

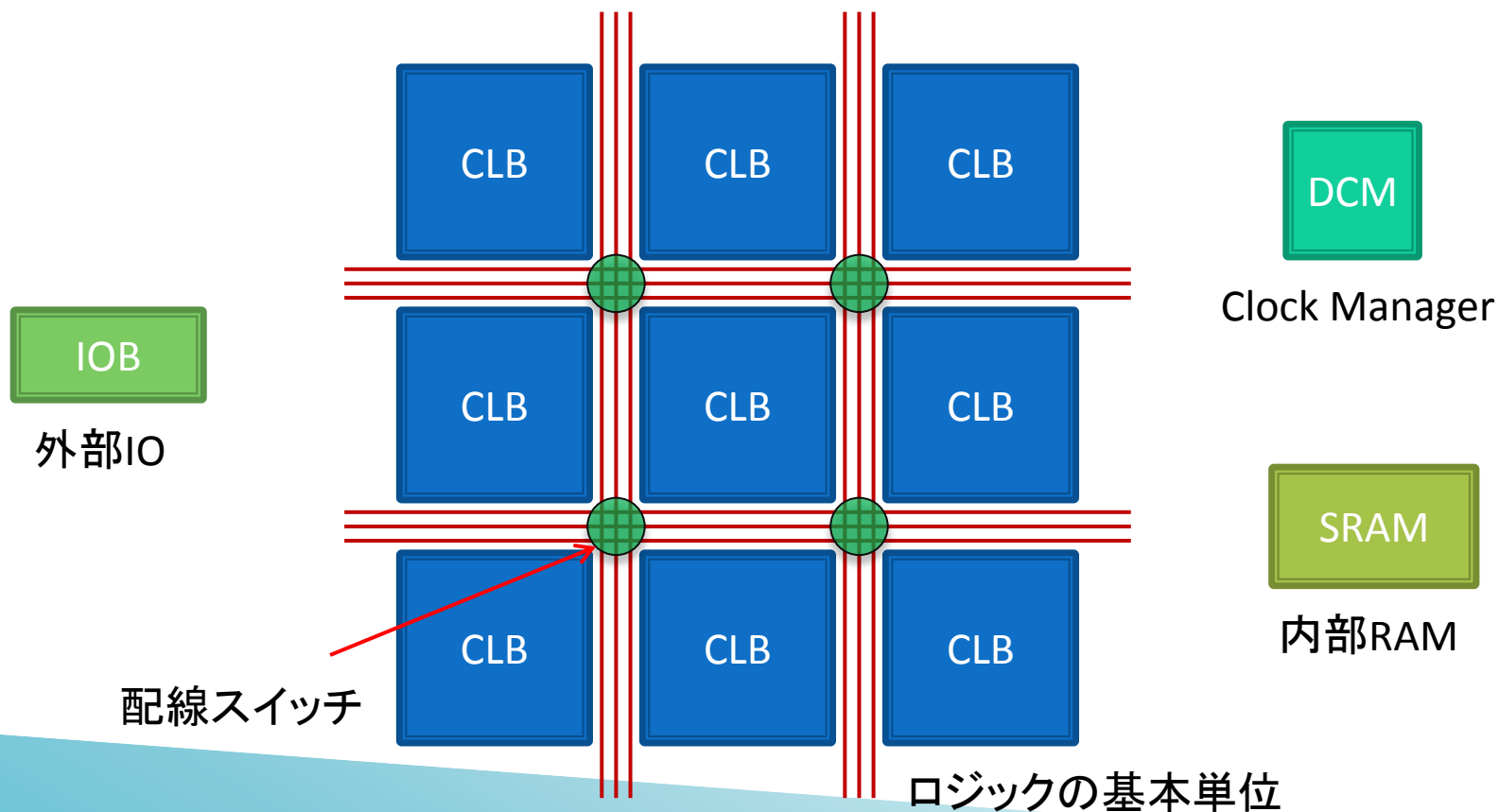
FPGA と LUPO その1

2010年8月31日
ばば

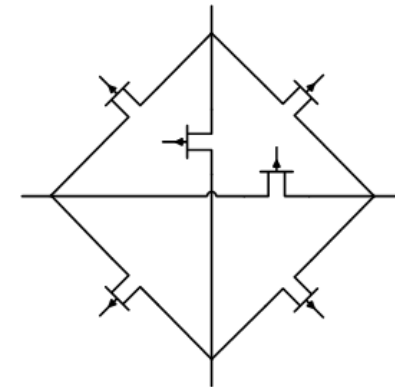
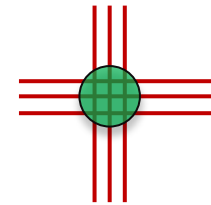
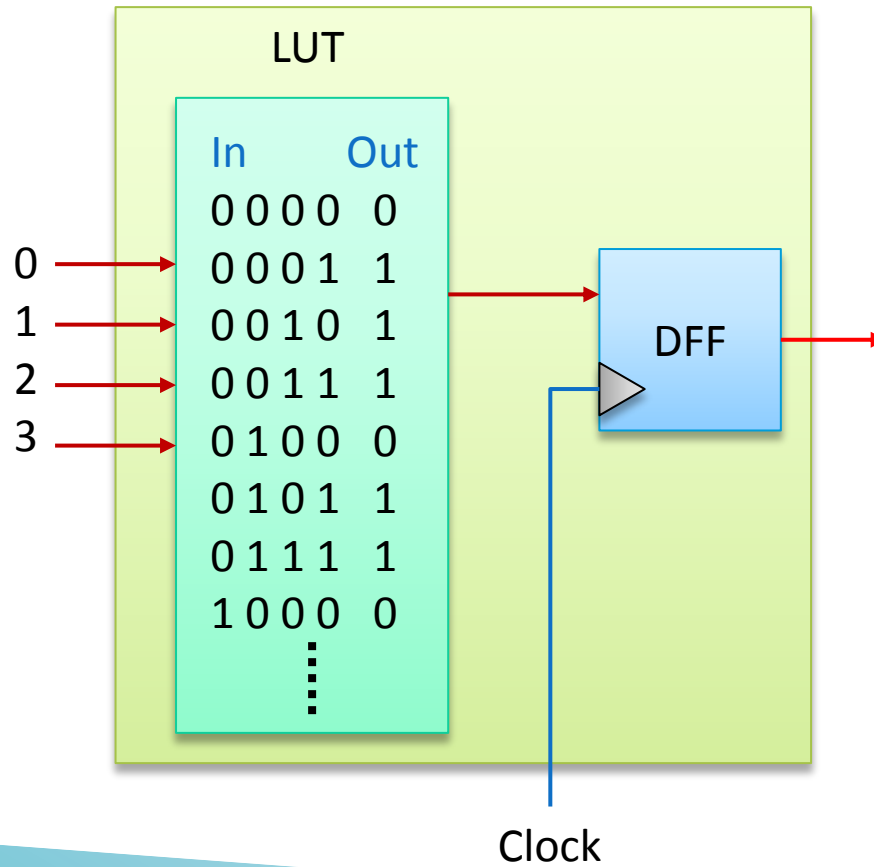


FPGAとは

▶ Field Programmable Gate Array



CLBと配線



LUPOについて

- ▶ NIMはLVTTTLに一回変換している
 - NIM->LVTTTL->FPGA
 - FPGA->LVTTTL->NIM
- ▶ LVDSはダイレクトに接続
 - IOBを2.5Vにしているので、2.5V系しか使えない

8LED

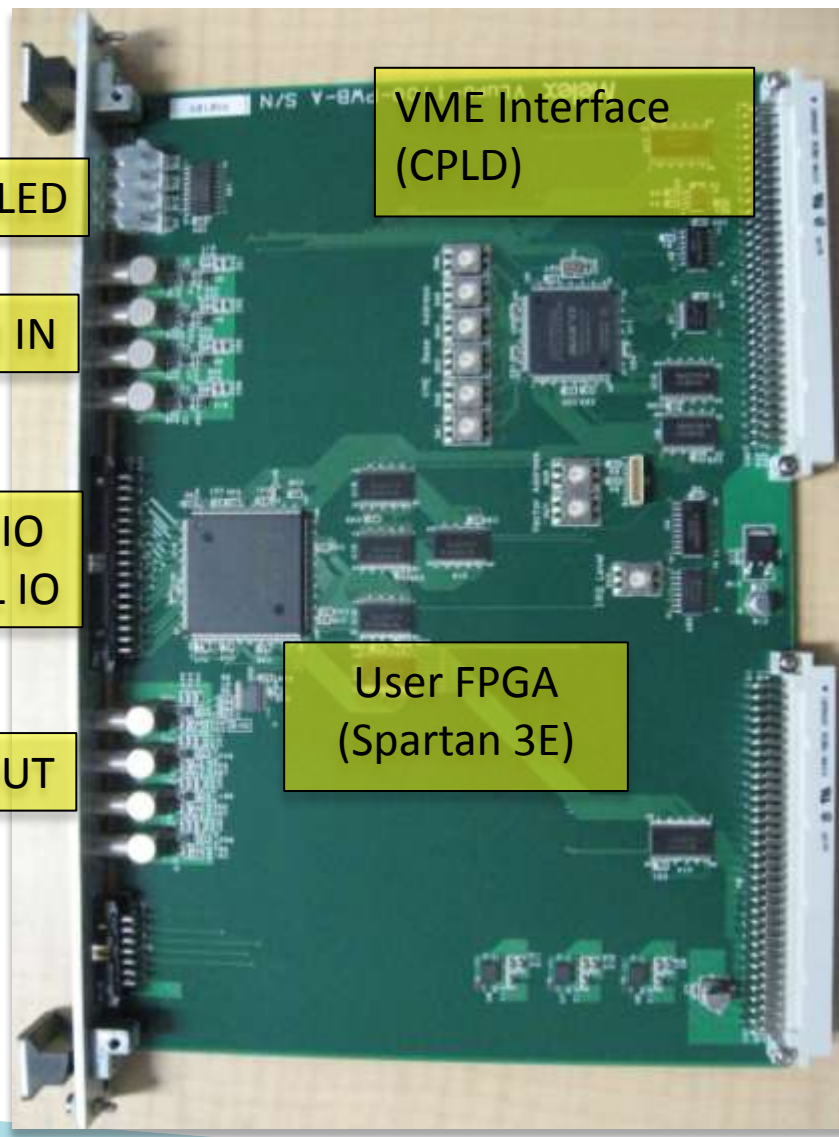
4NIM IN

16 LVDS IO
32 LVTTTL IO

4NIM OUT

VME Interface
(CPLD)

User FPGA
(Spartan 3E)



New Project

- ▶ DeviceとDesign flowは以下のとおり

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"/>



入出力の定義

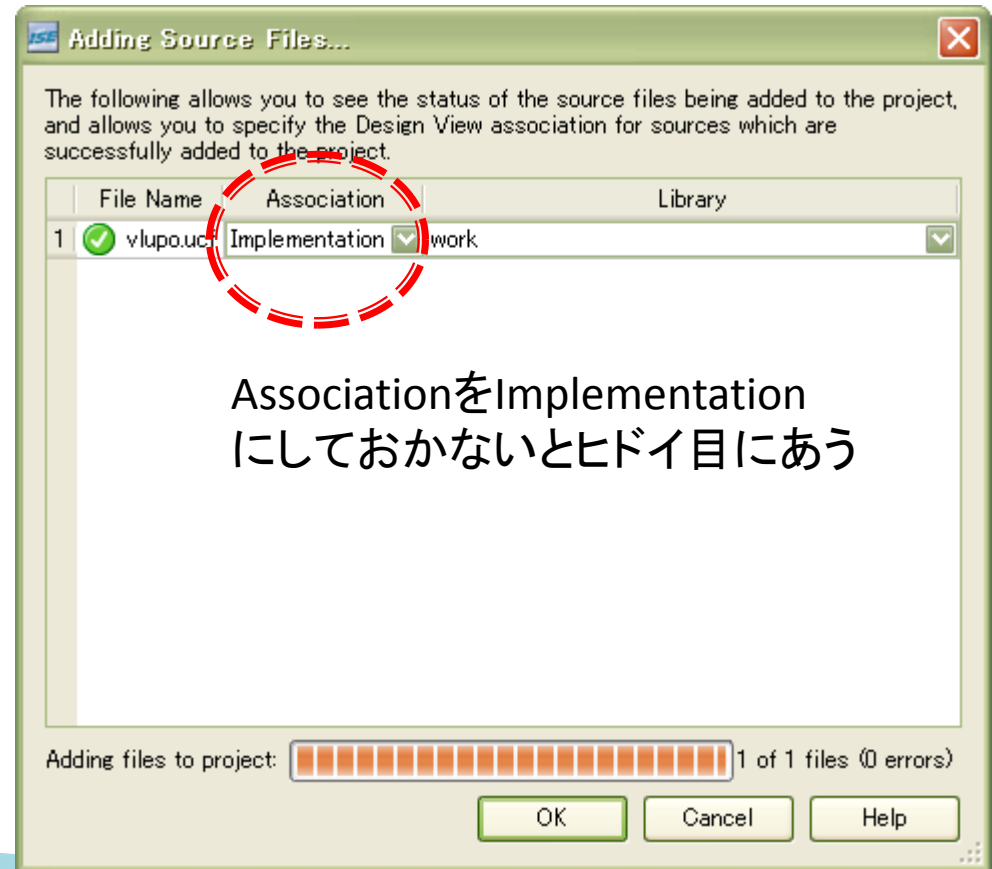
- ▶ LED : LED
- ▶ A : CAMAC/VMEアドレス
- ▶ CLOCK : 50MHz clock
- ▶ INIT : CAMAC Z/C, VME Init?
- ▶ IP : NIM Input 0-3
 - IP0: GCLK接続のIP0
- ▶ OP : NIM Output 0-3
- ▶ LVDSn/LVDSp : LVDS IO 0-15
 - LVDS_CLKn/p : GCLK接続のLVDS
- ▶ RD : Data Read
- ▶ WR : Data Write
- ▶ RD_STRB : Read strobe
- ▶ WR_STRB : Write 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;
    -- IP0       : 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;
```



制約ファイル(UCF)をコピー

- ▶ vlupo.ucf
- ▶ IO等の制約を書く



CAMAC/VMEを使わない場合は

▶ これで十分

```
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;
```

UCFに書いてあるNETは
entityに書いておかないとエラー

vlupo.ucfを Edit Constraints(Text)で編集

```
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 = LVCMOS25 ;
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 = LVCMOS25 ;
NET "OP[1]" LOC = "P62" | IOSTANDARD = LVCMOS25 ;
NET "OP[2]" LOC = "P61" | IOSTANDARD = LVCMOS25 ;
NET "OP[3]" LOC = "P60" | IOSTANDARD = LVCMOS25 ;
NET "CLOCK" TNM_NET = CLOCK;
TIMESPEC "TS_CLOCK" = PERIOD "CLOCK" 20 ns HIGH 50 %;
```



簡単に書くと

▶ こんな感じ。

```
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;

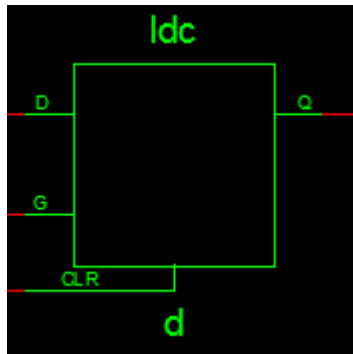
    OP(0) <= a;
    OP(1) <= b;
    OP(2) <= c;
    OP(3) <= CLOCK;
    LED(3 downto 0) <= IP(3 downto 0);
    LED(7 downto 4) <= (others => '1');
```

Synthesize -> Implement Design まで通ればたいていOK



同期回路を作る場合

- ▶ if, elseを使うときは使うsignalに対しすべて明示的に値を入れておくのが吉
 - Warningで確認すべし



非同期的な回路
で作られる

```
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;
```

この場合はちゃんとelseの部分で明示的に値を代入してあげること

xst

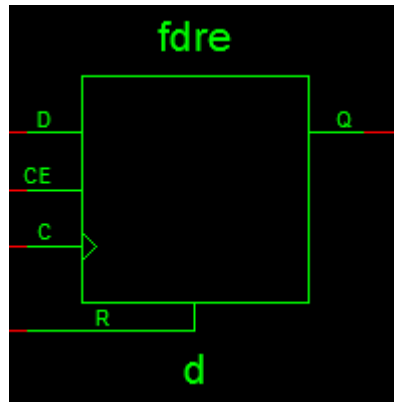


Xst:737 - Found 1-bit latch for signal <d>. Latches may be generated from incomplete case or if statements. We do not recommend the use of latches in FPGA/CPLD designs, as they may lead to timing problems.



こういうのは大丈夫な場合が多い

- ▶ event同期にしてあげると、DFFを使ってくれるので期待通りに値が保持される



Clockのタイミングでしか値が変化しない

```
signal d, e : std_logic := '0';

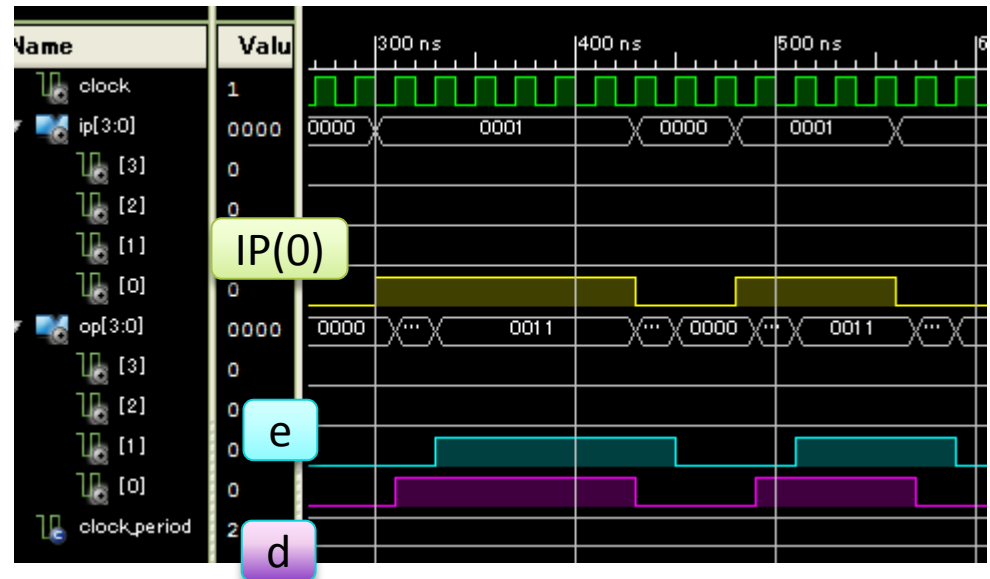
begin

    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;
```



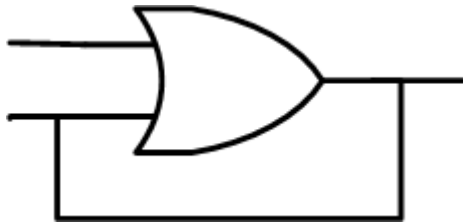
基本的にすべての行は同時に動作する

```
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;
```

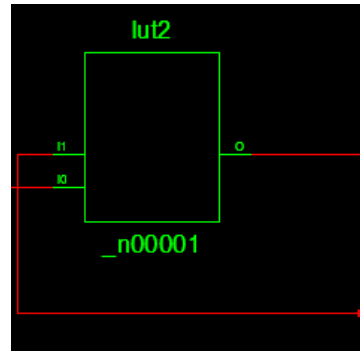



ループ回路は注意

- ▶ 回路図的には以下でLatchが作れるが...



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



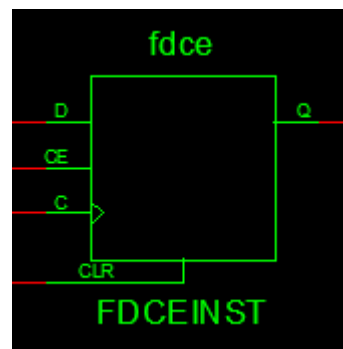
xst  Xst:2170 - Unit HOG: the following signal(s) form a combinational loop:

- ▶ Combinational loopがあると言われる
 - パルス幅、伝達経路delay、ジッタ等々でうまくいかないこと多し



明示的にDFFを使うと安心

- ▶ process文でeventを使うと基本的にDFFにしてくれる
- ▶ 回路要素としてFDCE(=DFF)を使うと期待通りの動作をしてくれるでしょう



```
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

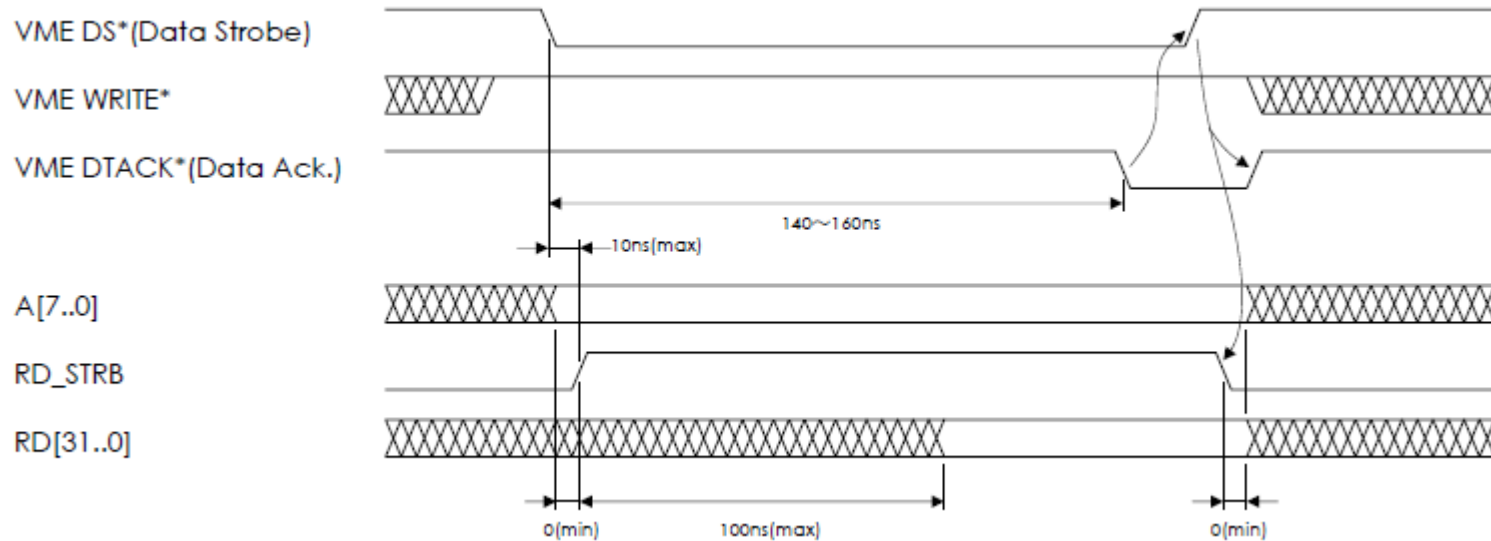
--a <= IP(0) or a;
FDCEINST: FDCE
generic map(
    INIT => '0')
PORT MAP(
    Q    => a,
    C    => IP(0),
    CE   => '1',
    CLR  => '0',
    D    => '1'
);

OP(0) <= a;
```



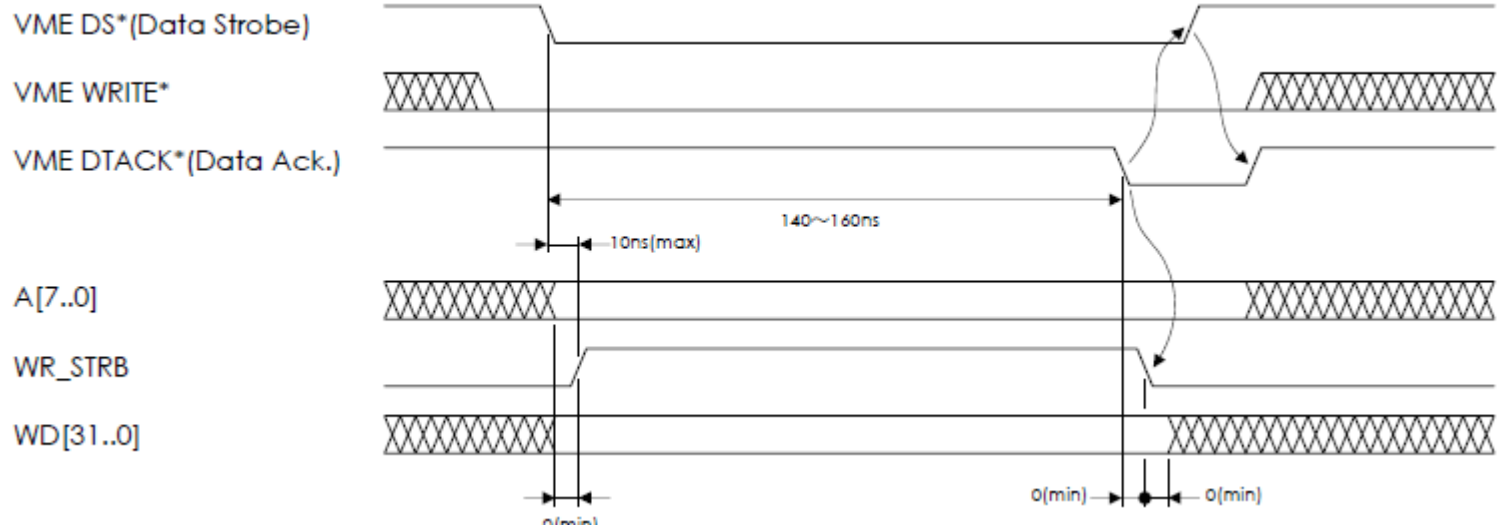
VME/CAMACバスとのやりとり(Read)

- ▶ RD_STRBが'1'の時にRDにデータを保持しておく



VME/CAMACバスとのやりとり (Write)

- ▶ WE_STRBが'1'から'0'に遷移したタイミングでWDからデータを取り込む



実際に値を読み出す場合

- ▶ whenでVME Address (CAMAC AF) を場合分け
- ▶ RD_STRB='1'の間だけRDに値を保持しておけばよい
 - RD <= val;
- ▶ それ以外はRDは切り離しておく
 - 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';
  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;
```

elsif (WR_STRB = '1') then



実際に値を書き込む場合

1. WR_STRB='1'の時にフラグを立てておく
 - set <= '1';
2. WR_STRB'event and WR_STRB='0'の時にフラグが立っていれば値を代入する
 - val <= WR;
3. WR_STRB='0'になった時でもsetの値は保持しておく
 - set <= set;

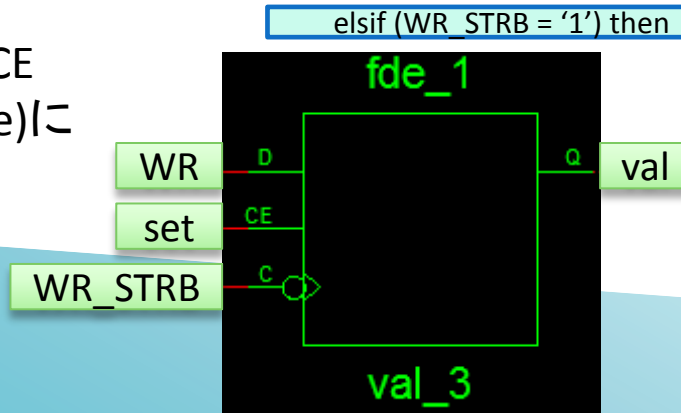
```
process(WR_STRB)
begin
  if(WR_STRB'event and WR_STRB='0') then
    if(set = '1') then
      val <= WR;
    end if;
  end if;
end process;
```

```
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がDFFのCE
(clock enable)に
接続される



スケーラ(カウンタ)を作る

- ▶ eventを使って同期回路を作ればよい
 - $\text{cnt} \leq \text{cnt} + 1$
- ▶ データを読み出したい場合はRD_STRB時に値をラッチしてあげないと時々不正な値を返す
 - 読み出し中にcntの値が変わるとアウト
 - 各ビット間で必ずskew(～20ps)がある

```
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;
```



スケーラ(カウンタ)値をラッチする

- ▶ 読み出し用のvectorを作っておける

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

iq <= cnt; ではない。
こうすると cnt - 1 を
読み出すことになる。

```
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;
```



使うライブラリを書いておく

- ▶ library IEEE;
- ▶ use IEEE.STD_LOGIC_1164.ALL;
- ▶ use IEEE.STD_LOGIC_ARITH.ALL;
- ▶ use IEEE.STD_LOGIC_UNSIGNED.ALL;
- これらARITH等が無いと `cnt <= cnt + 1` 等ができない

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

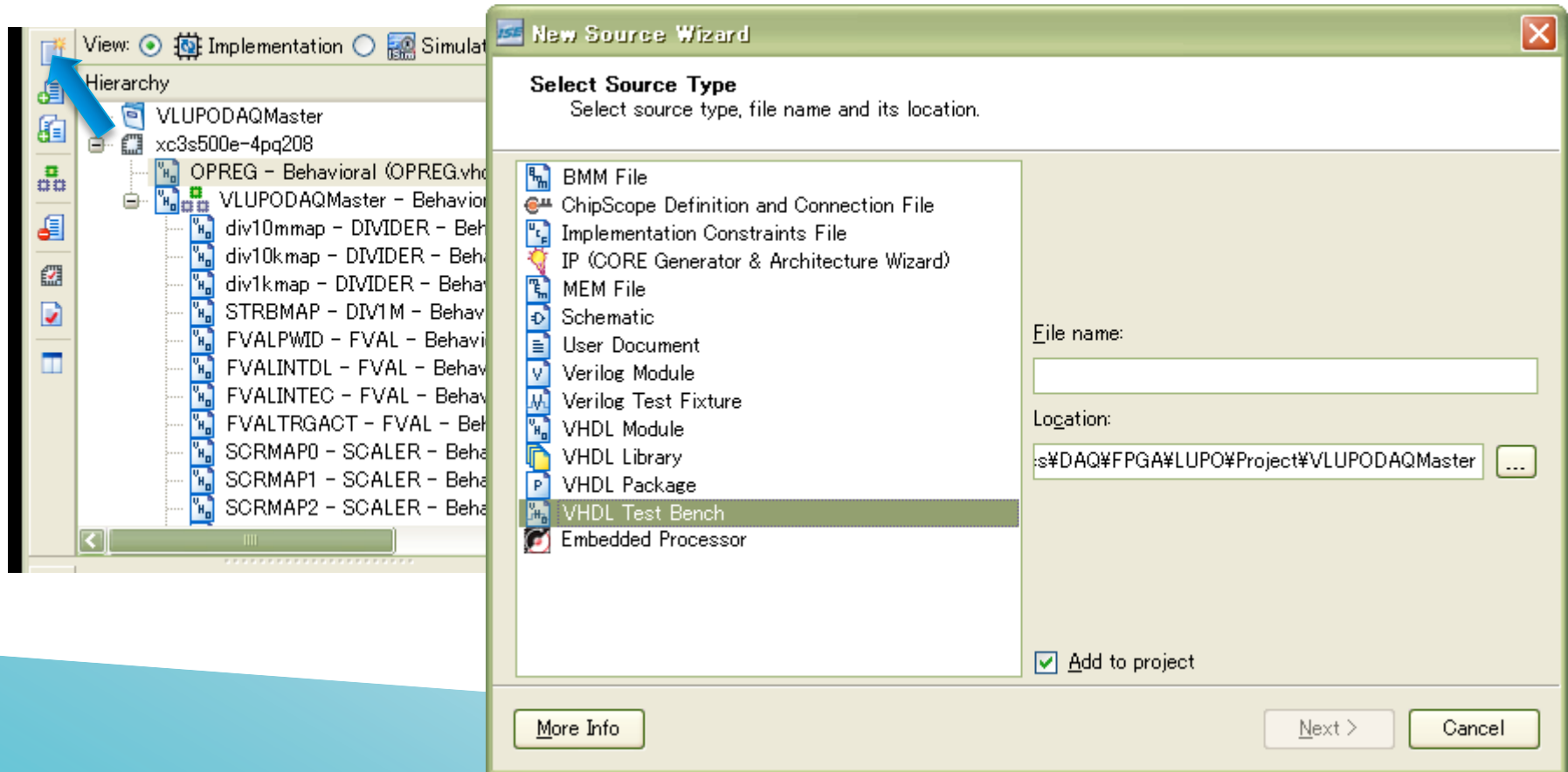
ちなみにFDCE等組み込みライブラリを使ったものをシミュレーションする場合は下記のようにコメントアウトしておく

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



シミュレーションする

▶ テストベンチをつくる



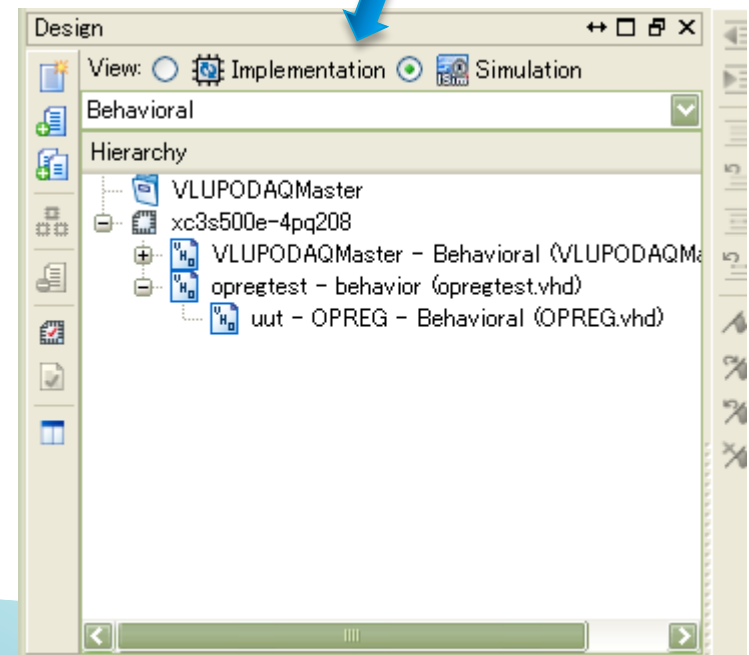
初期値を設定

```
--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;

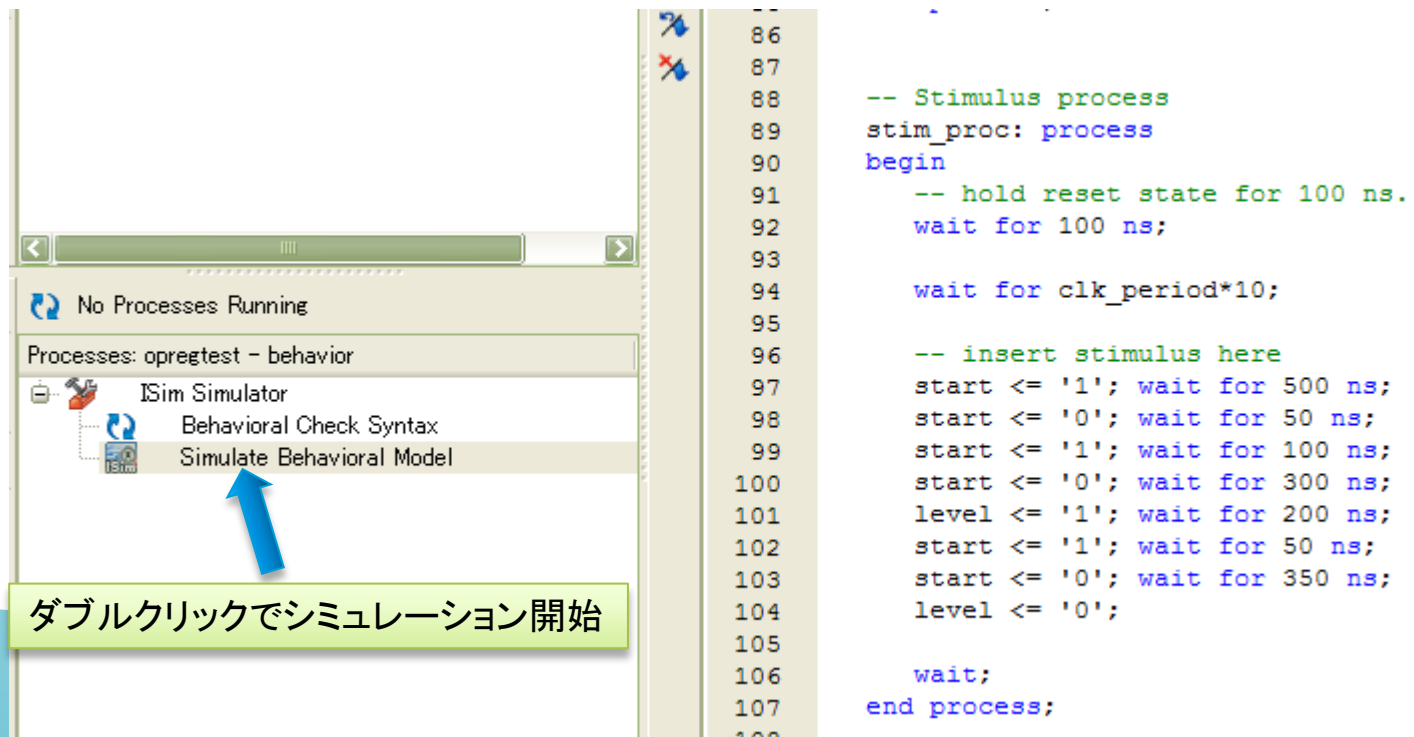
-- Clock period definitions
-- constant clk_period : time := 10 ns;
constant clk_period : time := 20 ns;
```

Implementation
Simulation
切り替え



こんな感じで書く

- ▶ clock部分は自動的に生成してくれる
- ▶ wait for 100ns; で100ns時間を進める



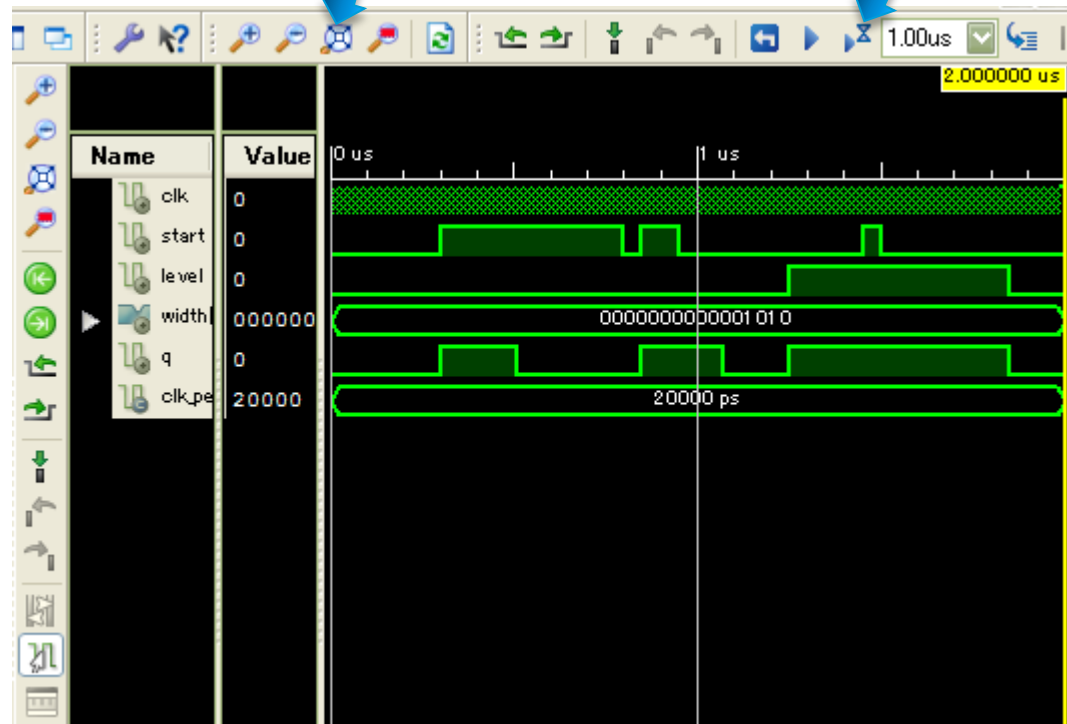
```
86
87
88 -- Stimulus process
89 stim_proc: process
90 begin
91     -- hold reset state for 100 ns.
92     wait for 100 ns;
93
94     wait for clk_period*10;
95
96     -- insert stimulus here
97     start <= '1'; wait for 500 ns;
98     start <= '0'; wait for 50 ns;
99     start <= '1'; wait for 100 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;
```



ISimの画面

全体を表示

1ステップ進める



LUPOにダウンロードする

- ▶ Generate Programming File
 - bitファイルができる
- ▶ Configure Target Device
 - iMPACTを起動
 - bitファイルからFlash memory用のファイルを作る
 - FPGAに書き込む

まずはPROM = Flash Memory用ファイル



PROM用ファイル設定

PROM File Formatter

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

Xilinx Flash/PROM
を選んで押す

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

PROMファイルを作る

The screenshot shows the ISE iMPACT (M.53d) - [PROM File Formatter: Xilinx Flash/PROM] window. The interface includes a menu bar (File, Edit, View, Operations, Output, Debug, Window, Help), a toolbar, and several panes. The 'iMPACT Flows' pane on the left lists 'Boundary Scan', 'SystemACE', and 'Create PROM File (PROM File Form...'. The 'iMPACT Processes' pane below it shows 'Available Operations are:' with 'Generate File...' highlighted. A blue arrow points from a text box to this option. The main workspace displays a diagram of the PROM file structure, showing a 'vlpoclockgenerator.bit' file being processed into a 'PROM / FLASH' block. The address range is from 0x0000_0000 to 0x0007_FFFF. A diagram on the right shows the connection between 'xc3s500e' and 'vlpoclockgenerator...'. The 'Console' pane at the bottom shows the message: 'INFO:iMPACT:501 - '1': Added Device xc3s500e successfully.' and 'Add one device.454b9'. The status bar at the bottom indicates 'PROM File Generation', 'Target Xilinx PROM', '2,270,208 Bits used', and the file path: 'File: vlpoclockmanager in Location: C:\Documents and Settings\baba\My Documents\Physics\DAQ\FPGA\LUPO\Project'.

ここをダブルクリックでファイルができる
MCSという拡張子

ウィザード形式で
1. デバイスを追加するか聞かれるのでYes
2. bitファイルを選ぶ
3. さらに用意するか聞かれるのでNo



やらかした例 `conv_std_logic_vector`

- ▶ 5000000という数字を24bitのベクターに変換したい。
- ▶ `conv_std_logic_vector(5000000, 24)`
 - 正しい。
- ▶ `conv_std_logic_vector(24, 5000000)`
 - 24という数字を5000000bitのvectorに変換しようとする。
 - Synthesizeがいつこうに終わらなくなる・・・(いつかは終わる)



downto (降順)と to (昇順)

- ▶ signal a : std_logic_vector(3 downto 0) := "3210"
 - vectorを使う時はdowntoが一般的
 - bitで初期値を書く時は一番右が0bit目なのが馴染み
- ▶ type ARINT is array(3 downto 0) of integer;
 - constant b : ARINT := (0,1,2,3);
 - b(0)=3, b(1)=2, b(2)=1, b(3)=0 になるので注意
- ▶ 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
 - 習慣の問題ですが・・・arrayの時はtoの方が馴染みやすい



black boxと言われる場合

- ▶ attribute box_type : string;
- ▶ attribute box_type of FDCE : component is "black_box";
- ▶ と書いておけばWarningが出なくなる

```
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";
```



LVDSを使う

▶ IBUFDS 入力



```
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;
```

- 入力の場合はUCFファイルのDIFF_TERMをTRUEにしておく
- これでターミネータが入る

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

▶ OBUFDS 出力



```
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とBUFG

- ▶ GCLKにアサインされているものはClock(配線時にSkew, Delayが少ない)として使える
 - IP0とLVDSClockp/n
- ▶ それ以外でClock(多数の回路のClockとなるもの)として使いたい場合はBUFGにつなげる
 - Place & Route時に必要に応じて自動でつなげてくれる
 - UCFで以下の記述が必要

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

- ▶ 回路規模が大きくなってきたらPlace & Route Reportは要チェック
 - BUFGMUXではなくLocalになっていて、Skewが大きい場合はPlanAheadを使って配線調整

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

