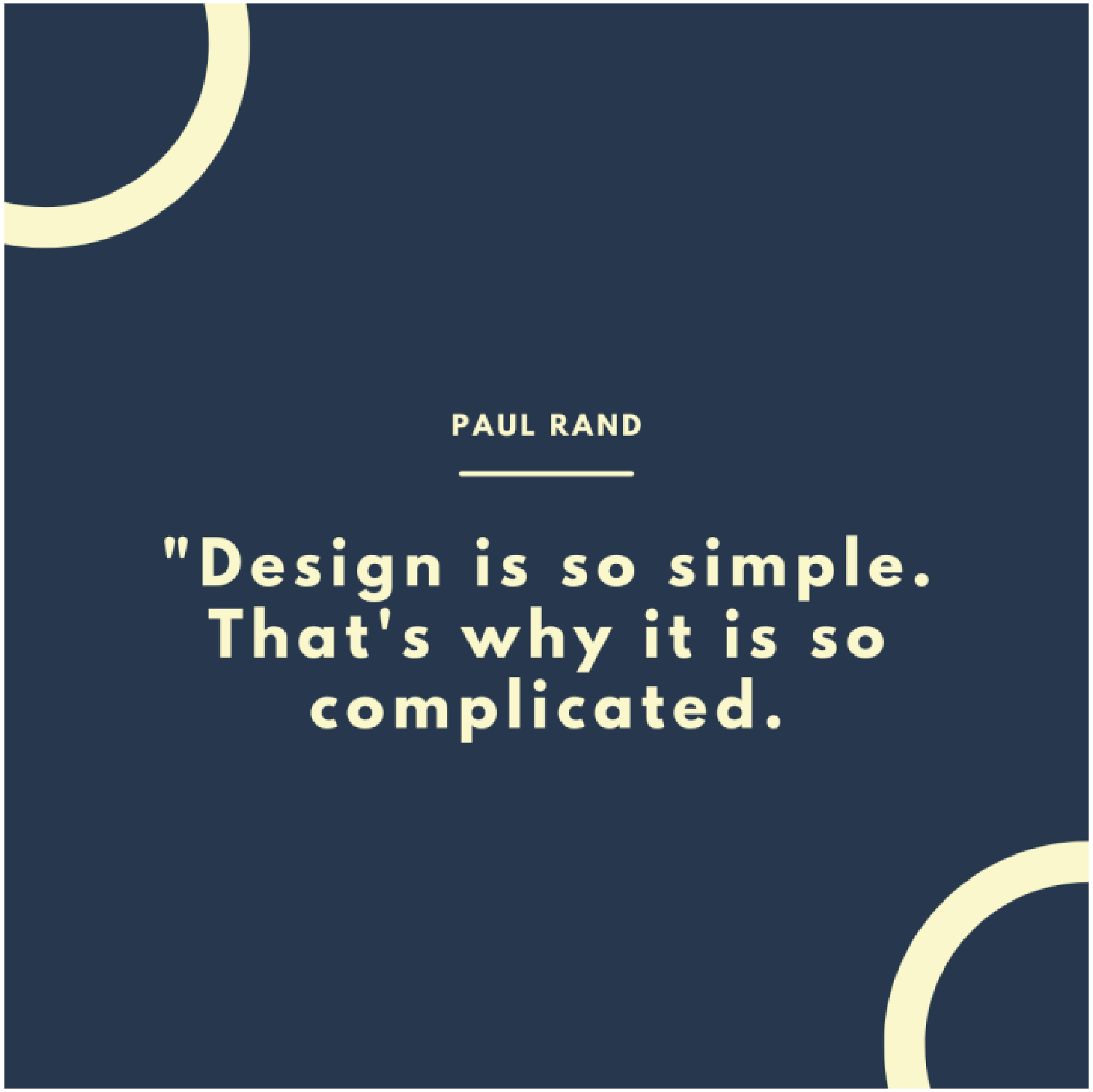
<u>In-House Chip Design</u>

Pravin Zode

Assistant Professor,
Department of Electronics Engg.
Yeshwantrao Chavan College of Engineering,
Nagpur
ppzode@ycce.edu

Outline

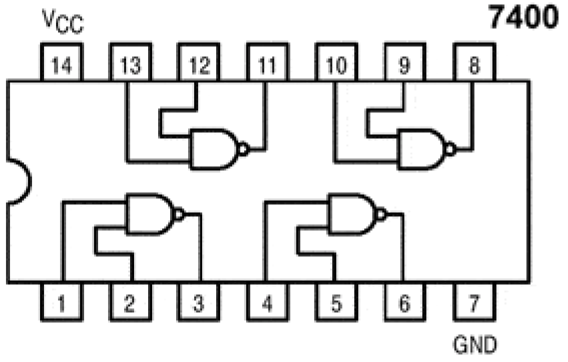
- Introduction to Design
- Older Ways of design
- Modern Digital Design
- Programmable Logic Devices (CPLD and FPGA)
- Demonstration of In-House Chip design



PAUL RAND

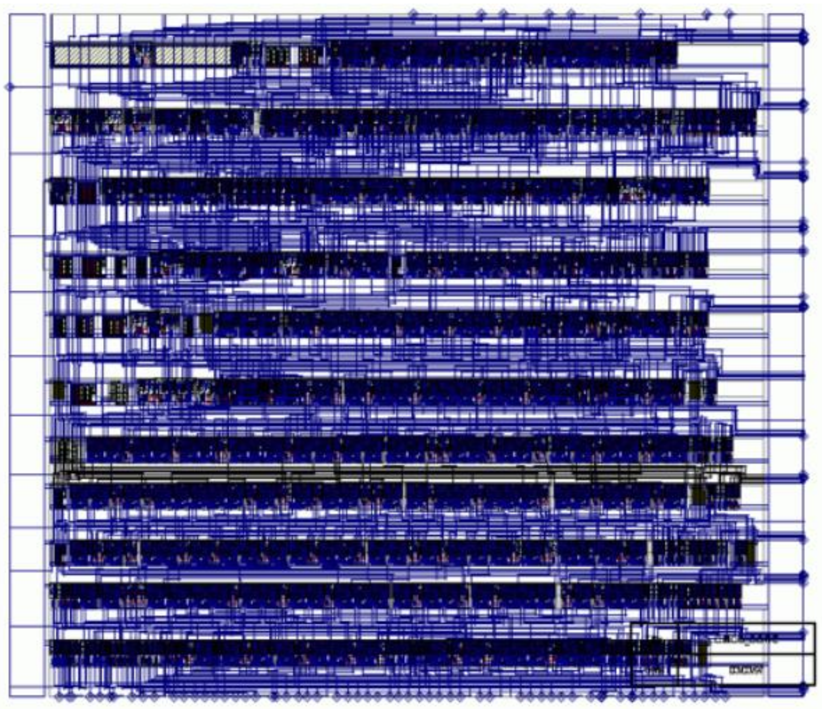
"Design is so simple. That's why it is so complicated.

Older ways of Digital Design



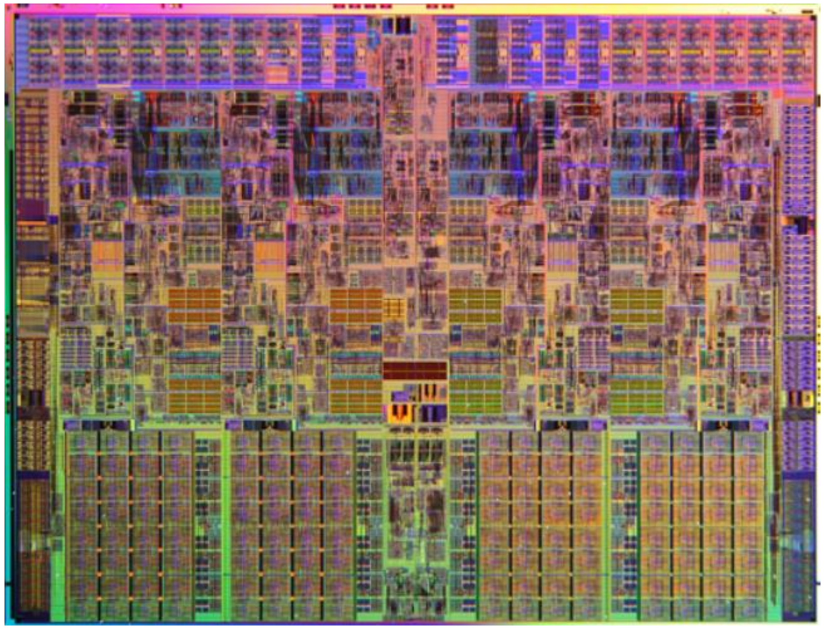
- Discrete Logic : Can build small circuits only
- De-Morgan’s Theorem : Theoretically we need only two input NAND and NOR gates
- Tedious , Expensive , slow, prone to wiring errors

Early Digital Design using Gate Arrays



- Rows of Gates : Often Identical in nature
- Connected to form custom specific circuits
- Can be full custom, completed design from scratch
- Can be semi-custom ; customization on metal layer only
- Once fabricated , the design is fixed

Modern Digital Designs

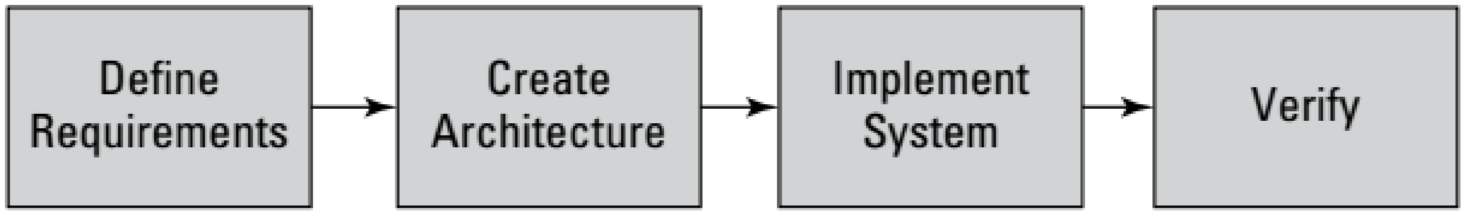


- Intel Core i7
- More than 3 billion transistors
- Very expensive to design
- Very expensive to manufacture
- Not viable unless the market is very large

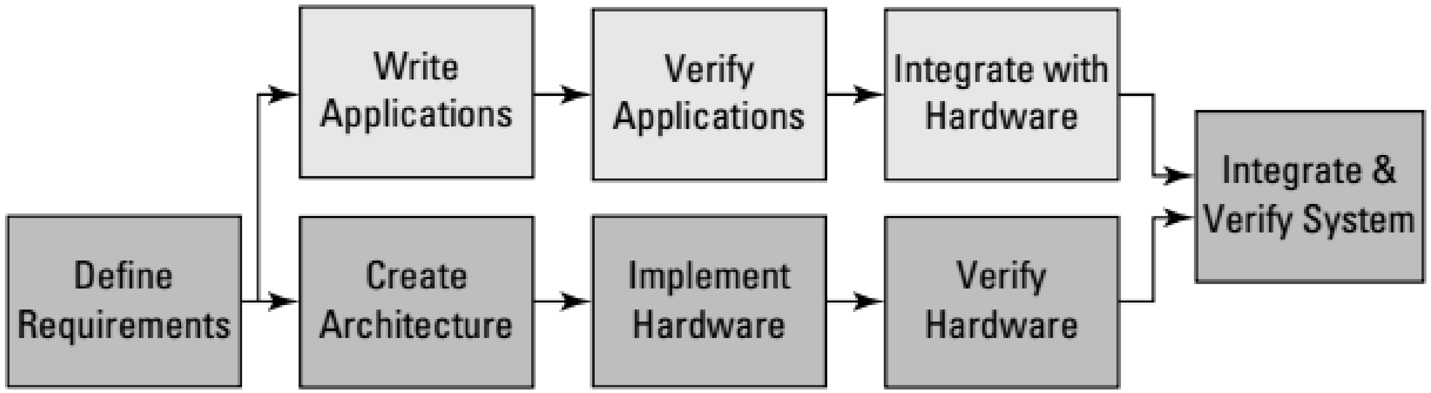
FPGA Vs Microcontroller

- With a microcontroller you write software
- The biggest limitation of software is that you can only do one thing at a time!
- With FPGAs you are not creating software
- You are designing the hardware!
- Instead of writing code to run on a fixed processor with fixed peripherals, you get to design your own circuit.
- If you really want to you can even create your very own processor and write software to run on it!
- A huge benefit of working with FPGAs is that every element of your design runs independently of each other
- That means one part of your design can be reading in some serial data, while another part is controlling a servo, while another is reading some sensors, and yet another is controlling a display

System Design Flow

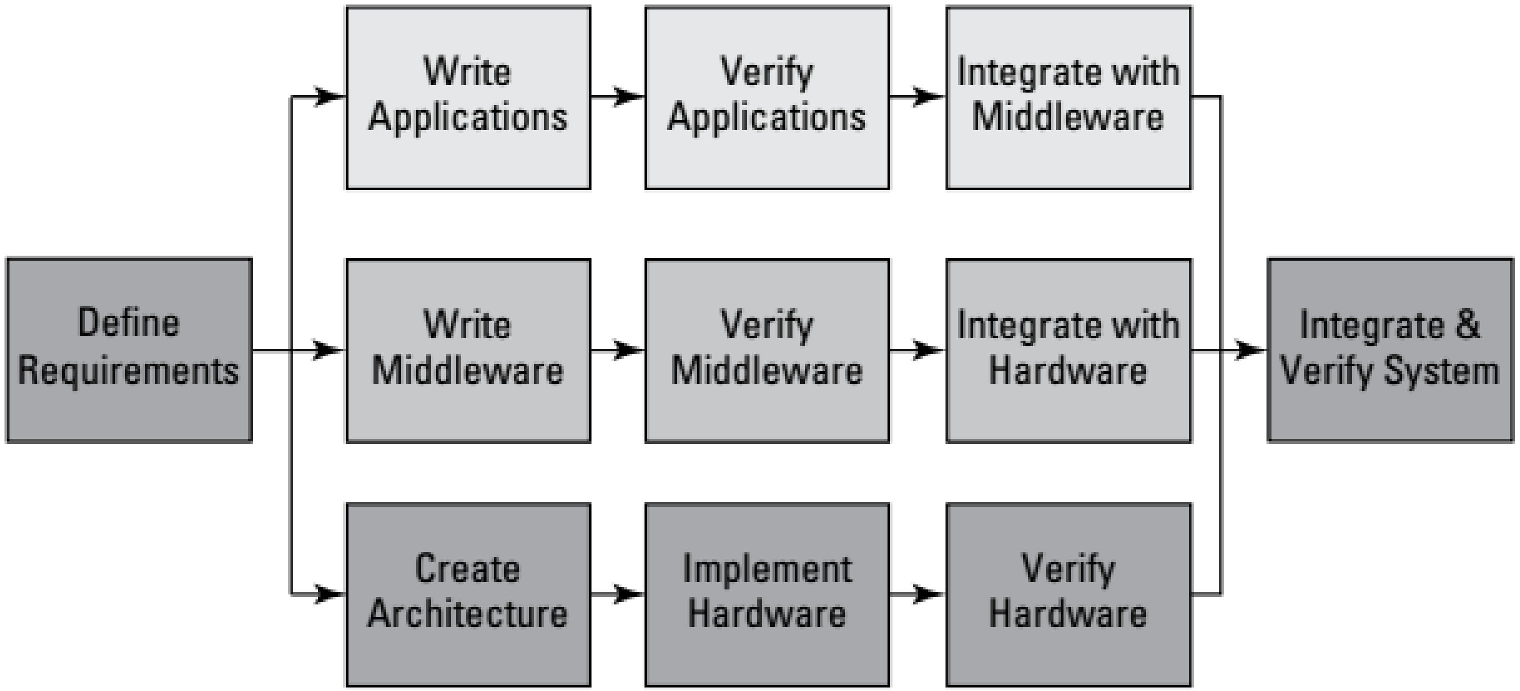


System design flow

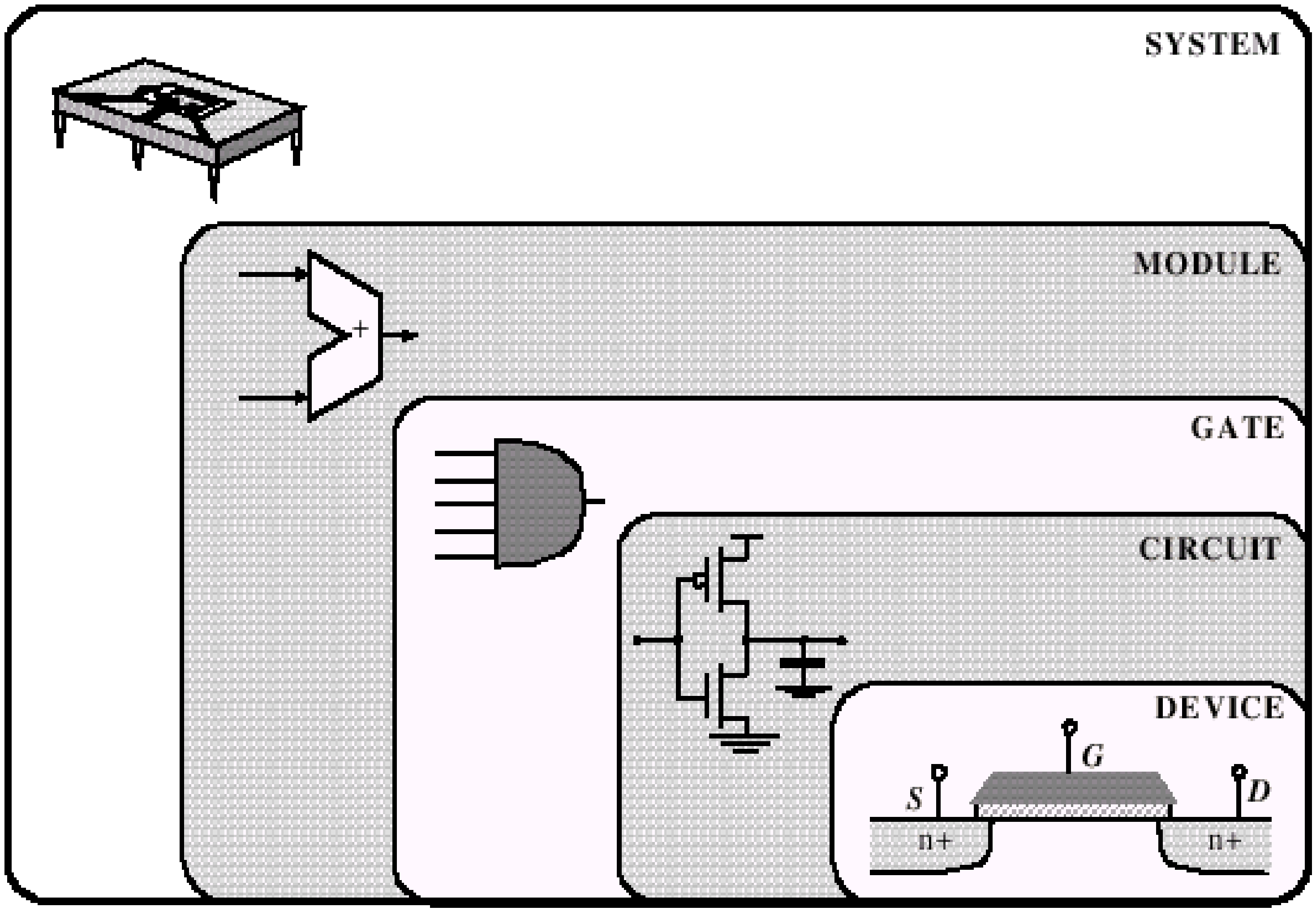


System design flow with the software application flow

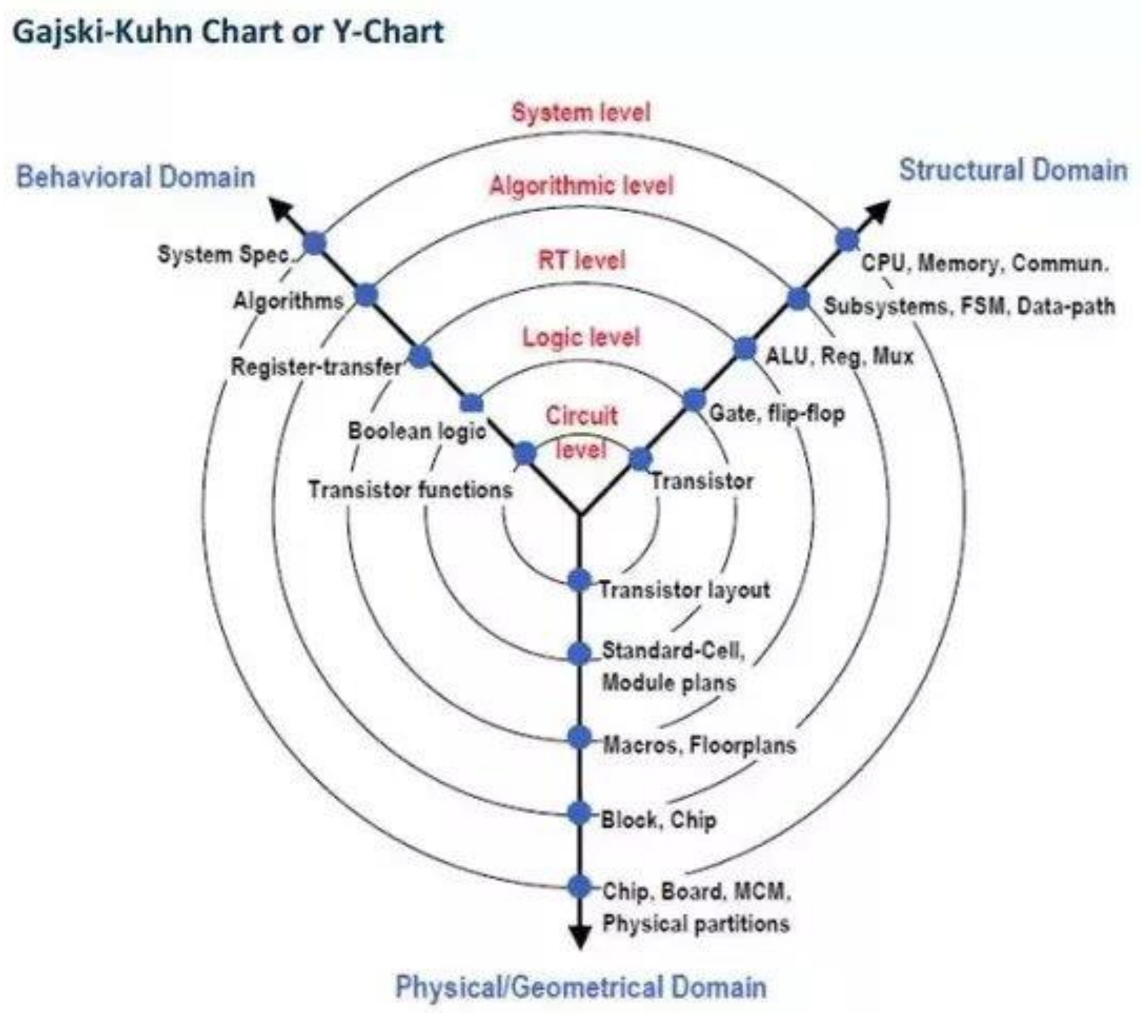
Middleware Design Flow



Design Abstraction Level



Design Abstraction Y Chart



Abstraction Levels in Hardware Design

Levels of Abstraction:

- Behavioral Level: High-level algorithm description
- Register Transfer Level (RTL): Data transfer between registers
- Gate Level: Logic gates and flip-flops
- Physical Level: Layout and fabrication

Register Transfer Level

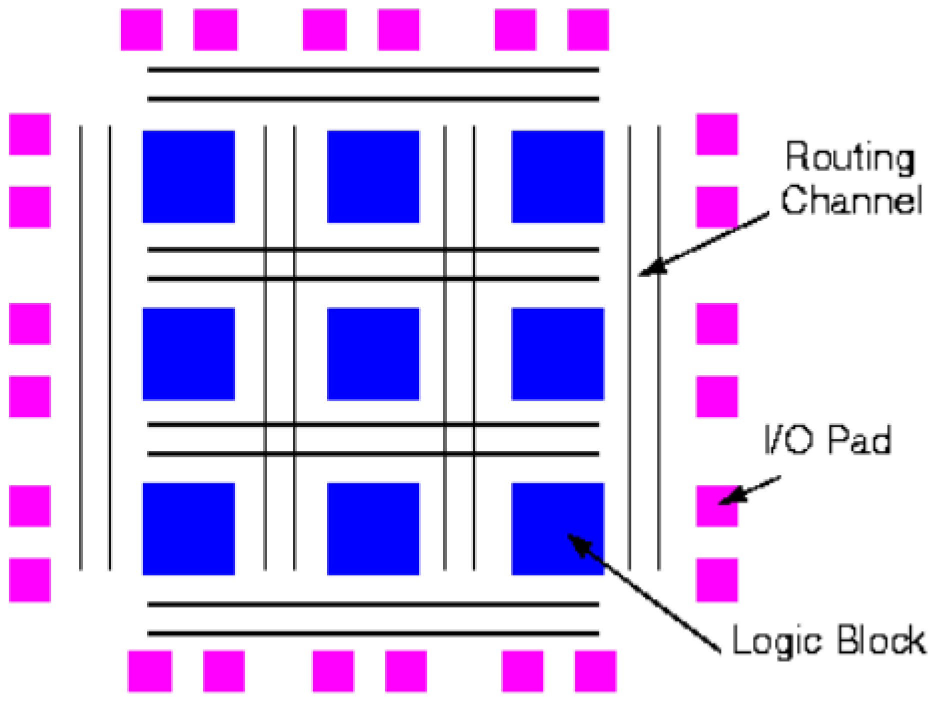
RTL : Data transfers between registers and the operations performed on that data.

- **Register (R):** A register is a small storage element that holds binary data (FF & Clock)
- **Transfer (T):** Data is moved from one register to another via combinational logic (e.g., multiplexers, adders, ALUs)
- The movement occurs in a clock cycle and follows specific control signals.
- **Level (L):**It describes how data moves between registers and how operations are performed, but it does not specify exact gate-level implementation.

Key Components of RTL Design

- **Registers:** Storage elements (D Flip-Flops, Latches)
- **Combinational Logic:** Data processing (AND, OR, MUX, ALU)
- **Control Logic:** State machines, control signals
- **Clock and Reset:** Timing control

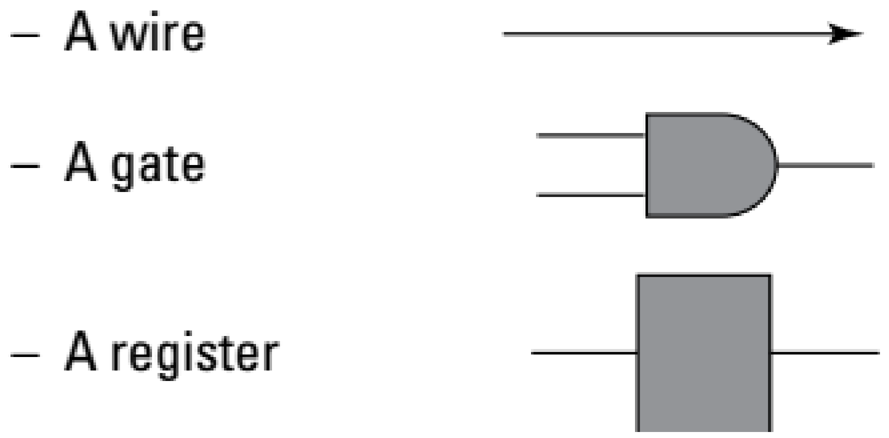
Field Programmable Grid Array



- Combined idea of PLD and gate arrays
- First introduced by Xilinx in 1985
- Array of logic blocks
- Lot of programmable wiring
- Very flexible I/O interfacing
- Two dominant FPGA makers
  - Xilinx and Altera
- Other specialist makers
  - Actel and Lattice Logic

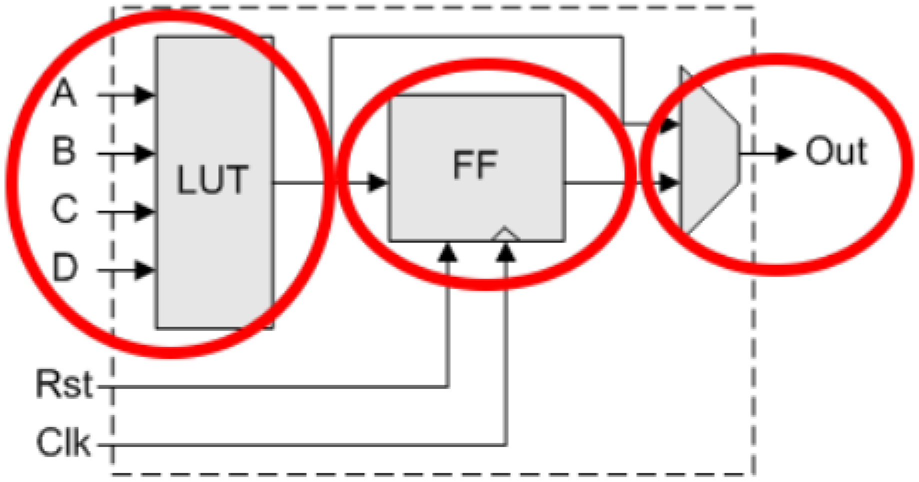
Building blocks of FPGA

- A *register* remembers a piece of information until it is told to remember something else
- A *logic gate* performs simple logic operations on signals
- *A wire* connects these other pieces

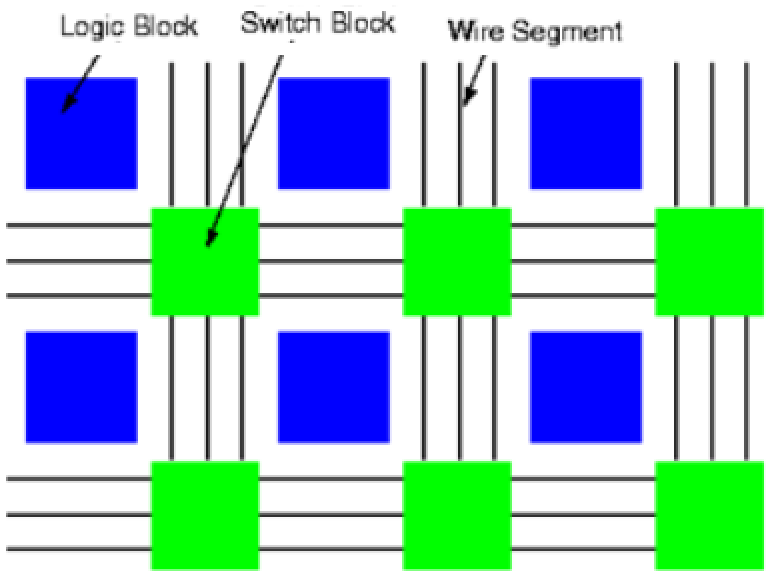


Older ways of Digital Design

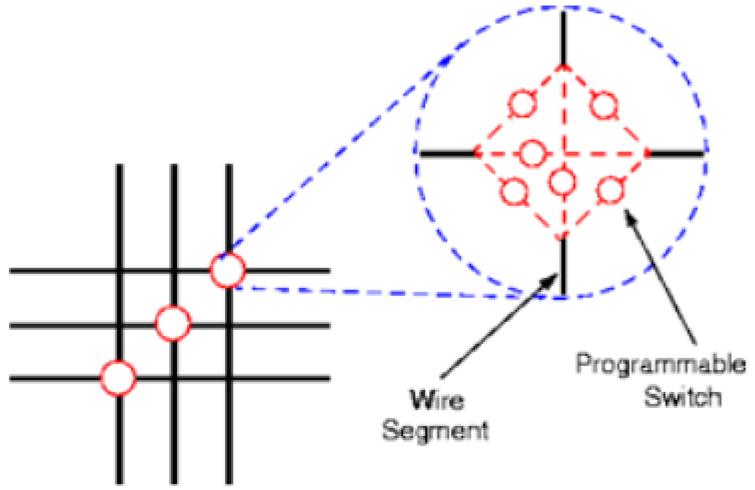
- Based on Lookup Table ,most common 4 input
- Optional D-FF at the output
- 4-input LUT can implement any Boolean function
- Special circuits for cascading logic blocks (carry chains)

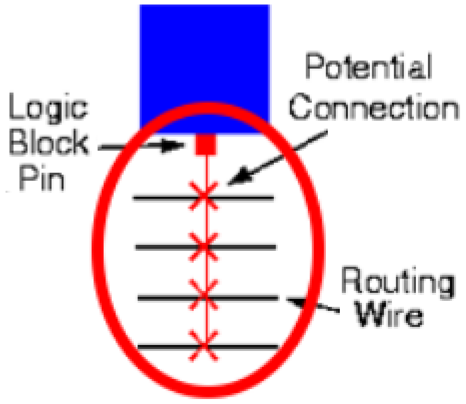


Programmable Routing



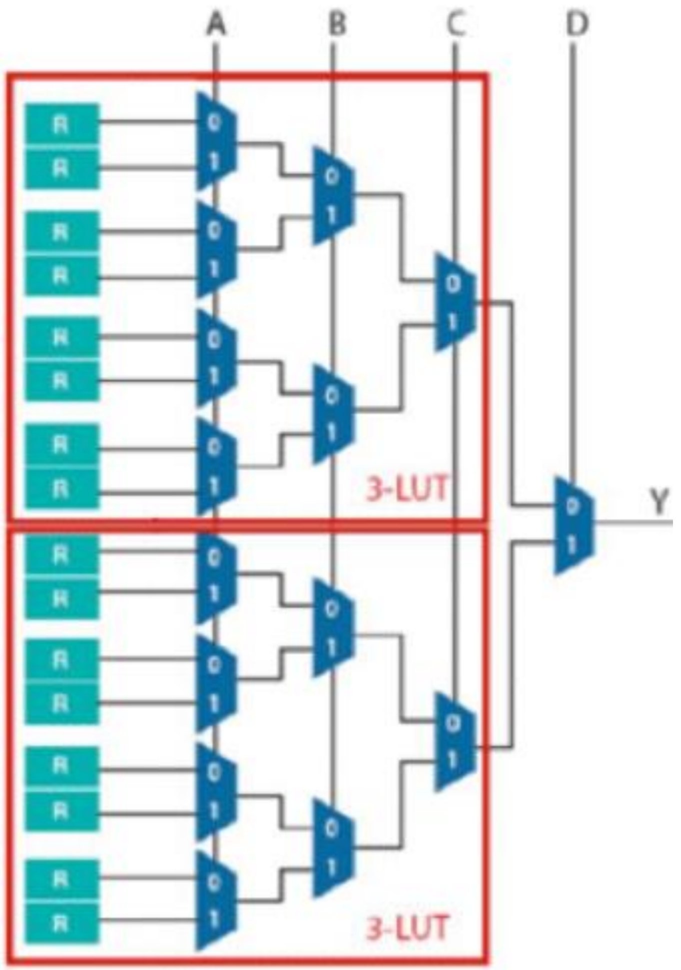
- Wiring channels between rows and column
- These are programmable
- Each wiring segment can be connected in one or many ways





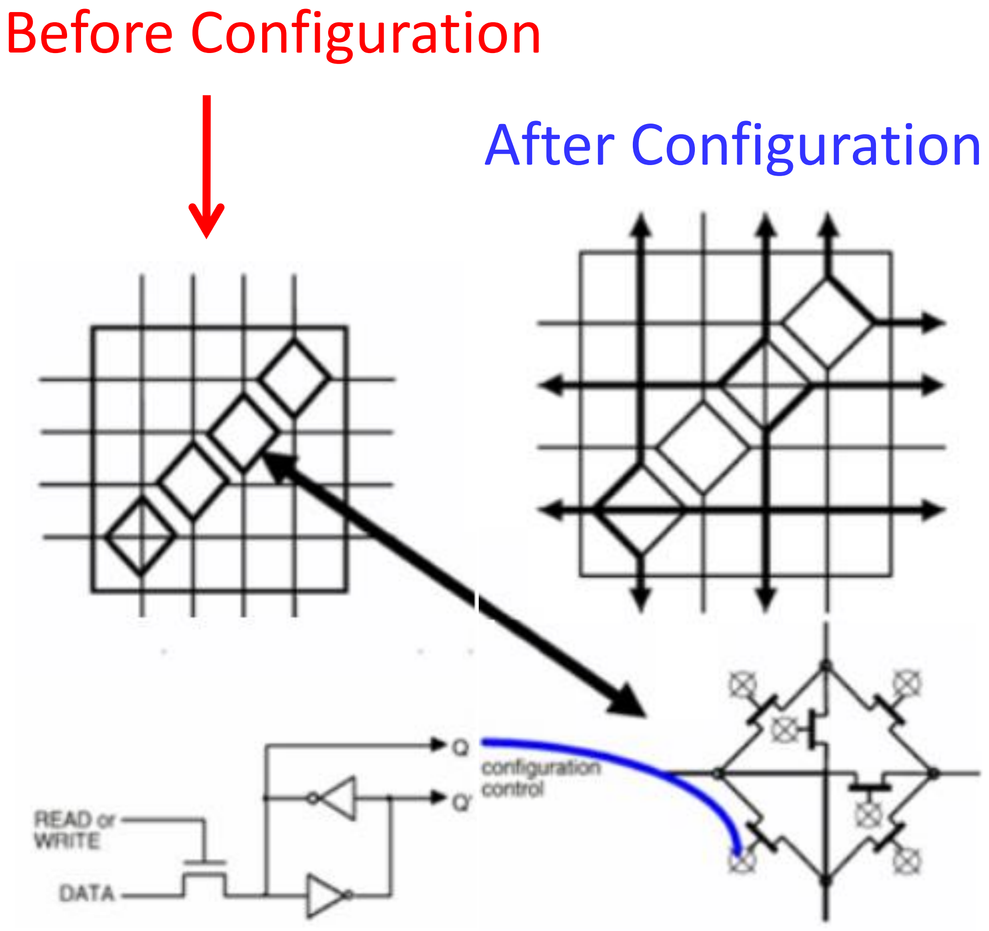
Idea of configuring FPGA

- Programming FPGA is not same as microprocessor
- We download bitstream (not a program) to an FPGA
- Programming FPGA is known as configuration
- LUTs are configured to implement the Boolean logic



Configuring the Routing of FPGA

- At each interconnect transistor switch OFF state
- Each switch is controlled by output of 1-bit configuration register
- Configuring the routing simply to put a ‘1’ or ‘0’in register
- Bitstream is either stored on local flash memory or download via computer
- Configuration happens on power-up



Digital Designs

- Washing machine control using basic AND and NOT gates
- Basics of OR gate and its application in industrial control
- Basics of NOT gate and its application in an eight bit ones complement circuit
- Basic NOT gate and its application in fuel level indicator
- Seat belt warning system using basic AND and NOT gates
- Basics of AND gate and its application in car wiper control
- Water level control using basic AND and NOT gates
- Electronic lock using basic logic gates
- Universal NAND gate and its application in level monitoring in chemical plant
- Universal NOR gate and its application in automobile alarm system
- XOR gate and its application in staircase light control
- Majority circuit using basic logic gates
- Cockpit warning light control using basic logic gates
- Build your own combinational logic circuit using generalized simulator

Low Fuel Warning System

Low Fuel Warning System

In a vehicle, the fuel level sensor provides a signal indicating whether the fuel level is above a certain threshold. A NOT gate can be used to trigger a warning light when the fuel level is below this threshold.

Inputs:

- **A**: Binary input signal from the fuel level sensor (1 if fuel level is above threshold, 0 if below).

Output:

- **Y**: Binary output signal to the warning light (1 to turn on the warning light, 0 to turn it off).

Industry Emergency Indicator

Emergency Stop System

Emergency Stop System: In an industrial environment, multiple emergency stop buttons are placed at different locations. Pressing any one of these buttons should stop the machinery to ensure safety.

Inputs:

- A: Signal from emergency stop button 1.
- B: Signal from emergency stop button 2.

Output:

- Y: Signal to stop the machinery (1 if any emergency stop button is pressed).

OR Gate:

- Output Y will be high (1) if either A or B is high (1), stopping the machinery.

Car Wiper Control System

Car Wiper Control System

Car wiper control system to ensure the wipers operate only when both the ignition is on and the wiper switch is activated.

Inputs:

- **A**: Binary input signal from the ignition switch (1 if ignition is on, 0 if off).
- **B**: Binary input signal from the wiper switch (1 if wiper switch is on, 0 if off).

Output:

- **Y**: Binary output signal to the wiper motor (1 to activate the wipers, 0 to deactivate them).

Water Level Control System

Water Level Control System

The water level control system should ensure that the water pump operates only when the water level is below a certain threshold and the system is enabled. Once the water reaches the desired level, the pump should automatically turn off.

Inputs:

L: Binary input signal from the low water level sensor (1 if water is below threshold, 0 otherwise).

H: Binary input signal from the high water level sensor (1 if water is at or above desired level, 0 otherwise).

E: Binary input signal for enabling the system (1 if system is enabled, 0 otherwise).

Outputs:

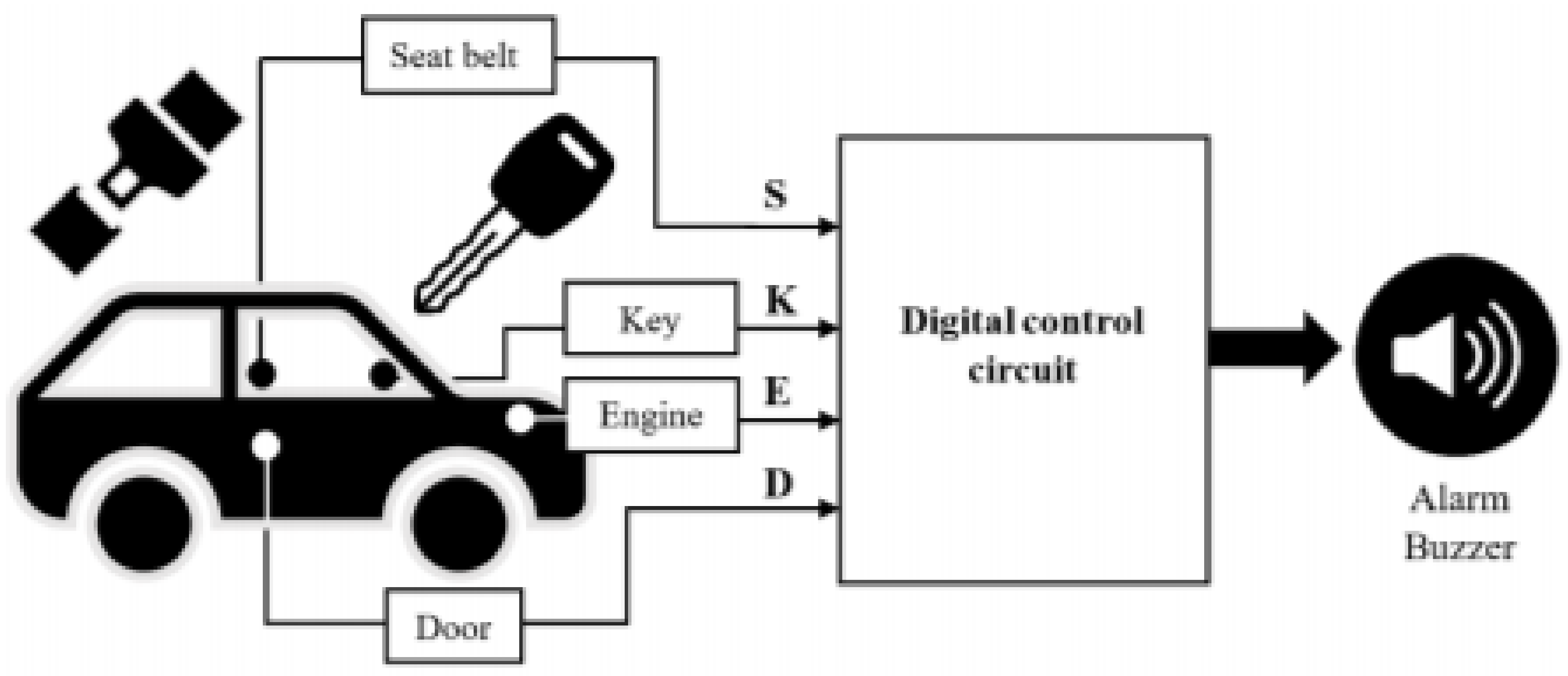
P: Binary output signal controlling the water pump (1 to turn on the pump, 0 to turn it off).

Water Level Control System

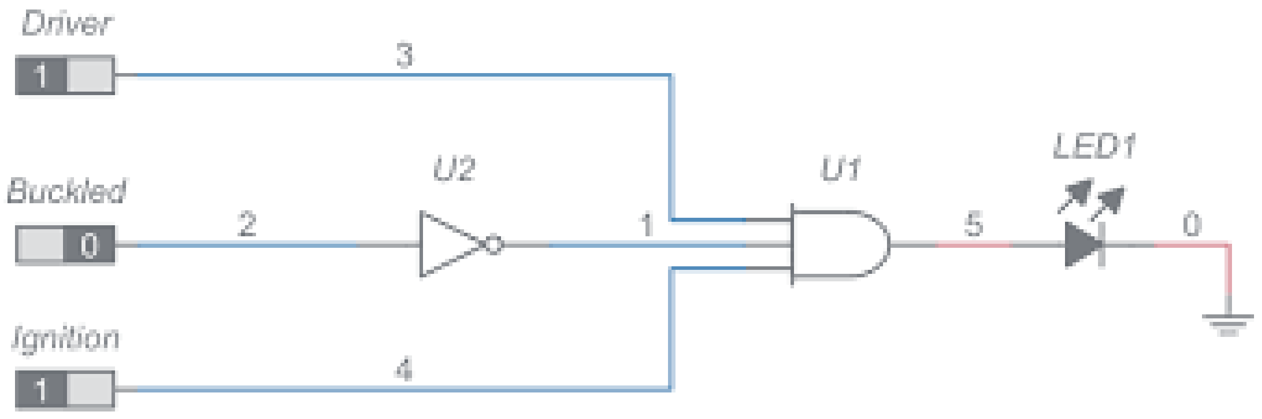
Low Level (L)	High Level (H)	System Enabled (E)	NOT H	Pump (P = NOT H AND L AND E)
0	0	0	1	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	0
1	0	0	1	0
1	0	1	1	1
1	1	0	0	0
1	1	1	0	0

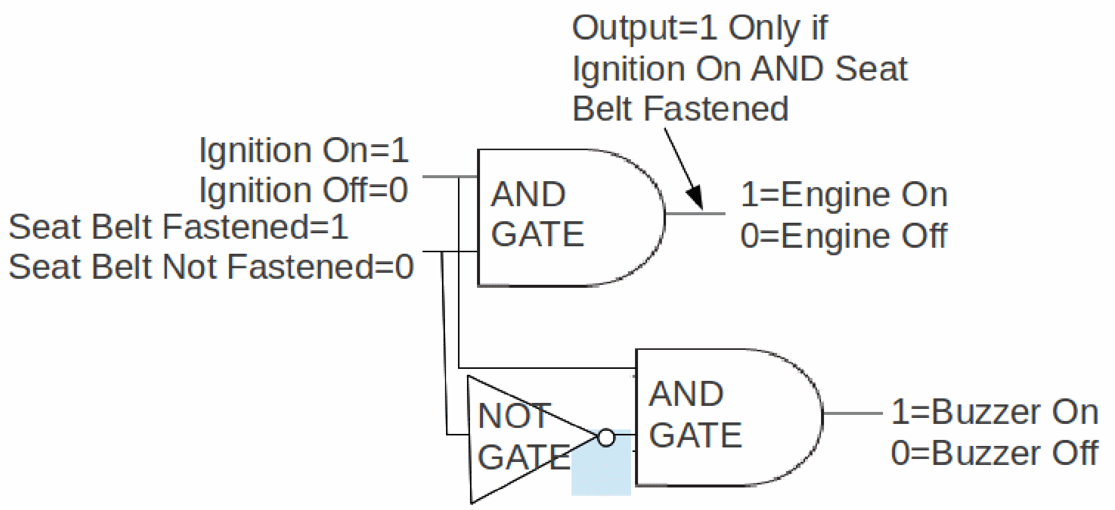
Seat Belt Warning Indicator

Seat Belt Indicator



Seat Belt Indicator





Electronic Lock

Electronic Lock

The electronic lock should only unlock when a specific combination of inputs is provided.

The lock should unlock only when A=1, B=0, and C=1.

Washing Machine Control

Washing Machine Control System

Problem Definition

The washing machine should only start its operation when the following conditions are simultaneously satisfied:

- Water is present
- Laundry is loaded
- The lid is closed

Solution : Washing Machine Control System

Inputs:

- **X**: Binary input signal indicating the presence of water (1 if water is present, 0 otherwise).
- **Y**: Binary input signal indicating the presence of laundry (1 if laundry is loaded, 0 otherwise).
- **Z**: Binary input signal indicating whether the lid is closed (1 if the lid is closed, 0 otherwise).

Output:

- C: Binary output signal controlling the washing machine (0 to start the machine, 1 to stop the machine).

Logic Design : Washing Machine Control System

AND Gate 1:

- Inputs: X, Y
- Output: A (X AND Y)
- Description: This gate ensures that both water and laundry are present.

AND Gate 2:

- Inputs: A, Z
- Output: B (A AND Z)
- Description: This gate ensures that the machine will only consider starting if both the previous condition (water and laundry present) is met and the lid is closed.

NOT Gate:

- Input: B
- Output: C (NOT B)
- Description: This gate inverts the signal, so the washing machine starts (C=0) only if all conditions are satisfied (B=1).

Logic Design : Washing Machine Control System

X Water	Y Laundry	Z Lid	A (X AND Y)	B (A AND Z)	C (NOT B)
0	0	0	0	0	1
0	0	1	0	0	1
0	1	0	0	0	1
0	1	1	0	0	1
1	0	0	0	0	1
1	0	1	0	0	1
1	1	0	1	0	1
1	1	1	1	1	0

- The designed control circuit ensures that the washing machine operates only when all required conditions (**water present, laundry loaded, lid closed**) are met.
- This prevents the machine from starting under unsafe or incorrect conditions, thereby **ensuring efficient and safe operation**.

Majority Circuit

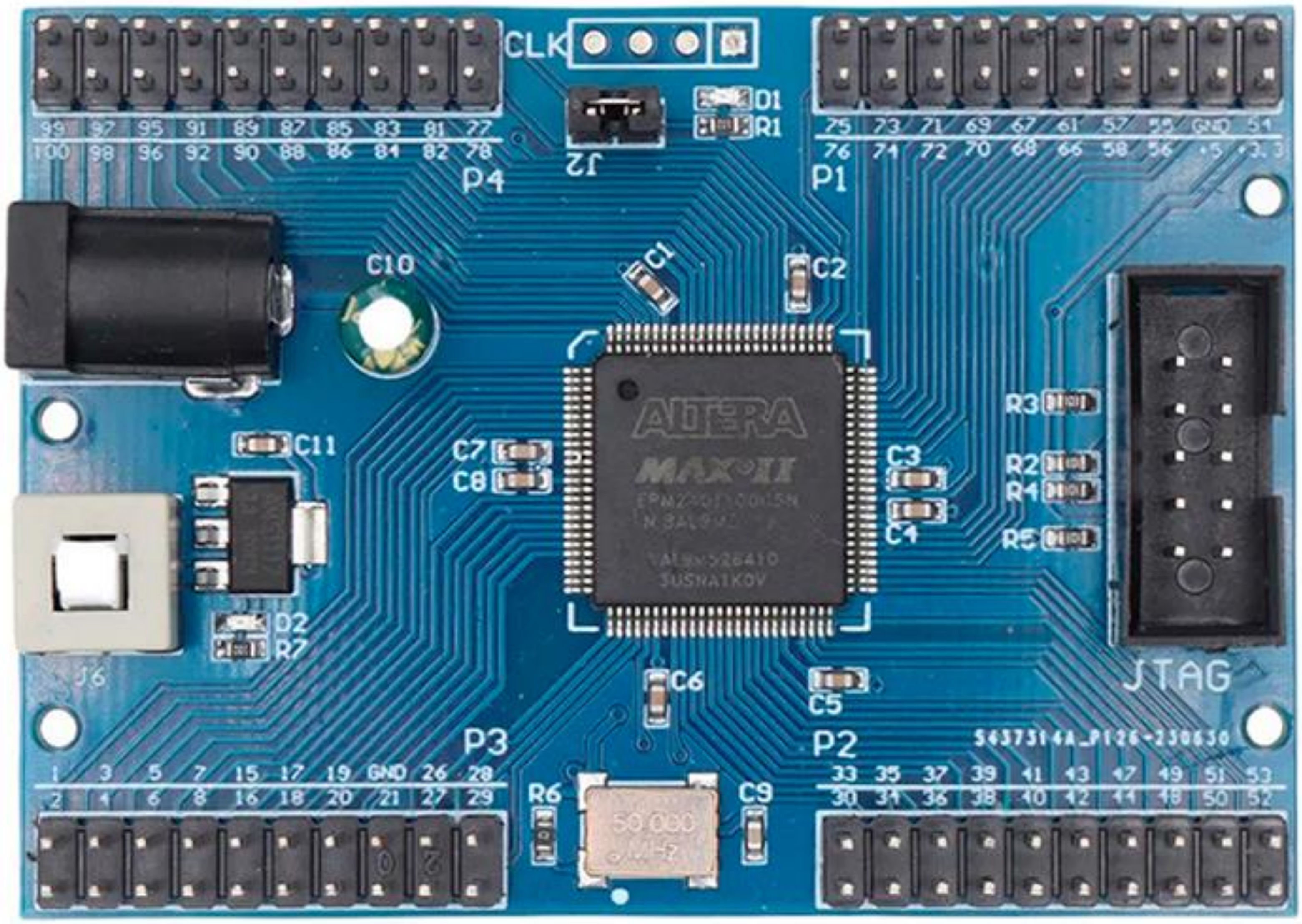
(Electronic Voting Machine)

Majority Circuit

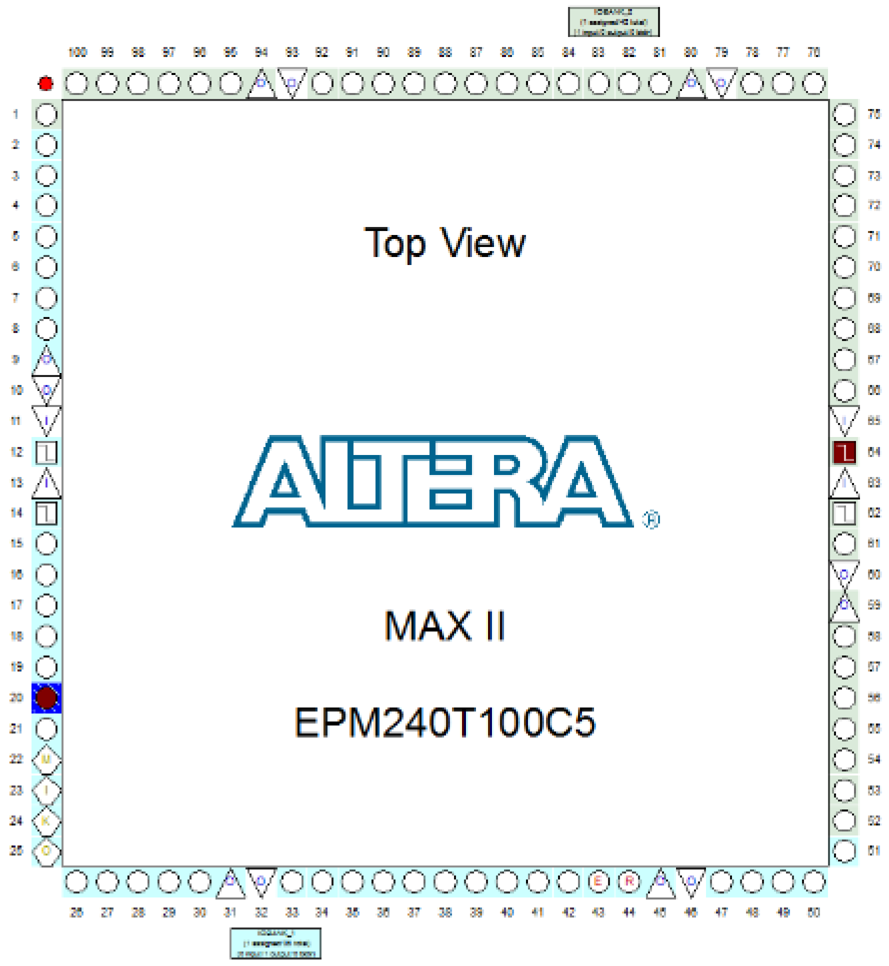
A majority circuit outputs true (1) if the majority of the inputs are true (1). For three inputs, the output should be true if at least two of the three inputs are true. (Voting Machine/Committee Decisions/Fault Tolerant Systems/ etc…..)

A	B	C	A AND B	A AND C	B AND C	Y (Majority)
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	0	0	0
1	0	1	0	1	0	1
1	1	0	1	0	0	1
1	1	1	1	1	1	1

CPLD Development Board



CPLD Development Board



HDL Programming

Verilog Code NOT Gate

Verilog Code

```
module inverter ( input wire a, output wire y );
        assign y = ~a;
endmodule
```

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

entity inverter is
    Port (
        a : in STD_LOGIC;   -- Input signal
        y : out STD_LOGIC   -- Output signal
    );
end inverter;

architecture Behavioral of inverter is
begin
    y <= not a;  -- Output is the negation
(inversion) of the input
end Behavioral;
```

VHDL Code

Slower Clock

```
module slower_clock(input clk, output reg clkOutput);

reg [22:0] counter;

always @(posedge clk)
    if(counter==5000000) counter <= 0; else counter <= counter+1;

always @(posedge clk)
    if(counter==5000000)
        begin
            clkOutput <= ~clkOutput;
        end
endmodule
```

LED is blinking at 5Hz
clock pins (12, 14, 62)

```
module ledblink(clk,led);
input clk; output led; reg led;
reg[23:0] cnt;
always @(posedge clk) begin cnt<= cnt + 1'b1; led<=cnt[23];
end
endmodule
```

Fine Tuning with PWM

```verilog
module LED_PWM(clk, PWM_input, LED);
input clk;
input [3:0] PWM_input;     // 16 intensity levels
output LED;

reg [4:0] PWM;
always @(posedge clk) PWM <= PWM[3:0] + PWM_input;

assign LED = PWM[4];
endmodule
```

```verilog
module LEDglow(clk, LED);
input clk;
output LED;

reg [23:0] cnt;
always @(posedge clk) cnt <= cnt+1;

reg [4:0] PWM;
wire [3:0] intensity = cnt[23] ? cnt[22:19] : ~cnt[22:19];
// ramp the intensity up and down
always @(posedge clk) PWM <= PWM[3:0] + intensity;

assign LED = PWM[4];
endmodule
```

Verilog Code for LED Blinking

```
module counter(clk, out);
  input clk;       // Clock input
  output out;      // Output signal
  reg out;         // Register for the output
  reg [23:0] count; // 24-bit register for the counter
  // Always block triggered on the positive edge of the clock
  always @(posedge clk) begin
    count <= count + 1'b1; // Increment the counter by 1
    out <= count[23];      // Assign the 24th bit of the counter to the output
  end
endmodule
```

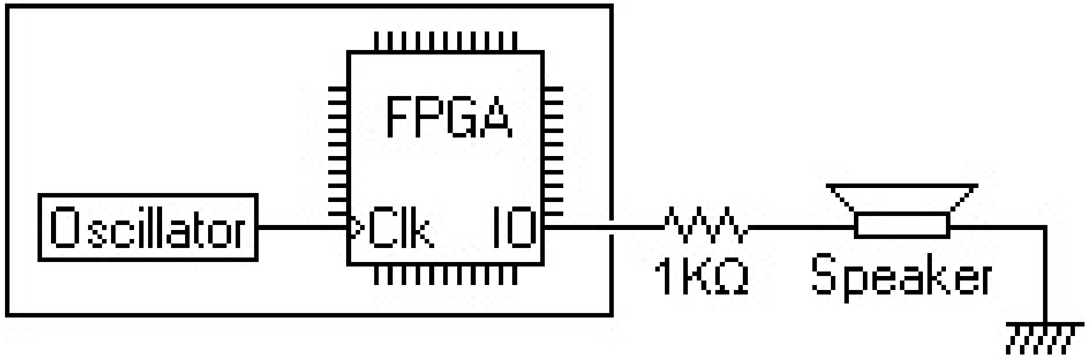
VHDL Code for LED Blinking

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

entity counter is
    Port (
        clk : in STD_LOGIC;  -- Clock input
        out : out STD_LOGIC  -- Output signal
    );
end counter;

architecture Behavioral of counter is
    signal count : STD_LOGIC_VECTOR(23 downto 0) := (others => '0'); -- 24-bit counter signal
begin
    process(clk)
    begin
        if rising_edge(clk) then
            count <= count + 1;        -- Increment the counter
            out <= count(23);          -- Assign the 24th bit of the counter to the output
        end if;
    end process;
end Behavioral;
```

Music Generation

Music

- Simple beeps
- Sirens
- Notes
- Tune



Music: Simple Beep (Verilog)

```
module music(clk, speaker);
input clk;
output speaker;

// first create a 16bit binary counter
reg [15:0] counter;
always @(posedge clk) counter <= counter+1;

// and use the most significant bit (MSB) of the counter to drive
the speaker
assign speaker = counter[15];
endmodule
```

Music: Simple Beep (VHDL)

```
entity music is
    Port (
        clk     : in  STD_LOGIC;  -- Clock input
        speaker : out STD_LOGIC   -- Speaker output
    );
end music;

architecture Behavioral of music is
signal counter : STD_LOGIC_VECTOR(15 downto 0) := (others => '0');
begin
process(clk)
    begin
        if rising_edge(clk) then
            counter <= counter + 1;
        end if;
    end process;

speaker <= counter(15);
end Behavioral;
```

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

Classroom and Feedback

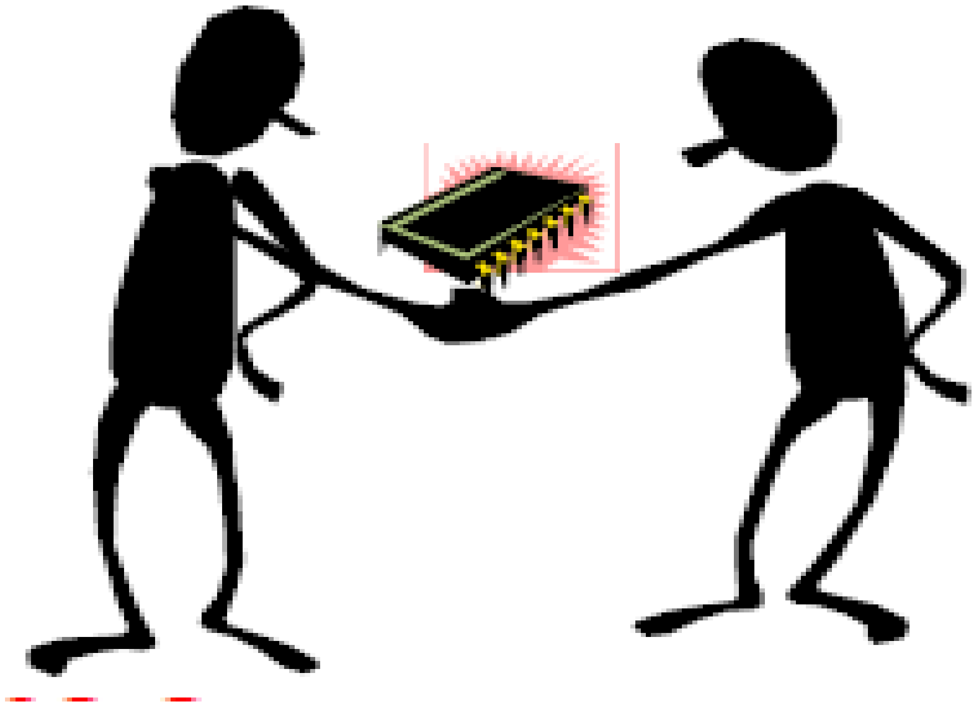
CDAC VLSI Verilog April 2025

WhatsApp group

Scan this QR code using the WhatsApp camera to join this group

Pre Course Delivery Feedback

Google Classroom



Thank you !

Happy Learning