KCPSM6 Reference Design for the KC705 Evaluation Board

ICAPE2 Communication and Transactions

including…

ReadBack CRC Experiments and BRAM Data Buffer

Ken Chapman

5th September 2014 – Initial Release

Disclaimer

Notice of Disclaimer

Xilinx is disclosing this Application Note to you "**AS-IS**" with no warranty of any kind. This Application Note is one possible implementation of this feature, application, or standard, and is subject to change without further notice from Xilinx. You are responsible for obtaining any rights you may require in connection with your use or implementation of this Application Note. XILINX MAKES NO REPRESENTATIONS OR WARRANTIES, WHETHER EXPRESS OR IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, IMPLIED WARRANTIES OF MERCHANTABILITY, NONINFRINGEMENT, OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL XILINX BE LIABLE FOR ANY LOSS OF DATA, LOST PROFITS, OR FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR INDIRECT DAMAGES ARISING FROM YOUR USE OF THIS APPLICATION NOTE.

This Document and Reference Design

The primary purpose of this document is to provide images and examples of the design being used to supplement the descriptions contained in the source VHDL and PSM code. It is assumed that you already have a copy of the KCPSM6 variant of PicoBlaze and are familiar with using it (i.e. this probably isn't the best design to start with if you have never seen or used PicoBlaze before). In particular, this design builds upon on the 'uart6_kc705.vhd' UART based reference design provided in the KCPSM6 package so this documentation only covers the additions to that design; specifically communication with ICAPE2, Readback CRC monitoring and the connection of a RAM buffer.

Who is this reference design for?

The reference design is presented on the Kintex-7 KC705 Evaluation Kit but the reference code is suitable for reuse with virtually all of the 7-Series devices. It is anticipated that different parts of the design will appeal to different people for use in various applications…

Anyone needing to communicate with ICAPE2 – The Internal Configuration Access Port (ICAP) provides access to the configuration memory and the configuration state machine registers of the device. These items are described in the '7 Series FPGAs Configuration User Guide' (UG470) but it is fair to say that actually implementing communication and transactions can be challenging! Having a known working reference with source code that contains comprehensive descriptions should be of value. This design may not implement the precise transactions that you require for your own application but the selection that are implemented cover the various types and should be an ideal starting point for your own code. You don't have to use PicoBlaze to work with ICAPE2 but hopefully this design will show you that it a convenient way.

Anyone interested in configuration Readback CRC error detection and ECC correction – If you are at all concerned about Single Eventt Upsets (SEU) to configuration memory cells and the effect that they *may* have on a design then your primary interest should be the SEM IP core provided in with the Xilinx development tools and described in the 'LogiCORE IP Soft Error Mitigation Controller Product Guide' (pg036). Whilst the SEM IP core is recommended, the fundamental Readback CRC mechanism and single bit error correction capability is built-in to the silicon of every 7-Series device and the SEM IP core isn't always required (see UG470). This reference design enables you to learn more about the built-in capabilities and evaluate how they work and how SEU may effect a design. Knowledge gained using this design will also be useful when the SEM IP core.

PicoBlaze users – In addition to the ICAP and Readback CRC focus, this design contains examples of PicoBlaze usage and code that can be of value for reuse in other designs. There are PSM routines that implement the entry of a line of characters with simple editing and interpretation of numerical values (see 'line_input_and_editing.psm'). The design also has a 4096 byte memory (BRAM) connected to KCPSM6 and routines to write and read bytes and 32-bit words to and from it (see 'RAM_2048x8_routines.psm').

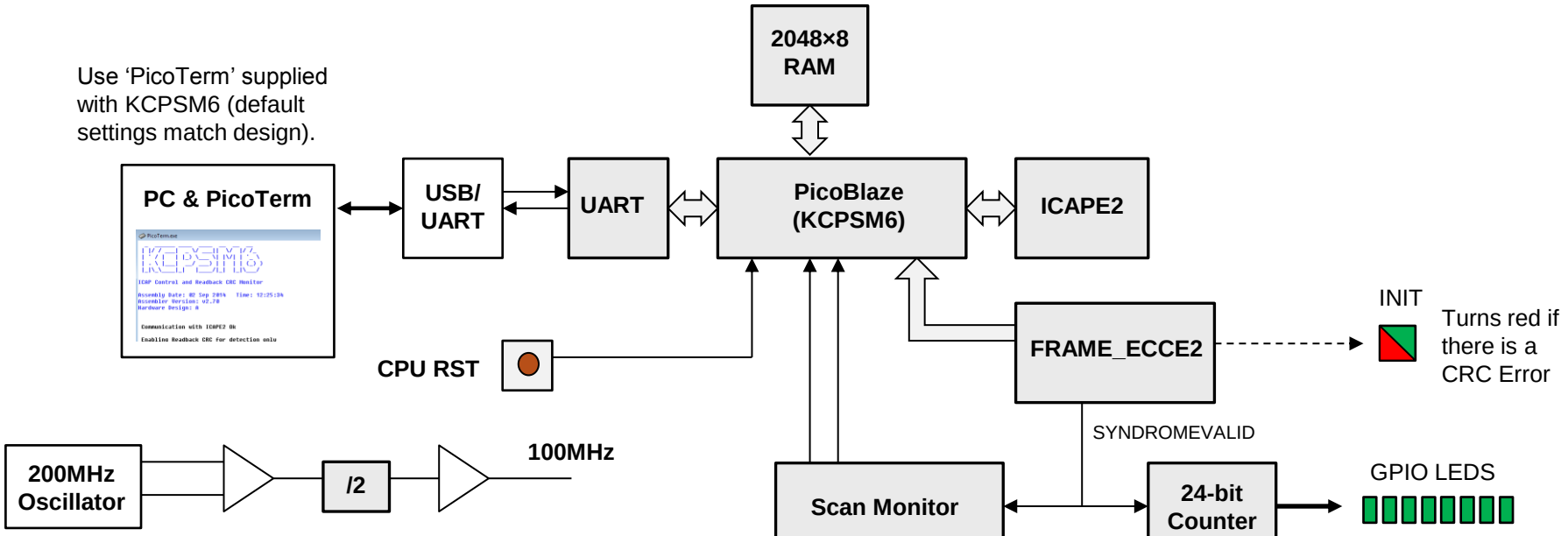
I do hope you find this reference design useful. Please provide any feedback related to this reference design (good or bad) to…

chapman@xilinx.com

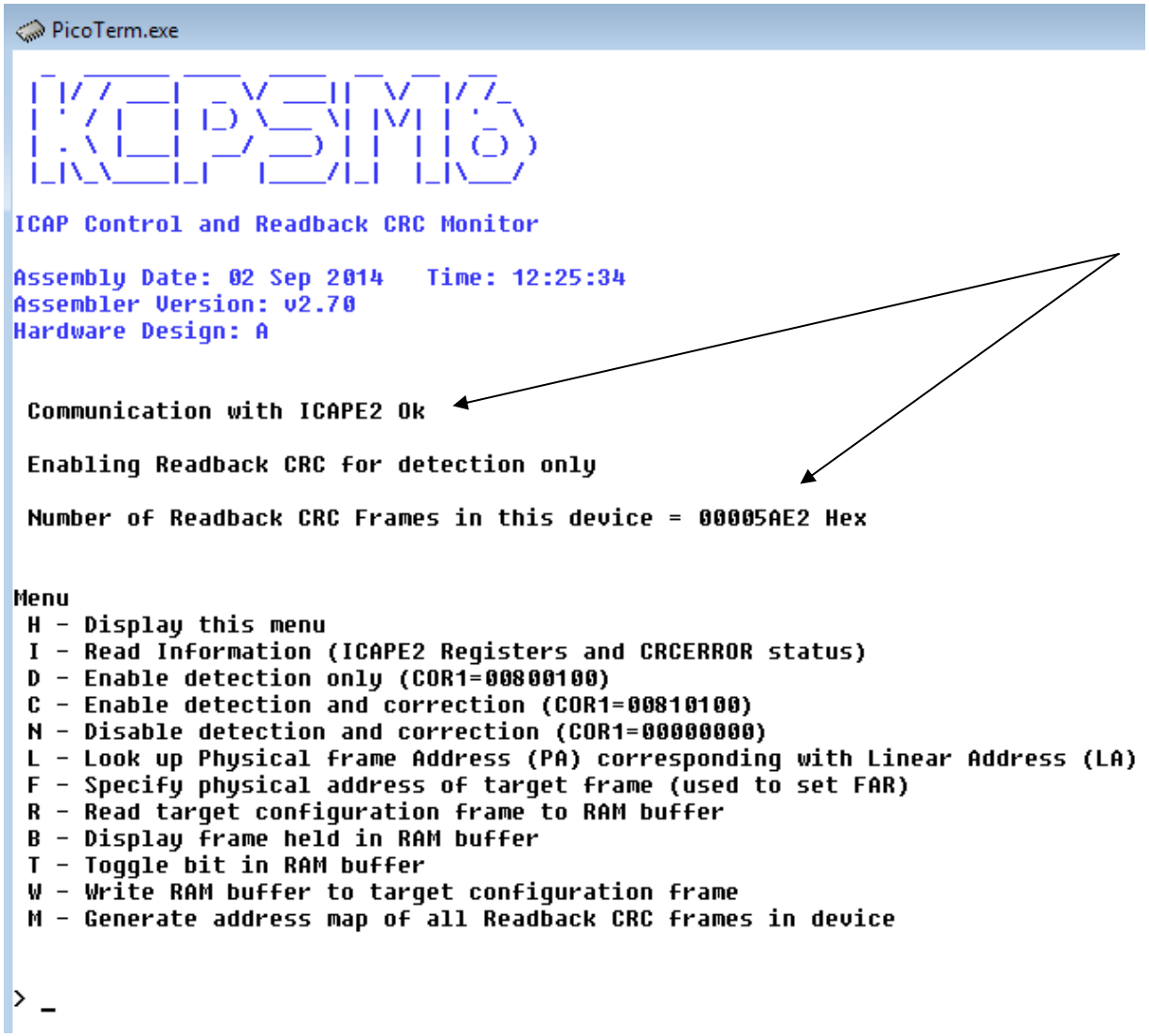
Overview of Reference Design

The design presents information and a menu of options on the PicoTerm terminal (many examples shown in this document). Please be aware that the PSM code provided consists in total of 2,873 instructions but the vast majority of these are related to user interaction. In fact, over 1,640 instructions are directly associated with the generation of text massages so any code actually implementing communication with ICAPE2 is much smaller ☺

The main focus of the design is an interface and communication with ICAPE2 for which there are some very exacting requirements in terms of both the hardware and subsequent transactions. The design has the ability to read, modify and write complete frames of configuration memory. Each frame is comprised of 101 words of 32-bits requiring 404 bytes of storage (i.e. larger than KCPSM6 scratch pad memory) so a BRAM has been connected to KCPSM6 to be used as a data buffer. KCPSM6 implements various ICAPE2 transactions including those that can enable and disable the Readback CRC error detection and frame ECC error correction capabilities of the device. KCPSM6, with the aid of a small amount of hardware, can observe outputs from the FRAME_ECCE2 primitive to observe behaviour and extract and present specific information concerning the Readback CRC scanning within the device.



Overview of Reference Design



This image of the PicoTerm window shows the opening display and menu implemented by presented by KCPSM6 in this reference design.

The design initially checks communication with ICAPE2 and determines the number of configuration frames being scanned by the Readback CRC mechanism (that KCPSM6 enabled by setting one of the ICAPE2 registers).

KCPSM6 then presents the user with a menu of 12 commands (options) each of which are described in more detail later in this document. All PSM source code includes detailed descriptions as well.

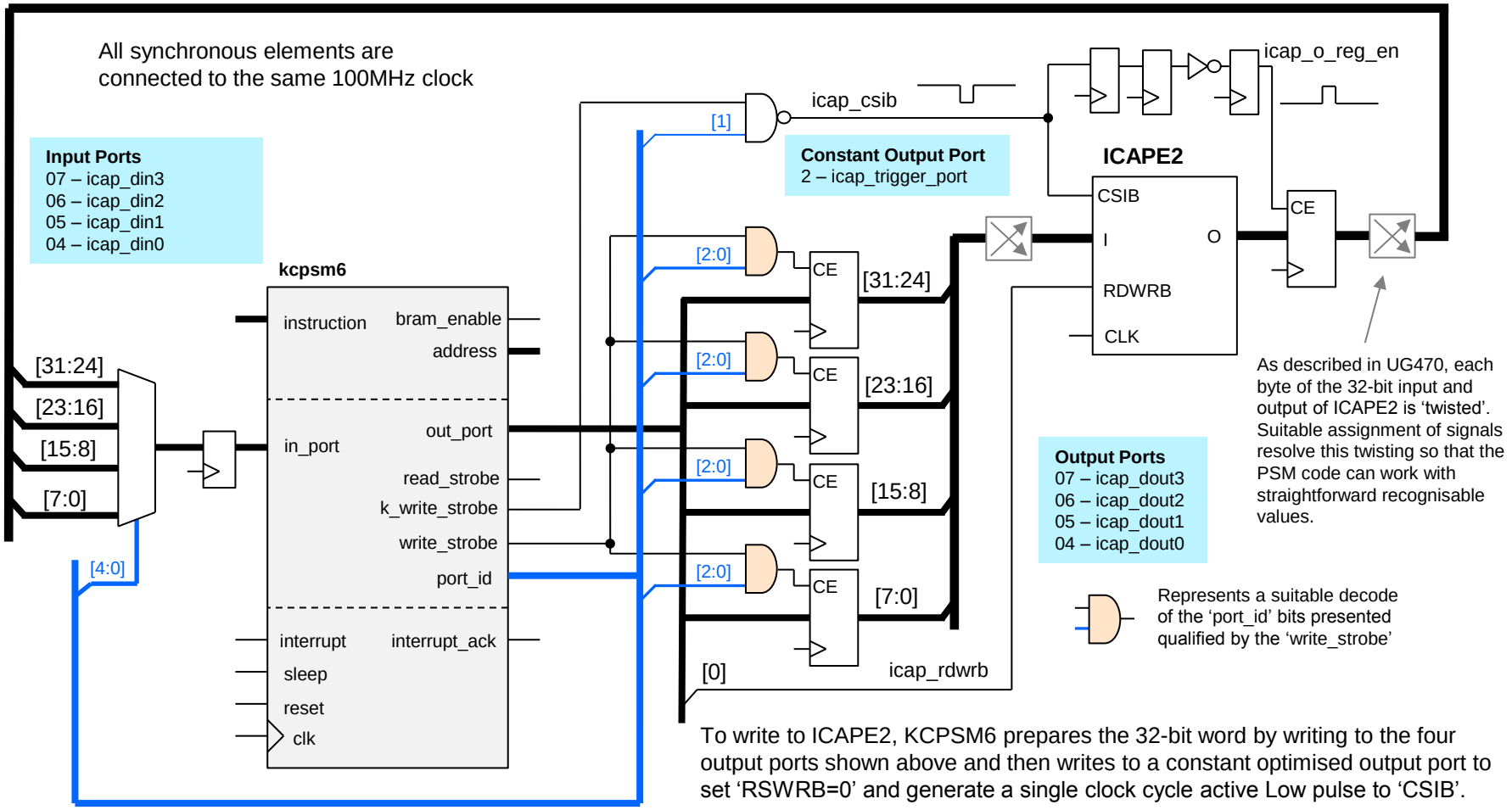
LOG Files

At the start of each session, KCPSM6 instructs PicoTerm to open a LOG file which will capture everything that appears on the screen. This complete log of activity can be very useful when conducting experiments. LOG files are automatically assigned names containing the date and time similar to 'PicoTerm_05Sep2014_121306.txt' and are written to the same directory as 'PicoTerm.exe'.

Connecting KCPSM6 to ICAPE2

For clarity this diagram only shows the interface and ports assigned to communicate with ICAPE2.

For KCPSM6 to read from ICAPE2 it writes to a constant optimised output port which sets 'RSWRB=1' and generates a single clock cycle active Low pulse to 'CSIB'. 3 clock cycles later, the value read is presented at the 'O' output of ICAPE2 and this is captured in a register as it is only remains valid for one clock cycle. Having allowed adequate clock cycles for the data to be captured, KCPSM6 can then read the 32-bit value via four input ports.

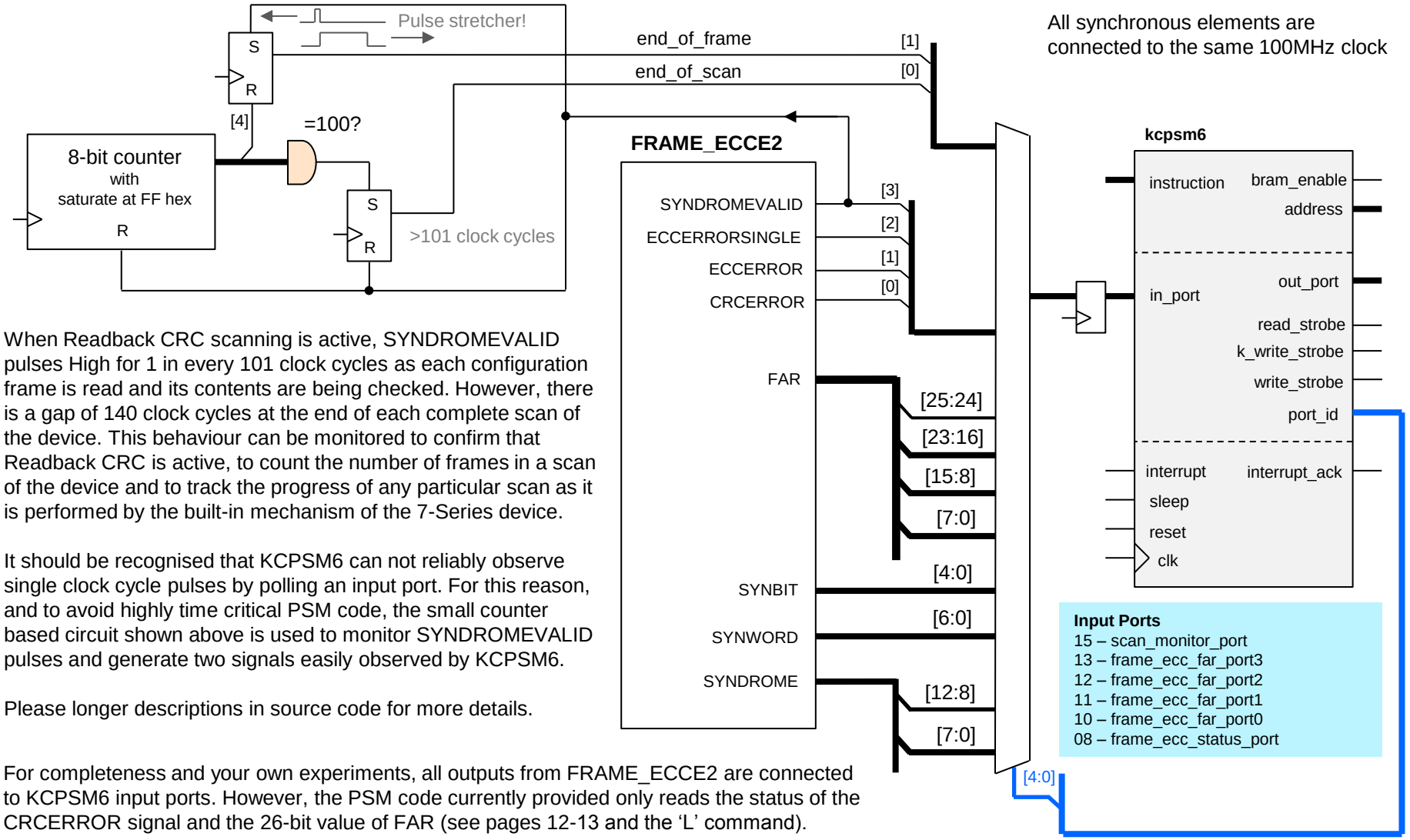


To write to ICAPE2, KCPSM6 prepares the 32-bit word by writing to the four output ports shown above and then writes to a constant optimised output port to set 'RSWRB=0' and generate a single clock cycle active Low pulse to 'CSIB'.

XILINX

Connecting KCPSM6 to FRAME_ECCE2

For clarity this diagram only shows the interface and ports assigned to monitor FRAME_ECCE2



When Readback CRC scanning is active, SYNDROMEVALID pulses High for 1 in every 101 clock cycles as each configuration frame is read and its contents are being checked. However, there is a gap of 140 clock cycles at the end of each complete scan of the device. This behaviour can be monitored to confirm that Readback CRC is active, to count the number of frames in a scan of the device and to track the progress of any particular scan as it is performed by the built-in mechanism of the 7-Series device.

It should be recognised that KCPSM6 can not reliably observe single clock cycle pulses by polling an input port. For this reason, and to avoid highly time critical PSM code, the small counter based circuit shown above is used to monitor SYNDROMEVALID pulses and generate two signals easily observed by KCPSM6.

Please longer descriptions in source code for more details.

For completeness and your own experiments, all outputs from FRAME_ECCE2 are connected to KCPSM6 input ports. However, the PSM code currently provided only reads the status of the CRCERROR signal and the 26-bit value of FAR (see pages 12-13 and the 'L' command).

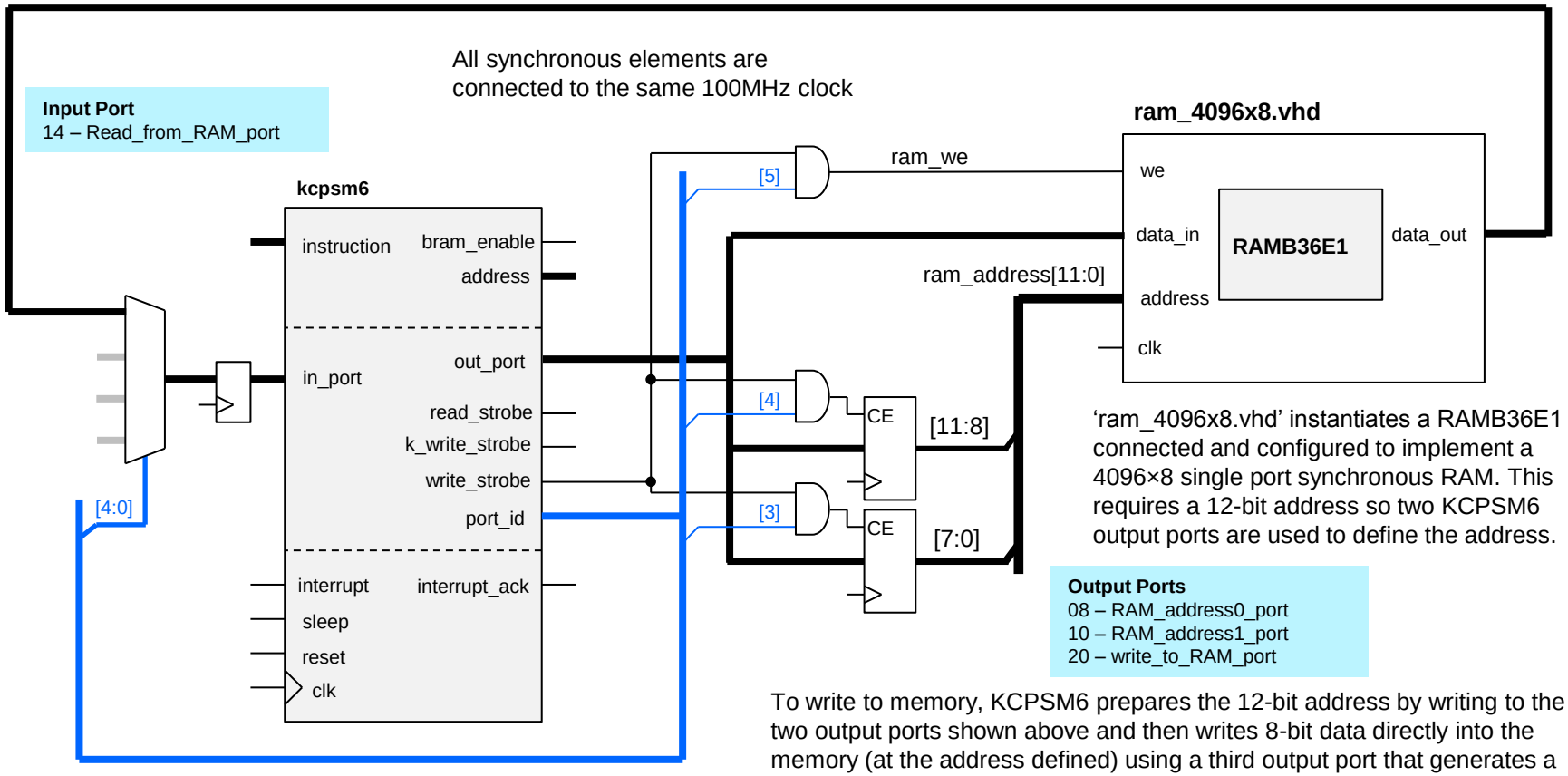
XILINX

Connecting KCPSM6 to BRAM

For clarity this diagram only shows the interface and ports assigned to communicate with a 4096-byte memory implemented by a BRAM (36kb).

The BRAM provides 4096-bytes of storage. As provided, this design only uses 404 bytes but the code is provided with reuse in mind.

To read from memory, KCPSM6 prepares the 12-bit address by writing to two output ports. The 8-bit data stored at that address is then read via an input port. Note that the 'read_byte_from_RAM' routine provided in 'RAM_2048x8_routines.psm' executes one additional instruction prior to the INPUT instruction which reads the memory contents. The additional instruction implements a delay of 2 clock cycles which allows for the synchronous nature of the BRAM.

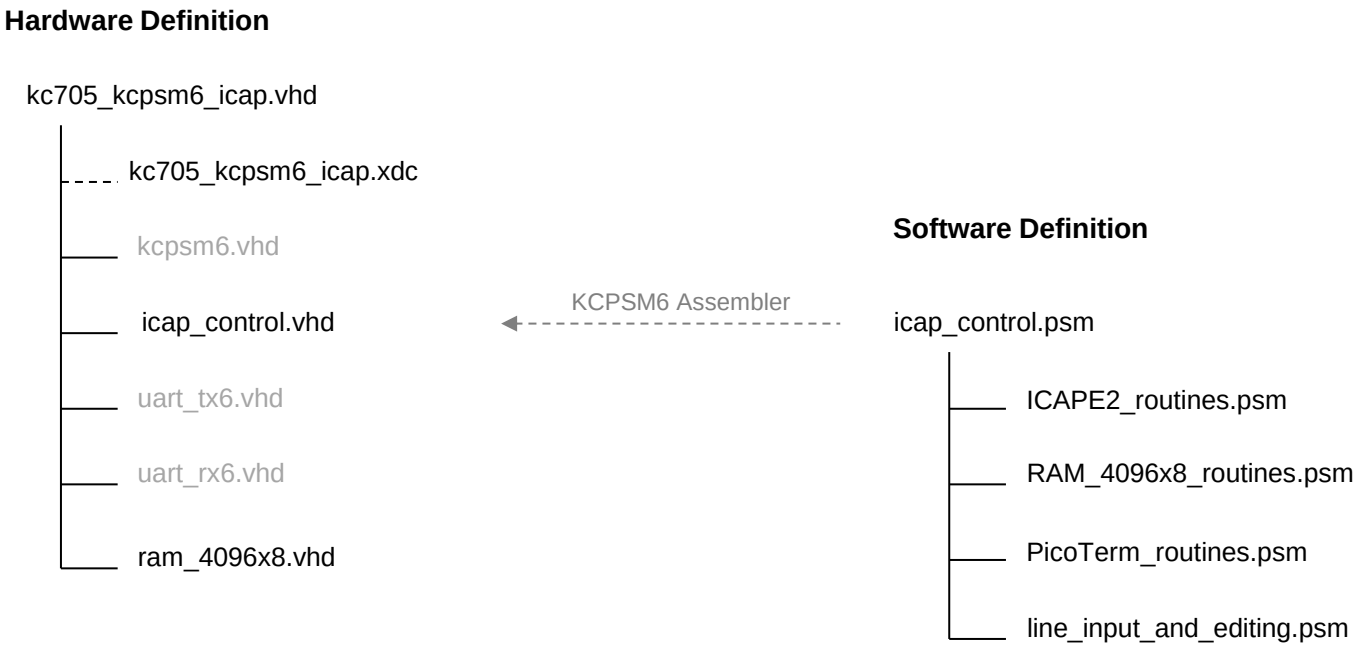


To write to memory, KCPSM6 prepares the 12-bit address by writing to the two output ports shown above and then writes 8-bit data directly into the memory (at the address defined) using a third output port that generates a write strobe ('ram_we').

XILINX

Reference Design Files

All source files contain detailed descriptions and comments. In fact, the descriptions and comments in the source code should be considered the *main* documentation for this reference design with this PDF mainly used to provide an introduction, user notes and complementary graphics.



Files shown in grey are provided in the KCPSM6 package and should be copied and added to your project directory

Hint – The 'icap_control.vhd' file is not provided. Assemble the PSM code in the normal way to generate this file.

Menu Commands - ‘H’ & ‘I’

```
Menu
 H - Display this menu
 I - Read Information (ICAPE2 Registers and CRCERROR status)
```

Menu

KCPSM6 waits for the user to enter a key and then acts on the character. KCPSM6 accepts upper and lower case letters. Use ‘H’ (‘Help’) to display the menu again.

```
> I

 ICAPE2 Registers

  IDCODE = 43651093
    CTL0 = 00000401
    STAT = 46107AFC
    COR0 = 02003FE5
    COR1 = 00000100
  WBSTAR = 00000000
 BOOTSTS = 00000001
```

Information (Reading from ICAPE2 Registers)

The ‘I’ command reads and displays the contents of seven of the configuration registers. The code provided clearly demonstrates the ability of KCPSM6 to read ICAPE2 registers and it would be a straightforward task to modify the code and read any of the remaining registers.

The ‘IDCODE’ and ‘COR1’ registers are of particular interest to the rest of this reference design.

Hint – The ‘7 Series FPGAs Configuration User Guide’ (UG470) describes the purpose of each configuration register. Below is an example table for the ‘COR1’ register. Note how setting Bit8 enables Readback CRC scanning.

Table 5-31: **Configuration Options Register 1**

Description	Reserved	PERSIST_DEASSERT_AT_DESYNC	RBCRC_ACTION	Reserved	RBCRC_NO_PIN	RBCRC_EN	Reserved	BPI_1ST_READ_CYCLE	BPI_PAGE_SIZE
Bit Index	31 30 29 28 27 26 25 24 23 22 21 20 19 18	17	16 15	14 13 12 11 10	9	8	7 6 5 4	3 2	1 0
Value	0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0 0	0 0 0 0 0	0	0	0 0 0 0	0 0	0 0

COR1 = 00000100 hex sets Bit8

NOTES

During the initialization phase of the ‘icap_control.psm’ program, KCPSM6 sets the COR1 register to the ‘00000100” value as shown above. This enables the Readback CRC scanning of the device. As such, you should also observe the LEDs flashing on the board.

GPIO LEDS 

With ‘COR1=00000100’, the device will use the initial Readback CRC scans of the device to calibrate the frame level ECC values and the device level golden CRC value. This defines the configuration image against which any deviations will be detected and reported as errors.

Menu Commands – ‘D’, ‘C’ & ‘N’

```
D - Enable detection only (COR1=00800100)
C - Enable detection and correction (COR1=00810100)
N - Disable detection and correction (COR1=00000000)
```

Writing to an ICAPE2 Register

The ‘D’, ‘C’ and ‘N’ commands write slightly different values to the ‘COR1’ configuration register. The code provided clearly demonstrates the ability of KCPSM6 to write to an ICAPE2 register and it would be a straightforward task to modify the PSM code to write values to other registers.

```
> N
Ok

> I

 ICAPE2 Registers

  IDCODE = 43651093
    CTL0 = 00000401
    STAT = 46107AFC
    COR0 = 02003FE5
    COR1 = 00000000
  WBSTAR = 00000000
 BOOTSTS = 00000001

> D
Ok

> I

 ICAPE2 Registers

  IDCODE = 43651093
    CTL0 = 00000401
    STAT = 46107AFC
    COR0 = 02003FE5
    COR1 = 00800100
  WBSTAR = 00000000
 BOOTSTS = 00000001
```

This sequence shows the ‘N’ and ‘D’ commands being used to set ‘COR1’ to different values.

The ‘I’ command allows us to read back and verify the current value of the ‘COR1’ register.

Hint - The LEDs on the board will stop flashing when Readback CRC scanning of the device is disabled.

GPIO LEDS 

IMPORTANT NOTE (‘Calibration’)

The ‘D’ and ‘C’ options set ‘COR1’ to 00800100 and 00810100 hex respectively. In both cases, these values not only set Bit8 to enable Readback CRC, but they also set Bit23. It is vital to appreciate the significance of Bit23 in relation to the use of the other options in this reference design. Unfortunately, Bit23 is not described in UG470 (v1.8) so a reasonably comprehensive description is contained in the ‘icap_control.psm’ program provided with this design.

As we will see later, the ‘R’, ‘T’ and ‘W’ commands can be used to deliberately modify the contents of the configuration memory with the aim of exercising and observing the error detection and correction capabilities of the device. Bit23 of the ‘COR1’ register must be set in order to tell the device *not* to recalibrate the frame level ECC values and the device level golden CRC value following what has been a deliberate and apparently meaningful change to the configuration of the device. In other words, the device needs to told to ignore the change that has been made so it will then go on to detect that change relative to the original ‘golden’ image and report it as being an error. In contrast, a normal application of partial reconfiguration would form a new valid configuration image and auto-calibration would be desired.

Hint – If you *do* want the device to recalibrate the ECC and CRC values then press the CPU_RST button on the bard so that KCPSM6 executes the initialisation sequence that sets ‘COR1’ to 00000100 hex. However, do be aware that in doing so you will be forcing the device to adopt all your deliberate corruptions as being part of a new valid image!

Menu Commands – ‘L’ & ‘M’

Configuration Frames: Linear Addresses (LA) and Physical Addresses (PA)

If you are familiar with the Soft Error Mitigation Controller (SEM IP) documented in User Guide pg036 then you will already know that each configuration frame is represented by both a Linear Address (LA) and a Physical Address (PA).

This example message generated by the Monitor Interface of the SEM IP core following the detection and correction of a single bit error shows both the Linear and Physical Addresses being reported. From this example captured whilst using the SEM IP on a KC705 board we know that LA=00004C07 corresponds with PA=0044038F in an XC7K325T device.

If you are not familiar with the SEM IP or its documentation (pg036) then you are strongly advised to investigate it. However, an objective of this reference design is to expand your knowledge of error detection and correction schemes so this design should be of use whether you use the SEM IP or not. With that in mind, we need to start by fully understanding Linear and Physical frame addresses.

An example error correction report generated by the SEM IP.

```
SC 04
SED OK
PA 0044038F
LA 00004C07
WD 16 BT 14
COR
WD 16 BT 14
END
FC 00
SC 08
FC 40
SC 02
O>
```

IACPE2 and Frame Addresses

In order to read or write a frame of configuration data it is necessary to know the Physical Address (PA) of that frame within the device. Assuming we do know the Physical Address (PA) it is first written into the Frame Address Register (FAR) as part of an ICAPE2 transaction *before* the actual frame data is either read or written. The issue is that we don’t immediately know what a valid PA is for the device that we are using (except for the example shown above!).

Due to different sizes of device the range of PA values will vary. Furthermore, the differences in device features and the actual physical layout of each device means that the address map of each device contains many ‘holes’ or ‘irregularities’. There is nothing wrong or worrying about these irregularities but they don’t make it any easier to guess which physical addresses (PA) are valid for a given device.

In contrast, the Readback CRC mechanism automatically implements a linear scan of all the configuration frames containing static** information automatically skipping over the ‘holes’ and ‘irregularities’ in a perfectly seamless way. So quite simply, we say that the first frame in the Readback CRC scan has a Linear Address (LA) of zero and then LA just increments for each frame in turn. For example, the 19,464th frame to be scanned would be LA 19,436 or LA=00004C07 which is the frame presented in the example SEM IP report shown above.

In this reference design, KCPSM6 and a small amount of logic connected to the FRAME_ECCE2 primitive (see page 7) exploits the built-in Readback CRC mechanism in order to determine the Physical Address (PA) corresponding with any Linear Address (LA).

** Readback CRC ignores frames associated with BRAM contents as they are typically associated with variable data during operation. Each BRAM has a local ECC option for data protection.

Menu Commands – ‘L’ & ‘M’

```
L - Look up Physical frame Address (PA) corresponding with Linear Address (LA)
```

The ‘L’ command prompts the user to enter a Linear Address (LA) within the range of the device being used.

```
> L

Enter a Linear Address (LA) in the range 00000000 to 00005AE1 Hex
4c07

LA=00004C07
PA=0044038F
```

Hint - KCPSM6 implements a very simple line editor allowing backspace to be used to modify the value before it is entered. For more details or to reuse this code please see ‘line_input_and_editing.psm’.

This example confirms that KCPSM6 has been able to determine that the Physical Address corresponding with a Linear Address of LA=00004C07 is PA=0044038F. This matches with the SEM IP report shown on the previous page.

In simple terms (and it isn’t complicated!), KCPSM6 waits for the start of a new Readback CRC scan to begin and then counts the number of SYNDROMEVALID pulses until the LA value is reached. It then reads the PA directly from the ‘FAR’ output of the FRAME_ECCE2 primitive before the Readback CRC advances to the next frame. For full details please read the comprehensive descriptions contained in ‘icap_control.psm’.

The conversion relies on Readback CRC scanning the device so KCPSM6 first checks to see that scanning is active and will generate an error message if it is not (i.e. if you previously used the ‘N’ command).

```
> L
Failed
Readback CRC scanning must be enabled to use this command
```

The ‘M’ command will automatically generate a complete Readback CRC memory map of the whole device.

```
M - Generate address map of all Readback CRC frames in device
```

```
> M

   [LA]       [PA]

00000000 = 00000000
00000001 = 00000001
00000002 = 00000002
...
00004C06 = 0044038E
00004C07 = 0044038F
00004C08 = 00440390
...
00005AE0 = 01C40300
00005AE1 = 02000000

>
```

Due to the large number of frames in a device, the ‘M’ command will take quite some time to complete! For example, it takes over 9 minutes to generate the map of an XC7K325T device. Fortunately you only need to map a device once because everything is captured in a PicoTerm LOG file that was automatically opened by KCPSM6 and PicoTerm at the start of the session (see page 5). So having used the ‘M’ command once, simply open the LOG file (e.g. a file with a name similar to ‘PicoTerm_05Sep2014_121306.txt’) in a text editor and extract the memory map for your future reference.

Hint - If you accidentally execute the ‘M’ command then you can press the ‘CPU_RST’ button to escape!

Menu Commands – ‘F’ & ‘R’

```
F - Specify physical address of target frame (used to set FAR)
```

Once you know a valid Physical Address (PA) it can be specified using the ‘F’ command. This command only informs KCPSM6 which frame you are interested in and it remembers this value in its scratch pad memory. KCPSM6 will then load this value into the Frame Address Register (FAR) as part of the ICAPE2 transactions used in the frame read (‘R’) and frame write (‘W’) commands.

```
> F

Enter a 32-bit Physical Address (PA)
44038f

 FAR = 0044038F
```

Hint – Using the backspace key, the physical address can be modified prior to entry. KCPSM6 will accept upper and lower case characters and verify that they are valid hexadecimal digits. KCPSM6 will also accept any number of digits up to the maximum of 8 expected for a Physical Address value. For more details or to reuse this code please see ‘line_input_and_editing.psm’.

```
R - Read target configuration frame to RAM buffer
```

The ‘R’ command reads the specified frame of configuration data out of configuration memory via ICAPE2 and stores it into the RAM buffer (BRAM) connected to KCPSM6. Each frame consists of 101 words of 32-bits requiring 404-bytes of memory (i.e. too large to be stored in scratch pad memory). If the intricacies of the ICAPE2 transaction required to read a frame of configuration data is of interest to you (it’s quite involved!), then please see the detailed descriptions provided in ‘icap_control.psm’.

```
> R

 FAR = 0044038F

00:00000000 01:00000000 02:00000000 03:00000000 04:00000000 05:00000000 06:00000000 07:00000000 08:00000000 09:00000000
0A:00000000 0B:00000000 0C:00000000 0D:00000000 0E:00000000 0F:00000000 10:00000000 11:00000000 12:00000000 13:00000000
14:00000000 15:00000000 16:00000000 17:00000000 18:00000000 19:00000000 1A:00000000 1B:00000000 1C:00000000 1D:00000000
1E:00000000 1F:00000000 20:00000000 21:00000000 22:00000000 23:00000000 24:00000000 25:00000000 26:00000000 27:00000000
28:00000000 29:00000000 2A:00000000 2B:00000000 2C:00000000 2D:00000000 2E:00000000 2F:00000000 30:00000000 31:00000000
32:00000000
33:00000000 34:00000000 35:00000000 36:00000000 37:00000000 38:00000000 39:00000000 3A:00000000 3B:00000000 3C:00000000
3D:00000000 3E:00000000 3F:00000000 40:00000000 41:00000000 42:00000000 43:00000000 44:00000000 45:00000000 46:00000000
47:00000000 48:00000000 49:00000000 4A:00000000 4B:00000000 4C:00000000 4D:00000000 4E:00000000 4F:00000000 50:00000000
51:00000000 52:00000000 53:00000000 54:00000000 55:00000000 56:00000000 57:00000000 58:00000000 59:00000000 5A:00000000
5B:00000000 5C:00000000 5D:00000000 5E:00000000 5F:00000000 60:00000000 61:00000000 62:00000000 63:00000000 64:00000000
```

KCPSM6 confirms the Physical Frame address that it read and then displays the frame contents that have been copied into the RAM buffer.

Menu Commands – ‘T’ & ‘B’

```
T - Toggle bit in RAM buffer
```

The ‘T’ command enables you to toggle the state of any bit of the frame data held in the RAM buffer (i.e. toggle a ‘0’ to become a ‘1’ or toggle a 1’ to become a ‘0’). The command prompts you to specify which one of the 32-bits in which one of the 101 words is to be toggled. Note that the change is only made to the contents of the RAM buffer (i.e. the actual device configuration is not changed until you use the write frame command).

```
> T
Specify the bit to be toggled in the RAM buffer

Enter a Word (WD) in the range 00 to 64
16

Enter a Bit (BT) in the range 00 to 1F
14

WD=16  BT=14
Ok
```

This example continues to shadow the SEM IP report shown on page 12. The bit to be toggled is contained in Word (WD) 16 hex and is Bit (BT) 14 hex. As before, KCPSM6 allows you to edit the values before you enter them and it will check that that they are both valid hexadecimal values in the required ranges before allowing you to continue.

Hint – The frame display (see below) identifies each word (WD) in blue next to the hexadecimal value of each 32-bit word displayed in black.

```
B - Display frame held in RAM buffer
```

The ‘B’ command allows you to see the current contents of the RAM buffer and confirm the changes that you have made.

This example shows that Word 16 is now value 00100000 hex = 0000 0000 0001 0000 0000 0000 0000 0000 in binary.
Hence Bit20 (14 hex) has been toggled from a ‘0’ (see previous page) to a ‘1’.
Note that this image shows what appears to be an ‘empty’ frame; other frames can look busy; toggling can change a ‘1’ to ‘0’ too!

```
> B

00:00000000 01:00000000 02:00000000 03:00000000 04:00000000 05:00000000 06:00000000 07:00000000 08:00000000 09:00000000
0A:00000000 0B:00000000 0C:00000000 0D:00000000 0E:00000000 0F:00000000 10:00000000 11:00000000 12:00000000 13:00000000
14:00000000 15:00000000 16:00100000 17:00000000 18:00000000 19:00000000 1A:00000000 1B:00000000 1C:00000000 1D:00000000
1E:00000000 1F:00000000 20:00000000 21:00000000 22:00000000 23:00000000 24:00000000 25:00000000 26:00000000 27:00000000
28:00000000 29:00000000 2A:00000000 2B:00000000 2C:00000000 2D:00000000 2E:00000000 2F:00000000 30:00000000 31:00000000
32:00000000
33:00000000 34:00000000 35:00000000 36:00000000 37:00000000 38:00000000 39:00000000 3A:00000000 3B:00000000 3C:00000000
3D:00000000 3E:00000000 3F:00000000 40:00000000 41:00000000 42:00000000 43:00000000 44:00000000 45:00000000 46:00000000
47:00000000 48:00000000 49:00000000 4A:00000000 4B:00000000 4C:00000000 4D:00000000 4E:00000000 4F:00000000 50:00000000
51:00000000 52:00000000 53:00000000 54:00000000 55:00000000 56:00000000 57:00000000 58:00000000 59:00000000 5A:00000000
5B:00000000 5C:00000000 5D:00000000 5E:00000000 5F:00000000 60:00000000 61:00000000 62:00000000 63:00000000 64:00000000
```

Menu Commands – ‘W’
Experiments – Error Detection

```
W - Write RAM buffer to target configuration frame
```

The ‘W’ command will write the frame of information currently contained in the RAM buffer into the configuration memory of the device at the Physical Address (PA) previously defined using the ‘F’ command. If the intricacies of the ICAPE2 transaction required to write a frame of configuration data is of interest to you (it’s even more involved than a read transaction!), then please see the detailed descriptions provided in ‘icap_control.psm’.

```
> W

FAR = 0044038F
Ok
```

KCPSM6 confirms the Physical Frame address that it has written the contents of the RAM buffer to.

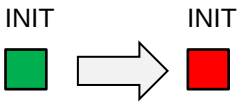
WARNING! – This design provides you with the ability to write any information to any frame. Hence, it also provides you with the potential to configure the device with illegal patterns that could ultimately stress the device. It is highly unlikely that you would ever want to do this intentionally. In practice, single event upsets (SEU) rarely flip more than one bit at a time and therefore most of your experiments would be expected to emulate similar events (i.e. only toggle one bit in a frame and then allow the device to correct it before toggling another one). However, this warning should remind you that you need to be thoughtful and logical when conducting any experiments especially if you modify the PSM code and/or create a PC based application to automate the injection of errors. In general, you should avoid creating situations in which large numbers of erroneous bits are present in the device at the same time; it just isn’t a situation that would occur in reality! Hint - Be careful *not* to copy the contents of one frame to another (i.e. using the ‘F’ command to change the Physical Address before using the ‘W’ command) as this has the potential to change a large number (theoretically all) of the bits in the frame in one go.

Experiment - Error Detection

Hopefully it is now clear how you can use the ‘F’, ‘R’, ‘T, and ‘W’ commands to flip the state of a single bit in a configuration frame. Doing so emulates the most common, but still very rare, type of single event upset (SEU). This processes is generally referred to as ‘error injection’ and is one of the important features provided by the SEM IP core as facilitated by this design.

```
> D
Ok
```

Firstly use the ‘D’ command to enable Readback CRC detection only. If you haven’t already done so, read the important note on page 11 regarding ‘calibration’. In order to deliberately inject an error you must use the ‘D’ or ‘C’ commands that appropriately set the ‘COR1’ register to disable the automatic calibration of the ECC and CRC values.



Then use the ‘F’, ‘R’, ‘T, ‘B’ and ‘W’ commands in the ways shown previously to flip one bit of a frame and write it back into the device. As you invoke the ‘W’ command you should expect to see the INIT LED on the KC705 board change from green to red as the error is detected by the Readback CRC circuit. The ‘I’ command will additionally display ‘CRCERROR’ reflecting the status of the internal signal KCPSM6 reads from the FRAME_ECCE2 primitive. If the INIT LED remains green, first check that you used the ‘D’ command prior to your frame write (see LOG file or try again). Secondly, not all bits of every frame can be flipped so use ‘R’ to read back the frame contents and see if you actually managed to change the state of that bit in *configuration* memory.

```
> I
CRCERROR
```

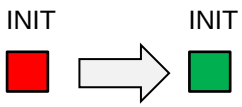
Experiments – Error Correction

Experiment - Error Correction

```
> I

 CRCERROR

 ICAPE2 Registers

  IDCODE = 43651093
    CTL0 = 00000401
    STAT = 46107AFC
    COR0 = 02003FE5
    COR1 = 00800100
  WBSTAR = 00000000
 BOOTSTS = 00000001

> C
Ok


> I

 ICAPE2 Registers

  IDCODE = 43651093
    CTL0 = 00000401
    STAT = 46107AFC
    COR0 = 02003FE5
    COR1 = 00810100
  WBSTAR = 00000000
 BOOTSTS = 00000001
```

Having successful injected a single bit error in one of the frames (see previous page) whilst operating in detection only mode the INIT LED on the KC705 will be red and the 'I' command will report 'CRCERROR' as well as displaying register values.

Now use the 'C' command to enable Readback CRC detection with ECC correction.



You should expect to see the INIT LED on the KC705 board return to green as the built in error correction mechanism of the device detects the erroneous frame and automatically uses the ECC syndrome to restore the corrupted bit to its original value.

The 'I' command confirms that the internal CRC signal is also cleared and you can further convince yourself of the correction using 'R' to read back and manually verify the frame contents following its correction by the device.

Note - If you inject an error whilst error correction is enabled, the error will be detected and corrected almost immediately and you will left with the impression that error injection is not working! In fact, any errors that you are injecting are being detected and corrected so quickly (i.e. in less than one Readback CRC period) and you just don't see anything. In fact, you are actually experiencing how quickly the device would detect and correct real SEUs ☺.

Experiments – Multiple Bit Errors

Simulating Correctable Multiple Bit Errors

As described in the '7 Series FPGAs Configuration User Guide' (UG470), the frame ECC is able to facilitate the correction of a single bit error. Very occasionally an SEU may lead to a double bit upset in which the contents of two adjacent configuration memory cells are flipped at the same time. Xilinx have designed for this rare situation by physically interleaving the cells of pairs of frames such that a typical double bit upset will take on the appearance of two single bit errors which can each be detected and corrected in turn.

Experiment – 1) Use the 'D' command to place the device into detection only mode.
We need to be in this mode whilst we inject two errors.

2) Use the 'F', 'R', 'T, and 'W' commands to flip one bit in a frame and write it back into the device.
E.g. FAR=0044038F, WD=16 and BT=14 as shown in the examples on the previous pages.
Note that the INIT LED turns red as this error is detected by the device level CRC mechanism.

3) Use the 'F', 'R', 'T, and 'W' commands to flip one bit in a different frame and write it back into the device. To most accurately simulate a real double bit error, flip the same bit of the adjacent frame (i.e. the next Linear Address converted to Physical Address).
E.g. FAR=00440390, WD=16 and BT=14.
Note that writing has no obvious effect because the INIT LED is already red so you may wish to use 'R' to read back the frame and convince yourself that you really did inject a second error in the second frame.

4) Then use the 'C' command to place the device into detection and correction mode.
The INIT LED will return to green indicating the correction of both bits (one after the other very quickly).
Use the 'F' and 'R' commands to read back both frames and convince yourself that both errors were indeed corrected.

Simulating Non-Correctable Multiple Bit Errors

Due to the physical interleaving of memory cells from different frames, the probability of a double bit error occurring within the same frame is low but it can occur. The SEM IP has an 'enhanced repair' option that augments the standard frame ECC to address these rare cases but the standard ECC-based scheme built in to the devices does not have this feature and any multiple bit (i.e. 2 or more) error in the same frame can not be corrected.

Experiment – Use the 'T' command twice to toggle two bits in a frame held in the RAM buffer *before* writing it back into the device. This will quickly show that the device is unable to correct this type of error and the INIT LED will turn red even when correction is enabled.

E.g. `16:00300000` = 0000 0000 00**11** 0000 0000 0000 0000 0000 in binary

Experiment – Using the 'T' command again, manually correct the errors that you created in the RAM buffer and then write this valid frame back into the device. The INIT LED returns to green indicating that the error has been removed. This illustrates how the SEM IP 'repair by replace' strategy can correct virtually any upset simply by taking a copy of the original frame definition and writing it into the device.