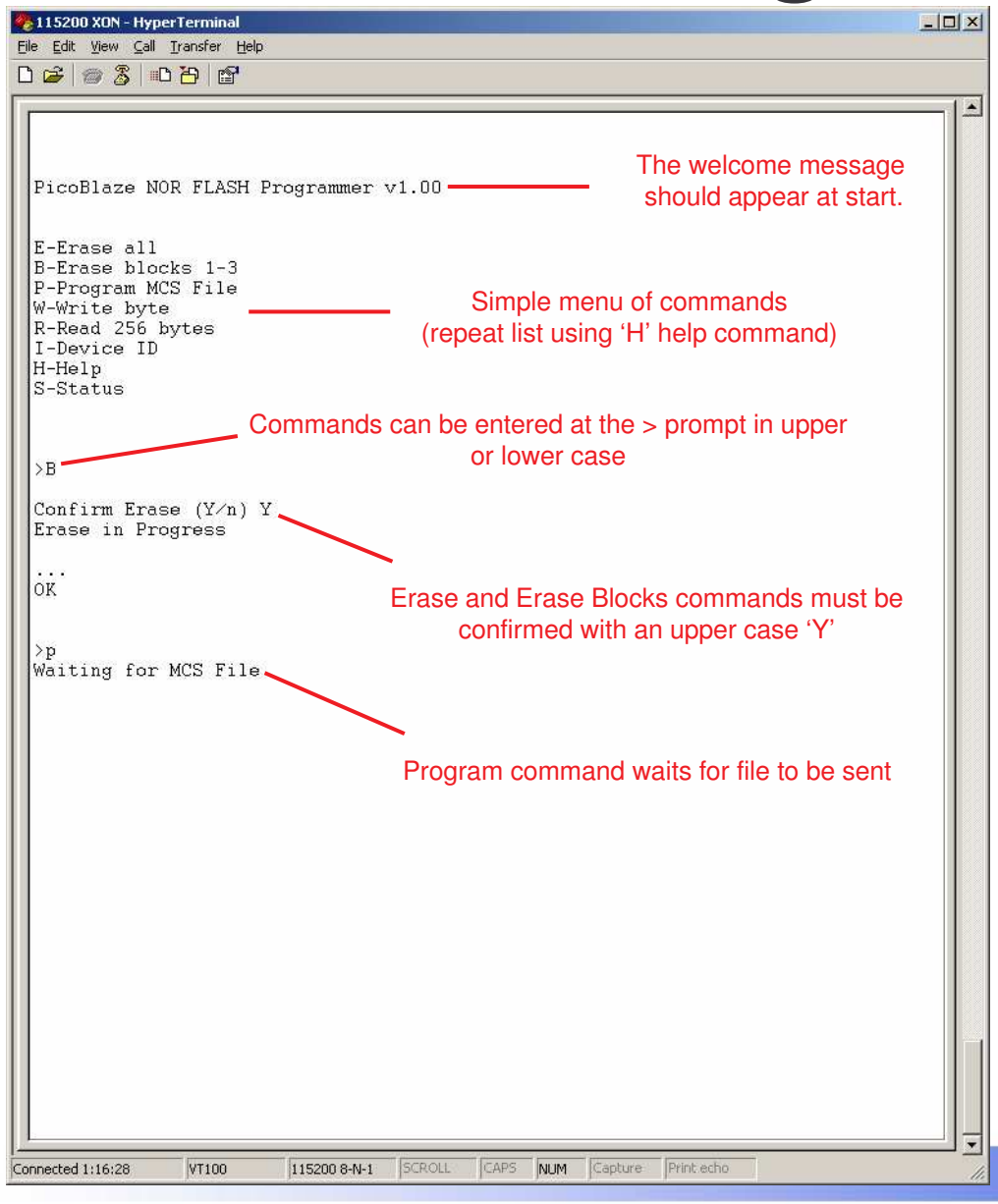
Configure XC3S500E with provided BIT file 'parallel_flash_memory_uart_programmer.bit'

Select xc3s500e device and right click to access program option



Hint – Any warning about 'JtagClk' can safely be ignored.

Talking to PicoBlaze



The welcome message should appear at start.

Simple menu of commands (repeat list using 'H' help command)

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

Erase and Erase Blocks commands must be confirmed with an upper case 'Y'

Program command waits for file to be sent

PicoBlaze implements a simple menu.

The following pages describe each command in detail.

‘H’, ‘I’, ‘S’, ‘E’ and ‘B’ Commands

H – Help command displays the simple menu again.

I – Read Identification code of the Intel StrataFLASH memory.

```
>i
ID= 89 18
```

This command is a good way to confirm communication with the NOR FLASH is working. The expected response is 89 18 where ’89’ is the Device Manufacturer Code (Intel) and ’18’ is the Memory ID code for the 128Mbit size device (please see Intel data sheet for more details)

S – Read the status register of the Intel StrataFLASH memory.

```
>s
80
```

The 8-bit status register is used during programming and erase operations. The MSB (bit7) indicated when the memory is ready (1) or busy (0). The lower bits all indicate errors of some kind and therefore the only desirable response ’80’ hex. This design performs no error checking or clearing but you could add these functions if required (please see Intel data sheet for more details).

E – Erase command will erase ALL of the 128Mbit memory.

```
>e

Confirm Erase (Y/n) Y
Erase in Progress
.....................
.............
OK
```

Note that the device will be completely erased using this command and hence you will be asked to confirm the operation with an upper case ‘Y’.

The 128Mbit device is organised into 128 blocks each of 128K-bytes. Each block could take up to 4 seconds to erase although typically it takes only 1 second. Therefore at best this command will take the best part of **2 minutes** to complete and at worst could take over 8 minutes (please see Intel data sheet for more details).

B – Erase Blocks command will erase blocks 0 to 2 only. This covers the address range 000000 to 05FFFF which is consistent with the storage of a configuration file for the XC3S500E device. This command is faster that the ‘E’ command and will leave the upper memory unchanged

```
>b

Confirm Erase (Y/n) Y
Erase in Progress
...
OK
```

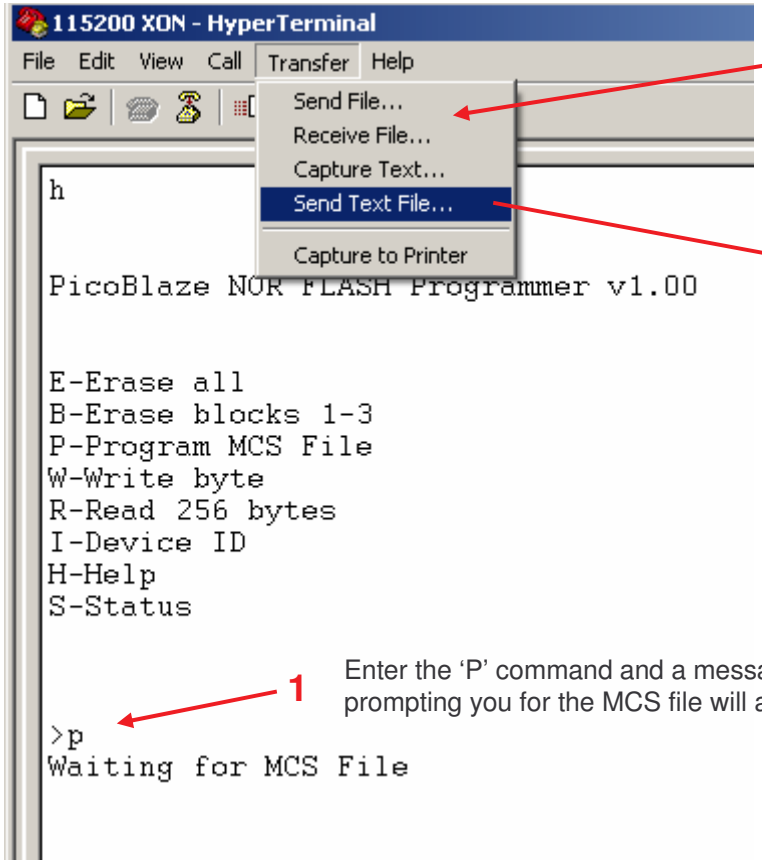
You will be asked to confirm the operation with an upper case ‘Y’.

The erase blocks command can take up to 12 seconds per sector (4 seconds per block). Typically this command will take 3 seconds to complete.

‘P’ Command

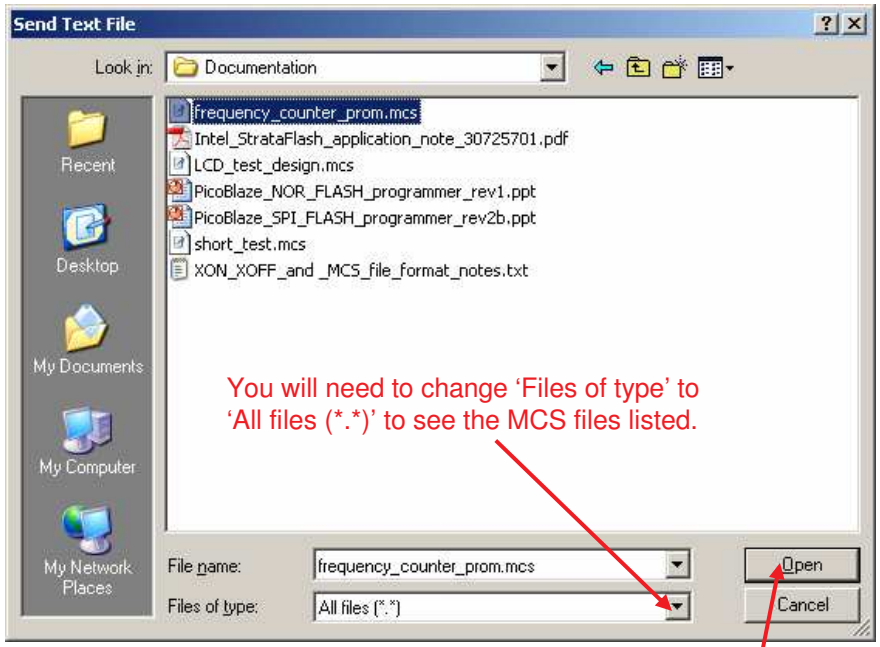
P – Program command.

This is the most important command as it will allow you to program the StrataFLASH device with a configuration bit stream suitable for the XC3S500E to load from at power up, pressing the PROG button or using multi-boot techniques. Later in this documentation we will consider how to prepare an MCS file and what is actually happening, but for now this page shows how to program the provided example file ‘frequency_counter_prom.mcs’ into the memory.



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

3 Navigate to the appropriate directory and select the desired MCS file which in this case is ‘frequency_counter_prom.mcs’.



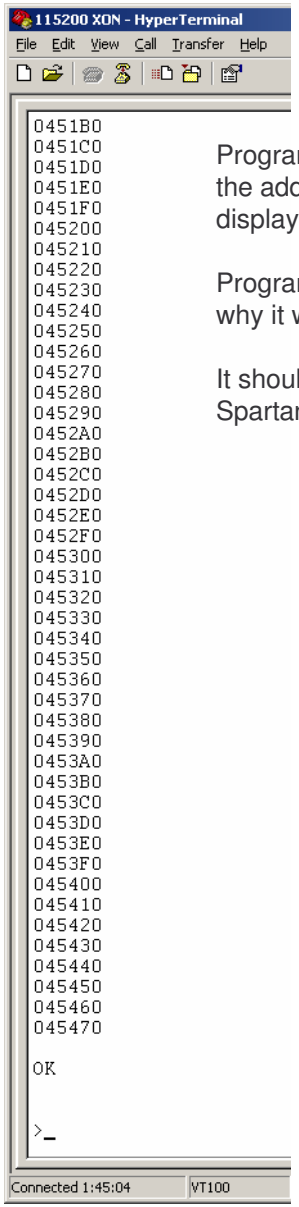
You will need to change ‘Files of type’ to ‘All files (*.*)’ to see the MCS files listed.

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

Hint If you accidentally enter the ‘P’ command you can get out by carefully typing the end of file record found in an MCS file which is…..
:00000001FF

Once you are happy with your selection click on ‘Open’. 4

‘P’ Command continued



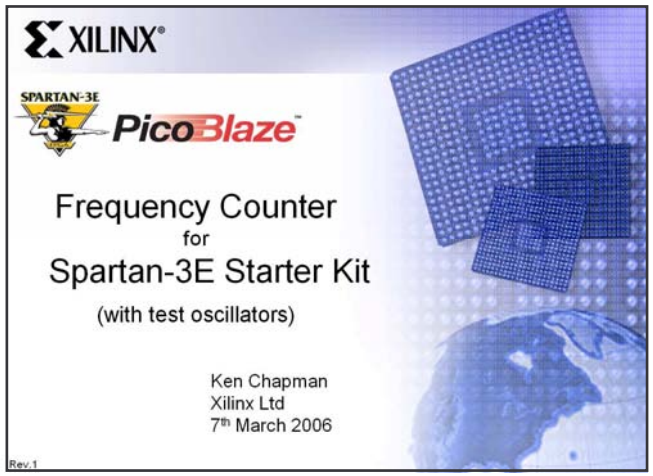
Programming will start immediately and will be indicated by a running display list of hexadecimal numbers. Each number indicates the address currently being programmed in the NOR FLASH memory as defined in the MCS file. For the XC3S500E the final address displayed is 045470 and hence this can be used to monitor progress.

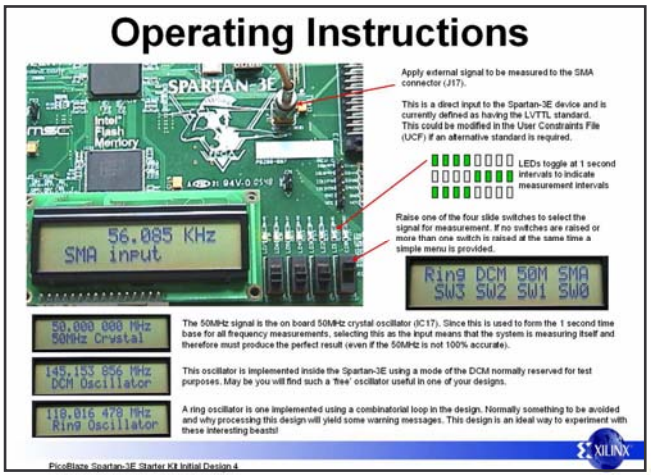
Programming will typically take **80 seconds** to complete. This time is almost entirely as a result of the RS232 serial interface and why it will be useful to investigate higher baud rates in future. The programming will complete with ‘OK’ and a return to the > prompt.

It should now be possible to set the J30 mode jumpers to BPI-UP mode and press the PROG button on the board to reconfigure the Spartan device directly from the new configuration image stored in the NOR FLASH..

Obviously once you have reconfigured the Spartan-3E using the image stored in FLASH memory the programmer design will have been replaced. So if you want to use the programmer design again, you must reload it via iMPACT.
If iMPACT fails to configure the Spartan-3E with the programmer then modify the J30 jumpers to select a mode other than BPI UP or DOWN (say Master serial) and try again. This is an issue with Spartan-3E devices of the ‘stepping 0’ version which were fabricated before April 2006.

If you used the supplied MCS file, then your board should now have been transformed into a 200MHz frequency counter. This design also uses PicoBlaze and is available as a reference design from **www.xilinx.com/s3estarter**.





‘R’ Command

R – Read command.

The read command allows you to observe 256 consecutive bytes stored in the StrataFLASH memory.

After entering the ‘R’ command you will be prompted to enter a start address.
You should then enter a 6 digit hexadecimal value 000000 to FFFFFF.
Entering an illegal hex character will result in the ‘address=‘ prompt being repeated.

```
>r
address=045400

045400  04 00 00 00 04 00 00 00 04 00 00 00 04 00 00 00
045410  04 00 00 00 04 00 00 00 04 00 00 00 04 00 00 00
045420  04 00 00 00 04 00 00 00 04 00 00 00 04 00 00 00
045430  04 00 00 00 04 00 00 00 04 00 00 00 04 00 00 00
045440  04 00 00 00 04 00 00 00 04 00 00 00 04 00 00 00
045450  0C 00 01 80 00 00 00 A0 0C 00 05 80 00 00 00 00
045460  0C 00 00 80 00 00 FA EA 0C 00 01 80 00 00 00 B0
045470  04 00 00 00 04 00 00 00 04 00 00 00 04 00 00 00
045480  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
045490  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0454A0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0454B0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0454C0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0454D0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0454E0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0454F0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

OK


>_
```

The display will indicate the address of the first of 16 bytes shown on each line followed by the 16 successive bytes.

This example shows the end of the configuration image loaded programmed using the ‘frequency_counter_prom.mcs’ file.

Hint: Data in an erased device will be ‘FF’ so if you read ’00’ it has been programmed. It is common for a configuration bit file to contain many ’00’ bytes especially if the design is relatively small.

‘W’ Command

R – Write Byte command.

The write byte command allows you to write a single byte at any address.

After entering the ‘W’ command you will be prompted to enter an address.
You should then enter a 6 digit hexadecimal value 000000 to FFFFFF.
Entering an illegal hex character will result in the ‘address=‘ prompt being repeated.

You will then be prompted to enter the data value and you should enter a 2 digit hexadecimal value 00 to FF.
Entering an illegal hex character will result in the ‘data=‘ prompt being repeated.

```
>w
address=0454BC
data=42

OK


>r
address=045400

045400  04 00 00 00 04 00 00 00 04 00 00 00 04 00 00 00
045410  04 00 00 00 04 00 00 00 04 00 00 00 04 00 00 00
045420  04 00 00 00 04 00 00 00 04 00 00 00 04 00 00 00
045430  04 00 00 00 04 00 00 00 04 00 00 00 04 00 00 00
045440  04 00 00 00 04 00 00 00 04 00 00 00 04 00 00 00
045450  0C 00 01 80 00 00 00 A0 0C 00 05 80 00 00 00 00
045460  0C 00 00 80 00 00 FA EA 0C 00 01 80 00 00 00 B0
045470  04 00 00 00 04 00 00 00 04 00 00 00 04 00 00 00
045480  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
045490  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0454A0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0454B0  FF FF FF FF FF FF FF FF FF FF FF FF FF 42 FF FF
0454C0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0454D0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0454E0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0454F0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

OK
```

This read display shows how address 0454BC has been modified to 42 hex.

Hint: FLASH memory only allows logic ‘1’ to be converted to logic ‘0’ when writing data. Bits can only be restored to logic ‘1’ when erasing complete blocks such that all locations contain ‘FF’. It is possible to modify data using the write command providing the ‘1’ to ‘0’ rule is observed.

MCS files and Device configuration

An MCS file contains additional information to define the storage address which PicoBlaze interprets as well as obtaining the configuration data. How an MCS file defines the addresses is beyond the scope of this document at this time, but in general the first lines of the MCS file defining an FPGA BPI-UP configuration from NOR FLASH will be associated with address zero (000000) and each line contains 16 data bytes to be stored in sequential locations.

If we look at the supplied MCS example file 'frequency_counter_prom.mcs' the first configuration data byes can be identified in each line. Having programmed the NOR FLASH memory, it is possible to read back those same data bytes with the 'R' command with start address '000000'.

Start of MCS file with byte data highlighted in blue

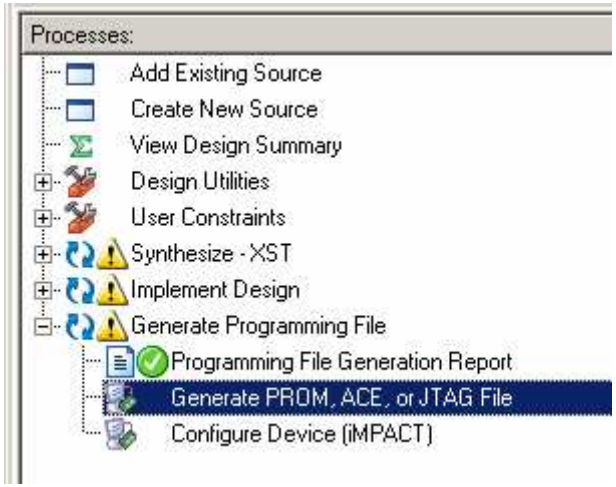
```
:020000040000FA
:10000000FFFFFFFF5599AA660C000180000000E089
:100010000C80068000000060C80048000008CA785
:100020000C800380804304C90C00038000000000A2
:100030000C000180000000900C0004800000000013
:100040000C000180000000800C0002000A8028598A
:10005000000000000000000000000000000000A0
:10006000000000000000000000000000000000090
:10007000000000000000000000000000000000080
:10008000000000000000000000000000000000070
:10009000000000000000000000000000000000060
:1000A000000000000000000000000000000000050
:1000B000000000000000000000000000000000040
:1000C000000000000000000000000000000000030
:1000D000000000000000000000000000000000020
:1000E000000000000000000000000000000000010
:1000F000000000000000000000000000000000000
etc
```

```
>r
address=000000

000000  FF FF FF FF 55 99 AA 66 0C 00 01 80 00 00 00 E0
000010  0C 80 06 80 00 00 00 06 0C 80 04 80 00 00 8C A7
000020  0C 80 03 80 80 43 04 C9 0C 00 03 80 00 00 00 00
000030  0C 00 01 80 00 00 00 90 0C 00 04 80 00 00 00 00
000040  0C 00 01 80 00 00 00 80 0C 00 02 00 0A 80 28 59
000050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000070  00 00 00 00 00 00 00 00 00 00 00 00 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

OK
```

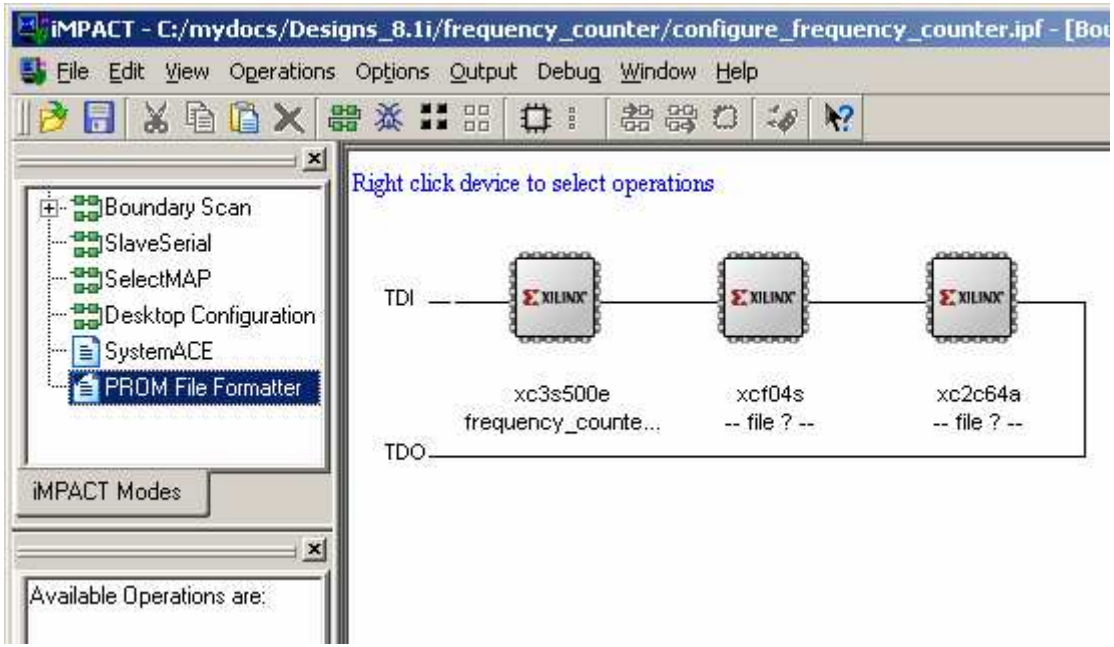
Preparing an MCS file

This design has been provided so that a ‘default’ MCS programming file generated by the ISE tools can be used. The following images indicate how that may be achieved but is not intended to replace existing documentation for PROM generation.

1) Select ‘Generate PROM’ in Project Manger

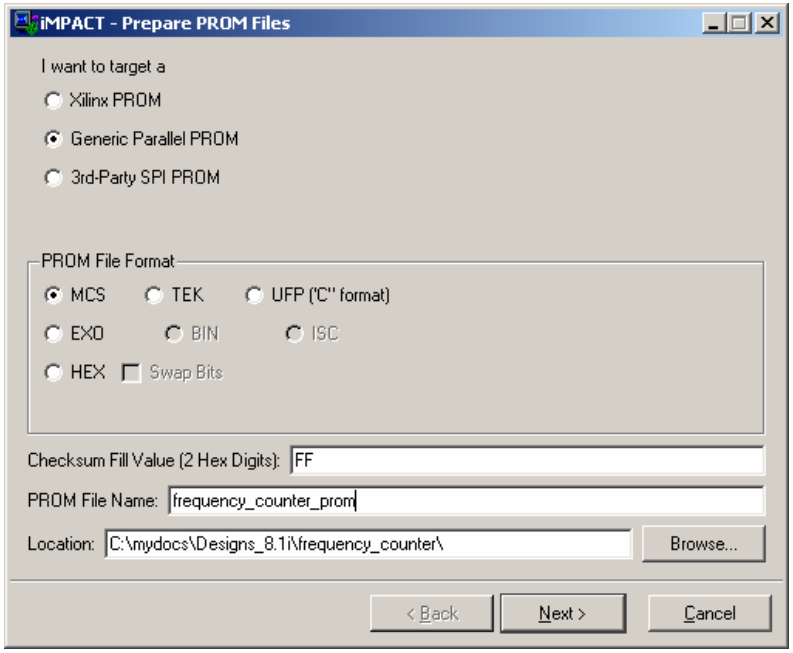


2) This launches iMPACT in which you need to select the PROM File Formatter mode.
(You probably need to expand the upper left window as shown here or pan down to see it).

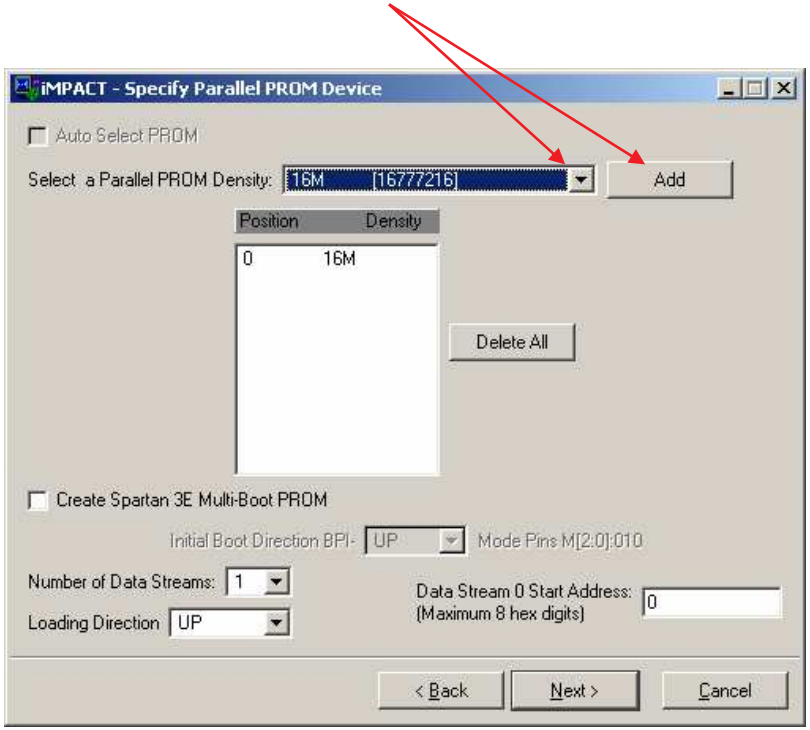


Preparing an MCS file

3) Select
'Generic parallel PROM'
'MCS' file format
and provide a file name and location.

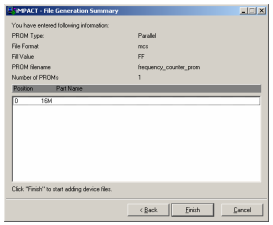


4) Select the density from the list (The 128Mbit device supplied on the Starter Kit board equates to 16M-bytes) from the drop down list and then click 'Add' so that it appears in the centre box.



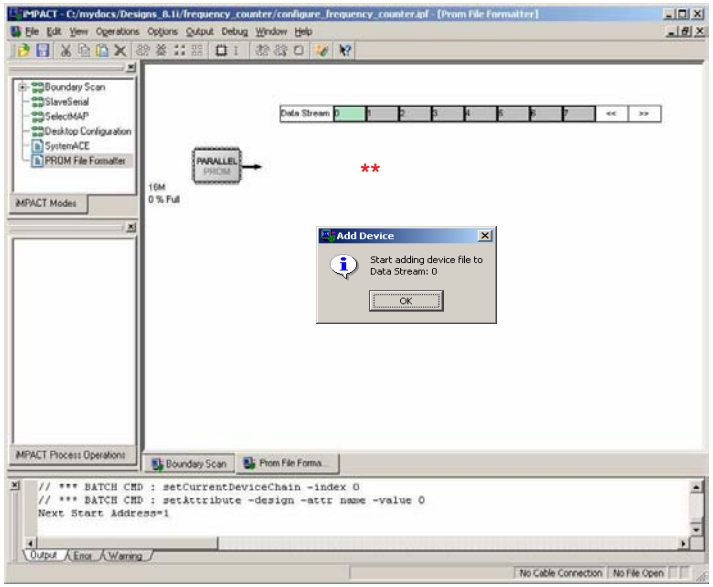
This stage also provides the ability to perform multi-boot designs and set the loading direction for BPI-UP and BPI-DOWN configuration modes. In this case the simple (default) BPI-UP mode will be used and therefore the configuration should be stored at address zero upwards.

5) Summary Page

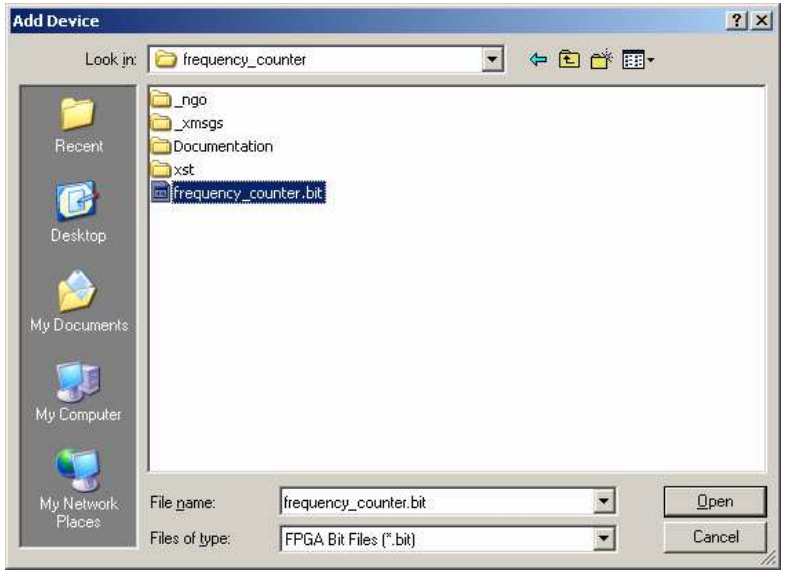


Preparing an MCS file

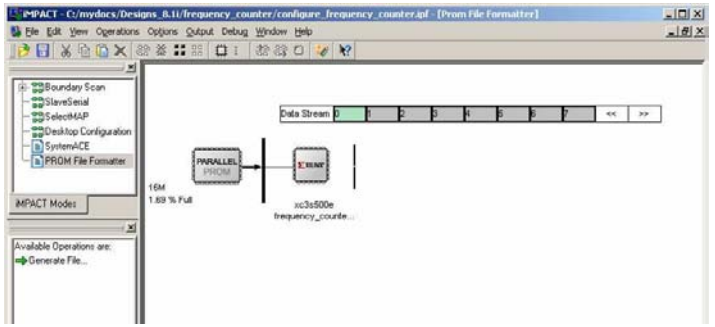
6) You are now presented with a picture of the PROM contents and an ‘Add Device’ box encouraging you to add your first device.
Click ‘OK’ to continue.
(If the ‘Add Device’ box does not appear, then right click where it is marked ** below and select ‘Add Xilinx Device…’)



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

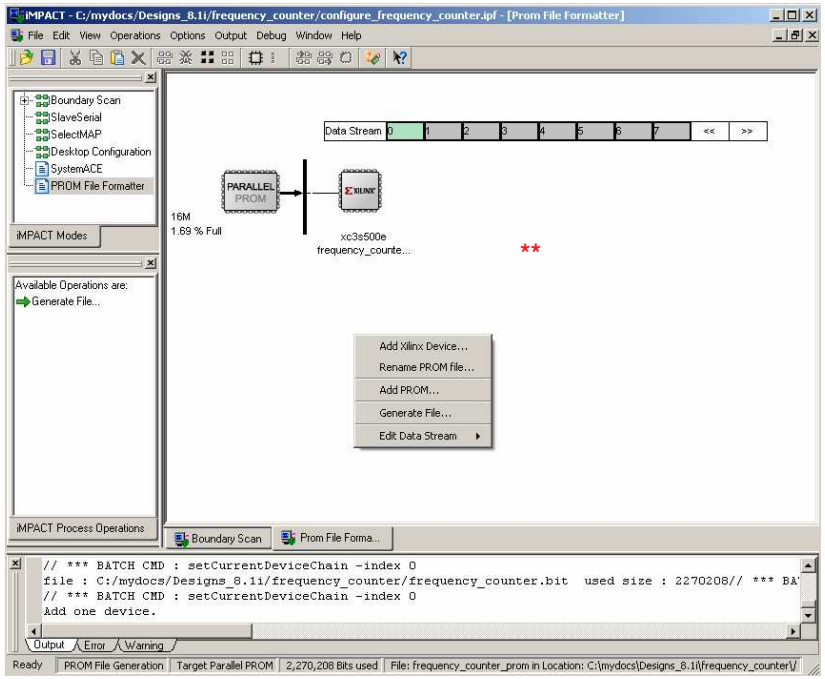


7) The picture updates to show the BIT file at the beginning of the PROM.

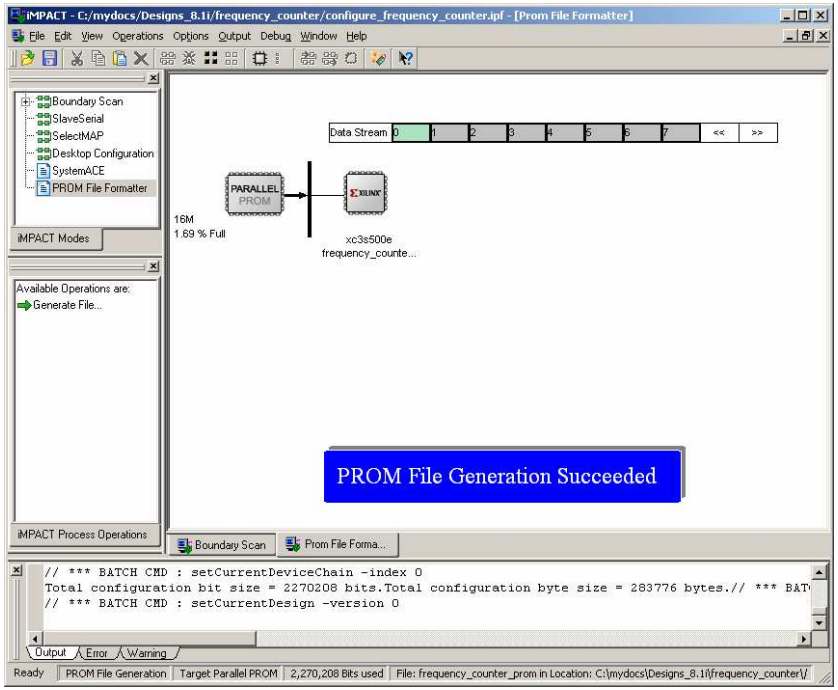


Preparing an MCS file

8) Right click where it is marked ** below and select ‘Generate File…’



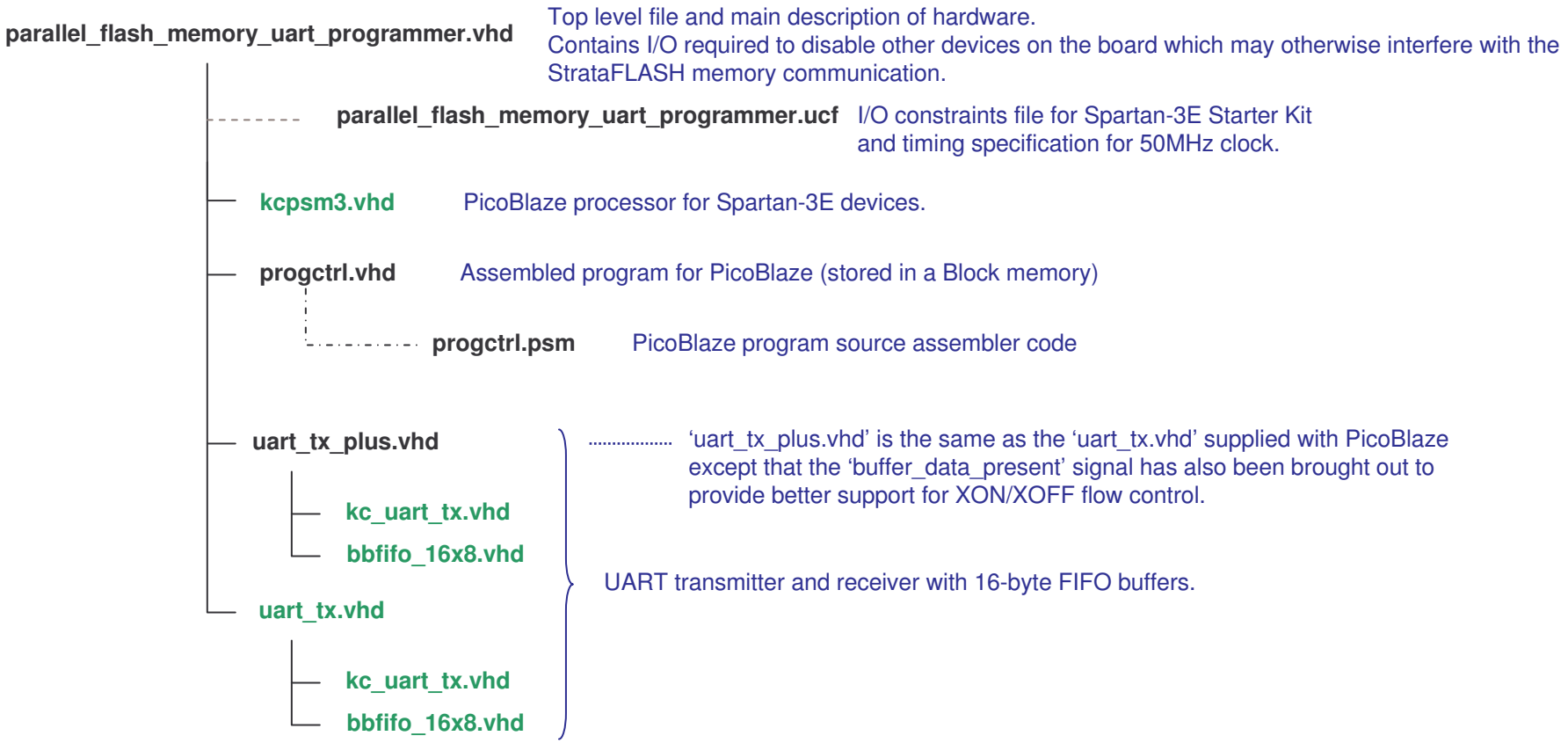
9) The file is written to the directory specified in step 3 and the process is complete.



Design Files

For those interested in the actual design implementation, the following pages provide some details and an introduction to the source files provided. This description may be expanded in future to form a more complete reference design. As well as these notes, the VHDL and PicoBlaze PSM files contain many comments and descriptions describing the functionality.

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

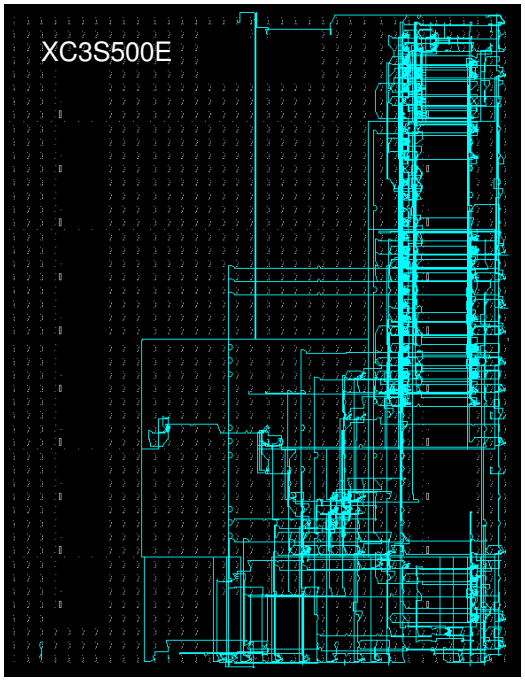
The images and statistics on this page show that the design occupies just 161 slices and 1 BRAM. This is only 3.5% of the slices and 5% of the BRAMs available in an XC3S500E device and would still be less than 17% of the slices in the smallest XC3S100E device.

MAP report

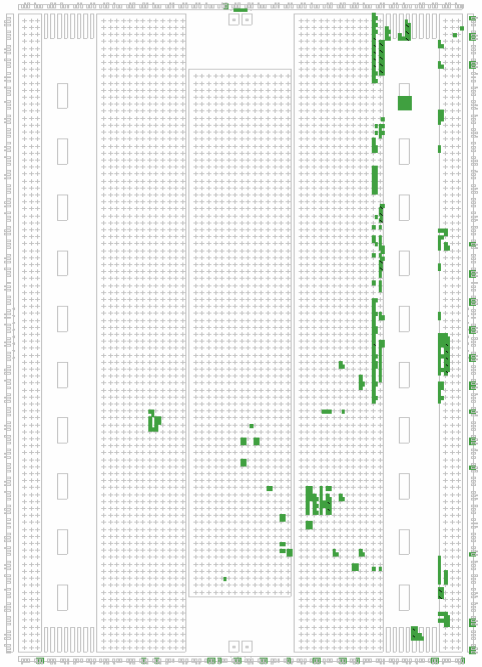
```
Number of occupied Slices:     161 out of    4,656     3%
Number of Block RAMs:            1 out of       20     5%

Total equivalent gate count for design:  79,043
```

PicoBlaze and the UART macros make extensive use of the distributed memory features of the Spartan-3E device leading to very high design efficiency. If this design was replicated to fill the XC3S500E device, it would represent the equivalent of over 1.5 million gates. Not bad for a device even marketing claims to be 500 thousand gates ☺

FPGA Editor view

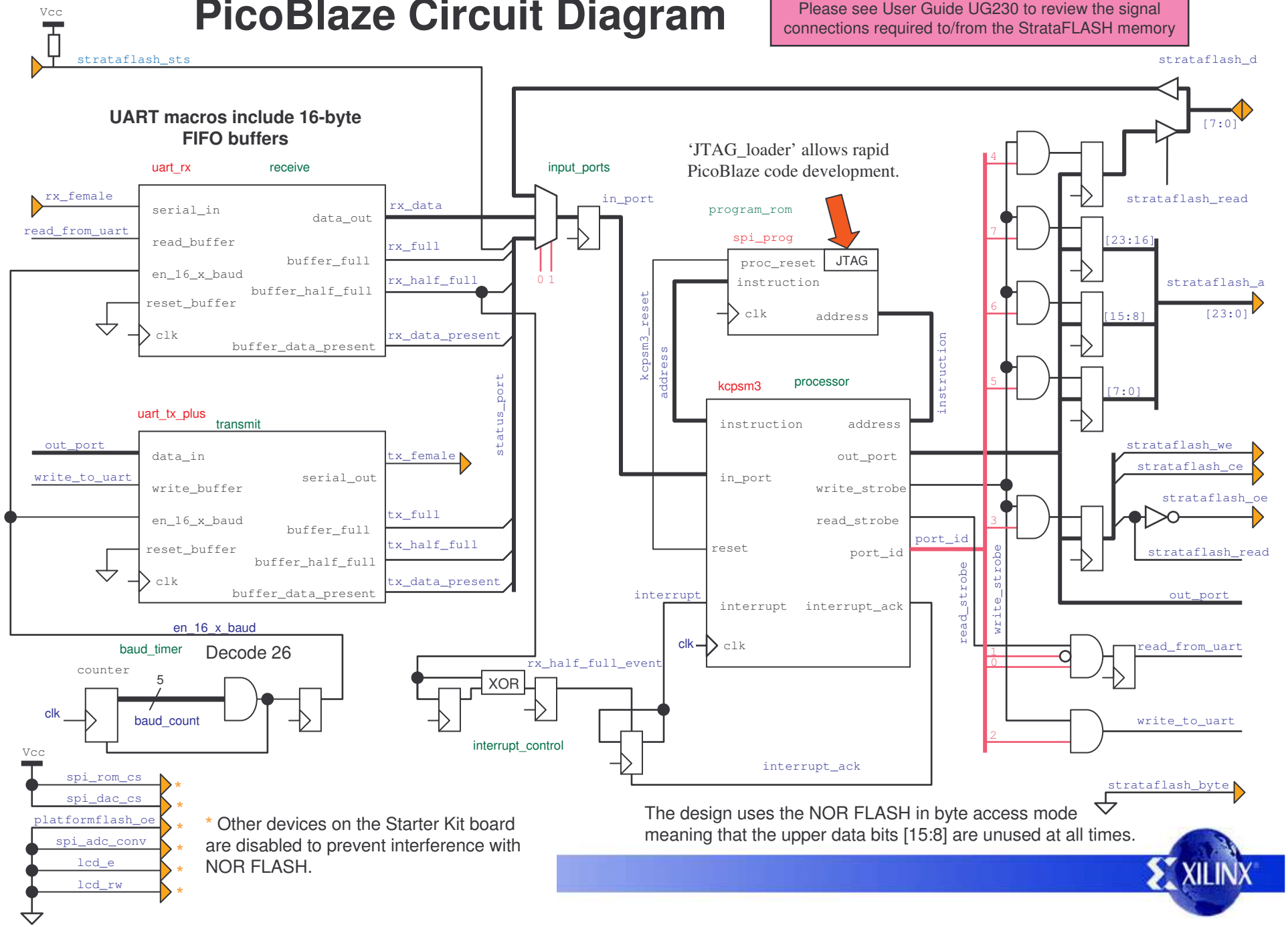


Floorplanner view



PicoBlaze Circuit Diagram

Please see User Guide UG230 to review the signal connections required to/from the StrataFLASH memory



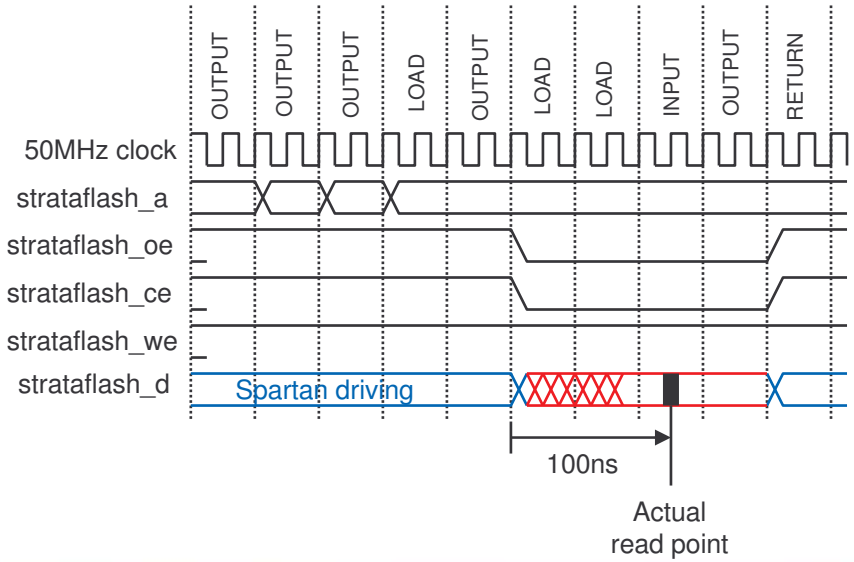
Reading StrataFLASH

Reading the StrataFLASH NOR memory is relatively straightforward. The only issue for PicoBlaze is that it does not have a 24-bit address range and therefore multiple ports are used to achieve the operation.

```
SF_byte_read: OUTPUT s9, SF_addr_hi_port
              OUTPUT s8, SF_addr_mi_port
              OUTPUT s7, SF_addr_lo_port
              LOAD s1, 05
              OUTPUT s1, SF_control_port
              LOAD s1, 06
              LOAD s1, 06
              INPUT s0, SF_data_in_port
              OUTPUT s1, SF_control_port
              RETURN
```

- Set 24-bit address form which to read
- Set controls for read
  - Bit 0 - strataflash_read='1'
    - Enables memory outputs (strataflash_oe='0')
    - Tri-states the Spartan outputs (strataflash_d='Z')
  - Bit 1 - strataflash_ce='0'
    - Enables memory
  - Bit 2 - strataflash_we='1'
    - Write enable is off (read operation)
- Read data
- Deselect StrataFLASH memory

All PicoBlaze instructions execute in 2 clock cycles and the design uses the 50MHz clock source on the board. This makes all timing of the design easy to predict and to ensure that the specifications for the StrataFLASH memory are met.



The access time of the memory is 75ns (see Intel data sheet for details). By including an additional LOAD instruction, the time between setting the controls to read the memory and the actual point of reading is increased by 40ns and the access time in adequate.

Note that the input port multiplexer is pipelined which means that the data from the memory is captured on the first clock edge of the INPUT instruction (as indicated) and then passed into the 's0' register on the second clock edge.

Hint – Data is read from the memory when it is in 'read array' mode (which is the default mode after power up). However, the same read operation is used to access memory status and device information when in other modes.

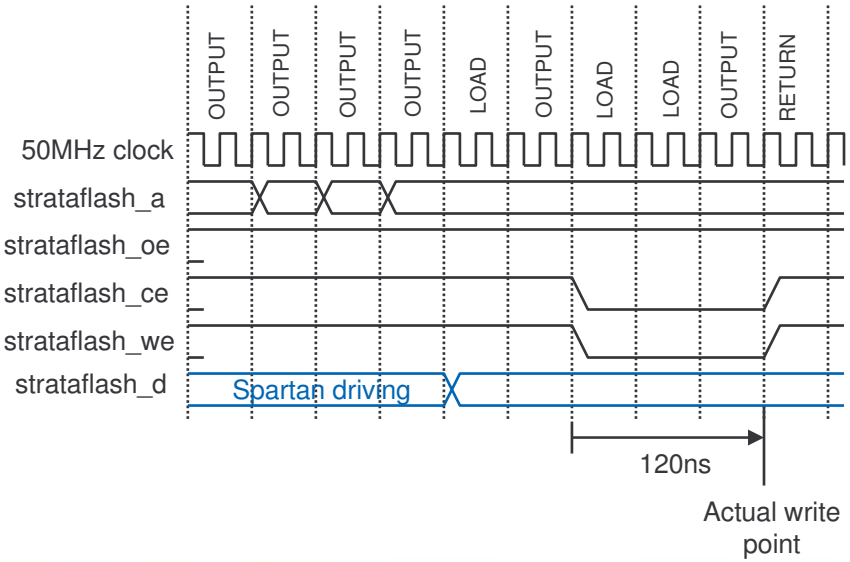
Writing to StrataFLASH

The basic format of a write operation is not so different to that of a read operation. However, the act of writing a byte to the StrataFLASH memory shown on this page is only one part of a process in actually writing data into the memory array such that it is stored and available to read (see following pages).

```
SF_byte_write: OUTPUT s9, SF_addr_hi_port
               OUTPUT s8, SF_addr_mi_port
               OUTPUT s7, SF_addr_lo_port
               OUTPUT s1, SF_data_out_port
               LOAD s1, 00
               OUTPUT s1, SF_control_port
               LOAD s1, 06
               LOAD s1, 06
               OUTPUT s1, SF_control_port
               RETURN
```

- Set 24-bit address form which to read
- Set byte to be written to memory
- Set controls for read
  - Bit 0 - strataflash_read='0'
    Disables memory outputs (strataflash_oe='1')
    Enables the Spartan data outputs
  - Bit 1 - strataflash_ce='0'
    Enables memory
  - Bit 2 - strataflash_we='0'
    Write enable is active
- Deselect memory

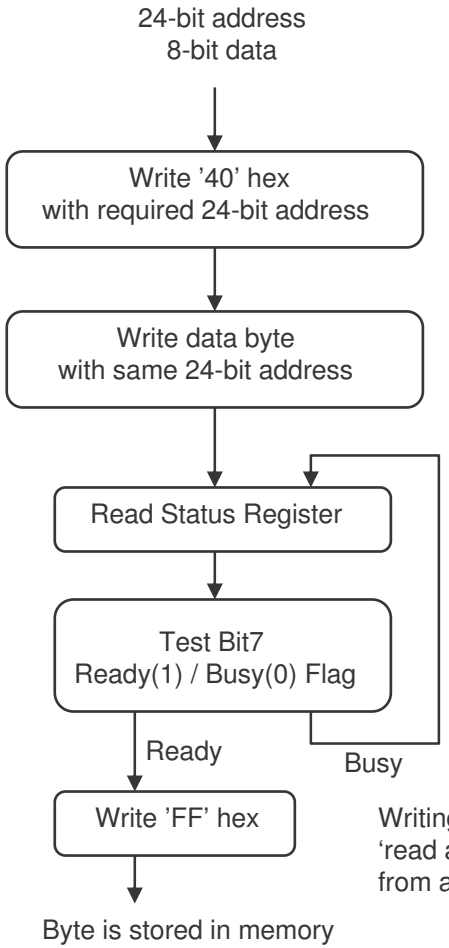
All PicoBlaze instructions execute in 2 clock cycles and the design uses the 50MHz clock source on the board. This makes all timing of the design easy to predict and to ensure that the specifications for the StrataFLASH memory are met.



The setup time to write data to the memory is 60ns(see Intel data sheet for details). As shown, the design allows plenty of margin by including an additional LOAD instruction.

Storing 1-Byte in StrataFLASH

To actually store a byte of data in the StrataFLASH memory something more than a simple write operation is required. The following describes the process which must be followed to store a single byte at a specified address.




```
SF_single_byte_write: LOAD s1, 40
                      CALL SF_byte_write
                      LOAD s1, s0
                      CALL SF_byte_write
                      CALL wait_SF_ready
                      RETURN
```

This process is implemented by the PicoBlaze code shown here. The data must be provided in register 's0' and the 24-bit address in register set [s9,s8,s7].

Before writing the actual data, a command byte '40' hex must be written to tell the memory that a single byte is to be written to the array at the specified address.

Then the actual data is written.

The memory then internally executes the write of the data into the FLASH array. This can take up to 175µs to complete and the memory is generally unavailable during this time. PicoBlaze is able to determine that the memory is busy by reading the status register and Testing bit7. This is achieved using a normal read operation because the device will automatically enter the 'read status register' during this process. The PicoBlaze code is shown to the right and includes a counter which could be used to determine exactly how long the program cycle takes to complete.

```
wait_SF_ready: LOAD sE, 00
               LOAD sD, 00
 wait_SF_loop: ADD sD, 01
               ADDCY sE, 00
               CALL SF_byte_read
               TEST s0, 80
               JUMP Z, wait_SF_loop
               CALL set_SF_read_array_mode
               RETURN
```

Writing 'FF' hex places the memory back to the default 'read array' mode such that data can once again be read from any address.

Hint – The strataflash_sts signal can also be used to determine the ready/busy status but care is needed because this signal can take up to 500ns to be asserted.

Storing up to 32 Bytes

Although bytes can be written individually, the write cycle time can become significant. For example, when this design is programming an XC3S500E configuration into the memory, most lines of the MCS file define 16 bytes of data. Programming these individually could take up to 16 × 175µs = 2.8ms which is equivalent to the time taken to transmit 32 characters on the 115200 baud UART. Since this would cause the UART FIFO buffer to overflow, the XON-XOFF flow control would come into play and ultimately slow the whole MCS programming sequence down as the communication is continuously interrupted.

The StrataFLASH offeres a 'buffer write' programming procedure which reduced the average programming time. The buffer allows up to 32 bytes to be written into a buffer at high speed which are then stored in the FLASH array incurring only one write cycle penalty of up to 654µs. Therefore writing more than 4 bytes using this buffer technique will be faster than individual byte programming. In this design, the 16-bytes defined in each line of the MCS file are programmed this way such that the 654µs maximum delay only equates to the transmission of 8 characters on the UART interface and the communication does not need to be interrupted resulting in the shortest download time.

```
  SF_buffer_write: LOAD s1, E8
                   CALL SF_byte_write
                   CALL SF_byte_read
                   TEST s0, 80
                   JUMP Z, SF_buffer_write
                   LOAD s1, sA
                   SUB s1, 01
                   CALL SF_byte_write
                   LOAD s3, data_start
write_buffer_loop: FETCH s1, (s3)
                   CALL SF_byte_write
                   ADD s7, 01
                   ADDCY s8, 00
                   ADDCY s9, 00
                   ADD s3, 01
                   SUB sA, 01
                   JUMP NZ, write_buffer_loop
                   LOAD s1, D0
                   CALL SF_byte_write
                   CALL wait_SF_ready
                   RETURN
```

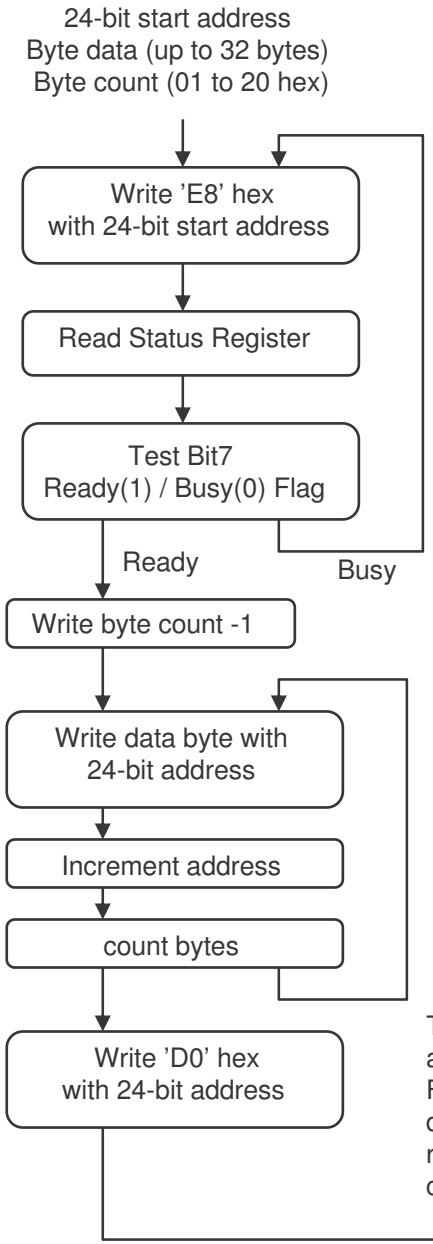
This buffer write process is implemented by the PicoBlaze code shown here and the process is described further on the next page.

PicoBlaze makes use of the scratch pad memory in this application. First it reads a line of the MCS file which it stores in scratch pad memory. It is then able to determine:-

- The 24-bit start address for the data which it holds in register set [s9,s8,s7].
- The number of bytes defined by the line, and hence the number of bytes to be written to the memory. This byte count is held in register 'sA'.

The buffer write routine reads the data from the scratch pad memory and writes it to the NOR memory.

StrataFLASH Write to Buffer Process



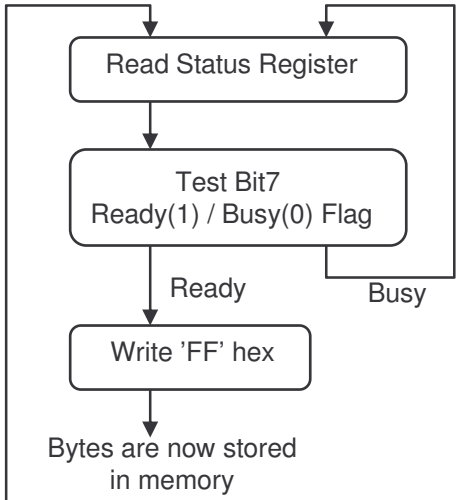
The buffer write command byte 'E8' hex must be written to tell the memory that a multiple byte write procedure is required.

It is possible that the memory is not ready to deal with the request, so the memory automatically enters the 'read status register' mode such that the ready/busy flag can be read and checked. If the device is busy the buffer write command must be repeated until it is ready to continue.

State number of bytes less one (now in the range 00 to 1F hex).

The corresponding number of data bytes must be written to the memory. The supplied routine copies each byte from scratch pad memory and also increments the 24-bit address as each byte as it is written.

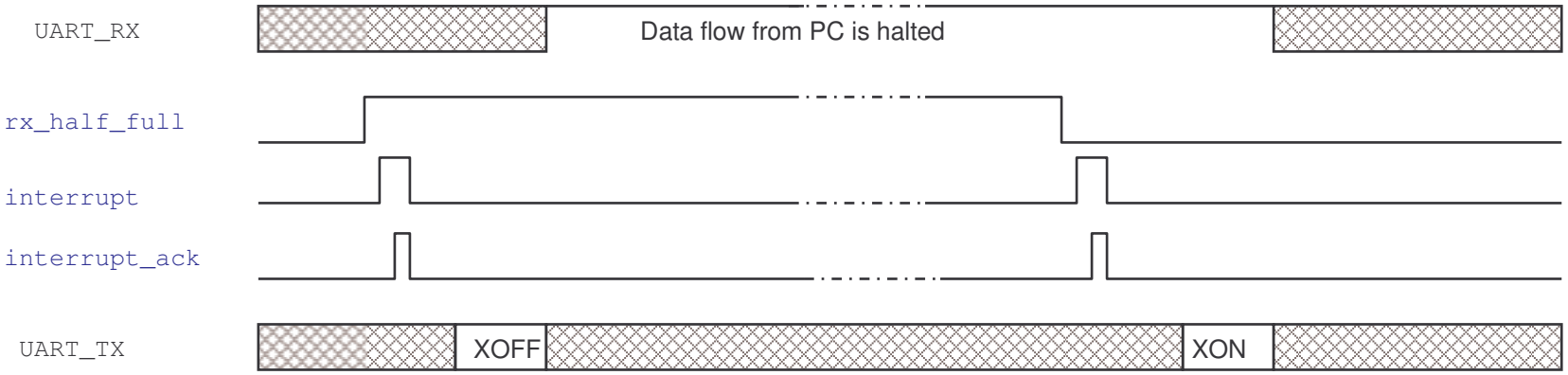
The 'confirm write' command byte 'D0' completes the buffer programming and the memory then performs the actual programming of the data into the FLASH array. This is the part which could take up to 654µs so again the device automatically enters 'read status register' mode such that the ready/busy flag can be read and checked. Once ready, the 'read array' mode can be restored with the 'FF' command byte.



XON/XOFF Flow Control

When the NOR FLASH device executes a program command it could take up to 654µs to complete. At the same time the PC will continue to transmit the MCS file at 115200 baud rate. This could mean that 8 characters are transmitted whilst PicoBlaze is waiting for the memory to be free for writing again. Officially the 16 byte FIFO buffer on the UART receiver should be adequate for this, but any additional delays could make this marginal and cause overflow. For this reason, the design incorporates a degree of XON/XOFF soft control to enable this design to work at without errors.

The principle requirement of flow control, as explained above, is to limit the flow from the PC to the PicoBlaze design. This is achieved by a combination of hardware and software employing interrupts.



The hardware detects when the ‘half_full’ flag on the receiver buffer changes state and generates an interrupt to the PicoBlaze. When PicoBlaze responds to the interrupt it clears the hardware interrupt automatically with the ‘interrupt_ack’ signal. The interrupt service routine then decides what action to take by reading the status of the ‘half_full’ flag. If the flag is High, then it indicates the buffer has at least 8 characters waiting to be read and so it immediately transmits and XOFF character on the UART transmitter. If the flag is Low, then it indicates the buffer has started to empty and it is able to immediately send an XON character to restore the data flow from the PC.

Note: Although the design includes soft flow control, it is not a comprehensive solution and should only be used as a starting point for other designs. In particular the response to XON/XOFF command characters received from the PC is handled entirely in software and is rather crude at this time.