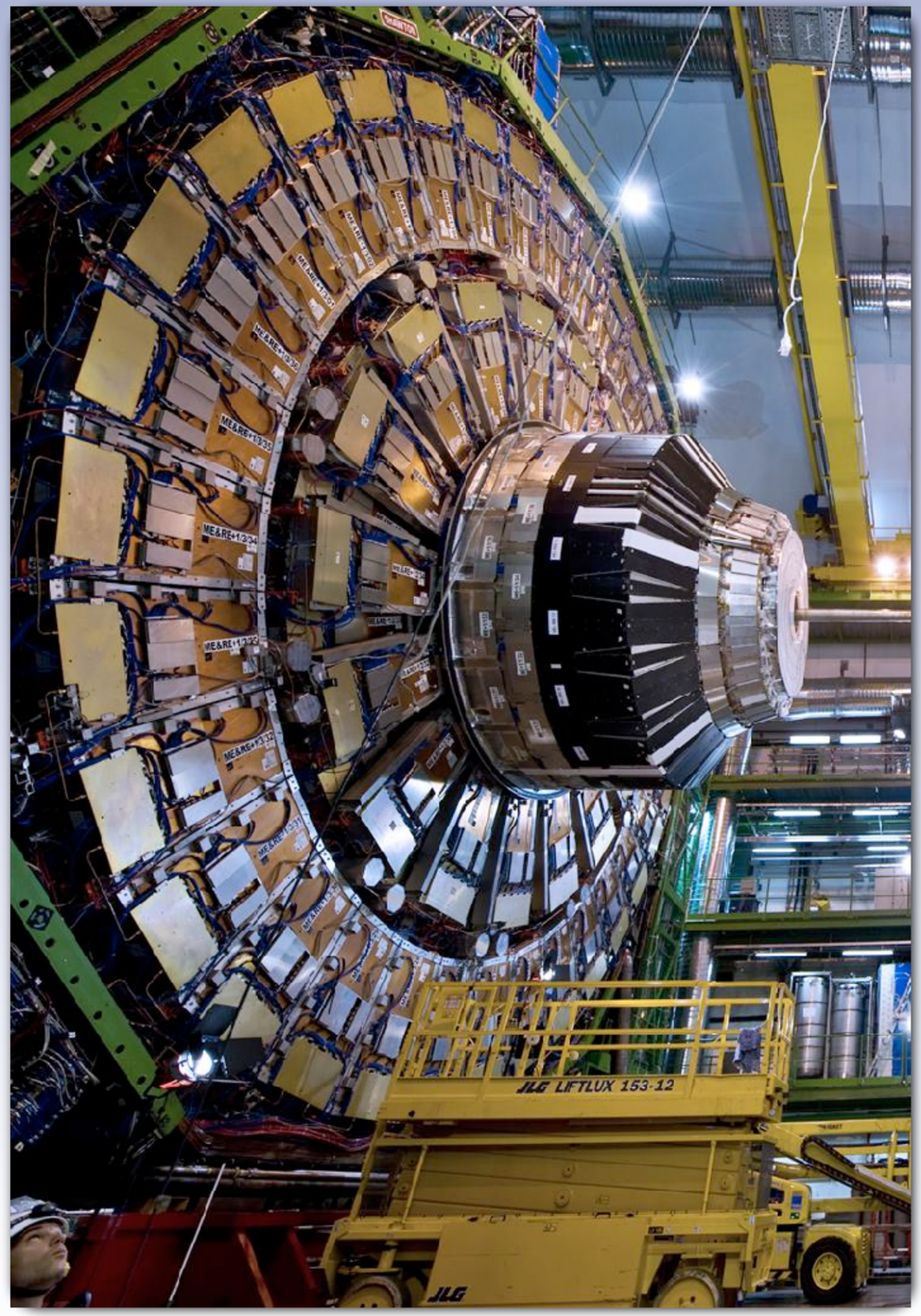
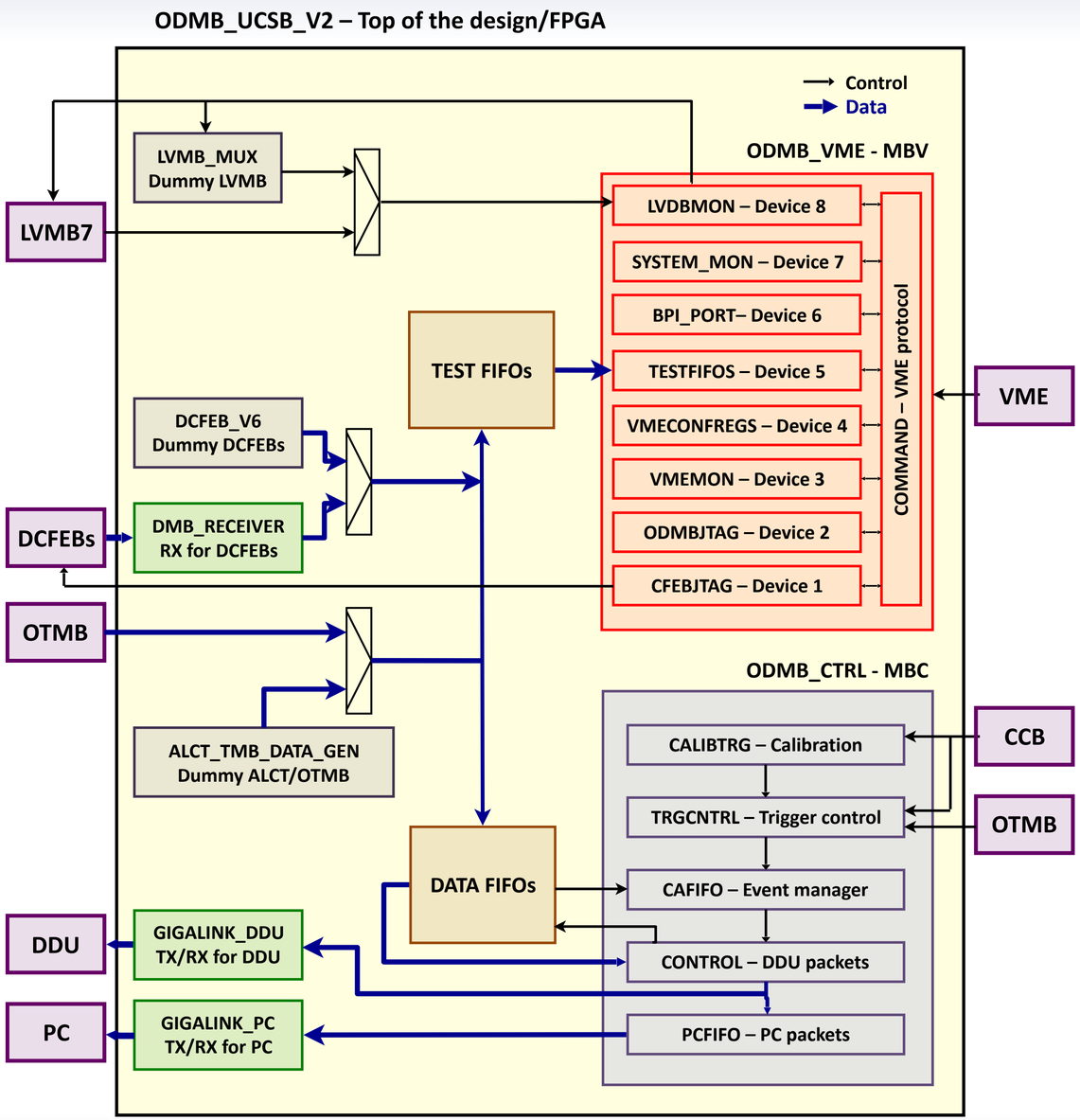


Lecture #5: VME devices in ODMB firmware

Manuel Franco Sevilla
UC Santa Barbara

19th June 2018
Richman group meeting

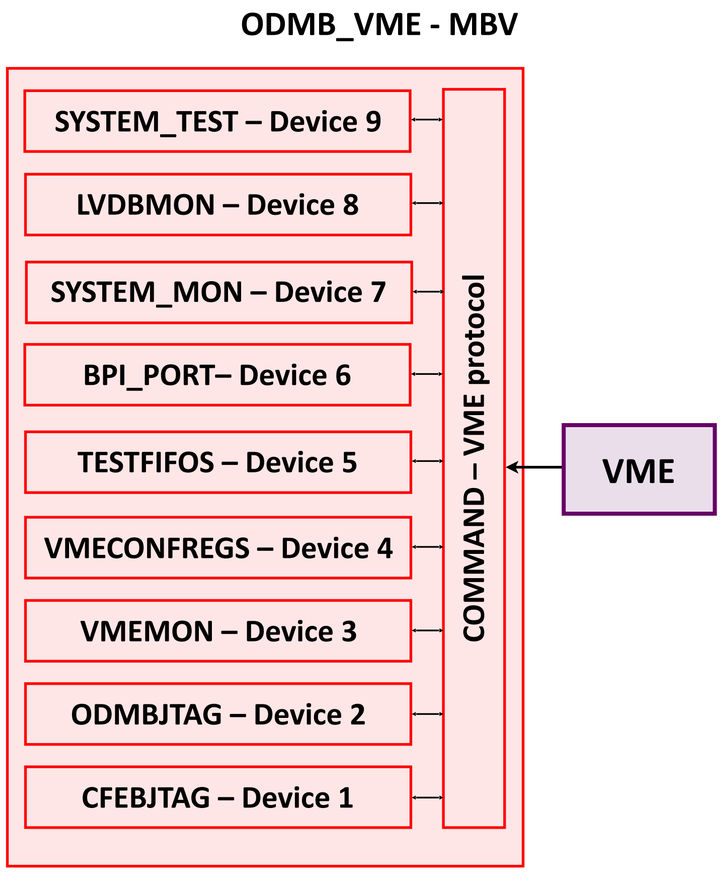
Outline



1. **Device 1: CFEBJTAG**
   - ✓ Detailed description because new features are needed
2. **Device 2: ODMBJTAG**
   - ✓ Brief overview
3. **Device 3: VMEMON**
   - ✓ Brief overview
4. **Device 4: CONFREGS**
   - ✓ Detailed but incomplete description

VME devices

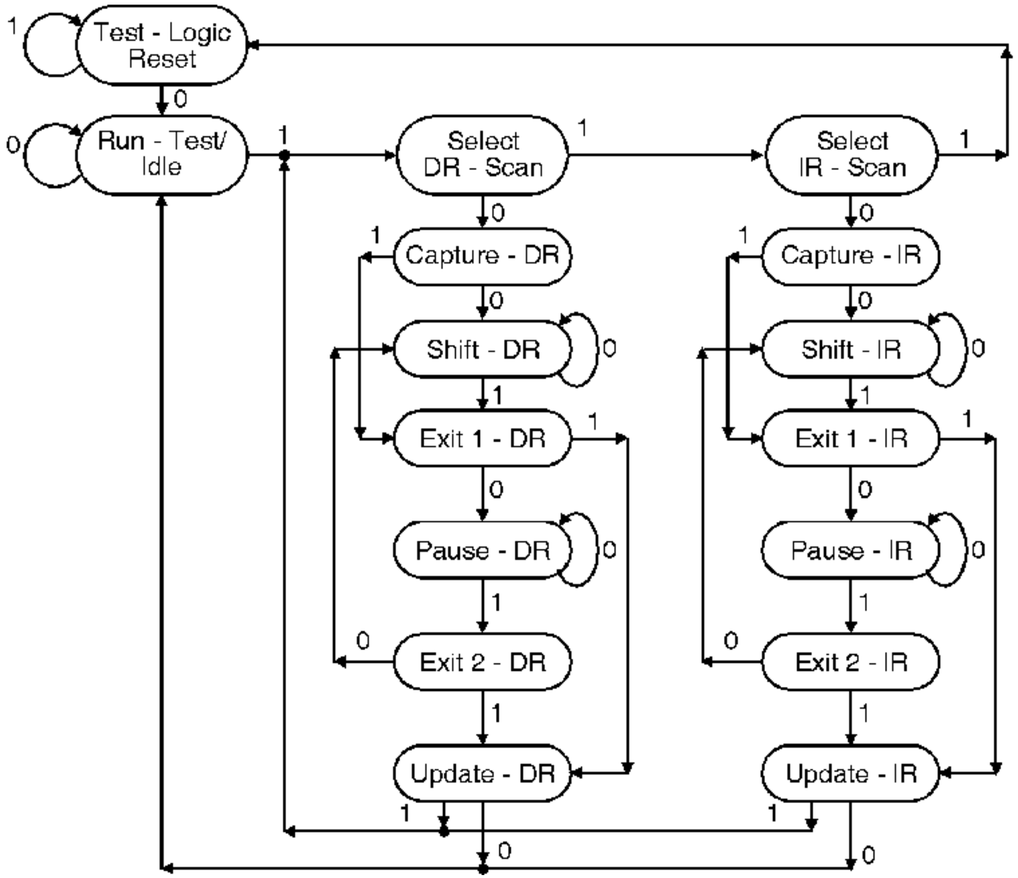
Overview of VME devices



- **Device 1** (`cfebjtag.vhd`): DCFEB JTAG
- **Device 2** (`odmbjtag.vhd`): ODMB JTAG
  - JTAG communication with own FPGA on ODMB
- **Device 3** (`vmemon.vhd`): ODMB/DCFEB control
  - Registers that allow experts to send resets and pulses, set multiplexers for dummy/real data, and read counters
- **Device 4** (`vmeconfregs.vhd`): Configuration registers
- **Device 5** (`testfifos.vhd`): Test FIFOs
  - Debug memories that store the last data received/sent from/to DCFEBs, DDU, PC
- **Device 6** (`bpi_port`): BPI Interface
  - Commands to read/write to the PROM (flash memory with configuration and parameters
- **Device 7** (`system_mon.vhd`): ODMB monitoring
  - Reports voltages and temperatures on the board
- **Device 8** (`lvdbmon.vhd`): Low voltage monitoring
  - Commands to control and read the LVMB7 mezzanine card on the LVDB7
- **Device 9** (`system_test.vhd`): System tests
  - High statistics tests for production quality control

Device 1: DCFEB JTAG

- Handles JTAG communication with DCFEBs
  - Read/write registers, send commands



"Y" refers to the number of bits to be shifted minus 1

Instruction	Description
W 1Y00	Shift Data; no TMS header; no TMS tailer
W 1Y04	Shift Data with TMS header only
W 1Y08	Shift Data with TMS tailer only
W 1Y0C	Shift Data with TMS header & TMS tailer
R 1014	Read TDO register
W 1018	Resets JTAG protocol to IDLE state (data sent with this command is disregarded)
W 1Y1C	Shift Instruction register
W 1020	Select DCFEB, one bit per DCFEB
R 1024	Read which DCFEB is selected

Example: Read DCFEB UserCode

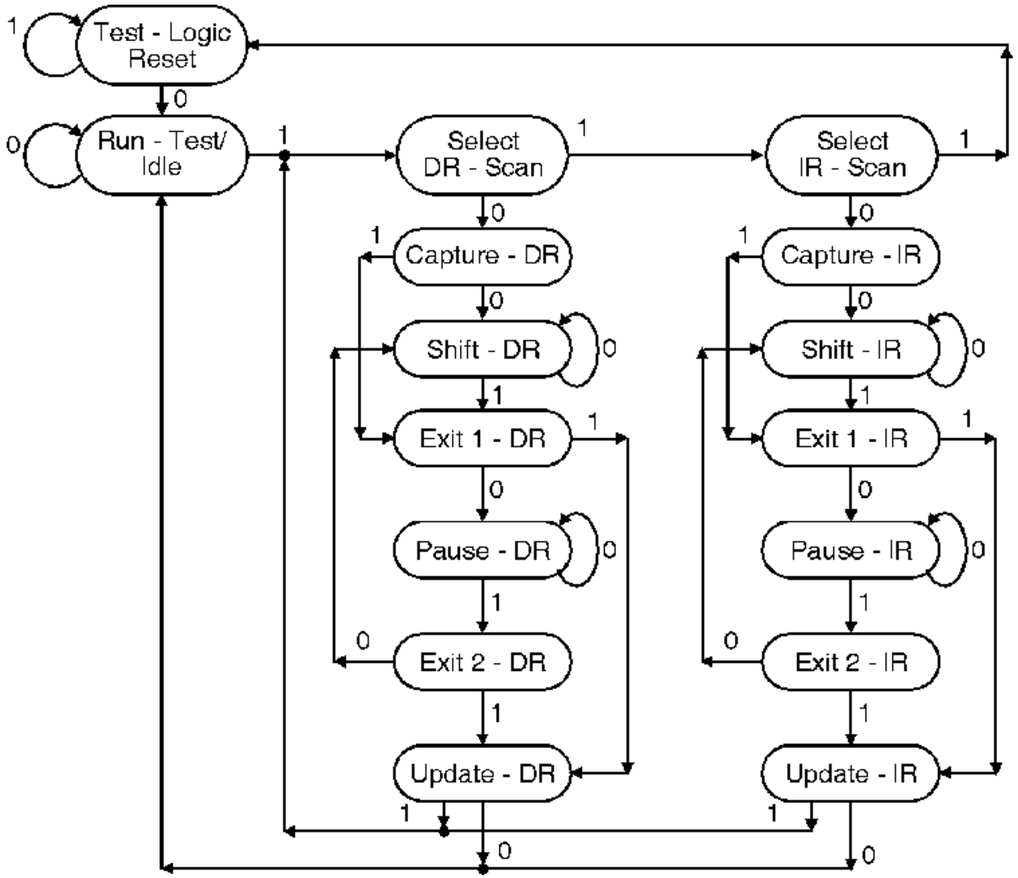
DCFEB registers are set and read via JTAG. The following procedure reads the 32-bit USERID of DCFEB 3:

```
W  1020    4      Select DCFEB 3 (one bit per DCFEB)

W  191c  3C8      Set instruction register to 3C8 (read UserCode)
W  1F04    0      Shift 16 lower bits
R  1014    0      Read last 16 shifted bits (DBDB)
W  1F08    0      Shift 16 upper bits
R  1014    0      Read last 16 shifted bits (XYZK)
```

Device 1: features needed

- Handles JTAG communication with DCFEBs
  - Read/write registers, send commands



"Y" refers to the number of bits to be shifted minus 1

Instruction	Description
W 1Y00	Shift Data; no TMS header; no TMS tailer
W 1Y04	Shift Data with TMS header only
W 1Y08	Shift Data with TMS tailer only
W 1Y0C	Shift Data with TMS header & TMS tailer
R 1014	Read TDO register
W 1018	Resets JTAG protocol to IDLE state (data sent with this command is disregarded)
W 1Y1C	Shift Instruction register
W 1020	Select DCFEB, one bit per DCFEB
R 1024	Read which DCFEB is selected

- **Two features to be added**

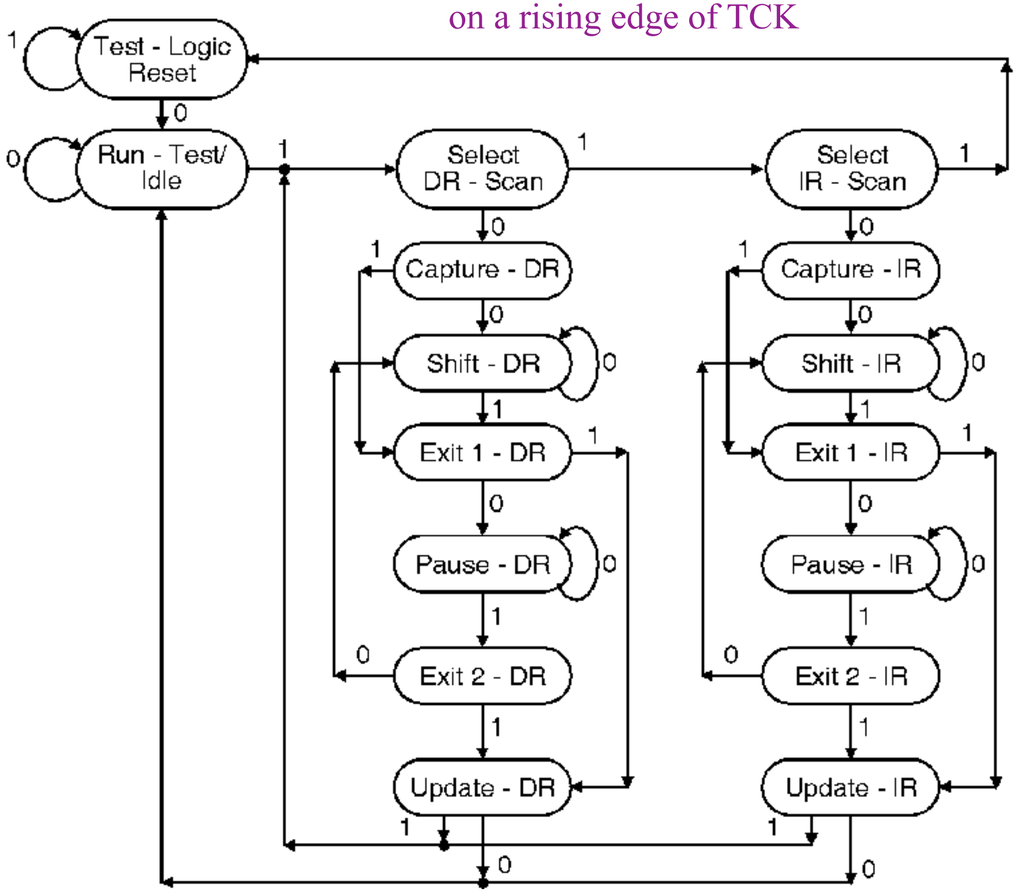
W 1Y30	Shift Instruction; no TMS header; no TMS tailer
W 1Y34	Shift Instruction with TMS header only
W 1Y38	Shift Instruction with TMS tailer only
W 1Y3C	Shift Instruction with TMS header & TMS tailer
W 1Y48	Shift Instruction; no TMS header; special TMS tailer to go directly to data
W 1Y4C	Shift Instruction with TMS header; special TMS tailer to go directly to data

Same flexibility for shifting instruction register as with data register

Ability to go from "Update - IR" directly to "Select DR - Scan"

Device 1: Example, data shift

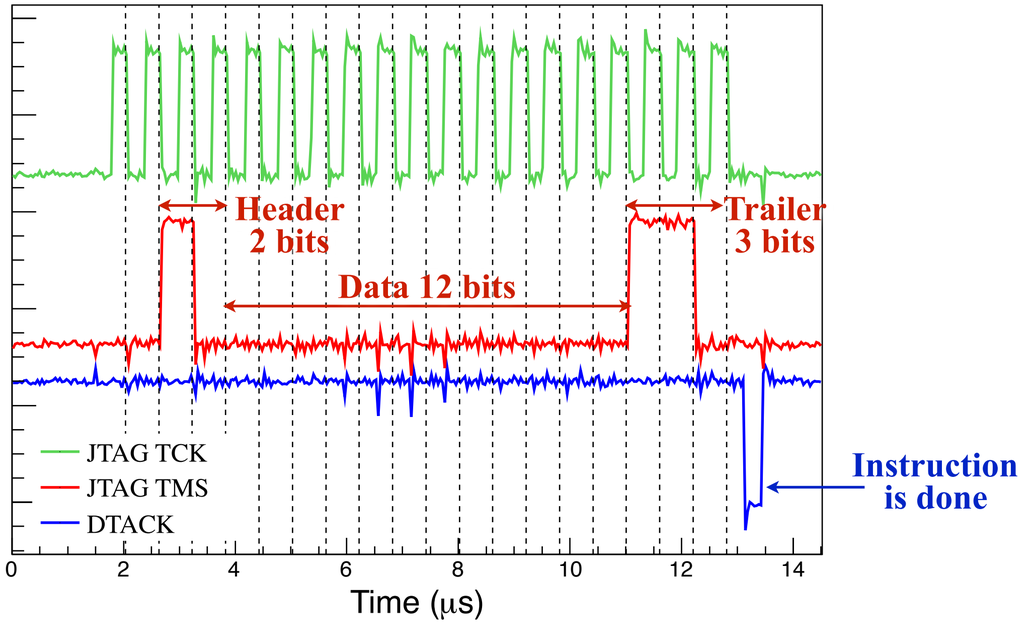
Transitions controlled by value of TMS
on a rising edge of TCK



"Y" refers to the number of bits to be shifted

Instruction	Description
`W 1Y00`	Shift Data; no TMS header; no TMS tailer
`W 1Y04`	Shift Data with TMS header only
`W 1Y08`	Shift Data with TMS tailer only
`W 1Y0C`	Shift Data with TMS header & TMS tailer

- Sending data with 0x1B0C (12 bits shift)



cfebjtag.vhd code

- This device is **very important**, but **code is** somewhat **difficult to follow**
  - Written before my time, directly translated from DMB schematics using sequential logic
  - Updated it to concurrent logic, but **no time to change Latches_Flipflops to the official unisim components**

```
use work.Latches_Flipflops.all;

FDCE(D, C, CE, CLR, Q);
```

Only used in cfebjtag.vhd, odmbjtag.vhd, odmb_ctrl.vhd

```
library unisim;
use unisim.vcomponents.all;

FDCE_ex : FDCE port map(Q, C, CE, CLR, D);
```

Note the different ordering of the ports!

- **Lots of confusing code**, such as the instruction decoder

```
-- COMMAND DECODER
CMDHIGH  <= '1' when (DEVICE = '1' and COMMAND(5) = '0' and COMMAND(4) = '0') else '0';
CMDDEV   <= CMDHIGH & COMMAND(3) & COMMAND(2) & COMMAND(1) & COMMAND(0);
DATASHFT <= '1' when (CMDDEV(4 downto 2) = "100")                             else '0';
INSTSHFT <= '1' when (CMDDEV = "10111")                                       else '0';
READTDO  <= '1' when (CMDDEV = "10101")                                       else '0';
SELCFEB  <= '1' when (CMDDEV = "11000")                                       else '0';
READCFEB <= '1' when (CMDDEV = "11001")                                       else '0';
RSTJTAG  <= '1' when (CMDDEV = "10110")                                       else '0';
```

Other modules

```
-- Decode instruction
CMDDEV <= "000" & DEVICE & COMMAND & "00";
DATASHFT <= '1' when (DEVICE = '1' and CMDDEV(7 downto 4) = x"0")  else '0';
INSTSHFT <= '1' when (DEVICE = '1' and CMDDEV(7 downto 0) = x"1C") else '0';
READTDO  <= '1' when (DEVICE = '1' and CMDDEV(7 downto 0) = x"14") else '0';
RSTJTAG  <= '1' when (DEVICE = '1' and CMDDEV(7 downto 0) = x"18") else '0';
```

Easier to read instruction when translated to hex

`cfebjtag.vhd` code: TCK generation

Up to 7 DCFEBs are selected with instruction 0x1020.

This instruction generates `SELCFEB`

Instruction	Description
`W 1020`	Select DCFEB, one bit per DCFEB
`R 1024`	Read which DCFEB is selected

```
-- COMMAND DECODER
CMDHIGH  <= '1' when (DEVICE = '1' and COMMAND(5) = '0' and COMMAND(4) = '0') else '0';
CMDDEV   <= CMDHIGH & COMMAND(3) & COMMAND(2) & COMMAND(1) & COMMAND(0);
DATASHFT <= '1' when (CMDDEV(4 downto 2) = "100")                             else '0';
INSTSHFT <= '1' when (CMDDEV = "10111")                                       else '0';
READTDO  <= '1' when (CMDDEV = "10101")                                       else '0';
SELCFEB  <= '1' when (CMDDEV = "11000")                                       else '0';
```

cfebjtag.vhd code: TCK generation

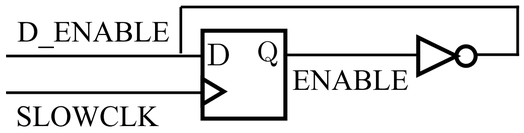
Up to 7 DCFEBs are selected with instruction 0x1020.

This instruction generates SELCFEB

Instruction	Description
W 1020	Select DCFEB, one bit per DCFEB
R 1024	Read which DCFEB is selected

```
-- COMMAND DECODER
CMDHIGH  <= '1' when (DEVICE = '1' and COMMAND(5) = '0' and COMMAND(4) = '0') else '0';
CMDDEV   <= CMDHIGH & COMMAND(3) & COMMAND(2) & COMMAND(1) & COMMAND(0);
DATASHFT <= '1' when (CMDDEV(4 downto 2) = "100")                             else '0';
INSTSHFT <= '1' when (CMDDEV = "10111")                                       else '0';
READTDO  <= '1' when (CMDDEV = "10101")                                       else '0';
SELCFEB  <= '1' when (CMDDEV = "11000")                                       else '0';
```

ENABLE is a clock twice as slow as SLOWCLK




```
-- Generate ENABLE
CE_ENABLE <= '1' when (RESETJTAG = '1' or BUSY = '1') else '0';
D_ENABLE  <= not ENABLE;
FDCE(D_ENABLE, SLOWCLK, CE_ENABLE, RST, ENABLE);
```

SELFEB is a **7 bit signal** loaded **from INDATA** that indicates **which DCFEBs are selected.**

The **"and" of SELFEB and ENABLE generates TCK**

```
-- Write SELFEB when SELCFEB=1
FDPE(INDATA(0), STROBE, SELCFEB, rst_init, SELFEB(1));
FDPE(INDATA(1), STROBE, SELCFEB, rst_init, SELFEB(2));
FDPE(INDATA(2), STROBE, SELCFEB, rst_init, SELFEB(3));
FDPE(INDATA(3), STROBE, SELCFEB, rst_init, SELFEB(4));
FDPE(INDATA(4), STROBE, SELCFEB, rst_init, SELFEB(5));
FDPE(INDATA(5), STROBE, SELCFEB, rst_init, SELFEB(6));
FDPE(INDATA(6), STROBE, SELCFEB, rst_init, SELFEB(7));
```

```
-- Generate TCK
TCK(1) <= SELFEB(1) and ENABLE;
TCK(2) <= SELFEB(2) and ENABLE;
TCK(3) <= SELFEB(3) and ENABLE;
TCK(4) <= SELFEB(4) and ENABLE;
TCK(5) <= SELFEB(5) and ENABLE;
TCK(6) <= SELFEB(6) and ENABLE;
TCK(7) <= SELFEB(7) and ENABLE;
```

cfebjtag.vhd code: data header

"Y" refers to the number of bits to be shifted

Instruction	Description
W 1Y00	Shift Data; no TMS header; no TMS tailer
W 1Y04	Shift Data with TMS header only
W 1Y08	Shift Data with TMS tailer only
W 1Y0C	Shift Data with TMS header & TMS tailer

DATASHFT: initiates all data shift instructions

DHEADEN: Data Header Enable generated by DATASHFT

SHDHEAD: Shift Data Header generated by DHEADEN

TMS: flip-flops send "00100" started by SHDHEAD.
FDCE reset to 0 and FDPE preset to 1

DONEDHEAD: Done Data Header. High after 5 clock cycles when the header is finished

```
-- COMMAND DECODER
DATASHFT <= '1' when (CMDDEV(4 downto 2) = "100") else '0';

-- generate DHEADEN
C_DHEADEN   <= '1' when (STROBE = '1' and BUSY = '0')  else '0';
CLR_DHEADEN <= '1' when (RST = '1' or DONEDHEAD = '1') else '0';
FDCE(COMMAND(0), C_DHEADEN, DATASHFT, CLR_DHEADEN, DHEADEN);

-- Generate SHDHEAD
SHDHEAD <= '1' when (BUSY = '1' and DHEADEN = '1') else '0';

-- Generate TMS when SHDHEAD=1
CE_SHDHEAD_TMS <= '1'              when ((SHDHEAD = '1') and (ENABLE = '1')) else '0';
FDCE(Q5_SHDHEAD_TMS, SLOWCLK, CE_SHDHEAD_TMS, RST, Q1_SHDHEAD_TMS);
FDCE(Q1_SHDHEAD_TMS, SLOWCLK, CE_SHDHEAD_TMS, RST, Q2_SHDHEAD_TMS);
FDPE(Q2_SHDHEAD_TMS, SLOWCLK, CE_SHDHEAD_TMS, RST, Q3_SHDHEAD_TMS);
FDCE(Q3_SHDHEAD_TMS, SLOWCLK, CE_SHDHEAD_TMS, RST, Q4_SHDHEAD_TMS);
FDCE(Q4_SHDHEAD_TMS, SLOWCLK, CE_SHDHEAD_TMS, RST, Q5_SHDHEAD_TMS);
TMS <= Q5_SHDHEAD_TMS when (SHDHEAD = '1') else 'Z';

-- Generate DONEDHEAD
R_DONEDHEAD <= '1' when (LOAD = '1' or RST = '1' or Q_DONEDHEAD = '1')       else '0';
CB4RE(SLOWCLK, SHDHEAD, R_DONEDHEAD, QV_DONEDHEAD, QV_DONEDHEAD, CEO_DONEDHEAD, TC_DONEDHEAD);
DONEDHEAD   <= '1' when ((QV_DONEDHEAD(1) = '1')and (QV_DONEDHEAD(3) = '1')) else '0';
FD(DONEDHEAD, SLOWCLK, Q_DONEDHEAD);
```

cfebjtag.vhd code: data shift

LOAD: generated by DATASHFT

BUSY: generated by LOAD

SHDATA: Shifting Data while device busy, header finished, and DONEDATA still not high

TAILEN: enable for the tailer, but it is high from the beginning (LOAD)

DONEDATA: counts backwards with CB4CLED (bidirectional counter) then number of bits specified by COMMAND(9 downto 6) and generates **TMS** (0 while shifting)

```
-- Generate LOAD
D1_LOAD  <= DATASHFT or INSTSHFT;
CLR_LOAD <= LOAD or RST;
FDC(D1_LOAD, STROBE, CLR_LOAD, Q_LOAD);
D2_LOAD  <= '1' when (Q_LOAD = '1' and BUSY = '0') else '0';
FDC(D2_LOAD, SLOWCLK, RST, LOAD);

-- Generate BUSY
FDC(LOAD, SLOWCLK, RST, Q_BUSY);
CLR_BUSY <= '1' when DONEDATA(1) = '1' and TAILEN = '0' or RST = '1' or DONETAIL = '1' else '0';
D_BUSY   <= '1' when Q_BUSY = '1' or BUSY = '1'                                       else '0';
FDC(D_BUSY, SLOWCLK, CLR_BUSY, BUSY);

-- Generate SHDATA
SHDATA  <= '1' when (BUSY = '1' and DHEADEN = '0' and IHEADEN = '0' and DONEDATA(1) = '0') else '0';

-- Generate TAILEN
CE_TAILEN  <= '1' when (INSTSHFT = '1' or DATASHFT = '1') else '0';
CLR_TAILEN <= '1' when (RST = '1' or DONETAIL = '1')      else '0';
FDCE(COMMAND(1), LOAD, CE_TAILEN, CLR_TAILEN, TAILEN);

-- Generate DONEDATA
DV_DONEDATA  <= COMMAND(9 downto 6);
CE_DONEDATA  <= '1' when (SHDATA = '1' and ENABLE = '1')                          else '0';
CLR_DONEDATA <= '1' when (RST = '1' or DONEDATA(1) = '1' or DONEDATA(0) = '1') else '0';
UP_DONEDATA  <= '0';
CB4CLED(SLOWCLK, CE_DONEDATA, CLR_DONEDATA, LOAD, UP_DONEDATA, DV_DONEDATA, QV_DONEDATA,
        QV_DONEDATA, CEO_DONEDATA, TC_DONEDATA);
D_DONEDATA   <= '1' when (QV_DONEDATA = "0000" and LOAD = '0')                    else '0';
FDCE(D_DONEDATA, SLOWCLK, SHDATA, LOAD, DONEDATA(0));
FDC(DONEDATA(0), SLOWCLK, LOAD, DONEDATA(1));

-- Generate TMS when SHDATA=1
TMS <= (TAILEN and D_DONEDATA) when (SHDATA = '1') else 'Z';
```

cfebjtag.vhd code: data tailer

SHTAIL: shift tailer, generated by TAILEN and the end of data shifting (DONEDATA(1) = 1)

```
-- Generate SHTAIL
SHTAIL <= '1' when (BUSY = '1' and DONEDATA(1) = '1' and TAILEN = '1') else '0';
```

TMS: flip-flops send "10" started by SHTAIL

```
-- Generate TMS when SHTAIL=1
CE_SHTAIL_TMS <= '1'  when ((SHTAIL = '1') and (ENABLE = '1')) else '0';
FDCE(Q2_SHTAIL_TMS, SLOWCLK, CE_SHTAIL_TMS, RST, Q1_SHTAIL_TMS);
FDPE(Q1_SHTAIL_TMS, SLOWCLK, CE_SHTAIL_TMS, RST, Q2_SHTAIL_TMS);
TMS  <= Q2_SHTAIL_TMS when (SHTAIL = '1') else 'Z';
```

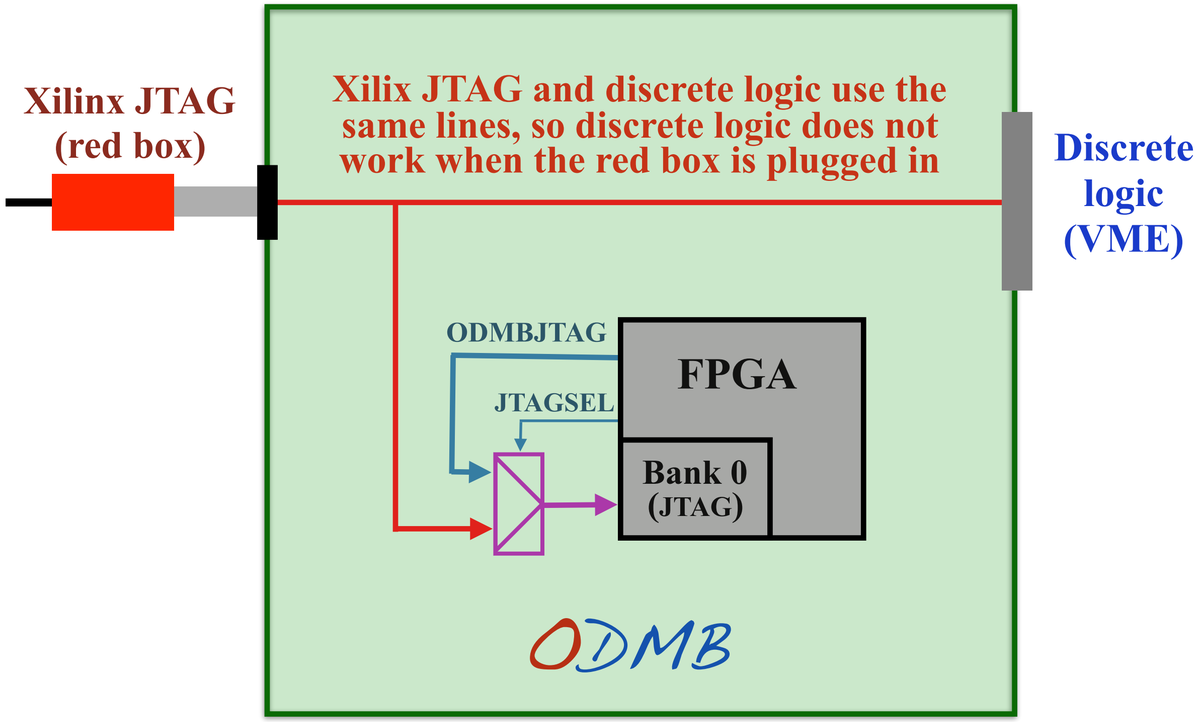
DONETAIL: high after two clock cycles of SLOWCLK, but only enabled half of the time due to ENABLE

```
-- Generate DONETAIL
CE_DONETAIL  <= '1' when (SHTAIL = '1' and ENABLE = '1') else '0';
CLR_DONETAIL <= '1' when (RST = '1' or Q_DONETAIL = '1') else '0';
CB4CE(SLOWCLK, CE_DONETAIL, CLR_DONETAIL, QV_DONETAIL, QV_DONETAIL, CEO_DONETAIL, TC_DONETAIL);
DONETAIL     <= QV_DONETAIL(1);
C_DONETAIL   <= SLOWCLK;
FD_1(DONETAIL, C_DONETAIL, Q_DONETAIL);
```

Shifting of instruction registers is very similar to data shifting

Device 2: ODMB JTAG

- Similar code to **`cfebjtag.vhd`** but used to talk to the ODMB FPGA
  - Allows to access bank 0 registers like UserCode
  - Generally not very useful, as the FPGA cannot program itself
    - Programming only from red box or discrete logic



"Y" refers to the number of bits to be shifted minus 1

Instruction	Description
W 2Y00	Shift Data; no TMS header; no TMS tailer
W 2Y04	Shift Data with TMS header only
W 2Y08	Shift Data with TMS tailer only
W 2Y0C	Shift Data with TMS header & TMS tailer
R 2014	Read TDO register
W 2018	Resets JTAG protocol to IDLE state (data sent with this command is disregarded)
W 2Y1C	Shift Instruction register
W 2020	Change polarity of V6_JTAG_SEL

Example: Read ODMB UserCode

Read FPGA UserCode:

```
W  291C  3C8     Set instruction register to 3C8 (read UserCode)
W  2F04    0     Shift 16 lower bits
R  2014    0     Read last 16 shifted bits (DBDB)
W  2F08    0     Shift 16 upper bits
R  2014    0     Read last 16 shifted bits (XYZK)
```

Device 3: ODMB/DCFEB control

- Device that allows experts to control the ODMB and DCFEBs
  - Sends resets and pulses
  - Set multiplexers for data/triggers
  - Reads counters

Bit specification DCFEB pulses command "W 3200"

- DCFEB_PULSE[0] - Sends INJPLS signal to all DCFEBs.
- DCFEB_PULSE[1] - Sends EXTPLS signal to all DCFEBs.
- DCFEB_PULSE[2] - Sends test L1A and L1A_MATCH to non-killed DCFEBs.
- DCFEB_PULSE[3] - Sends LCT request to OTMB.
- DCFEB_PULSE[4] - Sends external trigger request to OTMB.
- DCFEB_PULSE[5] - Sends BC0 to all DCFEBs.

Calibration pulses and ODMB-generated triggers

Information accessible via command "R 3YZC"

Trigger and packet counters

- YZ = 3F: Least significant 16 bits of L1A_COUNTER (also MSP in 33AC and LSP in 33BC)
- YZ = 5F: Least significant 16 bits of L1A_COUNTER (only reset by hard resets, no RESYNCs)
- YZ = 71-77: Number of LCTs for given DCFEB
- YZ = 78: Number of OTMBDAVs (available OTMB packets)
- YZ = 79: Number of ALCTDAVs (available ALCT packets)
- YZ = 21-29: Number of L1A_MATCHes for given DCFEB, OTMB, ALCT
- YZ = 41-49: Number of packets received for given DCFEB, TMB, or ALCT
- YZ = 4A: Number of packets sent to the DDU
- YZ = 4B: Number of packets sent to the PC
- YZ = 51-59: Number of packets shipped to DDU and PC for given DCFEB, TMB, or ALCT
- YZ = 61-67: Number of data packets received with good CRC for given DCFEB

ODMB counters (more in the manual)

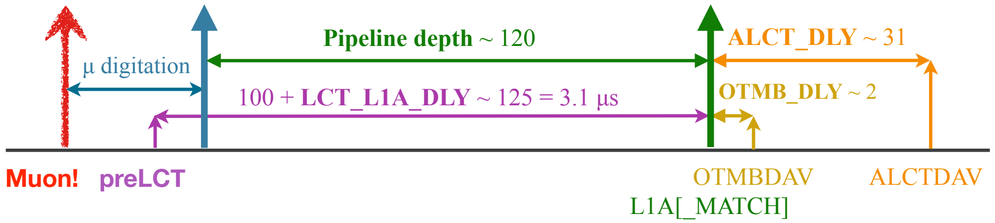
Instruction	Description
W/R 3000	0 → nominal mode, 1 → calibration mode (ODMB generates L1A with every pulse)
W 3004	ODMB soft reset
W 3008	ODMB optical reset
W 3010	Reprograms all DCFEBs
W 3014	L1A reset and DCFEB RESYNC
W/R 3020	TP_SEL register (selects which signals are sent to TP31, TP35, TP41, TP45)
W/R 3024	Number of words in DCFEB packet before autokill. 1024 by default and after each reset.
W/R 3100	LOOPBACK: 0 → no loopback, 1 or 2 → internal loopback
R 3110	DIFFCTRL (TX voltage swing): 0 → minimum ~100 mV, F → maximum ~1100mV
R 3120	Read DONE bits from DCFEBs (7 bits)
R 3124	Read if QPLL is locked
W 3200	Sends pulses to DCFEBs (see below)
W/R 3300	Data multiplexer: 0 → real data, 1 → dummy data
W/R 3304	Trigger multiplexer: 0 → external triggers, 1 → internal triggers
W/R 3308	LVMB multiplexer: 0 → real LVMB, 1 → dummy LVMB
W/R 3400	0 → normal, 1 → pedestal (L1A_MATCHes sent to DCFEBs for each L1A).
W/R 3404	0 → normal, 1 → OTMB data requested for each L1A (requires special OTMB FW)
W/R 3408	Bit 0 → kills L1A. Bits 1-7 → kills L1A_MATCHes
W/R 340C	MASK_PLS: 0 → normal, 1 → no EXTPLS/INJPLS (for non-pulsed pedestals from CCB)
R 3YZC	Read ODMB_DATA corresponding to selection YZ (see below)

Device 4: configuration registers

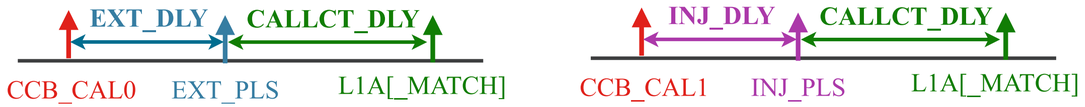
- Writes and reads configuration registers that can be written and read out from the PROM
  - Mostly delays, and also KILL mask, crate ID, ODMB ID, and firmware tag
- Only code that uses **Triple Modular Redundancy (TMS) or *Triple voting* for radiation hardness**
  - Creates 3 copies of each register
  - If 1 copy gets corrupted, the other 2 still win in the vote

Delay diagrams

1. **LCT_L1A_DLY**, **OTMB_DLY**, and **ALCT_DLY** match preLCT, OTMBDAV, and ALCTDAV to L1A, respectively



2. **EXT_DLY**/**INJ_DLY** set the distance between the CCB signals and the pulses. **CALLCT_DLY** sets the distance between the pulses and the L1A/L1A_MATCHes



Instruction	Description
W/R 4000	LCT_L1A_DLY[5:0] → Set to LCT/L1A gap - 100
W/R 4004	OTMB_DLY[5:0] → Set to L1A/OTMBDAV gap read with “R 338C”
W/R 4008	CABLE_DLY[0:0] → Delays sending L1A[_MATCH], RESYNC, BCO by 25 ns
W/R 400C	ALCT_DLY[5:0] → Set to L1A/ALCTDAV gap read with “R 339C”
W/R 4010	INJ_DLY[4:0] - Delay: 12.5*INJ_DLY [ns]
W/R 4014	EXT_DLY[4:0] - Delay: 12.5*EXT_DLY [ns]
W/R 4018	CALLCT_DLY[3:0] - Delay: 25*CALLCT_DLY [ns]
W/R 401C	KILL[9:1] (ALCT + TMB + 7 DCFEBs)
W/R 4020	CRATEID[6:0]
W/R 4028	Number of words generated by dummy DCFEBs, OTMB, and ALCT
R 4100	Read ODMB unique ID (if not set request UCSB to write it)
R 4200	Read firmware version
R 4300	Read firmware build
R 4400	Read month/day firmware was synthesized
R 4500	Read year firmware was synthesized

Configurable registers

Constant registers

vmeconfregs.vhd code: cfg registers

Instruction	Description
W/R 4000	LCT_L1A_DLY[5:0] → Set to LCT/L1A gap - 100
W/R 4004	OTMB_DLY[5:0] → Set to L1A/OTMBDAV gap read with “R 338C”
W/R 4008	CABLE_DLY[0:0] → Delays sending L1A[_MATCH], RESYNC, BCO by 25 ns
W/R 400C	ALCT_DLY[5:0] → Set to L1A/ALCTDAV gap read with “R 339C”
W/R 4010	INJ_DLY[4:0] - Delay: 12.5*INJ_DLY [ns]
W/R 4014	EXT_DLY[4:0] - Delay: 12.5*EXT_DLY [ns]
W/R 4018	CALLCT_DLY[3:0] - Delay: 25*CALLCT_DLY [ns]
W/R 401C	KILL[9:1] (ALCT + TMB + 7 DCFEBs)
W/R 4020	CRATEID[6:0]
W/R 4028	Number of words generated by dummy DCFEBs, OTMB, and ALCT

2D array `cfg_regs` contains configuration registers

Each **index** in **`cfg_regs` corresponds** to **one configuration register of 16 bits**

```
type cfg_regs_array is array (0 to 15) of std_logic_vector(15 downto 0);
signal cfg_regs       : cfg_regs_array;

LCT_L1A_DLY   <= cfg_regs(0)(5 downto 0);                          -- 0x4000
OTMB_PUSH_DLY <= to_integer(unsigned(cfg_regs(1)(5 downto 0)));    -- 0x4004
CABLE_DLY     <= to_integer(unsigned'("" & cfg_regs(2)(0)));       -- 0x4008
ALCT_PUSH_DLY <= to_integer(unsigned(cfg_regs(3)(5 downto 0)));    -- 0x400C
INJ_DLY       <= cfg_regs(4)(4 downto 0);                          -- 0x4010
EXT_DLY       <= cfg_regs(5)(4 downto 0);                          -- 0x4014
CALLCT_DLY    <= cfg_regs(6)(3 downto 0);                          -- 0x4018
KILL          <= cfg_regs(7)(NFEB+1 downto 0);                     -- 0x401C
CRATEID       <= cfg_regs(8)(7 downto 0);                          -- 0x4020
-- 0x4024 reserved for FW version
NWORDS_DUMMY  <= cfg_regs(10)(15 downto 0);                        -- 0x4028
BX_DLY        <= to_integer(unsigned(cfg_regs(11)(11 downto 0)));  -- 0x402C
```

vmeconfregs.vhd code: writing

- Three ways of writing
  - CHANGE_REG_INDEX: Internal condition in code
    - eg. auto-killed DCFEBs due to fiber errors
  - BPI_CFG_UL_PULSE: read from PROM
  - VME command

```
cfg_reg_we <= CHANGE_REG_INDEX when CHANGE_REG_INDEX < NREGS else
              CC_CFG_REG_WE when BPI_CFG_UL_PULSE = '1' else
              vme_cfg_reg_we;

cfg_reg_in <= CHANGE_REG_DATA when CHANGE_REG_INDEX < NREGS else
              CC_CFG_REG_IN when BPI_CFG_UL_PULSE = '1' else
              INDATA;
```

```
cfg_reg_proc : process (RST, CLK, cfg_reg_we, cfg_reg_in, cfg_regs)
begin
  for i in 0 to NREGS-1 loop
    for j in 0 to 2 loop
      if (RST = '1') then
        cfg_reg_triple(i)(j) <= cfg_reg_init(i);
      elsif (rising_edge(CLK) and cfg_reg_we = i and cfg_reg_mask_we(i) = '1') then
        cfg_reg_triple(i)(j) <= cfg_reg_in;
      else
        cfg_reg_triple(i)(j) <= cfg_regs(i);
      end if;
    end loop;
  end loop;
end process;
```

`vmeconfregs.vhd` code: triple voting

- Only code that uses **Triple Modular Redundancy (TMS) or *Triple voting* for radiation hardness**
  - Creates 3 copies of each register
  - If 1 copy gets corrupted, the other 2 still win in the vote

```
GEN_CFG_TRIPLEVOTING : for ind in 0 to NREGS-1 generate
begin
  GEN_TRIPLEBITS : for ibit in 0 to 15 generate
  begin
    cfg_regs(ind)(ibit) <= (cfg_reg_triple(ind)(0)(ibit) and cfg_reg_triple(ind)(1)(ibit)) or
                           (cfg_reg_triple(ind)(1)(ibit) and cfg_reg_triple(ind)(2)(ibit)) or
                           (cfg_reg_triple(ind)(2)(ibit) and cfg_reg_triple(ind)(0)(ibit));
  end generate GEN_TRIPLEBITS;
end generate GEN_CFG_TRIPLEVOTING;
```