
	<i>AGH University of Science and Technology</i>		
Faculty of Electrical Engineering, Automatics, Computer Science and Biomedical Engineering			
Texas Instruments C2000 MCU in automotive and industrial application			
author/author: Jakub Cios Damian Krakowiecki Maciej Duda		kierunek i rok stu field of study: Electrical engineering	academic year: 2023/2024
			date: 01.06.2024

Overview

This report details the activities undertaken to complete four tasks using the Texas Instruments C2000 MCU, specifically focusing on the TI Delfino F28379D LaunchPad. The first two tasks involved configuring and programming the LaunchPad to control LEDs based on button inputs. The third task aimed to create a model that counts the number of times a knob is turned, based on rectangular signals generated by an encoder. The fourth task involved programming an 8-segment display to show specific digits using MATLAB Simulink.

Task 1

Objective: Create a program that controls four LEDs (DLp1 to DLp4) based on specified conditions, using GPIO pins and digital inputs.

a. DLp1 Diode Always On

Implementation: Configured a Digital Output (GPIO DO) block for GPIO15 (DLp1).

Configuration: Set the block input to 1 (true).

Outcome: The DLp1 LED remained continuously lit, regardless of any button states

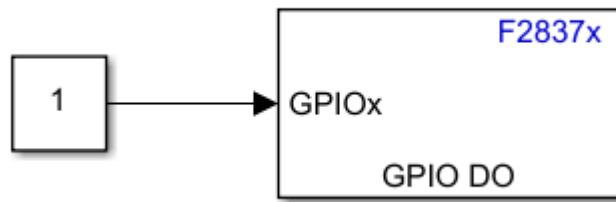


Fig. 1 DLp1 Diode Always On

b. DLp2 Diode Always Off

Implementation: Configured a Digital Output (GPIO DO) block for GPIO14 (DLp2).

Configuration: Set the block input to 0 (false).

Outcome: The DLp2 LED remained off at all times, unaffected by any button states

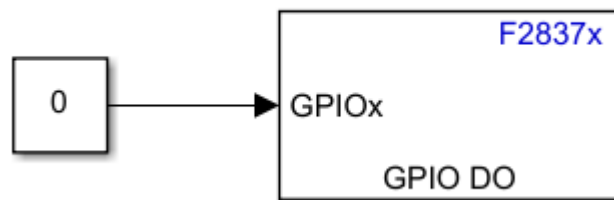


Fig. 2 DLp2 Diode Always Off

c. DLp3 LED On When SW1 Button is Pressed

Implementation: Used a Digital Input (GPIO DI) block for GPIO9 (SW1) and a Digital Output (GPIO DO) block for GPIO11 (DLp3).

Configuration: Connected the output of the DI block to the input of the DO block and added a NOT logic block between them due to the connection of the button in a pull up resistor configuration.

Outcome: The DLp3 LED lit up when the SW1 button was pressed and turned off when the button was not pressed.

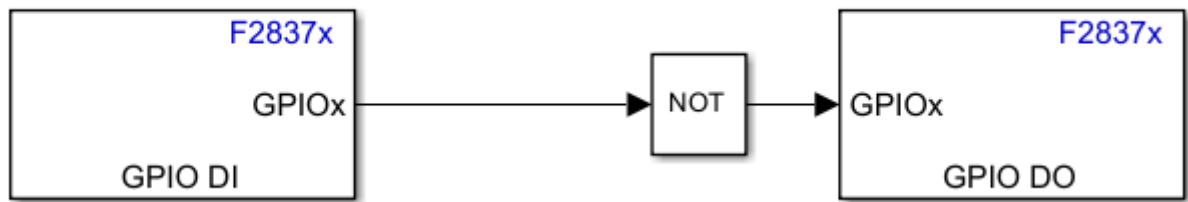


Fig. 3 DLp3 LED On When SW1 Button is Pressed

d. DLp4 Diode Off When SW2 Button is Pressed

Implementation: Used a Digital Input (GPIO DI) block for GPIO8 (SW2) and a Digital Output (GPIO DO) block for GPIO10 (DLp4).

Configuration: Connected the DI block output directly to the DO block input.

Outcome: The DLp4 LED turned off when the SW2 button was pressed and lit up when the button was not pressed.



Fig. 4 DLp4 Diode Off When SW2 Button is Pressed

Task 2

Objective: Develop a program that further manipulates the LEDs (DLp1 to DLp4) based on more complex button input conditions.

a. DLp1 Diode On When Both Buttons Are Pressed

Implementation: Configured two Digital Input (GPIO DI) blocks for GPIO9 (SW1) and GPIO8 (SW2), and a Digital Output (GPIO DO) block for GPIO15 (DLp1).

Configuration: Connected the DI block outputs to an AND Logical Operator block through NOT logic blocks, then to the DO block.

Outcome: The DLp1 LED lit up only when both SW1 and SW2 were pressed simultaneously.

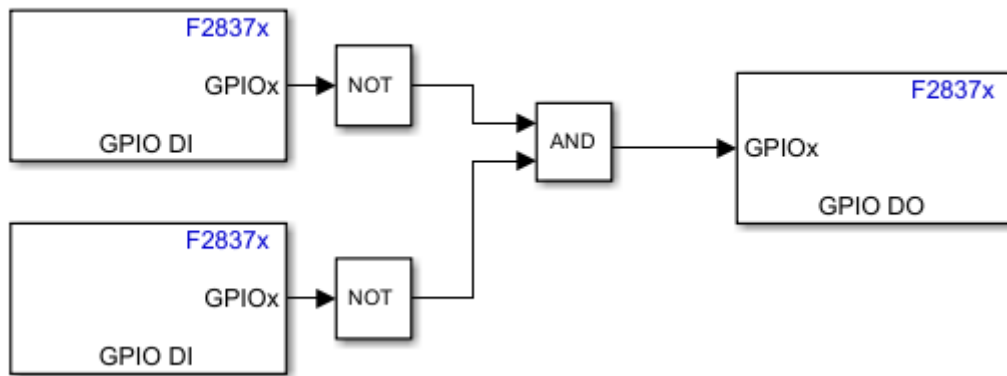


Fig. 5 DLp1 Diode On When Both Buttons Are Pressed

b. DLp2 LED On When At Least One Button is Pressed

Implementation: Configured two Digital Input (GPIO DI) blocks for GPIO9 (SW1) and GPIO8 (SW2), and a Digital Output (GPIO DO) block for GPIO14 (DLp2).

Configuration: Connected the DI block outputs to an OR Logical Operator block through NOT logic blocks, then to the DO block.

Outcome: The DLp2 LED lit up when either SW1 or SW2 (or both) were pressed.

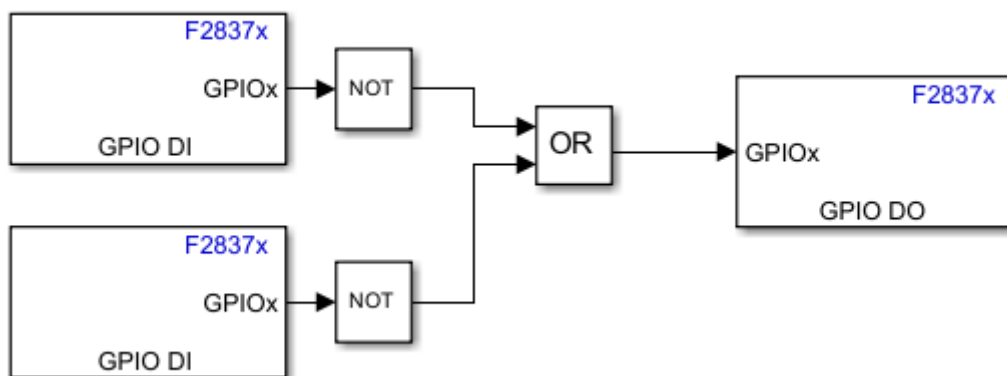


Fig. 6 DLp2 LED On When At Least One Button is Pressed

c. DLp3 Diode State Change Every 1 Second

Implementation: Configured a Digital Output (GPIO DO) block for GPIO11 (DLp3) and a Discrete Pulse Generator block.

Configuration: Set the pulse generator to have a period of 2 seconds and a duty cycle of 50%.

Outcome: The DLp3 LED toggled its state every second.

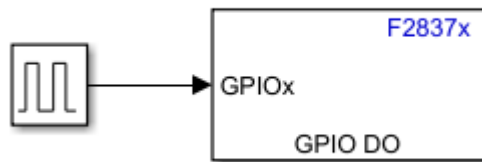


Fig. 7 DLp3 Diode State Change Every 1 Second

d. DLp4 Diode State Change When SW1 Button is Pressed

Implementation: Configured a Digital Input (GPIO DI) block for GPIO9 (SW1) and a Digital Output (GPIO DO) block for GPIO10 (DLp4), with a Toggle logic.

Configuration: Connected the DI block output directly to the DO block input.

Outcome: The DLp4 LED changed its state with each press of SW1 and maintained its state until the next press.

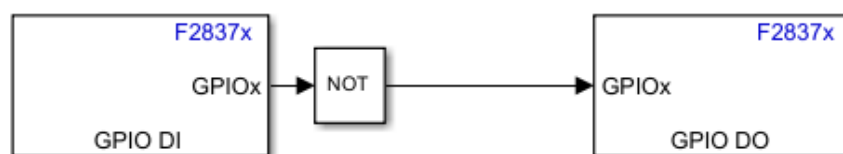


Fig. 9 DLp3 Diode State Change Every 1 Second

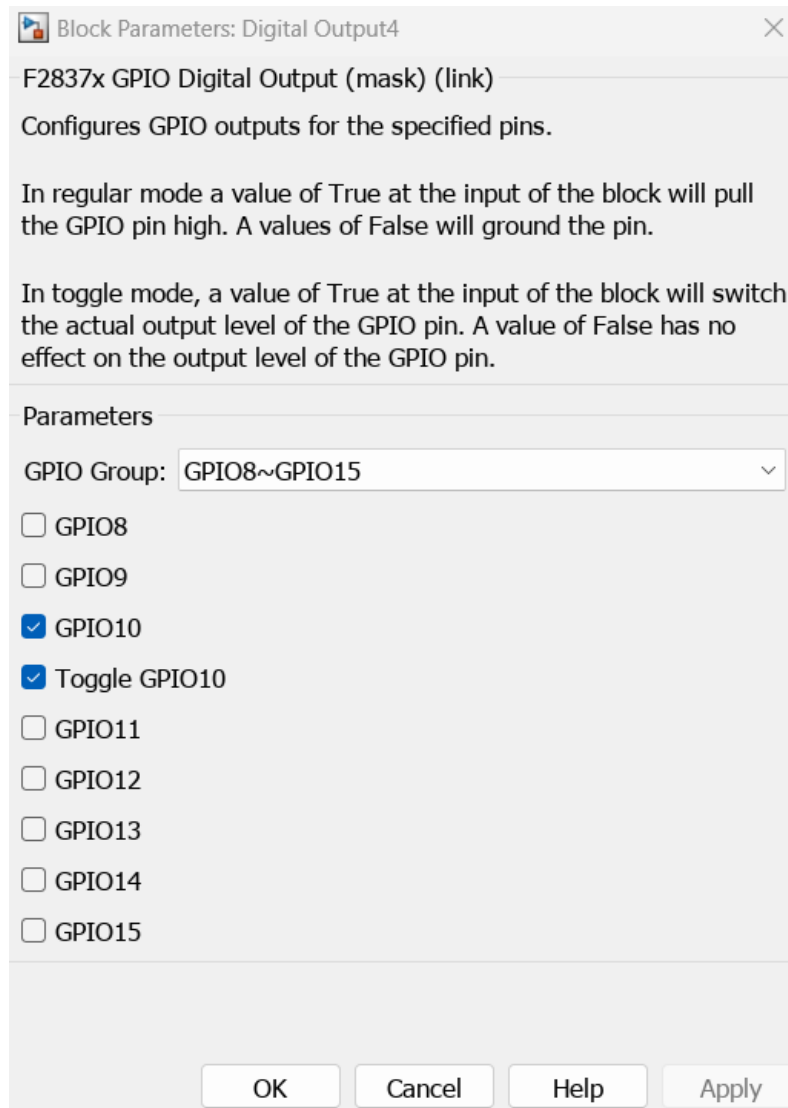


Fig. 9 DLp3 Diode State Change Every 1 Second– Digital Output window

Task 3

Objective: Develop a program that counts impulses generated by turning the pulser and increase value of a variable when turning right and decreases the value of that variable when turning left.

a) **increasing the value of the variable as a result of turning the pulser knob to the right and decreasing the value of this variable as a result of turning it to the left**

Implementation: Configured three Digital Input (GPIO DI) blocks for GPIO130 and GPIO131 which produce rectangular signals shifted relative to each other, which allows to recognise the direction of rotation when the knob is turned.

Configuration: Connected the DI block outputs to an AND Logical Operator block. Output

of that block triggers subsystem, which counts how many times the knob was turned, increasing or decreasing a variable depending on direction in which the knob was turned.

Outcome: Turning the knob right increased the variable, and turning left decreased it.

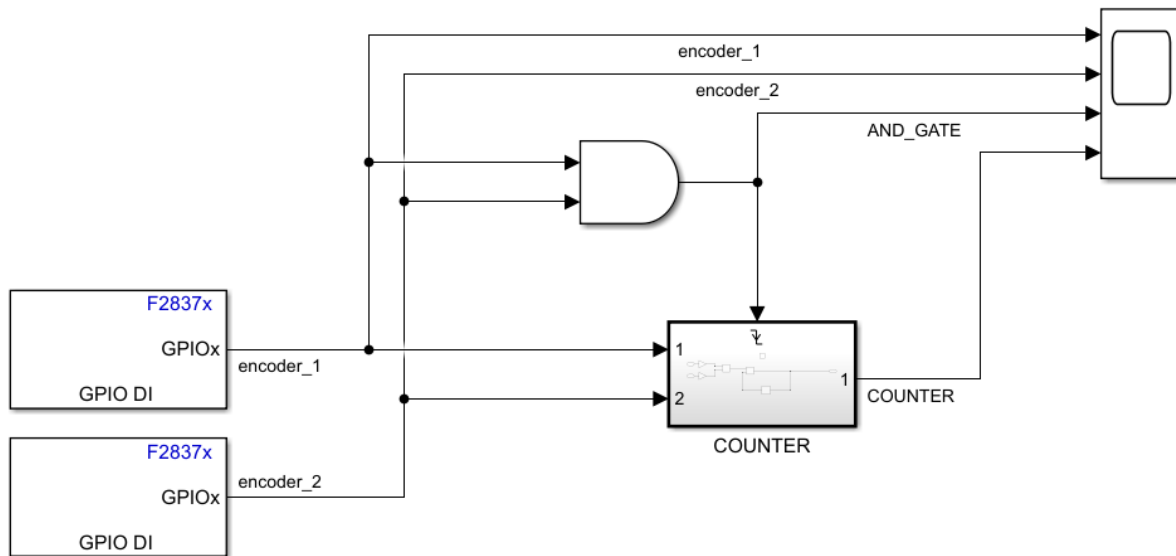


Fig. 10 Model of a system with signals observed in an oscilloscope and generation of the trigger signal

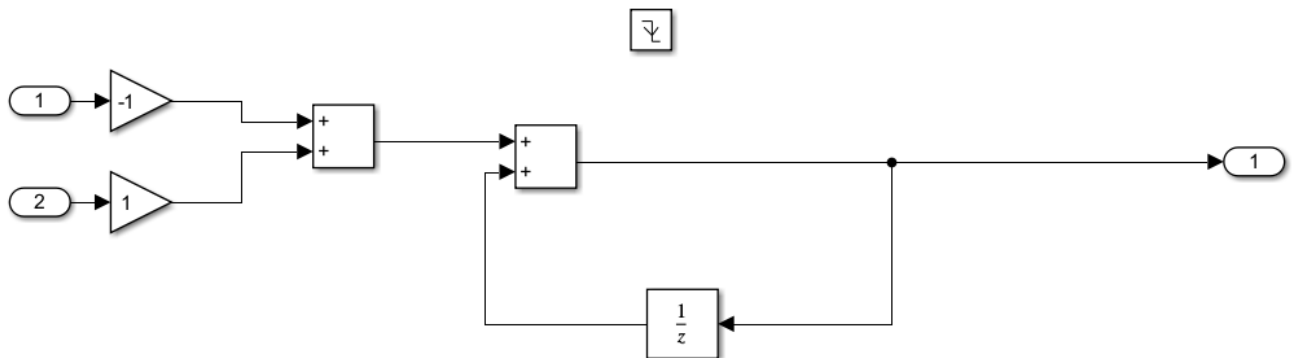


Fig. 11 Subsystem that counts signals after being triggered

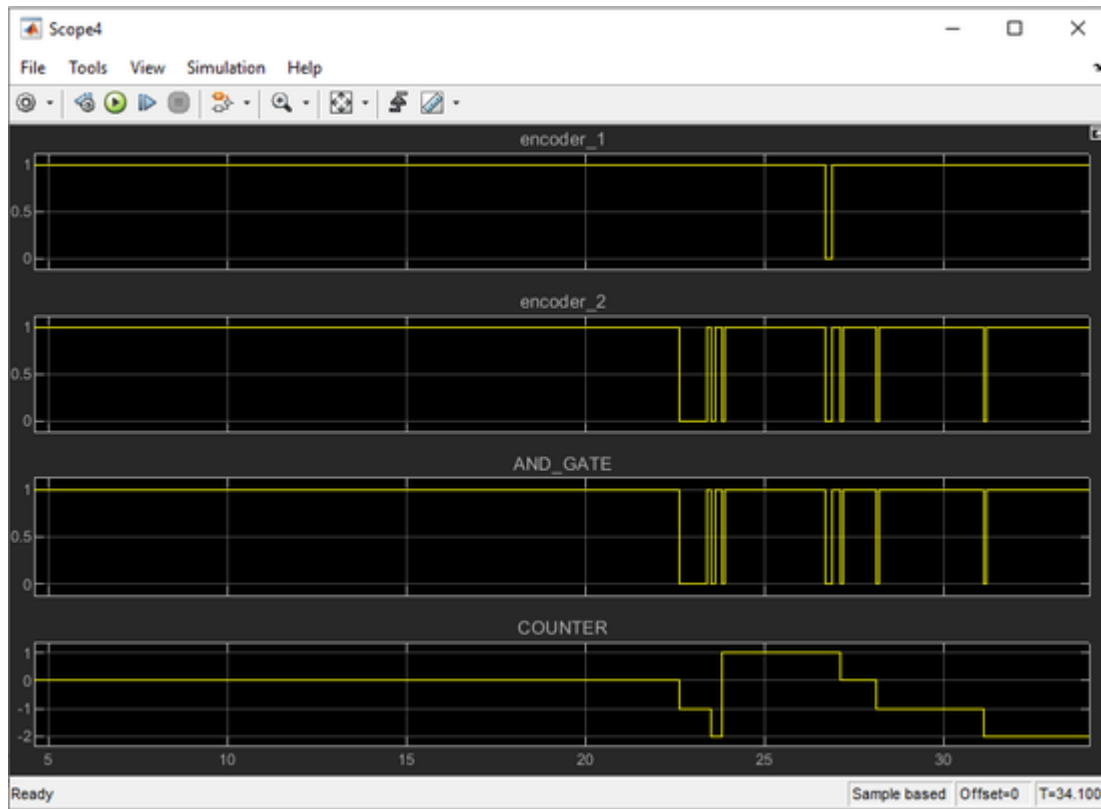


Fig. 12 Output of the oscilloscope from fig.10 model

b) limiting the value of the variable to the accepted range

Implementation: Saturation block was added in the subsystem, after the ADD block. If it was added after the subsystem, the counted value could still be higher than 5, but it wouldn't be displayed on the oscilloscope.

Configuration: Upper limit of the signal was set to 15, and lower limit to -15.

Outcome: Output of the counter increases and decreases while the knob is turned, but its value is in the range between -15 and 15.

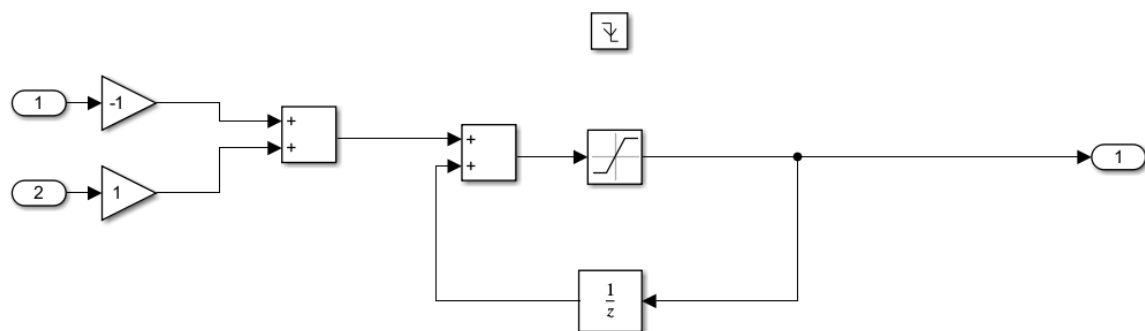


Fig. 13 Subsystem that counts signals with saturation

c) **changing the adjustment precision when the knob is pressed**

Implementation: Adding another GPIO Digital Input block for GPIO66 was necessary, because the state of that pin indicates if the knob is pressed down or not. Switch block was also added to the subsystem in order to select, how much should the variable change after each turn.

Configuration: Threshold of the Switch block was set to >0.5 , because the value of the third pulser output is either 0 or 1, depending on if the knob is pressed down.

Outcome: Turning the knob increments the variable by 5 when the knob is pressed and by 1 when it's not.

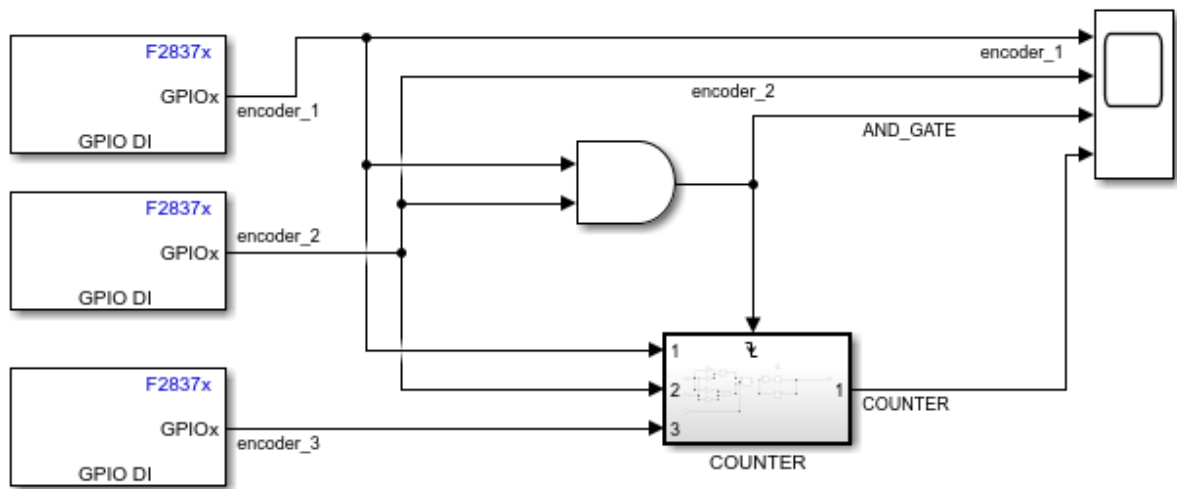


Fig. 14 Subsystem with third pulser input

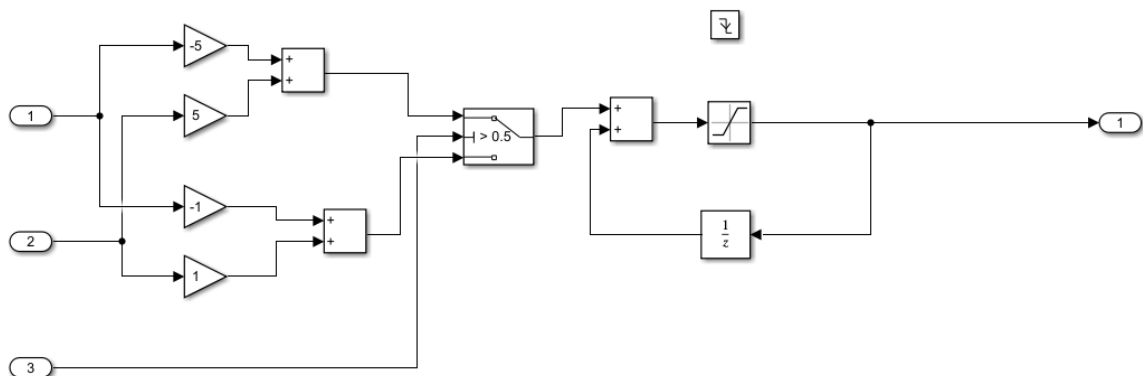


Fig. 15 Subsystem that adjusts precision depending on if the knob is pressed

8-segment displays ~ “Task 4”

Introduction

During the class on the available board, we were tasked with programming the board so that the display would show individual digits. To do this, we used the MATLAB SIMULINK environment.



Figure 16 Photo of used board

Course of the exercise

The first stage was the preparation of the project. Following the available instructions, it was possible to connect to the board.

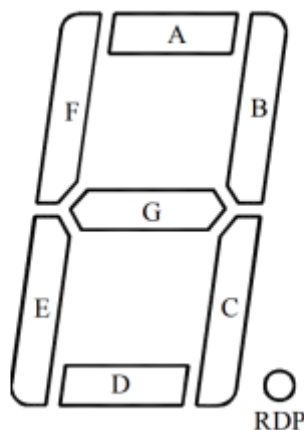


Figure 17 Markings of individual segments

	A	B	C	D	E	F	G	RDP
display 1	1	123	18	19	32	61	0	67
display 2	122	4	22	60	111	3	2	105
display 3	24	16	139	56	104	58	5	95

Figure 18 GPIO pin numbers for individual display segments

The board we had available had 3 displays consisting of 7 segments. Using the GPIO pins, we were able to change the supplied signal to the segment. The layout of the segments is shown in Figure 16, and the individual pins in Figure 17. The displays are numbered sequentially from left to right.

To achieve this, we built the following circuit in Simulink:

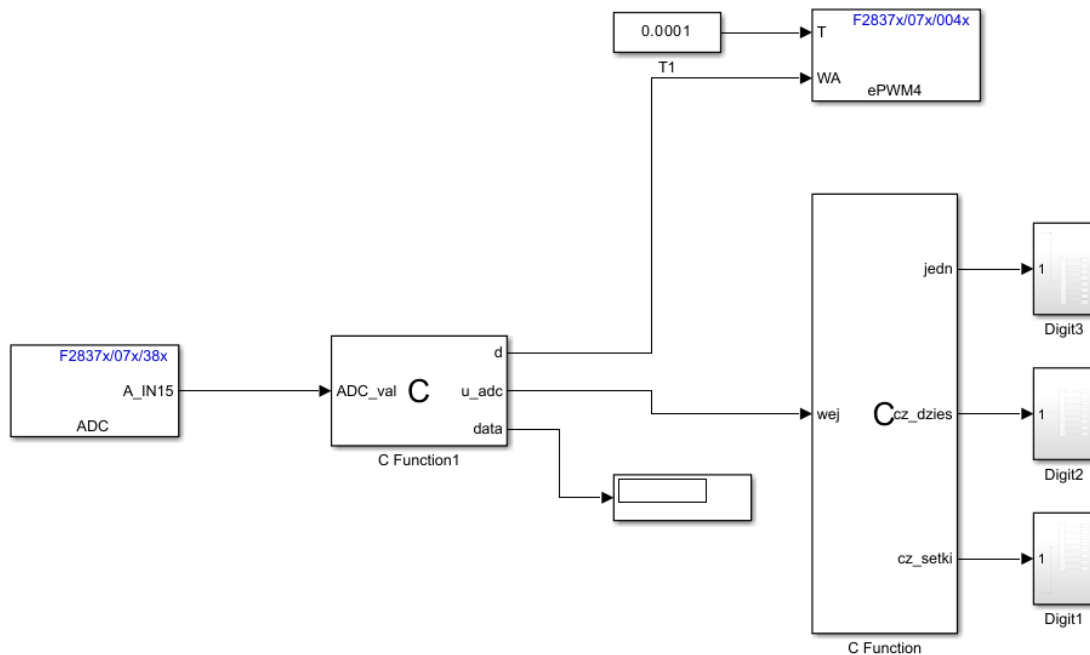


Figure 19 Circuit diagram

Initially, our goal was to display a specific number. We achieved this by using the “C funtion” block in digit 1, 2 and 3, whose code looks as follows:

```
int IN = (int)wej;
switch(IN)
{
    case 0:
        A=1;B=1;C=1;D=1;E=1;F=1;G=0;dot=0;
        break;

    case 1:
        A=0;B=1;C=1;D=0;E=0;F=0;G=0;dot=0;
        break;
}
```

```

case 2:
A=1;B=1;C=0;D=1;E=1;F=0;G=1;dot=0;
break;

case 3:
A=1;B=1;C=1;D=1;E=0;F=0;G=1;dot=0;
break;

case 4:
A=0;B=1;C=1;D=0;E=0;F=1;G=1;dot=0;
break;

case 5:
A=1;B=0;C=1;D=1;E=0;F=1;G=1;dot=0;
break;

case 6:
A=1;B=0;C=1;D=1;E=1;F=1;G=1;dot=0;
break;

case 7:
A=1;B=1;C=1;D=0;E=0;F=0;G=0;dot=0;
break;

case 8:
A=1;B=1;C=1;D=1;E=1;F=1;G=1;dot=0;
break;

case 9:
A=1;B=1;C=1;D=1;E=0;F=1;G=1;dot=0;
break;

default:
A=0;B=0;C=0;D=0;E=0;F=0;G=0;dot=0;
}

```

The outputs from this block are individual segments on a single display. The other two were done similarly.

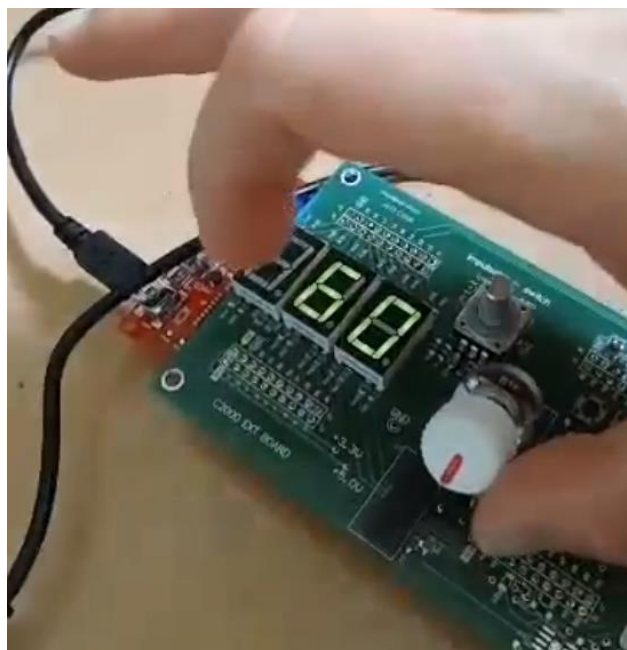


Figure 20 An example of a displayed number on a board

After a successful setup, we moved on to the next step, whereby by turning the potentiometer a specific value was calculated depending on the position. The first block from the schematic („C Function 1”) contains the corresponding potentiometer-dependent scaling. The signal is then passed to the next block with a function that appropriately divides the resulting value into unities, tens and hundreds. The configuration that lights up the corresponding segments on the display is then selected.

C Function 1 code:

```
u_adc = (ADC_val/4095)*3.30*100;  
d = (ADC_val/4095)*100;  
data=(ADC_val/4095)*100;
```

C Function code:

```
int liczba = (int)wej;  
jedn = liczba;  
cz_dzies = (liczba / 10)%10 ;  
cz_setki = liczba / 100;
```

Conclusion

The completion of the four tasks using the Texas Instruments C2000 MCU, specifically the TI Delfino F28379D LaunchPad, successfully demonstrated the ability to control LEDs based on both simple and complex button inputs, model a pulser rotation counter, and program an 8-segment display using MATLAB Simulink. Each task was implemented as planned: LEDs responded accurately to button presses and logical conditions, the pulser counter correctly adjusted a variable based on knob rotation direction and precision, and the 8-segment display showed the appropriate digits based on potentiometer input. These achievements underscore the versatility and effectiveness of the TI Delfino F28379D LaunchPad for various embedded system applications, despite challenges in program configuration and signal handling.