VHDL Intermediate

Tarek Eldeeb, Valeo Expert

Prerequisites

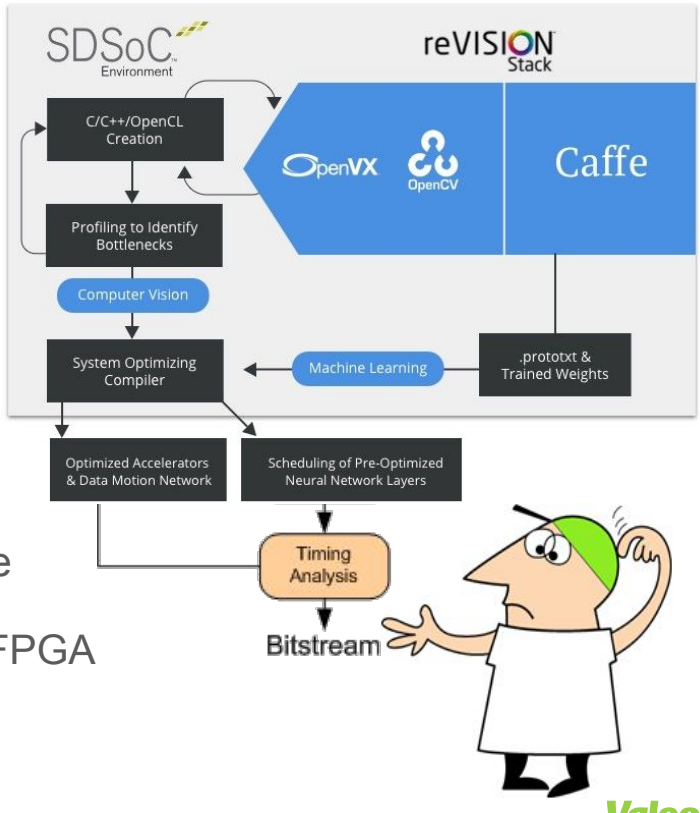
Knowledge

- Basic VHDL Syntax knowledge
- Basic FPGA Technology knowledge
  - Udemy Training

Installed Tools

- Xilinx tools (ISE/XPS 14.7) should be preinstalled with debugger drivers.
- Xilinx development kit with Spartan FPGA
  - ISE/XPS Download

SDSoc: Xilinx Tutorials - Demo Video



Training Outline

Day #1

- Revision: VHDL I
- Pipelining
- Static Timing Analysis
- Exercise

Day #2

- FPGA Advanced:
  - IO banks and Clocks
  - Special blocks
- AXI Bus: Specs and Applications
- Exercise

Day #3: Practical Workshop

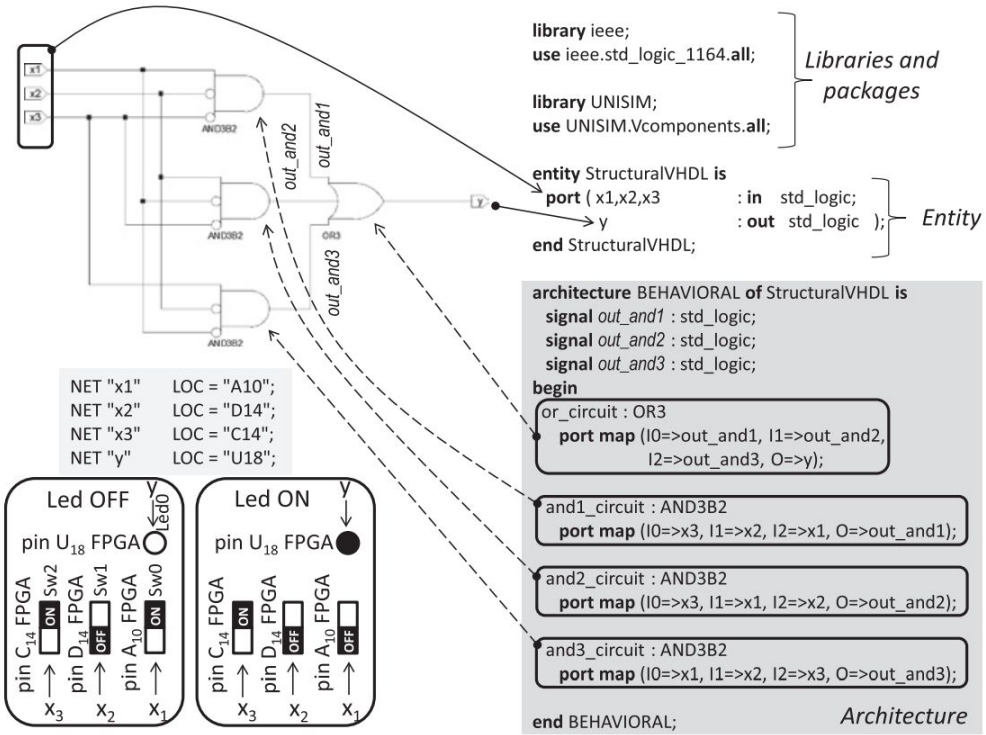
- Analyse Current Design
- Redesign for expandability

Revision

Rule #1: We are describing a HW circuit!

When you are describing hardware in VHDL, you are only describing the behaviour. The actual circuit will be synthesized (by the tools, eg. Xilinx ISE) to gates. The FPGA then implements the gates. The FPGA does NOT execute the VHDL code directly!!

Revision




```
library ieee;
use ieee.std_logic_1164.all;

library UNISIM;
use UNISIM.Vcomponents.all;
```

Libraries and packages

```
entity StructuralVHDL is
  port ( x1,x2,x3          : in  std_logic;
         y                 : out std_logic );
end StructuralVHDL;
```

Entity

```
architecture BEHAVIORAL of StructuralVHDL is
  signal out_and1 : std_logic;
  signal out_and2 : std_logic;
  signal out_and3 : std_logic;
begin
  or_circuit : OR3
    port map (I0=>out_and1, I1=>out_and2,
              I2=>out_and3, O=>y);

  and1_circuit : AND3B2
    port map (I0=>x3, I1=>x2, I2=>x1, O=>out_and1);

  and2_circuit : AND3B2
    port map (I0=>x3, I1=>x1, I2=>x2, O=>out_and2);

  and3_circuit : AND3B2
    port map (I0=>x1, I1=>x2, I2=>x3, O=>out_and3);

end BEHAVIORAL;
```

Architecture

Revision

Declaration of libraries:

```
library . . . ;
use . . . ;
. . . . . .
```

Description of entity

```
entity <name of entity> is
Declaration of generic parameters
Declaration of ports
end <name of entity>;
```

Entity declarative part

Description of architecture

```
architecture <name of architecture> of <name of entity> is
Declaration of signals
Declaration of constants
Declaration of types
Declaration of components
Declaration of functions (with bodies)
Declaration of procedures (with bodies)
Declaration of shared variables
begin

Body of architecture

end <name of architecture>;
```

Architecture declarative part

Connected components which are either library primitives or previously designed circuits from:

a) HDL specifications
b) schematic specifications
c) IP cores

- Structural
- Mixed
- Behavioral

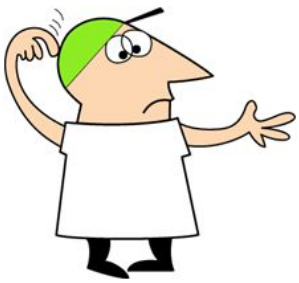
Examine VHDL code and component instantiation for schematic entries

Concurrent constructions:

a) signal assignments
b) VHDL processes

Xilinx ISE *Design* tab

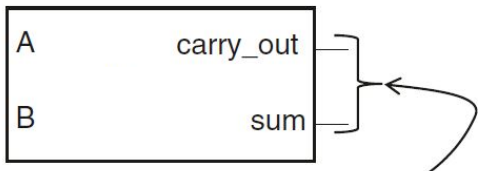
Design Utilities
. . . . . .
View HDL Functional Model
View HDL Instantiation Template



Revision

```
1  library IEEE;
   use IEEE.std_logic_1164.all;
```
use of libraries and packages

HA



Interface

```
entity half_adder is
 port (  A         : in STD_LOGIC;
         B         : in STD_LOGIC;
 2       carry_out : out STD_LOGIC;
         sum       : out STD_LOGIC);
end half_adder;
```

Architecture

```
architecture half_adder_behavior of half_adder is
begin
 3      sum       <= A xor B;
        carry_out <= A and B;
end half_adder_behavior;
```

A	B	carry_out	sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Revision

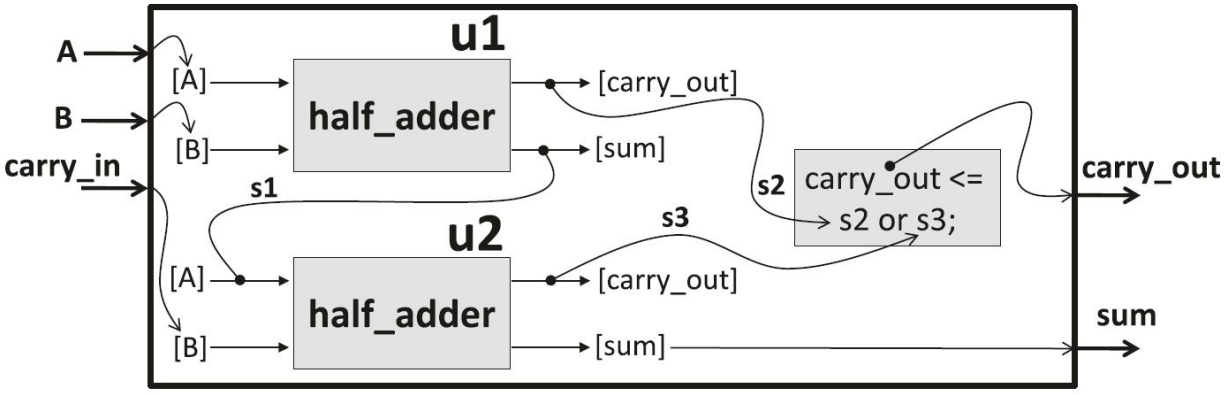
```
library IEEE;
use IEEE.std_logic_1164.all;
entity FULLADD is
port (  A, B, carry_in      : in  std_logic;
        sum, carry_out      : out std_logic );
end FULLADD;
architecture STRUCT of FULLADD is
signal s1, s2, s3 : std_logic;
begin
   u1: entity work.half_adder
               port map(A, B, s2, s1);
   u2: entity work.half_adder
               port map(s1, carry_in, s3, sum);
   carry_out <= s2 or s3;
end STRUCT;
```

Example of positional association

```
library IEEE;
use IEEE.std_logic_1164.all;
entity half_adder is
  port (
    A         : in STD_LOGIC;
    B         : in STD_LOGIC;
    carry_out : out STD_LOGIC;
    sum       : out STD_LOGIC    );
end half_adder;
architecture half_adder_behavior of half_adder is
begin
    sum       <= A xor B;
    carry_out <= A and B;
end half_adder_behavior;
```

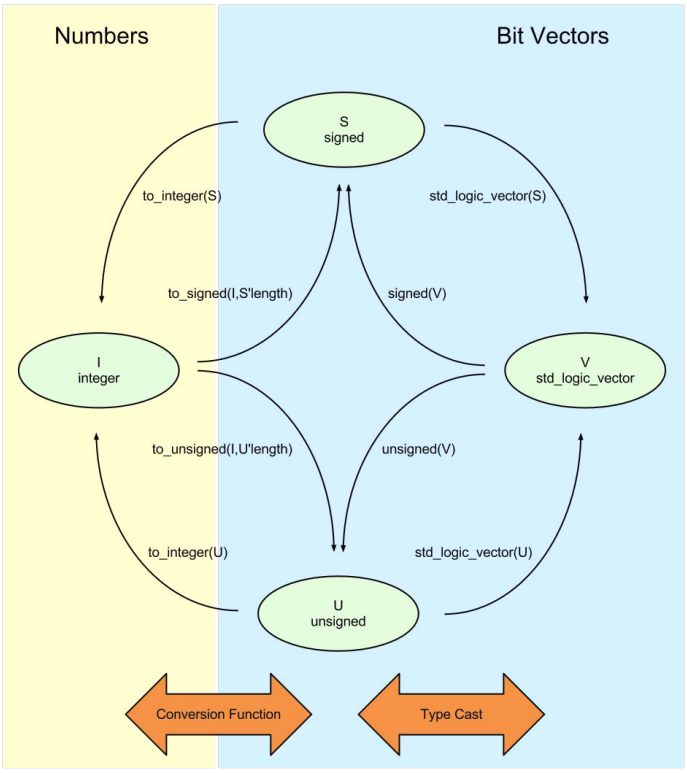
A	B	carry_in	carry_out	sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

FULLADD



TIPs: Packages/Types

```
library IEEE;
  use IEEE.std_logic_1164.all;
  use IEEE.std_logic_textio.all;
  use IEEE.std_logic_arith.all;
  use IEEE.numeric_bit.all;
  use IEEE.numeric_std.all;
  use IEEE.std_logic_signed.all;
  use IEEE.std_logic_unsigned.all;
  use IEEE.math_real.all;
  use IEEE.math_complex.all;
```



What to use?

```
library ieee;
  use ieee.std_logic_1164.all;   -- UX01ZWLH
  use ieee.numeric_std.all;   -- un/signed
  use ieee.std_logic_arith.all; -- bad, old!
```


TIPs: Attributes

- signal'event
- signal'left/right
- signal'low/high
- signal'length
- signal'range
- signal'reverse_range

This code is NOT accepted: VHDL is hard-typed

```
b(0 to 7) <= a(7 downto 0)
```

```
Sol#1
function reverse_any_vector (a: in std_logic_vector)
return std_logic_vector is
  variable result: std_logic_vector(a'RANGE);
  alias aa: std_logic_vector(a'REVERSE_RANGE) is a;
  begin
    for i in aa'RANGE loop
      result(i) := aa(i);
    end loop;
    return result;
  end; -- function reverse_any_vector
```

```
Sol#2
signal a : std_logic_vector(0 to 7);
signal b : std_logic_vector(7 downto 0);
..
for i in a'range generate
  b(i) <= a(i)
end generate;
```

TIPs: Process

Processes should be broken till each has a **SINGLE** type:

1. Pure Combinational
2. Pure Sequential
3. Sequential with async reset

TIPs: Process

1- Pure Combinational

- Rule 1: Every input (that can affect the output(s)) must be in the sensitivity list.
- Rule 2: Every output must be assigned a value for every possible combination of the inputs

```
process (A, B)
begin
   if (SEL = '0') then
      Y <= A;
   else
      Y <= B;
   end if;
end process;
```

```
process (SEL, A, B)
begin
      if (SEL = '0') then
            Y <= A;
      end if;
end process;
```

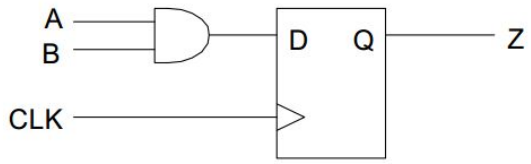
TIPs: Process

2- Pure Sequential

- Rule 1: Only the clock should be in the sensitivity list
- Rule 2: Only signals that change on the same edge of the same clock should be part of the same process

rising_edge(clk)

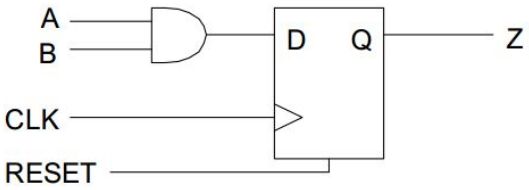
```
process (CLK)
begin
  if (CLK'event and CLK='1') then
     Z <= A and B;
  end if;
end process;
```



TIPs: Process

3- Sequential with async reset

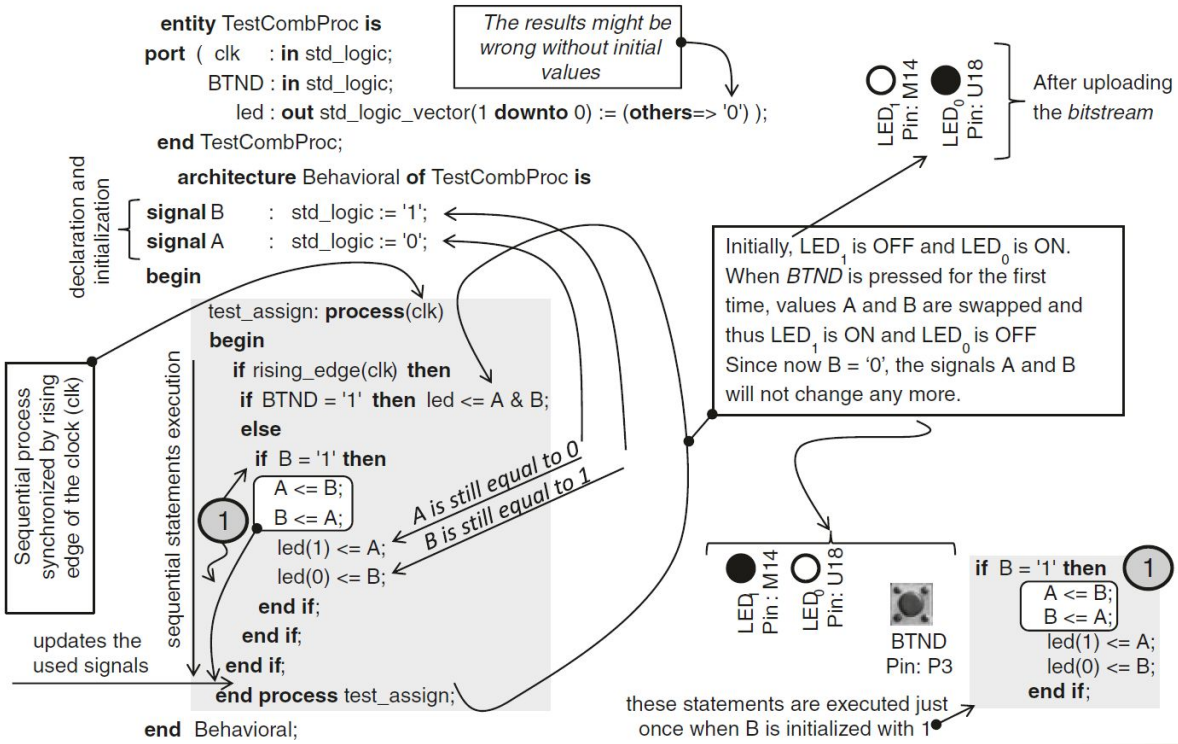
```
process (CLK, RESET)
begin
  if (RESET = '1') then
     Z <= '0';
  elsif (CLK'event and CLK='1') then
     Z <= A and B;
  end if;
end process;
```



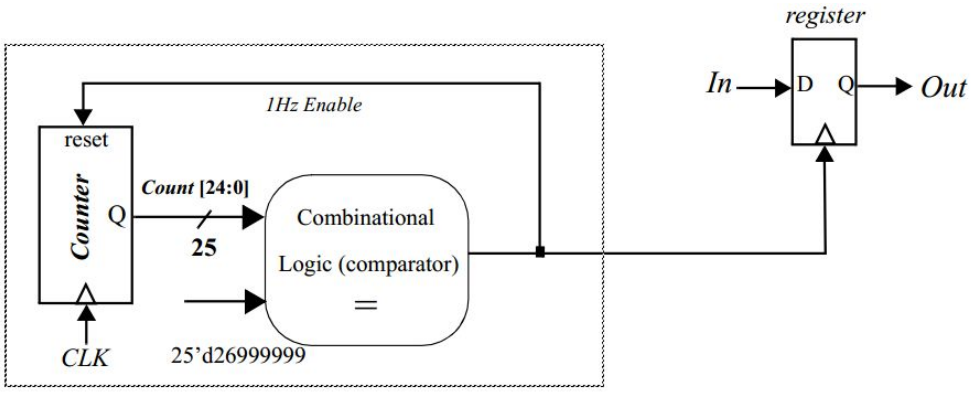
Quiz Time :-)

```
entity TestCombProc is
port ( clk : in std_logic;
       BTND : in std_logic;
       led : out std_logic_vector(1 downto 0) := (others=> '0') );
end TestCombProc;
architecture Behavioral of TestCombProc is
       Signal B : std_logic := '1';
       Signal A : std_logic := '0';
begin
test_assign: process(clk)
       begin
       if rising_edge(clk) then
              if BTND = '1' then led <= A & B;
              else
                     if B = '1' then
                            A <= B;
                            B <= A;
                            led(1) <= A;
                            led(0) <= B;
                     end if;
              end if;
       end if;
end process test_assign;
end Behavioral;
```

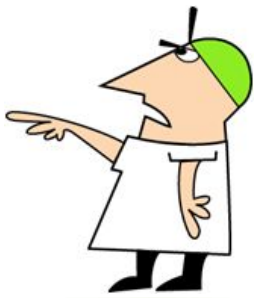
Quiz : Answered



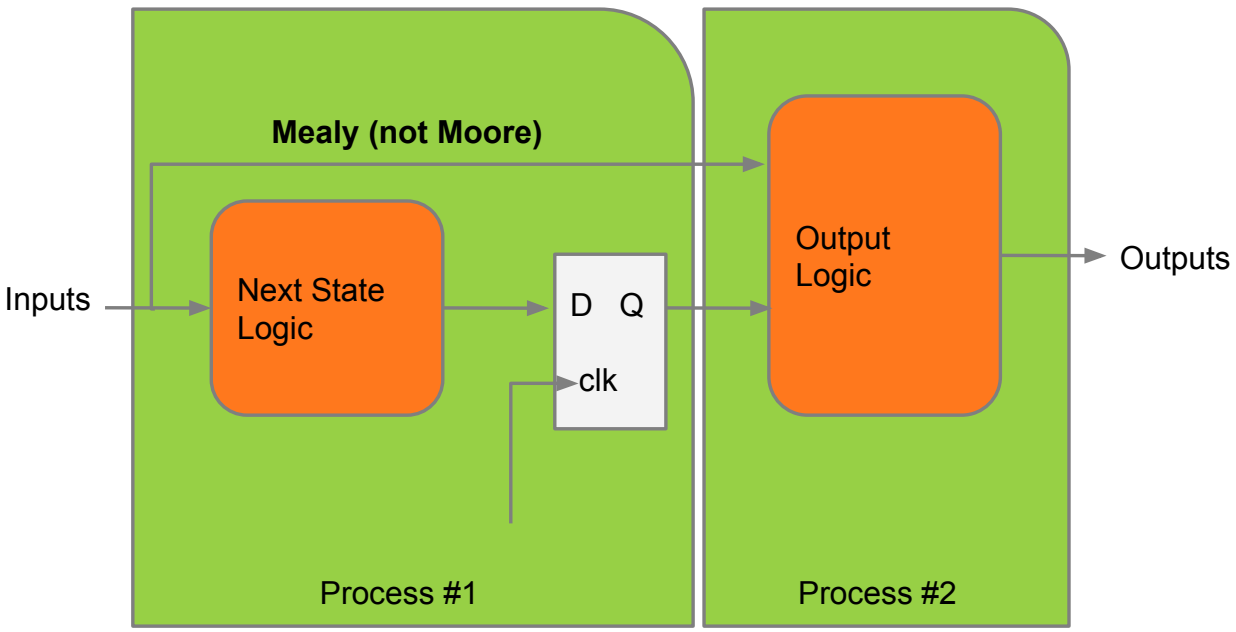
TIPs: Digital glitches!



- A 1 Hz signal is derived from a 27MHz to update a register
- Assume the counter reset is synchronous, a global reset (cnt=0) is not shown
- What can be wrong?

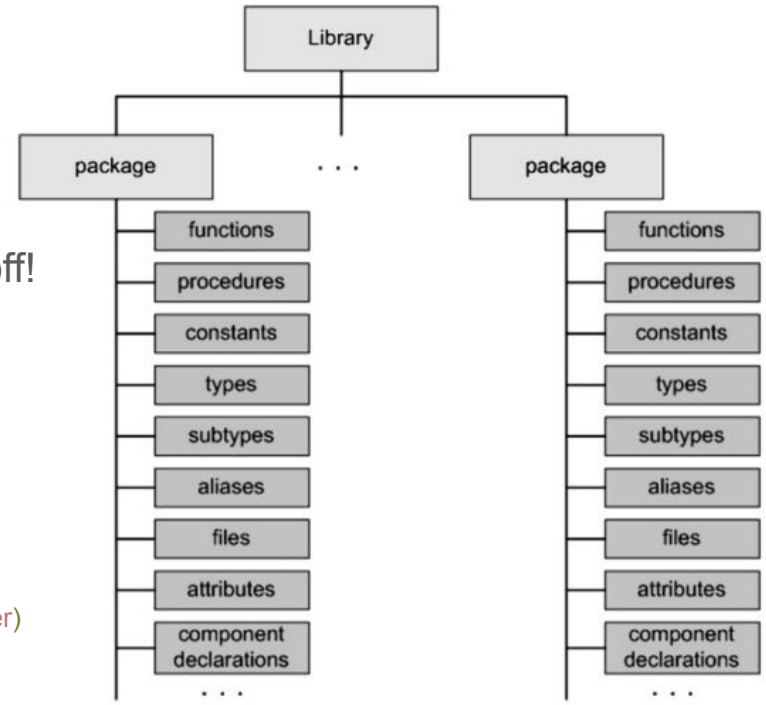


TIPs: State Machines



VHDL Libraries and Packages

- Code reusability and Modularity
- Best Practices / Shorthands
- Investing in Packages really pays off!



Examples:

```
type ArrOfSTDV8_t is array (natural range <>) of
  std_logic_vector(7 downto 0);

function ilog2(val : integer) return integer ;

function terni(cond : boolean; res_true, res_false : integer)
  return integer;
```


VHDL Records

```
example_inst : axi_example_peripheral
    port_map(
        clk              => clk,
        data             => data,
        s_axi_gp0_aclk   => aclk,
        s_axi_gp0_arvalid => periph_axi_arvalid,
        s_axi_gp0_awvalid => periph_axi_awvalid,
        s_axi_gp0_bready  => periph_axi_bready,
        s_axi_gp0_rready  => periph_axi_rready,
        s_axi_gp0_wlast   => periph_axi_wlast,
        s_axi_gp0_wvalid  => periph_axi_wvalid,
        s_axi_gp0_arid    => periph_axi_arid,
        s_axi_gp0_awid    => periph_axi_awid,
        s_axi_gp0_wid     => periph_axi_wid,
        s_axi_gp0_arburst => periph_axi_arburst,
        s_axi_gp0_arlock  => periph_axi_arlock,
        s_axi_gp0_arsize  => periph_axi_arsize,
        ... MANY MANY LINES REMOVED!
    );
```

```
example_inst : axi_example_peripheral
    port_map(
        clk            => clk,
        data           => data,
        s_axi_gp0_aclk => aclk,
        s_axi_gp0_in   => periph_axi.from_master,
        s_axi_gp0_out  => periph_axi.to_master
    );
```

VHDL Records

```
library ieee;
use ieee.std_logic_1164.all;
package my_package is
   type from_AXI_master is record
        arvalid : std_logic;
        awvalid : std_logic;
         ... removed!
    end record from_AXI_master;
    type to_AXI_master is record
        .. removed!
        rresp : std_logic_vector(1 downto 0);
        rdata : std_logic_vector(31 downto 0);
    end record to_AXI_master;
    type AXI_slave is record
        to_master : to_AXI_master;
        from_master : from_AXI_master;
    end record AXI_slave;
end my_package;
```

Packages can also include:

- Constants
- Types
- Common components
- Functions
- Procedures

Generate vs Loop!

Generate:

1. Replicating Logic in VHDL
2. Turning on/off blocks of logic in VHDL

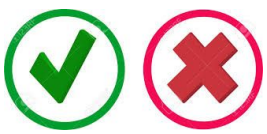
```
GEN_ADD: for I in 0 to 7 generate

  LOWER_BIT: if I=0 generate
    U0: HALFADD port map  (A(I),B(I),S(I),C(I));
  end generate LOWER_BIT;

  UPPER_BITS: if I>0 generate
    UX: FULLADD port map (A(I),B(I),C(I-1),S(I),C(I));
  end generate UPPER_BITS;

END generate GEN_ADD;

COUT <= C(7);
```

Loop:

1. Must be in a process
2. Executes in a single clock (take care!)

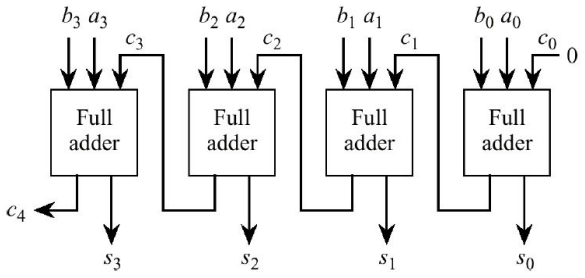
```
process (A)
begin
  Z <= "0000";
  for i in 0 to 3 loop
    if (A = i) then
      Z(i) <= '1';
    end if;
  end loop;
end process;
```

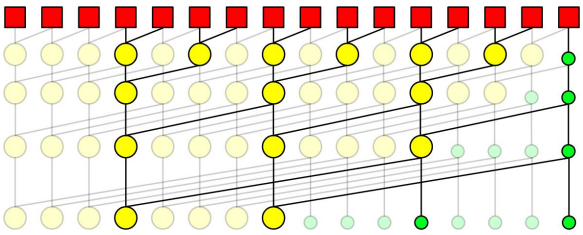
```
for i in mySrc'range loop
  my2Ddest(i) <= mySrc(i);
end loop;
```

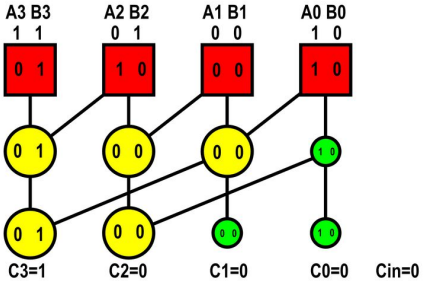


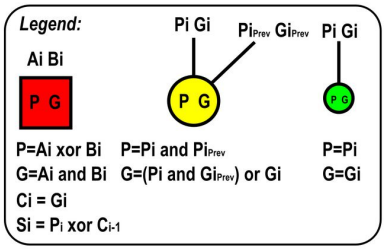
TIPs: Simple things are not simple

- Ripple Adder: Delay is *O*(n)
- Kogge-Stone: Delay is *O*(Log_2(n))
  - Area? Power?
  - Mid-way trees?



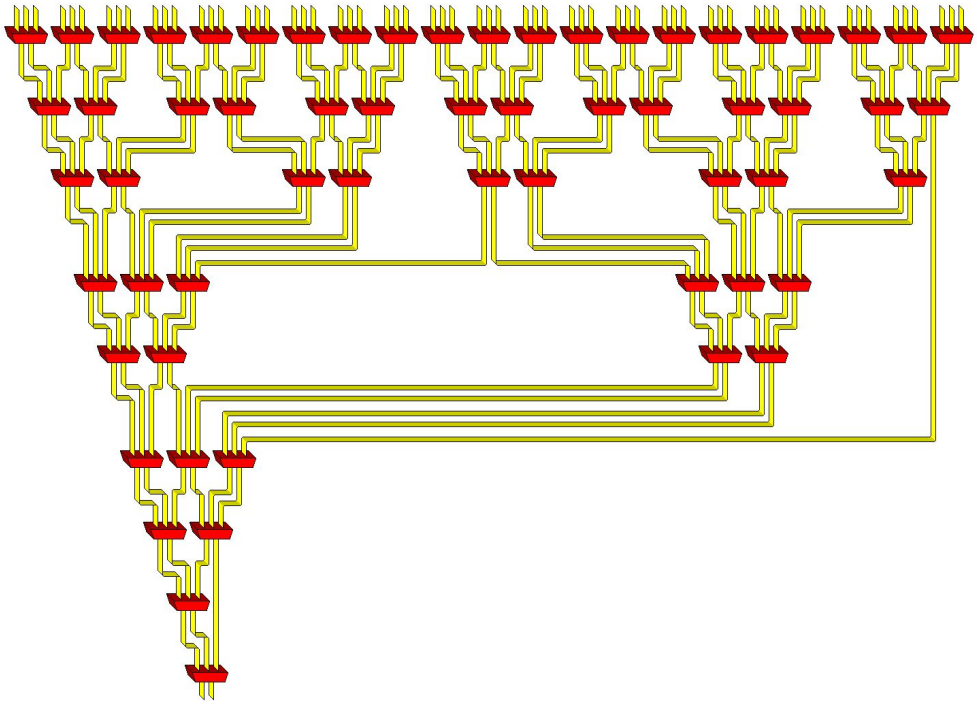






TIPs: Simple things are not simple

- Multiplication is easier ;-)
- Needs a CPA
- Half-width? Rounding?
- Fused Multiply-Add



TIPs: Simple things are not simple

Newton-Raphson: Division A/B

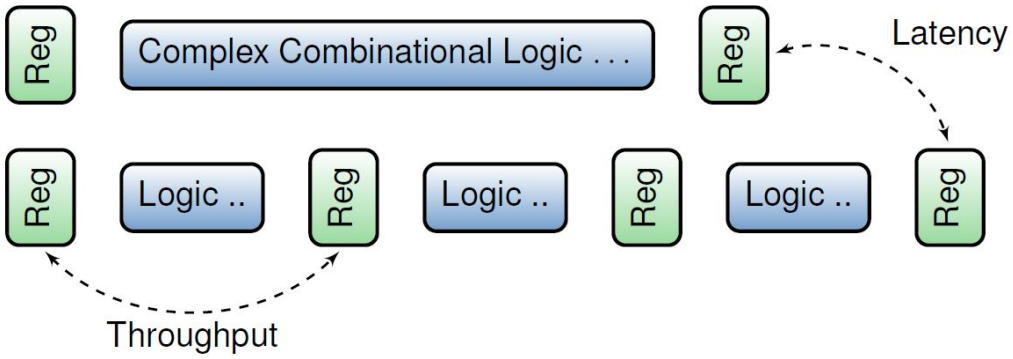
1. Look-up an estimate for the reciprocal of the divisor: X_0 [≈1/B]
2. Compute successively more accurate estimates of the reciprocal: X_i
3. Compute the quotient by multiplying the dividend by the reciprocal of the divisor Q= A x (1/B)

$$X_{i+1} = X_i - \frac{f(X_i)}{f'(X_i)} = X_i - \frac{1/X_i - D}{-1/X_i^2} = X_i + X_i(1 - DX_i) = X_i(2 - DX_i)$$

Newton-Raphson: Square-root

$$x_{i+1} = x_i(3 - bx_i^2)/2$$

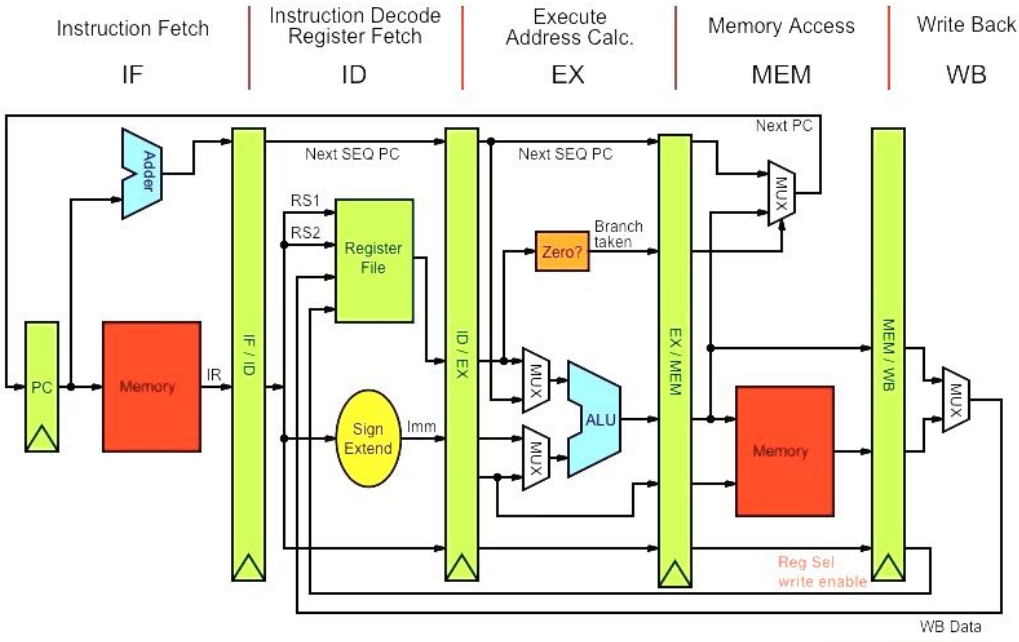
Pipelines



- Area?
- Power?
- Optimal number of stages?
- Automatic Register Balancing

Pipelines

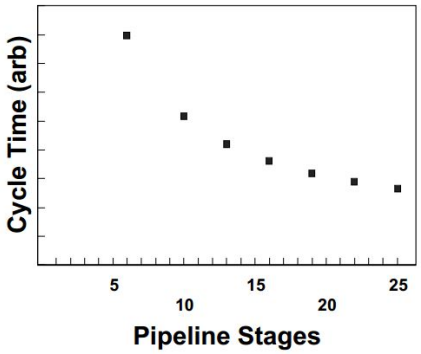
- Does **Feedback** break the pipe?

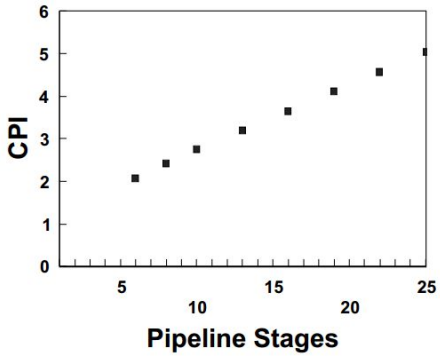


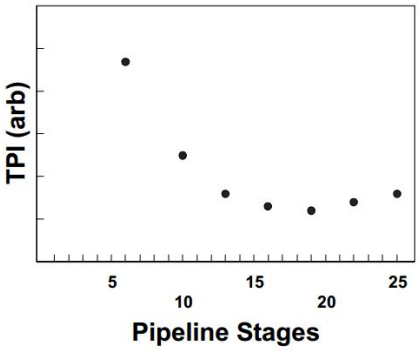
Lab #1

1. Write a vhdl module that adds 4 x 18bit registered inputs, and deliver the result at the clock positive edge
2. Write a TB to check results
3. Analyse
   a. RTL Schematics
   b. Technology Schematics
   c. Post PAR static timing
4. Add pipeline stages, calculate on 2 steps
5. Do steps 2~3
6. ~~Add 3 shift registers at output~~
7. ~~Do steps 2~3~~
8. ~~Enable register balancing~~
9. ~~Do steps 2~3~~

Pipelines

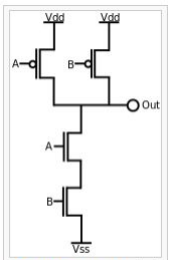






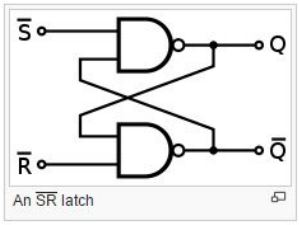
- CPU: Who breaks the pipeline?
- What sets the limits to other digital designs?

Pipelines: Latches



CMOS NAND gate

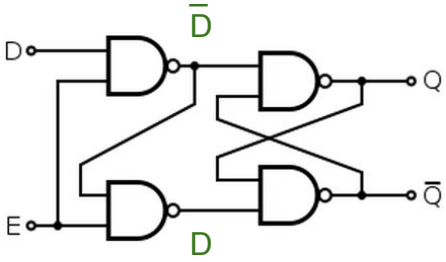
INPUT		OUTPUT
A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0



An $\overline{SR}$ latch

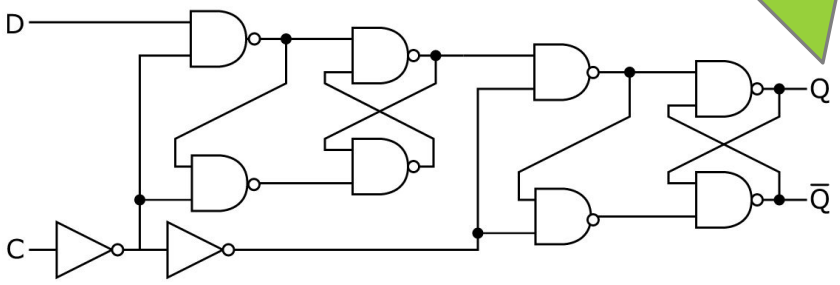
$\overline{SR}$ latch operation		
$\overline{S}$	$\overline{R}$	Action
0	0	not allowed
0	1	Q = 1
1	0	Q = 0
1	1	No Change

anim



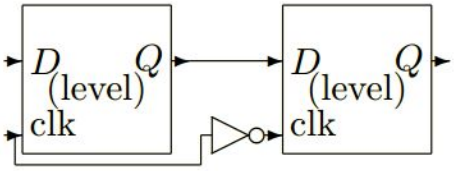
Latch:
- IO Ports
- Incomplete statements!

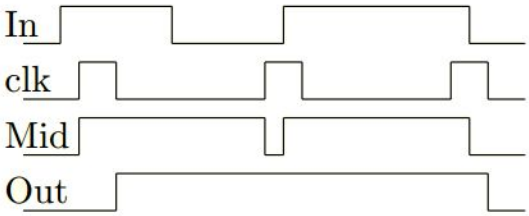
MS Edge D-Flip Flop:
- Registers
- Negative edge?

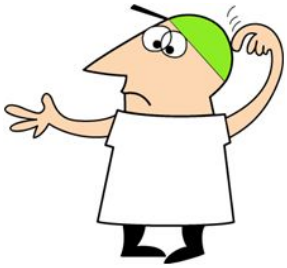


Why latches have poor timing?

Pipelines: Latches





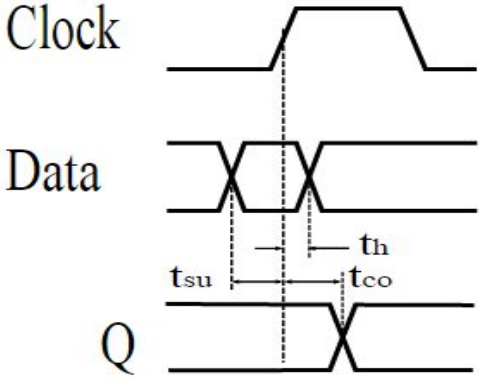


Pipelines: Latches

- Setup Time
- Hold time
- Clock-to-Output time

Maximum Combinational Delay

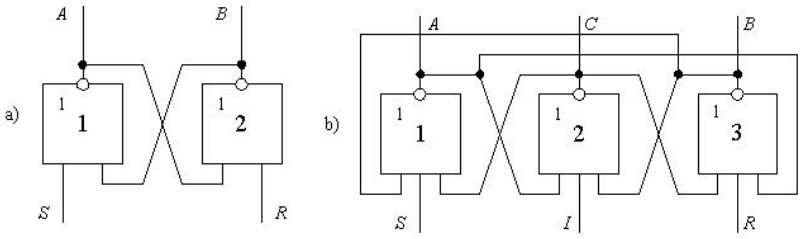
$$\frac{T_{clk} - T_{co} - T_{su}}{1/F_{max}}$$

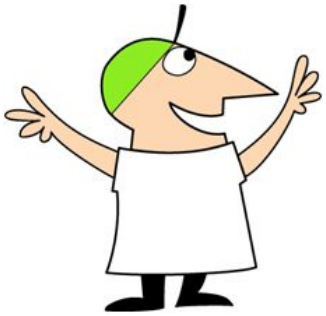


- Metastability:
  - Respect T_{su} and T_h
  - Practical range?
  - System inputs?
- In a shift register, Can $T_{co} < T_h$?

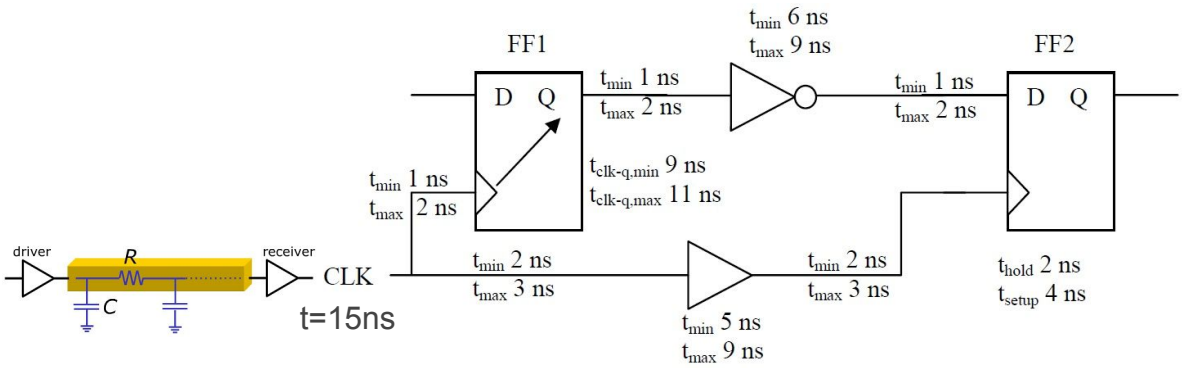
Pipelines: Out-of-the-box!

- Multi-valued Logic
- Wave pipelining!





Static Timing Analysis: Setup/Hold Violation?



Hold Analysis: Will new data overwrite the old at FF2?

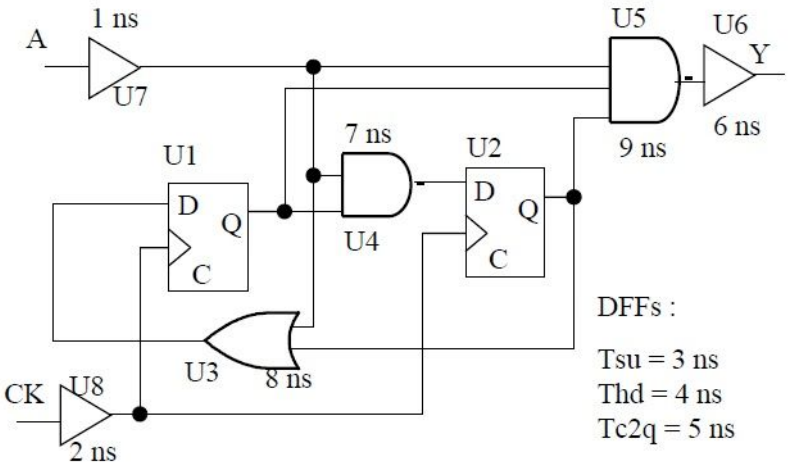
- Min Data Delay
- Max Clock Delay
- Tclk-Td = 18-17 = +ve slack

Setup Analysis: Will data miss the sampling edge at FF2?

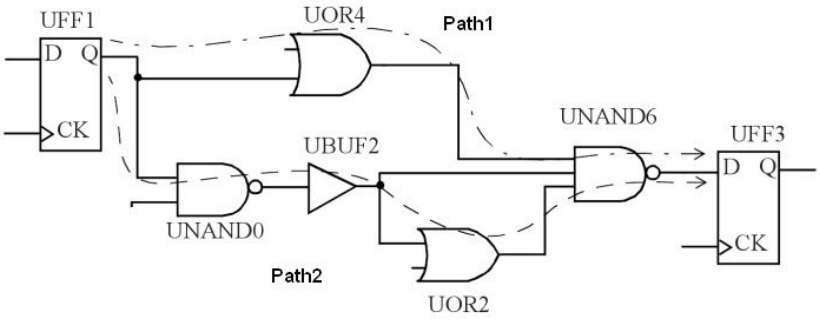
- Max Data Delay
- Min Clock Delay
- Tclk-Td = 20-26 = -ve slack

Static Timing Analysis

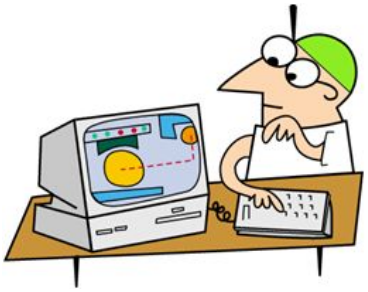
1. Reg-to-Reg delay [16ns]
   a. Possibilities?
2. Input Setup time [12-2ns]
   a. Tsu+Max.Td-Min.Tck
3. Input Hold time [-2ns]
   a. Thd+(Max.Tck-MinTd)
4. Clock-to-Output delay [22ns]
   a. Critical path?
5. Pin-to-Pin Combinational delay [16ns]
6. Max Clock frequency?
   a. 1/Max(Reg2Reg, Clk2Out, Pin2I



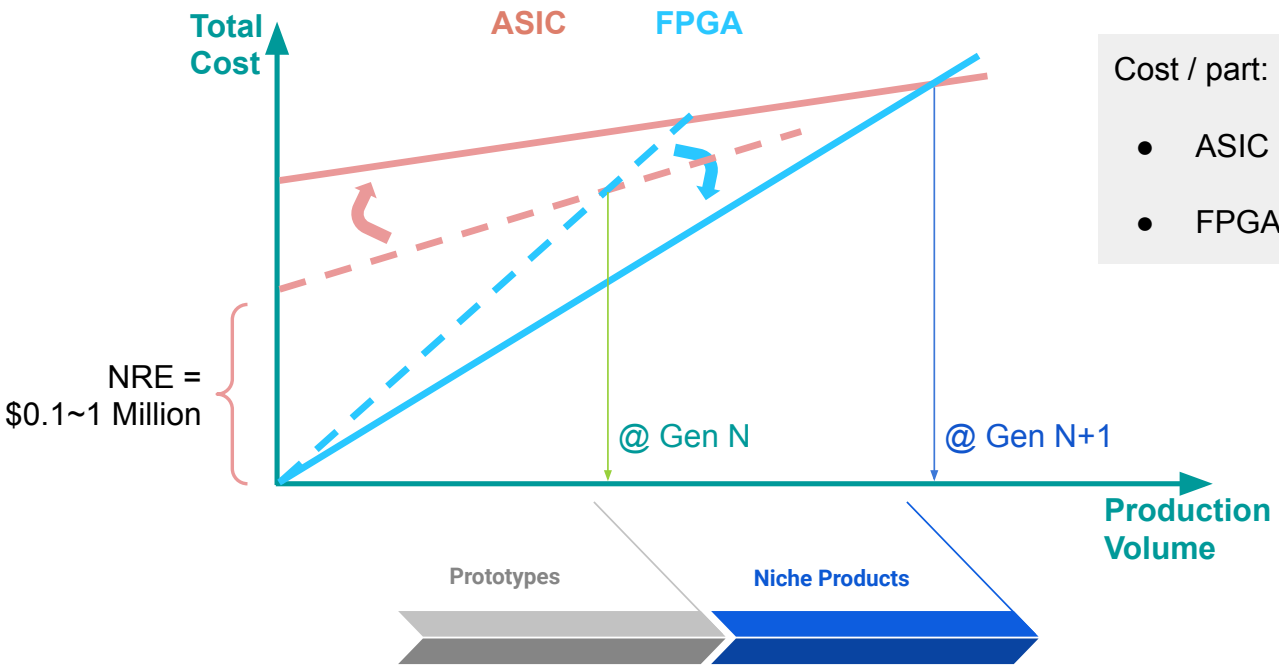
Static Timing Analysis



- Multiple Paths => Min, Max
- Backtrace method
- Combinational loops?
- System: Where to optimize?



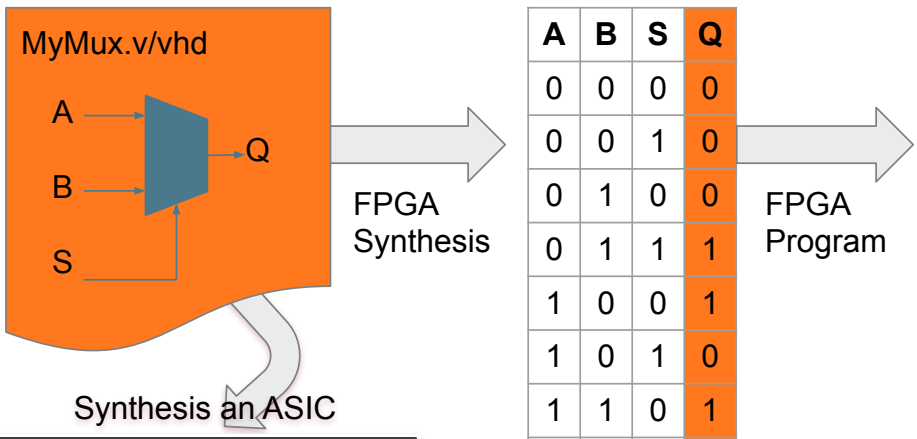
FPGAs for Prototyping



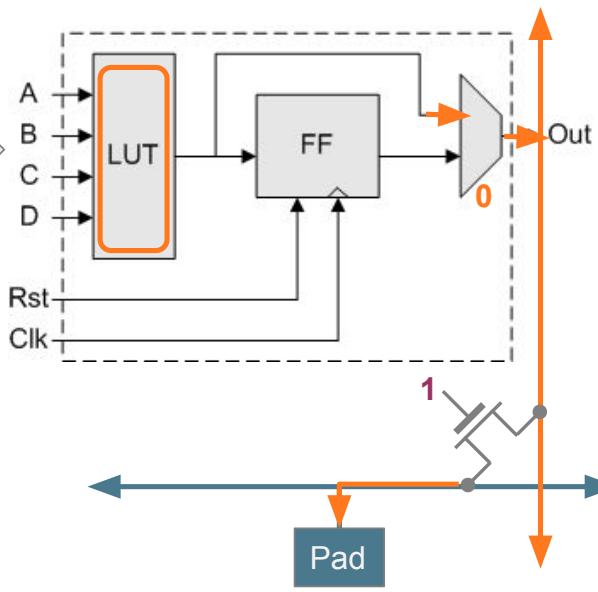
Cost / part:

- ASIC = $0.1~10
- FPGA = $10~100~4K

How FPGAs work?



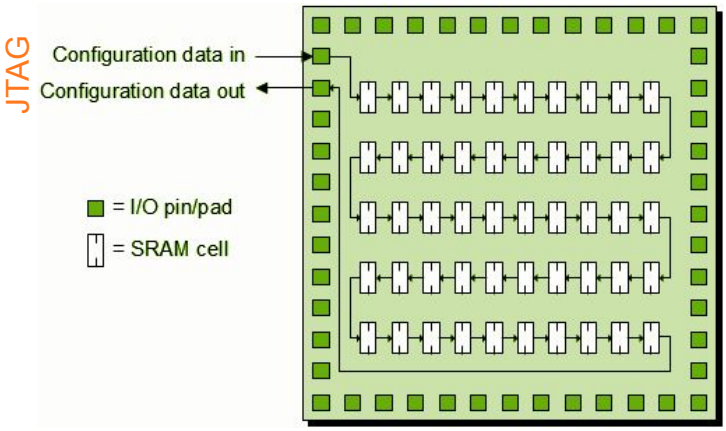
A	B	S	Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

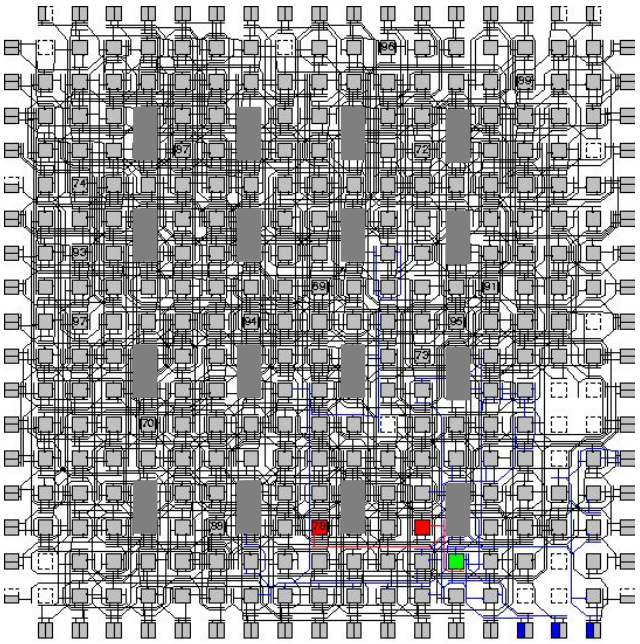


How FPGAs work?

More than LUTs:
DSP Blocks, RAM Blocks, MGT, DCM, ...

How do you Program an FPGA?





FPGA Evolution

Standardization (1980s)

- First FPGA 1984 by Xilinx
- 2.5-micron process technology
- More expensive than a µP

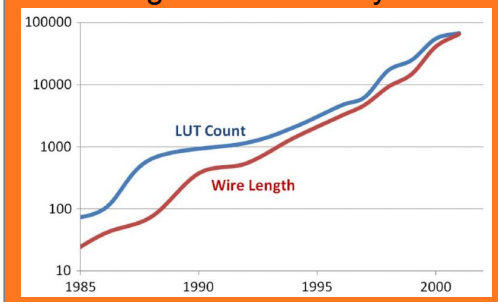
- Actel (now Microsemi) followed in 1988 with inventing the Antifuse, much smaller than SRAM

- Xilinx selected the 4-input LUT as a coarse-grained size

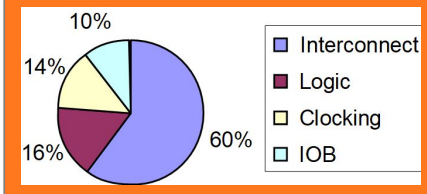
- Interconnects optimizations through long/short segments

Upsizing (1990s)

- Doubling transistors / 2 years



- Critical Path Delay:



Accumulating (2000s-Today)

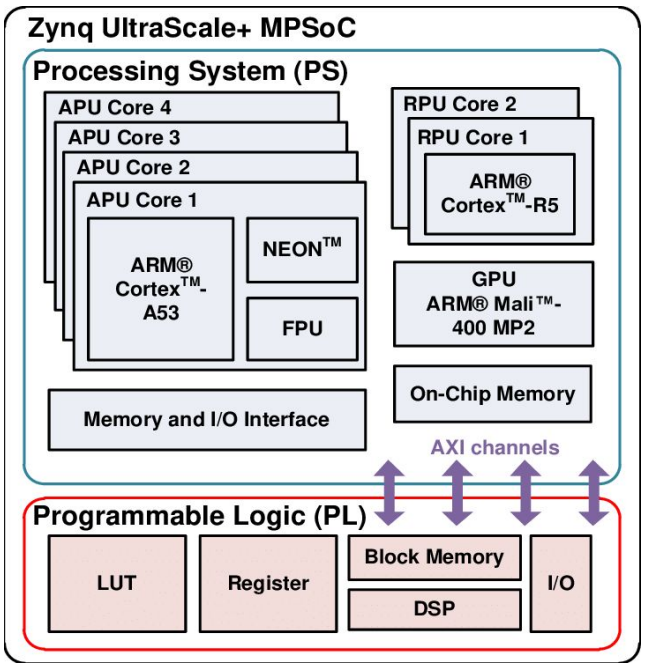
- Dedicated SoC blocks
  - CPU / GPU
  - Ethernet MAC
  - USB / PCIe endpoints

- Sharing IPs
  - Standard bus (AXI)
  - IP Security

- Design Tools
  - High Level Synthesis (C/Simulink)
  - Mixed-Signal interface
  - Power Management

FPGA Architecture Today

- Multiprocessor System-On-Chip (MPSoC)
- 16nm FinFET+ Technology
- 4K Video Processing (H2.265 Codec / Mali)
- Baseband Acceleration (5G)
- Extremely low-power variants for IoT



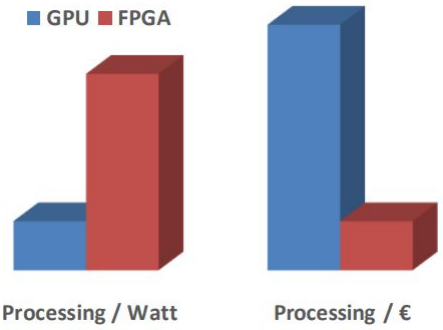
Valeo

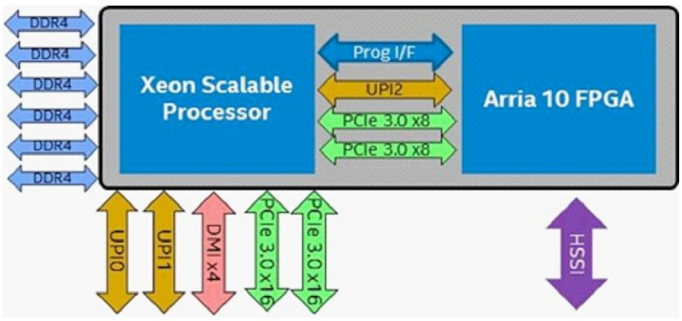
FPGA Acceleration

According to the application,

- GPU can show a speedup of 10X over multi-threaded CPU
- FPGA can show a speedup of 4X over GPU!

Intel has acquired Altera in 2015 and included Arria in its die!



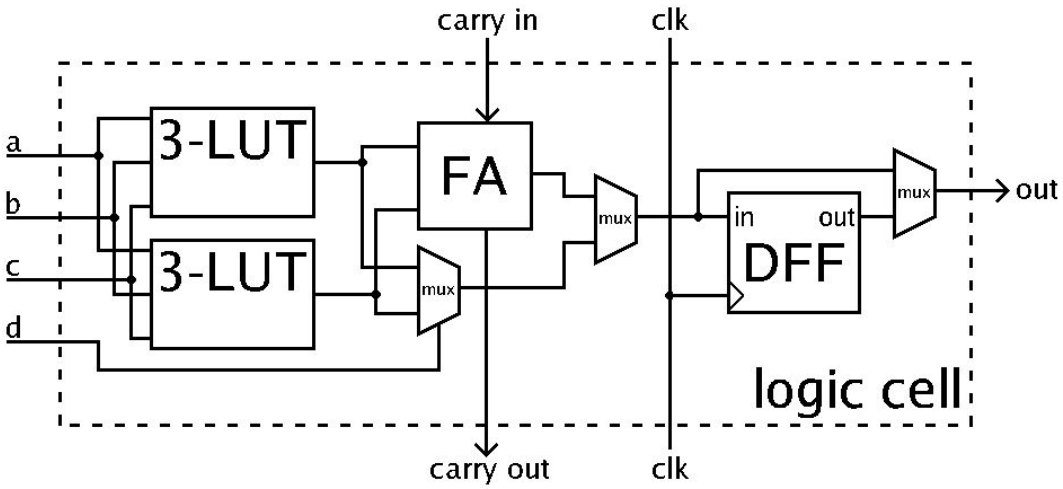


Details

Valeo

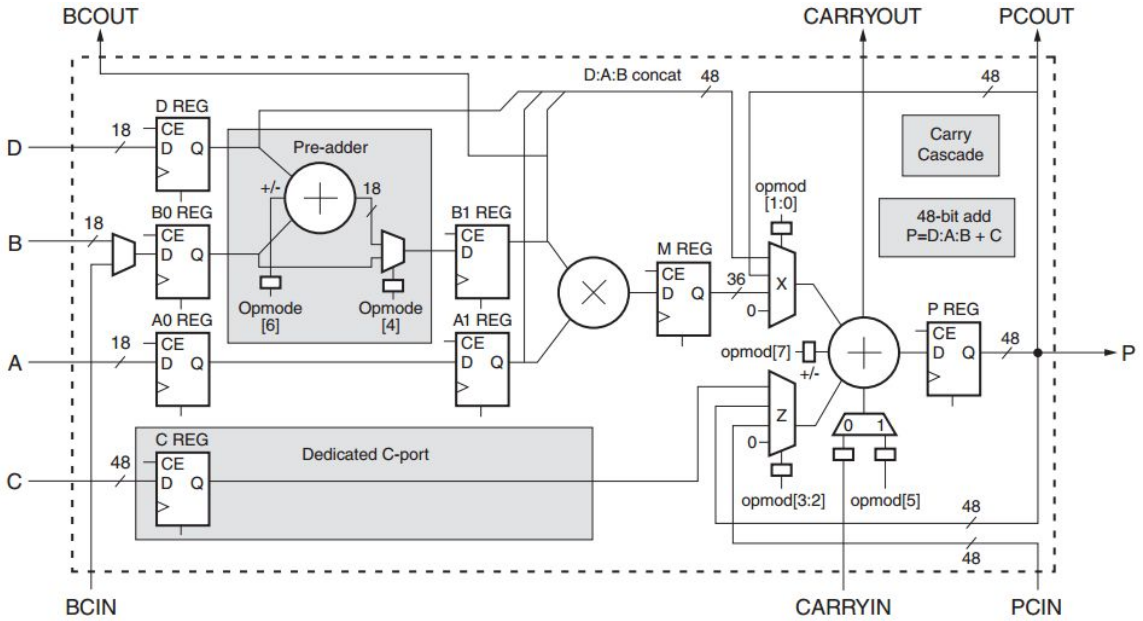
FPGA Contents

1- Programmable Logic (+Routing)



FPGA Contents

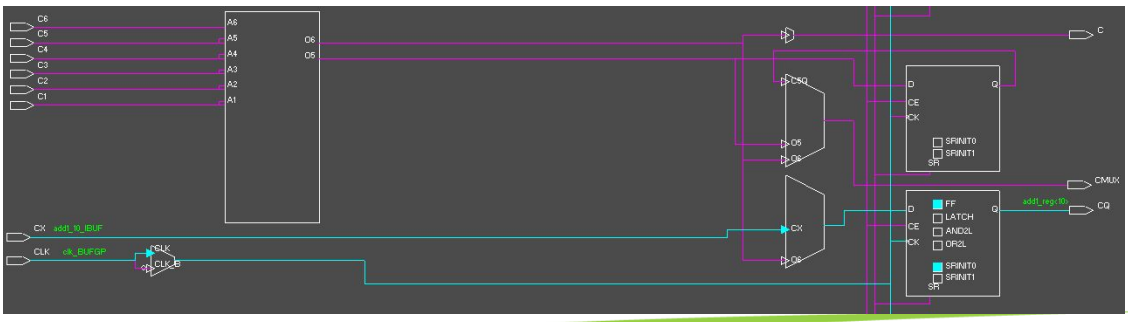
2- DSP Blocks



April 2020 I VHDL II

FPGA Contents: Exercise

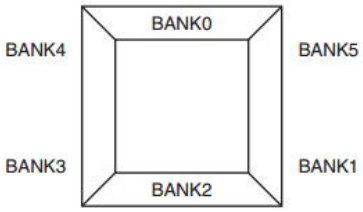
- Open your last project using Xilinx ISE
- PAR > View/Edit Routed Design (FPGA Editor)
- Identify:
  a. IOB of any pin
  b. Trace its route till its register
  c. Double click to view CLBs



FPGA Contents

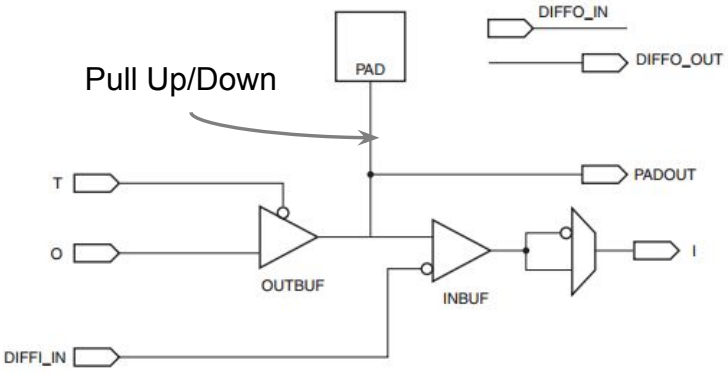
3- IO Pads

1. Single-Ended / Differential
2. Drive strength (Fan-out)
3. Slew rate
4. On-chip termination



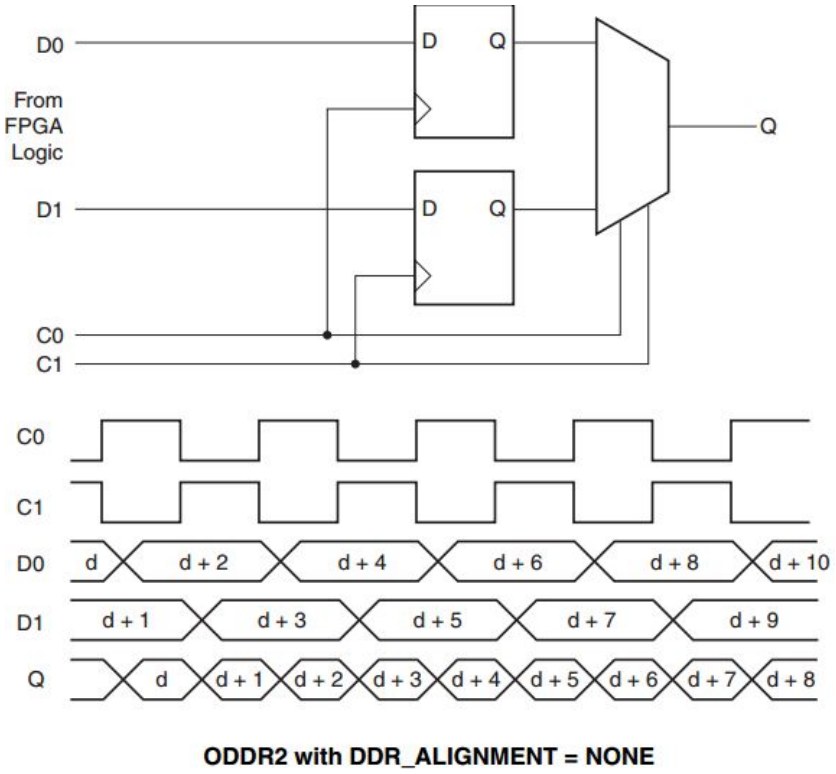
Xilinx Primitives [Single Ended]

1. IBUF (input buffer)
2. IBUFG (clock input buffer)
3. OBUF (output buffer)
4. OBUFT (3-state output buffer)
5. IOBUF (input/output buffer)



FPGA Contents

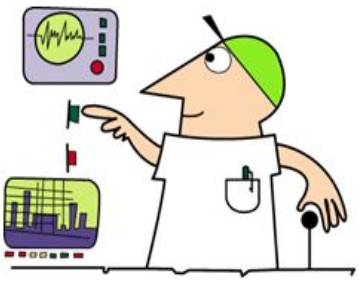
Many useful Blocks/Cells ..



ODDR2 with DDR_ALIGNMENT = NONE

FPGA Contents

- Low static and dynamic power (clock gating?)
- Multi-voltage, multi-standard IOs
- High-speed MGT serial transceivers
- Integrated Endpoint (PCIe)
- Integrated Memory Controllers
- Block RAM
- Clock Management
- Design Security



Lab #2

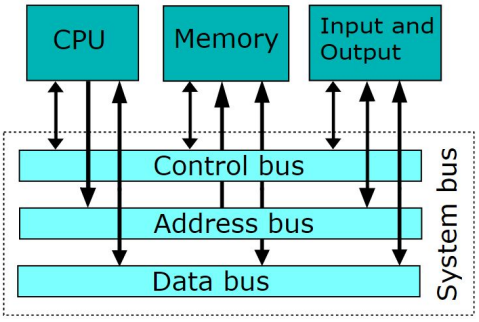
- Write a VHDL module to generate 2x and 0.5x of a given clock using PLL_BASE (xilinx primitive)
- Constrain the resulting clocks to IO pins, reimplement

```
pll_base_inst : PLL_BASE
generic map (
  BANDWIDTH => "OPTIMIZED",
  CLK_FEEDBACK => "CLKFBOUT",
  COMPENSATION => "SYSTEM_SYNCHRONOUS",
  DIVCLK_DIVIDE => a,
  CLKFBOUT_MULT => b,                  CLKFBOUT =>,
  CLKFBOUT_PHASE => 0.000,             CLKOUT0 =>,
  CLKOUT0_DIVIDE => c,                 CLKOUT1 =>,
  CLKOUT0_PHASE => 0.000,              CLKOUT2 =>,
  CLKOUT0_DUTY_CYCLE => 0.500,         CLKOUT3 =>,
  CLKOUT1_DIVIDE => d,                 CLKOUT4 =>,
  CLKOUT1_PHASE => 0.000,              CLKOUT5 =>,
  CLKOUT1_DUTY_CYCLE => 0.500,         LOCKED =>,
  CLKIN_PERIOD => e,                   RST => ,
  REF_JITTER => 0.010                  CLKFBIN =>,
  ) port map (                         CLKIN =>
       ..
```


Jump high: Computer Buses

Gen1: Single System Bus

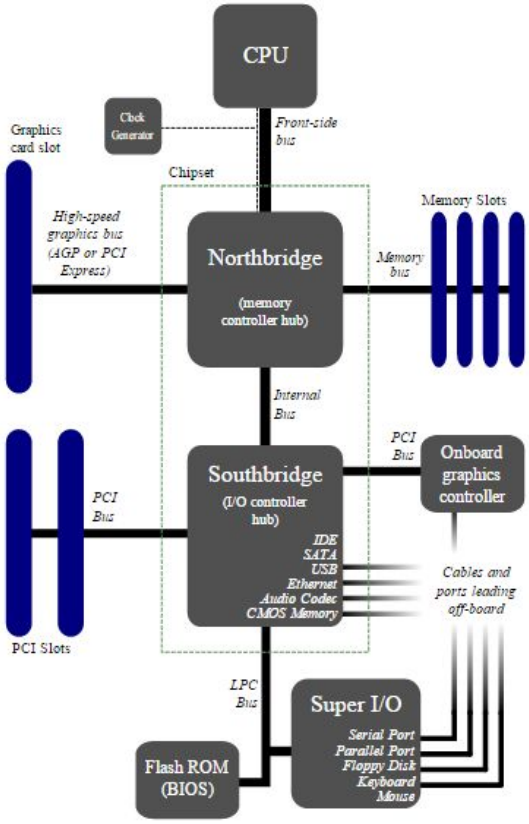
- Integrated
- Limit CPU Speed Evolution
- Configure using Jumpers:
  - Memory space
  - Interrupt Number and priority
- Good for embedded?



Jump high: Computer Buses

Gen2: Internal/External

- Decoupled fast CPUs
- AGP: Graphics cards
- Plug-n-Play
- 8bit → 64-bit PCI
- Multi-Gbps as in PCIe

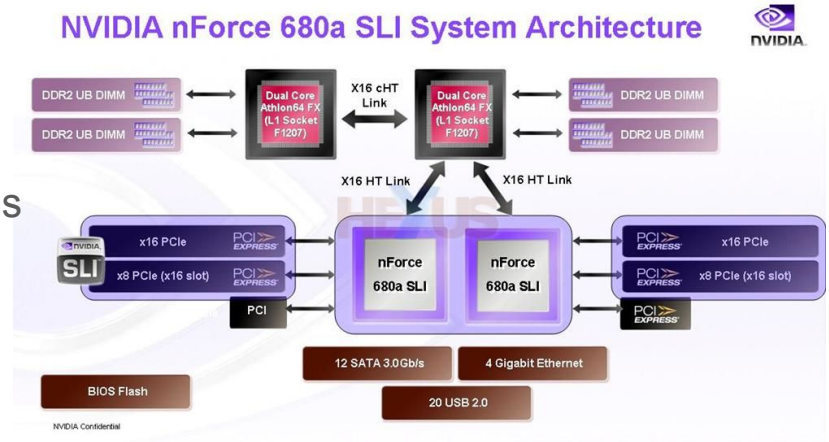


Valeo

Jump high: Computer Buses

Gen3: Bus of Everything

- Emerged since 2001
- HyperTransport, InfiniBand
- Physical form can be flexible
- Packet oriented, SW overheads
- Replaces:
  - Front-side bus
  - Multi-core interconnect
  - Co-processor interconnect
  - Pluggable Cards slots
  - Routers for networking!



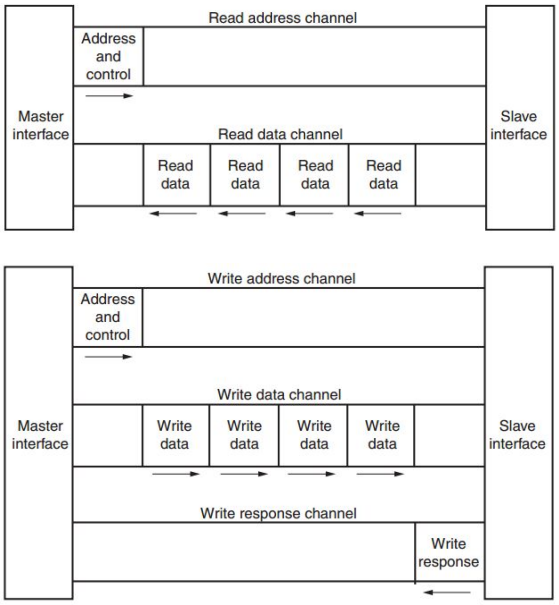
Valeo

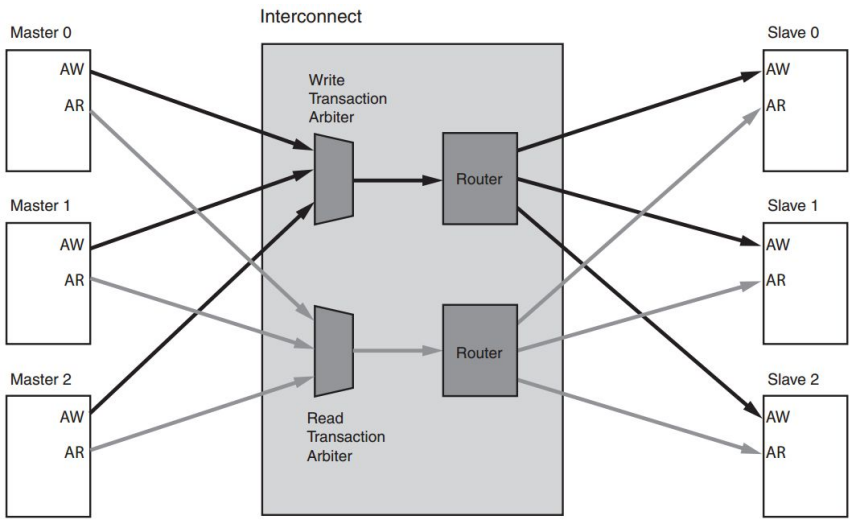
AXI

Advanced Microcontroller Bus Architecture (AMBA)

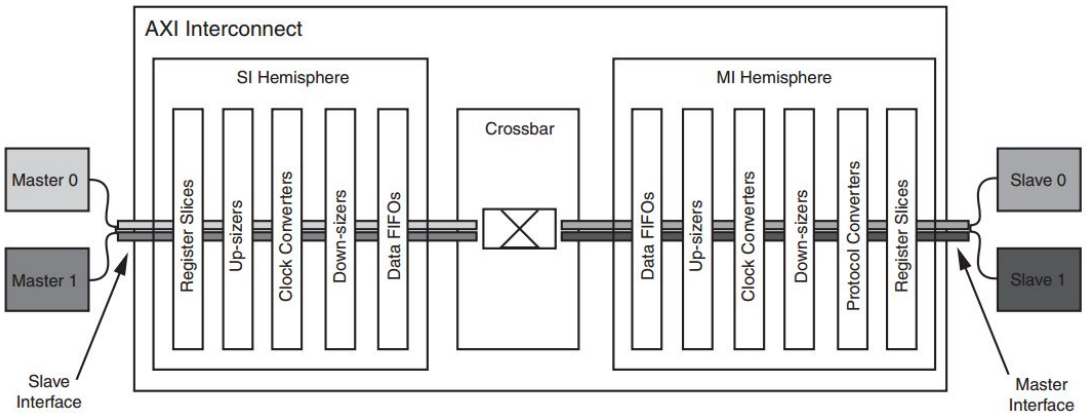
- open-standard,
- on-chip interconnect specification
- used on a range of ASIC and SoC
- introduced by ARM in 1996: ASB, APB, AHB
- In 2003, ARM introduced the third generation, AMBA 3, including AXI to reach even higher performance interconnect
- In 2010 the AMBA 4 with **AXI4**, widely used
- In 2013 the AMBA 5 CHI (Coherent Hub Interface)

AXI4: Master/Slave (5 Channels)





AXI4

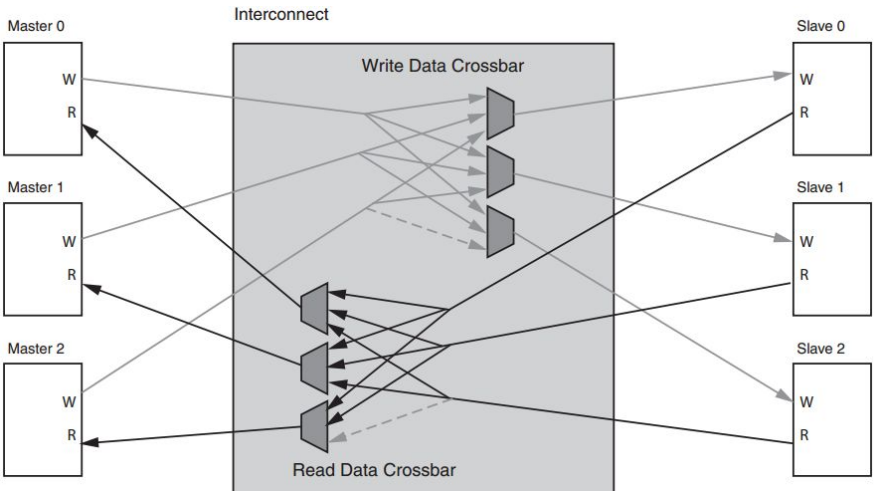


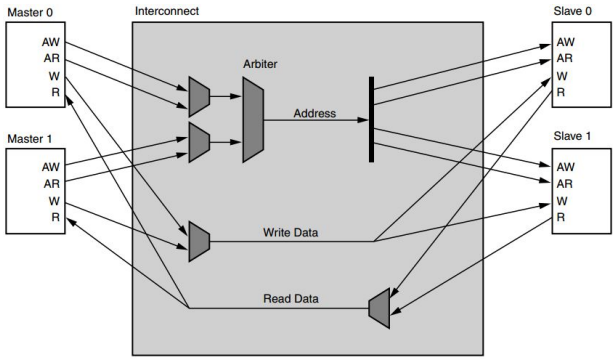
- AXI4— for memory mapped interfaces and allows burst of up to 256 data transfer cycles with just a single address phase
- AXI4-Lite— lightweight, single transaction memory mapped interface
- AXI4-Stream— Unidirectional P2P allows unlimited data burst size for high-speed streaming (ex. video)

AXI4

- Connects 1-16 masters to 1-16 slaves
- All Address channels are 32-bits
- Data channels:
  - AXI4— data channels ranges from 32, 64, 128, 256, 512 or 1024 bit wide
  - AXI4-Lite— data channels are 32-bit only
  - AXI4-Stream— no address channel, data channels is as the AXI4
- Interconnect:
  - Shared bus [best Area]
  - Crossbar switch [best Performance]

AXI4: Crossbar Interconnect





AXI4: Advanced Features

- Multiple outstanding transactions:
  - multiple reordering
  - Configurable write and read transaction acceptance limits for each connected master/slave
- “Single-Slave per ID” method of cyclic dependency (deadlock) avoidance
- Fixed priority and round-robin arbitration:
  - 16 configurable levels of static priority
  - Round-robin arbitration is used among all connected masters
- Supports TrustZone security for each connected slave
- Support for Read-only and write-only masters and slaves

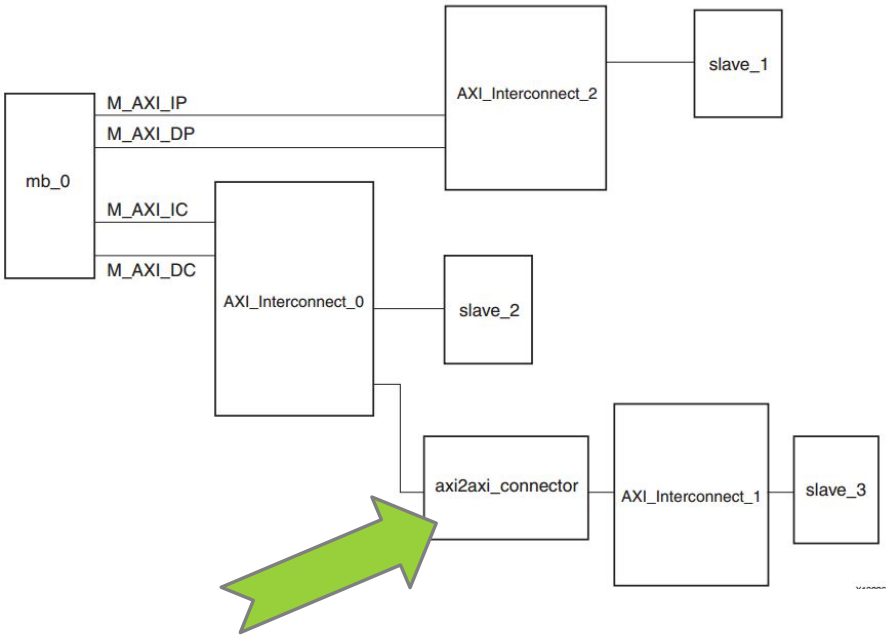
AXI4: Limitations

- No support for low-power mode
  - not clock-enable [Not std, Xilinx implementation: `ACLKEN`]
- AXI Interconnect core does not timeout if the slave doesn't respond
- no address remapping

AXI4: More slaves

AXI 2 AXI Connector

- Routes Master Interface Signals
- No logic
- No storage/buffering



AXI4

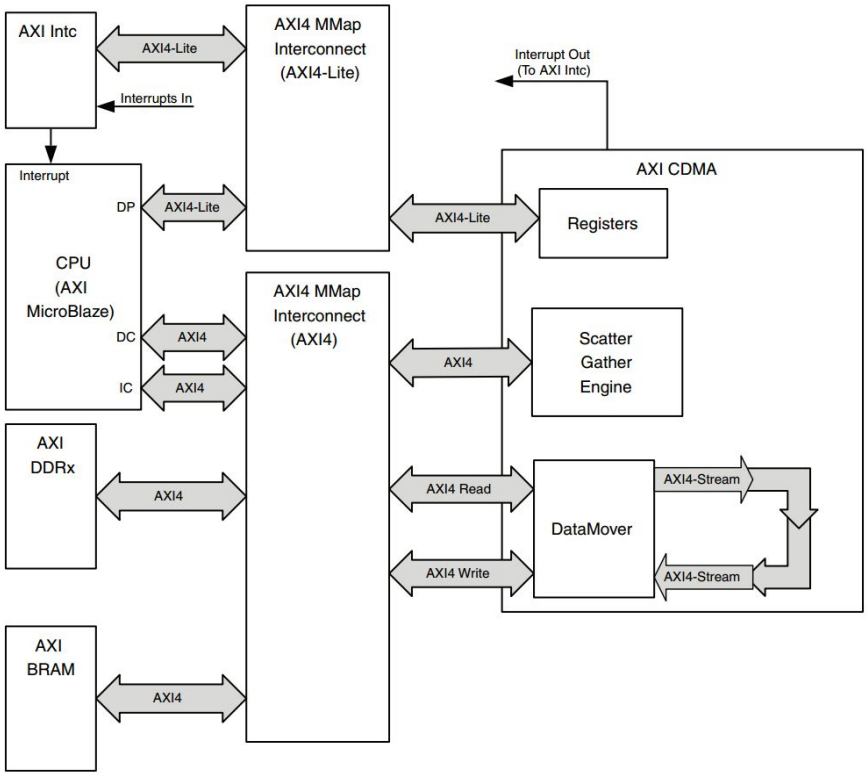
Infrastructure IPs

- AXI FIFOs (for buffering/clock conversion)
- AXI Interconnect IP (connects memory mapped IP together)
- AXI Direct Memory Access (DMA) engines
  - Ethernet DMA
  - Video DMA

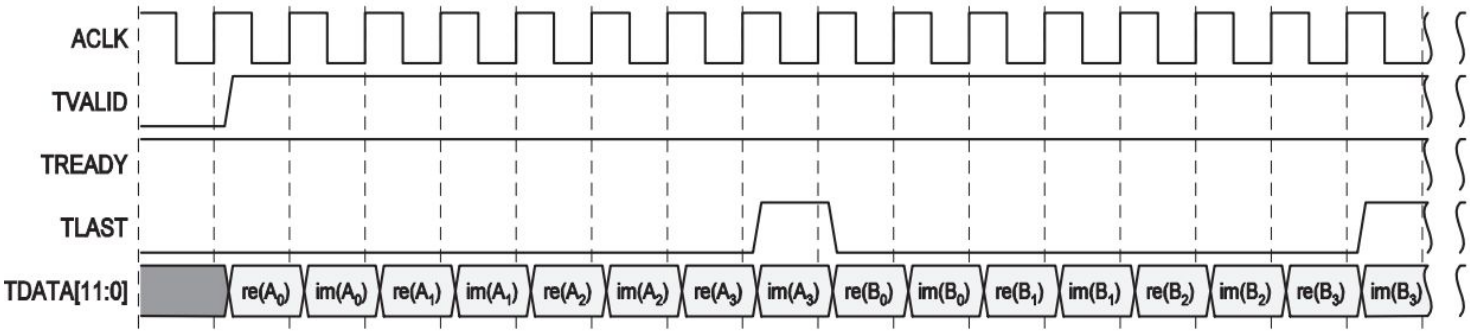
AXI4: CDMA

Central DMA

- Function:
  - Gets Configuration
  - Transfer Burst
  - Send Interrupt on finish
- Data Realignment Engine



AXI4 Stream



Flow:

1. Receiver asserts TREADY
2. Sender asserts TVALID and sends TDATA
3. According to application, sender may assert TLAST

Lab #3

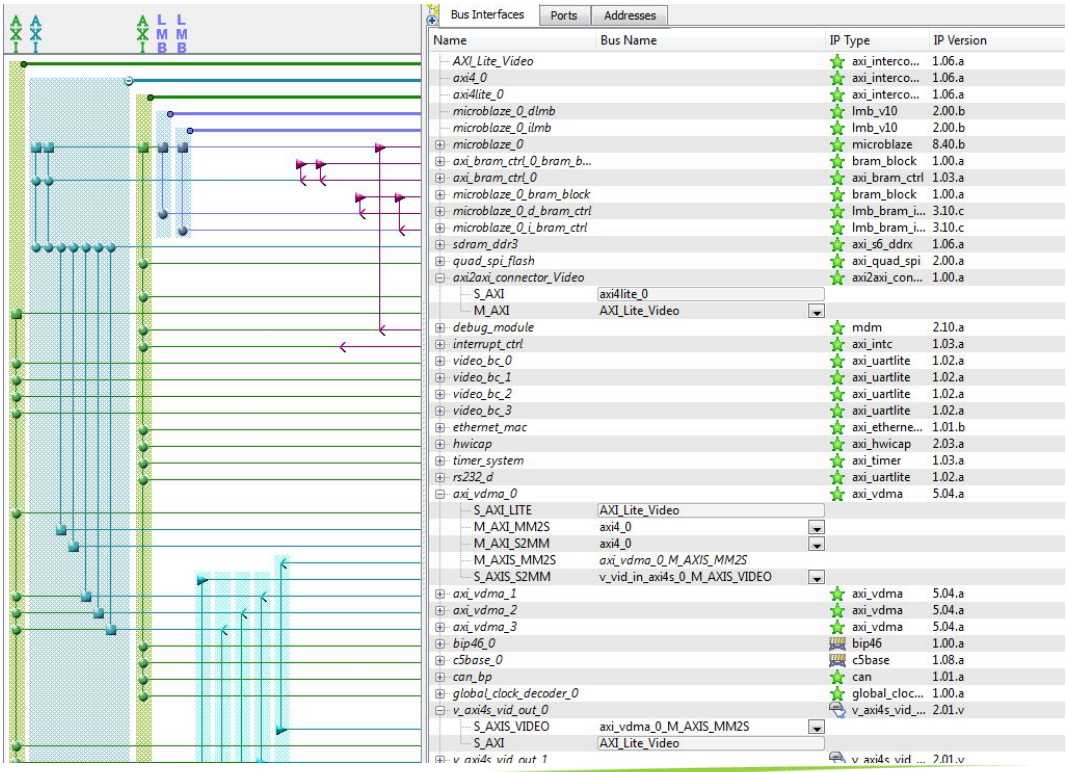
Rebuild the XPS Project :-)

AXI4: Quad Camera Simulator

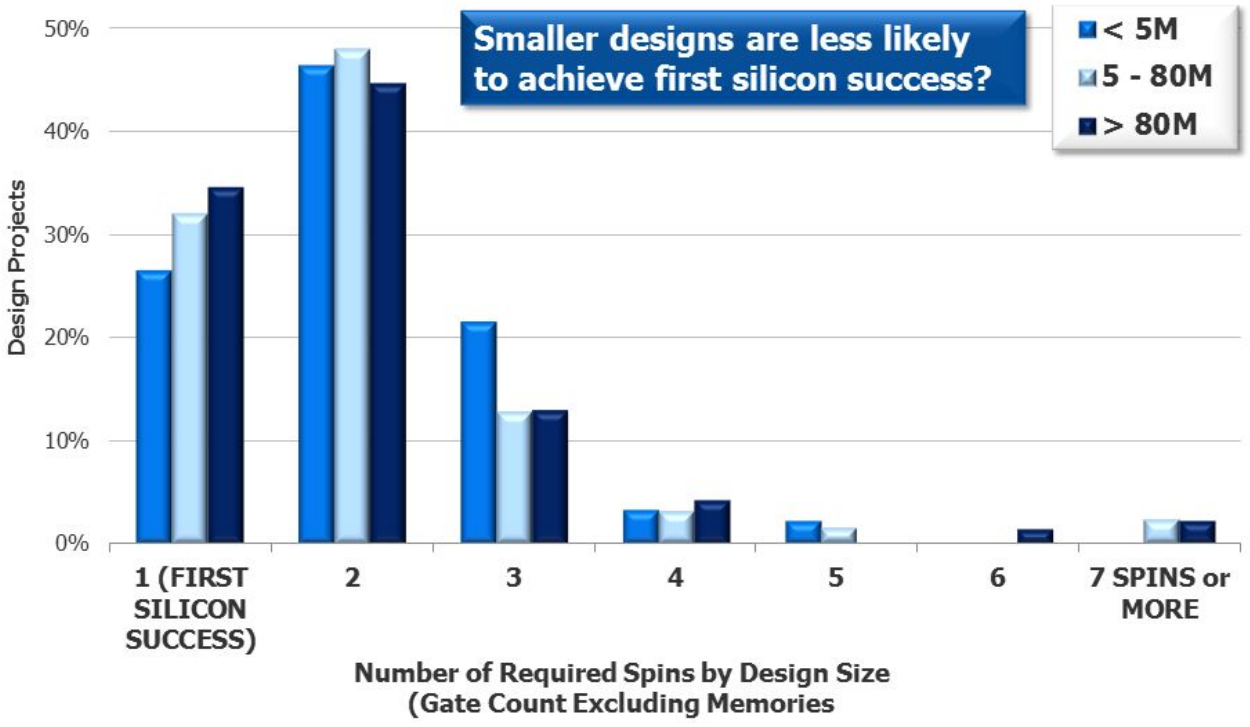
1x AXI

2x AXI-Lite

5x AXI-Stream



VHDL Verification



Source: Wilson Research Group and Mentor Graphics, 2014Functional Verification Study

VHDL Verification

	VHDL Simple	VHDL Advanced	UVM
Method	RTL style using processes	Records for GPIO, ADC, UART transactions. Procedures for stimulus and checking	UVM agents for GPIO, ADC, and UART interfaces. Built environment and isolated sequences for stimulus generation
Randomized Stimulus	No	No	Yes. ADC and GPIO randomized with constraints
Reusable	No	Yes, by creating packages. Connecting to user testbench is not intuitive	Yes. OOP features allow reuse
Code Coverage	78.59%	74.41%	79.71% with 1st test 85.58% with 2nd test

VHDL Verification

```
use std.textio.all;
```

```
PROCESS
  FILE vector_file: text OPEN read mode IS "test vec.txt"; -- declare and open file (1993 style)
  VARIABLE file_line           : line; -- text line buffer
  VARIABLE str_stimulus_in : string(11 DOWNTO 1);
  VARIABLE stimulus_in       : std_logic_vector(10 DOWNTO 0);
  VARIABLE y_expected       : std_logic_vector(7 DOWNTO 0);
BEGIN
WHILE NOT endfile(vector_file) LOOP  -- loop through lines in test file
  readline(vector_file, file_line); -- read one complete line into file_line
  read(file_line, str_stimulus_in); -- extract the first field from file line
  stimulus_in := str2vec(str_stimulus_in); -- convert string to vector
  WAIT UNTIL clk = '1'; -- rising edge of clock
  sel <= stimulus_in(10 DOWNTO 8); -- top 3 bits for input;
  y_expected := stimulus_in(7 DOWNTO 0); -- expected output
  WAIT UNTIL clk = '0'; -- falling edge
  -- now check if decoder outputs are correct
  IF y /= y_expected THEN
    ASSERT false
    REPORT "decoder failure" & CR & LF &
          "Expected y to be " & vec2str(y_expected) &
          " but its value was " & vec2str(y)
          SEVERITY ERROR;
  END IF;
END LOOP;
WAIT; -- suspend the simulation END PROCESS;
```

Multiple arguments may be parsed from a single line.

Tip: Add spaces for better visibility, and account for it when parsing.

Lab #4

Rewrite TB from Lab #1 with:

- Package
- Records
- Procedures
- FileIO
  - 5 Lines with inputs are reference result
- Rerun with post PAR model

Workshop / Project [To be defined per group]

- System Analysis
- Requirement Definition
- Unit Requirements (Block Diagram / Flow chart)
- Implement
- Integrate
- Validate

Questions. Discussions. Feedback