

# Application Note **125**

## Adding processors to the PB926EJ-S using Core Tiles

Document number: ARM DAI 0125B

Issued: January 2006

Copyright ARM Limited 2006

# ARM

## Application Note 125

### Adding additional processors to the PB926EJ-S using Core Tiles

Copyright © 2006 ARM Limited. All rights reserved.

#### Release information

The following changes have been made to this Application Note.

#### Change history

Date	Issue	Change
October 16, 2004	A.01	First release
August 22, 2005	B	Getting started section added

#### Proprietary notice

ARM, the ARM Powered logo, Thumb and StrongARM are registered trademarks of ARM Limited. The ARM logo, AMBA, Angel, ARMulator, EmbeddedICE, ModelGen, Multi-ICE, ARM7TDMI, ARM9TDMI, TDMI and STRONG are trademarks of ARM Limited. All other products, or services, mentioned herein may be trademarks of their respective owners.

#### Confidentiality status

This document is Open Access. This document has no restriction on distribution.

#### Feedback on this Application Note

If you have any comments on this Application Note, please send email to [errata@arm.com](mailto:errata@arm.com) giving:

- the document title
- the document number
- the page number(s) to which your comments refer
- an explanation of your comments.

General suggestions for additions and improvements are also welcome.

#### ARM web address

<http://www.arm.com>

# Table of Contents

<b>1</b>	<b>Introduction .....</b>	<b>2</b>
1.1	Purpose of this application note.....	2
1.2	CT and LT-XC2V6000 overview .....	2
<b>2</b>	<b>Getting Started .....</b>	<b>3</b>
<b>3</b>	<b>Architecture .....</b>	<b>4</b>
3.1	System Architecture .....	4
3.2	Clock architecture .....	5
3.3	Reset Architecture.....	7
<b>4</b>	<b>Software Interface .....</b>	<b>8</b>
4.1	Memory Map .....	8
4.2	Tile positions and extension addresses .....	8
4.3	GTC Regs .....	9
4.4	ZBT RAM .....	17
4.5	DPRAM .....	17
4.6	VPB Regs.....	18
4.7	MailBox .....	18
<b>5</b>	<b>Boot Operation .....</b>	<b>22</b>
<b>6</b>	<b>RTL .....</b>	<b>23</b>
6.1	Directory Structure .....	23
6.2	Logical.....	23
6.3	Targets.....	23
<b>7</b>	<b>Example Software .....</b>	<b>25</b>
7.1	Boot routine.....	25
7.2	Voltage Control .....	25
7.3	MailBox Example .....	26
<b>8</b>	<b>Clock frequency settings .....</b>	<b>28</b>
8.1	Default frequencies .....	28
8.2	Changing the startup frequencies .....	28
8.3	System bus frequency testing.....	29
8.4	Full frequency testing .....	29
<b>9</b>	<b>Variations .....</b>	<b>31</b>
9.1	Overview .....	31
9.2	CT7TDMI.....	31

# 1 Introduction

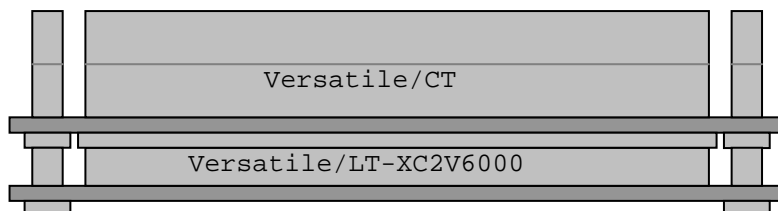
## 1.1 Purpose of this application note

This application note describes an example design implemented in an LT-XC2V6000 Logic Tile to allow use of the CT as an extension to the PB926EJ-S. The design is for a Logic Tile which accompanies each Core Tile.

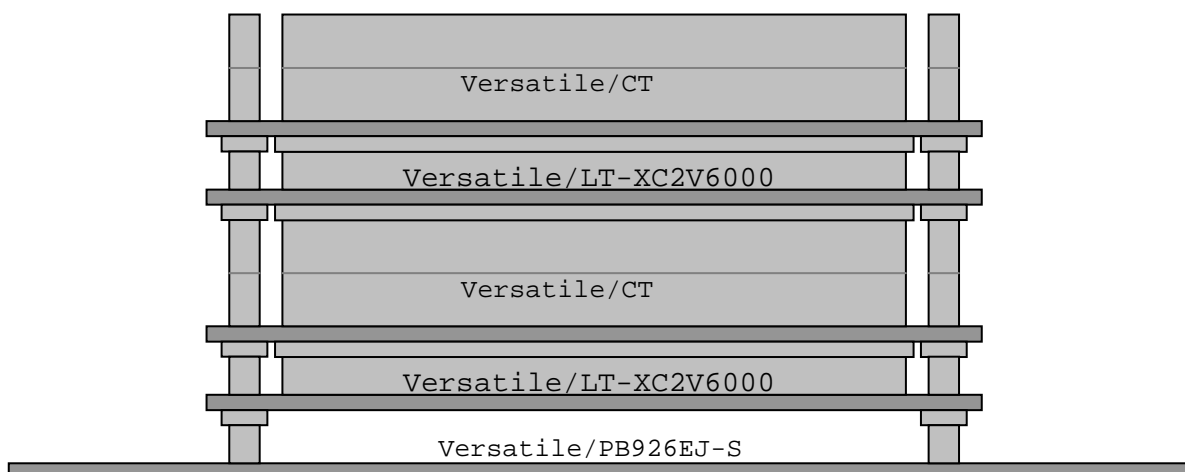
This design is intended as a starting point for hardware developers to build a multi-master, multi-layer AHB system based around the PB926EJ-S. It is also designed to work as a system for multi-core software development.

## 1.2 CT and LT-XC2V6000 overview

The CT provides a platform for an ARM Test Chip to be connected to a Logic Tile system. The LT-XC2V6000 provides an FPGA in which to synthesise all the logic required around a core to make a useful system. The example RTL is designed so that up to four pairs of Core Tile and Logic Tile can be stacked together on top of a PB926EJ-S without any modification to the RTL. The design is limited to four pairs as the address map needs to be correctly divided, and the position determined by each master. Four masters is determined to be a reasonable number for this example. Should more than four masters be required a modified memory map, and position encoding system will be required. The system is also designed to allow for the addition of additional tiles with other customized logic.



**Figure 1-1 Versatile/CT Platform Baseboard subsystem**



**Figure 1-2 Example system with a Versatile/PB926EJ-S and two core tiles**

## 2 Getting Started

Before you can use this application note, ensure that the PB926EJ-S baseboard is programmed and functioning correctly. Refer to the PB926EJ-S user guide for further information. Once you have done this please follow these steps to program the FPGA image in the Logic Tile with the image provided with this application note.

1. Plug the Logic Tile onto the PB926EJ-S. Plug the Core Tile onto the Logic Tile.
2. Fit the CONFIG jumper link on the PB926EJ-S (J32).
3. Connect RVI or Multi-ICE to the to the JTAG ICE connector on the PB926EJ-S (J31), or a USB cable to the PB926EJ-S USB Debug Port (J30).
4. Check the external supply voltage is +12V (positive on center pin, +/-10%, 35W), and connect it to the power connector (J35).
5. Power-up the board. The '3V3 OK' LED and '5V OK' on the PB926EJ-S should both be lit.
6. If using Multi-ICE, run Multi-ICE Server, press ctrl-L and load the relevant manual configuration file from the \boardfiles\multi-ice directory. Depending on the version of Multi-ICE used it may also be necessary to add new devices to Multi-ICE. Please refer to \boardfiles\irlength\_arm.txt for information on how to do this.
7. If using the USB connection, ensure that your PC has correctly identified an ARM® RealView™ ICE Micro Edition device is connected to the USB port. If the Windows operating system requires a USB driver to be installed please refer to \boardfiles\USB\_Debug\_driver\readme.txt.
8. If using Real View ICE (RVI), you must ensure that the RVI unit is powered and has completed its start-up sequence (check the LEDs on the front panel have stopped flashing).
9. You can now run the relevant 'progcards' utility for the connection you have prepared above.
  - progcards\_multiice.exe for your Multi ICE connection
  - progcards\_rvi.exe for your RealView ICE connectionWhen using RVI select the target RVI box you are using.
10. Select the option for the Core Tile you are using. The utility will report its progress, it may take several minutes to download. A successful configuration download will be terminated with the message "Programming Successful".
11. Power down the boards.
12. Set the configuration switches to load Logic Tile FPGA image 0 (S2 on the Logic Tile set to all OFF).
13. Remove the CONFIG jumper link (J32), and power-up the boards. Ensure 'FPGA\_OK' (Logic Tile D6) and the 'POWER' (Core Tile D1) LEDs are lit. The Character LCD should show the Firmware and Hardware versions indicating that the Boot monitor firmware is running.
14. The system will now be fully configured and ready for use.

Figure 3-1 shows the contents of the Core Tile system for use with the PB926EJ-S. The PB926EJ-S exposes three AHB buses onto the Logic Tile stack. These are the M1 (Master bus from the development chip), the M2 (second Master bus from the development chip, shared with the Platform Baseboard), and S (a slave port into the Development Chip shared between the Logic Tile stack and the PCI system). The arbiter for selection of the appropriate master is embedded into this design in a cascaded format in order to allow flexible extension (ie each and every logic tile containing a master requires an accompanying arbiter).



VPB M1:	This is the PB926EJ-S Master1 bus. It is a bus which is mastered by the Development Chip.
VPB M2:	This is the PB926EJ-S Master2 bus. This bus is not used by this system. A default slave is added to the bottom tile to correctly handle any accesses attempted on the M2 bus.
VPB S:	This is the PB926EJ-S Slave bus. This allows a master to access shared slaves inside the Development Chip (including bridges back out onto the M1 and M2 buses). The PCI subsystem on the Platform Baseboard is also a master on this bus, hence the need for an Arbiter.
LTS:	This is the cascaded Logic Tile Slave bus. It allows connection of additional tiles to be stacked on top of this system as if they were connected to the VPB S bus.
TC:	The Test Chip bus is the exposed AHB system from the Test Chip fitted to the Core Tile. The Test Chip is the only master on this bus.

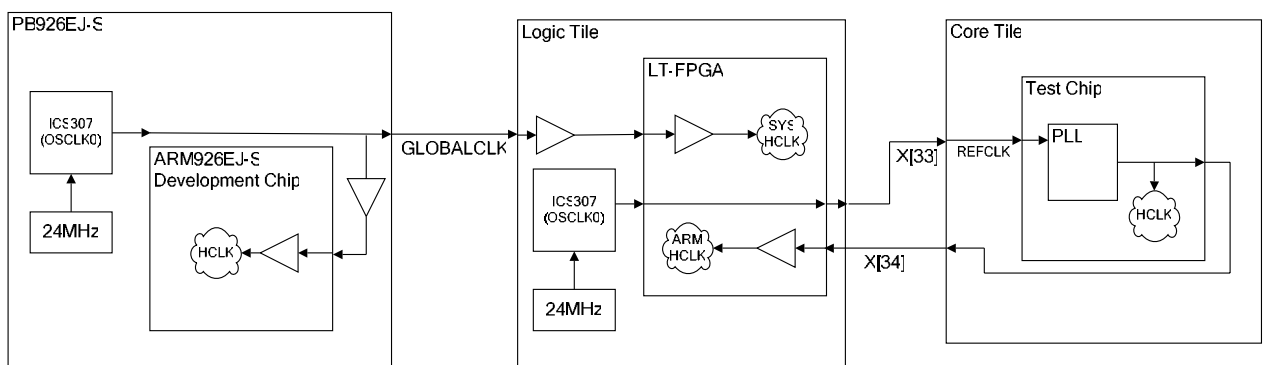
The function of each of these blocks is as follows:

1:1 Bridge:	This allows for cascaded arbitration of masters up the stack and to allow for reasonable timing to be met across a large number of subsystems in a stack.
Arbiter:	This allows for the sharing of the VPB S bus between multiple masters. The arbiter in this example arbitrates between three masters, the 1:1 bridge, the ASync Bridge, and any master on the system below (eg PCI on V/PB926EJ-S). Priority is given to the master on the tile below, and divided fairly between the two other masters.
ARM Decoder:	This selects which slave the Test Chip (TC bus) is currently accessing.
ASync Bridge:	This bridge allows the Test Chip to produce its own HCLK independent of the rest of the system. It also allows for better timing on the Test Chip AHB bus.
BPLogic:	This block is a matrix to allow cross triggering of multiple cores. The BP Logic blocks have a four channel shared communications line to allow each core to trigger other cores. There are two instantiations of this block in this design, as one is associated with the Test Chip, and in the case of the bottom tile positioned system one is associated with the Development Chip. Above the base position the second block is redundant.
Default:	This slave generates an error response to all otherwise unused addresses so as to avoid undefined bus responses.
DPRAM:	Dual Port RAM (FPGA Block Ram). It provides a pool of memory that both the local Test Chip and external masters can access simultaneously.
GTCRegs:	This block contains registers that control the Test Chip operation such as clock frequencies, power control and measurement, identification etc.
MailBox:	Allows the PB926EJ-S to interrupt the local Test Chip system and vice versa. It also allows for controlled flow of data through a mailbox register between the two systems.
Muxes:	These allow multiple slaves to be connected into a single AHB bus.
VPB Regs:	This block allows cross-triggering to the PB926EJ-S.
VPB M1 Decoder:	This determines whether the M1 bus is trying to access this system, and which particular slave is being accessed.
ZBTRam:	This is to provide each Test Chip with 2MB of relatively fast, low latency, private memory.

### 3.2 Clock architecture

The clock architecture is designed to make use of both the Test Chip and PB926EJ-S clock architectures.

The User Guides for all the boards used in this configuration explain the clock options they support.



**Figure 3-2 Clock architecture – HCLK**

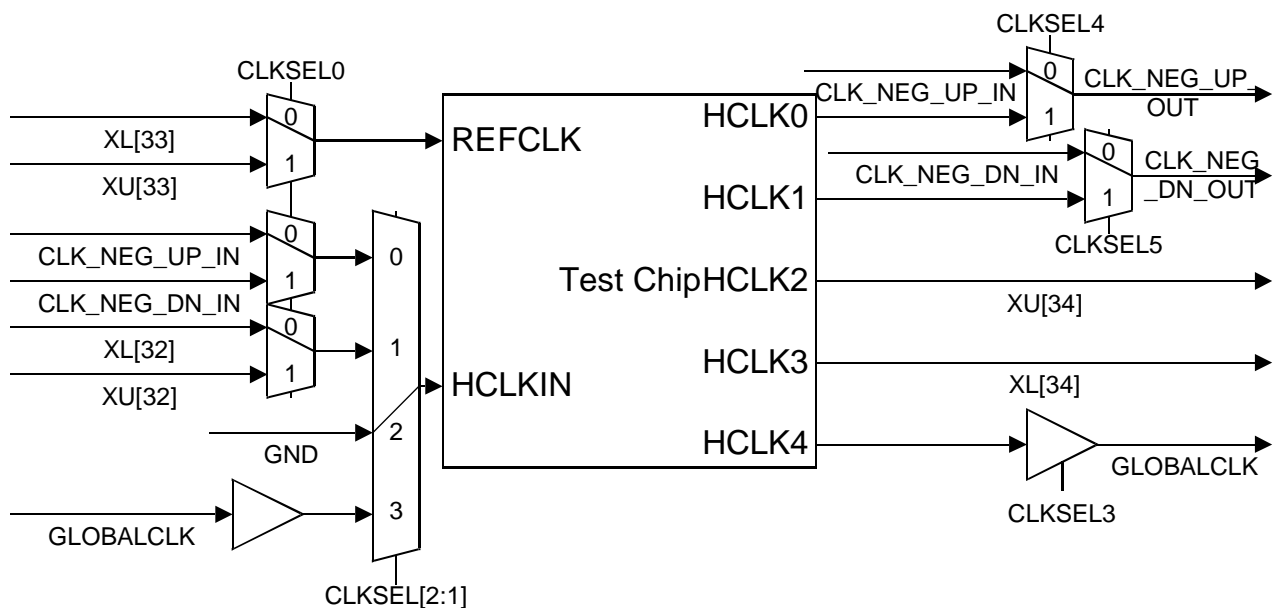
This clock architecture has been chosen for this system for a number of reasons.

1. The Core Tile can use either XL[33] or XU[33] as its reference clock. Since the Core Tile will always be connected on top of the PB926EJ-S, its clock multiplexers are configured so that REFCLK is sourced from XL[33]. Therefore the logic tile must drive a reference clock on XU[33].
2. The test chip then generates the HCLK for the ARM system based on its PLL configuration settings. REFCLK should not be used to clock any part of the design, since it may not be synchronous with HCLK.
3. The PLL is set into NORMAL mode (PLLBYPASS = 1'b0). This is so that the CPU on the Core Tiles can be used at its maximum frequency.

4. X[34] is chosen as the exported HCLK from the Test Chip, so that Test Chips with different VDDIO settings can be used. All signals on HDRX are able to support different voltage ranges. To understand how to set up boards for non 3.3V signaling read the appropriate User Guides.
5. GLOBALCLKIN is used to operate the AHB system busses synchronous to the baseboard. This signal is used, as its timing is guaranteed to be very close to the timing of GLOBALCLKIN on any other boards on Tile Site 1, including the baseboard itself.
6. There is an asynchronous bridge between the SYS clock domain and the ARM clock domain. This allows both the Logic Tile to interface to both the Core Tile and PB926EJ-S AHB domains.

### 3.2.1 Core Tile Clock configuration

In order to facilitate the above clock architecture the Core Tile Serial Stream must be correctly configured.



**Figure 3-3 Core Tile clock routing**

This configuration is chosen to match the overall clock architecture scheme. All the AHB Core Tiles designed for use in this system export their HCLK rather than synchronizing to an input HCLK (HCLKIN). Therefore HCLKIN is configured to always be connected to ground.

The CLK\_NEG\_UP signal, since it is not used for HCLKIN or HCLK, is configured to be passed through the Core Tile. This makes it available for use in other boards above the Core Tile.

Global Clock is not driven from the Core Tile directly, because there would be a delay between the Global Clock signal on Tile Site 1 and Tile Site 2.

### 3.2.2 Test Chip Clock Configuration

By default the test chip PLL is enabled (PLLBYPASS = 1'b0).

The settings for the PLL are dependent upon the specific Test Chip being used. Refer to the registers verilog file to see what defaults are being applied.



### 3.3 Reset Architecture

The baseboard has primary responsibility for reset control. Refer to the PB926EJ-S User Guide for how this works.

The Logic Tile and Core Tile resets are then generated from these signals (nSYSPOR and nSYSRST).

All Power On resets are directly driven from nSYSPOR. This signal is completely asynchronous to any system clocks.

The AHB resets both in the Logic Tile and Core Tile must be synchronized to the appropriate HCLK.

In the case of the System domain with the baseboard nSYSRST is synchronized to GLOBALCLK using flip flops inside the FPGA. This signal can safely occur a couple of clock cycles after it is released on the baseboard because the baseboard is guaranteed not to access the Logic Tile site in these first few clock cycles.

The ARM domain HRESETn are synchronized to HCLK (XU[34]) from the Test Chip. This can be safely delayed from the copy of nSYSRST provided to the Test Chip, as the Test Chip has exactly the same synchronization logic fitted inside it.

### 3.4 Interrupt Architecture

There is a simple interrupt controller embedded into the registers block at 0x10400040. For information on how to operate this block refer to section 4.

Some test chips also include their own Vectored Interrupt Controller inside the Test Chip. The following Test Chips presently have their own VIC:-

ARM926EJ-S UMC 0.18um LF712

ARM926EJ-S TSMC 0.18um LF711

This VIC allows the Test Chip to handle interrupts from embedded peripherals much faster (eg the DMA Controller). In order to allow interrupts through from outside the Test Chip, source 5 (nIRQ) and 4 (nFIQ) must be passed through this interrupt controller. To do this, do the following:-

Set 0x3FFE000C = 0x00000010. This Sets Source 4 to be routed to nFIQ rather than nIRQ.

Set 0x3FFE0014 = 0xFFFFFFFF. This clears any currently raised interrupts prior to enabling the interrupts.

Set 0x3FFE0010 = 0x00000030. This allows source 4 and source 5 to generate an interrupt to the processor.

## 4 Software Interface

The CT on PB926EJ-S example design provides a software interface with the majority of features required for a multi-core system. Note that all masters can access any slaves on the PB926EJ-S M1 bus, as the development chip loops the S bus back to the M1 and M2 bus.

### 4.1 Memory Map

Peripheral	Development Chip	CPU 0	CPU 1	CPU 2	CPU 3
GTC Regs	-	0x10400000	0x10400000	0x10400000	0x10400000
ZBT RAM	-	0x10800000	0x10800000	0x10800000	0x10800000
DPRAM	0x80100000, 0x90100000, 0xA0100000, 0xB0100000	0x0	0x0	0x0	0x0
VPB Regs	0x80000000, 0x90000000, 0xA0000000, 0xB0000000	-	-	-	-
MailBox	0x80200000, 0x90200000, 0xA0200000, 0xB0200000	0x10600000	0x10600000	0x10600000	0x10600000
Test Chip Internals	-	0x3FFC0000	0x3FFC0000	0x3FFC0000	0x3FFC0000

Figure 4-1 CT for PB926EJ-S example design memory map

### 4.2 Tile positions and extension addresses

The design can be used between one and four times in a logic tile stack with the option of additional logic added above this. The following table shows some examples of which address ranges are decoded by Tiles on the Logic Tile stack in different configurations.

Position	Example A - 1 Core Tile	Example B - 2 Core Tiles	Example C - 3 Core Tiles	Example D - 4 Core Tiles	Example E - 4 Core Tiles & Expansion	Example F - 2 Core Tiles & Expansion
Extension Logic	Not Fitted	Not Fitted	Not Fitted	Not Fitted	0xC0000000 - 0xFFFFFFFF	0xA0000000 - 0xFFFFFFFF
Core Tile 3	Not Fitted	Not Fitted	Not Fitted	0xB0000000 - 0xFFFFFFFF	0xB0000000 - 0xBFFFFFFF	Not Fitted
Core Tile 2	Not Fitted	Not Fitted	0xA0000000 - 0xFFFFFFFF	0xA0000000 - 0xAFFFFFFF	0xA0000000 - 0xAFFFFFFF	Not Fitted
Core Tile 1	Not Fitted	0x90000000 - 0xFFFFFFFF	0x90000000 - 0x9FFFFFFF	0x90000000 - 0x9FFFFFFF	0x90000000 - 0x9FFFFFFF	0x90000000 - 0x9FFFFFFF
Core Tile 0	0x80000000 - 0xFFFFFFFF	0x80000000 - 0x8FFFFFFF	0x80000000 - 0x8FFFFFFF	0x80000000 - 0x8FFFFFFF	0x80000000 - 0x8FFFFFFF	0x80000000 - 0x8FFFFFFF
ARM926EJ-S Development chip	0x0 - 0x7FFFFFFF	0x0 - 0x7FFFFFFF	0x0 - 0x7FFFFFFF	0x0 - 0x7FFFFFFF	0x0 - 0x7FFFFFFF	0x0 - 0x7FFFFFFF

Figure 4-2 Address Decoding Examples

The core tile subsystem position and address decoding is determined by using three pins on HDRY. Two pins driven from the tile below indicate the position of the Core Tile subsystem above the Baseboard and the third pin indicates whether there are any additional systems above this tile.

YL[61:60] : ID of this Core Tile (0-4)  
YU[59] : Additional logic present above this tile.

The tile position is also used to decide whether to drive the remaining signals required at the bottom of the PB926EJ-S stack, such as for the M2 bus, and usage of the HDRY fold over signals. The requirements for how to drive unused signals at the base of a PB926EJ-S system are explained in its accompanying documentation.

### 4.3 GTC Regs

This block is based upon the core module register block in Integrator Core Modules. Its function is to control all the local resources such as Test Chip control signals, clock generation and routing, power control, DAC readings, and interrupts. The GTC Regs area is private to each core as its function is to control that individual core. Most of these registers are similar to their Core Module (CM\_) counterparts. POR means Power On Reset (reset that occurs only at power up, and static means the contents never change. RESET indicates the contents are cleared by a normal reset.

Register	Address	read/write	reset by
-- generic registers			
CT_ID	0x10400000	r	static
CT_PROC	0x10400004	r	static
CT_OSC	0x10400008	r/w	POR
CT_CTRL	0x1040000C	r/w	RESET
CT_STAT	0x10400010	r	RESET
CT_LOCK	0x10400014	r/w	RESET
CT_LMBUSCNT	0x10400018	r	RESET
CT_AUXOSC	0x1040001C	r/w	POR
CT_SDRAM	0x10400020	r/w	POR
CT_INIT	0x10400024	r/w	POR
CT_REFCNT	0x10400028	r	RESET
-- flags registers			
CT_FLAGS	0x10400030	r	RESET
CT_FLAGSS	0x10400030	w	RESET
CT_FLAGSC	0x10400034	w	RESET
CT_NVFLAGS	0x10400038	r	POR
CT_NVFLAGSS	0x10400038	w	POR
CT_NVFLAGSC	0x1040003C	w	POR
-- interrupt controller			
CT_IRQ_STATUS	0x10400040	r	RESET
CT_IRQ_RAW	0x10400044	r	RESET
CT_IRQ_ENABLE	0x10400048	r	RESET
CT_IRQ_ENABLES	0x10400048	w	RESET
CT_IRQ_ENABLEC	0x1040004C	w	RESET
CT_SOFT_INT	0x10400050	r	RESET
CT_SOFT_INTS	0x10400050	w	RESET
CT_SOFT_INTC	0x10400054	w	RESET
CT_FIQ_STATUS	0x10400060	r	RESET
CT_FIQ_RAW	0x10400064	r	RESET
CT_FIQ_ENABLE	0x10400068	r	RESET
CT_FIQ_ENABLES	0x10400068	w	RESET
CT_FIQ_ENABLEC	0x1040006C	w	RESET
-- cross triggering			
CT_DBGXTRIG	0x10400070	r/w	RESET
-- voltage and power management			
CT_VOLTAGE0	0x10400080	r/w	POR
CT_VOLTAGE1	0x10400084	r/w	POR
CT_VOLTAGE2	0x10400088	r/w	POR
CT_VOLTAGE3	0x1040008C	r	POR
-- CT PLD Control			
CT_PLD_CTRL	0x10400094	r/w	POR
-- voltage and power management (cont.)			
CT_VOLTAGE4	0x104000A0	r	POR
CT_VOLTAGE5	0x104000A4	r	POR
CT_VOLTAGE6	0x104000A8	r	POR
CT_VOLTAGE7	0x104000AC	r	POR/RESET
-- Memory Expansion SPD memory			
CT_SPDBASE	0x10400100	r	POR

#### 4.3.1 CT\_ID

This register allows the user to determine the type of board that this is. It contains information about the build of the fpga, and the type of system that the FPGA image is designed for. It is a read-only register, with values coded into the RTL.

31	24	23	16	15	12	11	4	3	0
MAN			ARCH		FPGA		BUILD		REV

**CT\_ID Identification**

3:0 Revision  
 11:4 Build value (version) for internal development 0x00-0x99, binary coded decimal  
 15:12 FPGA type  
 23:16 Architecture  
 31:24 Manufacturer

All FPGA images associated with Core Tiles should contain identification registers. The allocation of bit fields is described in Table 4-1.

Bits	Name	Description	Allowable values
31:24	MAN	manufacturer	0x41 = ARM, other values may be used for special customer/partner applications. In general use the same encoding as for processor cores.
23:16	ARCH	architecture	Core module encoding as follows: bit [1:0] – processor bus type, 00 = ASB, 01 = 7TDMI, 10 = AHB bit [3:2] – system bus type, 00 = ASB, 10 = AHB bit 4 – SDRAM data width, 0 = 32-bit, 1 = 64-bit bit 5– SDRAM burst length, 0 = 4 b00011010 = AHB processor bus, AHB system bus, 4x64-bit burst SDRAM controller
15:12	FPGA	FPGA type	0x8 = Xilinx Virtex II XC2V6000/XC2V8000
11:4	BUILD	build number	Build value or version for ARM identification, binary coded decimal e.g. 0x40 = version 40
3:0	REV	revision number	Release revision 0x0 = Revision A (ASB system) 0x1 = Revision B (AHB system) 0x2 = Revision C (Bluetooth AHB system) 0x3 = Revision D (AHB-lite system, i.e. Integrator/CP) 0x4 = Revision E (926 DevChip, Multi-layer AHB)

**Table 4-1: CT\_ID Registers**

### 4.3.2 CT\_PROC

This register allows the user to determine the type of Test Chip fitted to this board. It is usually set to zero to indicate that the Test Chip can be directly interrogated in its CP15 registers (CP15 r0).

31	24	23	16	15	4	3	0
PROCESSOR							
IMP		ARCH		PART		REV	

**CT\_PROC Processor Identification**

31:0 If value = 0x00000000, read CP15r0 for details  
**or**  
 3:0 Revision  
 15:4 Part  
 23:16 Architecture  
 31:24 Implementer

### 4.3.3 CT\_OSC

This register allows the user to set a range of different Test Chip Reference Clock (REFCLK) frequencies. This clock is used by the Test Chip PLL to generate the CPU clock and any other synchronous clocks. The formula to calculate the clock frequency is:-

$$\text{REFCLK} = 48 * (\text{VDW} + 8) / (\text{RDW} + 2) * \text{OD}.$$

The value entered into this register is only applied on either a reset of the core, or if the Force Immediate Update bit is set (bit 26). A 1 in bit 27 (NewVal) indicates that the value of CT\_OSC has been changed, but that change has not yet been applied to the clock.

In order to understand the range of permissible values for each of these variables please refer to the ICS307 datasheet.

31 28 2 2 2 2 2 22 19 1 16 15 9 8 0

Reserved	N e w V a l	F o r c e	R e s e r v e d	BM ODE	Reserved	OD	RDW	VDW
----------	----------------------------	-----------------------	--------------------------------------	-----------	----------	----	-----	-----

#### CT\_OSC

#### Oscillator Divisors

$$\text{Freq} = 48(\text{VDW} + 8) / \text{OD}(\text{RDW} + 2)$$

8:0 CPU clock divider VDW

15:9 CPU clock divider RDW

18:16 CPU clock output divider (OD)

000 divide by 10

001 divide by 2

010 divide by 8

011 divide by 4

100 divide by 5

101 divide by 7

110 divide by 9

111 divide by 6 (default)

22:19 Unused (reserved)

24:23 Memory Bus mode (read only)

00 ARM7x0 processors (select processor bus mode by writing to CT\_CTRL register)

01 ARM9x0 processors (select processor bus mode by writing to CP15r1)

10 ARM9x6 processors (select processor bus mode by writing to CT\_INIT register)

11 ARM10x00 processors (select processor bus mode by writing to CT\_INIT and CP15r1)

25 Unused (reserved)

26 Force immediate update

27 New Value pending

31:28 Unused (reserved)

**Note:** This is the input clock to the processor and maybe the reference for the PLL, or the core clock in bypass mode. External memory clock is derived from within the processor device. A second oscillator is available but is not programmed via this register. See CT\_AUXOSC for details.

### 4.3.4 CT\_CTRL

This register allows control over extra hardware features. In this example design it is used to control the four LEDs fitted to the Logic Tile.

31	28	27			2	1	0
LED		Reserved				M B D E T	R e s e r v e d

**CT\_CTRL**      **Control**

0            Unused (reserved)  
 1            Motherboard detect, 0 = motherboard, 1 = stand alone (read)  
 27:2        Unused (reserved)  
 31:28       User LEDs 3:0

**4.3.5 CT\_STAT**

This register shows the current status of the SSRAM configuration, the User Switches fitted to the Logic Tile, and the position of this board in a stack.

31	28	27	24	23	16	15	8	7	0
SW		Reserved		SSRAMSIZE		SILICON ID		POS	

**CT\_STAT**      **Status**

7:0            Master Tile number in stack  
               0x00 = Core Tile 0  
               0x01 = Core Tile 1  
               0x02 = Core Tile 2  
               0x03 = Core Tile 3  
 15:8          Silicon ID (board specific field)  
 23:16        SSRAM memory size  
               Appending 0x0000 gives memory size:  
               0x20 => SSRAM=2MB (0x200000)  
 27:24        Unused (reserved)  
 31:28        User switches 3:0

**4.3.6 CT\_LOCK**

This register prevents accidental writes to important system registers. It locks the following registers: CT\_OSC, CT\_VOLTAGE0, CT\_VOLTAGE1, CT\_VOLTAGE2, CT\_VOLTAGE3, CT\_PLD, CT\_REFRESH, CT\_AUXOSC, and CT\_INIT. In order to write to any of these registers, unlock them by writing 0xA05F to CT\_LOCK. In order to relock these register write any value other than 0xA05F to this register.

31	17	1	15	0
Reserved				LOCKVAL
				L O C K E D

**CT\_LOCK**      **Lock**

15:0          Lock value, write 0xA05F to unlock, any other value locks  
 16            Lock bit, read 1 = locked, 0 = unlocked  
 31:17        Reserved

**4.3.7 CT\_LMBUSCNT**

This register allows the user to determine the number of Test Chip HCLK cycles between two accesses to this register. It can be used for basic profiling or time control.

31	0
LMBUSCNT	

**CT\_LMBUSCNT**    **Local Memory Bus Cycle Counter**

31:0            32-bit count value, resets at 0 and counts up

**4.3.8 CT\_AUXOSC**

This register allows the user to set a range of different auxiliary clock frequencies. This clock is unused in the default example design, but may be used by the user in future designs. The PLL control for the Test Chip can also be set in this register when the Test Chip makes use of the ARM\_PLL\* signals. To determine whether your specific Test Chip does, refer to the Reference Manual accompanying your Test Chip. The formula to calculate the clock frequency is:-

$REFCLK = 48 * (VDW + 8) / (RDW + 2) * OD$ .

In order to understand the range of permissible values for each of these variables please refer to the ICS307 datasheet.

31	28	27	24	2	2	2	2	1	1	16	15	9	8	0
				3	2	1	0	9	8					
PLLOUTDIV	PLLREFDIV	Reserved	PLL CTRL	Reserved	OD	RDW			VDW					

**CT\_AUXOSC****Auxiliary Oscillator Divisors**

Freq =  $48(\text{VDW} + 8)/\text{OD}(\text{RDW}+2)$   
 defaults give Freq =  $48 \times 263 / (6 \times 65) = 32.37\text{MHz}$

8:0	Aux clock divider VDW
	01111111 255 (default)
15:9	Aux clock divider RDW
	011111 63 (default)
18:16	Aux clock output divider (OD)
	000 divide by 10
	001 divide by 2
	010 divide by 8
	011 divide by 4
	100 divide by 5
	101 divide by 7
	110 divide by 9
	111 divide by 6 (default)
19	Unused (reserved)
21:20	PLLCTRL[1:0]
23:22	Unused (reserved)
27:24	PLLREFDIV[3:0]
31:28	PLLOUTDIV[3:0]

**4.3.9 CT\_SDRAM**

31	20	19	16	15	12	11	8	7	6	5	4	2	1	0
Reserved				Banks		Columns		Rows		Reserved	SPD	Size	CAS	

**CT\_SDRAM****SDRAM Control**

1:0	CAS latency
	0x = reserved
	10 = cas latency = 2 (default)
	11 = cas latency = 3
4:2	Memory size
	000 = 16MB
	001 = 32MB
	010 = 64MB
	011 = 128MB
	100 = 256MB
	101 = 512MB
	110 = 1GB
	111 = 2GB
5	SPD read, 0 = SPD information not loaded, 1 = loaded
7:6	Reserved
11:8	Number of rows
15:12	Number of columns
19:16	Number of banks
31:20	Reserved

**4.3.10 CT\_INIT**

This register is used for the initialization of the Test Chip. It controls the configuration of the Test Chip at reset. In order for these values to take effect a soft reset must be applied after changing the contents of the register.

3	3	29	24	23	17	1	15	8	7	6	4	3	2	1	0
1	0					6									

Reserved	USERIN	Reserved	INITRAM	PLLFBDIV	Reserved	HCLKDIV	Reserved	VINITHI	PLLBYPASS
----------	--------	----------	---------	----------	----------	---------	----------	---------	-----------

**CT\_INIT Test Chip Initialisation**

0	PLLBYPASS, 0 = off, 1=on (default), takes effect only after reset
1	PLLBYPASS (value read from pin after reset) - read only
2	VINITHI, 0 = vectors at 0x0, 1 = vectors at 0xffff0000
3	Reserved
6:4	HCLK divider
7	Reserved
15:8	PLLFBDIV[7:0]
16	Internal RAM enable, 0 = disabled (default), 1 = enabled
23:17	Reserved
29:24	USERIN[5:0]
31:30	Reserved

**4.3.11 CT\_REFCNT**

This register allows the user to determine the number of 24MHz clock cycles that have transpired between two accesses to this register. It can be used for basic profiling, or time control on a Core Tile System.

31	0
REFCNT	

**CT\_REFCNT Reference Clock (24MHz) Counter**

31:0	32-bit count value, resets at 0 and counts up
------	---

**4.3.12 CT\_FLAGS**

These flags can be used by software. The NVFlags are not cleared on a reset, so can be used for retaining data during a reset. For example this can be used to change some startup settings.

**CT\_FLAGS Flags**

31:0	1 = flag set, 0 = flag clear
------	------------------------------

**CT\_FLAGSS Flags Set**

31:0	1 = set flag, 0 = no change
------	-----------------------------

**CT\_FLAGSC Flags Clear**

31:0	1 = clear flag, 0 = no change
------	-------------------------------

**CT\_NVFLAGS Non-volatile Flags**

31:0	1 = flag set, 0 = flag clear
------	------------------------------

**CT\_NVFLAGSS Non-volatile Flags Set**

31:0	1 = set flag, 0 = no change
------	-----------------------------

**CT\_NVFLAGSC Non-volatile Flags Clear**

31:0	1 = clear flag, 0 = no change
------	-------------------------------

**4.3.13 CT\_IRQ**

This design implements a 6-bit IRQ interrupt controller and a 6-bit FIQ interrupt controller. Refer to the table of bit assignments below for details.

These registers can be used to determine which interrupt sources are active, enable and disable them, and configure whether they cause a normal or fast interrupt. This is for interrupts going to the Core Tile. The baseboard has its own interrupt control registers for interrupts going to the ARM926EJ-S Development Chip (see PB926EJ-S User Guide for a description of these).

31	6	5	4	3	2	1	0
Reserved						V	M
						P	B
						M	C
						B	O
						C	C
						O	S
						O	O



0	SOFTINT	Local Soft interrupt (from CT_SOFT_INT register)
1	COMMRX	Communications channel (DCC) receive interrupt source
2	COMMTX	Communications channel (DCC) transmit interrupt source
3	MAILBOXFULL	The inbound MAILBOX is currently full
4	MAILBOXEMPTY	The outbound MAILBOX is currently empty
5	VPBREGINT	Remote Soft interrupt from Mailbox

#### 4.3.14 CT\_DBGXTRIG

31	16	15	12	11	8	7	4	3	0
Reserved		REQINEN	ACKOUTEN	ACKSOFTEN	ACKSOFT				

Application Note 125  
ARM DAI 0125B

7:4	DBGACKSOFTEN[3:0]	- tri-state enable for DBGACKSOFT[3:0]
11:8	DBGACKOUTEN[3:0]	- tri-state enable for DBGACK from the processor
15:12	DBGREQINEN[3:0]	- enable DBGXTRIG[3:0] to drive processor DBGRQ

**Note:** This functionality allows cross triggering of breakpoints between CPUs. This is particularly important for DSP debugging. The implementation makes use of system bus Z[54:51] lines, designated DGBXTRIG[3:0] respectively. Each triggerable unit needs its own copy of these registers.

#### 4.3.15 CT\_VOLTAGE

This register can be used to access the DACs and ADCs fitted to the Core Tile.

31	20	19	8	7	0
ADC			ADC		DAC

**Note:** This register allows voltage control and read back over a number of DAC and ADC channels. Depending on the Core Tile build not all three Voltage domains may be present, and the scaling of the voltage domains varies between builds. It is recommended that when scaling the voltage, the algorithm to set the new voltage works by incrementing the DAC and using to ADC to determine the effect this is having. See the Core Tile User Guide for a full explanation of this circuitry, and how to make use of these values.

<b>CT_VOLTAGE_CTL0</b>	<b>Voltage Control 0</b>
7:0	Core voltage A DAC value
19:8	VDDCORE1 voltage ADC value (12 bits)
31:20	VDDCORE1 DIFF
<b>CT_VOLTAGE_CTL1</b>	<b>Voltage Control 1</b>
7:0	Core voltage B DAC value
19:8	VDDCORE2 voltage ADC value (12 bits)
31:20	VDDCORE2 DIFF
<b>CT_VOLTAGE_CTL2</b>	<b>Voltage Control 2</b>
7:0	Core voltage C DAC value
19:8	VDDCORE3 voltage ADC value (12 bits)
31:20	VDDCORE3 DIFF
<b>CT_VOLTAGE_CTL3</b>	<b>Voltage Control 3</b>
0	Core voltage A enable (nSHDN) (RESET)
1	Core voltage B enable (nSHDN) (RESET)
2	Core voltage C enable (nSHDN) (RESET)
7:3	Reserved
19:8	VDDCORE4 voltage ADC value (12 bits)
31:20	VDDCORE4 DIFF
<b>CT_VOLTAGE_CTL4</b>	<b>Voltage Control 4</b>
7:0	Reserved
19:8	VDDCORE5 voltage ADC value (12 bits)
31:20	VDDCORE5 DIFF
<b>CT_VOLTAGE_CTL5</b>	<b>Voltage Control 5</b>
7:0	Reserved
19:8	VDDCORE6 voltage ADC value (12 bits)
31:20	VDDCORE6 DIFF
<b>CT_VOLTAGE_CTL6</b>	<b>Voltage Control 6</b>
7:0	Reserved
19:8	VDDPLL1 voltage ADC value (12 bits)
31:20	VDDPLL2 voltage ADC value (12 bits)
<b>CT_VOLTAGE_CTL7</b>	<b>Voltage Control 7</b>
7:0	Reserved
19:8	VDDIO voltage ADC value (12 bits)
31:20	Test Point voltage ADC value (12 bits)

### 4.3.16 CT\_PLD\_CTRL

This register allows the user to communicate with the configurable options on the Core Tile PLD. These values are communicated across the Serial Interface to the Core Tile PLD.

31	28	2	25	2	23	13	1	1	1	9	4	3	0
		7		4			2	1	0				
PLD ID	Reserved	S D R A M	Reserved				P G O O D	Rese rved	CLKSEL			ZCTL	

#### CT\_PLD\_CTRL

#### Core Tile PLD Control/Status

0	ZCTL0: Set high to break Z[31:0] bus
1	ZCTL1: Set high to break Z[63:32] bus
2	ZCTL2: Set high to break Z[95:64] bus
3	ZCTL3: Set high to break Z[127:96] bus
4	CLKSEL0: Set low for tile below/high for tile above
6:5	CLKSEL[2:1]: HCLK gets CLK_NEG, X_HCLK, GND, CLK_GLOBAL
7	CLKSEL3: Set low to drive CLK_GLOBAL
8	CLKSEL4: NEG_UP_OUT assigned HCLK, NEG_UP_IN
9	CLKSEL5: NEG_DN_OUT assigned HCLK, NEG_DN_IN
12	PGOOD. Active high signal indicating that power supply on Core Tile board is working OK.
24	SDRAM mode. 1=MemExp. 0=CP Legacy
31:28	PLD_ID. Determines the current build/type of the PLD image.

### 4.3.17 CT\_SPDBASE

This is a copy of the contents of the SPD or other EEPROM associated with the SDRAM fitted to this system.

#### CT\_SPDBASE

#### SPD Base

Byte 2	Memory type
Byte 3	Number of Row addresses
Byte 4	Number of Column addresses
Byte 5	Number of Banks
Byte 18	CAS latencies supported
Byte 31	Module bank density (MB divided by 4)
Byte 63	Checksum
Byte 64-71	Manufacturer
Byte 73-90	Module Part Number

Some Tiles may have a memory expansion card fitted. These Memory Expansion Cards are fitted with a serial presence detect (SPD) EPROM, containing information about the manufacturer and memory type.

On power up, the FPGA reads this SPD EPROM and stores the data in 256 bytes of RAM which can be randomly accessed at addresses above 0x10400100. A bit is set in the CT\_SDRAM register to indicate when the EPROM has been read and information stored in RAM. If there is no DIMM fitted then this process reads invalid data, but the bit will still be set indicating completion. To check for valid SPD data it is necessary to checksum the memory. The algorithm is:

1. add up all bytes 0 to 62
2. logical AND the result with 0xff
3. compare the result with byte 63
4. if the two values match then the SPD data is valid

## 4.4 ZBT RAM

The ZBT RAM controller can access 2 MB of zero wait state memory private to the local CPU. The memory is aliased across the address range 0x10800000 – 0x10FFFFFF.

## 4.5 DPRAM

The DPRAM slave block contains 256kB of dual port block ram. It is full dual port memory so transactions can simultaneously take place in both the local CPU clock domain and in the Versaile/PB926EJ-S M1 clock domain.

The DPRAM appears at address 0x0 for the local CPUs, which is where the boot code for each processor is located. Refer to the section on booting for more information on the proposed boot mechanism.

The baseboard is accessed at memory addresses immediately above the DPRAM (0x00040000 onwards).

## 4.6 VPB Regs

These are registers that can be accessed by all masters in the system rather than being private to the local Core Tile.

Register	Address	read/write	reset by
-- cross triggering			
VPB_DBGXTRIG	0x80000000	r/w	RESET

### 4.6.1 VPB\_DBGXTRIG

31	16	15	12	11	8	7	4	3	0
Reserved					REQINEN	ACKOUTE N	ACKSOFT N	ACKSOFT	

CT_DBGXTRIG	Debug Cross-trigger
3:0	DBGACKSOFT[3:0] - soft DBGACK for cross-triggering in software
7:4	DBGACKSOFTEN[3:0] - tri-state enable for DBGACKSOFT[3:0]
11:8	DBGACKOUTEN[3:0] - tri-state enable for DBGACK from the processor
15:12	DBGREQINEN[3:0] - enable DBGXTRIG[3:0] to drive processor DBGRQ

**Note:** This functionality allows cross triggering of breakpoints between CPUs. This is particularly important for DSP debugging. The implementation makes use of system bus Z[54:51] lines, designated DGBXTRIG[3:0] respectively. Each triggerable unit needs its own copy of these registers.

## 4.7 MailBox

The MailBox allows flexible communication between processors in the system. It allows for data to be safely passed between the Test Chip and PB926EJ-S M1 clock domains, and allows for implementation of either a polling, or interrupt driven communications method.

The MailBox also contains registers to allow software driven interrupts to be sent between processors, and instantiates a basic interrupt controller to allow multiple interrupts to drive a PB926EJ-S interrupt line.

See MailBox Software example for more information on how to make use of this block.

Register	TC Address	read/write	VPB Address	read/write
-- generic registers				
ARM_MBDATA	0x10600000	r	MB_BASE	w
ARM_MBSTAT	0x10600004	r	MB_BASE+0x4	r
ASOFT_INTS	0x10600008	r	MB_BASE+0x8	r/w
ASOFT_INTC	0x1060000C	r/w	MB_BASE+0xC	r
VPB_MBDATA	0x10600010	w	MB_BASE+0x10	r
VPB_MBSTAT	0x10600014	r	MB_BASE+0x14	r
VSOFT_INTS	0x10600018	r/w	MB_BASE+0x18	r
VSOFT_INTC	0x1060001C	r	MB_BASE+0x1C	r/w
IRQ_STATUS	0x10600020	r	MB_BASE+0x20	r
IRQ_ENABLES	0x10600028	r	MB_BASE+0x28	r/w
IRQ_ENABLEC	0x1060002C	r	MB_BASE+0x2C	r/w

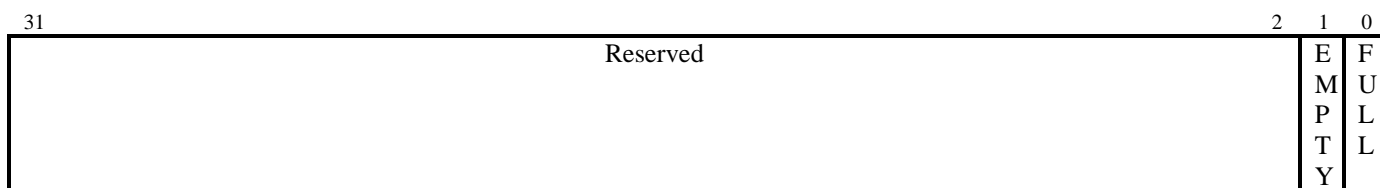
**Note:** The two different address columns in the above table refer to the address at which the Test Chip can access registers in the Test Chip AHB Clock domain, and the address at which the Development Chip can access registers in the PB926EJ-S M1 AHB Clock domain. The VPB address varies depending on the position of this system in the stack. The address shown here is for the system at the bottom of the stack, but the address could be any of 0x80200000, 0x90200000, 0xA0200000, 0xB0200000. Refer to section 2.2 to understand which address is applicable.

### 4.7.1 ARM\_MBDATA

31	0
MBDATA IN	

**ARM\_MBDATA MailBox Into Test Chip Data**

31:0 Data value to be passed from VPB domain to Test Chip.

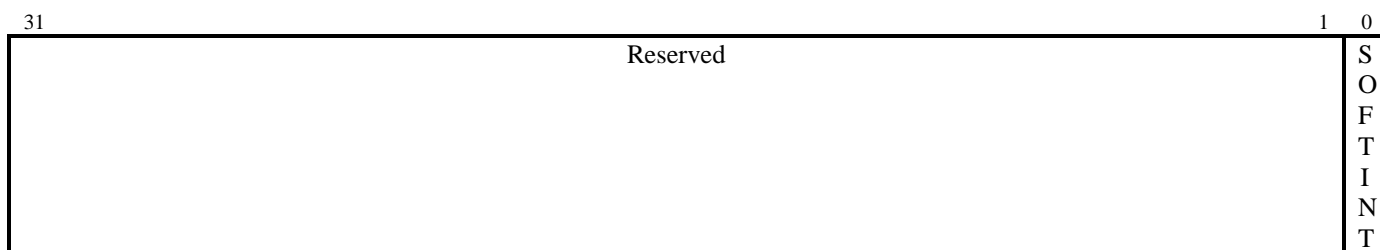
**4.7.2 ARM\_MBSTAT****ARM\_MBSTAT Test Chip Mailbox Status**

0 MailBox into Test Chip FULL

1 MailBox into Test Chip EMPTY

**4.7.3 ASOFT\_INTS/C**

These registers set and clear an interrupt to the PB926EJ-S. This is to allow software running on the Core Tile system to generate a specific interrupt to the Baseboard.

**ASOFT\_INTS Test Chip to PB926EJ-S Soft Interrupt Set**

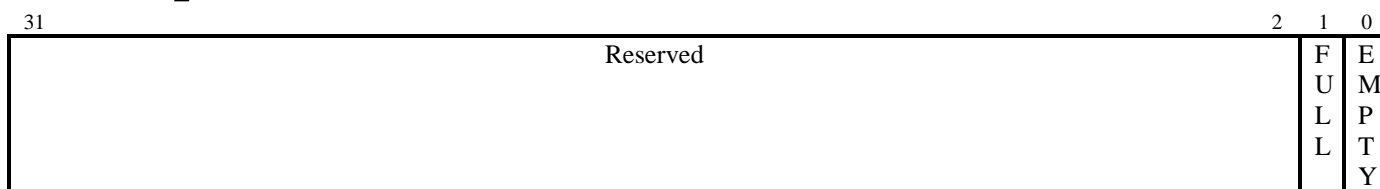
0 1 = Local Soft Interrupt set, 0 = no change

**ASOFT\_INTC Test Chip to PB926EJ-S Soft Interrupt Clear**

0 1 = Local Soft Interrupt clear, 0 = no change

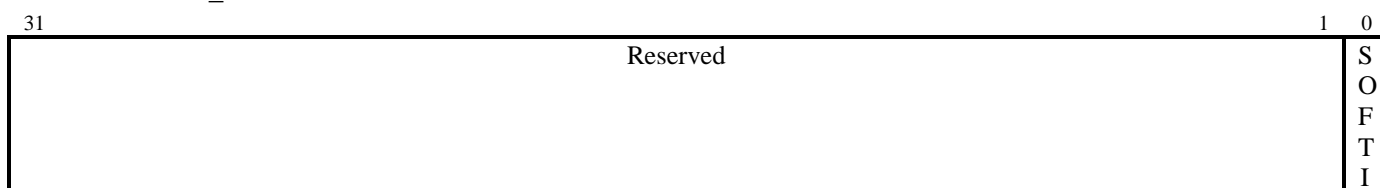
**4.7.4 VPB\_MBDATA****VPB\_MBDATA MailBox Out of Test Chip Data**

31:0 Data value to be passed from Test Chip domain to VPB.

**4.7.5 VPB\_MBSTAT****VPB\_MBSTAT PB926EJ-S to Test Chip Mailbox Status**

0 MailBox from Test Chip EMPTY

1 MailBox from Test Chip FULL

**4.7.6 VSOFINTS/C**

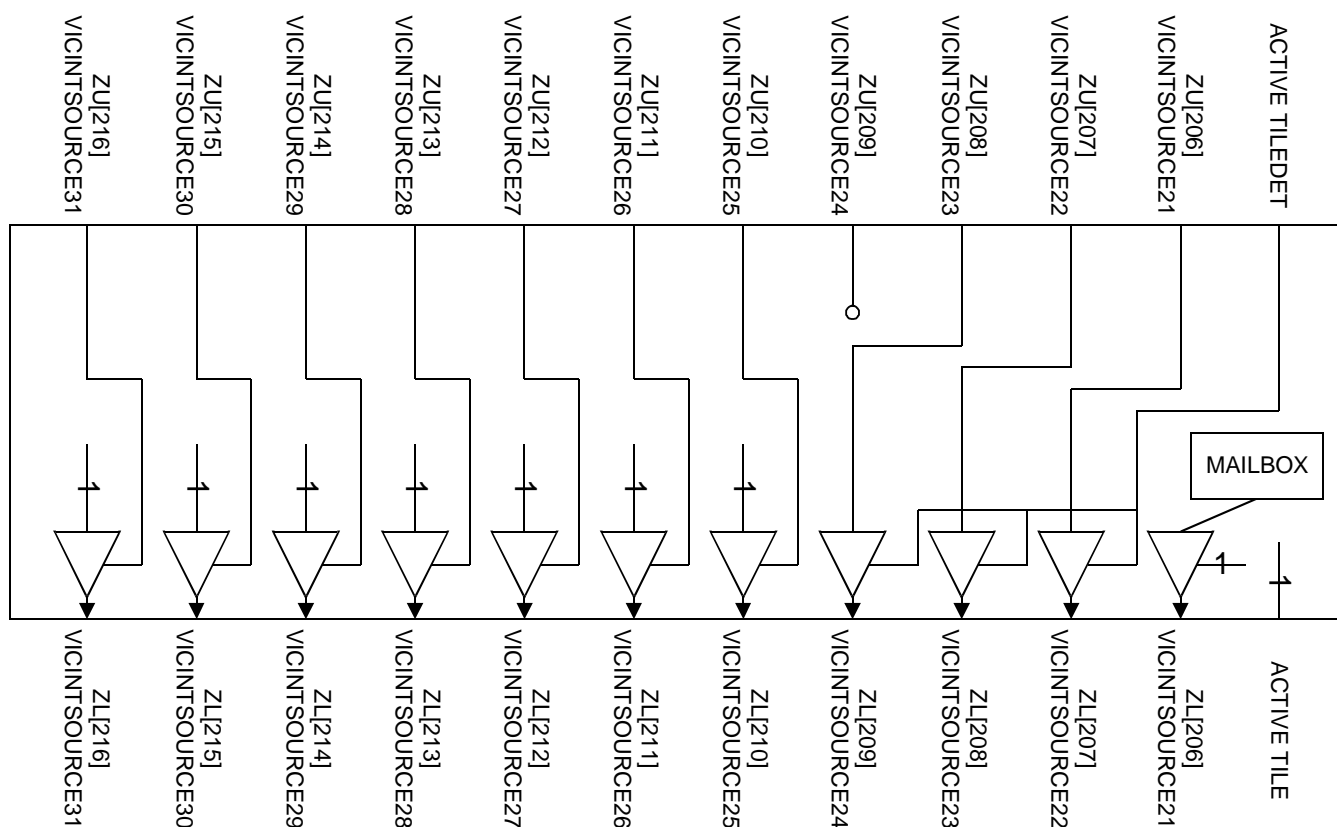
VSOFT_INTC	PB926EJ-S to Test Chip Soft Interrupt Clear
0	1 = Local Soft Interrupt clear, 0 = no change

#### 4.7.8 VPB IRQ

31	3	2	1	0
Reserved	M	M	S	
	B	B	O	
	E	F	F	
	M	U	T	
	P	L	I	
	T	L	N	
	Y		T	

<b>IRQ_ENABLEC</b>	<b>IRQ Enable Clear</b>
2:0	1 = Interrupt IRQ Enable clear, 0 = no change

Application Note 125  
ARM DAI 0125B



**Figure 4-2 CT System VPB Interrupt Overview**

## 4.8 Test Chip Internal Bus

Some test chips have some embedded peripherals and/or memory. The area 0x3FFC0000 – 0x3FFFFFFF is reserved for this purpose.

In the case of the ARM926EJ-S (LF711 and LF712) processors there are the following blocks:-

0x3FFC0000 – 0x3FFCFFFF Embedded GPIO(PL061 PrimeCell)/EPHEM IO controller.

0x3FFD0000 – 0x3FFDFFFF Embedded DMA controller (PL081 PrimeCell)

0x3FFE0000 – 0x3FFEFFFF Embedded VIC (PL190 PrimeCell)

0x3FFF0000 – 0x3FFFFFFF On-chip RAM.

The other Test Chips do not make use of this area at present.

For information on how to use these blocks refer to the appropriate PrimeCell manual.

## 5 Boot Operation

The system is designed as an asymmetric system (for example only the PB926EJ-S Development Chip can set up the main memory controllers). As such, the PB926EJ-S is responsible for bringing the systems out of reset, and providing suitable clocks to the global AHB buses.

Each Core Tile System has pre-initialized memory at address 0x0. This code is inside the DPRAM and is defined in the RTL for the FPGA build. The boot code is thus embedded into the FPGA image. This memory has the following program in it:-

Addr	Instruction	Mnemonic	Comments
0x0	0xEA000006	B 0x20	; Jumps past Exception Handlers
0x4	0xEAFFFFFEE	B 0x4	; Dummy Undefined instruction handler
0x8	0xEAFFFFFEE	B 0x8	; Dummy Software Interrupt handler
0xC	0xEAFFFFFEE	B 0xC	; Dummy Prefetch Abort handler
0x10	0xEAFFFFFEE	B 0x10	; Dummy Data Abort handler
0x14	0xEAFFFFFEE	B 0x14	; Reserved
0x18	0xEAFFFFFEE	B 0x18	; Dummy IRQ handler
0x1C	0xEAFFFFFEE	B 0x1C	; Dummy FIQ handler
0x20	0xE3A00000	MOV r0,#0	; Set counter to 0
0x24	0xE3A01B40	MOV R1,#0x40, 22	; Set maximum counter value to 0x10000
0x28	0xE59F2018	LDR R2,0x00000048	; Set pointer to LED control register 0x1040000C
0x2C	0xE5923000	LDR R3,[R2,#0]	; Load current contents of LED control register
0x30	0xE2800001	ADD R0,R0,#1	; Increment counter
0x34	0xE1500001	CMP R0,R1	; Check if counter is at maximum value
0x38	0x02833540	ADDEQ R3,R3,#0x40, 10	; If counter at max, add one to LED part register
0x3C	0x05823000	STREQ R3,[R2,#0]	; If counter at max, write new LED output
0x40	0x03A00000	MOVEQ R0,#0	; If counter at max, reset counter
0x44	0xE59FF000	LDR PC,0x4C	; Return to start of loop.
0x48	0x1040000C	DCD 0x1040000C	; Data value of CT_CTRL register
0x4C	0x00000030	DCD 0x00000030	; Address value of LDR PC command
0x50	0x00000000	DCD 0x00000000	; Test Pattern 0
0x54	0x55555555	DCD 0x55555555	; Test Pattern 1
0x58	0xAAAAAAAA	DCD 0xAAAAAAAA	; Test Pattern 2
0x5C	0xFFFFFFFF	DCD 0xFFFFFFFF	; Test Pattern 3

This code flashes the LEDs on the corresponding Logic Tile system. This shows the operator that each CPU has been successfully brought out of reset, and has appropriate clocks applied.

The Test Patterns at the end are so the user can read these values if a fault on the data bus is suspected.

In order to now execute some more useful code with each additional CPU the following should be done by the Development Chip for the ARM926EJ-S (or a JTAG tool):

- 1) Write the code to be executed into any available memory except where the code is running from (DPRAM 0x30-0x4C).
- 2) Once this is done the address at 0x4C should be replaced by the start address for the new code. This is done by writing to the shared port of the DPRAM at, for example 0x8010004C.

In order to see an example of this look at the example boot routine software provided.

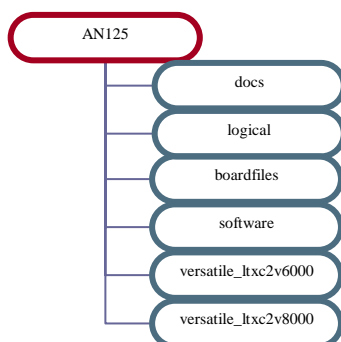
Note that in this code, the exception handlers (Undef, SWI, PF, Abort, D Abort, IRQ, FIQ) only trap the exception, by causing the processor to sit in a closed loop. On connection with a debugger it will be evident that an exception was hit, but in order to make proper use of exception handling the instructions at the relevant addresses must be replaced with branches to the exception handling routine.



## 6 RTL

All of the RTL for this design is provided as verilog. Example files are provided to allow building the system with Synplicity Synplify Pro and Xilinx ISE tools.

### 6.1 Directory Structure



The application note has directories. These are:

- Docs : Related documents including this document.
- Logical : All the verilog RTL required for the design.
- Progcards : The files required to program the design into ARM development boards.
- Software : ARM code to run on the AN125 system
- Versatile\_ltxc2v6000/versatile\_ltxc2v8000 : Synthesis and P&R scripts and builds for target boards.

### 6.2 Logical

The logical directory contains all the verilog required to build the system. The function of each block is shown earlier in Section 1.3.

Each PrimeCell or other large IP block has its own directory (for example Ahb2AhbSync).

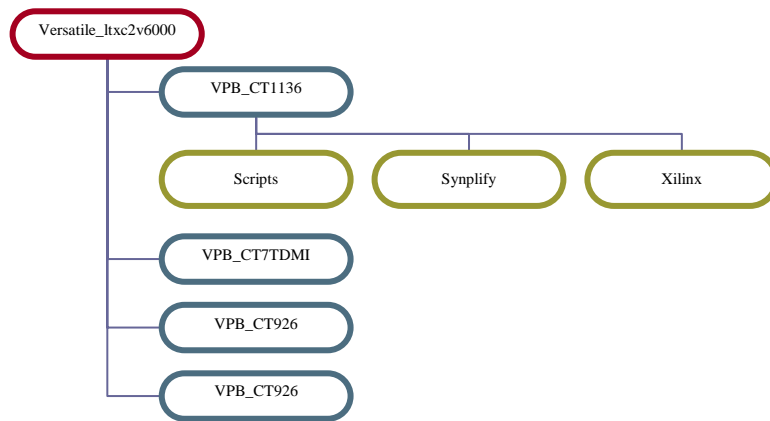
The top level for a system including a PB926EJ-S, LTXC2V4000+ and a CTxxx is in VPB\_CT.

The required registers, and their default settings are in the CTxxxRegs directories. Differences in hardware such as Test Chips means that often different Register settings are required for a system.

The remaining RTL to complete the system is in the CT\_xxx directories. There is a CT\_Generic directory containing RTL applicable to the majority of builds, and specific CT\_xxx directories for each different PCB in the Core Tile Range.

### 6.3 Targets

Each potential target has a directory (eg Versatile\_ltxc2v6000). Within these directories are all the different builds which exist for this target. In turn each of these has directories for each tool used in the build process.



## 7 Example Software

### 7.1 Boot routine

The 'boot' example, when executed by the Baseboard ARM926EJ-S Development Chip, changes the code being executed by the Core Tile Systems from their initial boot code to a function provided by the Development Chip. This is done by initializing a shared memory area, and then changing the branch address used in the Core Tile Boot Code.

#### 7.1.1 Source Code

This simple example consists of a header file with a definition of the registers to be used, and a C source code file with a function to be executed by the Core Tiles and a main() to be executed by the Development Chip.

#### 7.1.2 Functionality

At power up the Core Tile systems are flashing their own private LEDs to show they are functioning correctly. This code is stored in memory belonging to that Core Tile System. After the Development Chip has executed this example code the Core Tile systems will execute code from the baseboard SDRAM (0x100000). Each Core Tile reads the S6 switch bank on the Platform Baseboard, and flashes its associated GP\_LED accordingly. If the switch is down, the LED flashes. The mapping of the Core Tile System to the switches and LEDs, is that the system at the base of the stack corresponds to Switch 1. The next system corresponds to Switch 2. Up to 4 systems are supported by this example.

#### 7.1.3 Compilation Notes

It is necessary to observe the following guidelines if using this code:

- 1) The code must be compiled for an area of memory that is accessible at the same address by all processors in the system. The example uses 0x100000, which is baseboard SDRAM above the DPRAM range.
- 2) It is necessary to offset the function address by 0x4 in order to avoid the Stack save that each C function attempts. The compiler will add a stack save (STM R13!) instruction to the start of each function other than main(). Since the slave processors do not have a valid stack set up, they must not write to the address pointed to by R13.

### 7.2 Voltage Control

The 'control' example is intended to be executed directly by the Core Tile. It shows how to use the DAC and ADC parts fitted to the Core Tile. It can be used to confirm that the system is powered with correct voltages, and allows the user to implement changes to the programmable power supplies.

#### 7.2.1 Source Code

This example consists of a header file with a definition of the registers to be used, and a C source code file with functions to be executed by the Core Tile. This code cannot be run on the Development Chip, as the Development Chip does not have registers to read and set the voltage supplies.

#### 7.2.2 Functionality

The main() function provides the user with a menu to select the function to be run.

Read Identification parses the CT\_ID and CT\_STAT registers to determine the type of Core Tile fitted. This is important as different Core Tiles have different voltage requirements and specifications, so the user can use this information to determine the expected behaviour of this board.

The Read Voltages function reads all ADCs and presents the results to the user in a useful format (converted to voltage or current measurements).

The Set Voltage function allows the user to change the VDDCORE power supplies. Since different boards have different sensitivities to these numbers it is the responsibility of the user to ensure they know what they are doing prior to changing these values (see Core Tile User Guide for more information).

## 7.3 MailBox Example

The 'Mailbox' example when executed by the Development Chip sets up code which is run by the Development chip and a number of Core Tiles. Its purpose is to show how the MailBox and Software Interrupts can be used in a Multi-Core environment.

### 7.3.1 Source Code

This example has two header files with definitions of the registers to be used. One header file (apic.h) is taken from the PB926EJ-S software and defines the registers used for the Development Chip Interrupt handling routines. The other header file (mailbox.h) defines the remaining registers required for this code, including the mailbox register locations. A C source code file (mailbox.c) contains the bulk of the code. It contains functions to be executed by the Core Tiles and by the Development Chip. The Development Chip sets up the code in shared memory, and executes the main() function. All the Core Tiles then execute CoreTileCode(). The assembler file (irqsup.s) contains code for setting up the stack (prohibited in C), and adjusting the cpsr register (used to enable and disable interrupts amongst other things).

### 7.3.2 Functionality

At power up the Core Tile systems are flashing their own private LEDs to show they are functioning correctly. This code is stored in memory belonging to that Core Tile System. After the Development Chip has executed the main() function the Core Tile systems execute code from the baseboard SDRAM (0x100000).

The main() function establishes the number of Core Tiles to be used, and set them to run CoreTileCode(). It also sets up interrupts on the Development Chip, and writes a value to each Core Tiles MailBox.

Each Core Tile executes CoreTileCode() which sets up the Interrupts on each Core Tile. The interrupts are set to catch any interrupt generated as a result of a write to a mailbox. Since the development chip has written to each Mailbox, each Core Tile will then jump to their interrupt handling routine ctlIRQHandler(). They will fetch the data that was posted by the development chip. This data is defined as being the start of a range of numbers to examine for Prime Numbers. The Core Tiles will go through this range posting every positive result back to the Development Chip through the MailBox. Before posting a value the Core Tile will poll the MailBox status bits to ensure that the Development Chip has received the previous data before proceeding.

The Development Chip on receiving an Interrupt jumps to its Interrupt handling routine vpiIRQHandler. This checks to see whether the interrupt was generated by presence of data in the mailbox or a software generated interrupt (not a SWI) from the Core Tiles (the only possible sources in this example). If it was due to the MailBox having pending data in it, the Development Chip will read that data out of the Mailbox, and present it to the user as a result.

The Core Tiles will, on completion of finding prime numbers for their assigned range generate a software generated interrupt (not a SWI) to the Development Chip to indicate that they are now idle, and in need of a new range. They will then sit in an empty loop awaiting an interrupt as they were at the start.

The Development Chip, on seeing an interrupt being generated as a result of a Software Interrupt from a Core Tile, will post the value of the next range of numbers to be examined for Primes to that Core Tile. The entire process thus proceeds.

### 7.3.3 Compilation Notes

It is necessary to observe the following guidelines when using this code:

- 1) The code must be compiled for an area of memory that is accessible at the same address by all processors in the system. The example uses 0x100000, which is baseboard SDRAM above the DPRAM range.
- 2) It is necessary to offset the function address by 0x4 in order to avoid the Stack save that each C function attempts. The compiler will add a stack save (STM R13!) instruction to the start of each

function other than `main()`. Since the slave processors do not have a valid stack set up, they must not write to the address pointed to by R13.

- 3) In the case of an EPHEM enabled chip it is necessary to add in the commented code that sets up the VIC on the Test Chip Interrupts. Leaving this code in for a normal Test Chip will generate a Data Abort as this memory space is usually unused. This is done by adding in the line `#define EPHEMENABLED` at the top of Mailbox.c. EPHEM is an additional high speed IO interface embedded into the Test Chip for direct connection to user peripherals.

## 8 Clock frequency settings

### 8.1 Default frequencies

The default frequencies are set to generate a stable system with the system bus clock as close to the maximum operating frequency as possible, and the CPU set to a safe working frequency that is a multiple of the bus frequency.

For example the following systems have these default frequencies:-

Test Chip	REFCLK	CPUCLK	HCLKI	HCLKE
ARM926EJ-S UMC 0.18um LF712	20 MHz	170 MHz	85 MHz	28 MHz
ARM1136JZF-S TSMC 0.13um	40 MHz	240 MHz	120 MHz	30 MHz
ARM7TDMI EPSON T0464FA70	20 MHz	20 MHz	-	20 MHz

HCLKI is the bus frequency for any on-chip peripherals.

HCLKE is the external bus frequency used in the Logic Tile.

For benchmarking purposes it is simpler to scale your results from these frequencies, rather than attempt changing frequency as there are a number of factors that must be carefully considered prior to changing frequency.

### 8.2 Changing the startup frequencies

In circumstances where the user needs to change these startup clock frequencies then they should rebuild the FPGA image with their preferred startup values. This method will then mean that the user does not have to worry about running code at startup to deal with changing the clocks after power up.

In order to do this the following method is recommended:-

1. Determine the required frequencies that are desired for CPUCLK, HCLKI and HCLKE. HCLKE must be between 25 and 35 MHz with the standard clock architecture provided (required to meet the timing constraints and DCM requirements of the FPGA design). Refer to the release notes for the maximum CPUCLK settings for your CPU. The HCLKE frequency must be an integer factor of the CPUCLK frequency.

2. Calculate all the PLL settings that need to be applied to the Test Chip to generate the chosen CPUCLK.

$$F_{\text{CPUCLK}} = (F_{\text{REFCLK}} / \text{NR}) \times (\text{NF} / \text{OD})$$

$$\text{NR} = \text{Reference Divider} = (\text{PLLREFDIV}[3:0] + 1)$$

$$\text{NF} = \text{Feedback Divider} = (\text{PLLFBDIV}[7:0] + 1)$$

$$\text{OD} = \text{Output Divider} = (\text{PLLOUTDIV}[2:0] + 1)$$

$$137\text{kHz} < (F_{\text{REFCLK}} / \text{NR}) < 275 \text{ MHz}$$

$$500\text{kHz} < ((F_{\text{REFCLK}} / \text{NR}) \times \text{NF}) < 275 \text{ MHz}$$

3. Change the parameters declared within the Regs file for your Test Chip. These are:-

CM\_PLLREFDIV\_VAL

CM\_PLLFBDIV\_VAL

CM\_PLLOUTDIV\_VAL

4. If a different value of  $F_{\text{REFCLK}}$  is required to get the exact CPU frequency desired then the CM\_OSC values need to be changed accordingly. There is a calculator on the internet at [www.icst.com](http://www.icst.com). Otherwise refer to section 4.3.3 for how to calculate all the required fields. Change the parameters declared within the Regs file for your Test Chip. These are:-

CM\_CPU\_VDW\_VAL

CM\_CPU\_RDW\_VAL

CM\_CPU\_OD\_VAL

5. Calculate the CPUCLK:HCLKI settings.

For ARM9 based systems this can be either 1:1 or 2:1. To set 1:1 set CM\_USERIN\_VAL[5] to 1'b0, and for 2:1 set CM\_USERIN\_VAL[5] to 1'b1. Always set CM\_USERIN\_VAL[3] to 1'b1 to enable HCLKI.

For ARM11 based systems this can be any integer value from 1:1 to 64:1. Set the ratio value -1 to CM\_HCLKI\_VAL (ie tie to 6'b000000 for 1:1 or 6'b000001 for 2:1).

6. Calculate the HCLKI:HCLKE settings.

For ARM9 based systems this can be any integer value from 1:1 to 16:1.

Ratio = {USERIN[1:0], HCLKDIV[2:0]} + 1

For ARM11 based systems this can be any integer value from 1:1 to 64:1. Set the ratio value -1 to CM\_HCLKE\_VAL (ie tie to 6'b000000 for 1:1 or 6'b000001 for 2:1). This ratio is relative to the CPUCLK rather than HCLKI, but HCLKE must be equal to or slower than HCLKI.

Change the following parameters following these calculations:-

CM\_USERIN\_VAL[1:0]

CM\_HCLKDIV\_VAL

CM\_HCLKE\_VAL

7. Remember to change the CM\_BUILD and CM\_MANUFACTURER parameters to your own unique identification numbers so that you can tell this FPGA build apart from other builds. Then rebuild the FPGA with your new Regs file, and program into the Flash.

### 8.3 System bus frequency testing

It is sometimes desirable to temporarily change the frequency rather than doing a permanent FPGA build. For example if the user wants to test a frequency prior to building a new FPGA image.

In most cases the user will only care about the system bus frequency. In order to accommodate this CPU frequencies have been chosen that can be moved a little faster or slower. Therefore this change can be made to a live system by slowly changing the REFCLK frequency only.

In order to do this, do the following:-

1. Unlock the register bank by writing 0xA05F to 0x10400014.
2. Read register 0x10400008.
3. If decreasing frequency subtract 1 from the value in the last 9 bits. If increasing frequency add 1 to the value in the last 9 bits. Set bit 26 to 1'b1 (force update bit).
4. Write this value back to 0x10400008.
5. Calculate your new REFCLK setting (see section 4.3.3). This can then be used to calculate your new CPUCLK, HCLKI, and HCLKE frequencies (they all scale linearly with REFCLK).
6. Repeat steps 2-5 until you reach the desired operating range. Allow time for the clocks to settle to their new frequency between each loop.

### 8.4 Full frequency testing

If the user wishes to change the CPU frequency by a large amount, or wishes to change the ratios of the different frequencies then it is necessary to take some additional steps in order to allow the PLL/DCMs in the system to correctly lock to their new frequencies.

Since the ARM11 class Test Chips only read in the HCLKI and HCLKE values as part of a Power On Reset, it is not possible to change these values through software.

In order to do this, follow all the calculations in 8.2. However rather than changing the listed parameters above, do the following:-

1. Unlock the registers bank by writing 0xA05F to 0x10400008.
2. Read the value in CM\_AUXOSC register (0x1040001C).
3. Modify bits [31:28] to the new PLLOUTDIV value.
4. Modify bits [27:24] to the new PLLREFDIV value.
5. Write this value back to CM\_AUXOSC.
6. Read the value in CM\_INIT register (0x10400024).
7. Modify bits [29:24] to the new USERIN value.
8. Modify bits [15:8] to the new PLLFBDIV value.
9. Modify bits [6:4] to the new HCLKDIV value.
10. Write this value back to CM\_INIT.
11. Read the value in CM\_OSC (0x10400008).
12. Modify bits [18:16] to the new OD value.
13. Modify bits [15:9] to the new RDW value.
14. Modify bits [8:0] to the new VDW value.
15. Write this value back to CM\_OSC.
16. Push the RESET push button on the PB926EJ-S baseboard. This will now reset the system, with the Core Tile restarting in its new configuration. If connected with a debugger, it is probable that a breakpoint will have been left at 0x0, and so it will be necessary to use JTAG to instruct the processor to proceed with execution of its boot code.



## 9 Variations

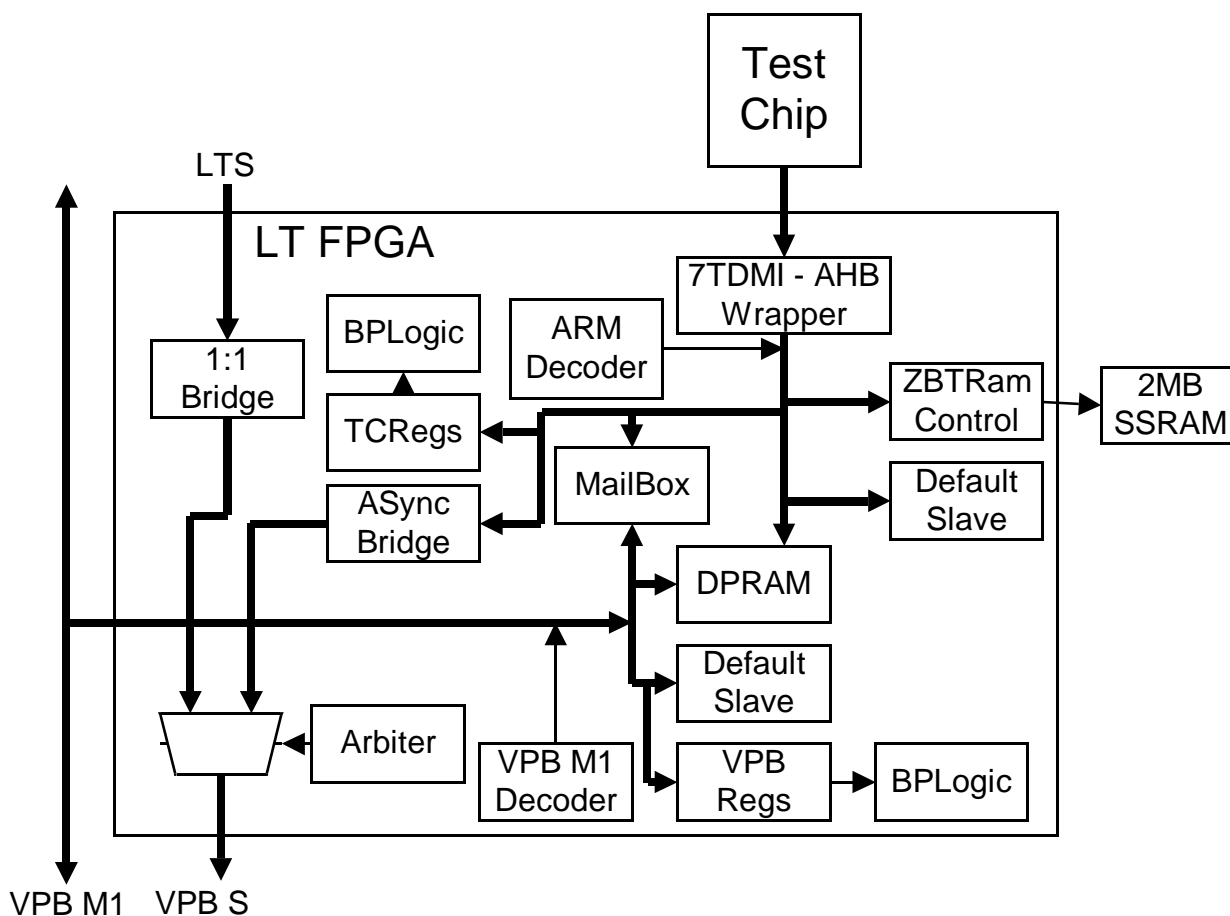
## 9.1 Overview

This Application Note is primarily aimed at using an AHB Test Chip. When used in conjunction with a different bus type, there are a number of differences to the design which the user should be aware of.

## 9.2 CT7TDMI

### 9.2.1 Architecture

In order to support a non-AHB core with an otherwise AHB system, a wrapper has been inserted between the Test Chip and the rest of the system.



The wrapper used in the RTL provided is taken from the ADK (AMBA Development Kit), which is available from ARM. The clock architecture is slightly modified to allow for the fact that the ARM7 Test Chip is not inside the same chip as the bridge. Instead of gating the local copy of the AHB clock to use as the 7TDMI clock, an early version is gated to compensate for the delay from the FPGA to the Test Chip.

### 9.2.2 Software Interface

The memory map for the 7TDMI system is the same as for a native AHB Test Chip. The only difference is in the timing of the 7TDMI, due to the insertion of an extra bridge on the Test Chip bus. The Platform Baseboard busses are unaffected.

### 9.2.3 GTC Regs

There are minor changes to the contents of some of the registers. These changes reflect that the system is different, and that the 7TDMI Test Chips are fitted to a PCB with different configuration options.

- **CT\_ID** : The architecture field reports the processor bus as 7TDMI.
- **CT\_PROC** : Since the 7TDMI does not feature a co-processor with identification information, this register contains information on the type of processor fitted.
- **CT\_STAT** : Since this is a different architecture chip, the Silicon ID values overlap with the native AHB chips.
- **CT\_AUXOSC** : There is no PLL fitted to 7TDMI test chips. The PLL control values have no effect.
- **CT\_INIT** : As with CT\_AUXOSC the PLL control values have no effect. The USERIN values also have no effect.
- **CT\_VOLTAGE** : The 7TDMI test chip has fewer voltage domains than the native AHB test chips. For this reason CT\_VOLTAGE\_CTL4,5,6,7 are not used. The other CT\_VOLTAGE\_CTL registers have the following effect:

<b>CT_VOLTAGE_CTL0</b>	<b>Voltage Control 0</b>
7:0	Core voltage A DAC value
19:8	VDDCOREA voltage ADC value (12 bits)
31:20	VDDCOREA DIFF
<b>CT_VOLTAGE_CTL1</b>	<b>Voltage Control 1</b>
7:0	Core voltage B DAC value
19:8	VDDCOREB voltage ADC value (12 bits)
31:20	VDDCOREB DIFF
<b>CT_VOLTAGE_CTL2</b>	<b>Voltage Control 2</b>
7:0	Reserved
19:8	PISMO 1.8V supply voltage ADC value (12 bits). *Not scaled by 2!
31:20	VDDIO voltage ADC value (12 bits)
<b>CT_VOLTAGE_CTL3</b>	<b>Voltage Control 3</b>
7:0	Reserved
19:8	TP2 voltage ADC value (12 bits)
31:20	TP1 voltage ADC value (12 bits)

- **CT\_PLD\_CTRL** : The CT-7TDMI does not sense whether all the core power nets are good (PGOOD). The Clock options are slightly different.

4	CLKSEL0: Set low for tile below/high for tile above
6:5	CLKSEL[2:1]: HCLK gets CLK_NEG, X_MCLK, GND, CLK_GLOBAL
7	CLKSEL3: NEG_UP_OUT assigned HCLK,NEG_UP_IN
8	CLKSEL4: NEG_DN_OUT assigned HCLK,NEG_DN_IN
9	CLKSEL5: Route X_MCLK and X_ECLK across Core Tile

### 9.2.4 Example Software

The example software has been coded to support both native AHB test chips and 7TDMI class test chips. By reading the processor bus type where applicable the software can operate across different systems. Since all the examples are compiled for the lowest common code architecture (v4 is the lowest expected with this hardware), the code will run on all test chips.