DIGITAL SYSTEM DESIGN USING FPGA

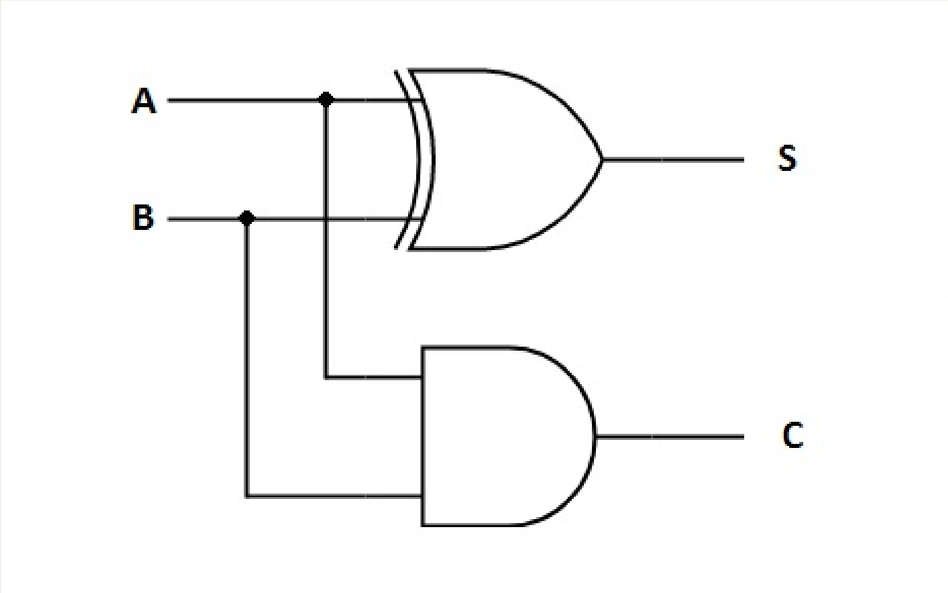
Advanced Verilog Implementation

ASCIEE SCHOOL OF EEE

tecs

DAKSH 2025
Build Things That Matter

What is Verilog




```
module HA(A, B, S, C);
    input A, B;
    output S, C;
    assign S = A^B;
    assign C = A&B;
endmodule
```

Why Verilog?

- Easy to use!
- Derived from C
- Verification & Simulation of circuit
- Scalable

Why Verilog? Why not VHDL?

```verilog
module HA(A, B, S, C);
    input A, B;
    output S, C;
    assign S = A^B;
    assign C = A&B;
endmodule

```

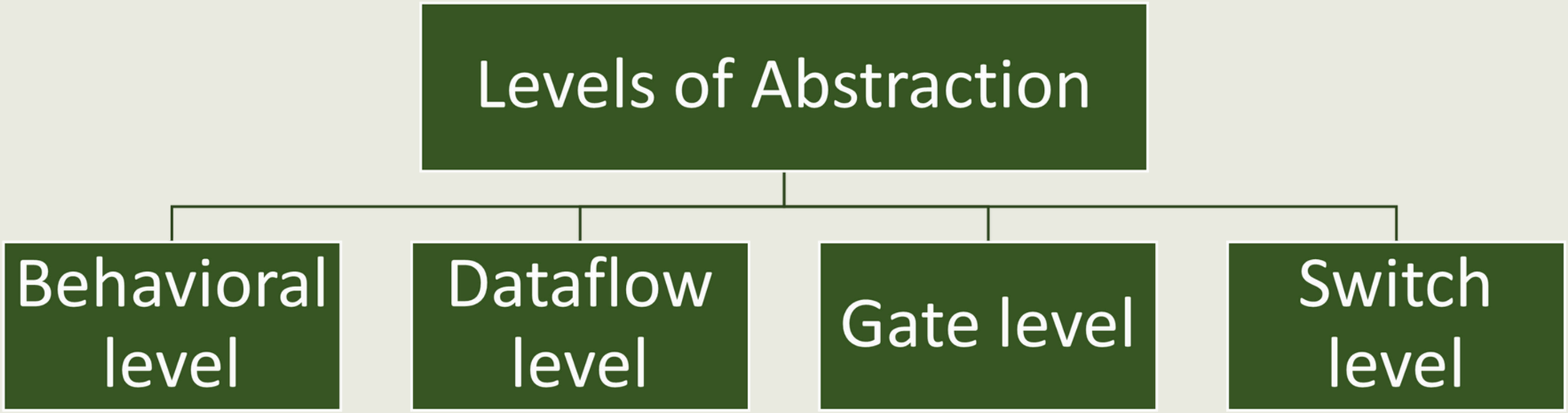
```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity half_adder is
  port (a, b: in std_logic;
    sum, carry_out: out std_logic);
  end half_adder;

architecture dataflow of half_adder is
  begin
  sum <= a xor b;
  carry_out <= a and b;
end dataflow;
```

Why Verilog? Why not VHDL?

```
1.  library ieee;
2.  use ieee.std_logic_1164.all;
3.
4.  entity half_adder is                        -- Entity declaratio
5.    port (a, b: in std_logic;
6.      sum, carry_out: out std_logic);
7.  end half_adder;
8.
9.  architecture structure of half_adder is      -- Architecture b
    adder
10.
11.   component xor_gate                       -- xor component decl
12.     port (i1, i2: in std_logic;
13.       o1: out std_logic);
14.   end component;
15.
16.   component and_gate                       -- and component decl
17.     port (i1, i2: in std_logic;
18.       o1: out std_logic);
19.   end component;
20.
21. begin
22.     u1: xor_gate port map (i1 => a, i2 => b, o1 => sum);
23.     u2: and_gate port map (i1 => a, i2 => b, o1 => carry_out);
24. -- We can also use Positional Association
25. --     => u1: xor_gate port map (a, b, sum);
26. --     => u2: and_gate port map (a, b, carry_out);
27. end structure;
```

```
1.  library ieee;
2.  use ieee.std_logic_1164.all;
3.
4.  entity half_adder is
5.    port (a, b: in std_logic;
6.      sum, carry_out: out std_logic);
7.    end half_adder;
8.
9.  architecture behavior of half_adder is
10.   begin
11.     ha: process (a, b)
12.     begin
13.       if a = '1' then
14.         sum <= not b;
15.         carry_out <= b;
16.       else
17.         sum <= b;
18.         carry_out <= '0';
19.       end if;
20.     end process ha;
21.
22. end behavior;
```

How does Verilog Work?

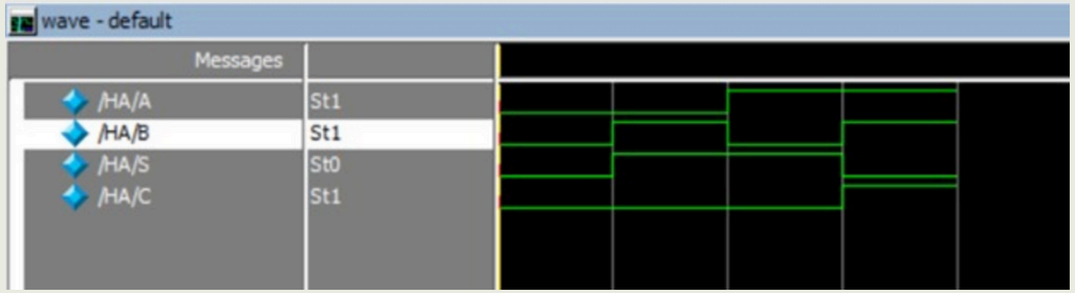


How does Verilog Work?

```c
#include <stdio.h>
int main() {
   // printf() displays the string inside quotation
   printf("Hello, World!");
   return 0;
}
```

```
Hello, World!
```

```verilog
module HA(A, B, S, C);
    input A, B;
    output S, C;
    assign S = A^B;
    assign C = A&B;
endmodule
```



Types of codes in HDL

Structural

```
assign a=b&c;
assign b=d&(~c);
```

Procedural

```
always @(posedge clk)
count <= count+1;
```

Syntax

```
module modulename(port_list);
    port declarations
    data_type declarations
    circuit functionality
    timing specification
endmodule
```

Datatypes

```
valuesets 0, 1, x, z
wire[lsb:msb] w
reg[lsb:msb]r
input
output
operators
```

Conditional statements

```
if(expr)
begin
    ....
end
else if(expr)
begin
    ...
end
else
begin
    ...
end
```

```
case(expr)
case_choice 1:
    begin
        ...
    end
case_choice 2:
    begin
        ...
    end
    default:
    begin
        ...
    end
```

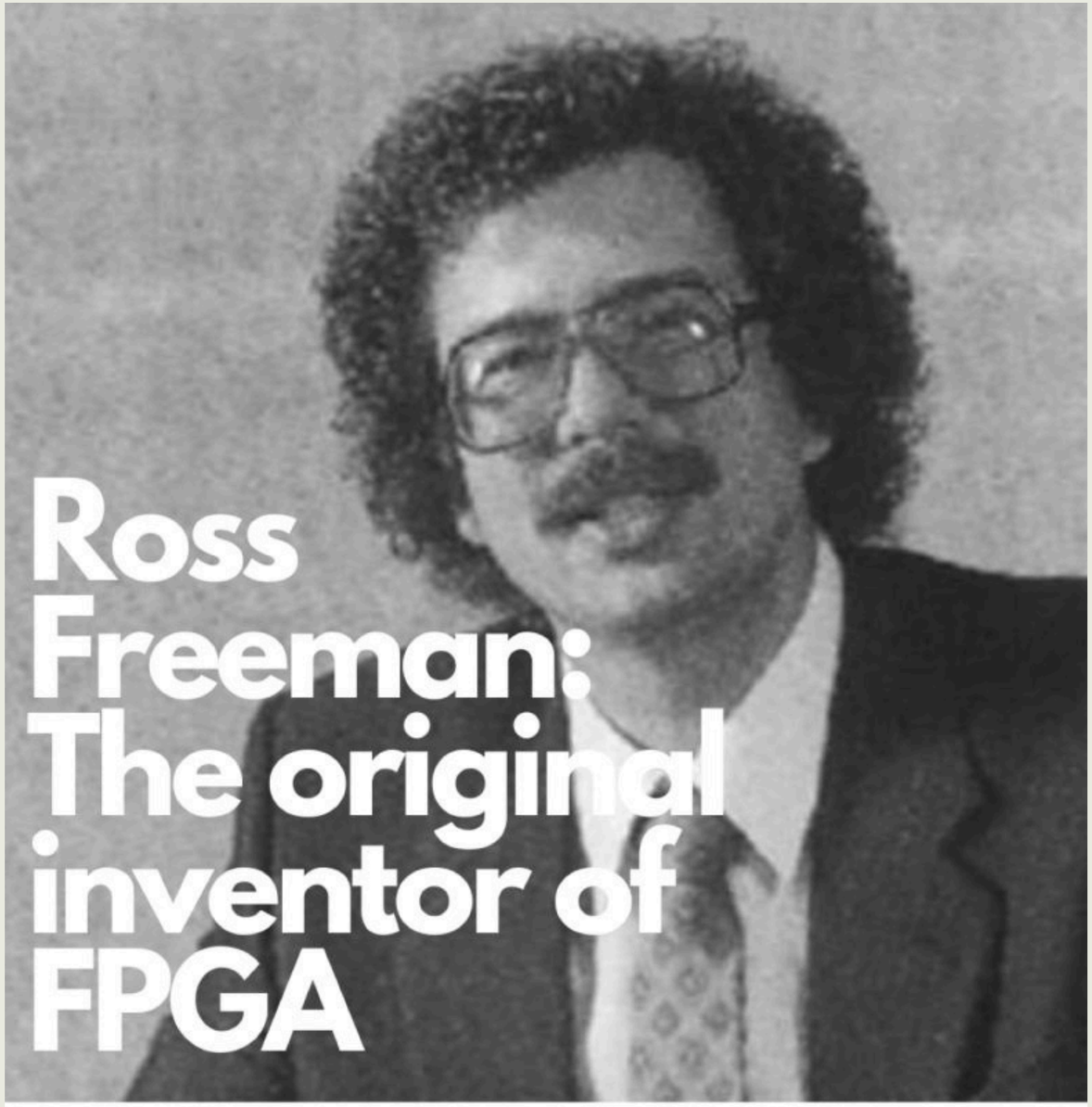
Module

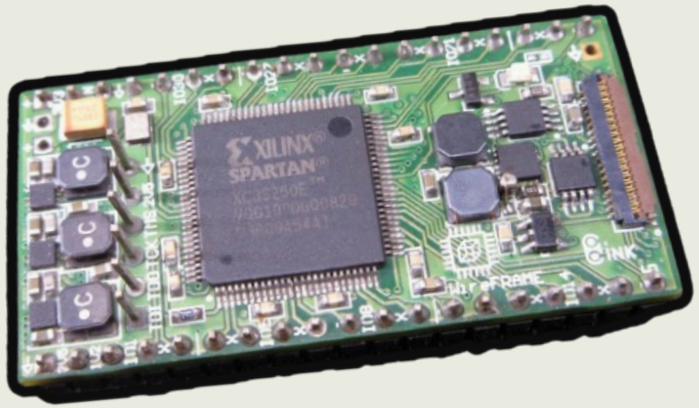
```
module name(name_list);
    input[msb:lsb] port list;
    output[msb:lsb] port list;
      ...
      statements
      ...
endmodule
```

Intro to FPGA

WHO IS THIS ?

XC2064 – 64 CLBs



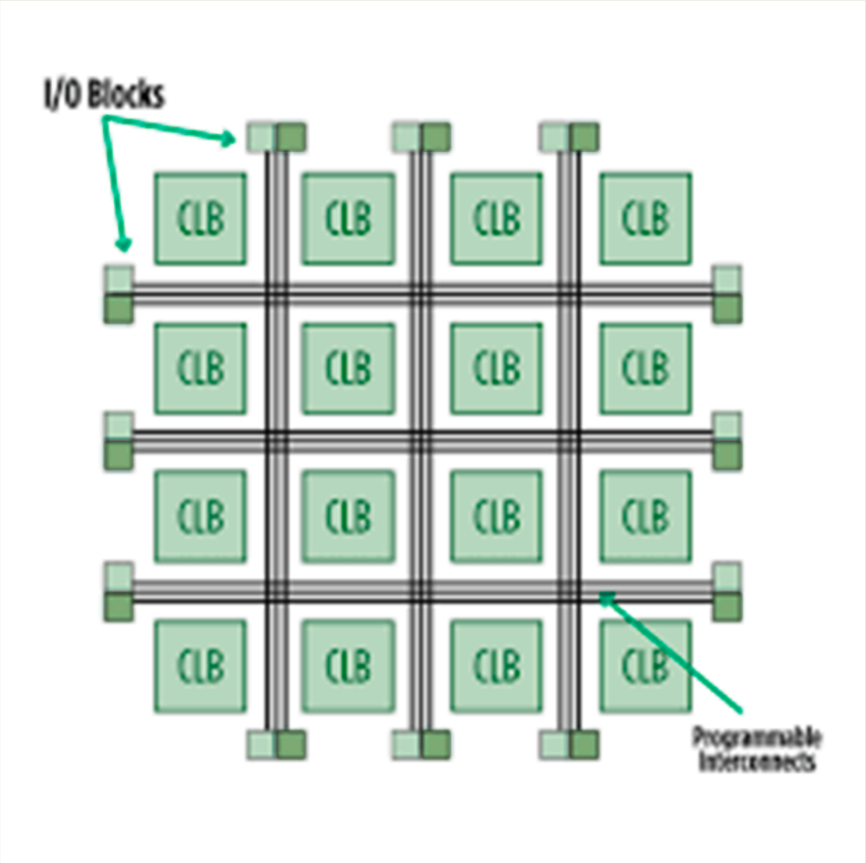


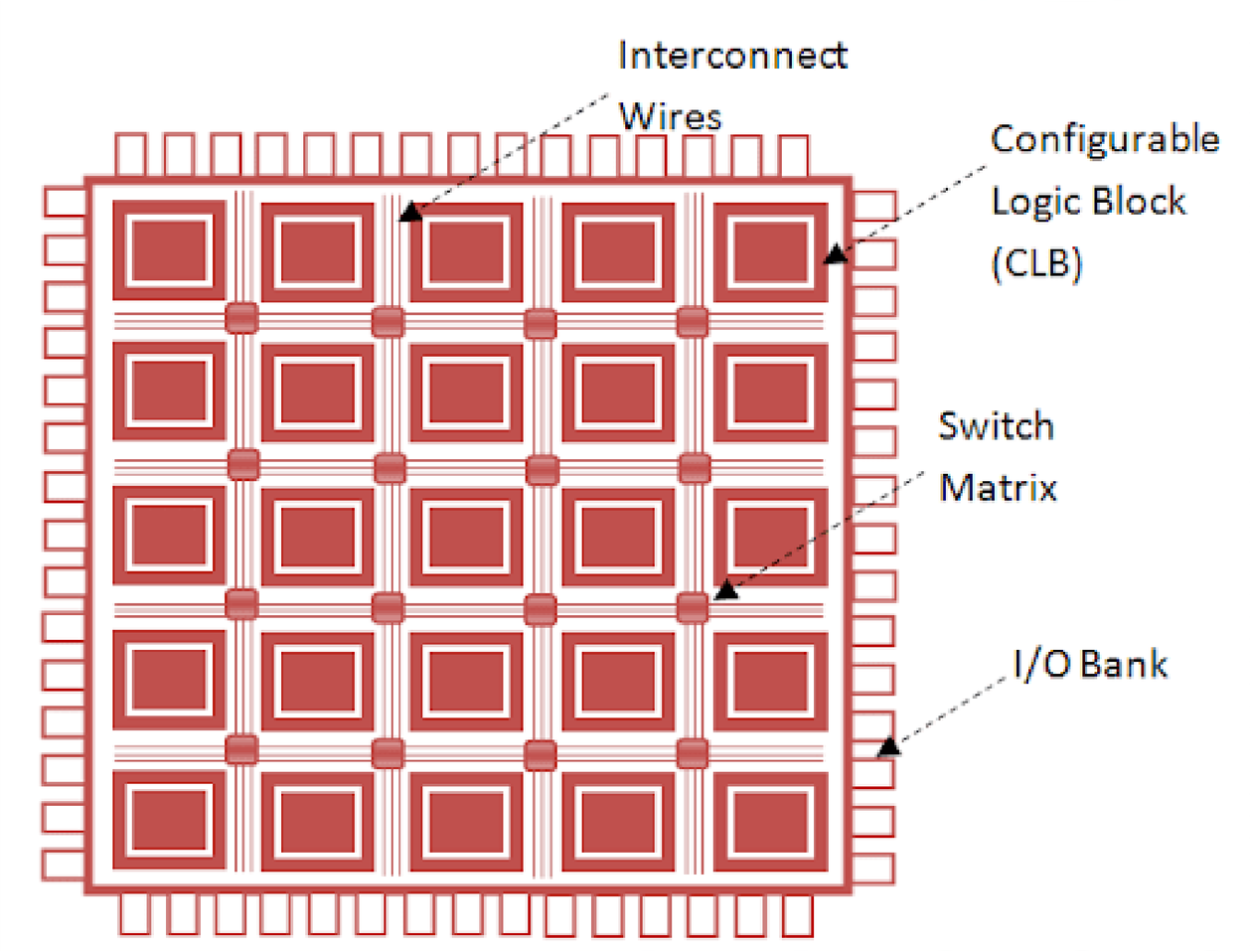
Ross Freeman, was an American electrical engineer and inventor, and co-founder of the leading FPGA developer Xilinx

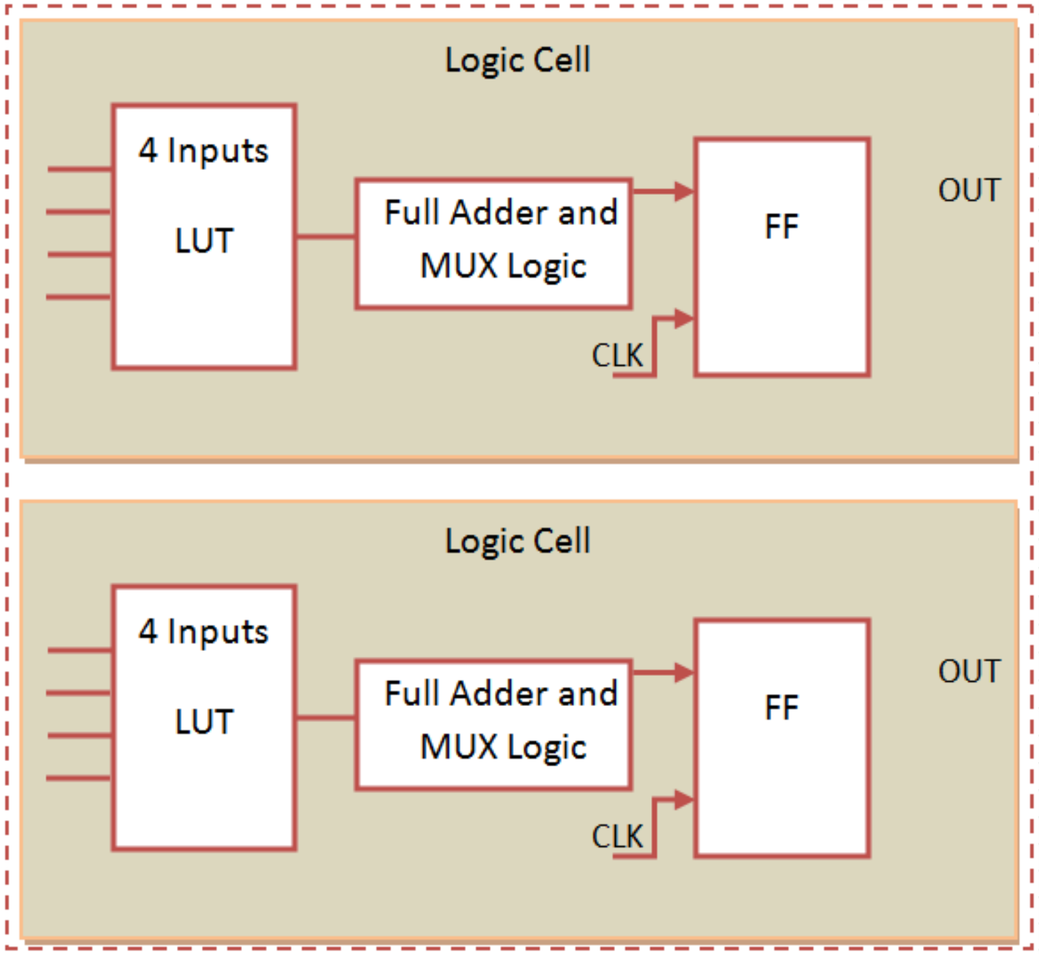
FPGA - FIELD PROGRAMMABLE GATE ARRAY

What is meant by Field programmable ?

What is meant by Gate Array ?







Whats inside FPGA chip ?

HOW IT IS WORKING ?

HOW IT IS WORKING ?

01
Verilog Code

02
Quartus Prime

03
Compiling (Synthesis and Fitting)

04
Generate .SOF/.POF

05
Load to FPGA via JTAG

06
Test using Peripherals

DE10-Lite

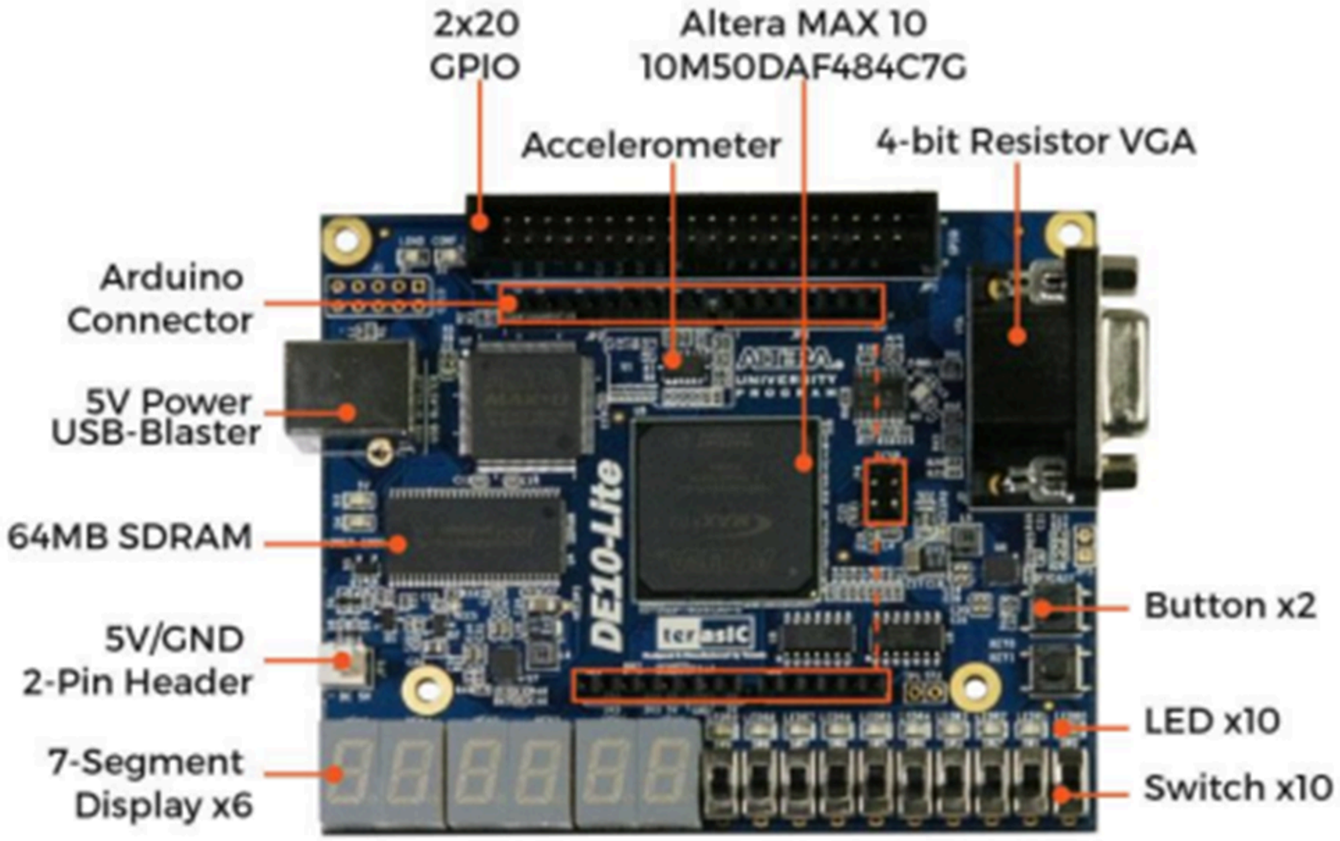
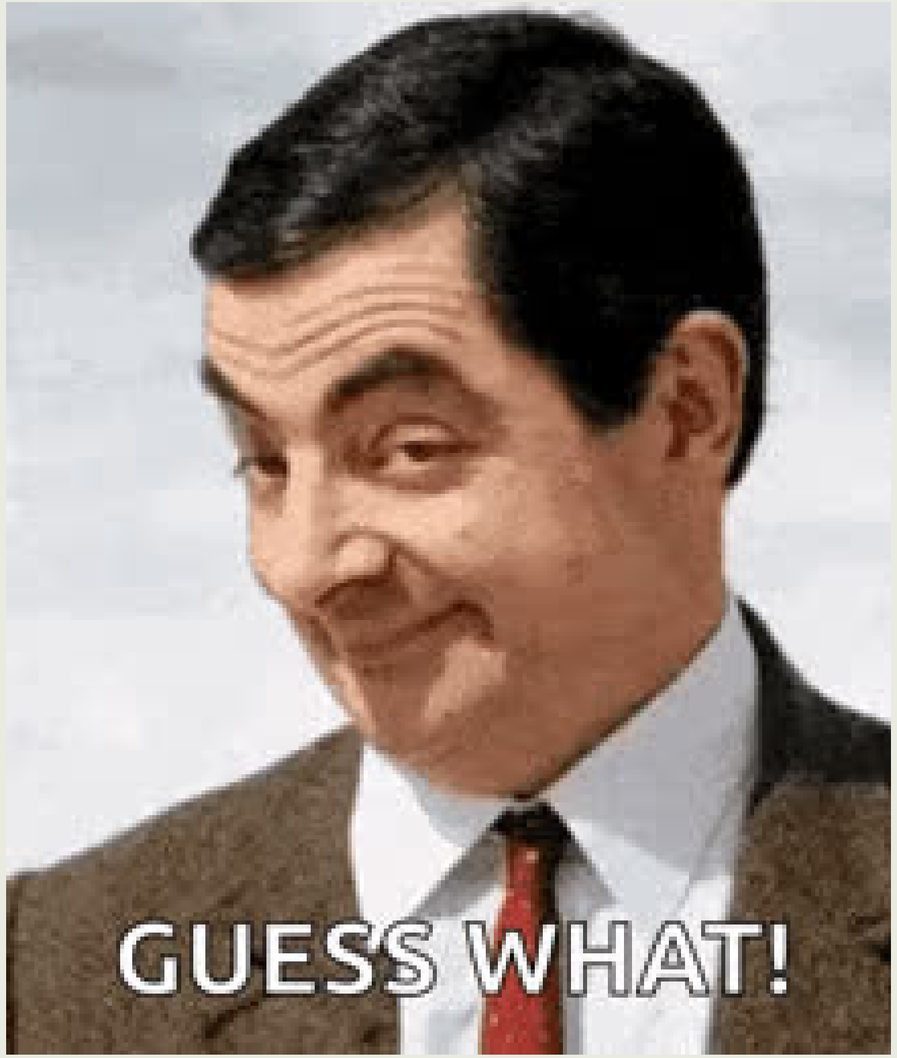


Figure 1-2 Development Board (top view)

Altera MAX 10 (10M50DAF484C7G) ?

- 10M -
- 50 -
- D -
- A -
- F484 -
- C -
- 7 -
- G -



Altera MAX 10 (10M50DAF484C7G)

10M : MAX 10 family (low cost, low power, small-medium designs).
50 : ~50K logic elements (mid-range MAX 10).
D : Dual configuration
A : Analog to Digital Converter
F484 : 484-pin BGA package.
C : Commercial temperature range (0°C to 85°C).
7 : Speed grade indicator.
G : Lead-free/RoHS compliant.

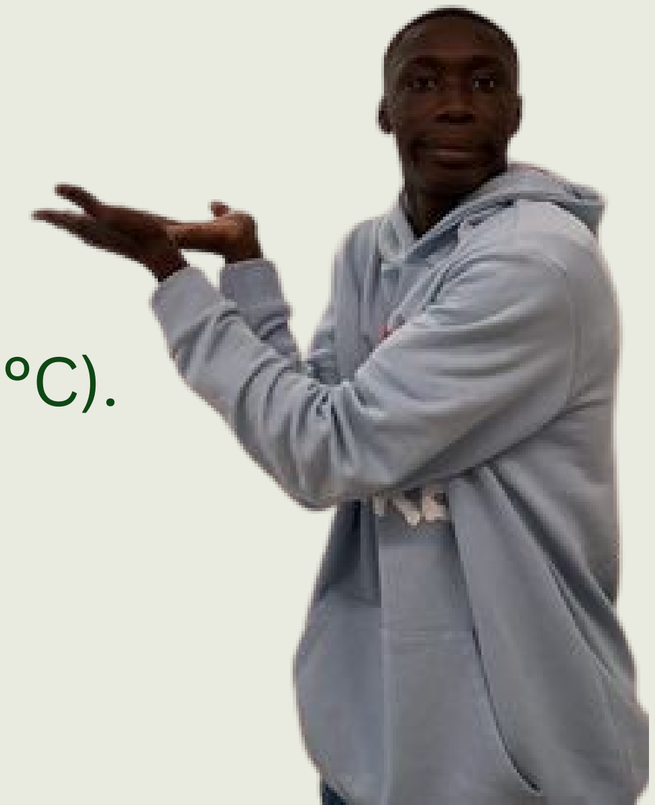


Figure 3-15 Connections between the slide switches and MAX 10 FPGA

Table 3-4 Pin Assignment of Slide Switches

Signal Name	FPGA Pin No.	Description	I/O Standard
SW0	PIN_C10	Slide Switch[0]	3.3-V LVTTL
SW1	PIN_C11	Slide Switch[1]	3.3-V LVTTL
SW2	PIN_D12	Slide Switch[2]	3.3-V LVTTL
SW3	PIN_C12	Slide Switch[3]	3.3-V LVTTL
SW4	PIN_A12	Slide Switch[4]	3.3-V LVTTL
SW5	PIN_B12	Slide Switch[5]	3.3-V LVTTL
SW6	PIN_A13	Slide Switch[6]	3.3-V LVTTL
SW7	PIN_A14	Slide Switch[7]	3.3-V LVTTL
SW8	PIN_B14	Slide Switch[8]	3.3-V LVTTL
SW9	PIN_F15	Slide Switch[9]	3.3-V LVTTL

Figure 3-16 Connections between the LEDs and MAX 10 FPGA

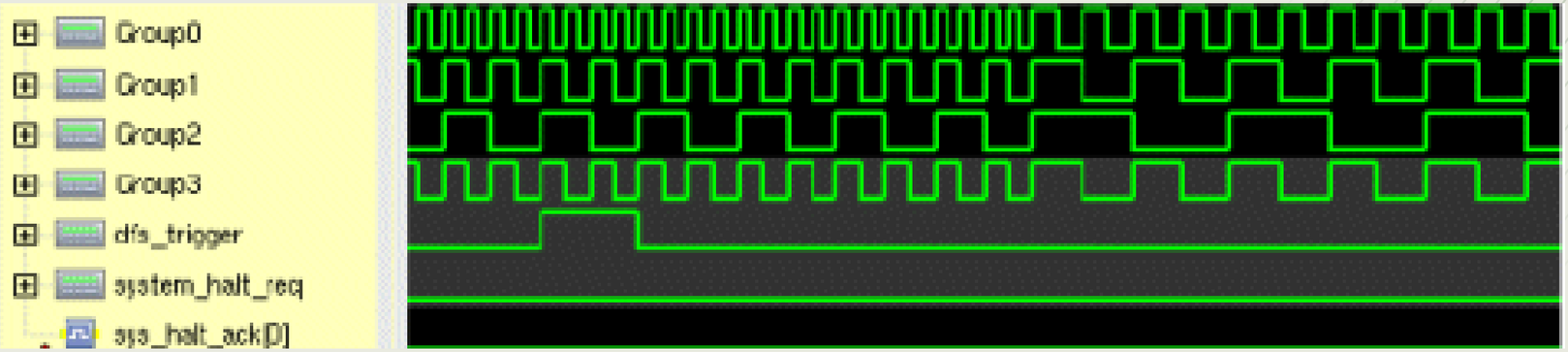
Table 3-5 Pin Assignment of LEDs

Signal Name	FPGA Pin No.	Description	I/O Standard
LEDR0	PIN_A8	LED [0]	3.3-V LVTTL
LEDR1	PIN_A9	LED [1]	3.3-V LVTTL
LEDR2	PIN_A10	LED [2]	3.3-V LVTTL
LEDR3	PIN_B10	LED [3]	3.3-V LVTTL
LEDR4	PIN_D13	LED [4]	3.3-V LVTTL
LEDR5	PIN_C13	LED [5]	3.3-V LVTTL
LEDR6	PIN_E14	LED [6]	3.3-V LVTTL
LEDR7	PIN_D14	LED [7]	3.3-V LVTTL
LEDR8	PIN_A11	LED [8]	3.3-V LVTTL
LEDR9	PIN_B11	LED [9]	3.3-V LVTTL

FREQUENCY SCALING

- Technique used to adjust system **operating speed.**
- Measured in Hertz (Hz), typically in Gigahertz (GHz) (billions of cycles per second).
- The purpose of frequency scaling is to **balance between performance and power consumption.**

FREQUENCY SCALING TYPES



FREQUENCY SCALING TYPES

- **Scaling Up**
  - Increases system speed.
  - Suitable for high-performance tasks (e.g., gaming, video editing).
  - More power consumption and heat generation.

- **Scaling Down**
  - Decreases system speed.
  - Ideal for low-power tasks (e.g., emails, document writing).
  - Saves energy and extends battery life.
  - Reduces heat generation.

.

Applications

- Used in laptops, smartphones, and embedded systems to optimize performance and power efficiency.

Conclusion

- Frequency scaling helps in achieving a balance between speed, power consumption, and temperature

Calculations

$$\text{OUTPUT} = \frac{\text{Operating Frequency}}{\text{Desired Frequency}}$$

Counter Bits = 2^n

Where n is output

LET'S START CODING!

PWM SIGNALS

- PWM is a technique used to control the speed, intensity, or power of devices like DC motors and LEDs.

Working Principle

- Rapidly switch the device ON and OFF.
- The duty cycle determines how long the signal stays ON during each cycle.
- Duty cycle is expressed as a percentage of the total cycle time.

Why Use PWM?

- Energy-efficient: Unlike resistors, PWM does not waste energy as heat.
- Provides precise control over speed or power.
- Commonly used in battery-powered devices.

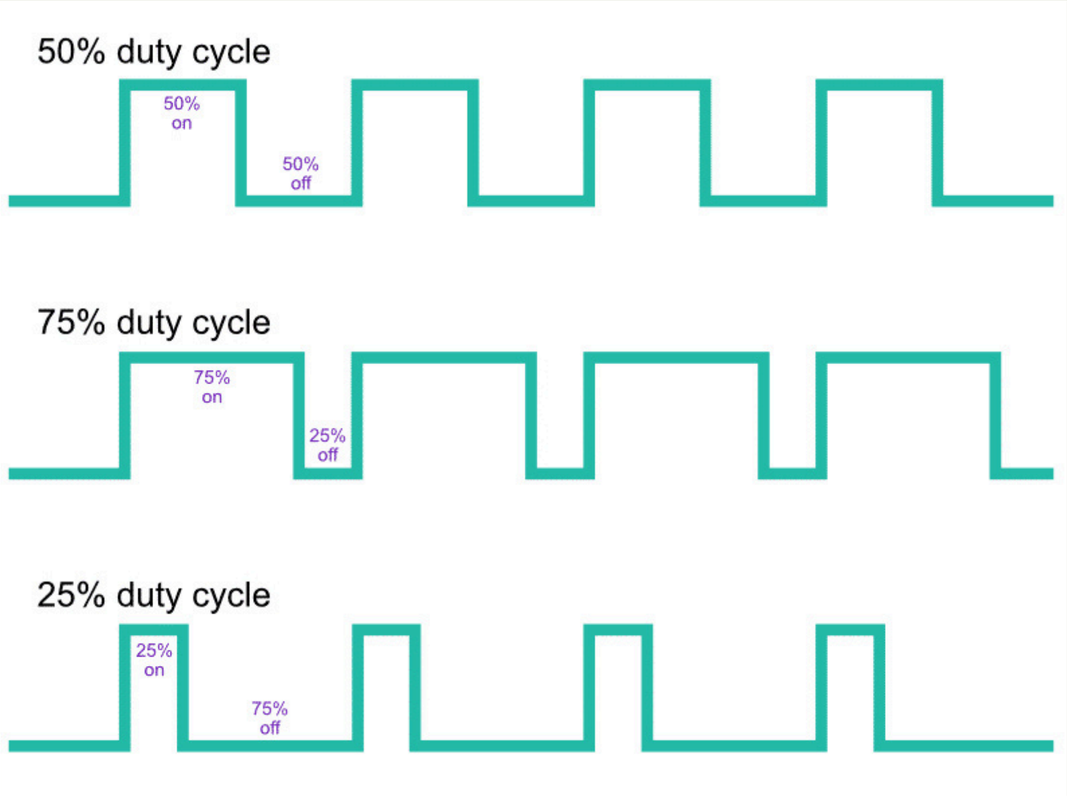
Duty Cycle Explanation

- 0% Duty Cycle: Device is always OFF.
- 50% Duty Cycle: Device is ON for half the time (moderate speed).
- 100% Duty Cycle: The Device is always ON (full speed).

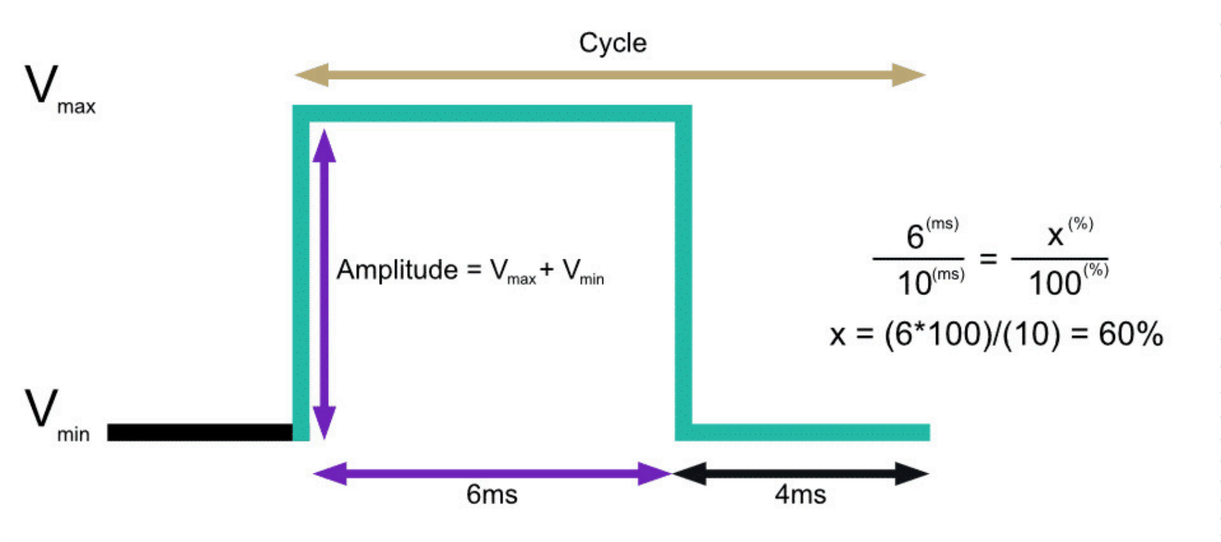
Pulse Width Control using a DC modulation Voltage - waveforms

Applications

- Motor speed control (DC motors).
- LED dimming.
- Power control in robotics and embedded systems.
- Fan speed regulation



PWM Signal with different duty cycles



IS IT POSSIBLE TO USE THE CONCEPT OF COUNTERS ?

Using a counter to create a PWM (Pulse Width Modulation) signal is a straightforward and commonly used method. Here's how we can implement it.

Counter Setup: First, we need a counter that counts up from 0 to a maximum value, typically referred to as the "period" or "cycle" value. The period value determines the time it takes for one complete cycle of the PWM signal. For example, if we want a PWM signal with a period of 16 clock cycles, our counter should count from 0 to 15.

Duty Cycle Control: To control the duty cycle (the percentage of time the signal is in the "on" state), we need another signal, often referred to as the duty_cycle value. This value is compared to the counter value. When the counter value is less than the duty_cycle value, the PWM output is high (on), and when it's equal to or greater than the duty_cycle value, the PWM output is low (off).

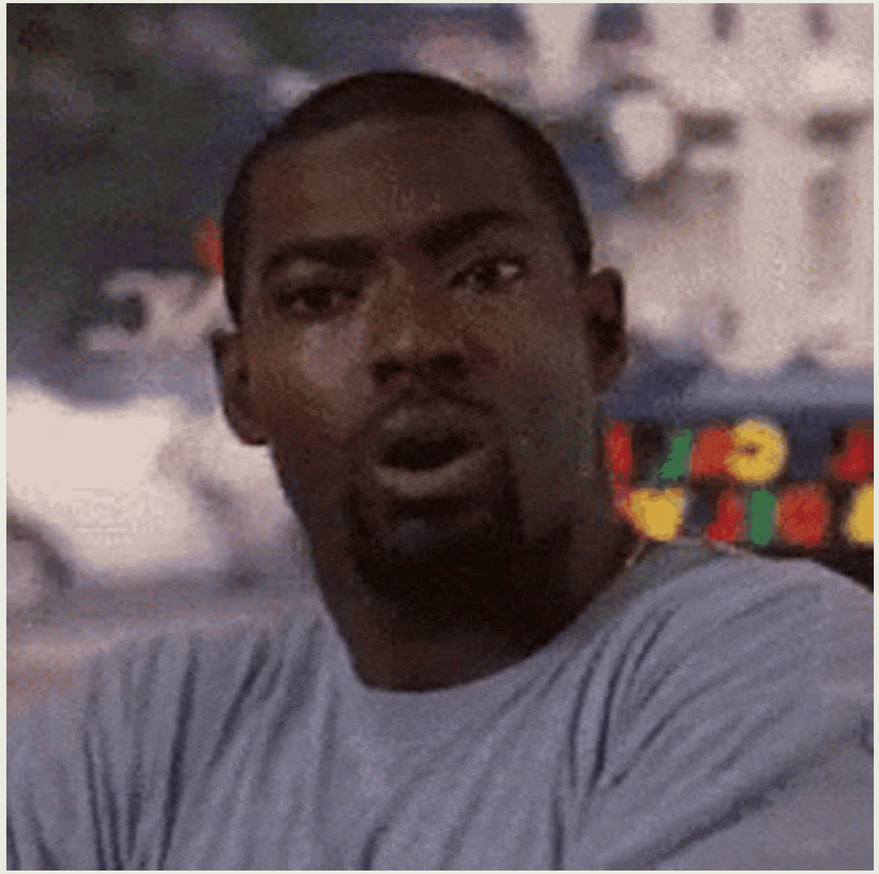
PWM Output: The output of the PWM signal is taken from the comparison result. When the counter is less than the duty cycle value, the output is high (logical '1'), and when the counter is greater or equal to the duty_cycle value, the output is low (logical '0').

LET'S START CODING!

TIME FOR LUNCH

QUIZ?

It’s Time to make your OWN LOGICAL UNIT



FSM - FINITE STATE MACHINES

used to recognize patterns in input sequences

What is an FSM (Finite State Machine) ?

An FSM (Finite State Machine) is a mathematical model of computation used to design and describe systems that have a limited number of states. An FSM can be in only one state at a time, and it can move between different states based on certain inputs.

- States
- Transitions
- Inputs
- Outputs

States :

- These represent different stages or conditions of the system. For example, in a vending machine, the states could represent different stages like waiting for coins, checking if enough money has been inserted, or dispensing an item.

Transitions :

- These are rules that decide how the FSM moves from one state to another. Transitions happen when certain inputs or conditions are met.

Inputs :

- Inputs are the signals or data that trigger state changes. In the case of the vending machine, inputs might include:
  - Amount of money inserted (amt)
  - Item choice (cho)
  - A reset signal (rst).

Outputs :

- Outputs are actions or information displayed when the FSM is in a certain state. In a vending machine, outputs could be:
  - A 7-segment display shows the money inserted or the item selected.
  - Signals controlling the mechanism to dispense the product.

MODEL	Output Depends on
Mealey Machine	Present state + Input
Moore Machine	Present state

FSM for Vending Machine :

1. Idle (ideal) State
2. Money Insertion
3. Item Selection
4. Done (de)
5. Display Control:

1 Idle (ideal) State:

- The FSM starts in the ideal state where no money has been inserted, and all 7-segment displays are turned off (m1, m2, m3, m4 are all 1111111).
- Based on the amt input, it transitions to the appropriate state:
  - If amt == 2'b00 (insert 5 cents), the FSM moves to r5.
  - If amt == 2'b01 (insert 10 cents), it moves to r10, and so on.

2 Money Insertion (States like r5, r10, r15):

- In states like r5, r10, r15, the FSM keeps track of how much money the user has inserted. It also updates the 7-segment displays to show the money inserted.
- The FSM transitions to the next state based on the amt input (e.g., amt == 2'b00 might move from r5 to r10).

3 Item Selection (cho):

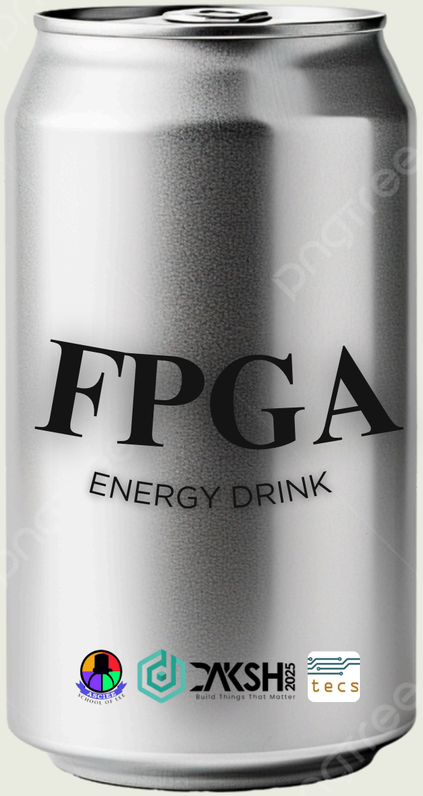
- In certain states, the FSM checks the cho input to determine which item the user wants to select. For example, in the r20 state, the FSM might check cho to see which product the user wants to buy.
- Depending on the cho value, the FSM may transition to another state or complete the transaction.

4 Done (de):

- The de signal is used to reset the FSM back to the ideal state when the transaction is complete or when the user is done interacting with the machine.

5 Display Control:

- The FSM updates the 7-segment displays (m1, m2, m3, m4) based on the state. For example, in state r20, it might display a particular pattern on the 7-segment displays to indicate 20 cents.



By the end of FSM for vending machine , u should have got your FPGA energy drink from vending machine

LET'S START CODING!

Mealy FSM for Sequence Detector :

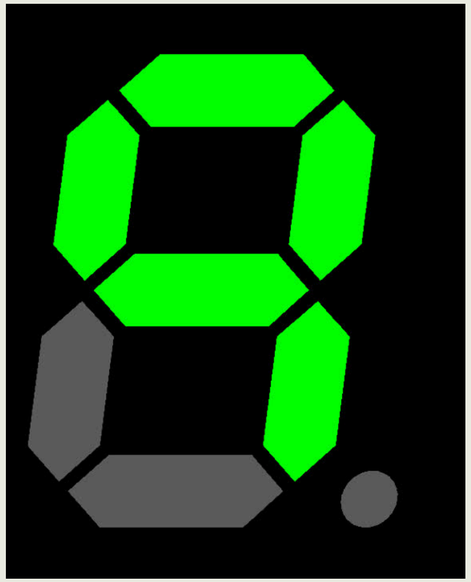
A Mealy FSM is where the output depends on both the current state and the current input. This makes it well-suited for tasks like sequence detection, where the output (detection signal) depends on both the current state in the sequence recognition process and the incoming bit.

What is a Sequence Detector?

A sequence detector is a digital circuit that detects a specific sequence of bits within a longer stream of input bits. When the target sequence is detected, the circuit typically asserts an output signal.

1. **Input:** The incoming bit stream.
2. **A (Idle):** Initial state, no part of the sequence has been detected.
3. **B:** First '1' has been detected.
4. **C:** '10' has been detected.
5. **D:** '101' has been detected.

6 Display Control:



- Displays **10** in a 7 segment display if the State is D, '101' has been detected and present input is 0 which is ‘1010’.

- Displays the state **A,B,C,D** in the 7 segment with the respective state currently in .

FEEDBACK FORM

SCAN ME

THANK YOU

ASCIEE
SCHOOL OF EEE

tecs