



# Low Cost Design Authentication for Spartan-3E



A Reference Design for the Spartan-3E Starter Kit

**PicoBlaze™**

Ken Chapman  
Xilinx Ltd

Rev1 – 12<sup>th</sup> December 2006



# Limitations

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

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

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

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

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

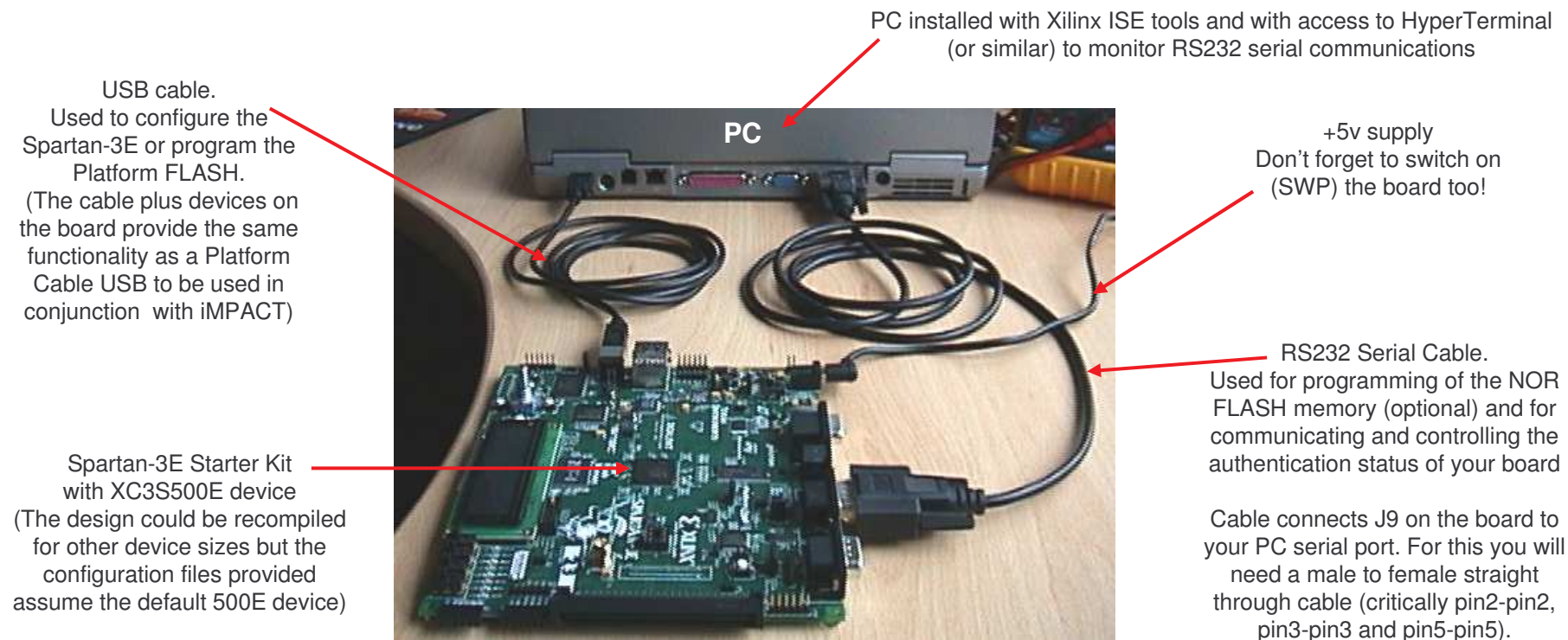
The author would also be pleased to hear from anyone using KCPSM3 or the UART macros with information about your application and how these macros have been useful.

# Design Overview

This reference design has been provided as a demonstration of anti-cloning design security using a technique known as 'Design Authentication'. The reference design also contains features which enable you evaluate the authentication procedure and it is also hoped that it will form the basis for your own experiments and commercial designs.

The design is provided for the Spartan-3E Starter Kit and employs the 8-bit PicoBlaze processor as the principle element implementing the design security. A second PicoBlaze processor is used to represent a real application which is being protected by the design authentication technique.

To use this design properly you will need the following equipment.....



# Before You Continue.....

- Are you familiar with the Spartan-3E Starter Kit board?  Yes  No
- Have you implemented a Xilinx FPGA design of your own or at least modified a existing reference design ?  Yes  No
- Have you successfully experimented with a different reference design for the Spartan-3E Starter Kit board?  Yes  No
- Have you successfully programmed at least one type of configuration FLASH memory on the Spartan-3E Starter Kit?  Yes  No
- Have you previously used the 8-bit PicoBlaze processor in a design or reviewed a reference design containing PicoBlaze?  Yes  No
- Have you previously establishing an RS232 communication between the Starter Kit board and your PC using HyperTerminal?  Yes  No

If you have answered 'No' to any of the above questions then it is highly recommended that you do not continue with this particular reference design until you have the required experience. This document is not written for the novice user and assumes that you are competent in the use of the Spartan-3E Starter Kit as well as fundamental FPGA design techniques.

Reference Designs you are recommended to download, study and try are....

**Pulse Width Modulation (PWM) Generation and Control with PicoBlaze**  
**PicoBlaze RS-232 StrataFlash™ Programmer**

These can be downloaded from..... [http://www.xilinx.com/products/boards/s3estarter/reference\\_designs.htm](http://www.xilinx.com/products/boards/s3estarter/reference_designs.htm)

# Ready for More?

So now I assume you are familiar with Spartan devices and the Spartan-3E Starter Kit and are suitably prepared to investigate design authentication security techniques. This document will attempt to use the reference design to introduce to you the fundamental concepts as well as highlight the factors which you should consider when going on to implement design security in your own products. This document will take you on a short 'journey'. Obviously you are free to take a different route, but I would ask you to resist that temptation because you may miss some vital concepts and recommendations if you do.

## Coming next.....

### **See your clone fail to work!**

In the first stage you will configure your board with the configuration image provided. Your Starter Kit is essentially a hardware clone of the Starter Kit that I have and now I am giving you the same configuration image. My board works but yours will not!

### **Theory**

Sorry about this, but there is some reading to do next. I will try to explain the basic concept being used to prevent your clone of my design from working. I hope this theory is made more interesting by observing the information the reference design provides as it is carrying out the authentication procedure.

### **Making authorised products**

You will be able to validate your Starter Kit and make it an authorised copy which will then work correctly. Likewise you will be able to return it to the original unauthorised state.

### **Attack!**

Although by this stage you will have already learnt things about my authentication design that would normally be hidden, I want to give you the opportunity to see if you can make your board work without using the obvious facility I have provided. At this point I will also tell you the potential weaknesses of the technique that a real attacker would be looking to exploit.

### **Revealing my Secrets**

Finally I will walk you through my design in more detail revealing how it works and what measures I have taken to make it difficult to bypass the security.

# Create Your Cloned Product

The reference design is supplied with a configuration image for you to program into the FLASH memory on your Spartan-3E Starter Kit board. Select one of the following options to complete the copy of the board and design that I have.

Configuration file provided: **low\_cost\_design\_authentication\_for\_spartan\_3e.mcs**

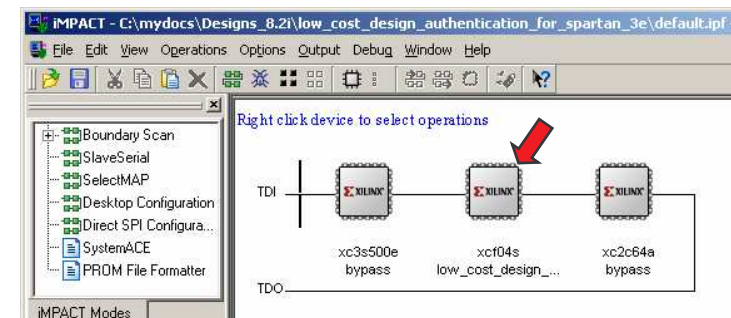
## Option 1 – Program the Intel StrataFLASH Parallel NOR FLASH (IC22)

This option is the most realistic because you will see later that it is the Intel FLASH device that is also playing a critical role in the design security. You can use the NOR FLASH Programmer reference design to enable you to do this. I recommend that you erase the whole device before programming with the above file. Once programmed, set the jumpers on J30 for BPI-UP mode.



## Option 2 – Program the Xilinx platformFLASH (IC11)

This is less realistic but still adequate for the purposes of demonstration and evaluation. The PlatformFLASH can be programmed directly from iMPACT using the USB cable so it is the fast and straightforward option. Once programmed, set the jumpers on J30 for Master Serial (M.S) mode.



## Option 3 – Program the ST Microelectronics SPI FLASH (IC4)

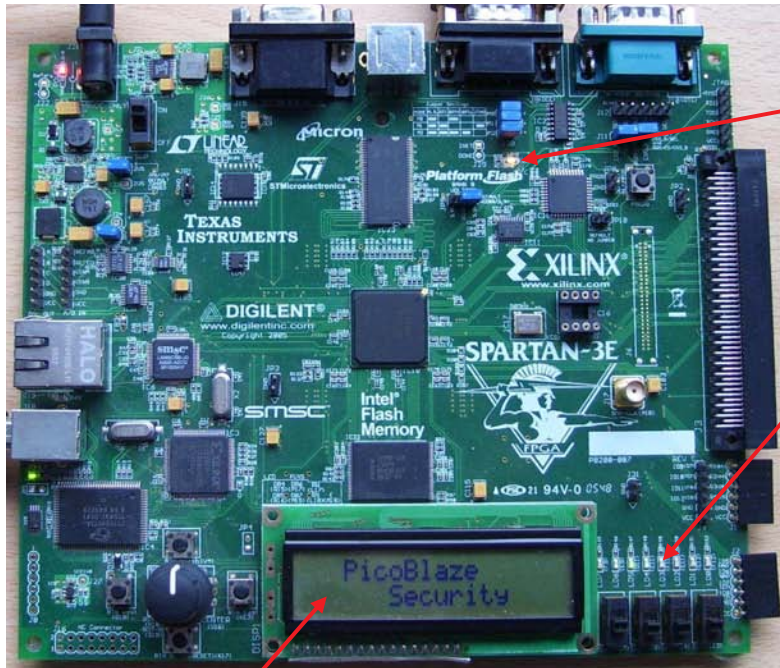
To be honest there is not many reasons to take this option, but if you want to, then go ahead and set the jumpers on J30 appropriately.

## Option 4 – Volatile

If you really do not want to program any of the FLASH devices on the board then you can download the provided configuration BIT file 'low\_cost\_design\_authentication\_for\_spartan\_3e.bit' directly into the Spartan-3E device. However, during your evaluations you will be cycling the power or reconfiguring the Spartan device many times and you will soon find having a FLASH programmed is easier, faster and far more realistic.

# Does Your Clone Work?

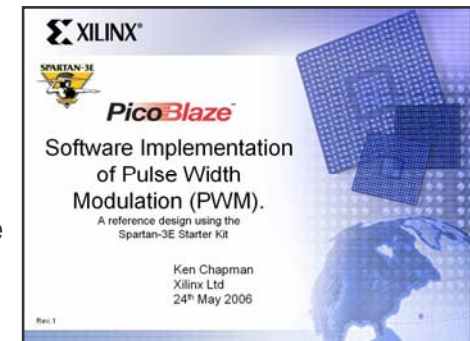
Assuming you have programmed a FLASH memory with the configuration image, you should be able to disconnect the USB cable and use the board standalone as if it were a real product. At this stage I would also like you to disconnect any RS232 Serial cables and/or close any HyperTerminal session on your PC. Then cycle the power or press PROG button.



You should initially see the following happen.....

XC-DONE LED should turn on indicating that the Spartan device does configure from the image programmed in the FLASH. Your product cloning task is complete!

The 8 general purpose LEDs should modulate left to right. These LEDs are controlled by a PicoBlaze processor and are being driven using 8-bit PWM with a PRF of 1KHz. You can find more details about this in the reference design shown to the right.



**NOTICE:** For the purposes of this reference design, it is this drive pattern of the LEDs which is representing the real application to be protected by the design authentication security.

This LCD display is being controlled by a second PicoBlaze processor which is responsible for design security. This 'welcome' message will be displayed for 10 seconds.

**For the first 10 seconds after configuration your cloned product will indeed be working 100%.** This was planned behaviour and illustrates how products can be fully tested in production even before they are authorised. This could also be the basis of a scheme in which your customers could have limited 'try before you buy' periods to evaluate optional upgrades. Later you will read that this behaviour is also part of the security scheme.

# It Worked, but now watch it Fail!

After the first 10 seconds of normal operation (LEDs nicely modulated from side to side), the design authentication security starts to take effect and you should observe the following.....



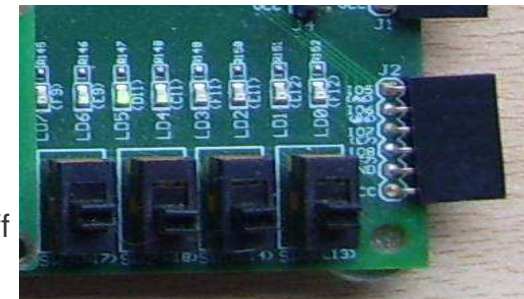
The security focused PicoBlaze will read and display on the LCD the 64-bit unique serial number which is provided in the Intel FLASH memory. Obviously the number displayed on your board should be different to the one shown here using my board. This number will also be display for 10 seconds and the LEDs will still be working normally because the 'real' application is still enabled to operate correctly.

In a real design you probably wouldn't display this serial number information. That said, it could be displayed and used as the serial number for the product or as a identification number which you would request your customers to provide when contacting your technical support.

Then things will go wrong! The first indication is that the LCD displays a failure message. At the same time that this message appears, the LEDs will all turn off. Your cloned product is not authorised and it has failed!



After approximately 5 seconds the LEDs suddenly appear to be working again but then after a short while that start flashing in a very strange way. Some LEDs are off and others are flashing almost randomly.



Finally, all the LEDs are turned on and then slowly fade to off and everything has stops. Only a power cycle or pressing PROG makes the whole sequence repeat. This has demonstrated that cloned hardware and a copied configuration image doesn't mean that a design will work. Indeed, it has failed by design. The way in which it failed was totally my choice and later you will discover why it failed in the rather strange ways that it did☺



# Low Cost Design Authentication

## Purpose

The advantage of using a programmable FPGA is that it is standard product enabling rapid product development and reduced time to market. A potential down side is that it makes products more susceptible to being cloned by unscrupulous organisations who can obtain the same devices and copy the configuration program image stored in the configuration FLASH memory (just as you have just done). The aim of Spartan low cost design authentication is to prevent illegal clones from operating even if they are a perfect hardware and firmware copy whilst keeping the cost of security to a minimum.

## Know Your Enemy

Low cost design authentication is intended to be a deterrent to an organisation or individual (hereafter called 'The Attacker') that has an interest in cloning your product. To be a viable proposition, your product would be of reasonable unit value and produced in significant volumes. The Attacker must be able to copy, manufacture and sell clones at a lower cost than your original product otherwise the exercise is pointless. Therefore, The Attacker will be deterred if their clone requires the additional cost of more components to bypass the security measures included in your design. Furthermore, adding components will require rework of the PCB and The Attacker will only have a certain amount of time and resources to commit to bypassing your security mechanism; by default they are second to market and will want to be in production quickly.

Low cost design authentication is analogous to home security. You probably secure your home using a reasonable quality of doors and windows fitted with reasonably sized locks or bolts. Your objective is to have adequate security to act as a deterrent to burglars that would be interested in seeing if you have some possessions worth stealing. Design authentication is initially intended to deter The Attacker from even attempting to clone your product and to move on to something else.

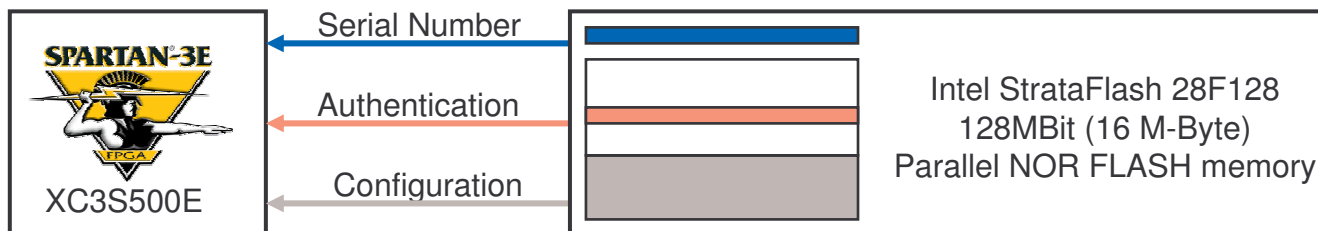
Some people feel that they need additional home security and will install a burglar alarm system. Typically this includes some obvious looking boxes and lights mounted outside the house which are again supposed to act as a deterrent to the burglar although some would argue that such visual indicators also imply that the house in question must contain items of value worth investigation! Regardless, the alarm system is intended to impede the progress of the burglar should they attempt to bypass one of the locks (e.g. break a window) and enter the house. Design authentication can also contain mechanisms to thwart The Attacker that attempts to bypass the design authentication 'lock'.

As with deciding a level of your home security, you must decide who your enemy is and how much to spend (engineering time and device resources) on design authentication security. Spartan devices allow you to decide what is sufficient.

# Low Cost Design Authentication

## Principle

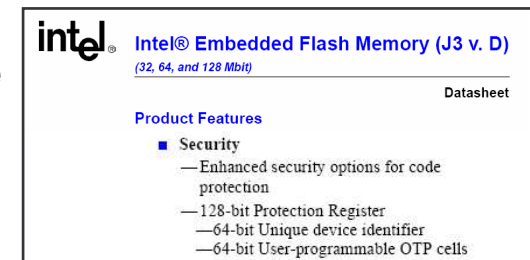
In its simplest form, design authentication involves reading two pieces of information and deciding if they are related in the way which is expected. A valid combination will then allow the design to operate and an invalid combination will result in the design being disabled. This is almost identical to the way a PC or a Web site requests you to enter a 'User Name' and a 'password' and then decides if the combination is valid before allowing you to continue.



The Spartan-3E device has no specific features for security. Therefore design authentication is implemented as a small part of your main application using the same device features and development tools as the rest of your design. Eventually the authentication process becomes part of the device configuration image typically stored in a configuration FLASH memory. In this reference design the configuration image is (ideally) stored in the Intel StrataFlash NOR FLASH memory (IC22). When the board is first turned on, the Spartan XC3S500E reads the configuration image from the memory and the design, including the design authentication circuits, begin to operate.

The design authentication circuit then reads two pieces of information from the memory. The first value is a 64-bit **unique** serial number that Intel provide for 'code' security. The serial number is like a 'User Name' that it is totally unique. It means that every Spartan-3E Starter Kit is also unique and no one can make a board identical to mine or to yours.

Secondly, the design authentication circuit reads an authentication value from the memory. This is like reading the 'password' and the design authentication circuit decides if this authentication value (password) is a correct partner to the serial number (User Name). The only difference between this scheme and the username/password analogy is that it is possible for someone to read the authentication value from the memory and copy it. Therefore the strength of the security lies in the fact that the serial number is unique combined with the algorithm which defines its relationship with the authentication value.



# Authorizing Your Clone

We are now going to see why your board is failing in slightly more detail and then authorise it to work.

To do this, you will need to connect a serial cable between J9 on the board and your PC serial port. The cable needs to be a male to female straight through cable (critically pin2-pin2, pin3-pin3 and pin5-pin5). You also need to set up HyperTerminal or a monitor utility using the settings advised on the next two pages.

## **CRITICAL – Understand what you are about to see and do**

It is important that you understand that much of what you are about to see and do is only because this is a design intended to help you learn the concepts and experiment. Much of what you actually see and do would NOT be included in a real production design because it would compromise security. You are about to see and do two major things....

### **Design Authentication Process in Action**

You will see the way in which the PicoBlaze performing design authentication is 'thinking' and the serial number and authentication values it is reading. You will see the values it computes using an algorithm and then see it make the decision to authorise or disable the design. A real design would NOT display such information.

### **Authorise a product**

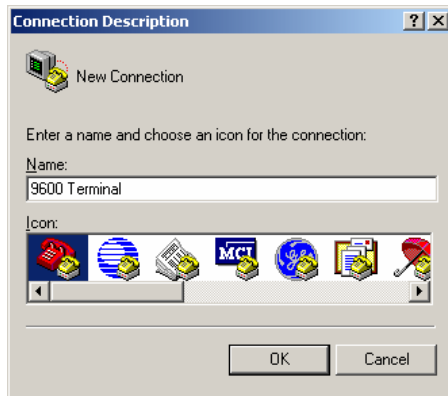
You will be able to authenticate your clone to make it a valid working 'product'. You will instruct PicoBlaze to generate the authentication value to match the unique serial number of the StrataFLASH memory and then the store that authentication value somewhere else in FLASH memory. This is the process that genuine products would need to go through at some stage during production by either shipping pre-programmed FLASH memories to the contract board manufacturer or by post programming the complete product (like your Starter Kit) at a trusted facility using a special FPGA configuration that is only loaded temporarily via JTAG. So what you are about to see and do would NOT be part of a normal production design unless there were significant measures taken to prevent it being used by an Attacker (i.e. the product could be authorised in the field using a web registration process or by inserting a SIM card but that technique would need to be secure in its own right).

# Serial Terminal Setup

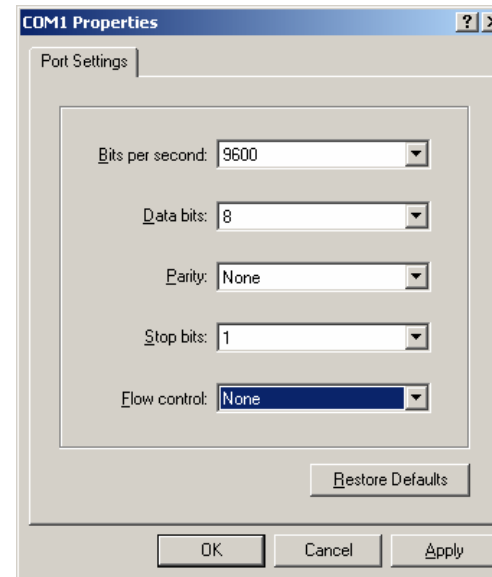
An RS232 serial link is used to communicate with the design. Any simple terminal program can be used, but HyperTerminal is adequate for the task and available on most PCs.

A new HyperTerminal session can be initiated and configured as shown in the following steps. The communication settings and protocol required by an alternative terminal utility are also included in these two pages.

- 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 : 9600**  
**Data bits: 8**  
**Parity: None**  
**Stop bits: 1**  
**Flow control: None**

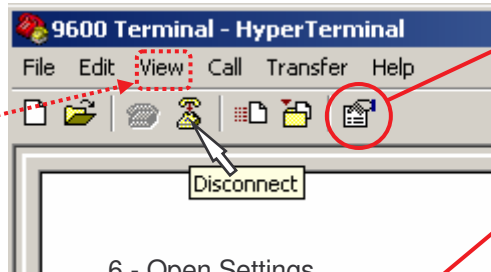
Go to next page to complete set up...



# HyperTerminal Setup

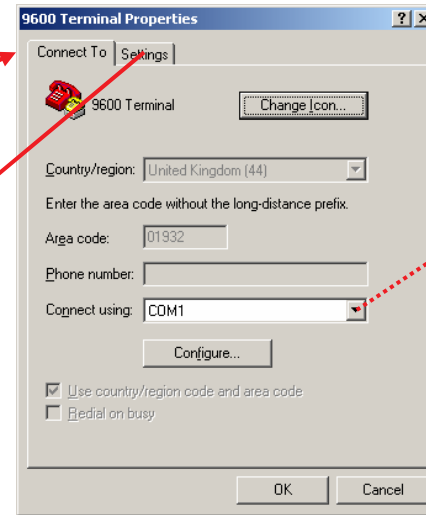
Although steps 1, 2 and 3 will actually create a Hyper terminal session, there are few other protocol settings which need to be set or verified for the PicoBlaze design to work as expected.

4 - Disconnect



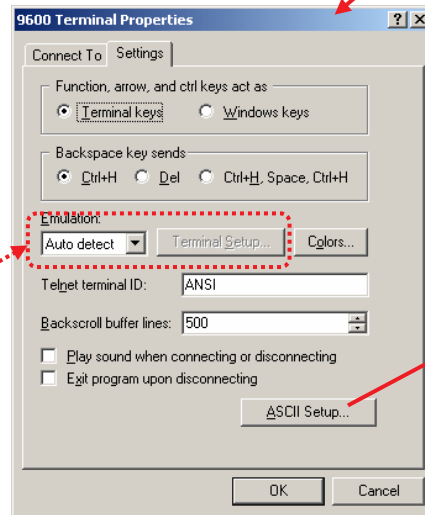
Optional step.....  
Set Font to  
Courier New,  
Regular, 10

5 - Open the properties dialogue

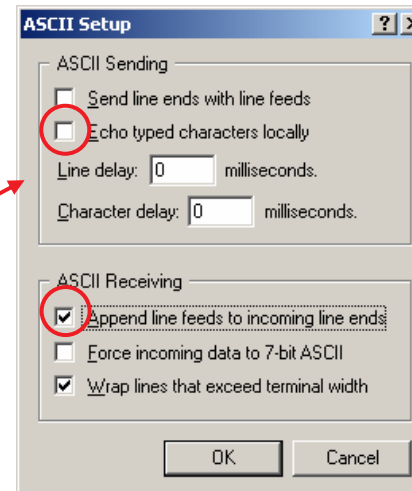
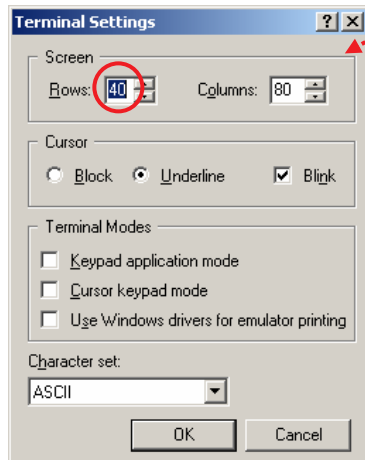


To select a different  
COM port and change  
settings (if not correct).

6 - Open Settings



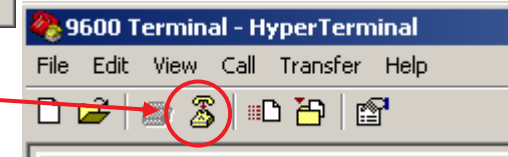
Optional steps.....  
Select VT100 and then click  
'Terminal Setup'  
Set 'Rows' to 40.  
(May require you to stretch main  
screen later to fit).



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<sub>HEX</sub>) to indicate end of  
line so you do need the 'Append line  
feeds to incoming line ends' option to  
be enabled.

8 - 'OK' the boxes to get back to  
main screen and then Connect.



# Why You Fail!

When you are ready, cycle the power or press PROG on your 'illegally cloned' Spartan-3E Starter Kit clone. You should see PicoBlaze performing the design authentication task and display the following information on your PC terminal as the board goes through the steps we reviewed previously (i.e. this all takes time to appear on your screen).

**IMPORTANT – A real security design should NEVER output such useful information.**

```
9600 - HyperTerminal
File Edit View Call Transfer Help
PicoBlaze Low Cost Design Security v1.00
Copyright Ken Chapman 2006
FLASH Serial Number = 72 9C 43 4D 00 3E 76 07
Computed CRC = 4F90
FLASH CRC = FFFF
Authentication FAILED
Menu
R - Read Authorisation
E - Erase Authorisation
A - Authorise
>_
```

So this is just a text message to confirm what is talking to you ☺

A copyright message is also just a text message and as such it would not prevent the design from being copied. However, it does have legal implications and a simple UART output on a spare pin could prove to be a powerful feature in a court of law. If The Attacker realises this copyright message exists then it is something will need to be removed (more of that later).

PicoBlaze reads the StrataFLASH serial number. It now has the knowledge about what is unique on your board.

PicoBlaze computes the authentication value that is required to match with the StrataFLASH serial number. The label is a big clue as to the nature of the algorithm which I used!

PicoBlaze then reads the authentication value stored in a particular location of the StrataFLASH memory.

Finally, PicoBlaze compares the authentication value read with the authentication value expected and clearly notices the mismatch with the result that it knows that this is an illegal copy and the failed product sequence begins.

**WHEN?** Carefully observe and note at what point during the failing design sequence that this menu is displayed. Then proceed to the next page.

# Authorizing Your Clone

This is the point at which we see where the authentication value is stored in the StrataFLASH memory and you get to authorise your clone as if you were a trusted programming facility.

```
R - Read Authorisation ← R
E - Erase Authorisation
A - Authorise
```

```
>
```

```
060000 FF FF FF FF FF FF FF FF FF FF FF FF FF FF ←
060010 FF FF FF FF FF FF FF FF FF FF FF FF FF FF
060020 FF FF FF FF FF FF FF FF FF FF FF FF FF FF
060030 FF FF FF FF FF FF FF FF FF FF FF FF FF FF
060040 FF FF FF FF FF FF FF FF FF FF FF FF FF FF
060050 FF FF FF FF FF FF FF FF FF FF FF FF FF FF
060060 FF FF FF FF FF FF FF FF FF FF FF FF FF FF
060070 FF FF FF FF FF FF FF FF FF FF FF FF FF FF
060080 FF FF FF FF FF FF FF FF FF FF FF FF FF FF
060090 FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0600A0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0600B0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0600C0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0600D0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0600E0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0600F0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

```
Menu
```

```
R - Read Authorisation
E - Erase Authorisation
A - Authorise ← A
```

```
>
```

```
Writing Authorisation ←
OK
```

Before authorising your board, enter 'R' at your keyboard to issue the 'Read Authorisation' command to the PicoBlaze processor.

This displays the area of memory in which the PicoBlaze processor looks for the authentication value. Assuming you have a blank StrataFLASH memory then you will see like this display that all memory locations contain 'FF' hex which is the default erased state of the memory.

As you can see, PicoBlaze is interested in the 256 bytes from address 060000 through to 0600FF hex. Again, such information should NOT be given away quite so quickly and obviously in a real design although we must accept that an Attacker could still determine these memory locations using a logic analyser and enough effort.

Now enter 'A' at your keyboard to issue 'Authorise Command'. This tells PicoBlaze to write the computed authentication value into the StrataFLASH memory to make your product a genuine 'product'.

**Once complete....** PLEASE cycle the power on your board or press PROG and watch what happens. Does your board now work continuously? Is your board authorised?

# Authorized Products

```
9600 - HyperTerminal
File Edit View Call Transfer Help
[Icons]

PicoBlaze Low Cost Design Security v1.00
Copyright Ken Chapman 2006
FLASH Serial Number = 72 9C 43 4D 00 3E 76 07

Computed CRC = 4F90
FLASH CRC = 4F90
Authentication PASSED

Menu
R - Read Authorisation
E - Erase Authorisation
A - Authorise

>
060000 3A DC 49 B2 C4 D5 BE B5 BC 83 64 D4 DD 2E 3F BA
060010 93 C2 FA 97 F0 24 C5 9F A1 A9 2E 6E 20 06 2C 34
060020 7D EB 7F BD 1F 4B 3E 58 27 CE 5F 9D CA 9B 14 0E
060030 6C A5 75 E1 17 17 9F 12 BC 1A 77 D8 E6 1E 28 31
060040 01 1F 5C 5A 2C 7F A3 08 9B 23 44 4B E6 53 69 75
060050 E1 6B 35 61 2A B9 81 B9 22 CB 50 62 09 13 D3 B9
060060 CA 96 70 96 67 BC 9F 2C 45 FB C1 5A 6C 5A E2 6E
060070 5C 20 24 E8 F1 CF C8 70 03 24 0C CD 4D DB 8C F0
060080 0C DE ED DD 29 EA 25 EC 90 B8 B5 86 61 81 32 60
060090 89 9F 95 A7 C7 77 53 3E 33 71 2E 48 2E A6 CF 7F
0600A0 BF A2 DE 9B C0 14 B3 90 01 D7 2F C8 E0 36 5E C8
0600B0 6A F8 A7 78 C5 6D C6 9C A2 3E CA 64 92 6D 14 6D
0600C0 3F E6 F0 22 39 D4 DD D1 7C E4 DD 34 29 4C A9 03
0600D0 EC 9D 42 0A 69 64 40 6B 87 B7 F9 EC 18 1B E2 43
0600E0 A9 AB 25 1B 70 31 46 F8 D6 A7 D6 96 C9 93 19 20
0600F0 AC BE FE FB C5 A3 32 52 EB 1E 1F 25 3E 29 6D 25
```

Hopefully you have seen your authorised board continue to work (LEDs continue to sequence left to right). The LCD display should now be indicating the authorised status along with the display on your PC screen.



This time PicoBlaze reads the correct authentication value stored in the StrataFLASH memory and allows the design to continue with normal operation.

Notice how fast the menu now appears compared with when the authorisation failed.

Feel free to experiment with the 'E' command. This erases block 3 of the StartaFLASH memory which covers the address range 060000 to 07FFFF. Hence it will erase the authentication value but will preserve the configuration image allowing you to repeat these steps again and again.

Use the 'R' command to display the authentication value. Are you surprised to see all of this rather than just the 16-bit value PicoBlaze said that it had read? Why do you think all 256 bytes are now programmed? Where is the authentication value?



# Attack!

Soon I will reveal you how the design authentication circuits of this reference design are actually implemented. However, as soon as I do that you will know all my 'secrets'. It is rather like putting my money in a good combination safe but then telling you the combination. At that point the security mechanism is virtually useless.

Before we get that far, I want you to recognise the potential strengths and weaknesses of the design authentication techniques such that you will always be able to provide adequate security for your own designs in future. The best way to do this is always to put yourself in the position of The Attacker and not the position of the defender (product designer).



So I now invite you to become a criminal for a moment and attempt to break into my reference design.

Erase the authentication value from your board and then see if you can find a way to make the design work (sequence the LEDs left and right continuously) without using the facility that I provided. The next pages will reveal potential ways to attack a Spartan design with authentication circuits to see how easy or difficult you think it is to 'break in'.

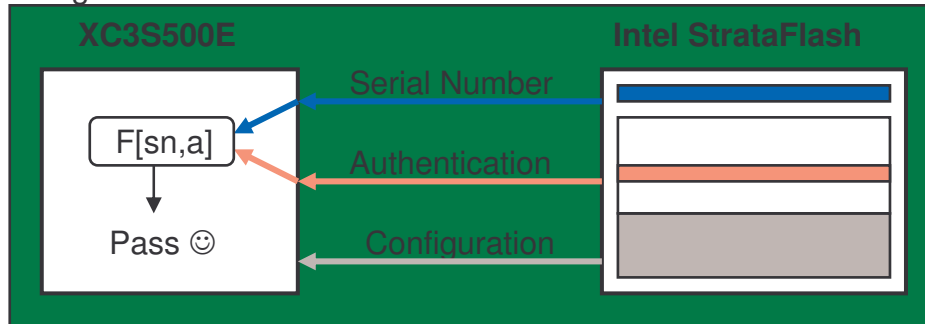
Remember that I have already provided you with helpful information that would already have taken an attacker some time to derive. I have also given you some significant clues you would not normally have; you know it uses PicoBlaze and the authentication algorithm has something to do with a 16-bit CRC value. So do stop to consider how The Attacker would even find that out that privileged information. Obviously at this stage you must not cheat by looking further into this document or reading the design source files but you can read the MCS file if you like and you can use the NOR FLASH programmer design to write or read the FLASH memory.

Always bear in mind that all security can be broken eventually (even banks and art galleries are not 100% safe!). Low cost design authentication security is about being 'good enough' to deter The Attacker. In this case, 'good enough' is when it takes too long for The Attacker to succeed. So see how long it takes you to break in. Even if you don't physically try yourself, then think of what you would try and estimate how long you think they would take to carry out. Always consider that in the long term you would need to make ANY Starter Kit work and not just the one that you have.

# Strengths and Weaknesses

**Algorithm** - The algorithm links the unique StrataFLASH serial number and the authentication value. Therefore it is the algorithm which ultimately makes authorised real products work and causes clones to fail.

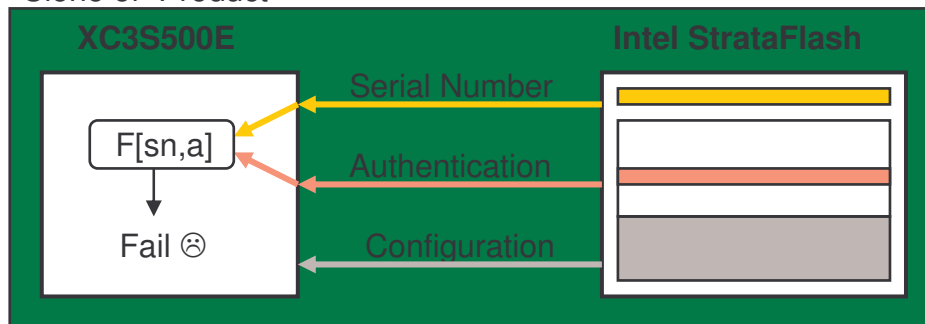
Original Product



In the original product, the authentication value programmed into the FLASH memory is the correct partner for the serial number in that same FLASH device. The authentication design confirms this match by reading both the serial number and authentication value and testing them with the algorithm or function (F[sn,a])

The algorithm (F[sn,a]) is kept secret, and if well chosen, it will mean that the authentication values required are also unique to each board.

Clone of Product



So if the board is cloned exactly, it does not matter from which original board the authentication value is copied, the clone will not have a correctly matching authentication value for the unique serial number contained in the Intel StrataFLASH device installed on that clone board.

Note that the configuration image loads in both cases but the design itself will fail to operate in the cloned product.

So the first point of attack is the authentication value itself. If the algorithm is too obvious or too simple to deduce then The Attacker will be able to program authentication values at will. It is therefore very important to use an algorithm and techniques that makes the task of deducing the algorithm adequately difficult and time consuming for The Attacker. It would be advisable to investigate cryptographically secure algorithms as well as incorporating your own ideas.

So can you work out the algorithm that I have used? Can you work out how to program the FLASH manually to make your board work. Does your theory hold true for my board too? If you succeed, then how long did it take you and how much did you base your deduction on the information that I had already revealed and how much time did that save you?

# Strengths and Weaknesses

**Brute Force Attacks** – One way to generate an authentication value for a given board would be to run through every permutation of authentication value until you stumble upon the value that works. In my design, the actual authentication value is only 16-bits which means that there are only 65,536 possible combinations to try. Such a *Brute force attack* is made much easier by having such a low number of bits and consequently your algorithm should yield larger authentication values. However, my design recognises this weakness and has taken two measures to strengthen it again.....

**Time Delays** – Every time you try the design it actually works for 20 seconds before starting to fail. Therefore it takes at least 20 seconds to try each combination. It would now take 15 days to try all 65,536 combinations and that would average out to 7½ days per board. The more time delay you add into a design authentication design then the stronger your security. Security schemes that instantly indicates failure enables an Attacker to make rapid iterations and should be avoided. In fact, consider how a delay of a week or more could make The Attacker think you had no security until they were in production!

**Concealment** – Although I stupidly gave away my ‘secret’ about having only a 16-bit authentication value, it appears that the authentication value stored in the FLASH is 256 bytes. I have hidden the actual authentication value somewhere in there and that makes it far less obvious which bits need to be changed. FLASH memory tends to be very big, so imagine how difficult you could make it to find those bits and remember that a real Attacker wouldn't even know what to look for to begin with. So did you manage where the actual authentication value is hiding in the 256 bytes?

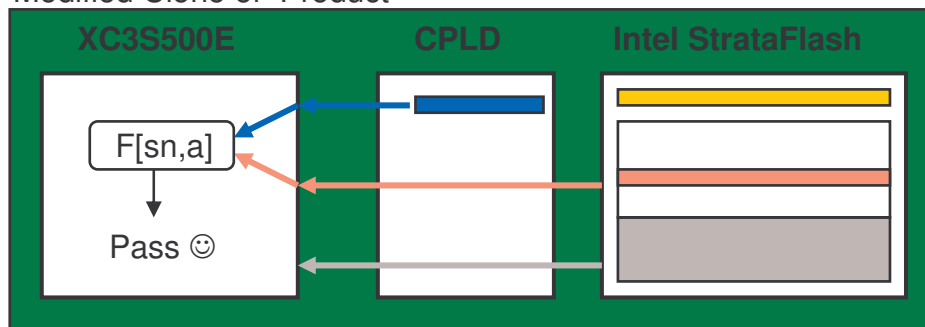
**Diversity** - In many way you can consider the concealment of the authentication value as being a second algorithm. The design authentication technique allows you to layer algorithms like having multiple layers of defence. There is also nothing to stop you having multiple authentication values each associated with a separate authentication value like a safe requiring three separate keys to open it. Each of your products and product revisions can (and should) use a totally different algorithm. Even if The Attacker succeeds in breaking your algorithm they are forced to start all over again because there is no one definitive receipt to breaking in.

**IMPORTANT** – Under no circumstances should you use this reference design without major modifications because I have revealed all the ‘secrets’ of my rather simplistic algorithm. This design provides ideas and it is not a solution to be copied.

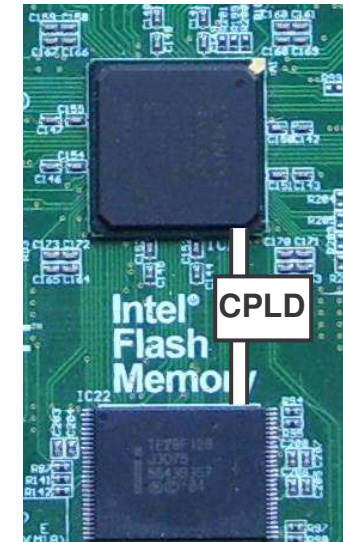
# Strengths and Weaknesses

**Spoof Attacks vs Economics** – The ultimate weakness in this scheme is that both the serial number and authentication value of an original product can be openly read. This is like providing your user name and password to someone. Therefore a clone product can be manufactured if it has the same serial number. The strength of this scheme is that every Intel FLASH device has a unique serial number so you simply can not purchase devices with the same number. This forces the Attacker to spoof the serial number. The diagram below illustrates how a CPLD could be used to intercept the request to read a serial number and respond with the same serial number of an original product.

Modified Clone of Product



This sounds easy in theory, but now look at your board and consider the implications. The Attacker is being forced to generate a modified PCB layout which is far more difficult than a pure photographic copy. It takes real engineering effort and time to do this.



As well as the time and effort required to modify the PCB and generate the CPLD design, The Attacker is being forced to add the cost of a CPLD to every cloned product. The Attacker will have a board with a larger bill of materials (BOM) than the original manufacturer and that is a bad starting point when attempting to copy high volume products and produce them at a lower unit price. Design authentication wins economically.

Hint – If your Spartan device can read other unique serial numbers in your system, then those too would need to be spoofed adding to the economic burden of The Attacker.

# Strengths and Weaknesses

**Configuration Image** – The configuration image for an XC3S500E device is 2,276,344 bits. A mass of ‘0’s and ‘1’s with no humanly obvious correlation to the design it represents once loaded into the FPGA. Obviously there is a mapping of these bits to the elements inside the Spartan device but this mapping information is not published. So although it is not impossible to reverse engineer a configuration image, it would be an incredibly time consuming task to do so. As if that wasn’t enough of a deterrent to The Attacker, the result of a reverse engineered image would be a circuit description with absolutely no structure (no design hierarchy) and no signal names to help understand functionality. If you have ever seen an schematic generated from an HDL description by a synthesis tool then you will have just what a mess The Attacker would be faced with trying to understand. Then somehow The Attacker would need to locate the small design authentication circuit and understand or bypass it. This reference design contains far more than the pure validation process and still occupies less than 5% of the XC3S500E device.

The potential for diversity of algorithms means that The Attacker doesn’t know exactly what to look for in your design. Furthermore, it is well known that even a slight modification to a design can result in PAR (place an route) making quite different use of the device resources so the location of the design authentication circuits could be anywhere on the device. In conclusion, the reverse engineering a complete configuration image is not considered economically viable and an adequate deterrent when combating high volume cloning of products.

**Design Manipulation** – Although reverse engineering a complete configuration image is not a practical proposition, it must be recognised that The Attacker may attempt to tamper with the configuration image looking for a simple way to bypass the design authentication process. In this situation The Attacker is less interested in the algorithm itself but more in the way the authentication circuit enables or disables the main application. If The Attacker can manipulate the configuration in a way that permanently enables the design then the serial number and authentication values are irrelevant.

A design authentication scheme should combat this type of attack by ensuring there is no single point of weakness. Your design can contain any number of design authentication tests, each test can then provide multiple enable/disable controls over the application and the design can contain deliberate anti-tampering logic. Again the use of time delays will significantly slow down The Attacker’s ability to determine if each attack attempt is a success or failure. Employing multiple enable/disable mechanisms each with a different time delay will lead The Attacker to wonder if they have ever fully succeed even if they bypass the first one or two signals.

Can you reverse engineer this reference design from the configuration image? Can you bypass the authentication process? Obviously you have to do it with the MCS file or image in the FLASH memory and not look at the design source files provided.

# Revealing my Secrets

From this point onwards, I will go into greater detail about how my particular design authentication circuit is implemented and operates. This totally gives away the secrets of my design and of course this totally compromises the security and therefore it is vital that you recognize that this design must not be used directly in your own designs. My design is intended to illustrate the techniques which can be employed but you must think of your own secrets and implement them. Good security is about keeping secrets, so if you have a brilliant idea, remember to keep that idea to yourself. Ensure that source code for your design are kept in safe storage and can not be stolen.

In this section, we will take another look at the reference design as it works on your board, but this time I will reveal exactly what is happening at each stage. At each stage we will examine what is happening in both the unauthorised and authorised situations and I encourage you to experiment as you read. At this stage I also invite you to look at the source files of the design which are as follows.....

- `low_cost_design_authentication_for_spartan_3e.vhd` Top level file and main description of hardware.
  - `low_cost_design_authentication_for_spartan_3e.ucf` I/O constraints and timing specifications for Spartan-3E Starter Kit.
  - `kcpsm3.vhd` PicoBlaze processor used as the 'application processor' controlling the LEDs.
  - `led_ctrl.vhd` Assembled program '`led_ctrl.psm`' for the 'application processor' generated using the standard '`ROM_form.vhd`' template.
  - `kcpsm3.vhd` PicoBlaze processor used as the 'security processor' performing design authentication.
  - `security.vhd` Assembled program '`security.psm`' for the 'security processor' generated using the template '`ROM_form_dual_port_ROM.vhd`'.
  - `bbfifo_16x8.vhd` FIFO buffer used to link between the two processors.
  - `uart_tx.vhd`
    - `kc_uart_tx.vhd`
    - `bbfifo_16x8.vhd`
  - `uart_rx.vhd`
    - `kc_uart_rx.vhd`
    - `bbfifo_16x8.vhd`
- UART transmitter and receiver with 16-byte FIFO buffers.

Note: The files shown in **green** are not included with the reference design as they are provided with the PicoBlaze download. Of course you already have these files because you answered 'yes' to one of the questions on page 4, but if you should still require a copy then please visit the PicoBlaze Web site.....

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

# 'Real' Application & Security

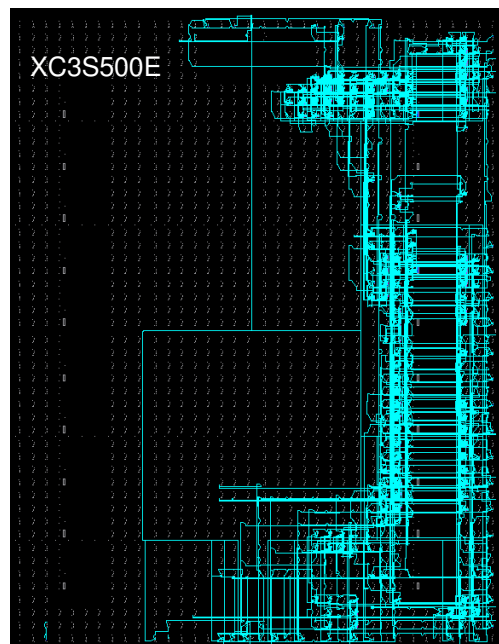
The whole point of having design security is to protect the real application. In this reference design, the 'real' application is represented by a PicoBlaze processor controlling the LEDs. However, this combined with all the security logic is still less than 10% of the XC3S500E device resources so it is not truly representative of a real design. It means that the security aspects of this small reference design are made easier to locate and attack than real design that is so much bigger. So were you successful?

## MAP report

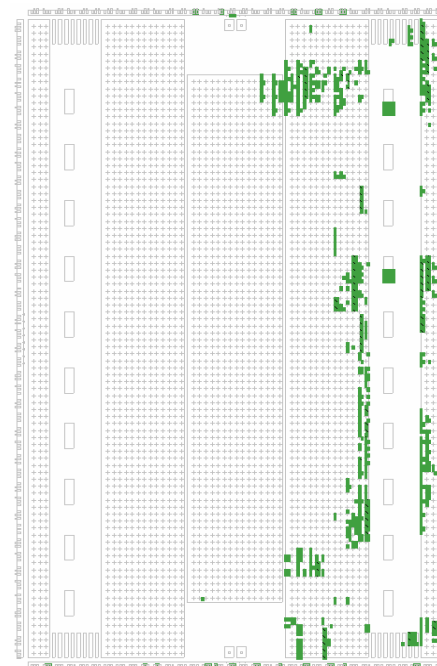
Number of occupied Slices:	320 out of	4,656	6%
Number of Block RAMs:	2 out of	20	10%
Total equivalent gate count for design: 155,540			

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



# The 'Real' Application PicoBlaze

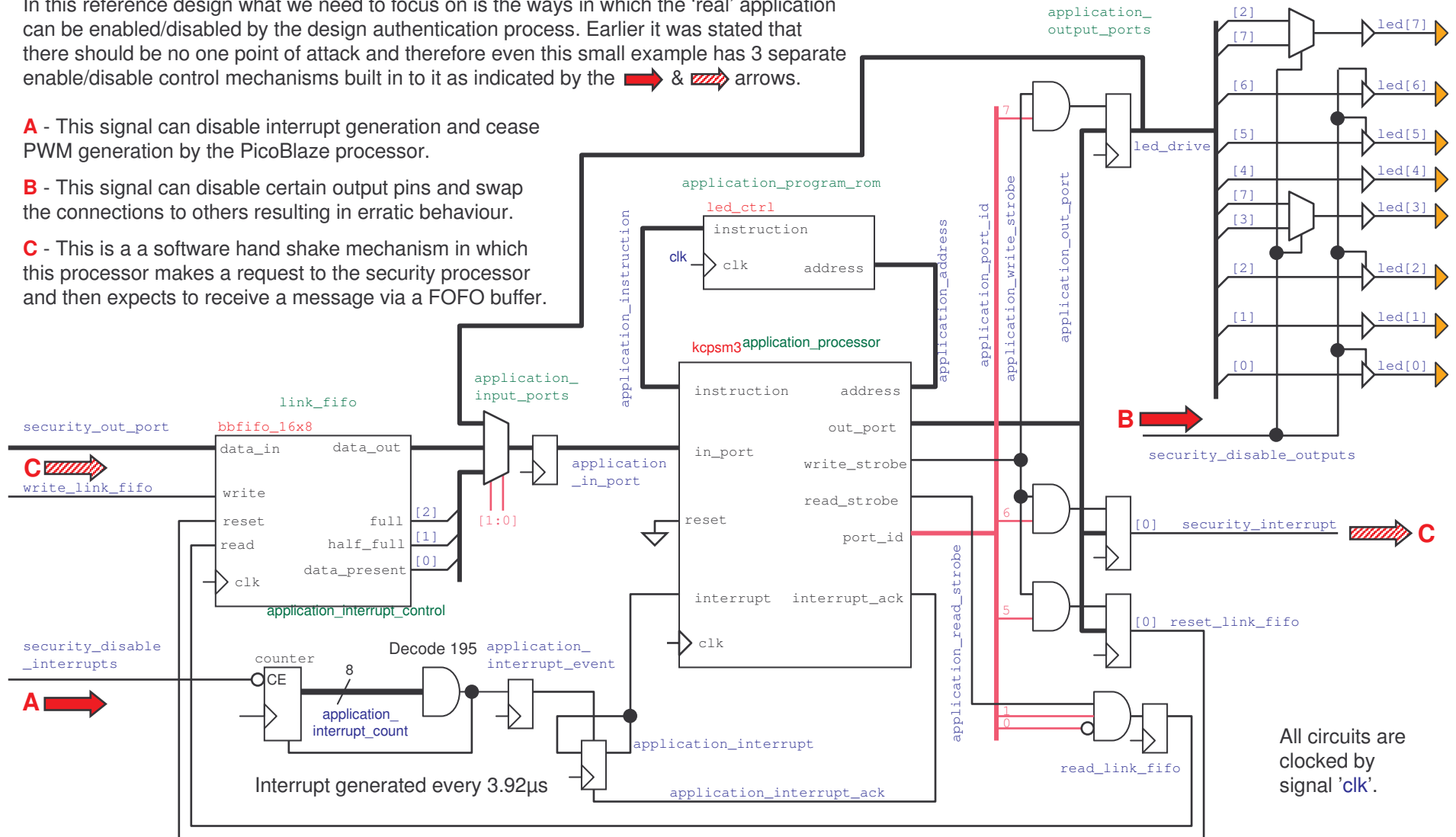
The 'real' application uses PicoBlaze to PWM (pulse width modulate) control the LEDs. You can learn more about this in the reference design 'Pulse Width Modulation (PWM) Generation and Control with PicoBlaze' (see web: [http://www.xilinx.com/products/boards/s3estarter/reference\\_designs.htm](http://www.xilinx.com/products/boards/s3estarter/reference_designs.htm)).

In this reference design what we need to focus on is the ways in which the 'real' application can be enabled/disabled by the design authentication process. Earlier it was stated that there should be no one point of attack and therefore even this small example has 3 separate enable/disable control mechanisms built in to it as indicated by the **→** & **↗** arrows.

**A** - This signal can disable interrupt generation and cease PWM generation by the PicoBlaze processor.

**B** - This signal can disable certain output pins and swap the connections to others resulting in erratic behaviour.

**C** - This is a software hand shake mechanism in which this processor makes a request to the security processor and then expects to receive a message via a FOFO buffer.

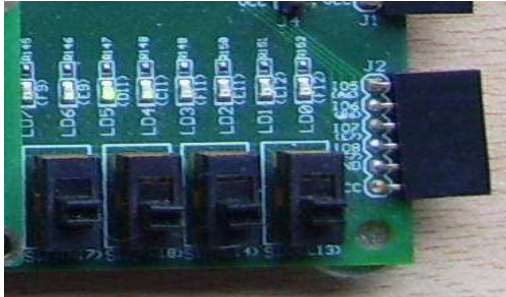


All circuits are clocked by signal 'clk'.



# A Closer Look at Failure

Make sure the authentication value is erased on your board and cycle the power or press PROG. Take a much closer look at what happens and relate this to the schematic and design source code.

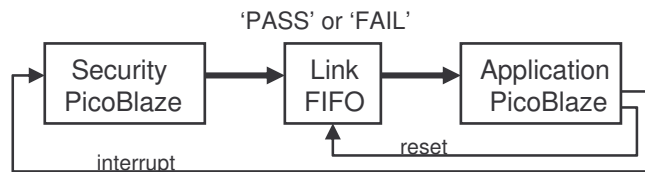


For the first 20 seconds the 'real' application is working normally. During this time all three enable/disable mechanisms are allowing things to proceed.



**A** – The 'security\_disable\_interrupts' signal initialises in the Low state and therefore the counter used to generate interrupts at 3.92µs intervals can operate. These interrupts mean that PicoBlaze executes an interrupt service routine at 256KHz and this results in a 1KHz PRF with 8-bit resolution.

**B** – The 'security\_disable\_outputs' signal also initialises in the Low state and therefore all LED outputs are driven with the corresponding PWM output channel.

**C** – This 'enable' is a little more complex and less obvious. To understand this one we need to look at the application PicoBlaze program file 'led\_ctrl.psm' as well as the schematic. Approximately once every 8 seconds the main program jumps to a routine called 'authentication\_check' (part of which is shown on the right).

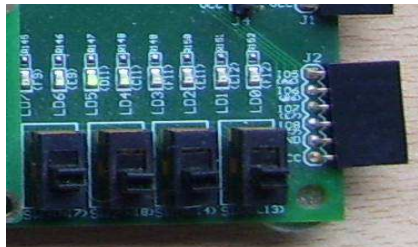


This routine begins by ensuring that the Link FIFO buffer is clear of any previous characters and then issues an interrupt to the security processor. The security processor responds by writing either the message 'PASS' or 'FAIL' into the Link FIFO. Until 20 seconds have elapsed, the security processor it will always respond with 'PASS' so the 'real' application processor receives the correct message and the normal main program continues.

<p>Clear Link FIFO buffer</p>  <p>reset_link_fifo</p>	}	<pre> LOAD s0, link_fifo_reset OUTPUT s0, link_fifo_control_port LOAD s0, 00 OUTPUT s0, link_fifo_control_port ;         </pre>
<p>Generate interrupt to security processor</p>  <p>security_interrupt</p>	}	<pre> LOAD s0, security_interrupt OUTPUT s0, security_request_port LOAD s0, 00 OUTPUT s0, security_request_port ;         </pre>
<p>Read Link FIFO and tests for the four character message 'PASS'</p>	}	<pre> CALL read_link_FIFO COMPARE s0, character_P JUMP NZ, fail_confirm ☹️ CALL read_link_FIFO COMPARE s0, character_A JUMP NZ, fail_confirm ☹️ CALL read_link_FIFO COMPARE s0, character_S JUMP NZ, fail_confirm ☹️ CALL read_link_FIFO COMPARE s0, character_S JUMP NZ, fail_confirm ☹️ JUMP normal_LED_sequence 😊         </pre>

# A Closer Look at Failure

After 20 seconds have elapsed the security processor performs the actual authentication check and recognises that the product is not authenticated. The reference design makes this clear by displaying the 'Authentication Failed' message on the LCD and PC displays. However, what interests now is how it actually disables the 'real application'. This is where you need to study really closely what happens during the failure sequence.



**A** – Initially all the LEDs appear to turn off. This is the effect caused by the security PicoBlaze driving the 'security\_disable\_interrupts' signal High and disabling preventing the generation of interrupts at 3.92 $\mu$ s intervals. The 'real' application processor is still actually working but the execution sequence has been altered significantly. The security PicoBlaze has total control over this disable signal so it could do anything with it. In this case it only activates it for 5 seconds and then returns it Low allowing the 'real' application to work again.

**B** – After another 5 seconds of apparently normal operation the LEDs then behave very erratically. This time the security PicoBlaze is driving the 'security\_disable\_outputs' signal High and although the 'real' application PicoBlaze is working perfectly normally the PWM outputs that it generates are scrambled or turned off at a hardware level. Look carefully and you will see that LEDs 0, 2, 5 and 6 are always off and LEDs 3 and 7 blink out of order; confusing isn't it! Once again the security PicoBlaze only activates this disable signal for 5 seconds and normal operation resumes again very briefly.

**C** – Moments after normal operation is restored, all the LEDs turn on together and then slowly fade away to off. This failure mechanism is now actually part of the 'real' application processor program (see routine called 'failed\_LED\_sequence' in 'led\_ctrl.psm'). The security processor is now responding with the message 'FAIL' rather than 'PASS' each time it receives an interrupt. So in this case the failure occurred once the 'real' application processor made the request and it decided in what way the operation should be effected.

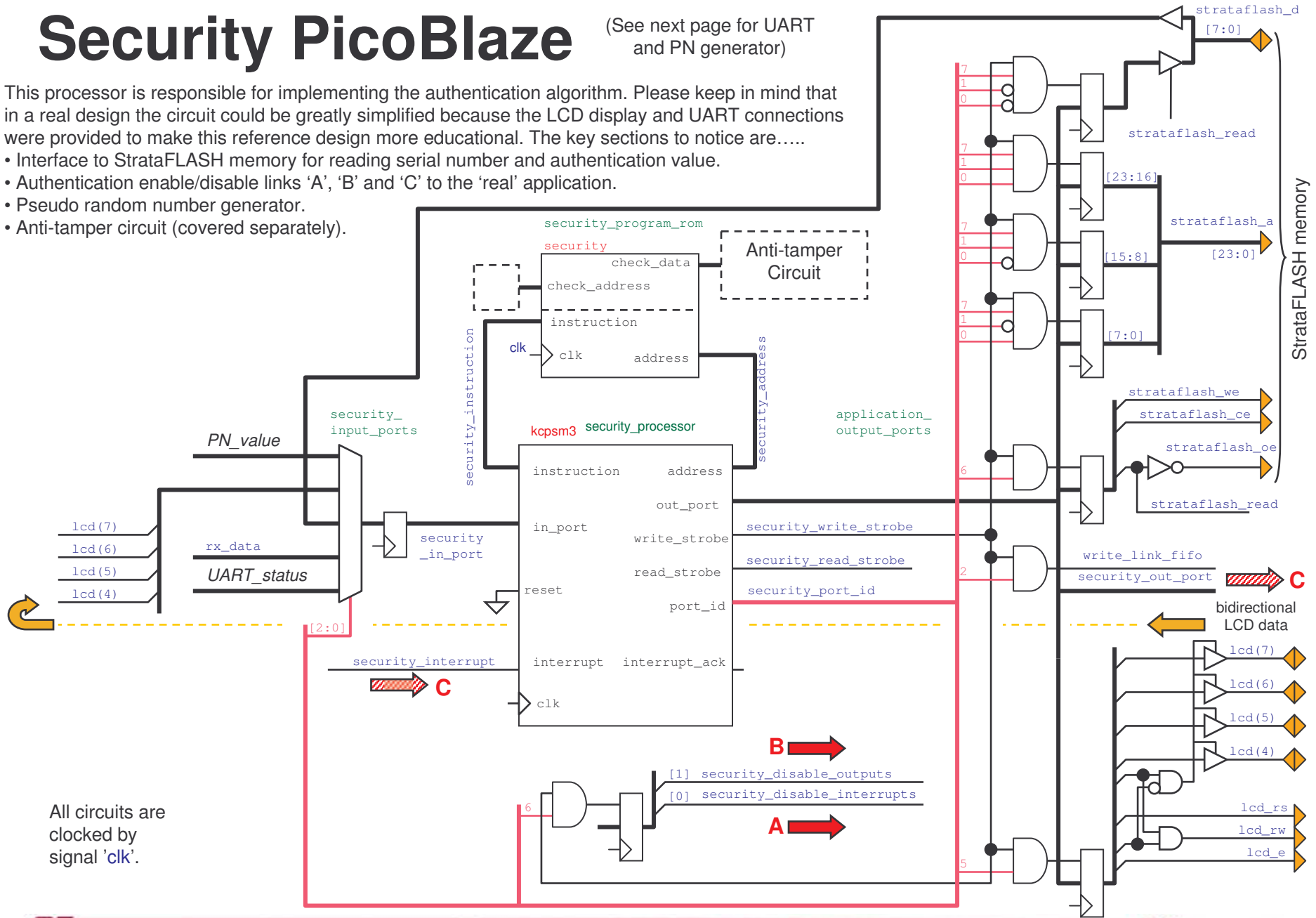
**Resistance to Attack** – Even this simple reference design has demonstrated enable/disable mechanisms in a small design. Both 'A' and 'B' are single points of attack which, in theory, could be found and permanently forced Low. In a real design, simultaneous activation of these signals would mean The Attacker would need to find both at the same time which would be extremely difficult without reverse engineering the whole configuration image. Mechanism 'C' can not be overridden by forcing signals High or Low which really means that the design must be reverse engineered and then modified. Dynamic signalling techniques will always offer stronger security and can be implemented with hardware state machines as well as software. In conclusion, the more enable/disable signals and mechanisms a design has, then the stronger the security.

# Security PicoBlaze

(See next page for UART and PN generator)

This processor is responsible for implementing the authentication algorithm. Please keep in mind that in a real design the circuit could be greatly simplified because the LCD display and UART connections were provided to make this reference design more educational. The key sections to notice are.....

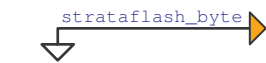
- Interface to StrataFLASH memory for reading serial number and authentication value.
- Authentication enable/disable links 'A', 'B' and 'C' to the 'real' application.
- Pseudo random number generator.
- Anti-tamper circuit (covered separately).



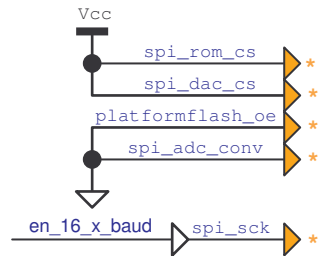
All circuits are clocked by signal 'clk'.

# Security PicoBlaze Peripherals

The design uses the StrataFLASH in byte access mode meaning that the upper data bits [15:8] are unused at all times.

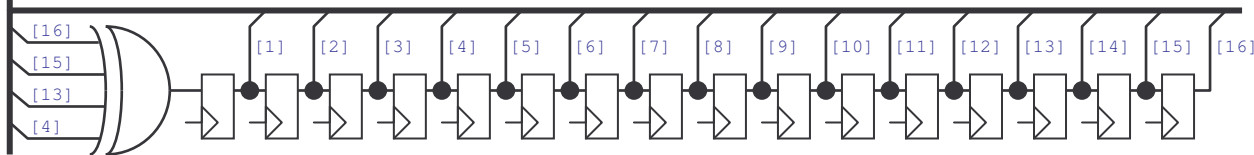


\* Other devices on the Starter Kit board are disabled to prevent interference when working with the StrataFLASH memory.



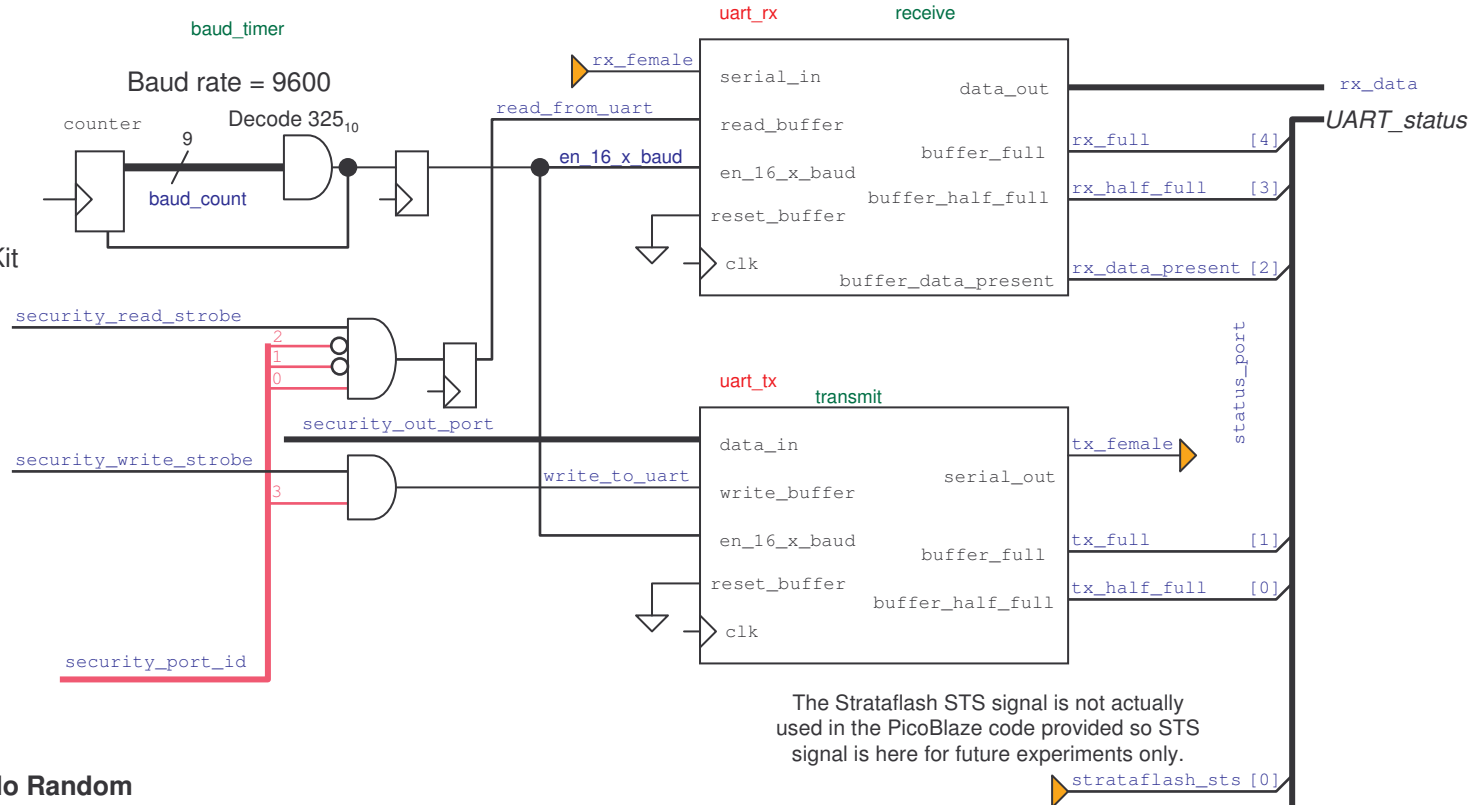
[12, 9, 4, 6, 11, 7, 16, 2] `PN_value`

**Pseudo Random Number Generator**



See XAPP052 and comments in source VHDL file for operation

**UART macros include 16-byte FIFO buffers**



The Strataflash STS signal is not actually used in the PicoBlaze code provided so STS signal is here for future experiments only.

All circuits are clocked by signal 'clk'.

# The Algorithm

The actual authentication algorithm at the heart of this reference design is a rather simple 16-bit Cyclic Redundancy Check CRC. As discussed previously, it would be advisable to have a more complex algorithm resulting in a value represented by a greater number of bits and ideally an algorithm which is cryptographically strong. Anyway, in this case the CRC value is computed by PicoBlaze as a software routine but it would also be possible to implement the computation in pure hardware or using a combination of hardware and embedded software depending on your preference.

The whole purpose of the algorithm is generate an authentication value from the unique serial number of the StrataFLASH memory. The algorithms needs to be of a type that produces different authentication values for each unique number in a way that does not make the relationship between the serial numbers and authentication values easy to determine. For example adding a 64-bit constant to the 64-bit serial number is an 'algorithm' but not very difficult to break. The CRC computation I have used is a reasonable demonstration but really something better should be used in a real design.

CRC details It is left as an exercise for you to discover the fully details of the CRC computation by examining the source code of the security PicoBlaze processor provided in the file 'security.psm'. All source files contain numerous comments to help you understand.

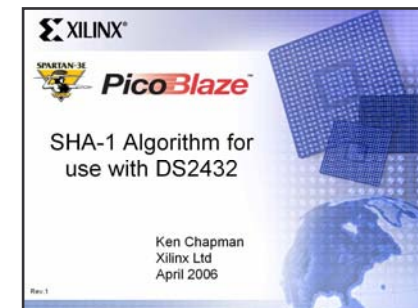
Question – Besides using the StrataFLASH serial number with the CRC polynomial, what else did I do to make the authentication value less obvious?

## ENIGMA



In 1918, Engineer Arthur Scherbius patented the famous cipher machine which used a combination of rotors, reflectors and wires with plugs. Even the simplest Enigma machine had in excess of 15,000,000,000,000,000,000 combinations. Between 1932 and 1939, Marian Rejewski, Henryk Zygalski and Jerzy Rozicki succeeded in breaking the first Enigma codes in the days before computers and electronic calculators existed so please do not underestimate the ability that some people have to deduce algorithms from data sets. That said, an Attacker of your designs will have far less data sets to work with (unless they buy many original products) and they do not have years available to break your code.

Hint - For more algorithm inspiration take a look at the 'SHA-1 Algorithm for use with DS2432' reference design for the Spartan-3E Starter Kit and XAPP780



# Hiding the Authentication Value

The second effective technique used in this reference design is one of hiding or obscuring the authentication value. In many respects the obscuring of the authentication value is also an algorithm making it difficult for an Attacker to know which bits stored in the FLASH memory are even used in the real algorithm.

When you authenticate this reference design on your board it writes 256 bytes of data into the FLASH memory . Of these 2048 bits only 16 are the real authentication CRC value.

To prevent a simple comparison of two or more boards, it is vital that all 256 bytes appear different in each original product. Therefore, each time you use PicoBlaze to authorise your board you will notice that it fills most locations with pseudo random numbers. Try erasing and writing the authorisation several times and you will see how it changes every time just as it should appear different in every original product.

## So where is the CRC value hiding?

I will leave this for you to discover. I have only used a very simple method so see how long it takes you to find it. Remember that unlike an Attacker, you know what value you are looking for.

Computed CRC = 4F90

060000	1B	7B	BC	1C	13	EC	0F	6A	D3	26	F0	62	83	19	55	B1
060010	95	62	8B	8A	1E	9A	7A	E8	52	11	BF	55	CD	57	39	1A
060020	D5	E9	E5	E2	28	4C	02	17	94	FF	9F	4F	AC	47	B6	DF
060030	09	33	B2	27	0D	09	79	80	31	47	04	56	0B	39	71	14
060040	2E	7F	AB	A6	7D	10	C4	2B	A4	10	E4	AC	5B	4F	0F	95
060050	DD	B5	6E	6B	BD	4D	22	E3	5A	6A	A4	39	77	E2	86	9B
060060	86	52	BB	35	76	3B	8E	C3	67	2E	B7	00	FD	8F	39	40
060070	2F	6B	90	51	F7	E9	46	39	F2	38	8D	23	0F	D8	79	50
060080	C9	A4	AD	62	E9	3A	6E	9F	60	72	00	9D	22	68	BC	CE
060090	14	61	64	30	08	32	BF	94	58	BE	E4	6F	62	28	92	0E
0600A0	AD	09	5C	51	32	15	8C	F9	EE	26	96	F3	C8	3F	53	DD
0600B0	2B	41	24	AE	5D	14	56	A8	95	24	4F	CC	E0	AE	D5	47
0600C0	53	33	CE	9C	A4	5A	0A	85	7C	82	10	09	17	26	3D	D8
0600D0	8B	DD	53	E1	26	80	0A	E2	72	3C	A7	E7	BD	34	78	DF
0600E0	64	D8	CB	61	D4	38	BB	F7	52	D5	8F	C0	8C	02	83	1D
0600F0	26	35	67	DA	C6	45	B0	BF	DA	80	0F	B6	A3	08	72	1C

If you really can not find the hidden authentication value then it is documented in the PicoBlaze source code 'security.psm'.

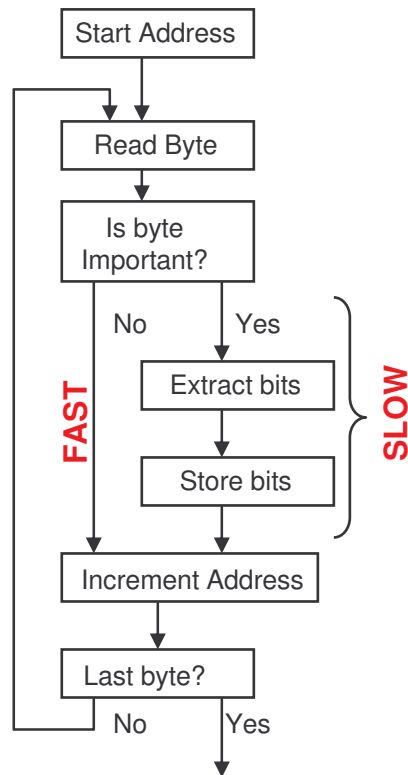
There are almost no limits to the ways in which you could hide authentication values. The random numbers could play a more active role rather than just looking like data. For example, using the values in the box above, consider the following possibility. The first random number '1B' could be used to mean move forward 27 locations where you then find the value '55' which in turn means move forward 85 locations where the real authentication value is located. In that way the real authentication value would be a moving target as well as looking hidden. Again remember that an Attacker doesn't even know what sort of authentication value to look for whilst doing this.

Design Hint – "Steganography"

# Do Not Provide Clues!

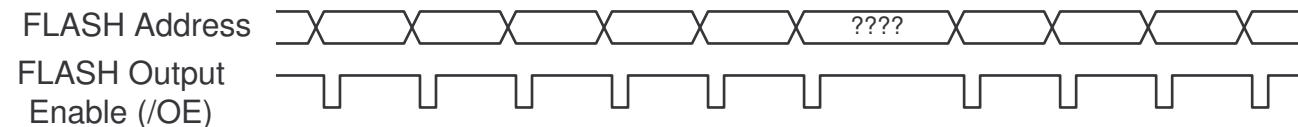
I think you will agree that trying to spot the location of the real authentication value in a block of random data is really very difficult. A child playing 'hide and seek' would soon be asking for a clue and the "warmer" and "colder" guidance messages from the child's parents keep the game alive 😊

So having done a marvellous job of hiding the real authentication value, it is very important not to give clues to an Attacker as to where the important information is hidden. In this reference design there are only 4 address locations that hold the important information. If PicoBlaze performing the authentication process only read those 4 key locations then it would be a very obvious clue to an Attacker equipped with a logic analyser. The address locations read by PicoBlaze would immediately identify the critical locations and all others could be ignored. So to prevent this type of attack. PicoBlaze always reads all 256 bytes of information and it then internally extracts the important information and ignores the rest.



However, even when reading all locations, some care needs to be taken. This flow diagram illustrates the actions a program needs to execute in order to read all locations and extract the key information. The sequential execution of instructions by PicoBlaze means that everything takes time. So when the program executes a different code sequence to handle the important bytes there is a high probability that the time taken will be longer than the cases where the data is simply ignored.

These small differences in time between FLASH read operations could be a clue for an attacker. In this simple case the observed timing 'glitches' would immediately identify important address locations.



Longer than normal time between OE pulses is a clue to important locations.

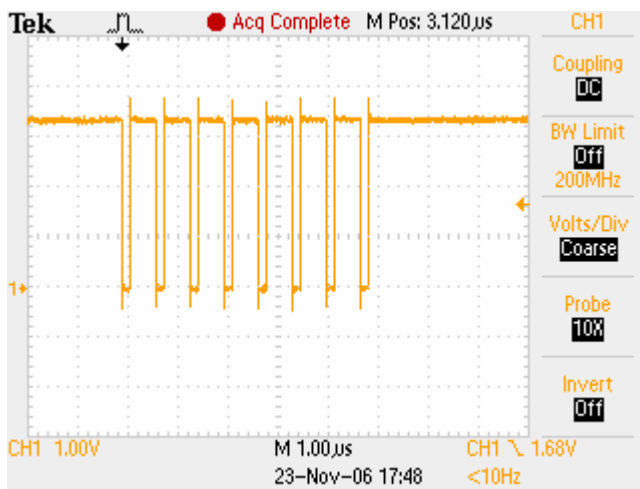
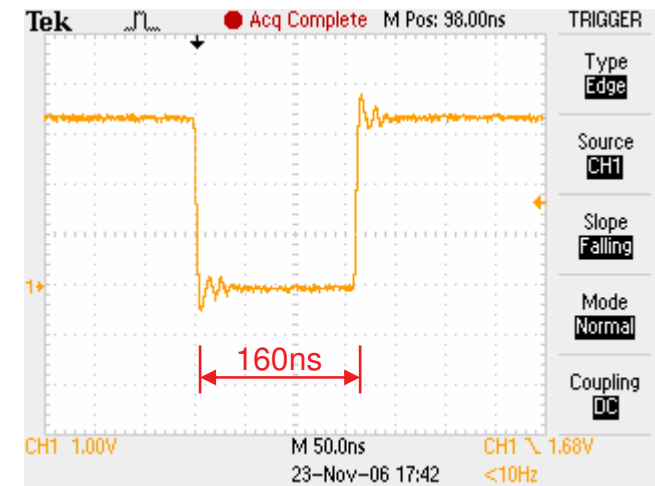
# Do Not Provide Clues!

PicoBlaze in this reference design ensures that it gives no outward clues as to which locations are important when reading the authentication bytes.

Firstly a review of PicoBlaze timing. Each instruction takes 2 clock cycles. Therefore each instruction takes 40ns to execute when using the 50MHz clock provided on the Spartan-3E Starter Kit. The following extract of code from 'security.psm' shows how PicoBlaze reads a byte from the StrataFLASH memory. There are 4 instructions between the OUTPUT instructions which control the OE signal which therefore result in OE pulses that are Low for 160ns as shown in the oscilloscope plot measured at pin 54 of the memory (also connects to R96).

```
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  
} Enable FLASH (/OE='1')  
} Read data (delay > 75ns required)  
— Disable FLASH (/OE='1')

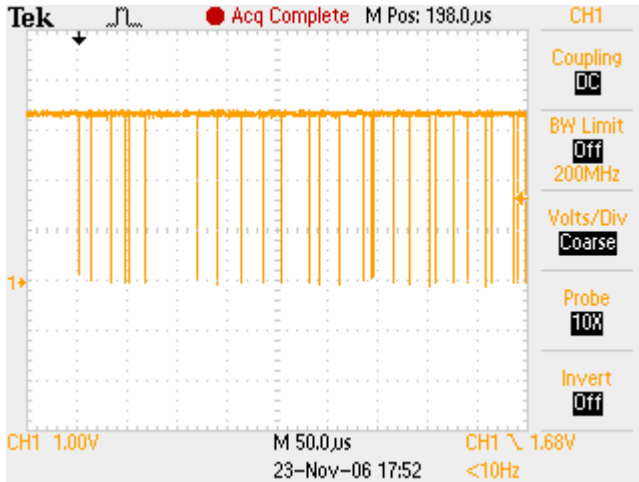


The plot to the left shows the OE pulses when PicoBlaze reads the 64-bit unique serial number. In this case each of the 8 bytes are read and stored in scratch pad memory; a process that has uniform timing. By monitoring the signals to the FLASH memory it is obvious that the serial number is being read but the regular timing of the read cycles provides no clue about the algorithm in which the serial number is subsequently used. So uniform reading is a good technique.



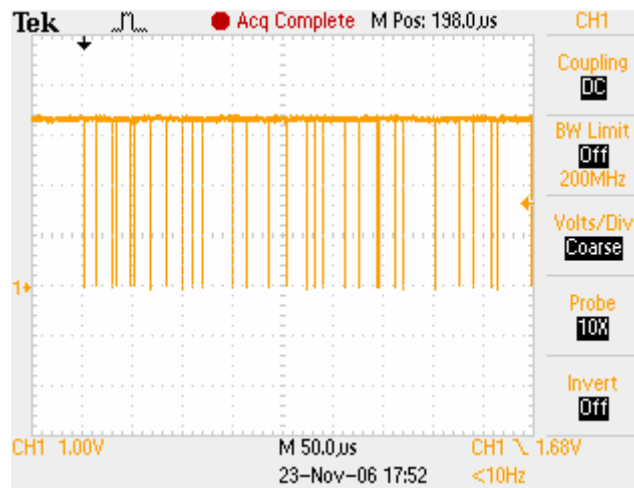
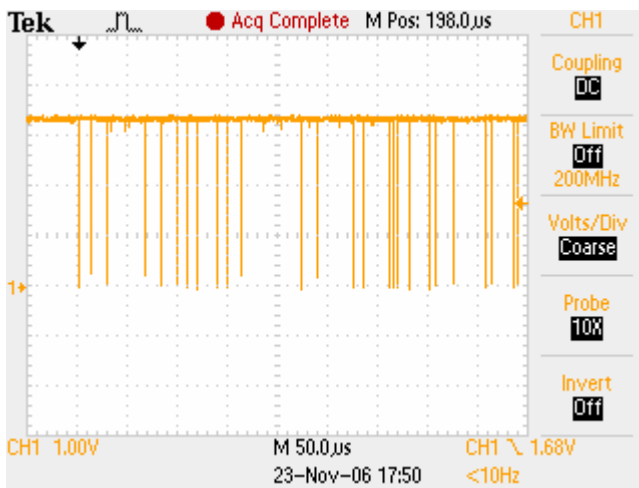
# Do Not Provide Clues!

The problem with reading the authentication value is that it would be a lot of data to store before extracting the important bits. It really is more desirable to extract the key information as it is read but this must avoid introducing timing difference clues for an Attacker to observe.



In some cases it may be possible to add artificial delay when ignoring data to equal the time taken when extracting key information but that soon becomes challenging software to write. So in this design I decided to deliberately insert a random duration delay between all read cycles. These random delays are large enough to obscure the small differences in execution times required to process the key information.

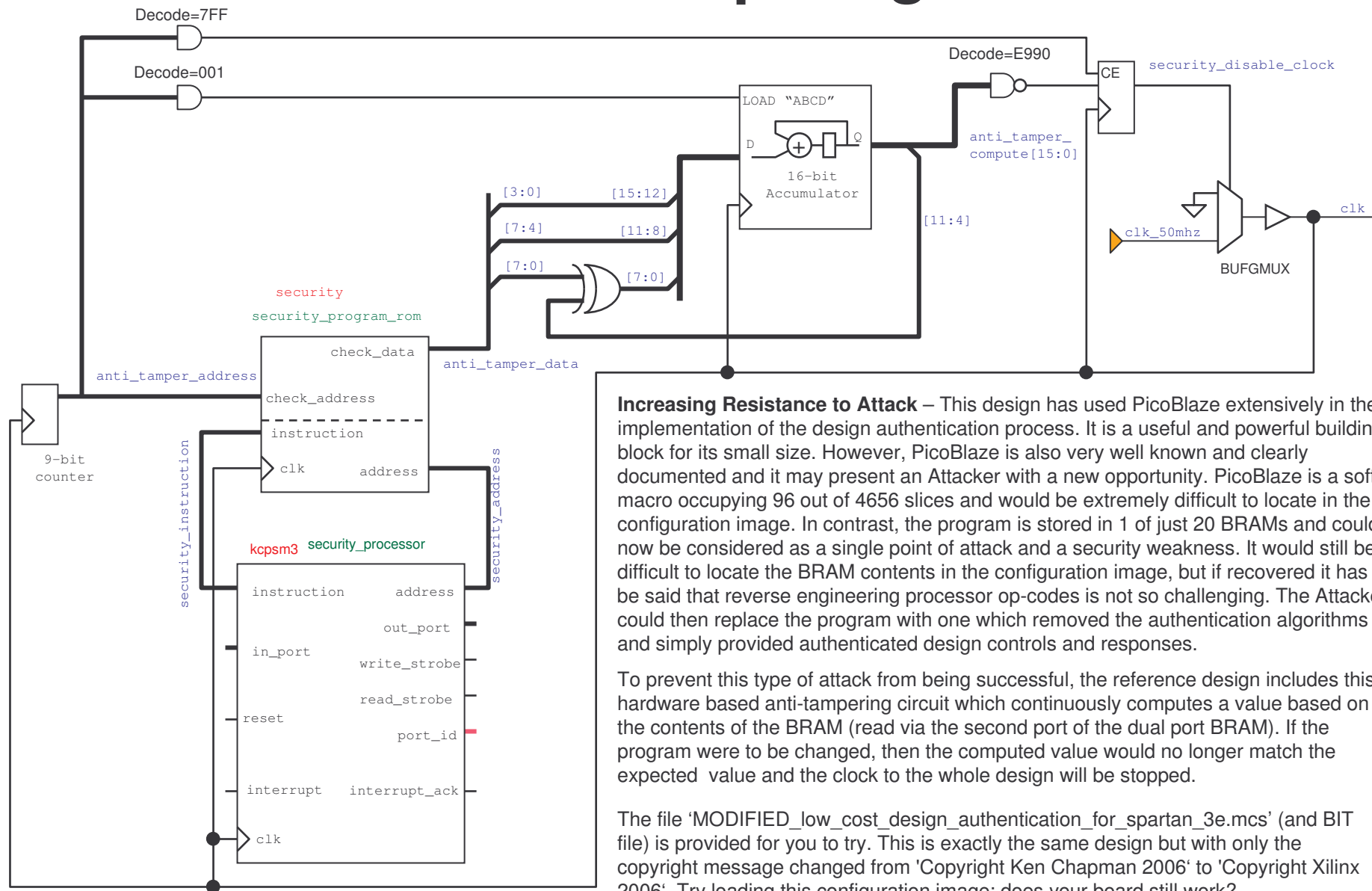
These three oscilloscope plots capture the OE pluses associated with reading the authentication data block from the FLASH memory. Approximately the first 30 of the 256 read cycles are shown in each case. It can be seen that every time the authentication value is read result in different read cycle timing (thanks to the PN generator). This random timing provides no clues to an Attacker.



Faster hardware based memory reading circuits may also be used to read the authentication values but care should also be taken with their implementation to ensure that no timing clues are presented to an Attacker.

Note – These traces were captured when the authentication process was executing. The 'R' command does not have this timing because it is for evaluation and diagnosis purposes only.

# PicoBlaze Anti-Tampering Detection



**Increasing Resistance to Attack** – This design has used PicoBlaze extensively in the implementation of the design authentication process. It is a useful and powerful building block for its small size. However, PicoBlaze is also very well known and clearly documented and it may present an Attacker with a new opportunity. PicoBlaze is a soft macro occupying 96 out of 4656 slices and would be extremely difficult to locate in the configuration image. In contrast, the program is stored in 1 of just 20 BRAMs and could now be considered as a single point of attack and a security weakness. It would still be difficult to locate the BRAM contents in the configuration image, but if recovered it has to be said that reverse engineering processor op-codes is not so challenging. The Attacker could then replace the program with one which removed the authentication algorithms and simply provided authenticated design controls and responses.

To prevent this type of attack from being successful, the reference design includes this hardware based anti-tampering circuit which continuously computes a value based on the contents of the BRAM (read via the second port of the dual port BRAM). If the program were to be changed, then the computed value would no longer match the expected value and the clock to the whole design will be stopped.

The file 'MODIFIED\_low\_cost\_design\_authentication\_for\_spartan\_3e.mcs' (and BIT file) is provided for you to try. This is exactly the same design but with only the copyright message changed from 'Copyright Ken Chapman 2006' to 'Copyright Xilinx 2006'. Try loading this configuration image; does your board still work?

# Your Own Security Designs

I do hope that you have found this reference design a useful introduction to low cost design authentication security. I know that for some of you, design protection is very important so again I hope that I have included sufficient details in this document and the design source code for you to fully understand the intricacies as well as the basic concepts.

Let me emphasize once again; DO NOT COPY THIS DESIGN FOR YOUR PRODUCT SECURITY because I have revealed all my secrets and it will be the first things that an Attacker would try. Please have fun creating your own schemes and variations.

The key to good security is recognising who your enemy is likely to be and thinking about the ways in which they may unlock or bypass your security measures. In this document I have been so open about the potential weaknesses of the design authentication technique; only by recognising the potential weaknesses was I able to introduce suitable counter measures. You must also take the time to identify the weaknesses of security schemes you implement and then strengthen your overall solution.

Be sure to design security protection into your project from the very beginning because, if you leave it until you have completed your main design, then you will then find it difficult to insert multiple authentication disable signals and mechanisms that penetrate deep enough into your design. However good the authentication algorithm is that you include, a single global enable/disable signal would offer a single point of attack which is to be avoided.

Finally, keep your secrets - If you have a good security idea, then keep it to yourself and use it! Make sure your design source code is also kept in a secure place and be careful who you share your code with in future.

## Make a feature of security

As a closing thought, consider how you can make design authentication a valuable addition to your product as well as providing the design security which can be so important. The scheme gives you total control over how authorised and unauthorised products do and do not work.

- Would it be useful for all your products to leave the factory only permitting an evaluation level of use until they are registered via the internet? Authentication value programmed into FLASH remotely under your direct control.
- Would it be useful to control the level of features available to your customers? 'Try before you buy' evaluations and charged upgrades all under the control of authentication values that are specific to the serial number of the product.