This thread has been locked.
If you have a related question, please click the "Ask a related question" button in the top right corner. The newly created question will be automatically linked to this question.

CCS/MSP430FR5969: SSD1306 OLED

Darwin Caina

Prodigy 50 points
Community Member

Part Number: MSP430FR5969

Tool/software: Code Composer Studio

hi everybody,

I am trying to connect the MCU MSP430FR5969 with SSD1306 OLED (by I2C communication). When I debug step by step, it works, but nothing happens when I try to run all in once. Please someone has any idea to guide me?. I am stuck. Many thanks in advance.

```
#include <msp430.h>

const unsigned char Init[] = {0xAE,0x81,0x07,0x20,0x01,0x21,0x00,0x7F,0x22,0x00,0x07,0x40,0xA0,0xA8,0x3F,
                              0xC0,0xD3,0x00,0x8D,0x14,0xDA,0x12,0xD5,0x80,0xD9,0x22,0xDB,0x20,0xA6,0xA4,0x
//const unsigned char Mod[] = {0xA5};

void main(void){
    WDTCTL = WDTPW | WDTHOLD;
    P1SEL1 |= BIT6 + BIT7;
    P1DIR |=BIT0;
    P1OUT &=~BIT0;
    PM5CTL0 &=~LOCKLPM5;

    UCB0CTLW0 |= UCSWRST;
    UCB0CTLW0 |= UCMODE_3 | UCMST | UCSYNC | UCSSEL_2;           // I2C master mode
    UCB0CTLW1 =  UCASTP_2; // Use SMCLK, keep SW reset
    UCB0BR0=0X40;
    UCB0I2CSA = 0x3C;       // address
    UCB0CTL1 &= ~UCSWRST;
    UCB0IE |= UCTXIE;

    while(1){
        __delay_cycles(20);
```

```
        while(UCB0CTL1 & UCTXSTP);
        UCB0CTLW0 |= UCTR | UCTXSTT;
        __bis_SR_register(LPM0_bits | GIE);
    }
}//FIN MAIN
//////////////////////////////////////
#pragma vector = USCI_B0_VECTOR
__interrupt void USCI_B0_ISR(void){
  switch(__even_in_range(UCB0IV, 0X1e))//USCI_I2C_UCBIT9IFG))
  {
    case USCI_NONE:          break;         // Vector 0: No interrupts
    case USCI_I2C_UCALIFG:   break;         // Vector 2: ALIFG
    case USCI_I2C_UCNACKIFG:                // Vector 4: NACKIFG
      UCB0CTL1 |= UCTXSTT;                  // I2C start condition- Resend start if NACK
      break;
    case USCI_I2C_UCSTTIFG:
        break;         // Vector 6: START
    case USCI_I2C_UCSTPIFG:
        UCB0IFG &=~UCSTPIFG;
        break;         // Vector 8: STOP DETECTED
    case USCI_I2C_UCRXIFG3:  break;         // Vector 10: RXIFG3
    case USCI_I2C_UCTXIFG3:  break;         // Vector 12: TXIFG3
    case USCI_I2C_UCRXIFG2:  break;         // Vector 14: RXIFG2
    case USCI_I2C_UCTXIFG2:  break;         // Vector 16: TXIFG2
    case USCI_I2C_UCRXIFG1:  break;         // Vector 18: RXIFG1
    case USCI_I2C_UCTXIFG1:  break;         // Vector 20: TXIFG1
    case USCI_I2C_UCRXIFG0:
        break;        // Vector 22: RXIFG0 MASTER 0
    case USCI_I2C_UCTXIFG0:
        P1OUT ^= BIT0;
        //__delay_cycles(20000);
        UCB0TXBUF = 0x80;
        unsigned int c;
        for(c = 0; c < 32; c++){
                        P1OUT ^= BIT0;
                        //__bis_SR_register(LPM0_bits + GIE);
                        UCB0TXBUF = Init[c];
                        //__bis_SR_register(LPM0_bits + GIE);
                    }
        //UCB0CTL1 |= UCTXSTP;
        //UCB0IFG &=~ UCTXIFG;
        break;         // Vector 24: TXIFG0 MASTER 0

    case USCI_I2C_UCBCNTIFG:                // Vector 26: BCNTIFG
      //P1OUT ^= BIT0;                        // Toggle LED on P1.0
      break;
    case USCI_I2C_UCCLTOIFG: break;         // Vector 28: clock low timeout
    case USCI_I2C_UCBIT9IFG: break;         // Vector 30: 9th bit
    default: break;
```

}

}

Sunny Regards

over 7 years ago

Cameron LaFollette *over 7 years ago*

TI__Genius 14875 points

Hi Darwin,

Where does it get stuck?

Try increasing your delay. Debug timing and free run timing often differ and you may be getting caught in that while loop.

Also, when inserting code, please use the code formatter ;-)

Darwin Caina *over 7 years ago in reply to Cameron LaFollette*

Prodigy 50 points

Thank you for replying,

I tried your suggestion, but It still does not work.
what I do not understand is, after transmiting
case USCI_I2C_UCTXIFG0: ....
what is going on?

Maybe, may you share me an I2C code example that is working, so I can guide me better. Actually I am a beginner with MCU.

Best Regards.

Cameron LaFollette *over 7 years ago in reply to Darwin Caina*

TI__Genius 14875 points

Hi Darwin,
Where are you getting the above code from?

The lines below that case appear to be toggling a GPIO (likely with an LED on it) and then transmitting 0x80 (register address?), and then transmitting the contents of the array at the beginning.

Try uncommenting Line 61 and see if an LED flashes on the launchpad.

Cameron LaFollette *over 7 years ago in reply to Cameron LaFollette*

TI__Genius 14875 points

Are you using pull up resistors? Because this issue sounds like the lines are being held low. If not tie the I2C lines high with 4.7K resistors.

Here is an appnote on i2c

http://www.ti.com/lit/an/slaa734/slaa734.pdf

and working example code which can be found on Resource Explorer at dev.ti.com

```
#include <msp430.h>

const unsigned char TXData[] = { 0xA1, 0xB1, 0xC1, 0xD1 };
const unsigned char SlaveAddress[] = { 0x0A, 0x0B, 0x0C, 0x0D };
volatile unsigned char TXByteCtr;
volatile unsigned char SlaveFlag = 0;

int main(void)
{
  WDTCTL = WDTPW | WDTHOLD;

  // Configure GPIO
  P1SEL1 |= BIT6 | BIT7;                    // I2C pins

  // Disable the GPIO power-on default high-impedance mode to activate
  // previously configured port settings
  PM5CTL0 &= ~LOCKLPM5;

  // Configure USCI_B0 for I2C mode
  UCB0CTLW0 = UCSWRST;                      // put eUSCI_B in reset state
  UCB0CTLW0 |= UCMODE_3 | UCMST | UCSSEL__SMCLK; // I2C master mode, SMCLK
  UCB0BRW = 0x8;                            // baudrate = SMCLK / 8
  UCB0CTLW0 &= ~UCSWRST;                    // clear reset register
  UCB0IE |= UCTXIE0 | UCNACKIE;             // transmit and NACK interrupt enable

  SlaveFlag = 0;                            // Initialize SlaveFlag

  while(1)
  {
    __delay_cycles(1000);                   // Delay between transmissions
    UCB0I2CSA = SlaveAddress[SlaveFlag];    // configure slave address
    TXByteCtr = 1;                          // Load TX byte counter
    while (UCB0CTLW0 & UCTXSTP);            // Ensure stop condition got sent

    UCB0CTLW0 |= UCTR | UCTXSTT;            // I2C TX, start condition

    __bis_SR_register(LPM0_bits | GIE);     // Enter LPM0 w/ interrupts
                                            // Remain in LPM0 until all data
                                            // is TX'd
    // Change Slave address
    SlaveFlag++;
    if (SlaveFlag > 3)                      // Roll over slave address
```

```
    {
      SlaveFlag = 0;
    }
  }
}

#if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
#pragma vector = USCI_B0_VECTOR
__interrupt void USCI_B0_ISR(void)
#elif defined(__GNUC__)
void __attribute__ ((interrupt(USCI_B0_VECTOR))) USCI_B0_ISR (void)
#else
#error Compiler not supported!
#endif
{
  switch(__even_in_range(UCB0IV, USCI_I2C_UCBIT9IFG))
  {
    case USCI_NONE:          break;         // Vector 0: No interrupts
    case USCI_I2C_UCALIFG:   break;         // Vector 2: ALIFG
    case USCI_I2C_UCNACKIFG:                // Vector 4: NACKIFG
      UCB0CTLW0 |= UCTXSTT;                 // resend start if NACK
      break;
    case USCI_I2C_UCSTTIFG:  break;         // Vector 6: STTIFG
    case USCI_I2C_UCSTPIFG:  break;         // Vector 8: STPIFG
    case USCI_I2C_UCRXIFG3:  break;         // Vector 10: RXIFG3
    case USCI_I2C_UCTXIFG3:  break;         // Vector 12: TXIFG3
    case USCI_I2C_UCRXIFG2:  break;         // Vector 14: RXIFG2
    case USCI_I2C_UCTXIFG2:  break;         // Vector 16: TXIFG2
    case USCI_I2C_UCRXIFG1:  break;         // Vector 18: RXIFG1
    case USCI_I2C_UCTXIFG1:  break;         // Vector 20: TXIFG1
    case USCI_I2C_UCRXIFG0:  break;         // Vector 22: RXIFG0
    case USCI_I2C_UCTXIFG0:                 // Vector 24: TXIFG0
      if (TXByteCtr)                        // Check TX byte counter
      {
        UCB0TXBUF = TXData[SlaveFlag];      // Load TX buffer
        TXByteCtr--;                        // Decrement TX byte counter
      }
      else
      {
        UCB0CTLW0 |= UCTXSTP;               // I2C stop condition
        UCB0IFG &= ~UCTXIFG;                // Clear USCI_B0 TX int flag
        __bic_SR_register_on_exit(LPM0_bits); // Exit LPM0
      }
      break;
    default: break;
  }
}
```

Cameron LaFollette *over 7 years ago in reply to Cameron LaFollette* TI__Genius 14875 points

Darwin,
I was able to recreate your issue using your code, and adding the pull-up resistors fixes it. Please try tying P1.6 and P1.7 to VCC with a 4.7K pull up.

sadasivam arumugam *over 7 years ago in reply to Cameron LaFollette* Intellectual 940 points

Dear Sir,

Kindly send us a final code which you have debugged and tested. We want to implement in our system for i2c interface.

sadasivam arumugam *over 7 years ago in reply to Cameron LaFollette* Intellectual 940 points

Hi Sir,
I want to know about the pin configuration for i2c interface from msp430fr5969(master) to ssd1306(slave). (To know, Whether I have to connect DC and Reset pins).

Darwin Caina *over 7 years ago in reply to sadasivam arumugam* Prodigy 50 points

Hi,

the pins are 31 [SDA] y 32 [SCL] from msp430fr5969.

I am still working on that. If you have some advices to my code. Please let me know [see above].

Regards.

sadasivam arumugam *over 7 years ago in reply to Darwin Caina* Intellectual 940 points

Hi sir,

I am using 6pin module(vcc, gnd, rst, dc, scl, sda). Could I have to connect all the pins for working this module or 4 pins are enough(vcc, gnd, sda, scl)

sadasivam arumugam *over 7 years ago in reply to Darwin Caina* Intellectual 940 points

Hi Sir,

Can you please attach the final code you working on this module.

sadasivam arumugam *over 7 years ago in reply to Darwin Caina* Intellectual 940 points

Hi Sir,

I have debugged the user prompted code and I haven't concluded with display module. I am attaching the firmware which i have debugged. And I tried with the pins mentioned above. The Result was not getting. Kindly, let us know the coding is correct. (Also, I have used External pull up resistor).

Below are my code:

```
#include <msp430.h>

const unsigned char Init[] = {0xAE,0x81,0x07,0x20,0x01,0x21,0x00,0x7F,0x22,0x00,0x07,0x40,0xA0,0xA8,0x3F,
0xC0,0xD3,0x00,0x8D,0x14,0xDA,0x12,0xD5,0x80,0xD9,0x22,0xDB,0x20,0xA6,0xA4,0xAF,0xA5};

void main(void)
{
WDTCTL = WDTPW | WDTHOLD;

//for scl and sda lines
P1SEL1 |= BIT6 + BIT7;
P1SEL0 &= ~(BIT6 | BIT7);

// gpio pins configuration
P1DIR |= BIT5; // for enable and disabling reset pin, default- DC pin as gnd
P1OUT &= ~BIT5;

//Disabling power on default high impedance state
PM5CTL0 &= ~LOCKLPM5;

//configuration for i2c mode
UCB0CTLW0 |= UCSWRST;
UCB0CTLW0 |= UCMODE_3 | UCMST | UCSYNC | UCSSEL_2; // I2C master mode
UCB0CTLW1 = UCASTP_2; // Use SMCLK, keep SW reset
UCB0BR0=0x40;
UCB0I2CSA = 0x3C; // slave address for SSD1306, 6-pin module
UCB0CTL1 &= ~UCSWRST;
UCB0IE |= UCTXIE;

while(1)
{
while(UCB0CTL1 & UCTXSTP);
UCB0CTLW0 |= UCTR | UCTXSTT;
__bis_SR_register(LPM0_bits | GIE);
}
}


#pragma vector = USCI_B0_VECTOR
__interrupt void USCI_B0_ISR(void){
switch(__even_in_range(UCB0IV, 0X1e))//USCI_I2C_UCBIT9IFG))
{
case USCI_NONE: break; // Vector 0: No interrupts
case USCI_I2C_UCALIFG: break; // Vector 2: ALIFG
case USCI_I2C_UCNACKIFG: // Vector 4: NACKIFG
UCB0CTL1 |= UCTXSTT; // I2C start condition- Resend start if NACK
break;
case USCI_I2C_UCSTTIFG:
```

```
break; // Vector 6: START
case USCI_I2C_UCSTPIFG:
break; // Vector 8: STOP DETECTED
case USCI_I2C_UCRXIFG3: break; // Vector 10: RXIFG3
case USCI_I2C_UCTXIFG3: break; // Vector 12: TXIFG3
case USCI_I2C_UCRXIFG2: break; // Vector 14: RXIFG2
case USCI_I2C_UCTXIFG2: break; // Vector 16: TXIFG2
case USCI_I2C_UCRXIFG1: break; // Vector 18: RXIFG1
case USCI_I2C_UCTXIFG1: break; // Vector 20: TXIFG1
case USCI_I2C_UCRXIFG0:
break; // Vector 22: RXIFG0 MASTER 0
case USCI_I2C_UCTXIFG0:
UCB0TXBUF = 0x80; // control byte configure for commands
unsigned int c;
for(c = 0; c < 32; c++)
{
P1OUT |= BIT5;
UCB0TXBUF = Init[c];
__bis_SR_register(LPM0_bits + GIE);
}
//automatic assertion of stop bits, after completion of dat transfer
UCB0CTL1 |= UCTXSTP;
UCB0IFG &= ~UCTXIFG;
break; // Vector 24: TXIFG0 MASTER 0
case USCI_I2C_UCBCNTIFG: // Vector 26: BCNTIFG
break;
case USCI_I2C_UCCLTOIFG: break; // Vector 28: clock low timeout
case USCI_I2C_UCBIT9IFG: break; // Vector 30: 9th bit
default: break;
}
}
```

Cameron LaFollette *over 7 years ago in reply to sadasivam arumugam*

TI__Genius 14875 points

Darwin,

Were you ever able to get your i2c working?

sadasivam arumugam *over 7 years ago in reply to Cameron LaFollette*

Intellectual 940 points

Hi Sir,

The I2c Interface is working fine. Now, I want to display any character in this module.(SSD1306). When I tried with normal strings it was not displaying. Is there a different approach to be made using those command table.

Cameron LaFollette *over 7 years ago in reply to sadasivam arumugam*

TI__Genius 14875 points

How are you formatting your strings?

Is the display showing anything at all?

Can you post a screenshot of an i2c transaction?

sadasivam arumugam *over 7 years ago in reply to Cameron LaFollette* Intellectual 940 points

Hi Sir,

Here, the Display is initialized by giving the command mentioned below.(ie, only display is in ON condition). But, We want to show a character in this display module. Is the Graphiics library supports OLED Module. How this library supports OLED Display Module.

Herewith,

```
const unsigned char Init[] = {0xAE,0x81,0x07,0x20,0x01,0x21,0x00,0x7F,0x22,0x00,0x07,0x40,0xA0,0x
                              0xC0,0xD3,0x00,0x8D,0x14,0xDA,0x12,0xD5,0x80,0xD9,0x22,0xDB,0x20,0x
```

Cameron LaFollette *over 7 years ago in reply to sadasivam arumugam* TI__Genius 14875 points

You'll need to write your own driver for the device, but driverlib contains all of the HAL components and fonts. You can see another example of this by downloading the software here:

http://doc.43oh.com/BoosterPack:OLED

It is written for a G2 part, but perhaps you could port it.

Texas Instruments has been making progress possible for decades. We are a global semiconductor company that designs, manufactures, tests and sells analog and embedded processing chips. Our products help our customers efficiently manage power, accurately sense and transmit data and provide the core control or processing in their designs.