I2C

GPIO configuration:

1. Enable the clock to the appropriate GPIO module via the RCGCGPIO register in the System
   Control module (see page 340). To find out which GPIO port to enable, refer to Table
   23-5 on page 1351.

PA6	23	-	-	-	I2C1SCL	-	M1PWM2	-	-	-	-	-	-
PA7	24	-	-	-	I2C1SDA	-	M1PWM3	-	-	-	-	-	-

PB2	47	-	-	-	I2C0SCL	-	-	-	T3CCP0	-	-	-	-
PB3	48	-	-	-	I2C0SDA	-	-	-	T3CCP1	-	-	-	-

PE4	59	AIN9	U5Rx	-	I2C2SCL	M0PWM4	M1PWM2	-	-	CAN0Rx	-	-	-
PE5	60	AIN8	U5Tx	-	I2C2SDA	M0PWM5	M1PWM3	-	-	CAN0Tx	-	-	-

PD0	61	AIN7	SSI3Clk	SSI1Clk	I2C3SCL	M0PWM6	M1PWM0	-	WT2CCP0	-	-	-	-
PD1	62	AIN6	SSI3Fss	SSI1Fss	I2C3SDA	M0PWM7	M1PWM1	-	WT2CCP1	-	-	-	-

2. In the GPIO module, enable the appropriate pins for their alternate function using the
   GPIOAFSEL register (see page 671). To determine which GPIOs to configure, see Table
   23-4 on page 1344
3. Enable the I2CSDA pin for open-drain operation. See page 676.
4. Configure the PMCn fields in the GPIOPCTL register to assign the I2C signals to the appropriate
   pins. See page 688 and Table 23-5 on page 1351.

Configuration for I2C as a master:

The following example shows how to configure the I2C module to transmit a single byte as a master.
This assumes the system clock is 20 MHz.

1. Enable the I2C clock using the RCGCI2C register in the System Control module (see page 348).
2. Initialize the I2C Master by writing the I2CMCR register with a value of 0x0000.0010.
3. Set the desired SCL clock speed of 100 Kbps by writing the I2CMTPR register with the correct
   value. The value written to the I2CMTPR register represents the number of system clock periods
   in one SCL clock period. The TPR value is determined by the following equation:
   TPR = (System Clock/(2*(SCL_LP + SCL_HP)*SCL_CLK))-1;*
   TPR = (20MHz/(2(6+4)*100000))-1;
   TPR = 9
   Write the I2CMTPR register with the value 9
4. Specify the slave address of the master and that the next operation is a Transmit by writing the
5. Place data (byte) to be transmitted in the data register by writing the I2CMDR register with the
   desired data.
6. Initiate a single byte transmit of the data from Master to Slave by writing the I2CMCS register
   with a value of 0x07 (STOP, START, RUN)., if we will send multiple bytes then 0x03.

   (((The STOP bit determines if the cycle stops at the end of the data cycle or continues on to a repeated START condition.)))

   —> To use Master receiver mode:

   When the I2C module operates in Master receiver mode, the ACK bit is normally set causing the I2C bus controller to transmit an acknowledge automatically after each byte. This bit must be cleared when the I2C bus controller requires no further data to be transmitted from the slave transmitter.

When operating in slave mode, the STARTRIS and STOPRIS bits in the I2C Slave Raw Interrupt
Status (I2CSRIS) register indicate detection of start and stop conditions on the bus and the I2C
Slave Masked Interrupt Status (I2CSMIS) register can be configured to allow STARTRIS and
STOPRIS to be promoted to controller interrupts (when interrupts are enabled

7. Wait until the transmission completes by polling the I2CMCS register's BUSBSY bit until it has
   been cleared. // we can do that by an interrupt
8. Check the ERROR bit in the I2CMCS register to confirm the transmit was acknowledged

Interrupts

Master interrupts: note that we can do these things by polling on flags in the I2CMCS register.

If the application doesn't require the use of interrupts, the raw interrupt status is always visible via
the I2C Master Raw Interrupt Status (I2CMRIS) register.

- The I2C master module generates an interrupt when a transaction completes (either transmit or
  receive), when arbitration is lost, or when an error occurs during a transaction.
- To enable the I2C master interrupt, software must set the IM bit in the I2C Master Interrupt Mask (I2CMIMR) register. When an interrupt condition is met. —> to enable the peripheral interrupt
- When an interrupt condition is met, software must check the ERROR and ARBLST bits in the I2C
  Master Control/Status (I2CMCS) register to verify that an error didn't occur during the last transaction and to ensure that arbitration has not been lost
- An error condition is asserted if the last transaction wasn't acknowledged by the slave. If an error is not detected and the master has not lost arbitration, the application can proceed with the transfer.

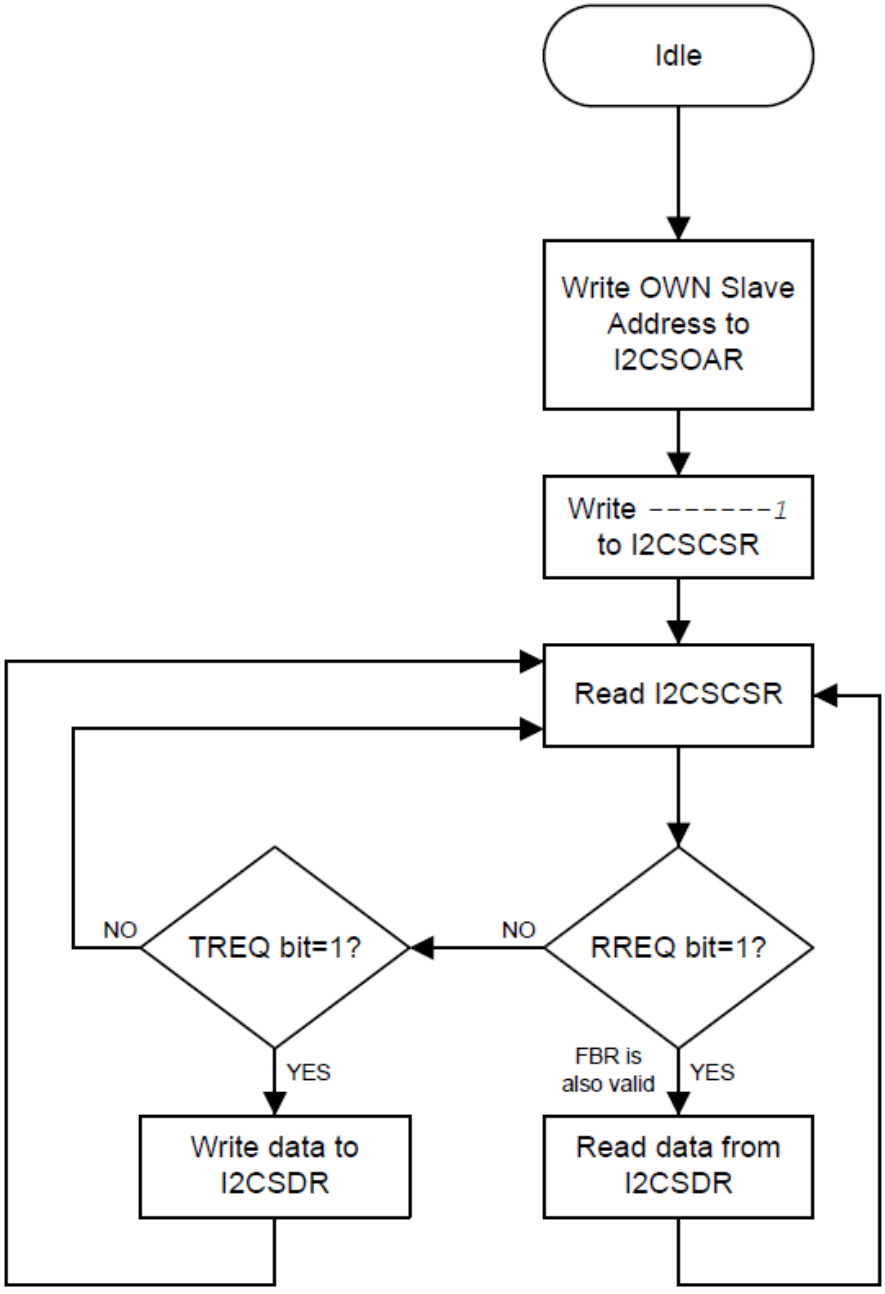
- The interrupt is cleared by writing a 1 to the IC bit in the I2C Master Interrupt Clear (I2CMICR) register.

—> I2C Master Masked Interrupt Status (I2CMMIS) register is used to check on whether an interrupt is signaled (for debugging)

Configuration for I2C as a slave:

1. Enable the I2C clock using the RCGCI2C register in the System Control module (see page 348).

Figure 16-15. Slave Command Sequence



- instead of checking on the RREQ bit, we can use interrupts.

Interrupts:

- The slave module can generate an interrupt when data has been received or requested

This interrupt is enabled by setting the DATAIM bit in the I2C Slave Interrupt Mask (I2CSIMR) register

- Software determines whether the module should write (transmit) or read (receive) data from the I2C Slave Data (I2CSDR) register, by checking the RREQ and TREQ bits of the I2C Slave Control/Status (I2CSCSR) register.
- If the slave module is in receive mode and the first byte of a transfer is received, the FBR bit is set along with the RREQ bit.
- The interrupt is cleared by setting the DATAIC bit in the I2C Slave Interrupt Clear (I2CSICR) register.

—> I2C Slave Masked Interrupt Status (I2CSMIS) register is used to check on whether an interrupt is signaled (for debugging)