F28335 – I2C MODULE

v1.1

Abstract

This document describes how to configure for I2C module of F28335.

Skype/Gmail: phien.ntace

Table of Contents

1. Overview

Step by step configure for I2C module:

- ✓ Configure SCL and SDA pin for I2C module.
- ✓ Configure clock for SCL.
- ✓ Select address bit mode: 7-bit or 10-bit, and data format.
- ✓ Select operation mode: master – transmitter, master – receiver, slave – transmitter, slave – receiver.
- ✓ Configure FIFO module.
- ✓ Select interrupts.

2. Configuration details

2.1 Initial for I2C module

```
/*=====================================================================
 * Description: Initial for I2C
 * Parameters:
 * Return:          NONE
 */
void init_I2C(void)
{
	EALLOW;

	/* Enable internal pull-up for the selected pins */
	// Pull-ups can be enabled or disabled disabled by the user.

	GpioCtrlRegs.GPBPUD.bit.GPIO32 = 1;    // Disable pull-up for GPIO32 (SDAA)
	GpioCtrlRegs.GPBPUD.bit.GPIO33 = 1;    // Disable pull-up for GPIO33 (SCLA)

	/* Set qualification for selected pins to asynch only */
	// This will select asynch (no qualification) for the selected pins.

	GpioCtrlRegs.GPBQSEL1.bit.GPIO32 = 3;  // Asynch input GPIO32 (SDAA)
	GpioCtrlRegs.GPBQSEL1.bit.GPIO33 = 3;  // Asynch input GPIO33 (SCLA)

	/* Configure SCI pins using GPIO regs */
	// This specifies which of the possible GPIO pins will be I2C functional pins.

	GpioCtrlRegs.GPBMUX1.bit.GPIO32 = 1;   // Configure GPIO32 for SDAA operation
	GpioCtrlRegs.GPBMUX1.bit.GPIO33 = 1;   // Configure GPIO33 for SCLA operation

	EDIS;
```

```
	I2caRegs.I2CPSC.all = 14;   // Prescaler need 7-12 Mhz on module clk (150/15 =
10MHz)

	// Configure SCL clock
	I2caRegs.I2CCLKL = 12;           // NOTE: must be non zero
	I2caRegs.I2CCLKH = 7;            // NOTE: must be non zero
	I2caRegs.I2CIER.all = 0x24;      // Enable SCD & ARDY interrupts

	I2caRegs.I2CMDR.all = 0x0020;    // Take I2C out of reset
	                                 // Stop I2C when suspended

	I2caRegs.I2CFFTX.all = 0x6000;   // Enable FIFO mode and TXFIFO
	I2caRegs.I2CFFRX.all = 0x2040;   // Enable RXFIFO, clear RXFFINT



	// Interrupts that are used in this example are re-mapped to
	// ISR functions found within this file.
	EALLOW; // This is needed to write to EALLOW protected registers
	PieVectTable.I2CINT1A    = &i2c_rxa_isr;
	EDIS;   // This is needed to disable write to EALLOW protected registers


	PieCtrlRegs.PIECTRL.bit.ENPIE = 1;

	// Enable I2C interrupt 1  in the PIE: Group 8
	PieCtrlRegs.PIEIER8.bit.INTx1 = 1;

	// Enable CPU INT8 which is connected to PIE group 8
	IER |= M_INT8;

}
```

2.2 Write function

```
/*=====================================================================
 * Description: Send data via I2C bus
 * Parameters:
 * Return:          NONE
 * ---------------------------------------------------------------------
 * Note: Data will be send store in i2c_write_buff[]
 */
void I2C_write_data(unsigned int slave_addr, unsigned int length)
{

	unsigned int jdex;


	while(I2caRegs.I2CMDR.bit.STP == 1);

	// Check if bus busy
	while(I2caRegs.I2CSTR.bit.BB == 1);
```

```
	// Setup slave address
	I2caRegs.I2CSAR = slave_addr;

	// Setup number of bytes to send
	// MsgBuffer + Address
	I2caRegs.I2CCNT = length;

	// Setup data to send
	for (jdex=0; jdex<length; jdex++)
	{
		I2caRegs.I2CDXR = *(i2c_write_buff+jdex);
	}

	// Send start as master transmitter
	I2caRegs.I2CMDR.all = 0x2E20;
}
```

2.3 Read function

```
/*=================================================================
 * Description: Send data via I2C bus
 * Parameters:
 * Return:          NONE
 * ----------------------------------------------------------------
 * Note: Data read from I2C will be stored in i2c_read_buff[]
 */
void I2C_read_data(unsigned int slave_addr, unsigned int cmd_length)
{

	unsigned int jdex;


	// Wait until the STP bit is cleared from any previous master communication.
	// Clearing of this bit by the module is delayed until after the SCD bit is
	// set. If this bit is not checked prior to initiating a new message, the
	// I2C could get confused.
	while(I2caRegs.I2CMDR.bit.STP == 1);

	// Check if bus busy
	while(I2caRegs.I2CSTR.bit.BB == 1);

	I2caRegs.I2CSAR = slave_addr;

	I2caRegs.I2CCNT = cmd_length;

	if(cmd_length != 0)
	{
		// Setup data to send
		for (jdex=0; jdex<cmd_length; jdex++)
```

```
            {
                  I2caRegs.I2CDXR = i2c_cmd[jdex];
            }

            I2caRegs.I2CMDR.all = 0x2620;    // Send cmd with no stop
            user_delay(10000);               // Wait I2C send
      }


      I2caRegs.I2CCNT = i2c_receive_length;  // Setup how many bytes to expect
      I2caRegs.I2CMDR.all = 0x2C20;          // Send restart as master receiver
}
```

2.4 Interrupt receive

```
/*=====================================================================
 * Description: Receive interrupt for I2C-A
 * Parameters:      NONE
 * Return:          NONE
 */
interrupt void i2c_rxa_isr(void)
{

      unsigned int i2c_dex;

      // Read I2C interrupt source
      i2c_int_source = I2caRegs.I2CISRC.all;

      // When i2c module is already receive enough data
      if(i2c_int_source == I2C_SCD_ISRC)
      {
            for(i2c_dex=0; i2c_dex<i2c_receive_length; i2c_dex++)
            {
                  i2c_read_buff[i2c_dex] = I2caRegs.I2CDRR;
            }
      }

      // When i2c module receive a NACK
      else if(i2c_int_source == I2C_ARDY_ISRC)
      {
            if(I2caRegs.I2CSTR.bit.NACK == 1)
            {

                  I2caRegs.I2CMDR.bit.STP = 1;                  // Send a STOP
                  I2caRegs.I2CSTR.all = I2C_CLR_NACK_BIT;       // Clear NACK flag
            }
      }


      // Enable future I2C (PIE Group 8) interrupts
      PieCtrlRegs.PIEACK.all |= PIEACK_GROUP8;
}
```

3. Reference

[1] **TMS320x2833x, 2823x Inter-Integrated Circuit (I2C) Module**

Revision

Revision	Date	Author	Description
1.1	Oct 21, 2014	Phien Nguyen Thanh	- Modify section 2