



# MultiBoot Control

## (Using ICAP in an XC3S700A)



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

**PicoBlaze™**

Ken Chapman  
Xilinx Ltd

Rev1 – 11<sup>th</sup> December 2007



# Limitations

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

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

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

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

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

*In memory of Poppy and Lily.  
My friends 1995 -2007*

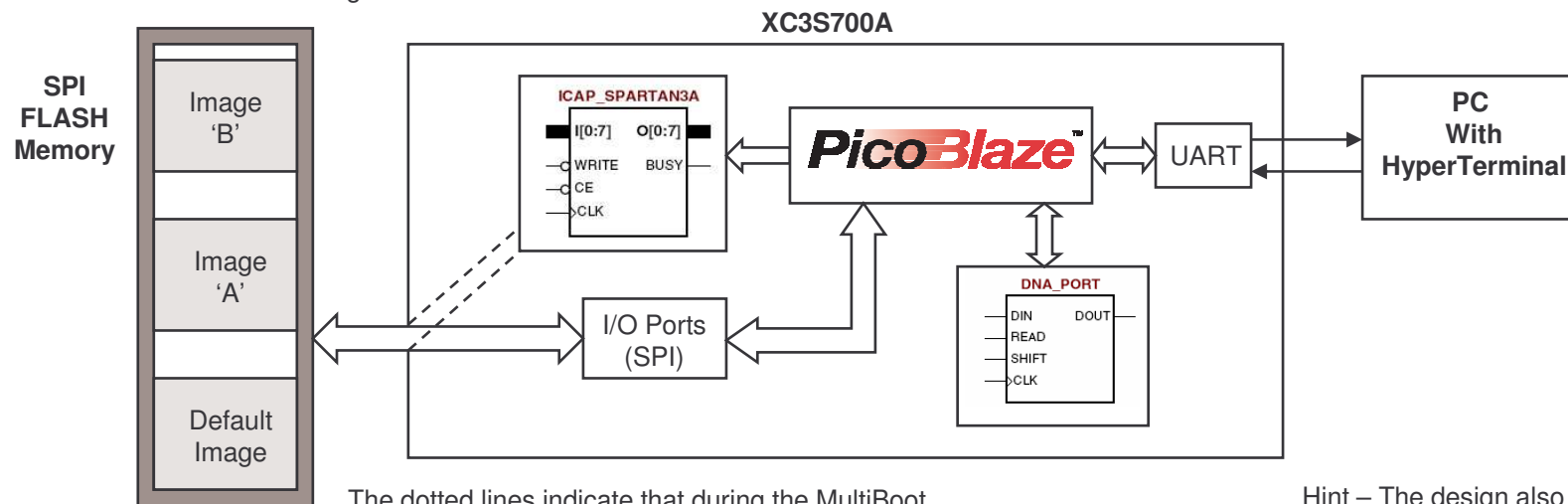


# Design Overview

This design will enable you to experiment with the MultiBoot capability of the Spartan-3A devices. Each Spartan-3A device, including the XC3S700A provided on the Spartan-3A Starter Kit, contains an Internal Configuration Access Port (ICAP) which can be used to initiate reconfiguration of the FPGA from a specified location of a specified FLASH memory. Providing the FLASH memory is of an adequate size, it is possible for several configuration images to be stored and for each design to use ICAP to initiate a reconfiguration to any other image which gives rise to the term 'MultiBoot'.

Under normal operational circumstances (i.e. in a real product) the Spartan-3A FPGA will typically configure automatically from FLASH memory using one of the master configuration modes as defined by the mode select pins M[2:0]. The initial configuration read by the FPGA will be the configuration image located at the base of the specified memory (address zero). Once that configuration has loaded and becomes active, it is then possible for that design to initiate a reconfiguration of the device by controlling the ICAP appropriately.

In this reference design the ICAP is controlled using a PicoBlaze processor and the Atmel SPI FLASH memory used to store the configuration images. Although the actual control of ICAP to initiate a reconfiguration is a relatively simple task in itself and there is no reason why a hardware state machine should not be used for this task, it is highly likely that a degree of 'configuration management' is desirable and that is when a small processor becomes very useful. In this reference design PicoBlaze presents you with a simple menu of commands on a PC HyperTerminal (or similar). You are able to use these commands to set the address at which the next configuration to be loaded is located in the SPI FLASH memory. It also enables you to confirm that the configuration image exists at that address before initiating the MultiBoot reconfiguration. In a real design PicoBlaze may be used to gather information from the system and make the decision about which configuration is most suitable. PicoBlaze may also program the FLASH memory with a new configuration delivered to it as a remote upgrade and then initiate a reconfiguration.



The dotted lines indicate that during the MultiBoot reconfiguration the FPGA's configuration state machine retakes control of the pins connected to the SPI FLASH memory.

Hint – The design also reads the Device DNA such that your own experiments could also include design authentication.

# Before You Continue.....

This design is not complex in itself but it is NOT for the novice user! It is vital that you can answer 'Yes' to all the following questions before continuing with this reference design.



I am familiar with the Spartan-3A Starter Kit board?

☐ Yes

☐ No

I have successfully experimented with the AT45DB161D Atmel SPI FLASH Programmer reference design including the programming of the Atmel SPI FLASH Memory with a UFP file?

☐ Yes

☐ No

In order to use the MultiBoot reference design as described in this document you will need to use the AT45DB161D Atmel SPI FLASH Programmer to store multiple configuration images in the memory and to correctly determine the addresses at which they are located.

[http://www.xilinx.com/products/boards/s3astarter/reference\\_designs.htm](http://www.xilinx.com/products/boards/s3astarter/reference_designs.htm)

My Spartan-3A Starter Kit and PC are connected to use the AT45DB161D Atmel SPI FLASH Programmer reference design with a baud rate of 115,200 and everything is working correctly?

☐ Yes

☐ No

Your board and PC need to be set up in exactly the same way for this MultiBoot reference design. The baud rate of the MultiBoot design is fixed to 115,200 baud. Although you can later modify the baud rate by changing the VHDL the initial configuration images have been generated to the 115,200 baud rate.

## Spartan-3 Generation Configuration User Guide

*Spartan™-3A, Spartan-3AN, Spartan-3A DSP, Spartan-3E, and Spartan-3 FPGA Families*

UG332 (v1.3) November 21, 2007

I have read the documentation for the Atmel programmer design and I am comfortable with the meaning of pages and 24-bit addresses within the FLASH memory?

☐ Yes

☐ No

I have a copy of the UG332 (see left) and I have read (at least quickly!) the sections describing 'ICAP' and 'Spartan-3A/3AN/3A DSP MultiBoot'?

☐ Yes

☐ No

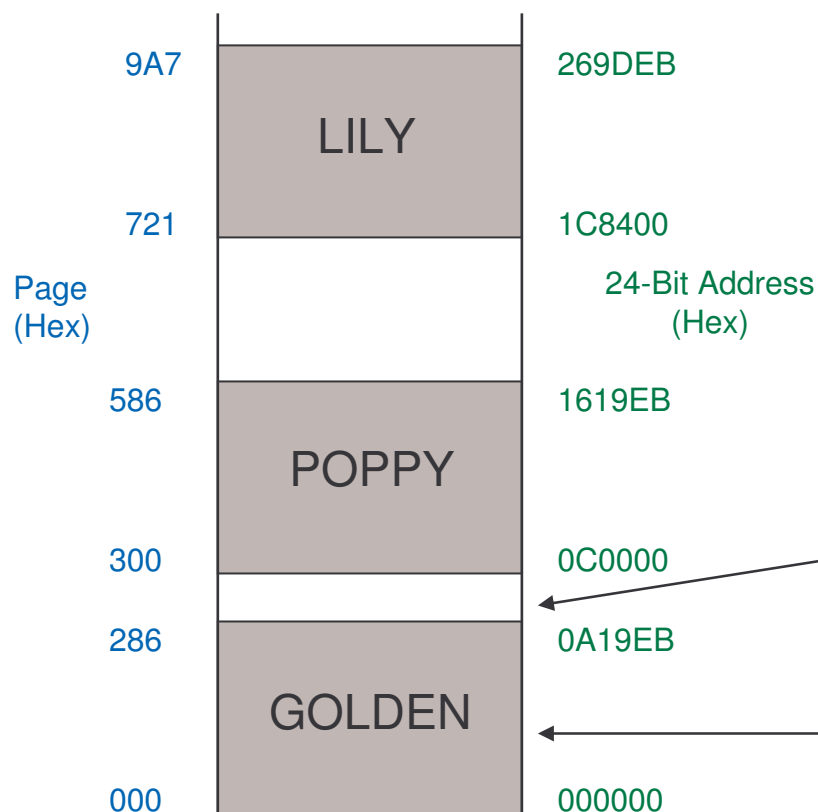
If you have answered 'Yes' 5 times then continue.....

# FLASH Programming

In order to conduct any meaningful experiments with MultiBoot you really need to have more than one configuration stored in the FLASH memory. Therefore your first task is to program the AT45DB161D Atmel SPI FLASH Programmer with some configuration images. Provided with this reference design are three configuration images for the XC3S700A device on the Starter Kit board.

multiboot\_control\_GOLDEN.ufp  
multiboot\_control\_POPPY.ufp  
multiboot\_control\_LILY.ufp

These configuration images are identical in all respect apart from the name of the file and the corresponding name which each will display on your PC terminal. The design is the MultiBoot controller described by this document and the name displayed on the terminal is defined in the PicoBlaze program. The variants were generated using DATA2MEM to modify the program directly in the configuration BIT file (this is described in more detail in the PicoBlaze download should you want to modify the PicoBlaze program further yourself).



Use the 'AT45DB161D Atmel SPI FLASH Programmer' reference design to program the FLASH with the configuration images. You are free to store the images in any order and at virtually any addresses but the following pages of this document will assume that you have programmed the memory as shown in this memory map and it is recommended that you replicate this set up before your own experiments.

When programming a configuration image (UFP file) into the FLASH, the AT45DB161D Atmel SPI FLASH Programmer prompts you to enter the page number and the byte address defining the start position. It will always tell you the final 24-bit address programmed and you can use the page read command to determine the 24-bit start address corresponding to the first page.

WRITE DOWN the 24-bit start address for each configuration image. This information is vital for MultiBoot operation.

Hint – Obviously the configuration images must not overlap but there are some practical reasons for leaving some additional space between images. These include the 'page' and 'sector' organisation of the FLASH memory. Please be sure to read the section 'Required Data Spacing between MultiBoot Images' in UG332 before implementing a MultiBoot scheme in a real design.

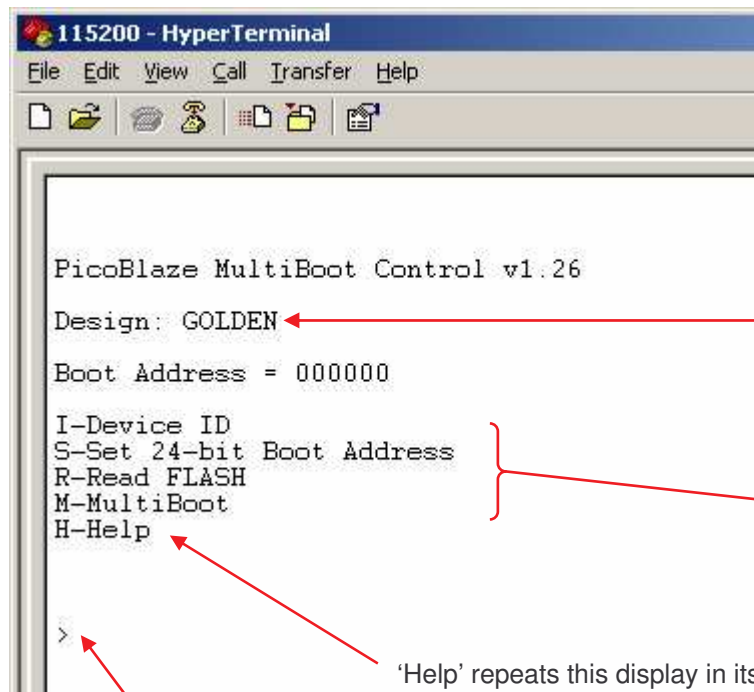
**IMPORTANT** – When Power is applied to the Spartan-3A device, the PROG\_B pin is cycled Low or a configuration error occurs the device will always attempt to configure from the image located at address zero. It is therefore vital that you have a *good* configuration image stored at this location. Due to the importance of this image, we refer to it as the 'Golden Image'.



# MultiBoot Menu

Once you have programmed the Atmel SPI FLASH memory with the configuration images, cycle the power supply or press PROG\_B such that the Spartan device configures with the MutliBoot Control design located at address zero.

The subsequent descriptions contained in this document will assume you have programmed the FLASH memory with the three UFP files as shown on the previous page. It is highly recommended that you do replicate this set up before embarking on your own experiments as this should provide you with a known to be good working environment and therefore help you learn what how MultiBoot is implemented.



```
115200 - HyperTerminal
File Edit View Call Transfer Help

PicoBlaze MultiBoot Control v1.26
Design: GOLDEN
Boot Address = 000000
I-Device ID
S-Set 24-bit Boot Address
R-Read FLASH
M-MultiBoot
H-Help
>
```

Assuming you have been using the 'AT45DB161D Atmel SPI FLASH Programmer' and have programmed the FLASH with the configuration images then your PC terminal should display the simple MultiBoot introduction and menu shown here.

The name displayed here indicates which configuration image has been loaded into the Spartan-3A device. In a real system it is likely that a version number or similar would be used to track progress. Note that this name is part of the PicoBlaze program and therefore part of the active design, i.e. it has not been read separately from anywhere else in the FLASH memory.

The four main commands are described in detail on the following pages. These allow you to manually control the MultiBoot sequence and **it is highly recommended that any real MultiBoot design implements and performs the same operations.**

'Help' repeats this display in its entirety.

Enter commands at the prompt.  
Upper and Lower case letters are accepted.

# I – Device ID Command

The ID command displays the identity information from the FLASH memory and the Spartan-3A FPGA. Although not strictly part of a MultiBoot sequence, it is highly likely that the FLASH memory will be accessed by a design to 'manage' configuration images. This may involve an upgrade of a configuration image or the selections of a suitable image depending on the situation the product finds itself in. Although not implemented directly in this or the 'AT45DB161D Atmel SPI FLASH Programmer' design, I hope it can be seen that it is possible to use some part of the FLASH memory to implement a simple form of File Allocation Table (FAT) or directory indicating which configuration images are stored in the FLASH, their locations, versions and if they are verified.

```
>i  
  
FLASH ID = 1F 26 00  
  
FLASH Security =  
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  
06 0B 15 0B 1C 00 1F 26 00 00 2F 0E FF FF 18 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  
  
Device DNA = 01131209165C8941  
  
>
```

Before performing any action with the FLASH memory (including a MultiBoot reconfiguration) it is very reassuring to know that communication with the FLASH memory is possible. The best way to determine this is to read the Manufacturer Code (1F hex for Atmel) and the density Device Code (26 00 hex for AT45DB161D). It actually quite common for the memory to respond 'FF FF FF' during a first attempt to read *immediately* after a Spartan device is configured. This is caused by the transitions of the SPI signals as the configuration read operation completes and the active design takes over. Reading the manufacture code ensures that the SPI signals are in the correct state and that both the FLASH memory and FPGA design are 'synchronised'. In this design, PicoBlaze automatically checks the manufacture code on start up so you would have to remove the jumper from 'J1' to see anything other than '1F 26 00' displayed.

The ID command goes on to read the security register of the Atmel FLASH device. The first 64 bytes of the security register are user one time programmable (OTP) and the second 64 bytes contain a read only unique factory programmed value. This can be very useful for product tracking especially when performing field upgrades or product registration.

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 used as a seed in design authentication.

Hint – Combine the unique FLASH security value with the unique device DNA to improve authentication security algorithms and device tracking 'serial numbers'.

# S – Set 24-bit Boot Address

```
PicoBlaze MultiBoot Control v1.26
```

```
Design: GOLDEN
```

```
Boot Address = 000000
```

```
I-Device ID
```

```
S-Set 24-bit Boot Address
```

```
R-Read FLASH
```

```
M-MultiBoot
```

```
H-Help
```

```
>s
```

```
Boot Address = 1c8400
```

```
OK
```

```
PicoBlaze MultiBoot Control v1.26
```

```
Design: GOLDEN
```

```
Boot Address = 1c8400
```

```
I-Device ID
```

```
S-Set 24-bit Boot Address
```

```
R-Read FLASH
```

```
M-MultiBoot
```

```
H-Help
```

```
>
```

The key to MultiBoot is to specify the start address of the next configuration image to be loaded into the Spartan-3A. In this reference design we do that manually, but hopefully you can easily imagine all kinds of schemes in which this process is automated. The simplest scheme would be one in which there is the 'golden image' and only one more image stored in the FLASH memory. By always locating the second image the same predefined location the 24-bit address is known from the outset.

MultiBoot requires a 24-bit value to define the start address of the next configuration image. The design displays the current 24-bit address as a 6-digit hexadecimal value. This reference design defaults to the address zero as this corresponds to the 'Golden image' which would be safe to use immediately in a MultiBoot operation.

Use the 'S' command to specify a new 24-bit address corresponding with the start of a configuration image. You will be prompted to enter the new address and you should then type in a 6-digit hexadecimal value. You can use upper or lower case characters but entering any non-hexadecimal characters will result in the 'Boot Address =' prompt being repeated and you must enter the address correctly from the beginning.

In this example, I have specified the address **1c8400** hex which is the start of the image 'LILY'.

Once you have entered the 6<sup>th</sup> hexadecimal digit to complete the address, your new address value will be displayed on the terminal.

You can set the address as many times as you like but remember that the objective is to define the start address for the next configuration image to be loaded into the Spartan-3A FPGA.



# R – Read FLASH

The Read FLASH command reads and displays 256 bytes contained in the FLASH memory starting at the currently defined 24-bit Boot Address. This enables you to confirm that there really is a configuration image stored at that address before initiating the MultiBoot attempt.

In this case **1C8400** is the start of image 'LILY'

At the start of the configuration (or MultiBoot reconfiguration) process, the Spartan-3A reads the memory serially looking for a **'synchronisation word'**. This word is stored as 'AA 99' hex when read as bytes. Therefore the easy way to confirm that the start of a configuration image is located at the specified 24-bit address is to observe the series of 'FF' bytes following by the synchronisation word 'AA 99'.

[illegible]

- A configuration imaged starts with a series of FF bytes so anything else here is suspicious!

**Recommendation** – Before initiating a MultiBoot reconfiguration it is a very good idea confirm that the specified address really does correspond with what appears to be a valid configuration image for the Spartan-3A FPGA. Although we will see later that the configuration state machine of the Spartan-3A includes protection against configuration failures due to corruption of an otherwise good configuration image (which may happen during a field upgrade) it is really not sensible to initiate a MultiBoot configuration at an address which contains an obviously invalid configuration image. By reading the first few bytes of memory and checking that the ‘FF’ bytes are followed by ‘AA 99’ is a reasonable way to validate that a configuration image will be found during MultiBoot.

# M – MultiBoot Command

PicoBlaze MultiBoot Control v1.26

Design: GOLDEN

Boot Address = 1C8400

I-Device ID

S-Set 24-bit Boot Address

R-Read FLASH

M-MultiBoot

H-Help

>m

Booting

PicoBlaze MultiBoot Control v1.26

Design: LILY

Boot Address = 000000

I-Device ID

S-Set 24-bit Boot Address

R-Read FLASH

M-MultiBoot

H-Help

>

Once you are happy that you have specified the correct start address for the next configuration image you are ready to initiate a MultiBoot reconfiguration.

In this case **1C8400** is the start of image 'LILY'

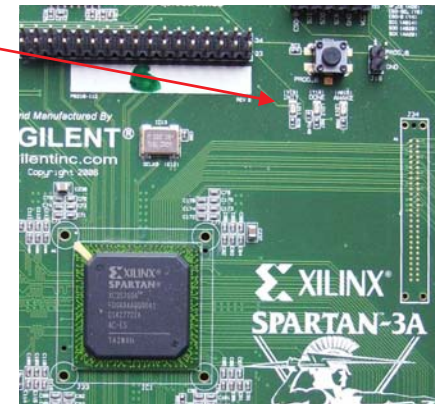
Enter the 'M' command and PicoBlaze will report 'Booting' before it writes to the ICAP port to initiate a MultiBoot reconfiguration from the currently defined 'Boot Address'.

Hint – Observe the 'INIT' and 'DONE' LED's as you enter the 'M' command to see the FPGA initialise (clear) and then reconfigure. Don't blink; it happens pretty fast!

Note that the display of 'Booting' is the last action performed by the current design (GOLDEN). After this the FPGA is initialised (cleared) ready for the next configuration.

Hint - If for any reason ICAP does not trigger a reconfiguration then the current PicoBlaze design will remain active and the program will display 'Failed!'. This really should not occur but it is good design practice to cover for the unexpected!

The name LILY indicates that the Spartan-3A is now configured with the configuration image located at address 1C8400. Since this is a virtually identical copy of the MultiBoot Control design you can repeat the operations to MultiBoot to another image as you wish (e.g. see next page).



# MultiBoot from LILY to POPPY

```
PicoBlaze MultiBoot Control v1.26

Design: LILY

Boot Address = 000000

I-Device ID
S-Set 24-bit Boot Address
R-Read FLASH
M-MultiBoot
H-Help

>>i

FLASH ID = 1F 26 00

FLASH 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 0B 15 0B 1C 00 1F 26 00 00 2F 0E FF FF 18 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

Device DNA = 01131209165C8941

>s
Boot Address = 0c0000
OK

PicoBlaze MultiBoot Control v1.26

Design: LILY

Boot Address = 0C0000

I-Device ID
S-Set 24-bit Boot Address
R-Read FLASH
M-MultiBoot
H-Help
```

This command sequence shows the MultiBoot from LILY to POPPY. Note the recommended sequence of checks being performed before the final MultiBoot is initiated.

Current active design is 'LILY'

Test FLASH communication

Specify boot address

0C0000 = 'POPPY'

```
>r
Address = 0C0000

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
AA 99 30 A1 00 07 20 00 31 61 89 EE 33 21 3C 0F
31 A1 00 C9 31 41 2F 00 31 C2 02 22 80 93 30 E1
FF CF 30 C1 00 85 31 81 08 81 32 01 00 1F 32 C1
00 05 32 E1 00 04 32 A1 00 0E 32 61 00 00 32 81
00 00 33 41 18 F2 33 62 00 00 00 00 30 22 00 00
00 00 30 A1 00 01 50 60 00 02 9A C2 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

>m
Booting

PicoBlaze MultiBoot Control v1.26

Design: POPPY

Boot Address = 000000

I-Device ID
S-Set 24-bit Boot Address
R-Read FLASH
M-MultiBoot
H-Help

>
```

Check that configuration is present  
FF FF...AA 99

MultiBoot

Active design is now 'POPPY'

# Automatic Fallback Error Recovery

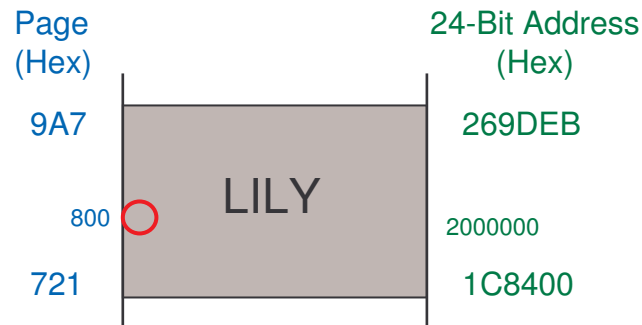
As a Spartan-3A device configures it computes a CRC value for the image being loaded. This is automatically compared with the CRC value contained in the configuration image and only if the two are identical will the device enter the start-up sequence and allow the design to become active.

In the CRC values do not match then the device will abort the configuration. Should this occur, two things can happen depending on how the configuration image was prepared. The default setting of the ISE tools is for the INIT\_B pin to be driven Low and for everything to stop until either the power supply or PROB\_B is cycled. The alternative is to enable the Spartan-3A to automatically initiate the equivalent of a MultiBoot using the image stored at address zero (the 'Golden image'). Clearly this is a valuable feature when performing MultiBoot operations especially when configuration images are upgraded in the field and have the potential to be corrupted during transmission or by an imperfection of the FLASH memory.

Page 14 illustrates how to configure ISE to enable the automatic fallback mode, but the GOLDEN, LILY and POPPY images were prepared with this option and are ready for you to try out this feature.

## Break it!

To evaluate and experiment with the automatic fallback feature, we will now deliberately corrupt one of the MultiBoot configuration images but NOT the GOLDEN one! (you would always preserve this image in a product after final production testing). In this example I have altered one byte in the image 'LILY' using the 'AT45DB161D Atmel SPI FLASH Programmer' reference design.



PicoBlaze AT45DB161D Programmer v1.21

Part of good configuration in page 800 hex

```
>r
Page = 800
200000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
200010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
200020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
>w
Page = 800
byte address = 000

Data = 42
Full address = 200000
```

Corrupt (damage) the configuration image by modifying a byte.

Configuration image is now invalid due to 'error' in page 800 hex

```
>r
Page = 800
200000 42 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
200010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
200020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Once you have corrupted a configuration image, press the PROG\_B button to reload the GOLDEN MultiBoot configuration design.....

# Automatic Fallback Error Recovery

You are free to experiment as much as you like. You should find that it is still possible to MultiBoot successfully between 'GOLDEN' at address 000000 and 'POPPY' at address 1C0000 without any problems. But then try a MultiBoot to the now corrupted image 'LILY' at address 1C8400 and you should see that you end up back at 'GOLDEN' as shown below.

```
PicoBlaze MultiBoot Control v1.26

Design: GOLDEN

Boot Address = 1C8400

I-Device ID
S-Set 24-bit Boot Address
R-Read FLASH
M-MultiBoot
H-Help

>m
Booting

PicoBlaze MultiBoot Control v1.26

Design: GOLDEN

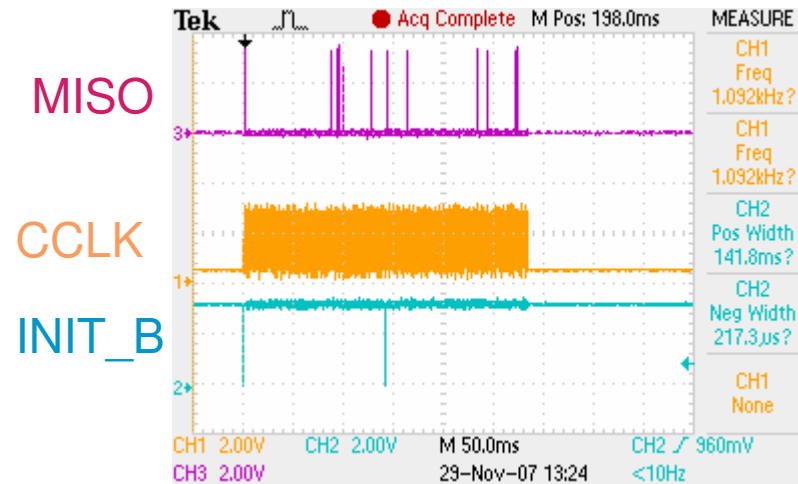
Boot Address = 000000

I-Device ID
S-Set 24-bit Boot Address
R-Read FLASH
M-MultiBoot
H-Help
```

Hint - Look very carefully at the INIT LED and you should see it blink twice; once for the clear before loading LILY and then again before loading GOLDEN.

1C8400 = 'LILY'

This oscilloscope screen shot shows a failed attempt to configure from the corrupted image LILY at address 1C8400. Note the two Low pulses on the INIT\_B pin.



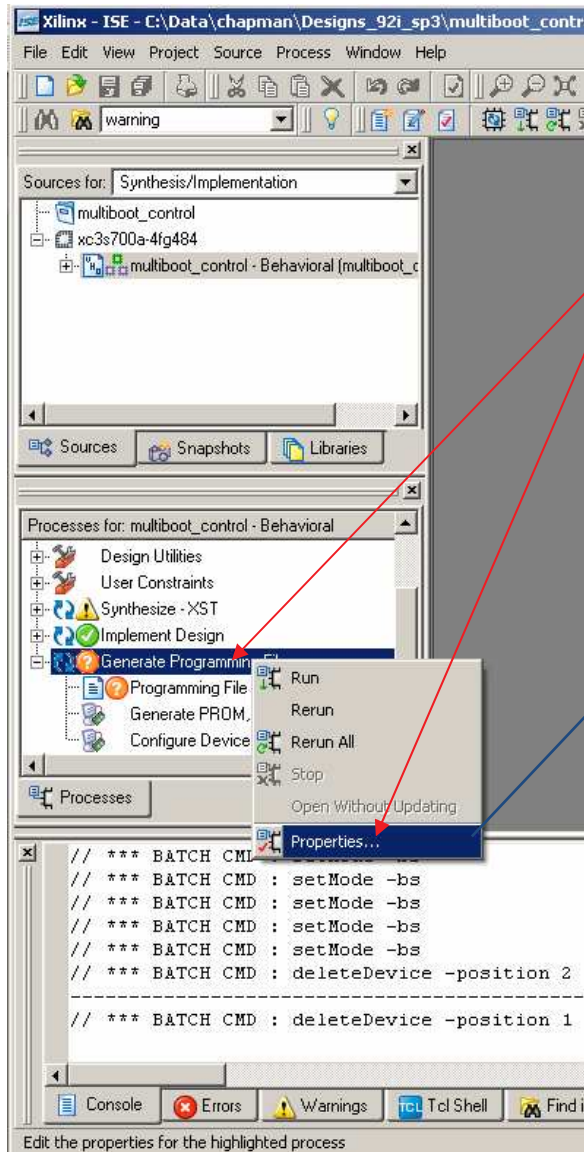
GOLDEN image loaded due to corrupted image.

000000 = 'GOLDEN'

The time between INIT\_B pulses is measured to be 141.8ms and is the time this device is taking to load one configuration image (ConfigRate=25).

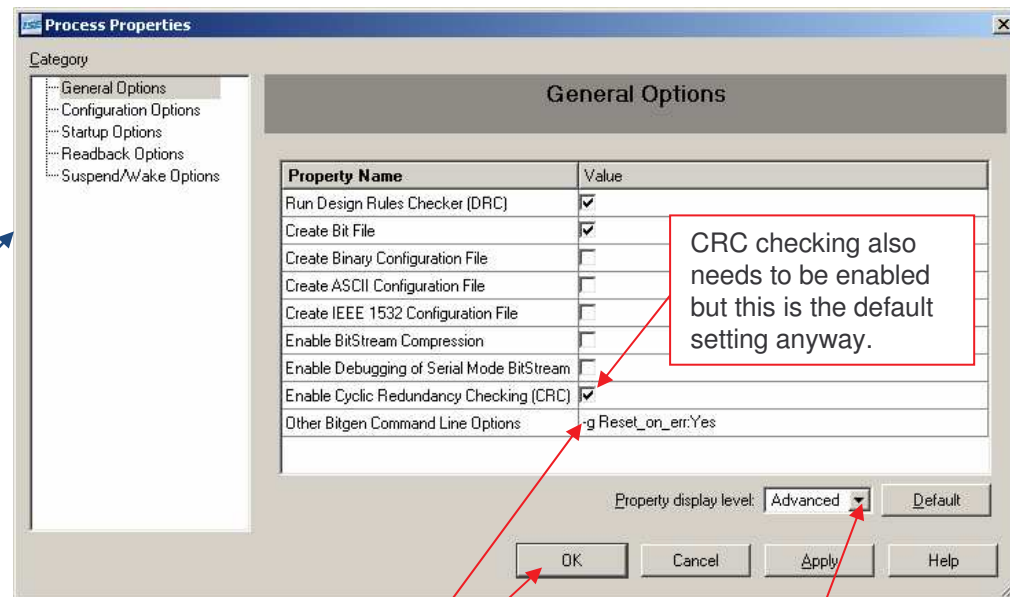
Hint – You can only perform three automatic fall back reconfigurations before INIT\_B becomes permanently Low (INIT LED stays on) and everything stops. This is to prevent the 'endless loop' scenario (see UG332 for details). A power or PROG\_B cycle resets this counter. Prove that this happens by performing three MultiBoot attempts to the now corrupted image 'LILY'.

# Automatic Fallback Mode



The screen shots on this page are taken from ISE v9.2i (with service pack 3) and show how to enable the automatic fallback mode when generating a configuration image.

Right click on 'Generate Programming File' and select 'Properties...'



CRC checking also needs to be enabled but this is the default setting anyway.

Hint – You may need to select 'Advanced' in order to reveal the 'Other Bitgen Command Line Options' box.

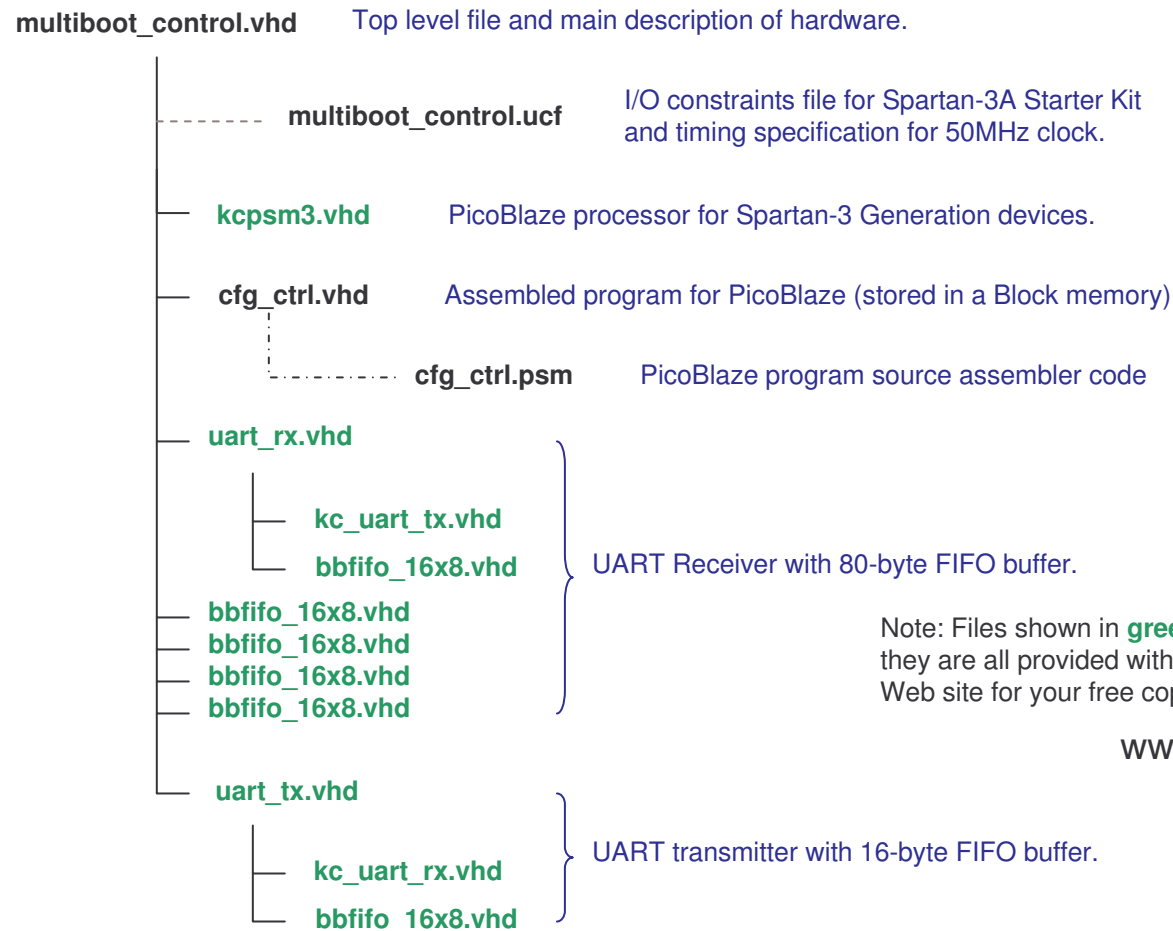
In the 'Other Bitgen Command Line Options' box enter the text string '**g Reset\_on\_err:Yes**' and then click OK.



# 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. As well as having UG332 and the 'AT45DB161D Atmel SPI FLASH Programmer' reference design available, you may also find it useful to have a copy of the Atmel data sheet for the AT45DB161D device.

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



**Note** – The descriptions contained in this document focus on ICAP and MultiBoot. Please use the other reference designs to learn more about other features such as reading SPI FLASH memory and device DNA and.

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](http://www.xilinx.com/picoblaze)

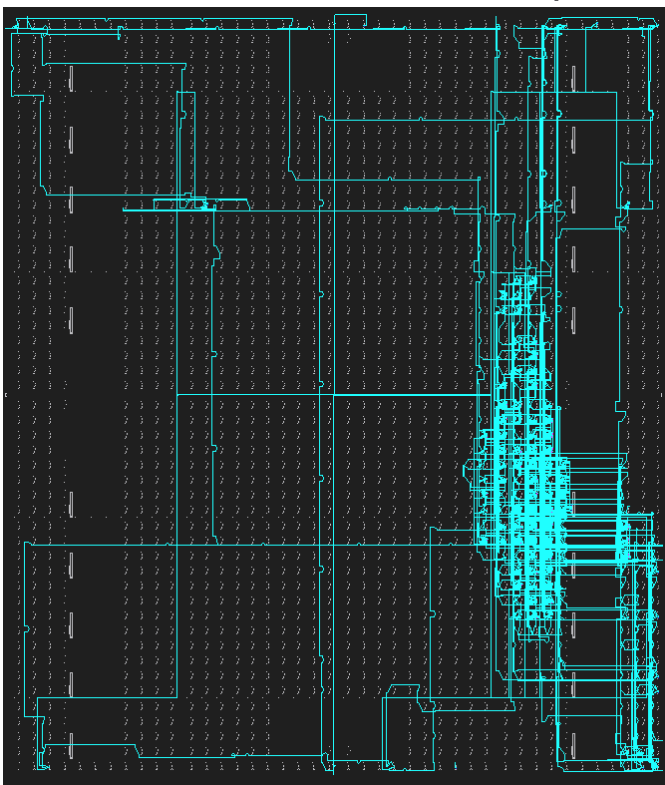
# PicoBlaze Design Size

This reference design occupies less than 5% of the XC3S700A device and indicates that a MultiBoot scheme is a very practical proposition in every density of Spartan-3A and Spartan-3AN device. The PicoBlaze program occupies approximately 70% of the single Block RAM (RAMB16BWE). However, nearly 30% of this program is consumed by the text strings displayed on the terminal so a real application would be able to perform many more functions with the PicoBlaze processor than those presented in this simple reference design.

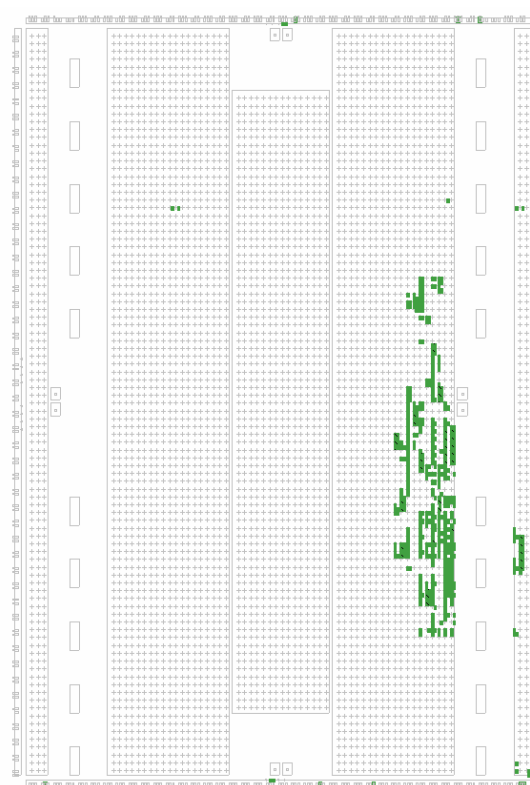
## MAP report

Number of occupied Slices:	204	out of	5,888	3%
Number of RAMB16BWEs:	1	out of	20	5%

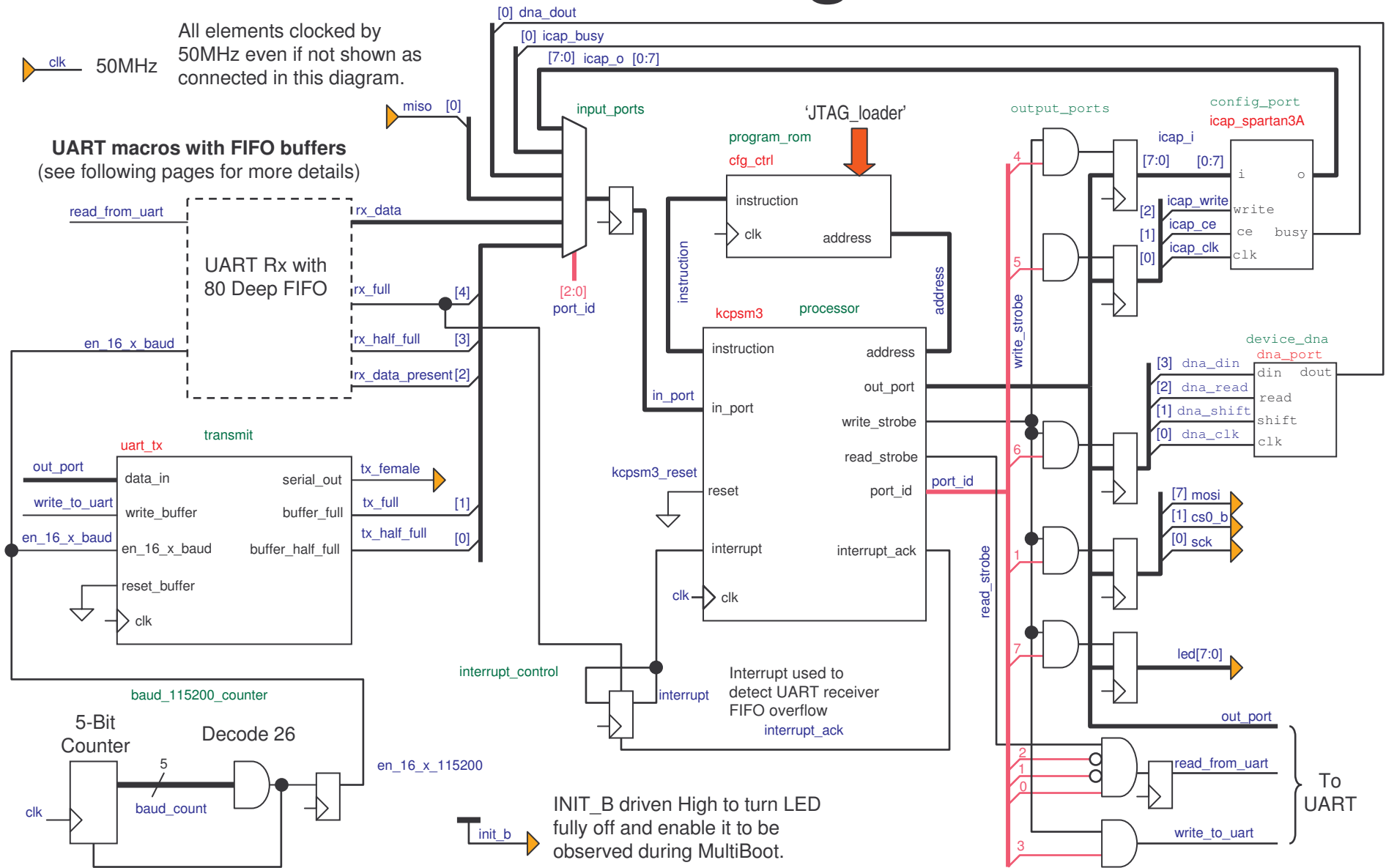
FPGA Editor view



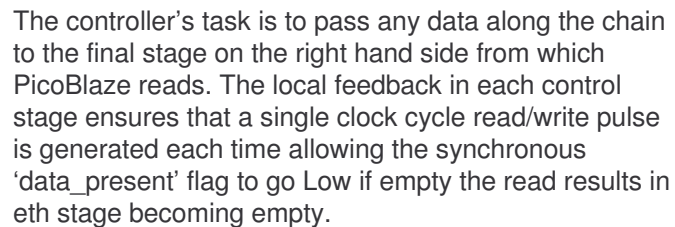
Floorplanner view



# PicoBlaze Circuit Diagram

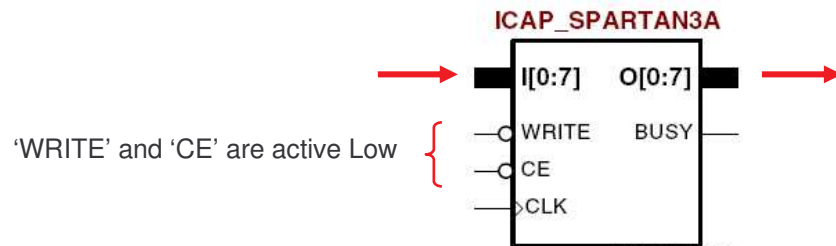


**Hint** – The only reason this reference design has the extended FIFO buffer is to be similar to the ‘AT45DB161D Atmel SPI FLASH Programmer’ reference design which does need it. By providing this hardware design it is hoped that it will facilitate you in your own experiments. An experimental field upgrade scheme would be expected to combine elements of both reference designs as it would enable a new configuration image to be downloaded and stored in the FLASH memory, checked (or verified) and then the MultiBoot initiated.



# Connecting ICAP

The internal Configuration Access Port (ICAP) primitive must be included in a design in order to perform MultiBoot reconfiguration. The primitive is a simple synchronous read/write module with 8-bit data ports. Please see the library guide and UG332 for more details. The following design notes combined with the schematic on page 17 and the source designs files provided with this reference design are intended to provide supplementary advice and design hints.



Be very careful connecting the 8-bit data ports as they have been defined with bit0 as the most significant bit. This reference design reverses the bit order as the ICAP ports are connected to the PicoBlaze I/O ports such that the assembler code is more 'natural'. This bit reversal has no cost or other implications but clearly there is potential for confusion if implementing your own design from scratch.

The physical connection to PicoBlaze is reflected in the CONSTANT directives within the 'cfg\_ctrl.psm' assembler program file. The corresponding port addresses have been assigned names as have the allocation of the bits on the signal ports. This makes the subsequent code easier to develop, read and maintain as well as providing you with code which should be easier to reuse in your own MultiBoot designs.

```
CONSTANT ICAP_i_port, 10          ; 8-bit data from PicoBlaze to ICAP
CONSTANT ICAP_o_port, 04          ; 8-bit data from ICAP to PicoBlaze
;
CONSTANT ICAP_control_port, 20    ; Control signals for ICAP communication
CONSTANT ICAP_clk, 01             ; CLK - bit0
CONSTANT ICAP_ce, 02              ; CE - bit1 Active Low
CONSTANT ICAP_write, 04           ; WRITE - bit2 Active Low
;
CONSTANT ICAP_status_port, 05     ; Status from ICAP to PicoBlaze
CONSTANT ICAP_busy, 01            ; BUSY - bit0
```

**Note-** The connection of the ICAP outputs 'O[0:7]' and 'BUSY' are for design completeness and to facilitate future experiments. The MultiBoot process only requires that data be written to the ICAP port, so apart from providing these connections this reference design makes no further use of them.

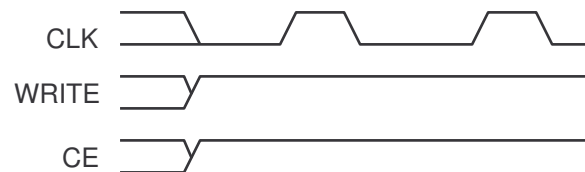
# Driving ICAP Signals

In this reference design PicoBlaze implements all signaling to the ICAP port using software. Although this is not as fast as using pure hardware it is probably always fast enough for the purpose. The following low level routines drive the ICAP control signals and allow data to be written.

```
ICAP_init: LOAD s0, 06
          OUTPUT s0, ICAP_control_port
          CALL ICAP_clock_cycle
          CALL ICAP_clock_cycle
          RETURN
```

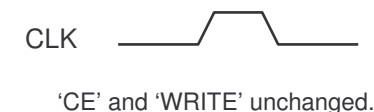
Before using the ICAP port, it is important to initialise the signals and prepare ICAP for future writing of data. The 'ICAP\_init' routine ensures that the 'WRITE' and 'CE' controls are disabled (High) and the 'CLK' is Low. It then applies two clock cycles which are required to initialise the ICAP port following configuration of the device.

Hint – The ICAP is not immediately able to receive any signals following configuration so ensure that there is a small delay before driving any ICAP signals. In normal operation the delay only needs to be the equivalent of a few CCLK cycles in order that the device fully completes the FPGA start up sequence but when using iMPACT and JTAG during development you need to allow at least 5ms for the configuration state machine to be released internally.



```
ICAP_clock_cycle: XOR s0, ICAP_clk
                  OUTPUT s0, ICAP_control_port
                  XOR s0, ICAP_clk
                  OUTPUT s0, ICAP_control_port
                  RETURN
```

The 'ICAP\_clock\_cycle' routine generates a single High level pulse to the ICAP clock input. Note that register 's0' is used in this process and the logic levels of 'CE' and 'WRITE' controls are defined by bits 1 and 2 respectively. It is vital that these bits of 's0' are defined appropriately before calling this routine.



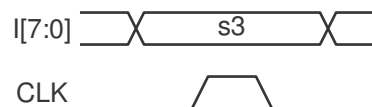
```
ICAP_write_enable: LOAD s0, 00
                   OUTPUT s0, ICAP_control_port
                   RETURN
```

The 'ICAP\_write\_enable' routine drives 'CE' and 'WRITE' into their active Low states ready to write data. 'CLK' is confirmed Low.



The 'ICAP\_write' routine presents the byte of data contained in register 's3' to the ICAP input port and then applies a clock pulse to write it.

```
ICAP_write: OUTPUT s3, ICAP_i_port
            CALL ICAP_clock_cycle
            RETURN
```



'CE' and 'WRITE' unchanged and should be Low to write to ICAP.



# MultiBoot Initiation

As detailed in UG332, a sequence of 20 bytes must be written to the ICAP port in order to initiate a MultiBoot reconfiguration. It can be seen that the majority of these bytes have fixed values but a few critical bytes need further definition.

Table 14-3: Command Sequence to Initiate MultiBoot to a Specified Address

CLK Cycle	Command	High or Low Byte	D0	D1	D2	D3	D4	D5	D6	D7	Hex
1	SYNC WORD	High	1	0	1	0	1	0	1	0	0xAA
2		Low	1	0	0	1	1	0	0	1	0x99
3	Type 1 Write GENERAL1 (1 Word)	High	0	0	1	1	0	0	1	0	0x32
4		Low	0	1	1	0	0	0	0	1	0x61
5	Lower 16 bits of MultiBoot Address	High	A15	A14	A13	A12	A11	A10	A9	A8	
6		Low	A7	A6	A5	A4	A3	A2	A1	A0	
7	Type 1 Write GENERAL2 (1 Word)	High	0	0	1	1	0	0	1	0	0x32
8		Low	1	0	0	0	0	0	0	1	0x81
9	Upper bits of MultiBoot Address (SPI mode example)	High	C7	C6	C5	C4	C3	C2	C1	C0	
10		Low	A23	A22	A21	A20	A19	A18	A17	A16	
11	Type 1 Write MODE_REG (1 Word)	High	0	0	1	1	0	0	1	0	0x32
12		Low	1	0	1	0	0	0	0	1	0xA1
13	Reserved	High	0	0	0	0	0	0	0	0	0x00
14	MODE_REG Register	Low	0	NEW MODE	BOOTMODE			Reserved			
15	Type 1 Write CMD (1 Word)	High	0	0	1	1	0	0	0	0	0x30
16		Low	1	0	1	0	0	0	0	1	0xA1
17	REBOOT Command	High	0	0	0	0	0	0	0	0	0x00
18		Low	0	0	0	0	1	1	1	0	0x0E
19	No Op	High	0	0	1	0	0	0	0	0	0x20
20		Low	0	0	0	0	0	0	0	0	0x00

## 24-Bit Address

Earlier in this document (and during your experiments) you saw that the key to MultiBoot reconfiguration is the specification of the 24-bit address corresponding with the start of the next configuration to be loaded. Therefore the most important task PicoBlaze is performing is to convert the 6-digit hexadecimal that you enter into a 24-bit binary format (3 bytes) and insert that value into the 20-byte MultiBoot initiation sequence.

'03' hex defines the Continuous Array Read (Low Frequency) command for the Atmel SPI FLASH.

'4F' composed as follows:-

'0' – Always '0'

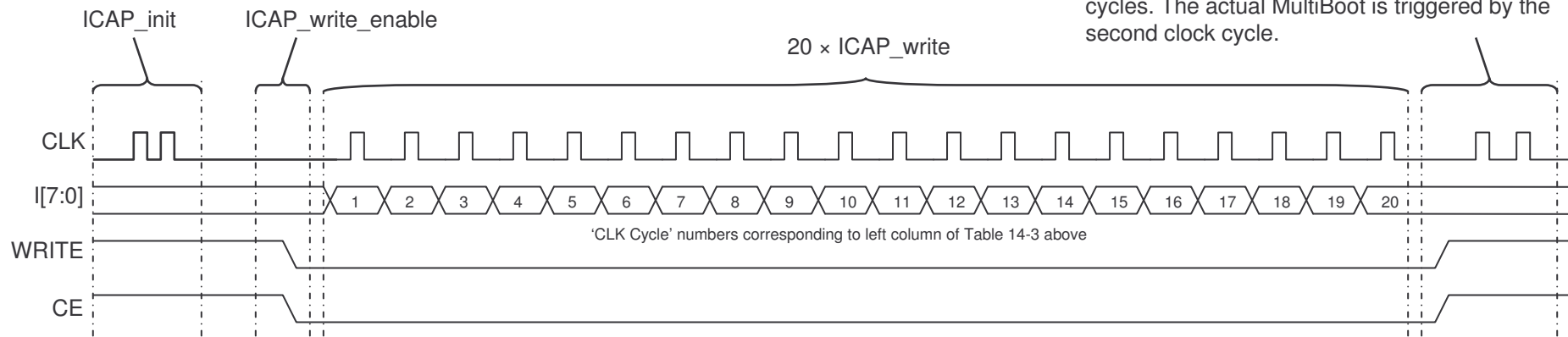
'1' – Next 3-bits define configuration mode

'001' – SPI Configuration

(Use '011' for Internal SPI in a Spartan-3AN)

'111' – Reserved and always '111'

Using 'ICAP\_init' routine completes the 20-byte write sequence but also generates two clock cycles. The actual MultiBoot is triggered by the second clock cycle.



# MultiBoot Initiation Code

The PicoBlaze code to implement the MultiBoot sequence is a simple series of subroutine calls. The 'cfg\_ctrl.psm' file contains comments but these have been removed in order to fit the entire code sequence onto this one page. A direct comparison can be made between this code and the table on the previous page.

```
MultiBoot_command: CALL send_CR
                   CALL send_Bootimg
                   CALL delay_20ms
                   ;
                   LOAD s0, LED5
                   OUTPUT s0, LED_port
                   ;
                   CALL ICAP_write_enable
                   ;
                   LOAD s3, AA
                   CALL ICAP_write
                   LOAD s3, 99
                   CALL ICAP_write
                   LOAD s3, 32
                   CALL ICAP_write
                   LOAD s3, 61
                   CALL ICAP_write
                   FETCH s3, MB_Address1
                   CALL ICAP_write
                   FETCH s3, MB_Address0
                   CALL ICAP_write
                   LOAD s3, 32
                   CALL ICAP_write
                   LOAD s3, 81
                   CALL ICAP_write
                   LOAD s3, SPI_read_command
                   CALL ICAP_write
                   FETCH s3, MB_Address2
                   CALL ICAP_write
```

The MultiBoot sequence will happen so fast that the 'Booting' message sent to the terminal would not be transmitted by the UART by the time the FPGA was initialized. This 20ms delay is long enough for the UART transmitter to transmit >200 characters which is more than enough!

Constant values are loaded into 's3' and then written to ICAP by the 'ICAP\_write' routine.

The 24-bit address has been prepared in three scratch pad memory locations and these are fetched into 's3' as required.

```
LOAD s3, 32
CALL ICAP_write
LOAD s3, A1
CALL ICAP_write
LOAD s3, 00
CALL ICAP_write
LOAD s3, 4F
CALL ICAP_write
LOAD s3, 30
CALL ICAP_write
LOAD s3, A1
CALL ICAP_write
LOAD s3, 00
CALL ICAP_write
LOAD s3, 0E
CALL ICAP_write
LOAD s3, 20
CALL ICAP_write
LOAD s3, 00
CALL ICAP_write
;
CALL ICAP_init
```