Embedded systems engineering CMPE3001

Lab Report

Laboratory 1:

Getting Started with Energia and MSP430 Experimenter's Board

Unit coordinator: Dr. Anuradhi Welhenge

Submission date: 19.09.2025

Group: Bentley (UG & PG) 35

Group members:

Name: Amiru Hangili Muhandiramlage (20818694)

Name: Dhrubo Troyee (22663281)

Name: Mohammed Ma'en Abu-Qamar (21026744)

Name: Nazinin Rahimi (21510234)

Name: Tashi Lhadon (22155746)

Table of contents

Sl No.	Contents	Page No.
1.	Introduction	3.
2.	Set up	4.
3.	Task 1: Turning a blue LED ON or OFF	5 – 7.
4.	Task 2: Flash LEDs in Round-robin sequence	8 – 9.
5.	Task 3: Working with an Interrupt	10 – 12.
6.	Task 4: Connecting a Servo Motor to MSP430	13 – 14.
7.	Task 5: Working with the onboard Temperature sensor	14 – 15.
8.	Conclusion	15.
9.	Discussion/Reflection	16.
10.	References	17.
11.	Appendices	18.
11. 1	Appendix A	18.
11. 2	Appendix B	19.
11. 3	Appendix C	20 – 21.

Introduction

This report reviews the feature set of the MSP-EXP430FR5739 Experimenter Board and demonstrates how its onboard hardware modules and external components can be programmed and controlled using the IDE (Energia Integrated Development Environment). Programming was conducted in C/C++, using Energia's Arduino-style libraries and functions to simplify microcontroller development.

Throughout the lab, structured, hands-on tasks were carried out to explore and understand key embedded system concepts. These included basic digital output control (blinking an onboard LED), implementing a round-robin LED sequence, handling external inputs through interrupt-driven programming using a push button, generating PWM signals to control a servo motor, and acquiring analog sensor data from the onboard temperature sensor using the ADC (Analog to Digital Converter) module as per the lab brief within Appendix C.

Each task was designed to demonstrate a specific hardware feature of the MSP430FR5739, utilizing built-in libraries, debugging UART (Universal Asynchronous Receiver-Transmitter) serial communication, and referencing the Energia documentation and the MSP430 online user manual [1][2]. Students were encouraged to engage with the development tools beyond just implementation, enabling an understanding of how microcontrollers interact with real-world hardware.

By the end of the lab, students developed practical skills in embedded programming, hardware-software integration, and microcontroller peripheral interfacing. These foundational competencies provide a solid base for more advanced embedded systems development and reflect essential concepts in modern embedded engineering. OBJ

Setup

The setup process for this laboratory required installation of Energia IDE version 16, as newer versions have known compatibility issues and bugs when used on Windows 10 systems. Once Energia was installed and the setup wizard completed, the MSP430FR5739 Experimenter Board was connected to the PC via USB. This USB connection serves multiple purposes: it supplies power to the board, enables flashing of compiled programs onto the microcontroller, and allows serial communication over UART (Universal Asynchronous Receiver-Transmitter).

After the board was connected, Windows Device Manager was used to identify the COM port assigned to the MSP430. If the COM port was not initially visible, running Windows Update or manually installing the appropriate Texas Instruments USB drivers resolved the issue. Once installed, the device typically appears as “TI MSP430 USB” under the Ports (COM & LPT) section.

With the COM port identified, the Energia IDE was configured accordingly. This involved navigating to the Tools menu in Energia, selecting "Board", and choosing "MSP-EXP430FR5739LP (LaunchPad/ MSP430FR5739)". Next, under "Port", the correct COM port (e.g., COM3, COM4, etc.) was selected, matching the one shown in Device Manager. This step is essential for enabling the IDE to upload code to the microcontroller and establish communication via the Serial Monitor.

Once these configurations were completed, the setup allowed for:

- Writing and compiling code within the Energia IDE
- Uploading the compiled code to the MSP430 board
- Monitoring serial data via the built-in Serial Monitor (Ctrl + Shift + M) or selecting the top right magnifying glass icon

This setup was used throughout all lab tasks to program the board and observe output via serial communication.

Task 1: Turning a blue LED ON or OFF

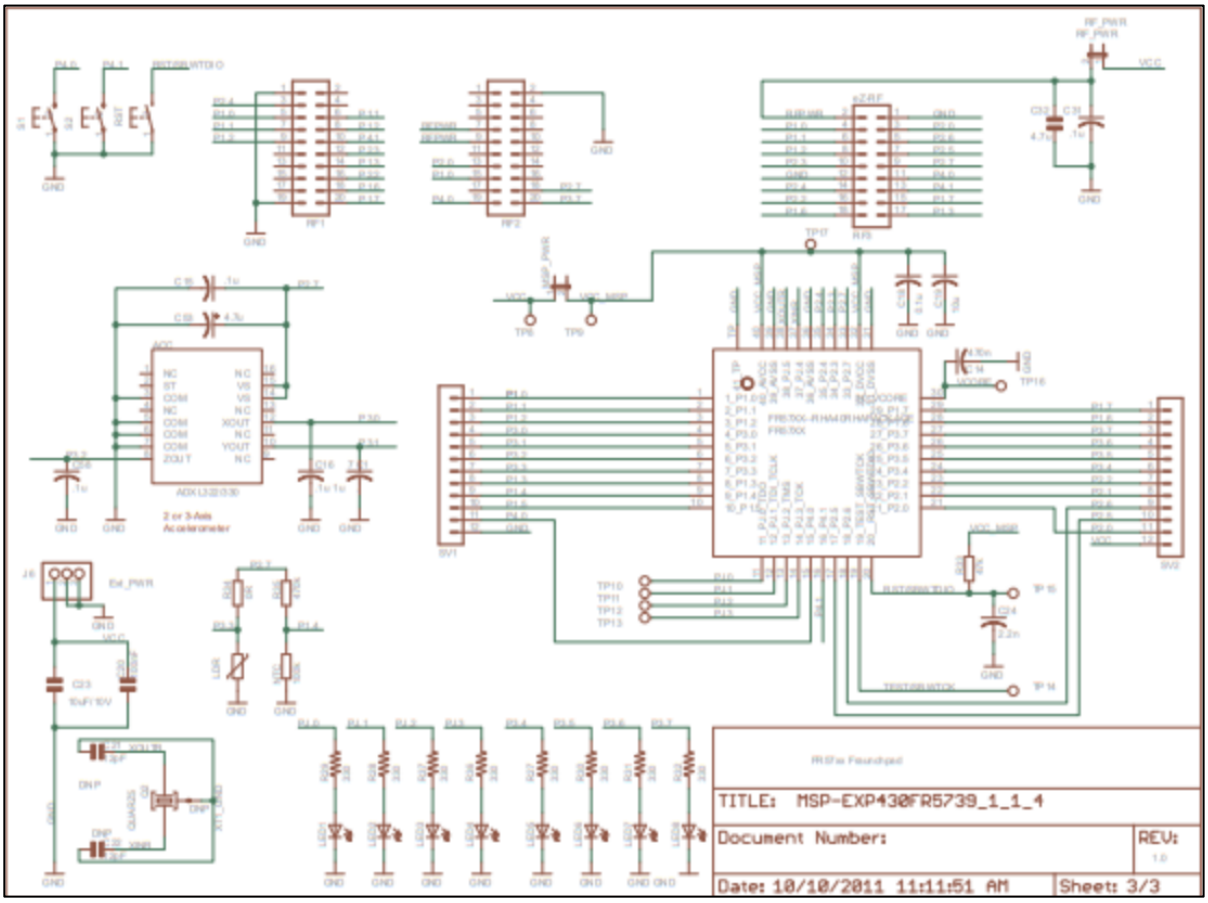
This task initially verified the development environment toolchain and hardware, confirming that the Energia IDE could correctly compile and upload code to the MSP430FR5739 microcontroller. The objective was to blink an onboard blue LED using Energia's built-in high-level macros/functions; however, the students also created code to directly memory-map the register. Reference code and documentation were obtained from the Energia website [2][3], which provided insight into MSP430 architecture, memory mapping, and peripheral control.

Circuit diagram

The circuit diagram for this task is relatively simple. The onboard LEDs (LED1 through LED8) are mapped to the following pins:

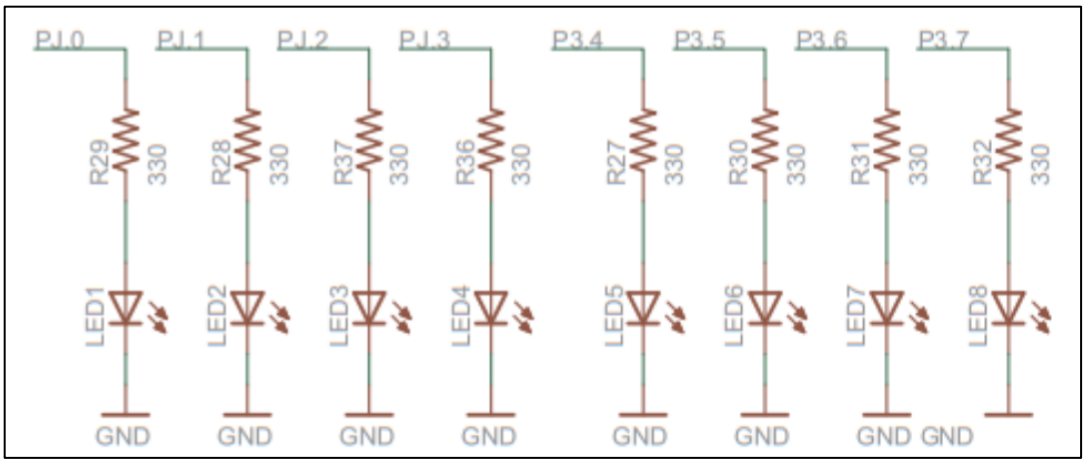
- PJ.0, PJ.1, PJ.2, PJ.3
- P3.4, P3.5, P3.6, P3.7

Each LED is connected in series with a 330Ω resistor to ground.



The diagram above displays the Circuit Diagram as shown in the User Guide [1]

Note: The section of the diagram used for this task is located at the bottom portion of the MSP430FR5739 user guide diagram [1].



Section referred to within the diagram [1]

Programming interface

MSP430 microcontrollers use a memory-mapped I/O architecture, allowing direct access to peripheral control via register addresses. Energia allows both low-level register manipulation and high-level abstractions through macros.

Direct Register Access (memory-mapped I/O)

The first implementation uses direct access to port direction and output registers to configure and control the LEDs. This approach gives full visibility into how the microcontroller sets pin directions and outputs.

```
void setup(){
    /* Stops Watch Dog Timer*/
    WDTCTL = WDTPW | WDTHOLD;

    /* Configures the LED pins as output*/
    PJDIR |= 0x0F;
    P3DIR |= 0xF0;
    /* Turns off all LEDs initially */
    PJOUT &= ~0x0F;
    P3OUT &= ~0xF0;
}
void loop(){
    /* Toggles ALL LEDs at the same time ON then OFF */
    PJOUT ^= 0x0F, P3OUT ^= 0xF0;
    /* 1 Second delay */
    delay(1000);
}
```

This code toggles all eight onboard LEDs on and off every second by inverting the output values of the corresponding ports.

In-built macros and functions

An alternative method uses Energia's high-level functions such as pinMode() and digitalWrite(), which abstract the low-level hardware interactions. This version is simpler and more readable but hides the underlying register operations.

```
char leds[] = {LED1, LED2, LED3, LED4, LED5, LED6, LED7, LED8}

void setup() {
    //Initialise the digital pin as an output.
    for(int i = 0; i < 8; i++){
        pinMode(leds[i], OUTPUT);
    }
}

// the loop routine runs over and over again forever:
void loop() {
    for(int i = 0; i < 8; i++){
        digitalWrite(leds[i], HIGH);   // turn the LED on (HIGH is the voltage
level)
    }
        delay(1000);               // wait for a second
    for(int i = 0; i < 8; i++){
        digitalWrite(leds[i], LOW);    // turn the LED off by making the voltage LOW
    }
    delay(1000);               // wait for a second
}
```

Energia example of doing the Blinking LED code [3][4]

While this code is easier to write and understand, it abstracts away the hardware-level operations, making it harder to learn what's happening "under the hood." For example, pinMode() and digitalWrite() are implemented using conditional logic that modifies the same memory-mapped registers as shown in the previous method.

These function definitions and macros can be found on the Energia GitHub repository [6], offering a deeper understanding of how these high-level calls translate to low-level operations, which was forked from the Arduino project.

Observation

This task confirmed that the MSP430FR5739 and Energia IDE worked correctly by blinking all onboard LEDs. Two approaches were tested: direct register access and high-level Energia functions. The register method gave more control and insight into hardware operation, while the high-level method was simpler and easier to write. Both achieved the same result, highlighting the trade-off between control and simplicity. This helped build a stronger understanding of embedded programming basics.

Task 2: Flash LEDs in Round-robin Sequence

The Setup

```
#define LEDA LED1
#define LEDB LED2
#define LEDC LED3
#define LEDD LED4
```

These #define statements are just aliases for the LED pins already predefined in the Energia.

```
void setup() {
  pinMode(LEDA, OUTPUT);
  pinMode(LEDB, OUTPUT);
  pinMode(LEDC, OUTPUT);
  pinMode(LEDD, OUTPUT);
}
```

The setup() function runs once when the board powers up. Each LED pin is configured as an **OUTPUT**, meaning the microcontroller will drive these pins HIGH (on) or LOW (off).

The loop

```
void loop() {
  digitalWrite(LEDA, HIGH);
  delay(500);
  digitalWrite(LEDA, LOW);
  delay(500);

  digitalWrite(LEDB, HIGH);
  delay(500);
  digitalWrite(LEDB, LOW);
  delay(500);

  digitalWrite(LEDC, HIGH);
  delay(500);
  digitalWrite(LEDC, LOW);
  delay(500);

  digitalWrite(LEDD, HIGH);
  delay(500);
  digitalWrite(LEDD, LOW);
  delay(500);
}
```

The loop() here runs forever. For each LED (LEDA → LEDB → LEDC → LEDD): It turns ON (HIGH) for 500 milli seconds with delay (500). Then turns OFF (LOW) for 500 milli seconds. Once LEDD finishes, the loop restarts back at LEDA → creating a round-robin / rotating sequence**.**

When the modified Blink sketch was uploaded, all four LEDs began blinking one after the other, in a round-robin fashion. Rather than designate one LED to blink repeatedly, the program cycled through each of the four LEDs one by one, in order (e.g., LED1 → LED2 → LED3 → LED4 → LED1). The pins were mapped in following order:

- LED A → pin 2
- LED B → pin 3
- LED C → pin 4
- LED D → pin 5

At any moment, only one LED would be turned on, and once its delay was exhausted, it would turn off and turn on the next LED in the sequence. This gave the effect of lights chasing each other, which is frequently seen in displays.

The delay time had a significant effect on how the sequence looked:
A very short delay time (50-100 ms), LEDs changed quick enough that the sequence looked blurred or too fast to see.
At medium delay time (200–500 ms), the sequence was very clear and easy to see.
At long delay time (1000 ms or more), the LEDs turned on one at a time slow enough to almost look like they were “taking turns” in the pattern.
This round-robin behavior illustrated the principles of fair scheduling, as each LED was granted a time slice before the LED passed the control to the next one, and no LED was skipped (or starved) in the process.

Observation and experimentation

We even experimented by changing pin addresses of the code like reversing order by putting LED 1 → pin 5, LED 2 → pin 4, LED 3 → pin 3 and LED 4 → pin2 and noticed that the LEDs turned on in the new order according to the pin number assignment. The changes also demonstrated the possibility of how the code controlled the mapping of a physical signal. Likewise, changing the delay values affected the speed of the pattern, higher the delay values longer it takes to blink, lower the delay values, lesser time the LEDs took to blink and thus the lights were changed to be faster or noticeably slower. These changes allowed us to easily see the effect of timing and pin assignments to the round-robin LED behavior.

Task 3: Working with an Interrupt

Circuit description

- Inputs include an internal pull-up resistor and an on-board push button (PUSH2).
- Outputs: The application defines four on-board LEDs (LED1–LED4).
- Since every component was a part of the MSP430FR5739 device, no external wiring was necessary.

Code explanation

```
#define LEDA LED1
#define LEDB LED2
#define LEDC LED3
#define LEDD LED4

volatile int flag = LOW;
int count = 0;

void setup()
{
  pinMode(LEDA, OUTPUT);
  pinMode(LEDB, OUTPUT);
  pinMode(LEDC, OUTPUT);
  pinMode(LEDD, OUTPUT);

  pinMode(PUSH2, INPUT_PULLUP);   // configure button with internal pull-up

  attachInterrupt(PUSH2, blink, FALLING); // Interrupt is fired whenever button is pressed
}

void loop()
{
if(flag)

{
 // LED1 Blink
 digitalWrite(LEDA, HIGH);   // turn the LED on (HIGH is the voltage level)
 delay(500);              // wait for a second
 digitalWrite(LEDA, LOW);    // turn the LED off by making the voltage LOW
 delay(500);            // wait for a secon

 // LED2 Blink
 digitalWrite(LEDB, HIGH);
 delay(500);
 digitalWrite(LEDB, LOW);
 delay(500);

 // LED3 Blink
 digitalWrite(LEDC, HIGH);
 delay(500);
```

```
 digitalWrite(LEDC, LOW);
 delay(500);

 // LED4 Blink
 digitalWrite(LEDD, HIGH);
 delay(500);
 digitalWrite(LEDD, LOW);
 delay(500);
flag = LOW;
 }
}

void blink()
{
 flag = HIGH;
}
```

Configuration

- As outputs, four LEDs (LEDA–LEDD) are selected and set up.
- To avoid floating input, the push button is assigned to INPUT_PULLUP.
- When the button is pushed, an interrupt is connected to the falling edge:

```
attachInterrupt(PUSH2, blink, FALLING); // whenever button (push2) is pressed ISRblink() executed
```

Loop

- Here, the flag variable is verified by the loop. However, the software executes the following sequence if the flag is HIGH (set by the ISR):
- Turn on and off/ blink LED1 for 0.5 seconds.
- Turn LED2 on and off for 0.5 seconds
- Turn LED3 on and off for 0.5 seconds.
- Turn LED4 on and off for 0.5 seconds.

Therefore, the program waits for the next button click after the entire sequence ended since the flag is cleared (flag = LOW).

Interrupt Service Routine (ISR):
The blink() function always sets flag = HIGH when the button is pressed.
This triggers the main loop to begin blinking the LEDs.

Program structure

- All the LEDs are off throughout startup.
- The mechanism needs the pressing of a button.
- ISR sets flag = HIGH when a button is pressed.
- The blinking sequence for LED1 → LED2 → LED3 → LED4 is initiated by the main loop upon detecting the flag.
- The system waits for the next button press once the four LEDs are finished and the flag is reset. Consequently, the entire 4-LED blinking sequence happens again with each new button press

Observation

None of the LED are lighted, at the beginning. All four LEDs blink sequentially (LED1 → LED2 → LED3 → LED4) when the push button is pressed once. Before the following LED lights on, the previous ones turn off after a half-second. The system only waits for another button press after the entire process has completed. The entire 4-LED blinking sequence is triggered by pressing a single button. This validate that the LED sequence was appropriately triggered by the push button interrupt as well as the ISR function, which handles code execution in the main loop, functioned according to plan.

Comparison with Task 4

Overall, it can be said that Push-button code (Task 5): Each button when pressed proceeds to the following LED in sequence, with only one LED turning on at a time.
However, the round-robin code (Task 4 without button) requires no button pressing; instead, the LEDs automatically blink in a loop.

Task 4: Connecting a Servo Motor to MSP430

Hardware Connection

Connecting to the Servo Motor

For this task, MSP430FR5739 board was used to interface a servo motor, which was then programmed using Energia. The three terminals of the Servo motor (VCC, GND, Signal) were connected to the MSP430FR5739 board as follows using jumper wires.

1. VCC was connected to the 5 V power output to provide operating voltage
2. GND was connected to the ground reference to complete the circuit
3. Signal was connected to GPIO P1.5

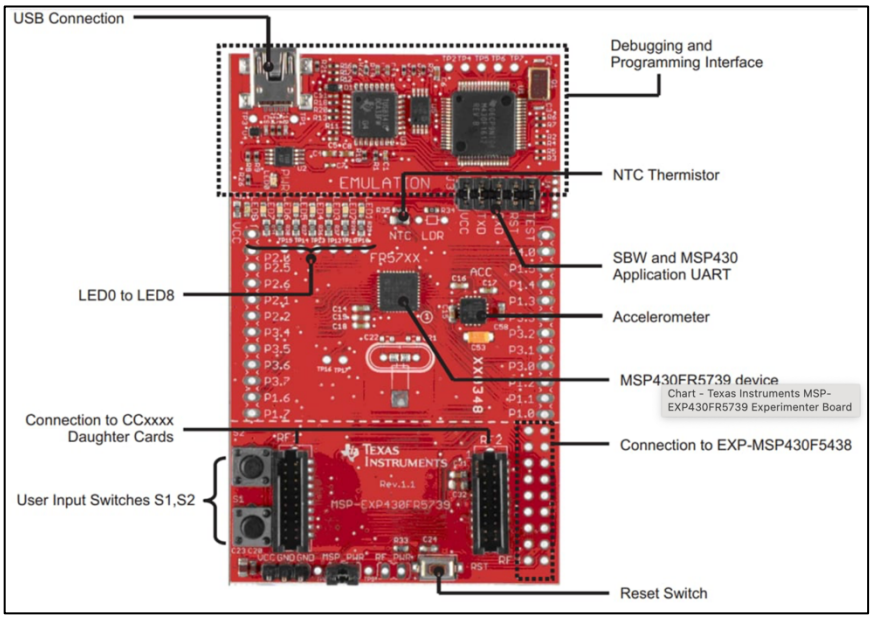
```
#include <Servo.h>
Servo myservo; // create servo object to control a servo
                            // a maximum of eight servo objects can be created

int pos = 0;// variable to store the servo positions

void setup()
{
   myservo.attach(19);  // attaches the servo on pin 9 to the servo object
}

void loop()
{
   for (pos = 0; pos < 180; pos += 1) // goes from 0 degrees to 180 degrees
    {                                 // in steps of 1 degree
        myservo.write(pos); // tell servo to go to position in variable 'pos'
        delay (15); // waits 15ms for the servo to reach the position
    }
   for (pos = 180; pos >= 1; pos -= 1) // goes from 180 degrees to 0 degree
    {
        myservo.write(pos); // tell servo to go to position in variable 'pos'
        delay (15); // waits 15ms for the servo to reach the position
}

}
```

Code explanation

This code manages a servo motor using the MSP430. A Servi object named “myservo” is created to interface with the servo. The starting position of the servo is set to 0 degrees by using an integer variable 'pos'. The servo is connected to pin 19, establishing the connection between myservo and the servo motor. The code uses a loop () method with two for-loops to alternate the servo's position. The first for-loop moves the servo from 0° to 180 degrees, in 1° progressive increments. There is a 15 ms delay following each movement to allow the acclamation of the server into the new position. The second for-loop then moves the servo back from 180° to 0°, with 1° steps and a similar delay. Together, these two for-loops result in a sweeping motion.

Task 5: Working with the onboard Temperature sensor

In this task, the NTC_FRAM program was used to read the onboard NTC temperature sensor of the MSP430FR5739 Board. The program starts the sensor, uses a library to read it, and shows the temperature on the Energia Serial Monitor. Temperature values are displayed in both Celsius and Fahrenheit with one decimal place, updated roughly every half second. An option is also included to stop the output by pressing the PUSH2 button.



Note: the label NTC Thermistor is the NTC sensor mentioned in the previous paragraph.

The results showed that the temperature readings increased when the board was warmed by touch and decreased when cooled by air, confirming the sensor's proper operation. The serial monitor displayed continuous values of temperature, as soon as it was cooled with air the values dropped significantly and once restored back to room temperature the values were noticeably increasing with around 1 to 2 degrees. This task demonstrated how an onboard analog sensor can be measured, processed, and displayed in real time, while also showing the role of libraries in simplifying hardware interfacing.

Conclusion

Lab 1 provided a practical introduction to the capabilities of the Energia IDE and the MSP430FR5739 microcontroller. and Energia IDE. It allowed the students to gain practical experience in essential embedded programming concepts such as digital input/output, pulse-width modulation (PWM), and interrupt handling. It also allowed them to attempt tasks that demonstrate how software interacts with hardware components, such as blinking LEDs, running a round-robin sequence, using button interrupts, reading the onboard temperature sensor, and sweeping a servo motor. These activities enriched their understanding of concepts like hardware integration, timing, and control signals. It helped to gain the necessary skills and confidence to successfully conduct advanced embedded systems projects in the future.

Discussion/Reflection

This lab provided practical experience with the MSP430FR5739 board and the Energia IDE. Setting up the board, selecting the correct COM port, and uploading code were the first steps, forming the foundation for later tasks. Although basic, these actions were essential for ensuring reliable communication between hardware and software.

The LED exercises demonstrated the principles of digital output control. Blinking and sequencing activities highlighted the use of functions such as pinMode(), digitalWrite(), and delay() to manage timing and code structure. The introduction of interrupts extended this knowledge by showing how a system can respond to external inputs, such as a push button, rather than relying only on fixed delays. This emphasized the importance of efficiency and responsiveness in embedded systems.

More advanced tasks expanded into practical applications. Controlling a servo motor through PWM signals illustrated how microcontrollers generate precise timing outputs to devices, a concept central to robotics and automation. Similarly, reading values from the onboard temperature sensor highlighted how analog signals are processed through the ADC and then displayed as meaningful data, linking sensing with interpretation.

Overall, these activities demonstrated how microcontrollers integrate inputs and outputs into functional systems. Concepts including digital I/O, interrupts, PWM, and ADC formed a progression from simple control to more complex interactions. The lab also shows the importance of debugging, careful wiring, and structured coding in achieving reliable results, providing a clear view of how embedded hardware and software operate as one system.

References

[1] Texas Instruments, “MSP-EXP430FR5739 FRAM Experimenter Board User's Guide” ti.com Accessed Sep. 15, 2025. [Online.] Available: https://www.ti.com/lit/ug/slau343b/slau343b.pdf

[2] Texas Instruments, “MSP430FR57xx Family User's Guide” ti.com Accessed Sep. 15, 2025. [Online.] Available: https://www.ti.com/lit/ug/slau272d/slau272d.pdf

[3] Energia, “Blink” energia.nu Accessed Aug. 08, 2025. [Online.] Available: https://energia.nu/guide/tutorials/basics/tutorial_blink

[4] Energia, “Example 00: Blinking LED (Output)” energia.nu Accessed Sep. 15, 2025. [Online.] Available: https://energia.nu/guide/tutorials/other/sidekick/sidekick_blink

[5] Energia, “Example: Spin the Servo (Servo Control)” energia.nu Accessed Sep. 15, 2025. [Online.] Available: https://energia.nu/guide/tutorials/other/sidekick/sidekick_spinservo

[6] Energia, “wiring_digital.c” https://github.com Accessed Sep. 15, 2025. [Online.] Available: https://github.com/energia/Energia/blob/master/hardware/msp430/cores/msp430/wiring_digital.c

Appendices

Appendix A

Task	Amiru H M	Dhrubo T	Mohammed A Q	Nazinin R	Tashi L
Introduction			√		
Setup			√		
Task 1			√		
Task 2					√
Task 3		√			
Task 4	√				
Task 5				√	
Conclusion	√				
Discussion/Reflection				√	
References					
Appendices			√		√
Formatting & Compiling					√
Proof reading		√			

Appendix B

Lab Report

1. Assignment overview

You are tasked with writing a review based on the lab component of Embedded Systems CMPE3001. You need to cover the following points:

- Diagrams of the physical circuit connections and explaining the connectivity (You can use photos from the lab or Tinkercad for explaining the connections)

- Explanation of lab 1 - You will need to explain your code clearly using written language and flow charts.

Marking Sheet for Lab Report

Criterion	**Unsatisfactory/ Unavailable 0% to 50%**	**Good 50% to 65%**	**Very Good 65% to 80%**	**Perfect 80% to 100%**	**Mark %**
Hardware Components [30%]	Not enough detail provided for hardware components	A brief discussion of hardware connections without any diagrams	There were some diagrams available, but they required a bit more detail	All hardware connections were clearly explained through photos and diagrams, and all connections were properly labelled and explained	
Software Components [40%]	Not enough detail provided for software components	A brief discussion of implemented code without evidence	Some code explanation was present, but some implementation lacked justification	All software implementations were clearly presented with detailed explanation of the code written for different tasks	
Communication Skills [30%] Technical writing, structure, content flow, and quality and presentation of engineering results	Informal tone and language used often. Writing is unclear, vague, or not concise. Engineering language not used correctly. Regular misuse of grammar, spelling and punctuation leading to difficulty in discerning meaning.	Rare misuse of grammar, spelling, and punctuation. Rare incorrect or insufficient use of recommended referencing style. Graphical representation of data is correct and can be interpreted.	Report has a good structure, and convincing argument. Professional and technically accurate engineering language with an appropriately formal tone used throughout. Correct use of grammar, spelling, and punctuation throughout. Minor thesis formatting	The report demonstrates high-level in all communication aspects: style, narrative, graphics, figures, format, etc. It is evident the student has much experience writing technical reports.	
Total [100%]					

Appendix C

Curtin University

EMBEDDED SYSTEMS ENGINEERING CMPE3001
Department of Electrical and Computer Engineering

Laboratory 1: Getting Started with Energia and MSP430 Experimenter's Board

Equipment Required:

1 × MSP430FR5739 Experimenter Board Curtin's Kit (USB programming cable, breadboard, wires, Servo Motor)
1 × PC with Energia software installed

1. Objectives

The purpose of this laboratory is to get started with the Energia integrated development environment (IDE) and use it to program an MSP430 supported development board, the MSP430FR5739 Experimenter Board by Texas Instrument.

The objectives of this laboratory session can be summarised as follows:
Learn how to use Energia IDE.

Get familiar with the MSP430FR5739 experimenter's board: Programming, compiling and loading.

2. Pre-laboratory

Before coming to the laboratory:

- Get familiar with the MSP430FR5739 experimenter's board, which will be referred to as the MSP430 development board or MSP430 board in this unit, by **reading the user's manual** available from Blackboard (Bb). Write any useful notes in your logbook.
- Review on Pulse Width Modulation (PWM) for the MSP430 and study the code from https://energia.nu/guide/tutorials/basics/tutorial_fade/
- Read this laboratory script and make sure you understand what is expected.

3. General Recommendations

Some general recommendations to follow:

- Follow your lab supervisor's safety instructions.
- This lab is an introduction to the hardware and software and is not marked. So, please take your time understanding the software and hardware and feel free to ask your lab supervisors for advice. Keep in mind the lab supervisors will only give you general directions and any debugging of the code will be on the students.
- You can download the Energia software and device driver from: Energia download link
- Please download version 16 because the later versions have some bugs specially for people who are using Windows 10.
- IMPORTANT: If you want to keep a copy of your work projects, compress the whole directory into a ZIP file and copy it to your thumb or I:\ drive.

4. Introduction

The MSP430 board is packed full of features. In this unit, we will be exploring some of the capabilities of the board by performing certain tasks which will make use of some inbuilt interfaces to the external world.

Working with the MSP430 board requires some prior knowledge of the external pin functions for program setup and hardware interfacing. The pinout is shown in Figure 1 is particularly useful.

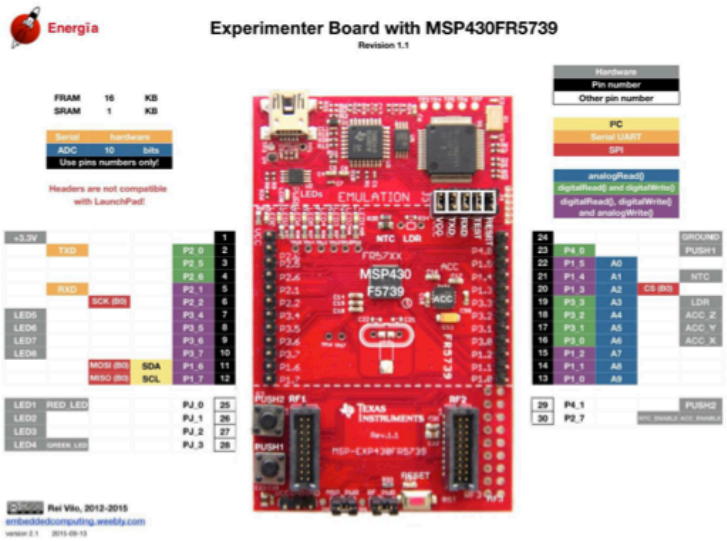


Figure 1 MSP430FR5737 pinout.

5. Tasks

1. After connecting your MSP430 to your computer, open Energia and go to Tools →Board and choose FraunchPad w/MSP430FR5739.

2. Go to Tools →Serial Port and choose the COM port connected to your MSP430. If you are working with the computers in the lab, it will be any COM port other than COM 1.

3. Turning a **blue LED** ON or OFF (Source Code: Blink from Energia)

The example sketch provided by the Energia IDE under the File →Examples (Blink) will allow you to pulse an LED on the experimenter's board using a software timer. Compile and upload to the MSP430 board. Show the programs running to your lab supervisor to get the marks for this task. Make sure that you record down important notes in your logbook that will help you achieve this task.

For this task, you will have to study the Blink sketch and get any hints from the comments to setup the correct blue LED for blinking. Define the correct BLUE_LED definition in your code by attaching to the correct pin by using the pin mapping shown in Figure 1. Also structure your code to make easier to modify for future changes. It is also a good practice to make use of the Help menu. For example, use Help if you do not know how to use a particular function by looking for the Language Reference.

4. Flash LEDs in Round-robin Sequence (Source Code: Round Robin Example from blackboard)

Modify your Blink sketch to toggle in a round-robin sequence, the four LEDs (use the pin map in Figure 1 for help). Remember to save as a different sketch if you want to keep a record of all your programs. Experiment between the delay times and use a delay that is not too short for the sequence to be visible. Show this task to your lab demonstrator upon completion.

5. Working with an Interrupt (Source Code: Push Button Example from blackboard)

Change your Task 4 sketch to round-robin sequence the four LEDs when an on-board pushbutton is pressed. In order to detect key presses, you will have to detect on interrupt. Setup your sketch so that the MSP430 operates in the low power mode. Make sure you understand what is happening with your sketch when you upload to the MSP430 board. Record any notes in your logbook.

For this task, toggle between any two chosen on-board LEDs when a button is pressed. Show this task to your laboratory supervisor upon completion.

An example on what your sketch may contain is shown below. This example is taken from the Energia Reference Help. You may look up for attachInterrupt and study the function and record down any important notes. Pay attention to the Note stated in the attachInterrupt reference and understand what it means. The example shows the use of the Serial function for printing results in the Serial Monitor (Ctrl+Shift+M). Using the serial monitor is useful to display status of your sketch and also assist in code debugging purposes.

Example

```
volatile int state = HIGH;
volatile int flag = HIGH;
int count = 0;
void setup()
{
	Serial.begin(9600);
	pinMode(GREEN_LED, OUTPUT);
	digitalWrite(GREEN_LED, state);
	/* Enable internal pullup.
	* Without it, the pin will float and the example will not work. */
	pinMode(PUSH2, INPUT_PULLUP);
	attachInterrupt(PUSH2, blink, FALLING); // Interrupt is fired whenever button is pressed
}

void loop()
{
	digitalWrite(GREEN_LED, state); //LED starts ON
	if(flag) {
		count++;
		Serial.println(count);
		flag = LOW;
	}
}

void blink()
{
	state = !state;
	flag = HIGH;
}
```

6. Connecting a Servo Motor to MSP430 (Source Code: Servo Sweep from Energia)

In this example you will be connecting a servo motor to your MSP430. Servo motor have 3 pins, VCC, GND, Signal. Connect the VCC pin and GND pin on the respective pins on your MSP430. Connect the signal pin to one of the generic GPIO pins of your microcontroller. Under the built in Energia examples, find the servo sweep example. This code generates a PWM signal to sweep the servo motor. Modify the pin in the source code to the one you have connected the signal pin into. Observe the behavior of the servo motor and try modifying the code so the servo motor goes to different angles.

7. Working with the onboard Temperature sensor (Source Code: NTC_FRAM from blackboard)

Download the NTC_FRAM source code from blackboard and put the program on the board. Note that the header file and library has been written in C++. This program read the data from the temperature sensor on the board and displays it on the serial monitor which can be seen on the top right of your Energia window with a magnifying glass icon.