A Beginner's Guide to
Field Programmable Gate Arrays

FPGA/VHDL WORKSHOP I

Brief History of FPGAs

- Progression of PROMs & PLDs
- late 1980s
  - Naval Surface Warfare Department funded an experiment
- 1985 – Xilinx
  - co-founders Ross Freeman and Bernard Vonderschmitt invented the first commercially viable FPGA
- early 1990s
  - Primarily used in telecommunications and networking
- 2000s
  - consumer, automotive, and industrial applications

PLD vs. Processers

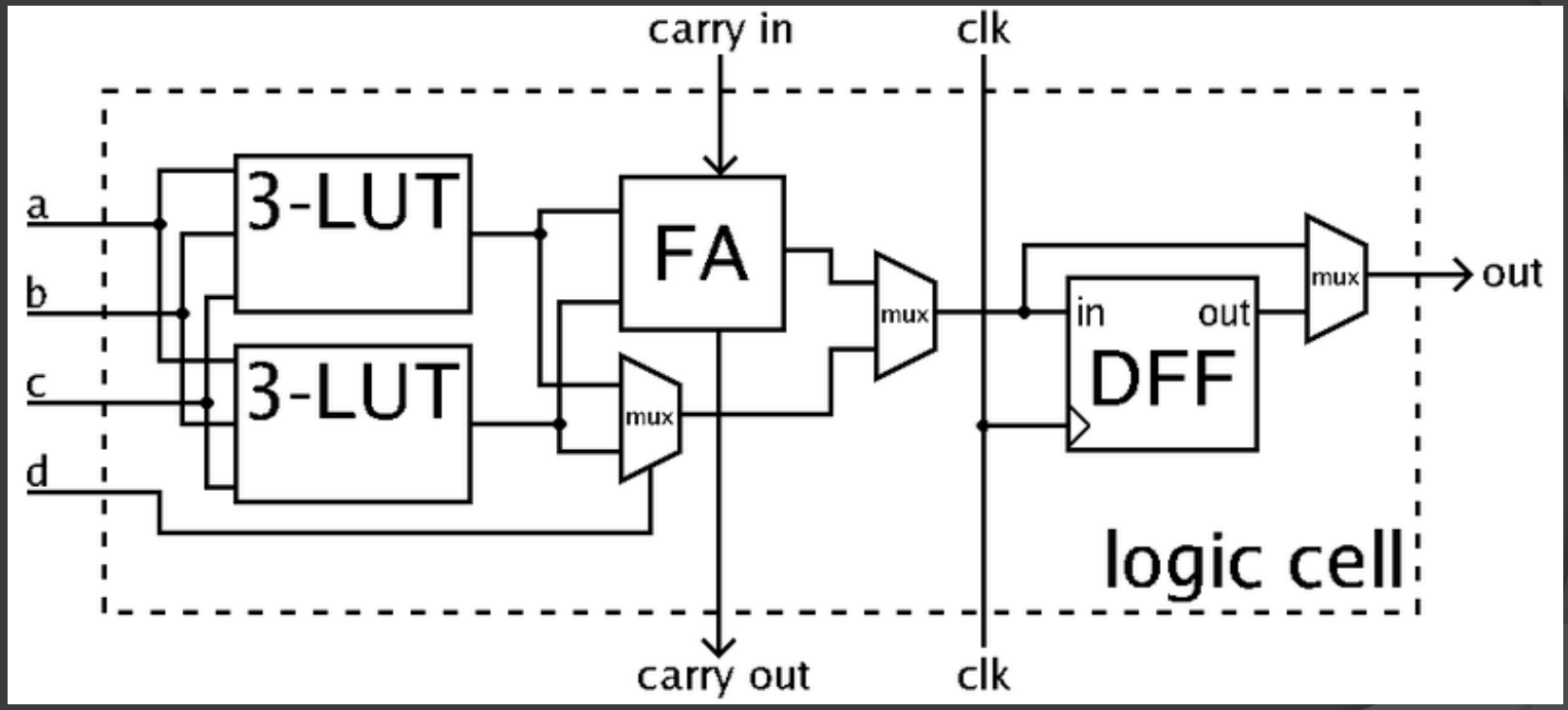
Processer

- Instructions
- Interrupts

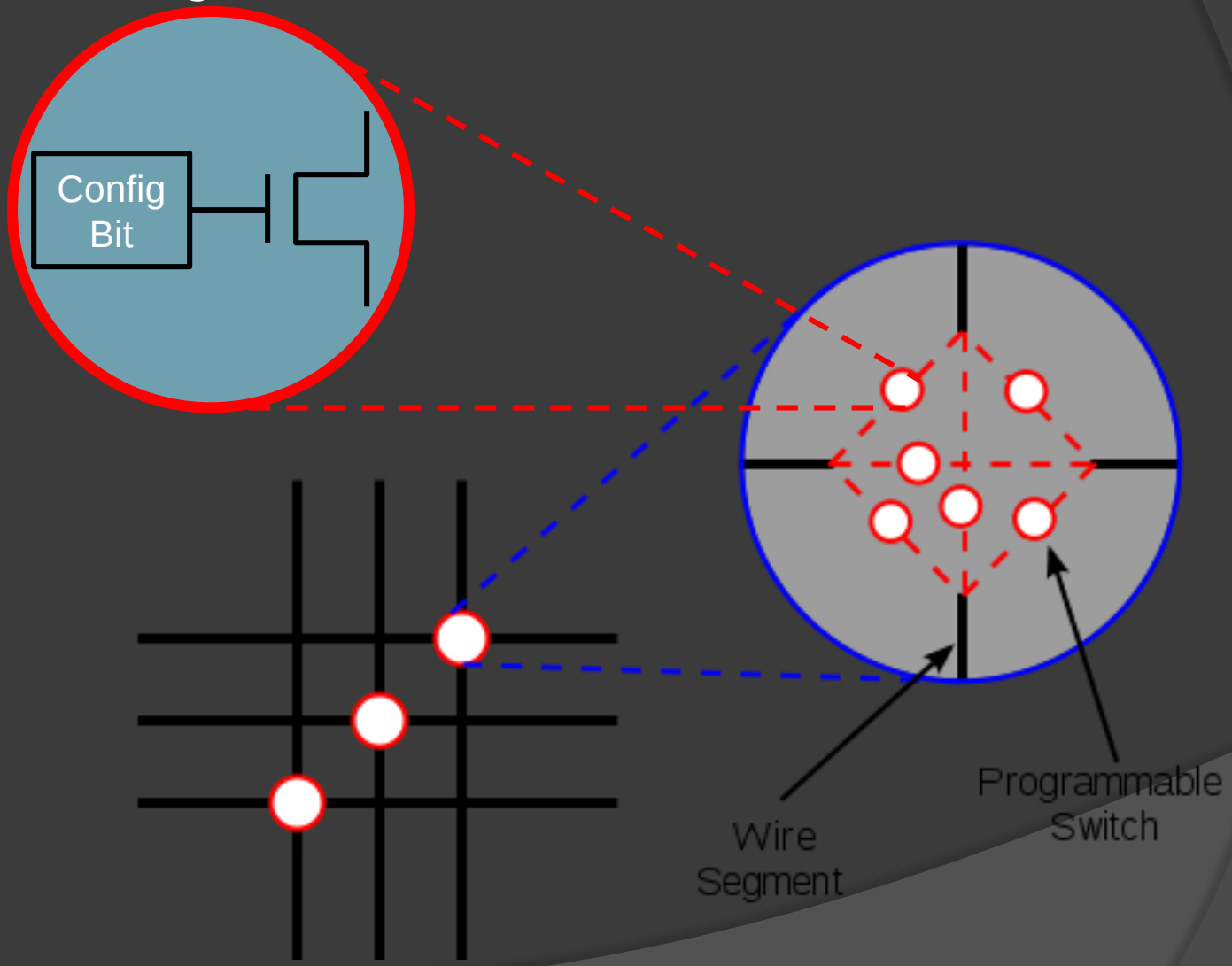
PLD

- “Physical” Hardware

FPGA Layout – CLBs



FPGA Layout – PIPs



The Language of Hardware

- HDL – Hardware Descriptive Language
  - VERILOG
  - VHDL – VHSIC Hardware Descriptive Language

In this workshop, we will focus on VHDL.

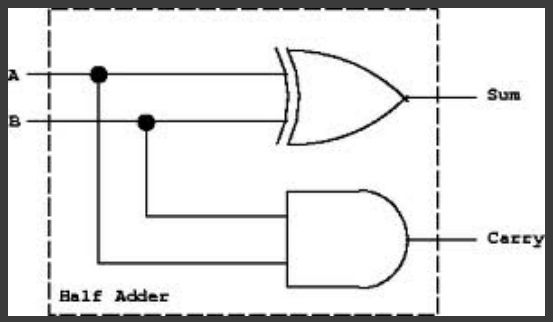
Thinking Logically

- When writing VHDL code you need to ask the question

"How would this be done with logic gate?"

Assume 1-bit addition

$$\begin{array}{rr} & A \\ + & B \\ \hline & SUM \end{array}$$



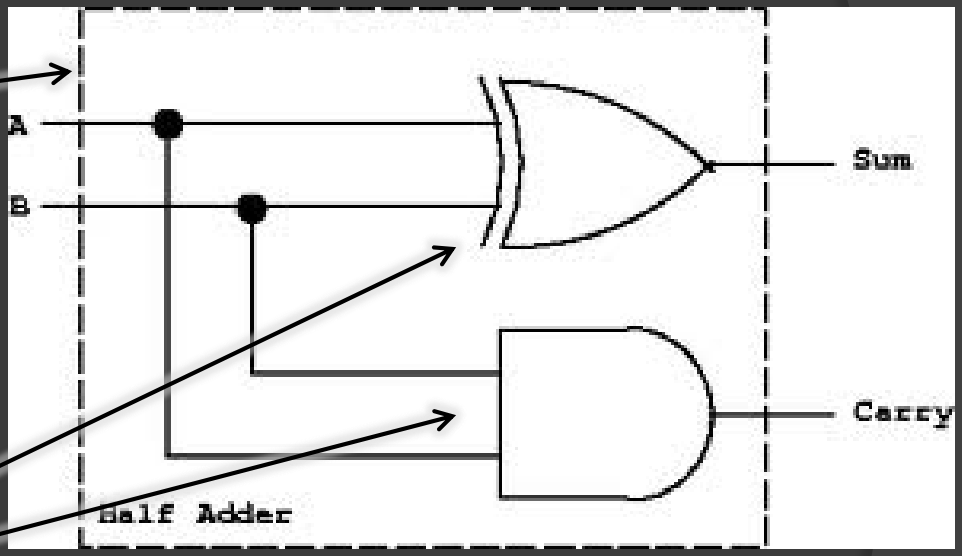
$SUM = A + B;$ $SUM <= A\ xor\ B;$

Diving into VHDL

```vhdl
-- import std_logic from the IEEE library
library IEEE;
use IEEE.std_logic_1164.all;

-- this is the entity
entity HALFADDER is
  port (
    A       : in  std_logic;
    B       : in  std_logic;
    SUM     : out std_logic;
    CARRY   : out std_logic);
end entity HALFADDER;

-- this is the architecture
architecture RTL of HALFADDER is
begin
  SUM   <= A xor B;
  CARRY <= A and B;
end architecture RTL;
```



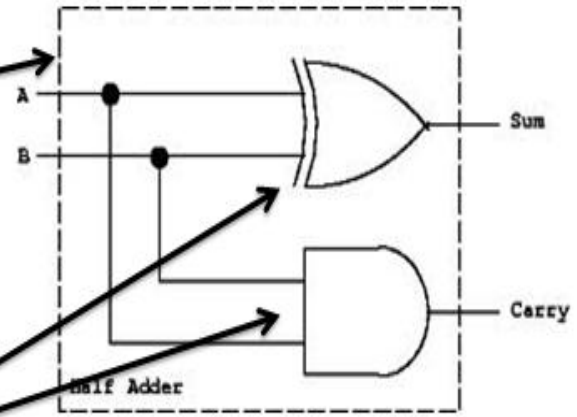
Diving into VHDL

- Library
  - Interpreter of syntax
- Entity
  - Define inputs and outputs
- Architecture
  - How the inputs affect the outputs

```vhdl
-- import std_logic from the IEEE library
library IEEE;
use IEEE.std_logic_1164.all;

-- this is the entity
entity HALFADDER is
  port (
    I1      : in  std_logic;
    I2      : in  std_logic;
    SUM     : out std_logic;
    CARRY   : out std_logic);
end entity HALFADDER;

-- this is the architecture
architecture RTL of HALFADDER is
begin
  SUM   <= I1 xor I2;
  CARRY <= I1 and I2;
end architecture RTL;
```



Libraries

- IEEE.STD_LOGIC_1164.ALL;

Entities

- A : in std_logic;
  - Signal Name
    - Alphanumeric or ‘_’
    - Must start with a letter
    - No spaces or double ‘_’
    - Not case sensitive

Entities

- I1 : in std_logic;
  - Direction
    - In – Signals into the entity
    - Out – Signals out of the entity
    - Inout – Bidirectional Signals
    - Buffer – Signals out of the entity and internally referenced

Entities

- I1 : in std_logic;
  - Signal Type
    - std_logic / std_logic_vector
      - 0,1,L,H
      - Z, –
    - boolean
      - TRUE,FALSE
    - bit / bit_vector
      - 0,1
    - std_ulogic / std_ulogic_vector
      - U,X,0,1,Z,W,L,H,X, –

Architecture

- Behavioral
  - Common Architecture
  - Algorithmic / Sequential
- Dataflow
  - How data is transferred from signal to signal and input to output
  - No sequential statements
- Structural
  - Primary use
    - connect entities in hierarchical design

Behavioral

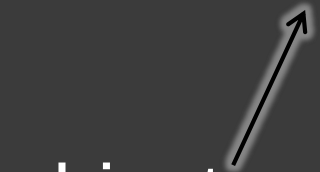
- Constants
  - Used for improved readability
- Signals
  - Think of wires
  - Sequential Statements
    - Assignment on next cycle
- Variables
  - Assignments are immediate
- Processes
  - contains sequential statements

Behavioral – Constants

```
constant Stop_All_CRC : STD_LOGIC_VECTOR :="11000111";
```

Behavioral – Signals

```
signal Custom_Raw          : STD_LOGIC_VECTOR (383 downto 0);
```



object name

object type

Behavioral – Variables

```
variable CUS_Cur_State          : STD_LOGIC;
```

object name

object type

Assignment Symbol

```
CUS_Cur_State :='0';
```

Behavioral – Processes

```
process_name : process(SYS_CLK)
    begin
    .
    .
    .
end process process_name;
```

```
if Logical_Test then
    statements;
elsif Logical_Test then
    statements;
else
    statements;
end if;
```

```
case STATE is
    when STATE1 =>
        statements;
    when STATE2 =>
        statements;
    .
    .
    .
    when others =>
        statements;
end case;
```

Netlist

- Used by the compiler to map
  Signal Names to Physical I/O

Our Platform

- Basys™2 Spartan-3E FPGA Board
  - 100K gate equivalent

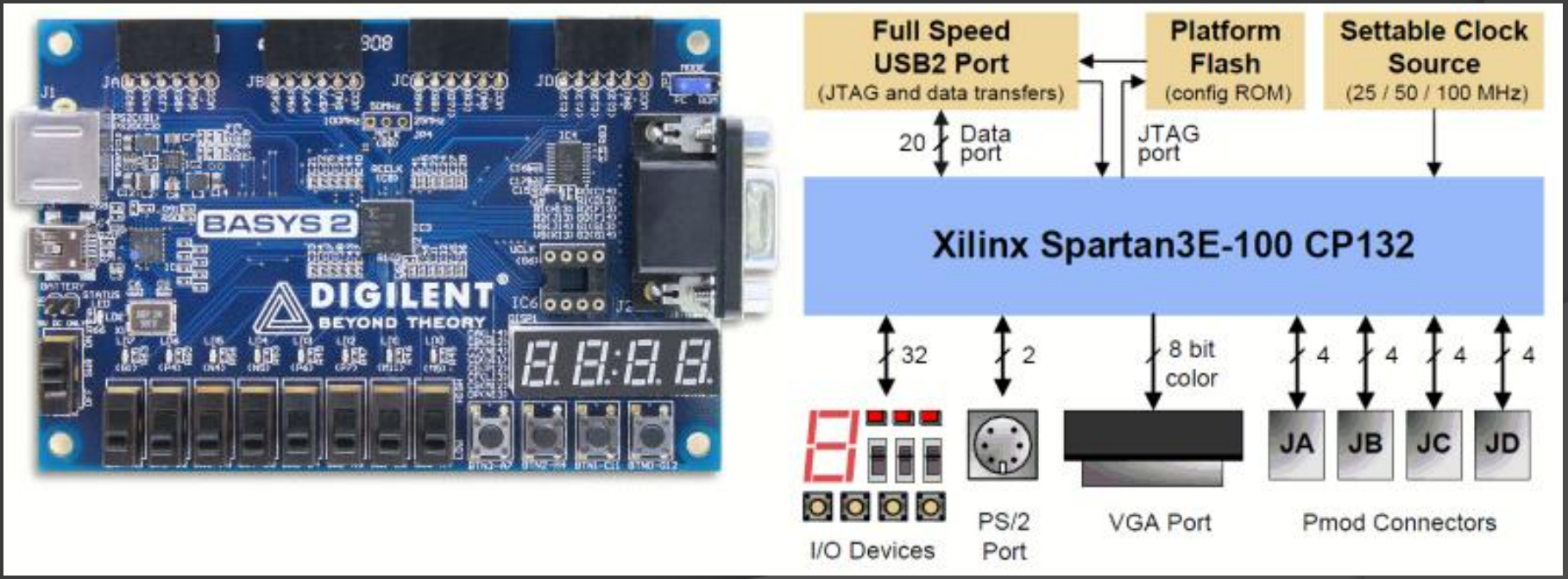


Image Source: digilentinc.com

First Design

- BLINKY – Flash a LED at a desired rate
- Components needed
  - Frequency Divider

First Design – BLINKY

```
entity ClockDivide is
        Port (
        clk_in : in  STD_LOGIC;
        reset  : in  STD_LOGIC;
        clk_out: out STD_LOGIC
        );
end ClockDivide ;
```

First Design – BLINKY

```
architecture Behavioral of ClockDivide is
        signal temp: STD_LOGIC;
        signal counter : integer range 0 to 124999 := 0;
begin
        frequency_divider: process (reset, clk_in)
        begin
        if (reset = '1') then
                temp <= '0';
                counter <= 0;
        elsif rising_edge(clk_in) then
                if (counter = 124999) then
                        temp <= NOT(temp);
                        counter <= 0;
                else
                        counter <= counter + 1;
                end if;
        end if;
        end process;
        clk_out <= temp;
end Behavioral;
```

Second Design

- Finite State Machine (FSM)
  - Used for data flow control
- Components needed
  - Case statements (FSM)
    - State Map
    - State Outputs

Resources

- VHDL Libraries - functions defined
  - http://www.csee.umbc.edu/portal/help/VHDL/stdpkg.html
- Digilent Classroom
  - http://www.digilentinc.com/NavTop/Classroom.cfm