Çankaya University
Department of
Electrical-Electronics Engineering

LED CONTROL WITH TWO SWITCHES

EE 315 - Microprocessors

Project Report

Name	**Surname**	**Number**	**Sign**
Aybüke	Şener	202326059	
Öykü Nilay	Aytuna	202326009	
Bilal	Erdoğan	202226031	
Ahmet Tuna	Fidan	202226033	

Supervisor: Assistant Professor Ulaş BELDEK, PhD

Date: 12 January 2026

This document was typeset in LaTeX using the Overleaf platform.

List of Figures

List of Tables

List of Source Codes

Contents

1 Introduction

In this part of the project, the problem, the aim of the project, literature survey and task distribution is going to be explained.

1.1 Problem Definition

The main problem of the project is developing a control algorithm on an embedded system. This algorithm has to detects the pressing and the releasing actions of two switches, PF0 and PF4, located on Port F of the microcontroller.

The system has to wait for a complete cycle where both switches are first pressed and then released. In this process, the order of events is irrelevant, it must correctly process any combination of sequential or simultaneous actions based on the principles of Discrete Event Systems [1].

The system must utilize the PF0 and PF4 pins on Port F as digital inputs. The user may press PF4 before PF0, or press both simultaneously. So tracking the status of switches has to be done independently. Also it must determine its output based on the history of events.

The waiting process must terminate only when both switches have reached the released state. When both switches reached the released state, the system should verify that both of them completed the press-and-release cycle.

1.2 Aim of The Project

The main purpose of this project is to practice the knowledge we learned in the EE315 course. This project was given to help us better understand how the development board works and how machine level (assembly) code is structured.

Through this project, we aimed to learn the interaction between hardware and software in a clearer way. By writing programs in assembly language, we gained experience with basic concepts such as registers, memory, and input/output operations as part of the hardware/software interface [2].

We also observed the differences between high-level programming languages and low-level programming. During the project, we also improved our problem solving and debugging skills and developed a better engineering perspective by working directly with the hardware.

1.3 Literature Survey

TM4C123G LaunchPad Evaluation Kit

The TM4C123G LaunchPad Evaluation Kit is an evaluation board designed by Texas

Instruments for ARM Cortex-M4F based microcontrollers. It is commonly used for educational purposes in embedded [3].

At the core of the board is the TM4C123GH6PM microcontroller, which operates with an 80 MHz, 32-bit ARM Cortex-M4F processor. The device includes 256 KB of flash memory, 32 KB of SRAM, and 2 KB of EEPROM, providing sufficient resources for a wide range of embedded applications [4].

The microcontroller offers extensive peripheral support, such as UART, SPI, I2C, CAN, and USB 2.0 Host/Device/OTG with an integrated PHY. In addition, it features dual 12-bit analog-to-digital converters, enabling analog signal processing with high sampling performance.

For user interaction and testing purposes, the LaunchPad board is equipped with programmable push buttons and an RGB LED. These components simplify basic input and output experiments during development.

The board includes stackable headers compatible with the BoosterPack™ XL interface, allowing users to easily extend the system by connecting external modules and additional peripherals.

An integrated In-Circuit Debug Interface (ICDI) is provided on the board, enabling programming and debugging directly through a USB connection without the need for extra hardware tools.

To support software development, Texas Instruments provides the TivaWare software development kit, which contains peripheral driver libraries and example projects. This SDK helps users quickly understand and utilize the microcontroller's features.

Due to its low cost, rich peripheral set, and ease of use, the TM4C123G LaunchPad is preferred in academic studies.

Keil µVision IDE

Keil µVision is a development environment specifically designed for embedded systems based on ARM microcontrollers. It is commonly used for writing, compiling, and debugging embedded software [5].

The µVision IDE integrates project management, source code editing, build tools, and debugging capabilities within a single interface, which improves development efficiency.

Its built-in debugger allows users to analyze and optimize their applications by using features such as breakpoints, execution control, and peripheral-level inspection.

Although the free version of Keil µVision has a code size limitation of 32 KB, it still offers advanced debugging and simulation features that are suitable for both educational and professional development purposes.

1.4 Distribution of Tasks

In accordance with the project requirements, the responsibilities were distributed among the group members to ensure an efficient workflow. The roles and contributions of each member are detailed below:

- **Aybüke Şener:** Tasked with the **comprehensive documentation** of the project. This included defining the problem, formulating the logical explanations of the proposed solutions, and authoring the conclusion. Furthermore, extensive literature research and information gathering were conducted to support the theoretical parts of the report.
- **Öykü Nilay Aytuna:** Responsible for the initial development of the control algorithm using **C++**. This high-level approach was utilized to establish a clear logical framework before migrating the design to low-level Assembly code, thereby reducing potential logic errors.
- **Bilal Erdoğan:** Responsible for the **technical implementation and coding**. This included writing the final ARM Assembly code in Keil uVision, developing the MATLAB simulation scripts, and performing the physical circuit connections on the breadboard. Additionally, the final integration of the report into the LaTeX format was managed by this member.
- **Ahmet Tuna Fidan:** Focused on the **visual representation and demonstration** of the system. This member designed and created the state diagram (Discrete Event System representation) and handled the recording, editing, and uploading of the implementation video to YouTube.
- **Collaborative Efforts (All Members):** The entire team participated in the **debugging and optimization** phases of the Assembly code. System testing and verification were performed collectively. Moreover, each member contributed specific technical content related to their individual tasks for the final report.

Table 1: Task Distribution

Name-Surname	**Tasks**
Aybüke Şener	Project Report
Öykü Nilay Aytuna	C++ Code
Bilal Erdoğan	Implementation, Assembly & Matlab Code
Ahmet Tuna Fidan	State Diagram and Video

2 Proposed Solution

The proposed solution of this project tested with different level programming languages to see the differences between the levels. These languages are Assembly, MATLAB and C++ which three different level of languages. Above machine code the most low-level language in the world is Assembly language. C++ is a mid/high level language. MATLAB is a very high level language. The MATLAB used to detect if state machine works correctly while C++ code was used as a logical bridge to develop the final Assembly implementation [6].

2.1 Keil uVision, MATLAB, C++ Codes and Their Explanations

Below are the codes developed for the simulation and implementation of the project.

2.1.1 MATLAB Simulation Code

This code simulates the logic of the system using high-level scripting to verify the state transitions before implementation, following the logic structures defined in the software documentation [7]

```
clear;
clc;

% --- Variable Initialization ---
flag_sw2 = 0; % Flag for Switch 2 (0: Not Pressed, 1: Pressed)
flag_sw1 = 0; % Flag for Switch 1 (0: Not Pressed, 1: Pressed)
curr_sw2 = 1; % Current State of SW2 (Negative Logic: 1=Released)
curr_sw1 = 1; % Current State of SW1 (Negative Logic: 1=Released)

% Port B Outputs
led_pb0 = 0;  % LED 1 Status
led_pb1 = 0;  % LED 2 Status
led_pb2 = 0;  % LED 3 (Done) Status

disp('--- SIMULATION START (Negative Logic) ---');

% --- Main Loop ---
while (flag_sw2 == 0 || flag_sw1 == 0 || curr_sw2 == 0 || curr_sw1 == 0)

    % Input
    curr_sw2 = input('Enter SW2 (PF0) State (0 or 1): ');
    curr_sw1 = input('Enter SW1 (PF4) State (0 or 1): ');

    % Logic for SW2 (PF0)
    if curr_sw2 == 0
        flag_sw2 = 1;
```

```
        led_pb0 = bitxor(led_pb0, 1);
    else
        if flag_sw2 == 1
            led_pb0 = 1;
        else
            led_pb0 = 0;
        end
    end

    % Logic for SW1 (PF4)
    if curr_sw1 == 0
        flag_sw1 = 1;
        led_pb1 = bitxor(led_pb1, 1);
    else
        if flag_sw1 == 1
            led_pb1 = 1;
        else
            led_pb1 = 0;
        end
    end

    % Output
    disp(['LED1: ' num2str(led_pb0) ' | LED2: ' num2str(led_pb1) ' |
   LED3: ' num2str(led_pb2)]);
end

% --- Program End ---
led_pb0 = 0;
led_pb1 = 0;
led_pb2 = 1;

disp('*** PROCESS COMPLETED ***');
disp(['LED1: ' num2str(led_pb0) ' | LED2: ' num2str(led_pb1) ' | LED3: '
    num2str(led_pb2)]);
```

Listing 1: MATLAB Simulation Code

2.1.2 C++ Algorithm Logic

Assembly is a low-level language, which makes it difficult to follow complex logic flows and very open to syntax errors.

This high-level approach allowed me to visualize four different button states and manage the (read-modify-write) operations more efficiently and easily.

This section represents an altarnative algorithmic flow in C language structure for better understanding of the logic flow.

```c
#include <stdint.h>

// EQU yani constant satirlar
#define SYSCTL_RCGCGPIO_R      (*((volatile uint32_t *)0x400FE608))
#define GPIO_PORTF_DATA_R      (*((volatile uint32_t *)0x400253FC))
#define GPIO_PORTF_DIR_R       (*((volatile uint32_t *)0x40025400))
#define GPIO_PORTF_PUR_R       (*((volatile uint32_t *)0x40025510))
#define GPIO_PORTF_DEN_R       (*((volatile uint32_t *)0x4002551C))
#define GPIO_PORTF_LOCK_R      (*((volatile uint32_t *)0x40025520))
#define GPIO_PORTF_CR_R        (*((volatile uint32_t *)0x40025524))



int main(void) {
    // Once clock aktif et
    SYSCTL_RCGCGPIO_R |= 0x20;        // Port F aktif
    volatile uint32_t delay = SYSCTL_RCGCGPIO_R; // Clock icin kisa bekleme
    // volatile ile degiskenin optimizasyonu engellenir, uint32_t 32 bitlik unsigned integer anlamina gelir

    // Kilidi ac (PF0 icin sart)
    GPIO_PORTF_LOCK_R = 0x4C4F434B;  // Lock sifresi
    GPIO_PORTF_CR_R |= 0x1F;         // Degisiklik izni verme (Pin 0-4)

    //  Yon ayari directive port (DIR)
    // PF0, PF4 input 0; PF1, PF2, PF3 output  1
    GPIO_PORTF_DIR_R &= ~0x11;       // PF0 ve PF4 Input, BIC'in yaptigini yapar bitleri sifirlar
    GPIO_PORTF_DIR_R |= 0x0E;        // PF1-3 Output (LED'ler), ORR'un yaptigi gibi bitleri 1 yapar
```

Listing 2: C++ Logic Structure Part 1

As seen in the implementation, we perform initialization at the very beginning of the code. This is the critical stage where we inform the microcontroller pins about their specific tasks. Without these hardware settings, it is impossible to read data from the pins or control the on-board peripherals like LEDs.

STEPS OF INITIALIZATION:

- **Clock Activation:** To make Port F work, the clock signal must be activated first via the *RCGCGPIO* register as specified in the microcontroller data sheet [4]. It is like a heartbeat; Port F remains inactive and inaccessible without an active clock signal.

- **Unlocking PF0:** By default, the PF0 pin is locked to prevent accidental configuration of the NMI (Non-Maskable Interrupt). Therefore, it must be unlocked using

a special lock code (0x4C4F434B) found in the board's technical documentation [8].

- **Direction Setting (DIR):** The data flow direction of the pins is determined in this stage. Buttons (PF0, PF4) are configured as **input (0)**, while LEDs (PF1, PF2, PF3) are configured as **output (1)**.
- **Pull-up Resistor (PUR):** Since the on-board switches are designed with negative logic, internal pull-up resistors are enabled to handle negative logic configurations [8].
- **Digital Enable (DEN):** Digital functions are explicitly activated for the required pins. This allows the pins to process digital high/low signals instead of remaining in their default analog state.

```
    //  Pull up (PUR)
    GPIO_PORTF_PUR_R |= 0x11;         // Butonlar icin aktif,
basilmadiginda 1 olur

    // Digital enable (DEN)
    GPIO_PORTF_DEN_R |= 0x1F;         // Tum pinleri dijital yap

    // Baslangicta tum LED'leri kapat
bool pf0_basildi = false;
    bool pf4_basildi = false;

    // ASAMA 1:IKISININ DE BASILMALARINI BEKLE
    while (!(pf0_basildi && pf4_basildi)) {
        uint32_t r2 = GPIO_PORTF_DATA_R & 0x11; // Maskeleme yapiyoruz

        // DURUM 1: Ikisine de basildi mi? (R2 == 0x00)
        if (r2 == 0x00) {
            pf0_basildi = true;
            pf4_basildi = true;
            GPIO_PORTF_DATA_R = (GPIO_PORTF_DATA_R & ~0x0E) | 0x08; //
Read modify write yapip yesil LED yaktik (PF3)
        }
        // DURUM 2: Sadece PF4 basili mi? (R2 == 0x01)
        else if (r2 == 0x01) {
            pf4_basildi = true;
            GPIO_PORTF_DATA_R = (GPIO_PORTF_DATA_R & ~0x0E) | 0x02; //
Kirmizi LED (PF1)
        }
        // DURUM 3: Sadece PF0 basili mi? (R2 == 0x10)
        else if (r2 == 0x10) {
            pf0_basildi = true;
            GPIO_PORTF_DATA_R = (GPIO_PORTF_DATA_R & ~0x0E) | 0x04; //
Mavi LED (PF2)
```

```
        }
        // DURUM 4: Hicbirine basilmamis (R2 == 0x11)
        else {
            // Eger daha once basilmislarsa LED'i sondurme, yoksa sondur
            if (!pf0_basildi && !pf4_basildi) GPIO_PORTF_DATA_R &= ~0x0E
    ;
        }
     }
```

Listing 3: C++ Logic Structure Part 2

In this part, we first perform masking, then we write the sections where we examine four different states for the buttons. Masking is a process used to select only the bits we are interested in (PF0 and PF4 in this project) within a register and ignore the rest. Briefly looking at the states:

STEPS OF INITIALIZATION:

- **State 1:** This is the case where '0' signals come from both PF0 and PF4 pins; this means we press both buttons and the Green LED turns on.
- **State 2:** This is the case where PF0 is '1' and PF4 is '0'. The Red LED turns on.
- **State 3:** We can say this is the exact opposite of State 2, and the system turns on the Blue LED to confirm that the PF0 button is pressed.
- **Final State:** Both flags (R4, R5) are set to 1 AND both buttons are currently released (Logic 1), triggering the exit condition.

```
    // ASAMA 2: HER IKISININ DE BIRAKILMASINI BEKLE (release)
    while ((GPIO_PORTF_DATA_R & 0x11) != 0x11) {
        // Ikisi de 1 (release) olana kadar burada don.
    }

    // Bitis: Her iki buton da birakildi, beyaz yak (Tum LED'ler)
    GPIO_PORTF_DATA_R |= 0x0E;

    while (1);
}
```

Listing 4: C++ Logic Structure Part 3

In the final part, it is ensured that both buttons return to the release position, and our code is completed at this point.

2.1.3 ARM Assembly Implementation (Keil uVision)

The final implementation deployed on the TM4C123G LaunchPad.

```
; EE315 Project
; Hardware: PF0/PF4 (Inputs), PB0/PB1/PB2 (Outputs)

; --- Register Definitions ---
SYSCTL_RCGCGPIO_R   EQU 0x400FE608
GPIO_PORTF_DATA_R   EQU 0x400253FC
GPIO_PORTF_DIR_R    EQU 0x40025400
GPIO_PORTF_DEN_R    EQU 0x4002551C
GPIO_PORTF_PUR_R    EQU 0x40025510
GPIO_PORTF_LOCK_R   EQU 0x40025520
GPIO_PORTF_CR_R     EQU 0x40025524

GPIO_PORTB_DATA_R   EQU 0x400053FC
GPIO_PORTB_DIR_R    EQU 0x40005400
GPIO_PORTB_DEN_R    EQU 0x4000551C

LOCK_KEY            EQU 0x4C4F434B

        AREA    |.text|, CODE, READONLY, ALIGN=2
        THUMB
        EXPORT  Start

Start
    ; --- Clock Init ---
    LDR R0, =SYSCTL_RCGCGPIO_R
    LDR R1, [R0]
    ORR R1, R1, #0x22       ; Enable Port F and Port B clock
    STR R1, [R0]
    NOP
    NOP

    ; --- Port F Init (Inputs) ---
    LDR R0, =GPIO_PORTF_LOCK_R
    LDR R1, =LOCK_KEY
    STR R1, [R0]            ; Unlock PF0

    LDR R0, =GPIO_PORTF_CR_R
    MOV R1, #0x01
    STR R1, [R0]

    LDR R0, =GPIO_PORTF_DIR_R
    LDR R1, [R0]
    BIC R1, R1, #0x11       ; Set PF0, PF4 as Input
    STR R1, [R0]

    LDR R0, =GPIO_PORTF_PUR_R
```

```
    LDR R1, [R0]
    ORR R1, R1, #0x11       ; Enable Pull-Up resistors
    STR R1, [R0]

    LDR R0, =GPIO_PORTF_DEN_R
    LDR R1, [R0]
    ORR R1, R1, #0x11       ; Digital Enable
    STR R1, [R0]

    ; --- Port B Init (Outputs) ---
    LDR R0, =GPIO_PORTB_DIR_R
    LDR R1, [R0]
    ORR R1, R1, #0x07       ; Set PB0, PB1, PB2 as Output
    STR R1, [R0]

    LDR R0, =GPIO_PORTB_DEN_R
    LDR R1, [R0]
    ORR R1, R1, #0x07
    STR R1, [R0]

    ; --- Variables ---
    MOV R4, #0  ; Flag for SW2 (0: Not Pressed)
    MOV R5, #0  ; Flag for SW1 (0: Not Pressed)

    LDR R0, =GPIO_PORTF_DATA_R
    LDR R2, =GPIO_PORTB_DATA_R

Loop
    LDR R1, [R0]            ; Read Inputs
    LDR R3, [R2]            ; Read Outputs

    ; --- SW2 Logic ---
    TST R1, #0x01
    BNE SW2_Released

    MOV R4, #1              ; SW2 Pressed
    EOR R3, R3, #0x01       ; Toggle LED1
    B   Check_SW1

SW2_Released
    CMP R4, #1
    BNE TurnOff_LED1
    ORR R3, R3, #0x01       ; Steady ON
    B   Check_SW1
TurnOff_LED1
    BIC R3, R3, #0x01

```

```
Check_SW1
    ; --- SW1 Logic ---
    TST R1, #0x10
    BNE SW1_Released

    MOV R5, #1              ; SW1 Pressed
    EOR R3, R3, #0x02       ; Toggle LED2
    B   Apply_Output

SW1_Released
    CMP R5, #1
    BNE TurnOff_LED2
    ORR R3, R3, #0x02       ; Steady ON
    B   Apply_Output
TurnOff_LED2
    BIC R3, R3, #0x02

Apply_Output
    STR R3, [R2]
    BL  Delay

    ; --- Exit Check ---
    CMP R4, #1
    BNE Loop
    CMP R5, #1
    BNE Loop
    TST R1, #0x01
    BEQ Loop
    TST R1, #0x10
    BEQ Loop

    B   Done

; --- Delay Subroutine ---
Delay
    LDR R6, =160000         ; Approx delay loop
Delay_Loop
    SUBS R6, R6, #1
    BNE  Delay_Loop
    BX   LR

; --- Final State ---
Done
    LDR R2, =GPIO_PORTB_DATA_R
    MOV R3, #0x04           ; LED1, LED2 OFF | LED3 ON
    STR R3, [R2]

```

```
141 DeadLoop
142     B   DeadLoop
143
144     END
```

Listing 5: ARM Assembly Code

2.2 State Diagram of the Problem

The logical framework of this project is designed based on the principles of **Discrete Event Systems (DES)** to manage asynchronous events [1]. In an embedded context, a DES is a system where the state evolves based on the occurrence of physical events such as a button press or release rather than the continuous passage of time.

The behavior of the system is modeled using a **Finite State Machine (FSM)**, which provides a formal representation of the discrete states and the transitions triggered by asynchronous external inputs. As illustrated in Figure 1, the system transitions between states based on the following DES characteristics:

- **Event-Driven Transitions:** The system remains in a steady state until an event occurs. Here, the "events" are defined as the falling and rising edges of the signals from PF0 and PF4.
- **Asynchronous Concurrency:** Since the user can press the switches independently or simultaneously, the DES model must account for all possible event sequences. The implemented algorithm tracks these events using status flags to ensure that the order of arrival does not affect the final reachability of the completion state.
- **State Memory (Historical Context):** Unlike purely combinational logic, this DES implementation utilizes the history of events. The system distinguishes between "instantaneous input levels" and the "historical context of events" (e.g., whether a button has already been pressed once) to determine the next transition.
- **Termination Criteria:** The final state in the DES model is reachable only after a specific language of events is completed:
  $\{Press(PF0), Press(PF4), Release(PF0), Release(PF4)\}$, regardless of their interleaving.

The diagram represents a non-deterministic interleaved event sequence, ensuring the system reaches the final state regardless of which button is pressed first.

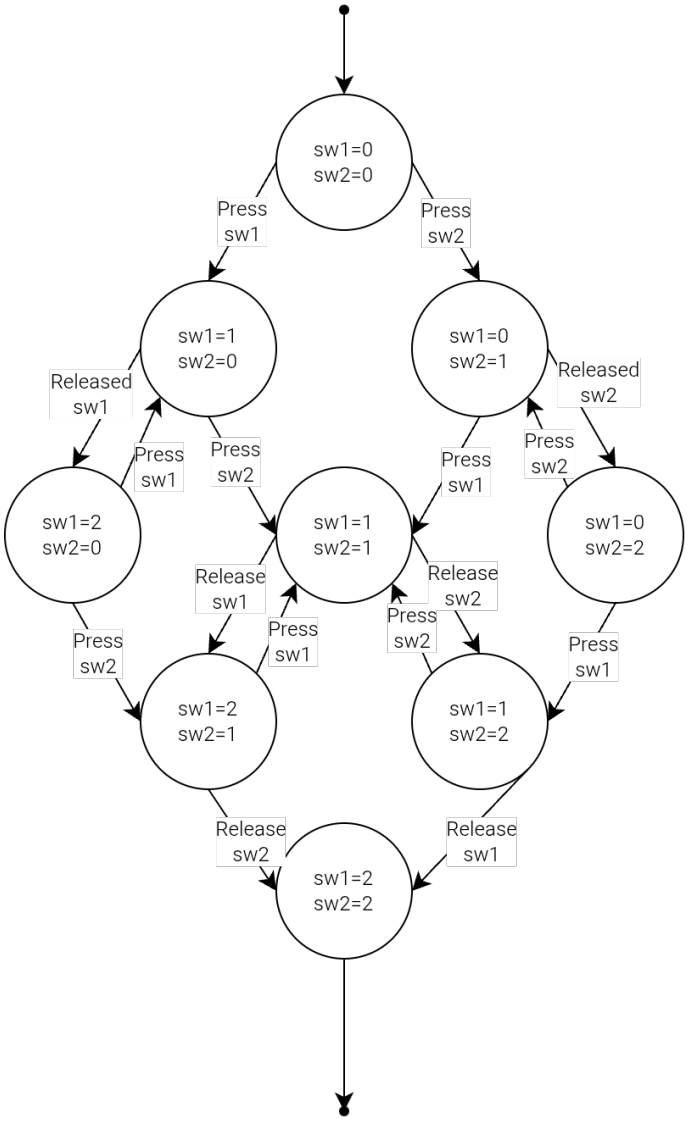


Figure 1: State Diagram

2.3 System Implementation

The complete implementation of the project, including the hardware setup on the breadboard and the execution of the code on the TM4C123G LaunchPad, has been recorded.

2.3.1 System Implementation Video

You can watch the demonstration video by scanning the QR code below or by clicking the direct link.

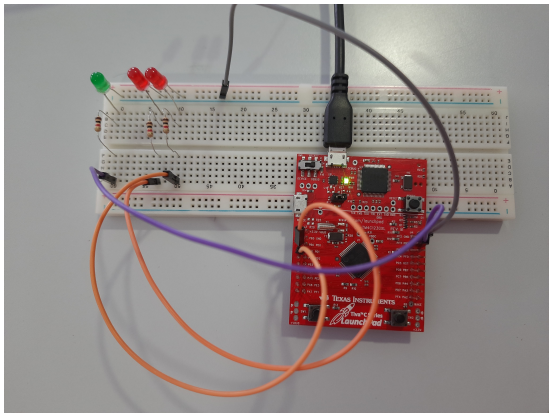
Figure 2: Scan this QR Code to watch the project demo

Direct Video Link:

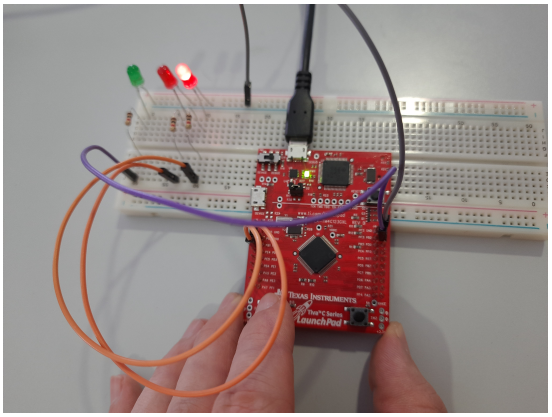
https://youtu.be/PVDyQyRDL90

2.3.2 System Implementation Photos

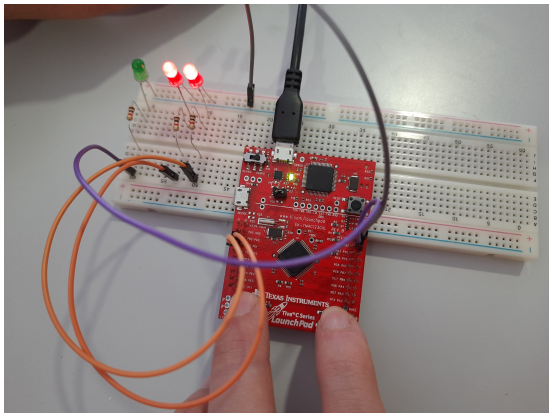
The following figures illustrate the different states of the system during operation:



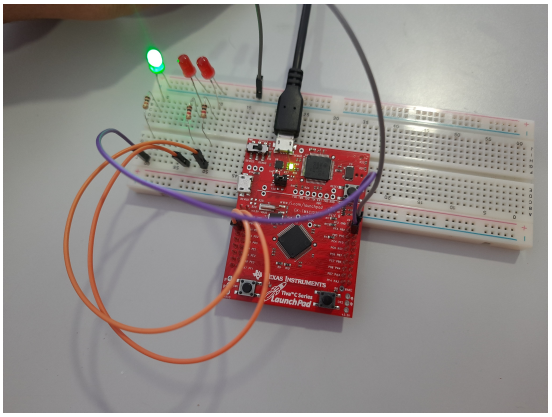
(a) Initial State (All LEDs OFF)



(b) Single Button Pressed (One LED Blinking)



(c) Both Buttons Pressed (Two LEDs Blinking)



(d) Final State (Task Completed LED ON)

Figure 3: System Implementation States

3 Conclusion

In this project, the system was successfully designed and implemented on an ARM based microcontroller to control input sequence of two switches. An algorithm that can detect the press and release cycle for both switches, non-dependent by order of the events, is successfully created.

The system was able to differentiate between instantaneous input levels and the historical context of events by converting switch inputs into logical state variables.

The code developed in different languages to see the differences between different level languages. The code developed in different languages effectively tracked asynchronous events and transitioned the state machine from the initial state to the final completed state.

The testing of the codes (especially the assembly code) confirmed that the system that designed in this project correctly waits until PF0 and PF4 reach the released state before terminating the process.

References

[1] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. Boston, MA: Springer, 2008.

[2] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, ARM Edition. Morgan Kaufmann, 2016.

[3] J. W. Valvano, *Embedded Systems: Introduction to ARM Cortex-M Microcontrollers*, vol. 1, 5th ed. Austin, TX: Jonathan Valvano, 2012.

[4] Texas Instruments, *Tiva™ TM4C123GH6PM Microcontroller Data Sheet*, Publication No. SPMS376E, June 2014. [Online]. Available: https://ee315.cankaya.edu.tr/uploads/files/tm4c123gh6pm.pdf. [Accessed: Jan. 12, 2026].

[5] armKEIL, *µVision User's Guide*, [Online]. Available: https://www.keil.com/products/uvision/. [Accessed: Jan. 12, 2026].

[6] ARM Limited, *Cortex-M3/M4F Instruction Set Technical User's Manual*, 2011. [Online]. Available: https://ee315.cankaya.edu.tr/uploads/files/CortexM_InstructionSet.pdf. [Accessed: Jan. 12, 2026].

[7] MathWorks, *MATLAB Documentation*, [Online]. Available: https://www.mathworks.com/help/matlab/. [Accessed: Jan. 12, 2026].

[8] Texas Instruments, *Stellaris® LM4F120 LaunchPad Evaluation Board User's Manual*, Publication No. SPMU289, 2012. [Online]. Available: https://ee315.cankaya.edu.tr/uploads/files/LaunchPadUsersManual.pdf. [Accessed: Jan. 12, 2026].