Digital System Design

Mathias Balling & Mads Thede

Last updated: June 25, 2024

Contents

1 Intergrated Circuits

An Intergrated Circuit (IC) is a micro chip that can function as an amplifier, oscillator, timer, microprocessor, or computer memory. IC can hold anywhere from hundreds to millions of transistors, resistors and capacitors. Embedded system is a combination of computer hardware and software designed for functions within a larger system or for a specific function.

Abstraction when designing a digital system is useful for understanding the system.

1.1 Moores law

Moores law states that the number of transistors on a microchip doubles every two years. This correlates to the transistor size halving every two years.

2 VHDL

Very High Speed Integrated Circuit Hardware Description Language. It is a programming language that has been designed and optimized for describing the behavior of digital systems.

2.1 VHDL Syntax

VHDL is not case sensitive, but it is a good practice to use upper case for keywords and lower case for everything else. VHDL describes hardware and so instructions are executed in a concurrent manner, meaning that all instructions are executed at once.

Comments are written with two hyphens (- -).

if, case and loop statements

Every if statement has a corresponding then component and each if statement is terminated with an end if; If an else if statement is needed then it can be written as elsif. Before each if statement a process statement must be written. The process statement includes the sensitivity list which is a list of signals that the process is sensitive to. Once one of the signals in the sensitivity list changes value the process is executed.

```
process (sel, I)
begin
  if (sel == "00") then
    a <= 'I';
    b <= '0';
    c <= '0';
    d <= '0';

  elsif (sel == "01") then
    a <= '0';
    b <= 'I';
    c <= '0';
    d <= '0';
  elsif (sel == "10") then
    a <= '0';
    b <= '0';
    c <= 'I';
    d <= '0';
  else
    a <= '0';
    b <= '0';
```

```
    c <= '0';
    d <= 'I';
  end if;
end process;
```

Each case statement is terminated with end case; Each loop statement has a corresponding end loop;

```
case (expression) is
  when choices =>
    <sequential statements>
  when choices =>
    <sequential statements>
  when others =>
    <sequential statements>
end case;
```

Signal and variable assignments

Among the most frequently used object types there is the signal object type, the variable object type, and the constant object type. The signal type is the software representation of a wire in the digital system. The variable type is used to store local information. The constant type is used to store constant information.

Signals can be used inside or outside processes. They take the value depend on if the signal is used in combinational or sequential code. In combinational code, signals immediately take the value of their assignment. In sequential code, signals do not immdeiately take the value of their assignment.

Signals are declared at the top of the architecture body, just before the keyword begin. Variables must be declared inside the process construct but before the begin and are local. Thus, variables can only be used inside processes.

Assigning a value to a signal is done using the signal assignment operator <=.

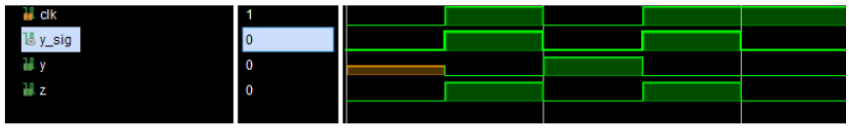
$$C <= not(A);$$

Assigning a value to a variable is done using the variable assignment operator :=.

$$B := A;$$

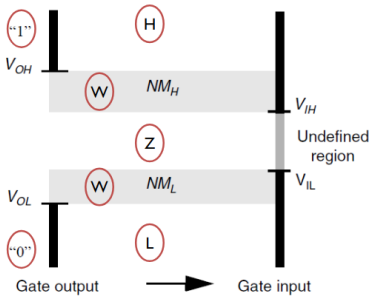
A variable changes its value immediately after the variable assignment is executed. Instead a signal changes its value "some time" after the signal assignment expression is evaluated.

```
entity signalvsVariable is
    Port ( clk : in STD_LOGIC;
           y : out STD_LOGIC;
           z : out STD_LOGIC);
end signalvsVariable;

architecture Behavioral of signalvsVariable is
    signal y_sig: std_logic;
begin
    process(clk)
        variable z_var: std_logic;
        begin
        if(rising_edge(clk)) then
        y_sig <='1';
        z_var :='1';
        else
        y_sig <='0';
        z_var :='0';
        end if;
        y <= y_sig;
        z <= z_var;
    end process;

end Behavioral;
```



2.2 VHDL Design Units

STD LOGIC



Character	Value
'U'	uninitialized
'X'	strong drive, unknown logic value
'0'	strong drive, logic zero
'1'	strong drive, logic one
'Z'	high impedance
'W'	weak drive, unknown logic value
'L'	weak drive, logic zero
'H'	weak drive, logic one
'-'	don't care

The noise margins indicate how much noise can be added to the logic level and still have the logic level recognized as the same.

Entity

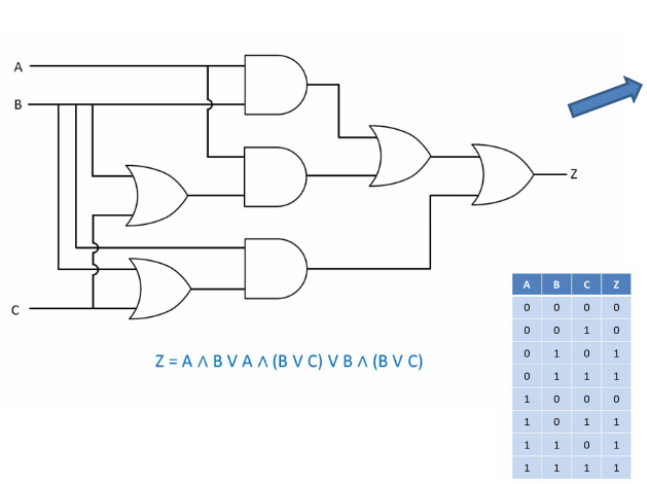
The entity describes how the unit interfaces with the outside world. The entity lists the various inputs and outputs of the underlying system.

```
entity my_entity is
port(
     port_name_1 : in    std_logic ;
     port_name_2 : out   std_logic;
     port_name_3 : inout std_logic ); --do not forget the semicolon
end my_entity; -- do not forget this semicolon either
```

Architecture

Describes what the circuit actually does. It describes the internal implementation of the associated entity. An architecture can be written by means of three modeling techniques plus any combination of these three: dataflow, behavioral, and structura plus any combination of these three.

Complex logig circuit



A	B	C	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$Z = A \wedge B \vee A \wedge (B \vee C) \vee B \wedge (B \vee C)$

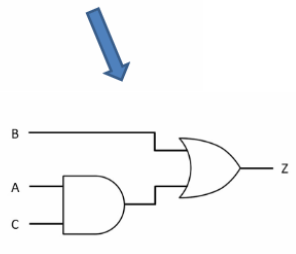
$= A \wedge B \vee (A \wedge B) \vee (A \wedge C) \vee (B \wedge B) \vee (B \wedge C)$

$= A \wedge B \vee (A \wedge B) \vee (A \wedge C) \vee B \vee (B \wedge C)$

$= (A \wedge B) \vee (A \wedge C) \vee B \vee (B \wedge C)$

$= (A \wedge B) \vee (A \wedge C) \vee B$

$= (A \wedge C) \vee B$



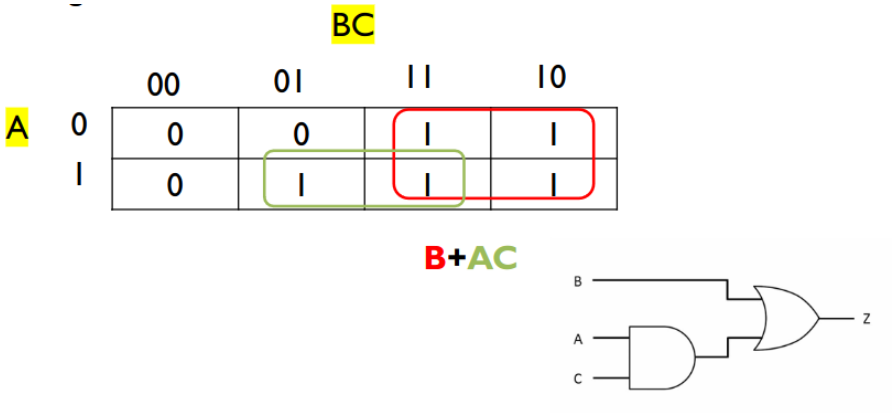
Laws of Boolean Algebra

A + corresponds to an OR gate. An · corresponds to an AND gate. And a ′ corresponds to a NOT gate.

In digital electronics, Boolean Algebra is used to simplify the design of complex circuits.

Karnaugh map (K-map)

A method of circuit optimization. Aims to reduce logic functions more quickly and easily compared to Boolean algebra.



Combinational circuits

It is defined as the time independent circuits which do not depend upon previous inputs to generate any output. e.g. Encoder, Decoder, Multiplexer, Demultiplexer etc.

Sequential circuits

Dependent on clock cycles and depends on present as well as past inputs to generate any output. e.g. Flip-flops, Counters, Registers etc.

Concurrent Statements CHDL has the ability to execute a virtually unlimited number of statements at the same time and in a concurrent manner. The key thing to remember is that we are designing hardware.

Conditional Signal Assignment when

The term conditional assignment is used to describe statements that have only one target but can have more than one associated expression assigned to the target.

The individual conditions are evaluated sequentially in the conditional signal assignment statement until the first condition evaluates as true.

Conditional signal assignment is a concurrent statement that assigns a value to a signal based on the value of a condition. The syntax is:

```
<target> <= <expression> when <condition> else
        <expression> when <condition> else
        <expression>;
```

Selected Signal Assignment with select

Selected signal assignments only have one assignment operator. Selected signal assignment differs from conditional assignment statements in that assignments are based upon the evaluation of one expression. The syntax is:

```
with <choose_expression> select
  <target> <= <expression> when <choices>,
              <expression> when <choices>
              <expression> when others;
```

The general form of the selected signal assignment statement is similar to switch statements as seen in algorithmic programming languages such as C.

Process Statements

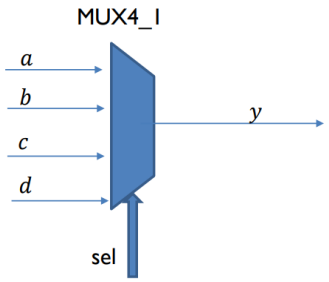
The process statement is a stement which contains a certain number of instructions that, when the process statement is executed, are executed sequentially. In other words the process statement is a tool that can be used for executing a certain number of instructions in a sequential manner.

The process statement is in itself a concurrent statement.

Multiplexer

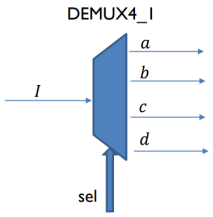
The function of a multiplexer is to select one of the inputs and connect it to the output.

A general multiplexer has n data inputs, one output, and m select inputs. The number of select inputs determines the number of data inputs. The number of data inputs is 2 to the power of the number of select inputs.



Demultiplexer

It selects one ouput from the multiple output line and feteches the single input through the selection line.



Encoders

Encoders are used to encode data into a coded form and decoders are used to convert it back into its original form. An encoder that has 2^n input lines and n output lines is called an $n - to - 2^n$ encoder. It is assumed that only one input has a value of 1 at any given time. It is used to minimize the number of input lines.

Priority encders are a modified version of an encoder. It gives a priority to an input signal and provides an output based on that priority.

Example of a 4-to-2 priority encoder:

```
entity Priority_Encoder is
  port (D: in std_logic_vector(3 downto 0);
        Y: out std_logic_vector(1 downto 0));
end Priority_Encoder;

architecture Behavioral of Priority_Encoder is
```

```
begin
  process (D)
  begin
    if (D(3) = '1') then
      Y <= "11";
    elsif (D(2) = '1') then
      Y <= "10";
    elsif (D(1) = '1') then
      Y <= "01";
    elsif (D(0) = '1') then
      Y <= "00";
    else
      Y <= "---";
    end if;
  end process;
end Behavioral;
```

Decoders

Decoders are used to decode data that has been encoded using a binary encoder. An n-bit code can represent 2^n different values. An encoder can be used with enable inputs. The enable inputs are used to enable or disable the decoder.

Latches

Level sensitive memory cell that is transparent to signals passing from the D input to Q output when enabled. It holds the value of D on Q at the time when it becomes disabled.

Latches with a clear input. The clear input means that when the clear signal goes to 1 the output is set to 0. The clear input is essentially the same as a reset signal.

D-type Flip-flops

Is an edge triggered memory device that transfers a signal's value on its D input to its Q output, when an active edge transition occurs on its clock input. The output value is held until the next active edge transition occurs.

Rising edge is preferred. Using an implemented function called rising_edge(clk) is more robust to noise compared to using clk'event and clk = '1'.

If you want to use flip-flops with reset it is good practice to use asynchronous reset. An example of asynchronous reset is shown below on the left, and a syncrhonous reset is shown on the right.

```
entity d_flip_flop is
  port (d, clk, rst: in std_logic;
        q: out std_logic);
end entity d_flip_flop;

architecture first of d_flip_flop
is begin
  process (rst, clk)
  begin
    if (rst = '1') then
      q <= '0';
    elsif (rising_edge(clk)) then
      q <= d;
    end if;
  end process;
end architecture first;
```

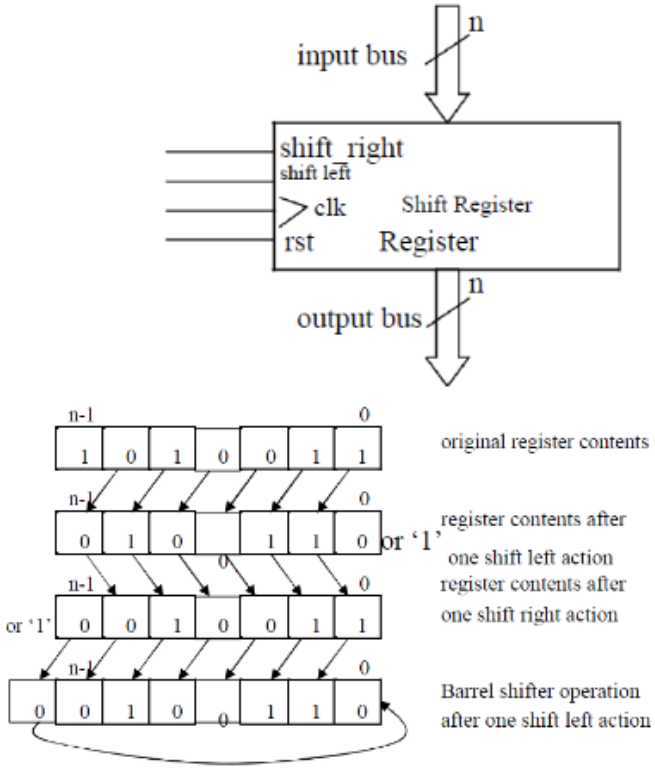
```
architecture second of d_flip_flop
is begin
  process (clk)
  begin
    if (rising_edge(clk)) then
      if (rst = '1') then
        q <= '0';
      else
        q <= d;
      end if;
    end if;
  end process;
end architecture second;
```

Registers

A parallel register is simply a bank of D-type flip-flops. Its implementation has the same structure as for one flip-flop but with extended inputs and outputs.

Shift registers

Registers with added features that enable them to shift their contents in either directions.



Keypad Encoder

A device that acts as the link between a keypad and a digital device. Its primary function is to provide a digital output equal to the key that is pressed on the keypad.

Comparators

Compares two or more inputs using one, or a number of different comparisons. When the given relationship(s) is true, an output signal is given logic 1, otherwise the output signal is given logic 0.

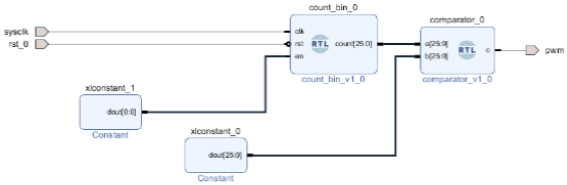
Only two data objects can be compared at once. Statements like... if (a = b = c) then... are not allowed in VHDL.

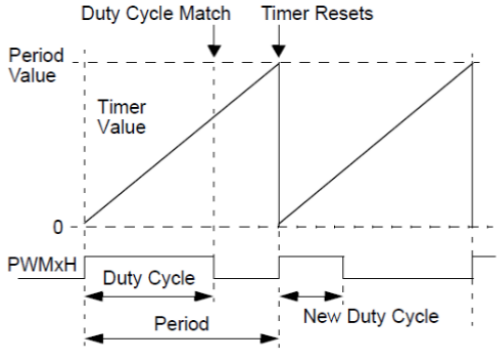
Comparators are only modeled using the if statement with an else clause and no else-if clauses.

Remember that std_logic_vector does not indicate a value of the bus. To compare two buses, unsigned or signes types must be used for comparision.

PWM

PWM shows how much time the signal is high in an analog fashion. Duty cycle is used to describe how much of the time that the signal is on.

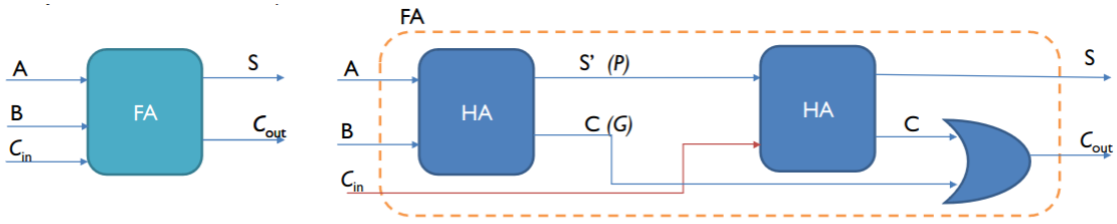




Calculating how many counter bits that are needed to generate a PWM signal that has a 1Hz frequency out of a 125MHz clock signal.

$$n = log_2(125 * 10^6)$$

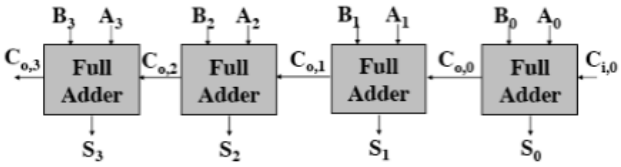
Adders



- $S = A \text{ XOR } B \text{ XOR } C_{in}$
- $C_{out} = (A \text{ AND } B) \text{ OR } (C_{in} \text{ AND } (A \text{ XOR } B))$

Ripple carry adder

An iterative array to perform binary addition, full adders are chained together.



Carry look-ahead adder

A hierarchical adder that is used to improve performance because it is faster than the ripple carry adder, but it takes more resources.

P = A XOR B. The same as sum for a half adder.

G = A AND B. The same as a carry for a half adder.

P and G can be calculated in advance since none of them depends on the carry bit (Cin).

$C_1=G_0+P_0C_0$

$C_2=G_1+P_1C_1=G_1+P_1(G_0+P_0C_0)$

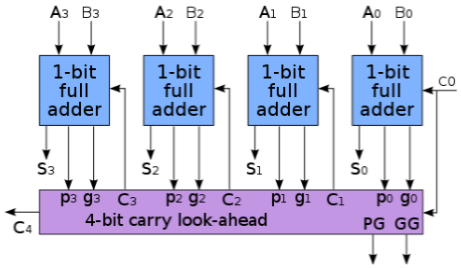
$=G_1+P_1G_0+P_1P_0C_0$

$C_3=G_2+P_2C_2=G_2+P_2(G_1+P_1G_0+P_1P_0C_0)$

$=G_2+P_2G_1+P_2P_1G_0+P_2P_1P_0C_0$

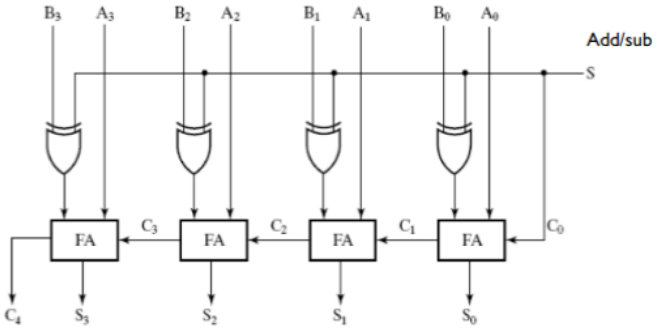
$C4=G_3+P_3C_3=G_3+P_3(G_2+P_2G_1+P_2P_1G_0+P_2P_1P_0C_0)$

$=G_3+P_3G_2+P_3P_2G_1+P_3P_2P_1G_0+P_3P_2P_1P_0C_0$



Subtractors

Subtraction can be done by adding 2's complement of the number. Example for A-B:



- For S=1, subtract, the 2's complement of B is formed by using XORs to form the 1's complement and adding the 1 applied to C0
- For S=0, B is passed through unchanged

Multiplication

It can be implemented by a shift register as wide as the product and an accumulator for the partial and final product.

Dividers

Division is the most complex of the four basic arithmetic operations. Hardware solutions are correspondingly larger and more complex than the solutions for other operations, it is best to minimize the number of divisions in any algorithm.

Division can be done using a clock divider, by subtraction and by multiplication.

$$A \cdot \frac{1}{B} = A \cdot (2^n/B)2^n$$

$$\frac{10}{4} = 10 \cdot \frac{2^2/4}{2^2} = 10\frac{1}{2^2} = 10.10$$

2.3 Standard models in VHDL Architecture

Data-flow style architecture (RTL)

Describes how data is transformed as it is passed from register to register. The transformation of data is performed by the combinational logic that exists between the registers. For example, logic gates, multiplexers, flip flops etc.

Behavioral style architecture

It provides no details as to how the design is implemented in actual hardware. The behavioral style models how the circuit outputs will react to the circuit inputs. Whereas in data-flow modeling you somewhat need to have a feel for the underlying logic in the circuit. In other words, data-flow modeling describes how the circuit should look in terms of logic gates whereas behavioral modeling describes how the circuit should behave.

Structural

Describes the interconnection of components within an architecture. For example block design.

Hybrid style

A combination of the previous design styles.

2.4 Simulation vs Synthesis

Simulation-only operations for error messages or reading files that can be simulated but not synthesized.

Constructs such as for loops, while loops, etc. are either un-synthesizable or (worse) synthesize poorly.

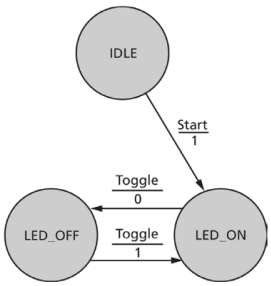
You need procedural code for testbench and only for testbench. Examples: AND gate is synthesizable but AND gate with 5ns wait is not. Printf is not synthesizable.

Use synthesizable code to describe the function of units that will be built into hardware. Use non-synthesizable code to create a testbench that checks to see if your synthesizable code does what you want.

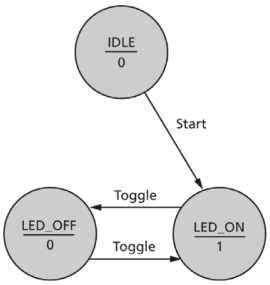
Synthesis is a process of converting high level FPGA logic design into gates.

3 Finite State Machines

FSM consists of states and transitions. State machiens are used for modeling systems that have a finite number of states and whose behavior is determined by the current state and the input. The states are drawn as circles and the transitions are drawn as arrows between the states. The transitions are labeled with the input that causes the transition. State machines can be made using two models: Mealy and Moore. In Mealy state machines, the output is a function of the current state and the input. When the input change, the outputs are updated without waiting for a clock edge. The mealy state machine is represented in the following figure:

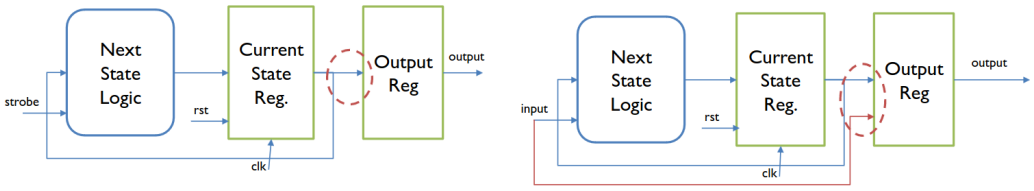


In Moore state the output is a function of the current state only. The outputs are only written when the state changes (on clock edge). The moore state machine is shown on the following figure:



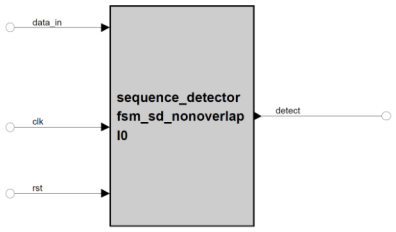
The hardware representation of a state machine can be visualized as:

Moore **Mealy**

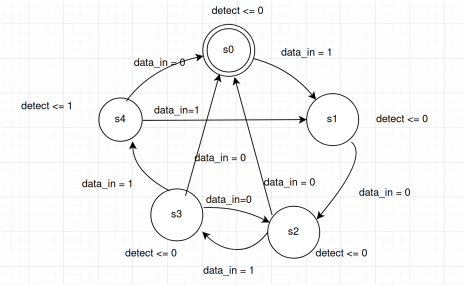


Sequence checker

Checks for the sequence 1011.



(a) Sequence checker FSM



(b) Sequence checker

Two types. Nonoverlapping sequence and overlapping sequence. With the overlapping sequence, the last 1 in the sequence is also the first 1 in the next sequence.

4 FPGA

Field Programmable Gate Arrays (FPGAs) are integrated circuits containing gate matrix which can be programmed by the user "in the field" without using expensive equipment.

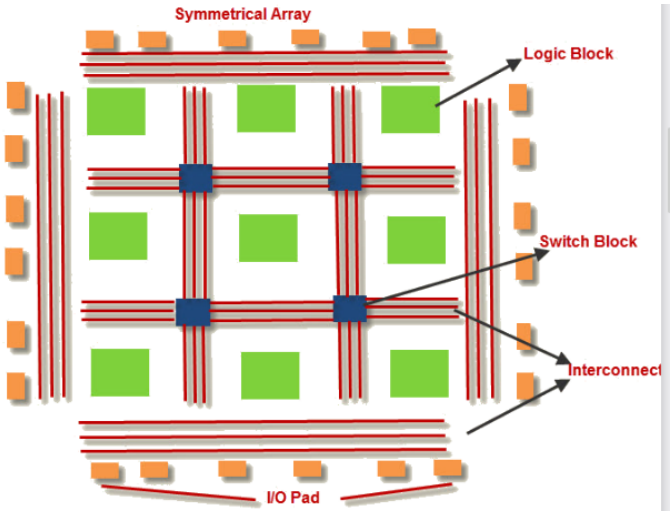
An FPGA contains a set of programmable logic gates and rich interconnect resources, making it possible to implement complex digital circuits.

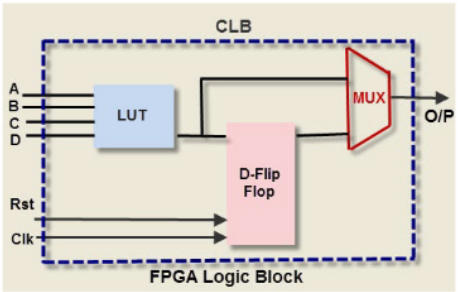
The majority of FPGAs are based on SRAM. (Static RAM).

FPGA mainly consists of three types of modules. I/O blocks, switch matrix/interconnection wires and configurable logic blocks (clb).

The I/O blocks are used to interface with the external world. This could be a sensor, a display or a communication device.

Configurable logic blocks consists of look-up tables (LUTs), flip flops and multiplexers. A LUT is a memory unit that can store data and output the stored data. The part that is programmed with VHDL is in the LUTs.





SRAM-based FPGAs

It stores logic cells configuration data in the static memory. Since SRAM is volatile, the FPGAs must be programmed upon start. There are two basic modes of programming:

Master mode: The FPGA is programmed by an external device such as an external flash memory chip. Slave mode: The FPGA is configured by an external master device, such as a processor.

SRAM based FPGAs with an internal flash memory: It contains internal flash memory blocks, thus eliminating the need to have an external non-volatile memory. It uses flash only during startup to load data to the SRAM configuration cells.

Programmable switch FPGAs

It uses flash as a primary resource for configuration storage, and doesn't require SRAM.

Switch is a floating-gate transistor that can be turned off by injecting charge onto its floating gate. It has a limited number of reprogramming, but is more secure than SRAM-FPGA.

Fuse-based FPGA

One time program FPGA. The fuse makes or breaks link between two wires. Making it more secure than SRAM (no load from an external device) In high radiation environments, radiation events can cause SRAM, that contains the program, to change state but not in fused FPGA.

Programmable switch matrix

All internal connections are composed of metal segments with programmable switching points and switching matrices to implement the desired routing.

Design flow

VHDL files. Goes to synthesis, then to implementation on the board and at last configuration where the FPGA is tested.

After the synthesis where the code is translated into logic gates, the logic gates are mapped into lut's on the logic blocks. After this placing routing is done, where the interconnections are wired using the switch blocks.

Once a design is implemented, the synthesis tool generates a file that the FPGA can understand. This file is called a bit stream. The bit file can be downloaded directly to the FPGA, or can be converted into a PROM file which stores the programming information.

4.1 CPLD

Complex Programmable Logic Devices (CPLDs) are similar to FPGAs, but they have fewer logic blocks and more interconnect.

Logic block signals only connect to the neighboring logic blocks. Nonvolatile logic cells (based on EEPROM Flash), on when powered.

CPLDs implement sum of product style logic not LUT as in FPGA. They are good for simple applications and cheaper.

5 UART

Serial port is a serial communication physical interface through which information transfers in or out one bit at a time. Data transfer through serial ports connected the computer to devices such as terminals and variuos peripherals.

Baudrate

It is necessary to choose a proper oscillator to get the correct baud rate with little or no error. Therefore, some common buad rates are known. E.g. 9600.

LSB is sent first known as little-endian. Also possible but rarely used is big-endian or MSB first.

Parity

Error detecting method. An extra data bit is sent with each data character, arranged so that the number of 1 bits in each character, including the parity bit, is always odd or even.

When setting up UART the clock frequency of the FPGA should be 16 times faster than the baud rate.

SPI

Serial Peripheral Interface is a synchronous serial communication interface for short-distance communication. Full duplex, single master.

Inter IC

Serial synchronous bus. Widely used to connect low-speed ICs processing units in short-distance, intra-board communication. Common speeds are 100kbps standard mode and 400kbps fast mode. Only two communication lines for all devices on the bus.

You can have multiple masters and multiple slaves.

Start condition is when the master node leaves the SCL high and pulls SDA low.

If two masters want to transmit at the same time, the first master to pull the SDA low wins the arbitration.

6 Memory

6.1 ROM

Read only memory.

6.2 RAM

RAM is a volatile memory meaning that the data is eventually lost when the memory is not powered.

SRAM: Static RAM

Faster than DRAM but larger in size. This is used for cache memory and has a low power consumption.

DRAM: Dynamic RAM

Lower speed than DRAM but smaller in size. This is used for main memory and has a higher power consumption. Refreshed frequently to avoid data loss.

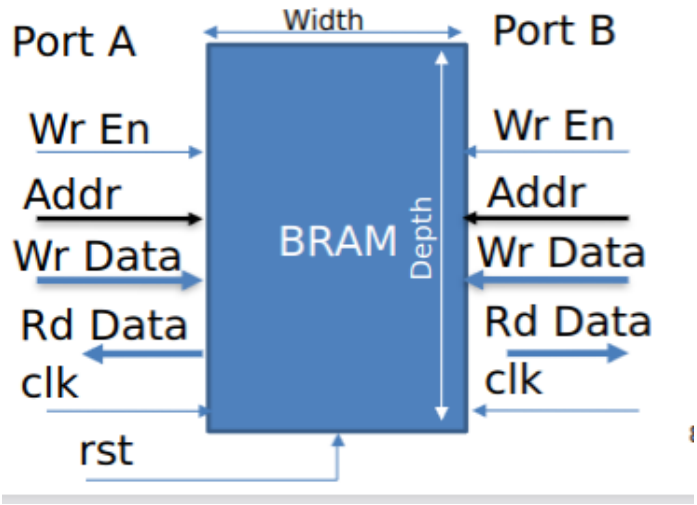
The more interfaces to the memory the more expensive it is. Can both read and write to the RAM.

6.3 BRAM

Storing large amount of data (e.g. sensor data). Used for storing read-only data. BRAM is a discrete part of the FPGA. Each FPGA chip has a limited number of BRAM.

Different configurations of BRAM:

Single port, Dual port, FIFO.



Dual port configuration allows each port to run in different clock frequencies. One operation at a time (read or write at the same address.)

The smart thing about using dual port is that you can read and write at the same time at different addresses.