IASc-INSA-NASI Summer Research Fellowship Programme (SRFP) 2025

Internship Project Report

Project Title:

Digital Design & Embedded Systems

Submitted by:

Ayush Tiwari
(Summer Research Fellow, IASc-INSA-NASI SRFP 2025)

Under the supervision of:

Dr. G. V. V. Sharma
Associate Professor
Department of Electrical Engineering
Indian Institute of Technology Hyderabad

Host Institute:

Indian Institute of Technology Hyderabad
Kandi, Sangareddy – 502285
Telangana, India

Duration:

14th May – 8th July 2025

Contents

1: Installation

1.1 Termux

1. Install Termux from the Play Store or F-Droid on your Android device.
2. Launch Termux and update its packages using:
   pkg update && pkg upgrade
3. Install Debian within Termux by following the instructions from this repository:
   https://github.com/gadepall/fwc-1
4. Confirm successful installation of Debian by checking its version inside Termux.

1.2 PlatformIO

1. Inside Debian (in Termux), install essential AVR toolchains with:
   apt install avra avrdude gcc-avr avr-libc
2. Download the PlatformIO installer script by referring to:
   https://docs.platformio.org/en/stable/core/installation/methods/installer-script.html#super-quick-macos-linux
3. Change to your project directory in Debian:
   cd ide/piosetup/codes
4. Build your project with PlatformIO:
   pio run
5. Connect the Arduino board to your laptop or Raspberry Pi, then upload the firmware using:
   pio run -t nobuild -t upload
6. Confirm successful upload — the LED beside pin 13 on your Arduino Uno should start blinking.

1.3 ArduinoDroid

1. Download ArduinoDroid from APKPure or the Play Store.
2. Install it on your Android phone and grant all requested permissions.
3. Connect the Arduino Uno to your phone using a USB-OTG cable.
4. Open ArduinoDroid and navigate to:
   Actions → Upload → Upload Precompiled
5. Locate and select the *.pio/build/uno/firmware.hex* file from your working directory to upload the precompiled HEX file.
6. After successful flashing, the LED beside pin 13 should start blinking.

2: Seven Segment Display

2.1 Components

1. The main components required are:
   - Resistor: 220 Ω (1 no.)
   - Arduino board (1 no.)
   - Seven Segment Display (1 no.)
   - 7447 decoder IC (1 no.)
   - 7474 flip-flop IC (2 nos.)
   - Jumper wires (about 20)
2. A breadboard is used to connect the components with proper voltage rails.
3. The breadboard rows are internally connected (green segments for power/ground, blue columns for signals).
4. This allows efficient distribution of 5V and GND to all connected components from a single Arduino supply.

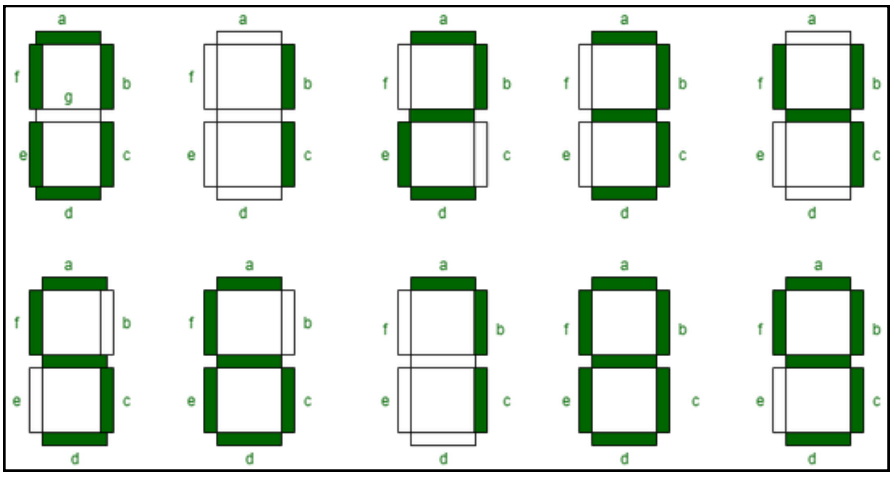
2.2 Display Control through Hardware

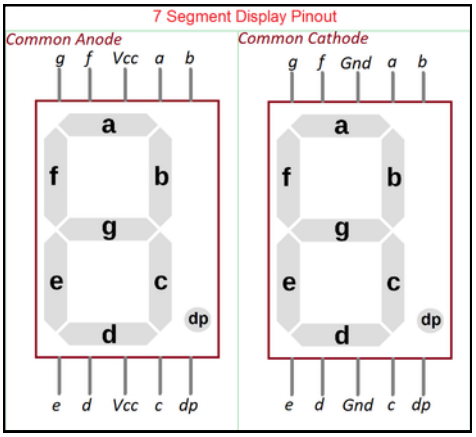
2.2.1 Powering the Display

5. The seven segment display has eight pins labeled a, b, c, d, e, f, g and a dot segment.
6. These segments are activated with an active LOW signal, meaning they glow when connected to ground (GND = 0).
7. Plug the display module into the breadboard as per the pinout diagram.
8. Connect the common pin (COM) of the display to the 5V line through a resistor (220Ω).
9. Connect the DOT pin of the display to the GND line.
10. Connect the breadboard power rails from the Arduino's 5V and GND pins for consistent power.

2.2.2 Controlling the Display

11. Connect each display segment pin to an Arduino I/O pin via a current-limiting resistor if needed.
12. For example:
    - Arduino pin 2 → segment a
    - Arduino pin 3 → segment b
    - and so on up to segment g
13. Once connected, you can control each segment by sending LOW signals to illuminate them.





4. As an exercise, connect only segments b and c to GND to display digit “1”.
5. Repeat by activating other segments to display other numbers (2, 3, … 9).
6. Record these patterns in a table for reference.

2.3 Display Control through Software

1. After completing the wiring, program the Arduino using the Arduino IDE.
2. Use a simple C++ program (for example, sevenseg.cpp) to drive the display.
3. Inside the code, map the Arduino pins to the correct segments using digitalWrite.
4. Generate numbers 0–9 by setting different segments LOW in the program’s logic.
5. Modify the loop section of the code to scroll through digits with delays.
6. Upload the program and observe the display change in real time on the breadboard.

3: 7447 – BCD to Seven Segment Decoder

3.1 Introduction

1. The IC 7447 is a BCD (Binary-Coded Decimal) to Seven Segment Decoder.
2. It converts a 4-bit binary input (A, B, C, D) to seven outputs (ā–ḡ) for directly driving common-anode 7-segment displays.
3. This simplifies wiring by reducing the number of digital pins required from the controller (Arduino).
4. The decoder IC activates specific segments to display digits 0–9 based on the BCD input.

3.2 Hardware Setup

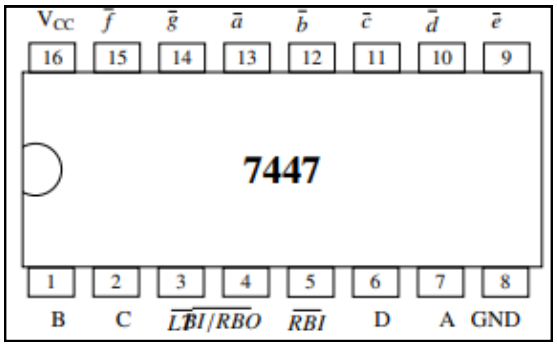
1. Connect the 7447 segment outputs (ā to ḡ) to the corresponding a–g pins of the 7-segment display.
2. Wire the BCD input pins of the 7447 to Arduino pins:
    - D → Arduino pin 5
    - C → Arduino pin 4
    - B → Arduino pin 3
    - A → Arduino pin 2
3. Provide power supply to the 7447 IC:
    - VCC (pin 16) → 5V
    - GND (pin 8) → Ground
4. Connect control pins LT, RBI, and BI/RBO as required (usually grounded for normal operation).
5. After connections, the IC will automatically light up the correct segments based on BCD inputs.

3.3 Software Testing with Arduino

1. Connect the Arduino to your computer and open the Arduino IDE.
2. Navigate to the source file location:
   ide/7447/codes/gvv_ard_7447/gvv_ard_7447.cpp
3. In the code, pins 2–5 are used to send binary input (A–D) to the 7447 IC.
4. Upload the sketch and verify that digits 0–9 appear sequentially on the 7-segment display.
5. To test the logic for incrementing decoder, use:
    - *ide/7447/codes/inc_dec/inc_dec.ino*
    - This sketch implements logic for incrementing binary numbers and displaying the output.
6. You can also test input through physical switches by modifying:
    - *ide/7447/codes/ip_inc_dec/ip_inc_dec.cpp*
    - This code takes input (e.g., number 5: 0101) and displays it on the 7-segment using 7447.

3.4 Truth Table & Logic Understanding

1. Each BCD input corresponds to a unique digit output on the display:
    - 0000 (0) → All segments except g
    - 0001 (1) → Only b and c
    - ... and so on up to 1001 (9)



Z	Y	X	W	D	C	B	A
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0

2. Table 3.2 in the manual provides all mappings from BCD to display.
3. The logic behind the IC can be recreated manually using truth tables and Boolean equations.
4. Students are encouraged to experiment with writing their own decoder logic in Arduino by analyzing the output and using functions like digitalWrite().

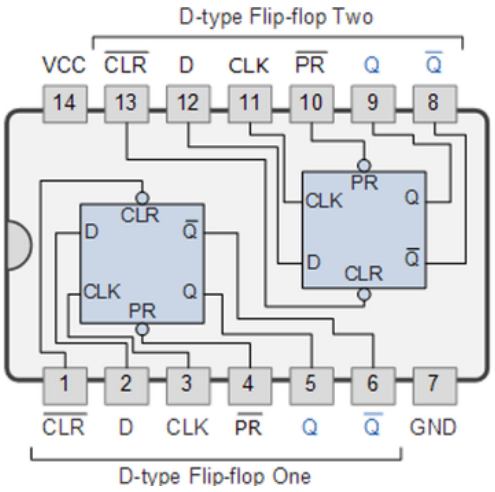
4: 7474 – Dual D Flip-Flop

4.1 Introduction

1. The IC 7474 is a Dual D-type Positive-Edge-Triggered Flip-Flop.
2. Each flip-flop within the IC has separate inputs: Data (D), Clock (CLK), Preset (PRE), and Clear (CLR), and outputs Q and $\bar{Q}$.
3. This IC is used in sequential logic circuits for storing and transferring binary data at clock edges.
4. In this course, it is used to implement a counter circuit interfaced with a 7-segment display via the 7447 decoder.

4.2 Hardware Connections

5. One 7474 IC contains two D flip-flops with independent inputs and outputs.
6. Connect power pins:
   - Pin 14 → VCC (5V)
   - Pin 7 → GND
7. Connect D and CLK inputs to Arduino pins for control.
8. Use Preset (PRE) and Clear (CLR) pins to reset or set the flip-flop as needed (often active low).
9. Output Q of the first flip-flop can be fed to the D input of the second for cascading.



	INPUT				OUTPUT				CLOCK		5V			
	W	X	Y	Z	A	B	C	D						
Ar-duino	D6	D7	D8	D9	D2	D3	D4	D5	D13					
7474	5	9			2	12			CLK1	CLK2	1	4	10	13
7474			5	9			2	12	CLK1	CLK2	1	4	10	13
7447					7	1	2	6			16			

4.3 Testing the 7474 as a Counter

1. The IC 7474 can be used to construct a binary counter when connected properly.
2. The count output from the flip-flops is sent to the 7447 decoder, which drives a 7-segment display.
3. This setup allows visual tracking of the binary count from 0 to 9.
4. Test circuit using the provided code file:
   - *ide/7474/codes/gvv_ard_7474/gvv_ard_7474.cpp*
5. Upload this sketch to Arduino and observe the display incrementing with each clock pulse.

4.4 Clock Generation and Observations

6. Clock pulses can be generated from Arduino pins via digitalWrite() toggles in the code.
7. Alternatively, physical push-buttons may be used to manually trigger clock edges.
8. The output Q changes only at the rising edge of the CLK signal.
9. This experiment demonstrates the basic behavior of D flip-flops, which is foundational to sequential digital systems.

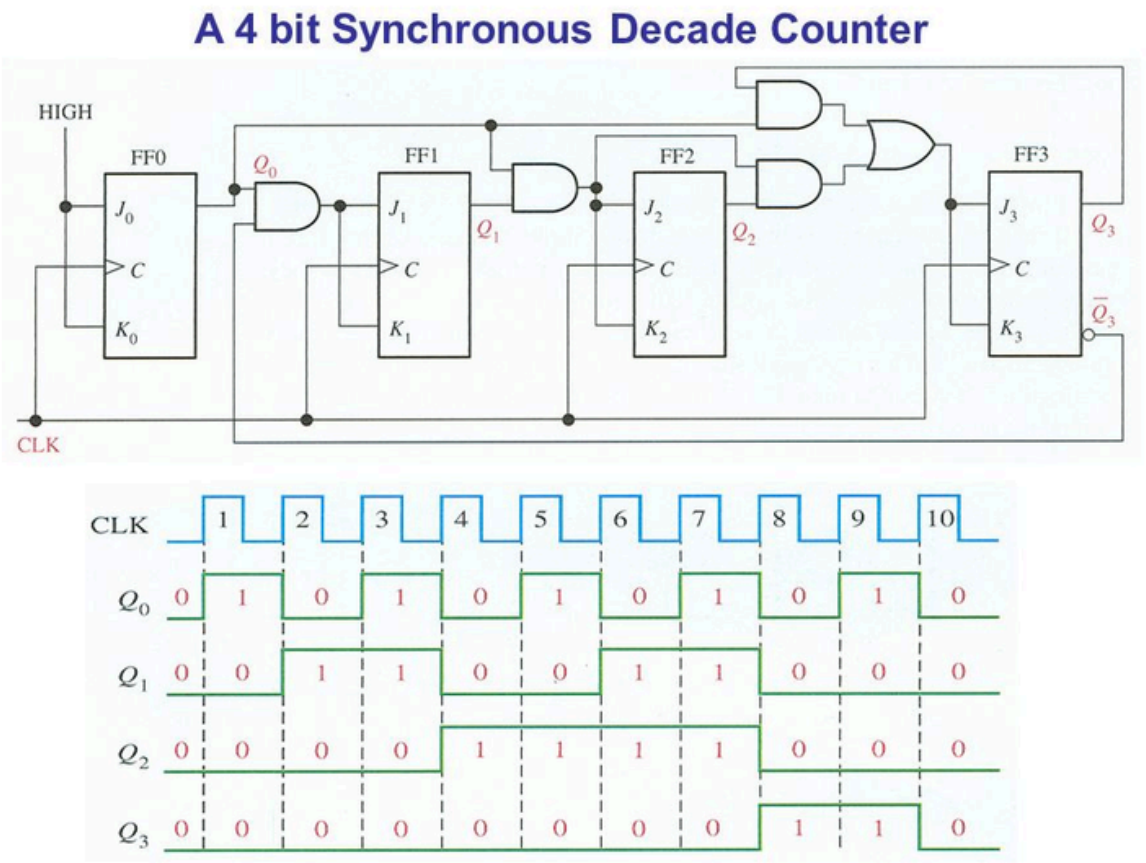
5: Finite State Machines (FSM)

5.1 Introduction

1. A Finite State Machine (FSM) is a sequential digital circuit with a finite number of defined states.
2. It transitions between these states based on input signals and clock pulses.
3. FSMs are commonly used in control systems, pattern detectors, counters, and protocol design.

5.2 Implementation using Flip-Flops

1. FSMs are constructed using D flip-flops (like 7474) to store the current state.
2. Combinational logic determines the next state and output based on present state and inputs.
3. The FSM transitions on the rising edge of the clock signal.



5.3 Course Experiments

1. A basic FSM was implemented using 7474 flip-flops, connected to a 7447 decoder and a 7-segment display.
2. Clock pulses were generated using the Arduino to drive the FSM transitions.
3. The FSM was verified by observing state changes visually through display outputs.

5.4 Source File Reference

1. The main code used for FSM testing is located at: *ide/fsm/codes/fsm_1/fsm_1.cpp*
2. This sketch includes logic for defining state transitions and clock-driven output.

6: Assembly Programming

6.1 Introduction

1. Assembly language is a low-level programming language that provides direct control over hardware.
2. It is written using mnemonics and opcodes, each corresponding to a specific instruction executed by the CPU.
3. In this project, assembly programming is done for the AVR microcontroller on the Arduino Uno board.
4. The AVR-GCC toolchain and AVRA assembler are used for compiling and uploading assembly code.

6.2 Setting Up the Environment

5. Use the Debian environment inside Termux, as set up during installation.
6. Ensure the following packages are installed:
   - *avra* (assembler)
   - *avrdude* (uploader)
7. Navigate to the folder:
   - *ide/avr/codes/blink/blink.asm*
   - This is a sample assembly file that blinks the LED on pin 13.

6.3 Sample Program: Blinking LED

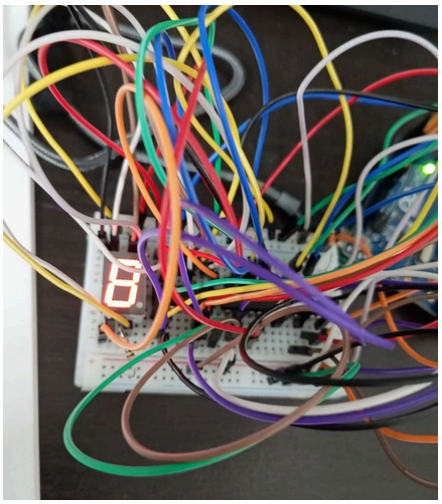
8. The blink.asm file demonstrates basic port manipulation:
   - Set pin 13 (PB5) as output
   - Toggle the pin HIGH and LOW with delays
9. Use the following commands to compile and upload:
   - avra blink.asm → compiles to blink.hex
   - avrdude -c arduino -p m328p -P /dev/ttyUSB0 -b 115200 -U flash:w:blink.hex → uploads to Arduino
10. Upon successful upload, the LED on pin 13 blinks continuously.

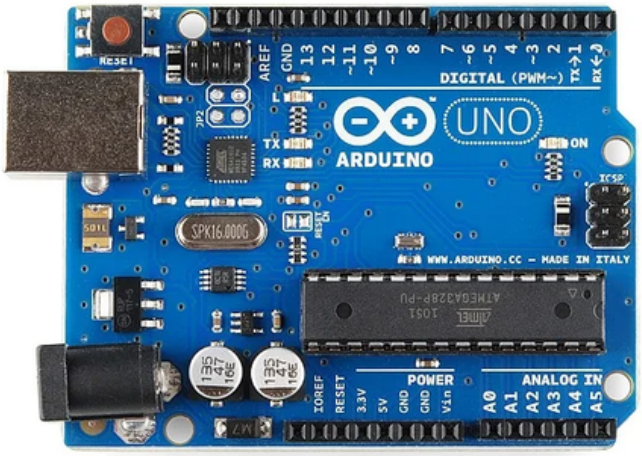
6.4 Understanding the Code

11. Key instructions used in the blink.asm file:
    - *ldi* → load immediate data into a register
    - *out* → write to an I/O port
    - *sbi / cbi* → set/clear individual bits
    - *rjmp* → relative jump (used for looping)
12. A delay loop is implemented using decrement and branch instructions.
13. The program uses direct register-level programming for PORTB.

6.5 Exercises and Modifications

1. Try modifying the delay to change the blink frequency.
2. Create your own program to blink multiple LEDs on different ports.
3. Add an input switch and toggle LED based on button press.
4. These files are available at:
    - *ide/avr/codes/blink/blink.asm*
    - *ide/avr/codes/blink_n/blink_n.asm*
    - *ide/avr/codes/switch/switch.asm*





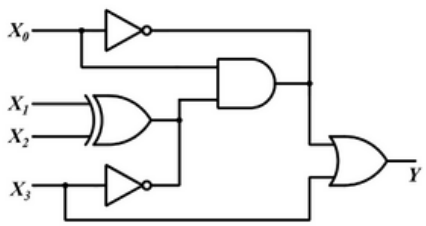
6.6 Benefits of Learning Assembly

1. Gives complete control over microcontroller behavior.
2. Helps understand hardware architecture and registers.
3. Useful in time-critical and resource-constrained embedded applications.
4. Encourages optimization and minimal memory usage.

6.7 Assignment

https://github.com/ayush-1910/GATE-Papers-fwc-1/tree/main/fwc-1/assembly/assignment

Q.47 The logic gates shown in the digital circuit below use strong pull-down nMOS transistors for LOW logic level at the outputs. When the pull-downs are off, high-value resistors set the output logic levels to HIGH (i.e. the pull-ups are weak). Note that some nodes are intentionally shorted to implement "wired logic". Such shorted nodes will be HIGH only if the outputs of all the gates whose outputs are shorted are HIGH.



The number of distinct values of $X_3X_2X_1X_0$ (out of the 16 possible values) that give $Y = 1$ is _____.

7: Embedded C Programming

7.1 Introduction

1. Embedded C is a variant of the C language used for programming microcontrollers and embedded systems.
2. It provides low-level access to hardware registers and ports, allowing full control of the microcontroller.
3. In this project, Embedded C is used to control the AVR microcontroller (ATmega328P) on the Arduino Uno.

7.2 Programming Environment

4. The programming is done using the AVR-GCC compiler, which is part of the Debian setup in Termux.
5. Files are located under:
   - *ide/avr/codes/blink_c/blink_c.c*
   - *ide/avr/codes/blink_c/switch_c.c*
6. Compilation is done using:
   - *avr-gcc -mmcu=atmega328p -Os -o blink_c.elf blink_c.c*
   - *avr-objcopy -O ihex -R .eeprom blink_c.elf blink_c.hex*
   - *avrdude -c arduino -p m328p -P /dev/ttyUSB0 -b 115200 -U flash:w:blink_c.hex*

7.3 Sample Code: Blinking LED

7. The blink_c.c program sets PB5 (pin 13) as output and toggles it ON and OFF in a loop.
8. Registers like DDRB (Data Direction Register) and PORTB are directly accessed in code.
9. A simple delay is implemented using a for loop.

7.4 Switch-Controlled LED

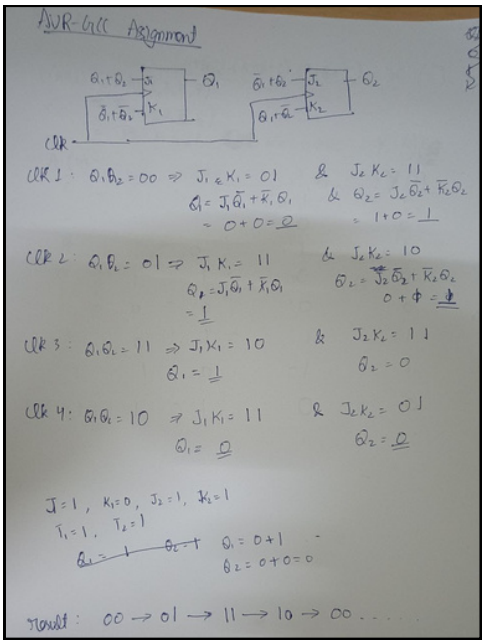
10. The switch_c.c file demonstrates using PB0 as input to control PB5.
11. When the button connected to PB0 is pressed, PB5 (LED) is turned ON.
12. This demonstrates use of conditional statements and input handling in Embedded C.

7.5 Importance

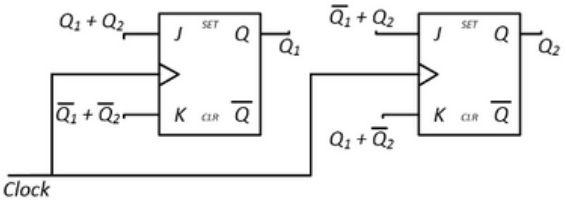
13. Embedded C offers more readability and maintainability than assembly while still offering low-level hardware access.
14. It is widely used in industrial embedded systems development.
15. Understanding Embedded C is essential for real-time embedded applications.

7.6 Assignment

https://github.com/ayush-1910/GATE-Papers-fwc-1/tree/main/fwc-1/avr-gcc/assignment

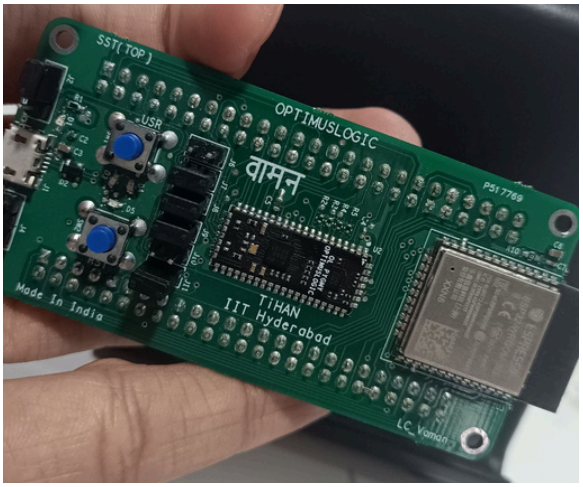


Q.44 A 2-bit synchronous counter using two J-K flip flops is shown. The expressions for the inputs to the J-K flip flops are also shown in the figure. The output sequence of the counter starting from $Q_1Q_2 = 00$ is

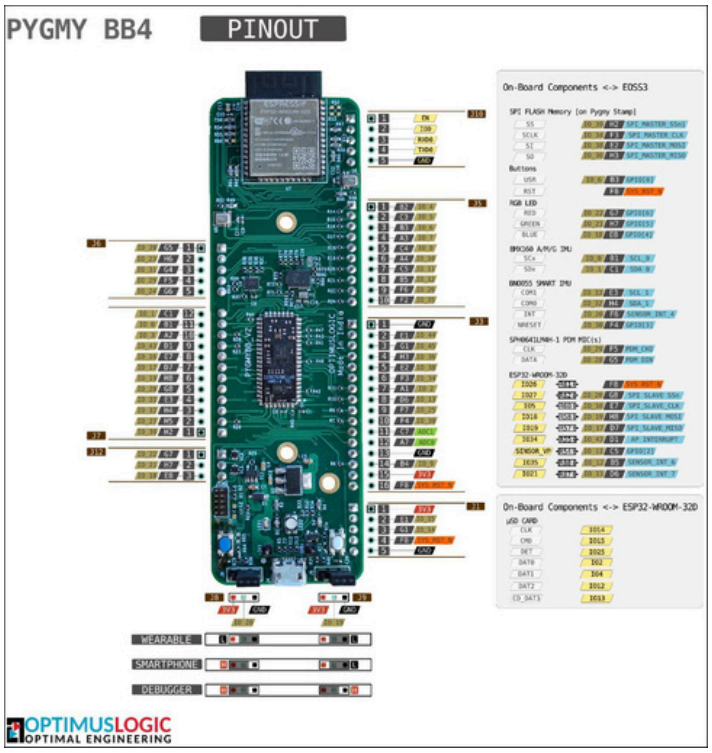


8: Vaman-ESP32

8.1 Vaman – Introduction



1. Vaman (वामन) is IIT Hyderabad's indigenously developed hardware board, created through a collaboration between TiHAN–IIT Hyderabad and OptimusLogic • Systems India.
2. The board integrates key components:
   - QuickLogic QORC-based programmable logic (FPGA + ARM)
   - Espressif ESP-32 WiFi + BLE module for wireless connectivity
   - 100 GPIO pins (2×50), USB-Micro port for power and programming
3. Designed to promote hardware–software co-design across Android, Raspberry Pi, and PC platforms; it's fully open-source in hardware, firmware, and tools.
4. It’s IIT Hyderabad’s flagship platform for autonomous UAV/UGV kits, AI/ML, VLSI, FPGA, and microcontroller labs since 2021.



PD64		
IO Locatic	Alias	IO Type
B1	IO_0	BIDIR
C1	IO_1	BIDIR
A1	IO_2	BIDIR
A2	IO_3	BIDIR
B2	IO_4	BIDIR
C3	IO_5	BIDIR
B3	IO_6	BIDIR
A3	IO_7	BIDIR/CLOCK
D4	IO_8	BIDIR/CLOCK
B4	IO_9	BIDIR
A4	IO_10	BIDIR
C5	IO_11	BIDIR
B5	IO_12	BIDIR
D6	IO_13	BIDIR
A5	IO_14	BIDIR
C6	IO_15	BIDIR
E7	IO_16	BIDIR
D7	IO_17	BIDIR
E8	IO_18	BIDIR
H8	IO_19	BIDIR
G8	IO_20	BIDIR
H7	IO_21	BIDIR
G7	IO_22	BIDIR/CLOCK
H6	IO_23	BIDIR/CLOCK
G6	IO_24	BIDIR/CLOCK
F7	IO_25	BIDIR
F6	IO_26	BIDIR
H5	IO_27	BIDIR
G5	IO_28	BIDIR
F5	IO_29	BIDIR
F4	IO_30	BIDIR
G4	IO_31	BIDIR
H4	IO_32	SDIOMUX
F3	IO_33	SDIOMUX
F3	IO_34	SDIOMUX
F2	IO_35	SDIOMUX
H3	IO_36	SDIOMUX
G2	IO_37	SDIOMUX
F2	IO_38	SDIOMUX
H2	IO_39	SDIOMUX
D2	IO_40	SDIOMUX
F1	IO_41	SDIOMUX
H1	IO_42	SDIOMUX
D1	IO_43	SDIOMUX
E1	IO_44	SDIOMUX
G1	IO_45	SDIOMUX

PU64		
IO Locatic	Alias	IO type
4	IO_0	BIDIR
5	IO_1	BIDIR
6	IO_2	BIDIR
2	IO_3	BIDIR
3	IO_4	BIDIR
64	IO_5	BIDIR
62	IO_6	BIDIR
63	IO_7	BIDIR/CLOCK
61	IO_8	BIDIR/CLOCK
60	IO_9	BIDIR
59	IO_10	BIDIR
57	IO_11	BIDIR
56	IO_12	BIDIR
55	IO_13	BIDIR
54	IO_14	BIDIR
53	IO_15	BIDIR
40	IO_16	BIDIR
42	IO_17	BIDIR
38	IO_18	BIDIR
36	IO_19	BIDIR
37	IO_20	BIDIR
39	IO_21	BIDIR
34	IO_22	BIDIR/CLOCK
33	IO_23	BIDIR/CLOCK
32	IO_24	BIDIR/CLOCK
31	IO_25	BIDIR
30	IO_26	BIDIR
28	IO_27	BIDIR
27	IO_28	BIDIR
26	IO_29	BIDIR
25	IO_30	BIDIR
23	IO_31	BIDIR
22	IO_32	SDIOMUX
21	IO_33	SDIOMUX
20	IO_34	SDIOMUX
18	IO_35	SDIOMUX
17	IO_36	SDIOMUX
15	IO_37	SDIOMUX
16	IO_38	SDIOMUX
11	IO_39	SDIOMUX
13	IO_40	SDIOMUX
14	IO_41	SDIOMUX
10	IO_42	SDIOMUX
7	IO_43	SDIOMUX
8	IO_44	SDIOMUX
9	IO_45	SDIOMUX

WN42		
IO Locatic	Alias	IO Type
A7	IO_0	BIDIR
B7	IO_1	BIDIR
C7	IO_3	BIDIR
A6	IO_6	BIDIR
B6	IO_8	BIDIR/CLOCK
A5	IO_9	BIDIR
B5	IO_10	BIDIR
A4	IO_14	BIDIR
B4	IO_15	BIDIR
E1	IO_16	BIDIR
D1	IO_17	BIDIR
C1	IO_19	BIDIR
F2	IO_20	BIDIR
E2	IO_23	BIDIR/CLOCK
D2	IO_24	BIDIR/CLOCK
D3	IO_25	BIDIR
F3	IO_28	BIDIR
E3	IO_29	BIDIR
F4	IO_30	BIDIR
E4	IO_31	BIDIR
D5	IO_34	SDIOMUX
F5	IO_36	SDIOMUX
E6	IO_38	SDIOMUX
F6	IO_39	SDIOMUX
D7	IO_43	SDIOMUX
E7	IO_44	SDIOMUX
F7	IO_45	SDIOMUX

8.2 ESP-32 Module

1. The onboard ESP-32 module provides dual-core WiFi and Bluetooth Low Energy connectivity.
2. Programming is supported through standard toolchains (Espressif ESP-IDF, Arduino-ESP32), and integrates easily with the Vaman board via shared USB power and GPIO routing.
3. Firmware development is done on ide/vaman-esp32/ directory, containing setup, WiFi, BLE, and GPIO control examples.
4. Example application: a simple WiFi–Blinky that connects to an access point and toggles GPIO outputs, demonstrating basic wireless control and hardware interfacing.
5. Advanced examples include ESP-32 acting as a sensor hub, communicating via MQTT/HTTP to Raspberry Pi or Android through UART or BLE.

8.3 Setup & Testing Procedure

6. Power the Vaman board via USB-Micro, enabling ESP-32 auto-power and FPGA/ARM sections.
7. Connect the board to your host machine and navigate to:
   - *ide/vaman-esp32/codes/wifi_blinky/wifi_blinky.ino*
8. In Arduino IDE:
   - Ensure “ESP32 Dev Module” is selected
   - Set correct COM port
   - Modify SSID/password in code and upload

4. On successful upload, ESP-32 connects to the configured WiFi and controls GPIO-2 to blink an LED or drive an external circuit.
5. Confirm wireless operation using a smartphone or laptop pinging the ESP-32 IP address.

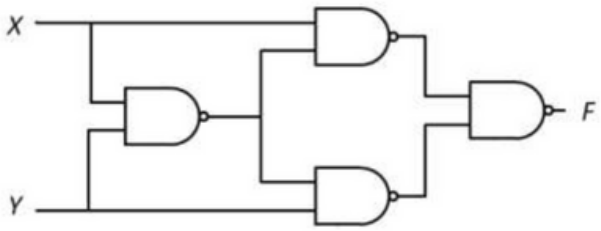
8.4 Applications & Impact

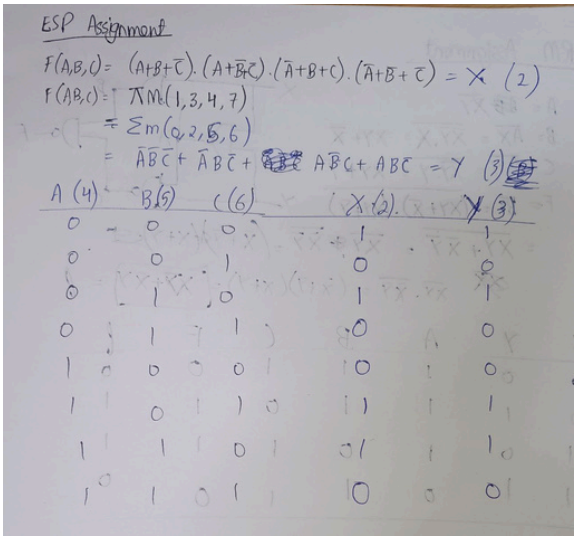
1. This integration enables rapid prototyping of IoT and autonomous control systems.
2. Wireless sensor interfacing, data logging, and remote actuation are easily demonstrated in classroom and lab environments.
3. Use cases include ESP-32 collecting sensor data and sending it to the Vaman ARM/FPGA, which processes it or triggers actions in UAV/UGV kits.

8.5 Assignment

https://github.com/ayush-1910/GATE-Papers-fwc-1/tree/main/fwc-1/esp32/assignment

Q.46 The Boolean function $F(X, Y)$ realized by the given circuit is





9: Vaman – FPGA

9.1 Introduction

1. The Vaman board also includes an onboard FPGA based on the QuickLogic EOS S3 SoC.
2. It allows students to explore digital logic design, soft processors, and hardware acceleration.
3. Programming the FPGA on Vaman is supported via SymbiFlow open-source toolchain.

9.2 Environment Setup

1. *Set up the environment using the prebuilt archive from the project directory:*
   - *wget https://raw.githubusercontent.com/gadepall/fwc-1/main/scripts/setup.sh*
   - *bash setup.sh*

9.3 Compiling and Uploading a Verilog Design

2. Navigate to the example directory:
   - *cd vaman/fpga/setup/codes/blink*
   - *source ~/.vamenv/bin/activate*
   - *ql symbiflow -compile -src vaman/fpga/setup/codes/blink -d ql-eos-s3 -P PU64 -v helloworldfpga.v -t helloworldfpga -p pygmy.pcf -dump binary*
   - *scp blink/helloworldfpga.bin pi@192.168.0.114:*
3. After a successful build, upload the bitstream to the FPGA using rpi:
   - *python3 -m venv ~/.vamenv*
   - *source ~/.vamenv/bin/activate*
   - *git clone --recursive https://github.com/QuickLogic-Corp/TinyFPGA-Programmer-Application.git*
   - *pip3 install tinyfpgab*
   - *deactivate*
   - *sudo reboot*
   - *source ~/.vamenv/bin/activate*
   - *python3 TinyFPGA-Programmer-Application/tinyfpga-programmer-gui.py -- port /dev/ttyACM0 --appfpga /home/pi/helloworldfpga.bin --mode fpga --reset*
4. This will program the onboard FPGA to execute the blink module (e.g., blinking an LED connected to a GPIO).

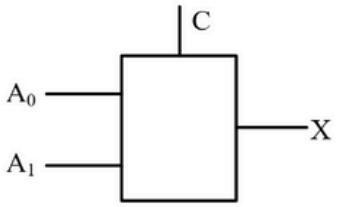
9.4 Applications

5. Vaman FPGA examples include:
   - *GPIO control (blinking LEDs)*
   - *PWM signal generation*
   - *Soft processor design*
6. These serve as introductory labs for FPGA development with open-source tools.

9.5 Assignment

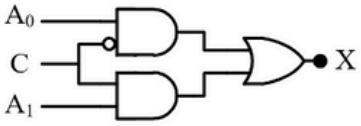
https://github.com/ayush-1910/GATE-Papers-fwc-1/tree/main/fwc-1/fpga/assignment

Q.13 In a 2-to-1 multiplexer as shown below, the output $X = A_0$ if $C = 0$, and $X = A_1$ if $C = 1$.

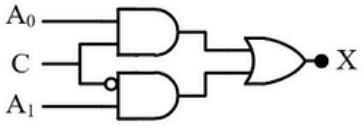


Which one of the following is the correct implementation of this multiplexer?

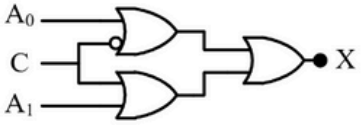
(A)



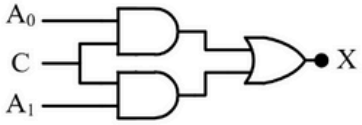
(B)



(C)



(D)



10: Vaman – ARM (Cortex-M4)

10.1 LED Control using ARM M4

1. The ARM Cortex-M4 core on the Vaman board can control onboard peripherals such as LEDs.
2. This section demonstrates compiling and flashing a simple blink program.

10.2 Configuration and Compilation

3. Navigate to the project directory:
   - *cd vaman/arm/setup/blink/GCC_Project*
4. Open and edit the config.mk file:
   - *nvim config.mk*
5. Modify the PROJ_ROOT line as below (for root users):
   - *export PROJ_ROOT=/root/pygmy-dev/pygmy-sdk*
6. Compile the project using:
   - *make -j4*
7. Transfer the binary to the Raspberry Pi:
   - *scp output/bin/blink.bin pi@192.168.0.114:*
8. (Replace the IP address with your Pi's address.)

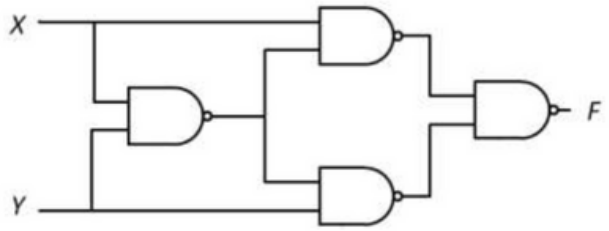
10.3 Upload to Vaman via Raspberry Pi

1. On the Pi, run the TinyFPGA programmer GUI:
   - *sudo python3 /home/pi/Vaman-dev/Vaman-sdk/TinyFPGA-Programmer-Application/tinyfpga-programmer-gui.py --port /dev/ttyACM0 --m4app blink.bin --mode m4-fpga*
     (Modify /dev/ttyACM0 if needed.)
2. After the script completes, press the onboard button to start the M4 app.
3. The onboard RGB LED will start blinking.

10.4 Assignment

https://github.com/ayush-1910/GATE-Papers-fwc-1/tree/main/fwc-1/arm/assignment

Q.46 The Boolean function $F(X, Y)$ realized by the given circuit is



(A) $\bar{X}Y + X\bar{Y}$ (B) $\bar{X}\bar{Y} + XY$ (C) $X + Y$ (D) $\bar{X} \cdot \bar{Y}$