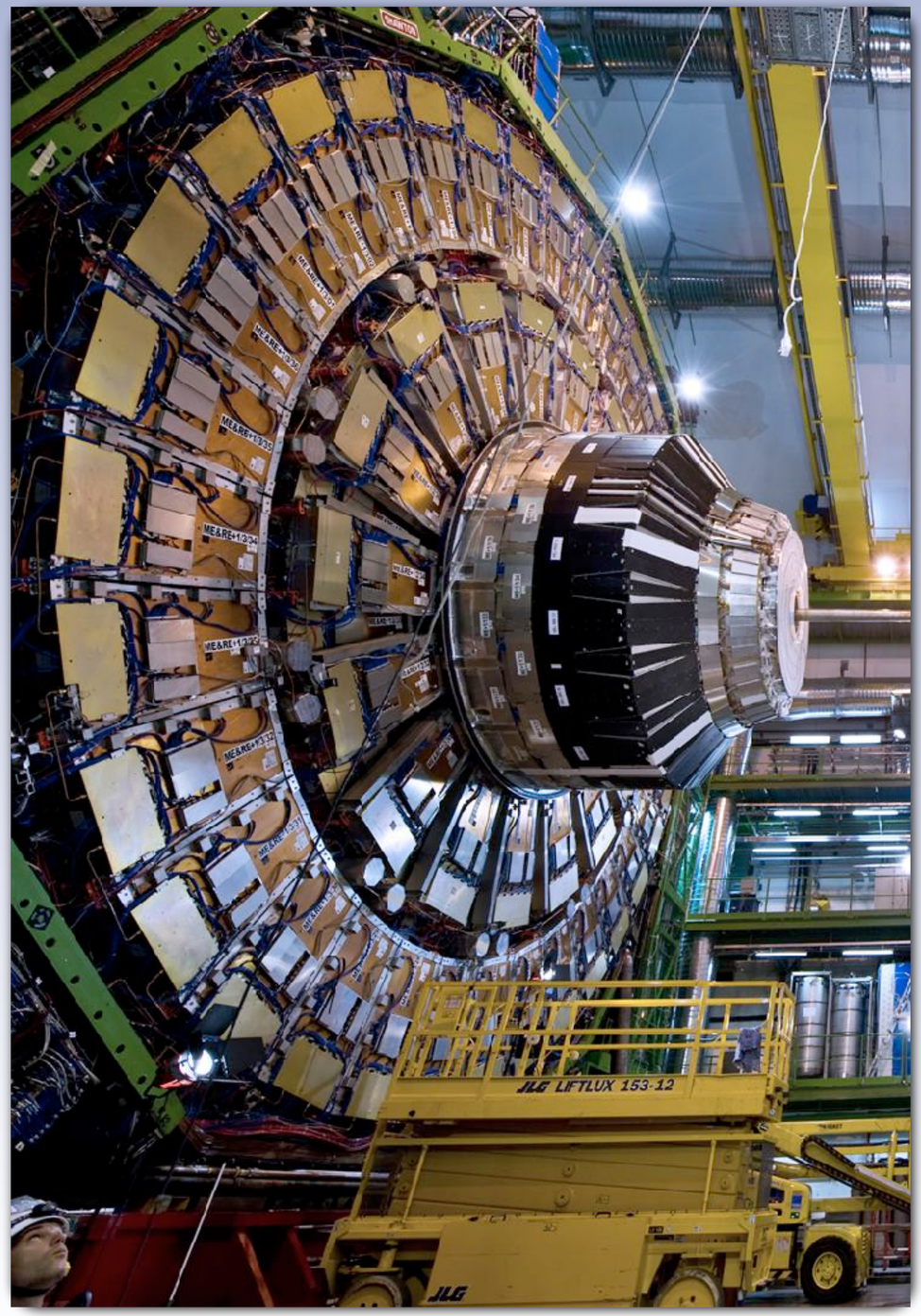
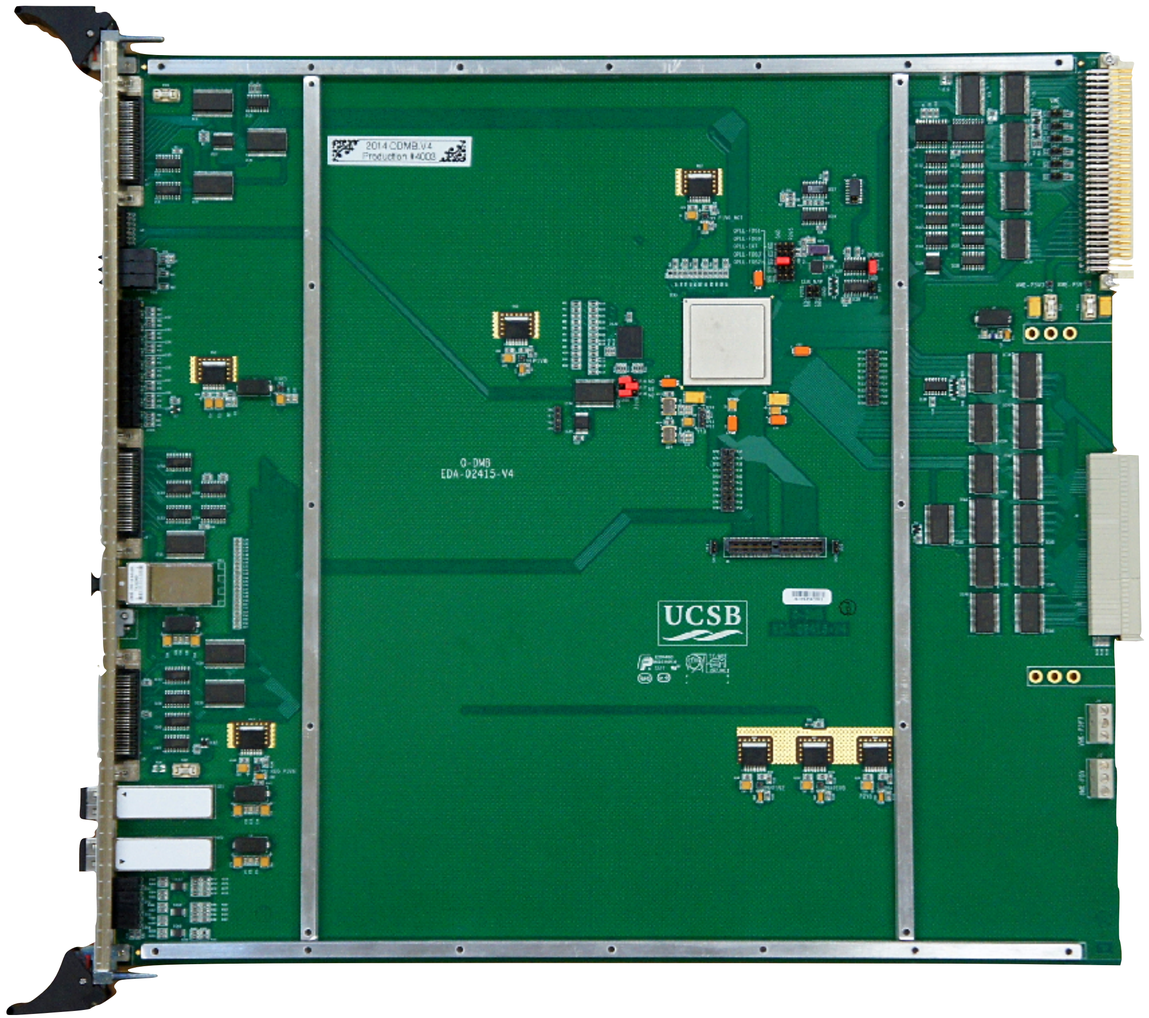


Lecture #4:
ODMB firmware

Manuel Franco Sevilla
UC Santa Barbara

11th May 2018
Richman group meeting

Outline



1. Some firmware tips

- ✓ So that you learn them earlier/easier than I did

2. ODMB firmware

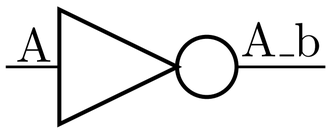
- ✓ Firmware project
- ✓ Functionality of main blocks
- ✓ VME protocol
- ✓ JTAG protocol

Some firmware tips I wish somebody taught me from the beginning

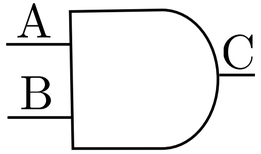
Firmware

- **Low-level software** that directly **controls** a device **specific hardware**
  - ODMB firmware mostly written in VHDL, some in Verilog
  - Good VHDL primer: http://www.seas.upenn.edu/~ese171/vhdl/vhdl_primer.html
- **Code specifies** how **actual logic gates** and **components in FPGA resources are connected**

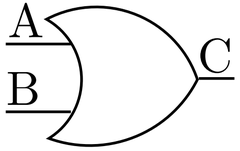
```
A_b <= not A;
```



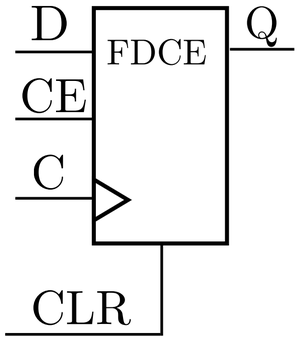

```
C <= A and B;
```



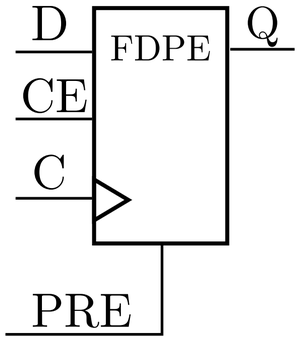

```
C <= A or B;
```




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




```
FDPE_ex : FDPE port map(Q, C, CE, PRE, D);
```



Inputs		FDCE		Outputs
CLR	**CE**	**D**	**C**	**Q**
1	X	X	X	0
0	0	X	X	No Change
0	1	D	↑	D

Inputs		FDPE		Outputs
PRE	**CE**	**D**	**C**	**Q**
1	X	X	X	1
0	0	X	X	No Change
0	1	D	↑	D

Concurrent logic

All** standard **statements happen at the same time

```
bad_dcfeb_pulse160_longpacket(dev) <= bad_dcfeb_longpacket(dev) and not kill(dev);
bad_dcfeb_pulse160_fiber(dev) <= (bad_dcfeb_fiber_q(dev) and not bad_dcfeb_fiber_qq (dev))  and not kill(dev);
bad_dcfeb_pulse160(dev) <= bad_dcfeb_pulse160_longpacket(dev) when IS_SIMULATION = 1 else
                           bad_dcfeb_pulse160_longpacket(dev) or bad_dcfeb_pulse160_fiber(dev);
-- Creating 40 MHz pulse, and long pulse to reset the FIFOs
PULSE_BADDCFEB : PULSE2SLOW port map(bad_dcfeb_pulse(dev), clk40, clk160, reset, bad_dcfeb_pulse160(dev));
PULSE_BADDCFEB_LONG : NPULSE2SAME port map(bad_dcfeb_pulse_long(dev), clk160, reset, 50, bad_dcfeb_pulse160(dev));
```

*Also possible to use **sequential logic using processes***

```
done_fsm_regs : process (done_next_state, pon_reset, clk10khz)
begin
  for dev in 1 to NFEB loop
    if (pon_reset = '1') then
      done_current_state(dev) <= DONE_LOW;
    elsif rising_edge(clk10khz) then
      done_current_state(dev) <= done_next_state(dev);
      if done_cnt_rst(dev) = '1' then
        done_cnt(dev) <= 0;
      elsif done_cnt_en(dev) = '1' then
        done_cnt(dev) <= done_cnt(dev) + 1;
      end if;
    end if;
  end loop;
end process;
```

- Whenever possible, **concurrent logic** is preferred
  - Less overhead and delays, clearer hardware implementation

```
int_tdo(I) <= dcfeb_tdo(I) when (gen_dcfeb_sel = '0') else gen_tdo(I);
```

Concurrent

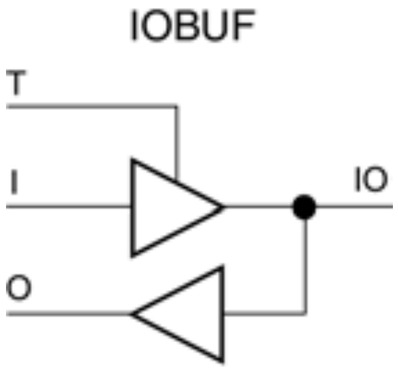
```
TDO_pro : process (gen_dcfeb_sel, dcfeb_tdo, gen_tdo)
begin
  if (gen_dcfeb_sel = '0') then
    int_tdo(I) <= dcfeb_tdo(I);
  else
    int_tdo(I) <= gen_tdo(I);
  end if;
end process;
```

Bad sequential logic

Signal directionality

- **All signals have *direction***: modules have well defined inputs and outputs
  - When trying to understand new code, you look for which signal are driving others

- Possible to change direction, for instance, with IOBUF
  - eg. In ODMB.V2 by mistake DCFEB_TMS/TDI were inputs, and in V3/V4 we made them outputs, so we wrote the following code that works both with V2 and V3/V4

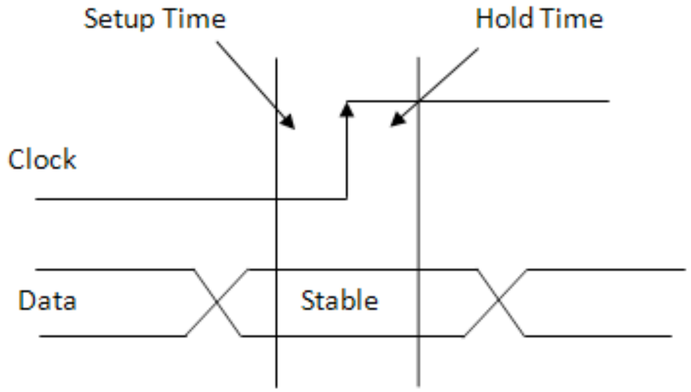



```
is_odmb_v2 <= '1' when (odmb_id(15 downto 12) = x"2") else '0';
BUF_DCFEBTMS : IOBUF port map(O => odmb_tms, IO => DCFEB_TMS, I => dcfeb_tms_out, T => is_odmb_v2);
BUF_DCFEBTDI : IOBUF port map(O => odmb_tdi, IO => DCFEB_TDI, I => dcfeb_tdi_out, T => is_odmb_v2);
```

  - **Single numbers** between **single quotes** (eg. '1') and **multiple numbers** between **double quotes** (eg. "**11001**") are **binary**
  - Numbers between **double quotes** and **preceded by an "x"** are **hexadecimal** (eg. **x"10"** = "00010000")

Timing issues

- Need to be careful with timing issues
  - **Signals** take time to **travel in FPGA**, and each **logic gate introduces a delay (~0.2 ns)**
  - Need to respect the **setup** and **hold times for flip-flops**
  - In the *ODMB*, signal up to 160 MHz → 6.25 ns period, clock edge every 3.125 ns



Xilinx DS152 - Virtex-6 FPGA Data Sheet: DC and Switching Characteristics. ***Table 49***

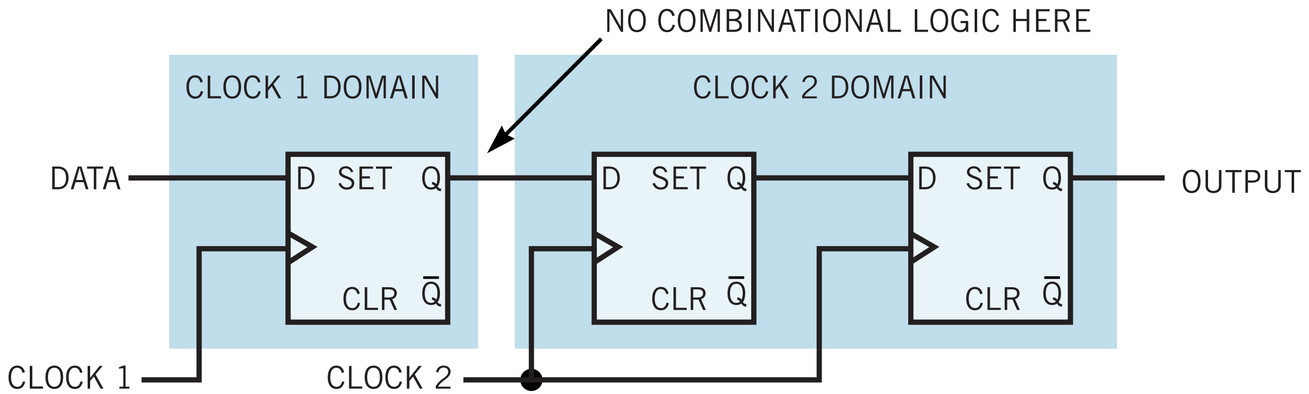
Symbol	Description	Speed Grade				Units
		-3	-2	-1	-1L	
T_{IDOCK}/T_{IOCKD}	D pin Setup/Hold with respect to CLK without Delay	0.07/ 0.41	0.08/ 0.46	0.10/ 0.54	0.11/ 0.64	ns
Combinatorial						
T_{IDI}	D pin to O pin propagation delay, no Delay	0.15	0.17	0.20	0.23	ns

- **Timing errors lead to random behavior, difficult to debug**
  - **We will use synchronous design with most signals changing on edges of clock**
    - **Do no use gated clocks** (flip-flops with non-clocks in the clock input)
  - Based on this paper https://inst.eecs.berkeley.edu/~cs150/sp10/Collections/Papers/ClockCrossing.pdf

Clock synchronizer

- Used to synchronize a **level signal** to a new clock
  - Latency of 2 clock cycles in new clock domain

"Crossing the abyss: asynchronous signals in a synchronous world"
Mike Stein, **designfeature** 59 (2013)



- Implemented in **`source/utils/crossclock.vhd`**

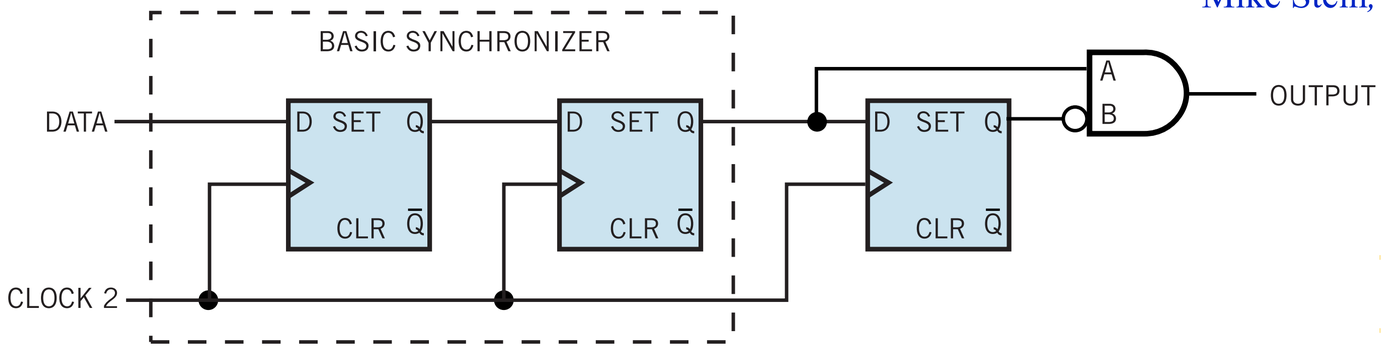
```
FD1 : FDC port map(level(0), CLK_DIN, RST, DIN);

FD2 : FDC port map(level(1), CLK_DOUT, RST, level(0));
FD3 : FDC port map(DOUT, CLK_DOUT, RST, level(1));
```

Note convention of using **ports** (inputs/outputs) in **all capitals**

Edge-detecting synchronizer

- Creates a **one-clock-cycle long pulse** in a **faster or equal clock domain** when there is a **rising edge** in the **original clock domain**

"Crossing the abyss: asynchronous signals in a synchronous world"
Mike Stein, **designfeature** 59 (2013)



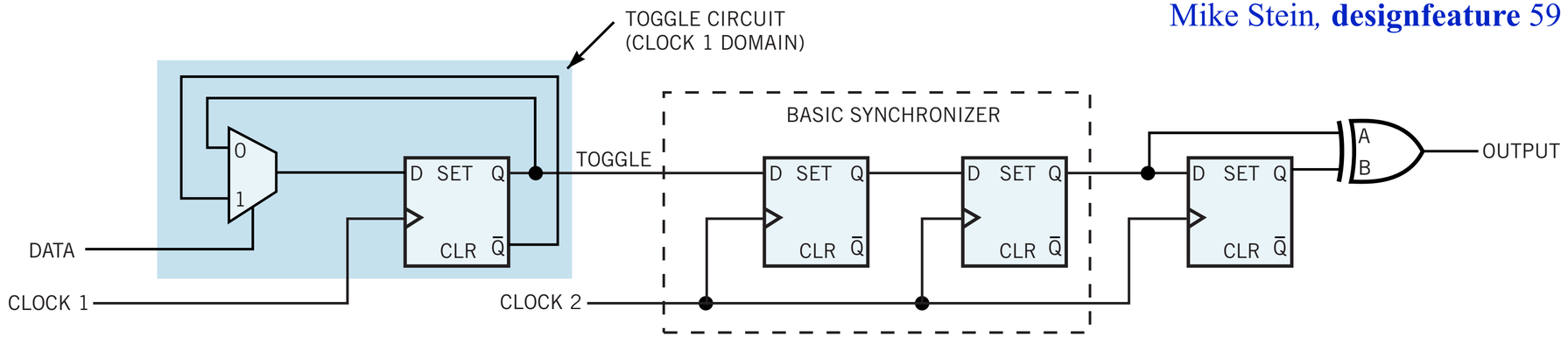
- Implemented in **`source/utils/pulse2fast.vhd`**

```
FD0 : FDC generic map(INIT => '1') port map(pulse0, CLK_DOUT, RST, DIN);
FD1 : FDC generic map(INIT => '1') port map(pulse(1), CLK_DOUT, RST, pulse0);
FD2 : FDC generic map(INIT => '1') port map(pulse(2), CLK_DOUT, RST, pulse(1));

DOUT <= pulse(1) and not pulse(2);
```

Pulse synchronizer

- Creates a **one-clock-cycle long pulse** in a **slower, faster or equal clock domain** when there is a **pulse** in the **original clock domain**

"Crossing the abyss: asynchronous signals in a synchronous world"
Mike Stein, **designfeature** 59 (2013)



- Implemented in **`source/utils/pulse2slow.vhd`**

```
pulsein(0) <= pulsein(1) when DIN = '0' else not pulsein(1);
FD1 : FDC generic map(INIT => '1') port map(pulsein(1), CLK_DIN, RST, pulsein(0));

FD2 : FDC generic map(INIT => '1') port map(pulse(2), CLK_DOUT, RST, pulsein(1));
FD3 : FDC generic map(INIT => '1') port map(pulse(3), CLK_DOUT, RST, pulse(2));
FD4 : FDC generic map(INIT => '1') port map(pulse(4), CLK_DOUT, RST, pulse(3));

DOUT <= pulse(3) xor pulse(4);
```

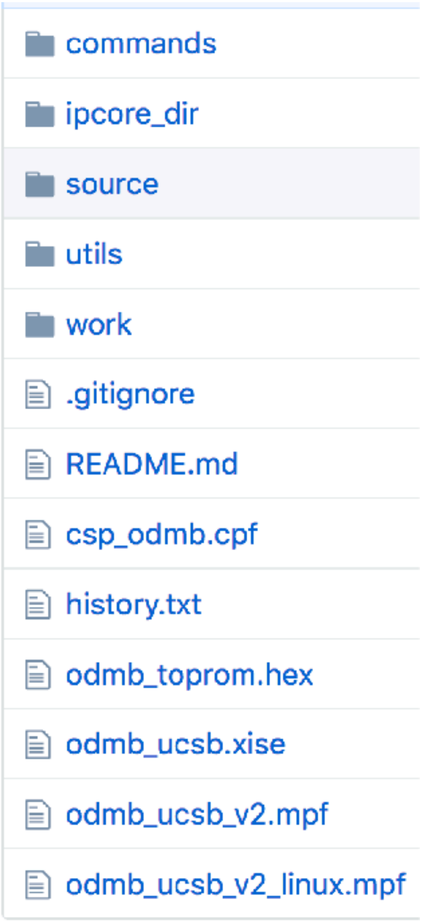
The ODMB firmware

ODMB firmware project

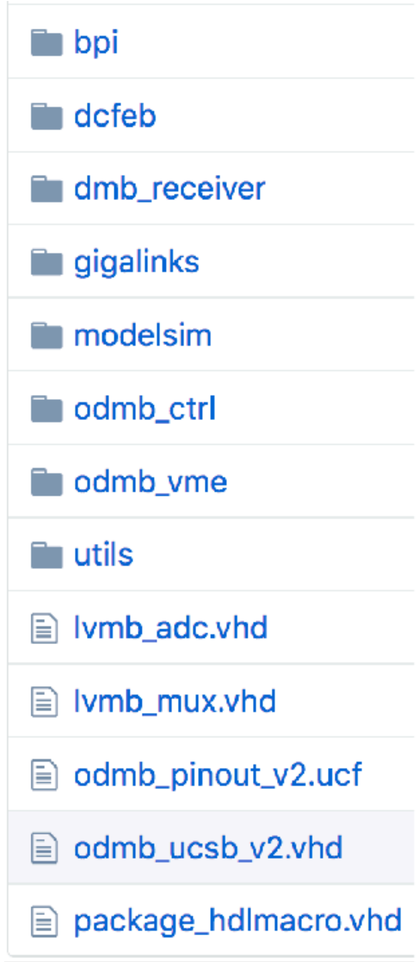
- Can be found at https://github.com/csc-fw/odmb_ucsb_v2
  - Typically committing to https://github.com/odmb/odmb_ucsb_v2 and syncing the fork, but we probably should end that

- Top folders
  - **commands**: text files with input VME commands to test firmware (typically simulation)
  - **ipcore_dir**: IP cores from Xilinx that handle some sophisticated resources (GTX, big FIFOs)
    - Generated and configured from ISE, which also produces VHDL files to be able to simulate them
  - **source**: folder with VHDL and Verilog code
  - **utils**: a utility to transform a Verilog function into VHDL

- Top files
  - **history.txt**: to be updated with each firmware release
  - **odmb_toprom.hex**: marker to be introduced into each PROM configuration file (.mcs)
    - Allows us to identify ODMB firmware from DCFEB's, since both use the same FPGA
  - **odmb_ucsb.xise**: ISE project
  - **odmb_ucsb_v2.mpf**: ModelsSim Windows project
  - **odmb_ucsb_v2_linux.mpf**: ModelSim Linux project



ODMB firmware code

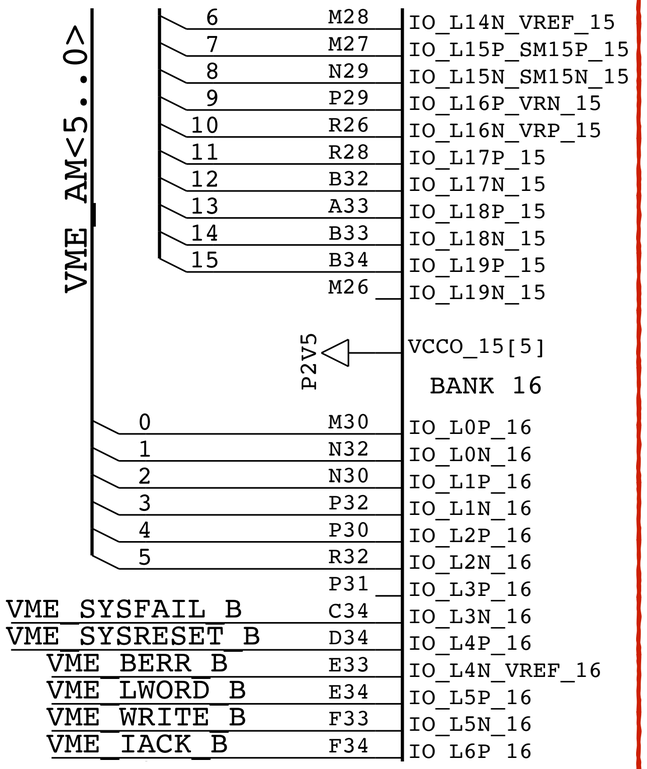


- Top folders
  - **bpi**: programs the PROM using BPI engine
  - **dcfeb**: simulates response from DCFEB (data and JTAG control)
  - **dmb_receiver**: handles the 7 optical receivers from the DCFEBs
  - **gigalinks**: handles the TX/RX with the DDU and spy PC
  - **modelsim**: simulated test bench and other utilities only used during simulation
  - **odmb_ctrl**: modules in MBC that handle the data flow (fast control)
  - **odmb_VME**: modules in MBV that handle the VME communication (slow control)
  - **utils**: utilities to safely cross clock domains, count pulses, etc
- Top files
  - **lvmb_adc.vhd**: simulates 1 of the 7 ADCs in an LVMB7
  - **lvmb_mux.vhd**: simulates response from an LVMB7
  - **odmb_pinout_v2.ucf**: correspondence between firmware signals and FPGA pins
  - **odmb_ucsb_v2.vhd**: top of the design that sends/receives signals from the board or test bench
  - **package_hdlmacro.vhd**: VHDL implementation of basic Xilinx resources such as FDCE

UCF: User Constraints File

Schematics

Connections on the board



UCF

Correspondence between firmware signals and FPGA pins

```
NET  "vme_am[0]"          LOC = "M30"  | IOSTANDARD = LVCMOS25;
NET  "vme_am[1]"          LOC = "N32"  | IOSTANDARD = LVCMOS25;
NET  "vme_am[2]"          LOC = "N30"  | IOSTANDARD = LVCMOS25;
NET  "vme_am[3]"          LOC = "P32"  | IOSTANDARD = LVCMOS25;
NET  "vme_am[4]"          LOC = "P30"  | IOSTANDARD = LVCMOS25;
NET  "vme_am[5]"          LOC = "R32"  | IOSTANDARD = LVCMOS25;
NET  "vme_sysfail_b"      LOC = "C34"  | IOSTANDARD = LVCMOS25;
NET  "vme_sysreset_b"     LOC = "D34"  | IOSTANDARD = LVCMOS25;
NET  "vme_berr_b"         LOC = "E33"  | IOSTANDARD = LVCMOS25;
NET  "vme_lword_b"        LOC = "E34"  | IOSTANDARD = LVCMOS25;
NET  "vme_write_b"        LOC = "F33"  | IOSTANDARD = LVCMOS25;
NET  "vme_iack_b"         LOC = "F34"  | IOSTANDARD = LVCMOS25;
```

Firmware

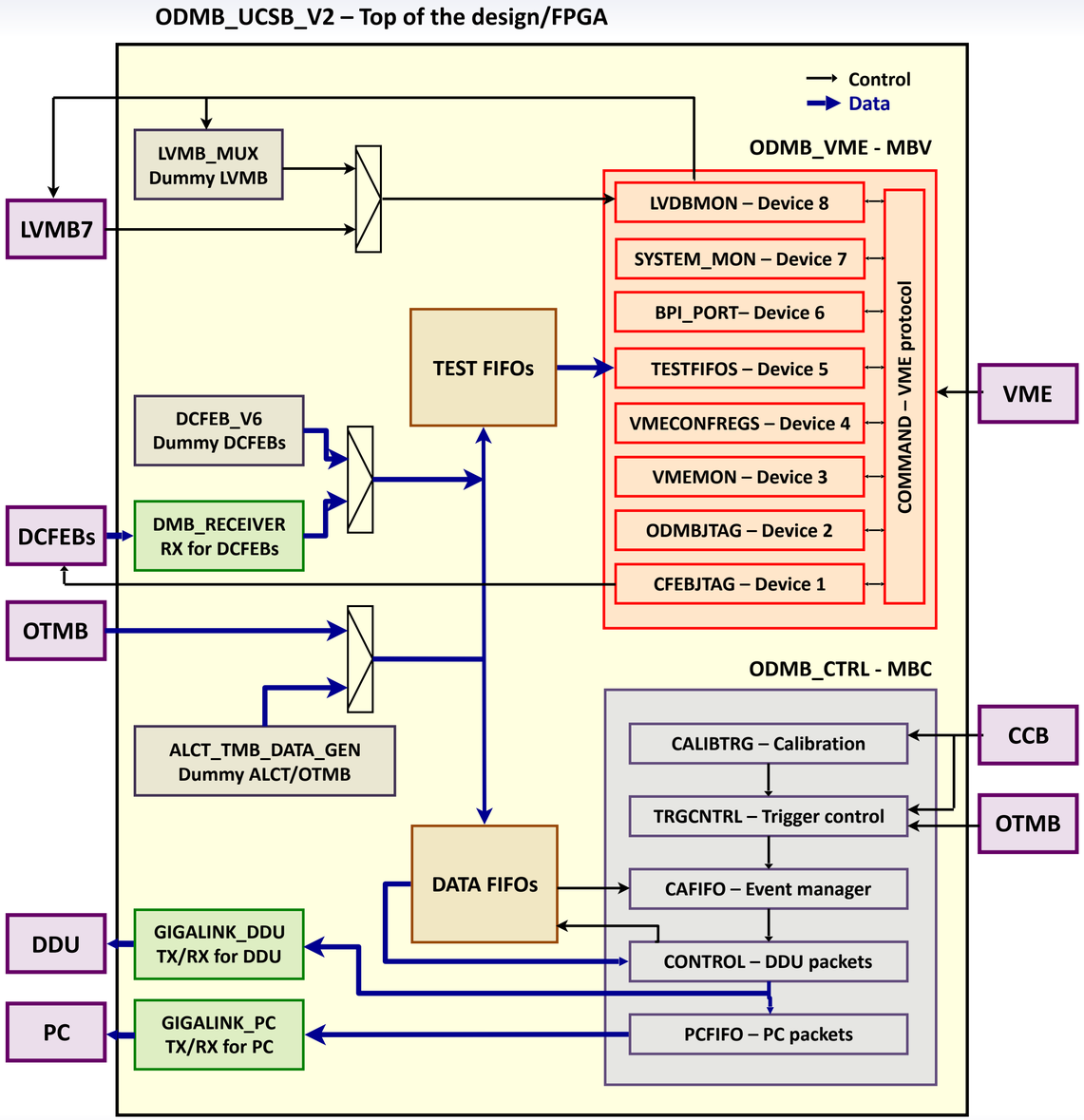
Signals internal to FPGA

```
entity ODMB_UCSB_V2 is
  generic (
    IS_SIMULATION : integer range 0 to 1 := 0;
    NREGS         : integer := 16;
    NFEB          : integer range 1 to 7 := 7
    );
  port (

    vme_data        : inout std_logic_vector(15 downto 0);
    vme_addr        : in    std_logic_vector(23 downto 1);
    vme_am          : in    std_logic_vector(5 downto 0);
    vme_gap         : in    std_logic;
    vme_ga          : in    std_logic_vector(4 downto 0);
    vme_as_b        : in    std_logic;
    vme_ds_b        : in    std_logic_vector(1 downto 0);
    vme_sysfail_b   : in    std_logic;
    vme_sysfail_out : out   std_logic;
    vme_berr_b      : in    std_logic;
    vme_berr_out    : out   std_logic;
    vme_iack_b      : in    std_logic;
    vme_lword_b     : in    std_logic;
    vme_write_b     : in    std_logic;
```

- UCF also specifies other constraints
  - eg. clocking, to know which clocking resources to use

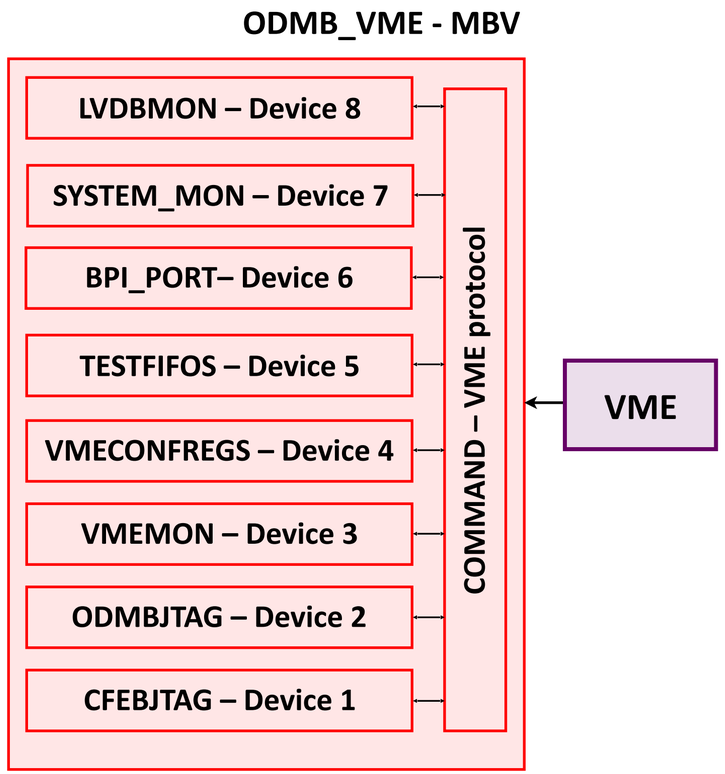
```
# User Clock Constraints
NET "qpll_clk40mhz_n" TNM_NET = qpll_clk40mhz_n;
TIMESPEC TS_qpll_clk40mhz_n = PERIOD "qpll_clk40mhz_n" 25 ns HIGH 50%;
NET "qpll_clk40mhz_p" TNM_NET = qpll_clk40mhz_p;
TIMESPEC TS_qpll_clk40mhz_p = PERIOD "qpll_clk40mhz_p" 25 ns HIGH 50%;
```

Code structure



- Two big blocks: MBV and MBC
  - They used to be two different FPGAs in the DMB
- **MBV** handles the **VME communication**
  - Translates VME instructions sent by PC to hardware commands
  - Slow control since it runs at 2.5 MHz
- **MBC** handles the **data flow**
  - Stores data as it comes, handles triggers, construct data packet
  - Fast control since it runs at 40-80 MHz
- Dummy devices simulate real LVMB7, DCFEBs, ALCT, OTMB
  - Can be used in simulation or in actual board

The VME address



- The VME address contains both the requested slot in the crate and the command
- The first 5 bits correspond to the slot
  - eg. 0x541980 corresponds to slot 0x15 = 21

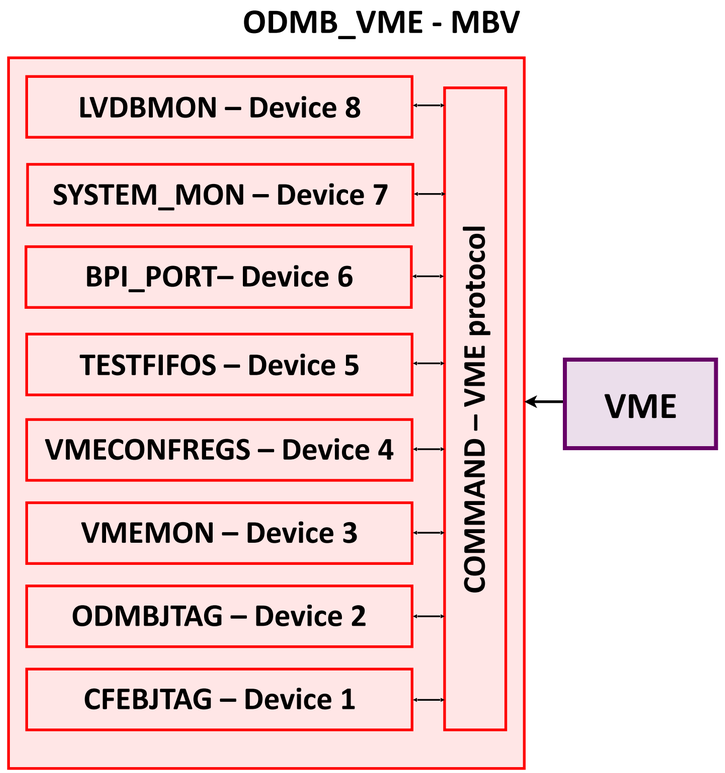
	0x5	0x4	0x1	0x9	0x8	0x0
Slot in crate = 0x15	101	0100	0001	1001	1000	0000

- Bits 18 to 1 correspond to the instruction
  - It is quite confusing because bit 0 is not passed to the FPGA, so the instruction is shifted by one bit in the firmware
  - eg. 0x541980 corresponds to instruction 0x3300 in slot 21

0x0	0x3	0x3	0x0	0x0
000	0011	0011	0000	0000

- The first hex digit of the 16-bit instruction corresponds to the device being addressed

VME protocol decoding



- **`source/odmb_vme/command.vhd`**: decodes the key signals
  - Compares **requested slot** with **board geographical address**
    - Bits 23-19 of VME address indicate the slot

```
CGA <= not GA;
BOARD_SEL_NEWEW <= '1' when (ADRS_INNER(23 downto 19) = CGA(4 downto 0)) else '0';
```

  - **Generates COMMAND** as a subset of the VME address

```
-- Generate COMMAND
COMMAND(9 downto 0) <= ADRS_INNER(11 downto 2);
```

Drops another bit from VME address!

  - **Generates DEVICE** which selects which set of VME instructions will be executed

```
-- Generate DEVICE
ADRSHIGH <= '1' when (ADRS_INNER(18) = '1' or ADRS_INNER(17) = '1' or ADRS_INNER(16) = '1') else '0';
ADRSDEV  <= ADRSHIGH & ADRS_INNER(15) & ADRS_INNER(14) & ADRS_INNER(13) & ADRS_INNER(12);

with ADRSDEV select
  DEVICE <= "0000000001" when "00000",
  "0000000010"           when "00001",
  "0000000100"           when "00010",
  "0000001000"           when "00011",
  "0000010000"           when "00100",
  "0000100000"           when "00101",
  "0001000000"           when "00110",
  "0010000000"           when "00111",
  "0100000000"           when "01000",
  "1000000000"           when "01001",
  "0000000000"           when others;
```

Example of VME decoding

- Example: setting the ODMB to use dummy data and internal triggers
  - As shown in the ODMB user's manual, these are commands 3300 and 3304 on device 3 (VMEMON)

Instruction	Description
W/R 3300	Data multiplexer: 0 → real data, 1 → dummy data
W/R 3304	Trigger multiplexer: 0 → external triggers, 1 → internal triggers

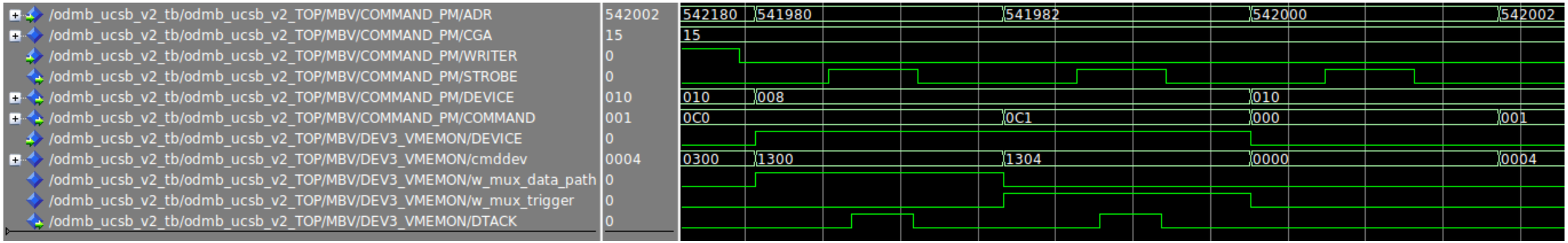
  - **source/odmb_vme/vmemon.vhd** decodes the instruction directed at device 3
    - cmddev is a human readable instruction that does not require shifts

```
cmddev <= "000" & DEVICE & COMMAND & "00";
w_mux_data_path <= '1' when (CMDDEV = x"1300" and WRITER = '0') else '0';
r_mux_data_path <= '1' when (CMDDEV = x"1300" and WRITER = '1') else '0';
w_mux_trigger   <= '1' when (CMDDEV = x"1304" and WRITER = '0') else '0';
r_mux_trigger   <= '1' when (CMDDEV = x"1304" and WRITER = '1') else '0';
```

- WRITER indicates whether the instruction is a write ('0') or a read ('1')
  - Active-low VME signal

Example of VME decoding

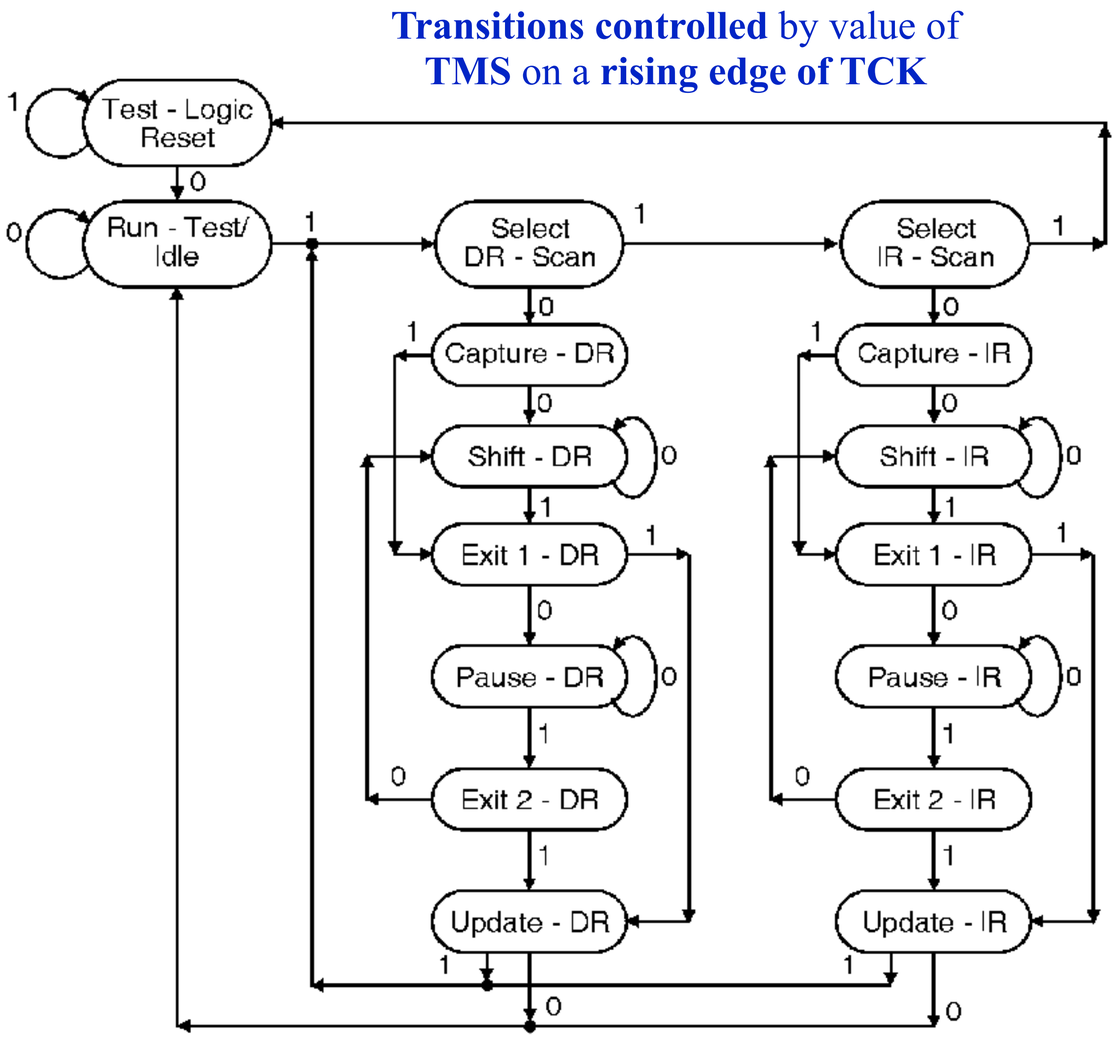
- Example: setting the ODMB to use dummy data and internal triggers
  - 0x541980 corresponds to instruction 0x3300 in slot 0x15
  - DEVICE in **`command.vhd`** is 0x8 (= b1000), so DEVICE in **`vmemon.vhd`** is '1'
  - **`command.vhd`** also generates STROBE which is high while the command is being executed



  - When the instruction is done, the device sends a DTACK (data acknowledge)
    - In this case instruction is done automatically, so it sends it immediately

```
dd_dtack <= STROBE and DEVICE;
FD_D_DTACK : FDC port map(d_dtack, dd_dtack, q_dtack, '1');
FD_Q_DTACK : FD port map(q_dtack, SLOWCLK, d_dtack);
DTACK    <= q_dtack;
```

The JTAG protocol

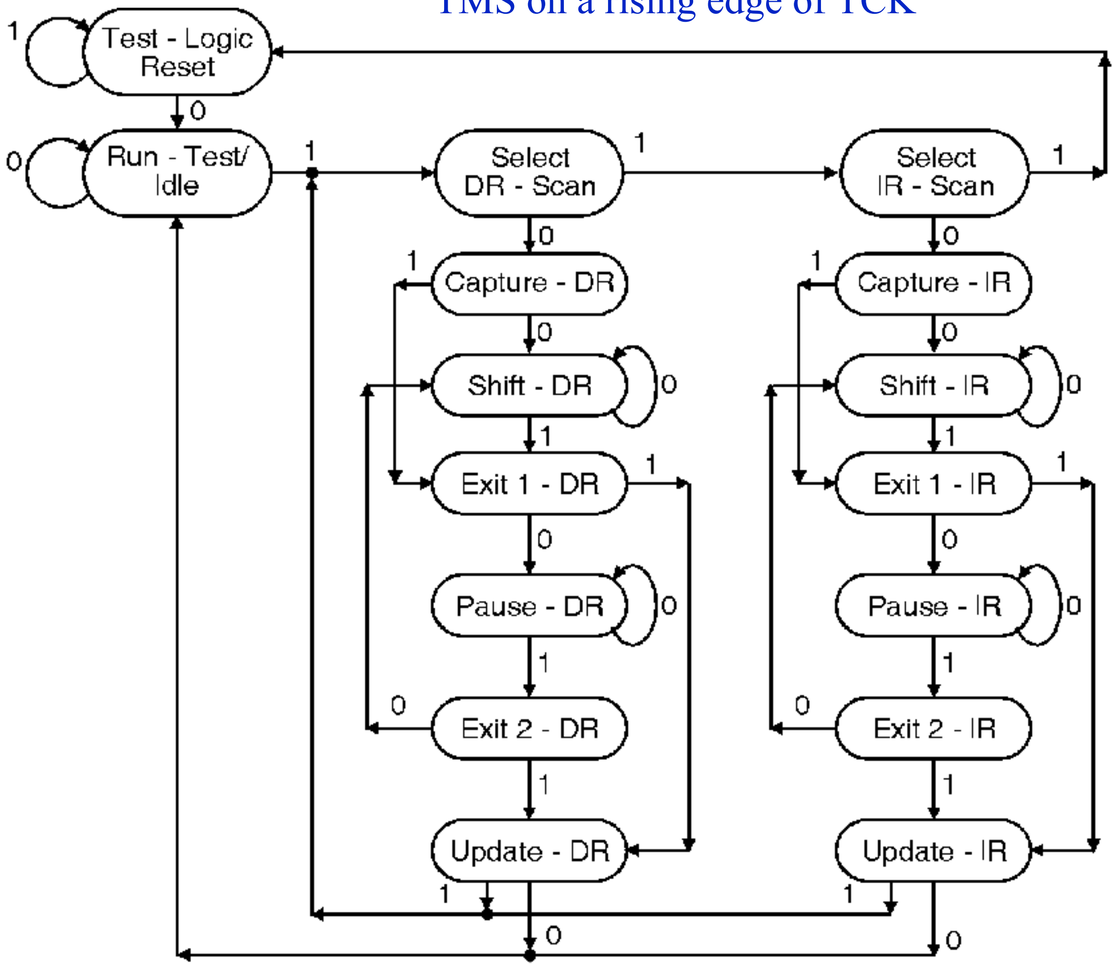


- Important serial protocol that use to communicate with the DCFEBs
  - **TCK**: JTAG clock (to the DCFEBs)
  - **TMS**: mode select (to the DCFEBs)
  - **TDI**: data input (to the DCFEBs)
  - **TDO**: data ouput (from the DCFEBs)

- Also used to program the ODMB FPGA via the Xilinx red box or the emergency logic

Example: data shift

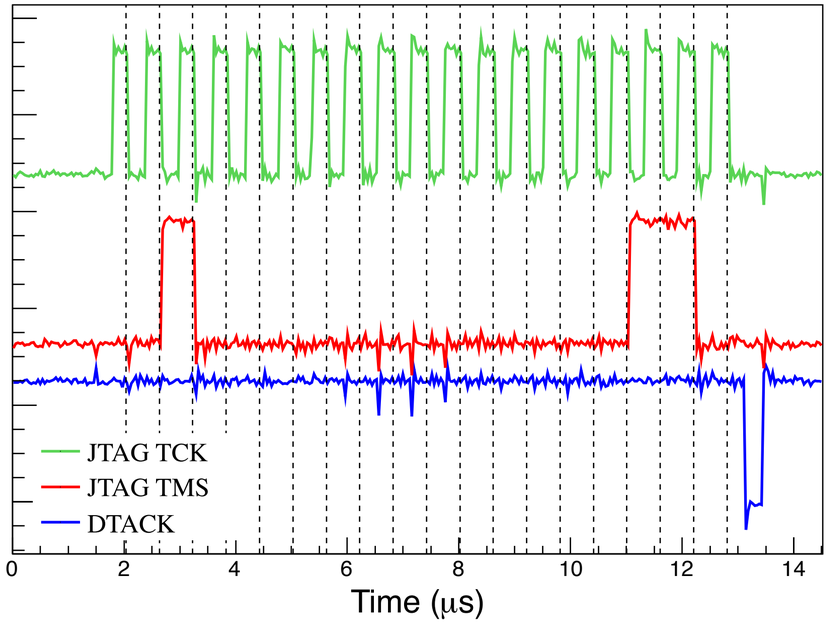
Transitions controlled by value of TMS on a rising edge of TCK



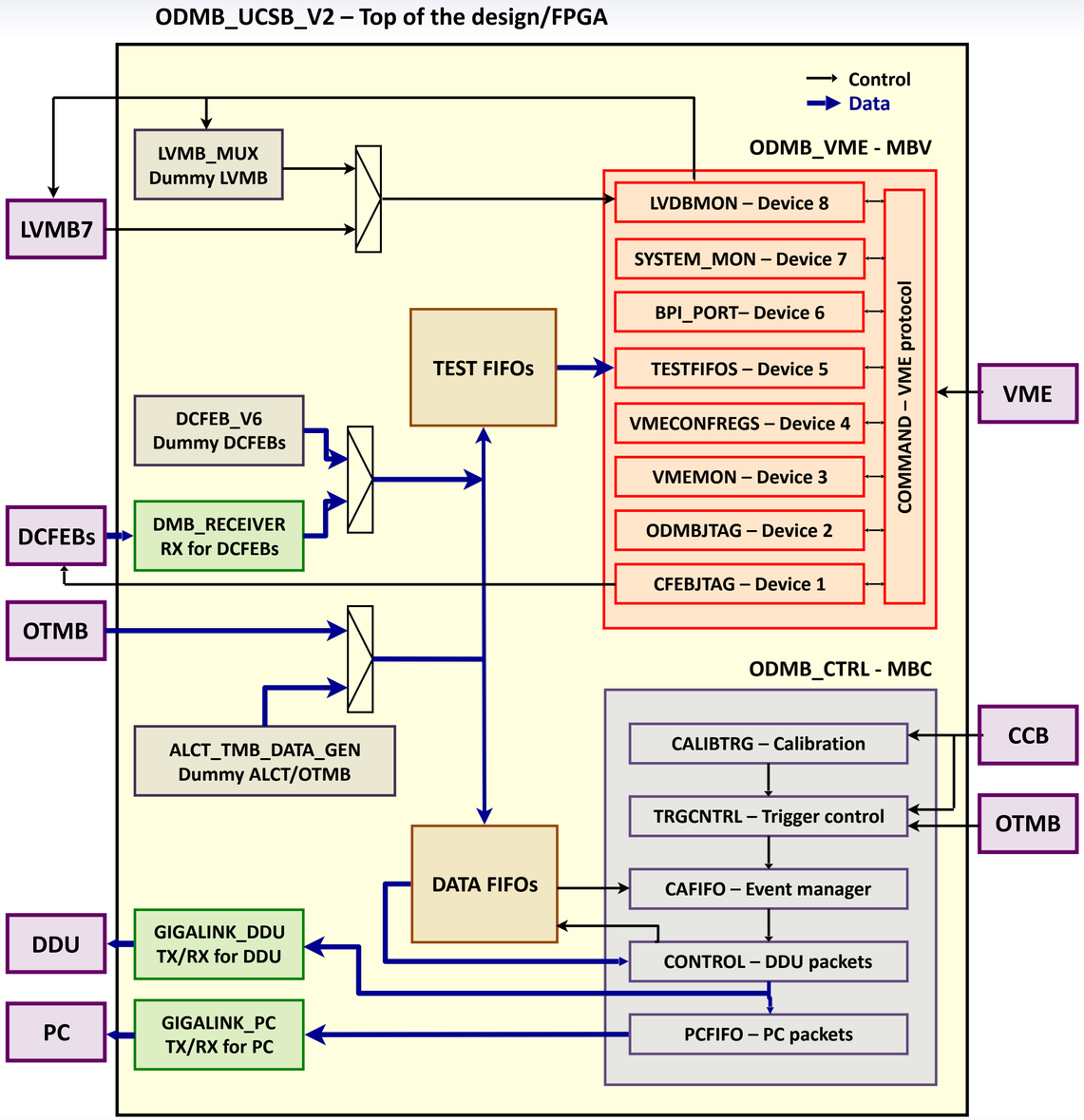
"Y" refers to the number of bits to be shifted

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 instruction 0x1B0C



Summary



- Learned basic firmware tips
- Saw overview of ODMB firmware project
- Basics of VME protocol
- Basics of JTAG protocol
- **Still to see**
  - **Description of each device**
  - **Packet building**
- Summer task will be to implement arbitrarily-long instructions
  - Just like we can send arbitrarily-long data

Instruction	Description
`W 1Y1C`	Shift Instruction register