Voltage Identification (VID) For Virtex-7 Devices

A KCPSM6 Reference Design for the VC707 Evaluation Board

Ken Chapman

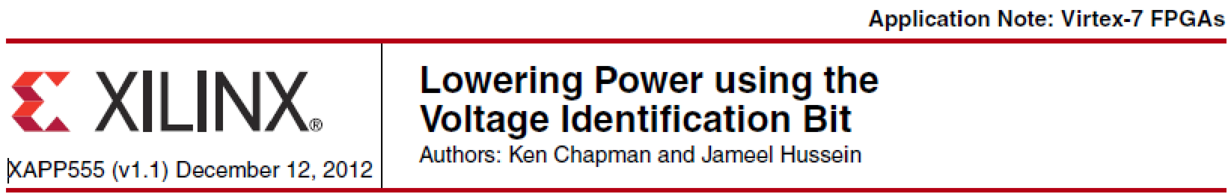
18th March 2014

Disclaimer

This Document

This document is provided as a *supplement* to XAPP555 ‘Lowering Power using the Voltage Identification Bit’ which can be obtained from the Xilinx web site…

http://www.xilinx.com/support/documentation/application_notes/xapp555-Lowering-Power-Using-VID-Bit.pdf



Application Note: Virtex-7 FPGAs

XILINX

XAPP555 (v1.1) December 12, 2012

Lowering Power using the Voltage Identification Bit

Authors: Ken Chapman and Jameel Hussein

Use the link inside XAPP555 to download the reference design source files.

Hopefully primary focus of this application note will be if interest to you, but regardless, it can be seen that the reference design to accompany the application note is presented on a VC707 board and includes KCPSM6 and the UART macros. In fact the design contains over 1,000 KCPSM6 processors and this supplement will make it clearer why!

As shown in this overview figure taken form XAPP555, the principle KCPSM6 is focused on reading **DEVICE_DNA**, implementing **PMBus** (to control and monitor power supplies) and UART based communication with a terminal.

Reference Design

The reference design files for this application note can be downloaded from:

https://secure.xilinx.com/webreg/clickthrough.do?cid=185933

Figure 3 shows an overview of the reference design presented on the VC707 evaluation kit.

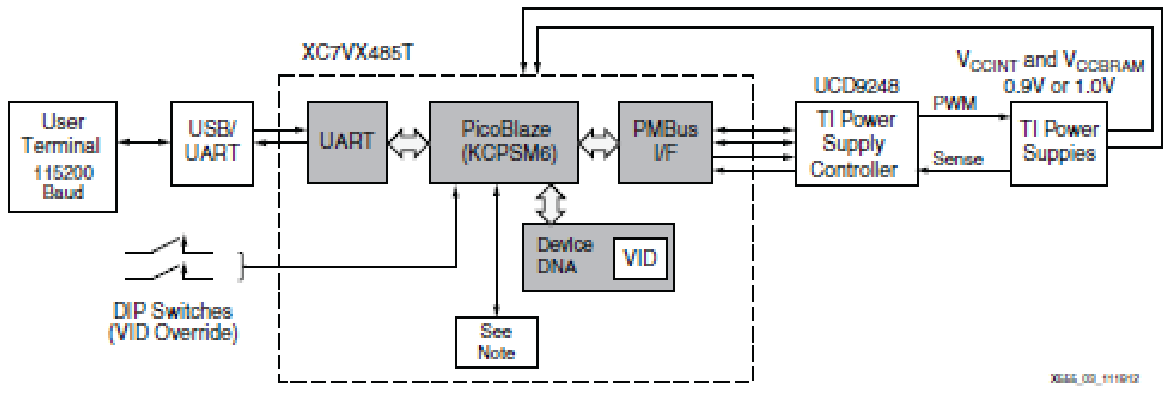


Figure 3: **Overview of the Reference Design on a VC707 Board**

XILINX

This Document

As will be seen in the remainder of this document, the reference design also includes features that demonstrate the effect that the 'VID' technique will have on a design. This involves a set of 1,000 identical 'PCAST' Modules that also contain KCPSM6 and the UART macros. In terms of reuse, these modules show how KCPSM6 could be used for the **distributed control and monitoring** of functions within a large device. The main KCPSM6 also implements **I2C** to control the **Si570 Programmable Oscillator** on the board.

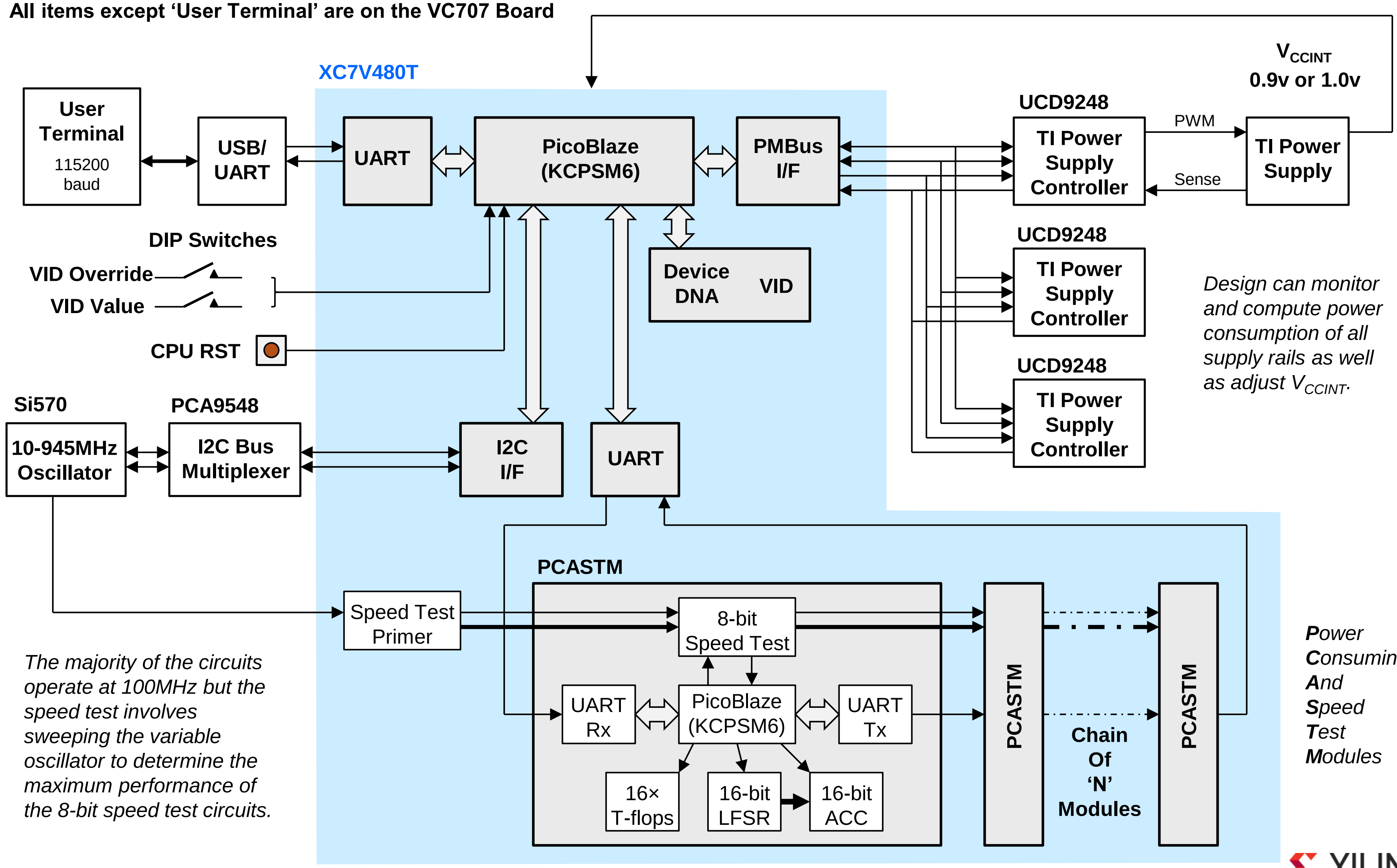
One way or another, this design contains and illustrates a variety of ways in which KCPSM6 can be used. Many of the functions implemented are not directly associated with the 'VID' technique or are equally suitable for other applications.

The 'VID' Reference Design source files contain full text descriptions of the design as a whole as well as localised descriptions relating to each section. This document is provided mainly as a pictorial supplement to further aid your understanding of the design and to facilitate reuse of any sections in your designs. The 'README.txt' file provided in the XAPP555 download introduces the reference design, lists all the source files and the describes how to begin using the design on your VC707 board.

This design is dedicated to the memory of Jeffers Emmanuel.
Thank you for helping me to implement this design.
We all miss you.

Expanded Overview of Reference Design

All items except 'User Terminal' are on the VC707 Board



Design can monitor and compute power consumption of all supply rails as well as adjust V_{CCINT}.

The majority of the circuits operate at 100MHz but the speed test involves sweeping the variable oscillator to determine the maximum performance of the 8-bit speed test circuits.

***P**ower **C**onsuming **A**nd **S**peed **T**est **M**odules*

Design Facts and Figures: XC7VX485T Device

```
Design Information:
-------------------
Target Device   : xc7vx485t
Target Package  : ffg1761
Target Speed    : -1
```

The BIT file provided (vc707_485t_vid_reference_design.bit) defines a design containing **1,000 PCASTM** occupying the majority of the device. Below are the key facts figures from the implementation reports.

'VID' is only applicable to the '-1C' grade. The device on your board may be of a different grade but will still facilitate experiments that will provide you with power consumption and performance results indicating the relative effects of operating VCCINT at a lower voltage.

```
Design Summary
--------------

   Number of occupied Slices:   63,728 out of  75,900   83%

Number of RAMB36E1/FIFO36E1s:    1,002 out of   1,030   97%

   Number of Slice Registers:  214,271 out of 607,200   35%

        Number of Slice LUTs:  199,173 out of 303,600   65%
```

Remember that the 'VID' process is tiny; the 1,000 PCASTM are to facilitate experiments and to be representative of the logic of a real design that almost completely fills the device.

~83% Slices occupied

199,173 LUTs

214,271 flip-flops

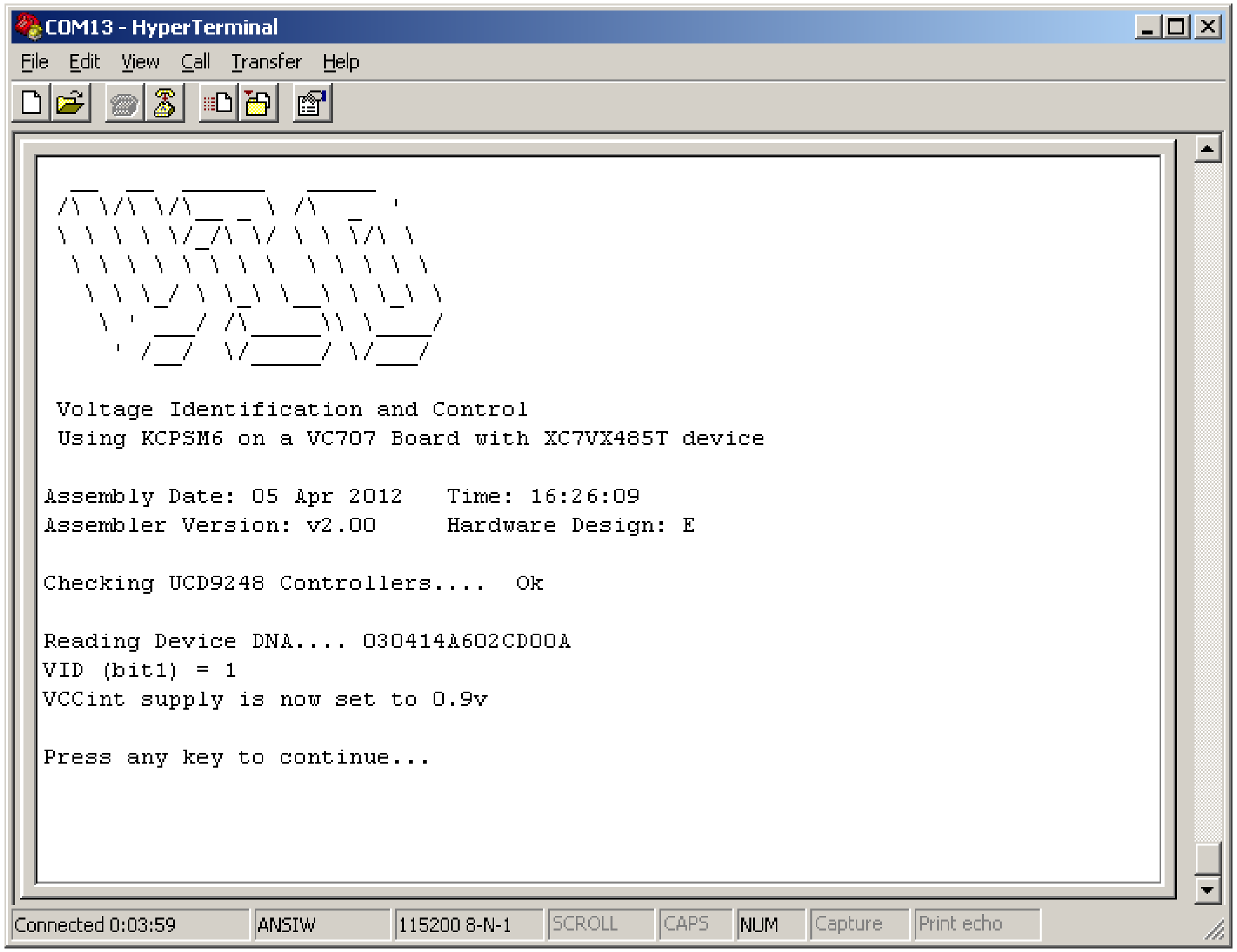
~97% BRAM occupied

The speed test circuits were given a 500MHz specification which ISE v13.4 just failed to achieve using the default settings. However, for the purpose of experimentation using this reference design, all that matters is that the tools have indicated a minimum performance of 471MHz for the 'speed test' circuits in a '-1C' device. This is the clock frequency that should en exceeded when running the speed test at 0.9v and 1.0v depending on the 'VID' capability of your device.

```
-----------------------------------------------------------------------------------------------------
  Constraint                                   |    Check     | Worst Case |  Best Case | Timing |    Timing
                                               |              |    Slack   | Achievable | Errors |     Score
-----------------------------------------------------------------------------------------------------
* TS_fast_clk = PERIOD TIMEGRP "fast_clk" 2 | SETUP        |    -0.123ns|     2.123ns|     127|         6757
   ns HIGH 50%                                 | HOLD         |     0.004ns|            |       0|            0
-----------------------------------------------------------------------------------------------------
```

The VID Process in Action

This screen capture shows the initial output from the reference design covering the actual ‘VID’ process....



```
  __    __  ______    _____
 /\ \/\ \/\__ _ \ /\  _  '
 \ \ \ \ \/_/\ \/ \ \ \/\ \
  \ \ \ \ \ \ \ \  \ \ \ \ \
   \ \ \_/ \ \_\ \__\ \ \_\ \
    \ ' ___/ /\_____\\ \____/
     ' /__/  \/_____/ \/___/

 Voltage Identification and Control
 Using KCPSM6 on a VC707 Board with XC7VX485T device

Assembly Date: 05 Apr 2012   Time: 16:26:09
Assembler Version: v2.00     Hardware Design: E

Checking UCD9248 Controllers....  Ok

Reading Device DNA.... 030414A602CD00A
VID (bit1) = 1
VCCint supply is now set to 0.9v

Press any key to continue...
```

```
Connected 0:03:59   ANSIW   115200 8-N-1   SCROLL   CAPS   NUM   Capture   Print echo
```

Even this simple illustrates a simple example of what PicoBlaze (KCPSM6) can implement in less than 50 Slices of logic.

Comprehensive version tracking is built in to the PicoBlaze solution.

Check that power supplies are accessible ready for any change.

Read and validate DNA and extract VID bit.

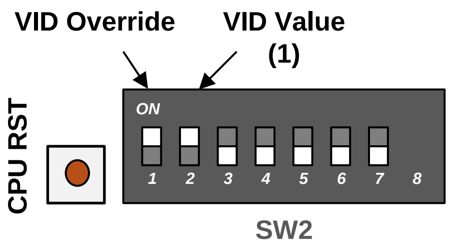
Reduce V_{CCINT} to 0.9v when VID=1.
(PMBus control of supply)

Hint – The VID is Bit1 of the DVA value. In this case the least significant hex digit of the DNA is ‘A’ which is the bit pattern “10**1**0” so Bit1 is indeed a ‘1’.

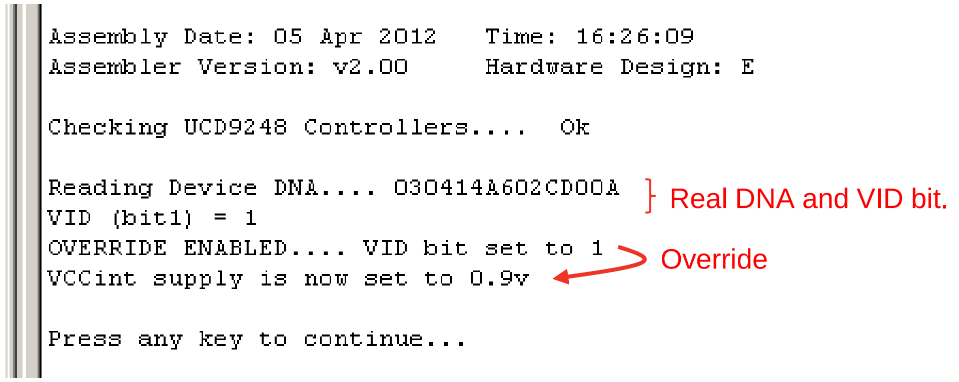
VID Emulation Feature

The Device DNA in your device is unique and fixed (factory programmed) so the VID bit is also fixed in the device that you have on your board. As a result, the correct response of the VID process will always be to set V_{CCINT} to the same voltage. Whilst this is correct behaviour (as shown on the previous page) it makes for a boring reference design and limits the value of conducting experiments.

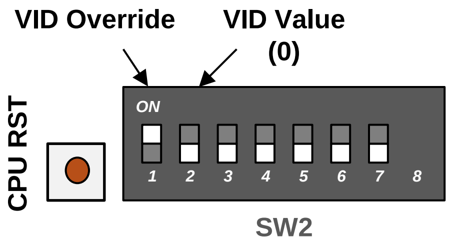
The VID emulation feature allow you to override the actual value of the VID bit in your device and force the process to continue setting V_{CCINT} supply according to the value you have defined using the DIP switches.

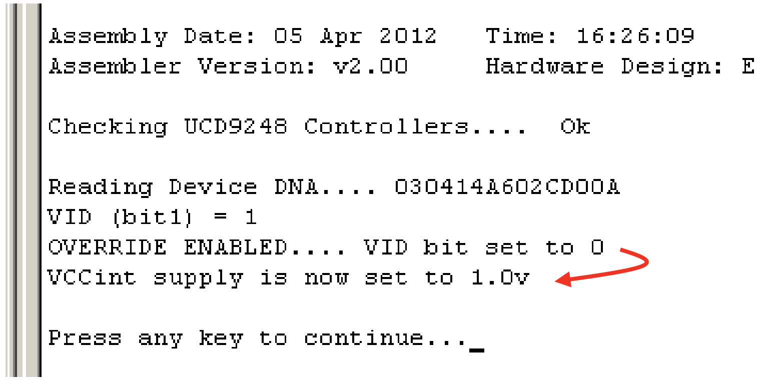


Set the DIP switches and then press the 'CPU RST' button to emulate a power cycle and a device with different VID bit value.



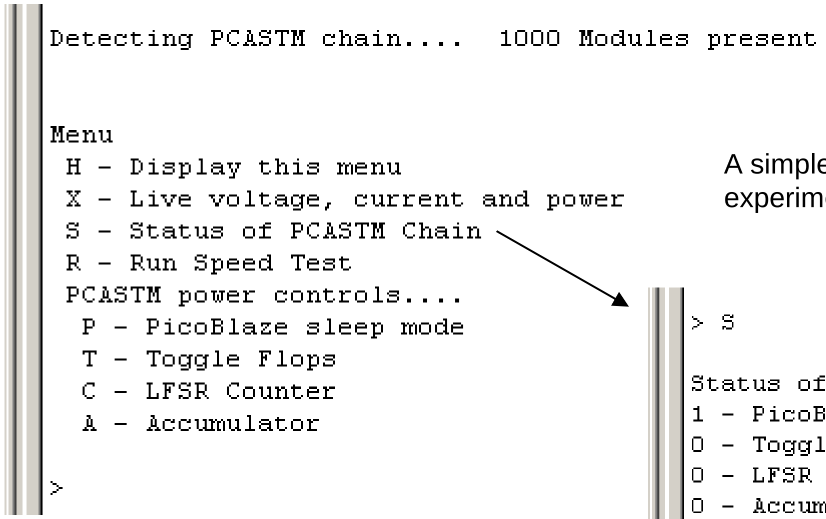
```
Assembly Date: 05 Apr 2012    Time: 16:26:09
Assembler Version: v2.00      Hardware Design: E

Checking UCD9248 Controllers....  Ok

Reading Device DNA.... 030414A602CD00A
VID (bit1) = 1
OVERRIDE ENABLED.... VID bit set to 1
VCCint supply is now set to 0.9v

Press any key to continue...
```





```
Assembly Date: 05 Apr 2012    Time: 16:26:09
Assembler Version: v2.00      Hardware Design: E

Checking UCD9248 Controllers....  Ok

Reading Device DNA.... 030414A602CD00A
VID (bit1) = 1
OVERRIDE ENABLED.... VID bit set to 0
VCCint supply is now set to 1.0v

Press any key to continue..._
```

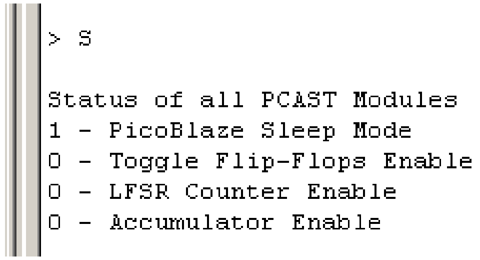
PCASTM Chain and Static Power



```
Detecting PCASTM chain....  1000 Modules present


Menu
 H - Display this menu
 X - Live voltage, current and power
 S - Status of PCASTM Chain
 R - Run Speed Test
 PCASTM power controls....
  P - PicoBlaze sleep mode
  T - Toggle Flops
  C - LFSR Counter
  A - Accumulator

>
```

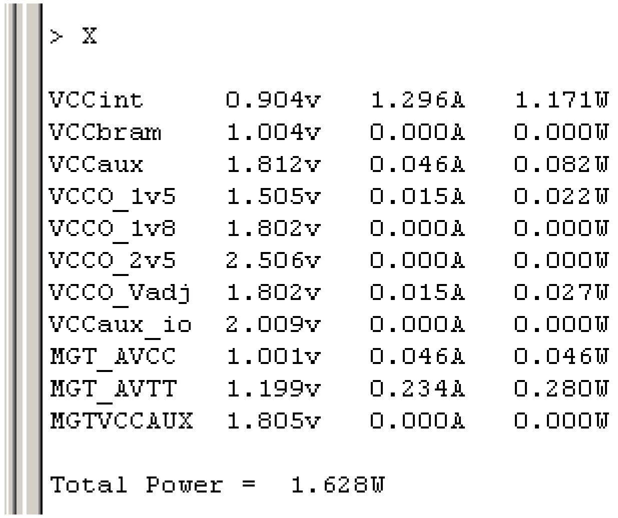
The design tells you how many Power Consuming AND Speed Test Modules (PCASTM) are contained in the design. The supplied BIT file contains 1,000 PCASTM in the chain.

A simple menu of options are then presented and which you can use to conduct your experiments. PicoBlaze accepts upper or lower case characters.



```
> S

Status of all PCAST Modules
1 - PicoBlaze Sleep Mode
0 - Toggle Flip-Flops Enable
0 - LFSR Counter Enable
0 - Accumulator Enable
```

The initial status should confirm that all the PCASTM are in their lowest power state. Whilst there is some activity associated with the main controller and distribution of some clock the device is otherwise close to being in a static state.



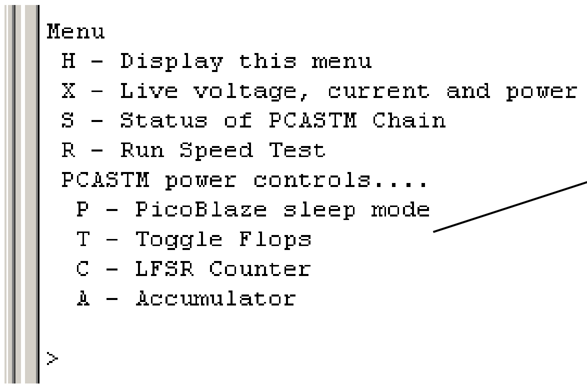
```
> X

VCCint      0.904v   1.296A   1.171W
VCCbram     1.004v   0.000A   0.000W
VCCaux      1.812v   0.046A   0.082W
VCCO_1v5    1.505v   0.015A   0.022W
VCCO_1v8    1.802v   0.000A   0.000W
VCCO_2v5    2.506v   0.000A   0.000W
VCCO_Vadj   1.802v   0.015A   0.027W
VCCaux_io   2.009v   0.000A   0.000W
MGT_AVCC    1.001v   0.046A   0.046W
MGT_AVTT    1.199v   0.234A   0.280W
MGTVCCAUX   1.805v   0.000A   0.000W

Total Power =  1.628W
```

The 'X' option makes PicoBlaze read all the power supplies and compute the power consumption of each rail as well as the total power consumption. Of most interest is the V_{CCINT} supply.

In this example we can see the V_{CCINT} supply has been reduced to 0.9v and the 'static' power consumption is 1.171W.

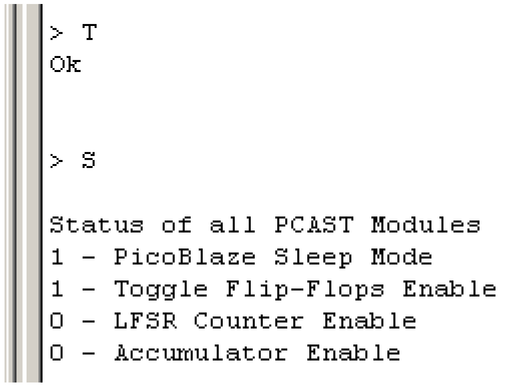
Experiment – Try the device on your board (P) and see what difference supply voltage (V) makes to static power consumption by using the 'VID Override' feature.

Dynamic Power



```
Menu
 H - Display this menu
 X - Live voltage, current and power
 S - Status of PCASTM Chain
 R - Run Speed Test
 PCASTM power controls....
  P - PicoBlaze sleep mode
  T - Toggle Flops
  C - LFSR Counter
  A - Accumulator

>
```

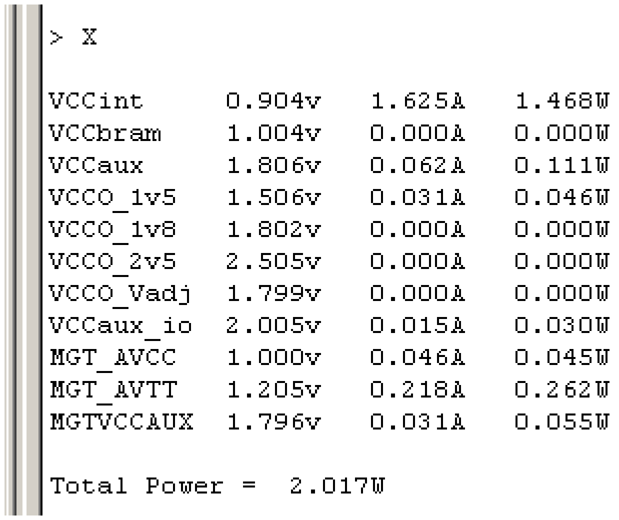
Use the 'P', 'T', 'C' and 'A' options to toggle the enable state of each of the power consuming features in *all* the PCASTM as desired. The status command ('S') can be used to confirm the operational state of all PCASTM in the chain.



```
> T
Ok


> S

Status of all PCAST Modules
1 - PicoBlaze Sleep Mode
1 - Toggle Flip-Flops Enable
0 - LFSR Counter Enable
0 - Accumulator Enable
```

The descriptions contained in the source files describe at some length the ways in which the various power consuming features can be used to evaluate power consumption relative to toggle rates.

In this case 1000 × 16 = 16,000 flip-flops will now be toggling with a 100MHz clock and increase the power consumption.



```
> X

VCCint      0.904v   1.625A   1.468W
VCCbram     1.004v   0.000A   0.000W
VCCaux      1.806v   0.062A   0.111W
VCCO_1v5    1.506v   0.031A   0.046W
VCCO_1v8    1.802v   0.000A   0.000W
VCCO_2v5    2.505v   0.000A   0.000W
VCCO_Vadj   1.799v   0.000A   0.000W
VCCaux_io   2.005v   0.015A   0.030W
MGT_AVCC    1.000v   0.046A   0.045W
MGT_AVTT    1.205v   0.218A   0.262W
MGTVCCAUX   1.796v   0.031A   0.055W

Total Power =  2.017W
```

In comparison with the 'static' power consumption of 1.171W shown on the previous page then this figure indicates an increase of dynamic power consumption of 0.297W (equivalent to 18.5µW per toggle flip-flop).

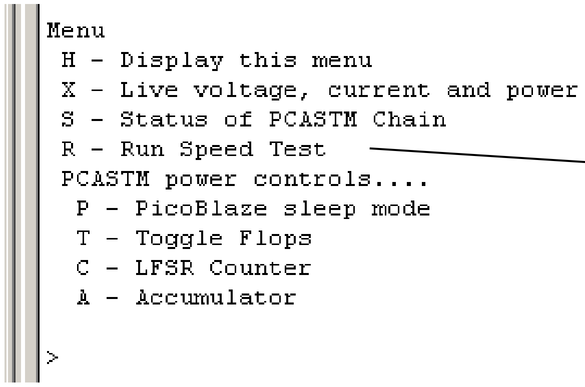
Experiments

Try different combinations of features understand more about toggle rates and the impact on power consumption. For example, what is the equivalent toggle rate of a 16-bit LFSR counter relative to 16 toggle flip-flops?

Compare dynamic power consumption at 0.9v and1.0v.

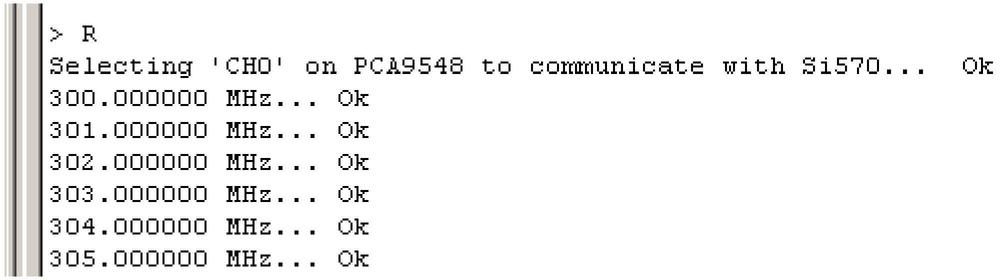
Caution - Do not allow the device to over heat!

Speed Test



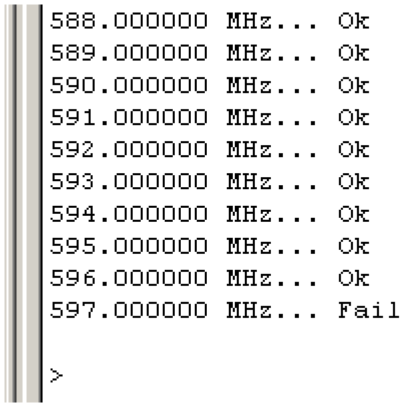
```
Menu
 H - Display this menu
 X - Live voltage, current and power
 S - Status of PCASTM Chain
 R - Run Speed Test
 PCASTM power controls....
  P - PicoBlaze sleep mode
  T - Toggle Flops
  C - LFSR Counter
  A - Accumulator

>
```

During the speed test, PicoBlaze (KCPSM6) controls the I2C bus multiplexer (PCA9528) on the board and establishes communication with the Si570 programmable oscillator which it then controls.



```
> R
Selecting 'CH0' on PCA9548 to communicate with Si570...   Ok
300.000000 MHz... Ok
301.000000 MHz... Ok
302.000000 MHz... Ok
303.000000 MHz... Ok
304.000000 MHz... Ok
305.000000 MHz... Ok
```

Starting at 300MHz and increasing in 1MHz increments, PicoBlaze performs a speed test at each clock frequency and reports the result. When eventually the frequency is too high for one or more of the speed testing circuits to operate correctly the test run ends. In this example the design was good at all frequencies up to 596MHz.



```
588.000000 MHz... Ok
589.000000 MHz... Ok
590.000000 MHz... Ok
591.000000 MHz... Ok
592.000000 MHz... Ok
593.000000 MHz... Ok
594.000000 MHz... Ok
595.000000 MHz... Ok
596.000000 MHz... Ok
597.000000 MHz... Fail

>
```

For the BIT file provided, the minimum performance for the 'speed test' circuits in a '-1C' device was reported by the implementation tools to be 471MHz (see page 5). This result clearly shows that this worst case figure was exceeded and in this case the VID capable device was operating with a V_{CCINT} set to 0.9v.

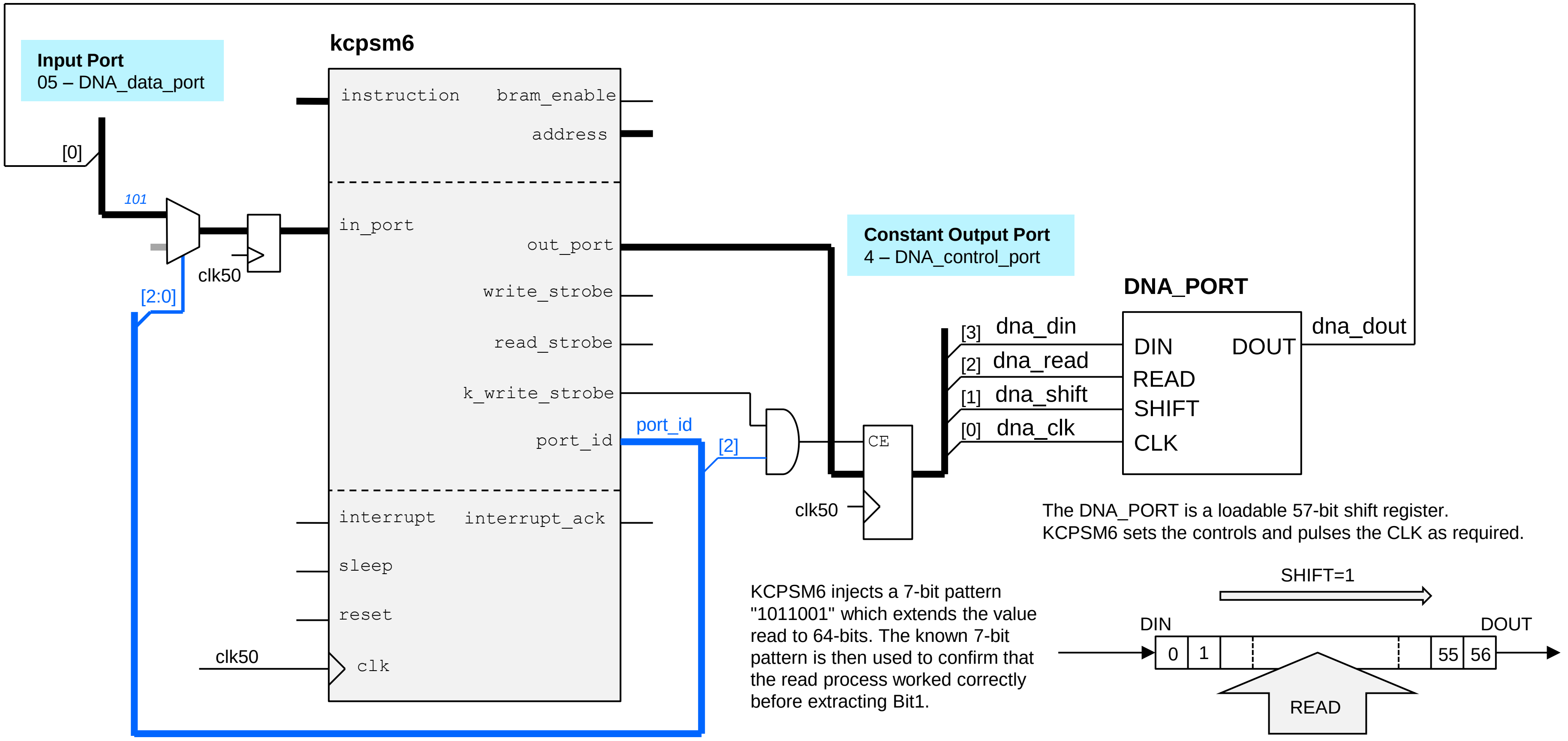
Experiments

How fast is the device on your board (P) ?
How does operating at 0.9v and1.0v effect the maximum performance (V) ?
How does temperature effect performance (T) ?
  (Hint – Increase dynamic power to warm the device up)

Remember that lowering the supply voltage should lower performance and decrease power consumption (static and dynamic). The 'VID' scheme is there to reduce performance level that exceeds the worst case specification of a '-1C' device.

Reading Device DNA

For clarity, this diagram only shows the connections directly associated with interfacing the DNA_PORT to KCPSM6.



The DNA_PORT is a loadable 57-bit shift register.
KCPSM6 sets the controls and pulses the CLK as required.

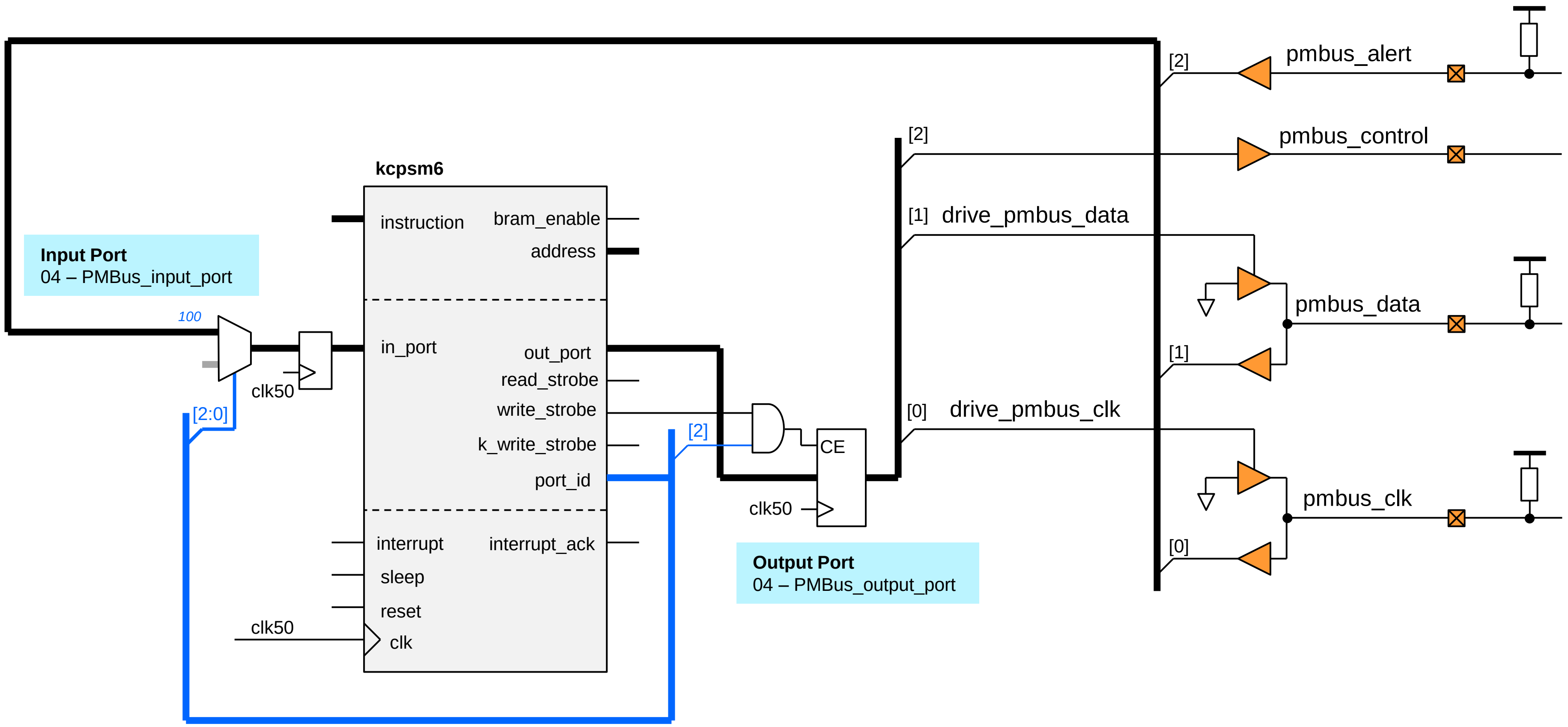
KCPSM6 injects a 7-bit pattern "1011001" which extends the value read to 64-bits. The known 7-bit pattern is then used to confirm that the read process worked correctly before extracting Bit1.

READ=1 is used to load the shift register with the unique 57-bit 'DNA' value. 'VID' is Bit1.

Bit1 = '0' - Device must be operated at V_{CCINT} = 1.0v.
Bit1 = '1' - Device can be operated at V_{CCINT} = 0.9v or 1.0v.

PMBus Control and Monitor

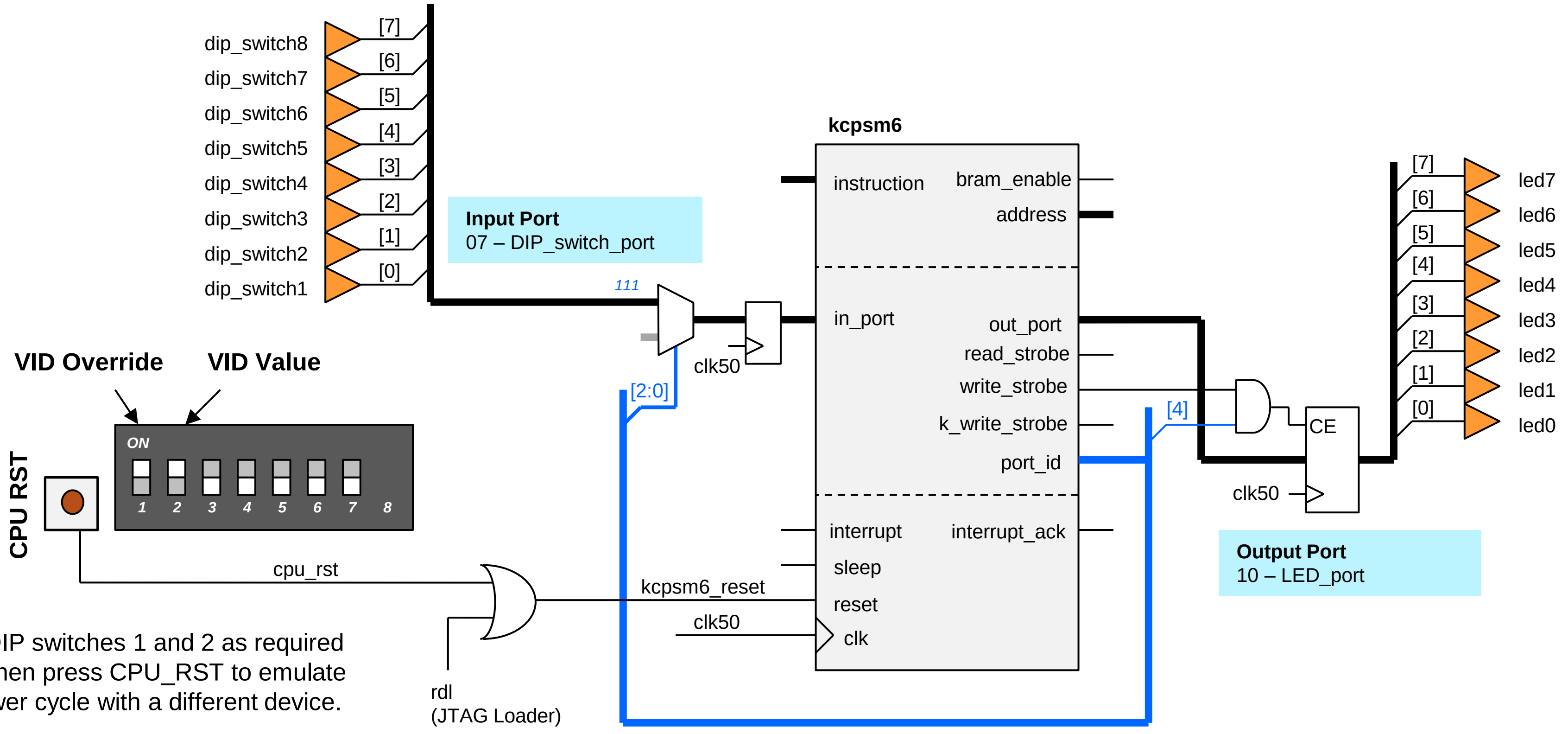
For clarity this diagram only shows the ports assigned to drive and monitor the PMBus in the reference design. The PMBus is used to automatically set the V_{CCINT} supply rail to 0.9v if the VID bit is set. The user is then able to perform experiments and monitor all supply rails.



‘VID’ Override (DIP Switches and LEDs)

As the name suggests, the ‘VID Override’ feature of the design enable you to override the physical value of the VID bit in the Device DNA of the device that you have on your VC707 board. Using this feature you can conduct power and performance experiments with V_{CCINT} supply rail set to 0.9v or 1.0v. Please remember that results are indicative of the effects observed with real ‘-1C’ device with VID of value ‘0’ and ‘1’ and should not be interpreted as absolute figures for real devices.

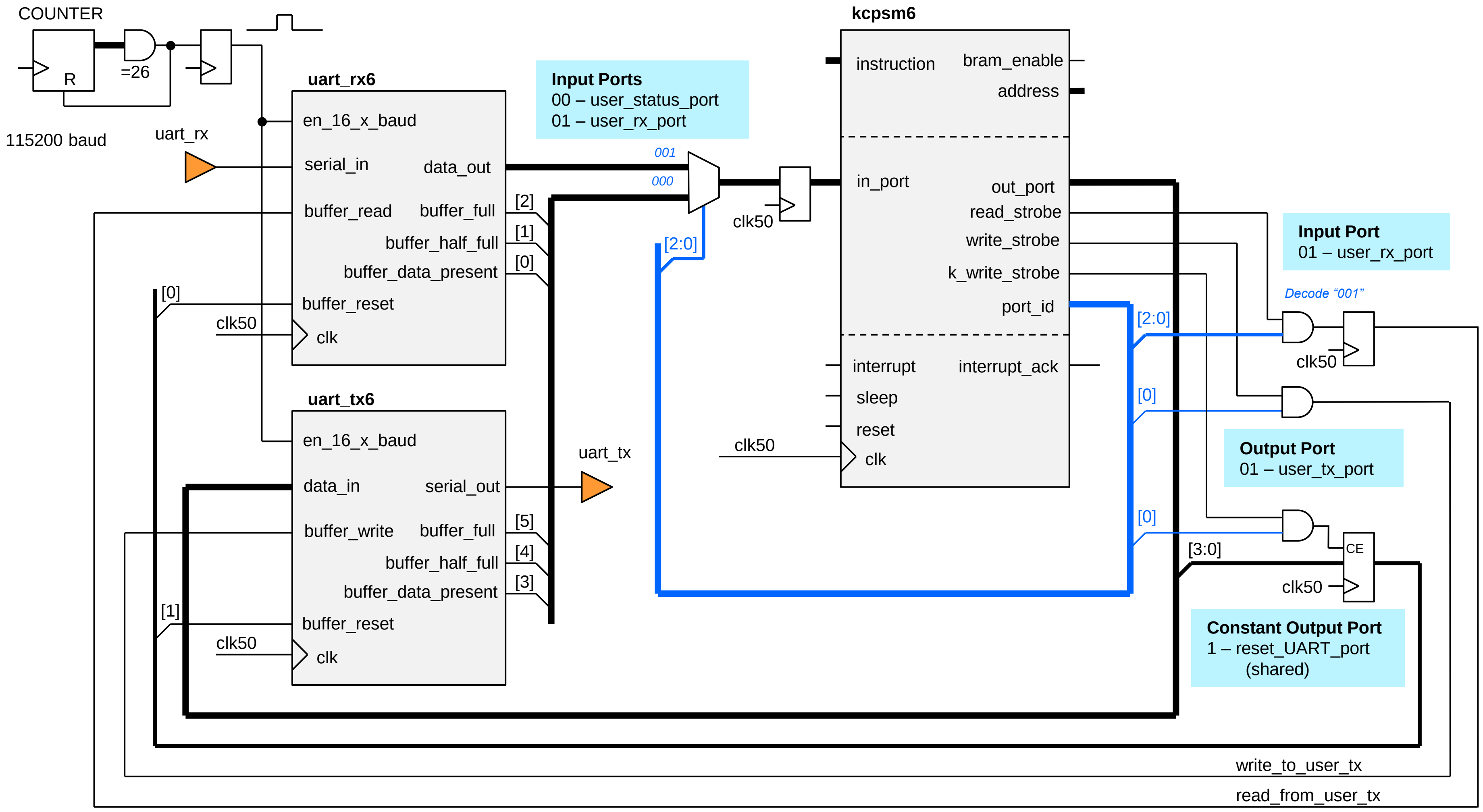
For clarity this diagram only shows the ports assigned to the DIP Switches and LEDs. All switches and LEDs have been connected for future use by the author or by you in your own designs (e.g. The design has JTAG_Loader enabled so you can reprogram PicoBlaze to do anything you like).



Set DIP switches 1 and 2 as required and then press CPU_RST to emulate a power cycle with a different device.

User Terminal UART Macros

For clarity this diagram only shows the ports assigned to connect to the UART macros used to communicate with the user at 115,200 baud. For more details about the UART macros please see 'UART6_User_Guide_8July11.pdf' and a simple reference design called 'uart6_ml605.vhd' supplied with KCPSM6.

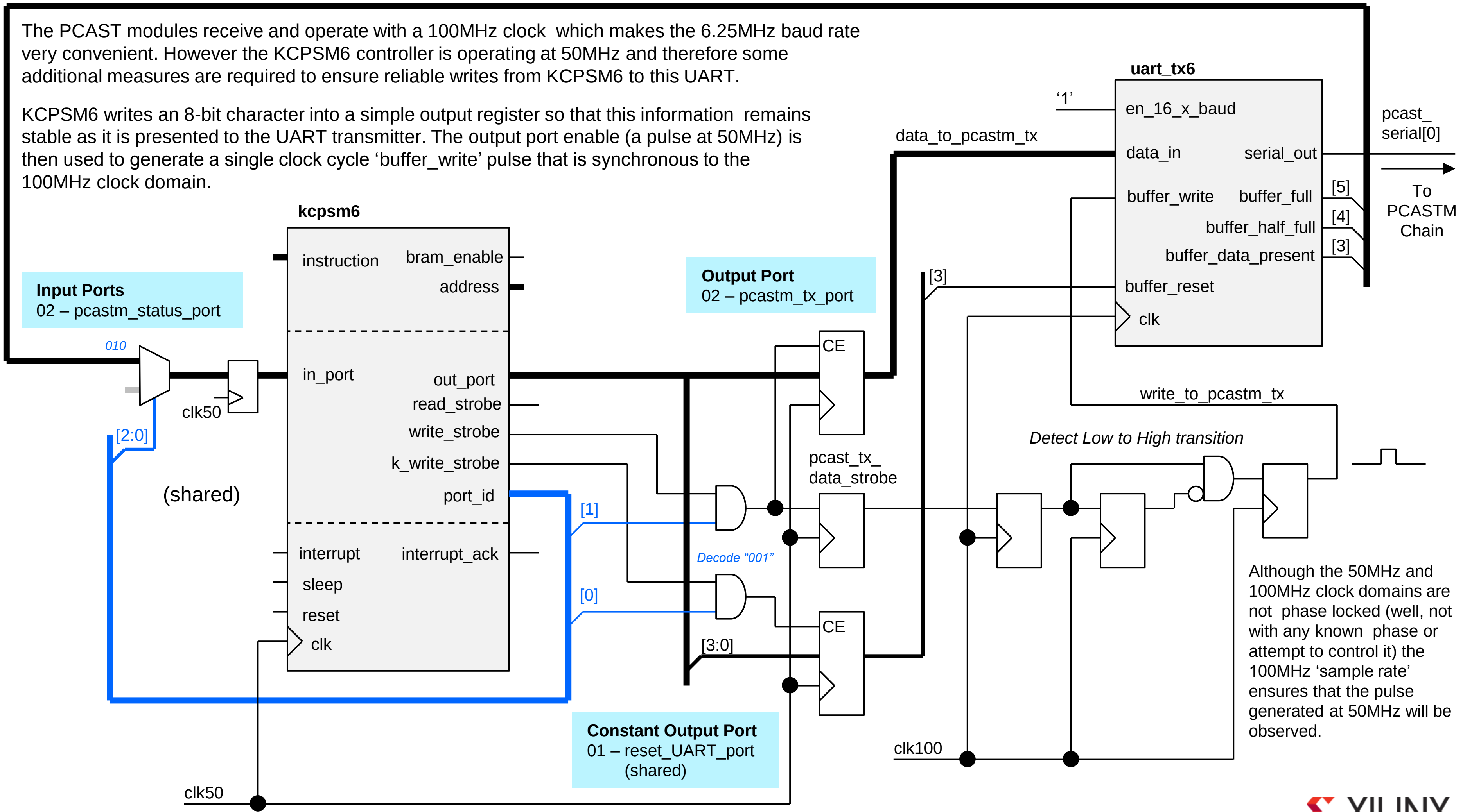


UART Connection to PCASTM Chain - Transmitter

For clarity this diagram only shows the ports assigned to connect to the transmitter UART macro connecting to the PCASTM chain. Communication with the PCASTM chain is at a baud rate of 6.25mbps which is the maximum rate supported by the UART6 macros when using a 100MHz clock (i.e. 100MHz/16).

The PCAST modules receive and operate with a 100MHz clock which makes the 6.25MHz baud rate very convenient. However the KCPSM6 controller is operating at 50MHz and therefore some additional measures are required to ensure reliable writes from KCPSM6 to this UART.

KCPSM6 writes an 8-bit character into a simple output register so that this information remains stable as it is presented to the UART transmitter. The output port enable (a pulse at 50MHz) is then used to generate a single clock cycle 'buffer_write' pulse that is synchronous to the 100MHz clock domain.

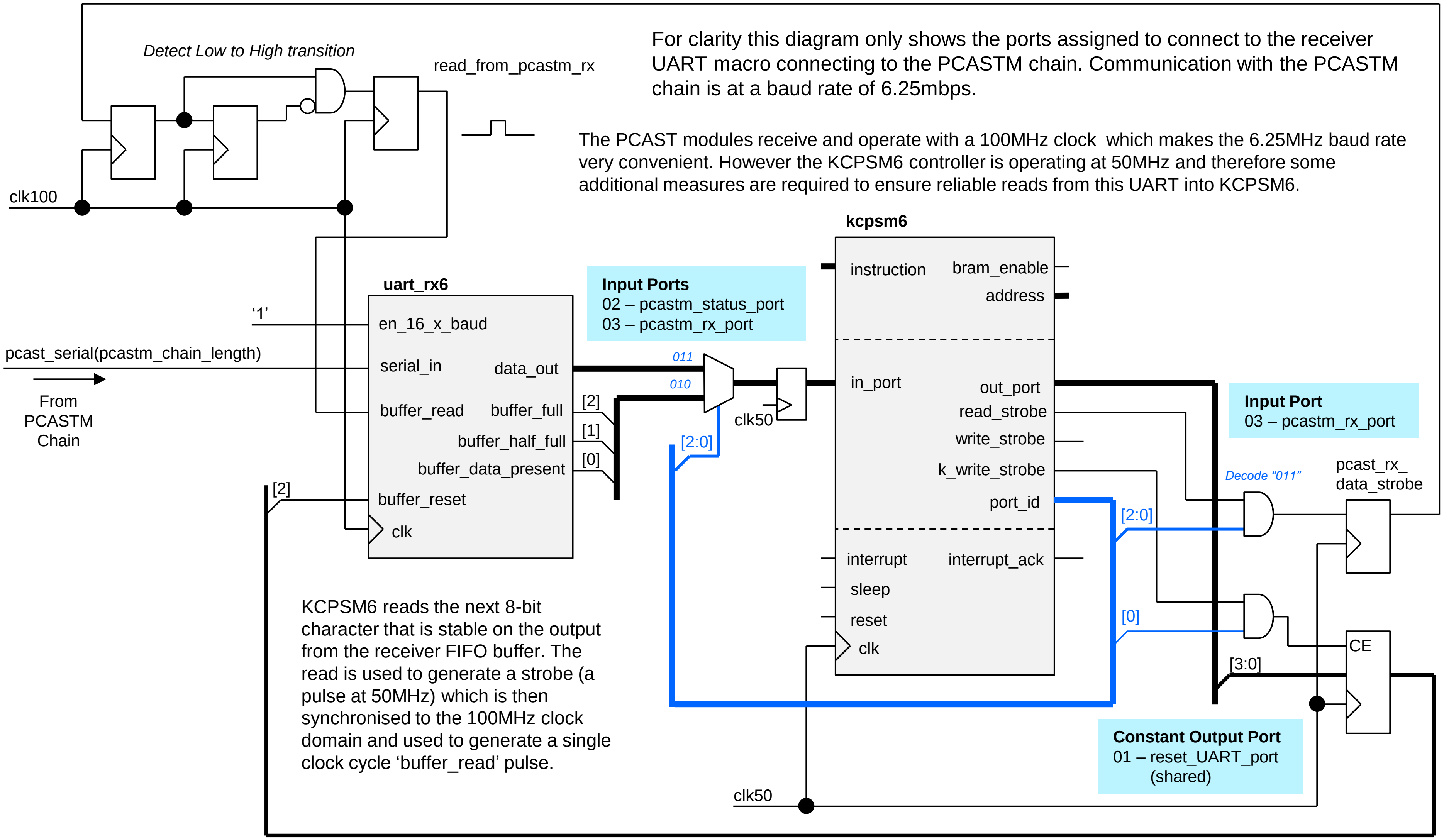


Although the 50MHz and 100MHz clock domains are not phase locked (well, not with any known phase or attempt to control it) the 100MHz 'sample rate' ensures that the pulse generated at 50MHz will be observed.

UART Connection to PCASTM Chain - Receiver

For clarity this diagram only shows the ports assigned to connect to the receiver UART macro connecting to the PCASTM chain. Communication with the PCASTM chain is at a baud rate of 6.25mbps.

The PCAST modules receive and operate with a 100MHz clock which makes the 6.25MHz baud rate very convenient. However the KCPSM6 controller is operating at 50MHz and therefore some additional measures are required to ensure reliable reads from this UART into KCPSM6.



Input Ports
02 – pcastm_status_port
03 – pcastm_rx_port

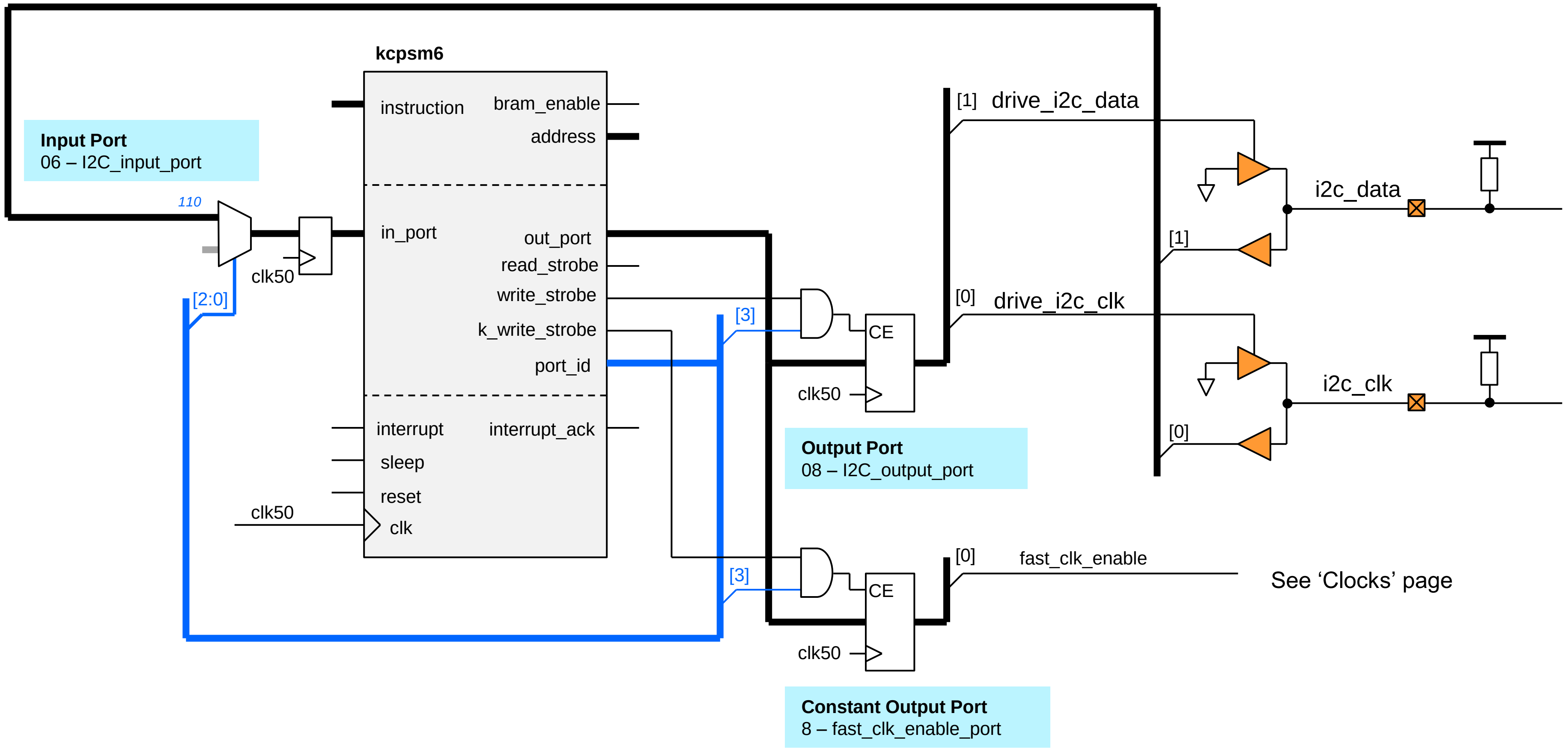
Input Port
03 – pcastm_rx_port

Constant Output Port
01 – reset_UART_port
(shared)

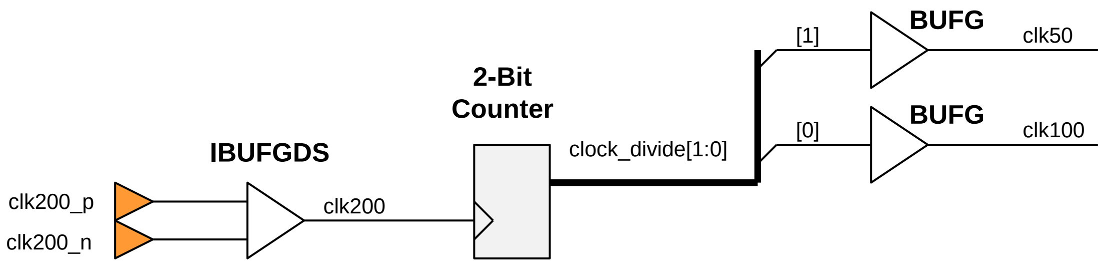
KCPSM6 reads the next 8-bit character that is stable on the output from the receiver FIFO buffer. The read is used to generate a strobe (a pulse at 50MHz) which is then synchronised to the 100MHz clock domain and used to generate a single clock cycle 'buffer_read' pulse.

I2C Bus for Si570 Control

For clarity this diagram only shows the ports assigned to drive and monitor the I2C bus in the reference design. The I2C bus is used to control the 8-channel I2C bus multiplexer on the board to select and then control the Si570 programmable oscillator.

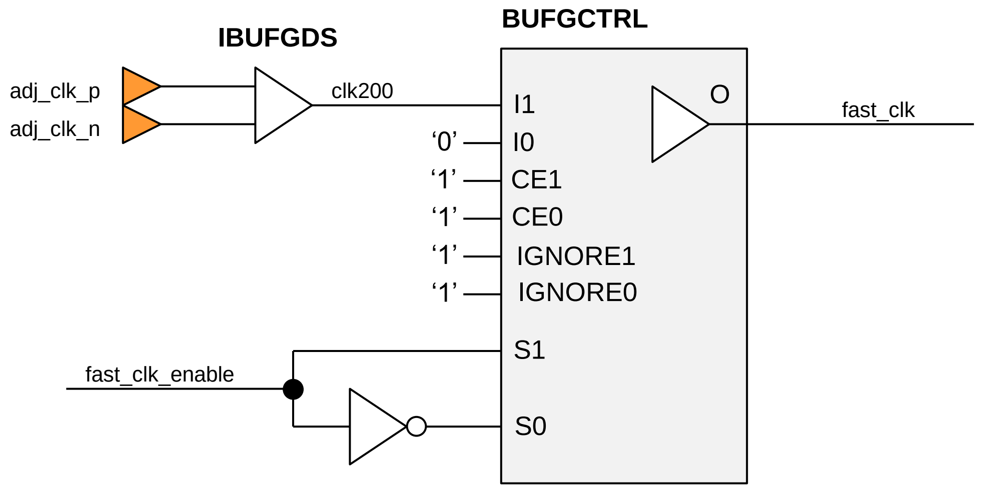


Clocks



50MHz – Used by the main KCPSM6 controller.

100MHz – Used by all the Power Consuming and Speed Test Modules (PCASTM) to run the KCPSM6 and the 16 toggle flops, 16-bit LFSR counter and 16-bit accumulator peripherals.



'Fast Clock'– Applied to the 8-bit LFSR counters and comparators forming the 'speed test' elements in all PCAST modules.

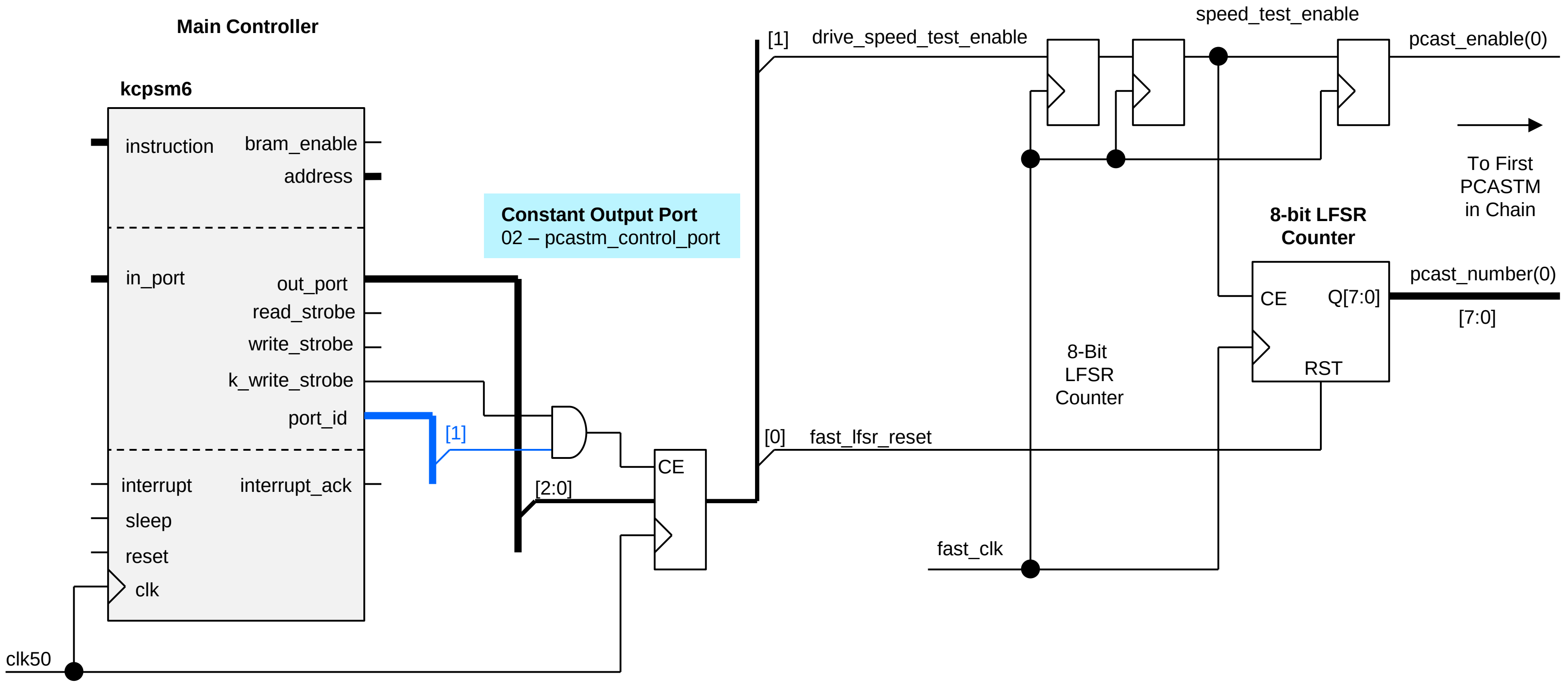
KCPSM6 will disable the distribution of 'Fast Clock' to illustrate a method for limiting dynamic power consumption of a design prior to V_{CCINT} being set to the appropriate level for the VID bit.

Speed Test 'Primer' LFSR Counter and Control

For clarity this diagram only shows the port assigned to the main KCPSM6 controller used to control the speed test circuits.

As shown in the following pages, the 'speed test' in each PCASTM involves the comparison of its 8-bit LFSR counter with that of the 8-bit LFSR counter in the previous PCASTM. In order that the first PCASTM in the chain can also perform a valid speed test a 'primer' LFSR counter is included in the top level design. This counter is controlled by the main KCPSM6 processor.

As each speed test is implemented by the main KCPSM6 processor, the speed test enable signal is activated for ~20ms. It is absolutely vital that the enable signal is synchronised to the 'fast_clk' as it is asserted and de-asserted so synchronising flip-flops are included to ensure this happens.

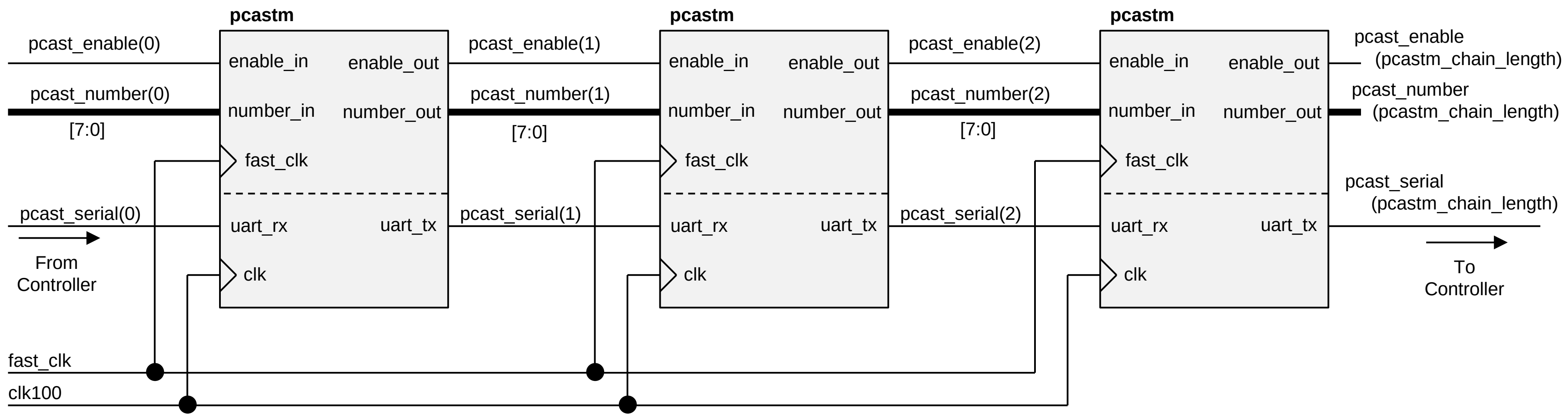


The PCASTM Chain

The Power Consuming and Speed Test Modules (PCASTM) modules are connected in a chain of whatever length is desired and which can fit into the target device. For ease of design manipulation a constant called 'pcast_chain_length' has been defined and is used to define the number of PCASTM in the chain without requiring any additional changes to be needed. This illustration shows a chain formed of only 3 modules but it would be more normal to have hundreds or even thousands in a large device.

Each PCASTM is formed of two sections; a 'Power Consuming' section and a 'Speed Test' section. Of course the speed test will also result in the consumption of additional power but this is not its main purpose.

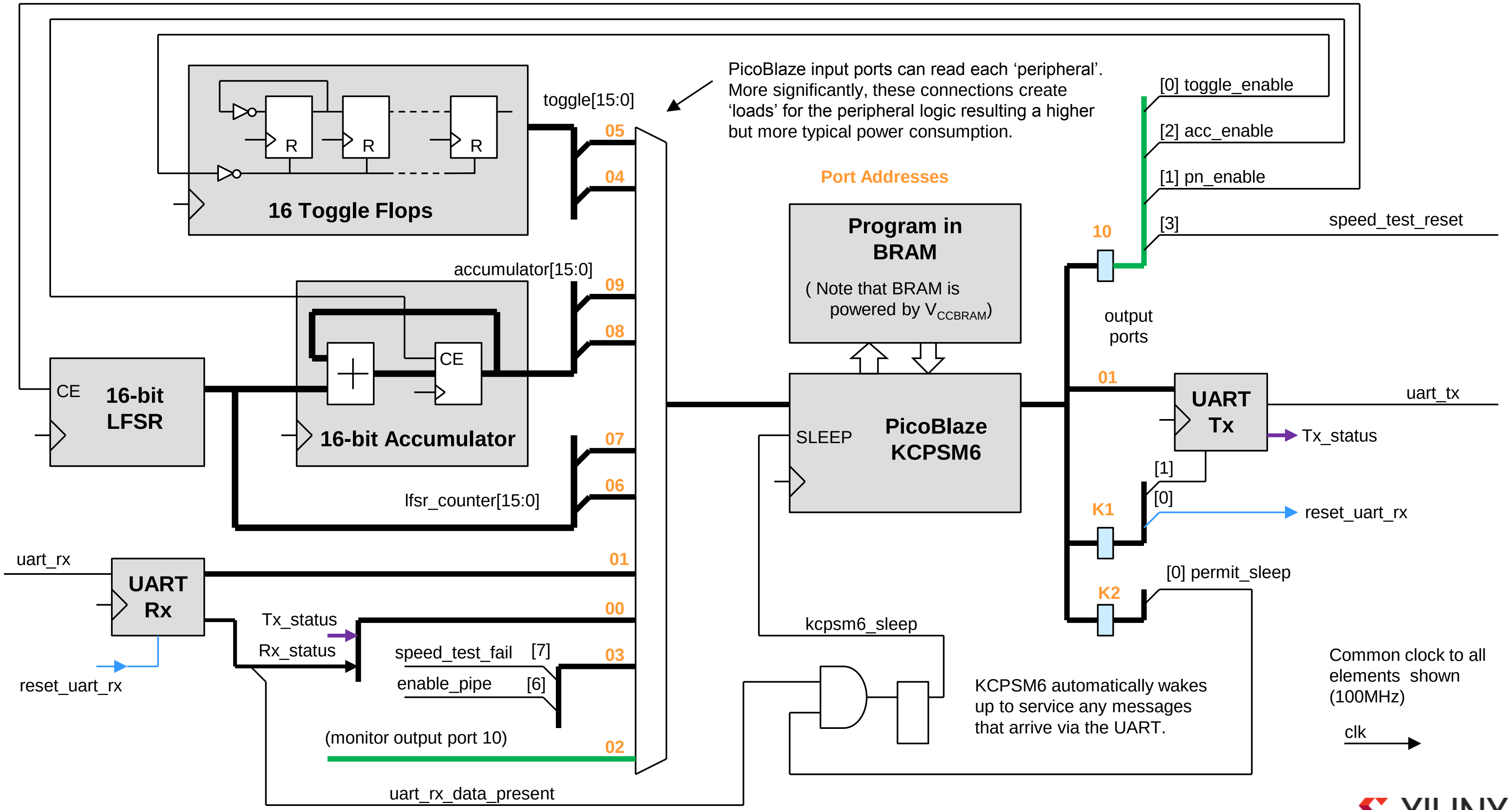
'Speed Test' – The speed test is operated at a higher clock frequency ('fast_clk') and is intended to verify the ability of the device to exceed the specified design performance especially if the V_{CCINT} supply is reduces to 0.9v. In the reference design 'fast_clk' is driven by the Si570 programmable oscillator and used to test the design at 1MHz intervals starting at 300MHz. The speed test requires that logic functions within each module operate correctly at each frequency and also that the 'enable' and 8-bit 'number' signals all have less than single clock cycle propagation times. As soon as one path is too slow for the clock frequency the incorrect operation will be detected and hence reveal the maximum performance of the speed test circuits in the given device (P) at that voltage (V) and temperature (T).



Control and 'Power Consuming' Logic – The bulk of the logic resources in each module are responsible for the control and monitoring of each module in the chain and to increase or decrease the dynamic power consumption. All this logic is operated at 100MHz which is very conservative and reliable whilst having the ability to consume enough power for experimental purposes. The modules are controlled and monitored by passing ASCII commands through the chain using UART serial communications at a baud rate of 6.25mbps. This scheme provides flexibility and scalability whilst only requiring one wire to link each pair of modules, which in turn, allows the links of the speed test to dominate the connectivity between modules.

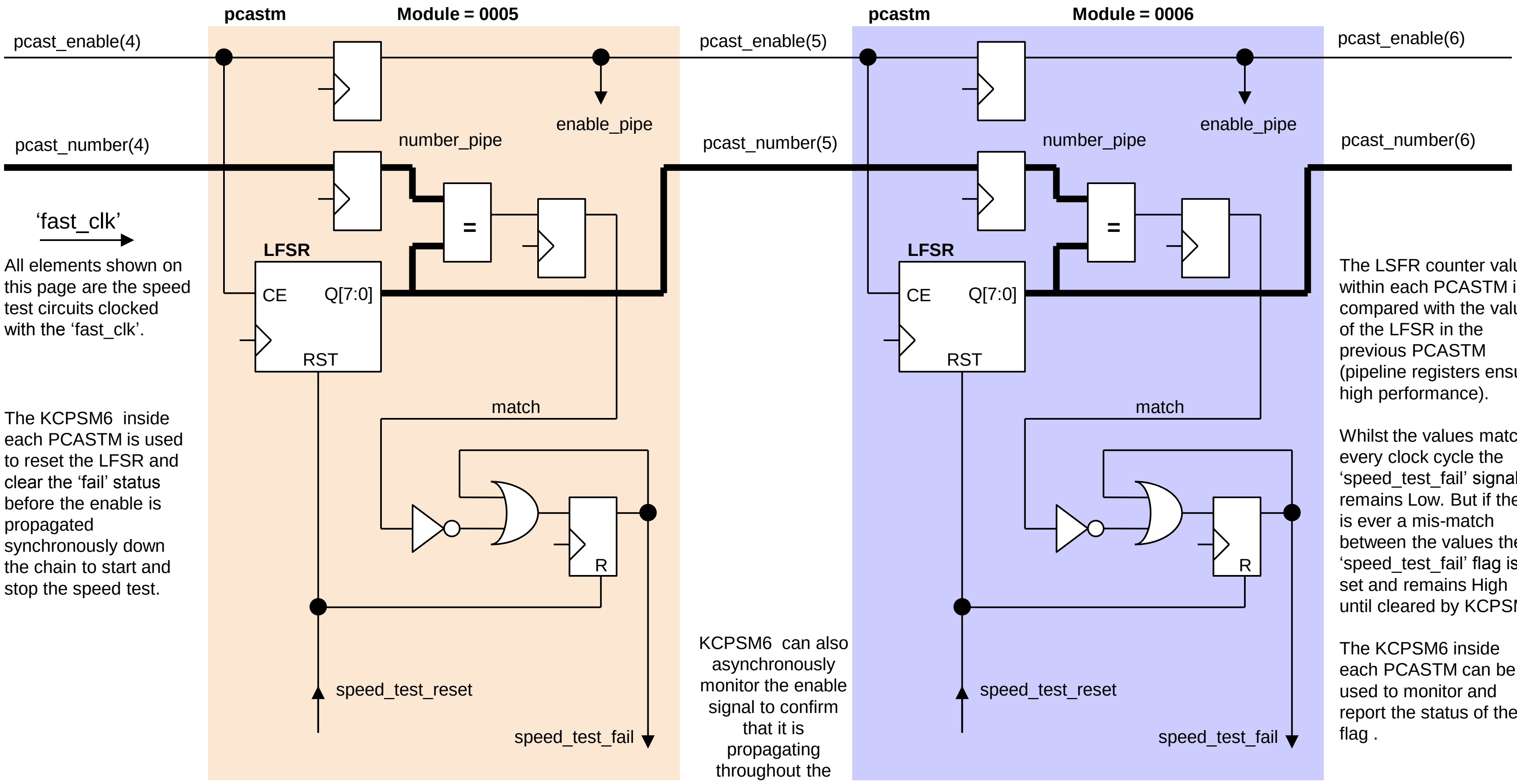
‘Power Consuming Logic’, Control & Monitoring

This diagram represents the communication, control and power consuming (PC) circuits of each PCASTM (note the signals to and from the ‘speed test’). Waking PicoBlaze and enabling the 16 toggle flip-flops, the 16-bit LFSR counter and the 16-bit accumulator in any combination will increase or decrease dynamic power consumption. Please read notes in the source files to appreciate how these simple functions can be used to demystify the most elusive of factors; toggle rate.



The 'Speed Test'

For clarity, this diagram only shows only the 'speed test' circuits contained in the 5th and 6th PCAST modules in the chain. Each link of the chain represents a separate test of logic and interconnect speed. As such, it is possible for some links to fail whilst others are able to continue operating perfectly. If desired, it would be possible to enhance the reference design to further interrogate the chain and identify which link(s) fail the speed test first.



All elements shown on this page are the speed test circuits clocked with the 'fast_clk'.

The KCPSM6 inside each PCASTM is used to reset the LFSR and clear the 'fail' status before the enable is propagated synchronously down the chain to start and stop the speed test.

KCPSM6 can also asynchronously monitor the enable signal to confirm that it is propagating throughout the chain.

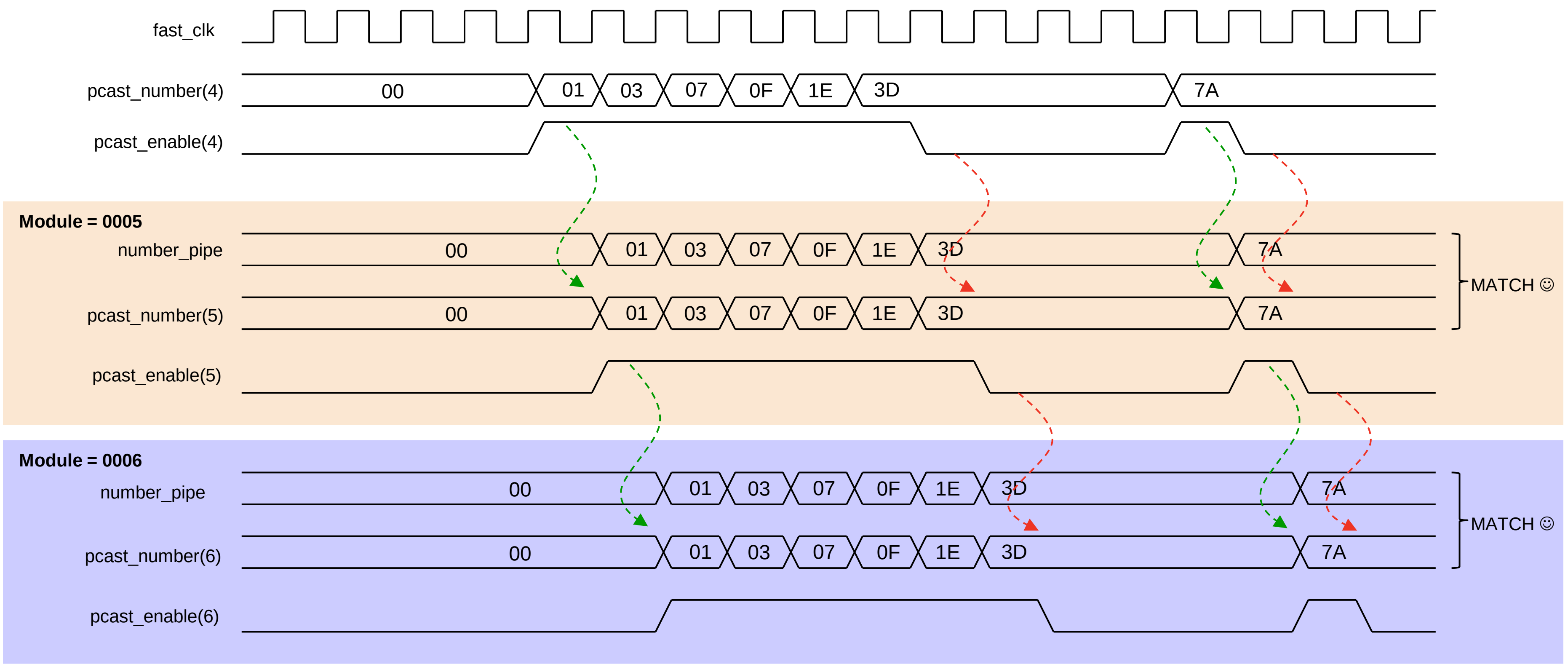
The LSFR counter value within each PCASTM is compared with the value of the LFSR in the previous PCASTM (pipeline registers ensure high performance).

Whilst the values match every clock cycle the 'speed_test_fail' signal remains Low. But if there is ever a mis-match between the values the 'speed_test_fail' flag is set and remains High until cleared by KCPSM6.

The KCPSM6 inside each PCASTM can be used to monitor and report the status of the flag .

‘Speed Test’ Waveforms

These waveforms illustrate how, if desired, the ‘enable’ to the speed test can be any waveform. As the enable passes through the chain the LFSR counters in adjacent modules should start and stop synchronously and always remain synchronised. They will do this providing the fundamental logic and interconnect is good and if the performance is adequate for the clock rate. In the reference design provides, then enable is active for ~20ms at each test frequency.



At all times the value of the local LFSR counter should be the same as the value of the LFSR counter in the previous PCASTM delayed by one clock cycle (‘number_pipe’). This pipeline delay compensates for the pipeline delay in the enable signal.

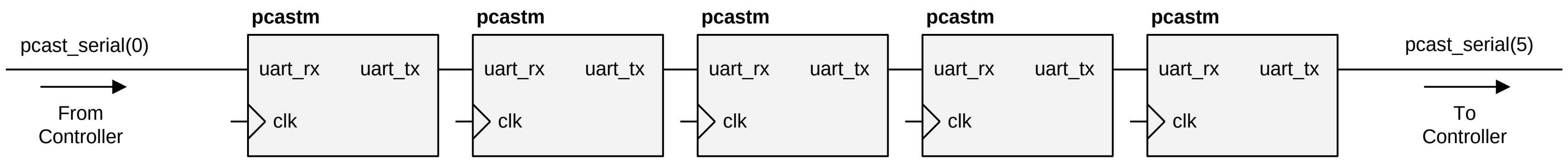
Communication with the PCAST Chain

Although this is going beyond the scope of the 'VID' Reference design, it is anticipated that the arrangement presented in the PCASTM chain could have value in other designs. The fundamental concept of a chain of PicoBlaze linked with a UART communication thread could be applied to the control and monitoring of virtually anything within a device and could be useful in any large design. Think in terms of it being like having a set of ultra-lite ChipScope Analysers in your design. Rather that each module being identical as it is in this design, the hardware and program in each module could be enhanced to perform much more intelligent tasks and tuned to the logic it is required to control and monitor. Note that a program of up to 256 instructions can be implemented in 18 Slices avoiding the need for a BRAM and making each module a very small footprint in your designs.

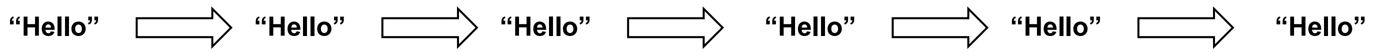
As shown in the previous diagrams, all PCASTM in the chain are linked by serial (UART) communications that only flows in one direction. The main controller (shall we call it the master?) transmits messages to the start of the chain and then waist for responses to emerge from the end of the chain

Echo Mode

The most fundamental state of each PCASTM is the echo mode. This is the mode the KCPSM6 in each PCASTM is in whenever it isn't actively servicing a command. Unsurprisingly the echo mode simply means that KCPSM6 will read any characters that arrive on the UART receiver and write them to the UART transmitter. If KCPSM6 is sleeping then the arrival of a character will automatically wake KCPSM6 up so that it will do this (KCPSM6 makes sure that it completes this simple task before permitting itself to go back to sleep).



Providing the # character is <u>not</u> used then all characters will be echoed through each PCAST in the chain. <u>Hint</u> - This can be used to test the chain.



The serial baud rate is 6.25mbps so the time to transmit one character is 1.6µs. As a character is being transmitted the corresponding receiver is converting it back into a character so there is no additional delay associated with the 'link' itself. However, KCPSM6 in each PCASTM can only see the character once it has been fully received, and only then, can it start to transmit onwards. KCPSM6 itself does this almost immediately but there can be up to a 1 bit delay (160ns) before the start of the serial transmission. All this means that it takes time for a character to ripple all the way through the chain. As an approximation the total time will be in the order of 1.7µs × the number of PCASTM in the chain. In practice, it is normal for the master controller just to wait for the expected character or message to appear at the end of the chain however long it takes.

Communication with the PCAST Chain

PCASTM Commands

Although rather a precise ‘language’, the commands currently implemented by PicoBlaze inside each PCASTM are all formed of readable ASCII characters. This is intended to make any communication humanly readable and writable which is particularly useful during system debugging.

> Note – In order to keep the current PCASTM PicoBlaze program small (enough to fit in 18 Slices if required) there is no error checking or trapping. This doesn’t normally present any issues when commands sent to the chain are generated by the program in the main controller (at least not after the code had been debugged!) but can definitely lead to issues when manually entering commands (i.e. human error!). So just try to be careful if you attempt any manual communication with the chain.

In the current implementation only **upper case characters** should be used in the definition of any commands or information.

#

A command always begins with the # character. All PCASTM will remain in ‘echo mode’ until the # character is encountered.

Although a # character will also be echoed, PicoBlaze will transition into a ‘command mode’ in which the next character will define the command, and when relevant, further characters will provide information to be used with that command.

When in the command mode PicoBlaze will remain permanently awake. Once a command has been processed, PicoBlaze will return to the ‘echo mode’ and may go to sleep if it is set to be in the sleep mode.

#T - Transparent Command

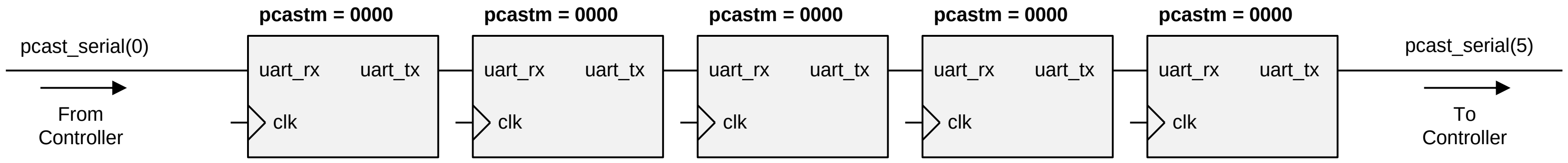
This is the most simple (and seemingly pointless!) command. This command forces the PCASTM back into ‘echo mode’ immediately. Any subsequent characters are then echoed until the next # character is encountered.

Communication with the PCAST Chain

#Eaaaa - Enumerate Command

In this design all PCASTM are identical and any number of them can be connected in a chain. Since they are all identical they will all react in the same way to begin with and this would limit your ability to control them and monitor them. For this reason, each module has a 16-bit address intended to service a chain consisting of up to 65,535 modules. However, all modules have the same initial address of zero (0000 hex) which is of no help whatsoever!

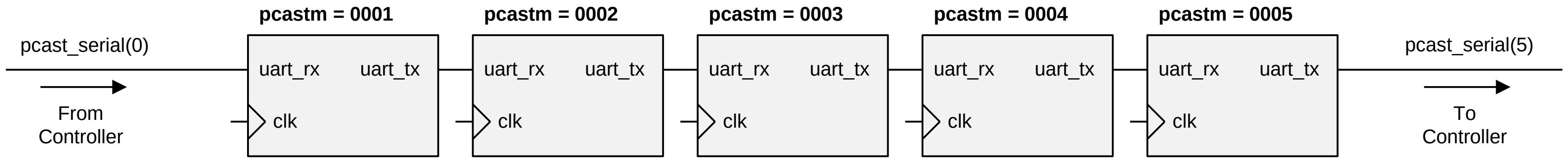
Before enumerate command....



The enumerate command is the way to assign a unique address to each module. In most applications the command sent to the chain specifies an address of zero (specified using the 4-digit hex representation '0000'). As this is received by the first module the address value is incremented to '0001' and becomes the address of that module. It then transmits its own address to the next module. Hence the address increments as it ripples through the chain.

#E0000 ⇨ **#E0001** ⇨ **#E0002** ⇨ **#E0003** ⇨ **#E0004** ⇨ **#E0005**

After enumerate command....



Hint – The message observed at the end of the chain reveals the number of modules in the chain. This is how the master can report the number of modules without being specifically programmed. It is also a good way to confirm the integrity of the chain and that it really does contain the number of modules you think it should.

Note – PicoBlaze needs to receive all 4 characters forming 'aaaa' in order that it can increment the address and adopt that value. This also means that it is only able to transmit its own 'aaaa' value to the next module in the chain once the new address is known. This means that whilst the whole command propagates through the chain and is observed at the end, the 'echo' of 'aaaa' does not occur on a character by character basis. This can be somewhat disconcerting when manually entering commands (i.e. No echo when entering each of the 4 hex digits and then all appear!).

Communication with the PCAST Chain

#Clllluuuubbbb - Control Command

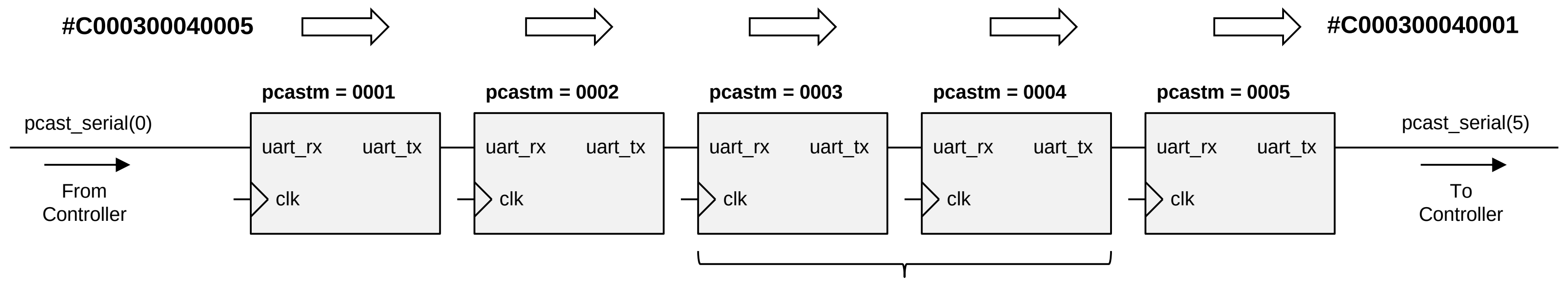
This command enables up to 16 control bits to be set as required in one or more PCASTM in the chain. The command string passes unaltered through each module to re-emerge at the end. As each module receives and echoes the control command it decides if the command should be applied to it and updates the control value with the specified value if it should.

Note – Each group of 4 characters forming a 16-bit hex value 'llll', 'uuuu' or 'bbbb' will only be echoed by a module after all 4 characters have been received. This can be somewhat disconcerting when manually entering commands!

The #C command is followed by three 16-bit values (each specified using a 4-digit hex representation). The first two values 'llll' and 'uuuu' define the range of module addresses that are to respond to the command.

Hint – Specifying an address range of '0000' to 'FFFF' will mean that all modules will respond to the control word regardless of the number of modules in the chain.

Hint – To control only one module, specify the same address for both 'llll' and 'uuuu'.



In this example only the modules in the address range '0003' to '0004' will apply the command.

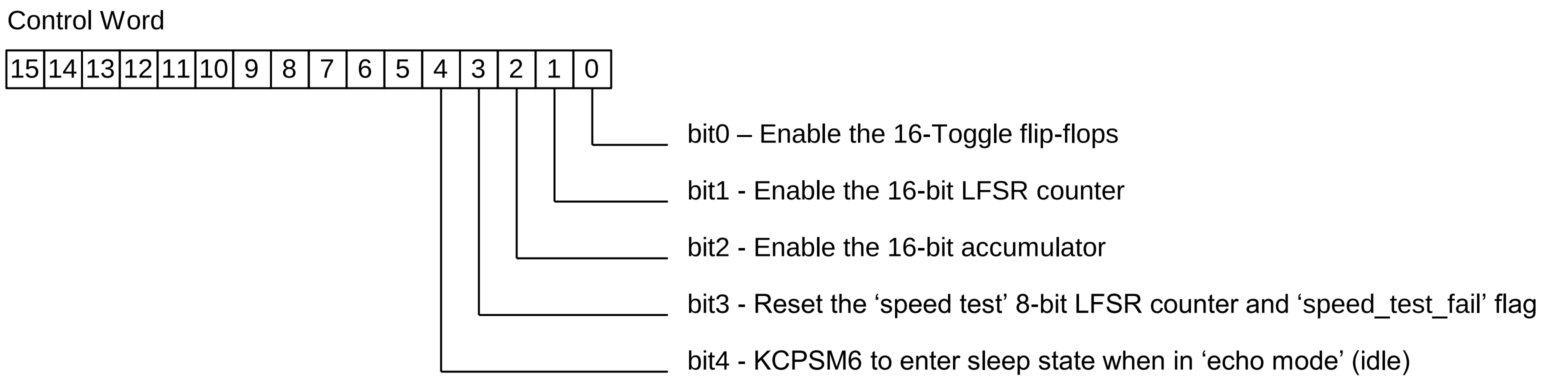
The final 16-bit value 'bbbb' defines the state that the 16 control bits in the PCAST should become. These are used to control the features and operation of the PCASTM and are described on the following page.

In this example the control bits in modules '0003' and '0004' have been set to 0005 hex = 0000 0000 0000 0101.
The control bits in the remaining modules will remain unchanged.

Communication with the PCAST Chain

Control Bits

Whilst it is possible to define 16 control bits only 5 are actually used in the current design (i.e. Clearly room for expansion ☺).


Control Word
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
bit0 – Enable the 16-Toggle flip-flops
bit1 - Enable the 16-bit LFSR counter
bit2 - Enable the 16-bit accumulator
bit3 - Reset the 'speed test' 8-bit LFSR counter and 'speed_test_fail' flag
bit4 - KCPSM6 to enter sleep state when in 'echo mode' (idle)

Default post configuration value is 0000 0000 0001 1000 = 0018 hex and is the lowest power state of the PCASTM. All power consuming peripherals are disabled, the speed test LFSR counter is held in reset (regardless of the state of the speed test enable signal) and KCPSM6 will be sleeping except to service any serial communications that occur.

Note – That in the current implementation all control bits are defined by the control command and therefore all bits must be specified with the desired value. It may be desirable in the future to implement a different command or include a 'mask' word in the existing command that would enable only one or a few bits to be adjusted whilst others remain the same.

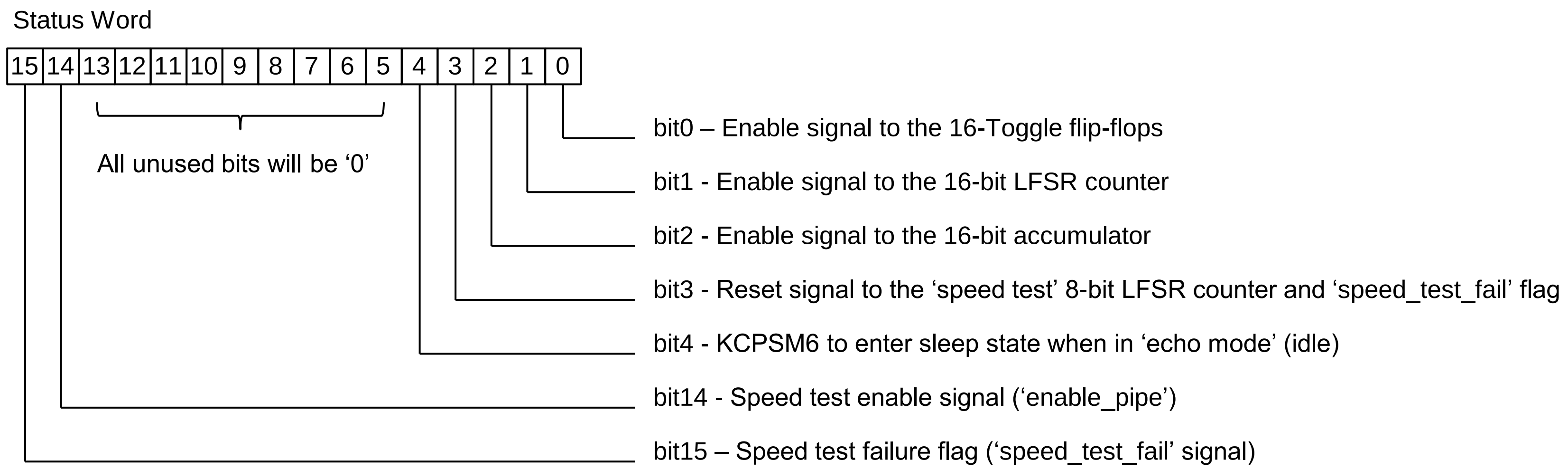
Communication with the PCAST Chain

Status Bits

There are two commands to read the status which are described on the following pages. Before looking at the status commands, below is the format of the status word in each PCASTM.

Bits 0 to 4 mirror the bits in the control word and would therefore be expected to be the same. However, it should be noted that in the case of bits 0 to 3, KCPSM6 is truly observing the state of the physical signals via an input port and therefore proves that the control signals are actually being driven by the corresponding KCSPM6 output port. In contrast, the sleep state indicated by bit4 has to be a report of the value held within KCPSM6 rather than a physical signal because KCSPM6 has to be awake when servicing a status command. As such it can only ever be an indication of the fact that it will go to sleep when it is otherwise in the idle mode.

Bits 14 and 15 allow the status of the speed test circuit to be observed. The status is a snapshot sample of the 'enable_pipe' and 'speed_test_fail' signals. The generally static nature of these signals means that is perfectly acceptable for KCPSM6 to observe them even though it is operating with a different clock. Note that the speed test result is for the particular PCASTM and completely independent of the other PCASTM in the chain. Hence it is possible to isolate any point(s) of failure by individually interrogating the status of each PCAST module.


Status Word
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
All unused bits will be '0'
bit0 – Enable signal to the 16-Toggle flip-flops
bit1 - Enable signal to the 16-bit LFSR counter
bit2 - Enable signal to the 16-bit accumulator
bit3 - Reset signal to the 'speed test' 8-bit LFSR counter and 'speed_test_fail' flag
bit4 - KCPSM6 to enter sleep state when in 'echo mode' (idle)
bit14 - Speed test enable signal ('enable_pipe')
bit15 – Speed test failure flag ('speed_test_fail' signal)

Hint – Bit14 can be used to confirm that the speed test enable signal is propagating all the way through the chain.

Communication with the PCAST Chain

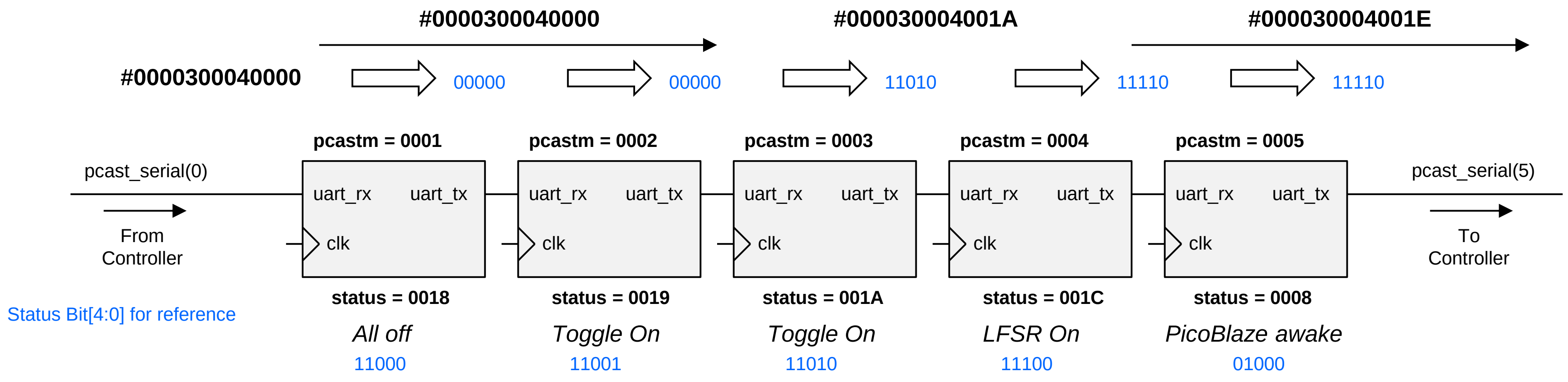
#Olllluuuussss - 'OR' Status Command

This 'OR' status command allows the 16 status bits of one or more PCASTM in the chain to be observed. In other words, it is possible to use this command to observe the status word of just one particular module or to observe the combined status of many or all modules in one go. In typical application the status of all modules would be checked first to determine if everything is operating as expected or if a failure has occurred anywhere. Then if desired, the precise location of the failure could be identified by reading the status of fewer or individual modules.

The #O command is followed by three 16-bit values (each specified using a 4-digit hex representation). The first two values 'llll' and 'uuuu' define the range of module addresses that are to respond to the command. The last word is the 16-bit status. In the majority of applications the command sent to the chain will specify and initial status of '0000'. The command ripples through the chain and is otherwise ignored by any modules that do not fall within the specified address range. When a module is within the specified range then it will take a snapshot of its current status and then superimpose its own status word onto the status word contained in the command using a bit-wise OR operation. The resulting combined status is then transmitted to the next module in the chain.

Note – Each group of 4 characters forming a 16-bit hex value 'llll', 'uuuu' or 'ssss' will only be echoed by a module after all 4 characters have been received. This can be somewhat disconcerting when manually entering commands!

In this example the combined 'OR' status of modules 0003 and 0004 is determined. Modules 0001 and 0002 are not in the specified range so they simply echo the command as received. Module 0003 superimposes its status word onto the command as it passes it on (0000 OR 001A = 001A). Likewise, module 0004 superimposes its status word onto the command as it passes it on (001A OR 001C = 001E). Finally, module 0005 simply echoes the command containing the superimposed status as it is outside of the address range.



Hint – 'OR' status is the easiest way to determine if a particular status it is set ('1') in one or more modules in the chain.

Communication with the PCAST Chain

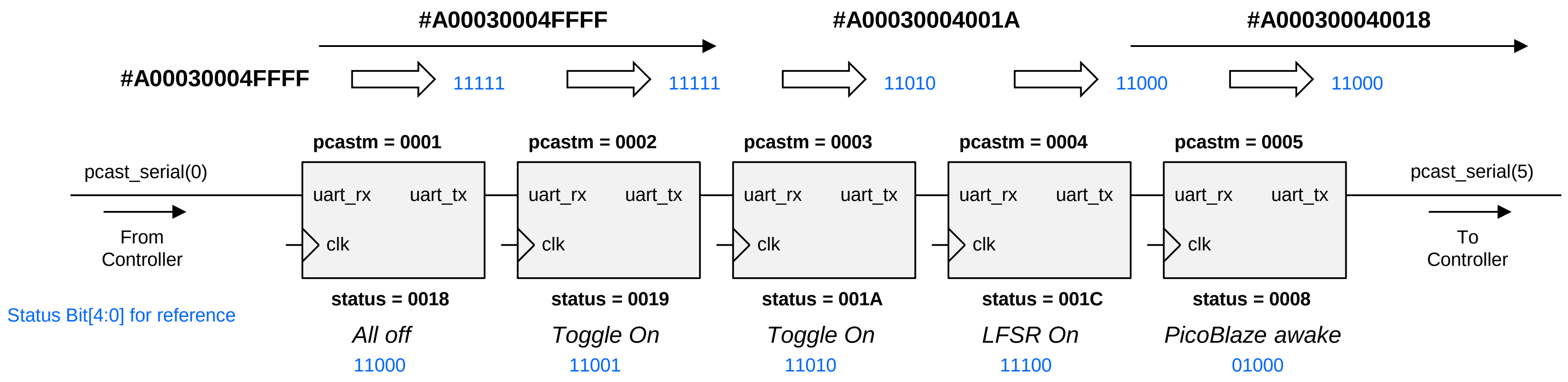
#Allllluuuusssss - 'AND' Status Command

As with the 'OR' status command, the 'AND' status command allows the 16 status bits of one or more PCASTM in the chain to be observed.

The #A command is followed by three 16-bit values (each specified using a 4-digit hex representation). The first two values 'llll' and 'uuuu' define the range of module addresses that are to respond to the command. The last word is the 16-bit status. In the majority of applications the command sent to the chain will specify and initial status of 'FFFF'. The command ripples through the chain and is otherwise ignored by any modules that do not fall within the specified address range. When a module is within the specified range then it will take a snapshot of its current status and then superimpose its own status word onto the status word contained in the command using a bit-wise AND operation. The resulting combined status is then transmitted to the next module in the chain.

Note – Each group of 4 characters forming a 16-bit hex value 'llll', 'uuuu' or 'ssss' will only be echoed by a module after all 4 characters have been received. This can be somewhat disconcerting when manually entering commands!

In this example the combined 'AND' status of modules 0003 and 0004 is determined. Modules 0001 and 0002 are not in the specified range so they simply echo the command as received. Module 0003 superimposes its status word onto the command as it passes it on (FFFF AND 001A = 001A). Likewise, module 0004 superimposes its status word onto the command as it passes it on (001A AND 001C = 0018). Finally, module 0005 simply echoes the command containing the superimposed status as it is outside of the address range.



Hint – 'AND' status is the easiest way to determine if a particular status it is reset ('0') in one or more modules in the chain.

Communication with the PCAST Chain

#I - Initialise Command

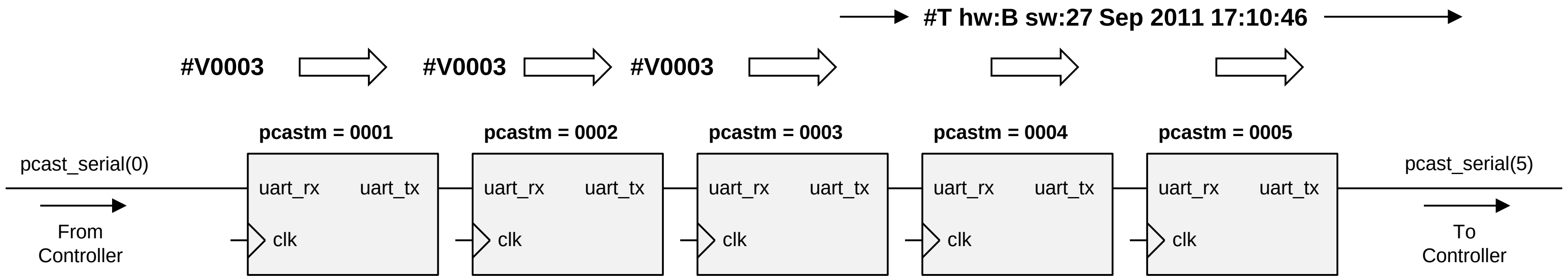
This command will ripple all the way through the chain and cause all modules to revert to the initial default state. This is the lowest power state in which all power consuming peripherals are disabled, the speed test LFSR counter is held in reset and KCPSM6 will be sleeping. All modules will also revert to their default address of '0000' so the enumerate command would normally be used again afterwards.

> Hint – This could be very useful in an emergency such as a device overheating situation. It is very easy to type manually ☺

> Note – When PicoBlaze in the PCASTM receives the #I command it will almost immediately disable the power consuming peripherals and hold the speed test circuit in reset. However there will be a delay of ~100ms before PicoBlaze enters its sleep state. This ensures that the #I command is definitely passed on to the next module before the buffers in the UART macros are also cleared.

#Vaaaa - Version Command

This command will cause the particular PCASTM being addressed to return a version test string indicating the hardware version of the PCASTM design and the date and time that the KCPSM6 program was assembled. Note that this command will return many more characters than the original command. It is also worth noting that the response is a string of 31 characters so the master must ensure that it reads the message before the 16-character buffer in the UART receiver becomes full (not normally an issue providing the master does not then become occupied doing or waiting for something else).

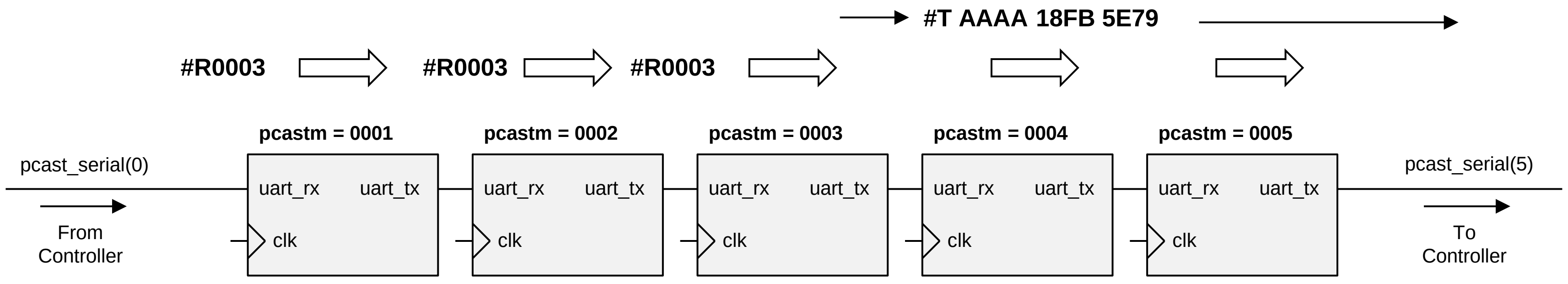


As the #V command ripples through the chain only the module with the specified address will respond. So prior to the target module the command is simply echoed and otherwise ignored. The responding module transmits a transparent command (you see, it wasn't pointless after all ☺) so that all following modules will simply echo the text string through to the end of the chain where it can be captured by the master.

Communication with the PCAST Chain

#Raaaa - Read Information Command

This command will cause the particular PCASTM being addressed to return a string containing a snapshot of the 16-bit values of the ‘power consuming’ peripherals in that PCASTM. The first value is that of the 16 toggle flip-flops, the second is the 16-bit LFSR counter value and the last is the 16-bit value of the accumulator. This illustrates how a module can be used to monitor virtually any circuit.



As the #A command ripples through the chain only the module with the specified address will respond. So prior to the target module the command is simply echoed and otherwise ignored. The responding module transmits a transparent command (again you see it wasn’t pointless ☺) so that all following modules will simply echo the string through to the end of the chain where it can be captured by the master.

Hint – The only real purpose of this command is to prove that the PCASTM peripherals are operating when enabled and static when disabled. There is no accurate control over the enables to these peripherals so it is not possible to enable the peripherals for a precise number of clock cycles and hence stop them at a known value. However, it is possible to use multiple read commands to see if the values are static or changing.

Note – Each 16-bit value is read by KCPSM6 requires the read of two input ports. This means that the lower byte is read two clock cycle before the upper byte so the ‘snapshot’ will not actually be a true ’16-bit snapshot’ value if the peripheral is enabled. However, this is somewhat irrelevant in terms of the PCASTM given the basic purpose is only to observe any activity at all but worthy of consideration if migrated to a real design.

Note – The 16-bit toggle flip-flops are held in reset when not enabled and therefore the static value should always be zero (‘0000’). When toggling, the bit pattern will rapidly settle on alternating ‘5555’ and ‘AAAA’ values. However, because every operation in a KCPSM6 takes 2 clock cycles it is an interesting quirk of fate that only the ‘AAAA’ value is ever captured (although it is just possible it could get our of phase due to KCPSM6 going in an out of sleep mode). So generally speaking interpret ‘0000’ as toggle not enabled and ‘AAAA’ as enabled. Power consumption should also increase and decrease accordingly anyway.