



# Report

## Temperature measurement using ADC.

*Your Name Here*

## Contents

<b>Abstract:</b> .....	2
<b>Introduction:</b> .....	2
Functional Block Diagram: .....	2
Pin Diagram: .....	3
Operating Modes: .....	3
UART: .....	4
USCI (UART Mode) Clock Frequency: .....	4
12-Bit ADC, Power Supply, and Input Range Conditions: .....	5
12-Bit ADC, Temperature Sensor and Built-In VMID: .....	5
<b>Interface Diagram:</b> .....	6
<b>Software Design:</b> .....	7
Pseudocode: .....	7
Flowchart: .....	8
Software Code: .....	9
<b>Working Principle:</b> .....	15
<b>Results and Discussion:</b> .....	15
<b>Conclusion:</b> .....	16
<b>References:</b> .....	16

## Abstract:

In this assignment, I am using the MSP430F5529 microcontrollers ADC internal temperature sensor, which is placed on channel 10 of ADC. Getting temperature samples after specific intervals from ADC, converting them into digital values and displaying them on any serial terminal using UART with a Baud rate of 115200, no parity and one stop bit.

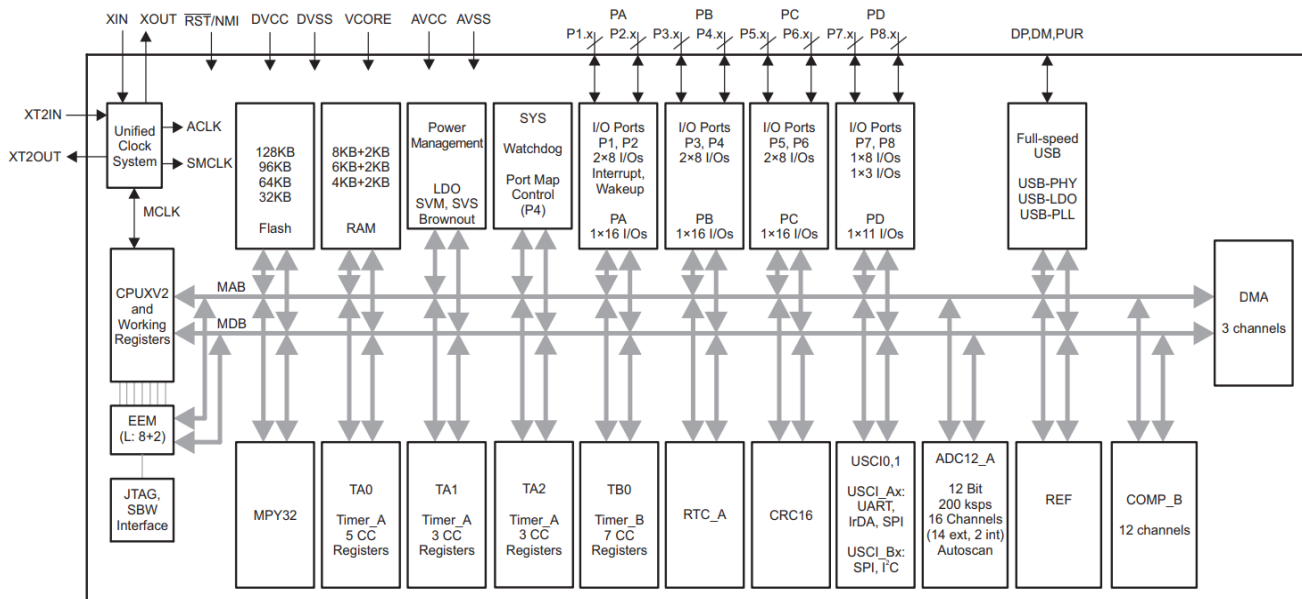
I am using an MSP430F5529 controller with ADC and UART peripherals, the temperature sensor is built inside the controller.

## Introduction:

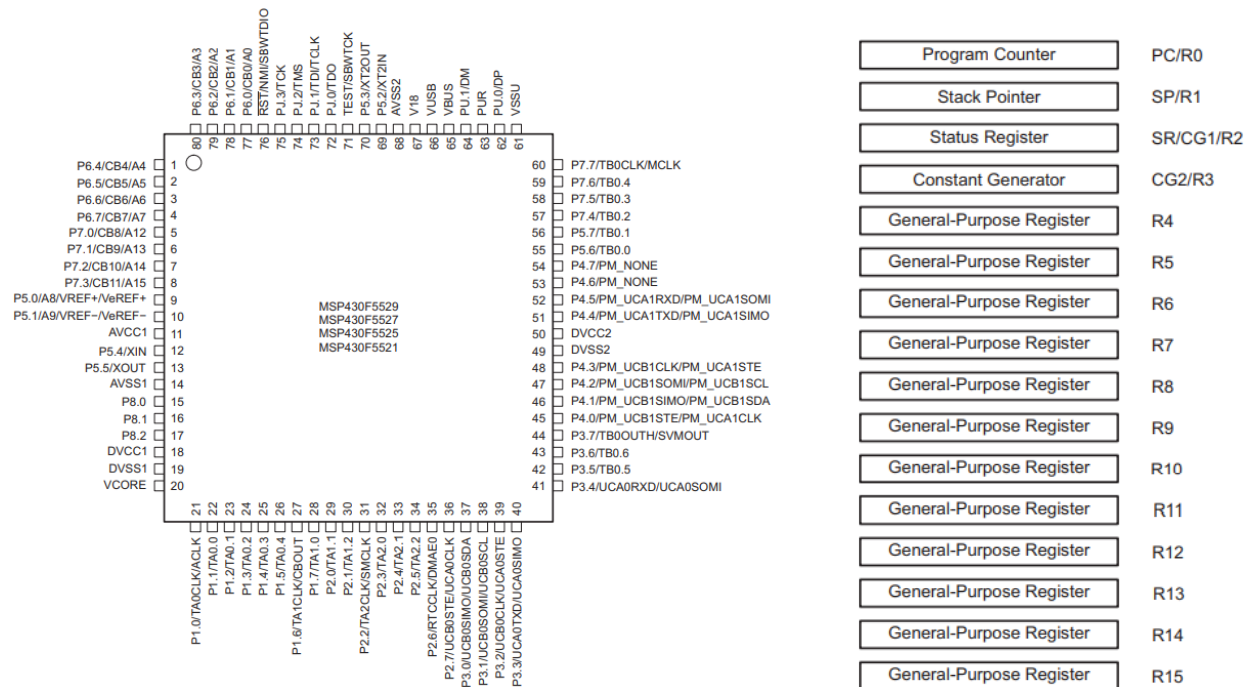
The Texas Instruments MSP430F5529 microcontroller (MCU) is important for the MSP430™ framework control and correspondence group of super low-power microcontrollers and comprises of a few elements highlighting fringe sets focused on for an assortment of uses. The design, joined with broad low-power modes, is streamlined to accomplish expanded battery duration in compact estimation applications. The microcontroller highlights a 16-cycle RISC CPU, 16-bit registers, and steady generators that add to most extreme code productivity. The carefully controlled oscillator (DCO) permits the peripherals to awaken from low-power modes to dynamic mode in 3.5  $\mu$ s (average).

The MSP430F5529 microcontrollers have incorporated USB and PHY supporting USB 2.0, four 16-cycle clocks, an elite presentation 12-bit ADC, two USCIs, 63 I/O pins a hardware multiplier, DMA, and an RTC module with alert abilities.

## Functional Block Diagram:



## Pin Diagram:



The MSP430 CPU has a 16-cycle RISC design that is profoundly straightforward to the application. All tasks, other than program-stream directions, are proceeded as register activities related to seven tending to modes for the source operand and four tending to modes for the objective operand. The CPU is coordinated with 16 registers that give decreased program execution time. The R-to-R activity execution time is one pattern of the CPU clock. Four registers, R0 to R3, are committed as program counters, stack pointers, status registers, and consistent generators. The other registers are general-purpose registers. components/elements/peripherals are associated with the CPU utilizing information, address, and control transports. The peripherals can be made do with all guidelines. The guidance set comprises the first 51 directions with three organizations and seven location modes and extra guidelines for the extended location range. Every code can work on word and byte information..

## Operating Modes:

These microcontrollers have one dynamic mode and six programming selectable low-power methods of working. An interrupt can awaken the controller from any of the low-power modes, administer the solicitation, and reestablish it to the low-power mode on getting back from the ISR program. we can easily change and configure the operating modes in our firmware:

- Controller Active mode
- Low Power mode 0
  1. CPU is off
  2. ACLK and SMCLK is active, and MCLK is off
- Low-power mode 1
  1. CPU is off

- 2. ACLK and SMCLK is active, and MCLK is off
- Low-power mode 2
  1. CPU is off
  2. MCLK, FLL loop control, and DCOCLK are off
  3. DC generator of the DCO always enabled
  4. ACLK is active
- Low-power mode 3
  1. CPU is off
  2. MCLK, FLL loop control, and DCOCLK are off
  3. DC generator of the DCO is off
  4. ACLK is active
- Low-power mode 4
  1. CPU is off
  2. ACLK is off
  3. MCLK, FLL loop control, and DCOCLK are off
  4. DC oscillator of the DCO is off
  5. Crystal oscillator is stopped/paused
- Low-power mode 4.5/5
  1. Internal regulator is turned off
  2. No data retention is there
  3. Wake up signal is generated from RST/NMI, P1, and P2

## UART:

I have used Timer A module in this assignment, so I am only providing its default configuration and settings here.

PARAMETER		TEST CONDITIONS	V <sub>CC</sub>	MIN	MAX	UNIT
f <sub>TA</sub>	Timer_A input clock frequency	Internal: SMCLK or ACLK, External: TACLK, Duty cycle = 50% ±10%	1.8 V, 3 V		25	MHz
t <sub>TA,cap</sub>	Timer_A capture timing	All capture inputs, minimum pulse duration required for capture	1.8 V, 3 V	20		ns

## USCI (UART Mode) Clock Frequency:

Since I am using UCA1 UART mode, so I am providing default configuration and settings.

PARAMETER		CONDITIONS	MIN	MAX	UNIT
f <sub>USCI</sub>	USCI input clock frequency	Internal: SMCLK or ACLK, External: UCLK, Duty cycle = 50% ±10%		f <sub>SYSTEM</sub>	MHz
f <sub>BITCLK</sub>	BITCLK clock frequency (equals baud rate in MBaud)			1	MHz

## 12-Bit ADC, Power Supply, and Input Range Conditions:

PARAMETER	TEST CONDITIONS	V <sub>CC</sub>	MIN	TYP	MAX	UNIT
AV <sub>CC</sub> Analog supply voltage	AVCC and DVCC are connected together, AVSS and DVSS are connected together, V <sub>(AVSS)</sub> = V <sub>(DVSS)</sub> = 0 V		2.2		3.6	V
V <sub>(Ax)</sub> Analog input voltage range <sup>(2)</sup>	All ADC12 analog input pins Ax		0		AV <sub>CC</sub>	V
I <sub>ADC12_A</sub> Operating supply current into AVCC terminal <sup>(3)</sup>	f <sub>ADC12CLK</sub> = 5.0 MHz <sup>(4)</sup>	2.2 V		125	155	μA
		3 V		150	220	
C <sub>i</sub> Input capacitance	Only one terminal Ax can be selected at one time	2.2 V		20	25	pF
R <sub>i</sub> Input MUX ON resistance	0 V ≤ V <sub>Ax</sub> ≤ AVCC		10	200	1900	Ω

(1) The spillage current is determined by the computerized I/O input spillage.

(2) The simple information voltage range should be inside the chose reference voltage range VR+ to VR-for substantial change results. Assuming the reference voltage is provided by an outer source or on the other hand on the off chance that the inside reference voltage is utilized and REFOUT = 1, decoupling capacitors are required.

(3) The interior reference current is excluded from the ongoing utilization boundary IADC12\_A.

(4) ADC12ON = 1, REFON = 0, SHT0 = 0, SHT1 = 0, ADC12DIV = 0

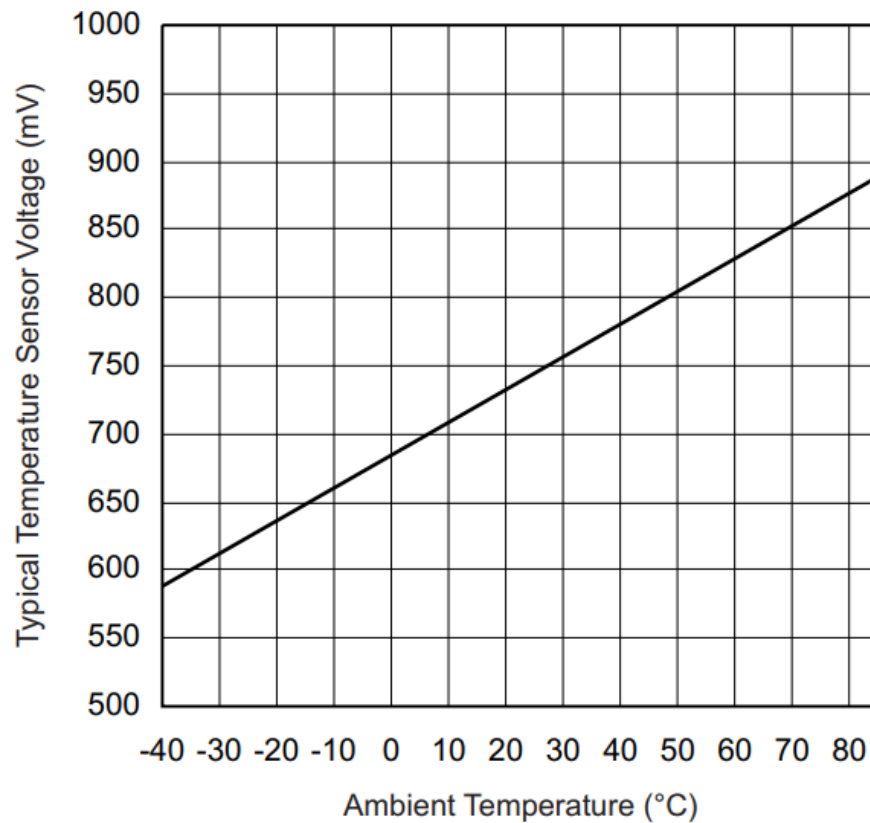
## 12-Bit ADC, Temperature Sensor and Built-In VMID:

PARAMETER <sup>(1)</sup>	TEST CONDITIONS	V <sub>CC</sub>	MIN	TYP	MAX	UNIT
V <sub>SENSOR</sub> See <sup>(2)</sup>	ADC12ON = 1, INCH = 0Ah, T <sub>A</sub> = 0°C	2.2 V		680		mV
		3 V		680		
TC <sub>SENSOR</sub>	ADC12ON = 1, INCH = 0Ah	2.2 V		2.25		mV/°C
		3 V		2.25		
t <sub>SENSOR(sample)</sub> Sample time required if channel 10 is selected <sup>(3)</sup>	ADC12ON = 1, INCH = 0Ah, Error of conversion result ≤ 1 LSB	2.2 V		100		μs
		3 V		100		
V <sub>MID</sub>	AV <sub>CC</sub> divider at channel 11, V <sub>AVCC</sub> factor		0.48	0.5	0.52	V <sub>AVCC</sub>
	AV <sub>CC</sub> divider at channel 11	2.2 V	1.06	1.1	1.14	V
		3 V	1.44	1.5	1.56	
t <sub>VMID(sample)</sub> Sample time required if channel 11 is selected <sup>(4)</sup>	ADC12ON = 1, INCH = 0Bh, Error of conversion result ≤ 1 LSB	2.2 V, 3 V		1000		ns

The temperature sensor is given by the REF module. The REF module parametric, IREF+, in regard to the ongoing utilization of the temperature sensor.

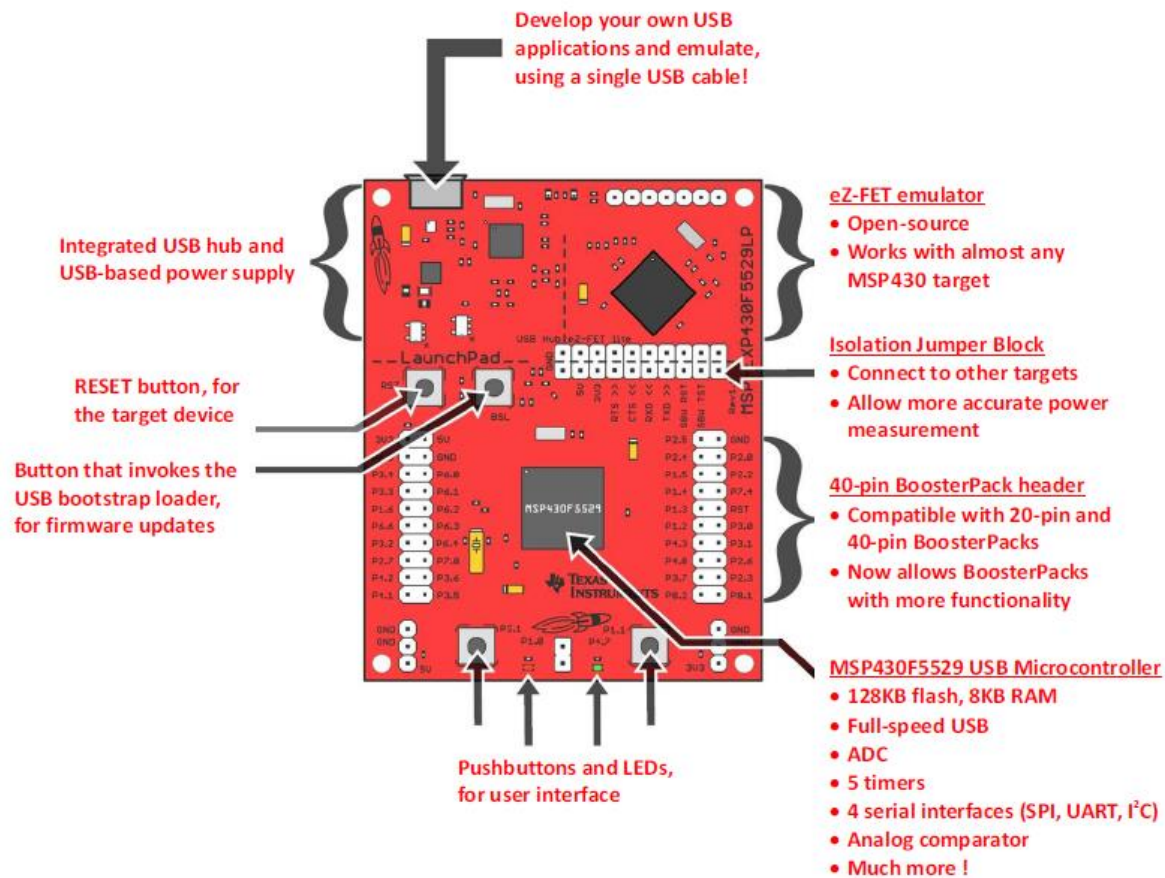
The temperature sensor offset can be critical. TI prescribes a solitary guide adjustment to limit the offset blunder of the inherent temperature sensor. The TLV structure contains adjustment values for 30°C ±3°C and 85°C ±3°C for every one of the accessible reference voltage levels. The sensor voltage can be figured as VSENSE = TCSENSOR × (Temperature) + VSENSOR, where TCSENSOR and VSENSOR can be registered from the alignment values for higher exactness.

The normal identical impedance of the sensor is  $51\text{ k}\Omega$ . The example time required remembers the sensor-for time  $t_{\text{SENSOR(on)}}$ . The on-time  $t_{\text{VMID(on)}}$  is remembered for the examining time  $t_{\text{VMID(sample)}}$ ; no extra on time is required.



## Interface Diagram:





## Software Design:

Software design consists of three components, which I will discuss one by one in detail below:

### Pseudocode:

```

Include Libraries
Initialize global variables
Declare all function definitions
Begin main:
    Stop watchdog timer
    Timer Setup
    UART Setup
    ADC Setup

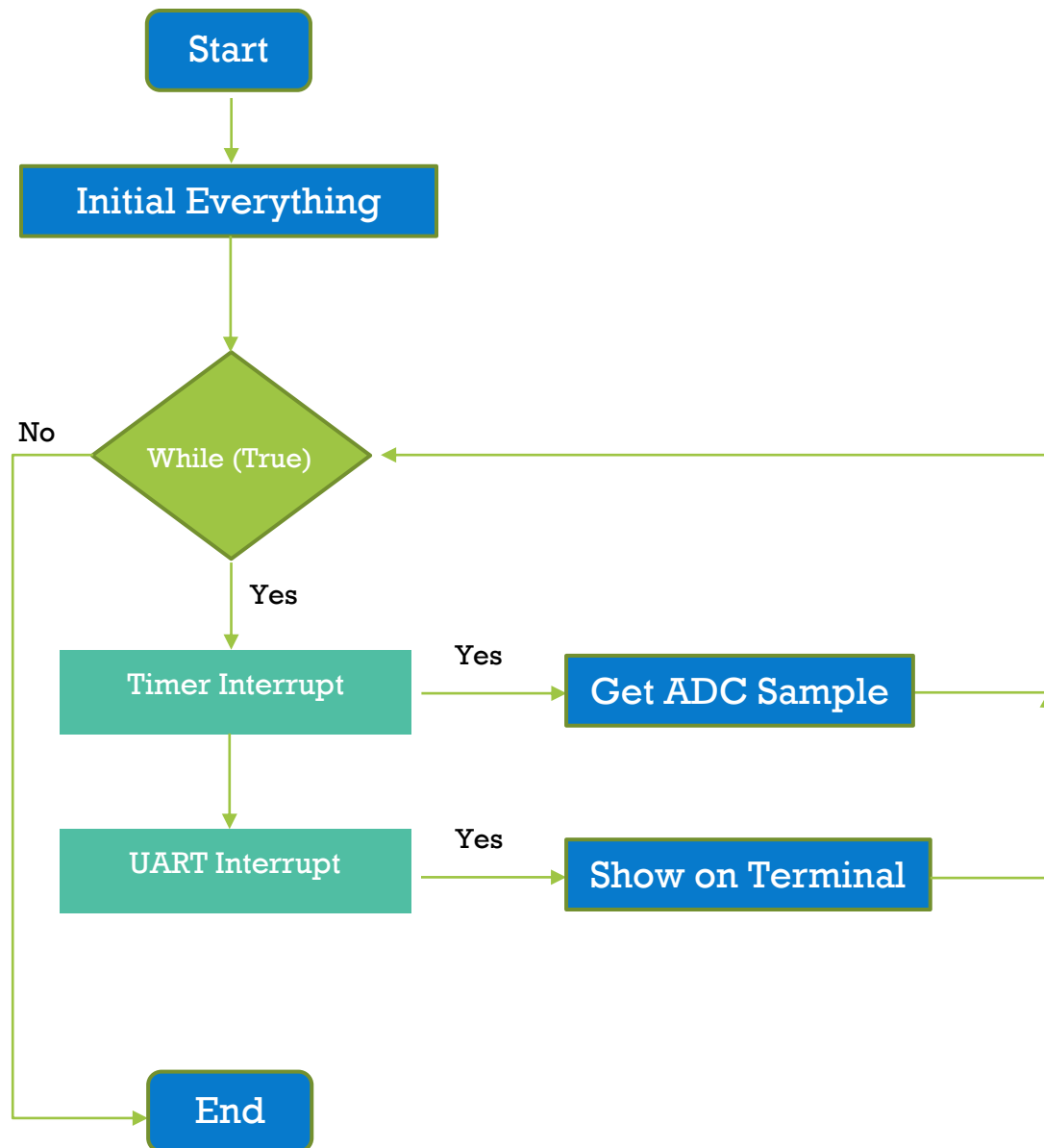
    Start:
        Read ADC Value
        Convert into Digital
        Send to PC

    Goto Start
End program

```



Flowchart:



## Software Code:

```
/*
 * INCLUDES
 * BEGIN
 */
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "driverlib.h"
/*
 * INCLUDES
 * END
 */

/*
 * GLOBALS
 * BEGIN
 */
long temp;
volatile long IntDegC;
unsigned long debug_timeout = 250;
/*
 * GLOBALS
 * END
 */

/*
 * MACROS
 * BEGIN
 */
#define COMPARE_VALUE 50000
/*
 * MACROS
 * END
 */

/*
 * Functions declerations
 * BEGIN
 */
void UART_setup(void);
void UART_Send_Character(char c);
void UART_Send_String(char* string);
char UART_Get_Character();
void UART_Get_Word(char* buffer, int limit);
```

```

/*
 * Functions declerations
 * END
 */

void main(void)
{
    //stop watch dog
    WDT_A_hold(WDT_A_BASE);

    /*
     * GPIO Setup
     */

    /*
     * Timer Setup
     */
    Timer_A_initContinuousModeParam initContParam = {0};
    initContParam.clockSource = TIMER_A_CLOCKSOURCE_SMCLK;
    initContParam.clockSourceDivider = TIMER_A_CLOCKSOURCE_DIVIDER_1;
    initContParam.timerInterruptEnable_TAIE = TIMER_A_TAIE_INTERRUPT_DISABLE;
    initContParam.timerClear = TIMER_A_DO_CLEAR;
    initContParam.startTimer = false;
    Timer_A_initContinuousMode(TIMER_A1_BASE, &initContParam);

    //Initiaze compare mode
    Timer_A_clearCaptureCompareInterrupt(TIMER_A1_BASE,
                                         TIMER_A_CAPTURECOMPARE_REGISTER_0
                                         );

    Timer_A_initCompareModeParam initCompParam = {0};
    initCompParam.compareRegister = TIMER_A_CAPTURECOMPARE_REGISTER_0;
    initCompParam.compareInterruptEnable =
        TIMER_A_CAPTURECOMPARE_INTERRUPT_ENABLE;
    initCompParam.compareOutputMode = TIMER_A_OUTPUTMODE_OUTBITVALUE;
    initCompParam.compareValue = COMPARE_VALUE;
    Timer_A_initCompareMode(TIMER_A1_BASE, &initCompParam);

    Timer_A_startCounter(TIMER_A1_BASE,
                        TIMER_A_CONTINUOUS_MODE
                        );
    // UART setup
    UART_setup();

    /*
     * ADC Setup
     */

```

```

ADC12_A_init(ADC12_A_BASE,
             ADC12_A_SAMPLEHOLDSOURCE_SC,
             ADC12_A_CLOCKSOURCE_ADC12OSC,
             ADC12_A_CLOCKDIVIDER_1);

ADC12_A_enable(ADC12_A_BASE);
ADC12_A_enable(ADC12_A_BASE);
ADC12_A_setupSamplingTimer(ADC12_A_BASE,
                           ADC12_A_CYCLEHOLD_768_CYCLES,
                           ADC12_A_CYCLEHOLD_4_CYCLES,
                           ADC12_A_MULTIPLESAMPLESDISABLE);
ADC12_A_configureMemoryParam param = {0};
param.memoryBufferControlIndex = ADC12_A_MEMORY_0;
param.inputSourceSelect = ADC12_A_INPUT_TEMPSENSOR;
param.positiveRefVoltageSourceSelect = ADC12_A_VREFPOS_INT;
param.negativeRefVoltageSourceSelect = ADC12_A_VREFNEG_AVSS;
param.endOfSequence = ADC12_A_NOTENDOFSEQUENCE;
ADC12_A_configureMemory(ADC12_A_BASE,&param);

ADC12_A_clearInterrupt(ADC12_A_BASE,
                      ADC12IFG0);
ADC12_A_enableInterrupt(ADC12_A_BASE,
                      ADC12IE0);

/*
 * ADC reference Setup
 */
while(REF_ACTIVE == Ref_isRefGenBusy(REF_BASE))
{
    ;
}
Ref_setReferenceVoltage(REF_BASE,
                      REF_VREF1_5V);
Ref_enableReferenceVoltage(REF_BASE);

__delay_cycles(75);
__bis_SR_register(GIE);

// trigger ADC conversion
ADC12_A_startConversion(ADC12_A_BASE,
                       ADC12_A_MEMORY_0,
                       ADC12_A_SEQOFCHANNELS);

while(1)
{
}
}

```

```

/*
 * ADC ISR
 */
#pragma vector=ADC12_VECTOR
__interrupt void ADC12ISR(void)
{
    switch(__even_in_range(ADC12IV,34))
    {
        case 0: break;    //Vector 0: No interrupt
        case 2: break;    //Vector 2: ADC overflow
        case 4: break;    //Vector 4: ADC timing overflow
        case 6:           //Vector 6: ADC12IFG0
            //Move results, IFG is cleared
            temp = ADC12_A_getResults(ADC12_A_BASE,
                                      ADC12_A_MEMORY_0);

            IntDegC = (((temp - 1855) * 667) / 4096);

            // Trigger conversion for next time
            ADC12_A_startConversion(ADC12_A_BASE,
                                     ADC12_A_MEMORY_0,
                                     ADC12_A_SEQOFCHANNELS);

            //Exit active CPU
            __bic_SR_register_on_exit(LPM4_bits);
            break;
        case 8: break;    //Vector 8: ADC12IFG1
        case 10: break;   //Vector 10: ADC12IFG2
        case 12: break;   //Vector 12: ADC12IFG3
        case 14: break;   //Vector 14: ADC12IFG4
        case 16: break;   //Vector 16: ADC12IFG5
        case 18: break;   //Vector 18: ADC12IFG6
        case 20: break;   //Vector 20: ADC12IFG7
        case 22: break;   //Vector 22: ADC12IFG8
        case 24: break;   //Vector 24: ADC12IFG9
        case 26: break;   //Vector 26: ADC12IFG10
        case 28: break;   //Vector 28: ADC12IFG11
        case 30: break;   //Vector 30: ADC12IFG12
        case 32: break;   //Vector 32: ADC12IFG13
        case 34: break;   //Vector 34: ADC12IFG14
        default: break;
    }
}

/*
 * TIMER ISR
 */
#pragma vector=TIMER1_A0_VECTOR

```

```

__interrupt void TIMER1_A0_ISR(void)
{
    char console_buffer[100] = {};
    uint16_t compVal = Timer_A_getCaptureCompareCount(TIMER_A1_BASE,
        TIMER_A_CAPTURECOMPARE_REGISTER_0)
        + COMPARE_VALUE;

    /*
    * Display temperature messages
    */
    if (!debug_timeout--) {
        sprintf(console_buffer, "TEMP: %.2f\r\n", (float)IntDegC);
        UART_Send_String(console_buffer);
        debug_timeout = 250;
    }

    //Add Offset to CCR0
    Timer_A_setCompareValue(TIMER_A1_BASE,
        TIMER_A_CAPTURECOMPARE_REGISTER_0,
        compVal
    );
}

/*
* UART initialization function
*/
void UART_setup(void)
{
    P4SEL |= BIT4 + BIT5; // Set USCI_A1 RXD/TXD to receive/transmit data
    UCA1CTL1 |= UCSWRST; // Set software reset during initialization
    UCA1CTL0 = 0; // USCI_A1 control register
    UCA1CTL1 |= UCSSEL_2; // Clock source SMCLK
    UCA1BR0 = 0x09; // 1048576 Hz / 115200 lower byte
    UCA1BR1 = 0x00; // upper byte
    UCA1MCTL = 0x02; // Modulation (UCBRS0=0x01, UCOS16=0)
    UCA1CTL1 &= ~UCSWRST; // Clear software reset to initialize USCI state machine
    //IE2 |= UCA1RXIE; // Enable USCI_A1 RX interrupt
    //UCA1IE |= UCRXIE; // Enable USCI_A1 RX interrupt
}

/*
* UART sending a single character
*/
void UART_Send_Character(char c)
{
    while (!(UCA1IFG & UCTXIFG)); // Wait for previous character to transmit
    UCA1TXBUF = c; // Put character into tx buffer
}

```

```

}

/*
 * UART sending a string
 */
void UART_Send_String(char* string)
{
    unsigned int i;
    for(i = 0; i < strlen(string); i++)
    {
        UART_Send_Character(string[i]);
    }
}

/*
 * UART getting character from terminal
 */
char UART_Get_Character()
{
    while(!(UCA1IFG & UCRXIFG)); // Wait for a new character
    return UCA1RXBUF;
}

/*
 * UART getting string from terminal
 */
void UART_Get_Word(char* buffer, int limit)
{
    char Current_Character = UART_Get_Character();
    UART_Send_Character(Current_Character);
    unsigned int i = 0;

    do
    {
        buffer[i] = Current_Character;
        Current_Character = UART_Get_Character();
        UART_Send_Character(Current_Character);
        if(i < limit-2)
        {
            i++;
        }
        else
        {
            break;
        }
    }
}

```

```
while(Current_Character != '\r');  
UART_Send_Character('\r');  
UART_Send_Character('\n');  
buffer[i] = 0;  
}
```

## Working Principle:

The working principle of the project is very simple. First, we initialize the UART, ADC and Timer control registers.

Timer A1 is used with the clock source of SMCLK, the divider is 1 in continuous mode and a capture compares the value of 50000 cycles.

UART USCI\_A1 RDX/TXD to receive/transmit data on ports 4 pins 4 and 5 is used with clock source of SMCLK and Baud rate of 115200 bits per second. UART interrupt is disabled and operated based on the timer module.

ADC12\_A\_INPUT\_TEMPSENSOR is selected as a source for the ADC channel to get the data from internal temperature which is converted into digital value from the given formula.

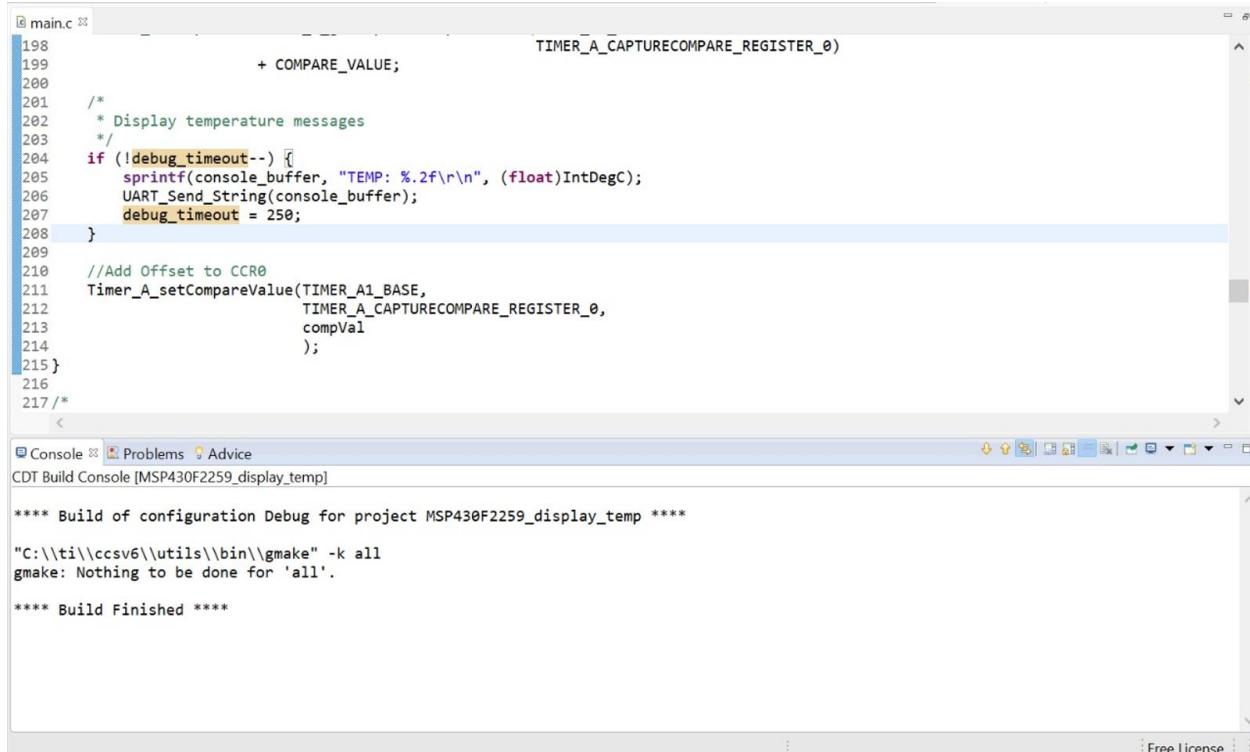
$$\text{temp} = \text{ADC12\_A\_getResults}(\text{ADC12\_A\_BASE}, \text{ADC12\_A\_MEMORY\_0});$$
$$\text{IntDegC} = (((\text{temp} - 1855) * 667) / 4096);$$

## Results and Discussion:



When I connect the hardware with my laptop and build and debug the program on code composer studio version 6 the values instantly appear on the terminal when I chose the correct baud rate of 115200 and no parity bits with a single stop bit.





The screenshot shows an IDE with two windows. The top window, titled 'main.c', contains C code for an embedded system. The code includes a loop that checks a 'debug\_timeout' variable, prints the temperature using 'printf' and 'UART\_Send\_String', and sets a compare value for a timer. The bottom window, titled 'Console', shows the output of a 'make' command, indicating a successful build for the project 'MSP430F2259\_display\_temp'.

```
198                                     TIMER_A_CAPTURECOMPARE_REGISTER_0)
199                                     + COMPARE_VALUE;
200
201 /*
202  * Display temperature messages
203  */
204 if (!debug_timeout--) {
205     sprintf(console_buffer, "TEMP: %.2f\r\n", (float)IntDegC);
206     UART_Send_String(console_buffer);
207     debug_timeout = 250;
208 }
209
210 //Add Offset to CCR0
211 Timer_A_setCompareValue(TIMER_A1_BASE,
212                         TIMER_A_CAPTURECOMPARE_REGISTER_0,
213                         compVal
214                         );
215 }
216
217 /*
```

Console Problems Advice  
CDT Build Console [MSP430F2259\_display\_temp]

```
**** Build of configuration Debug for project MSP430F2259_display_temp ****
"C:\ti\ccsv6\utils\bin\gmake" -k all
gmake: Nothing to be done for 'all'.
**** Build Finished ****
```

The code is successfully built and run with zero errors, working fine.

## Conclusion:

In this assignment I have learned a lot of things ranging from UART, to ADC using analogue sensors, handling their values converting into a digital value which makes sense to human beings and displayed indefinitely on serial terminal using UART module.

I feel confident and motivated to make more diverse projects and solve some real-world problems. It instils in me a desire to work more and build on embedded systems-based applications to explore more about the controller and its peripherals.

## References

- [1] "Embedded Lab," [Online]. Available:  
<https://www.google.com/url?sa=i&url=https%3A%2F%2Fembedded-lab.com%2Fblog%2Ftinkering-ti-msp430f5529%2F&psig=AOvVaw0li9UPUNAC4iMjNOMdkeAF&ust=1652195673105000&source=images&cd=vfe&ved=0CAwQjRxqFwoTCMjSzuva0vcCFQAAAAAdAAAAABAD>.
- [2] "texas instrument," [Online]. Available:  
[https://www.ti.com/lit/ds/symlink/msp430f5529.pdf?ts=1652083427642&ref\\_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FMSP430F5529](https://www.ti.com/lit/ds/symlink/msp430f5529.pdf?ts=1652083427642&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FMSP430F5529).