Spartan-6 Muon Port Card Mezzanine Firmware and Software Manual

Rice University

Version 2.0

26 May, 2020

Introduction

This document describes the updated firmware and software for the Spartan-6 Muon Port Card (MPC) mezzanine board for the CSC System at CMS. The hardware parts (the baseboard MPC and the mezzanine card) remain unchanged; their descriptions can be found in [1] and [2]. The Spartan-6 FPGA residing on the mezzanine supports 8 optical channels for data transmission from nine Trigger Motherboards (TMB) residing in the peripheral Endcap Muon (EMU) crates to the Endcap Muon Track Finder (EMTF). The previous design was based on industry standard 8B/10B encoding scheme for all optical links. The current design is using a more data efficient 38B/40B decoding allowing to compress all the info from 9 TMBs into 8 optical links operating at 3.2Gbps rate.

1. Interface to Trigger Motherboards and Sorter Unit

Every 25 ns the TMB can send to MPC up to two Local Charged Tracks (LCT). Each LCT is represented by 32 bits that are transmitted in two frames at 80 MHz. The frame format is shown in Table 1. LCTs from any TMB (or from any FIFO_A) can be masked out (disabled) using the CSR7 register. This register is implemented in the discrete logic; it can be read out via VME through the FPGA only. When disabled (CSR7(i)=0, i=0..8), the selected TMB(i) inputs are ignored. By default (after power cycling) CSR7(15:0)="1", so all the TMBs are enabled.

The original MPC2004 board performed sorting of up to 18 incoming LCT patterns based on 4-bit "quality", selection of three "best" ones (with the highest "quality") and transmission them in ranked order to the three TLK2501 serializers residing on the MPC baseboard. This functionality is not needed any more, but we keep the FIFO_A and FIFO_B buffers to be able to test the TMB interface with the MPC.

LCTs with "Quality"=0 and "vpf"=0 are cancelled by the sorter unit. LCTs with "quality=0" and "vpf=1" do participate in sorting. If several patterns happen to have the same valid "Quality" value, then the pattern arriving from the TMB with the highest slot number in the crate will have the precedence. If two LCT from the same TMB are selected and both have the same "quality" value, then the LCT0 will have the precedence over LCT1. Selected 32-bit patterns are multiplexed into two 16-bit frames on the FPGA outputs and sent directly to TLK2501 serializers in ranked order.

Table 1

TMB-to-MPC data Format (i=1..9)

		First frame transmitted at 80Mhz		Second frame transmitted at 80Mhz	
Line	**Old Name**	**Signal**	**LCT**	**Signal**	**LCT**
TMBi[0]	**Lcti_vpf**	**Wire Group_0**	**0**	**½-strip_0**	**0**
TMBi[1]	**Lcti_qual0**	**Wire Group_1**	**0**	**½-strip_1**	**0**
TMBi[2]	**Lcti_qual1**	**Wire Group_2**	**0**	**½-strip_2**	**0**
TMBi[3]	**Lcti_qual2**	**Wire Group_3**	**0**	**½-strip_3**	**0**
TMBi[4]	**Lcti_qual3**	**Wire Group_4**	**0**	**½-strip_4**	**0**
TMBi[5]	**Lcti_qual4**	**Wire Group_5**	**0**	**½-strip_5**	**0**
TMBi[6]	**Lcti_qual5**	**Wire Group_6**	**0**	**½-strip_6**	**0**
TMBi[7]	**Lcti_qual6**	**CLCT Pattern_ID0**	**0**	**½-strip_7**	**0**
TMBi[8]	**Lcti_qual7**	**CLCT Pattern_ID1**	**0**	**L/R Bend Angle**	**0**
TMBi[9]	**Lcti_qual8**	**CLCT Pattern_ID2**	**0**	**SYNC_ER**	**0**
TMBi[10]	**Lcti_hs0**	**CLCT Pattern_ID3**	**0**	**BXN[0]**	**0**
TMBi[11]	**Lcti_hs1**	**Quality_0**	**0**	**BC0**	**0**
TMBi[12]	**Lcti_hs2**	**Quality_1**	**0**	**CSC_ID0**	**0**
TMBi[13]	**Lcti_hs3**	**Quality_2**	**0**	**CSC_ID1**	**0**
TMBi[14]	**Lcti_hs4**	**Quality_3**	**0**	**CSC_ID2**	**0**
TMBi[15]	**Lcti_hs5**	**Valid Pattern Flag**	**0**	**CSC_ID3**	**0**
TMBi[16]	**Lcti_hs6**	**Wire Group_0**	**1**	**½-strip_0**	**1**
TMBi[17]	**Lcti_hs7**	**Wire Group_1**	**1**	**½-strip_1**	**1**
TMBi[18]	**Lcti_wg0**	**Wire Group_2**	**1**	**½-strip_2**	**1**
TMBi[19]	**Lcti_wg1**	**Wire Group_3**	**1**	**½-strip_3**	**1**
TMBi[20]	**Lcti_wg2**	**Wire Group_4**	**1**	**½-strip_4**	**1**
TMBi[21]	**Lcti_wg3**	**Wire Group_5**	**1**	**½-strip_5**	**1**
TMBi[22]	**Lcti_wg4**	**Wire Group_6**	**1**	**½-strip_6**	**1**
TMBi[23]	**Lcti_wg5**	**CLCT Pattern_ID0**	**1**	**½-strip_7**	**1**
TMBi[24]	**Lcti_wg6**	**CLCT Pattern_ID1**	**1**	**L/R Bend Angle**	**1**
TMBi[25]	**Lcti_accmu**	**CLCT Pattern_ID2**	**1**	**SYNC_ER**	**1**
TMBi[26]	**Lcti_bx0**	**CLCT Pattern_ID3**	**1**	**BXN[0]**	**1**
TMBi[27]	**Lcti_bx1**	**Quality_0**	**1**	**BC0**	**1**
TMBi[28]	**Lcti_rsv0**	**Quality_1**	**1**	**CSC_ID0**	**1**
TMBi[29]	**Lcti_rsv1**	**Quality_2**	**1**	**CSC_ID1**	**1**
TMBi[30]	**Lcti_rsv2**	**Quality_3**	**1**	**CSC_ID2**	**1**
TMBi[31]	**Lcti_rsv3**	**Valid Pattern Flag**	**1**	**CSC_ID3**	**1**

2. Serialization and Interface to the Endcap Muon Track Finder (EMTF)

The ser40a serializer has been created in the Xilinx ISE 14.7 Core Generator with the following options:

- Reference clock: 160MHz (originating from the QPLL ASIC on the mezzanine board)
- Encoding: none
- Line rate: 3.2Gbps
- Data path: 40 bit
- TX buffer: enabled
- USERCLK=320MHz, USERCLK2=80MHz; both clocks originate from the fabric 40MHz clock (sourced by the CCB 40Mhz clock)

There are eight ser40a serializers in the project utilizing all 4 GTP tiles in the Spartan-6 FPGA. Custom mpcx_tx module (Appendix A) applies 38B/40B encoding scheme to the TMB path and provides inputs to 8 ser40a serializers. 38 bits are assigned to MPC inputs and

two are used for alignment header in each frame (Fig.1, based on presentation [3]). The total bandwidth per link per bunch crossing is increasing from 64 bits (8B/10B) to 76 bits (38B/40B). The scrambler (also in Appendix A) randomizes input data without adding bits to provide DC balance for the serial link. On the receiver end the alignment logic finds the frame edge using header transmitted with data. 12-channel Avago AFBR-810B optical transmitter (4 channels are not used) is connected to the EMTF receiver.

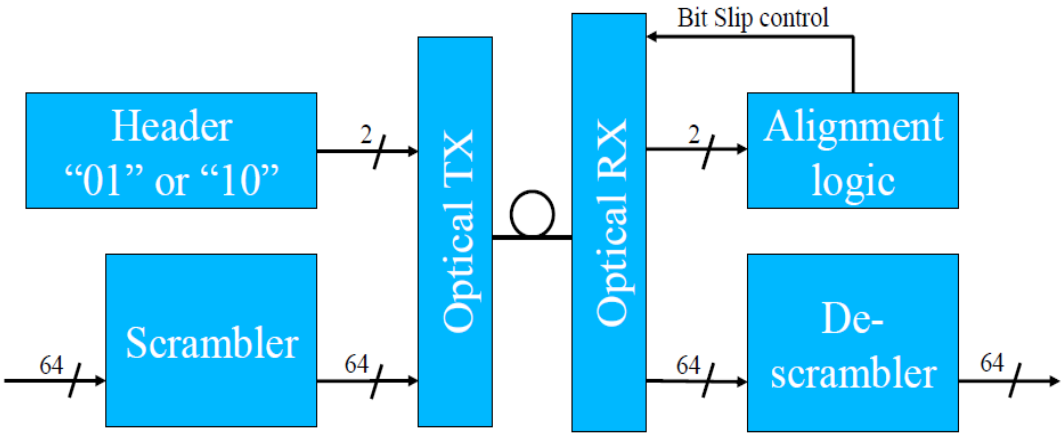

Figure 1: Block diagram of the 38B/40B encoding scheme

3. FIFO Buffers

Two groups of FIFO Buffers (FIFO_A and FIFO_B) are implemented in the main FPGA in order to test the MPC internal functionality and its communications with the Trigger Motherboards. 3 FIFO_B keep the results of sorting for three old optical links. All buffers are 511-word deep and available from VME for read and write. Since two muon patterns are packed into FIFO_A in two frames at 80MHz, each FIFO effectively comprises 255 patterns. Each buffer represents data corresponding to one TMB board, or two muon patterns. FIFO_A1 corresponds to TMB1, FIFO_A2 corresponds to TMB2 and so on. Its format is shown in Table 2. In a "Test" mode the test patterns representing 18 muons are sent out simultaneously from all FIFO_A buffers at 80Mhz upon specific TTC or VME commands. They pass through the sorting unit that selects the 3 best patterns and transmits them to the TLK2501 serializers and FIFO_B buffers. FIFO_B format is shown in Table 3.

One important feature of all FIFO_B buffers is that the data from FIFO_A ("Test" mode) or TMB's ("Trigger" mode) can be saved in FIFO_B only if there is at least one valid muon pattern, or pattern with "vpf"=1. The "vpf" bit from the first best selected muon acts as a "write enable" signal for all FIFO_B buffers. This means that if there is just one valid pattern coming after sorting from FIFO_A or TMB, it will be stored in FIFO_B1 and "0" will be written into FIFO_B[2..3]. This mechanism assures that all three FIFO_B buffers will contain an equal number of words inside. As for VME access, any data can be loaded and read back out of any FIFO_B buffer independently. The FULL and EMPTY flags (common to all FIFO_A and FIFO_B buffers) are available for read from CSR2. After an asynchronous FIFO reset all "EMPTY" flags are active "1" and "FULL" flags are "0".

Table 2

FIFO_A Data Format

FIFO_A Frame 1 (LCT1)															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
vpf	**Quality[3..0]**				**CLCT Pattern[3..0]**				**Wire Group ID[6..0]**						
FIFO_A Frame 1 (LCT0)															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
vpf	**Quality[3..0]**				**CLCT Pattern[3..0]**				**Wire Group ID[6..0]**						
FIFO_A Frame 2 (LCT1)															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSC_ID[3..0]				**Bc0**	**Bx0**	**ER**	**L/R**	**CLCT Half-strip ID[7..0]**							
FIFO_A Frame 2 (LCT0)															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CSC_ID[3..0]				**Bc0**	**Bx0**	**ER**	**L/R**	**CLCT Half-strip ID[7..0]**							

Table 3

FIFO_B Data Format (80MHz)

FIFO_B Frame 1															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Vpf	**Quality[3..0]**				**CLCT Pattern ID[3..0]**				**Wire Group ID[6..0]**						
FIFO_B Frame 2															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSC_ID[3..0]				**Bc0**	**Bx0**	**ER**	**L/R**	**CLCT Half-strip Pattern_ID[7..0]**							

CLCT Half-Strip Pattern ID is between 0 and 159
CLCT Pattern encodes the number of layers and whether the pattern consists of half-strips or di-strips
Wire Group ID is between 0 and 111 and indicates the position of the pattern within the chamber
L/R – Bend Angle Bit indicates whether the track is heading towards lower or higher strip number
VPF – Valid Pattern Flag indicates a valid LCT that has been found by TMB and being sent in the current clock cycle
ER - Synchronization Error
BX0 – The less significant bits of the Bunch Crossing Counter.
BC0 – Bunch Crossing Zero Flag arriving from the TMB

4. VME Interface

The MPC can be accessed in the VME crate using geographical addressing that utilizes the address pins GA<4-0> available on the VME64x backplane. The MPC resides in slot 12 of the peripheral EMU backplane, so its base geographical address is 600000(hex). The board responds to AM codes 39(hex), 3D(hex) and supports A24D16 slave operations. It does not respond to byte-addressing modes, so all valid addresses must be even numbers. Decoded addresses and VME commands are listed in Table 4.

The main part of the VME interface (data drivers, address latches, address comparators, DACK response logic, CSR0) is implemented in discrete logic. This means that the board is accessible even without mezzanine card that carries the Xilinx FPGA. The registers marked red have changed vs the previous (2013) revision of firmware.

Table 4

Address (hex)	Access	Function
600000	Read/Write	CSR0 (implemented on the main MPC2004 board)
600002	Write	Hard_Reset command to FPGA (decoded on the main MPC board)
600004	Write	Soft_Reset command to FPGA (decoded on the main MPC board)
600006	Write	CSR7 (TMB mask register). Read through the FPGA, address 6000CA
600080 (0)	Read/Write	FIFO_A1[15..0]. Corresponds to TMB1_LCT1
600082 (1)	Read/Write	FIFO_A1[31..16]. Corresponds to TMB1_LCT0
600084 (2)	Read/Write	FIFO_A2[15..0]. Corresponds to TMB2_LCT1
600086 (3)	Read/Write	FIFO_A2[31..16]. Corresponds to TMB2_LCT0
600088 (4)	Read/Write	FIFO_A3[15..0]. Corresponds to TMB3_LCT1
60008A (5)	Read/Write	FIFO_A3[31..16]. Corresponds to TMB3_LCT0
60008C (6)	Read/Write	FIFO_A4[15..0]. Corresponds to TMB4_LCT1
60008E (7)	Read/Write	FIFO_A4[31..16]. Corresponds to TMB4_LCT0
600090 (8)	Read/Write	FIFO_A5[15..0]. Corresponds to TMB5_LCT1
600092 (9)	Read/Write	FIFO_A5[31..16]. Corresponds to TMB5_LCT0
600094 (10)	Read/Write	FIFO_A6[15..0]. Corresponds to TMB6_LCT1
600096 (11)	Read/Write	FIFO_A6[31..16]. Corresponds to TMB6_LCT0
600098 (12)	Read/Write	FIFO_A7[15..0]. Corresponds to TMB7_LCT1
60009A (13)	Read/Write	FIFO_A7[31..16]. Corresponds to TMB7_LCT0
60009C (14)	Read/Write	FIFO_A8[15..0]. Corresponds to TMB8_LCT1
60009E (15)	Read/Write	FIFO_A8[31..16]. Corresponds to TMB8_LCT0
6000A0 (16)	Read/Write	FIFO_A9[15..0]. Corresponds to TMB9_LCT1
6000A2 (17)	Read/Write	FIFO_A9[31..16]. Corresponds to TMB9_LCT0
6000A4 (18)	Read/Write	FIFO_B1[15..0]. Corresponds to 1st best selected LCT
6000A6 (19)	Read/Write	FIFO_B2[15..0]. Corresponds to 2nd best selected LCT
6000A8 (20)	Read/Write	FIFO_B3[15..0]. Corresponds to 3rd best selected LCT
6000AA (21)	Read	CSR1 (date of the current firmware version)
6000AC (22)	Read/Write	CSR2 (control)
6000AE (23)	Read	CSR3 (FIFO Status)
6000B0 (24)	Read	L1ACC Counter
6000B2 (25)	Write	Transmit 511 words of data from all FIFO_A buffers in “Test” mode
6000B2 (25)	Read	CSR9 (various board statuses)
6000B4 (26)	Read/Write	N/A
6000B6 (27)	Write	Send a 3.2 us TxEn “0” pulse to all three TLK2501 transmitters
6000B8 (28)	Read/Write	N/A
6000BA (29)	Read/Write	N/A
6000BC (30)	Read	CSR6 (access to DS2401 serial number and AFBR-810 registers)
6000BE (31)	Read	N/A
6000C0 (32)	Write	Generate 800 us “Reset” pulse on a 1-Wire bus to initialize the DS2401
6000C2 (33)	Write	Generate 2 us “Read” pulse on a 1-Wire bus to read data from DS2401
6000C4 (34)	Write	Reset CSR6
6000C6 (35)	Write	Generate “Write-zero” 50 us pulse on a 1-Wire bus to DS2401
6000C8 (36)	Write	Generate “Write-one” 12 us pulse on a 1-Wire bus to DS2401
6000CA (37)	Read	CSR7 (readable through the FPGA only)
6000CC (38)	Write	CSR11 (reset various sources, see section 8.10 below)
6000CE (39)	Read	CSR8 (GTP statuses)
6000D0 (40)	Write/Read	N/A
6000D2 (41)	Read	N/A
6000D4 (42)	Read	N/A
6000D6 (43)	Read	N/A
6000D8 (44)	Read	N/A
6000DA (45)	Read	N/A

6000DC (46)	Read	N/A
6000DE (47)	Read	N/A
6000E0 (48)	Read	N/A
6000E2 (49)	Read	N/A
6000E4 (50)	Read/Write	N/A
6000E6 (51)	Read/Write	N/A
6000E8 (52)	Read/Write	N/A
6000EA (53)	Read/Write	N/A
6000EC (54)	Read/Write	N/A
6000EE (55)	Read/Write	N/A
6000F0 (56)	Read/Write	N/A
6000F2 (57)	Read/Write	N/A
6000F4 (58)	Write	N/A
6000F6 (59)	Write	N/A
6000F8 (60)	Write	N/A
6000FA (61)	R/W	
6000FC (62)	R/W	
6000FE (63)	R/W	
6000FE (63)	R/W	

5. Control and Status Registers

The 16-bit CSR0 is implemented in a discrete logic on the main MPC board and is available for write and read over VME even if the mezzanine card is not installed. Bit assignment is shown in Table 5. The other registers are implemented inside the FPGA (see Tables 6-13).

5.1. CSR0 (600000h, discrete logic, main MPC2004 board)

Table 5

Bit and access	Function
0 (R/W)	FPGA_Mode (Trigger mode if "0", Test mode (FIFO_A is a source of data) if "1")
1(R/W)	Board_ID[0]
2 (R/W)	Board_ID[1]
3 (R/W)	Board_ID[2]
4 (R/W)	Board_ID[3]
5 (R/W)	TDI (JTAG signal for FPGA/EPROM access)
6 (R/W)	TMS (JTAG signal for FPGA/EPROM access)
7 (R/W)	TCK (JTAG signal for FPGA/EPROM access)
8 (R)	TDO (JTAG signal for FPGA/EPROM access)
9 (R/W)	TxEn signal for TLK2501 transmitters ("1" for normal data transfer mode)
10 (R/W)	Board_ID[4]
11 (R/W)	Board_ID[5]
12 (R)	FPGA Configuration Done (read only, active "1")
13 (R/W)	If "0", the input FPGA clock should be adjusted with the CSR2[15..8] If "1", the input FPGA clock is adjusted automatically in the middle of the "safe window"
14 (R/W)	Enable TLK2501 serializers when "1". If "0", all TLK2501 are in power-down mode
15 (R/W)	PRBSEN (Enable PRBS test mode for all TLK2501 serializers when "1")

5.2. CSR1 (6000AAh, FPGA, contains the date of the firmware version)

Table 6

Bit and access	Function
0 (R)	Day, LSB
1(R)	Day
2 (R)	Day
3 (R)	Day
4 (R)	Day, MSB
5 (R)	Month, LSB
6 (R)	Month
7 (R)	Month
8 (R)	Month, MSB
9 (R)	Year, LSB (*)
10 (R)	Year
11 (R)	Year
12 (R)	Year, MSB (*)
13 (R)	"0"
14 (R)	"0"
15 (R)	"0"

(*) The code at CSR1<11..9> should be added to 2000 to get an actual year. For example, CSR1=1507(hex) corresponds to August 7, 2010.

5.3. CSR2 (6000ACh, FPGA, control register, by default set to "0")

Table 7

Bit and access	Function
0 (R/W)	When "0", all TLK2501 transmitters are in "Normal data character" mode. When "1", all TLK2501 are in IDLE mode, unless there is a Valid Pattern or BC0 from TMB
1(R/W)	Not used
2 (R/W)	Allows to increase (when "1") and decrease (when "0") the phase shift of the 160Mhz and 320Mhz clocks by writing any data "n" times to address 6000F6h.
3 (R/W)	SDA input/output of the AFBR-810 optical transmitter
4 (R/W)	SCL input of the AFBR-810 optical transmitter
5 (R/W)	Enable read from AFBR-810 (when "1") via CSR6[15]
6 (R/W)	Not used
7 (R/W)	Controls the enp_vme input of mpcx_tx module
8 (R/W)	Delay of the 40Mhz input clock for the FPGA, LSB (*)
9 (R/W)	Delay of the 40Mhz input clock for the FPGA (*)
10 (R/W)	Delay of the 40Mhz input clock for the FPGA (*)
11 (R/W)	Delay of the 40Mhz input clock for the FPGA (*)
12 (R/W)	Delay of the 40Mhz input clock for the FPGA (*)
13 (R/W)	Delay of the 40Mhz input clock for the FPGA (*)
14 (R/W)	Delay of the 40Mhz input clock for the FPGA (*)
15 (R/W)	Delay of the 40Mhz input clock for the FPGA, MSB (*)

(*) - 1 step = 0.25 ns. Implemented using 3D7408-0.25 programmable delay chip from Data Delay Devices

5.4. CSR3 (6000AEh, FPGA, FIFO status register)

Table 8

Bit and access	Function
0 (R)	FIFO_A FULL. Active “1” if at least one out of nine FIFO_A buffers is full. “0” after reset.
1(R)	FIFO_A EMPTY. Active “1” if ALL nine FIFO_A buffers are empty. Also “1” after reset
2 (R)	FIFO_B FULL. Active “1” if at least one out of three FIFO_B buffers is full. “0” after reset.
3 (R)	FIFO_B EMPTY. Active “1” if ALL three FIFO_B buffers are empty. Also “1” after reset
4 (R)	“0”
5 (R)	“0”
6 (R)	“0”
7 (R)	“0”
8 (R)	“0”
9 (R)	“0”
10 (R)	“0”
11 (R)	“0”
12 (R)	“0”
13 (R)	“0”
14 (R)	“0”
15 (R)	“0”

5.5. CSR6 (6000BCh, access to DS2401 serial number)

Table 9

Bit	Access	Function
0	R	“0” indicates the “Presence” pulse from the DS2401 after the “Reset” pulse
1	R	Data bit from DS2401
2	R	Status of the initialization. When “1” after the “Reset” pulse, the CSR6[0] is valid.
3	R	Status of the read cycle. When “1” after the “Read” pulse, the CSR6[1] is valid.
4	R	Status of the command cycle. When “1” after the “Write-one” or “Write-zero” command, the next command can be sent
5	R	“0”
6	R	“0”
7	R	“0”
8	R	“0”
9	R	“0”
10	R	“0”
11	R	“0”
12	R	“0”
13	R	“0”
14	R	“0”
15	R	SDA line from AFBR-810 for read

5.6. CSR7 (6000CAh, discrete logic; readable through the FPGA)

Table 10

Bit	Access	Function
0	R/W	Enable (if « 1 ») or Disable (if « 0 ») TMB1
1	R/W	Enable (if « 1 ») or Disable (if « 0 ») TMB2
2	R/W	Enable (if « 1 ») or Disable (if « 0 ») TMB3
3	R/W	Enable (if « 1 ») or Disable (if « 0 ») TMB4
4	R/W	Enable (if « 1 ») or Disable (if « 0 ») TMB5
5	R/W	Enable (if « 1 ») or Disable (if « 0 ») TMB6
6	R/W	Enable (if « 1 ») or Disable (if « 0 ») TMB7
7	R/W	Enable (if « 1 ») or Disable (if « 0 ») TMB8

8	R/W	Enable (if « 1 ») or Disable (if « 0 ») TMB9
9	R/W	-
10	R/W	-
11	R/W	-
12	R/W	-
13	R/W	-
14	R/W	-
15	R/W	-

5.7. CSR8 (6000CEh, FPGA, GTP status register, read only)

Table 11

Bit	Access	Function
0	R	PLLKDET0 from serializer ser40a_1
1	R	PLLKDET1 from serializer ser40a_1
2	R	PLLKDET0 from serializer ser40a_2
3	R	PLLKDET1 from serializer ser40a_2
4	R	PLLKDET0 from serializer ser40a_3
5	R	PLLKDET1 from serializer ser40a_3
6	R	PLLKDET0 from serializer ser40a_4
7	R	PLLKDET1 from serializer ser40a_4
8	R	RESETDONE0 from serializer se40a_1
9	R	RESETDONE1 from serializer se40a_1
10	R	RESETDONE0 from serializer se40a_2
11	R	RESETDONE1 from serializer se40a_2
12	R	RESETDONE0 from serializer se40a_3
13	R	RESETDONE1 from serializer se40a_3
14	R	RESETDONE0 from serializer se40a_4
15	R	RESETDONE1 from serializer se40a_4

5.8. CSR9 (6000B2h, FPGA, status register, read only)

Table 12

Bit	Access	Function
0	R	QPLL_LOCKED status (160MHz clock); “1” = locked
1	R	INTL signal from the Avago AFBR-810 optical transmitter (“0” – active interrupt)
2	R	SEU error status from the QPLL (“1” – error)
3	R	DCM block inthe FPGA is locked (active « 1 » when locked)
4	R	QPLL_LOCKED error counter bit[0]
5	R	QPLL_LOCKED error counter bit[1]
6	R	QPLL_LOCKED error counter bit[2]
7	R	QPLL_LOCKED error counter bit[3]
8	R	QPLL_LOCKED error counter bit[4]
9	R	QPLL_LOCKED error counter bit[5]
10	R	QPLL_LOCKED error counter bit[6]
11	R	QPLL_LOCKED error counter bit[7]
12	R	QPLL_LOCKED error counter bit[8]
13	R	QPLL_LOCKED error counter bit[9]
14	R	QPLL_LOCKED error counter bit[10]
15	R	QPLL_LOCKED error counter bit[11]

5.9. CSR10 (6000B4h, FPGA, general purpose i/o register)

5.10. CSR11 (6000CCh, FPGA, reset various sources, write "1" once to reset)

Table 13

Bit	Access	Function
0	W	n/a
1	W	n/a
2	W	n/a
3	W	n/a
4	W	Reset all GTP blocks
5	W	n/a
6	W	Reset QPLL
7	W	Reset L1A counter
8	W	Reset Avago AFBR-810 optical transmitter
9	W	n/a
10	W	n/a
11	W	n/a
12	W	n/a
13	W	n/a
14	W	n/a
15	W	n/a

References

[1] http://padley.rice.edu/cms/MPC2004_100808.pdf
[2] http://padley.rice.edu/cms/MPCMEZ6_050518.pdf
[3] https://indico.cern.ch/event/911653/contributions/3834318/subcontributions/304474/attachments/2025880/3389027/MPC-EMTF-format-2020.pdf

Appendix A

Transmitter mpcx_tx with 38B/40B encoding (A.Madorsky, 05/25/2020)

```
module mpcx_tx
(
        input [7:0] tmb1_lct0_hs,
        input [6:0] tmb1_lct0_wg,
        input [3:0] tmb1_lct0_qual,
        input [3:0] tmb1_lct0_cpat,
        input       tmb1_lct0_lr,
        input       tmb1_lct0_vpf,
        input       tmb1_lct0_bc0,
        input       tmb1_lct0_bx0,
        input       tmb1_lct0_syer,
        input [3:0] tmb1_lct0_cscid,

        input [7:0] tmb1_lct1_hs,
        input [6:0] tmb1_lct1_wg,
        input [3:0] tmb1_lct1_qual,
        input [3:0] tmb1_lct1_cpat,
        input       tmb1_lct1_lr,
        input       tmb1_lct1_vpf,
        input       tmb1_lct1_bc0,
        input       tmb1_lct1_bx0,
        input       tmb1_lct1_syer,
        input [3:0] tmb1_lct1_cscid,

        input [7:0] tmb2_lct0_hs,
        input [6:0] tmb2_lct0_wg,
        input [3:0] tmb2_lct0_qual,
        input [3:0] tmb2_lct0_cpat,
        input       tmb2_lct0_lr,
        input       tmb2_lct0_vpf,
        input       tmb2_lct0_bc0,
        input       tmb2_lct0_bx0,
        input       tmb2_lct0_syer,
        input [3:0] tmb2_lct0_cscid,

        input [7:0] tmb2_lct1_hs,
        input [6:0] tmb2_lct1_wg,
        input [3:0] tmb2_lct1_qual,
        input [3:0] tmb2_lct1_cpat,
        input       tmb2_lct1_lr,
        input       tmb2_lct1_vpf,
        input       tmb2_lct1_bc0,
        input       tmb2_lct1_bx0,
        input       tmb2_lct1_syer,
        input [3:0] tmb2_lct1_cscid,

        input [7:0] tmb3_lct0_hs,
        input [6:0] tmb3_lct0_wg,
        input [3:0] tmb3_lct0_qual,
        input [3:0] tmb3_lct0_cpat,
        input       tmb3_lct0_lr,
```

```
input       tmb3_lct0_vpf,
input       tmb3_lct0_bc0,
input       tmb3_lct0_bx0,
input       tmb3_lct0_syer,
input [3:0] tmb3_lct0_cscid,

input [7:0] tmb3_lct1_hs,
input [6:0] tmb3_lct1_wg,
input [3:0] tmb3_lct1_qual,
input [3:0] tmb3_lct1_cpat,
input       tmb3_lct1_lr,
input       tmb3_lct1_vpf,
input       tmb3_lct1_bc0,
input       tmb3_lct1_bx0,
input       tmb3_lct1_syer,
input [3:0] tmb3_lct1_cscid,

input [7:0] tmb4_lct0_hs,
input [6:0] tmb4_lct0_wg,
input [3:0] tmb4_lct0_qual,
input [3:0] tmb4_lct0_cpat,
input       tmb4_lct0_lr,
input       tmb4_lct0_vpf,
input       tmb4_lct0_bc0,
input       tmb4_lct0_bx0,
input       tmb4_lct0_syer,
input [3:0] tmb4_lct0_cscid,

input [7:0] tmb4_lct1_hs,
input [6:0] tmb4_lct1_wg,
input [3:0] tmb4_lct1_qual,
input [3:0] tmb4_lct1_cpat,
input       tmb4_lct1_lr,
input       tmb4_lct1_vpf,
input       tmb4_lct1_bc0,
input       tmb4_lct1_bx0,
input       tmb4_lct1_syer,
input [3:0] tmb4_lct1_cscid,

input [7:0] tmb5_lct0_hs,
input [6:0] tmb5_lct0_wg,
input [3:0] tmb5_lct0_qual,
input [3:0] tmb5_lct0_cpat,
input       tmb5_lct0_lr,
input       tmb5_lct0_vpf,
input       tmb5_lct0_bc0,
input       tmb5_lct0_bx0,
input       tmb5_lct0_syer,
input [3:0] tmb5_lct0_cscid,

input [7:0] tmb5_lct1_hs,
input [6:0] tmb5_lct1_wg,
input [3:0] tmb5_lct1_qual,
input [3:0] tmb5_lct1_cpat,
input       tmb5_lct1_lr,
input       tmb5_lct1_vpf,
```

```
input       tmb5_lct1_bc0,
input       tmb5_lct1_bx0,
input       tmb5_lct1_syer,
input [3:0] tmb5_lct1_cscid,

input [7:0] tmb6_lct0_hs,
input [6:0] tmb6_lct0_wg,
input [3:0] tmb6_lct0_qual,
input [3:0] tmb6_lct0_cpat,
input       tmb6_lct0_lr,
input       tmb6_lct0_vpf,
input       tmb6_lct0_bc0,
input       tmb6_lct0_bx0,
input       tmb6_lct0_syer,
input [3:0] tmb6_lct0_cscid,

input [7:0] tmb6_lct1_hs,
input [6:0] tmb6_lct1_wg,
input [3:0] tmb6_lct1_qual,
input [3:0] tmb6_lct1_cpat,
input       tmb6_lct1_lr,
input       tmb6_lct1_vpf,
input       tmb6_lct1_bc0,
input       tmb6_lct1_bx0,
input       tmb6_lct1_syer,
input [3:0] tmb6_lct1_cscid,

input [7:0] tmb7_lct0_hs,
input [6:0] tmb7_lct0_wg,
input [3:0] tmb7_lct0_qual,
input [3:0] tmb7_lct0_cpat,
input       tmb7_lct0_lr,
input       tmb7_lct0_vpf,
input       tmb7_lct0_bc0,
input       tmb7_lct0_bx0,
input       tmb7_lct0_syer,
input [3:0] tmb7_lct0_cscid,

input [7:0] tmb7_lct1_hs,
input [6:0] tmb7_lct1_wg,
input [3:0] tmb7_lct1_qual,
input [3:0] tmb7_lct1_cpat,
input       tmb7_lct1_lr,
input       tmb7_lct1_vpf,
input       tmb7_lct1_bc0,
input       tmb7_lct1_bx0,
input       tmb7_lct1_syer,
input [3:0] tmb7_lct1_cscid,

input [7:0] tmb8_lct0_hs,
input [6:0] tmb8_lct0_wg,
input [3:0] tmb8_lct0_qual,
input [3:0] tmb8_lct0_cpat,
input       tmb8_lct0_lr,
input       tmb8_lct0_vpf,
input       tmb8_lct0_bc0,
```

```
            input       tmb8_lct0_bx0,
            input       tmb8_lct0_syer,
            input [3:0] tmb8_lct0_cscid,

            input [7:0] tmb8_lct1_hs,
            input [6:0] tmb8_lct1_wg,
            input [3:0] tmb8_lct1_qual,
            input [3:0] tmb8_lct1_cpat,
            input       tmb8_lct1_lr,
            input       tmb8_lct1_vpf,
            input       tmb8_lct1_bc0,
            input       tmb8_lct1_bx0,
            input       tmb8_lct1_syer,
            input [3:0] tmb8_lct1_cscid,

            input [7:0] tmb9_lct0_hs,
            input [6:0] tmb9_lct0_wg,
            input [3:0] tmb9_lct0_qual,
            input [3:0] tmb9_lct0_cpat,
            input       tmb9_lct0_lr,
            input       tmb9_lct0_vpf,
            input       tmb9_lct0_bc0,
            input       tmb9_lct0_bx0,
            input       tmb9_lct0_syer,
            input [3:0] tmb9_lct0_cscid,

            input [7:0] tmb9_lct1_hs,
            input [6:0] tmb9_lct1_wg,
            input [3:0] tmb9_lct1_qual,
            input [3:0] tmb9_lct1_cpat,
            input       tmb9_lct1_lr,
            input       tmb9_lct1_vpf,
            input       tmb9_lct1_bc0,
            input       tmb9_lct1_bx0,
            input       tmb9_lct1_syer,
            input [3:0] tmb9_lct1_cscid,

            input [5:0] mpc_id,
            input enp_vme,

    input rst,
            input clk40,

    output [39:0] tx_data_gtp_0,
    output [39:0] tx_data_gtp_1,
    output [39:0] tx_data_gtp_2,
    output [39:0] tx_data_gtp_3,
    output [39:0] tx_data_gtp_4,
    output [39:0] tx_data_gtp_5,
    output [39:0] tx_data_gtp_6,
    output [39:0] tx_data_gtp_7,
            input tx_clk
);

            wire enp = enp_vio | enp_vme; // test pattern can be enabled from VIO or VME
            wire [75:0] link_tx_data [7:0];
```

```
(* mark_debug *) wire [37:0] tx_data_38_s [7:0], tx_data_38 [7:0];
(* mark_debug *) wire [7:0] frame_0;
reg  [1:0] tx_header [7:0];
(* mark_debug *) wire [1:0] tx_header_0 = tx_header[0];
(* mark_debug *) wire [1:0] tx_header_1 = tx_header[1];
(* mark_debug *) wire [1:0] tx_header_2 = tx_header[2];
(* mark_debug *) wire [1:0] tx_header_3 = tx_header[3];
(* mark_debug *) wire [1:0] tx_header_4 = tx_header[4];
(* mark_debug *) wire [1:0] tx_header_5 = tx_header[5];
(* mark_debug *) wire [1:0] tx_header_6 = tx_header[6];
(* mark_debug *) wire [1:0] tx_header_7 = tx_header[7];
reg [5:0] tx_header_cnt [7:0];


// transmit frame_0 marker only once per 64 BX, to prevent unlocking frame alignment in RX
// count frame_0 markers
        integer i;
always @(posedge tx_clk)
        begin
                for (i = 0; i < 8; i=i+1)
                begin
                        tx_header[i] =
     (rst == 1'b1) ? 2'b00 : // this is for header alignment tests on rx side
     (frame_0[i] == 1'b0 && tx_header_cnt[i] == 6'h0) ? 2'b10 : // frame_0 header
     2'b01; // regular header

                        if (frame_0[i] == 1'b1) tx_header_cnt[i] = tx_header_cnt[i] + 6'b1;
                end
        end

        reg [11:0] bxc;
        always @(posedge clk40)
        begin
                if (bxc == 12'd3563) bxc = 12'h0;
                else bxc = bxc + 12'b1; // just free-running BX counter
        end

        assign tx_data_gtp_0 = {tx_header[0], tx_data_38_s[0]};
        assign tx_data_gtp_1 = {tx_header[1], tx_data_38_s[1]};
        assign tx_data_gtp_2 = {tx_header[2], tx_data_38_s[2]};
        assign tx_data_gtp_3 = {tx_header[3], tx_data_38_s[3]};
        assign tx_data_gtp_4 = {tx_header[4], tx_data_38_s[4]};
        assign tx_data_gtp_5 = {tx_header[5], tx_data_38_s[5]};
        assign tx_data_gtp_6 = {tx_header[6], tx_data_38_s[6]};
        assign tx_data_gtp_7 = {tx_header[7], tx_data_38_s[7]};

        genvar gi, gj;
        generate
        for (gi = 0; gi < 8; gi=gi+1)
        begin: link_loop


                mpc_scrambler mpc_s
                (
                        .dtx (tx_data_38[gi]), // TX data to scramble
```

```
                .stx (tx_data_38_s[gi]), // scrambled TX data
                .clk (tx_clk),
                .rst (1'b0)
        );

        tx_reclock txr
        (
                .din     (link_tx_data [gi]),
                .clk40   (clk40),
                .dout    (tx_data_38[gi]),
                .frame_0 (frame_0[gi]),
                .clk80   (tx_clk)
        );
 end
 endgenerate

(* mark_debug *) reg [7:0] hs    [9:1][1:0]; // [tmb][lct]
(* mark_debug *) reg [6:0] wg    [9:1][1:0];
(* mark_debug *) reg [3:0] qual  [9:1][1:0];
(* mark_debug *) reg [3:0] cpat  [9:1][1:0];
(* mark_debug *) reg       lr    [9:1][1:0];
(* mark_debug *) reg       vpf   [9:1][1:0];
(* mark_debug *) reg       bc0   [9:1][1:0];
(* mark_debug *) reg       bx0   [9:1][1:0];
(* mark_debug *) reg       syer  [9:1][1:0];
(* mark_debug *) reg [3:0] cscid [9:1][1:0];

always @(*)
begin

        hs    [1][0] = (enp == 1'b0) ? tmb1_lct0_hs     : bxc[7:0];
        wg    [1][0] = (enp == 1'b0) ? tmb1_lct0_wg     : bxc[6:0];
        qual  [1][0] = (enp == 1'b0) ? tmb1_lct0_qual   : bxc[3:0];
        cpat  [1][0] = (enp == 1'b0) ? tmb1_lct0_cpat   : bxc[3:0];
        lr    [1][0] = (enp == 1'b0) ? tmb1_lct0_lr     : bxc[0];
        vpf   [1][0] = (enp == 1'b0) ? tmb1_lct0_vpf    : bxc[0];
        bc0   [1][0] = (enp == 1'b0) ? tmb1_lct0_bc0    : (bxc == 12'd3563);
        bx0   [1][0] = (enp == 1'b0) ? tmb1_lct0_bx0    : bxc[1];
        syer  [1][0] = (enp == 1'b0) ? tmb1_lct0_syer   : 0;
        cscid [1][0] = (enp == 1'b0) ? tmb1_lct0_cscid  : 1;
        hs    [1][1] = (enp == 1'b0) ? tmb1_lct1_hs     : bxc[7:0];
        wg    [1][1] = (enp == 1'b0) ? tmb1_lct1_wg     : bxc[6:0];
        qual  [1][1] = (enp == 1'b0) ? tmb1_lct1_qual   : bxc[3:0];
        cpat  [1][1] = (enp == 1'b0) ? tmb1_lct1_cpat   : bxc[3:0];
        lr    [1][1] = (enp == 1'b0) ? tmb1_lct1_lr     : bxc[0];
        vpf   [1][1] = (enp == 1'b0) ? tmb1_lct1_vpf    : ~bxc[0];
        bc0   [1][1] = (enp == 1'b0) ? tmb1_lct1_bc0    : (bxc == 12'd3563);
        bx0   [1][1] = (enp == 1'b0) ? tmb1_lct1_bx0    : bxc[1];
        syer  [1][1] = (enp == 1'b0) ? tmb1_lct1_syer   : 1;
        cscid [1][1] = (enp == 1'b0) ? tmb1_lct1_cscid  : 1;
        hs    [2][0] = (enp == 1'b0) ? tmb2_lct0_hs     : bxc[7:0];
        wg    [2][0] = (enp == 1'b0) ? tmb2_lct0_wg     : bxc[6:0];
        qual  [2][0] = (enp == 1'b0) ? tmb2_lct0_qual   : bxc[3:0];
        cpat  [2][0] = (enp == 1'b0) ? tmb2_lct0_cpat   : bxc[3:0];
        lr    [2][0] = (enp == 1'b0) ? tmb2_lct0_lr     : bxc[0];
        vpf   [2][0] = (enp == 1'b0) ? tmb2_lct0_vpf    : bxc[0];
```

```
bc0   [2][0] = (enp == 1'b0) ? tmb2_lct0_bc0    : (bxc == 12'd3563);
bx0   [2][0] = (enp == 1'b0) ? tmb2_lct0_bx0    : bxc[1];
syer  [2][0] = (enp == 1'b0) ? tmb2_lct0_syer   : 0;
cscid [2][0] = (enp == 1'b0) ? tmb2_lct0_cscid  : 2;
hs    [2][1] = (enp == 1'b0) ? tmb2_lct1_hs     : bxc[7:0];
wg    [2][1] = (enp == 1'b0) ? tmb2_lct1_wg     : bxc[6:0];
qual  [2][1] = (enp == 1'b0) ? tmb2_lct1_qual   : bxc[3:0];
cpat  [2][1] = (enp == 1'b0) ? tmb2_lct1_cpat   : bxc[3:0];
lr    [2][1] = (enp == 1'b0) ? tmb2_lct1_lr     : bxc[0];
vpf   [2][1] = (enp == 1'b0) ? tmb2_lct1_vpf    : ~bxc[0];
bc0   [2][1] = (enp == 1'b0) ? tmb2_lct1_bc0    : (bxc == 12'd3563);
bx0   [2][1] = (enp == 1'b0) ? tmb2_lct1_bx0    : bxc[1];
syer  [2][1] = (enp == 1'b0) ? tmb2_lct1_syer   : 1;
cscid [2][1] = (enp == 1'b0) ? tmb2_lct1_cscid  : 2;
hs    [3][0] = (enp == 1'b0) ? tmb3_lct0_hs     : bxc[7:0];
wg    [3][0] = (enp == 1'b0) ? tmb3_lct0_wg     : bxc[6:0];
qual  [3][0] = (enp == 1'b0) ? tmb3_lct0_qual   : bxc[3:0];
cpat  [3][0] = (enp == 1'b0) ? tmb3_lct0_cpat   : bxc[3:0];
lr    [3][0] = (enp == 1'b0) ? tmb3_lct0_lr     : bxc[0];
vpf   [3][0] = (enp == 1'b0) ? tmb3_lct0_vpf    : bxc[0];
bc0   [3][0] = (enp == 1'b0) ? tmb3_lct0_bc0    : (bxc == 12'd3563);
bx0   [3][0] = (enp == 1'b0) ? tmb3_lct0_bx0    : bxc[1];
syer  [3][0] = (enp == 1'b0) ? tmb3_lct0_syer   : 0;
cscid [3][0] = (enp == 1'b0) ? tmb3_lct0_cscid  : 3;
hs    [3][1] = (enp == 1'b0) ? tmb3_lct1_hs     : bxc[7:0];
wg    [3][1] = (enp == 1'b0) ? tmb3_lct1_wg     : bxc[6:0];
qual  [3][1] = (enp == 1'b0) ? tmb3_lct1_qual   : bxc[3:0];
cpat  [3][1] = (enp == 1'b0) ? tmb3_lct1_cpat   : bxc[3:0];
lr    [3][1] = (enp == 1'b0) ? tmb3_lct1_lr     : bxc[0];
vpf   [3][1] = (enp == 1'b0) ? tmb3_lct1_vpf    : ~bxc[0];
bc0   [3][1] = (enp == 1'b0) ? tmb3_lct1_bc0    : (bxc == 12'd3563);
bx0   [3][1] = (enp == 1'b0) ? tmb3_lct1_bx0    : bxc[1];
syer  [3][1] = (enp == 1'b0) ? tmb3_lct1_syer   : 1;
cscid [3][1] = (enp == 1'b0) ? tmb3_lct1_cscid  : 3;
hs    [4][0] = (enp == 1'b0) ? tmb4_lct0_hs     : bxc[7:0];
wg    [4][0] = (enp == 1'b0) ? tmb4_lct0_wg     : bxc[6:0];
qual  [4][0] = (enp == 1'b0) ? tmb4_lct0_qual   : bxc[3:0];
cpat  [4][0] = (enp == 1'b0) ? tmb4_lct0_cpat   : bxc[3:0];
lr    [4][0] = (enp == 1'b0) ? tmb4_lct0_lr     : bxc[0];
vpf   [4][0] = (enp == 1'b0) ? tmb4_lct0_vpf    : bxc[0];
bc0   [4][0] = (enp == 1'b0) ? tmb4_lct0_bc0    : (bxc == 12'd3563);
bx0   [4][0] = (enp == 1'b0) ? tmb4_lct0_bx0    : bxc[1];
syer  [4][0] = (enp == 1'b0) ? tmb4_lct0_syer   : 0;
cscid [4][0] = (enp == 1'b0) ? tmb4_lct0_cscid  : 4;
hs    [4][1] = (enp == 1'b0) ? tmb4_lct1_hs     : bxc[7:0];
wg    [4][1] = (enp == 1'b0) ? tmb4_lct1_wg     : bxc[6:0];
qual  [4][1] = (enp == 1'b0) ? tmb4_lct1_qual   : bxc[3:0];
cpat  [4][1] = (enp == 1'b0) ? tmb4_lct1_cpat   : bxc[3:0];
lr    [4][1] = (enp == 1'b0) ? tmb4_lct1_lr     : bxc[0];
vpf   [4][1] = (enp == 1'b0) ? tmb4_lct1_vpf    : ~bxc[0];
bc0   [4][1] = (enp == 1'b0) ? tmb4_lct1_bc0    : (bxc == 12'd3563);
bx0   [4][1] = (enp == 1'b0) ? tmb4_lct1_bx0    : bxc[1];
syer  [4][1] = (enp == 1'b0) ? tmb4_lct1_syer   : 1;
cscid [4][1] = (enp == 1'b0) ? tmb4_lct1_cscid  : 4;
hs    [5][0] = (enp == 1'b0) ? tmb5_lct0_hs     : bxc[7:0];
wg    [5][0] = (enp == 1'b0) ? tmb5_lct0_wg     : bxc[6:0];
```

```
qual  [5][0] = (enp == 1'b0) ? tmb5_lct0_qual   : bxc[3:0];
cpat  [5][0] = (enp == 1'b0) ? tmb5_lct0_cpat   : bxc[3:0];
lr    [5][0] = (enp == 1'b0) ? tmb5_lct0_lr     : bxc[0];
vpf   [5][0] = (enp == 1'b0) ? tmb5_lct0_vpf    : bxc[0];
bc0   [5][0] = (enp == 1'b0) ? tmb5_lct0_bc0    : (bxc == 12'd3563);
bx0   [5][0] = (enp == 1'b0) ? tmb5_lct0_bx0    : bxc[1];
syer  [5][0] = (enp == 1'b0) ? tmb5_lct0_syer   : 0;
cscid [5][0] = (enp == 1'b0) ? tmb5_lct0_cscid  : 5;
hs    [5][1] = (enp == 1'b0) ? tmb5_lct1_hs     : bxc[7:0];
wg    [5][1] = (enp == 1'b0) ? tmb5_lct1_wg     : bxc[6:0];
qual  [5][1] = (enp == 1'b0) ? tmb5_lct1_qual   : bxc[3:0];
cpat  [5][1] = (enp == 1'b0) ? tmb5_lct1_cpat   : bxc[3:0];
lr    [5][1] = (enp == 1'b0) ? tmb5_lct1_lr     : bxc[0];
vpf   [5][1] = (enp == 1'b0) ? tmb5_lct1_vpf    : ~bxc[0];
bc0   [5][1] = (enp == 1'b0) ? tmb5_lct1_bc0    : (bxc == 12'd3563);
bx0   [5][1] = (enp == 1'b0) ? tmb5_lct1_bx0    : bxc[1];
syer  [5][1] = (enp == 1'b0) ? tmb5_lct1_syer   : 1;
cscid [5][1] = (enp == 1'b0) ? tmb5_lct1_cscid  : 5;
hs    [6][0] = (enp == 1'b0) ? tmb6_lct0_hs     : bxc[7:0];
wg    [6][0] = (enp == 1'b0) ? tmb6_lct0_wg     : bxc[6:0];
qual  [6][0] = (enp == 1'b0) ? tmb6_lct0_qual   : bxc[3:0];
cpat  [6][0] = (enp == 1'b0) ? tmb6_lct0_cpat   : bxc[3:0];
lr    [6][0] = (enp == 1'b0) ? tmb6_lct0_lr     : bxc[0];
vpf   [6][0] = (enp == 1'b0) ? tmb6_lct0_vpf    : bxc[0];
bc0   [6][0] = (enp == 1'b0) ? tmb6_lct0_bc0    : (bxc == 12'd3563);
bx0   [6][0] = (enp == 1'b0) ? tmb6_lct0_bx0    : bxc[1];
syer  [6][0] = (enp == 1'b0) ? tmb6_lct0_syer   : 0;
cscid [6][0] = (enp == 1'b0) ? tmb6_lct0_cscid  : 6;
hs    [6][1] = (enp == 1'b0) ? tmb6_lct1_hs     : bxc[7:0];
wg    [6][1] = (enp == 1'b0) ? tmb6_lct1_wg     : bxc[6:0];
qual  [6][1] = (enp == 1'b0) ? tmb6_lct1_qual   : bxc[3:0];
cpat  [6][1] = (enp == 1'b0) ? tmb6_lct1_cpat   : bxc[3:0];
lr    [6][1] = (enp == 1'b0) ? tmb6_lct1_lr     : bxc[0];
vpf   [6][1] = (enp == 1'b0) ? tmb6_lct1_vpf    : ~bxc[0];
bc0   [6][1] = (enp == 1'b0) ? tmb6_lct1_bc0    : (bxc == 12'd3563);
bx0   [6][1] = (enp == 1'b0) ? tmb6_lct1_bx0    : bxc[1];
syer  [6][1] = (enp == 1'b0) ? tmb6_lct1_syer   : 1;
cscid [6][1] = (enp == 1'b0) ? tmb6_lct1_cscid  : 6;
hs    [7][0] = (enp == 1'b0) ? tmb7_lct0_hs     : bxc[7:0];
wg    [7][0] = (enp == 1'b0) ? tmb7_lct0_wg     : bxc[6:0];
qual  [7][0] = (enp == 1'b0) ? tmb7_lct0_qual   : bxc[3:0];
cpat  [7][0] = (enp == 1'b0) ? tmb7_lct0_cpat   : bxc[3:0];
lr    [7][0] = (enp == 1'b0) ? tmb7_lct0_lr     : bxc[0];
vpf   [7][0] = (enp == 1'b0) ? tmb7_lct0_vpf    : bxc[0];
bc0   [7][0] = (enp == 1'b0) ? tmb7_lct0_bc0    : (bxc == 12'd3563);
bx0   [7][0] = (enp == 1'b0) ? tmb7_lct0_bx0    : bxc[1];
syer  [7][0] = (enp == 1'b0) ? tmb7_lct0_syer   : 0;
cscid [7][0] = (enp == 1'b0) ? tmb7_lct0_cscid  : 7;
hs    [7][1] = (enp == 1'b0) ? tmb7_lct1_hs     : bxc[7:0];
wg    [7][1] = (enp == 1'b0) ? tmb7_lct1_wg     : bxc[6:0];
qual  [7][1] = (enp == 1'b0) ? tmb7_lct1_qual   : bxc[3:0];
cpat  [7][1] = (enp == 1'b0) ? tmb7_lct1_cpat   : bxc[3:0];
lr    [7][1] = (enp == 1'b0) ? tmb7_lct1_lr     : bxc[0];
vpf   [7][1] = (enp == 1'b0) ? tmb7_lct1_vpf    : ~bxc[0];
bc0   [7][1] = (enp == 1'b0) ? tmb7_lct1_bc0    : (bxc == 12'd3563);
bx0   [7][1] = (enp == 1'b0) ? tmb7_lct1_bx0    : bxc[1];
```

```
        syer  [7][1] = (enp == 1'b0) ? tmb7_lct1_syer   : 1;
        cscid [7][1] = (enp == 1'b0) ? tmb7_lct1_cscid  : 7;
        hs    [8][0] = (enp == 1'b0) ? tmb8_lct0_hs     : bxc[7:0];
        wg    [8][0] = (enp == 1'b0) ? tmb8_lct0_wg     : bxc[6:0];
        qual  [8][0] = (enp == 1'b0) ? tmb8_lct0_qual   : bxc[3:0];
        cpat  [8][0] = (enp == 1'b0) ? tmb8_lct0_cpat   : bxc[3:0];
        lr    [8][0] = (enp == 1'b0) ? tmb8_lct0_lr     : bxc[0];
        vpf   [8][0] = (enp == 1'b0) ? tmb8_lct0_vpf    : bxc[0];
        bc0   [8][0] = (enp == 1'b0) ? tmb8_lct0_bc0    : (bxc == 12'd3563);
        bx0   [8][0] = (enp == 1'b0) ? tmb8_lct0_bx0    : bxc[1];
        syer  [8][0] = (enp == 1'b0) ? tmb8_lct0_syer   : 0;
        cscid [8][0] = (enp == 1'b0) ? tmb8_lct0_cscid  : 8;
        hs    [8][1] = (enp == 1'b0) ? tmb8_lct1_hs     : bxc[7:0];
        wg    [8][1] = (enp == 1'b0) ? tmb8_lct1_wg     : bxc[6:0];
        qual  [8][1] = (enp == 1'b0) ? tmb8_lct1_qual   : bxc[3:0];
        cpat  [8][1] = (enp == 1'b0) ? tmb8_lct1_cpat   : bxc[3:0];
        lr    [8][1] = (enp == 1'b0) ? tmb8_lct1_lr     : bxc[0];
        vpf   [8][1] = (enp == 1'b0) ? tmb8_lct1_vpf    : ~bxc[0];
        bc0   [8][1] = (enp == 1'b0) ? tmb8_lct1_bc0    : (bxc == 12'd3563);
        bx0   [8][1] = (enp == 1'b0) ? tmb8_lct1_bx0    : bxc[1];
        syer  [8][1] = (enp == 1'b0) ? tmb8_lct1_syer   : 1;
        cscid [8][1] = (enp == 1'b0) ? tmb8_lct1_cscid  : 8;
        hs    [9][0] = (enp == 1'b0) ? tmb9_lct0_hs     : bxc[7:0];
        wg    [9][0] = (enp == 1'b0) ? tmb9_lct0_wg     : bxc[6:0];
        qual  [9][0] = (enp == 1'b0) ? tmb9_lct0_qual   : bxc[3:0];
        cpat  [9][0] = (enp == 1'b0) ? tmb9_lct0_cpat   : bxc[3:0];
        lr    [9][0] = (enp == 1'b0) ? tmb9_lct0_lr     : bxc[0];
        vpf   [9][0] = (enp == 1'b0) ? tmb9_lct0_vpf    : bxc[0];
        bc0   [9][0] = (enp == 1'b0) ? tmb9_lct0_bc0    : (bxc == 12'd3563);
        bx0   [9][0] = (enp == 1'b0) ? tmb9_lct0_bx0    : bxc[1];
        syer  [9][0] = (enp == 1'b0) ? tmb9_lct0_syer   : 0;
        cscid [9][0] = (enp == 1'b0) ? tmb9_lct0_cscid  : 9;
        hs    [9][1] = (enp == 1'b0) ? tmb9_lct1_hs     : bxc[7:0];
        wg    [9][1] = (enp == 1'b0) ? tmb9_lct1_wg     : bxc[6:0];
        qual  [9][1] = (enp == 1'b0) ? tmb9_lct1_qual   : bxc[3:0];
        cpat  [9][1] = (enp == 1'b0) ? tmb9_lct1_cpat   : bxc[3:0];
        lr    [9][1] = (enp == 1'b0) ? tmb9_lct1_lr     : bxc[0];
        vpf   [9][1] = (enp == 1'b0) ? tmb9_lct1_vpf    : ~bxc[0];
        bc0   [9][1] = (enp == 1'b0) ? tmb9_lct1_bc0    : (bxc == 12'd3563);
        bx0   [9][1] = (enp == 1'b0) ? tmb9_lct1_bx0    : bxc[1];
        syer  [9][1] = (enp == 1'b0) ? tmb9_lct1_syer   : 1;
        cscid [9][1] = (enp == 1'b0) ? tmb9_lct1_cscid  : 9;

end

mpc_formatter mpcf
(

        .tmb1_lct0_hs     (hs    [1][0]),
        .tmb1_lct0_wg     (wg    [1][0]),
        .tmb1_lct0_qual   (qual  [1][0]),
        .tmb1_lct0_cpat   (cpat  [1][0]),
        .tmb1_lct0_lr     (lr    [1][0]),
        .tmb1_lct0_vpf    (vpf   [1][0]),
        .tmb1_lct0_bc0    (bc0   [1][0]),
        .tmb1_lct0_bx0    (bx0   [1][0]),
```

```
.tmb1_lct0_syer   (syer  [1][0]),
.tmb1_lct0_cscid  (cscid [1][0]),

.tmb1_lct1_hs     (hs    [1][1]),
.tmb1_lct1_wg     (wg    [1][1]),
.tmb1_lct1_qual   (qual  [1][1]),
.tmb1_lct1_cpat   (cpat  [1][1]),
.tmb1_lct1_lr     (lr    [1][1]),
.tmb1_lct1_vpf    (vpf   [1][1]),
.tmb1_lct1_bc0    (bc0   [1][1]),
.tmb1_lct1_bx0    (bx0   [1][1]),
.tmb1_lct1_syer   (syer  [1][1]),
.tmb1_lct1_cscid  (cscid [1][1]),

.tmb2_lct0_hs     (hs    [2][0]),
.tmb2_lct0_wg     (wg    [2][0]),
.tmb2_lct0_qual   (qual  [2][0]),
.tmb2_lct0_cpat   (cpat  [2][0]),
.tmb2_lct0_lr     (lr    [2][0]),
.tmb2_lct0_vpf    (vpf   [2][0]),
.tmb2_lct0_bc0    (bc0   [2][0]),
.tmb2_lct0_bx0    (bx0   [2][0]),
.tmb2_lct0_syer   (syer  [2][0]),
.tmb2_lct0_cscid  (cscid [2][0]),

.tmb2_lct1_hs     (hs    [2][1]),
.tmb2_lct1_wg     (wg    [2][1]),
.tmb2_lct1_qual   (qual  [2][1]),
.tmb2_lct1_cpat   (cpat  [2][1]),
.tmb2_lct1_lr     (lr    [2][1]),
.tmb2_lct1_vpf    (vpf   [2][1]),
.tmb2_lct1_bc0    (bc0   [2][1]),
.tmb2_lct1_bx0    (bx0   [2][1]),
.tmb2_lct1_syer   (syer  [2][1]),
.tmb2_lct1_cscid  (cscid [2][1]),

.tmb3_lct0_hs     (hs    [3][0]),
.tmb3_lct0_wg     (wg    [3][0]),
.tmb3_lct0_qual   (qual  [3][0]),
.tmb3_lct0_cpat   (cpat  [3][0]),
.tmb3_lct0_lr     (lr    [3][0]),
.tmb3_lct0_vpf    (vpf   [3][0]),
.tmb3_lct0_bc0    (bc0   [3][0]),
.tmb3_lct0_bx0    (bx0   [3][0]),
.tmb3_lct0_syer   (syer  [3][0]),
.tmb3_lct0_cscid  (cscid [3][0]),

.tmb3_lct1_hs     (hs    [3][1]),
.tmb3_lct1_wg     (wg    [3][1]),
.tmb3_lct1_qual   (qual  [3][1]),
.tmb3_lct1_cpat   (cpat  [3][1]),
.tmb3_lct1_lr     (lr    [3][1]),
.tmb3_lct1_vpf    (vpf   [3][1]),
.tmb3_lct1_bc0    (bc0   [3][1]),
.tmb3_lct1_bx0    (bx0   [3][1]),
.tmb3_lct1_syer   (syer  [3][1]),
```

```
.tmb3_lct1_cscid  (cscid [3][1]),

.tmb4_lct0_hs     (hs    [4][0]),
.tmb4_lct0_wg     (wg    [4][0]),
.tmb4_lct0_qual   (qual  [4][0]),
.tmb4_lct0_cpat   (cpat  [4][0]),
.tmb4_lct0_lr     (lr    [4][0]),
.tmb4_lct0_vpf    (vpf   [4][0]),
.tmb4_lct0_bc0    (bc0   [4][0]),
.tmb4_lct0_bx0    (bx0   [4][0]),
.tmb4_lct0_syer   (syer  [4][0]),
.tmb4_lct0_cscid  (cscid [4][0]),

.tmb4_lct1_hs     (hs    [4][1]),
.tmb4_lct1_wg     (wg    [4][1]),
.tmb4_lct1_qual   (qual  [4][1]),
.tmb4_lct1_cpat   (cpat  [4][1]),
.tmb4_lct1_lr     (lr    [4][1]),
.tmb4_lct1_vpf    (vpf   [4][1]),
.tmb4_lct1_bc0    (bc0   [4][1]),
.tmb4_lct1_bx0    (bx0   [4][1]),
.tmb4_lct1_syer   (syer  [4][1]),
.tmb4_lct1_cscid  (cscid [4][1]),

.tmb5_lct0_hs     (hs    [5][0]),
.tmb5_lct0_wg     (wg    [5][0]),
.tmb5_lct0_qual   (qual  [5][0]),
.tmb5_lct0_cpat   (cpat  [5][0]),
.tmb5_lct0_lr     (lr    [5][0]),
.tmb5_lct0_vpf    (vpf   [5][0]),
.tmb5_lct0_bc0    (bc0   [5][0]),
.tmb5_lct0_bx0    (bx0   [5][0]),
.tmb5_lct0_syer   (syer  [5][0]),
.tmb5_lct0_cscid  (cscid [5][0]),

.tmb5_lct1_hs     (hs    [5][1]),
.tmb5_lct1_wg     (wg    [5][1]),
.tmb5_lct1_qual   (qual  [5][1]),
.tmb5_lct1_cpat   (cpat  [5][1]),
.tmb5_lct1_lr     (lr    [5][1]),
.tmb5_lct1_vpf    (vpf   [5][1]),
.tmb5_lct1_bc0    (bc0   [5][1]),
.tmb5_lct1_bx0    (bx0   [5][1]),
.tmb5_lct1_syer   (syer  [5][1]),
.tmb5_lct1_cscid  (cscid [5][1]),

.tmb6_lct0_hs     (hs    [6][0]),
.tmb6_lct0_wg     (wg    [6][0]),
.tmb6_lct0_qual   (qual  [6][0]),
.tmb6_lct0_cpat   (cpat  [6][0]),
.tmb6_lct0_lr     (lr    [6][0]),
.tmb6_lct0_vpf    (vpf   [6][0]),
.tmb6_lct0_bc0    (bc0   [6][0]),
.tmb6_lct0_bx0    (bx0   [6][0]),
.tmb6_lct0_syer   (syer  [6][0]),
.tmb6_lct0_cscid  (cscid [6][0]),
```

```
.tmb6_lct1_hs     (hs    [6][1]),
.tmb6_lct1_wg     (wg    [6][1]),
.tmb6_lct1_qual   (qual  [6][1]),
.tmb6_lct1_cpat   (cpat  [6][1]),
.tmb6_lct1_lr     (lr    [6][1]),
.tmb6_lct1_vpf    (vpf   [6][1]),
.tmb6_lct1_bc0    (bc0   [6][1]),
.tmb6_lct1_bx0    (bx0   [6][1]),
.tmb6_lct1_syer   (syer  [6][1]),
.tmb6_lct1_cscid  (cscid [6][1]),

.tmb7_lct0_hs     (hs    [7][0]),
.tmb7_lct0_wg     (wg    [7][0]),
.tmb7_lct0_qual   (qual  [7][0]),
.tmb7_lct0_cpat   (cpat  [7][0]),
.tmb7_lct0_lr     (lr    [7][0]),
.tmb7_lct0_vpf    (vpf   [7][0]),
.tmb7_lct0_bc0    (bc0   [7][0]),
.tmb7_lct0_bx0    (bx0   [7][0]),
.tmb7_lct0_syer   (syer  [7][0]),
.tmb7_lct0_cscid  (cscid [7][0]),

.tmb7_lct1_hs     (hs    [7][1]),
.tmb7_lct1_wg     (wg    [7][1]),
.tmb7_lct1_qual   (qual  [7][1]),
.tmb7_lct1_cpat   (cpat  [7][1]),
.tmb7_lct1_lr     (lr    [7][1]),
.tmb7_lct1_vpf    (vpf   [7][1]),
.tmb7_lct1_bc0    (bc0   [7][1]),
.tmb7_lct1_bx0    (bx0   [7][1]),
.tmb7_lct1_syer   (syer  [7][1]),
.tmb7_lct1_cscid  (cscid [7][1]),

.tmb8_lct0_hs     (hs    [8][0]),
.tmb8_lct0_wg     (wg    [8][0]),
.tmb8_lct0_qual   (qual  [8][0]),
.tmb8_lct0_cpat   (cpat  [8][0]),
.tmb8_lct0_lr     (lr    [8][0]),
.tmb8_lct0_vpf    (vpf   [8][0]),
.tmb8_lct0_bc0    (bc0   [8][0]),
.tmb8_lct0_bx0    (bx0   [8][0]),
.tmb8_lct0_syer   (syer  [8][0]),
.tmb8_lct0_cscid  (cscid [8][0]),

.tmb8_lct1_hs     (hs    [8][1]),
.tmb8_lct1_wg     (wg    [8][1]),
.tmb8_lct1_qual   (qual  [8][1]),
.tmb8_lct1_cpat   (cpat  [8][1]),
.tmb8_lct1_lr     (lr    [8][1]),
.tmb8_lct1_vpf    (vpf   [8][1]),
.tmb8_lct1_bc0    (bc0   [8][1]),
.tmb8_lct1_bx0    (bx0   [8][1]),
.tmb8_lct1_syer   (syer  [8][1]),
.tmb8_lct1_cscid  (cscid [8][1]),
```

```
                .tmb9_lct0_hs     (hs    [9][0]),
                .tmb9_lct0_wg     (wg    [9][0]),
                .tmb9_lct0_qual   (qual  [9][0]),
                .tmb9_lct0_cpat   (cpat  [9][0]),
                .tmb9_lct0_lr     (lr    [9][0]),
                .tmb9_lct0_vpf    (vpf   [9][0]),
                .tmb9_lct0_bc0    (bc0   [9][0]),
                .tmb9_lct0_bx0    (bx0   [9][0]),
                .tmb9_lct0_syer   (syer  [9][0]),
                .tmb9_lct0_cscid  (cscid [9][0]),

                .tmb9_lct1_hs     (hs    [9][1]),
                .tmb9_lct1_wg     (wg    [9][1]),
                .tmb9_lct1_qual   (qual  [9][1]),
                .tmb9_lct1_cpat   (cpat  [9][1]),
                .tmb9_lct1_lr     (lr    [9][1]),
                .tmb9_lct1_vpf    (vpf   [9][1]),
                .tmb9_lct1_bc0    (bc0   [9][1]),
                .tmb9_lct1_bx0    (bx0   [9][1]),
                .tmb9_lct1_syer   (syer  [9][1]),
                .tmb9_lct1_cscid  (cscid [9][1]),

                .link0_tx_data (link_tx_data [0]),
                .link1_tx_data (link_tx_data [1]),
                .link2_tx_data (link_tx_data [2]),
                .link3_tx_data (link_tx_data [3]),
                .link4_tx_data (link_tx_data [4]),
                .link5_tx_data (link_tx_data [5]),
                .link6_tx_data (link_tx_data [6]),
                .link7_tx_data (link_tx_data [7]),

                .mpc_id (mpc_id),

                .clk40 (clk40)

        );

        wire [35:0] CONTROL0, CONTROL1;

        vio_control vio_c
        (
                .CONTROL0 (CONTROL0) // INOUT BUS [35:0]
//              .CONTROL1 (CONTROL1) // INOUT BUS [35:0]
        );


        mpcx_tx_vio vio
        (
                .CONTROL  (CONTROL0), // INOUT BUS [35:0]
                .CLK      (clk40), // IN
                .SYNC_OUT (enp_vio) // OUT BUS [0:0]
        );
/*
        mpcx_ila mila
        (
                .CONTROL(CONTROL1), // INOUT BUS [35:0]
```

```
                .CLK(clk40), // IN
                .DATA({
                                link_tx_data[7],
                                link_tx_data[6],
                                link_tx_data[5],
                                link_tx_data[4],
                                link_tx_data[3],
                                link_tx_data[2],
                                link_tx_data[1],
                                link_tx_data[0]
                        }), // IN BUS [623:0]
                .TRIG0(link_tx_data[0][0]) // IN BUS [0:0]
        );
        */
endmodule
--------------------------------------------------------------------------------------------------------------
module mpc_scrambler #
(
   parameter BW = 38
)
(
   input  wire  [(BW-1):0] dtx,
   output reg   [(BW-1):0] stx,

   input  wire         clk,
   input  wire         rst
);


   integer           i;
   reg    [57:0]     poly;
   reg    [(BW-1):0]  scrambled_data_i;
   reg    [57:0]     scrambler;
   reg    [(BW-1):0]  tempData;
   reg               xorBit;

   always @(*)
   begin
      poly = scrambler;
      for (i = 0; i < BW; i=i+1)
      begin
         xorBit = dtx[i] ^ poly[38] ^ poly[57];
         poly = {poly[56:0], xorBit};
         tempData[i] = xorBit;
      end
   end

   always @(posedge clk)
   begin
      if (rst)
      begin
         stx      = 38'h1555555555;//'h0;
         scrambler = 58'h155_5555_5555_5555;
      end
      else
      begin
```

```
        stx      = tempData;
        scrambler = poly;
      end
    end

endmodule
```

```
module tx_reclock
(
    input [75:0] din,
    // this is fabric clock, in phase = 0 with clk80
    input clk40,

    output [37:0] dout,
    output reg frame_0,
    input clk80 // this is tx_clk
);

    reg [75:0] inreg;
    (* mark_debug *) wire [75:0] inreg_w = inreg;
            assign dout = inreg[37:0];
    reg clk40_ff;

    always @(posedge clk40)
    begin
        clk40_ff = ~clk40_ff; // just inverts on every clk40
    end

    (* async_reg = "TRUE" *) reg clk40_r;
    (* mark_debug *) wire clk40_w = clk40;
    (* mark_debug *) wire clk80_w = clk80;

    // mux output according to clk40 phase
    always @(posedge clk80)
    begin
        if (clk40_r != clk40_ff)
        begin
            frame_0 = 1'b0;
                                        inreg[37:0] = inreg[75:38];
        end

        if (clk40_r == clk40_ff)
        begin
            frame_0 = 1'b1;
                                        inreg = din;
        end

        clk40_r = clk40_ff; // clk40 history
    end

endmodule
```
