DOCUMENTATION AMBSLP

AUTHOR: DANIEL MAGALOTTI

REVISION 1: PAOLA GIANNETTI, AUGUST 2014

REVISION 2: PANOS NEROUTSOS, OCTOBER 2014

REVISION 3: CALLIOPE-LOUISA SOTIROPOULOU, FEBRUARY 2015 – SYNC MODULE

REVISION 4: PAOLA, MARCH 2015 –LAMB INFOS, SOME REGISTER ISSUES, LOOP DESCRIPTION.

FIRMWARE AMBFTK SYSTEM

1 The AMBSLP logic 3
1.1 HIT FPGA 5
1.1.1 Implemented logic 5
1.1.2 GTP Serial Link Mapping 7
1.1.3 Spy Buffers 8
1.1.4 Synchronization module 9
1.1.5 HIT Vme Registers and memories 12
1.2 Control FPGA 20
1.2.1 Logic block 20
1.2.2 Error monitor logic and freeze signal 22
1.2.3 Control Vme registers 22
1.3 VME Chip 23
1.3.1 VME chip registers 23
1.4 Road FPGA 25
1.4.1 GTP Serial Link Mapping 25
1.4.2 Implemented logic 26
1.4.3 ROAD VME registers 27
2 AMBSLP JTAG CHAIN 32
2.1 AMBSLP V1 32
2.2 AMBSLP V2 32
3 LAMB 33
3.1 LAMB Chain configuration 33
3.2 Input distribution chain .... **Error! Bookmark not defined.**
3.3 Output collection chaiN .... **Error! Bookmark not defined.**

3.4 JTAG connection chain 33
4 VME commands 34
4.1 Internal AMBoard loop command 34
4.1.1 Simulation of the AM bank to produce roads 37

1 THE AMBSLP LOGIC

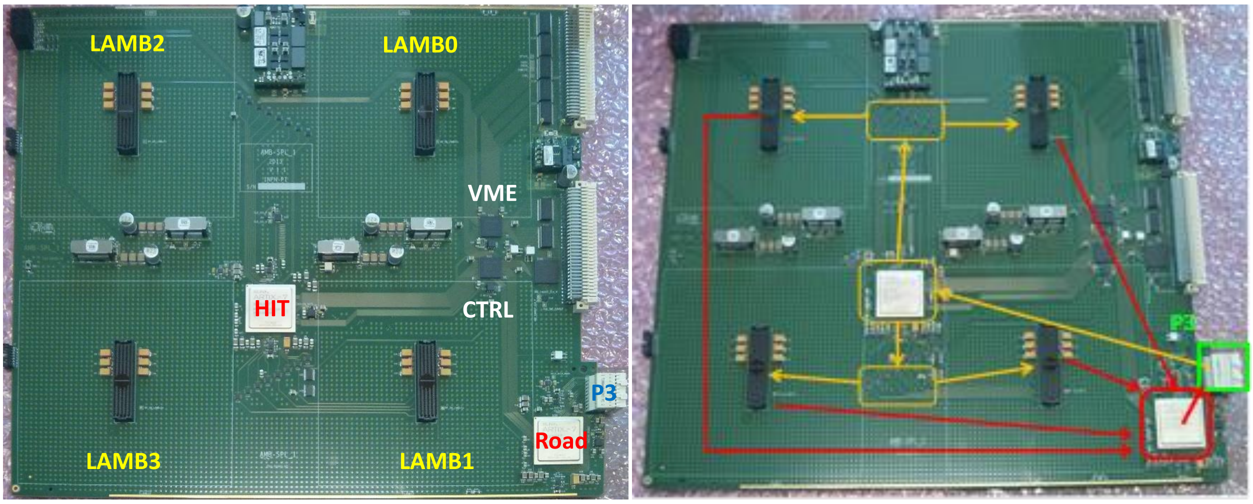


Figure 0a: the AMBSLP version 1 and the data traffic in the motherboard..

A 9U-VME board filled with 64 AM chips can hold 8 million patterns. To simplify input/output operations, the AM chips are grouped into AM units composed of 16 chips each, called Little Associative Memory Boards (LAMB, Figure 0b). A 9U-VME board has been implemented to hold 4 such units. Figure 0a shows the motherboard. The LAMB and the motherboard communicate through a high frequency and high pin-count connector placed in the center of the LAMB. A network of high speed serial links handles the data distribution from the input (the high density P3 connector on the bottom-right side of Figure 1a, called P3) to the 64 AM chips and back to the connector, for a total of ~750 point-to-point connections. Twelve input serial links (in yellow) carry the silicon data from the P3, and 16 output serial links (4 links from each LAMB represented by a red arrow in the figure) carry the fired patterns from the LAMBs to P3.

Figure 0b: The LAMB and the input data distribution to AM chips.

The data traffic is handled by 2 Xilinx FPGAs, the HIT and ROAD chips. They are 2 Xilinx-Artix7 which have 16 Gigabit Transceivers (GTP) each providing ultra-fast data transmission. HIT handles the input data, while ROAD in the red box near the P3 handles the output data. Two separate Xilinx Spartan-6 FPGAs implement the data control logic. The 12 input serial links are merged into the 8 buses received by each AM chip, one bus for each detector layer used for pattern matching.

The data rate is very challenging. A huge quantity of silicon data must be distributed at high rate (2 Gb/s on each serial link, for a total of 24 Gb/s maximum rate), with extremely large fan-out. Events are fed to the board at a maximum rate of 100 kHz. Each 10µs on average, 8 thousand words (16 bits) have to reach the patterns through 8 buses and a similarly large number of output words must be collected and sent back to the P3 (32 Gb/s maximum output rate). Each input word has to reach the 8 million patterns on the board.

The large input fan-out is obtained through 3 levels of serial fan-out chips to reach each of the 64 AM chips and a very powerful data distribution tree inside each AM chip itself. The AM chip compares 8 input words with 128k locations every 10 ns. The first level of 1:2 fan-out is visible inside the 2 yellow boxes of Figure 1a, which distribute each of the 8 buses to the 4 LAMBs. The other two levels are placed on the LAMBs and are visible in Figure 0b. Each LAMB has 40 1:4 fan-outs. The 8 red ones around the central connector (orange box) replicate each of the 8 incoming buses 4 times to make them available to a quartet of AM chips. For the input data distribution AM chips are organized into vertical quartets as shown by the blue dotted lines in Figure 6. The second level of fan-outs (yellow little squares) replicates again the bus 4 times, one for each single AM device in the quartet. The placement of chips on the LAMB has been studied and optimized with the goal of minimizing the crossing of the serial links.

Figure 0c shows how the output words are collected from the 16 AM chips in 4 daisy chains. Each AM device has the capability to receive outputs from other two AM chips and merge them internally with patterns that fired in the chip itself. Each daisy chain has a single output that goes directly to the connector. Each quartet also shares a 100 MHz low jitter clock necessary for the 11 serial links handled by each AM chip. The oscillator and the 1:4 fan-out for its output distribution are placed exactly in the middle of the quartet in the red boxes.

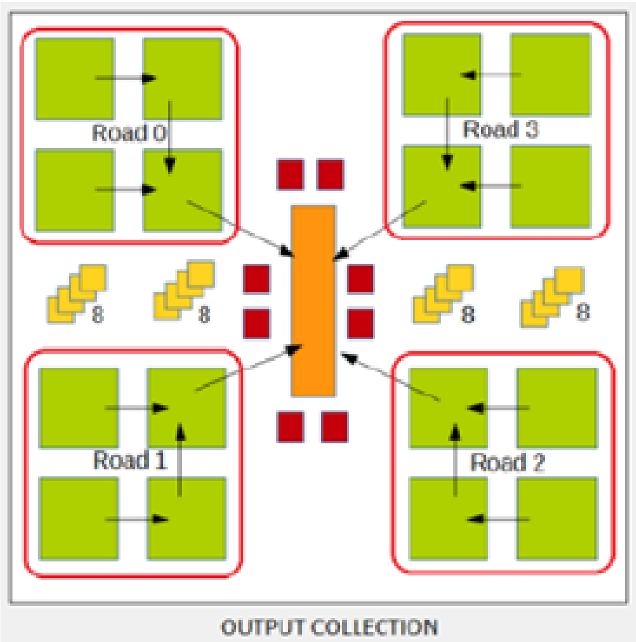


Figure 0c: Output data collection from AM chips.

1.1 HIT FPGA

The **HIT chip** (Artix7 xc7a200t) manages the distribution of the input hits coming from the the detector (4 SCT layers and the 4 PIXEL layers) to be downloaded in the AMChips in parallel on 8 independent buses. The Hit chip (blue box in fig 1), receives 12 serial buses at 2Gbps from the AUX board, through the external P3 connector (from the AUX board). Four buses are for the SCT layers and eight buses are for the PIXEL layers (each pixel bus is duplicated to fully exploit the P3 connector and because we expect higher data bandwidth needs from Pixel modules). A merge module combines each pixel couple of buses to provide a single output to be sent to the LAMBs. In conclusion HIT chip sends only four buses for both SCT and Pixel layers to the AMChips. Each HIT output bus is duplicated, one is distributed by fanouts (yellow boxes in figure 1) to the 2 LAMBs on the top half of AMBSLP (LAMB0 and LAMB2) and one is distributed (pink boxes in figure 1) to the 2 LAMBs on the bottom half of AMBSLP (LAMB1 and LAMB3).

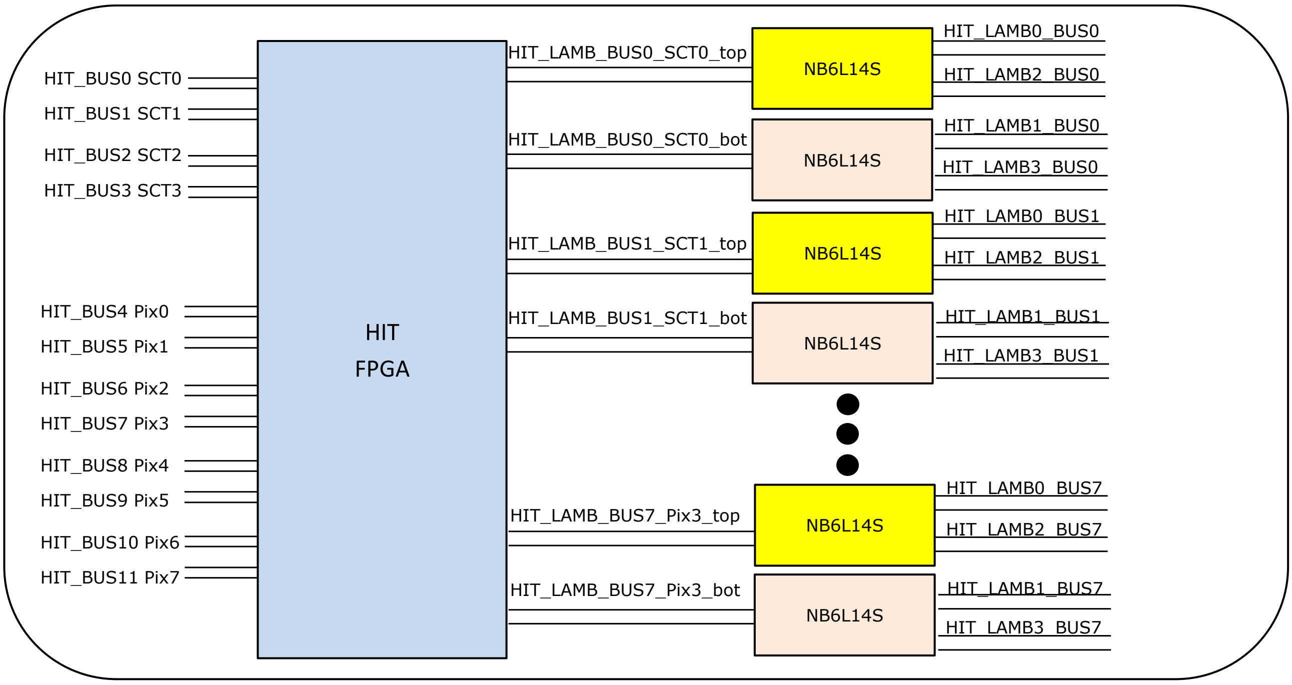


FIGURE 1: HIT DISTRIBUTION TO LAMBS

The internal logic implemented in the FPGA for each link is:

- a monitoring module (spy buffer) to control the data flow;
- a module to check the error in the data;
- a module to emulate the data flow, loading data from VME in a writable FIFO;

1.1.1 IMPLEMENTED LOGIC

The architecture of the logic implemented in the HIT FPGA is shown in figure 2 that reports the architecture for a single bus. The logic is the same for all the

12 input buses. For each couple of Pixel buses there is an additional multiplex that merges the couple of inputs into a single output.

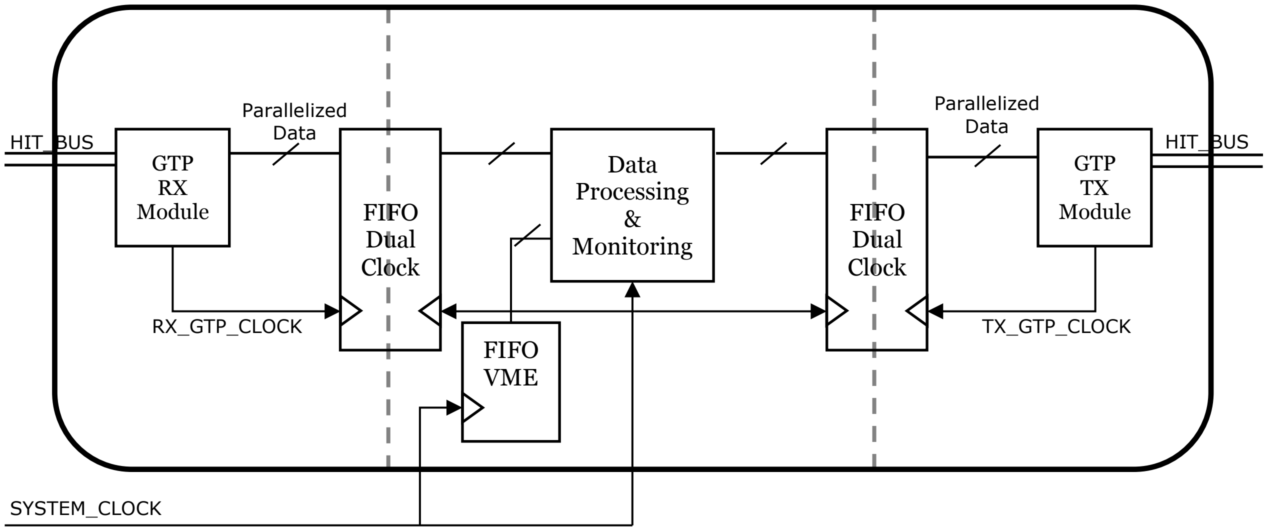


FIGURE 2: DATA PROCESSING LOGIC FOR EACH SERIAL LINK RECEIVED BY HIT CHIP

The input serial link is managed by the GTP Transceiver module. The 32 bit input bus is codified with the 8B/10B encoding, so the width of the bus becomes 40bits. They are transferred at 50MHz rate, so the frequency of the link is 2Gbp.

A simple protocol is used between the AUX card and the AMBSLP. We use two types of control words: **idle words** (BCBC1C1C, K character F) are sent when no data valid is present on the input bus; an **End Event word** (F7RRNNNN, K character 8) is sent when data words of an event are finished.

To summarize the data flow in input to the AMBSLP

Type of word	Value	K character
IDLE WORD	BCBC1C1C	1111
HITS DATA	XXXXYYYY XXXXYYYY	0000 0000
End Event WORD	F7RRNNNN	1000

where the XXXX and YYYY are two 16-bit hits that are transmitted in the same word;RR are bits reserved for error codes , NNNN is the ID number of the specific event under process. The protocol used between HIT chip and the AMchip is similar and is described in the following table:

Type of word	Value	K character
IDLE WORD	BCBC1C1C	1111
HITS DATA	XXXXYYYY XXXXYYYY	0000 0000
End Event WORD	F7RRNNNN	1000

Add the encoding of the opcode to send to AMchip after the init event		

Looking at figure 2 from the left, the *"GTP module"* is the first part of the process. It de-serializes the incoming bitstream to send to the output a parallelized data bus (the 32 bits data word and the 4-bit control k word described in the tables above) and the recovery clock. An elastic buffer is enabled in the GTP module to help the module to synchronize the internal clock domain .

The output parallel bus is stored in a *"Dual clock FIFO"*. The writing clock is the recovery clock of the GTP module, the depth of the FIFO is 1024k words and the program full signal is used as hold signal to be sent to the AUX card to stop the data flow if the Fifo becomes full. The output of the FIFO is read with the system clock signal.

A *"VME FIFO"*, 8Kword depth, has been added to perform standalone tests, downloading data from VME. It is written by the VME module. A mux, controlled by the TMODE signal, selects the normal data flow from the GTP module (TMODE-0) or the test data flow from the VME FIFO (TMODE=1).

The *"Data processing & monitoring module"* controls the event flow in the chip. It looks for the end event word (data word = F7RRNNNN and control word=1000) and waits for the initialization signal in order to enable the processing of the next event. The monitoring module is used to copy the input data in a memory called spy buffer. Itis realized with a circular buffer. A logic is implemented to freeze the spy buffer in case of error and the content of the memory is read by the VME interface. The depth of the spy buffer memory is 8Kword of 16 bits.

The module that controls the transmitter GTX is again composed by a dual clock FIFO and a logic to insert the idle word when no data valid is present on the bus.

1.1.2 GTP Serial Link Mapping

A list of the input and output data serial link mapping is reported in table 1The IO pad need to be placed.

TABLE 1

Data Bus	PAD		QUAD	RX TILE	TX TILE	AUX Processor
	P	N				
SCT_HIT_0	AJ13	AK13	113	3	-	
SCT_HIT_1	AJ19	AK19	213	1	-	
SCT_HIT_2	AJ21	AK21	213	3	-	
SCT_HIT_3	AL20	AM20	213	2	-	
PIX_HIT_0	AL16	AM16	113	1	-	

PIX_HIT_1	AJ17	AK17	113	0	-	
PIX_HIT_2	F13	E13	116	0	-	
PIX_HIT_3	F19	E19	216	2	-	
PIX_HIT_4	F17	E17	116	3	-	
PIX_HIT_5	D16	C16	116	2	-	
PIX_HIT_6	D20	C20	216	1	-	
PIX_HIT_7	F21	E21	216	0	-	
RX_LAMB_0	D18	C18	216	3	-	
RX_LAMB_0	AL18	AM18	213	0	-	
RX_LAMB_0	F15	E15	116	1	-	
RX_LAMB_0	AJ15	AK15	113	2	-	
HIT_A_0	B13	A13	116	-	0	
HIT_A_1	D14	C14	116	-	1	
HIT_A_2	B15	A15	116	-	2	
HIT_A_3	B17	B17	116	-	3	
HIT_A_4	B23	A23	216	-	3	
HIT_A_5	D22	C22	216	-	2	
HIT_A_6	B21	A21	216	-	1	
HIT_A_7	B19	A19	216	-	0	
HIT_B_0	AN19	AP19	213	-	3	
HIT_B_1	AN21	AP21	213	-	2	
HIT_B_2	AL22	AM22	213	-	1	
HIT_B_3	AN23	AP23	213	-	0	
HIT_B_4	AN17	AP17	113	-	0	
HIT_B_5	AN15	AP15	113	-	1	
HIT_B_6	AL14	AM14	113	-	2	
HIT_B_7	AN13	AP13	113	-	3	
MGTREFCLK0 X1Y1	H16	G16	116	-	-	
MGTREFCLK0 X0Y1	H18	G18	216	-	-	
MGTREFCLK1 X1Y0	AG16	AH16	113	-	-	
MGTREFCLK1 X0Y0	AG18	AH18	213	-	-	

1.1.3 SPY BUFFERS

Each Spy Buffer is composed by a circular memory and a status register. The spy buffer memory is attached to the bus to monitor the incoming data. It is a dual port memory: the Spy buffer cannot be written by VME, it is Read Only memory. The write operation is controlled by the port A to store all the words coming in input during the normal data flow, the read operation is controlled by port B with VME interface signals.

The status register contains tree information:

- ✓ the pointer at the first free address location of the memory;
- ✓ the flag of the freeze status;
- ✓ the flag of the memory overflowstatus.

A VME write to the status register (data is irrelevant) resets to zero the pointer, reset the overflow flag, but does not change the freeze status, since it is controlled by specific logic functions. The freeze bit is Read-Only (check with Pierluigi).

The memory write operation during the normal data flow is stopped (to preserve the data inside) when a freeze signal is asserted. The freeze signal is a global signal for all the Spy Buffers in the board and can also be propagated to the upstream board. These are the particular conditions that generate the freeze signal:

- ✓ when an error is detected on a board freeze is asserted to all the Spy Buffers on that board; it is also sent to the board immediately upstream to freeze its Spy Buffers since the error could be generated by that board;
- ✓ there is a bit in the End Event word that tells all boards to freeze their Spy Buffers after processing that specified event.

This last option enables events to be read out and compared with simulation to ensure that there aren't subtle problems in the hardware. After freeze is set, no data can be written into the memory and the content of the memory is read through VME access.

1.1.4 Synchronization module

The synchronization module (LossOfSync) receives the 12 Hit Bus data input and its task is to identify whether the 12 Input Buses carry synchronized data (the same event -EventID- appears in all 12 buses). The module traces the End Event word at each Hit Bus, produces an End Event Reference Word and compares this value to event id in every input bus to find if and which of those buses are different in comparison to the reference.

The synchronization module consists of:

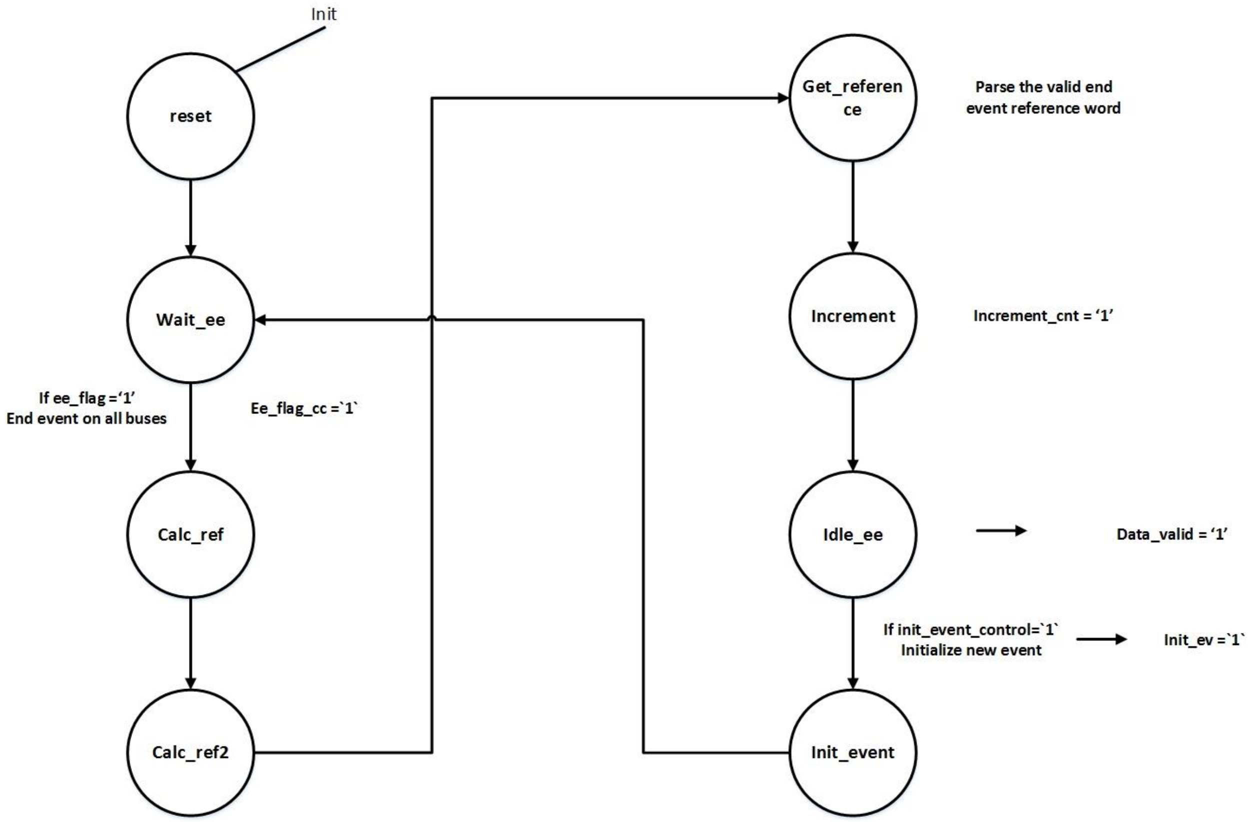
- the Majority Vote module that produces the End Event Reference Word
- the Sync Module that implements all the logic for the loss of synchronization detection
- the control module (FSM)

Majority Vote Module

The End Event Reference Word is calculated using the majority vote module. This module receives all the signals from the Hit buses and a data valid signal that is active when all input buses contain End Events. wordsIt then implements a comparison of all the End Event words and produces a reference word that is the most common Event ID among the Hit buses. Additionally it reports in how many buses the reference word exists. The module is designed to accept 12 or 8 input bus input by using a switch signal.

FSM Module

The FSM block diagram that describes the functionality of the module is shown in the figure below.



Analysis of the FSM states:

- ***Reset:*** The system starts from here only when *init*='1' (RESET='1').
- ***Wait_ee:*** waiting for the end event tag and data valid flags from the FIFOs. If a set of values has the end event tag on every bus as well as all the data valid signals on all FIFOs are valid ('1'), then we proceed with the rest of the process, that is calculate the End Event Reference word.
- ***Calc_ref:*** Lasts 1 clk cycle. Calculates the End Event Reference word. The end event reference word is the most common value among the HitBuses.
- ***Calc_ref2:*** The process of comparison between the hitbuses lasts two clk cycles.
- ***Get_reference:*** The state where the end event reference word is valid (flag signal *parse_reference*). At this state the ee_flag_cc signal is sent to the Control Chip.

- ***Increment***: In this state we increment the appropriate counters which correspond to the HitBuses that don't have the same value with the end event reference word (*HBSn_LoS_Counters*).
- ***Idle_ee***: Waiting to the init_event_control (*INIT_HIT_REG*) signal from the Control Chip.
- ***Init_event***: Initialize all the registers except for the counters and goes back to the *wait_ee* state.

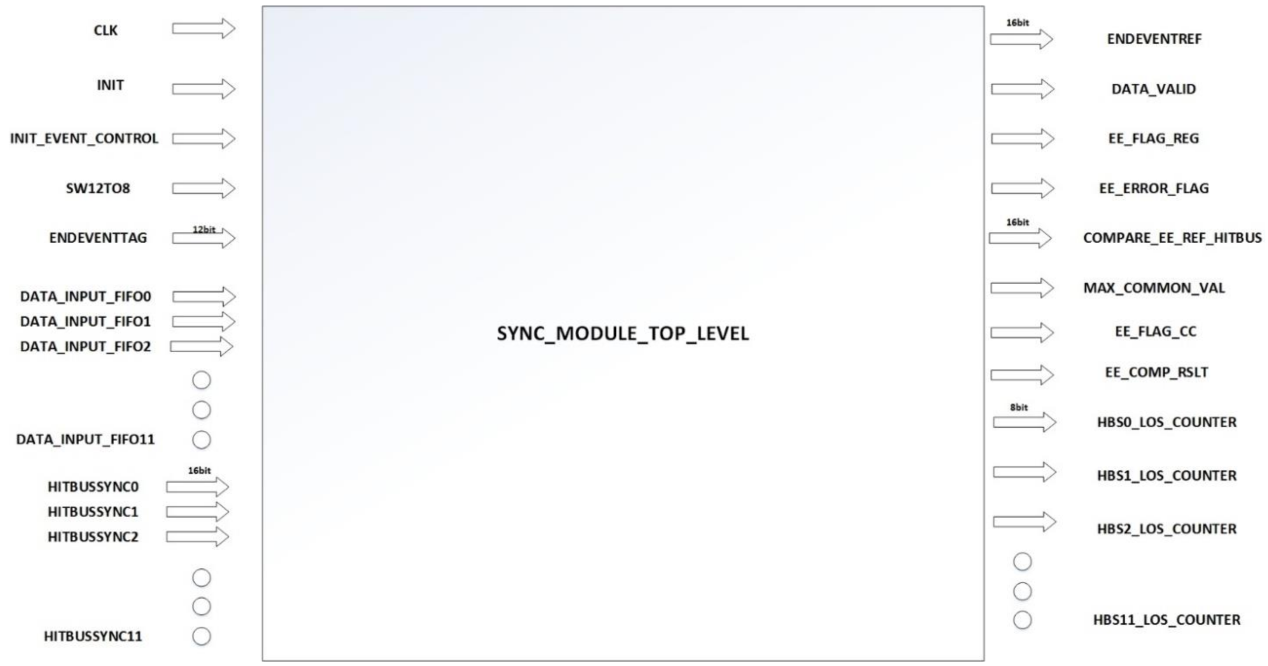
The output of the Sync Module is valid only when "data_valid" is asserted.

<u>Sync Module</u>

The sync module accepts all the words from the Hit buses, the End Event Reference word from the Majority Vote module and the control signals from the FSM. The module identifies whether there is a loss of syncronization and increases the values of the counters that correspond to the bus that has lost sync.

<u>Synchronization Module (Top level)</u>

List of Input and Output Signals for the Sync Module Top Level Entity:



Input signals:

- <u>*Clk:*</u> clock
- <u>*init :*</u> Master reset.
- <u>*sw12to8*</u> *:* '0' for 12 Hit Bus input, '1' for 8 Hit Bus Input.
- <u>*init_event_control :*</u> Initialization signal that comes from the Control Chip.

- *HitBusSyncN* (where N from 0 to 11) 16-bit bus width: Input Data from the Hit buses.
- *Data_Input_FifoN* (where N from 0 to 11): Data valid flag on each input FIFO.
- *Endeventtag (12-bit) :* A single cycle pulse that indicates the end event tag for each hit bus.

Output signals:

- *EndEventRef* : End Event Reference value. Read only when data_valid is '1'.
- *HBSn_LoS_Counter* (where n from 0 to 11): Counters for the losses of synchronization on each bus respectively.
- *ee_comp_rslt*: Registered signal that indicates whether all the End Event Words have the same value. The signal is '1' when there is end event and equal values of the HitBuses.
- *ee_flag_reg*: Indicates the presence of an end event on all buses (value '1' in the end event tag for all the HIT buses).
- *ee_error_flag*: The signal arises if only the end event has arrived in all HitBuses and at least one comparison is not equal.
- *ee_flag_cc*: End Event signal that goes to the Control Chip
- *data_valid:* Indicates whether all the module output data (end event reference word and error counters) are valid.
- *compare_eeref_hitbus* (12-bit): compares each hitbus with the end event reference word. '0' when they are equal, '1' when they are not.
- *max_common_val:* Signal that represents how many times the EndEventRef is repeated among the Hit Buses

1.1.5 HIT VME REGISTERS AND MEMORIES

In the Table 2 the list of the internal register in the HIT chip is reported.

TABLE 2: THE ADDRESSER FOR THE VME REGISTER OF THE HIT CHIP

ADDR FPGA [26:2]	ADDR AMBFTK [31:0]	DESCRIPTION	N of bits	Type
000800	xx002000	Fifo_SCT_link0	32	R/W
000801	xx002004	isK_SCT_link0	4	R/W
000802	xx002008	Fifo_SCT_link1	32	R/W

000803	xx00200C	isK_SCT_link1	4	R/W
000804	xx002010	Fifo_SCT_link2	32	R/W
000805	xx002014	isK_SCT_link2	4	R/W
000806	xx002018	Fifo_SCT_link3	32	R/W
000807	xx00201C	isK_SCT_link3	4	R/W
000808	xx002020	Fifo_PIX_link0	32	R/W
000809	xx002024	isK_PIX_link0	4	R/W
00080A	xx002028	Fifo_PIX_lin1	32	R/W
00080B	xx00202C	isK_PIX_link1	4	R/W
00080C	xx002030	Fifo_PIX_link2	32	R/W
00080D	xx002034	isK_PIX_link2	4	R/W
00080E	xx002038	Fifo_PIX_link3	32	R/W
00080F	xx00203C	isK_PIX_link3	4	R/W
000810	xx002040	Fifo_PIX_link4	32	R/W
000811	xx002044	isK_PIX_link4	4	R/W
000812	xx002048	Fifo_PIX_lin5	32	R/W
000813	xx00204C	isK_PIX_link5	4	R/W
000814	xx002050	Fifo_PIX_link6	32	R/W
000815	xx002054	isK_PIX_link6	4	R/W
000816	xx002058	Fifo_PIX_link7	32	R/W
000817	xx00205C	isK_PIX_link7	4	R/W
000818	XX002060	SPY_SCT_link0_STATUS *Bit 12-0: pointer* *Bit 14: overflow* *Bit 15: freeze*	16	RO WR clear
00819	XX002064	SPY_SCT_link1_STATUS *Bit 12-0: pointer* *Bit 14: overflow* *Bit 15: freeze*	16	RO WR clear
0081A	XX002068	SPY_SCT_link2_STATUS *Bit 12-0: pointer* *Bit 14: overflow* *Bit 15: freeze*	16	RO WR clear

0081B	XX00206C	SPY_SCT_link3_STATUS *Bit 12-0: pointer* *Bit 14: overflow* *Bit 15: freeze*	16	RO WR clear
0081C	XX002070	SPY_PIX_link0_STATUS *Bit 12-0: pointer* *Bit 14: overflow* *Bit 15: freeze*	16	RO WR clear
0081D	XX002074	SPY_PIX_link1_STATUS *Bit 12-0: pointer* *Bit 14: overflow* *Bit 15: freeze*	16	RO WR clear
0081E	XX002078	SPY_PIX_link2_STATUS *Bit 12-0: pointer* *Bit 14: overflow* *Bit 15: freeze*	16	RO WR clear
0081F	XX00207C	SPY_PIX_link3_STATUS *Bit 12-0: pointer* *Bit 14: overflow* *Bit 15: freeze*	16	RO WR clear
00820	XX002080	SPY_PIX_link4_STATUS *Bit 12-0: pointer* *Bit 14: overflow* *Bit 15: freeze*	16	RO WR clear
00821	XX002084	SPY_PIX_link5_STATUS *Bit 12-0: pointer* *Bit 14: overflow* *Bit 15: freeze*	16	RO WR clear
00822	XX002088	SPY_PIX_link6_STATUS *Bit 12-0: pointer* *Bit 14: overflow* *Bit 15: freeze*	16	RO WR clear
00823	XX0028C	SPY_PIX_link7_STATUS	16	RO

		Bit 12-0: pointer *Bit 14: overflow* *Bit 15: freeze*		WR clear
00824	XX002090	HOLD_FLAG	12	RO
00825	XX002094	HIT_FIRMWARE_VERSION *Bit 0: tmode* *Bit 3-1 NOT USED"* *Bit 7-4 version* *Bit 11-8 year* *Bit 15-12 PllLock* *Bit 19-16 PllLostRef* *Bit 23-20 isKstable* *Bit 27-24 isKchaged*	20	RO
00826	XX002098	HIT_CONTROL *Bit 15-0: SET operation* *Bit 31-16: CLEAR operation* *Bit 0: StartReadFifoVME* *Bit 1: LoopFifoVME*	3	RW
00827	XX00209C	PIX_ALIGNMENT_STATUS *Bit 3-0: gt0_quad113* *Bit 7-4: gt0_quad116* *Bit 11-8: gt0_quad213* *Bit 15-12: gt0_quad216*	16	RO
00828	XX0020A0	PIX_RESETDONE_STATUS *Bit 3-0: tx_resetdone_quad113* *Bit 7-4: rx_resetdone_quad113* *Bit 11-8: tx_resetdone_quad116* *Bit 15-12: rx_resetdone_quad116* *Bit 19-16: tx_resetdone_quad213* *Bit 23-20: rx_resetdone_quad213* *Bit 27-24: tx_resetdone_quad216* *Bit 31-28: rx_resetdone_quad216*	32	RO

00829	XX0020A4	EMPTY_FLAG	12	RO
0082A	XX0020A8	CONFIGURE TEST *Bit 3-0: Board Version* *Bit 6-4: PRBS pattern selection* *Bit 8: PRBS enable checker*		RW NOT RESET BY INIT
0082B	XX0020AC	CONFIGURE POLARITY *Bit 15-0: TX polarity* *213-216-116-113* *Bit 31-16: RX polarity*	32	RW NOT RESET BY INIT
0082C	XX0020B0	A_LOSSOFSYNCH0 *Bit 7-0: sct link 0* *Bit 15-8: sct link 1* *Bit 23-16: sct link 2* *Bit 31-24: sct link 3*	32	RO
0082D	XX0020B4	A_LOSSOFSYNCH1 *Bit 7-0: pix link 0* *Bit 15-8: pix link 0* *Bit 23-16: pix link 1* *Bit 31-24: pix link 1*	32	RO
0082E	XX0020B8	A_LOSSOFSYNCH2 *Bit 7-0: pix link 2* *Bit 15-8: pix link 2* *Bit 23-16: pix link 3* *Bit 31-24: pix link 3*	32	RO
0082F	xx0020BC	K WORD MONITORING		
00830	XX0020C0	TIMER_PIX	-	-
00831	XX0020C4	ERROR_PIX *Bit 0: Error flag* *Bit 1: Error critical*		
00832	xx0020C8	PRBS ERROR LINK 113 *Bit 7-0: gt0* *Bit 15-8: gt1* *Bit 23-16: gt2*	32	RO

		Bit 31-24: gt3		
00833	xx0020CC	PRBS ERROR LINK 116 *Bit 7-0: gt0* *Bit 15-8: gt1* *Bit 23-16: gt2* *Bit 31-24: gt3*	32	RO
00834	xx0020D0	PRBS ERROR LINK 213 *Bit 7-0: gt0* *Bit 15-8: gt1* *Bit 23-16: gt2* *Bit 31-24: gt3*	32	RO
00835	xx0020D4	PRBS ERROR LINK 216 *Bit 7-0: gt0* *Bit 15-8: gt1* *Bit 23-16: gt2* *Bit 31-24: gt3*	32	RO
00836	xx0020D8	DEGUB_DISABLE_HOLD_HIT	12	RW
00837	xx0020DC	DEBUG_FORCE_HOLD_HIT *Disable hold has priority*	12	RW
00838	xx0020E0	PRBS CHECKER LINK 113 *Bit 7-0: gt0* *Bit 15-8: gt1* *Bit 23-16: gt2* *Bit 31-24: gt3*	32	RO
00839	xx0020E4	PRBS CHECKER LINK 116 *Bit 7-0: gt0* *Bit 15-8: gt1* *Bit 23-16: gt2* *Bit 31-24: gt3*	32	RO
0083A	xx0020E8	PRBS CHECKER LINK 213 *Bit 7-0: gt0* *Bit 15-8: gt1* *Bit 23-16: gt2*	32	RO

		Bit 31-24: gt3		
0083B	xx0020EC	PRBS CHECKER LINK 216 *Bit 7-0: gt0* *Bit 15-8: gt1* *Bit 23-16: gt2* *Bit 31-24: gt3*	32	RO
0083C 00FFF	xx0020F0 xx003FFC			

In the Table 2 there is the list of the memory in the HIT chip

ADDR FPGA [26:2]	ADDR AMBFTK [31:0]	DESCRIPTION	N of bits	Type
S: 20000 E: 21FFF	S: XX080000 E: XX087FFC	ISPY_SCT0	32 bit	RO
S: 22000 E: 23FFF	S: XX088000 E: XX08FFFC	ISPY_SCT1	32 bit	RO
S: 24000 E: 25FFF	S: XX090000 E: XX097FFC	ISPY_SCT2	32 bit	RO
S: 26000 E: 27FFF	S: XX098000 E: XX09FFFC	ISPY_SCT3	32 bit	RO
S: 28000 E: 29FFF	S: XX0A0000 E:XX0A7FFC	ISPY_PIX0	32 bit	RO
S: 2A000 E: 2BFFF	S: XX0A8000 E: XX0AFFFC	ISPY_PIX1	32 bit	RO
S: 2C000 E: 2DFFF	S: XX0B0000 E: XX0B7FFC	ISPY_PIX2	32 bit	RO
S: 2E000 E: 2FFFF	S: XX0B8000 E: XX0BFFFC	ISPY_PIX3	32 bit	RO
S: 30000 E: 31FFF	S: XX0C0000 E:XX0C7FFC	ISPY_PIX4	32 bit	RO
S: 32000 E: 33FFF	S: XX0C8000 E: XX0CFFFC	ISPY_PIX5	32 bit	RO

S: 34000 E: 35FFF	S: XX0D0000 E: XX0D7FFC	ISPY_PIX6	32 bit	RO
S: 36000 E: 37FFF	S: XX0D8000 E: XX0DFFFC	ISPY_PIX7	32 bit	RO
S: 38000 E: 3FFFF	S: XX0E0000 E: XX0FFFFC	NOT USED	32 bit	RO

1.2 Control FPGA

The control chip manages the event processing in the AMBSLP, both for the input hit distribution and the road readout. Control chip receive the END event from both HIT and ROAD chips and send back the INIT event signal. An additional bus of 32 lines is available between Control and the two FPGAs for any possible future need. It is already partially used to communicate the event ID and information about the errors. Control chip is also used to propagate commands, control signals to both HIT and ROAD: it makes the final decision to assert freeze, stopping all the Spy Buffers, it propagates INIT and TMODE asserted by VME etc. etc. Figure 3 summarizes all the connections between Control and the other parts of the system.

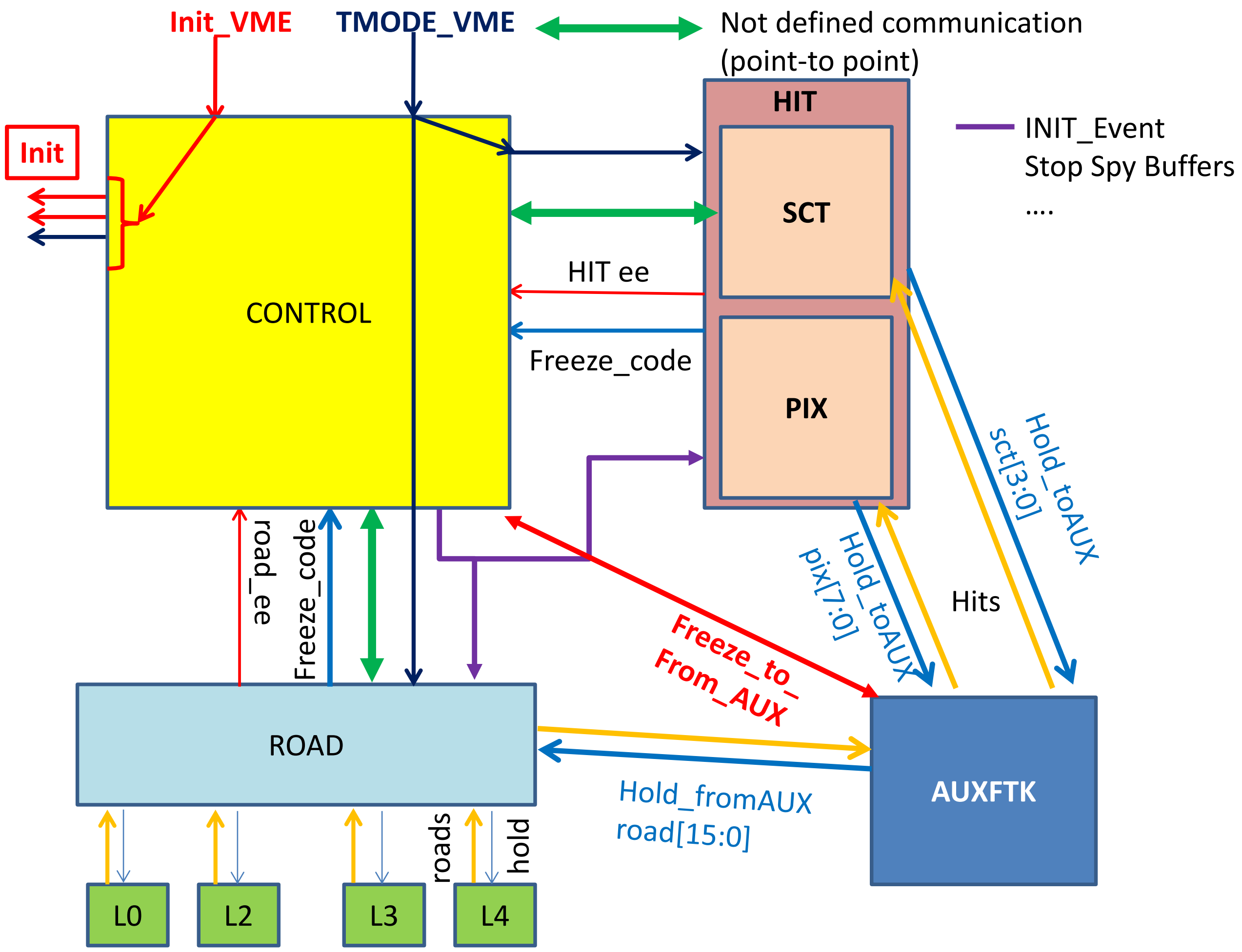


Figure 3: connections between Control and the other chips

1.2.1 Logic block

The logic that controls the event processing is organized in two Finite State Machines, one for the HIT distribution and one for the ROAD readout. Communication between the two FSMs controls the correct data processing inside the AMchip: during the loading of the hits of the N+1 event, the road of the event N are read out.

Figure 4 shows the FSM diagrams.

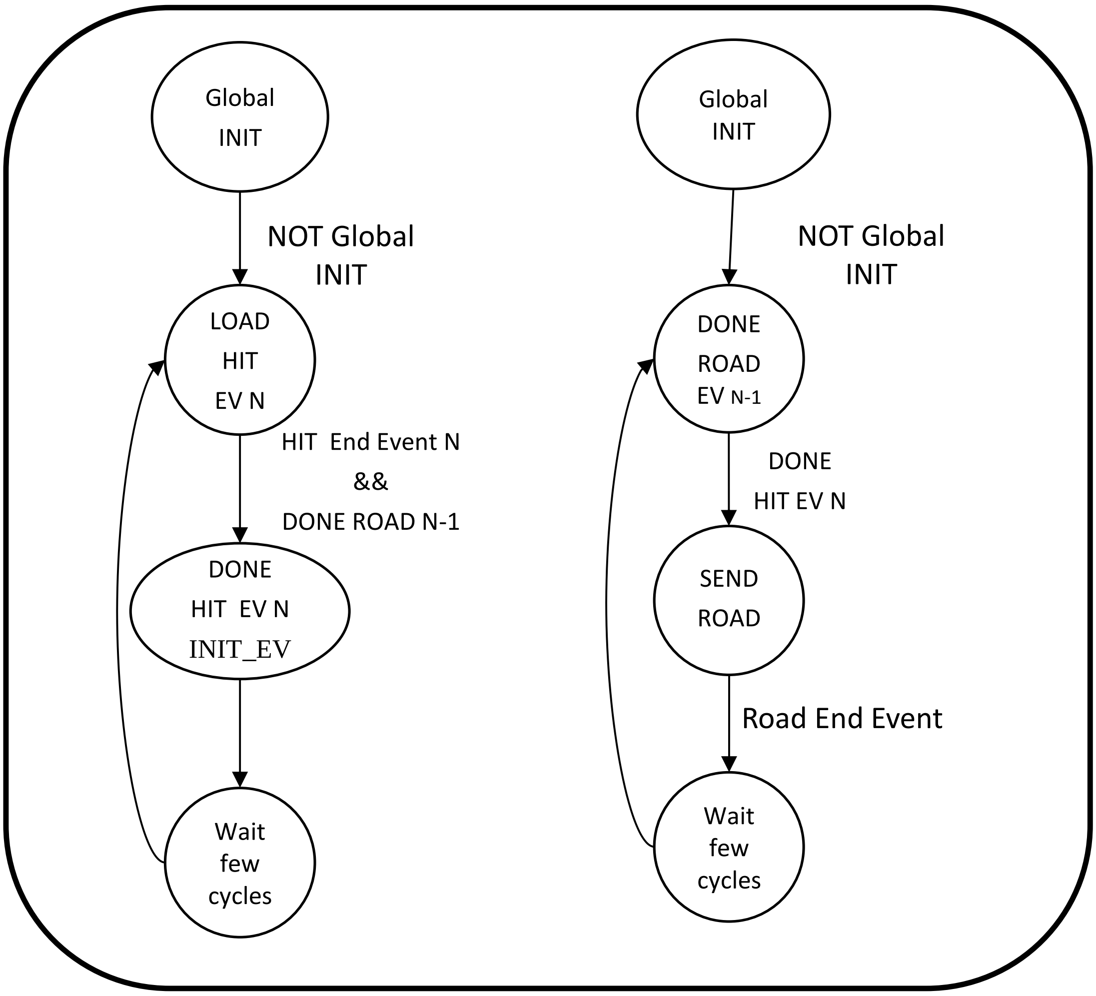


Figure 4

Gloabal INIT puts the two FSMs in the initial state. When Global INIT is removed both FSMs move to the next state.

Looking at the two FSM: in the LOAD HIT state the HIT FSM waits until the End Event (event N) is received, so all the hits have been sent to the AMchip; When the HIT FSM goes to the next state, DONE HIT, also the roads of the previous event have been finished since the ROAD FSM is in the state DONE ROAD, so INIT_event is asserted to the AMchips, the new event ID is stored and after a programmable number of wait cycles the HIT FSM goes back to the LOAD HIT state. The ROAD FSM goes to the next state "SEND ROAD" at the same time the HIT FSM transits to the DONE HIT state. Control chip enables the "send ROADs" signal to ROAD chip (event N) that receive Roads from the LAMBs and sends them to the AUX board. When the Roads are finished ROAD sends the end event word to AUX, end event signal to Control chip and the FSM transits to the next state to wait a programmable number of clock cycles, and after that it goes back to DONE ROAD state where the FSM waits the loading of HITs of the next event. So also the Road FSM is ready to process the next event.

1.2.2 *Error monitor logic and freeze signal*

1.2.3 *Control VME registers*

In the following table is the list of the control register

ADDR FPGA [26:2]	ADDR AMBFTK [31:0]	DESCRIPTION	N of bits	Type
01800	XX006000	CONTROL_STATUS_REGISTER Bit 3-0: version Bit 7-4 year Bit 11:8: FSMState Bit 15:12: Presence LAMB	RO	RO
01801	XX006004	HOLD FLAGS	RO	RO
01802	XX006008	Error Severity Register		R/W
01803	XX00600C	PRESENCE_LAMB Bit 0: LAMB 0 Bit 1: LAMB 1 Bit 2: LAMB 2 Bit 3: LAMB 3	4	RO
01804	XX006010	FREEZE	RW	1
01805	XX006014	MASK_EE Bit 0: mask HIT Bit 1: mask ROAD	RW	2
01806	XX006018			
01807	XX00601C			

1.3 VME CHIP

We describe the AMBSLP slave interface. The VME interface controls registers and memories of three different FPGA: HIT, ROAD and CONTROL. In the different FPGAs we have to address both the registers and the memories so we divide the address space in the following way

The GREEN is to identify the internal register

The VIOLET is to identify the internal AMBSLP memories

The YELLOW is to identify which memory of the possible 16 in total

The RED is reserved for the AUX card

26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2
VME INFERFACE REGISTER 0x00 – 0x3FF																								
0	0	0	0	0	0	0	0	0	0	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x
HIT REGISTER 0x40 – 0x7F																								
0	0	0	0	0	0	0	0	0	0	0	0	0	1	x	x	x	x	x	x	x	x	x	x	x
ROAD REGISTER 0x80 – 0xBF																								
0	0	0	0	0	0	0	0	0	0	0	0	1	0	x	x	x	x	x	x	x	x	x	x	x
CONTROL REGISTER 0xC0 – 0xFF																								
0	0	0	0	0	0	0	0	0	0	0	0	1	1	x	x	x	x	x	x	x	x	x	x	x
HIT MEMORY 0x10000																								
0	0	0	0	0	0	0	1	0000 to 1111				x	x	x	x	x	x	x	x	x	x	x	x	x
ROAD MEMORY 0x20000																								
0	0	0	0	0	0	1	0	0000 to 1111				x	x	x	x	x	x	x	x	x	x	x	x	x

1.3.1 VME CHIP REGISTERS

ADDR FPGA [26:2]	ADDR AMBFTK [31:0]	DESCRIPTION	N of bits	Type
000000	XX000000	STATUS_REGEGISTER_VME		R
000001	XX000004	INIT	1	W

000002	XX000008	TMODE	1	R/W
000003	XX00000C	TCK_ENABLE	1	R/W
000004	XX000010	TEST32BIT	32	R/W
Not yet impl.	XX000014	TMS_FPGA	1	R/W
Not yet impl.	XX000018	TDI - FPGA	1	R/W
Not yet impl.	XX00001C	TDO - FPGA	1	RO
000008	XX000020	TMS_BSCAN	8x4=32	W
000009	XX000024	TDI_BSCAN	8x4=32	W
00000A	XX000028	TDO_BSCAN	8x4=32	R
00000B	XX00002C	TRS_BSCAN	8x4=32	W

1.4 ROAD FPGA

The ROAD chip receives the roads from the LAMBs and send them to the AUX card. The same chip controls all the 16 links in input and output.

The structure of the logic (see figure 4) is similar to the HIT chip: there are two FIFOs to solve the asynchronism of the GTP clock with respect to the system clock. A VME FIFO is also used to be able to download data from VME.

1.4.1 GTP SERIAL LINK MAPPING

A list of the input and output data serial link mapping is reported in the table. In the last column there is the mapping with the AUX processor. [QUALE E' LA DIFFERENZA FRA "ROAD_LAMB3_0" E "ROAD_OUT_L3_0"? QUALE E' L'INPUT E QUALE L'OUTPUT?]

TABLE

Data Bus	PAD		QUAD	RX TILE	TX TILE	AUX Processor
	P	N				
ROAD_LAMB3_0	AJ15	AK15	113	1	-	2_0
ROAD_LAMB3_1	AL18	AM18	213	3	-	4_2
ROAD_LAMB3_2	F13	E13	116	3	-	1_2
ROAD_LAMB3_3	F15	E15	116	2	-	1_1
ROAD_LAMB2_0	F19	E19	216	1	-	1_0
ROAD_LAMB2_1	D18	C18	216	0	-	3_0
ROAD_LAMB2_2	F17	E17	116	0	-	3_1
ROAD_LAMB2_3	D16	C16	116	1	-	1_3
ROAD_LAMB1_0	AJ21	AK21	213	0	-	4_1
ROAD_LAMB1_1	AL20	AM20	213	1	-	4_0
ROAD_LAMB1_2	D20	C20	216	2	-	3_3
ROAD_LAMB1_3	F21	E21	216	3	-	3_2
ROAD_LAMB0_0	AJ19	AK19	213	2	-	4_3
ROAD_LAMB0_1	AJ17	AK17	113	3	-	2_2
ROAD_LAMB0_2	AJ13	AK13	113	0	-	2_1
ROAD_LAMB0_3	AL16	AM16	113	2	-	2_3
ROAD_OUT_L3_0	AN23	AP23	213	-	0	
ROAD_OUT_L3_1	AL22	AM22	213	-	1	
ROAD_OUT_L3_2	AN21	AP21	213	-	2	
ROAD_OUT_L3_3	AN19	AP19	213	-	3	
ROAD_OUT_L2_0	AL14	AM14	113	-	1	
ROAD_OUT_L2_1	AN15	AP15	113	-	2	
ROAD_OUT_L2_2	AN13	AP13	113	-	0	
ROAD_OUT_L2_3	AN17	AP17	113	-	3	

ROAD_OUT_L1_0	B19	A19	216	-	0	
ROAD_OUT_L1_1	B17	A17	116	-	0	
ROAD_OUT_L1_2	D22	C22	216	-	2	
ROAD_OUT_L1_3	B23	A23	216	-	3	
ROAD_OUT_L0_0	D14	C14	116	-	2	
ROAD_OUT_L0_1	B21	A21	216	-	1	
ROAD_OUT_L0_2	B15	A15	116	-	1	
ROAD_OUT_L0_3	B13	A13	116	-	3	
ROAD_MGTREFCLK0_X1Y1_116	H16	G16	116	-	-	
ROAD_MGTREFCLK0_X0Y1_216	H18	G18	216	-	-	
ROAD_MGTREFCLK0_X1Y0_113	AG16	AH16	113	-	-	
ROAD_MGTREFCLK0_X0Y0_213	AG18	AH18	213	-	-	

1.4.2 IMPLEMENTED LOGIC

In figure 4 there is the architecture of the single link processing module (the same logic of the HIT chip).

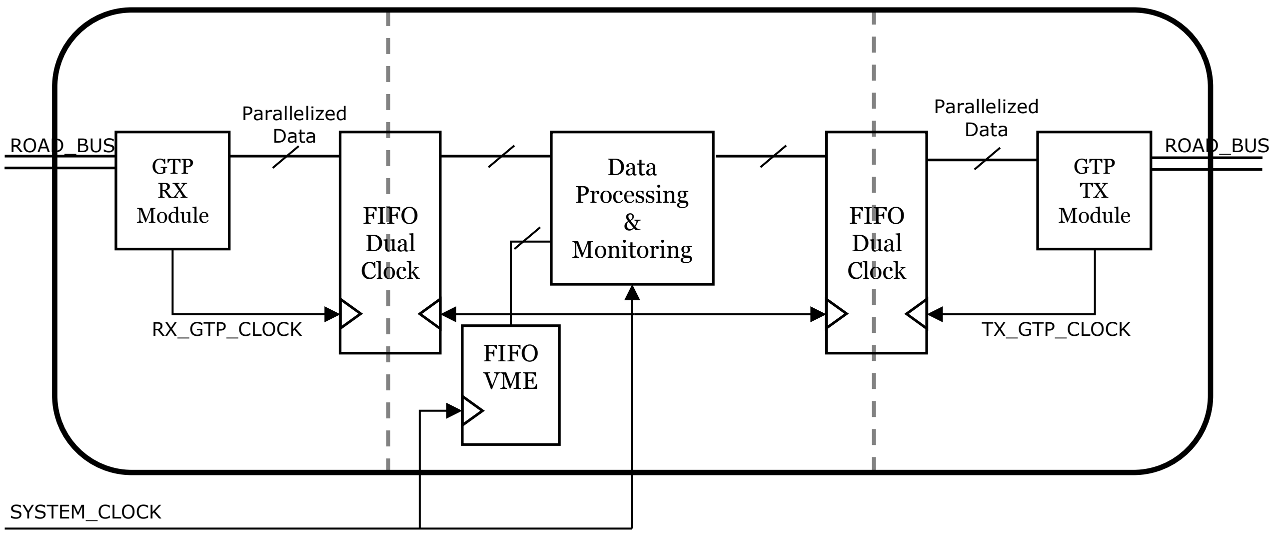


FIGURE 4

The input serial links are managed with the GTP Transceiver module. The 32 bit of the input buses are codified with the 8B/10B encoding, so the width of the buses became 40bit, and are transferred at 50MHz rate, so the frequency of the link is 2Gbp.

A simple communication protocol has been used between the AMchip and ROAD chip: **idle word** (BCBC1C1C) is sent when no data valid is present on the bus. After the AMchips receive the INIT event we have to wait a latency time of N clock cycles to have consecutive roads. The end event of roads is asserted by ROAD chip on each link when two conditions match: (1) N cycles have been counted since INIT event and (2) no more consecutive roads are received from the LAMB.

To summarize the data flow from AMChip to the ROAD chip is:

Type of word	**Value**	**K character**
IDLE WORD	BCBC1C1C	1111
ROAD DATA	BBAAAAAA	0000

where the BB bit field is the bitmap and AAAAAA bit field is the ROAD ID:

- Bits 18-0 of the AAAAAA field identifie the 512 k patterns inside one quartet of AMchips (see section 3 describing the LAMB mezzanine) on the LAMB. The 2 MSBs are the Geographical Address configured in the AMchip, that is the code that identifies each AMchip in the quartet.
- Bits 20-19 of the AAAAAA field identifie the 4 output links of the LAMB, that is the 4 quartets of chips on the LAMB.
- Bit 23 of the AAAAAA field is used inside the AUX board to flag the last road in the event;

The protocol between the ROAD chip and the AUX card is

Type of word	**Value**	**K character**
IDLE WORD	0000BC50	0010
ROAD DATA	BBAAAAAA	0000
END EVENT WORD	F7RRMMMM	1000

where RR are reserved bits for error codes, MMMM is the event ID.

1.4.3 ROAD VME REGISTERS

In the following table is the list of the internal registers of the ROAD chip

ADDR FPGA [26:2]	ADDR AMBFTK [31:0]	DESCRIPTION	N of bits	Type
0001000	XX004000	Fifo_RoadL0_link0	32	R/W
0001001	XX004004	Fifo_RoadL0_link1	32	R/W
0001002	XX004008	Fifo_RoadL0_link2	32	R/W
0001003	XX00400C	Fifo_RoadL0_link3	32	R/W
0001004	XX004010	Fifo_RoadL1_link0	32	R/W

0001005	XX004014	Fifo_RoadL1_link1	32	R/W
0001006	XX004018	Fifo_RoadL1_link2	32	R/W
0001007	XX00401C	Fifo_RoadL1_link3	32	R/W
0001008	XX004020	Fifo_RoadL2_link0	32	R/W
0001009	XX004024	Fifo_RoadL2_link1	32	R/W
000100A	XX004028	Fifo_RoadL2_link2	32	R/W
000100B	XX00402C	Fifo_RoadL2_link3	32	R/W
000100C	XX004030	Fifo_RoadL3_link0	32	R/W
000100D	XX004034	Fifo_RoadL3_link1	32	R/W
000100E	XX004038	Fifo_RoadL3_link2	32	R/W
000100F	XX00403C	Fifo_RoadL3_link3	32	R/W
0001010	XX004040	SPY_RoadL0_link0_STATUS	16=2+14	RO WR clear
0001011	XX004044	SPY_RoadL0_link1_STATUS	16=2+14	RO WR clear
0001012	XX004048	SPY_RoadL0_link2_STATUS	16=2+14	RO WR clear
0001013	XX00404C	SPY_RoadL0_link3_STATUS	16=2+14	RO WR clear
0001014	XX004050	SPY_RoadL1_link0_STATUS	16=2+14	RO WR clear
0001015	XX004054	SPY_RoadL1_link1_STATUS	16=2+14	RO WR clear
0001016	XX004058	SPY_RoadL1_link2_STATUS	16=2+14	RO WR clear
0001017	XX00405C	SPY_RoadL1_link3_STATUS	16=2+14	RO WR clear
0001018	XX004060	SPY_RoadL2_link0_STATUS	16=2+14	RO WR clear
0001019	XX004064	SPY_RoadL2_link1_STATUS	16=2+14	RO WR clear
000101A	XX004068	SPY_RoadL2_link2_STATUS	16=2+14	RO WR clear

000101B	XX00406C	SPY_RoadL2_link3_STATUS	16=2+14	RO WR clear
000101C	XX004070	SPY_RoadL3_link0_STATUS	16=2+14	RO WR clear
000101D	XX004074	SPY_RoadL3_link1_STATUS	16=2+14	RO WR clear
000101E	XX004078	SPY_RoadL3_link2_STATUS	16=2+14	RO WR clear
000101F	XX00407C	SPY_RoadL3_link3_STATUS	16=2+14	RO WR clear
0001020	XX004080	FF_REVENL_HF_FF_EF_flags	3 BITS x 8	RO
0001021	XX004084	ROAD_STATUS_REGISTER *Bit 0: tmode* *Bit 2-1 PllDetect* *Bit 7-3 date* *Bit 11-8 version* *Bit 15-12 month* *Bit 19-16 ResetDone* *Bit 28-20 RxByteIsAlign* *Bit 31:28: FSMState*	32bit	RO
0001022	XX004088	ROAD_CONTROL *Bit 15-0: SET operation* *Bit 31-16: CLEAR operation* *Bit 0: StartReadFifoVME* *Bit 1: LoopFifoVME* *Bit 3-2: Tmode*	1	RW
0001023	XX00408C	TIMER_SCT	32	RO
0001024	XX004090	ERROR_EVENL		RO WR clear
0001025	XX004094	ROAD_ALIGNMENT_STATUS *Bit 3-0: gt0_quad113* *Bit 7-4: gt0_quad116*	16	RO

		Bit 11-8: gt0_quad213 *Bit 15-12: gt0_quad216*		
0001026	XX004098	ROAD_RESETDONE_STATUS *Bit 3-0: tx_resetdone_quad113* *Bit 7-4: rx_resetdone_quad113* *Bit 11-8: tx_resetdone_quad116* *Bit 15-12: rx_resetdone_quad116* *Bit 19-16: tx_resetdone_quad213* *Bit 23-20: rx_resetdone_quad213* *Bit 27-24: tx_resetdone_quad216* *Bit 31-28: rx_resetdone_quad216*	16	RO
0001027	XX00409C	CONFIGURE *Bit 3-0: Board version* *Bit 6-4: PRBS pattern selection* *Bit 8: PRBS generator*	4	RW
0001028	XX0040A0	CONFIGURE_POLARITY *Bit 15-0: TX polarity* *213-216-116-113* *Bit 31-16: RX polarity*	32	RW
0001029	xx0040A4	KWORD MONITORING	32	RO
000102A	xx0040A8	PRBS ERROR QUAD 113	32	RO
000102B	xx0040AC	PRBS ERROR QUAD 116	32	RO
000102C	xx0040B0	PRBS ERROR QUAD 213	32	RO
000102D	xx0040B4	PRBS ERROR QUAD 216	32	RO
000102E	xx0040B8	DEBUG_DISABLE_HOLD_AUX	16	RW
000102F	xx0040BC	HOLD_AUX_STATUS	16	RO
000102E 00017FF	XX0040C0 XX005FFC			

In the following table is the list of the ROAD memories

ADDR FPGA [26:2]	ADDR AMBFTK [31:0]	DESCRIPTION	N of bits	Type

S: 40000 E: 41FFF	S: XX100000 E:XX107FFC	OSPY_L0_link0	32	RO
S: 42000 E: 43FFF	S: XX108000 E:XX10FFFC	OSPY_L0_link1	32	RO
S: 44000 E: 45FFF	S: XX110000 E:XX117FFC	OSPY_L0_link2	32	RO
S: 46000 E: 47FFF	S: XX118000 E:XX11FFFC	OSPY_L0_link3	32	RO
S: 48000 E: 49FFF	S: XX120000 E:XX127FFC	OSPY_L1_link0	32	RO
S: 4A000 E: 4BFFF	S: XX128000 E:XX12FFFC	OSPY_L1_link1	32	RO
S: 4C000 E: 4DFFF	S: XX130000 E:XX137FFC	OSPY_L1_link2	32	RO
S: 4E000 E: 4FFFF	S: XX138000 E:XX13FFFC	OSPY_L1_link3	32	RO
S: 50000 E: 51FFF	S: XX140000 E:XX147FFC	OSPY_L2_link0	32	RO
S: 52000 E: 53FFF	S: XX148000 E:XX14FFFC	OSPY_L2_link1	32	RO
S: 54000 E: 55FFF	S: XX150000 E:XX157FFC	OSPY_L2_link2	32	RO
S: 56000 E: 57FFF	S: XX158000 E:XX15FFFC	OSPY_L2_link3	32	RO
S: 58000 E: 59FFF	S: XX160000 E:XX167FFC	OSPY_L3_link0	32	RO
S: 5A000 E: 5BFFF	S: XX168000 E:XX16FFFC	OSPY_L3_link1	32	RO
S: 5C000 E: 5DFFF	S: XX170000 E:XX177FFC	OSPY_L3_link2	32	RO
S: 5E000 E: 5FFFF	S: XX178000 E:XX17FFFC	OSPY_L3_link3	32	RO

2 AMBSLP JTAG CHAIN

2.1 AMBSLP V1

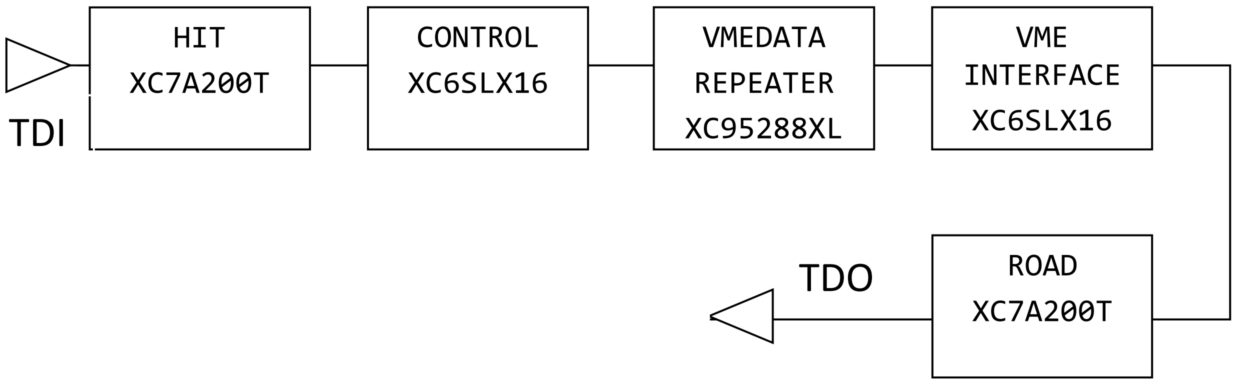


FIGURE 5

2.2 AMBSLP V2

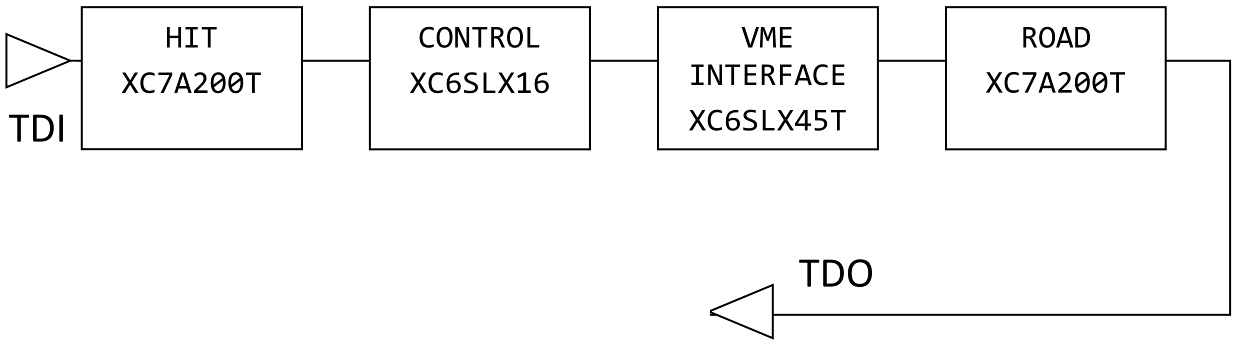


FIGURE 6

SPI FLASH ROAD & HIT: N25Q128A   Package: xxxxx
SPI FLASH VME & CTRL: AT45DB321D   Package: xxxxx

3 LAMB CONFIGURATION

Four LAMBs are mounted on an AM board, and each LAMB has a BOUSCA chip (blue box in figure 7) for JTAG configuration. Figure 7, shows the organization of the 16 AMchips into 8 JTAG chains on a LAMB. In this section a description of the picture in figure 7 is presented.

3.1 LAMB CHAIN CONFIGURATION

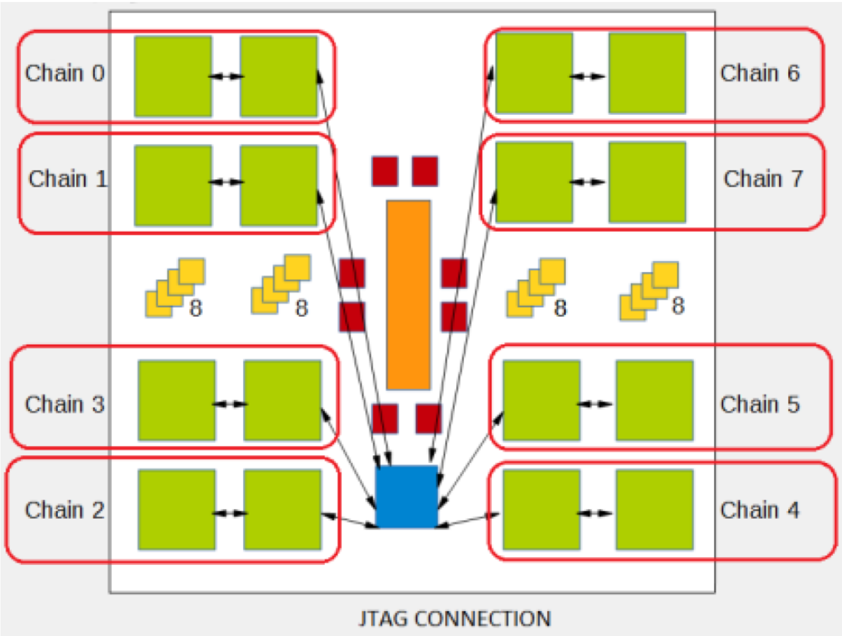


FIGURE 7

3.2 JTAG CONNECTION CHAIN

The JTAG connection on an AM board is made of 32 chains, and each chain contains 2 AMchips. One chain requires one data bit, thus a total of 32 chains can be accessed in parrales exploiting the 32 bit of the VMEDATA bus. On each LAMB there are eight chains as shown on the right of Figure 7. The corresponding VME registers are: TMS_BSCAN, TDI_BSCAN, TDO_BSCAN, and an eight bit slice of these registers is allocated to each of the four BOUSCA chips. The task of the VME chip is to generate the address signals for them. Four LAMBs are accessed in parallel. For example with one VME operation the 32 TDI signals, which are distributed to eight chains on all the four LAMBs, are updated. The VME data bus is split in four slices, each consists of eight bits, between the four LAMBs as shown in Table 1. Each one bit in a slice is allocated to one of the JTAG chains on a LAMB as shown in Table 2.

VMEDATA[32:24]	VMEDATA[23:16]	VMEDATA[15:8]	VMEDATA[7:0]
LAMB0	LAMB1	LAMB2	LAMB3

TABLE 1

VMEDATA bit	[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
Chain	7	6	5	4	3	2	1	0

TABLE 2

Table 3 and 4 summarise the JTAG registers and JTAG operations of AMchip05. The IR values of the registers, and their width is described in Table 3. Table 4 lists the IR values for JTAG operations, which does not access a particular register but trigger a specific action on the chip.

IR value	Width	Name	Description	Type
0xFF	1	BYPASS	Bypass	RW
0x01	32	IDCODE	ID Code (0x50004071 for AMchip05)	R
0xC5	145	JPATT_DATA	Pattern data	RW
0xE5	145	JPATT_DATA	Read back pattern data	R
0xC4	16	JPATT_ADDR	Pattern address	RW
0xE4	16	JPATT_ADDR	Read back pattern address	R
0xC6	97	JPATT_CTRL	Pattern bank configuration	RW
0xE6	97	JPATT_CTRL	Read back pattern bank configuration	R
0xE8	25	REC_ADDRESS	Bank output status	R
0xC9	7	SERDES_SEL	Select target SER/DES register	RW
0xCA	32	SERDES_REG	Write register selected by SERDES_SEL	RW
0xCB	32	IDLE_CFG	Idle output configuration	RW
0xEA	32	SERDES_STAT_CFG	Read information selected by SERDES_SEL	R
0xED	32	CRC_REG	Reads output stream CRC32	R
0xCD	42	PATT_TEST_REG	Internal pattern test configuration	R

TABLE 3

IR value	Name	Description
0xD4	OP_WRITE_INC	Write pattern and increment JTAG_ADDR
0xD5	OP_SEL_BANK	Select next pattern from bank
0xD6	OP_INIT_EV	Init event

TABLE 4

4 VME FUNCTIONS AND COMMANDS

4.1 JTAG LOW LEVEL COMMANDS

Jtag Commands are included in the files ambslp.cxx, ambslp_jtag_func.cxx, ambslp_VME_jtag_func.cxx

4.2 ONE EXAMPLE OF STUCTURED PROCEDURE: STANDALONE AMBOARD LOOP FUNCTION

This section describes an example of procedure that can be run on the AMBSLP as a standalone board.

We describe the vme commands to configure the AMBSLP and how to run the commands to enable in the event internal loop mode.

Loop of events using the LAMBs and AM05 chips

- **initialization_ambslp.sh** reset procedure
- **ambslp_amchip_8b_10b.cxx** setup of links. The bus0 clock (recovered) used for input logic; out clock used for majority and readout;
- **ambslp_status.cxx** prints on screen the status of all input/output links to identify if all of them are correctly aligned;
- **reset_gtp_road.sh** reset of Road GTPs – usually if problems are found they are in Road, so please use this fuction to reset again the GTPs;
- **ambslp_amchip_jpatt_cfg.cxx** configure the AMchips, like the geographical addresses of the chips, the patflow status, the testmode, the majority THR, the pattern enable/disable control…. We use this function only to change the parameters that need to be used differently from the default, in this case they are: (a) THR=15; (b) testmode=1; (c) disable pattflow (=1). This is important to do before writing patterns.
- **ambslp_amchip_init_evt.cxx** we give an INIT so that the parameters set before becomes active in AMchip.
- **ambslp_amchip_disable_bank.cxx** we configure the chip to disable all the patterns turning off the match in the majority.
- **ambslp_amchip_init_evt.cxx** we give an INIT so that the patterns are really disabled.
-
- **ambslp_amchip_enable_bank.cxx** we enable the bunch of patterns we want to use. We can enable a number of patterns set by the parameter "npatt" starting from the address "offset". By default they are all the AM05 patterns (~2000) staring from offset 0.
- **ambslp_amchip_init_evt.cxx** we give an INIT so that the patterns are really enabled.
- **ambslp_amchip_jpatt_cfg.cxx** configure the AMchips to start running: (a) tmode=0; (b) THR=0 (or whatever is preferred) (c) pattflow=0;
- **ambslp_amchip_init_evt.cxx** we give an INIT so that the parameters set before becomes active in AMchip.
- **ambslp_feed_hit.cxx** reads hitfile from disk and download hits on input fifos.
- **ambslp_out_spy.cxx** reads spy buffers and print the content. With this is possible to check if the expected roads really came out. If the THR is zero, all the enabled patterns should be readable.

Loop of fake events without LAMBs and AM05 chips, but using simulated Roads preloaded on output fifos. Probably the description is **obsolete**.

The first operation is a reset procedure:to set-up the configuration registers of the AMBSLP board and to send a reset. The command are executed by *ambslp_init_main*, *ambslp_config_reg_main* and *ambslp_reset_spy_main*. The option --help after each command give a list of all input parameters (ambslp_init_main --help).

- ex: **ambslp_config_reg_main --slot 15 --fixpolarity 1 --boardver 0** #Configure the register and fix the polarity of the link from/to AUX; select the board version
  - –boardver 0 #configuration for the AMBSLP v1
  - –boardver 1 #configuration for the AMBSLP v2
- **ambslp_init_main --slot 15** #send a reset pulse to the AMBSLP
- **ambslp_reset_spy_main --slot 15** #Resetting the input and output spybuffer

[OPTIONAL] In case that you have a real pattern bank file this command is useless.

To generate a random pattern bank, the command *ambslp_patt_gen_main* defines the number of patterns that compose the pattern bank

- Ex: **ambslp_patt_gen_main --npatt 500 exmple_patternbank.patt**

After generating the random pattern bank or starting from a real pattern bank file, the *ambslp_gen_hits_main* command generates a file with the hit of events (the option ambslp_gen_hits_main --help give a description of all input parameters)

- Ex: **ambslp_gen_hits_main -e 3 -r 10 --rs example_patternbank.patt > example_hit.hit**

[OPTIONAL] In case that you have a stream of roads coming from AMchips this command is useless

To generate random roads to be sent to the AUX card, the command * ambslp_gen_roads_main* creates a file with roads and end event.

- Ex: **ambslp_gen_roads_main --nroad 10 --nevent 3 example_road.road**

After generating the Hit file and the road file, the command *ambslp_feed_hit_main* and *ambslp_feed_road_main* enable the loading of the VME fifos

- Ex: **ambslp_feed_hit_main --slot 15 -- loopfifovme 0 example_hit.hit** #loading fifos and sending hits without loop; you can load several input files

- Ex: **ambslp_feed_hit_main --slot 15 -- loopfifovme 1 example_hit.hit** #loading fifos and sending hits with loop; to stop the loop you have to send the ambslp_init_main command

- **ambslp_feed_road_main --slot 15 --loopfifovme 0 example_hit.hit** #loading fifos and sending roads without loop; you can load several input files

- **ambslp_feed_road_main --slot 15 -- loopfifovme 1 example_hit.hit** #loading fifos and sending roads with loop; to stop the loop you have to send the ambslp_init_main command

The command *ambslp_status_main* monitors some internal registers of the AMBSLP FPGAs such as: the alignment, the reset done of the gtp, the counter of loss of synchronism, the PRBS error links, the hold status signal.

- ex: **ambslp_status_main --slot 15**

The command *ambslp_inp_spy_main* and *ambslp_out_spy_main* dump the content of the 12 input spybuffer and 16 output spybuffer with the status register associated to each links

- ex: **ambslp_inp_spy_main --slot 15**
- ex: **ambslp_out_spy_main --slot 15**

Usefull links

- **https://twiki.cern.ch/twiki/bin/viewauth/Atlas/FastTrackerHardwareDocumentation**

- **svn+ssh://svn.cern.ch/reps/atlasgroups/Trigger/FastTracker/Docs/Specs** #directory with the documentation

- **svn+ssh://svn.cern.ch/reps/atlasftkfw/AMboard** #directory with the AMBSLP firmware

4.2.1 Simulation of the AM bank to produce roads

This section describes the commands to simulate the expected roads from the AMchips, given a certain pattern bank, in order to compare with the road coming out from the hardware

The *ambslp_expected_raod_DC* takes two files in input, the pattern bank file and the hit file, and performs the simulation to find the expected roads. The command has the possibility to set the main following parameters: the threshold of the matched patterns, the bitmap enable and the number of used Don't Care bits

- Ex: **ambslp_expected_road_DC_main --thr 7 --laymap 1 --sort 1 --eebit 15 PattBank.txt HitFile.txt** # the output of the simulation is a file with the list of matched roads and end event words.

The list of matched roads from the AMchip can be obtained dumping the out spybuffers. The comparison between this list and the HW output is performed by the *ambslp_road_diff* script.

- Ex: **ambslp_road_diff_main matched_road_simulation.txt matched_road_hardware.txt** # the output is the result of the comparison between the two files.