Inter Integrated Communication I2C

TM4C123GH6PM Launchpad

Dr. Munesh Singh

Indian Institute of Information Technology Design and Manufacturing
Kancheepuram, Chennai, Tamil Nadu

March 9, 2020

Introduction to I2C

- The inter-integrated circuit I2C interface was proposed by Philips in the late 1980s
- To connect external devices to the micro controller using just two wires
- I2C is ideal to attach low-speed peripherals
- I2C provides a connection oriented communication with acknowledge
- The interface will operate at baud rates of up to 100 kbps with maximum capacitive bus loading
- The maximum interconnect length and the number of devices that can be connected to the bus are limited by a maximum bus capacitance of 400pF in all instances.

I2C Communication Port

What is I2C

- Inter-integrated Circuit
- Bidirectional Data Transfer
- Half Duplex (have only one data-line)
- Synchronous bus so data is clocked with clock signal
- Clock is controlled when data line is changed

Speed of I2C

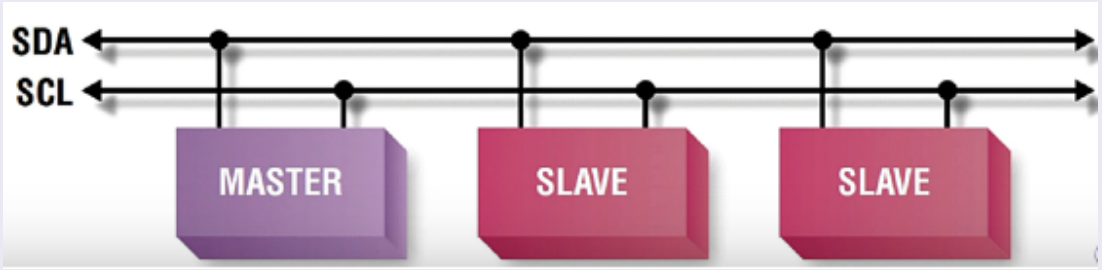
- Low (under 100 kbps)
- Fast (400 Kbps)
- High speed (3.4 mbps) (I2C V2.0)
- 2 Wire communication:
- SDA and SCL

I2C Communication Signal

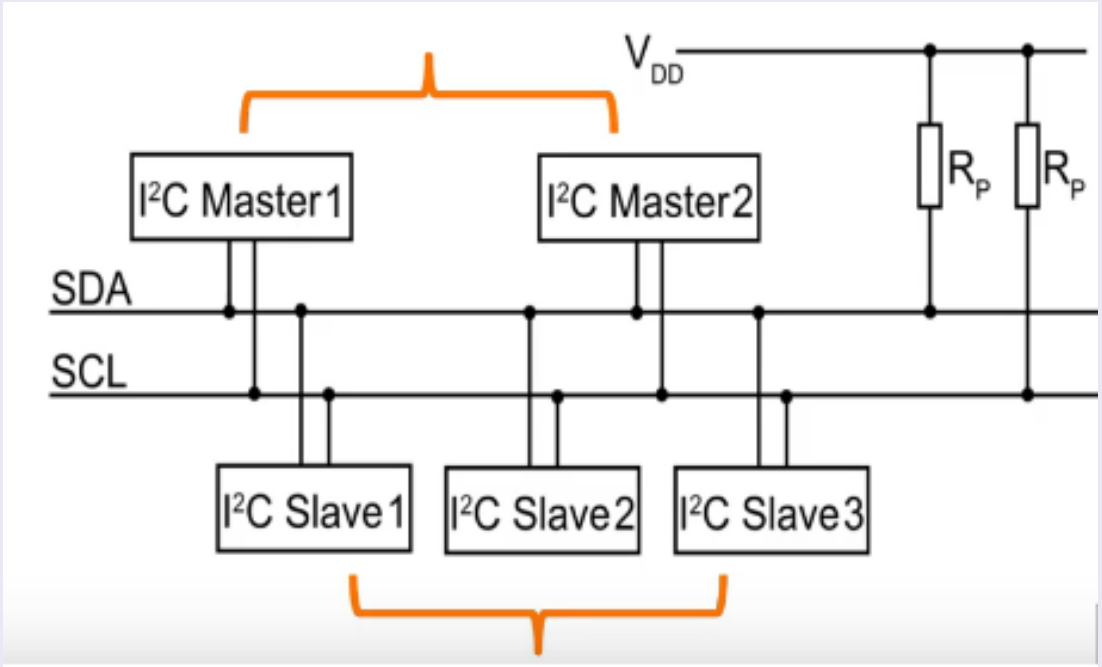
- The Serial Clock Line (SCL) and the Serial Data line (SDA) are both bi-directional.
- Each line is open-drain, meaning a device may drive it low or let it float.
- A logic high occurs if all devices let the output float, and a logic low occurs when at least one device drives it low.
- The value of the pull-up resistor depends on the speed of the bus.
  - 4k7 is recommended for baud rates below 100 kbps
  - 2k2 is recommended for standard mode
  - and 1k is recommended for fast mode.

I2C Communication Port

What is I2C



Basic protocol is master slave protocol



I2C Communication Signal

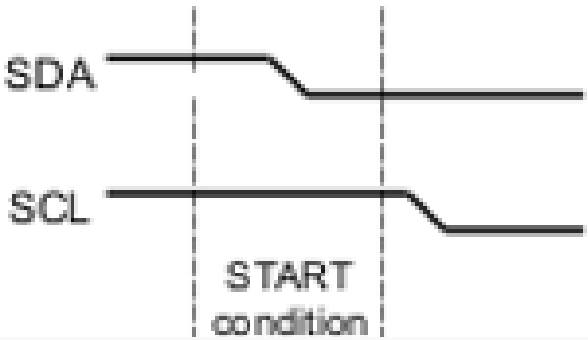
- The SCL clock is used in a synchronous fashion to communicate on the bus.
- Even though data transfer is always initiated by a master device, both the master and the slaves have control over the data rate.
- The master starts a transmission by driving the clock low, but if a slave wishes to slow down the transfer, it too can drive the clock low (called clock stretching).
- In this way, devices on the bus will wait for all devices to finish.
- Both address (from Master to Slaves) and information (bi-directional) are communicated in serial fashion on SDA.

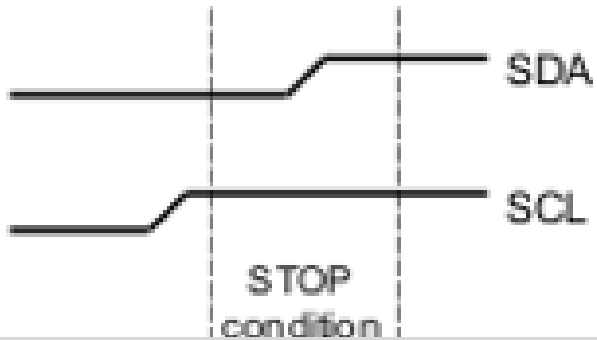
I2C Nodes

- In I2C protocol, more than 100 devices can share an I2C bus.
- Each node can operate as either master or slave.
- Master is a device that generate the Clock for the system, it also initiate and terminate a transmission.
- Slave is node that receives the clock and is addressed by the master.
- In I2C, both master and slave can receive or transmit data.
  - Master Transmitter
  - Master Receiver
  - Slave Transmitter
  - Slave Receiver

START and STOP conditions

- I2C is a connection oriented communication protocol
- it means that each transmission is initiated by a START condition and is terminated by STOP condition.
- Remember that the START and STOP conditions are generated by the master.
  - A High-to-Low transition on the SDA line while the SCL is High is defined as a START condition
  - Low-to-High transition on the SDA line while SCL is High is defined as a STOP condition
- The bus is considered busy after a START condition and free after a STOP condition.

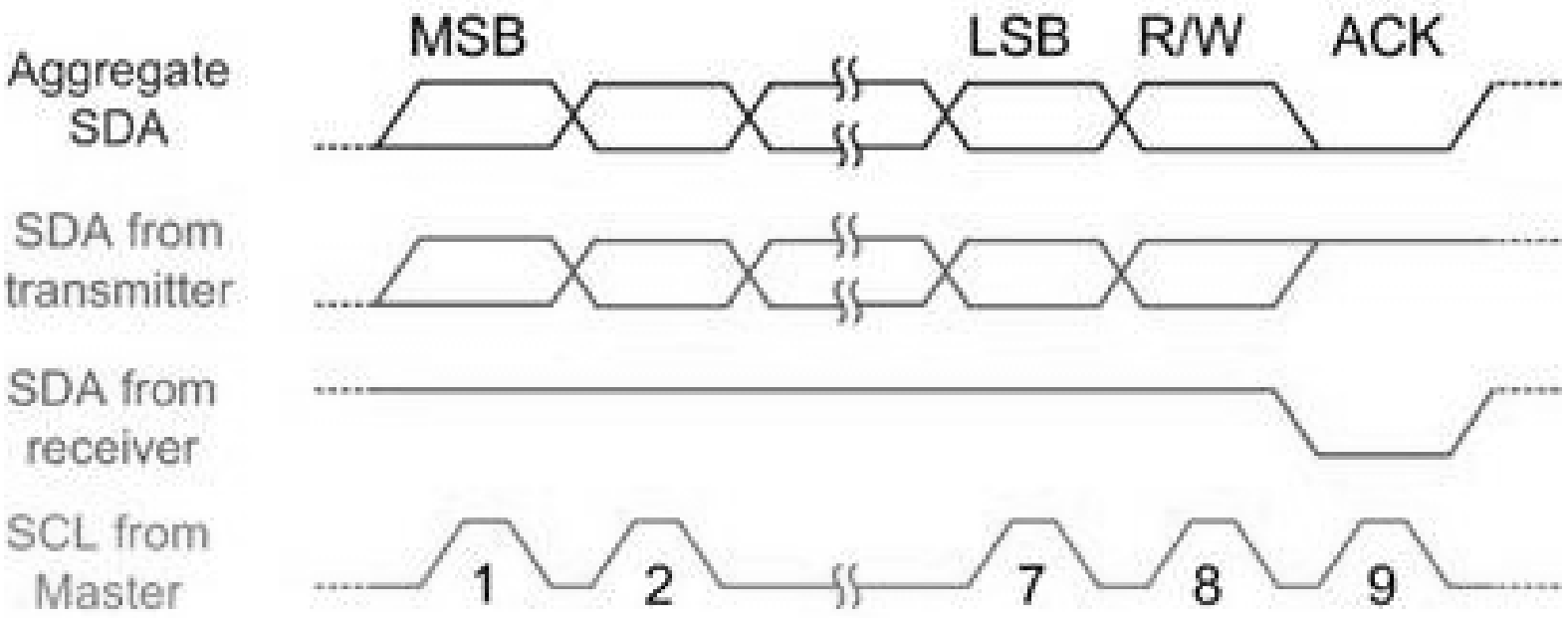




Message format in I2C

- In I2C, each address or data to be transmitted must be framed in 9-bit long.
- The first 8-bits are put on SDA line by the transmitter and the 9th bit is the acknowledge (ACK) by the receiver or it may be NACK (negative acknowledge).
- Notice that the clock is always generated by the master, regardless of it being transmitter or receiver.
- To allow acknowledge, the transmitter releases the SDA line during the 9th clock so the receiver can pull the SDA line low to indicate an ACK.
- If the receiver doesnt pull the SDA line low, it is considered as NACK.

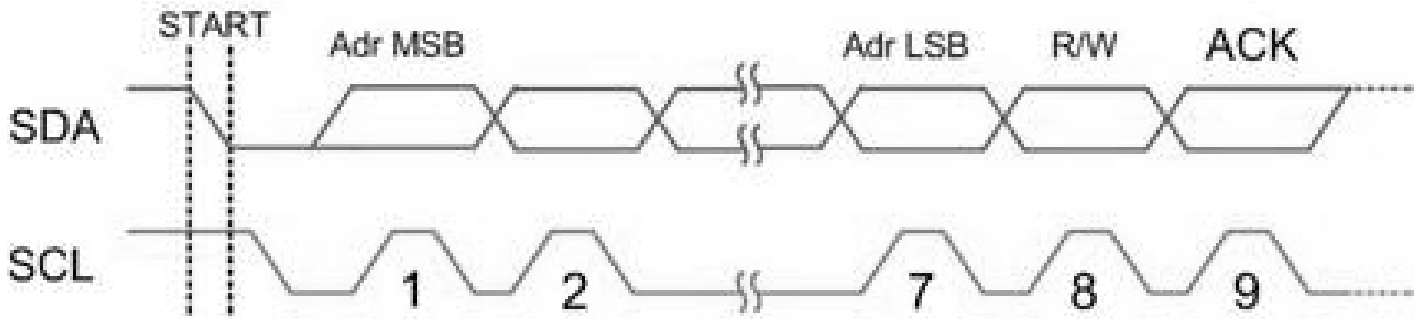
Message format in I2C



- In I2C, each byte may contain either address or data.
  START condition + slave address byte + one or more data byte + STOP condition

Address Byte Format

- All address bytes transmitted on the I2C bus are nine bits long.
- It consists of seven address bits, one READ/WRITE control bit and an acknowledge bit.
- Slave address bits are used to address a specific slave device on the bus
- 7 bit address let the master to address maximum of 128 slaves on the bus.
- Although address 0000 000 is reserved for general call and all address of the format 1111 xxx are reserved in many devices.
- That means 119 devices can share an I2C bus.



Address Byte Format

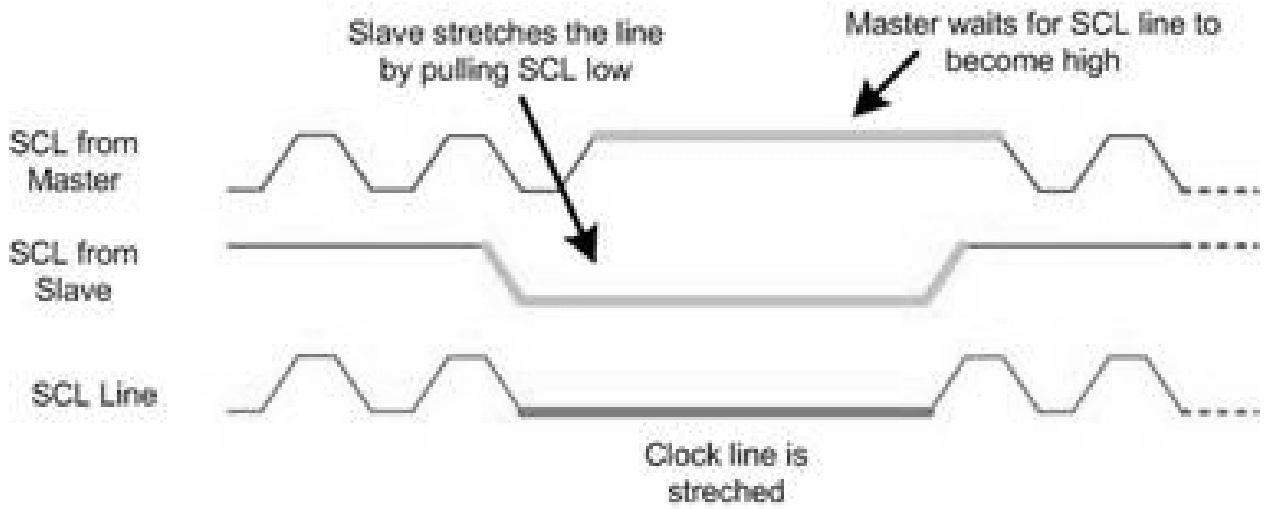
- In I2C bus the MSB of the address is transmitted first.
- The 8th bit in the byte is READ/WRITE control bit.
- If this bit is set, the master will read the next byte from the slave, otherwise, the master will write the next byte on the bus to the slave.
- When a slave detects its address on the bus, it knows that it is being addressed and it should acknowledge in the ninth clock cycle by pulling SDA to low.
- If the addressed slave is not ready or for any reason does not want to respond to the master, it should leave the SDA line high in the 9th clock cycle.
- In case of NACK, the master can transmit a STOP condition to terminate the transmission, or a REPEATED START condition to initiate a new transmission.

Data Byte Format

- Like other bytes, data bytes are 9 bits long too.
- The first 8 bits are a byte of data to be transmitted and the 9th bit, is ACK.
- If the receiver has received the last byte of data and does not wish to receive more data, it may signal a NACK by leaving the SDA line high.
- The master should terminate the transmission with a STOP after a NACK appears.
- In data bytes, like address bytes, MSB is transmitted first.
- Combining Address and Data Bytes: In I2C, normally, a transmission is started by a START condition, followed by an address byte (SLA+R/W), one or more data bytes and finished by a STOP condition.

Clock stretching

- One of the features of the I2C protocol is clock stretching.
- It is a kind of flow control.
- If an addressed slave device is not ready to process more data it will stretch the clock by holding the clock line (SCL) low after receiving (or sending) a bit of data
- Master will wait until the slave releases the SCL line to show it is ready for the next bit.



Arbitration

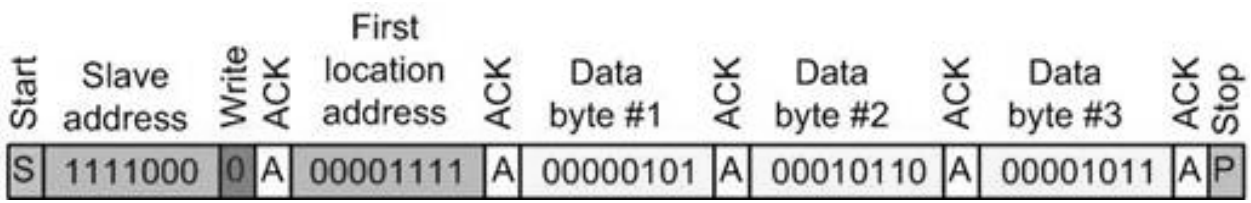
- I2C protocol supports multi-master bus system.
- It doesnt mean that more than one master can use the bus at the same time.
- Each master waits for the current transmission to finish and then start to use the bus.
- But it is possible that two or more masters initiate a transmission at about the same time called arbitration.
- Each master has to check the level of the bus and compare it with the levels it is driving:
  - if it doesn't match, that master has lost the arbitration, and will switches to slave mode.
  - In the case of arbitration, the winning master will continue its job.

Multi-byte burst write

- Burst mode writing is an effective means of loading data into consecutive memory locations.
- It is supported in I2C like SPI and many other serial protocols.
- In burst mode, we provide the address of the first memory location, followed by the data for that location.
- From then on, consecutive bytes are written to consecutive memory locations.
- In this mode, the I2C device internally increments the address location as long as STOP condition is not detected

Multi-byte burst write

- The following steps are used to send (write) multiple bytes of data in burst mode for I2C devices:
  - The master generates a START condition.
  - The master transmits the slave address followed by a zero bit (for write).
  - The master transmits the address of the first location.
  - The master transmits the data for the first location and from then on, the master simply provides consecutive bytes of data to be placed in consecutive memory locations in the slave.
  - The master generates a STOP condition.
- The following figure shows how to write 0x05, 0x16 and 0x0B to 3 consecutive locations starting from location 00001111 of slave 1111000.

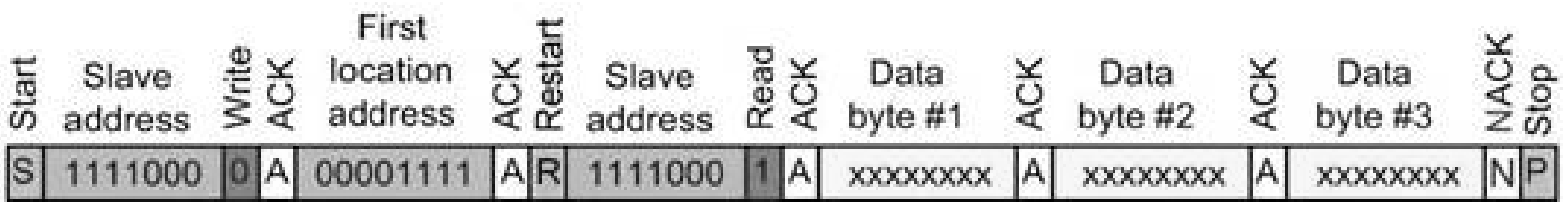


Multi-byte burst read

- The following steps are used to get (read) multiple bytes of data using burst mode for I2C devices.
  - The master generates a START condition.
  - The master transmits the slave address followed by a zero bit (for writing the address).
  - The master transmits the address of the first memory location.
  - The master generates a RESTART condition to switch the bus direction from write to read.
  - The master transmits the slave address followed by a one bit (for read).
  - The master clocks the bus 8 times and the slave device provides the data for the first location.
  - The master provides an ACK.
  - The master reads the consecutive locations and provides an ACK for each byte.
  - The master gives a NACK for the last byte received to signal the slave that the read is complete.
  - The master generates a STOP condition.

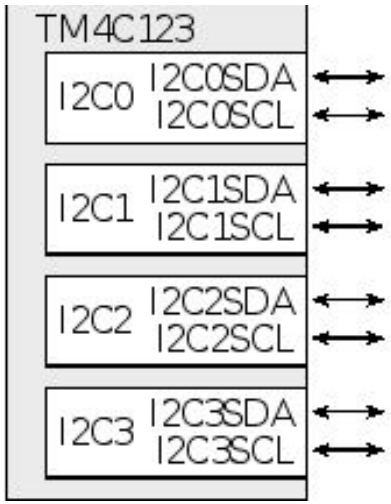
Multi-byte burst read

- The following figure shows how to read three consecutive locations starting from location 00001111 of slave number 1111000.



I2C Programming in TI ARM Tiva

- The TI ARM Tiva micro controllers have zero to three I2C modules.
- Microcontroller pins SDA and SCL can be connected directly to an I2C network
- The I2C modules are located at the following base addresses:
  - I2C0 - 0x4002.0000
  - I2C1 - 0x4002.1000
  - I2C2 - 0x4002.2000
  - I2C3 - 0x4002.3000



Enabling Clock to I2C Module

- To enable and use any of the peripherals, we must enable the clock to it.
- We use RCGCI2C register to enable the clock to I2C modules.
- Notice again RCGCI2C is part of the SYSCTL registers.
- We need SYSCTL_RCGCI2C_R |= 0x0F, enables the clock to all four I2C modules

Inter-Integrated Circuit Run Mode Clock Gating Control (RCGCI2C)

Base 0x400F.E000
Offset 0x620
Type RW, reset 0x0000.0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved												R3	R2	R1	R0
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

I2C Clock speed

- The I2CMTPR (I2C Master Timer Period) register allows us to set the clock rate for the SCL.
- The SCL clock comes from the system clock and goes through clock divider circuit controlled by I2CMTPR register.
- The I2C standard defines four clock speeds for the SCL
  - Standard mode of 100Kbps (bits per second),
  - Fast mode of 400Kbps,
  - Fast Plus mode of 1Mbps. and there is also a
  - High-Speed (HS) mode which can be as high as 3.3Mbps.
- The lower 7 bits (D6-D0, TPR) of I2CMTPR register are used to set the I2C clock speed.

I2C Master Timer Period (I2CMTPR)

I2C 0 base: 0x4002.0000
I2C 1 base: 0x4002.1000
I2C 2 base: 0x4002.2000
I2C 3 base: 0x4002.3000
Offset 0x00C
Type RW, reset 0x0000.0001

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

I2C Clock speed

- The lower 7 bits (D6-D0, TPR) of I2CMTPR register are used to set the I2C clock speed.
- We use the following formula to set the I2C clock speed:
  $SCL\_PERIOD = 2(1 + TPR)(SCL\_LP + SCL\_HP)CLK\_PRD$
- In the above formula, SCL_PERIOD is the I2C clock period, CLK_PRD is the system clock period.
  - The SCL_LP is the SCL Low Period and is fixed at 6.
  - The SCL_HP is the SCL High Period and is fixed at 4.
- $SCL\_PERIOD = 2(1 + TPR)(4 + 6)CLK\_PRD$
  $SCL\_PERIOD = 2(1 + TPR)10CLK\_PRD$
  $SCL\_PERIOD = (20(1 + TPR))/System\_Clock\_Freq$
- Now, solving for TPR (the value we need for the I2CMTPR register) we have:
- $TPR = ((System\_Clock\_Freq \times SCL\_PERIOD) / 20) - 1$
  $TPR = (System\_Clock\_Freq) / (20 \times I2C\ Clock) - 1$

I2C Clock speed

- **Example:** Assume the system clock frequency is 16MHz. Find the values for the I2CMTPR register if we want I2C clock of (a) 100 Kbps, (b) 400 Kbps, and (c) 1 Mbps.

I2C Master Timer Period (I2CMTPR)

I2C 0 base: 0x4002.0000
I2C 1 base: 0x4002.1000
I2C 2 base: 0x4002.2000
I2C 3 base: 0x4002.3000
Offset 0x00C
Type RW, reset 0x0000.0001

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

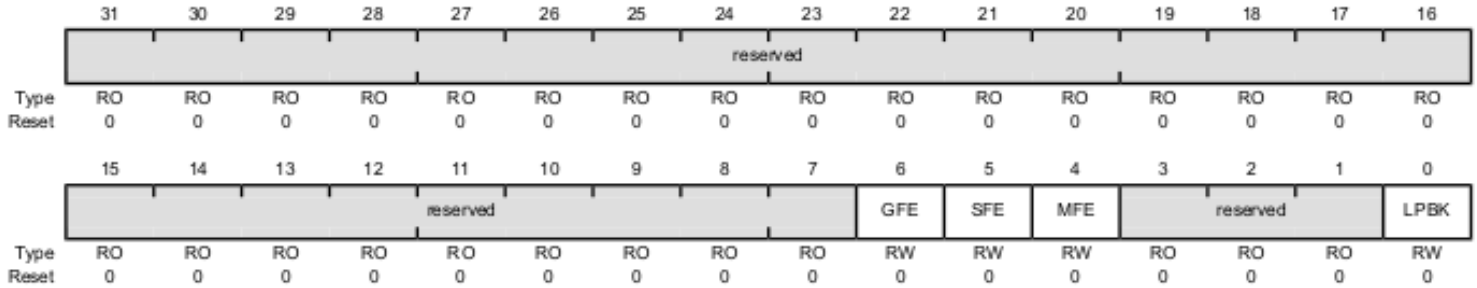
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								HS	TPR						
Type	RO	RO	RO	RO	RO	RO	RO	RO	WO	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Master or Slave?

- The I2C Module inside the ARM chip can be Master or Slave.
- We use I2CMCR (I2C Master Configuration register) to designate the ARM chip as master or slave.
- Setting bit D4 to 1 makes the I2C of ARM chip as Master.

I2C Master Configuration (I2CMCR)

I2C 0 base: 0x4002.0000
I2C 1 base: 0x4002.1000
I2C 2 base: 0x4002.2000
I2C 3 base: 0x4002.3000
Offset 0x020
Type RW, reset 0x0000.0000



Bits	Name	Function	Description
0	LPBK	I²C Loopback	0: Normal operation, 1: Loopback
4	MFE	I²C Master Function Enable	1: Enable Master Function, 0: Disable
5	SFE	I²C Slave Function Enable	1: Enable Slave Function, 0: Disable
6	GFE	I²C Glitch Filter Enable	1: Enable Glitch Filter, 0: Disable

Slave Address

- To transmit a byte of data to an I2C device, we must specify its I2C address.
- We use I2CMSA (I2C Master Slave Address) register to hold the address of the slave device
- In the I2CMSA register, the D7-D1 bits are used for the slave address and the LSB of D0 is used to indicate if we are transmitting to slave device or receiving from a slave device.

I2C Master Slave Address (I2CMSA)

I2C 0 base: 0x4002.0000
I2C 1 base: 0x4002.1000
I2C 2 base: 0x4002.2000
I2C 3 base: 0x4002.3000
Offset 0x000
Type RW, reset 0x0000.0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved								SA							R/S
Type	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Data Register

- In Master transmit mode, we place a byte of data in I2CMDR (I2C Master Data register) for transmission.
- Only the lower 8 bits of this register is used.

I2C Master Data (I2CMDR)

I2C 0 base: 0x4002.0000
I2C 1 base: 0x4002.1000
I2C 2 base: 0x4002.2000
I2C 3 base: 0x4002.3000
Offset 0x008
Type RW, reset 0x0000.0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								DATA							
Type	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

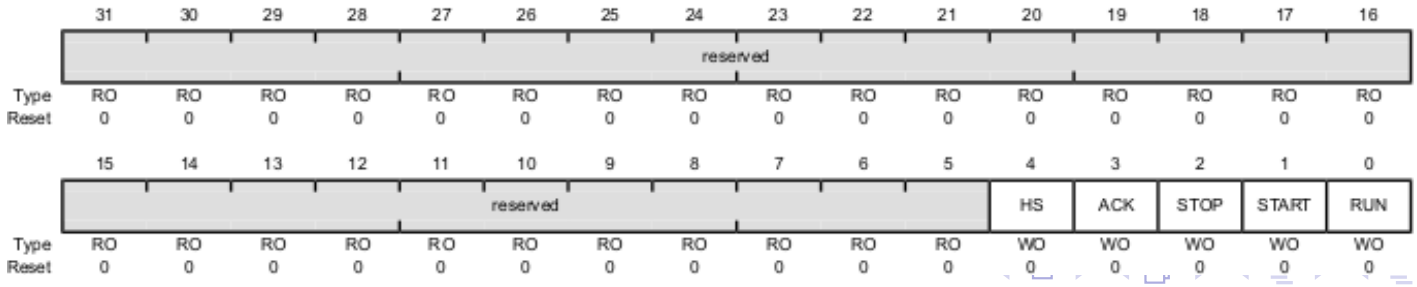
Control and Status Flag Register

- We have two registers with the same name.
- One we write to control the I2C module, the other one we read from to get the status of the I2C module.
- We use the I2CMCS (I2C Master Control/Status) register for both control and status
- Upon reading it, we get the status to see if a byte has been transmitted and transmission buffer is empty and ready for the next byte.

Write-Only Control Register

I2C Master Control/Status (I2CMCS)

I2C 0 base: 0x4002.0000
I2C 1 base: 0x4002.1000
I2C 2 base: 0x4002.2000
I2C 3 base: 0x4002.3000
Offset 0x004
Type WO, reset 0x0000.0020



Control and Status Flag Register

- After placing a byte of data in I2C Data register and the slave address in I2C Master Slave address register
- We may write a value of 0x07 to I2CMCS register for the I2C to start a single byte data transmission from Master (microcontroller) to slave device.
- Notice, writing 0x07 to this register has all the three of STOP = 1, RUN = 1, and START = 1 in it.
  - generate a START condition,
  - send the slave address with R/W bit,
  - check the acknowledge bit,
  - send the data byte,
  - check the acknowledge bit and
  - generate the STOP condition to terminate the transmission.

Control and Status Flag Register

Read-Only Status Register

I2C Master Control/Status (I2CMCS)

I2C 0 base: 0x4002.0000
I2C 1 base: 0x4002.1000
I2C 2 base: 0x4002.2000
I2C 3 base: 0x4002.3000
Offset 0x004
Type RO, reset 0x0000.0020

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								CLKTO	BUSBSY	IDLE	ARBLST	DATACK	ADRACK	ERROR	BUSY
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

Control and Status Flag Register

Bits	Name	Function	Description
0	BUSY	I2C Busy	0: The controller is idle 1: The controller is busy.
1	ERROR	Error	0: No error was detected on the last operation. 1: An error occurred on the last operation.
2	ADRACK	Acknowledge Address	0: The transmitted address was acknowledged 1: The transmitted address was not acknowledged.
3	DATACK	Acknowledge Data	0: The transmitted data was acknowledged 1: The transmitted data was not acknowledged.
4	ARBLST	Arbitration Lost	0: The I2C controller won arbitration 1: The I2C controller lost arbitration.
5	IDLE	I2C Idle	0: The I2C controller is not idle. 1: The I2C controller is idle.
6	BUSBSY	Bus Busy	0: The I2C bus is idle 1: The I2C bus is busy.
7	CLKTO	Clock Timeout Error	0: No clock timeout error 1: The clock timeout error has occurred.

Control and Status Flag Register

- For a single byte write, after the START, the bus will go busy until the data byte is written and the STOP is sent.
- To monitor the progress of the transmission, we need to check the BUSBSY bit (bit 6) of the I2CMCS register. When this bit goes low, the transmission is complete.
- When the transmission is complete, the program should check the ERROR bit (bit 1) to make sure there was no error in the transmission.
- Transmission error may be caused by either slave address was not acknowledged or the data write was not acknowledged.
- In either case, the ADRACK (bit 2) or the DATACK (bit 3) will be set.

- In a system with multiple I2C masters, the program should check the BUSBSY (bit 6) of I2CMCS register to be sure that the bus is idle before starting a transmission.
- After the transmission is started, the program should check the ARBLST (bit 4) to see whether it lost the arbitration or not.
- If the arbitration is lost, the program should attempt the transmission after the bus is not busy anymore.

Enabling Open Drain

- For the I2C Module inside the TI ARM Tiva, we must enable the open-drain option for the I/O pins used by the I2C buses.
- We use the GPIOODR (GPIO Open Drain) register to enable the open drain.
- Depending on which pins we use for the I2C connection, we have to enable the GPIOODR register for that port.

GPIO Open Drain Select (GPIOODR)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
GPIO Port F (APB) base: 0x4002.5000
GPIO Port F (AHB) base: 0x4005.D000
Offset 0x50C
Type RW, reset 0x0000.0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								ODE							
Type	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Enabling Open Drain

SSI Module Pin	GPIO Pin
I2C0SCL	PB2
I2C0SDA	PB3
I2C1SDA	PA6
I2C1SDA	PA7
I2C2SCL	PB4
I2C2SDA	PB5
I2C3SCL	PD0
I2C3SDA	PD1

Thank You