

FPGA Lecture for LUPO and GTO Vol. 1

2010, 31 August (revised 2013, 19 November)

H. Baba



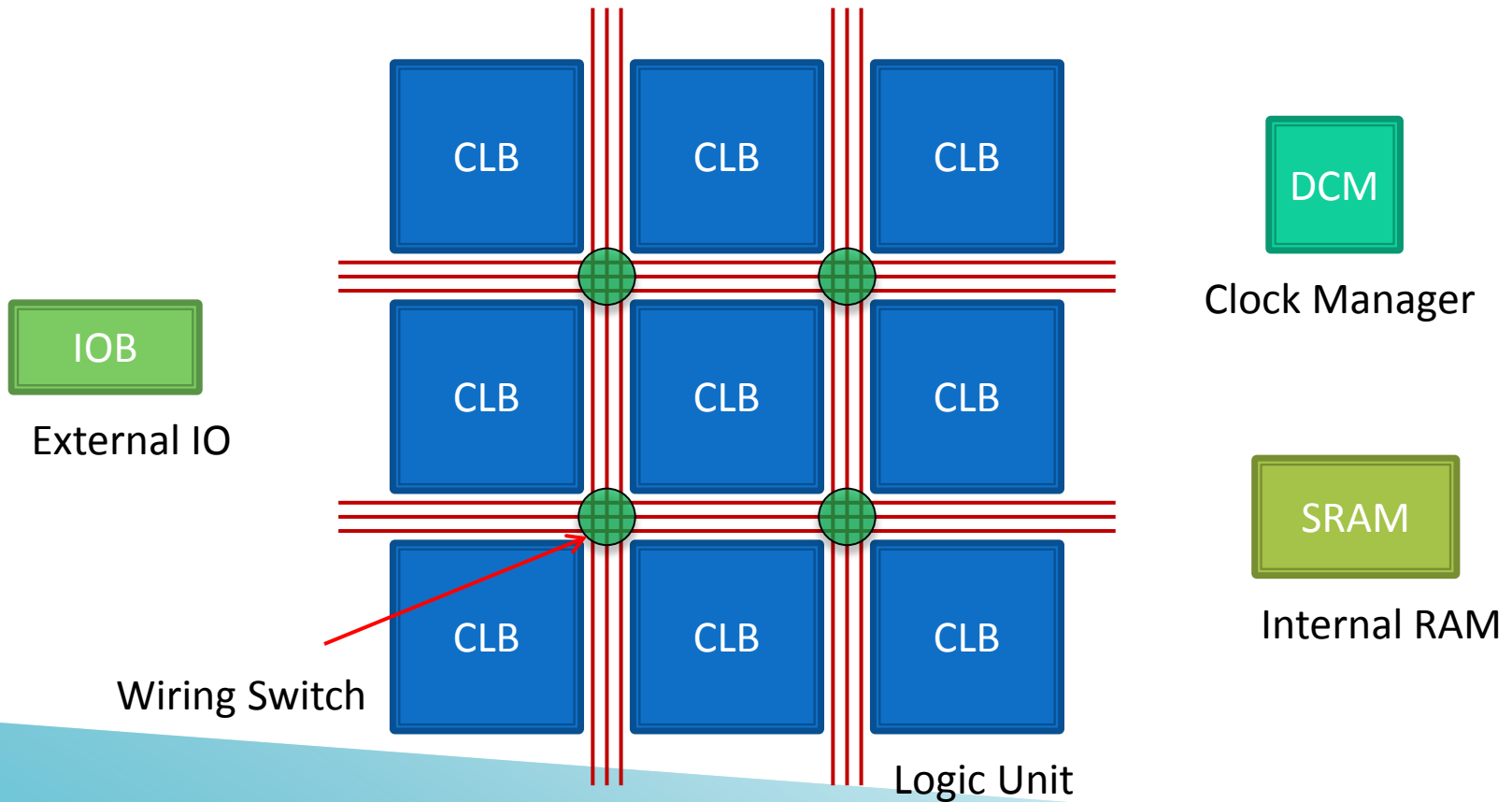
Contents

- ▶ Basic feature of FPGA
- ▶ Basic usage for LUPO
 - New project
 - Read and Write
- ▶ Basic behavioral VHDL simulation
- ▶ Download firmware

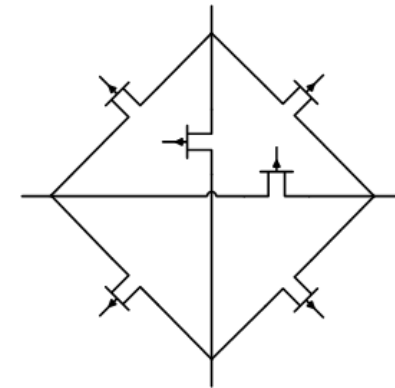
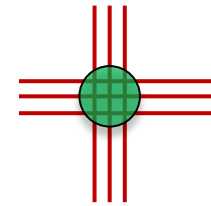
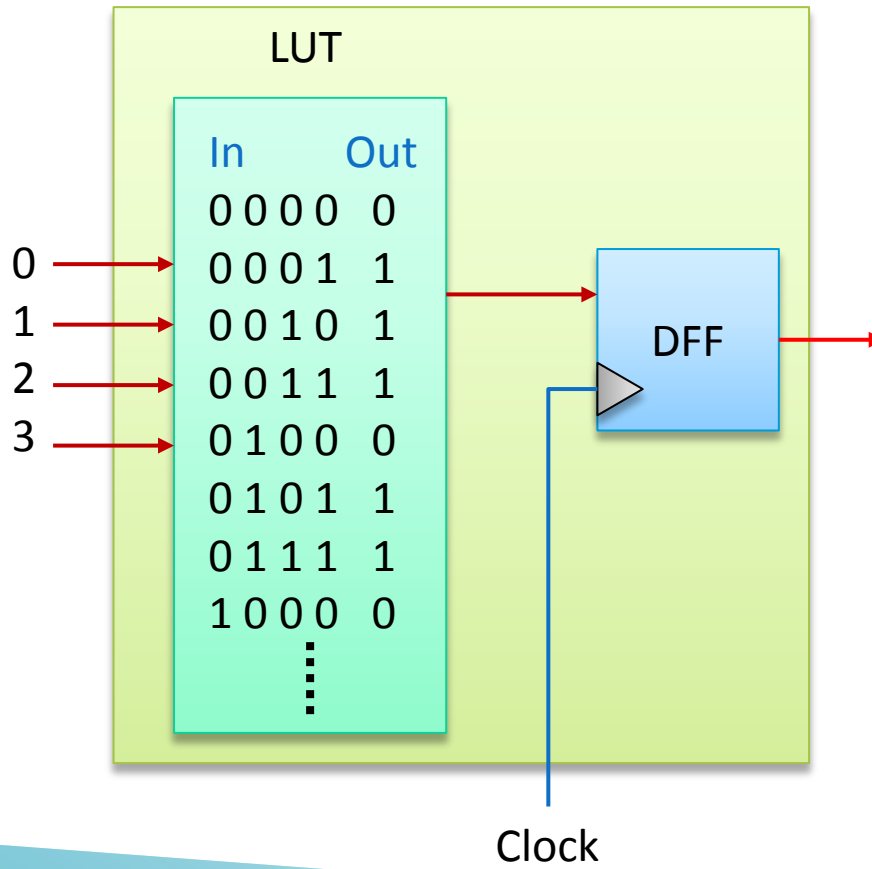


What is FPGA ?

▶ Field Programmable Gate Array

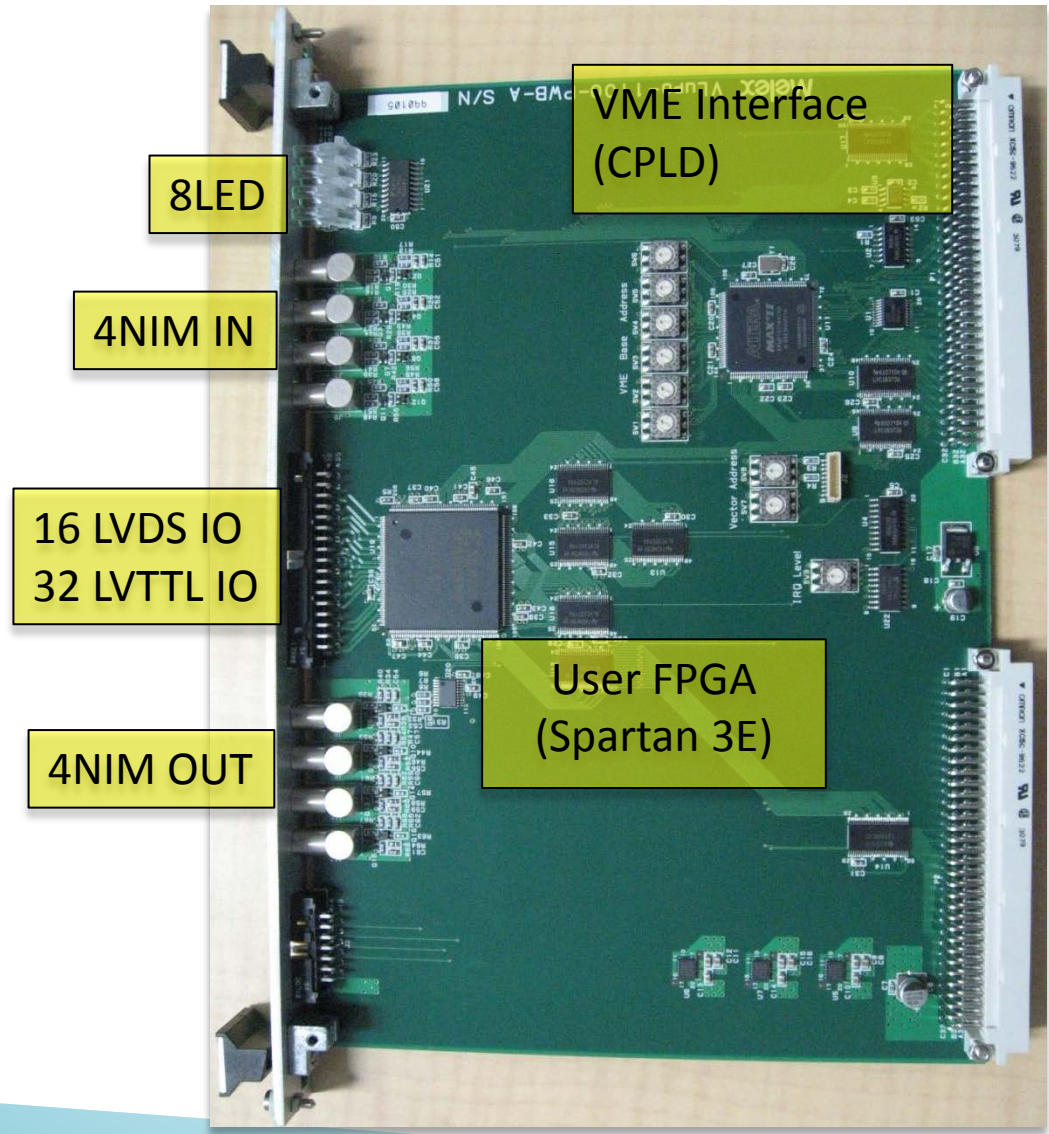


CLB and wiring



About LUPO

- ▶ NIM signals are converted to LVTTTL
 - NIM->LVTTTL->FPGA
 - FPGA->LVTTTL->NIM
- ▶ LVDS ports are direct connection
 - Because of this IOB is 2.5V, this port can be accept 2.5V IO signals



New Project for LUPO

- ▶ Device and Design flow

Select the device and design flow for the project

Property Name	Value
Product Category	All
Family	Spartan3E
Device	XC3S500E
Package	PQ208
Speed	-4
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	VHDL
Property Specification in Project File	Store all values
Manual Compile Order	<input type="checkbox"/>
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	<input type="checkbox"/>



Design property for GTO

Project Settings

Property Name	Value
Top-Level Source Type	HDL
Evaluation Development Board	None Specified
Product Category	All
Family	Spartan3A and Spartan3AN
Device	XC3S200A
Package	VQ100
Speed	-4
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	VHDL
Property Specification in Project File	Store all values
Manual Compile Order	<input type="checkbox"/>
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	<input type="checkbox"/>

OK Cancel Help



The definition of IO (LUPO)

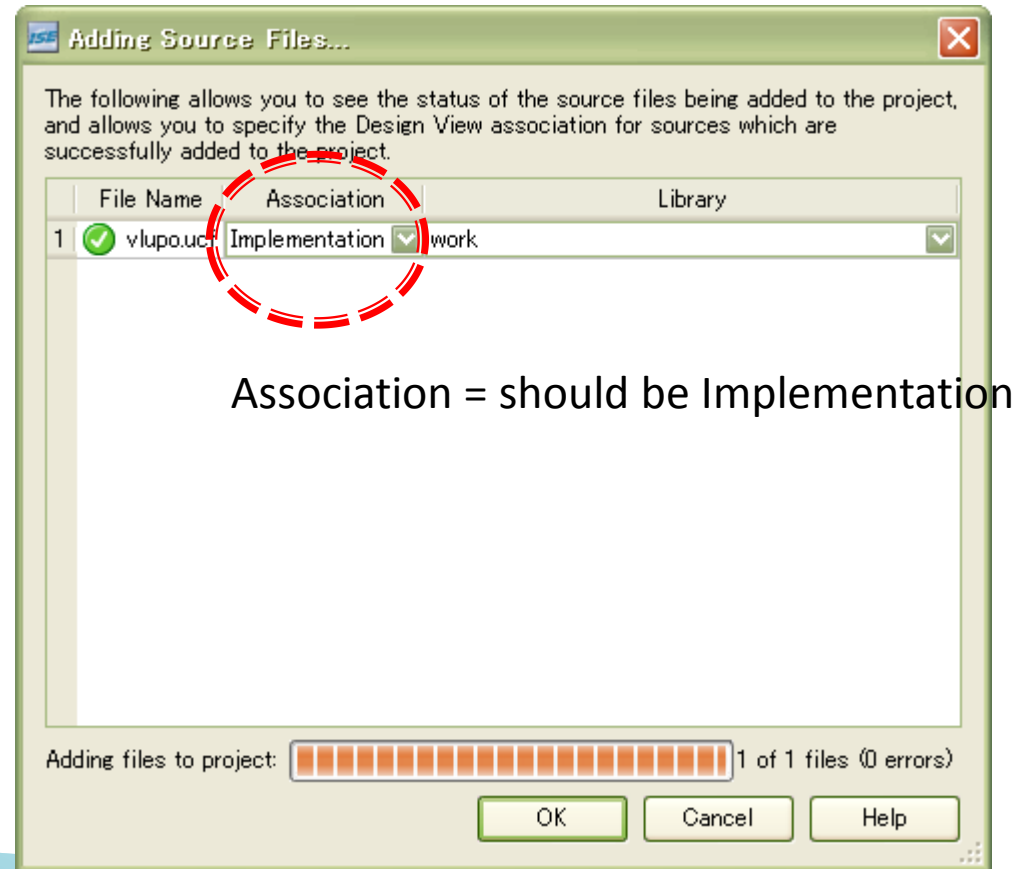
- ▶ LED : LED
- ▶ A : CAMAC/VME address
- ▶ CLOCK : 50MHz clock
- ▶ INIT : CAMAC Z/C, VME Init?
- ▶ IP : NIM Input 0-3
 - IPO: IPO to GCLK
- ▶ OP : NIM Output 0-3
- ▶ LVDSn/LVDSp : LVDS IO 0-15
 - LVDS_CLKn/p : LVDS0 to GCLK
- ▶ RD : Data Read
- ▶ WR : Data Write
- ▶ RD_STRB : Read strobe
- ▶ WR_STRB : Write strobe

```
entity HOGE is
  port (
    LED          : out STD_LOGIC_VECTOR (7 downto 0);
    -- LVDS_CLKn : in  STD_LOGIC;
    -- LVDS_CLKp : in  STD_LOGIC;
    -- LVDSn     : in  STD_LOGIC_VECTOR (15 downto 0);
    -- LVDSp     : in  STD_LOGIC_VECTOR (15 downto 0);
    -- LVDSn     : out STD_LOGIC_VECTOR (15 downto 0);
    -- LVDSp     : out STD_LOGIC_VECTOR (15 downto 0);
    A            : in  STD_LOGIC_VECTOR (7 downto 0);
    CLOCK        : in  STD_LOGIC;
    INIT         : in  STD_LOGIC;
    -- IPO       : in  STD_LOGIC;
    IP           : in  STD_LOGIC_VECTOR (3 downto 0);
    -- UDI       : in  STD_LOGIC_VECTOR (3 downto 0);
    -- UDO       : out STD_LOGIC_VECTOR (3 downto 0);
    IRQ         : out STD_LOGIC;
    OP           : out STD_LOGIC_VECTOR (3 downto 0);
    RD           : out STD_LOGIC_VECTOR (31 downto 0);
    WR           : in  STD_LOGIC_VECTOR (31 downto 0);
    RD_STRB     : in  STD_LOGIC;
    WR_STRB     : in  STD_LOGIC);
end HOGE;
```



Copy the Constraint file (UCF)

- ▶ vlupo.ucf
- ▶ gto.ucf
- ▶ Write constraints for IO and others



Without CAMAC/VME Buses (LUPO)

- ▶ This is enough

```
entity HOGE is
  port (
    LED      : out STD_LOGIC_VECTOR (7 downto 0);
    CLOCK    : in  STD_LOGIC;
    IP       : in  STD_LOGIC_VECTOR (3 downto 0);
    OP       : out STD_LOGIC_VECTOR (3 downto 0));
end HOGE;
```

NETs written in UCF are must be written in “entity” also

Edit vlupo.ucf by “Edit Constraints(Text)”

```
NET "CLOCK" LOC = "P177" | IOSTANDARD = LVTTTL ;
NET "IP[0]" LOC = "P202" | IOSTANDARD = LVTTTL ;
NET "IP[1]" LOC = "P203" | IOSTANDARD = LVTTTL ;
NET "IP[2]" LOC = "P204" | IOSTANDARD = LVTTTL ;
NET "IP[3]" LOC = "P205" | IOSTANDARD = LVTTTL ;
NET "IRQ" LOC = "P64" | IOSTANDARD = LVCOS25 ;
NET "LED[0]" LOC = "P189" | IOSTANDARD = LVTTTL ;
NET "LED[1]" LOC = "P190" | IOSTANDARD = LVTTTL ;
NET "LED[2]" LOC = "P192" | IOSTANDARD = LVTTTL ;
NET "LED[3]" LOC = "P193" | IOSTANDARD = LVTTTL ;
NET "LED[4]" LOC = "P196" | IOSTANDARD = LVTTTL ;
NET "LED[5]" LOC = "P197" | IOSTANDARD = LVTTTL ;
NET "LED[6]" LOC = "P199" | IOSTANDARD = LVTTTL ;
NET "LED[7]" LOC = "P200" | IOSTANDARD = LVTTTL ;
NET "OP[0]" LOC = "P63" | IOSTANDARD = LVCOS25 ;
NET "OP[1]" LOC = "P62" | IOSTANDARD = LVCOS25 ;
NET "OP[2]" LOC = "P61" | IOSTANDARD = LVCOS25 ;
NET "OP[3]" LOC = "P60" | IOSTANDARD = LVCOS25 ;
NET "CLOCK" TNM_NET = CLOCK;
TIMESPEC "TS_CLOCK" = PERIOD "CLOCK" 20 ns HIGH 50 %;
```



Simple case (LUPO)

▶ Like this

```
architecture Behavioral of HOGE is
begin
OP(0) <= IP(0) and IP(1);
OP(1) <= IP(2) or IP(3);
OP(2) <= IP(0) and not IP(1);
OP(3) <= CLOCK;
LED(3 downto 0) <= IP(3 downto 0);
LED(7 downto 4) <= (others => '1');
```

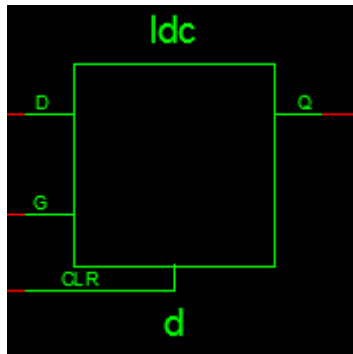
```
architecture Behavioral of HOGE is
signal a, b, c : std_logic;
begin
a <= IP(0) and IP(1);
b <= IP(2) or IP(3);
c <= a and not b;
OP(0) <= a;
OP(1) <= b;
OP(2) <= c;
OP(3) <= CLOCK;
LED(3 downto 0) <= IP(3 downto 0);
LED(7 downto 4) <= (others => '1');
```

If Synthesize -> Implement Design are passed, it will work



To make the synchronous circuit

- ▶ When you use “if, else”, you should put values into all signals explicitly
 - Check Warning



In this case, created circuit will be asynchronous circuit

```
process(IP(0), IP(1))
begin
  if(IP(0) = '1') then
    d <= '0';
    e <= '1';
  elsif(IP(1) = '1') then
    d <= '1';
    e <= '0';
  -- else
  --   d <= '0';
  --   e <= '0';
  end if;
end process;
```

Should write values in “else” explicitly

xst

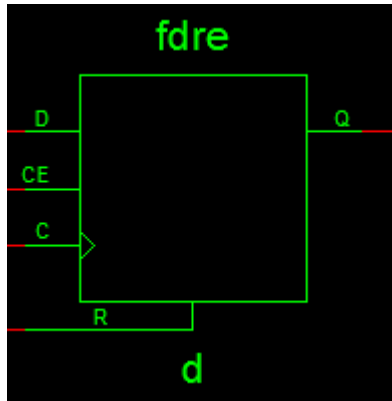


Xst:737 - Found 1-bit latch for signal <d>. Latches may be generated from incomplete case or if statements. We do not recommend the use of latches in FPGA/CPLD designs, as they may lead to timing problems.



This type works well, usually

- ▶ By using “event process”, DFF will be used, and values are kept as you had expected



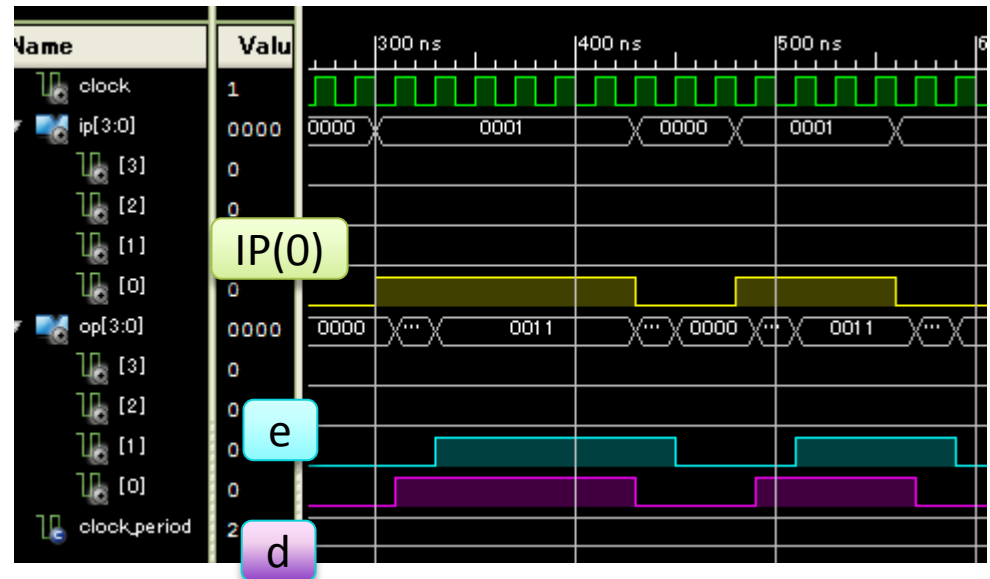
Values are changed at the Clock timing

```
signal d, e : std_logic := '0';  
  
begin  
  
  process(CLOCK)  
  begin  
    if(CLOCK'event and CLOCK='1') then  
      if(IP(0) = '1') then  
        d <= '1';  
        e <= '1';  
      else  
        d <= '0';  
        -- e <= '0';  
      end if;  
    end if;  
  end process;
```



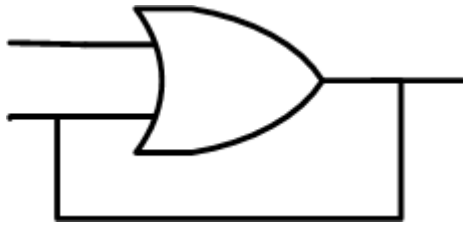
Basically, all lines run in parallel

```
OP(0) <= d;  
OP(1) <= e;  
OP(2) <= '0';  
OP(3) <= '0';  
  
process(CLOCK)  
begin  
  if(CLOCK'event and CLOCK='1') then  
    if(IP(0) = '1') then  
      d <= '1';  
      e <= d;  
    else  
      d <= '0';  
      e <= d;  
    end if;  
  end if;  
end process;
```

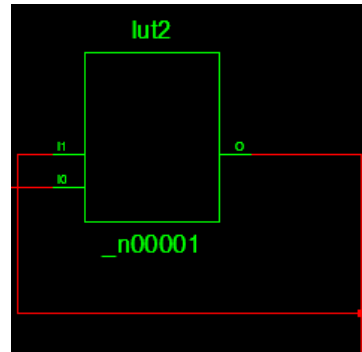



Be careful about the loop circuit

- ▶ As this schematic, you can make the “Latch circuit”, but...



```
a <= IP(0) or a;
```



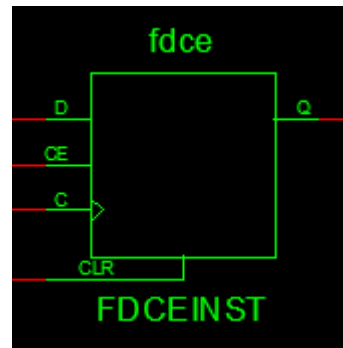
xst  Xst:2170 - Unit H0GE : the following signal(s) form a combinational loop:

- ▶ Warnings of Combinational loop
 - It will not work, depending on the pulse width, line delay, jitter and so on



Expressly using FF is safe

- ▶ “event” in the process statement makes DFF, basically
- ▶ As a circuit component, if you use FDCE(=DFF), it will work as you expected



```
COMPONENT FDCE
generic (INIT : bit := '1');
PORT (
    Q   : OUT std_logic;
    C   : IN  std_logic;
    CE  : IN  std_logic;
    CLR : IN  std_logic;
    D   : IN  std_logic);
END COMPONENT;

signal a : std_logic;

begin

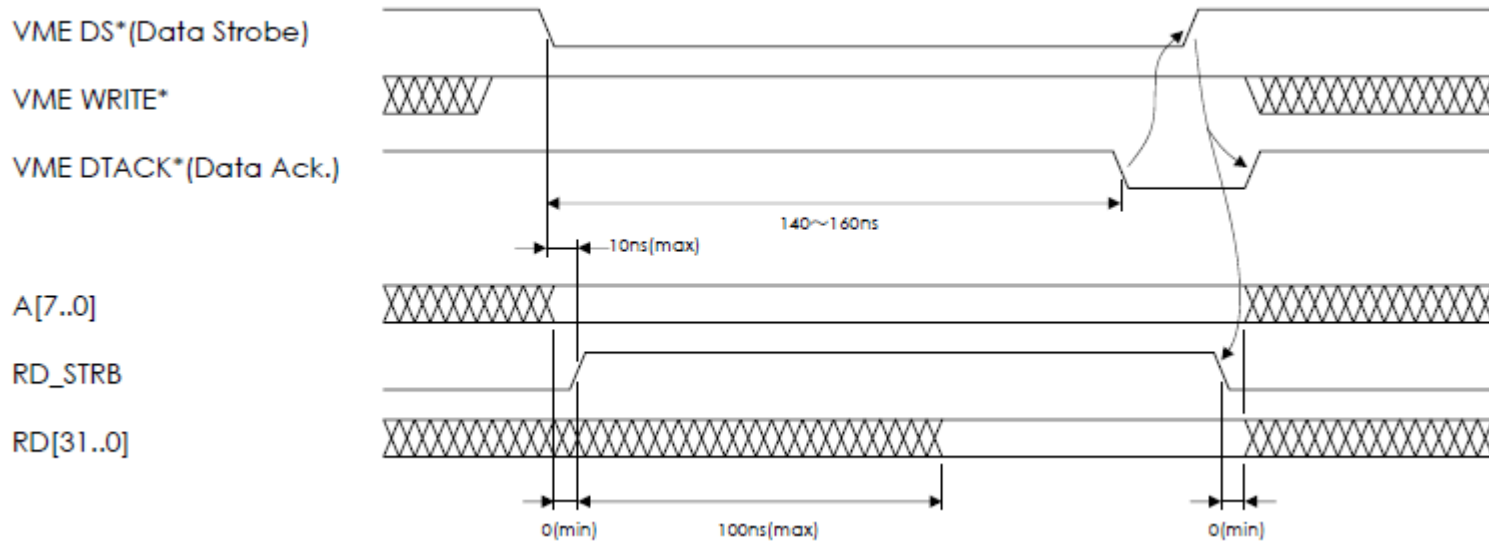
--a <= IP(0) or a;
FDCEINST: FDCE
generic map(
    INIT => '0')
PORT MAP(
    Q   => a,
    C   => IP(0),
    CE  => '1',
    CLR => '0',
    D   => '1'
);

OP(0) <= a;
```



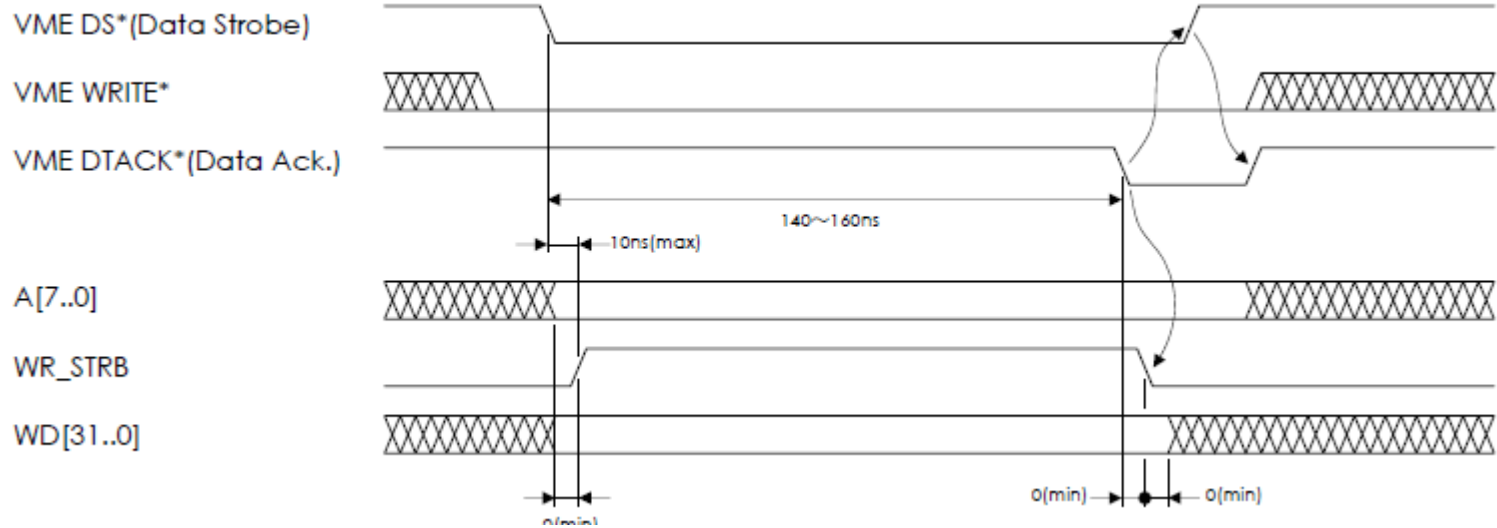
Accessing VME/CAMAC buses (Read for LUPO)

- ▶ Latch data during RD_STRB = '1'



Accessing VME/CAMAC buses (for LUPO)

- ▶ Data are read from WD at the transition timing of WE_STRB '1' to '0'



Example of data readout (LUPO)

- ▶ By using when statement, VME Address (CAMAC AF) is selected
- ▶ During RD_STRB='1', data must be latched on RD
 - RD <= val;
- ▶ Other cases, RD is unconnected
 - RD <= (others => 'z');

```
process(WR_STRB)
begin
if(WR_STRB'event and WR_STRB='0') then
  if(set = '1') then
    val <= WR;
  end if;
end if;
end process;

process(INIT, RD_STRB, WR_STRB)
begin
if(INIT = '1') then
  RD <= (others => 'Z');
  set <= '0';
elseif(RD_STRB = '1') then
  set <= '0';
  case A is
  when x"00" => RD <= val;
  when others => RD <= (others => '0');
  end case;
elseif(WR_STRB'event and WR_STRB = '1') then
  RD <= (others => 'Z');
  case A is
  when x"00" => set <= '1';
  when others => set <= '0';
  end case;
else
  RD <= (others => 'Z');
  set <= set;
end if;
end process;
```

elseif (WR_STRB = '1') then



Example of data write (LUPO)

1. Set a flag at WR_STRB='1'
 - set <= '1';
2. At WR_STRB'event and WR_STRB='0' timing, if flag is 1, set values
 - val <= WR;
3. When WR_STRB='0', value of set should be kept
 - set <= set;

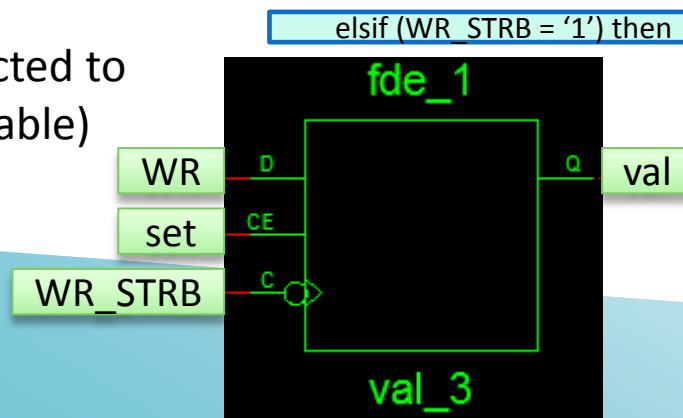
```
process(WR_STRB)
begin
if(WR_STRB'event and WR_STRB='0') then
  if(set = '1') then
    val <= WR;
  end if;
end if;
end process;
```

```
process(INIT, RD_STRB, WR_STRB)
begin
if(INIT = '1') then
  RD <= (others => 'Z');
  set <= '0';
elsif(RD_STRB = '1') then
  set <= '0';
  case A is
  when x"00" => RD <= val;
  when others => RD <= (others => '0');
  end case;
```

```
  elsif(WR_STRB'event and WR_STRB = '1') then
    RD <= (others => 'Z');
    case A is
    when x"00" => set <= '1';
    when others => set <= '0';
    end case;
```

```
  else
    RD <= (others => 'Z');
    set <= set;
  end if;
end process;
```

set is connected to
CE (clock enable)
of DFF



Make a scaler (counter)

- ▶ By using event, make a synchronous circuit
 - $cnt \leq cnt + 1$
- ▶ When data readout, data must be latched at the RD_STRB, otherwise you get invalid values, sometimes
 - If cnt value is changed during readout, it is bad
 - There are skew (- 20ps) between all bits

```
entity SCRWORK is
    Port ( c : in  STD_LOGIC;
          rst : in  STD_LOGIC;
          q : out  STD_LOGIC_VECTOR (3 downto 0));
end SCRWORK;

architecture Behavioral of SCRWORK is

    signal cnt : std_logic_vector(3 downto 0) := "0000";

begin

    q <= cnt;

    process(rst, c)
    begin
        if(rst = '1') then
            cnt <= (others => '0');
        elsif(c'event and c='1') then
            cnt <= cnt + 1;
        end if;
    end process;

end Behavioral;
```



Latching the value of Scaler (Counter)

- ▶ Prepare the vector for readout

```
elsif(c'event and c='1') then
    cnt <= cnt + 1;
    if(f = '0') then
        iq <= cnt + 1;
    else
        iq <= iq;
    end if;
end if;
```

It is not "iq <= cnt;"
If you this, you get value
of cnt - 1

```
entity SCRWORK is
    Port ( c : in  STD_LOGIC;
          f : in  STD_LOGIC;
          rst : in  STD_LOGIC;
          q : out STD_LOGIC_VECTOR (31 downto 0));
end SCRWORK;

architecture Behavioral of SCRWORK is

    signal cnt : std_logic_vector(31 downto 0) := x"00000000";
    signal iq : std_logic_vector(31 downto 0) := x"00000000";

begin

    q <= iq;

    process(rst, c)
    begin
        if(rst = '1') then
            cnt <= (others => '0');
            iq <= (others => '0');
        elsif(c'event and c='1') then
            cnt <= cnt + 1;
            if(f = '0') then
                iq <= cnt + 1;
            else
                iq <= iq;
            end if;
        end if;
    end process;

end Behavioral;
```



Declare useful libraries

- ▶ library IEEE;
- ▶ use IEEE.STD_LOGIC_1164.ALL;
- ▶ use IEEE.STD_LOGIC_ARITH.ALL;
- ▶ use IEEE.STD_LOGIC_UNSIGNED.ALL;
 - cnt <= cnt + 1 requires ARITH

```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use IEEE.STD_LOGIC_ARITH.ALL;  
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

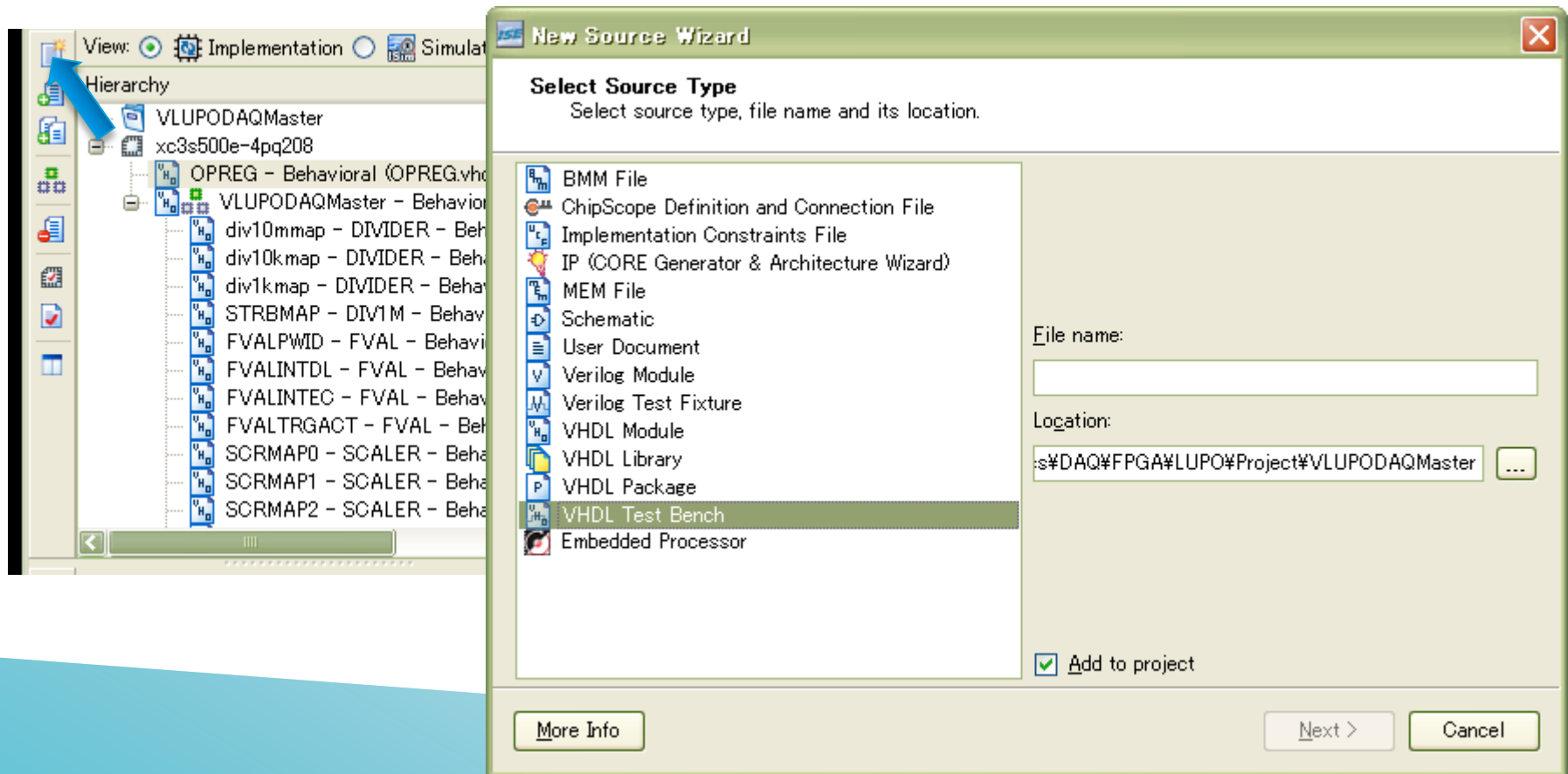
If you simulate with built in libraries such as FDCE, you have to uncomment like this

```
-- Uncomment the following library declaration if instantiating  
-- any Xilinx primitives in this code.  
library UNISIM;  
use UNISIM.VComponents.all;
```



Do simulation

▶ Create the Test Bench



The screenshot shows the Xilinx ISE IDE with the 'New Source Wizard' dialog box open. The wizard is titled 'New Source Wizard' and has a close button (X) in the top right corner. The main heading is 'Select Source Type' with the instruction 'Select source type, file name and its location.' Below this is a list of source types, with 'VHDL Test Bench' selected and highlighted. Other options include BMM File, ChipScope Definition and Connection File, Implementation Constraints File, IP (CORE Generator & Architecture Wizard), MEM File, Schematic, User Document, Verilog Module, Verilog Test Fixture, VHDL Module, VHDL Library, VHDL Package, and Embedded Processor. To the right of the list are two text input fields: 'File name:' and 'Location:'. The 'Location' field contains the path 's:\DAQ\FPGA\LUPO\Project\VLUPODAQMaster' and has a browse button (...). At the bottom right, there is a checked checkbox labeled 'Add to project'. At the bottom of the dialog are three buttons: 'More Info', 'Next >', and 'Cancel'. In the background, the ISE IDE is visible, showing the 'Hierarchy' pane with a tree view of the project 'VLUPODAQMaster' under 'xc3s500e-4pq208'. A blue arrow points to the 'New Source Wizard' icon in the top-left toolbar of the IDE.



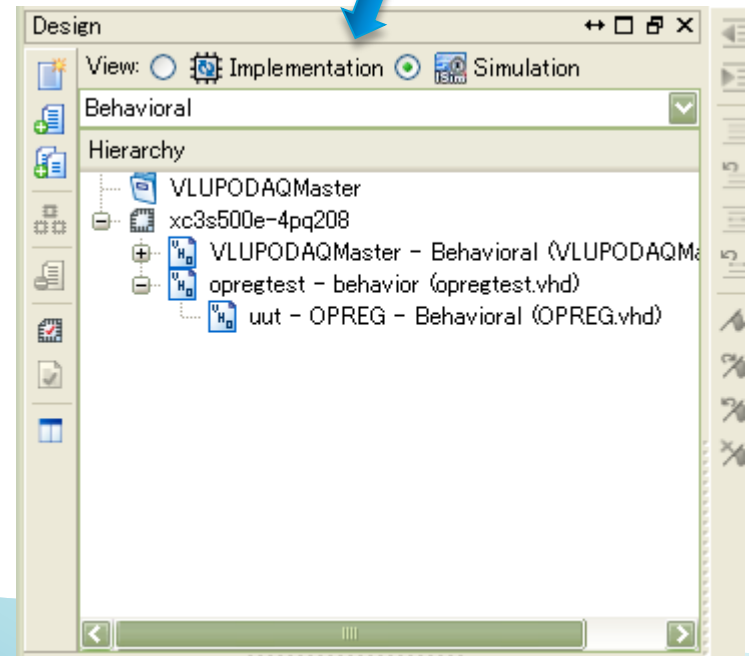
Set initial values

```
--Inputs
signal clk : std_logic := '0';
signal start : std_logic := '0';
signal level : std_logic := '0';
-- signal width : std_logic_vector(15 downto 0) := (others => '0');
signal width : std_logic_vector(15 downto 0) := x"000a";

--Outputs
signal q : std_logic;

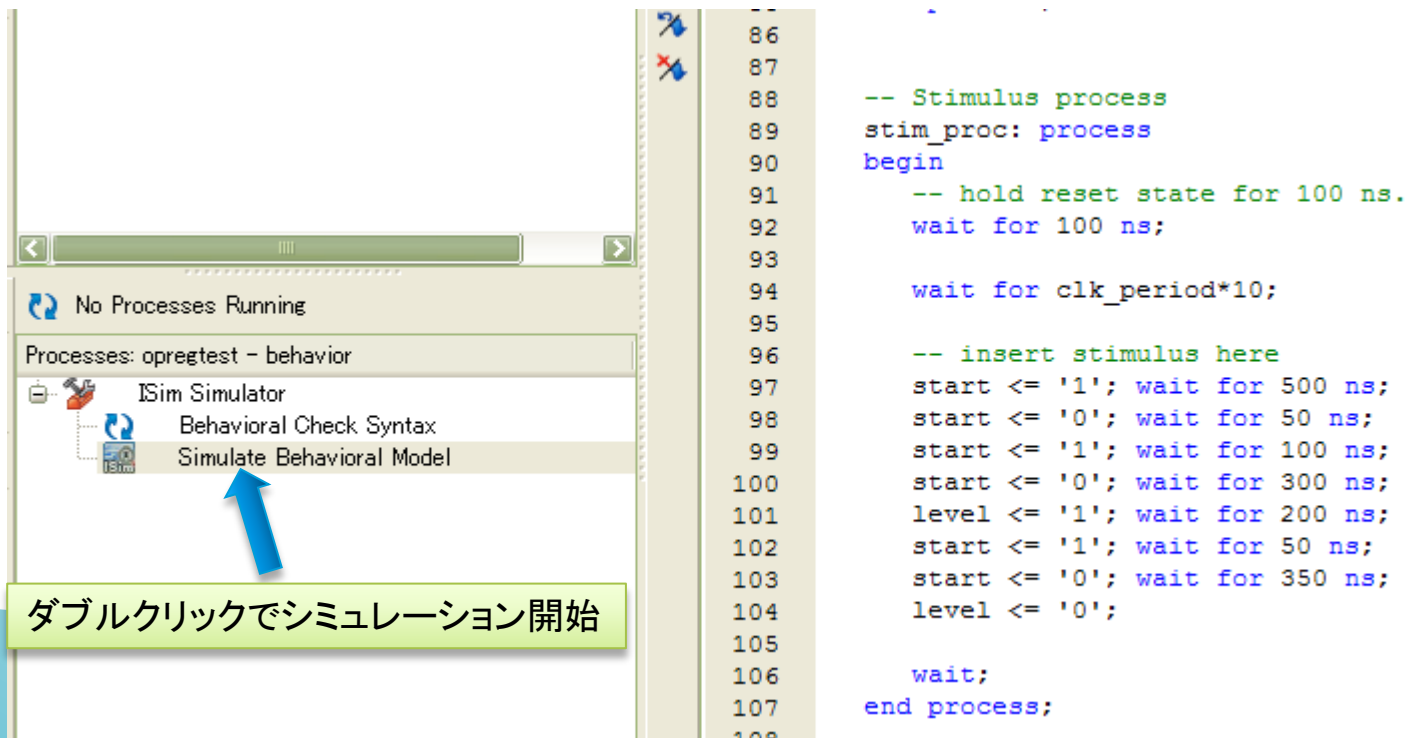
-- Clock period definitions
-- constant clk_period : time := 10 ns;
constant clk_period : time := 20 ns;
```

Implementation
Simulation
Switch



Write like this

- ▶ clock part are generated automatically
- ▶ wait for 100ns; means 100ns passing

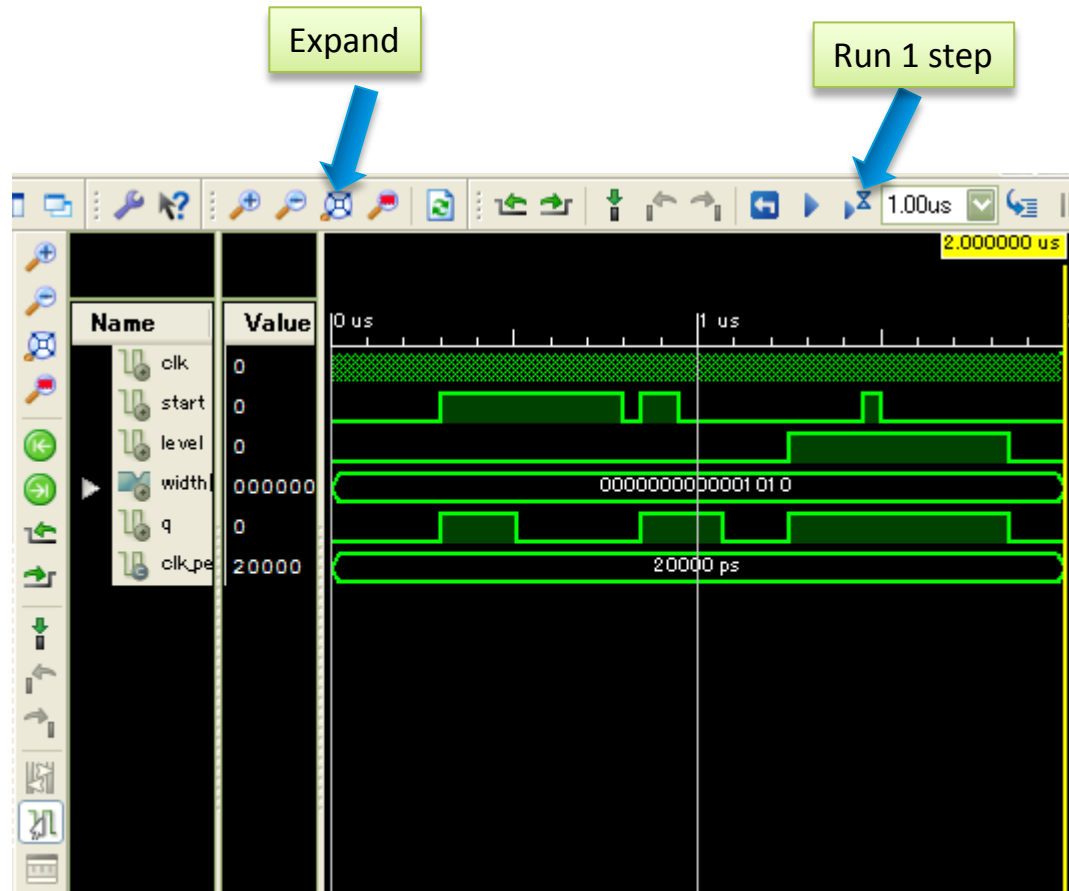


```
86
87
88 -- Stimulus process
89 stim_proc: process
90 begin
91     -- hold reset state for 100 ns.
92     wait for 100 ns;
93
94     wait for clk_period*10;
95
96     -- insert stimulus here
97     start <= '1'; wait for 500 ns;
98     start <= '0'; wait for 50 ns;
99     start <= '1'; wait for 100 ns;
100    start <= '0'; wait for 300 ns;
101    level <= '1'; wait for 200 ns;
102    start <= '1'; wait for 50 ns;
103    start <= '0'; wait for 350 ns;
104    level <= '0';
105
106    wait;
107 end process;
108
```

ダブルクリックでシミュレーション開始



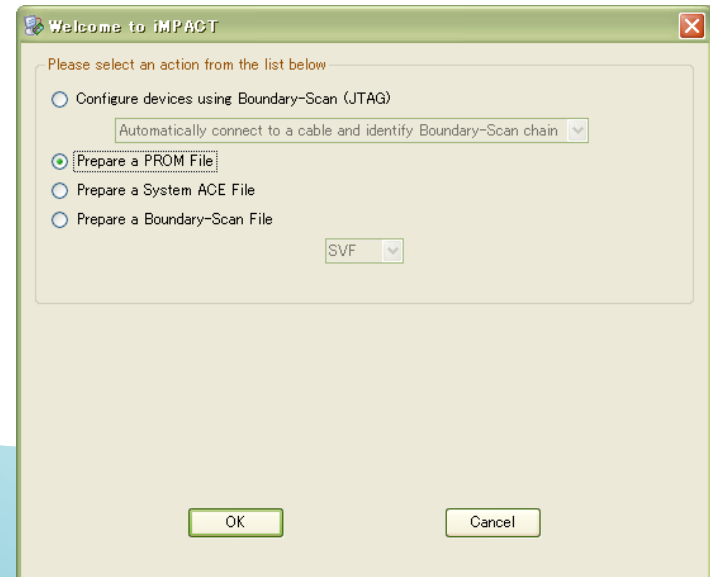
ISim



Download firmware to LUPO/GTO

- ▶ Generate Programming File
 - bit file is generated
- ▶ Configure Target Device
 - Launch iMPACT
 - Make file for the flash memory from bit file
 - Load into FPGA

First step, PROM = Flash Memory file



Setting of PROM file

GTO case, it is xcf02s

Step 1. Select Storage Target

Storage Device Type :

- Xilinx Flash/PROM
 - Non-Volatile FPGA
 - Spartan3AN
 - SPI Flash
 - Configure Single FPGA
 - Configure MultiBoot FPGA
 - BPI Flash
 - Configure Single FPGA
 - Configure MultiBoot FPGA
 - Configure from Paralleled PROMs
 - Generic Parallel PROM

Step 2. Add Storage Device(s)

Platform Flash

Device (bits): xcf04s [4 M]

Add Storage Device Remove Storage Device

xcf04s [4 M]

Auto Select PROM

Step 3. Enter Data

General File Detail	Value
Checksum Fill Value	FF
Output File Name	vlupoclockmanager
Output File Location	O:/Project/MLUPOClockGenerator

Flash/PROM File Property	Value
File Format	MCS
Add Non-Configuration Data Files	No

Description:

In this step, you will enter information to assist in setting up and generating a PROM file for the targeted storage device and mode.

- Checksum Fill Value:** When data is insufficient to fill the entire memory of a PROM, the value specified here is used to calculate the checksum of the unused portions.
- Output File Name:** This allows you to specify the base name of the file to which your PROM data will be written
- Output File Location:** This allows you to specify the directory in which the file named above will be created
- File Format:** PROM files can be generated in any number of industry standard formats. Depending on the PROM file format your PROM programmer uses, you output a MCS, HEX, UFP, ISC or BIN file. MCS is the most popular. ISC is used when targeting programming flows that utilize IEEE Std 1532. Third Party socket-based programmers usually accept any of the listed

OK Cancel Help



Generate PROM file

The screenshot shows the ISE iMPACT PROM File Formatter interface. The main window displays a diagram of the PROM file structure, with a green block labeled 'vlpoclockgenerator.bit' and a vertical bar labeled 'PROM / FLASH Bitstream'. The address range is from 0x0000_0000 to 0x0007_FFFF. A diagram on the right shows the flow from 'xc3s500e vlpoclockgenerator...' to 'xc3s500e PROM'.

By double click, file (*.mcs) will be generated**

By the wizard

1. [add a device ?] -> Yes
2. Select bit file
3. [any other device ?] -> No

Console output:

```
INFO:iMPACT:501 - '1': Added Device xc3s500e successfully.  
-----  
Add one device.454b9
```

PROM File Generation | Target Xilinx PROM | 2,270,208 Bits used | File: vlpoclockmanager in Location: C:\Documents and Settings\baba\My Documents\Physics\DAQ\FPGA\LUPO\Project



Download firmware

ISE iMPACT (R49d) - [Boundary Scan]

IMPACT Flows

- Boundary Scan
 - gscrgto.bit
 - gscrgto10.mcs
- SystemACE
- Create PROM File (PROM File For...)
- WebTalk Data

IMPACT Processes

Available Operations are:

- Program
- Verify
- Erase
- Blank Check
- Readback
- Get Device ID
- Get Device Checks
- Get Device Signature

Console

```
Writing file "C:\Users\baba\Documents\FPGA\GTO\Project\GSCRGTO10\gscrgto10.c  
'2': Loading file 'C:/Users/baba/Documents/FPGA/GTO/Project/GSCRGTO10/gscrgt  
done.
```

PROM File Formatter: Xilinx Flash/PROM

For FPGA,
cleared by
power cycle
(bit file)

For Flash ROM,
non-volatile
(mcs file)

Load FPGA option is useful, FPGA
will be reconfigured when Flash
ROM is updated

Device Programming Properties - Device 2 Programming Properties

Category

- Boundary-Scan
 - Device 1 (FPGA xc3s200a)
 - Device 2 (PROM xcf02s)

Property Name	Value
Verify	<input checked="" type="checkbox"/>
General CPLD And PROM Properties	
Erase The Entire Device	<input checked="" type="checkbox"/>
Read Protect	<input type="checkbox"/>
PROM/CoolRunner-II Usercode (8 Hex Digits)	
PROM Specific Properties	
Load FPGA	<input checked="" type="checkbox"/>

OK Cancel Apply Help



Ex. mistake case, conv_std_logic_vector

- ▶ Converting to the numerical value of 5000000 to the 24bits vector
- ▶ `conv_std_logic_vector(5000000, 24)`
 - Good
- ▶ `conv_std_logic_vector(24, 5000000)`
 - Converting the numerical value of 24 to the 5000000bits vector
 - Synthesize take so long time



“downto” and “to”

- ▶ signal a : std_logic_vector(3 **downto** 0) := “3210”
 - Using vector, **downto** is usual
 - Writing initial value with “bit”, it is common that the most right position is 0-th bit
- ▶ type ARINT is array(3 **downto** 0) of integer;
 - constant b : ARINT := (0,1,2,3);
 - It will be b(0)=3, b(1)=2, b(2)=1,b(3)=0
- ▶ type ARINT is array(0 **to** 3) of integer;
 - constant b : ARINT := (0,1,2,3);
 - b(0)=0, b(1)=1, b(2)=2,b(3)=3
 - Depending on custom,,, when array, **to** is easier to understand



If you find a warning of black box

- ▶ attribute box_type : string;
- ▶ attribute box_type of FDCE : component is "black_box";
- ▶ Warning will be solved

```
COMPONENT FDCE
generic (INIT : bit := '1');
PORT(
    Q   : OUT std_logic;
    C   : IN  std_logic;
    CE  : IN  std_logic;
    CLR : IN  std_logic;
    D   : IN  std_logic);
END COMPONENT;

attribute box_type : string;
attribute box_type of FDCE : component is "black_box";
```

- ▶ Or, you don't need to declare COMPONENT that are defined in Xilinx Library, simply uncomment these lines

```
-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
library UNISIM;
use UNISIM.VComponents.all;
```



Using LVDS

▶ IBUFDS Input



```
component IBUFDS
  port (
    O : out STD_LOGIC;
    I : in  STD_LOGIC;
    IB : in  STD_LOGIC);
end component;
```

```
LVDSIN_MAPgene : for i in 0 to 15 generate
LVDSIN_MAP : IBUFDS
  port map (
    O => LVDSio(i),
    I => LVDSp(i),
    IB => LVDSn(i));
end generate;
```

- For input case, write DIFF_TERM = TRUE
- Terminator is installed

```
INST "LVDSn*" DIFF_TERM = "TRUE";
INST "LVDSp*" DIFF_TERM = "TRUE";
```

▶ OBUFDS Output



```
component OBUFDS
  port (
    I : in  STD_LOGIC;
    O : out STD_LOGIC;
    OB : out STD_LOGIC);
end component;
```

```
LVDSOUT_MAPgene : for i in 0 to 15 generate
LVDSOUT_MAP : OBUFDS
  port map (
    I => LVDSio(i),
    O => LVDSp(i),
    OB => LVDSn(i));
end generate;
```

```
signal LVDSio : std_logic_vector(15 downto 0);
```



GCLK and BUFG

- ▶ Inputs assigned to GCLK can be used as a Clock (having small Skew and Delay)
 - IPO and LVDSClockp/n
- ▶ By connecting BUFG, other inputs are also can be used as a Clock (it will be the Clock of many other circuit components)
 - Basically, it is automatically connected at the Place & Route
 - Following declaration is necessary in UCF

```
NET "WR_STRB" CLOCK_DEDICATED_ROUTE = FALSE;
```

- ▶ With larger circuit, please check “Place & Route Report”
 - In case of Local (not BUFGMUX) and Skew is large, the wiring tuning with PlanAhead is necessary

Clock Net	Resource	Locked	Fanout	Net Skew(ns)	Max Delay(ns)
IP_2_IBUF	BUFGMUX_X1Y1	No	32	0.023	0.168
div1kmap/iq	BUFGMUX_X2Y0	No	32	0.017	0.165
trg	BUFGMUX_X2Y1	No	48	0.043	0.165
CLOCK_BUFGP	BUFGMUX_X2Y11	No	180	0.086	0.203
div10kmap/iq	BUFGMUX_X1Y0	No	32	0.019	0.164
IP_1_IBUF	BUFGMUX_X1Y11	No	32	0.018	0.168
rawtrg	BUFGMUX_X3Y8	No	33	0.033	0.092
IP_3_IBUF	BUFGMUX_X0Y8	No	32	0.021	0.113
div10mmap/iq	BUFGMUX_X3Y4	No	32	0.013	0.085
IP_0_IBUF	BUFGMUX_X1Y10	No	32	0.018	0.168
wrstrbg	BUFGMUX_X3Y9	No	76	0.063	0.122
Inst_TRGVETO/q1	Local		5	0.000	1.744

