Android Bluetooth Temperature Monitor

By Romita Mullick

An MSP430 based project using the HC-05 Bluetooth module and a phone Android App. The project monitors the surrounding temperature and transfers the live feed to the user's phone via Bluetooth. It is a simple user friendly application which enables the user to keep track of the temperature simply by a few clicks on the phone, which continuously receives temperature feeds serially from the MSP430.

Mentor : Mr. Kishor Narang, MD, Narnix Technolabs

Vasant Kunj, New Delhi

June-July 2014

Summer Internship Project

INDEX

Acknowledgements

I am sincerely indebted to Narang Sir, MD Narnix Technolabs for granting me a summer internship at Narnix Technolabs. I was granted the flexibility and time to pick a comprehensive project which would be both practical and useful to the common man. My sincere thanks to sir for guiding me and encouraging me to keep pursuing what I thought best and for giving his valuable ideas and inputs. It was a great learning curve being associated in Narnix as an intern. My project focussed in the area of serial communication. Being from Electronics and Communications Engineering, doing such a project greatly widened my technical understanding and microcontroller application skills. Using Bluetooth mode of communication as the medium, I gained experience in the very basic and yet most important areas of asynchronous serial communication.

ABOUT THE ORGANISATION

NARNIX: THE MISSION

Narnix harbours the mission to introduce people as not merely an industrial or business organization, but an institution where we propose to learn, maneuver and interface technology to the 'super industrial society' of today.

Narnix is one of the pioneer “Independent Design Houses” in India engaged in design & technology consultancy since 1981 providing complete Design & Development Support to Electronic Products & Systems Manufacturing Organizations.

UNIQUE DESIGN STRENGTH

- Hardware Critical Embedded Designs
- Mixed Signal Designs
- Power n Energy Efficient Designs

In the fields of High Frequency Power, Industrial, Telecom, Convergence & Consumer Electronics.

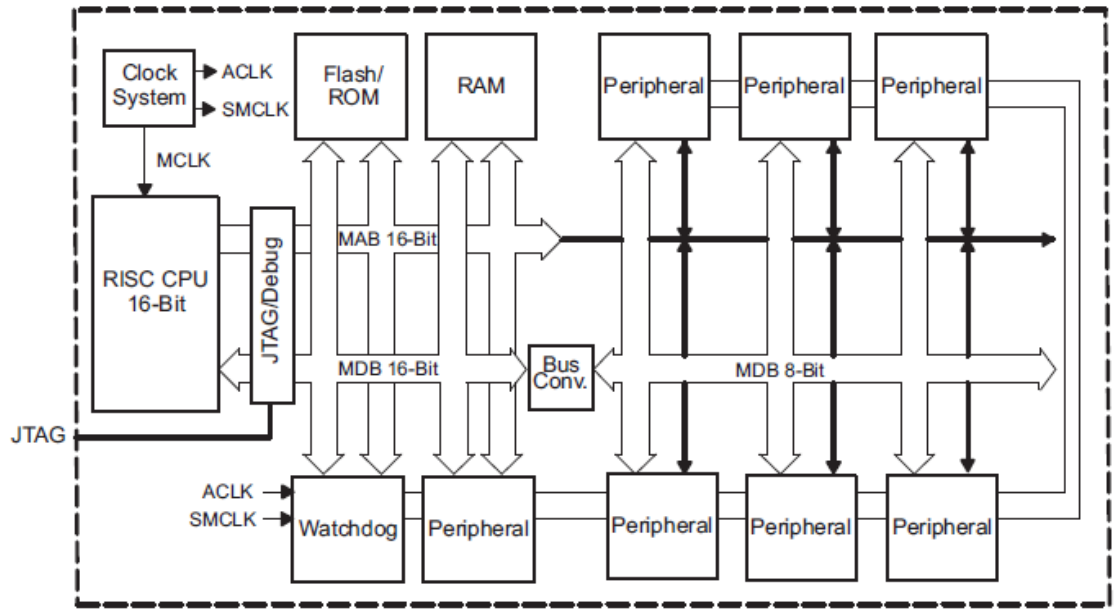
INTRODUCTION

This is an MSP 430 G2553 based project. The main modules of the MSP430 used here are the ADC and the USCI modules which are explained in detail below. The ADC used is a 10 bit adc with 1024 steps. The USCI stands for the Universal Serial Communication Interface. Particularly, the USCI_A module is used here which includes the UART (Universal Asynchronous Receive Transmit). A Bluetooth module HC-05 is used to link the MSP430 to the Android phone. The aim of the project is to send live temperature data from the microcontroller to the user's phone via Bluetooth connectivity. Temperature is being read using the inbuilt temperature sensor on the TI MSP430 Launchpad. This temperature is being transmitted serially to the phone's Bluetooth feeds continually. It has been coded in such a way that the temperature can be received in three different modes, depending on the type of command the user sends to the HC-05. They are the digital, Celsius and Farenheit modes.

MSP430 MICROCONTROLLER : OVERVIEW

Architecture

The MSP430 incorporates a 16-bit RISC CPU, peripherals, and a flexible clock system that interconnect
using a von-Neumann common memory address bus (MAB) and memory data bus (MDB). Partnering a modern CPU with modular memory-mapped analog and digital peripherals, the MSP430 offers solutions for demanding mixed-signal applications.
Key features of the MSP430x2xx family include:
• Ultralow-power architecture extends battery life
– 0.1 µA RAM retention
– 0.8 µA real-time clock mode
– 250 µA/MIPS active
• High-performance analog ideal for precision measurement
– Comparator-gated timers for measuring resistive elements
• 16-bit RISC CPU enables new applications at a fraction of the code size.
– Large register file eliminates working file bottleneck
– Compact core design reduces power consumption and cost
– Optimized for modern high-level programming
– Only 27 core instructions and seven addressing modes
– Extensive vectored-interrupt capability
• In-system programmable Flash permits flexible code changes, field upgrades and data logging



MSP430 ARCHITECTURE

Flexible Clock System

The clock system is designed specifically for battery-powered applications. A low-frequency auxiliary clock (ACLK) is driven directly from a common 32-kHz watch crystal. The ACLK can be used for a background real-time clock self wake-up function. An integrated high-speed digitally controlled oscillator (DCO) can source the master clock (MCLK) used by the CPU and high-speed peripherals. By

design, the DCO is active and stable in less than 2 µs at 1 MHz. MSP430-based solutions effectively use the high-performance 16-bit RISC CPU in very short bursts.
• Low-frequency auxiliary clock = Ultralow-power stand-by mode
• High-speed master clock = High performance signal processing

Address Space

The MSP430 von-Neumann architecture has one address space shared with special function registers (SFRs), peripherals, RAM, and Flash/ROM memory as shown in Figure 1-2. See the device-specific data sheets for specific memory maps. Code access are always performed on even addresses. Data can be accessed as bytes or words. The addressable memory space is currently 128 KB.

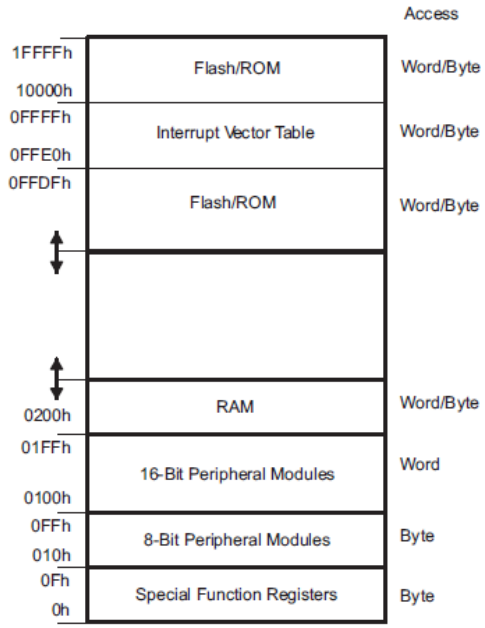


Figure 1-2. Memory Map

Flash/ROM

The start address of Flash/ROM depends on the amount of Flash/ROM present and varies by device. The end address for Flash/ROM is 0x0FFFF for devices with less that 60KB of Flash/ROM. Flash can be used for both code and data. Word or byte tables can be stored and used in Flash/ROM without the need to copy the tables to RAM before using them. The interrupt vector table is mapped into the upper 16 words of Flash/ROM address space, with the highest priority interrupt vector at the highest Flash/ROM word address (0x0FFFE).

RAM

RAM starts at 0200h. The end address of RAM depends on the amount of RAM present and varies by device. RAM can be used for both code and data.

Peripheral Modules

Peripheral modules are mapped into the address space. The address space from 0100 to 01FFh is

reserved for 16-bit peripheral modules. These modules should be accessed with word instructions. If byte instructions are used, only even addresses are permissible, and the high byte of the result is always 0. The address space from 010h to 0FFh is reserved for 8-bit peripheral modules. These modules should be accessed with byte instructions. Read access of byte modules using word instructions results in unpredictable data in the high byte. If word data is written to a byte module only the low byte is written into the peripheral register, ignoring the high byte.

Special Function Registers (SFRs)

Some peripheral functions are configured in the SFRs. The SFRs are located in the lower 16 bytes of the address space, and are organized by byte. SFRs must be accessed using byte instructions only.

.

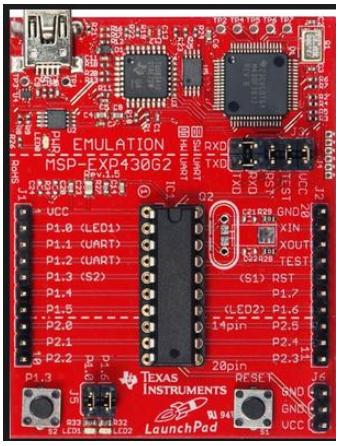
Memory Organization

Bytes are located at even or odd addresses. Words are only located at even addresses. When using word instructions, only even addresses may be used. The low byte of a word is always an even address. The high byte is at the next odd address. For example, if a data word is located at address xxx4h, then the low byte of that data word is located at address xxx4h, and the high byte of that word is located at address xxx5h.

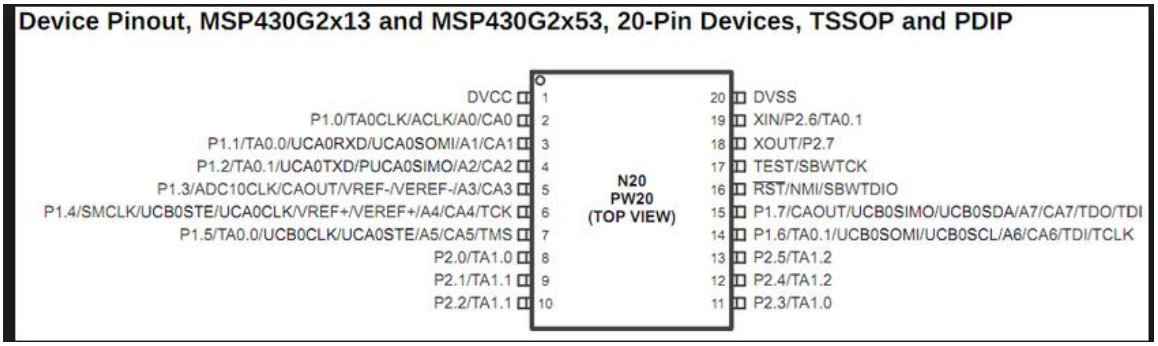
MSP430 G2553

The G2553 is a G family series microcontroller. The 20 pin DIP package is used here. The MSP430 G2553 was chosen for its ultra low power consumption.

Active Mode: 230 µA at 1 MHz, 2.2 V
– Standby Mode: 0.5 µA
– Off Mode (RAM Retention): 0.1 µA
• Five Power-Saving Modes
• Ultra-Fast Wake-Up From Standby Mode in less Than 1 µs
• On-Chip Comparator for Analog Signal
• 16-Bit RISC Architecture, 62.5-ns Instruction Compare Function or Slope Analog-to-Digital Cycle Time (A/D) Conversion
• Basic Clock Module
– Internal Frequencies up to 16 MHz With Four Calibrated Frequency
• Brownout Detector
– Internal Very-Low-Power Low-Frequency (LF) Oscillator
– 32-kHz Crystal
– External Digital Clock Source
• Two 16-Bit Timer_A With Three Capture/Compare Registers



LAUNCHPAD(right)



PIN DIAGRAM

ABOUT THE CORE MODULS : USCI AND ADC

The core modules of the MSP430 that are being used are the USCI and the ADC modules.

THE UNIVERSAL SERIAL COMMUNICATION INTERFACE (USCI)

The USCI enables either asynchronous or synchronous serial communication. The USCI are of different types, such as USCI_A and USCI_B. When more than one module of the same type is present in the device, it is named serially as 0, 1 etc as USCA_0 and USCA_1.

The USCI_Ax modules support:
• UART mode
• Pulse shaping for IrDA communications
• Automatic baud rate detection for LIN communications
• SPI mode
The USCI_Bx modules support:
• I2C mode
• SPI mode

Here we use the UART mode (Universal Asynchronous Receive Transmit). Hence, the USCA_x connect the msp430 to the external serial communication device via the USCARXD and USCATXD pins.

UART mode features include:
• 7- or 8-bit data with odd, even, or non-parity
• Independent transmit and receive shift registers
• Separate transmit and receive buffer registers
• LSB-first or MSB-first data transmit and receive
• Built-in idle-line and address-bit communication protocols for multiprocessor systems
• Receiver start-edge detection for auto-wake up from LPMx modes
• Programmable baud rate with modulation for fractional baud rate support
• Status flags for error detection and suppression
• Status flags for address detection
• Independent interrupt capability for receive and transmit

In UART, the receiving and transmitting devices are asynchronous to each other, i.e , their bit rates are different. However, the receive and transmit functions use the same baud rate. Also, UCMODE_x = 00.

Character Format

The UART character format consists of a start bit, seven or eight data bits, an even/odd/no parity bit, an address bit (address-bit mode), and one or two stop bits. The UCMSB bit controls the direction of the transfer and selects LSB or MSB first. LSB-first is typically required for UART communication.

Basically, the receive and transfer operations are expounded hereunder.

The USCI module is enabled by clearing the UCSWRST bit in the UCA0CTL1 register. The baud rate generator is activated by the falling edge of the start bit. If the start bit detected is valid, then a character is received. If the start bit isn't valid, then the baud rate generator will be turned off and the UART state machine will go back to idle state.

Characters are being received by the mcu from some serial device such as the hyperterminal window (putty, realterm etc) or from the phone(via Bluetooth) or from an RFID transceiver module. The character is received on the UCA0RXD in the receive shift register, then the last character is transferred to the UCA0RXBUF. The UCA0RXIFG flag gets set every time a character needs to be loaded into the RXBUF. Once the character is read from the RXBUF, the RXIFG interrupt flag gets reset automatically.

When the mcu needs to serially send characters to an external portal such as the putty (hyperterminal) window or to a phone (via Bluetooth), it performs a serial 'Write' function. When there is no data waiting to be written, the UCA0TXIFG flag bit is zero. But once a character to be written arrives, this TXIFG flag gets set and the data is stored in the UCA0TXBUF register. From here the character is transferred into the transmit shift register to be sent on the UCA0TXD to the mcu. Once data is written into the TXBUF, the TXIFG resets automatically.

The USCI baud rate generation can be achieved by dividing the BRCLK (baud rate clock source) by a suitable division factor N to achieve the desired baud rate. There are two baud rate generation modes selected by the UCOS16 bit. When zero, it selects the low frequency mode in which baud rates are generated from low frequency clock sources. This also reduces power consumption. When UCOS16 bit is set, the oversampling baud rate generation is used which uses higher input clock frequencies.
Here the low frequency clock source is used. It uses one prescalar and one modulator. For example, if we wish to use a 1MHz clock source for BR= 9600, then N = 1,000,000/9600 = 104.16667

For a given BRCLK clock source, the baud rate used determines the required division factor N:

$$N = \frac{f_{BRCLK}}{\text{Baud rate}}$$

The division factor N is often a non-integer value thus at least one divider and one modulator stage is used to meet the factor as closely as possible.

In the low-frequency mode, the integer portion of the divisor is realized by the prescaler:
UCBRx = INT(N)
and the fractional portion is realized by the modulator with the following nominal formula:
UCBRSx = round((N – INT(N)) × 8)
Incrementing or decrementing the UCBRSx setting by one count may give a lower maximum bit error for any given bit. To determine if this is the case, a detailed error calculation must be performed for each bit for each UCBRSx setting.

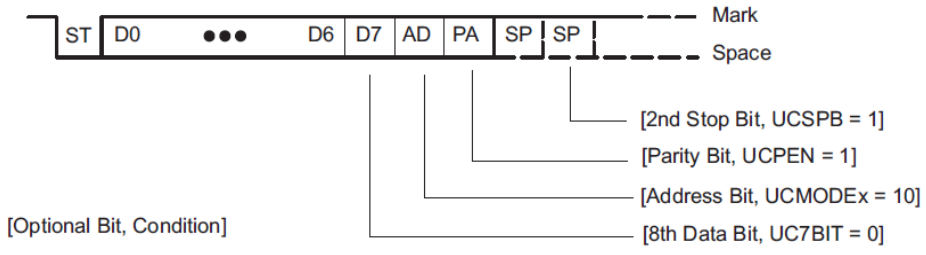


Figure 15-2. Character Format

Therefore, in this example, UCBR0 = 104
And UCBRS0= round ((104.16667-104) * 8)= round(0.16667 * 8)= round(1.3333)= 1.
UCBR0 and UCBRS0 are the prescalars and modulators respectively.

Using the USCI Module in UART Mode with Low Power Modes

The USCI module provides automatic clock activation for SMCLK for use with low-power modes. When SMCLK is the USCI clock source, and is inactive because the device is in a low-power mode, the USCI module automatically activates it when needed, regardless of the control-bit settings for the clock source. The clock remains active until the USCI module returns to its idle condition. After the USCI module returns to the idle condition, control of the clock source reverts to the settings of its control bits. Automatic clock activation is not provided for ACLK.

A few registers are explained below with their bit configuration.

UCA0CTL0 – Control Register 0

7	6	5	4	3	2	1	0
UCPEN	UCPAR	UCMSB	UC7BIT	UCSPB	UCMODEx		UCSYNC
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

UCPEN	Bit 7	Parity enable
	0	Parity disabled.
	1	Parity enabled. Parity bit is generated (UCAxTXD) and expected (UCAxRXD). In address-bit multiprocessor mode, the address bit is included in the parity calculation.
UCPAR	Bit 6	Parity select. UCPAR is not used when parity is disabled.
	0	Odd parity
	1	Even parity
UCMSB	Bit 5	MSB first select. Controls the direction of the receive and transmit shift register.
	0	LSB first
	1	MSB first
UC7BIT	Bit 4	Character length. Selects 7-bit or 8-bit character length.
	0	8-bit data
	1	7-bit data
UCSPB	Bit 3	Stop bit select. Number of stop bits.
	0	One stop bit
	1	Two stop bits
UCMODEx	Bits 2-1	USCI mode. The UCMODEx bits select the asynchronous mode when UCSYNC = 0.
	00	UART mode
	01	Idle-line multiprocessor mode
	10	Address-bit multiprocessor mode
	11	UART mode with automatic baud rate detection
UCSYNC	Bit 0	Synchronous mode enable
	0	Asynchronous mode
	1	Synchronous mode

UCA0CTL1 – Control Register 1

7	6	5	4	3	2	1	0
UCSSELx		UCRXEIE	UCBRKIE	UCDORM	UCTXADDR	UCTXBRK	UCSWRST
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

UCSSELx Bits 7-6 USCI clock source select. These bits select the BRCLK source clock.
- 00 UCLK
- 01 ACLK
- 10 SMCLK
- 11 SMCLK

UCRXEIE Bit 5 Receive erroneous-character interrupt-enable
- 0 Erroneous characters rejected and UCAxRXIFG is not set
- 1 Erroneous characters received will set UCAxRXIFG

UCBRKIE Bit 4 Receive break character interrupt-enable
- 0 Received break characters do not set UCAxRXIFG.
- 1 Received break characters set UCAxRXIFG.

UCDORM Bit 3 Dormant. Puts USCI into sleep mode.
- 0 Not dormant. All received characters will set UCAxRXIFG.
- 1 Dormant. Only characters that are preceded by an idle-line or with address bit set will set UCAxRXIFG. In UART mode with automatic baud rate detection only the combination of a break and synch field will set UCAxRXIFG.

UCTXADDR Bit 2 Transmit address. Next frame to be transmitted will be marked as address depending on the selected multiprocessor mode.
- 0 Next frame transmitted is data
- 1 Next frame transmitted is an address

UCTXBRK Bit 1 Transmit break. Transmits a break with the next write to the transmit buffer. In UART mode with automatic baud rate detection 055h must be written into UCAxTXBUF to generate the required break/synch fields. Otherwise 0h must be written into the transmit buffer.
- 0 Next frame transmitted is not a break
- 1 Next frame transmitted is a break or a break/synch

UCSWRST Bit 0 Software reset enable
- 0 Disabled. USCI reset released for operation.
- 1 Enabled. USCI logic held in reset state.

UCA0BR0 and UCA0BR1 – Baud Rate registers 0 and 1

They form the LSB and MSB respectively of the prescalar value.

15.4.3 UCAxBR0, USCI_Ax Baud Rate Control Register 0

7	6	5	4	3	2	1	0
UCBRx							
rw	rw	rw	rw	rw	rw	rw	rw

15.4.4 UCAxBR1, USCI_Ax Baud Rate Control Register 1

7	6	5	4	3	2	1	0
UCBRx							
rw	rw	rw	rw	rw	rw	rw	rw

UCBRx 7-0 Clock prescaler setting of the Baud rate generator. The 16-bit value of (UCAxBR0 + UCAxBR1 × 256) forms the prescaler value.

UCA0MCTL

UCABRS0 bits in the UCA0MCTL register forms the modulator values from 0-7

15.4.5 UCAxMCTL, USCI_Ax Modulation Control Register

7	6	5	4	3	2	1	0
UCBRFx				UCBRSx			UCOS16
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

UCBRFx	Bits 7-4	First modulation stage select. These bits determine the modulation pattern for BITCLK16 when UCOS16 = 1. Ignored with UCOS16 = 0. Table 15-3 shows the modulation pattern.
UCBRSx	Bits 3-1	Second modulation stage select. These bits determine the modulation pattern for BITCLK. Table 15-2 shows the modulation pattern.
UCOS16	Bit 0	Oversampling mode enabled 0 Disabled 1 Enabled

UCA0RXBUF and UCA0TXBUF

UCA0RXBUF is the receive buffer which receives the last character in the receive shift register when the UCA0RXIFG is set. Characters are read from the UCA0RXBUF and the UCA0RXIFG resets.

Similarly, the UCA0TXBUF holds the character which needs to be transmitted to the transmit shift register. The UCA0TXIFG then resets. When a character needs to be written, the UCA0TXIFG gets set.

15.4.7 UCAxRXBUF, USCI_Ax Receive Buffer Register

7	6	5	4	3	2	1	0
UCRXBUFx							
rw	rw	rw	rw	rw	rw	rw	rw

UCRXBUFx	Bits 7-0	The receive-data buffer is user accessible and contains the last received character from the receive shift register. Reading UCAxRXBUF resets the receive-error bits, the UCADDR or UCIDLE bit, and UCAxRXIFG. In 7-bit data mode, UCAxRXBUF is LSB justified and the MSB is always reset.

15.4.8 UCAxTXBUF, USCI_Ax Transmit Buffer Register

7	6	5	4	3	2	1	0
UCTXBUFx							
rw	rw	rw	rw	rw	rw	rw	rw

UCTXBUFx	Bits 7-0	The transmit data buffer is user accessible and holds the data waiting to be moved into the transmit shift register and transmitted on UCAxTXD. Writing to the transmit data buffer clears UCAxTXIFG. The MSB of UCAxTXBUF is not used for 7-bit data and is reset.

IE2 – interrupt enable 2 register

Only the UCA0RXIE bit is set, meaning interrupt is generated on receiving data, not on transmitting data.

15.4.12 IE2, Interrupt Enable Register 2

7	6	5	4	3	2	1	0
						UCA0TXIE	**UCA0RXIE**
						rw-0	rw-0

	Bits 7-2	These bits may be used by other modules (see the device-specific data sheet).
UCA0TXIE	Bit 1	USCI_A0 transmit interrupt enable
		0 Interrupt disabled
		1 Interrupt enabled
UCA0RXIE	Bit 0	USCI_A0 receive interrupt enable
		0 Interrupt disabled
		1 Interrupt enabled

IFG2- Interrupt Flag 2 register

It contains the UCA0RXIFG flag which gets set when a character needs to be read from the RX buffer. The UCA0TXIFG gets set when character needs to be written into the TX buffer.

15.4.13 IFG2, Interrupt Flag Register 2

7	6	5	4	3	2	1	0
						UCA0TXIFG	**UCA0RXIFG**
						rw-1	rw-0

	Bits 7-2	These bits may be used by other modules (see the device-specific data sheet).
UCA0TXIFG	Bit 1	USCI_A0 transmit interrupt flag. UCA0TXIFG is set when UCA0TXBUF is empty.
		0 No interrupt pending
		1 Interrupt pending
UCA0RXIFG	Bit 0	USCI_A0 receive interrupt flag. UCA0RXIFG is set when UCA0RXBUF has received a complete character.
		0 No interrupt pending
		1 Interrupt pending

ADC10 (ANALOG TO DIGITAL) MODULE

Next, we proceed to the next core module that is implemented in the project, ie, the ADC. The MSP430 uses its 10 bit ADC for converting the analog voltages produced by the internal temperature sensor to a digital value. The analog voltage is sampled through the sampling channel 10 (A10) at the desired adc sampling rate. Since 2^{10} = 1024, so the voltage reference selected will be divided into 1024 steps. For example, if a reference of 1.5V is chosen, each step will correspond to 1.5V/1024 = 1.46mV.

Inside the module, what really happens is that the analog voltage which is input Vin is converted to its corresponding digital value which gets stored in the ADC10MEM register.

$$N_{ADC} = 1023 \times \frac{V_{IN} - V_{R-}}{V_{R+} - V_{R-}}$$

Eg, if Vin= 0.5V

Then, Nadc= (Vin/Vref)*1024 = (0.5/1.5) * 1024 = 341.33 ~ 341

Hence, an analog voltage of 0.5V corresponds to a digital reading of 341 or 155H. Using the digital value present in the ADC10MEM, we can retrieve the corresponding analog value. The reverse calculation which our code performs is as follows :

Vin = (Nadc/1024)* Vref

Example, Vin = (341/1024) * 1.5 = 0.5V

The ADC10 module supports fast, 10-bit analog-to-digital conversions. The module implements a 10-bit SAR core, sample select control, reference generator, and data transfer controller (DTC).
The DTC allows ADC10 samples to be converted and stored anywhere in memory without CPU intervention. The module can be configured with user software to support a variety of applications.

ADC10 features include:

- Greater than 200-ksps maximum conversion rate
- Monotonic 10-bit converter with no missing codes
- Sample-and-hold with programmable sample periods
- Conversion initiation by software or Timer_A
- Software selectable on-chip reference voltage generation (1.5 V or 2.5 V)
- Software selectable internal or external reference
- Up to eight external input channels (twelve on MSP430F22xx devices)
- Conversion channels for internal temperature sensor, VCC, and external references
- Selectable conversion clock source
- Single-channel, repeated single-channel, sequence, and repeated sequence conversion modes
- ADC core and reference voltage can be powered down separately
- Data transfer controller for automatic storage of conversion results

10-Bit ADC Core

The ADC core converts an analog input to its 10-bit digital representation and stores the result in the ADC10MEM register. The core uses two programmable/selectable voltage levels (VR+ and VR-) to define the upper and lower limits of the conversion. The digital output (NADC) is full scale (03FFh) when the input signal is equal to or higher than VR+, and zero when the input signal is equal to or lower than VR-. The input channel and the reference voltage levels (VR+ and VR-) are defined in the conversion-control memory. Conversion results may be in straight binary format or 2s-complement format. The conversion formula for the ADC result when using straight binary format is:

The ADC10 core is configured by two control registers, ADC10CTL0 and ADC10CTL1. The core is enabled with the ADC10ON bit. With few exceptions the ADC10 control bits can only be modified when ENC = 0. ENC must be set to 1 before any conversion can take place.

Conversion Clock Selection

The ADC10CLK is used both as the conversion clock and to generate the sampling period. The ADC10 source clock is selected using the ADC10SSELx bits and can be divided from 1 to 8 using the ADC10DIVx bits. Possible ADC10CLK sources are SMCLK, MCLK, ACLK, and internal oscillator ADC10OSC . The ADC10OSC, generated internally, is in the 5-MHz range, but varies with individual devices, supply voltage, and temperature. See the device-specific data sheet for the ADC10OSC specification. The user must ensure that the clock chosen for ADC10CLK remains active until the end of a conversion. If the clock is removed during a conversion, the operation does not complete, and any result is invalid.

There are eight external sampling channels and four internal sampling channels. A0-A7 and A12-A15 are external and A8-A11 are internal. The ADC core can be configured while the ENC bit is cleared. Once the ENC bit is set , the ADC settings cannot be changed. There is an option to select the reference voltages using the SREFx. Eg, SREF_0 sets V+ = Vcc and V- = Vss. SREF_1 sets V+ = Vref – Vss etc. Once the SREFx is selected, the internal reference can be turned on after giving it some delay to settle down. Setting the REFON bit enables the internal voltage reference. If the REF2_5V = 1 bit is set, the reference selected is 2.5V. When REF2_5V= 0 , the reference selected is 1.5V by default. The ADC10ON bit turns on the ADC module when its set. ADC10ON =1 is done only after giving some delay after selecting and turning on the reference. The ADC10SHTx determines the sampling period. ADC10SHTx from 0 to 3 generates a sample period equal to 4,8,16 or 64 times the ADC10CLK period.

The ADC10 has 4 operation modes determined by CONSEQx,of which we use the CONSEQ_0 which is the single channel converted once. A single channel selected by INCHx is sampled and converted once. The ADC result is written to ADC10MEM.

Stopping ADC10 activity depends on the mode of operation. The recommended ways to stop an active conversion or conversion sequence are:
Resetting ENC in single-channel single-conversion mode stops a conversion immediately and the results are unpredictable. For correct results, poll the ADC10BUSY bit until reset before clearing ENC.

Using the Integrated Temperature Sensor

To use the on-chip temperature sensor, select the analog input channel INCHx = 1010. Any other configuration is done as if an external channel was selected, including reference selection,

conversion-memory selection, etc. When using the temperature sensor, the sample period must be greater than 30 µs. The temperature sensor offset error is large. Deriving absolute temperature values in the application requires calibration. See the device-specific data sheet for the parameters. Selecting the temperature sensor automatically turns on the on-chip reference generator as a voltage source for the temperature sensor. However, it does not enable the VREF+ output or affect the reference selections for the conversion. The reference choices for converting the temperature sensor are the same as with any other channel.

ADC10CTL0 – CONTROL REGISTEL 0

Here according to our requirements we have chosen SREF_1 with Vr+ = Vref+ and Vr- = Vss. The sampling period has been set at ADC10CLK * 64 cycles by setting ADC10SHT_3 and the adc sampling rate at 50ksps. Also, the REF2_5V being 0, 1.5V is taken as the reference. After a delay of 50 us , the REFON is set. And after a further 50 us delay, the ADC10ON is set, filanny turning on the ADC module. All these changes are done with ENC =0.
Once all the settings are done, then after checking whether the AD10BUSY flag is busy or not, ENC and ADC10SC can be set to enable a new conversion and start the conversion.

15	14	13	12	11	10	9	8
SREFx			ADC10SHTx		ADC10SR	REFOUT	REFBURST
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
MSC	REF2_5V	REFON	ADC10ON	ADC10IE	ADC10IFG	ENC	ADC10SC
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Can be modified only when ENC = 0

SREFx Bits 15-13 Select reference.
- 000 $V_{R+} = V_{CC}$ and $V_{R-} = V_{SS}$
- 001 $V_{R+} = V_{REF+}$ and $V_{R-} = V_{SS}$
- 010 $V_{R+} = V_{eREF+}$ and $V_{R-} = V_{SS}$. Devices with V_{eREF+} only.
- 011 V_{R+} = Buffered V_{eREF+} and $V_{R-} = V_{SS}$. Devices with V_{eREF+} pin only.
- 100 $V_{R+} = V_{CC}$ and $V_{R-} = V_{REF-}/V_{eREF-}$. Devices with V_{eREF-} pin only.
- 101 $V_{R+} = V_{REF+}$ and $V_{R-} = V_{REF-}/V_{eREF-}$. Devices with $V_{eREF+/-}$ pins only.
- 110 $V_{R+} = V_{eREF+}$ and $V_{R-} = V_{REF-}/V_{eREF-}$. Devices with $V_{eREF+/-}$ pins only.
- 111 V_{R+} = Buffered V_{eREF+} and $V_{R-} = V_{REF-}/V_{eREF-}$. Devices with $V_{eREF+/-}$ pins only.

ADC10SHTx Bits 12-11 ADC10 sample-and-hold time
- 00 4 × ADC10CLKs
- 01 8 × ADC10CLKs
- 10 16 × ADC10CLKs
- 11 64 × ADC10CLKs

ADC10SR Bit 10 ADC10 sampling rate. This bit selects the reference buffer drive capability for the maximum sampling rate. Setting ADC10SR reduces the current consumption of the reference buffer.
- 0 Reference buffer supports up to ~200 ksps
- 1 Reference buffer supports up to ~50 ksps

REFOUT Bit 9 Reference output
- 0 Reference output off
- 1 Reference output on. Devices with V_{eREF+} / V_{REF+} pin only.

REFBURST Bit 8 Reference burst.
- 0 Reference buffer on continuously
- 1 Reference buffer on only during sample-and-conversion

MSC Bit 7 Multiple sample and conversion. Valid only for sequence or repeated modes.
- 0 The sampling requires a rising edge of the SHI signal to trigger each sample-and-conversion.
- 1 The first rising edge of the SHI signal triggers the sampling timer, but further sample-and-conversions are performed automatically as soon as the prior conversion is completed

REF2_5V Bit 6 Reference-generator voltage. REFON must also be set.
- 0 1.5 V
- 1 2.5 V

REFON Bit 5 Reference generator on
- 0 Reference off
- 1 Reference on

ADC10ON Bit 4 ADC10 on
- 0 ADC10 off
- 1 ADC10 on

ADC10IE Bit 3 ADC10 interrupt enable
- 0 Interrupt disabled
- 1 Interrupt enabled

ADC10IFG	Bit 2	ADC10 interrupt flag. This bit is set if ADC10MEM is loaded with a conversion result. It is automatically reset when the interrupt request is accepted, or it may be reset by software. When using the DTC this flag is set when a block of transfers is completed.
		0 No interrupt pending
		1 Interrupt pending
ENC	Bit 1	Enable conversion
		0 ADC10 disabled
		1 ADC10 enabled
ADC10SC	Bit 0	Start conversion. Software-controlled sample-and-conversion start. ADC10SC and ENC may be set together with one instruction. ADC10SC is reset automatically.
		0 No sample-and-conversion start
		1 Start sample-and-conversion

ADC10CTL1 – CONTROL REGISTER 1

The bits of this register used are the INCHx. INCH_10 is used for for sampling analog channel A10 which is connected to the internal temperature sensor. CONSEQx bit used here is CONSEQ_0 to select the single channel once conversion mode. For clock selection, ADC10SSELx is used as ADC10SSEL_0 for selecting the internal oscillator ADC10OSC. ADC10OSC generates upto 5MHz of clock frequency. The ADC10DIVx divides the clock frequency by a factor from 1 to 8. Here we use ADC10DIV_3 to divide the clock by a factor of 4. Eg, if the ADC10OSC produces 5MHz, then the resulting frequency is 5MHz/4 = 1.25 MHz.

When a conversion is going on, an input is being sampled, then the ADC10BUSY flag is set. When the ADC10BUSY=0 , it means conversion is complete and the result can be obtained from the ADC10MEM register now.

15	14	13	12	11	10	9	8
INCHx				SHSx		ADC10DF	ISSH
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC10DIVx			ADC10SSELx		CONSEQx		ADC10BUSY
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-0

Can be modified only when ENC = 0

INCHx Bits 15-12 Input channel select. These bits select the channel for a single-conversion or the highest channel for a sequence of conversions. Only available ADC channels should be selected. See device specific datasheet.

- 0000 A0
- 0001 A1
- 0010 A2
- 0011 A3
- 0100 A4
- 0101 A5
- 0110 A6
- 0111 A7
- 1000 V_{eREF+}
- 1001 V_{REF-}/V_{eREF-}
- 1010 Temperature sensor
- 1011 $(V_{CC} - V_{SS}) / 2$
- 1100 $(V_{CC} - V_{SS}) / 2$, A12 on MSP430F22xx devices
- 1101 $(V_{CC} - V_{SS}) / 2$, A13 on MSP430F22xx devices
- 1110 $(V_{CC} - V_{SS}) / 2$, A14 on MSP430F22xx devices
- 1111 $(V_{CC} - V_{SS}) / 2$, A15 on MSP430F22xx devices

SHSx Bits 11-10 Sample-and-hold source select.

- 00 ADC10SC bit
- 01 Timer_A.OUT1(1)
- 10 Timer_A.OUT0(1)
- 11 Timer_A.OUT2 (Timer_A.OUT1 on MSP430F20x0, MSP430G2x31, and MSP430G2x30 devices)(1)

ADC10DF Bit 9 ADC10 data format

- 0 Straight binary
- 1 2s complement

ISSH Bit 8 Invert signal sample-and-hold

- 0 The sample-input signal is not inverted.
- 1 The sample-input signal is inverted.

ADC10DIVx Bits 7-5 ADC10 clock divider

- 000 /1
- 001 /2
- 010 /3
- 011 /4
- 100 /5
- 101 /6
- 110 /7
- 111 /8

ADC10SSELx Bits 4-3 ADC10 clock source select

- 00 ADC10OSC
- 01 ACLK
- 10 MCLK
- 11 SMCLK

CONSEQx Bits 2-1 Conversion sequence mode select

- 00 Single-channel-single-conversion
- 01 Sequence-of-channels
- 10 Repeat-single-channel
- 11 Repeat-sequence-of-channels

ADC10BUSY Bit 0 ADC10 busy. This bit indicates an active sample or conversion operation

- 0 No operation is active.
- 1 A sequence, sample, or conversion is active.

ADC10AE0 and ADC10AE1 – analog enable registers.

ADC10AE0 register enables the external analog inputs for channels A0 to A7. BIT0 is set for A0, BIT1 is set for A1 and so on. ADC10AE1 uses only bits 4-7. BIT4 corresponds to A12, BIT 5 corresponds to A13 , BIT6 to A14 and BIT7 to A15. Bits 0-3 are reserved. ADC10AE1 cannot be set for the internal input channels A8- A11.

22.3.3 ADC10AE0, Analog (Input) Enable Control Register 0

7	6	5	4	3	2	1	0
ADC10AE0x							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

ADC10AE0x	Bits 7-0	ADC10 analog enable. These bits enable the corresponding pin for analog input. BIT0 corresponds to A0, BIT1 corresponds to A1, etc. The analog enable bit of not implemented channels should not be programmed to 1. 0 Analog input disabled 1 Analog input enabled

22.3.4 ADC10AE1, Analog (Input) Enable Control Register 1 (MSP430F22xx only)

7	6	5	4	3	2	1	0
ADC10AE1x				Reserved			
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

ADC10AE1x	Bits 7-4	ADC10 analog enable. These bits enable the corresponding pin for analog input. BIT4 corresponds to A12, BIT5 corresponds to A13, BIT6 corresponds to A14, and BIT7 corresponds to A15. The analog enable bit of not implemented channels should not be programmed to 1. 0 Analog input disabled 1 Analog input enabled
Reserved	Bits 3-0	Reserved

ADC10MEM

Holds the 10 bit digital value from bits 0 to 9. They are right justified.

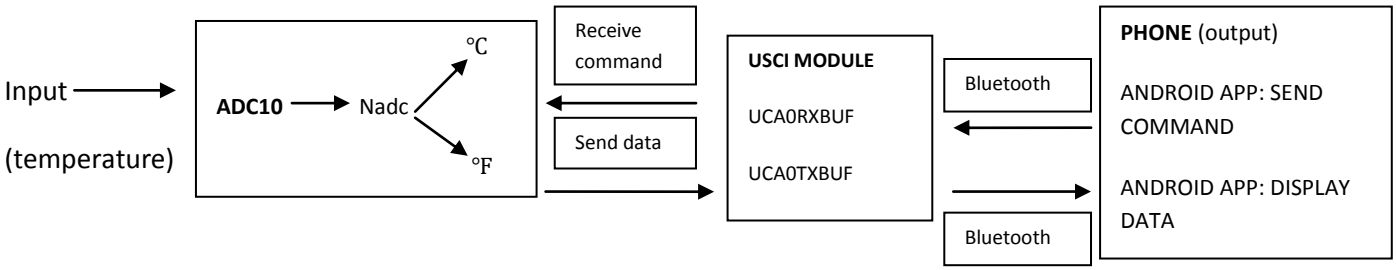
22.3.5 ADC10MEM, Conversion-Memory Register, Binary Format

15	14	13	12	11	10	9	8
0	0	0	0	0	0	Conversion Results	
r0	r0	r0	r0	r0	r0	r	r
7	6	5	4	3	2	1	0
Conversion Results							
r	r	r	r	r	r	r	r

Conversion Results	Bits 15-0	The 10-bit conversion results are right justified, straight-binary format. Bit 9 is the MSB. Bits 15-10 are always 0.

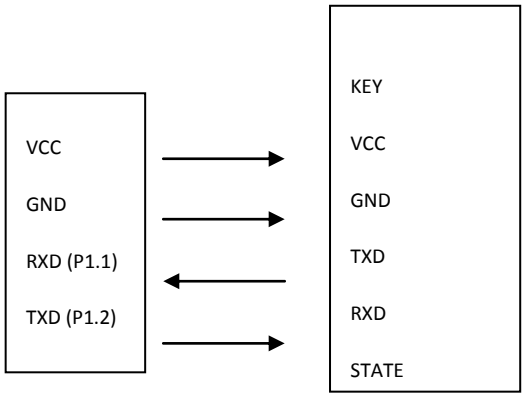
INTERFACING

Now that the basic core modules have been explained, we move onto their application in the project. It process is illustrated with a simple block representation.



The temperature sensor takes in the input and feeds it to the ADC. The ADC processes and does sample and conversion. The digital value gets stored in the ADC10MEM register as Nadc. Algorithms applied in the code convert the Nadc to Celsius and Farenheit. These values are sent via the 'send data' arrow to the UCA0TXBUF from which it is sent onto the display on the App via Bluetooth pairing. The user intern sends commands to the UCA0RXBUF. This sends the 'receive command' to the mcu which processes it to intern send back corresponding data in the same manner.

The connections between the MCU and the Bluetooth module are made as below :



MSP 430 HC-05

BLUETOOTH

Bluetooth serial module is used for converting serial port to Bluetooth. The HC series modules have two modes: master and slaver device. The device named after even number is defined to be master or slaver when out of factory and can't be changed to the other mode. But for the device named after odd number, users can set the work mode (master or slaver) of the device by AT commands.

The main function of Bluetooth serial module is replacing the serial port line, such as:

1. There are two MCUs want to communicate with each other. One connects to Bluetooth master device while the other one connects to slave device. Their connection can be built once the pair is made. This Bluetooth connection is equivalently liked to a serial port line connection including RXD, TXD signals. And they can use the Bluetooth serial module to communicate with each other.
2. When MCU has Bluetooth salve module, it can communicate with Bluetooth adapter of computers and smart phones. Then there is a virtual communicable serial port line between MCU and computer or smart phone.
3. The Bluetooth devices in the market mostly are salve devices, such as Bluetooth printer, Bluetooth GPS. So, we can use master module to make pair and communicate with them. Bluetooth Serial module's operation doesn't need drive, and can communicate with the other Bluetooth device who has the serial. But communication between two Bluetooth modules requires at least two conditions:

1. The communication must be between master and slave.
2. The password must be correct.

HC-05

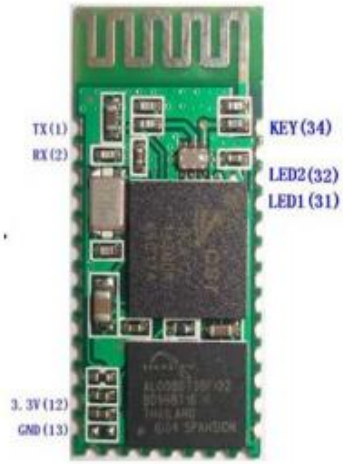
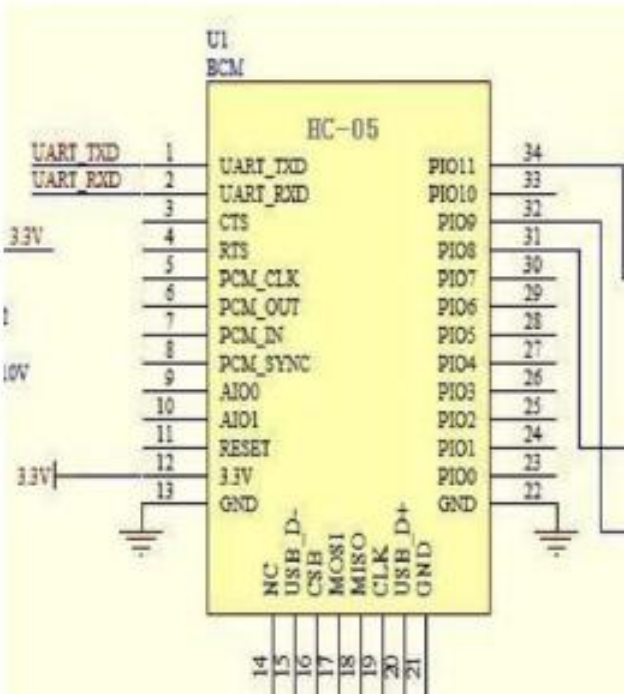
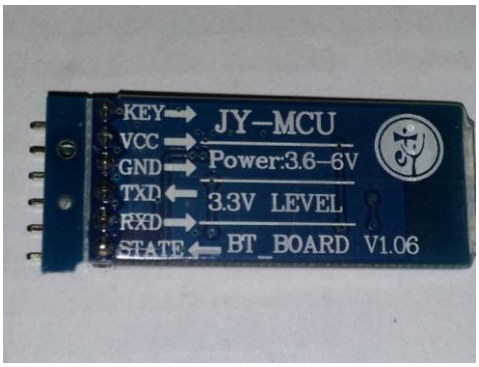


Figure 2 HC-05

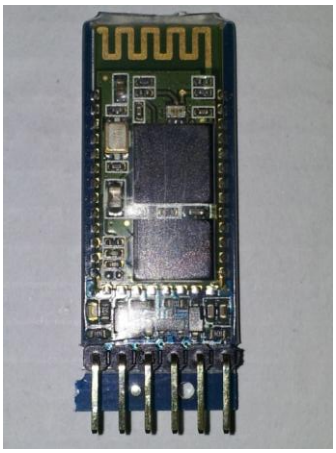


Master and slave mode can be switched Master
Bluetooth name: HC-05
Password:1234



HC-05 embedded Bluetooth serial communication module (can be short for module) has two work modes: order-response work mode and automatic connection
work mode. And there are three work roles (Master, Slave and Loopback) at the automatic connection work mode. When the module is at the automatic connection
work mode, it will follow the default way set lastly to transmit the data automatically. When the module is at the order-response work mode, user can send the AT command to the module to set the control parameters and sent control order. The work mode of module can be switched by controlling the module PIN (PIO11) input level.

Serial module PINs:



1. PIO8 connects with LED. When the module is power on, LED will flicker. And the
flicker style will indicate which work mode is in using since different mode has
different flicker time interval.

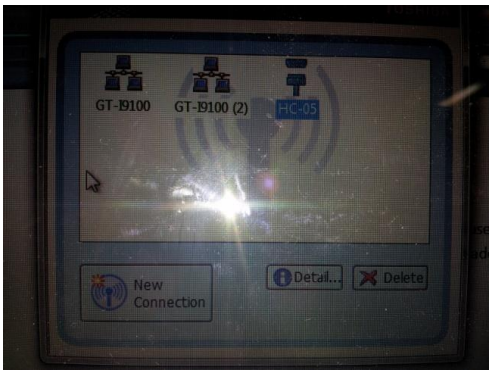
2. PIO9 connects with LED. It indicates whether the connection is built or not. When
the Bluetooth serial is paired, the LED will be turned on. It means the connection is
built successfully.

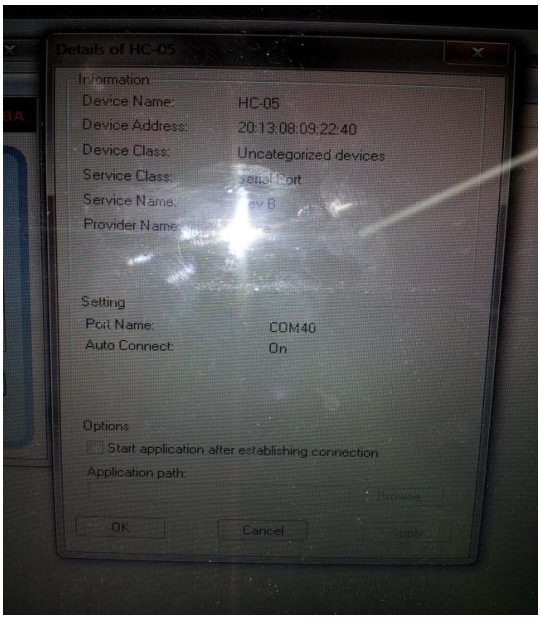
3. PIO11 is the work mode switch. When this PIN port is input high level, the work
mode will become order-response work mode. While this PIN port is input low level or suspended in air, the work mode will become automatic connection work mode.

4. The module can be reset if it is re-powered since there is a reset circuit at the module.

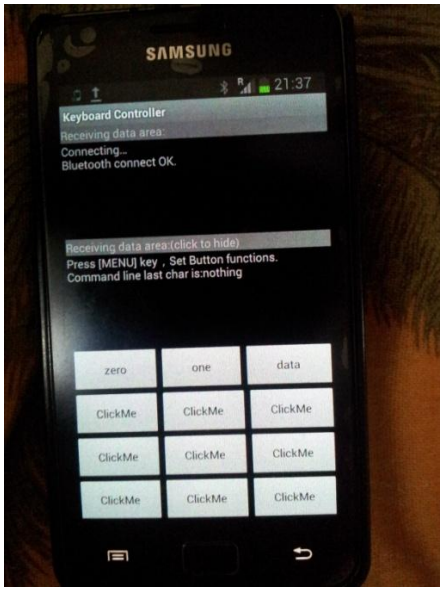
In the following project, the HC-05 is being used in the slave mode to communicate with the phone's Bluetooth.

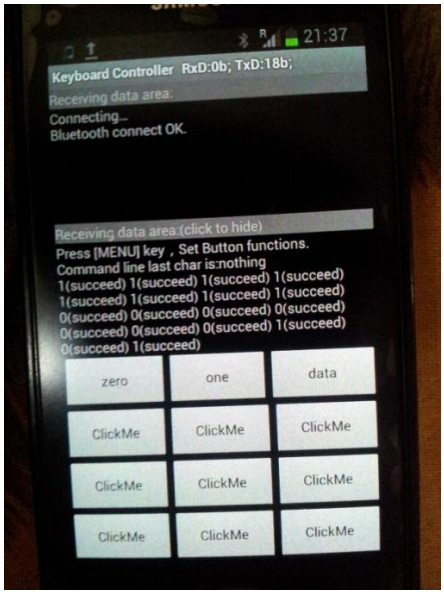
PAIRING AND CONNECTIVITY





These pictures illustrate the port number and device name. The HC-05 was detected to be at COM port 40 with name HC-05 as detected by the PC.





Similarly, this module can also be detected by the phone. The two devices are paired by entering the correct pin number. In this case, the default pin number is 1234. Once paired, connection is established. The phone used is a Samsung galaxy S2. A standard Bluetooth android app could be downloaded from the Google Play store. The one being used here is Bluetooth SPP by Jerry Li.

THE BASIC IDEA

The app is capable of both receiving serial data in the real time mode and sending commands in the keyboard mode. As can be seen in the picture, there are 9 buttons, any number of which can be customised according to the requirement to send command.

In the project, only three of these buttons have been used. Its been set to

1) Digital – command > send character '0'
2) Celsius – command > send character '1'
3) Farenheit – command > send character '2'

What the code does is, as the user sends a command from the phone via Bluetooth, this command is received by the microcontroller via the HC-05 Bluetooth. The code will serialRead the character received and store it in a unsigned character variable, say 'a'. The 'a' character is received in the receive shift register which sends it to the UCA0RXBUF from which this 'a' is read. Now, a simple if else algorithm is implemented for the three possible values of 'a'.

If (a== '0')

then send Digital

else if (a== '1')

then send Celsius

else if (a=='2')

then send Farenheit

What send basically implies is the data, whether in the digital/celsius/farenheit format is serially sent from the MCU to the phone via Bluetooth again and displayed on the app screen.

The working is again completely based on the ADC10 and the USCI modules. The ADC10MEM register stores the digital values. The formula for translating the digital Nadc value to Celsius is :

temp_C=(((nadc*420.0)+512.0)/1024.0) - 278.0

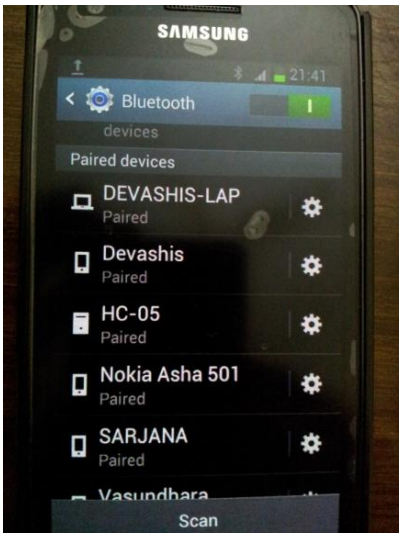
Further, the Farenheit value can be directly obtaine from the Celsius value as follows :

temp_F= ((temp_C)*(9.0/5.0)) + 32.0

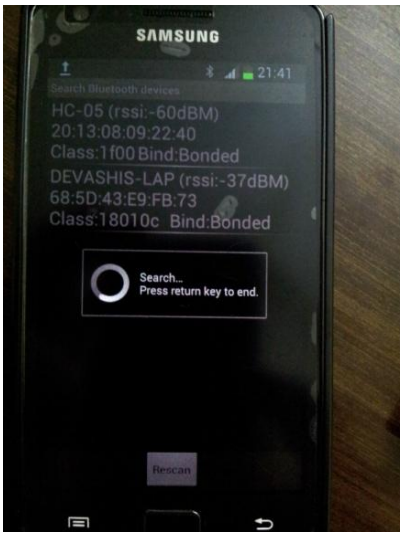
the Nadc, temp_C and temp_F values are serially sent via serialWrite to the UCA0TXBUF from which it is sent to the transmit shift register onto the TXD line.

This explains the working of the project. Simply pressing a button on your phone will show you the temperature anywhere within the Bluetooth range.

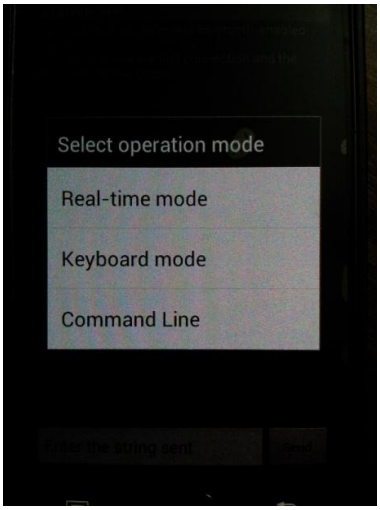
Step 1 : pairing the HC-05 with the phone the Bluetooth SPP



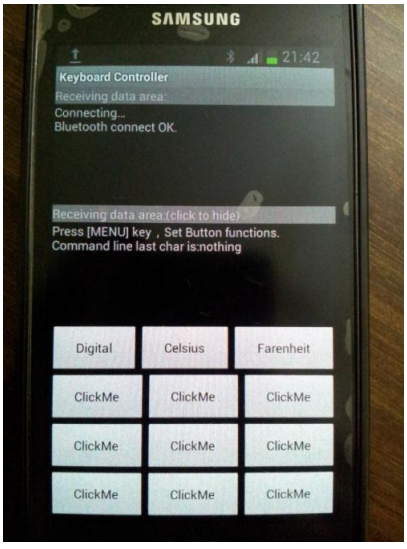
Step 2 : detecting the device on

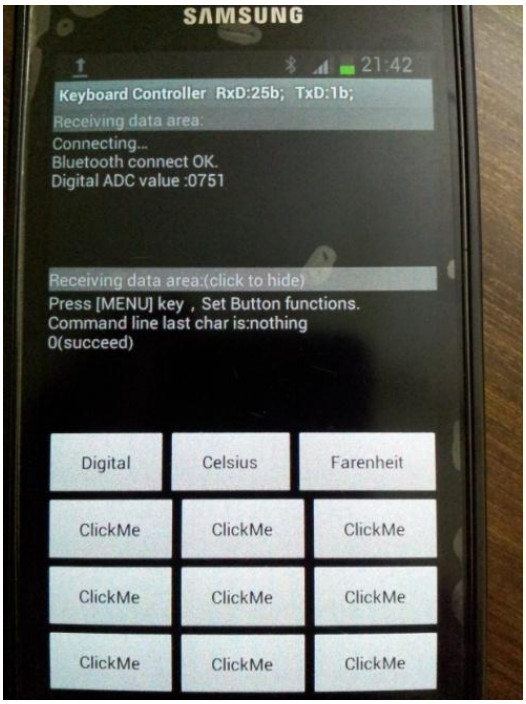


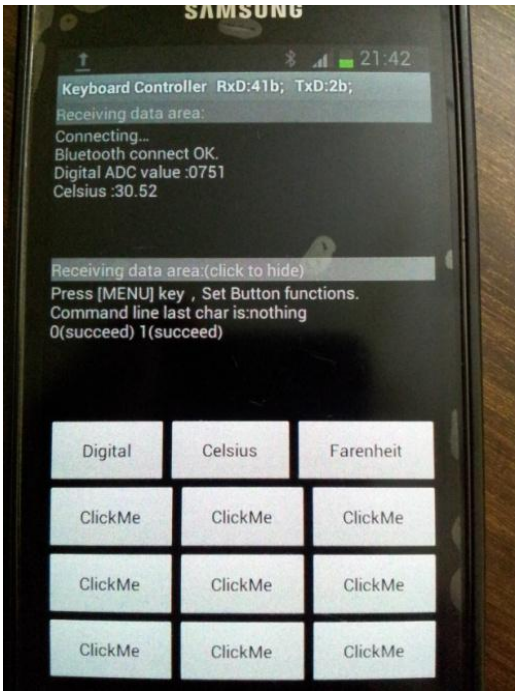
Step 3 : selecting the keyboard mode



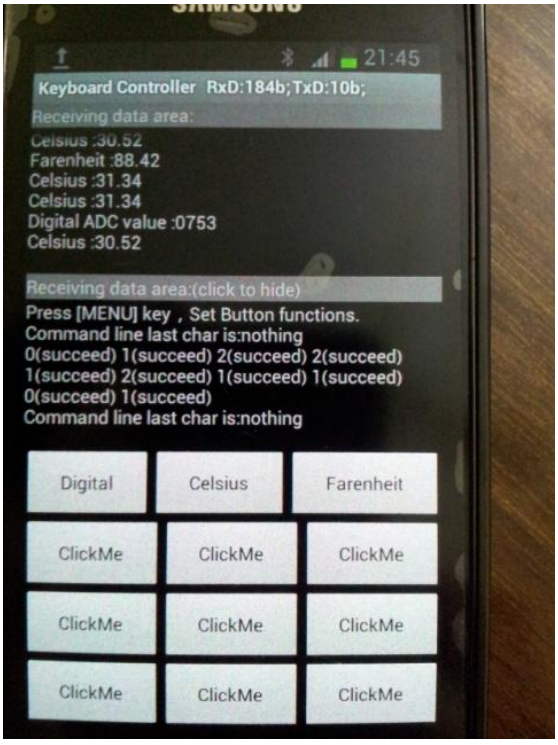
Step 4 : preparing to display, viewing the digital, Celsius and Farenheit readings in any random order by pressing the customised keys



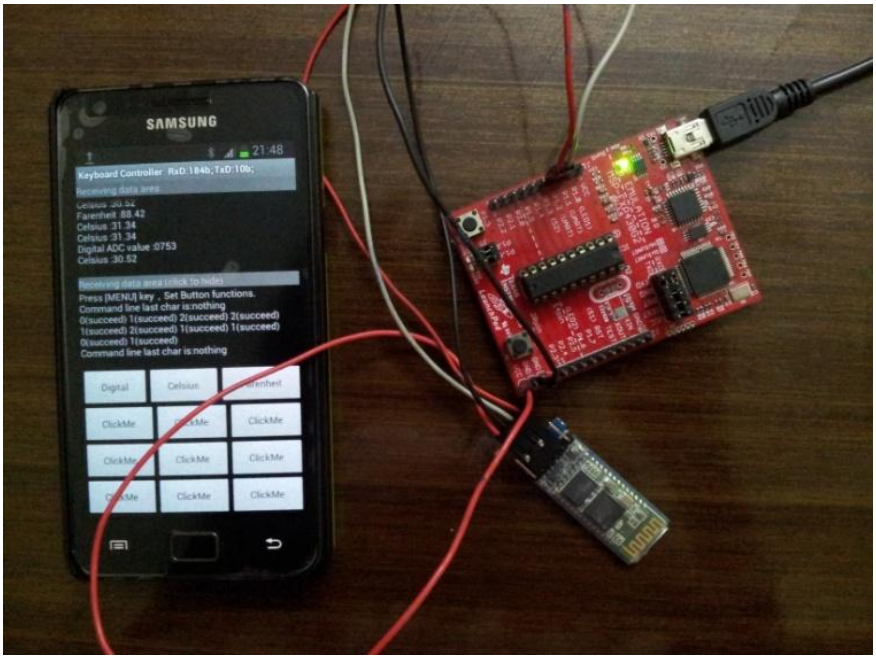

SAMSUNG
21:42
Keyboard Controller RxD:25b; TxD:1b;
Receiving data area:
Connecting...
Bluetooth connect OK.
Digital ADC value :0751
Receiving data area:(click to hide)
Press [MENU] key , Set Button functions.
Command line last char is:nothing
0(succeed)
Digital
Celsius
Farenheit
ClickMe


21:42
Keyboard Controller RxD:41b; TxD:2b;
Receiving data area:
Connecting...
Bluetooth connect OK.
Digital ADC value :0751
Celsius :30.52
Receiving data area:(click to hide)
Press [MENU] key , Set Button functions.
Command line last char is:nothing
0(succeed) 1(succeed)
Digital
Celsius
Farenheit
ClickMe


21:42
Keyboard Controller RxD:59b; TxD:3b;
Receiving data area:
Connecting...
Bluetooth connect OK.
Digital ADC value :0751
Celsius :30.52
Farenheit :86.94
Receiving data area:(click to hide)
Press [MENU] key , Set Button functions.
Command line last char is:nothing
0(succeed) 1(succeed) 2(succeed)
Digital
Celsius
Farenheit
ClickMe


21:45
Keyboard Controller RxD:184b;TxD:10b;
Receiving data area:
Farenheit :88.42
Celsius :31.34
Celsius :31.34
Digital ADC value :0753
Celsius :30.52
Receiving data area:(click to hide)
Press [MENU] key , Set Button functions.
Command line last char is:nothing
0(succeed) 1(succeed) 2(succeed) 2(succeed)
1(succeed) 2(succeed) 1(succeed) 1(succeed)
0(succeed) 1(succeed)
Command line last char is:nothing
Digital
Celsius
Farenheit
ClickMe

Summary



This is a very handy, user friendly project. Its easy to use and very cheap to make. The only costs incurred were for the HC-05 module. The launchpad was available from before. Only four jumper wires were needed for connecting. Any android phone will serve the purpose. And a Code Composer Studio (Limited version for free) installed on the PC is used for coding. Hence, useful data could be achieved with almost no hardware and very minimal cost. To summarize, this project is the very basic, fundamental illustration of bigger applications that can be achieved along similar lines, using the same principles. This project simply serves as a template for more complex applications such as home automation, wireless communication, media file transfer etc. While this project only explores the transmission of ASCII characters and character arrays, more complex data files such as audio and video files, word and document formats etc are transferred via Bluetooth in practice today. This project successfully illustrates the basic functionality and judiciously uses he minimal basic tools available.

PROGRAM CODE

```
#include <msp430g2553.h>

float temp_C=0.0;
float temp_F=0.0;
int j,i,w =0;
unsigned int nadc=0;
unsigned char b[4], c[5];
unsigned char a=2;

void adcinit()
{
        ADC10CTL0 = SREF_1 + ADC10SHT_3  + ADC10SR ;
        ADC10CTL1 = ADC10SSEL_0 + ADC10DIV_3 + INCH_10 + CONSEQ_0;
        __delay_cycles(50);
        ADC10CTL0 |= REFON;
        __delay_cycles(50);
        ADC10CTL0 |= ADC10ON;

}



void serialInit()
{
        P1SEL= BIT1 + BIT2; //P1.1 = RXD P1.2=TXD
        P1SEL2= BIT1 +BIT2; // P1.1=RXD & P1.2=TXD
        UCA0CTL1|= UCSSEL_2; // SMCLK
        UCA0BR0=104; // BAUDRATE AT 1 MHz 9600
        UCA0BR1=0;//1MHz 9600
   UCA0MCTL= UCBRS0; // MODULATION UCBRSx=1
   UCA0CTL1&=~UCSWRST; // ** INITIALIZE USCI STATE MACHINE
   IE2|= UCA0RXIE; // ENABLE VSCI_A0 RX INTERRUPT
}

unsigned char serialRead()
{
        while(!(IFG2&UCA0RXIFG));   //USCI_A0 RX buffer ready ?
        return UCA0RXBUF;

}
void serialWrite(unsigned char c)
{
        while(!(IFG2&UCA0TXIFG));  // USCI_A0 TX buffer ready ?
        UCA0TXBUF=c; // TX
}
void serialwriteString(const char *str)
{
        while(*str)
                serialWrite(*str++);
}
```

```
void serial_write_int(unsigned int temp)
{
        for( i=0;i<4;i++)
        {
                b[i]=temp%10;
                //serialWrite(b[i]);
                temp=temp/10;
        }

        for(j=3;j>=0;j--)
        {
                serialWrite(b[j] + 48);
        }

        serialWrite(' ');
        serialWrite('\n');

}


serial_write_float(float v)
{

        w = v*100;

        for(i=0;i<4;i++)
        {
                c[i]=w%10;
                w=w/10;
        }

        serialWrite(c[3]+48);
        serialWrite(c[2] +48);
        serialWrite('.');
        serialWrite(c[1] + 48);
        serialWrite(c[0] + 48);

        serialWrite(' ');
        serialWrite('\n');




}

void main()
{
        WDTCTL = WDTPW + WDTHOLD;
        BCSCTL1 = CALBC1_1MHZ;
        DCOCTL = CALDCO_1MHZ;
   P1DIR = 0x41;
   P1OUT=0x00;
```

```
        adcinit();
        serialInit();

        while(1)
        {

                ADC10CTL0 |= ENC + ADC10SC;
                while(ADC10CTL1 & ADC10BUSY)
                {}
//serialwriteString("hello !!");
//serialWrite('\n');
                nadc= ADC10MEM;

                temp_C=(((nadc*420.0)+512.0)/1024.0) - 278.0;
                temp_F= ((temp_C)*(9.0/5.0)) + 32.0;

                a=serialRead();

                if(a== '0')

                {

                        P1OUT = 0x41;
                                serialwriteString("Digital ADC value :");
                        serial_write_int(nadc);
                }

                else if (a== '1')
                {

                        P1OUT=0x01;
                        __delay_cycles(5000);
                        P1OUT=0x00;
                        serialwriteString("Celsius :");
                serial_write_float(temp_C);

                }

                else if (a =='2')

                {
                        P1OUT=0x40;
                        __delay_cycles(5000);
                        P1OUT=0X00;
                        serialwriteString("Farenheit :");
                        serial_write_float(temp_F);

                }

                }

}
```