Guide to Connect HC-06 Bluetooth Module

1. Power the HC-06 module with 5/3.3V to Vcc pin, and ground to GND pin. The LED will start flashing if the module is ready to connect.
2. Select find Bluetooth device on the laptop, make sure to choose “find all devices”. The name “HC-06” will show up. Enter the default pin “1234” to connect.
3. After pairing is completed, the device will be shown as “not connected”. This is expected. Under the Bluetooth device settings, there should be two new COM Ports added called “HC-06 port” and “HC-06”.
4. Use a terminal monitor, like TeraTerm or CoolTerm, to connect to serial port with name “HC-06 port” with outgoing direction. The LED on the device will stay on if the connection is successfully established.

Communicate with HC-06 with UART

1. HC-06 has two pins, “TX” and “RX”, for UART communication. Any serial input to the RX pin will be transmitted to the connected serial port by the HC-06 module. Similarly, any serial data received by the HC-06 will be transmitted to the TX pin.
2. A white sheet with black text and numbers

   Description automatically generatedMSP432P401R has UART port available on the pins. The UART mode can be initiated by adjusting the control bit. For example, to set pin3.2 and pin3.3 to the UART mode, we need follow the table below

Set the P3SEL1.2/P3SEL1.3 = 0 and the P3SEL0.2/P3SEL0.3 = 1. It can be done by the code below.

A close up of a text

Description automatically generated

1. For the wire connection, pin 3.2 RX need to be connected to TX on HC-06, and pin 3.3 TX need to be connected to RX on HC-06.
2. Then for transmitting data, it will be the same for regular UART functions. Wait for the flag for buffer to be ready and write the char data to the buffer.A screenshot of a computer program

   Description automatically generated

Guide to Capture Serial Input with CoolTerm

1. Download CoolTerm on the official website <https://freeware.the-meiers.org/>
2. Unzip and start the program
3. Select Connection -> File Capture -> Start. A window will pop out to ask to save the file.A screenshot of a computer

   Description automatically generated
4. After saving the txt file, the connection can be started by clicking the arrow at the bottom left corner.
5. The setting for the connection can also be adjusted at the drop down manual.

Nearest distance by USG sensor – 1.97cm, longest sensor – 2.97m.

Distance between triggers should be more than 60 ms.

Appendix

#include "msp.h"

#include <stdint.h>

#include <stdio.h>

//higher number has lower priority

#define PORT2\_PRIO 4

#define TIMEA0\_PRIO 3

#define MCLKFREQ

#define INTR\_PRD 1000

#define DIST\_DIV 1400

//according to HCSR-04 datasheet, trigger length is 10us and max echo length is 60ms

//by my measurement, the echo length is from 135us to 55ms

long intrcnt = 0;

long distance = 0;

long clktick = 0;

long timestamp = 0;

long start = 0;

long end = 0;

void UART0\_init();

int UART0\_puts(const char \*str);

void UART2\_init();

int UART2\_puts(const char \*str);

void Delay(uint32\_t tick);

long measure\_dist();

void get\_time\_stamp();

int min = 0;

int sec = 0;

/\*\*

\* The main function initializes all port, including, UART, trigger, echo. And keep triggering

\* the ultrasonic sensor and send the distance converted from echo length to the console

\*/

int main(void)

{

WDT\_A->CTL = WDT\_A\_CTL\_PW | // Stop watchdog timer

WDT\_A\_CTL\_HOLD;

//setting up clock signals

CS->KEY = CS\_KEY\_VAL; // Unlock CS module for register access

CS->CTL0 = 0; // Reset tuning parameters

CS->CTL0 = CS\_CTL0\_DCORSEL\_3; // Set DCO to 24MHz

CS->CTL1 = CS\_CTL1\_SELA\_2 | // Select ACLK = REFO

CS\_CTL1\_SELS\_3 | // SMCLK = DCO

CS\_CTL1\_SELM\_3; // MCLK = DCO

CS->KEY = 0; // Lock CS module from unintended accesses

UART2\_init();

// Configure GPIO

// Setting up LED for distance detecting

P1->DIR |= BIT0; // Set P1.0 as output

//set p2 as GPIO

P2->SEL0 = 0;

P2->SEL1 = 0;

// uses p2.7 as echo

P2->DIR &= ~BIT7; // P2.7 as input pin

P2->REN |= BIT7; // P2.7 pull resistor enabled

P2->OUT &= ~BIT7; // P2.7 selected as pull down (active low)

// uses p2.6 as trigger

P2->DIR |= BIT6; // trigger pin as output

// receiving interrupt from p2.7

P2->IFG = 0; // clean pending interrupt flag

P2->IES &= ~BIT7; // enable rising edge interrupt

P2->IE |= BIT7; // enable interrupt

//setting up timer A0

TIMER\_A0->CCTL[0] = TIMER\_A\_CCTLN\_CCIE; // CCR0 interrupt enabled

TIMER\_A0->CCR[0] = INTR\_PRD - 1; // interrupt is raised for every 1000 clock tick

TIMER\_A0->CTL = TIMER\_A\_CTL\_TASSEL\_2 | TIMER\_A\_CTL\_MC\_\_UP | TIMER\_A\_CTL\_CLR; // SMCLK, upmode, TA clear

// Enable Port2 interrupt for echo and set priority as 4

NVIC\_SetPriority(PORT2\_IRQn, PORT2\_PRIO);

NVIC\_EnableIRQ(PORT2\_IRQn);

// Enable Timer interrupt for timing and set priority as 3

NVIC\_SetPriority(TA0\_0\_IRQn, TIMEA0\_PRIO);

NVIC\_EnableIRQ(TA0\_0\_IRQn);

\_\_enable\_irq(); // Enables interrupts to the system

//blink twice before it begin

P1->OUT |= BIT0;

Delay(24000000);

P1->OUT &= ~BIT0;

Delay(24000000);

P1->OUT |= BIT0;

Delay(24000000);

P1->OUT &= ~BIT0;

intrcnt = 0;

while (1)

{

distance = measure\_dist();

get\_time\_stamp();

char buffer[50];

sprintf(buffer, "%d:%d, %d\n", min,sec, distance); // format the output string

UART2\_puts(buffer);

//turning LED on if distance is less than certain range and if distance isn't 0.

if (distance < 50 && distance != 0)

P1->OUT |= BIT0;

else

P1->OUT &= ~BIT0;

Delay(300000);

}

}

/\*\*

\* This function is initializing the UART0 for sending distance message to the console with baud rate of 9600

\*

\*/

void UART0\_init()

{

// Configure UART pins

P1->SEL0 |= BIT2 | BIT3; // set 2-UART pin as secondary function

// Configure UART

EUSCI\_A0->CTLW0 |= EUSCI\_A\_CTLW0\_SWRST; // Put eUSCI in reset

EUSCI\_A0->CTLW0 |= EUSCI\_B\_CTLW0\_SSEL\_\_SMCLK; // Configure eUSCI clock source for SMCLK

// Baud Rate calculation

// 12000000/(16\*9600) = 78.125

// Fractional portion = 0.125

// User's Guide Table 21-4: UCBRSx = 0x10

// UCBRFx = int ( (78.125-78)\*16) = 2

EUSCI\_A0->BRW = 78; // 12000000/16/9600

EUSCI\_A0->MCTLW = (2 << EUSCI\_A\_MCTLW\_BRF\_OFS) |

EUSCI\_A\_MCTLW\_OS16;

EUSCI\_A0->CTLW0 &= ~EUSCI\_A\_CTLW0\_SWRST; // take eUSCI out of reset mode

EUSCI\_A0->IFG &= ~EUSCI\_A\_IFG\_RXIFG; // Clear eUSCI RX interrupt flag

EUSCI\_A0->IE &= ~EUSCI\_A\_IE\_RXIE; // Disable USCI\_A0 RX interrupt

}

/\*\*

\* This function puts a string to transmit buffer in the UART0, which will be sent to console

\*/

int UART0\_puts(const char \*str)

{

int status = -1;

if (str != '\0')

{

status = 0;

while (\*str != '\0')

{

/\* Wait for the transmit buffer to be ready \*/

while (!(EUSCI\_A0->IFG & EUSCI\_A\_IFG\_TXIFG))

;

/\* Transmit data \*/

EUSCI\_A0->TXBUF = \*str;

/\* If there is a line-feed, add a carriage return \*/

if (\*str == '\n')

{

/\* Wait for the transmit buffer to be ready \*/

while (!(EUSCI\_A0->IFG & EUSCI\_A\_IFG\_TXIFG))

;

EUSCI\_A0->TXBUF = '\r';

}

str++;

}

}

return status;

}

void UART2\_init()

{

//pin 3.2 RX pin 3.3 TX

// Configure UART pins

P3->SEL0 |= BIT2 | BIT3; // set 2-UART pin as secondary function

P3->SEL1 &= ~(BIT2 | BIT3);

// Configure UART

EUSCI\_A2->CTLW0 |= EUSCI\_A\_CTLW0\_SWRST; // Put eUSCI in reset

EUSCI\_A2->CTLW0 |= EUSCI\_B\_CTLW0\_SSEL\_\_SMCLK; // Configure eUSCI clock source for SMCLK

// Baud Rate calculation

// 12000000/(16\*9600) = 78.125

// Fractional portion = 0.125

// User's Guide Table 21-4: UCBRSx = 0x10

// UCBRFx = int ( (78.125-78)\*16) = 2

EUSCI\_A2->BRW = 78; // 12000000/16/9600

EUSCI\_A2->MCTLW = (2 << EUSCI\_A\_MCTLW\_BRF\_OFS) |

EUSCI\_A\_MCTLW\_OS16;

EUSCI\_A2->CTLW0 &= ~EUSCI\_A\_CTLW0\_SWRST; // take eUSCI out of reset mode

EUSCI\_A2->IFG &= ~EUSCI\_A\_IFG\_RXIFG; // Clear eUSCI RX interrupt flag

EUSCI\_A2->IE &= ~EUSCI\_A\_IE\_RXIE; // Disable USCI\_A2 RX interrupt

}

int UART2\_puts(const char \*str)

{

int status = -1;

if (str != '\0') {

status = 0;

while (\*str != '\0') {

/\* Wait for the transmit buffer to be ready \*/

while (!(EUSCI\_A2->IFG & EUSCI\_A\_IFG\_TXIFG));

/\* Transmit data \*/

EUSCI\_A2->TXBUF = \*str;

/\* If there is a line-feed, add a carriage return \*/

if (\*str == '\n') {

/\* Wait for the transmit buffer to be ready \*/

while (!(EUSCI\_A2->IFG & EUSCI\_A\_IFG\_TXIFG));

EUSCI\_A2->TXBUF = '\r';

}

str++;

}

}

return status;

}

/\*\*

\* This function delay for the given amount of clock cycle

\*/

void Delay(uint32\_t tick)

{

// initialize timer32 1 with give amount of tick

TIMER32\_1->LOAD = tick;

//no prescaler, periodic wrapping mode, disable interrupt, 32-bit timer

TIMER32\_1->CONTROL = 0xc2;

//spin wait until the time is reached

while ((TIMER32\_1->RIS & 1) == 0)

;

TIMER32\_1->INTCLR = 0; //clear raw interrupt flag

}

long measure\_dist()

{

// uses pin 2.6 for trigger

P2->OUT |= BIT6; // generate pulse

Delay(240); // 240 cycle in 24mhz is about 10us

P2->OUT &= ~BIT6; // stop pulse

P2->IFG = 0; // clear P2 interrupt just in case anything happened before

P2->IES &= ~BIT7; // wait for rising edge on ECHO pin

Delay(660000);

if(end > start)

{

return (end - start) / DIST\_DIV; // converting ECHO time into cm

}

else

{

return 0;

}

}

void get\_time\_stamp()

{

sec = intrcnt / 24000;

min = sec / 60;

sec %= 60;

}

/\*\*

\* Port2 interrupt service routine, start timing echo from rising edge to falling edge

\*/

void PORT2\_IRQHandler(void)

{

// check if interrupt is pending for p2.7

if (P2->IFG & BIT7)

{

//if we are checking for raising edge, we are at the beginning of the echo

if (!(P2->IES & BIT7 ))

{

//record starting time of the echo

start = intrcnt \* 1000 + (long) TIMER\_A0->R;

//now checks for falling edge, which is the end of the echo

P2->IES |= BIT7;

}

else

{

end = intrcnt \* 1000 + (long) TIMER\_A0->R; //calculating ECHO length

//now back to checking rising edge, which is the beginning of the echo

P2->IES &= ~BIT7;

}

P2->IFG &= ~BIT7; //clear flag

}

}

/\*\*

\* TimerA0 interrupt service routine, updates count of interrupt and clears interrupt flag

\*/

void TA0\_0\_IRQHandler(void)

{

// Interrupt gets triggered for every 1000 clock cycle in SMCLK

intrcnt++;

TIMER\_A0->CCTL[0] &= ~TIMER\_A\_CCTLN\_CCIFG;

}