

ESD Assignment No. 2

NAME : ASHUTOSH RAJENDRA KARVE
BITS ID : 2024HT01021
BITS MAIL : 2024ht01021@wilp.bits-pilani.ac.in
PHONE NO : +91 9765541324
DATE : 19-10-2024
PLACE : PUNE

GITHUB LINK

https://github.com/Ashutoshkarve007/BitsLearning/tree/main/Embedded_System_Design/ESD_Assignments/Lab_Assignment_2

Q.1. Write a C program for displaying your BITS ID on 1st Row and voltage difference between the terminals of the potentiometer, along with the date in the DD/M format, on the 2nd row of LCD Display present in the LPC2378 kit.

Introduction

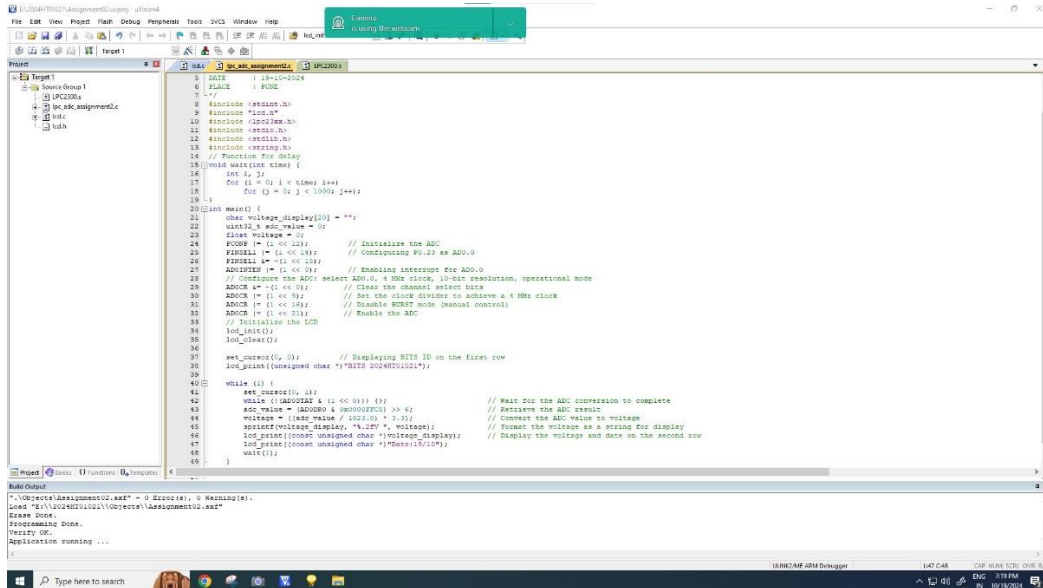
For this question, I implemented a C program that reads the analog input from a potentiometer connected to the LPC2378 microcontroller. The program calculates the corresponding voltage from the potentiometer and displays both the voltage and the current date (in DD/M format) on an LCD screen. The BITS ID is displayed on the first row, and the voltage and date are shown on the second row. This task was performed using Keil uVision IDE and the LPC2378 remote lab kit.

Hardware Setup

- **LPC2378 Kit**
- **Potentiometer** connected to AD0.0 pin (P0.23)
- **LCD Display** connections:
 - DB4: P1.24
 - DB5: P1.25
 - DB6: P1.26
 - DB7: P1.27
 - RS: P1.28
 - RW: P1.29
 - E: P1.31

Code Explanation

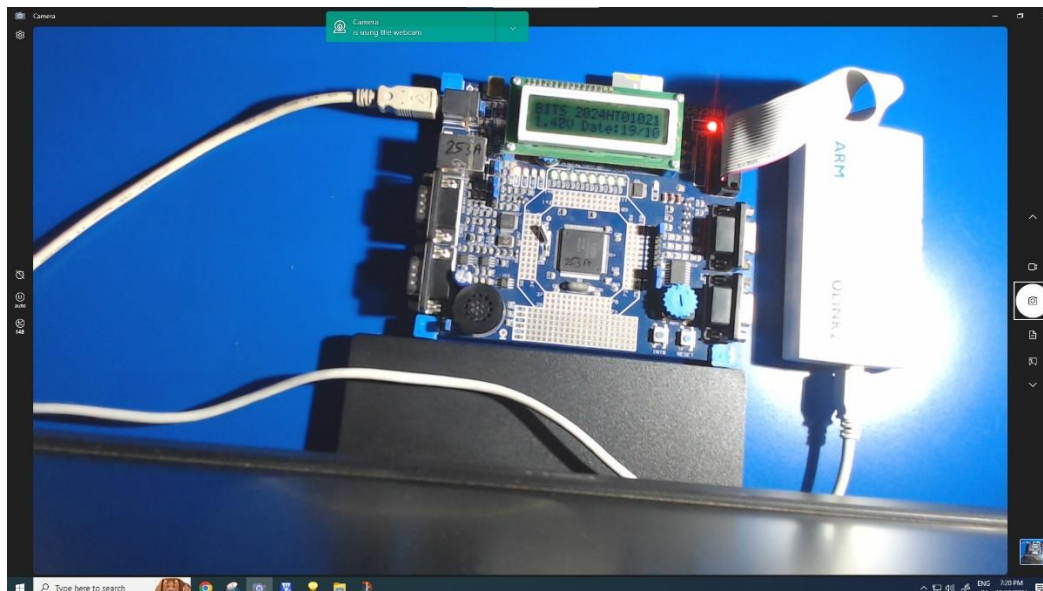
The following code initializes the ADC to read the potentiometer values, converts the ADC result into a voltage (based on a 3.3V reference), and displays both the BITS ID, voltage, and the date on the LCD screen.



```
1  #include <stm32f10x.h>
2  #include <math.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <string.h>
6  #include <time.h>
7  #include <stdarg.h>
8  #include <stdbool.h>
9  #include <stdint.h>
10 #include <sys/time.h>
11 #include <sys/types.h>
12 #include <unistd.h>
13 #include <fcntl.h>
14 // Function for delay
15 void wait(int time) {
16     int i;
17     for (i = 0; i < time; i++)
18         ;
19 }
20 int main() {
21     char voltage_display[20] = "";
22     uint32_t adc_value = 0;
23     float voltage = 0;
24     PCORR = (1 << 23); // Initialize the ADC
25     FSRHLL = (1 << 10); // Comparing 95.23 as ADC.0
26     FSRHLL = (1 << 10); // Enabling interrupt for ADC.0
27     // Configure the ADC: select ADC.0, 4 MHz clock, 10-bit resolution, operational mode
28     ADCCR = (1 << 0); // Clear the channel select bits
29     ADCCR = (1 << 0); // Set the clock divider to achieve a 4 MHz clock
30     ADCCR = (1 << 0); // Enable SMCT mode (manual control)
31     ADCCR = (1 << 0); // Enable the ADC
32     // Initialize the LCD
33     lcd_init();
34     lcd_clear();
35     // Displaying BITS ID on the first row
36     lcd_puts("BITS 2024HT01021");
37     // Displaying date on the second row
38     lcd_puts("Date:12/10");
39     while (1) {
40         // Wait for the ADC conversion to complete
41         while ((ADCR & (1 << 0)) != 0) {}
42         // Retrieve the ADC result
43         adc_value = ADCDR & 0xFFFF;
44         // Convert the ADC value to voltage
45         voltage = (float)adc_value / 1023.0 * 3.3;
46         // Round the voltage as a string for display
47         sprintf(voltage_display, "%.2fV", voltage);
48         // Display the voltage and date on the second row
49         lcd_puts(voltage_display);
50         wait(1);
51     }
```

Testing

The program was tested on the LPC2378 kit using Keil uVision IDE. The ADC correctly reads the potentiometer's analog value and converts it to a corresponding voltage, which is displayed on the LCD along with the BITS ID and the date in the specified format. The system outputs were verified by entering debug mode in Keil IDE and monitoring the ADC values, voltage, and other variables.



Q.2. Answer the following questions related to LPC2378:

Question:

a) What is the smallest change in input voltage that the ADC can detect? (+Vref = 3.3 V)

Explanation and Solution:

The LPC2378 microcontroller features a 10-bit Analog-to-Digital Converter (ADC). The resolution of an ADC is determined by the number of bits used to represent the input voltage. For a **10-bit ADC**, the input voltage range from **0V** to **+Vref** (which in this case is 3.3V) is divided into $2^{10}=1024$ discrete steps.

Step 1: Understanding ADC Resolution

The resolution of the ADC defines the smallest change in input voltage that it can detect. For a **10-bit ADC**, the input voltage is divided into **1024 steps**.

The formula to calculate the smallest detectable change (ΔV_{min}) is:

$$\Delta V_{min} = \frac{V_{ref}}{2^n} = \frac{V_{ref}}{1024}$$

Where:

- $V_{ref} = 3.3V$ (the reference voltage)
- $n = 10$ (since it's a 10-bit ADC)

Substituting the value into the formula:

$$\Delta V_{min} = \frac{3.3V}{1024} = 0.0032226V$$

This gives us:

$$\Delta V_{min} \approx 3.22mV$$

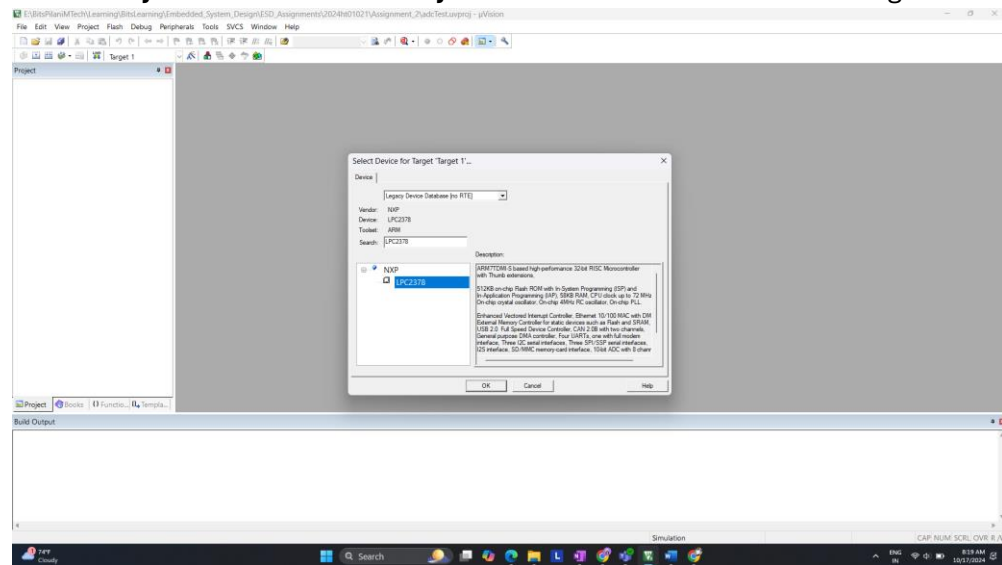
Step 2: Verifying the Calculation in Keil uVision5 Simulation

To simulate the behavior of the ADC and verify the result, we can use the Keil uVision5 built-in simulator. Below are the detailed steps to simulate the ADC and see how this minimum voltage change is represented in the ADC result.

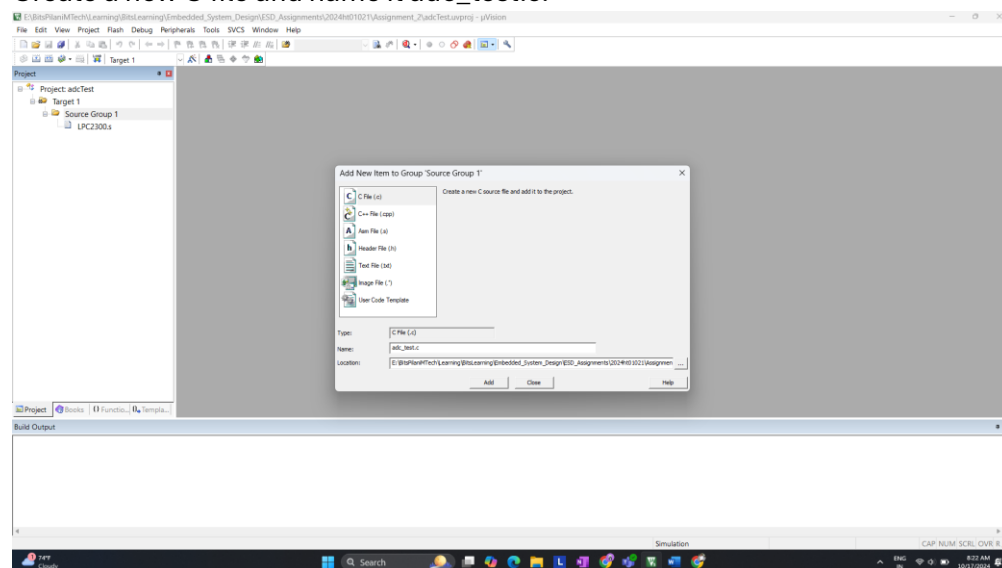
Step-by-Step Guide for Simulation in Keil uVision5

1. Creating the Project:

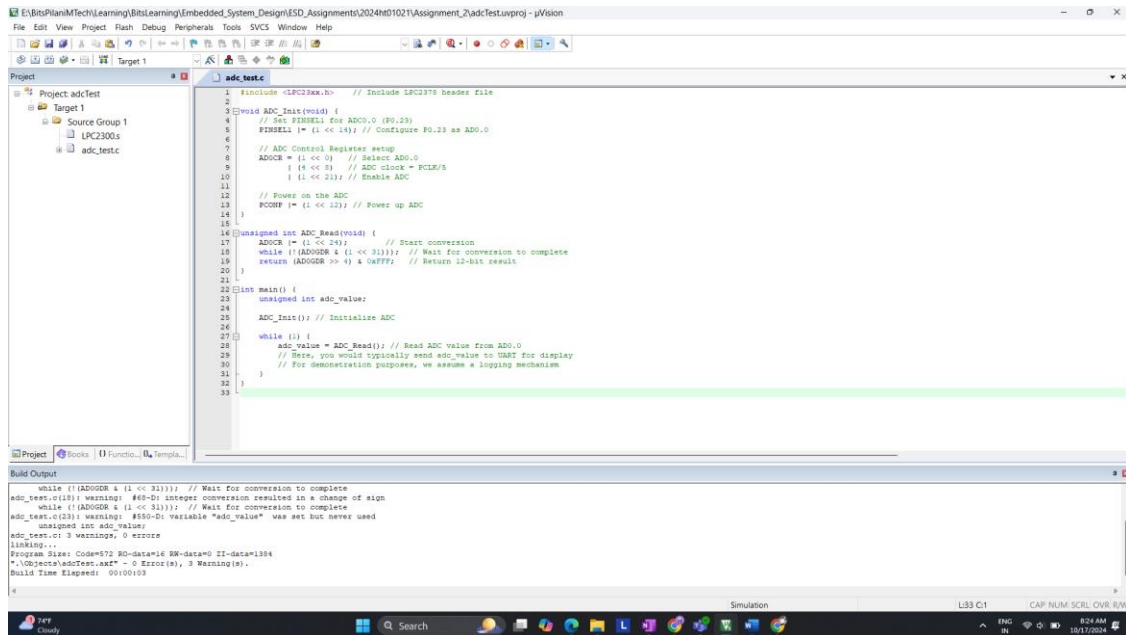
- Open **Keil uVision5**.
- Select **Project > New uVision Project** and choose **LPC2378** as the target device.



- Create a new C file and name it **adc_test.c**.

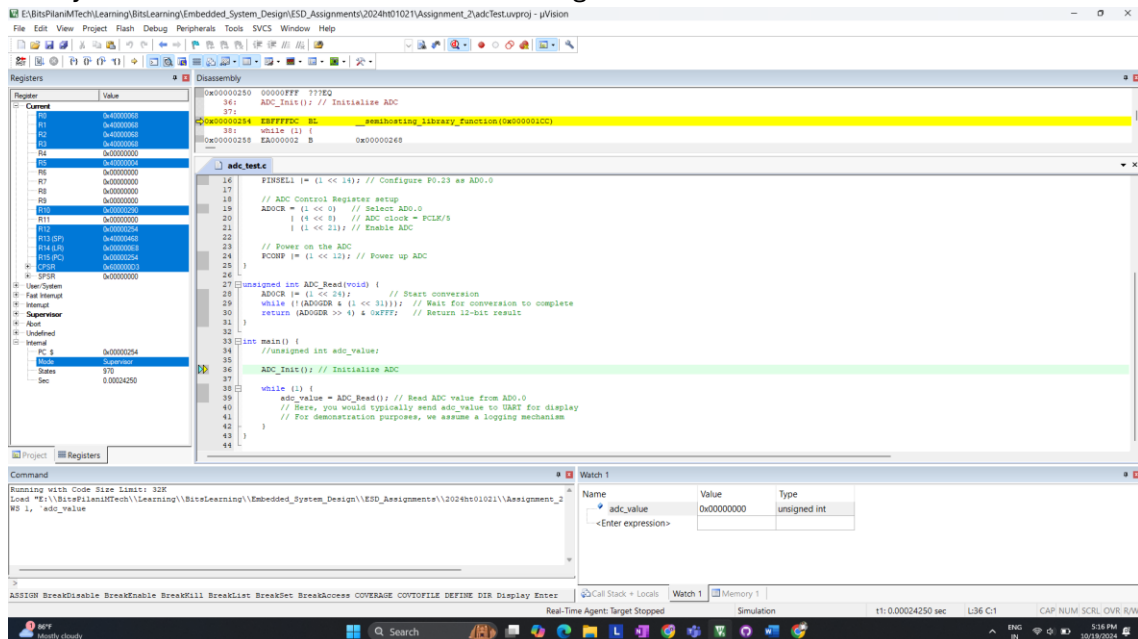


2. **Write the Code:** Below is an example code that initializes the ADC on LPC2378 and reads from channel 0 (AD0.0). The program reads the ADC value and stores it in the `adc_value` variable.



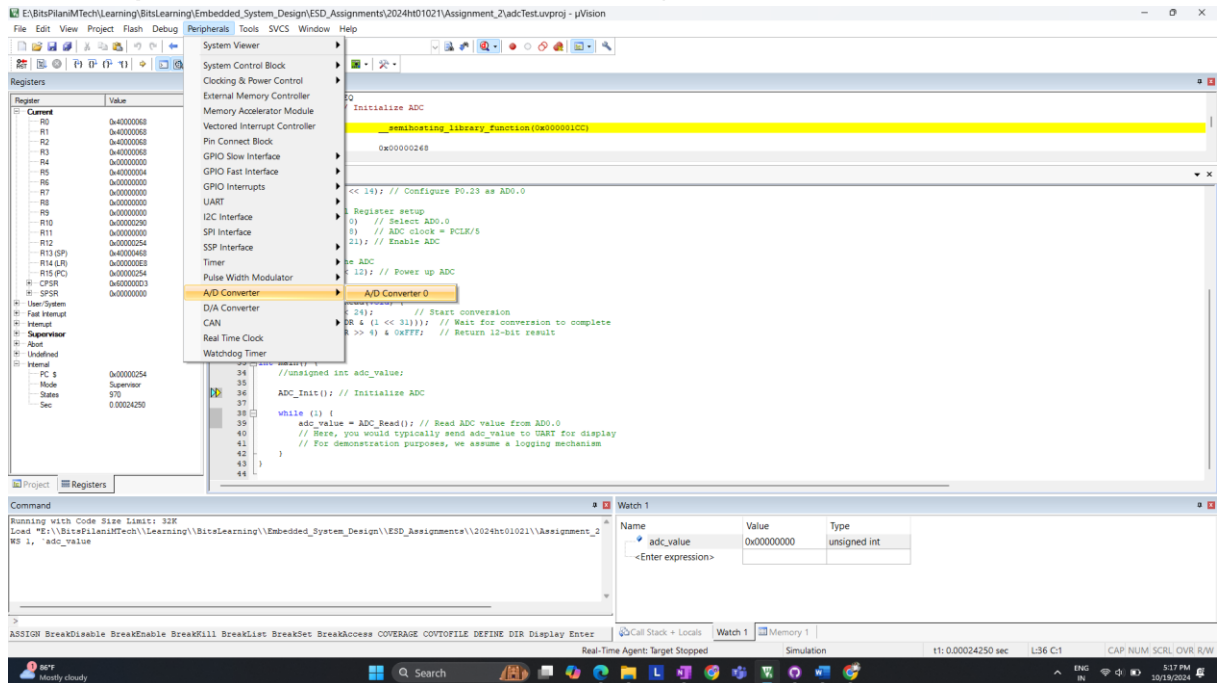
3. Building and Running the Simulation:

- **Build** the project by pressing **F7**.
- Start a **Debug Session** by clicking **Debug > Start/Stop Debug Session**.
- Now you can simulate the ADC behavior using the simulator.

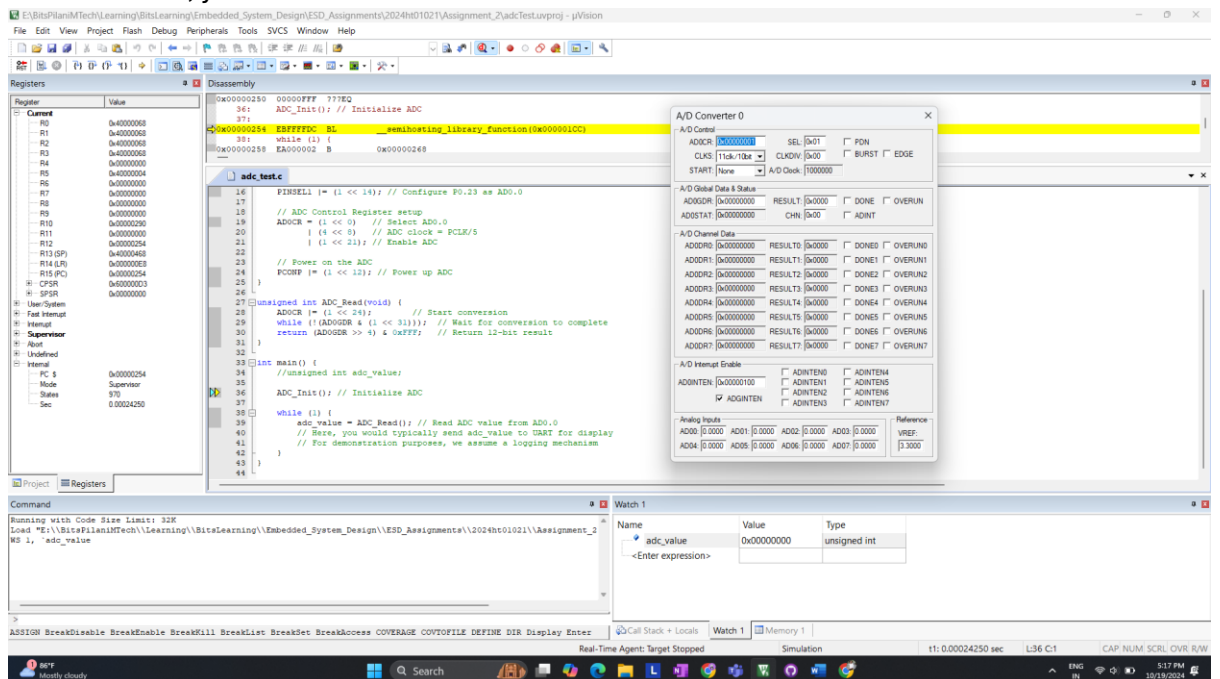


4. Accessing the ADC Peripheral in the Simulator:

- Go to **Peripherals > A/D Converter** in the menu to open the ADC window.



- In this window, you can see the result of the ADC conversion.



5. Simulating results

ADC Test with Input Voltage of 0.3300V

In this experiment, we simulated the LPC2378 ADC with a reference voltage of **3.3V** and an input voltage of **0.3300V** applied to channel AD0.0. The goal was to observe how the ADC converts the analog input voltage into a corresponding digital value.

Expected ADC Value:

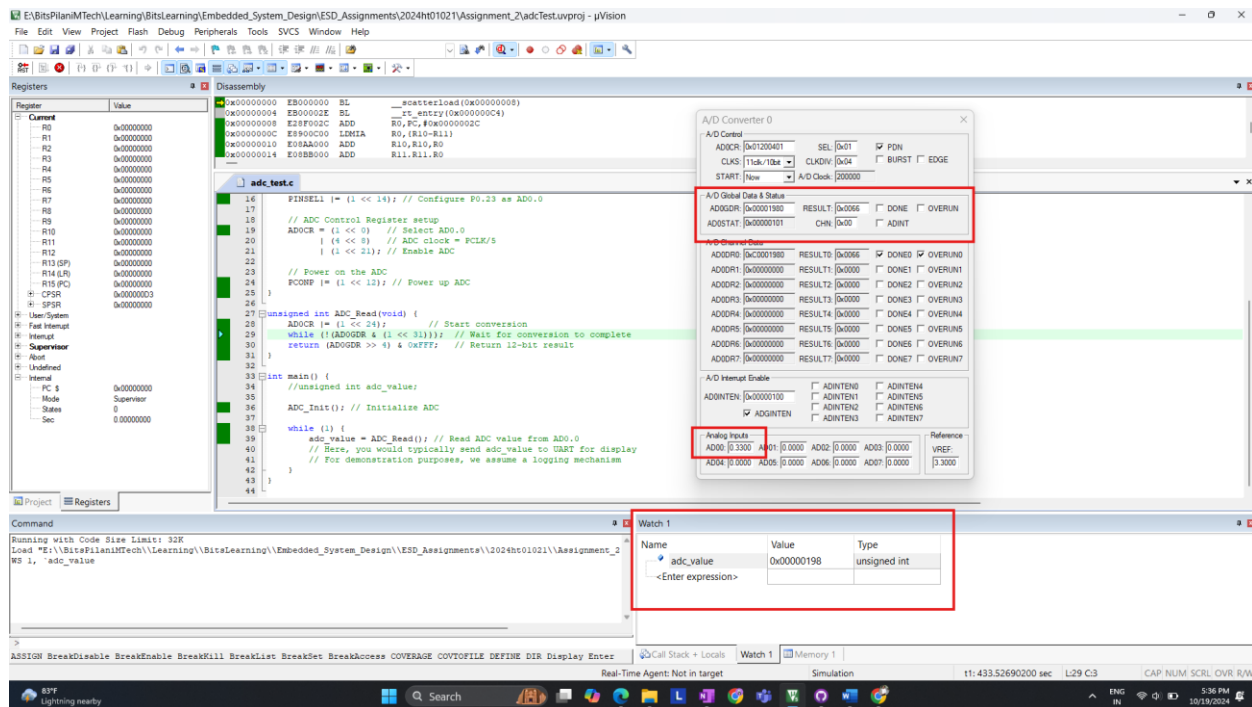
for an input voltage of 0.3300V,
the expected ADC Value can be calculated
as follows.

$$\text{ADC Value} = \frac{0.33\text{V}}{3.3\text{V}} \times 1024 = 102$$

This Calculation shows that the
ADC should o/p a digital value of
102 for an i/p vty of 0.3300v

Observed ADC Result:

In the **A/D Converter 0** window, the result of the ADC conversion is stored in the **AD0GDR** register and the **RESULT0** field. The observed value is **0x0066**, which corresponds to **102** in decimal. This matches the calculated value, confirming that the ADC is working correctly.



Smallest Detectable Voltage Change:

The smallest detectable voltage change for the LPC2378 ADC is **3.22mV**, calculated by dividing the reference voltage by 1024:

$$\Delta V_{\min} = \frac{3.3V}{1024} = 3.22mV$$

The simulation verified that the ADC accurately converted an input voltage of **0.3300V** to the corresponding digital value of **102**. The smallest detectable voltage change of **3.22mV** demonstrates the precision of the ADC.

(b): What is the maximum clock frequency needed by the ADC of LPC2378?

Answer:

The maximum clock frequency for the ADC of the LPC2378 is **4.5 MHz**.

Explanation:

- The ADC clock is derived from the **Peripheral Clock (PCLK)**, and it is divided by a programmable value set in the **CLKDIV** field of the **AD0CR** register.
- The clock used by the ADC must not exceed **4.5 MHz** to ensure proper operation.

Example:

If the **PCLK** is set to **48 MHz**, you must divide it down to meet the 4.5 MHz limit using the formula:

$$\text{ADC Clock} = \frac{\text{PCLK}}{\text{CLKDIV} + 1}$$

For **PCLK = 48 MHz**, to get the ADC clock under **4.5 MHz**, set **CLKDIV = 10**. This results in:

$$\text{ADC Clock} = \frac{48 \text{ MHz}}{10 + 1} = 4.36 \text{ MHz}$$

This value is within the allowed limit.

Summary:

The ADC clock for the LPC2378 must be below **4.5 MHz**, and this can be controlled using the **CLKDIV** setting.

(c): Give the steps to program a timer for 2-second delay generation with calculation. Assume CCLK = 48 MHz.

Steps to Program Timer for 2-Second Delay on LPC2378

We will use **Timer 0** of the LPC2378 to generate a 2-second delay. The timer clock (PCLK) is derived from the CPU clock (CCLK). The steps include setting up the timer, calculating the required values for the prescaler and match register, and writing the code to achieve the 2-second delay.

Step 1: Understand the Timer Setup

- **CCLK = 48 MHz** is the CPU clock.
- **PCLK = CCLK / 1 = 48 MHz** (assuming no additional divider).

The timer increments every **PCLK** tick, so we need to calculate how many ticks are needed to generate a 2-second delay.

Step 2: Calculate the Prescaler and Match Register

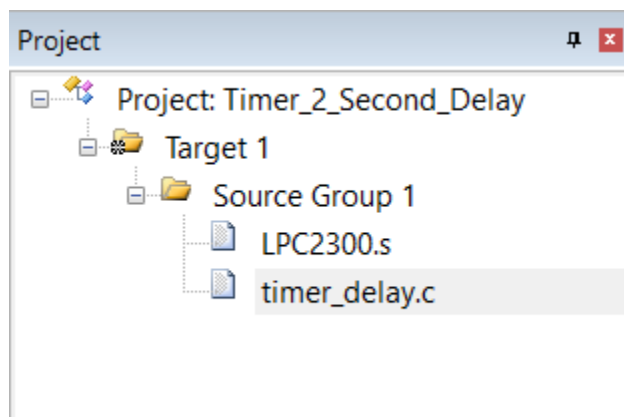
- **Timer Prescaler (PR):** The prescaler is used to divide the clock down to a manageable frequency for generating delays.
 - For simplicity, let's set the prescaler so that the timer clock is **1 MHz** (1 tick = 1 microsecond).
 - Since **PCLK = 48 MHz**, set the prescaler to divide by 48:

$$PR = 48 - 1 = 47$$

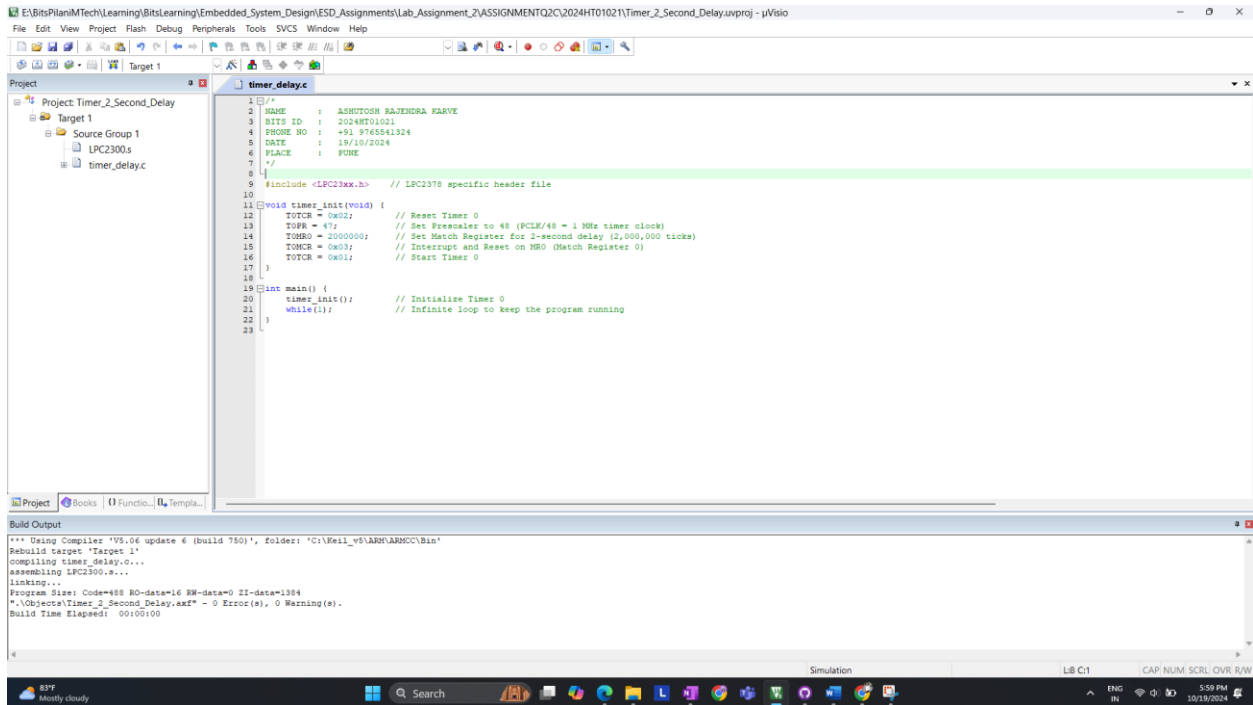
- **Match Register (MR):** We need the timer to count up to the number of ticks required for 2 seconds. Since the timer increments every 1 microsecond (1 MHz clock), the number of ticks for 2 seconds is:

$$MR = 2 \times 1,000,000 = 2,000,000 \text{ ticks}$$

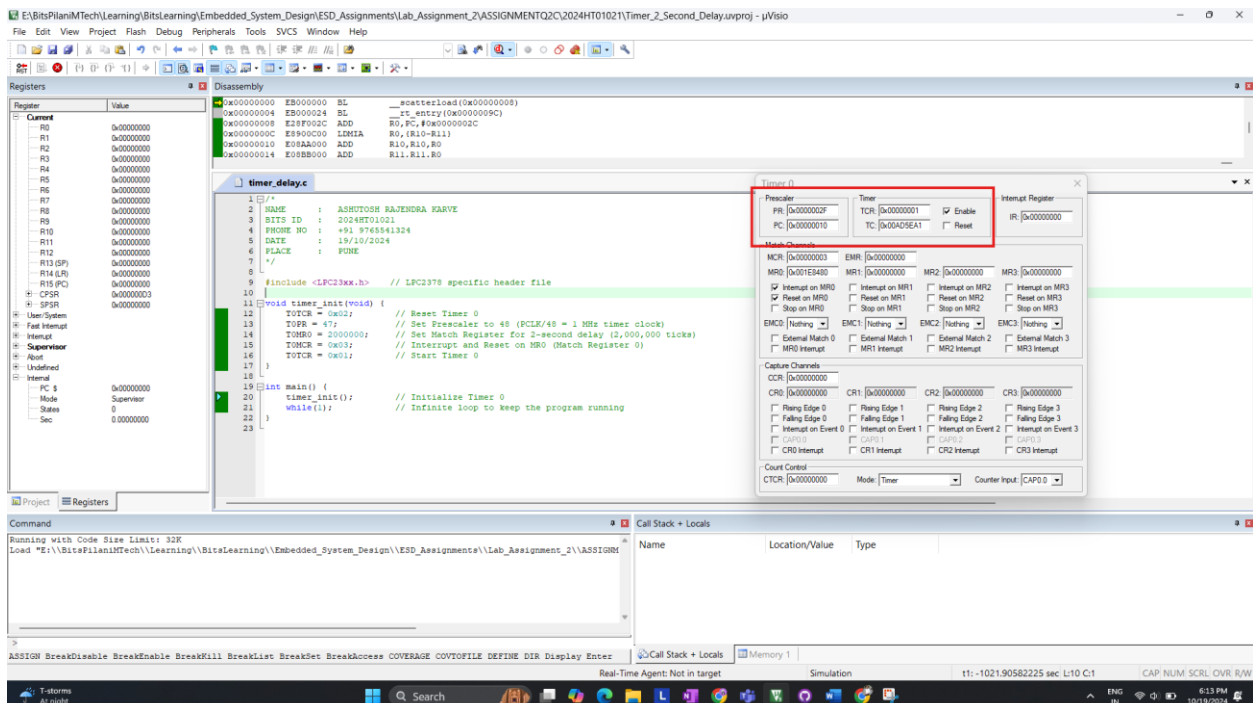
Step 3: Timer Programming Code



timer_delay.c code for timer functionality.,



DEBUG WINDOW check TC is incrementing.



Step 4: Explanation of Code

1. **T0TCR = 0x02**: Resets **Timer 0**.
2. **T0PR = 47**: Sets the prescaler to divide the **PCLK (48 MHz)** by 48, resulting in a **1 MHz** timer clock.
3. **T0MR0 = 2,000,000**: Sets the match register for a 2-second delay (2 million ticks).
4. **T0MCR = 0x03**: Configures the timer to:
 - Reset the timer when it reaches the match value.
 - Generate an interrupt (optional, not used here).
5. **T0TCR = 0x01**: Starts the timer.

Summary:

- The timer is set up to generate a 2-second delay by configuring the prescaler to **47** and the match register to **2,000,000** ticks.
- The prescaler reduces the 48 MHz PCLK to 1 MHz, and the timer counts 2,000,000 ticks for a 2-second delay.

- END -

