Application Note 151

Example AXI design for a Logic Tile on top of AXI Versatile base boards

Document number: ARM DAI 0151G
Issued: June 2008
Copyright ARM Limited 2008

ARM

Application Note 151
Example AXI design for a Logic Tile on top of AXI Versatile base boards

Copyright © 2008 ARM Limited. All rights reserved.

Release information

The following changes have been made to this Application Note.

Change history

Date	Issue	Change
July 21, 2005	A	First release
December 1, 2005	B	Getting started section added
April 20, 2006	C	Corrected table 4-4.1 to show both HRDX and HRDY signals, format updates
June 27, 2007	D	Supports all AXI Versatile base boards
July 24, 2007	E	Added support for LT-XC4VLX100+ Virtex 4 logic tiles
September 10, 2007	F	Added support for LT-XC5VLX330 Virtex 5 logic tile, details for PB11MPCore
Feburary 29, 2008	G	Updated for new RTL structure, added details for PB-A8

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 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

Copyright © 2008 ARM Limited. All rights reserved.

1 Introduction

1.1 Purpose of this application note

This application note discusses the operation of the example AXI Logic Tile (LT) with an AXI Versatile baseboard. It will examine the contents of the Logic Tile FPGA, the system interconnect, the clock structure, and specifics of the LT programmer's model directly relevant to Logic Tile operation.

On reading this Application Note the user should be in a position to make changes to the provided Logic Tile FPGA design, connect their own AHB or AXI based masters and slaves, or debug and analyze the operation of the provided image.

1.2 AXI Versatile baseboard and LT overview

Table 1.1 References the AXI Versatile baseboards at time of publishing.

AXI Versatile base boards
Emulation Baseboard (EB)
PB1176JZF-S
PB11MPCore
PB-A8

Table 1.1 AXI Versatile base boards

This application note is designed to work on an AXI Versatile baseboard, for example EB with a CT11MPCore Tile fitted in Tile Site 1 and Logic Tile fitted in Tile Site 2, as shown in Figure 1 Example AXI LT on an AXI Versatile baseboard. This LT example is clocked asynchronously to the EB. It is synthesized for up to 30MHz operation. It will also work on PB1176JZFS, PB11MPCore and PB-A8.

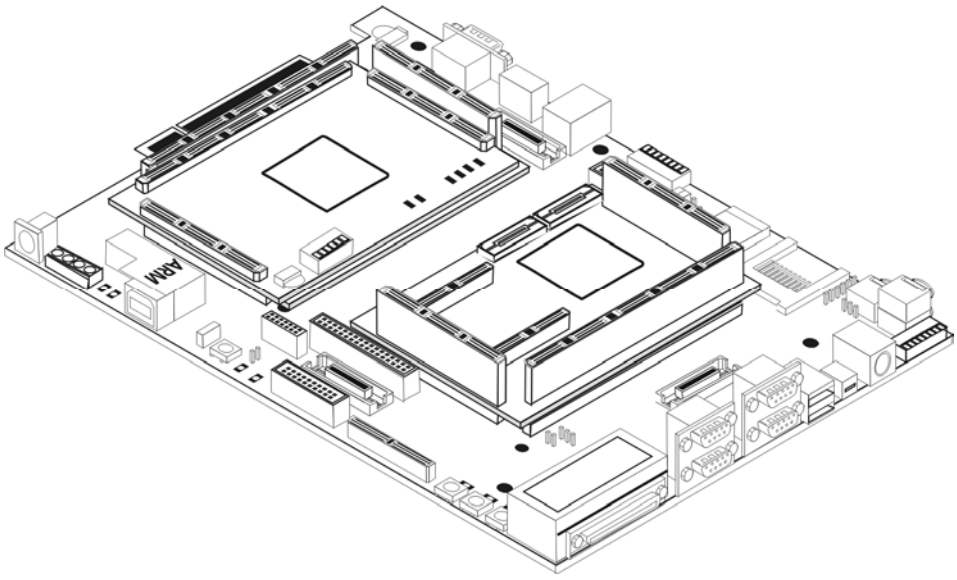


Figure 1 Example AXI LT on an AXI Versatile baseboard

Copyright © 2008 ARM Limited. All rights reserved.

2 Getting Started

Before you can use this application note with EB, you will need to program the Baseboard with the required FPGA image to enable the Core Tile fitted in TILE SLOT 1 to function correctly. Please refer to the relevant application note for details on how to do this (for example AN152). 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 TILE SITE 2 of the Emulation Baseboard or TILE SITE 1 on PB1176JZF-S, PB11MPCore or PB-A8.
2. Slide the CONFIG switch (S1) to the ON position.
3. Connect RVI or Multi-ICE to the Emulation Baseboard JTAG ICE connector (J18), or a USB cable to the USB Debug Port.
4. For EB and PB1176JZF-S check the external supply voltage is +12V (positive on center pin, +/-10%, 35W), and connect it to the power connector (J28).
5. Power-up the boards. The '3V3 OK' and '5V OK' LEDs on the Baseboard 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 EB \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_usb.exe for your USB connection
    - progcards_rvi.exe for your RealView ICE connection
      When using RVI select the target RVI box you are using.

10. Select the option for the Logic 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. Slide the CONFIG switch to the OFF position, and power-up the boards. Ensure the Logic Tile 'FPGA_OK' and Emulation Baseboard 'GLOBAL_DONE' 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.

Copyright © 2008 ARM Limited. All rights reserved.

3 System architecture

This system is an AXI (AMBA 3.0) based system. This LT image exposes one master port and one slave port (both muxed 64 bit AXI) to the base board.

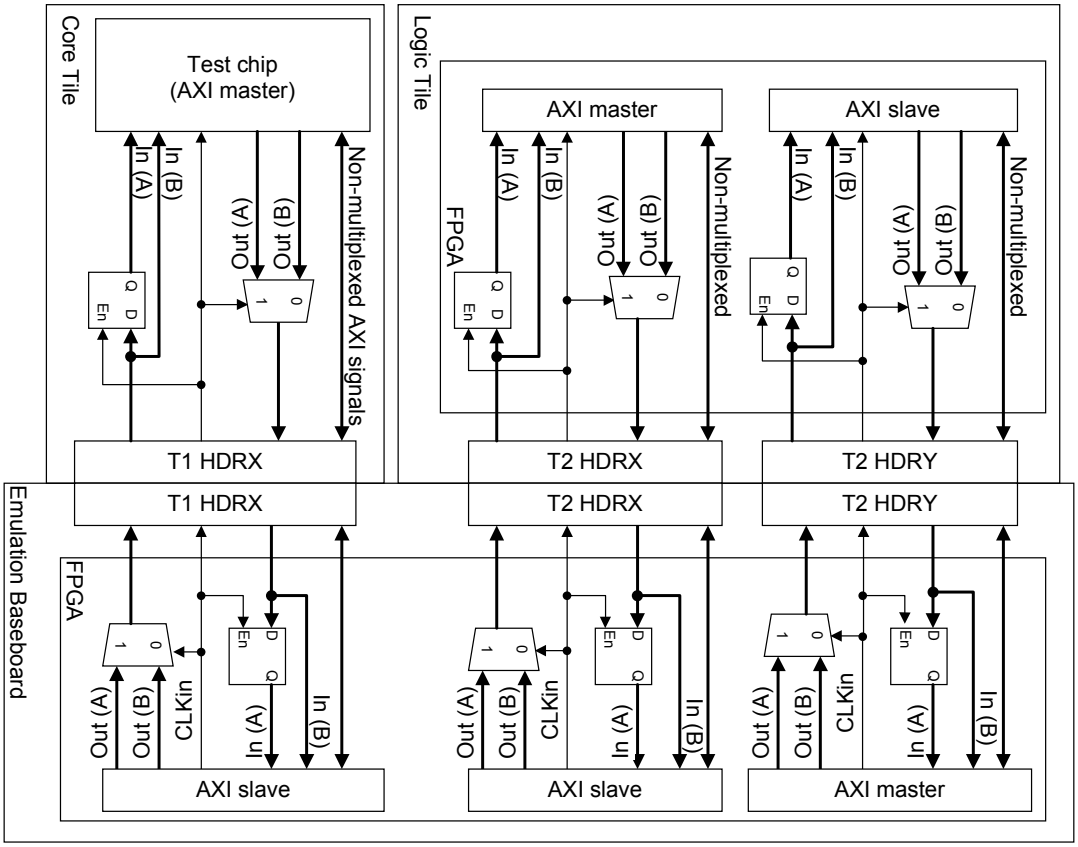


Figure 2 Block level architecture

Note that the direction of the arrows indicates the direction of control, i.e. it points from the Master to the Slave. An AXI bus contains signals going in both directions.

Copyright © 2008 ARM Limited. All rights reserved.

3.1 LT architecture

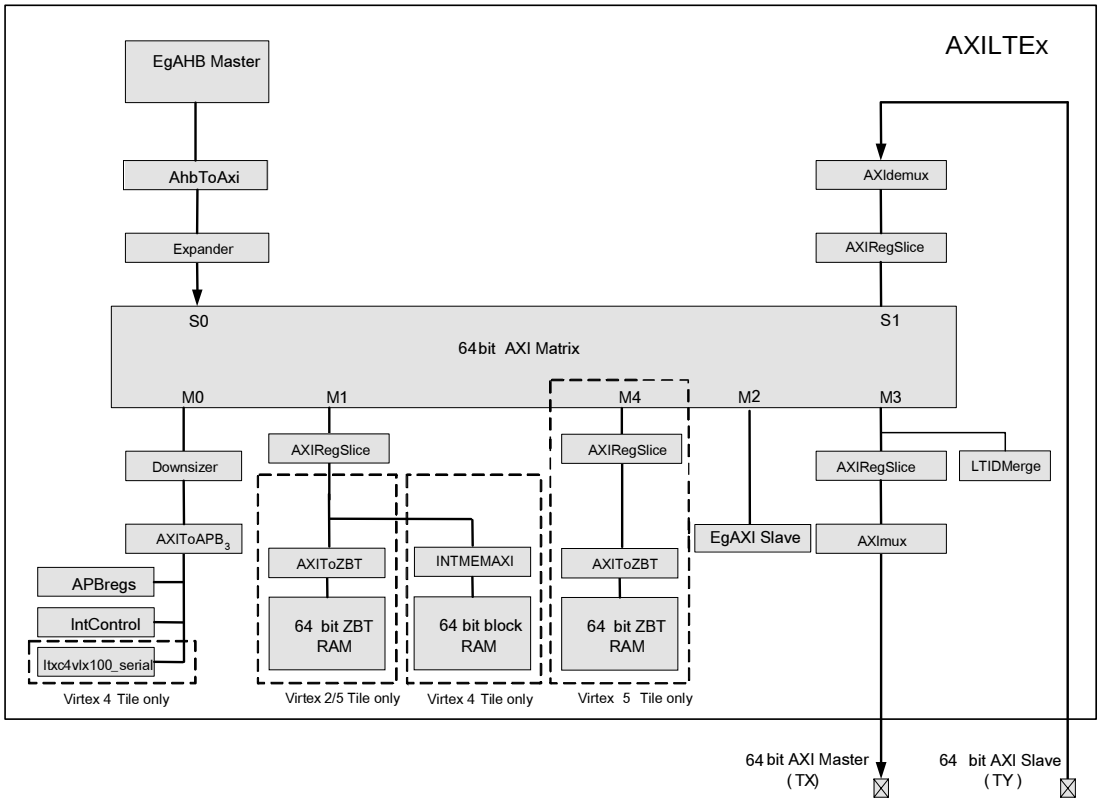


Figure 3 AXI Logic Tile Example

3.2 Module functionality

The function of each of these blocks is as follows:

EgAHBMaster

This is a simple example 32 bit AHB Master, which can perform a single AHB transfer. In the case of a write, it accepts a destination address and 32 bit data value from the register block. In the case of a read, it accepts a read address and returns a 32 bit data value into a register in the register block.

AHBToAXI

This is a 32 bit bridge which converts an AHBLite transfer into 32 bit AXI transfer (AMBA 3.0).

Expander

This is an AXI Expander that converts 32bit AXI transfers into 64 bit transfers.

AXIdemux

This is an AXI demux block, which demultiplexes the muxed 64 bit AXI bus on TY. Refer to section 4.3 on the mux/demux strategy.

AXIRegSlice

This block provides timing isolation between master and slave interfaces on an AXI interconnect.

AXI Matrix

Copyright © 2008 ARM Limited. All rights reserved.

This 64 bit AXI matrix provides the bulk of the interconnect structure. It allows either of the 2 slave ports to connect to any of the 4 slave ports (5 slave ports on a Virtex5 design) without blocking the other master (unless they both try to access the same slave). It also contains the decoder mapping to determine the address map, and a scheme to determine priority of competing masters to a single slave.

Downsizer

This is an AXI Downsizer which converts 64bit AXI transfers into 32 bit transfers.

AXIToAPB$_3$

This is a bridge that converts 32 bit AXI transactions into APB$_3$ transfers.

APBRegs

This is the APB register block, which allows the user to program the system registers.

IntControl

This block allows the user to generate an interrupt to the base board.

AXIToZBT (Virtex2 and 5 only)

This is a 64 bit bridge which converts AXI transfers into ZBT SRAM transfers.

INTMEMAXI (Virtex4 only)

This is a 64 bit bridge which converts AXI transfers into Xilinx BRAM transfers.

LTXC4VLX100_serial (Virtex4 only)

This is a serial interface to the LT PLD for control of LEDs, Switches and clocks.

EgAXISlave

This is a 64bit example AXI slave, which contains 16 64bit registers.

LTIDMerge

This block merges the ID field on the AXI bus to reduce the width of each ID channel. See section 4.7

AXImux

This is an AXI mux block, which multiplexes the 64 bit AXI bus M3 from the AXI Matrix onto TX. Refer to section 4.3 on the mux/demux strategy.

Copyright © 2008 ARM Limited. All rights reserved.

3.3 Clock architecture

The clock architecture is carefully designed so as to minimize the skew (difference) in the clock edge position between different components across the system. The User Guides for all the boards used in this design explain the clock options they support. Please refer to the baseboard user guide for baseboard clock structure, the EB system is shown below as an example.

The maximum frequency the example Logic Tile design can work at is dependant on the tile used, typically this is in the 30MHz range.

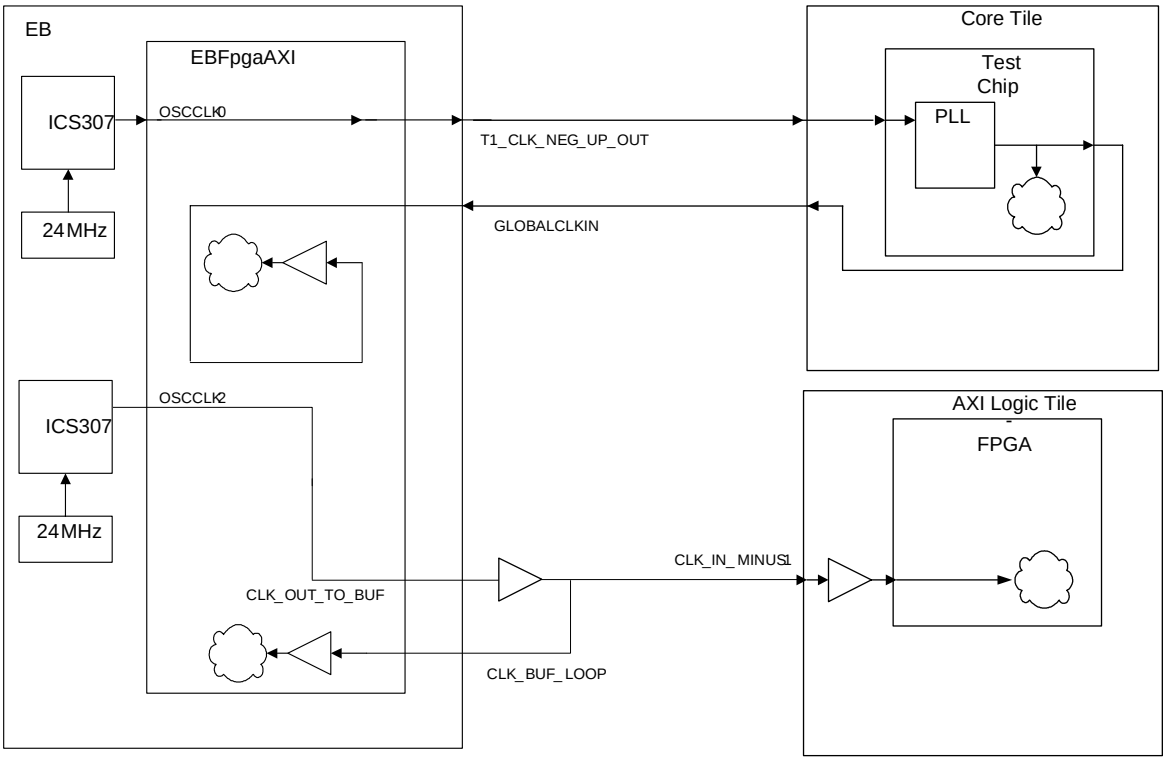


Figure 4 Clock architecture for EB and LT

Copyright © 2008 ARM Limited. All rights reserved.

3.4 Interrupt architecture

The interrupt scheme makes use of a simple Interrupt controller to facilitate the connection of LTINT to the baseboard. The same system is used on PB1176JZF-S, PB11MPCore and PB-A8.

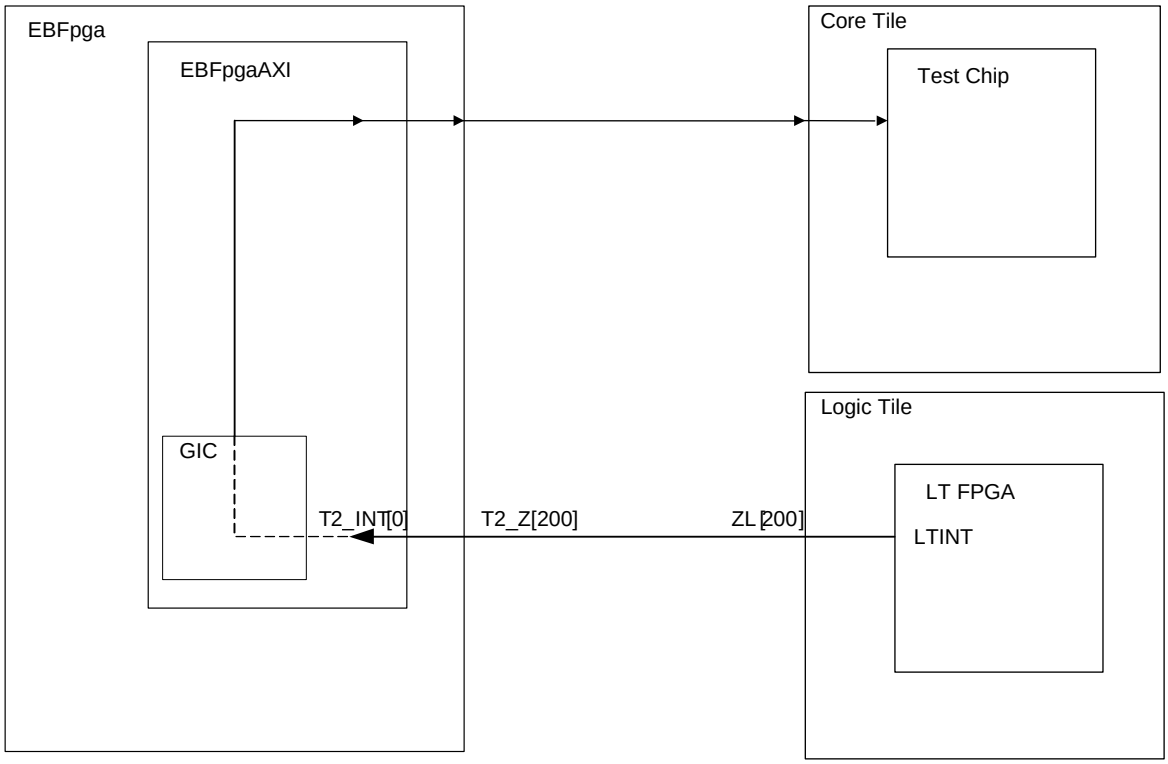


Figure 5 Interrupt architecture

The interrupt controller in the EB FPGA is made up of four Generic Interrupt Controllers (GIC), which can be configured to generate an IRQ or FIQ on the ARM test chip. The GIC has a large number of interrupt inputs, you can have up to 8 interrupt sources from the LT without having to cascade interrupt controllers. The example design in this application note only uses one of these 8 interrupt signals.

For more details about the GIC refer to the EB user guide and the appropriate application note, for example AN152 for the ARM11MPCore Core Tile.

3.5 Reset architecture

The signal nSYSRST is used to reset all the peripherals in the Logic Tile. nSYSRST is driven up from the baseboard.

Copyright © 2008 ARM Limited. All rights reserved.

4 Hardware description

4.1 Top level (AXILTEx)

The top level of the design is of particular importance for a number of reasons. This level defines the mapping from the HDRX, HDRY and HDRZ busses from the tile site into their functional allocations.

4.2 AXI subsystem (AXITopLevel)

This level connects all the components together and ties off static pins. This includes all the major blocks as shown in Figure 2 Block level architecture. The top level RTL is provided so blocks can be added and removed. AXI blocks are provided as .NGO netlists.

4.3 AXI multiplexing scheme

By using a 2:1 multiplexer and latch scheme as shown below it is possible to reduce the pin count for the AXI buses into a realistic size for implementation on the Tile XL and YL headers.

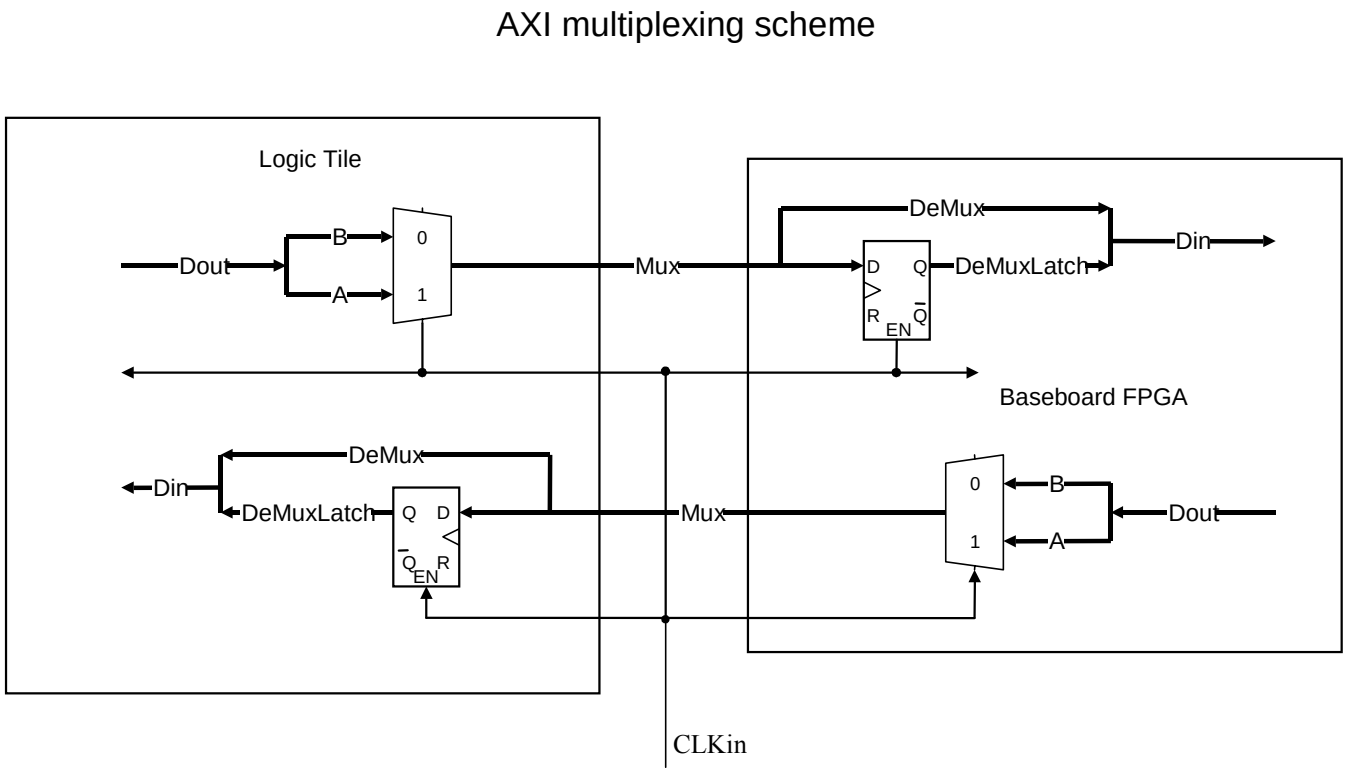


Figure 6 AXI multiplexing scheme

The output data is multiplexed on the level of CLKin, to generate the multiplexed bus (Mux). The de-multiplexing is performed by latching the data (A) generated on the high level of CLKin when CLKin goes low (DeMuxLatch). The data (B), generated on the low side of CLKin is passed straight through (DeMux). This design assumes data is always generated and captured on the rising edge of CLKin.

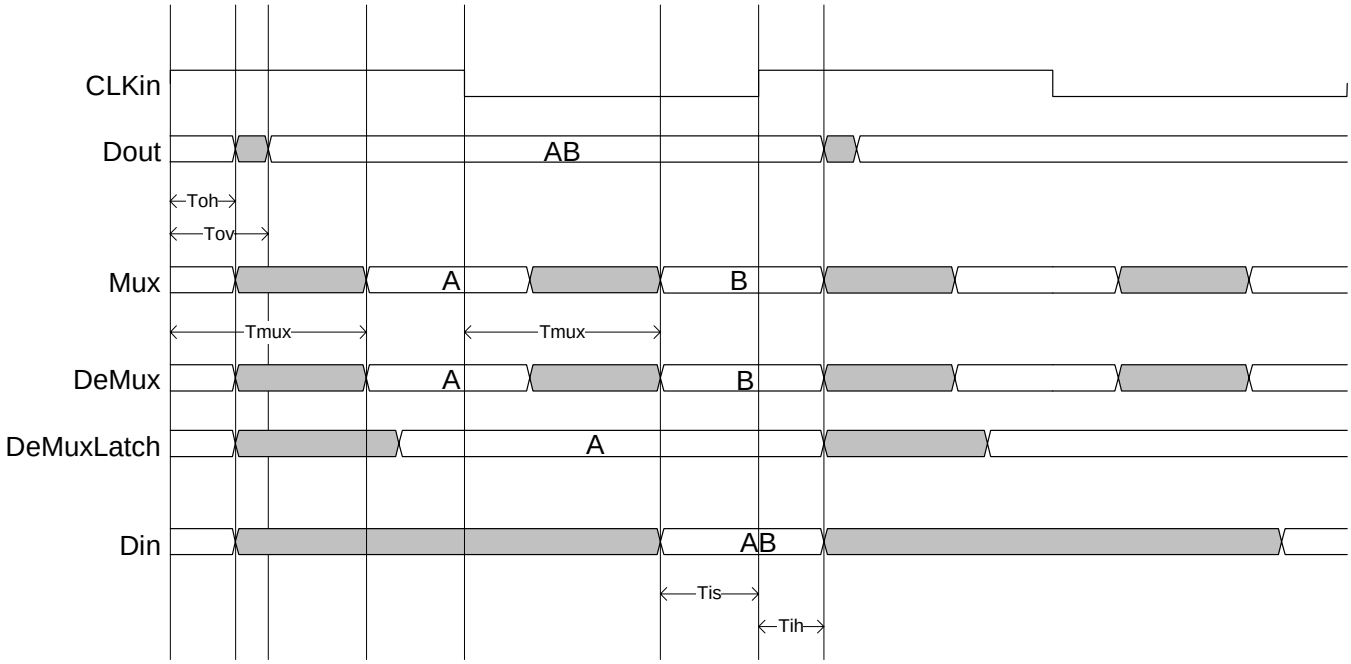
The **Valid** and **Ready** signals on AXI can not be multiplexed in this way due to their timing requirements and must be passed directly between devices.

The Logic Tile supplies two multiplexed AXI buses (TX and TY).

Copyright © 2008 ARM Limited. All rights reserved.

The following timing diagram shows the data flow through the design with expected delays from standard components.

AXI timing requirements



Timing requirements;

Toh	min = 0ns	(output hold)
Tov	max = 2ns	(output valid)
Tis	max = 2ns	(Input setup)
Tih	max = 0ns	(input hold)
Tmux	max = 6ns	(multiplexer and board delay)

The CLKin is the clock driven into the MPCore from the board. All I/O timing must be with respect to this clock.

Figure 7 AXI timing requirements

Copyright © 2008 ARM Limited. All rights reserved.

4.4 Header HDRX and HDRY AXI pin allocation

The two AXI buses connect to the HDRX and HDRY Tile headers as shown. AXI port TX connects to header HDRX and AXI port TY connects to header HDRY.

X/Y Bus	HDRX pin	HDRY pin	signal	X/Y Bus	HDRX pin	HDRY pin	signal
0	180	179	WDATA0/32	72	36	35	BID1/3
1	178	177	WDATA1/33	73	34	33	BID4/BID5
2	176	175	WDATA2/34	74	32	31	BRESP0/1
3	174	173	WDATA3/35	75	30	29	BVALID
4	172	171	WDATA4/36	76	28	27	BREADY
5	170	169	WDATA5/37	77	26	25	ARADDR0/16
6	168	167	WDATA6/38	78	24	23	ARADDR1/17
7	166	165	WDATA7/39	79	22	21	ARADDR2/18
8	164	163	WDATA8/40	80	20	19	ARADDR3/19
9	162	161	WDATA9/41	81	18	17	ARADDR4/20
10	160	159	WDATA10/42	82	16	15	ARADDR5/21
11	158	157	WDATA11/43	83	14	13	ARADDR6/22
12	156	155	WDATA12/44	84	12	11	ARADDR7/23
13	154	153	WDATA13/45	85	10	9	ARADDR8/24
14	152	151	WDATA14/46	86	8	7	ARADDR9/25
15	150	149	WDATA15/47	87	6	5	ARADDR10/26
16	148	147	WDATA16/48	88	4	3	ARADDR11/27
17	146	145	WDATA17/49	89	2	1	ARADDR12/28
18	144	143	WDATA18/50	90	1	2	ARADDR13/29
19	142	141	WDATA19/51	91	3	4	ARADDR14/30
20	140	139	WDATA20/52	92	5	6	ARADDR15/31
21	138	137	WDATA21/53	93	7	8	ARID0/2
22	136	135	WDATA22/54	94	9	10	ARID1/3
23	134	133	WDATA23/55	95	11	12	ARLEN0/2
24	132	131	WDATA24/56	96	13	14	ARLEN1/3
25	130	129	WDATA25/57	97	15	16	ARSIZE0/1
26	128	127	WDATA26/58	98	17	18	ARID4/ARPROT2
27	126	125	WDATA27/59	99	19	20	ARPROT0/1
28	124	123	WDATA28/60	100	21	22	ARBURST0/1
29	122	121	WDATA29/61	101	23	24	ARLOCK0/1
30	120	119	WDATA30/62	102	25	26	ARCACHE0/2
31	118	117	WDATA31/63	103	27	28	ARCACHE1/3
32	116	115	WID0/2	104	29	30	ARVALID/ARID5
33	114	113	WID1/3	105	31	32	ARREADY
34	112	111	WSTRB0/4	106	33	34	RDATA0/32
35	110	109	WSTRB1/5	107	35	36	RDATA1/33
36	108	107	WSTRB2/6	108	37	38	RDATA2/34
37	106	105	WSTRB3/7	109	39	40	RDATA3/35
38	104	103	WLAST/WID4	110	41	42	RDATA4/36
39	102	101	WVALID/WID5	111	43	44	RDATA5/37
40	100	99	WREADY	112	45	46	RDATA6/38
41	98	97	AWADDR0/16	113	47	48	RDATA7/39
42	96	95	AWADDR1/17	114	49	50	RDATA8/40
43	94	93	AWADDR2/18	115	51	52	RDATA9/41
44	92	91	AWADDR3/19	116	53	54	RDATA10/42
45	90	89	AWADDR4/20	117	55	56	RDATA11/43
46	88	87	AWADDR5/21	118	57	58	RDATA12/44
47	86	85	AWADDR6/22	119	59	60	RDATA13/45
48	84	83	AWADDR7/23	120	61	62	RDATA14/46
49	82	81	AWADDR8/24	121	63	64	RDATA15/47
50	80	79	AWADDR9/25	122	65	66	RDATA16/48
51	78	77	AWADDR10/26	123	67	68	RDATA17/49

Copyright © 2008 ARM Limited. All rights reserved.

X/Y Bus	HDRX pin	HDRY pin	signal	X/Y Bus	HDRX pin	HDRY pin	signal
52	76	75	AWADDR11/27	124	69	70	RDATA18/50
53	74	73	AWADDR12/28	125	71	72	RDATA19/51
54	72	71	AWADDR13/29	126	73	74	RDATA20/52
55	70	69	AWADDR14/30	127	75	76	RDATA21/53
56	68	67	AWADDR15/31	128	77	78	RDATA22/54
57	66	65	AWID0/2	129	79	80	RDATA23/55
58	64	63	AWID1/3	130	81	82	RDATA24/56
59	62	61	AWLEN0/2	131	83	84	RDATA25/57
60	60	59	AWLEN1/3	132	85	86	RDATA26/58
61	58	57	AWSIZE0/1	133	87	88	RDATA27/59
62	56	55	AWID4/AWPROT2	134	89	90	RDATA28/60
63	54	53	ARM_nRESET	135	91	92	RDATA29/61
64	52	51	AWPROT0/1	136	93	94	RDATA30/62
65	50	49	AWBURST0/1	137	95	96	RDATA31/63
66	48	47	AWLOCK0/1	138	97	98	RID0/2
67	46	45	AWCACHE0/2	139	99	100	RID1/3
68	44	43	AWCACHE1/3	140	101	102	RRESP0/1
69	42	41	AWVALID/AWID5	141	103	104	RLAST/RID4
70	40	39	AWREADY	142	105	106	RVALID/RID5
71	38	37	BID0/2	143	107	108	RREADY

Table 4.1 Header HRDX and HRDY AXI pin allocation

Copyright © 2008 ARM Limited. All rights reserved.

4.5 Header HDRZ pin allocation

The ZL bus is used to connect the Logic Tile interrupt to the baseboard. Only signal ZL[200] and ZL[233:232] are used for this example Logic Tile.

ZL Bus	HDRZ	signal
200	112	LTINT
232	48	BOARDDET0
233	46	BOARDDET1

Table 4.2 Header HDRZ pin allocation

The ZL[233:233] are used to detect which base board the LT is connected to, and change the base address of the peripherals to suit the baseboard.

ZL[233:232]	Baseboard	Peripheral base address
b00	EB	0x80000000
b01	PB1176JZF-S PB11MPCore PB-A8	0xC0000000
b10	*Reserved*	0xC0000000
b11	*Reserved*	0xC0000000

Table 4.3 LT Peripheral base address

4.6 Description of the example master

The example master generates a single word (32 bits) access in the system bus when the logic tile push button is pressed.

The logic tile switch S1 selects the type of transfer:

- If S1-1 is OFF, the enabled master generates a write transfer
- If S1-1 is ON, the enabled master generates a read transfer

The address for the system bus accesses is programmed by writing to logic tile register: LT_HADDR_TRANSFER.

- LT_HADDR_TRANSFER contains the transfer address for master and is located at offset address 0x24
- LT_EGMASTER_WRITE contains the data to be written in the case of a write transfer (offset address 0x14).
- LT_EGMASTER_READ contains the data read after a read transfer (offset address 0x20).

For example, if you want example master to write the data 0x12345678 to address 0x82000000 using an EB, you must follow these steps:

- Set S1-1 to OFF, so the transfer is a write transfer
- Write 0x82000000 to register LT_HADDR_TRANSFER at address 0x80000024
- Write 0x12345678 to register LT_EGMASTER_WRITE at address 0x80000014
- Push the logic tile push button, 0x12345678 should be transferred to address 0x82000000.

If you want example master to read back the data at address 0x82000000,using an EB you must follow these steps:

- Set S1-1 to ON, so the transfer is a read transfer

Copyright © 2008 ARM Limited. All rights reserved.

- Write 0x82000000 to register LT_HADDR_TRANSFER at address 0x80000024
- Push the logic tile push button, 0x12345678 should be transferred from address 0x82000000 to register LT_EGMASTER_READ at address 0x80000020.

By default after reset, HADDR_TRANSFER is programmed to access the LT ZBT SSRAM. Please note this default value changes depending on baseboard.

- HADDR_TRANSFER = 0x82000000 for EB

If the BOARDDET signals (ZL[233:232]) detected a baseboard other an EB, the base address of all the peripherals in the LT would change from 0x80000000 to 0xC0000000.

4.7 LTIDMerge

The purpose of this block is to reduce the ID width from 7 to 6 bits before connecting to the baseboard. Bit 4 is removed as it is always set to b0. Doing this reduces the ID width of each channel by 1 bit.

There is a restriction on the ID value if the example design is to be used with other ARM baseboards. The ID value must be one of the following to allow the example design to work with future ARM baseboards.
The allowed hex values are 0x02, 0x03, 0x06, 0x07,0x10, 0x11, 0x14, 0x15, 0x20, 0x21, 0x24, 0x25, 0x26, 0x27, 0x30, 0x31, 0x32, 0x33, 0x36 & 0x37.

```
//Example master ID value
assign AWIDS0 = 6'h24;
assign WIDS0  = 6'h24;
assign ARIDS0 = 6'h24;
```

4.8 Flash interface

The interface to the configuration flash is not used in this example so the interface is tied off.

```
assign FnOE   = 1'b1;
assign FnWE   = 1'b1;
assign FA0    = 1'b0;
assign FA1    = 1'b0;
assign FA21   = 1'b0;
assign FA22   = 1'b0;
assign FnBYTE = 1'b1;
assign FnCE   = 1'b1;
```

4.9 TAG route through

It is possible to add TAP controllers into the debug scan chain inside this FPGA. This design does not add a TAP controller, so the JTAG signals are routed through the design.

```
assign D_TDO  = D_TDI;
assign D_RTCK = D_TCK;
```


Copyright © 2008 ARM Limited. All rights reserved.

5 Programmer's model

The LT on EB example design provides memory mapped registers and ZBT SSRAM. Note that all masters in the system can access the slaves on the LT. The LT example master is also able to access slaves on the base board.

5.1 LT Memory map on EB

	Memory range		**Bus type**	**Memory region size**
Peripheral	**Lower limit**	**Upper limit**		
EB	0x00000000	0x7FFFFFFF	AXI	2GB
APB Peripherals on LT	0x80000000	0x81FFFFFF	APB	2MB
ZBT SSRAM (Virtex 2 LT only)	0x82000000	0x823FFFFF	AHB	2MB
Block RAM (Virtex 4 LT only)	0x82000000	0x8207FFFF	AHB	512KB
ZBT SSRAM (Virtex 5 LT only)	0x82000000	0x83FFFFFF	AHB	32MB
Example AXI slave on LT	0x84000000	0x84000FFF	AXI	4KB
Reserved	*0x84001000*	*0xFFFFFFFF*	*AXI*	*~1995MB*

Table 5.1 Memory map

If the BOARDDET signals (ZL[233:232]) detected a baseboard other an EB, the base address of all the peripherals in the LT would change from 0x80000000 to 0xC0000000.

5.2 LT APB peripherals

	Memory range		**Bus type**	**Memory region size**
Peripheral	**Lower limit**	**Upper limit**		
System Registers	0x80000000	0x80000FFF	APB	4K
System Interrupt Controller	0x80001000	0x80001FFF	APB	4K
Reserved PSEL2	*0x80002000*	*0x80002FFF*	*APB*	*4K*
Reserved PSEL3	*0x80003000*	*0x80003FFF*	*APB*	*4K*
Reserved PSEL4	*0x80004000*	*0x80004FFF*	*APB*	*4K*
Reserved PSEL5	*0x80005000*	*0x80005FFF*	*APB*	*4K*
Reserved PSEL6	*0x80006000*	*0x80006FFF*	*APB*	*4K*
Reserved PSEL7	*0x80007000*	*0x80007FFF*	*APB*	*4K*
Reserved PSEL8	*0x80008000*	*0x80008FFF*	*APB*	*4K*
Reserved PSEL9	*0x80009000*	*0x80009FFF*	*APB*	*4K*
Reserved PSEL10	*0x8000A000*	*0x8000AFFF*	*APB*	*4K*
Reserved PSEL11	*0x8000B000*	*0x8000BFFF*	*APB*	*4K*
Reserved PSEL12	*0x8000C000*	*0x8000CFFF*	*APB*	*4K*
Reserved PSEL13	*0x8000D000*	*0x8000DFFF*	*APB*	*4K*
Reserved PSEL14	*0x8000E000*	*0x8000EFFF*	*APB*	*4K*
Reserved PSEL15	*0x8000F000*	*0x8000FFFF*	*APB*	*4K*
Reserved	*0x80010000*	*0x81FFFFFF*	*APB*	*1936K*

Table 5.2 LT APB peripherals

If the BOARDDET signals (ZL[233:232]) detected a baseboard other an EB, the base address of all the peripherals in the LT would change from 0x80000000 to 0xC0000000. The offsets and size remain the same.

Copyright © 2008 ARM Limited. All rights reserved.

5.3 System registers

Table 4-3.1 shows the location of the system registers in the example design. The addresses shown are offsets from the logic tile base address.

Offset address	Name	Reset value	Type	Size	Function
0x000	LT_OSC0	0x30406	Read/write	19	Oscillator 0 divisor register
0x004	LT_OSC1	0x30406	Read/write	19	Oscillator 1 divisor register
0x008	LT_OSC2	0x30406	Read/write	19	Oscillator 2 divisor register
0x00C	LT_LOCK	0x0000	Read/write	17	Oscillator lock register
0x010	LT_LEDS	0xF/0xFF	Read/write	4/8[1]	User LEDs control register (LT)
0x014	LT_EGMASTER_WRITE	0x00000000	Read/write	32	Data for next write transfer
0x018	LT_INT	b0	Read/write	1	Push button interrupt register
0x01C	LT_SW	bxxxx/bxxxxxxxx	Read	4/8[1]	Switches register (LT S1)
0x020	LT_EGMASTER_READ	0x00000000	Read	32	Data after last read transfer
0x024	LT_HADDR_TRANSFER	0x82000000	Read/write	32	Address for next transfer from example master

Table 5.3 System registers

If the BOARDDET signals (ZL[233:232]) detected a baseboard other an EB, the default LT_HADDR_TRANSFER would change from 0x82000000 to 0xC2000000.

[1] 8 LT_LEDS and 8 LT_SWITCHES for LTXC4VLX100+ and LT-XC5VLX330 designs.

Copyright © 2008 ARM Limited. All rights reserved.

5.3.1 Oscillator divisor registers

The oscillator registers LT_OSC0, LT_OSC1 and LT-OSC2 control the frequency of the clocks generated by the three clock generators on the logic tile.

Before writing to the oscillator registers, you must unlock them by writing the value 0x0000A05F to the LT_LOCK register. After writing the oscillator register, relock it by writing any other value to the LT_LOCK register.

Bits	Name	Access	Function	Default
[18:16]	OD	Read/write	Output divider: b000 = divide by 10 b001 = divide by 2 b010 = divide by 8 b011 = divide by 4 b100 = divide by 5 b101 = divide by 7 b110 = divide by 3 b111 = divide by 6	b000
[15:9]	RDW	Read/write	VCO divider word. Defines the binary value of the RV[6:0] pins of the clock generator	b0010110
[8:0]	VDW	Read/write	Reference divider word. Defines the binary value of the V[7:0] pins of the clock generator	b001110000

Table 5.4 LT_OSCx bit pattern

The reset value of these registers sets the oscillators to 24MHz. More information about setting up the frequency of the logic tile oscillators is available in the logic tile User Guide.

5.3.2 Oscillator lock register

The lock register LT_LOCK (at offset 0x0C) controls access to the oscillator registers and allows you to lock them and unlock them. This mechanism prevents the oscillator registers from being overwritten accidentally.

Bits	Name	Access	Function
[16]	LOCKED	Read	This bit indicates if the oscillator registers are locked or unlocked: b0 = unlocked b1 = locked
[15:0]	LOCKVAL	Read/write	Write the value 0xA05F to this field to enable write accesses to the oscillator registers. Write any other value to lock the oscillator registers.

Table 5.5 LT_LOCK bit pattern

Copyright © 2008 ARM Limited. All rights reserved.

5.3.3 User LEDs control registers

The LT_LED register (at offset 0x10) controls the 4/8 user LEDs on the logic tile.

Writing the value b1111/b11111111 will light all 4/8 LED's. LED's can be lit individually for example writing b00000011 will light only the LED0 and LED1.

5.3.4 Push button interrupt register

The push button interrupt register LT_INT (at offset 0x18) contains 1 bit. It is a latched indication that the push button has been pressed. The contents of this register are fed to the interrupt controller registers.

Bits	Name	Access	Function
[0]	LT_INT	Read	If the push button has been pressed, this bit is set.
		Write	Write b0 to this register to clear the latched push button indication.
			Writing b1 to this register has the same effect as pressing the push button.

Table 5.6 LT_INT bit pattern

5.3.5 Switch registers

Use the LT_SW register (at offset 0x1C) to read the setting of the 4/8-way DIP switch on the logic tile.

5.3.6 Registers used by example masters

LT_HADDR_TRANSFER contains the transfer address for the example master and is located at offset address 0x24

LT_EGMASTER_WRITE contains the data that will be written in the next write transfers and is located at offset address 0x14.

LT_EGMASTER_READ contains the data read in the last read transfers and is located at offset address 0x20.

5.4 Interrupt controller

Table 4.4 shows the location of the interrupt controller memory mapped registers.

Offset address	Name	Type	Size (bits)	Function
0x1000000	LT_ISTAT	Read	8	Interrupt status
0x1000004	LT_IRSTAT	Read	8	Interrupt raw status
0x1000008	LT_IENABLE	Read	8	Interrupt enable
0x1000008	LT_IENSET	Write	8	Interrupt enable set
0x100000C	LT_IENCLR	Write	8	Interrupt enable clear
0x1000010	LT_SOFTINT	Read/write	4	software interrupt

Table 5.7 Location of interrupt controller registers

Copyright © 2008 ARM Limited. All rights reserved.

The interrupt controller included in the example design generates an interrupt signal from a number of interrupt sources. The output of the interrupt controller is routed to the base board:

The logic tile interrupt controller is designed so that it can accept up to four sources of interrupts and four software interrupts. In the example design only one source of interrupt is used, which is connected to the logic tile push button.

The other three sources or interrupts are unused in the example design, but users can use them to connect interrupt request signals for the peripherals they implement in the logic tile.

The interrupt controller contains registers to enable, disable and monitor the status of the different interrupt sources.

Table 4.5 shows the interrupt number assigned to each source of interrupt. Each interrupt source is associated with a bit number in the interrupt controller registers.

Bit	Name	Function
[7:5]	-	Spare (not used by the example image)
[4]	PBINT	Push button interrupt
[3:0]	SOFTINT[3:0]	Software interrupt generated by writing to LT_SOFTINT

Table 5.8 Interrupt register bit assignment

For example, in order to enable the push button interrupt you must set bit 4 of the interrupt enable register.

The way that the interrupt enable, clear, status and raw status registers work is illustrated in Figure 8 Interrupt controller internal design. This figure shows the control logic for one interrupt source, corresponding to one bit of all these registers.

Copyright © 2008 ARM Limited. All rights reserved.

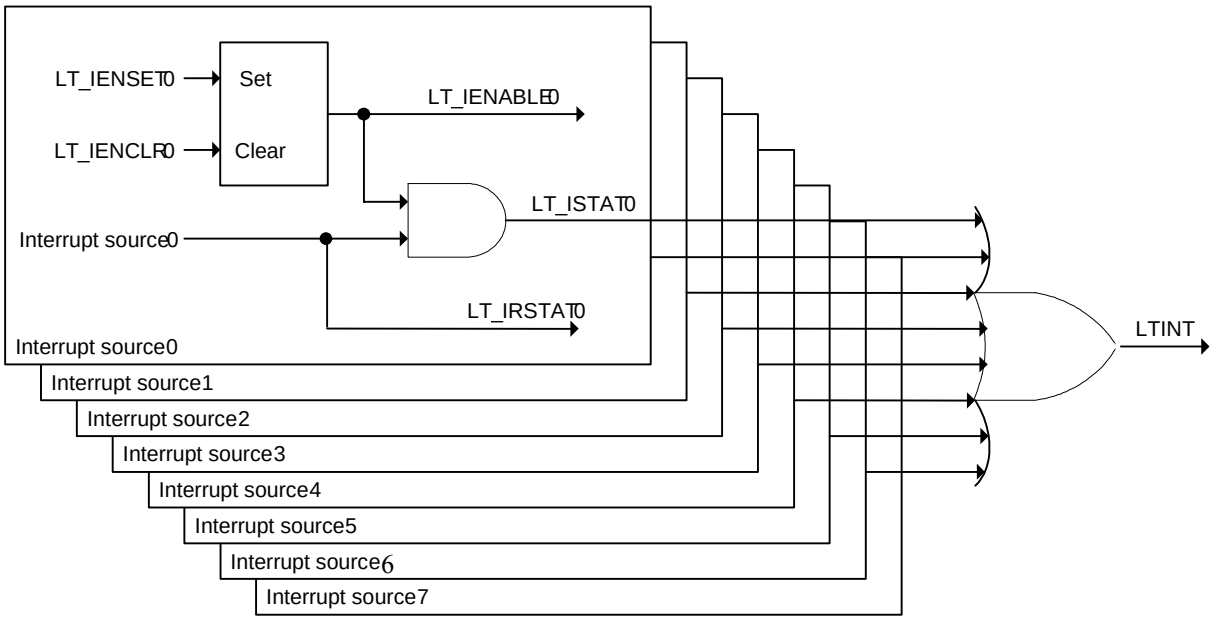


Figure 8 Interrupt controller internal design

nINT in the figure is the interrupt request output from the logic tile interrupt controller. This signal is activated when any bit of LT_ISTAT is set. This signal is passed to the base board as T2_INT[0].

5.4.1 Interrupt status register

The status register LT_ISTAT, contains the logical AND of the bits in the raw status register and the enable register.

Therefore a bit of the status register is 1 when its corresponding interrupt source is active and its corresponding interrupt enable bit has been set.

5.4.2 Interrupt raw status register

The raw status register LT_IRSTAT indicates the signal levels on the interrupt request inputs. A bit set to 1 indicates that the corresponding interrupt request is active.

5.4.3 Interrupt enable, interrupt enable set and interrupt enable clear

Reading from the interrupt enable register LT_IENABLE returns the current state of the interrupt source enable bits.

Writing 1 to a bit of the interrupt enable set register LT_IENSET sets the corresponding bit of LT_IENABLE

Writing 1 to a bit of the interrupt enable clear register LT_IENCLR clears the corresponding bit of LT_IENABLE

Writing 0 to a bit of LT_IENSET or LT_IENCLR leaves the corresponding bit of LT_IENABLE unchanged

LT_IENABLE and LT_IENSET share the same address in the memory map.

Copyright © 2008 ARM Limited. All rights reserved.

5.4.4 Software interrupt register

This register is used to generate interrupts by software.

Writing 1 to any bit position in LT_SOFTINT register sets the corresponding bit in the interrupt controller registers. The LT_SOFTINT register has four bits, corresponding to the four software interrupts.

Writing a 0 to this register clears any software interrupts.

Reading from this register shows the raw status of the software interrupts.

The software interrupts should not be confused with the ARM SWI instruction.

5.5 64bit ZBT SSRAM (LT-XC2V4000+ only)

The SSRAM is configured as 64 bits wide using two 32 bit chips, it occupies the address space from 0x82000000 to 0x823FFFFF on an EB or from 0xC2000000 to 0xC23FFFFF on any other AXI Versatile base board. It can be accessed by any master in the system, including those on the baseboard. It is 4MB in size and aliased over this address range.

5.6 64bit Block RAM (LT-XC4VLX100+ only)

The FPGA Block RAM is configured as 64 bits wide, it occupies the address space from 0x82000000 to 0x8207FFFF on an EB or from 0xC2000000 to 0xC207FFFF on any other AXI Versatile base board. It can be accessed by any master in the system, including those on the baseboard. It is 512KB in size and aliased over this address range.

5.7 64bit Block RAM (LT-XC5VLX330 only)

The SSRAM is configured as two independently controlled 64 bit wide memory controllers. The first SSRAM block occupies the address space from 0x82000000 to 0x82FFFFFF on an EB or from 0xC2000000 to 0xC2FFFFFF on any other AXI Versatile base board, and the second SSRAM block occupies the address space from 0x83000000 to 0x83FFFFFF on an EB or from 0xC3000000 to 0xC3FFFFFF. It can be accessed by any master in the system, including those on the baseboard. Each memory region is 16MB in size and can operate contiguously between the two memory regions and is aliased over the whole 32MB address range.

5.8 Example slave

This is a 64bit example AXI slave, which contains 16 64bit registers. It occupies the address space from 0x84000000 to 0x84000FFF on EB or from 0xC4000000 to 0xC4000FFF on any other AXI Versatile base board.

5.9 Reserved and undefined memory

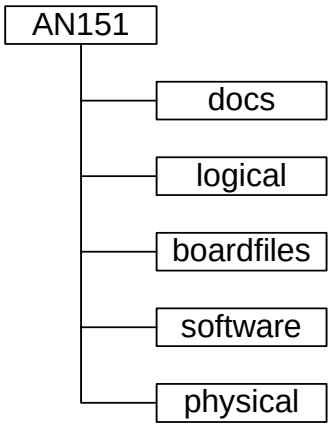
If reserved memory is accessed, it will be caught by the AXI bus matrix and return a decode error ('DECERR') which generates a data abort. The only exception to this is the APB subsystem, which has 14 spare PSEL signals for expansion. If this address range is accessed the AXIToAPB bridge will return an 'OKAY' response.

Copyright © 2008 ARM Limited. All rights reserved.

6 RTL

All of the AHB RTL for this design is provided as verilog. AXI components are supplied as netlists. 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.
- boardfiles : The files required to program the design into ARM development boards.
- physical : Synthesis and place and route (P&R) scripts and builds for target board.
- software : ARM code to run on the AN151 system

6.2 logical

The logical directory contains all the verilog required to build the system. The physical directory contains pre-synthesised components. The function of each block is shown earlier in 3.2 Module functionality.

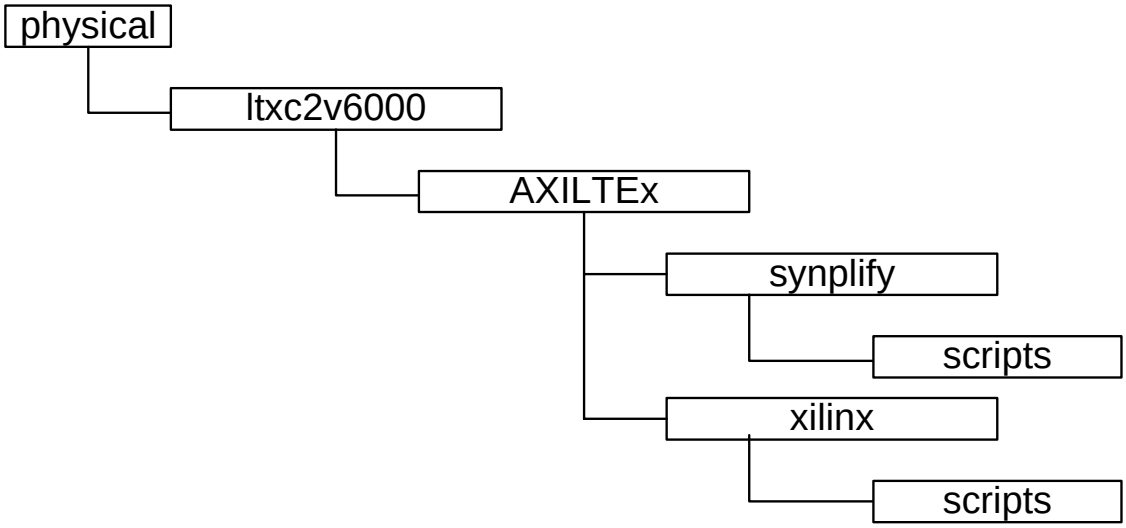
Each PrimeCell or other large IP block has its own directory (for example AHBToAXI).

The top level for this system is in AXILTEx.

6.3 physical

The physical directory contains the scripts for the tools used in the build process.

Copyright © 2008 ARM Limited. All rights reserved.



6.4 Building the App Note using Microsoft Windows

To build the App Note using Microsoft Windows first run the synplify_synth.bat batch file in the following directory : AN151/physical/ltxc2v6000/AXILTEx/synplify/scripts. This synthesizes the design.

Next change directory to AN151/physical/ ltxc2v6000/AXILTEx/xilinx/scripts and run the xilinx_par.bat batch file. This runs place and route on the design pulling in pre synthesized components.

A programmable bit file is generated under

AN151/physical/ ltxc2v6000/AXILTEx/xilinx/netlist

The process is exactly the same for a ltxc2v8000/ltxc4vlx160//ltxc4vlx200 tile, just replace ltxc2v6000 with ltxc2v8000/ltxc4vlx160/ltxc4vlx200 above.

6.5 Building the App Note using Unix

To build the App Note using Unix first run the synplify_synth.scr script file in the following directory : AN151/physical/ ltxc2v6000/AXILTEx/synplify/scripts. This synthesizes the design

Next change directory to AN151/physical/ ltxc2v6000/AXILTEx/xilinx/scripts and run the xilinx_par.scr script file. This runs place and route on the design pulling in pre synthesized components.

A programmable bit file is generated under

AN151/physical/ ltxc2v6000/AXILTEx/xilinx/netlist

The process is exactly the same for a ltxc2v8000/ltxc4vlx160//ltxc4vlx200 tile, just replace ltxc2v6000 with ltxc2v8000/ltxc4vlx160//ltxc4vlx200 above.

6.6 Board file selection

To use the pre-built bit file, use one of the following board files.

an151_ltxc2v4000_102c_xc2v…._axi_mast_slave_buildx.brd

an151_ltxc4vlx100_158a_xc4vlx…_axi_mast_slave_buildx.brd

To use a customer version, use one of the following board files.

an151_ltxc2v4000_102c_xc2v…._customer_rebuild.brd

an151_ltxc4vlx100_158a_xc4vlx…_customer_rebuild.brd

Copyright © 2008 ARM Limited. All rights reserved.

7 Example software

Example software is provided to verify the example design and the logic tile hardware.

The source files included are logic.c, logic.h. Logic.c rw_support.s. Logic.c contains the main code, written in C. rw_support.s contains several assembler functions to perform word, half-word and byte accesses to the ZBT SSRAM or FPGA Block RAM.

A batch file with calls to the compiler, assembler and linker; and a built image are also provided for each of the configurations. The software can be re-built with both ADSv1.2 and RVDSv2.1 or later.

After the FPGA is configured, as indicated by the FPGA_OK LED, you can download and execute the example software on any ARM processor in the system. For example the software can be executed by an ARM11MPCore processor on a CT11MPCore tile plugged on Tile Site 1 of the EB baseboard.

The example code communicates with the user via the debugger's console window. It operates as follows:

1. Reads the baseboard identification register to ensure that the software is executed on the correct system.
2. Sets the logic tile clocks
3. Flashes the LEDs on the logic tile and interface module (if present)
4. Tests the logic tile push button and interrupt controller.
5. Tests the ZBT SSRAM/Block RAMs for word, half-word and byte accesses.
6. Test the User LEDs and Switches.

Copyright © 2008 ARM Limited. All rights reserved.