# The full tutorial video link:

<https://drive.google.com/file/d/19TDm8Dt0lAyq8qOB-2iKjvqkOJg2104O/view?usp=sharing>

# Creating Thing Speak Channel:

<https://thingspeak.com/channels>



* Create your account
* Create a new channel
* Create new fields for each sensor
* Find your channel id and write API Keys

Graphical user interface, text, application, email

Description automatically generated

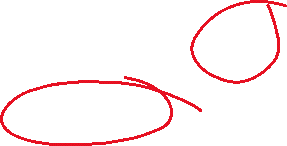


Table

Description automatically generated

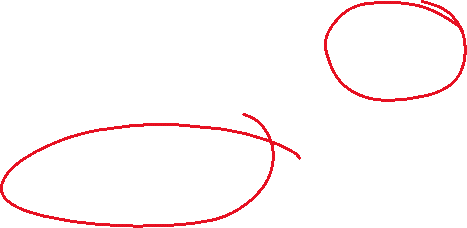
Graphical user interface, text, application, chat or text message

Description automatically generated



Graphical user interface, text, application, chat or text message

Description automatically generated



# IoT application using C#

Graphical user interface, text, application, email

Description automatically generated

* After creating project you will get a screen like the following. Go to the Toolbox

Graphical user interface, application, Word

Description automatically generated



* Create 3buttons, 2 labels, 1 textbox, 1 timer and 1 serial port. Go to the properties by putting cursor on each of the tool. Change the design name and text name like the following figure

Graphical user interface, application

Description automatically generated



Graphical user interface, application

Description automatically generated



* Now double click on each of the tool except the Serial Port in Form1.cs[Design] file and it will create a function for you.
* Fill every function according to the provided code. Make sure you have mention right COM port, Baud rate, channel name and API keys.
* Create a function (SerialPort1\_DataReceived) for Serial Port and connect with serial port.

Serial Port Setup:



Graphical user interface, application, table

Description automatically generated



Table

Description automatically generated



Graphical user interface, application

Description automatically generated



Timer Setup:

Table

Description automatically generated



Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated



* Now run this. Before you run the C# application, make sure using CCS, you are able to send data to the terminal. When you see the data in the terminal, close/disconnect the terminal which will close the serial port and now the serial port can be used for C# application.

# Code:

## **CCS**

**#include** <msp430.h>

**#define** CALADC\_15V\_30C \*((**unsigned** **int** \*)0x1A1A) // Temperature Sensor Calibration-30 C //6682 // See device datasheet for TLV table memory mapping //6684

**#define** CALADC\_15V\_85C \*((**unsigned** **int** \*)0x1A1C) // Temperature Sensor Calibration-High Temperature (85 for Industrial, 105 for Extended)

**volatile** **long** temp1;

**volatile** **float** IntDegF1;

**volatile** **float** IntDegC1;

**volatile** **long** temp2;

**volatile** **float** IntDegF2;

**volatile** **float** IntDegC2;

**char** result[100];

**int** count;

**void** **uart\_init**(**void**);

**void** **ConfigClocks**(**void**);

**void** **strreverse**(**char**\* begin, **char**\* end);

**void** **itoa**(**int** value, **char**\* str, **int** base);

**void** **Software\_Trim**();

**void** **port\_init**();

**void** **ConfigureAdc\_temp1**();

**void** **ConfigureAdc\_temp2**();

**void** **initialize\_Adc**();

**void** **main**(**void**)

{

WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer

PM5CTL0 &= ~LOCKLPM5;

P1DIR = BIT6; // P1.6 outputs

P1OUT = 0; // LEDs off

**int** m=0;

ConfigClocks();

port\_init();

uart\_init();

//spi\_init();

//lcd\_init();

\_delay\_cycles(5); // Wait for ADC Ref to settle

**while**(1){

//Transmit a check byte B

**if**(m == 0){

\_delay\_cycles(20000);

**int** acount =0;

result[acount]='B';

**while**((UCA1IFG & UCTXIFG)==0);

UCA1TXBUF = result[acount] ; //Transmit the received data.

m++;

**if**(m==1){

//initialize\_Adc();

PMMCTL0\_H = PMMPW\_H; // Unlock the PMM registers read 2.2.8 & 2.2.9 form the manual

PMMCTL2 |= INTREFEN | TSENSOREN | REFVSEL\_0; // Enable internal 1.5V reference and temperature sensor

ConfigureAdc\_temp1();

ADCCTL0 |= ADCENC + ADCSC +ADCMSC; // Converter Enable, Sampling/conversion start

**while**((ADCCTL0 & ADCIFG) == 0); // check the Flag, while its low just wait

\_delay\_cycles(200000);

temp1 = ADCMEM0; // read the converted data into a variable

ADCCTL0 &= ~ADCIFG;

IntDegC1 = (temp1-CALADC\_15V\_30C)\*(85-30)/(CALADC\_15V\_85C-CALADC\_15V\_30C)+30;

itoa(IntDegC1,result,10);

acount =0;

**while**(result[acount]!='\0')

{

**while**((UCA1IFG & UCTXIFG)==0); //Wait Unitl the UART transmitter is ready //UCTXIFG

UCA1TXBUF = result[acount++] ; //Transmit the received data.

}

m=0;

//m=2;

}

/\* if(m==2){

PMMCTL0\_H = PMMPW\_H; // Unlock the PMM registers read 2.2.8 & 2.2.9 form the manual

PMMCTL2 |= INTREFEN | TSENSOREN | REFVSEL\_0; // Enable internal 1.5V reference and temperature sensor

//initialize\_Adc();

ConfigureAdc\_temp1();

ADCCTL0 |= ADCENC + ADCSC +ADCMSC; // Converter Enable, Sampling/conversion start

while((ADCCTL0 & ADCIFG) == 0); // check the Flag, while its low just wait

\_delay\_cycles(20000000);

temp2 = ADCMEM0; // read the converted data into a variable

ADCCTL0 &= ~ADCIFG;

IntDegC2 = (temp2-CALADC\_15V\_30C)\*(85-30)/(CALADC\_15V\_85C-CALADC\_15V\_30C)+30;

// Temperature in Fahrenheit

// Tf = (9/5)\*Tc | 32

IntDegF2 = 9\*IntDegC2/5+32;}

itoa(IntDegC2,result,10);

acount =0;

while(result[acount]!='\0')

{

while((UCA1IFG & UCTXIFG)==0); //Wait Unitl the UART transmitter is ready //UCTXIFG

UCA1TXBUF = result[acount++] ; //Transmit the received data.

}

m=0;

\*/

}

}

}

**void** **uart\_init**(**void**){

UCA1CTLW0 |= UCSWRST;

UCA1CTLW0 |= UCSSEL\_\_SMCLK;

UCA1BRW = 8; // 115200

UCA1MCTLW = 0xD600;

UCA1CTLW0 &= ~UCSWRST; // Initialize eUSCI

UCA1IE |= UCRXIE; // Enable USCI\_A0 RX interrupt

}

**void** **ConfigClocks**(**void**)

{

CSCTL3 = SELREF\_\_REFOCLK; // Set REFO as FLL reference source

CSCTL1 = DCOFTRIMEN\_1 | DCOFTRIM0 | DCOFTRIM1 | DCORSEL\_0;// DCOFTRIM=3, DCO Range = 1MHz

CSCTL2 = FLLD\_0 + 30; // DCODIV = 1MHz

**\_\_delay\_cycles**(3);

**\_\_bic\_SR\_register**(SCG0); // Enable FLL

Software\_Trim(); // Software Trim to get the best DCOFTRIM value

CSCTL4 = SELMS\_\_DCOCLKDIV | SELA\_\_REFOCLK; // set default REFO(~32768Hz) as ACLK source, ACLK = 32768Hz

// default DCODIV as MCLK and SMCLK source

}

**void** **strreverse**(**char**\* begin, **char**\* end) // Function to reverse the order of the ASCII char array elements

{

**char** aux;

**while**(end>begin)

aux=\*end, \*end--=\*begin, \*begin++=aux;

}

**void** **itoa**(**int** value, **char**\* str, **int** base) { //Function to convert the signed int to an ASCII char array

**static** **char** num[] = "0123456789abcdefghijklmnopqrstuvwxyz";

**char**\* wstr=str;

**int** sign;

// Validate that base is between 2 and 35 (inlcusive)

**if** (base<2 || base>35){

\*wstr='\0';

**return**;

}

// Get magnitude and th value

sign=value;

**if** (sign < 0)

value = -value;

**do** // Perform interger-to-string conversion.

\*wstr++ = num[value%base]; //create the next number in converse by taking the modolus

**while**(value/=base); // stop when you get a 0 for the quotient

**if**(sign<0) //attch sign character, if needed

\*wstr++='-';

\*wstr='\0'; //Attach a null character at end of char array. The string is in revers order at this point

strreverse(str,wstr-1); // Reverse string

}

**void** **port\_init**(){

P1DIR |= BIT0;

P1OUT |= BIT0;

P1SEL0 |= BIT6 | BIT7; // set 2-UART pin as second function

P4SEL0 |= BIT2 | BIT3; // set 2-UART pin as second function

P4SEL1 &= ~BIT2; // set 2-UART pin as second function

P4SEL1 &= ~ BIT3; // set 2-UART pin as second function

}

**void** **Software\_Trim**()

{

**unsigned** **int** oldDcoTap = 0xffff;

**unsigned** **int** newDcoTap = 0xffff;

**unsigned** **int** newDcoDelta = 0xffff;

**unsigned** **int** bestDcoDelta = 0xffff;

**unsigned** **int** csCtl0Copy = 0;

**unsigned** **int** csCtl1Copy = 0;

**unsigned** **int** csCtl0Read = 0;

**unsigned** **int** csCtl1Read = 0;

**unsigned** **int** dcoFreqTrim = 3;

**unsigned** **char** endLoop = 0;

**do**

{

CSCTL0 = 0x100; // DCO Tap = 256

**do**

{

CSCTL7 &= ~DCOFFG; // Clear DCO fault flag

}**while** (CSCTL7 & DCOFFG); // Test DCO fault flag

//\_\_delay\_cycles((unsigned int)3000 \* MCLK\_FREQ\_MHZ);// Wait FLL lock status (FLLUNLOCK) to be stable

// Suggest to wait 24 cycles of divided FLL reference clock

**while**((CSCTL7 & (FLLUNLOCK0 | FLLUNLOCK1)) && ((CSCTL7 & DCOFFG) == 0));

csCtl0Read = CSCTL0; // Read CSCTL0

csCtl1Read = CSCTL1; // Read CSCTL1

oldDcoTap = newDcoTap; // Record DCOTAP value of last time

newDcoTap = csCtl0Read & 0x01ff; // Get DCOTAP value of this time

dcoFreqTrim = (csCtl1Read & 0x0070)>>4;// Get DCOFTRIM value

**if**(newDcoTap < 256) // DCOTAP < 256

{

newDcoDelta = 256 - newDcoTap; // Delta value between DCPTAP and 256

**if**((oldDcoTap != 0xffff) && (oldDcoTap >= 256)) // DCOTAP cross 256

endLoop = 1; // Stop while loop

**else**

{

dcoFreqTrim--;

CSCTL1 = (csCtl1Read & (~DCOFTRIM)) | (dcoFreqTrim<<4);

}

}

**else** // DCOTAP >= 256

{

newDcoDelta = newDcoTap - 256; // Delta value between DCPTAP and 256

**if**(oldDcoTap < 256) // DCOTAP cross 256

endLoop = 1; // Stop while loop

**else**

{

dcoFreqTrim++;

CSCTL1 = (csCtl1Read & (~DCOFTRIM)) | (dcoFreqTrim<<4);

}

}

**if**(newDcoDelta < bestDcoDelta) // Record DCOTAP closest to 256

{

csCtl0Copy = csCtl0Read;

csCtl1Copy = csCtl1Read;

bestDcoDelta = newDcoDelta;

}

}**while**(endLoop == 0); // Poll until endLoop == 1

CSCTL0 = csCtl0Copy; // Reload locked DCOTAP

CSCTL1 = csCtl1Copy; // Reload locked DCOFTRIM

**while**(CSCTL7 & (FLLUNLOCK0 | FLLUNLOCK1)); // Poll until FLL is locked

}

// Configure ADC Temperature

**void** **ConfigureAdc\_temp1**(){

ADCCTL0 |= ADCSHT\_8 | ADCON; // ADC ON,temperature sample period>30us

ADCCTL1 |= ADCSHP; // s/w trig, single ch/conv, MODOSC

ADCCTL2 &= ~ADCRES; // clear ADCRES in ADCCTL

ADCCTL2 |= ADCRES\_2; // 12-bit conversion results

ADCMCTL0 |= ADCSREF\_1 | ADCINCH\_12; // ADC input ch A12 => temp sense

ADCIE |=ADCIE0;

}

// Configure ADC Temperature

**void** **ConfigureAdc\_temp2**(){

ADCCTL0 &= ~ADCENC;

ADCCTL0 |= ADCSHT\_8 | ADCON; // ADC ON,temperature sample period>30us

ADCCTL1 |= ADCSHP|ADCCONSEQ\_1; // s/w trig, single ch/conv, MODOSC

ADCCTL2 &= ~ADCRES; // clear ADCRES in ADCCTL

ADCCTL2 |= ADCRES\_2; // 12-bit conversion results

ADCMCTL0 |= ADCSREF\_1 | ADCINCH\_3; // ADC input ch A12 => temp sense

ADCIE |=ADCIE0;

}

**void** **initialize\_Adc**(){

ADCCTL0 &= ~ADCIFG;//CLEAR FLAG

ADCMEM0=0x00000000;

//ADCAE0=0x00;

ADCCTL0=0x0000;

ADCCTL1=0x0000;

}

## C#:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Net;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Project\_F22\_3

{

public partial class Form1 : Form

{

int startflag = 0;

int flag\_sensor;

string RxString;

string temp, voice = "0";

char charb = 'B';

public Form1()

{

InitializeComponent();

}

private void Serial\_start\_Click(object sender, EventArgs e)

{

serialPort1.PortName = "COM3";

serialPort1.BaudRate = 115200;

serialPort1.Open();

if (serialPort1.IsOpen)

{

textBox1.ReadOnly = false;

}

}

private void Serial\_stop\_Click(object sender, EventArgs e)

{

serialPort1.Close();

textBox1.ReadOnly = true;

}

private void Read\_In\_TS\_Click(object sender, EventArgs e)

{

WebClient client = new WebClient();

label1.Text = client.DownloadString("http://api.thingspeak.com/channels/1563508/field/field1/last.text");//use your channel id

}

private void Form1\_Load(object sender, EventArgs e)

{

if (serialPort1.IsOpen)

serialPort1.Close();

serialPort1.PortName = "COM3";

serialPort1.BaudRate = 115200;

}

private void label2\_Click(object sender, EventArgs e) //current data

{

textBox1.AppendText(RxString);

}

private void textBox1\_TextChanged(object sender, EventArgs e)

{

}

private void timer1\_Tick(object sender, EventArgs e)

{

if (!string.Equals(textBox1.Text, ""))

{

if (serialPort1.IsOpen) serialPort1.Close();

try

{

if (RxString[0] == 'B')

{

flag\_sensor = 10;

}

const string WRITEKEY = "F0252SK1A7O3D2J9"; ////use your channel API keys

string strUpdateBase = "http://api.thingspeak.com/update";

string strUpdateURI = strUpdateBase + "?key=" + WRITEKEY;

string strField1 = textBox1.Text;

//string strField1 = temp;

//string strField2 = voice;

//string strField2 = "42";

HttpWebRequest ThingsSpeakReq;

HttpWebResponse ThingsSpeakResp;

if (flag\_sensor == 11)

{

strUpdateURI += "&field1=" + strField1;

//strUpdateURI += "&field1=" + temp;

}

/\* else if (flag\_sensor == 12)

{

// strUpdateURI += "&field2=" + voice;

strUpdateURI += "&field2=" + strField1;

}\*/

else

{

}

flag\_sensor++;

ThingsSpeakReq = (HttpWebRequest)WebRequest.Create

(strUpdateURI);

ThingsSpeakResp = (HttpWebResponse)

ThingsSpeakReq.GetResponse();

ThingsSpeakResp.Close();

if (!(string.Equals(ThingsSpeakResp.StatusDescription,

"OK")))

{

Exception exData = new Exception

(ThingsSpeakResp.StatusDescription);

throw exData;

}

}

catch (Exception ex)

{

}

textBox1.Text = "";

serialPort1.Open();

}

}

private void SerialPort1\_DataReceived(object sender, System.IO.Ports.SerialDataReceivedEventArgs e)

{

Console.WriteLine("Data Received");

RxString = serialPort1.ReadExisting();

if (RxString.Contains(charb))

{

startflag = 9;

}

else { }

if (startflag == 9)

{

startflag++;

}

else if (startflag == 10)

{

temp = RxString;

startflag++;

}

else if (startflag == 11)

{

voice = RxString;

startflag++;

}

else

{

// startflag = 0;

}

this.Invoke(new EventHandler(label2\_Click));

}

}

}