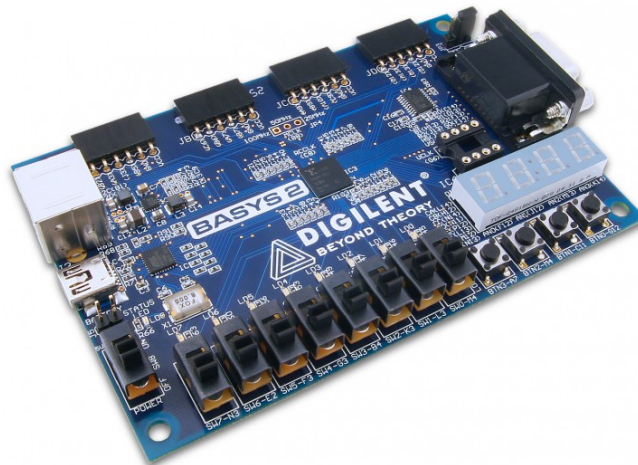
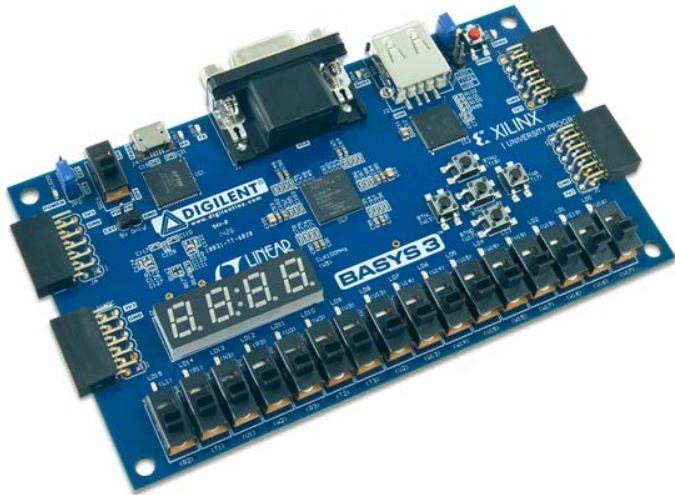


Interfaces



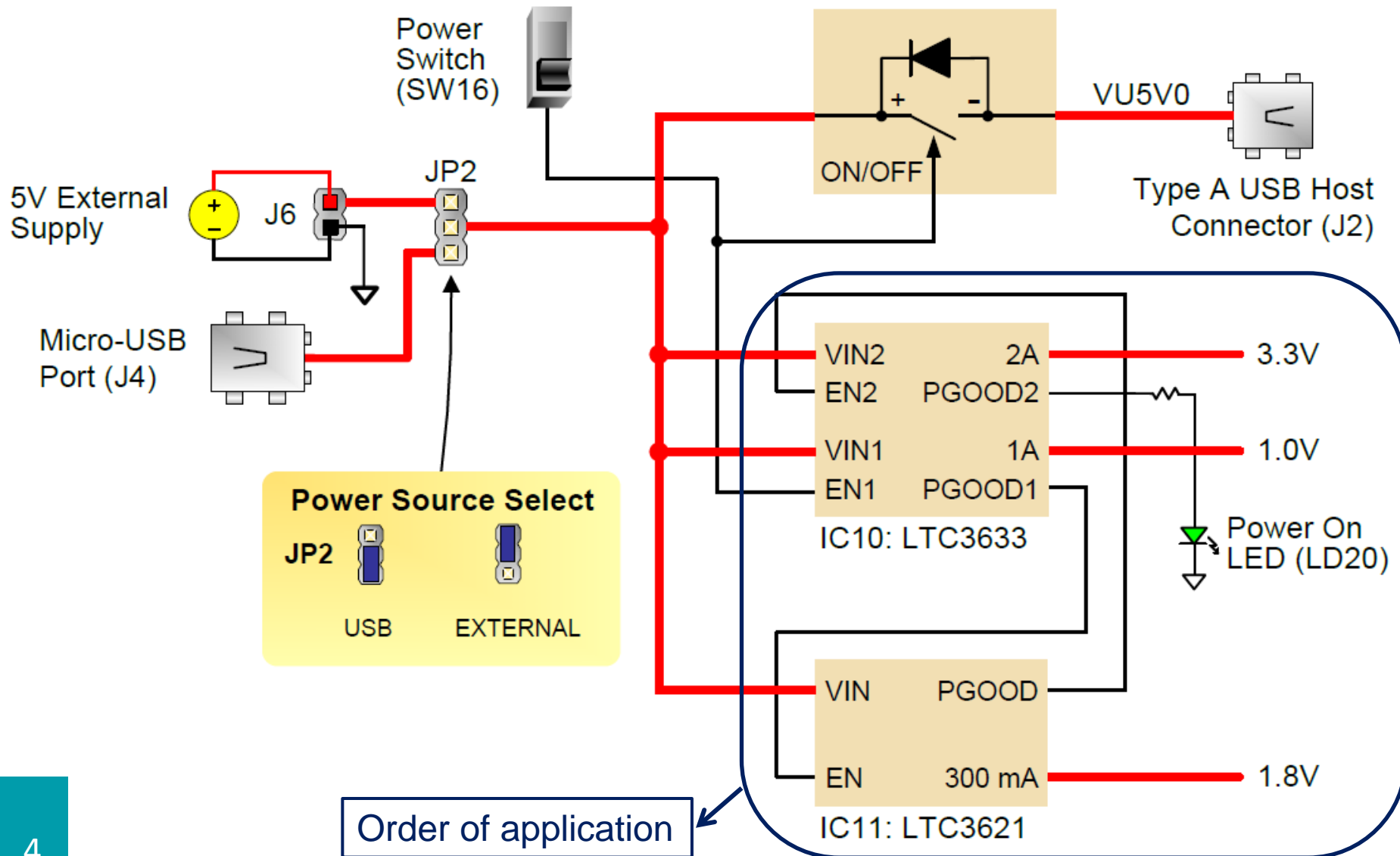
Content

- Basys3
 - Power Supplies
 - Configuration
 - VGA
 - With simple memory
 - With Block RAM IP
 - With Clock IP
 - Oscillators / Clocks
 - USB-UART Bridge
 - USB HID Host
 - PS2
 - Basic I/O
 - Pmod Controllers

Content

- Buttons
- Rotary Encoder
- Language templates
- IP catalog

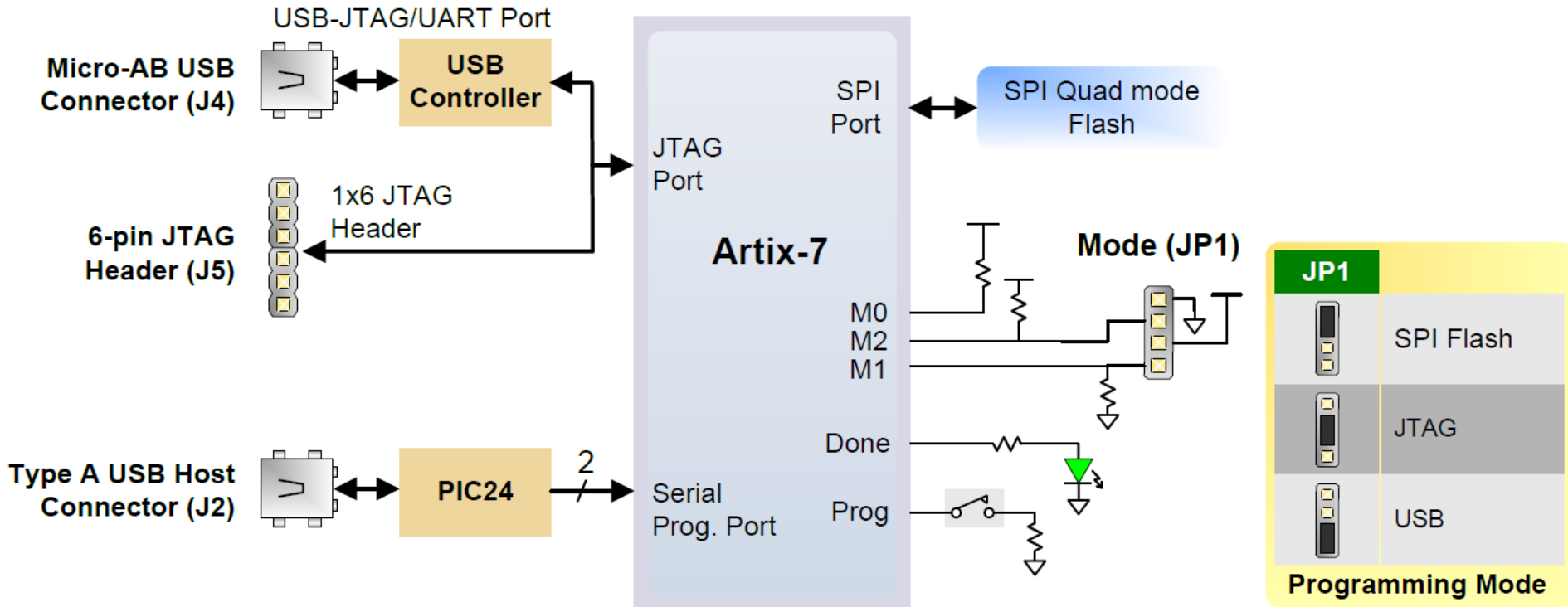
Basys3 : Power Supplies



Basys3 : Power Supplies

Supply	Circuits	Device	Current (max/typical)
3.3V	FPGA I/O, USB ports, Clocks, Flash, PMODs	IC10: LTC3633	2A/0.1 to 1.5A
1.0V	FPGA Core	IC10: LTC3633	2A/ 0.2 to 1.3A
1.8V	FPGA Auxiliary and Ram	IC11: LTC3621	300mA/ 0.05 to 0.15A

Basys3 : Configuration

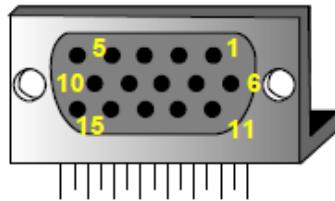


Basys3 : Configuration

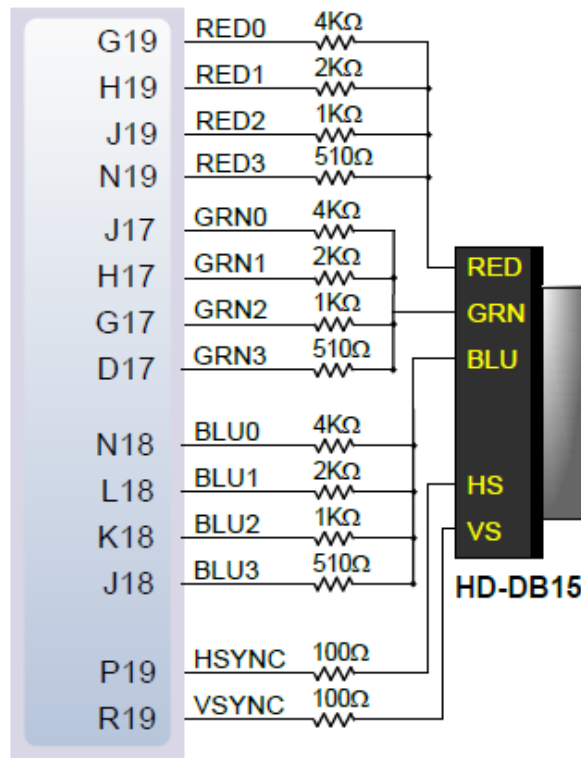
You can program the FPGA from a pen drive attached to the USB-HID port (J2) by doing the following:

1. Format the storage device (Pen drive) with a FAT32 file system.
2. Place a single .bit configuration file in the root directory of the storage device.
3. Attach the storage device to the Basys3.
4. Set the JP1 Programming Mode jumper on the Basys3 to “USB”.
5. Push the PROG button or power-cycle the Basys3.

Basys3: VGA port



Pin 1: Red	Pin 5: GND
Pin 2: Grn	Pin 6: Red GND
Pin 3: Blue	Pin 7: Grn GND
Pin 13: HS	Pin 8: Blu GND
Pin 14: VS	Pin 10: Sync GND

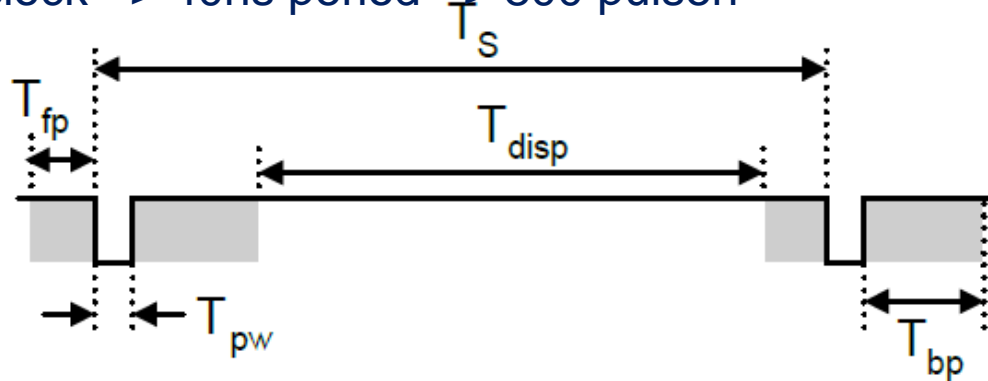


Artix-7

4096 colors = 2^{12}

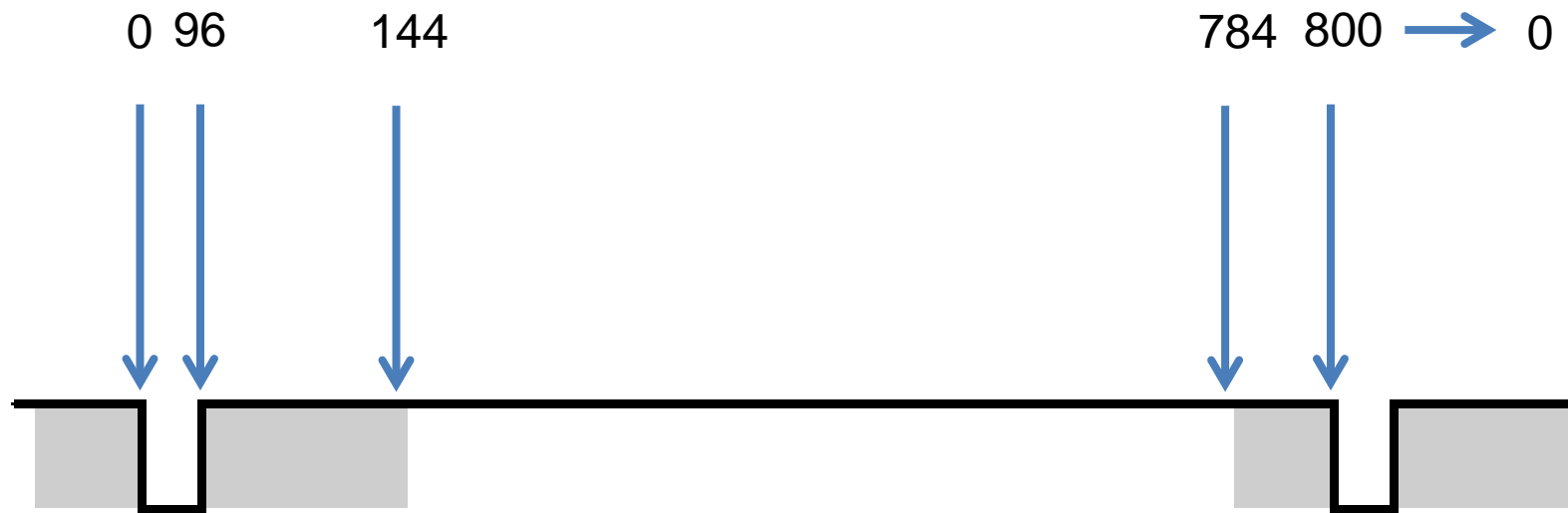
Basys3: VGA port

- 640 by 480 mode
- 60Hz refresh rate \Rightarrow 16.7ms period \Rightarrow $16.7\text{ms}/521 = 32\mu\text{s}$ line time
- 25MHz clock \Rightarrow 40ns period \Rightarrow 800 pulsen

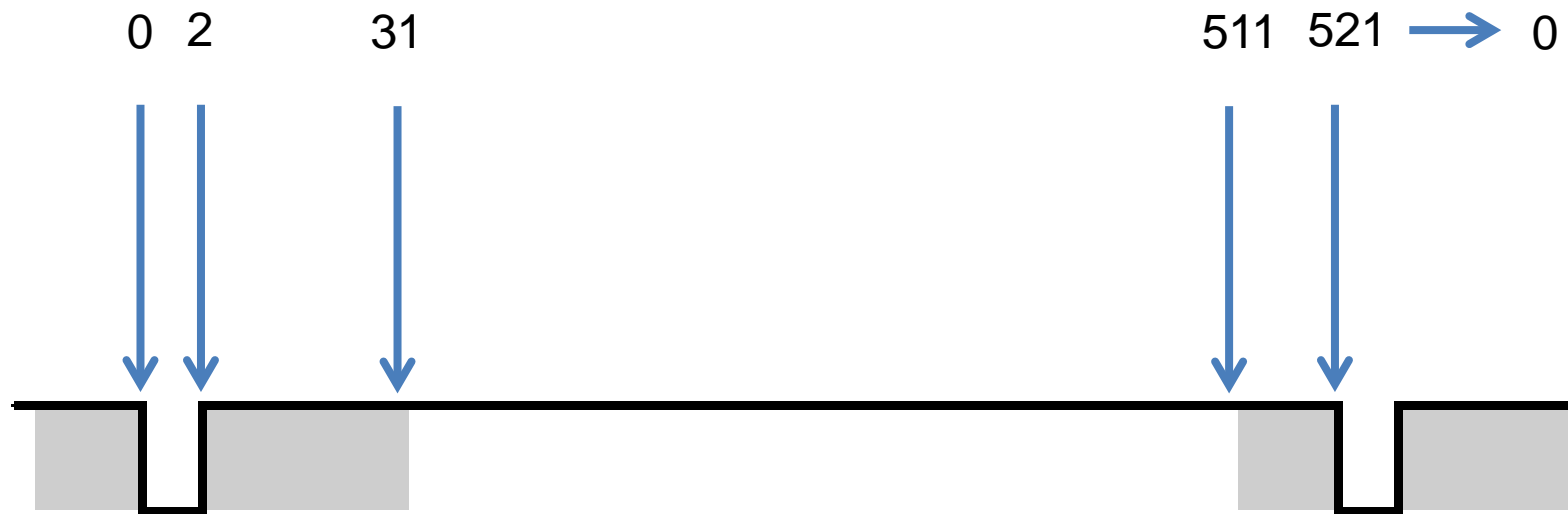


Symbol	Parameter	Vertical Sync			Horiz. Sync	
		Time	Clocks	Lines	Time	Clks
T_S	Sync pulse	16.7ms	416,800	521	32 μs	800
T_{disp}	Display time	15.36ms	384,000	480	25.6 μs	640
T_{pw}	Pulse width	64 μs	1,600	2	3.84 μs	96
T_{fp}	Front porch	320 μs	8,000	10	640 ns	16
T_{bp}	Back porch	928 μs	23,200	29	1.92 μs	48

Basys3: VGA port: horizontal counter



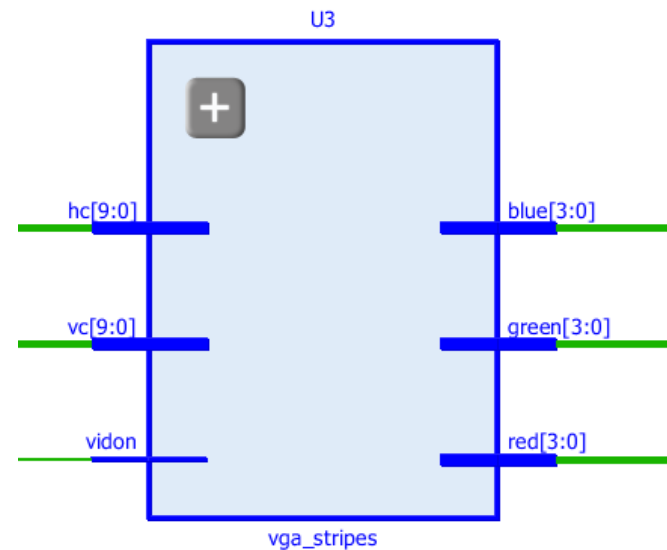
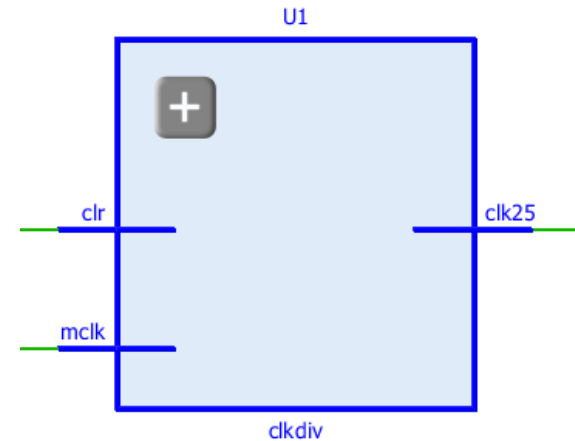
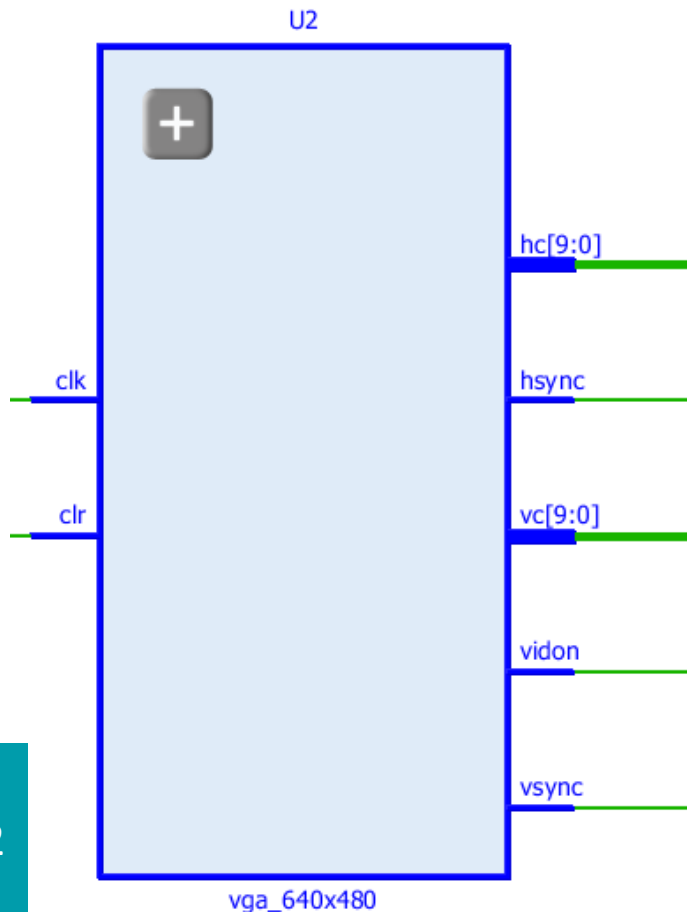
Basys3: VGA port: vertical counter



Basys3: VGA port

We need a component for:

- Build a 25MHz clock signal
- Synchronization
- vgaRed / vgaGreen / vgaBlue output



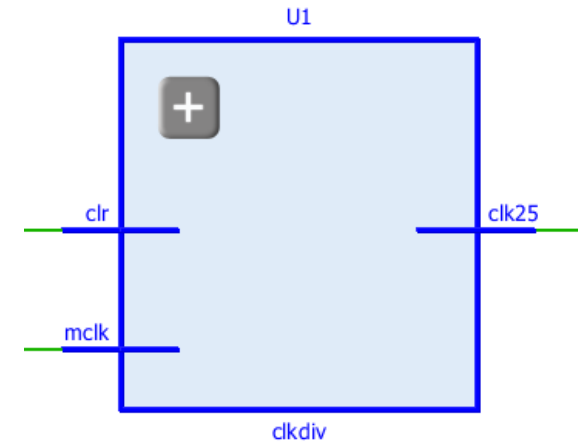
Basys3: VGA port

We need a component for:

- Build a 25MHz clock signal

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;

entity clkdiv is
    port(
        clk : in STD_LOGIC;
        clr : in STD_LOGIC;
        clk25 : out STD_LOGIC
    );
end clkdiv;
```



Basys3: VGA port

We need a component for:

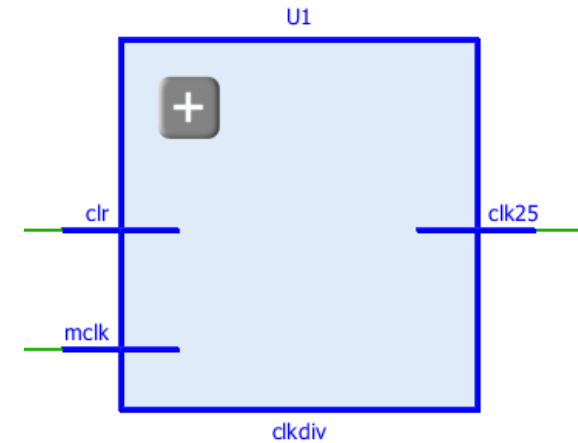
- Build a 25MHz clock signal

```
architecture div of clkdiv is
  signal q:STD_LOGIC_VECTOR(3 downto 0);
begin
```

```
  process(clk, clr)
  begin
    if clr = '1' then
      q <= "0000";
    elsif rising_edge (clk) then
      q <= q + 1;
    end if;
  end process;
```

```
  clk25 <= q(1);    -- 25 MHz
```

```
end div;
```



- `clk50 <= q(0);` -- 50 MHz
- `clk25 <= q(1);` -- 25 MHz
- `Clk12.5 <= q(2);` -- 12.5 MHz

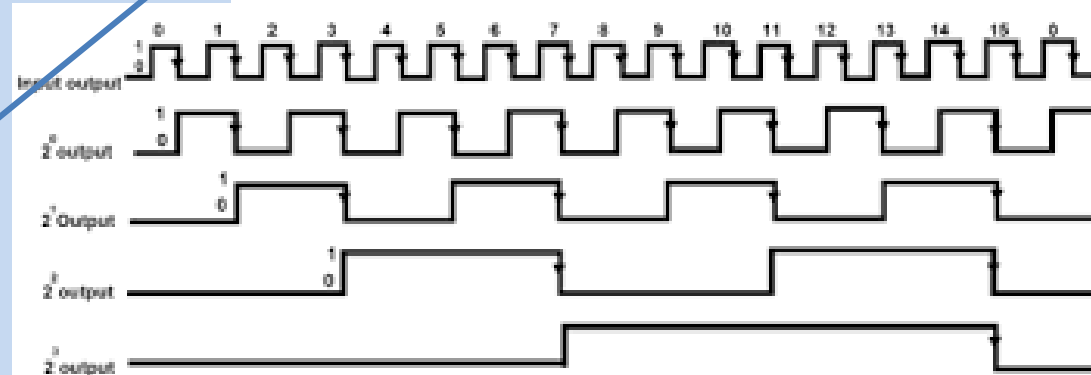


Fig 4: Waveforms of four-stage count-up counter

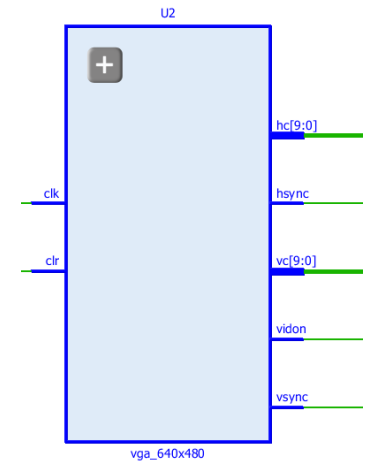
Basys3: VGA port

We need a component for:

- Synchronization

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;

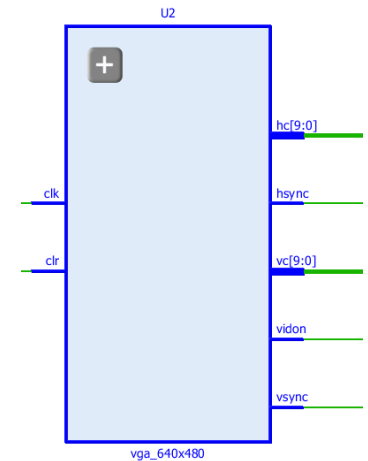
entity vga_640x480 is
    port ( clk, clr : in std_logic;
          hsync : out std_logic;
          vsync : out std_logic;
          hc : out integer range 0 to 800;
          vc : out integer range 0 to 800;
          vidon : out std_logic
        );
end vga_640x480;
```



Basys3: VGA port

- We need a component for:
- Synchronization

architecture behavior of vga_640x480 is
 constant hpixels: integer := 800;
 constant vlines: integer := 521;
 constant hbp: integer := 144;
 constant hfp: integer := 784;
 constant vbp: integer := 31;
 constant vfp: integer := 511;
 signal hcs, vcs: integer range 0 to 800;
 signal vsenable: std_logic;



Value of pixels in a horizontal line = 800
 Number of horizontal lines in the display = 52
 Horizontal back porch = 144 (96+48)
 Horizontal front porch = 784 (96+48+640)
 Vertical back porch = 31 (2+29)
 Vertical front porch = 511 (2+29+480)
 These are the Horizontal and Vertical counters
 Enable for the Vertical counter

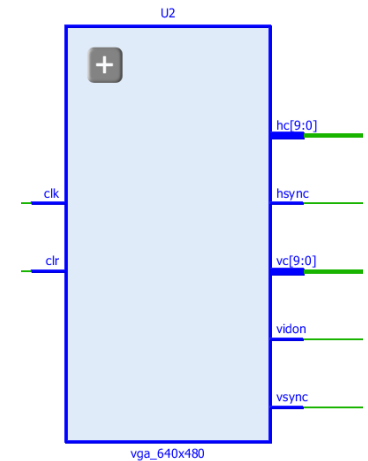
Basys3: VGA port

We need a component for:

- Synchronization

```
begin
    process(clk, clr)
    begin
        if clr = '1' then
            hcs <= 0;
        elsif (clk'event and clk = '1') then
            if hcs = hpixels - 1 then
                hcs <= 0;
                vsenable <= '1';
            else
                hcs <= hcs + 1;
                vsenable <= '0';
            end if;
        end if;
    end process;

    hsync <= '0' when hcs < 96 else '1';
```



- Count horizontal pulses of 25MHz clock
- Generate Hsync

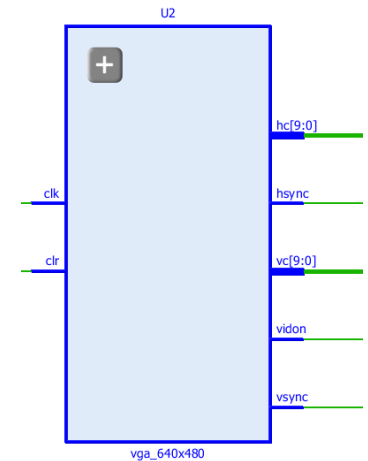
Basys3: VGA port

We need a component for:

- Synchronization

```
process(clk, clr, vsenable)
begin
  if clr = '1' then
    vcs <= 0;
  elsif(clk'event and clk = '1' and vsenable='1') then
    if vcs = vlines - 1 then
      vcs <= 0;
    else
      vcs <= vcs + 1;
    end if;
  end if;
end process;

vsync <= '0' when vcs < 2 else '1';
```



- Count vertical lines
- Generate Vsync

Basys3: VGA port

We need a component for:

- Synchronization

```
vidon <= '1' when (((hcs < hfp) and (hcs >= hbp))  
                  and ((vcs < vfp) and (vcs >= vbp))) else '0';
```

```
hc <= hcs;
```

```
vc <= vcs;
```

```
end behavior
```

- Enable video in visible area
- Output counters

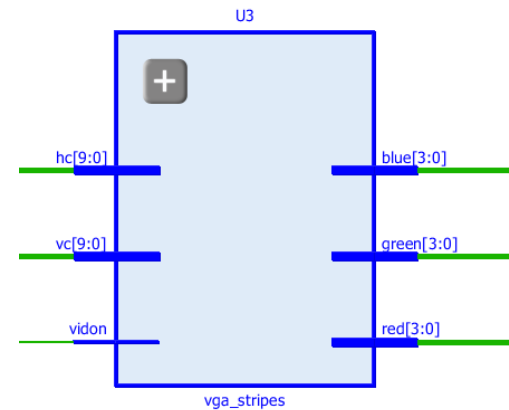
Basys3: VGA port

We need a component for:

- vgaRed / vgaGreen / vgaBlue output

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_arith.all;

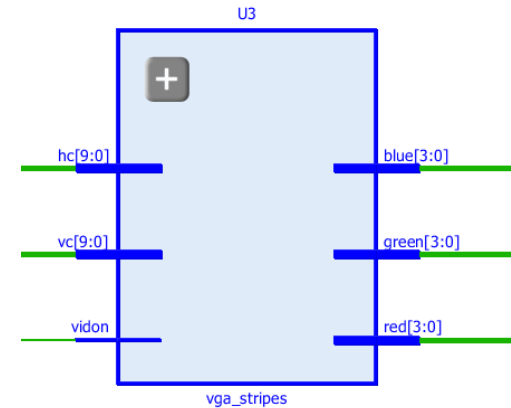
entity vga_strips is
    port (
        vidon: in std_logic;
        hc : in integer range 0 to 800;
        vc : in integer range 0 to 800;
        red : out std_logic_vector(3 downto 0);
        green : out std_logic_vector(3 downto 0);
        blue : out std_logic_vector(3 downto 0));
end vga_strips;
```



Basys3: VGA port

We need a component for:

- vgaRed / vgaGreen / vgaBlue output



architecture behavior of vga_stripes is

```
signal vc_vec: std_logic_vector (9 downto 0);
```

```
Begin
```

```
vc_vec <= conv_std_logic_vector(vc,10);
```

```
process(vidon, vc_vec)
```

```
begin
```

```
    red <= "0000";
```

```
    green <= "0000";
```

```
    blue <= "0000";
```

```
    if vidon = '1' then
```

```
        red <= vc_vec(5) & vc_vec(5) & vc_vec(5) & vc_vec(5);
```

```
        green <= not (vc_vec(5) & vc_vec(5) & vc_vec(5) & vc_vec(5));
```

```
    end if;
```

```
end process;
```

```
end behavior;
```

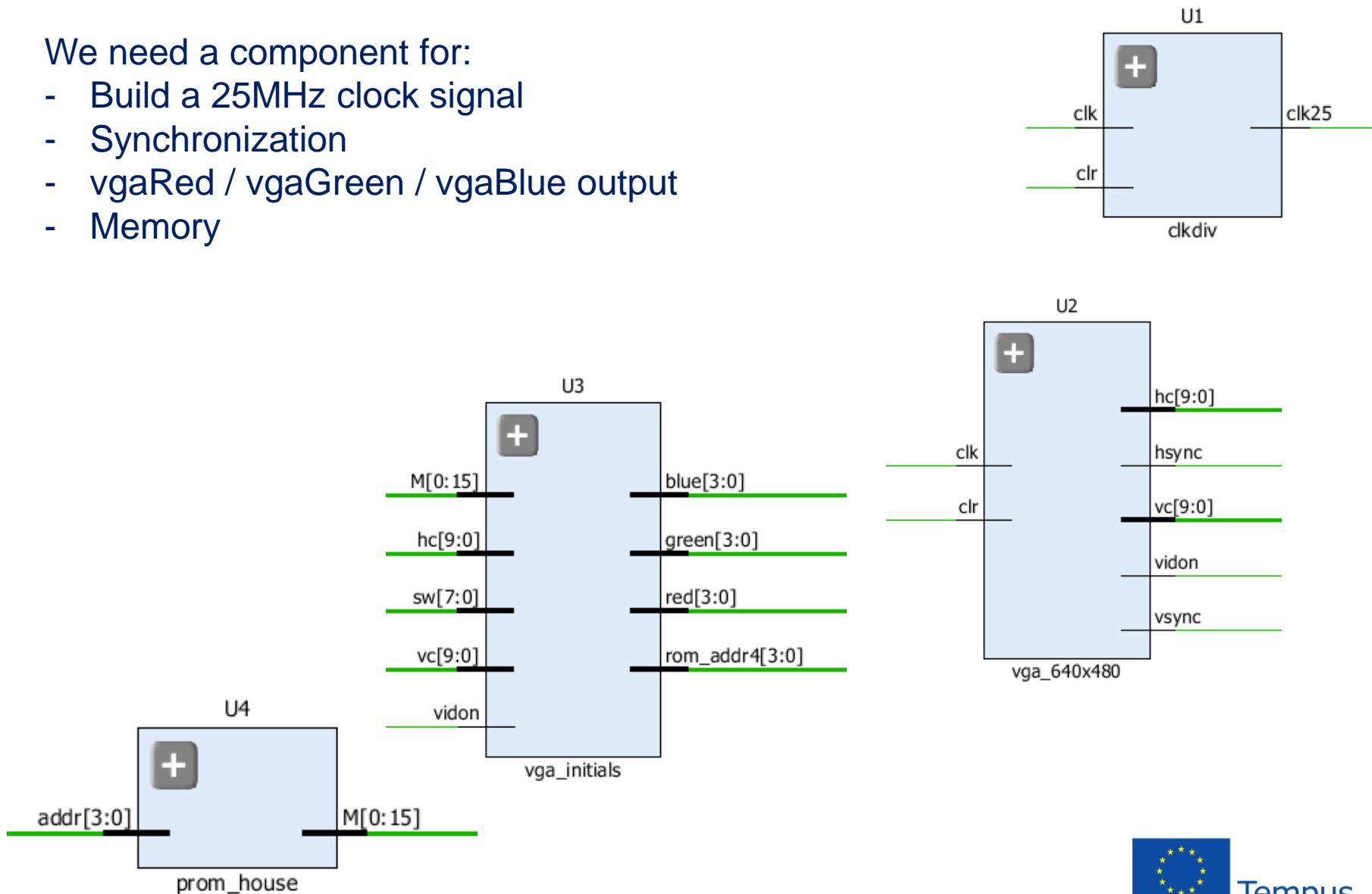
- convert to binary vector
- red when vc_vec (5) is '1', green when not: horizontal stripes

Basys3: VGA port with sprite from memory



We need a component for:

- Build a 25MHz clock signal
- Synchronization
- vgaRed / vgaGreen / vgaBlue output
- Memory



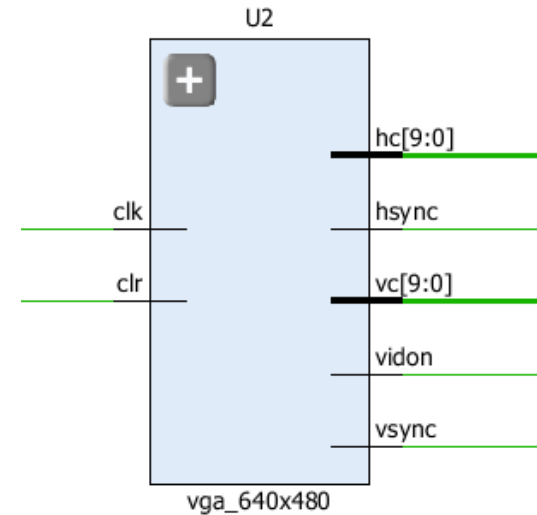
Basys3: VGA port with sprite

We need a component for:

- Synchronization

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

entity vga_640x480 is
    port ( clk, clr : in std_logic;
          hsync : out std_logic;
          vsync : out std_logic;
          hc : out std_logic_vector(9 downto 0);
          vc : out std_logic_vector(9 downto 0);
          vidon : out std_logic
        );
end vga_640x480;
```



Basys3: VGA port with sprite

We need a component for:

- Synchronization

```
architecture vga_640x480 of vga_640x480 is
  constant hpixels: std_logic_vector(9 downto 0) := "1100100000";      --800
  constant vlines: std_logic_vector(9 downto 0) := "1000001001";      --521
  constant hbp: std_logic_vector(9 downto 0) := "0010010000";          --144 = 96+48
  constant hfp: std_logic_vector(9 downto 0) := "1100010000";          -- 784 = 96+48+640
  constant vbp: std_logic_vector(9 downto 0) := "0000011111";          --31 = 29 + 2
  constant vfp: std_logic_vector(9 downto 0) := "0111111111";          --511 = 2+29+480
  signal hcs, vcs: std_logic_vector(9 downto 0);
  signal vsenable: std_logic;
```


Basys3: VGA port with sprite

We need a component for:

- Synchronization

```
begin
    process(clk, clr)
    begin
        if clr = '1' then
            hcs <= "0000000000";
        elsif (clk'event and clk = '1') then
            if hcs = hpixels - 1 then
                hcs <= "0000000000";
                vsenable <= '1';    --Enable the vertical counter
            else
                hcs <= hcs + 1;
                vsenable <= '0';
            end if;
        end if;
    end process;

    hsync <= '0' when hcs < 96 else '1';
```

Basys3: VGA port with sprite

We need a component for:

- Synchronization

```
process(clk, clr, vsenable)
begin
  if clr = '1' then
    vcs <= "0000000000";
  elsif(clk'event and clk = '1' and vsenable='1') then
    if vcs = vlines - 1 then
      vcs <= "0000000000";
    else
      vcs <= vcs + 1;
    end if;
  end if;
end process;

vsync <= '0' when vcs < 2 else '1';
```

Basys3: VGA port with sprite

We need a component for:

- Synchronization

```
--Enable video out when within the porches  
vidon <= '1' when (((hcs < hfp) and (hcs >= hbp))  
    and ((vcs < vfp) and (vcs >= vbp))) else '0';
```

```
-- output horizontal and vertical counters  
hc <= hcs;  
vc <= vcs;
```

```
end vga_640x480;
```

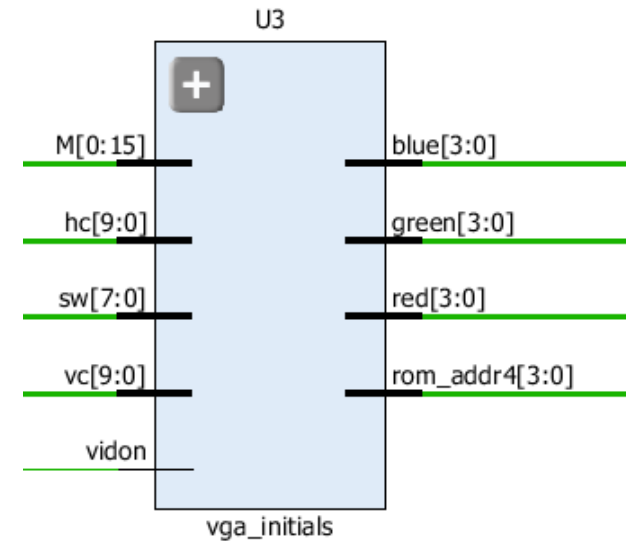
Basys3: VGA port with sprite

We need a component for:

- vgaRed / vgaGreen / vgaBlue output

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

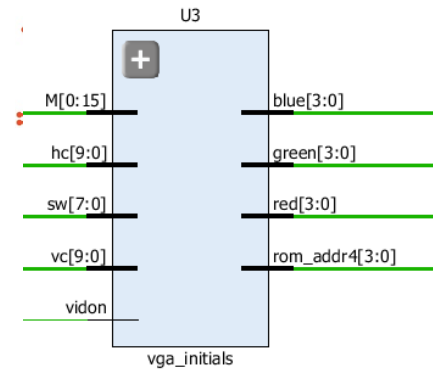
entity vga_sprite is
    port ( vidon: in std_logic;
          hc : in std_logic_vector(9 downto 0);
          vc : in std_logic_vector(9 downto 0);
          M: in std_logic_vector(0 to 15);
          sw: in std_logic_vector(7 downto 0);
          rom_addr4: out std_logic_vector(3 downto 0);
          red : out std_logic_vector(3 downto 0);
          green : out std_logic_vector(3 downto 0);
          blue : out std_logic_vector(3 downto 0)
        );
end vga_sprite;
```



Basys3: VGA port with sprite

We need a component for:

- vgaRed / vgaGreen / vgaBlue output



architecture behavior of vga_sprite is

```
constant hbp: std_logic_vector(9 downto 0) := "0010010000"; --144
```

```
constant vbp: std_logic_vector(9 downto 0) := "0000011111"; --31
```

```
constant w: integer := 16;
```

```
constant h: integer := 16;
```

```
signal C1, R1: std_logic_vector(10 downto 0);
```

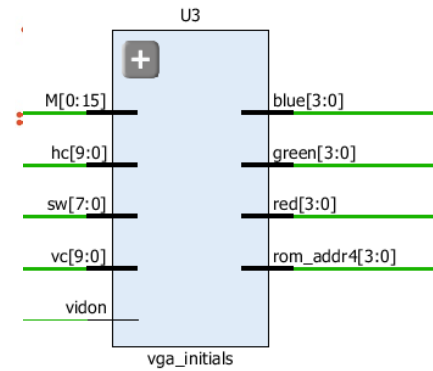
```
signal rom_addr, rom_pix: std_logic_vector(10 downto 0);
```

```
signal spriteon, R, G, B: std_logic;
```

Basys3: VGA port with sprite

We need a component for:

- vgaRed / vgaGreen / vgaBlue output



begin

--set C1 and R1 using switches

C1 <= "00" & SW(3 downto 0) & "00001";

R1 <= "00" & SW(7 downto 4) & "00001";

rom_addr <= vc - vbp - R1;

rom_pix <= hc - hbp - C1;

rom_addr4 <= rom_addr(3 downto 0);

- ROM lijnen
- ROM pixels

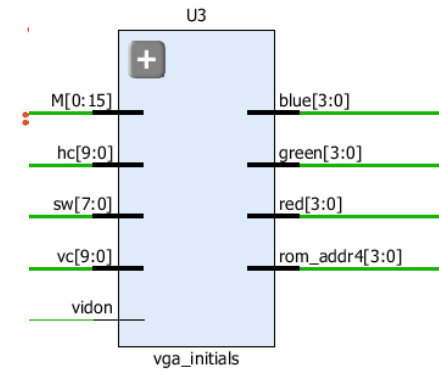
--Enable sprite video out when within the sprite region

spriteon <= '1' when (((hc >= C1 + hbp) and (hc < C1 + hbp + w))
and ((vc >= R1 + vbp) and (vc < R1 + vbp + h))) else '0';

Basys3: VGA port with sprite

We need a component for:

- vgaRed / vgaGreen / vgaBlue output



```
process(spriteon, vidon, rom_pix, M)
variable j: integer;
begin
    red <= "0000";
    green <= "0000";
    blue <= "0000";
    if spriteon = '1' and vidon = '1' then
        j := conv_integer(rom_pix);
        R <= M(j);
        G <= M(j);
        B <= M(j);
        red <= R & R & R & R;
        green <= G & G & G & G;
        blue <= B & B & B & B;
    end if;
end process;
end behavior;
```

vidon: visible area
spriteon: sprite area
⇒ look for memory line
and pixel
⇒ If '1' all on: white

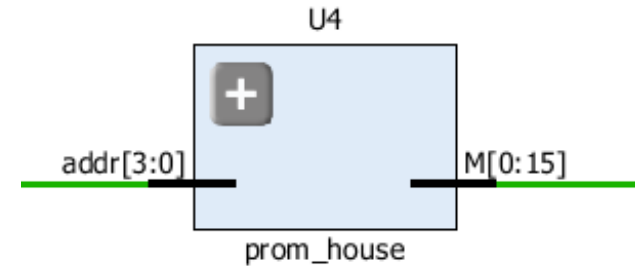
Basys3: VGA port with sprite

We need a component for:

- Memory

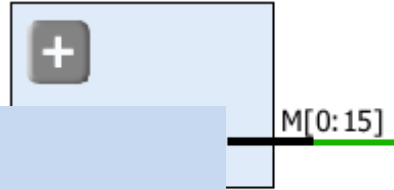
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity prom_house is
  port (
    addr: in STD_LOGIC_VECTOR (3 downto 0);
    M: out STD_LOGIC_VECTOR (0 to 15)
  );
end prom_house;
```



Basys3: VGA port with sprite

```
architecture prom of prom_house is
type rom_array is array (NATURAL range <>)
of STD_LOGIC_VECTOR (0 to 15);
constant rom: rom_array := (
"000000001100000000",
"0100001111000010",
"0000011111100000",
"0000111111110000",
"0001111111111000",
"0011111111111100",
"0111111111111110",
"1111111111111111",
"1100000000000011",
"1100000001110011",
"1100000001110011",
"1100000001110011",
"1100111000000011",
"1100111000000011",
"1100111000000011",
"1100111000000011"
);
```

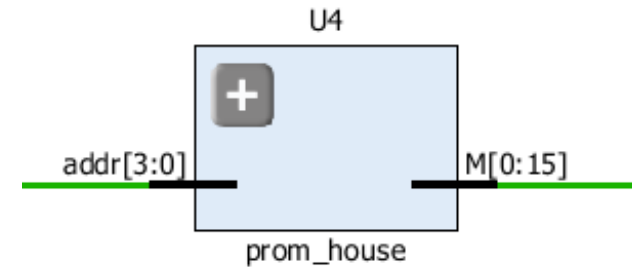


Basys3: VGA port with sprite

We need a component for:

- Memory

```
begin
  process(addr)
    variable j: integer;
    begin
      j := conv_integer(addr);
      M <= rom(j);
    end process;
  end prom;
```

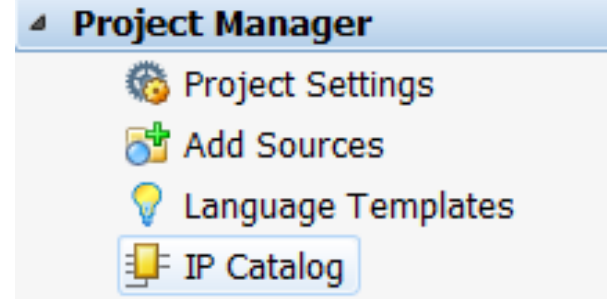


Find line

Basys3: VGA port Block RAM IP

We need a component for:

- Memory

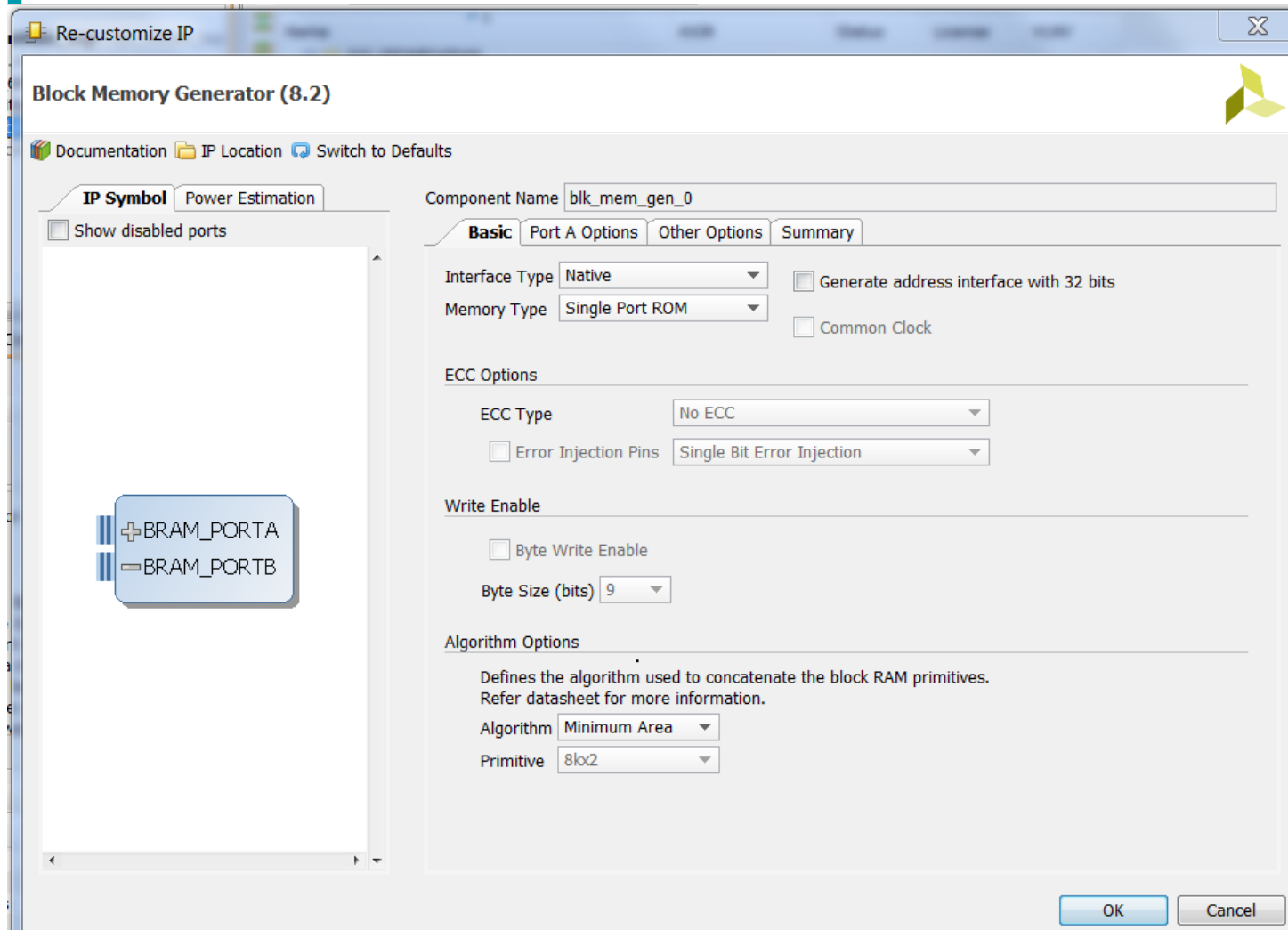


Name	AXI4	Status	License	VLNV
AXI Infrastructure				
BaseIP				
Basic Elements				
Accumulators				
Concat		Production	Included	xilinx.com...
Constant		Production	Included	xilinx.com...
Counters				
DSP48 Macro		Production	Included	xilinx.com...
Memory Elements				
Block Memory Generator	AXI4	Production	Included	xilinx.com...
Distributed Memory Generator		Production	Included	xilinx.com...
Registers, Shifters & Pipelining				
Slice		Production	Included	xilinx.com...
Communication & Networking				
Debug & Verification				
Digital Signal Processing				
Embedded Processing				
FPGA Features and Design				
Math Functions				
Memories & Storage Elements				
Partial Reconfiguration				
Standard Bus Interfaces				
Video & Image Processing				

Basys3: VGA port Block RAM IP

We need a component for:

- Memory



Basys3: VGA port Block RAM IP

We need a component for:

- Memory

Component Name

Basic **Port A Options** Other Options Summary

Memory Size

Port A Width Range: 1 to 4608 (bits)

Port A Depth Range: 2 to 1048576

The Width and Depth values are used for Read Operation in Port A

Operating Mode Enable Port Type

Port A Optional Output Registers

☒ Primitives Output Register ☐ Core Output Register

☐ SoftECC Input Register ☐ REGCEA Pin

Port A Output Reset Options

☐ RSTA Pin (set/reset pin) Output Reset Value (Hex)

☐ Reset Memory Latch Reset Priority

READ Address Change A

☐ Read Address Change A

Basys3: VGA port Block RAM IP

We need a component for:

- Memory

Component Name

Basic Port A Options **Other Options** Summary

Pipeline Stages within Mux Mux Size: 1x1

Memory Initialization

☒ Load Init File

Coe File

☐ Fill Remaining Memory Locations

Remaining Memory Locations (Hex)

Structural/UniSim Simulation Model Options

Defines the type of warnings and outputs are generated when a read-write or write-write collision occurs.

Collision Warnings

Behavioral Simulation Model Options

☐ Disable Collision Warnings ☐ Disable Out of Range Warnings

Basys3: VGA port Block RAM IP

We need a component for:

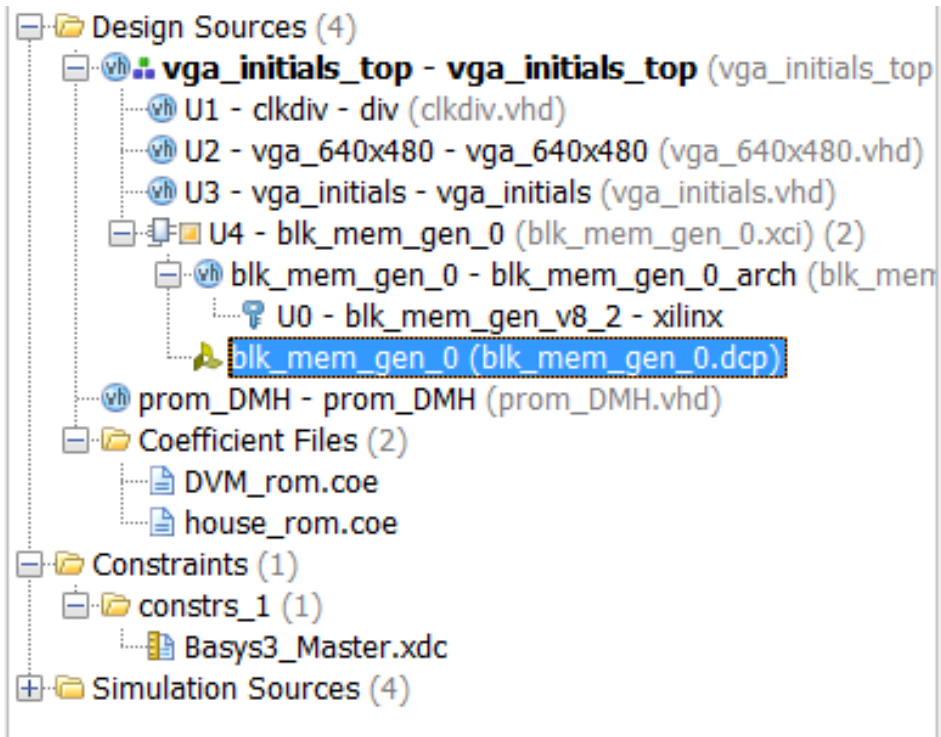
- Memory: COE

```
memory_initialization_radix = 2;
memory_initialization_vector =
0000000110000000
0100001111000010
0000011111100000
0000111111110000
0001111111111000
0011111111111100
0011111111111100
0111111111111110
1111111111111111
1100000000000011
1100000001110011
1100000001110011
1100000001110011
1100111000000011
1100111000000011
1100111000000011
1100111000000011
;
```

Basys3: VGA port Block RAM IP

We need a component for:

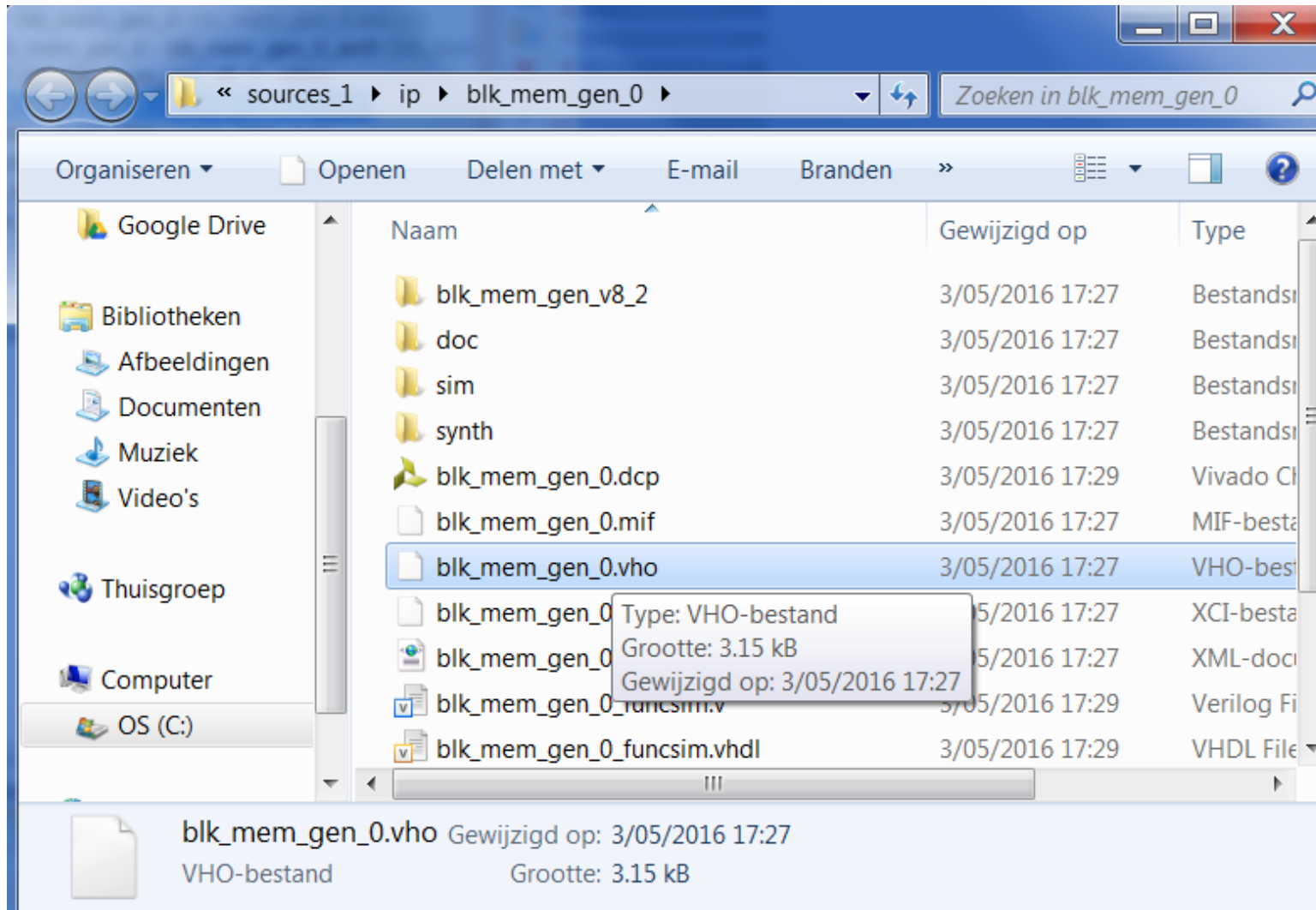
- Memory



Basys3: VGA port Block RAM IP

We need a component for:

- Memory



Basys3: VGA port Block RAM IP

We need a component for:

- Memory

```
COMPONENT blk_mem_gen_0 PORT (  
    clka : IN STD_LOGIC;  
    addra : IN STD_LOGIC_VECTOR(3 DOWNT0 0);    douta : OUT  
STD_LOGIC_VECTOR(15 DOWNT0 0) );  
END COMPONENT;
```

```
your_instance_name : blk_mem_gen_0  
PORT MAP (  clka => clka,  addra => addra,  douta => douta );
```

Basys3: VGA port Block RAM IP

We need a component for:

- Memory





```
U4 : blk_mem_gen_0  
  PORT MAP (  
    clka => clk,  
    addra => rom_addr4,  
    douta => M  
  );
```

Basys3: VGA port clock with IP

We need a component for:

- Build a 25MHz clock signal

Project Manager

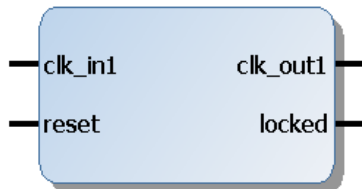
-  Project Settings
-  Add Sources
-  Language Templates
-  IP Catalog

Name	AXI4	Status	License	VLNV
<div> <div>1</div> <div> <div>Vivado Repository</div> <div> <div>Alliance Partners</div> <div>Automotive & Industrial</div> <div>AXI Infrastructure</div> <div>BaseIP</div> <div>Basic Elements</div> <div>Communication & Networking</div> <div>Debug & Verification</div> <div>Digital Signal Processing</div> <div>Embedded Processing</div> <div>FPGA Features and Design</div> <div>Clocking</div> <div>Clocking Wizard</div> <div>IO Interfaces</div> <div>Soft Error Mitigation</div> <div>XADC</div> <div>Math Functions</div> <div>Memories & Storage Elements</div> <div>Partial Reconfiguration</div> <div>Standard Bus Interfaces</div> <div>Video & Image Processing</div> </div> </div> </div>	AXI4	Production	Included	xilinx.com...

Basys3: VGA port clock with IP

We need a component for:

- Build a 25MHz clock signal



Clocking Options

Primitive: ☒ MMCM ☐ PLL

Cloning Features:

- ☒ Frequency Synthesis
- ☒ Phase Alignment
- ☐ Dynamic Reconfig
- ☐ Safe Clock Startup
- ☐ Minimize Power
- ☐ Spread Spectrum
- ☐ Dynamic Phase Shift

Jitter Optimization:

- ☒ Balanced
- ☐ Minimize
- ☐ Maximize

Dynamic Reconfig Interface Options:

- ☒ AXI4Lite
- ☐ DRP
- ☐ Phase Duty Cycle Config

Input Clock Information:

	Input Clock	Input Frequency(MHz)	Time Period (ns)		Jitter Options	Input Jitter	Source
<input type="checkbox"/>	Primary	100.000	10.000	10,000 - 800,000	UI	0.010	Single ended clock capab
<input type="checkbox"/>	Secondary	100.000	10.000	150,000 - 300,000		0.010	Single ended clock capab

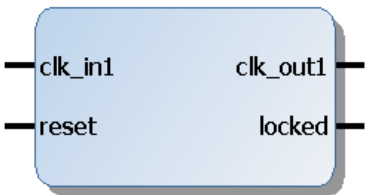
Balanced : selecting this option allows the software choosing the correct bandwidth for jitter optimization Minimize output Jitter Selecting this option minimizes the jitter on the output clocks at the expense of power and output clock phase error Maximize input jitter filtering : Selecting this option allows larger input jitter on the input clocks

Basys3: VGA port clock with IP

We need a component for:

- Build a 25MHz clock signal

Now disabled ports



Clocking Options **Output Clocks** MMCM Settings Port Renaming Summary

The phase is calculated relative to the active input clock.

Output Clock	Output Freq (MHz)		Phase (degrees)		Duty Cycle (%)		Drives
	Requested	Actual	Requested	Actual	Requested	Actual	
<input checked="" type="checkbox"/> clk_out1	25.000	XXX	0.00	0.000	50.000	50.000	BUFG
<input type="checkbox"/> clk_out2	100.000	N/A	0.000	N/A	50.000	N/A	BUFG
<input type="checkbox"/> clk_out3	100.000	N/A	0.000	N/A	50.000	N/A	BUFG
<input type="checkbox"/> clk_out4	100.000	N/A	0.000	N/A	50.000	N/A	BUFG
<input type="checkbox"/> clk_out5	100.000	N/A	0.000	N/A	50.000	N/A	BUFG
<input type="checkbox"/> clk_out6	100.000	N/A	0.000	N/A	50.000	N/A	BUFG
<input type="checkbox"/> clk_out7	100.000	N/A	0.000	N/A	50.000	N/A	BUFG

Actual phase for the output clock5

Could not obtain valid VCO, M and D values with the given input freqs and the requested frequencies.

☐ USE CLOCK SEQUENCING

Clocking Feedback

Output Clock	Sequence Number
clk_out1	1
clk_out2	1
clk_out3	1

Source Signaling

☒ Automatic Control On-Chip
 ☐ Automatic Control Off-Chip
 ☐ User Controlled On-Chip
 ☒ Single-ended
 ☐ Differential

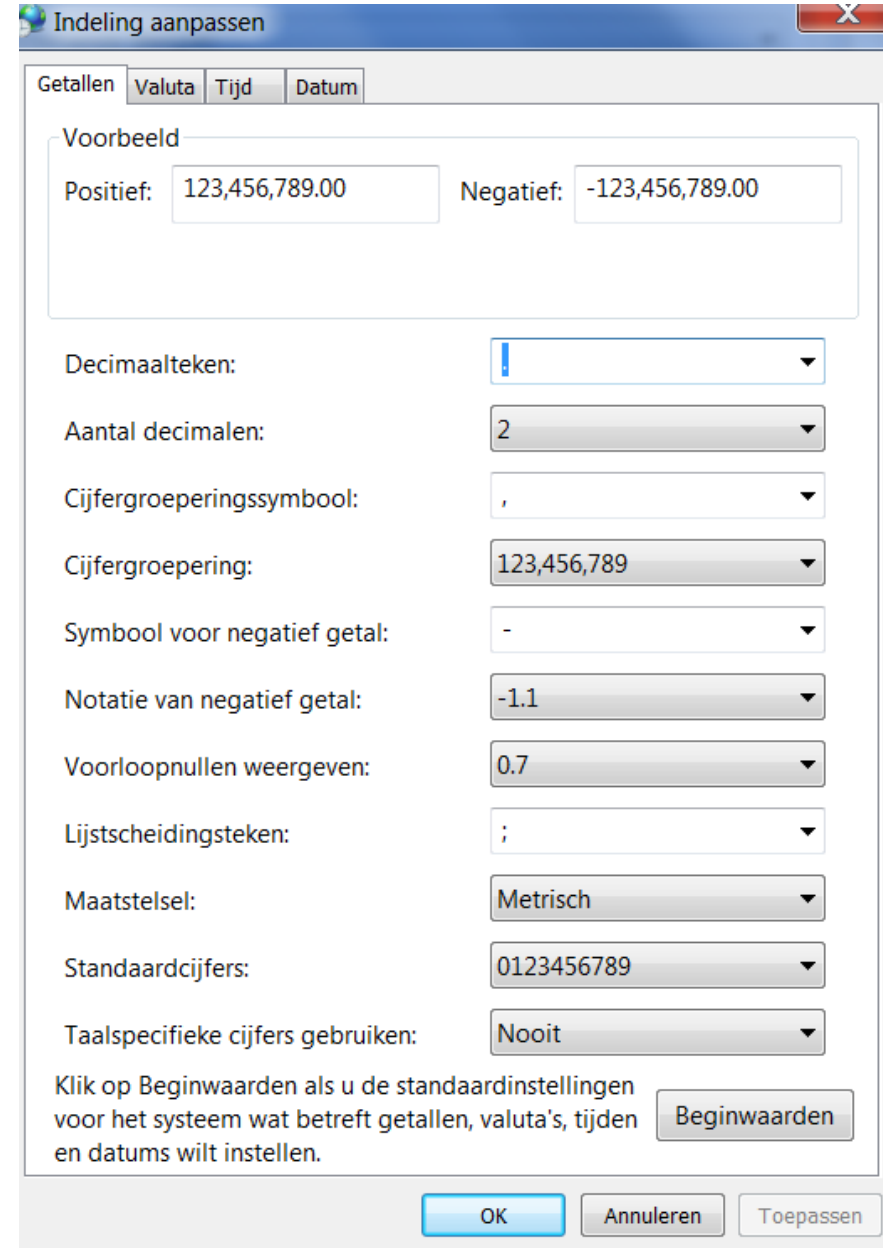
Basys3: VGA port clock with IP

We need a component for:

- Build a 25MHz clock signal

Attention: language settings!!!

=> Configuration: country and language



Indeling aanpassen

Getallen Valuta Tijd Datum

Voorbeeld

Positief: 123,456,789.00 Negatief: -123,456,789.00

Decimaalteken:

Aantal decimalen:

Cijfergroeperingssymbool:

Cijfergroepering:

Symbool voor negatief getal:

Notatie van negatief getal:

Voorloopnullen weergeven:

Lijstscheidingsteken:

Maatstelsel:

Standaardcijfers:

Taalspecifieke cijfers gebruiken:

Klik op Beginwaarden als u de standaardinstellingen voor het systeem wat betreft getallen, valuta's, tijden en datums wilt instellen.

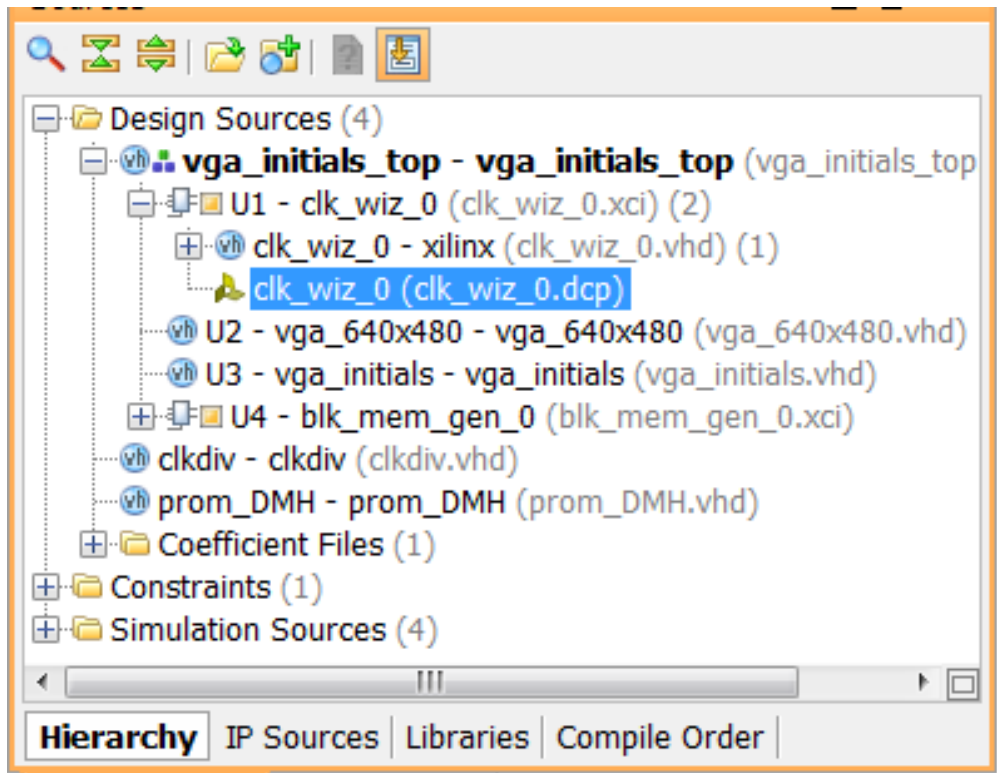
Beginwaarden

OK Annuleren Toepassen

Basys3: VGA port clock with IP

We need a component for:

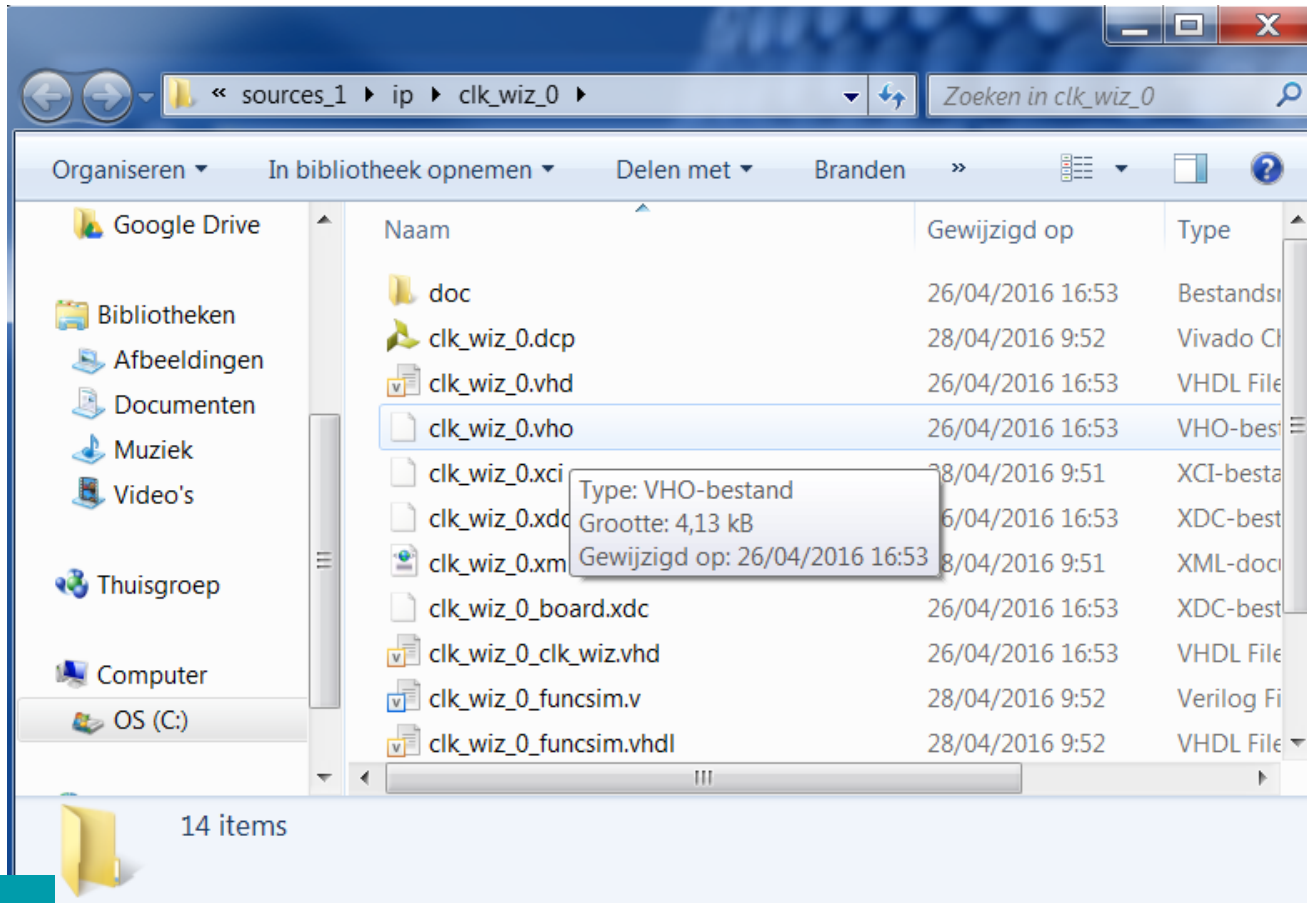
- Build a 25MHz clock signal



Basys3: VGA port clock with IP

We need a component for:

- Build a 25MHz clock signal



Basys3: VGA port clock with IP

We need a component for:

- Build a 25MHz clock signal

```
component clk_wiz_0
port
(-- Clock in ports
clk_in1      : in  std_logic;
-- Clock out ports
clk_out1     : out  std_logic;
-- Status and control signals
reset       : in  std_logic;
locked      : out  std_logic );
end component;

ATTRIBUTE SYN_BLACK_BOX : BOOLEAN;
ATTRIBUTE SYN_BLACK_BOX OF clk_wiz_0 : COMPONENT IS TRUE;

ATTRIBUTE BLACK_BOX_PAD_PIN : STRING;
ATTRIBUTE BLACK_BOX_PAD_PIN OF clk_wiz_0 : COMPONENT IS
"clk_in1,clk_out1,reset,locked";
```

Basys3: VGA port clock with IP

We need a component for:

- Build a 25MHz clock signal

```
your_instance_name : clk_wiz_0
port map (
-- Clock in ports
clk_in1 => clk_in1,
-- Clock out ports
clk_out1 => clk_out1,
-- Status and control signals
reset => reset,
locked => locked
);
```

Basys3: VGA port clock with IP

We need a component for:

- Build a 25MHz clock signal

```
U1 : clk_wiz_0
  port map (

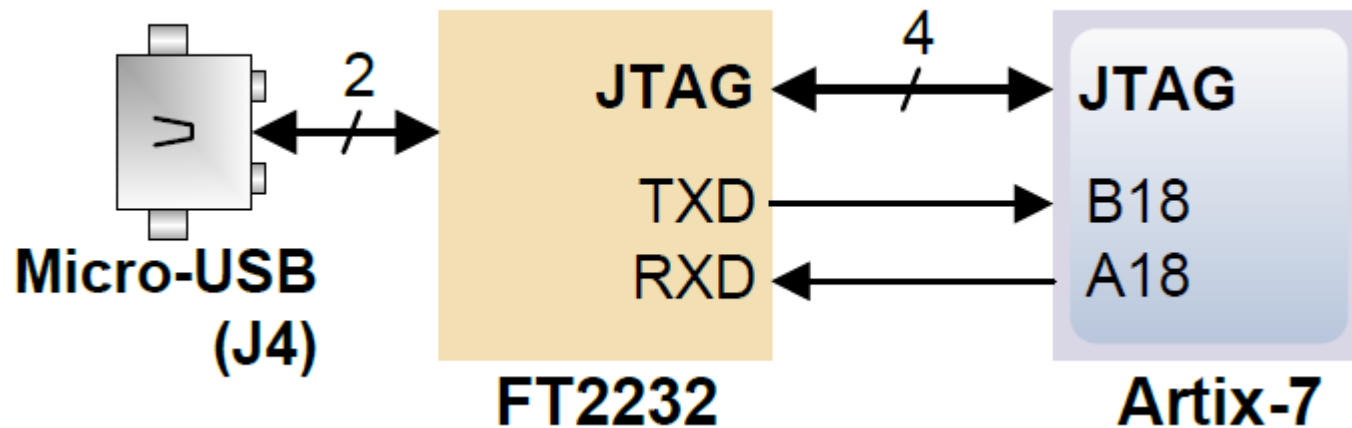
    -- Clock in ports
    clk_in1 => clk,
    -- Clock out ports
    clk_out1 => clk25,
    -- Status and control signals
    reset => clr,
    locked => open
  );
```

Basys3: Oscillators/Clocks

- The Basys3 board includes a single 100 MHz oscillator connected to pin W5
- The input clock can drive MMCMs or PLLs to generate clocks of various frequencies and with known phase relationships.

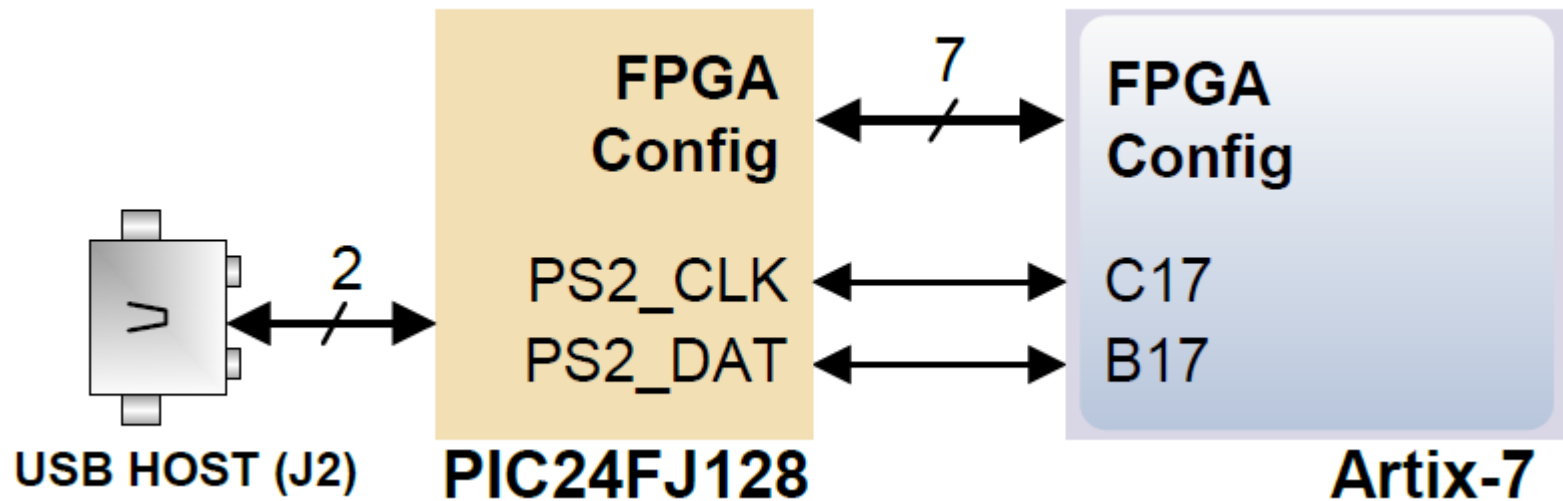
Basys3: USB-UART Bridge (Serial Port)

- The Basys3 includes an FTDI FT2232HQ USB-UART bridge
- This allows to use PC applications to communicate with the board using standard Windows COM port commands
- Serial port data is exchanged with the FPGA using a two-wire serial port (TXD/RXD)

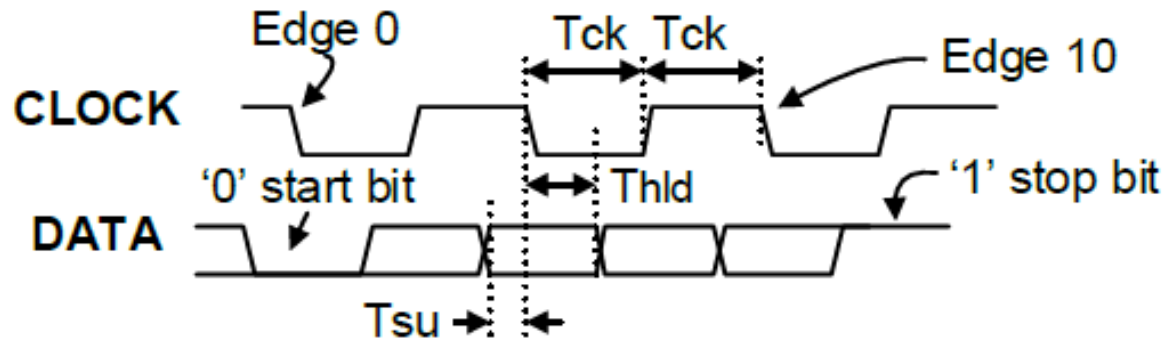


Basys3: USB HID Host

- The Auxiliary Function microcontroller (Microchip PIC24FJ128) provides the Basys3 with USB HID host capability
- The PIC24 drives two signals which are used to implement a standard PS/2 interface for communication with a mouse or keyboard



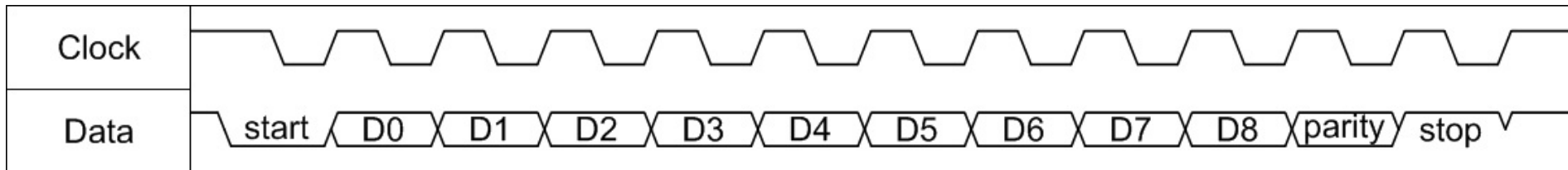
Basys3: PS2



Symbol	Parameter	Min	Max
T_{CK}	Clock time	30us	50us
T_{SU}	Data-to-clock setup time	5us	25us
T_{HLD}	Clock-to-data hold time	5us	25us

Basys3: PS2

- Both clock and data signals are logic level high when inactive
- The keyboard provides both the clock and data
- The data begins with a start bit (logic low), followed by one byte of data, a parity bit, and finally a stop bit (logic high)
- The data is sent LSB first
- Each bit should be read on the falling edge of the clock signal
- Once complete, both the clock and data signals return to logic level high



Basys3: PS2

- Scan Codes

ESC 76	F1 05	F2 06	F3 04	F4 0C	F5 03	F6 0B	F7 83	F8 0A	F9 01	F10 09	F11 78	F12 07	
~ 0E	1! 16	2@ 1E	3# 26	4\$ 25	5% 2E	6^ 36	7& 3D	8* 3E	9(46	0) 45	- 4E	=+ 55	BackSpace ← 66
TAB 0D	Q 15	W 1D	E 24	R 2D	T 2C	Y 35	U 3C	I 43	O 44	P 4D	[{ 54]} 5B	\ 5D
Caps Lock 58	A 1C	S 1B	D 23	F 2B	G 34	H 33	J 3B	K 42	L 4B	:: 4C	'" 52	Enter ↵ 5A	
Shift 12	Z 1Z	X 22	C 21	V 2A	B 32	N 31	M 3A	,< 41	>. 49	/? 4A	↑ 59	Shift 59	
Ctrl 14	Alt 11	Space 29							Alt E0 11	Ctrl E0 14			

Basys3: PS2

- 4 actions
 - Synchronize & Filter
 - Input
 - Output

Basys3: PS2

entity PS2 is

Generic(clk_freq: INTEGER := 100_000_000);

--system clock frequency in Hz

Port (clk : in STD_LOGIC;

clr : in STD_LOGIC;

ps2d : in STD_LOGIC;

ps2c : in STD_LOGIC;

ps2_out_new : out STD_LOGIC;

--flag new PS2 output

ps2_out : out STD_LOGIC_VECTOR (7 downto 0));

--output received from PS2

end PS2;

architecture Behavioral of PS2 is

signal ps2c_int : STD_LOGIC;

--debounced clock signal

signal ps2d_int : STD_LOGIC;

--debounced data signal

signal ps2c_filter: STD_LOGIC_VECTOR(7 DOWNT0 0);

signal ps2d_filter: STD_LOGIC_VECTOR(7 DOWNT0 0);

signal ps2_word : STD_LOGIC_VECTOR(10 DOWNT0 0); --stores the ps2 data word

signal count_idle : INTEGER RANGE 0 TO clk_freq/18_000; --counter to determine idle

Basys3: PS2

- Synchronize & Filter

```
filter: process(clk, clr)
begin
  if clr = '1' then
    ps2c_filter <= (others => '1');
    ps2d_filter <= (others => '1');
```

Basys3: PS2

- Filter

```
elsif rising_edge (clk) then
    ps2c_filter(7) <= ps2c;
    ps2c_filter(6 downto 0) <= ps2c_filter(7 downto 1);
    ps2d_filter(7) <= ps2d;
    ps2d_filter(6 downto 0) <= ps2d_filter(7 downto 1);
    if ps2c_filter = X"FF" then
        ps2c_int <= '1';
    elsif ps2c_filter = X"00" then
        ps2c_int <= '0';
    end if;
    if ps2d_filter = X"FF" then
        ps2d_int <= '1';
    elsif ps2d_filter = X"00" then
        ps2d_int <= '0';
    end if;
end if;
end process filter;
```

Basys3: PS2

- Input

```
input: process(ps2c_int)
begin
  if(ps2c_int'event and ps2c_int = '0') then  --falling edge of PS2 clock
    ps2_word <= ps2d_int & ps2_word(10 downto 1); --shift in PS2 data bit
  end if;
end process;
```

Basys3: PS2

- Output

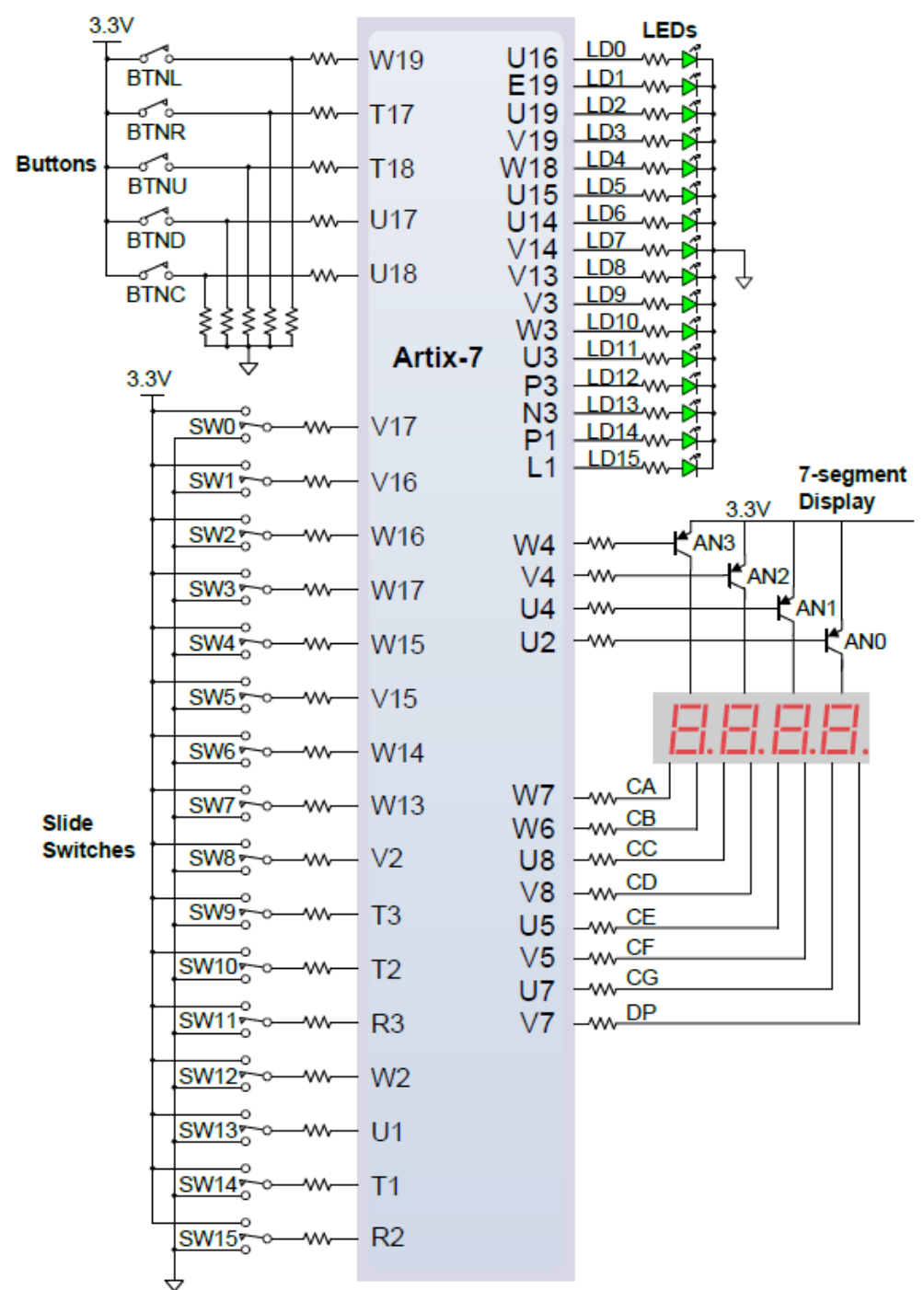
```
output: process(clk)
begin
  if rising_edge (clk) then
    if(ps2c_int = '0') then
      count_idle <= 0;
    elsif(count_idle /= clk_freq/18_000) then

      count_idle <= count_idle + 1;
      ps2_out_new <= '0';
    else
      ps2_out_new <= '1';
      ps2_out <= ps2_word(8 DOWNT0 1);
    end if;
  end if;
end process;
end Behavioral;
```

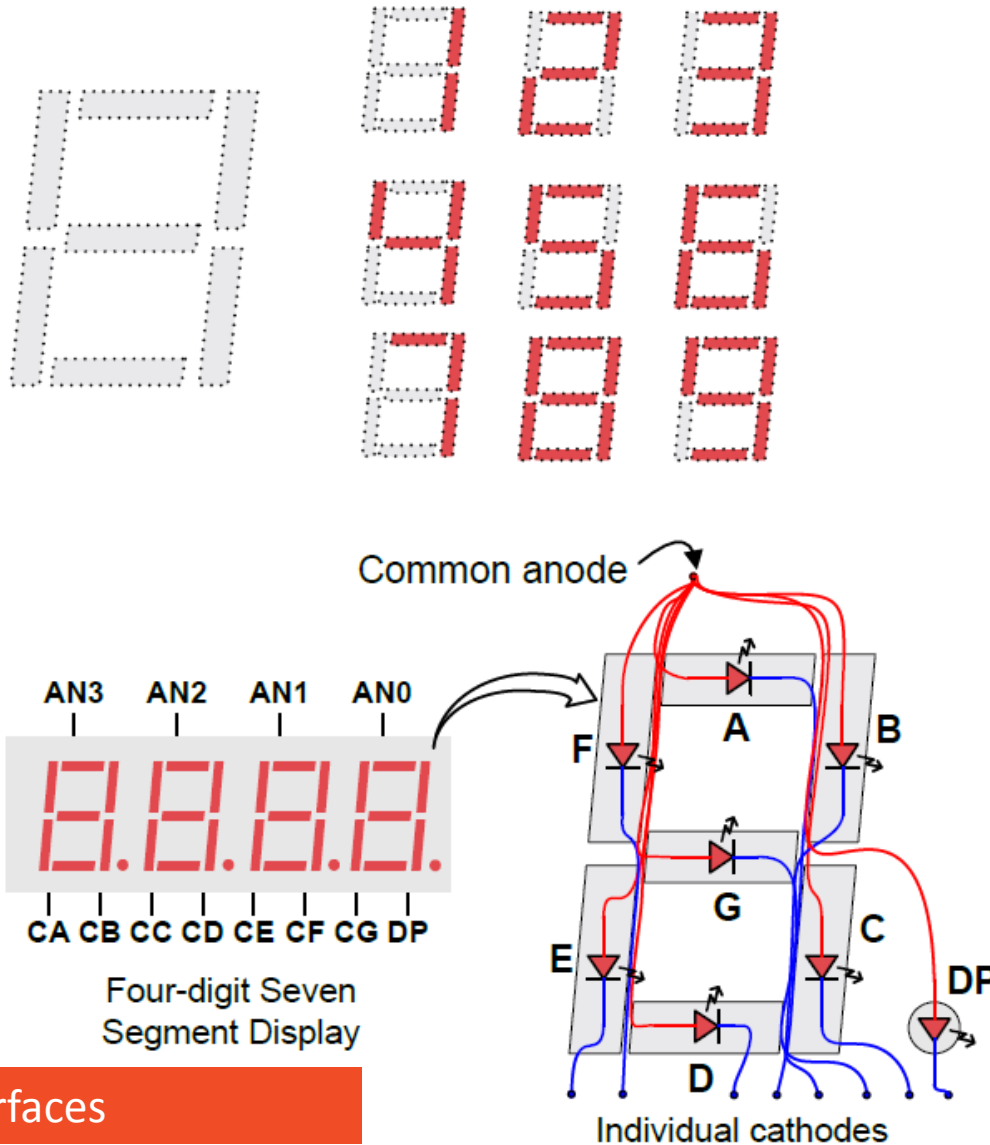
--rising edge of system clock
--low PS2 clock, PS/2 is active
--reset idle counter
--PS2 clock has been high less than a half clock period, max 50us
--continue counting

-- clk_freq/18000 = 55us
--set flag that new PS/2 code is available

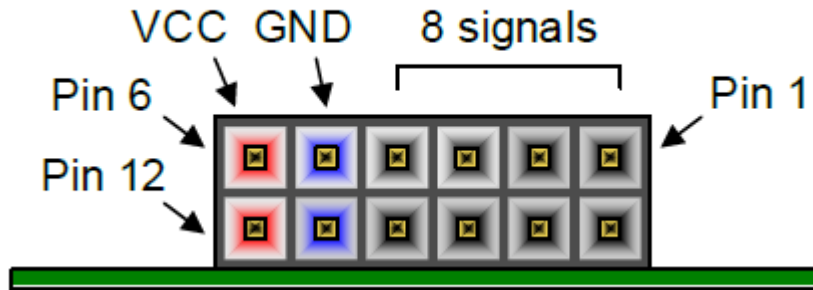
Basys3: Basic I/O



Basys3: Basic I/O

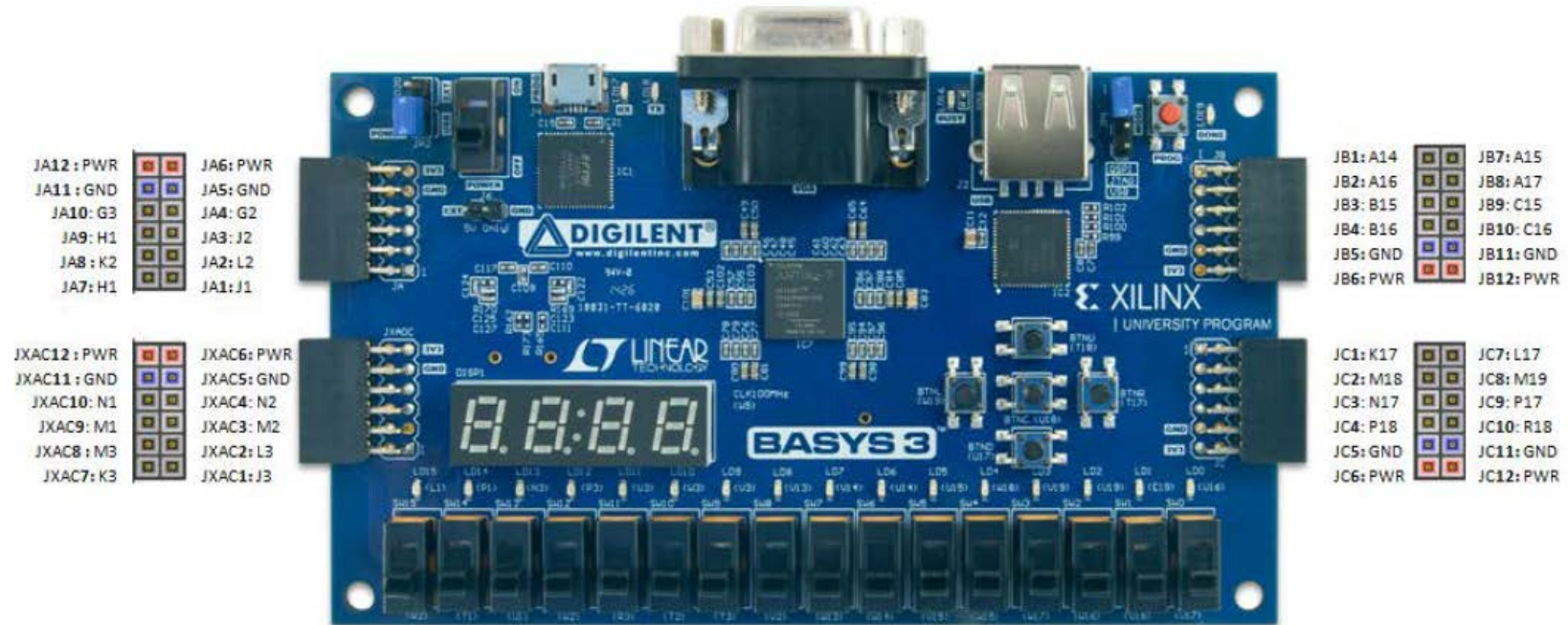


Basys3: Pmod Controllers



Pmod JA	Pmod JB	Pmod JC	Pmod XDAC
JA1: J1	JB1: A14	JC1: K17	JXADC1: J3
JA2: L2	JB2: A16	JC2: M18	JXADC2: L3
JA3: J2	JB3: B15	JC3: N17	JXADC3: M2
JA4: G2	JB4: B16	JC4: P18	JXADC4: N2
JA7: H1	JB7: A15	JC7: L17	JXADC7: K3
JA8: K2	JB8: A17	JC8: M19	JXADC8: M3
JA9: H2	JB9: C15	JC9: P17	JXADC9: M1
JA10: G3	JB10: C16	JC10: R18	JXADC10: N1

Basys3: Pmod Controllers

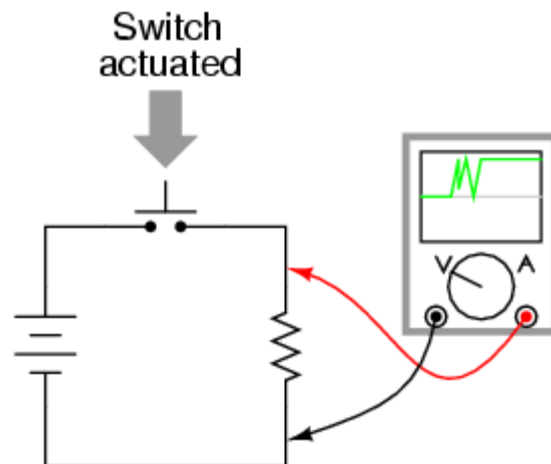


Buttons, rotary encoder

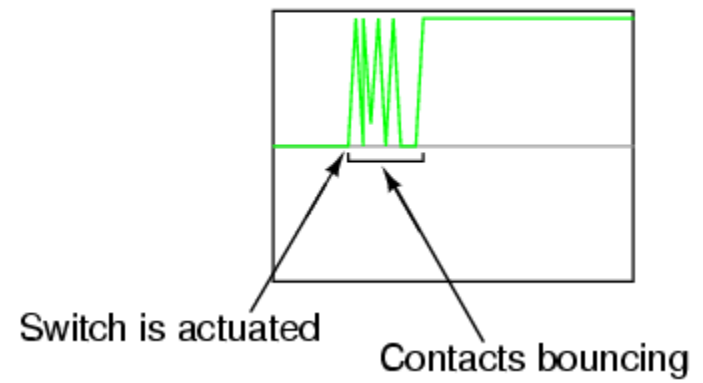
- Debouncing

- Often sequential circuits are depended on signal edges.
- Mechanical switches and buttons are prone to bounce: this produces more than one edge

contacts will “bounce” upon closure for a period of milliseconds before coming to a full rest and providing unbroken contact



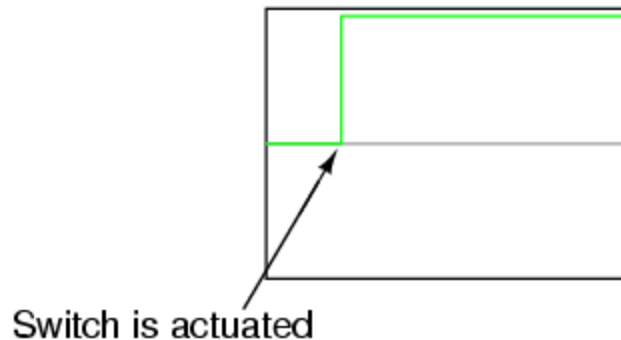
Close-up view of oscilloscope display:



Buttons, rotary encoder

- Debouncing
 - For proper operation we need one clean edge

"Bounceless" switch operation



Buttons, rotary encoder

- Always debounce switches and buttons
- Spartan 3E: push buttons need pull-up constraints
- Rotary Encoder also needs to be debounced

In XDC: `set_property PULLUP true [get_ports <ports>]`

Debouncing:

entity debouncer is

```
Port (    sig_in : in  STD_LOGIC;
         clk : in  STD_LOGIC;
         sig_out : out STD_LOGIC);
```

end debouncer;

architecture debounce of debouncer is

```
signal Q1, Q2, Q3 : std_logic;begin
```

```
process(clk) begin
```

```
if (clk'event and clk = '1') then
```

```
Q1 <= sig_in;
```

```
Q2 <= Q1;
```

```
Q3 <= Q2;
```

```
end if; end process;
```

```
sig_out <= Q1 and Q2 and Q3;
```

```
end debounce;
```


Debouncing: (one shot pulse)

entity debouncer is

```
Port (    sig_in : in  STD_LOGIC;
         clk : in  STD_LOGIC;
         sig_out : out STD_LOGIC);
```

end debouncer;

architecture debounce of debouncer is

```
signal Q1, Q2, Q3 : std_logic;begin
```

```
process(clk) begin
```

```
if (clk'event and clk = '1') then
```

```
Q1 <= sig_in;
```

```
Q2 <= Q1;
```

```
Q3 <= Q2;
```

```
end if; end process;
```

```
sig_out <= Q1 and Q2 and (not Q3);
```

```
end debounce;
```

Debouncing: (active low logic)

entity debouncer is

```
Port (    sig_in : in  STD_LOGIC;
         clk : in  STD_LOGIC;
         sig_out : out STD_LOGIC);
```

end debouncer;

architecture debounce of debouncer is

```
signal Q1, Q2, Q3 : std_logic;begin
```

```
process(clk) begin
```

```
if (clk'event and clk = '1') then
```

```
Q1 <= sig_in;
```

```
Q2 <= Q1;
```

```
Q3 <= Q2;
```

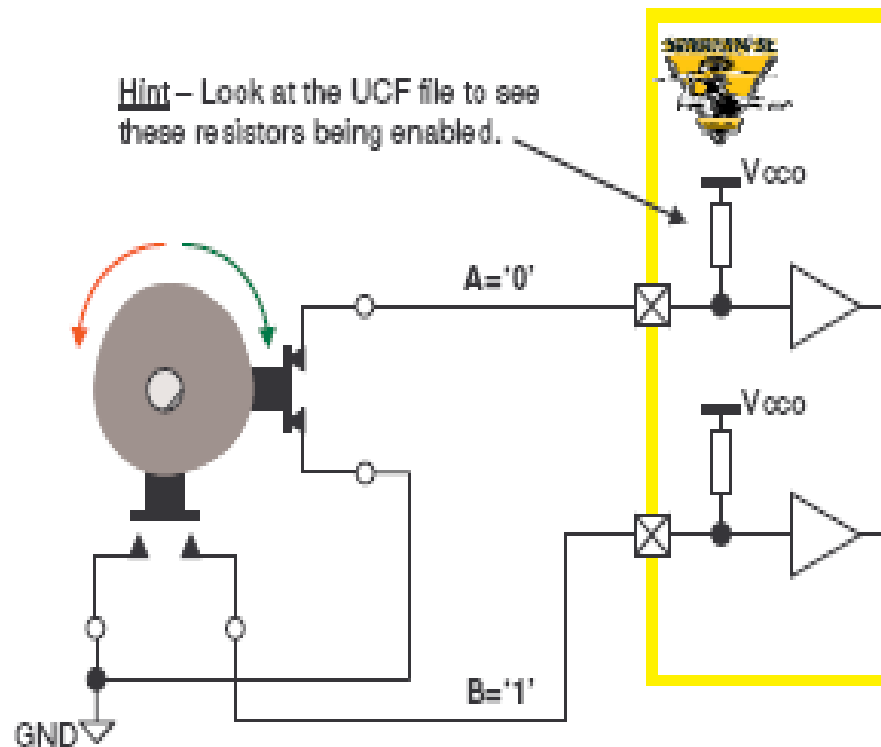
```
end if; end process;
```

```
sig_out <= Q1 or Q2 or Q3;
```

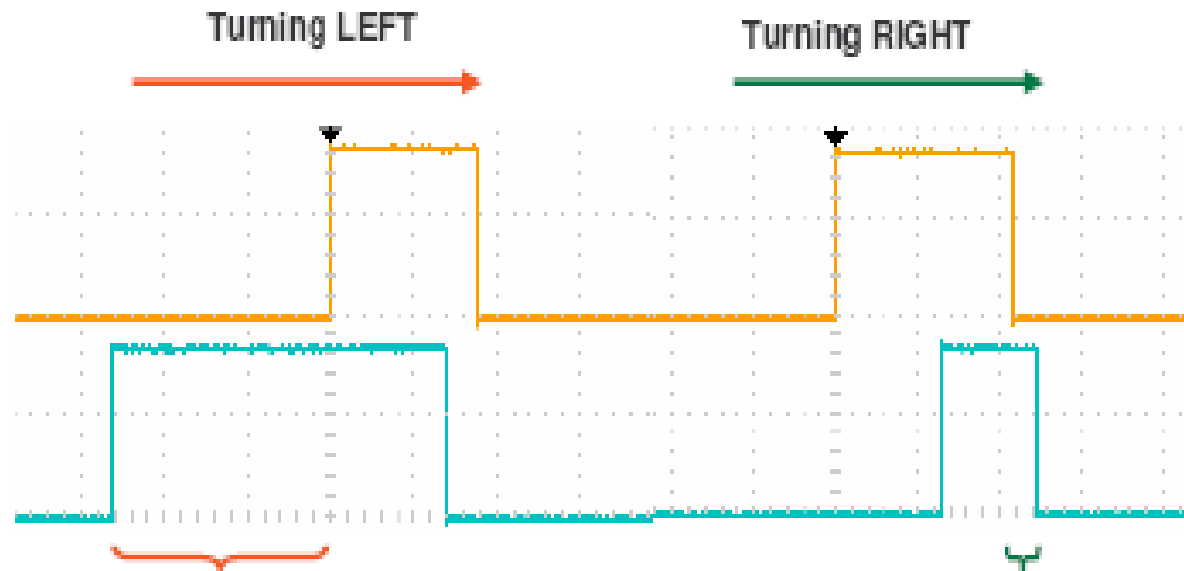
```
end debounce;
```

Rotary encoder:

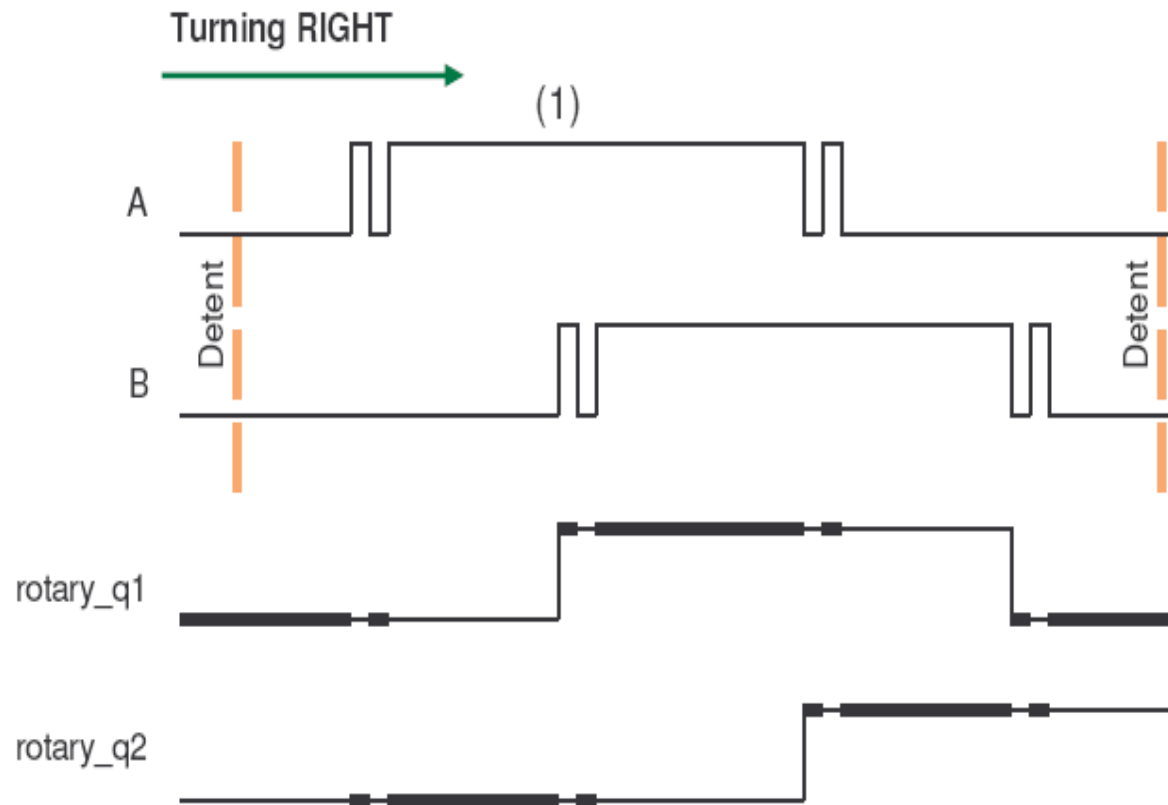
http://www.xilinx.com/products/boards/s3estarter/files/s3esk_rotary_encoder_interface.pdf



Rotary encoder:



Rotary encoder: debouncing



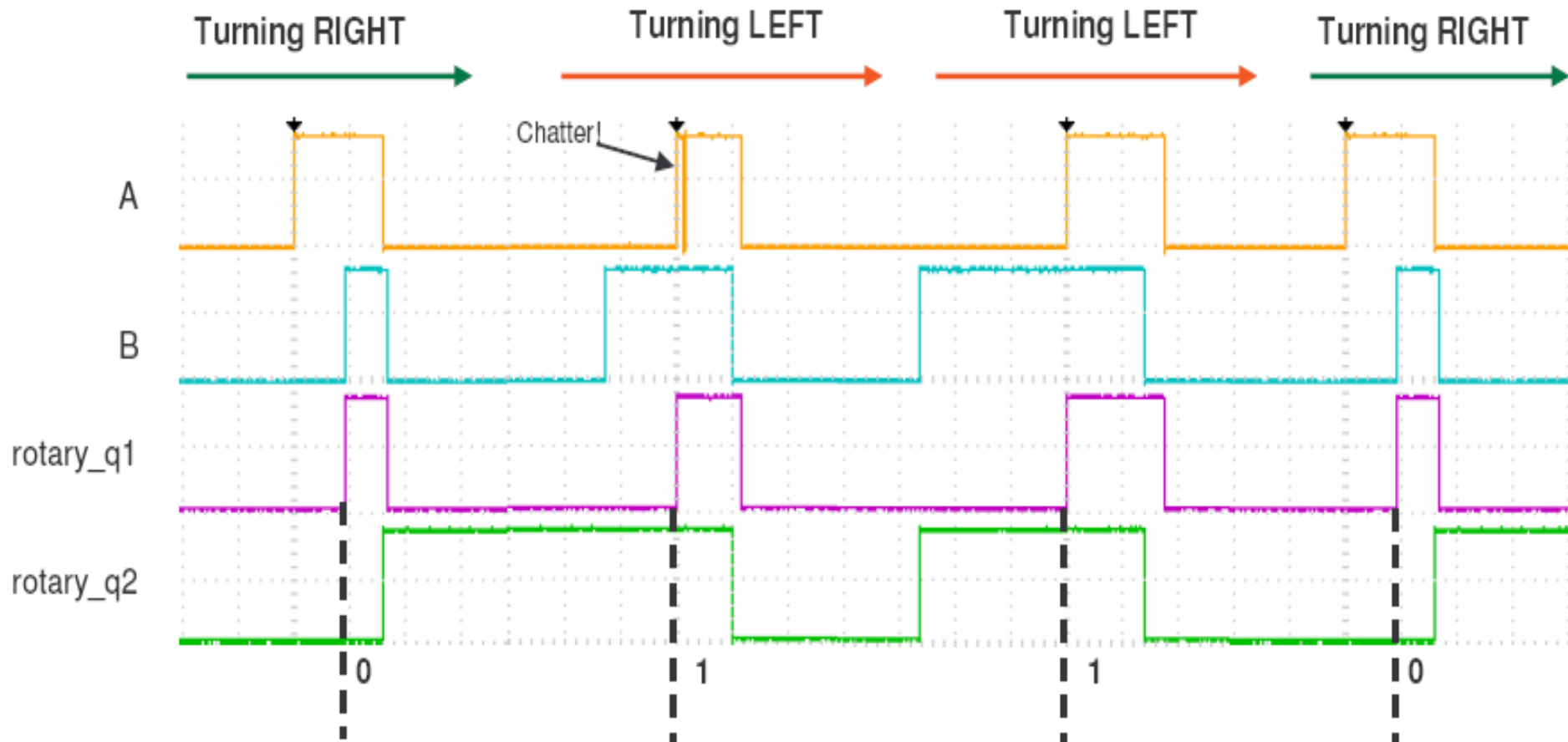
Rotary encoder: debouncing

```
rotary_filter: process(clk)
begin
    if clk'event and clk='1' then
        rotary_in <= rotary_b_in & rotary_a_in;
```

Rotary encoder: debouncing

```
case rotary_in is
  when "00" => rotary_q1 <= '0';
                  rotary_q2 <= rotary_q2;
  when "01" => rotary_q1 <= rotary_q1;
                  rotary_q2 <= '0';
  when "10" => rotary_q1 <= rotary_q1;
                  rotary_q2 <= '1';
  when "11" => rotary_q1 <= '1';
                  rotary_q2 <= rotary_q2;
  when others => rotary_q1 <= rotary_q1;
                  rotary_q2 <= rotary_q2;
end case;
```

Rotary encoder: direction



Rotary encoder: direction

```
direction: process(clk)
begin
    if clk'event and clk='1' then
        delay_rotary_q1 <= rotary_q1;
        if rotary_q1='1' and delay_rotary_q1='0' then
            rotary_event <= '1';
            rotary_left <= rotary_q2;
        else
            rotary_event <= '0';
            rotary_left <= rotary_left;
        end if;
    end if;
end if;
end process direction;
```

Demo: Use language templates

In Vivado

Edit = > Language templates

e.g. debounce circuit: coding examples => misc
 entity declaration: common constructs
 XDC properties
 State-Machines coding examples

Demo: Use IP Catalog

In Vivado

Project Manager = > IP Catalog

e.g. Block Memory Generator
 Clocking Wizard

PS2 : interface: other version

```
process (PS2C) begin --polling new data and redirecting outputs
if (falling_edge (PS2C)) then
    count <= count + 1;
    data_driver (count)<=PS2D;
    if count = 10 then
        count <= 0;
        led_in <= data_driver (8 downto 1);
    end if;
end if;
end process;
led <= led_in;
```

“Clock dedicated route = false”

Contact

Ing. Dirk Van Merode MSc.
Project Coordinator DESIRE
Thomas More | Campus De Nayer
Technology & Design
J. P. De Nayerlaan 5
2860 Sint-Katelijne-Waver

Belgium
Tel. + 32 15 31 69 44
Gsm + 32 496 26 84 15
dirk.vanmerode@thomasmore.be
Skype dirkvanmerode
www.thomasmore.be

Dr. Ing. Peter Arras MSc.
International Relations Officer
KU Leuven | Campus De Nayer
Faculty of engineering technology
J. P. De Nayerlaan 5
2860 Sint-Katelijne-Waver

Belgium
Tel. + 32 15 31 69 44
Gsm + 32 486 52 81 96
peter.arras@kuleuven.be
Skype pfjlaras
www.iw.kuleuven.be

Sources

<http://www.allaboutcircuits.com/textbook/digital/chpt-4/contact-bounce/>

https://eewiki.net/pages/viewpage.action?pageId=28278929&preview=/28278929/28508225/ps2_keyboard.vhd#PS/2KeyboardInterface%28VHDL%29-CodeDownloads

Digital Design Using Digilent FPGA Boards

Richard E. Haskell

Darrin M. Hanna

LBE Books – Third Edition, 2014