# SPI & I$^2$C

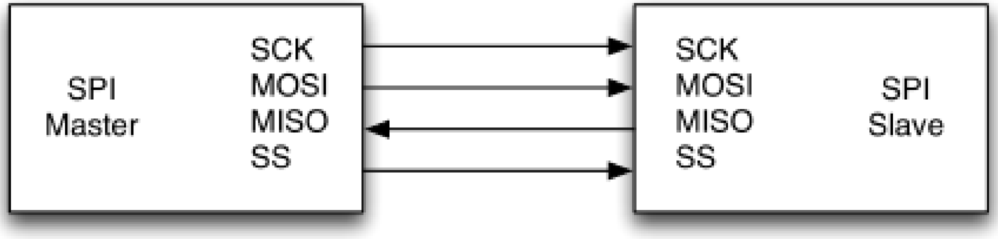
Serial Communication Protocols

# SPI Protocol

4-Wire Serial Synchronous Protocol

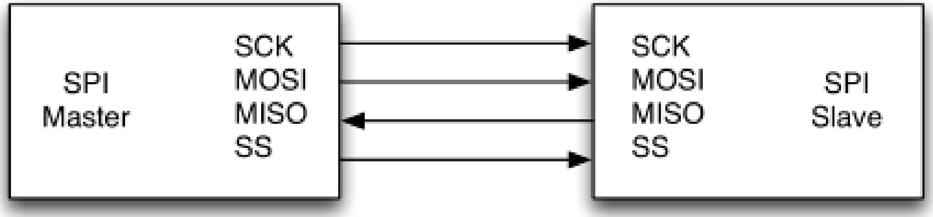
# SPI Interface

- **SPI** is short for **S**ynchronous **P**eripheral **I**nterface.
- **Full-duplex** communications.
- Developed by Motorola in mid-1980s.
- Various manufacturers implemented their versions.
    - Motorola (now **Freescale**) version is the de-facto standard.
- Multiple Devices can be connected on SPI bus.
    - Supports **one-Master** with **multiple-Slaves**.
- Master selects **clock frequency** – one that is supported by the Slave.
- Applications: LCD displays, SD cards, sensors, …



# SPI Interface

- SPI communications consist of **4** signals:
  - **SCK** : Serial Clock
  - **MOSI** : Master Output -> Slave Input
  - **MISO** : Master Input <- Slave Output
  - **SS** : Slave Select



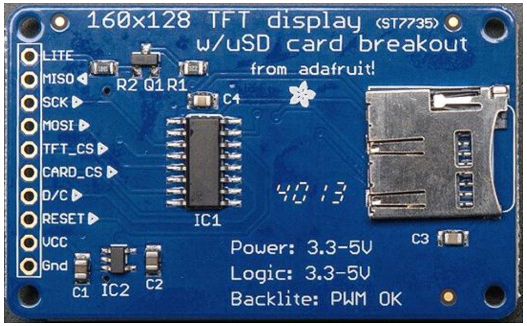
SPI Master

SCK
MOSI
MISO
SS

SCK
MOSI
MISO
SS

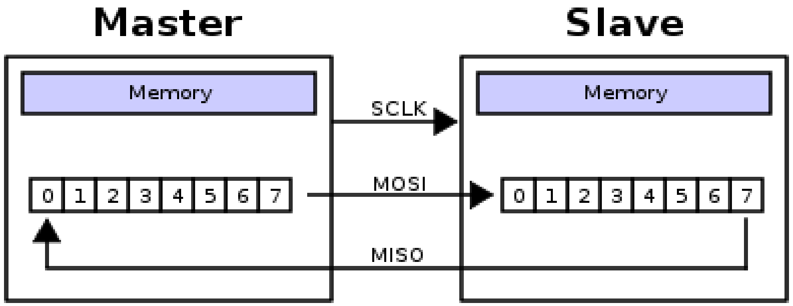
SPI Slave

TFT LCD panel used in Lab setup
=> **SPI interface:**
- MISO
- SCK
- MOSI
- TFT_CS (SS)



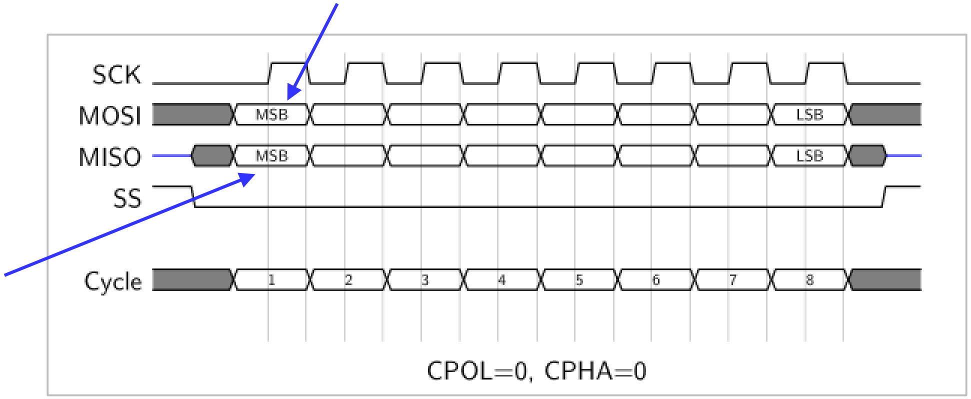
# SPI Protocol – Data Transmission

- SPI communication is controlled by the **Master**.
- During each SPI clock cycle, a **full-duplex** data transmission takes place.
- Master asserts **SS** line to initiate data transfer.
- Bits are shifted out to the slave over **MOSI** (over 8 clock cycles in example, at right).
  - MSB bit is shifted out first.
- Simultaneously, slave responds via the **MISO** signal.



Data sent from **Master to Slave** (8-bits)

Tiva LaunchPad supports 4- to 16-bit length data.



Data sent from **Slave to Master** (8-bits)

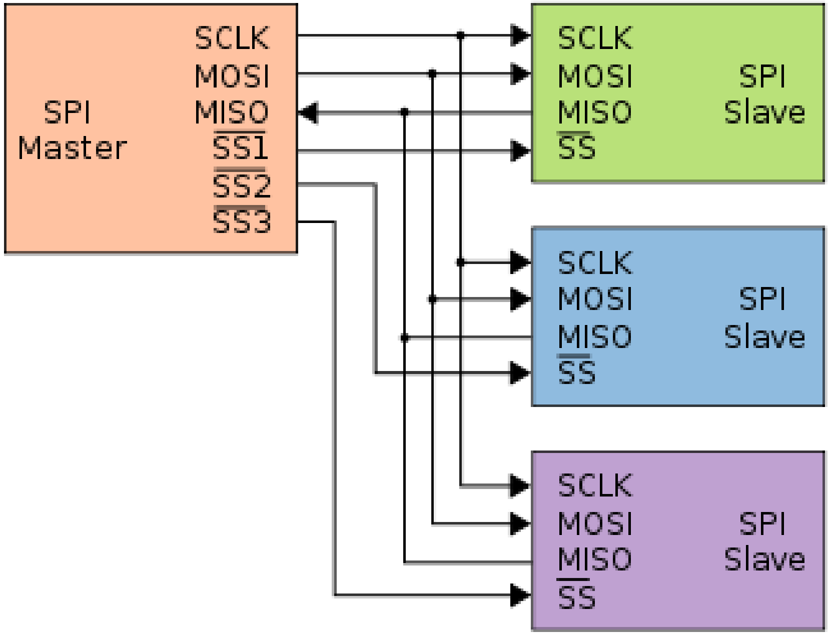
**CPOL** (Clock Polarity), **CPHA** (Phase Control)

# SPI Master & Slave Modes

- **Master Mode**
  - Only a Master initiates a transmission.
  - Data is shifted out via Master-Output-Slave-Input (MOSI) line.
  - Data is shifted in via Master-Input-Slave-Output (MISO) line.
  - Transmission ends after all data bit shifted out, synchronized to SCK.
- **Slave Mode**
  - Transfer synchronized to serial clock (SCK) from Master.
  - Data is shifted in via the Master-Output-Slave-Input (MOSI) line.
  - Data is shifted out from Slave via the Master-Input-Slave-Output (MISO) line.

# SPI Independent Slave Configuration

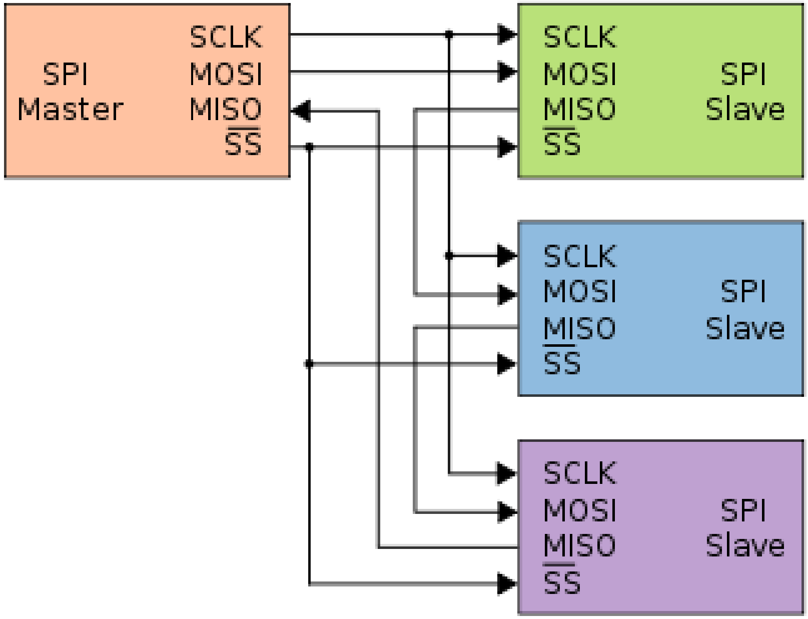
- Each Slave has an independent SS signal: **One SS for each Slave**.
- This is the most common SPI configuration used.



*Note: A pull-up resistor between power source and SS line is usually connected for each independent device to reduce cross-talk between devices.*

# SPI Daisy-Chain Configuration

- The 1st Slave output (MISO) is connected to the 2nd Slave input (MOSI) and so on.
- MISO port of each Slave sends out during the 2nd group of clock pulses an exact copy of the data it received during the 1st group of clock pulses.
- The daisy-chain link acts as a communication shift register.
- Each slave copies input to output in the next clock cycle until SS (active low) goes high.



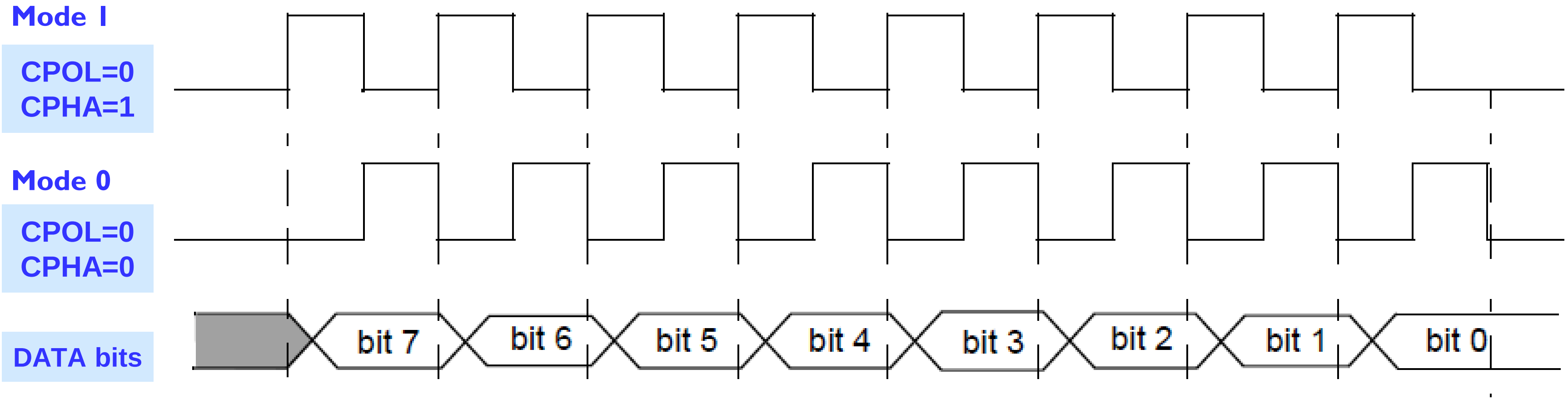
# SPI Modes of Operation:

## Clock Polarity & Phase

- Four **SPI Modes**, determined through Clock Polarity (**CPOL**) & Clock Phase (**CPHA**).
- Clock Polarity determines the starting logic state of the clock.
  - When **CPOL** is 'L', the clock generated by the Master i.e. SCK is 'L' when idle & toggles to 'H' during active state (during a transfer).
  - When **CPOL** is 'H', SCK is 'H' during idle & 'L' during active state.
- Clock Phase determines the clock transition, whether rising or falling, at which data is transmitted.
  - When **CPHA** is '0', data is transmitted on 1st clock edges.
  - When **CPHA** is '1', data is transmitted on 2nd clock edges.

In Tiva LaunchPad, Clock Polarity & Phase are set in the SSI Control Register (**SSICR0**) - bits **SPO** (Polarity) & **SPH** (Phase).

# SPI Modes of Operation
## (Modes 0 & 1)



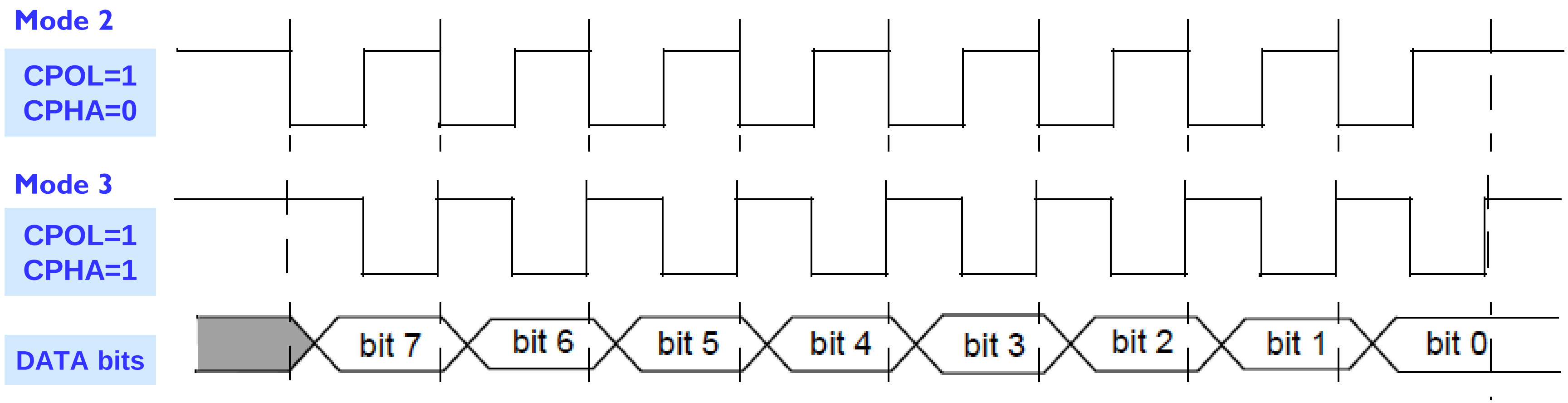
**Mode 0:** (CPOL = 0 & CPHA = 0)

- Clock Polarity is ‘L’,
- Clock Phase is 0.
- Data transmission occurs during 1st (rising) clock edge.

**Mode 1:** (CPOL = 0 & CPHA = 1)

- Clock Polarity is ‘L’,
- Clock Phase is 1.
- Data transmission occurs during 2nd (falling) clock edge.

# SPI Modes of Operation
## (Modes 2 & 3)



**Mode 2:** (CPOL = 1 & CPHA = 0)

- Clock Polarity is ‘H’,
- Clock Phase is 0.
- Data transmission occurs during 1st (falling) clock edge.

**Mode 3:** (CPOL = 1 & CPHA = 1)

- Clock Polarity is ‘H’,
- Clock Phase is 1.
- Data transmission occurs during 2nd (rising) clock edge.

# Tiva LaunchPad SSI Module

SSI0, SSI1, SSI2, SSI3

# SSI Module

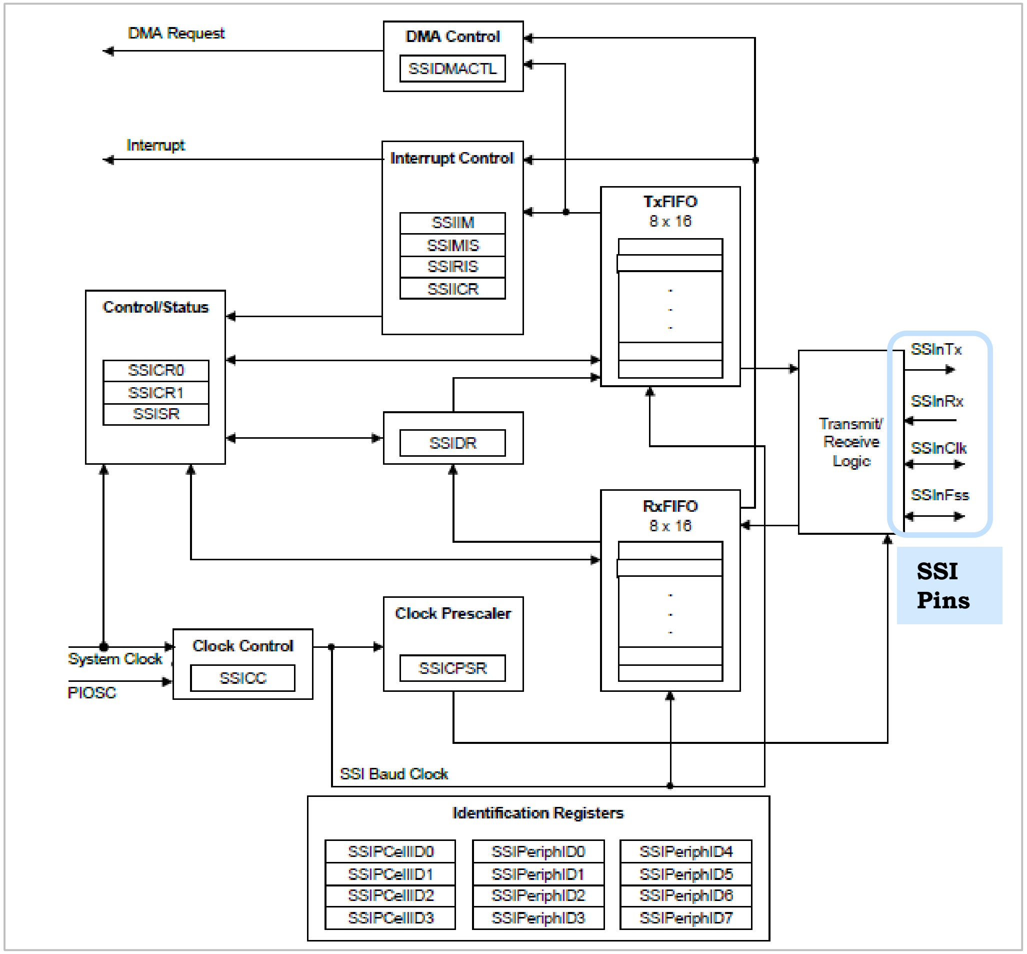
- TM4C123GH6PM microcontroller includes **four** Synchronous Serial Interface (SSI) modules:
  - **SSI0, SSI1, SSI2, SSI3**.
- Each SSI module supports:
  - Master or Slave operation.
  - 3 types of Synchronous Serial supported:
    - **Freescale SPI**, - *de-facto standard*
    - MICROWIRE,
    - Texas Instruments Synchronous Serial interfaces.
  - Programmable bit clock rate & pre-scaler.
  - Programmable data frame width from 4 bits to 16 bits.
  - Separate Tx and Rx FIFOs ( 8 levels by 16-bits).
  - Interrupts & DMA support.

# SSI Modules

Each SSI modules has 4 pins that are multiplexed with other functions.

| SSI pin | SPI Function |
|---|---|
| SSI*n*Clk | SPI Clock |
| SSI*n*Fss | SPI Slave Select |
| SSI*n*Rx | MISO (Receive ) |
| SSI*n*Tx | MOSI (Transmit) |

where *n* = 0 to 3 (total of 4 SSI modules).



SSI Module Block Diagram

# SSI Modules

| SPI | LaunchPad | SSI0 (PMC#) | SSI1 (PMC#) | SSI2 (PMC#) | SSI3 (PMC#) |
|---|---|---|---|---|---|
| **SCK** | SSI*n*Clk | PA2 (2) | PF2 (2)<br>PD0 (2) | PB4 (2) | PD0 (1) |
| **SS** | SSI*n*Fss | PA3 (2) | PF3 (2)<br>PD1 (2) | PB5 (2) | PD1 (1) |
| **MISO** | SSI*n*Rx | PA4 (2) | PF0 (2)<br>PD2 (2) | PB6 (2) | PD2 (1) |
| **MOSI** | SSI*n*Tx | PA5 (2) | PF1 (2)<br>PD3 (2) | PB7 (2) | PD3 (1) |

- All SSI functions are GPIO Alternate functions.

# SPI Frame Control

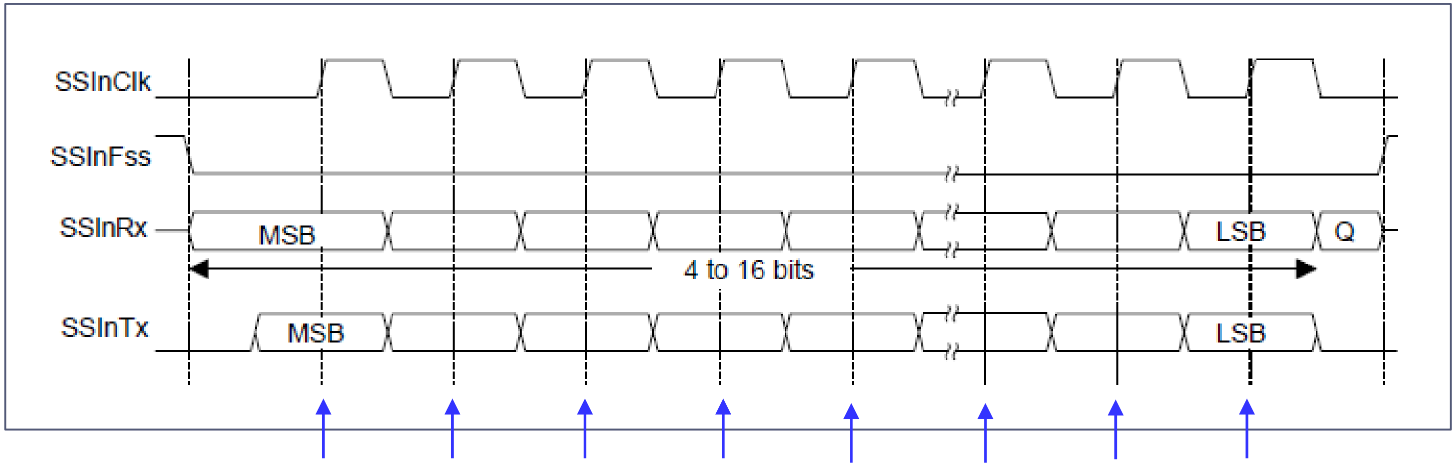
- **SPI Clock:** SPO = Clock Polarity Bit (set through **SSICR0** register)
  - When the SPO=‘0’, it produces a steady state ‘L’ on the SSI*n*Clk pin.
  - If the SPO=‘1’, a steady state ‘H’ is placed on the SSI*n*Clk pin when data is not being transferred.

- **SPI Phase:** SPH = Phase Control Bit (set through **SSICR0** register)
  - SPH phase control bit selects the clock edge that captures data.
  - SPH = ‘0’, data is captured on the **1st clock edge** transition.
  - SPH = ‘1’, data is captured on the **2nd clock edge** transition.

| SSI | | SPI |
|---|---|---|
| SPO | ≡ | CPOL |
| SPH | ≡ | CPHA |

**SSICR0** = SSI Control Register 0.

# SPI Mode 0 (SPO=0, SPH=0)

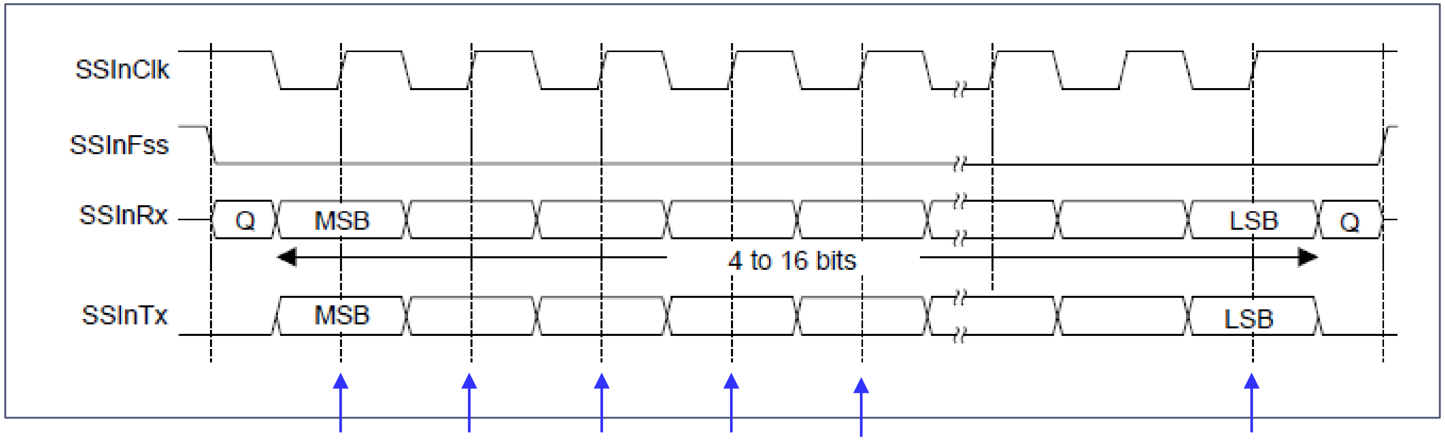


*Data is clocked-in at these instances.*

- SSI*n*Clk is ‘0’ when not active → SPO = 0.
- Data is clocked in during 1st SSI*n*Clk transition → SPH = 0.
- Therefore, above is **SPI Mode 0**.

| SSI | | SPI |
|---|---|---|
| SPO | ≡ | CPOL |
| SPH | ≡ | CPHA |

# SPI Mode 3 (SPO=1, SPH=1)



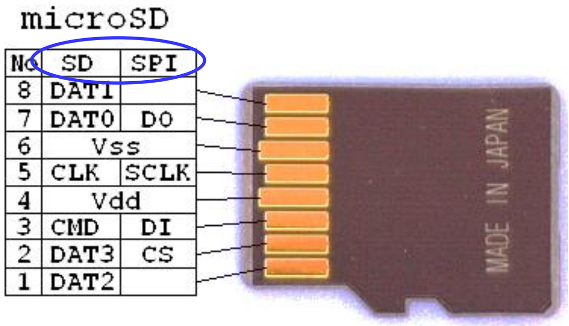
*Data is clocked-in at these instances.*

- SSIClk is ‘1’ when not active → SPO = 1.
- Data is clocked in during 2nd SSI*n*Clk transition → SPH = 1.
- Therefore, above is **SPI Mode 3**.

| <u>SSI</u> | | <u>SPI</u> |
|---|---|---|
| SPO | ≡ | CPOL |
| SPH | ≡ | CPHA |

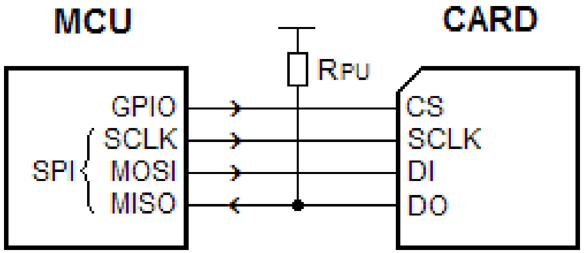
# Micro SD Card: SPI Mode

- SD Card can operate in **SD mode** (parallel) or **SPI mode** (serial).
- SPI mode operation is a subset of the SD Card protocol.
- SD Mode is default for SD cards.
- SD card enters SPI Mode if it detects CS is ‘L’ when Reset command (CMD0) is received.
- SD Mode:
  - 4 bit bi-directional data lines - DAT[3:0] – high performance.
- SPI Mode:
  - Serial data transfer (*lower performance*).
  - Byte (8-bits) transfer for command & response.
  - Uses **Mode 0** operation (CPHA = 0; CPOL = 0).



microSD

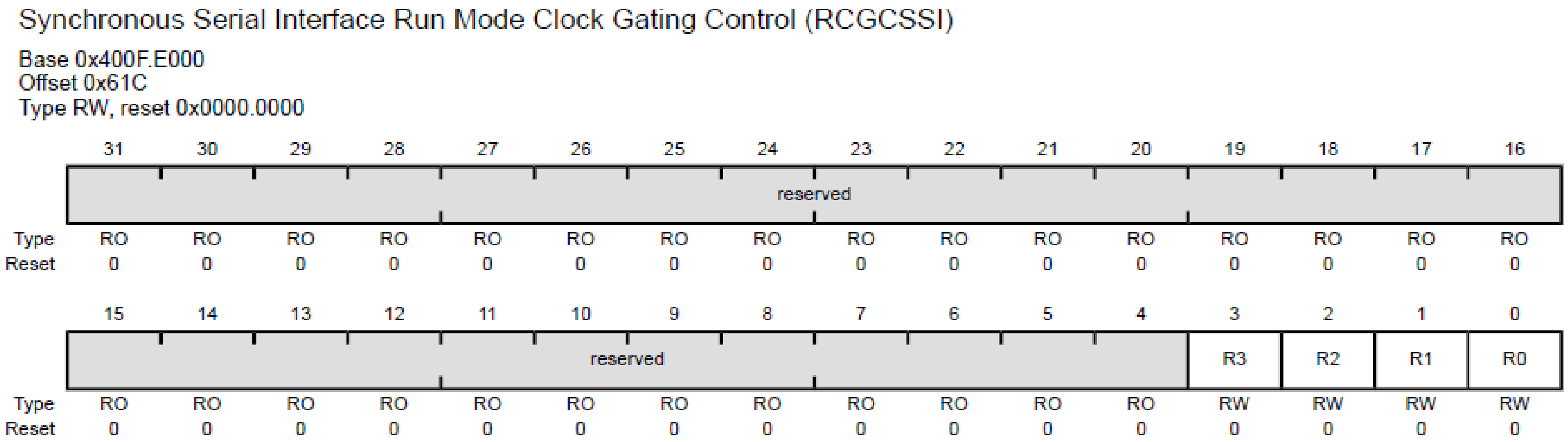
| No | SD | SPI |
|---|---|---|
| 8 | DAT1 | |
| 7 | DAT0 | DO |
| 6 | Vss | |
| 5 | CLK | SCLK |
| 4 | Vdd | |
| 3 | CMD | DI |
| 2 | DAT3 | CS |
| 1 | DAT2 | |



# SSI Registers

SSI Module Registers for SPI Operation

# SSI Clock Gating Register (RCGCSSI)



Synchronous Serial Interface Run Mode Clock Gating Control (RCGCSSI)

Base 0x400F.E000
Offset 0x61C
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 |

SSI3 SSI2 SSI1 SSI0

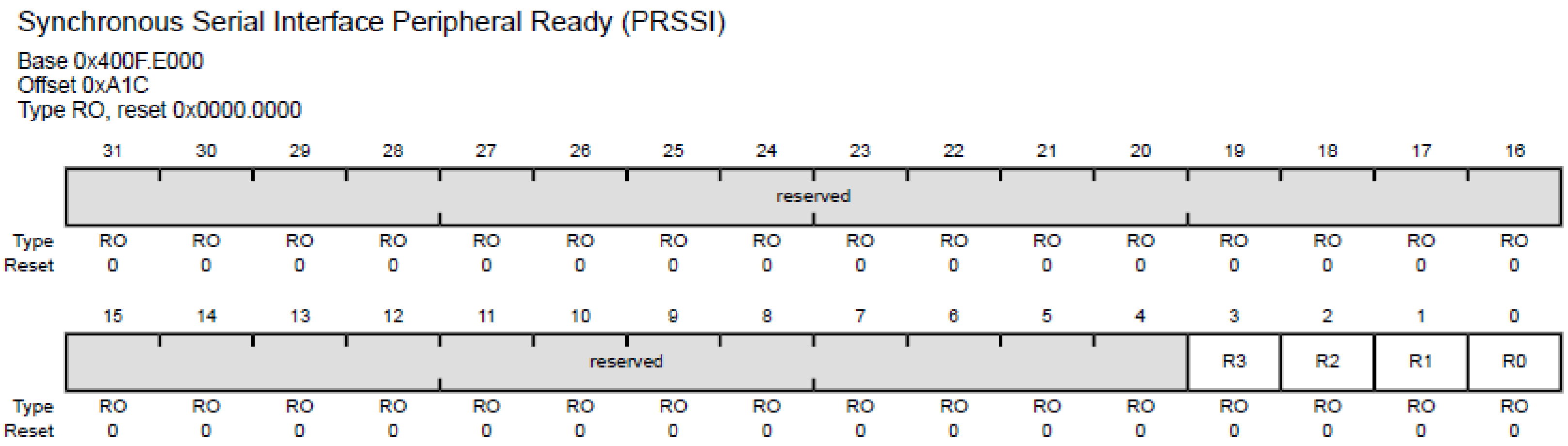
- **RCGCSSI** provides a clock & enables accesses to SSI module registers.
- When disabled, the clock is disabled to save power & accesses to module registers generate a bus fault.

**R*n*** bit:
‘0’ disable clock gating;
‘1’ enables clock gating.

**R0:** SSI module 0
**R1:** SSI module 1
**R2:** SSI module 2
**R3:** SSI module 3

Source: TM4C123GH6PM Datasheet (spms376e), pg 346

# SSI Peripheral Ready Register (PRSSI)



Synchronous Serial Interface Peripheral Ready (PRSSI)

Base 0x400F.E000
Offset 0xA1C
Type RO, 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 | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | SSI3 | SSI2 | SSI1 | SSI0 |

- **PRSSI** indicates whether the SSI modules are ready to be accessed by software following a change in status of power, Run mode clocking, or reset.

***Rn*** bit:
'0' = SSI module not ready;
'1' = SSI module ready for access.

**R0:** SSI module 0
**R1:** SSI module 1
**R2:** SSI module 2
**R3:** SSI module 3

Source: TM4C123GH6PM Datasheet (spms376e), pg 412

# SSI Module Registers

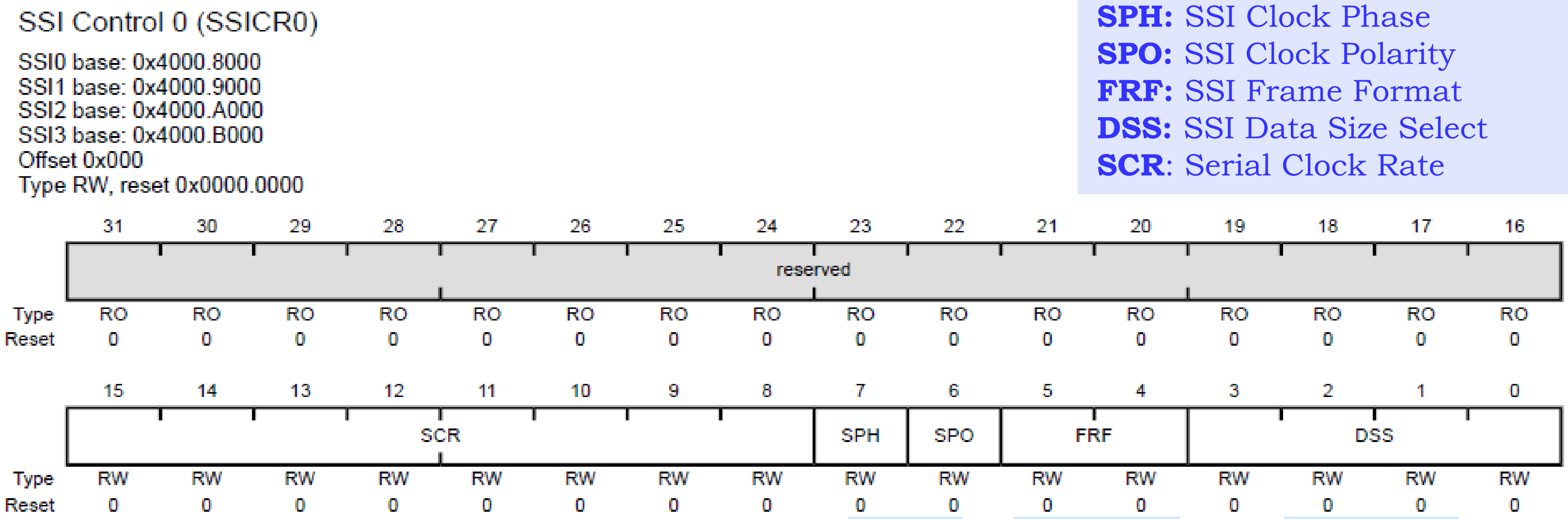
Similar to GPIO & UART modules!

- SSI Clock & Ready:
  - SSI Clock Gating (**RCGCSSI**)
  - SSI Peripheral Ready (**PRSSI**).
- Initialization (Protocol Settings):
  - Control 0 (**SSICR0**)
  - Control 1 (**SSICR1**)
  - Clock Pre-scale (**SSICPSR**)
- Data Transmit/Receive:
  - SSI Data register (**SSIDR**)
  - SSI Status register (**SSISR**)
- Interrupt Handling:
  - Interrupt Mask (**SSIIM**)
  - Raw Interrupt Status (**SSIRIS**)
  - Masked Interrupt Status (**SSIMIS**)
  - Interrupt Clear (**SSIICR**).

For a complete list of SSI registers, see datasheet page 967 (SSI Register Map).

# SSI Control Register 0 (SSICR0) – [1/2]

**SPH:** SSI Clock Phase
**SPO:** SSI Clock Polarity
**FRF:** SSI Frame Format
**DSS:** SSI Data Size Select
**SCR**: Serial Clock Rate



SSI Control 0 (SSICR0)

SSI0 base: 0x4000.8000
SSI1 base: 0x4000.9000
SSI2 base: 0x4000.A000
SSI3 base: 0x4000.B000
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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | SCR | | | | | | | | SPH | SPO | FRF | | DSS | | | |
| Type | RW | RW | RW | RW | RW | RW | RW | RW | 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 |

SPI Mode

Frame Format

No of Data Bits

**SSICR0** register: configures SPI mode, clock rate & data size.

**SPH:** ‘0’ = data captured on $1^{st}$ clock edge;
‘1’ = data captured on $2^{nd}$ clock edge.

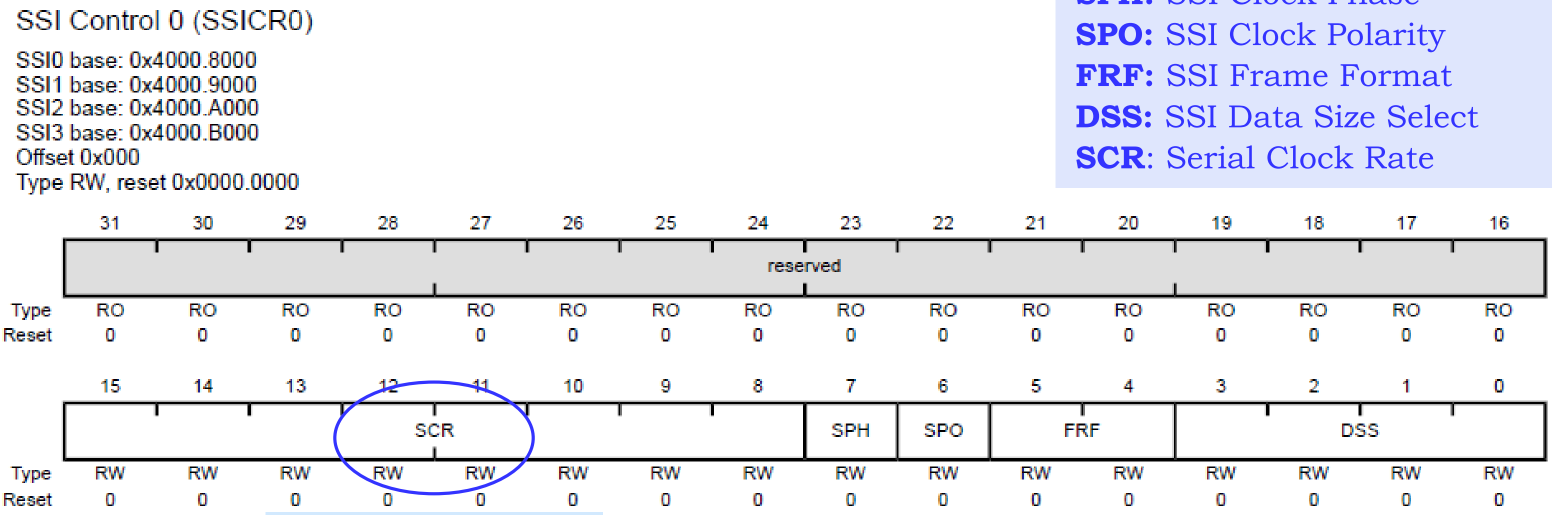
**SPO:** ’0’ = clock pin ‘L’ when not active.
‘1’ = clock pin ‘H’ when not active..

**FRF:** 0x0 = **Freescale SPI Format**
0x1 = TI Synchronous Serial Frame
0x2 = MICROWIRE Frame Format

**DSS:** 0x6 = 7-bit data; 0x7 = 8-bit data
0x8 = 9-bit data; 0x9 =10-bit data
*(data can range from 4- to 16-bits; see full list in datasheet)*

Source: TM4C123GH6PM Datasheet (spms376e), pg 969

# SSI Control Register 0 (SSICR0) – [2/2]



SSI Control 0 (SSICR0)

SSI0 base: 0x4000.8000
SSI1 base: 0x4000.9000
SSI2 base: 0x4000.A000
SSI3 base: 0x4000.B000
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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | SCR | | | | | | | | SPH | SPO | FRF | | DSS | | | |
| Type | RW | RW | RW | RW | RW | RW | RW | RW | 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 |

Serial Clock Rate (8-bits)

**SPH:** SSI Clock Phase
**SPO:** SSI Clock Polarity
**FRF:** SSI Frame Format
**DSS:** SSI Data Size Select
**SCR**: Serial Clock Rate

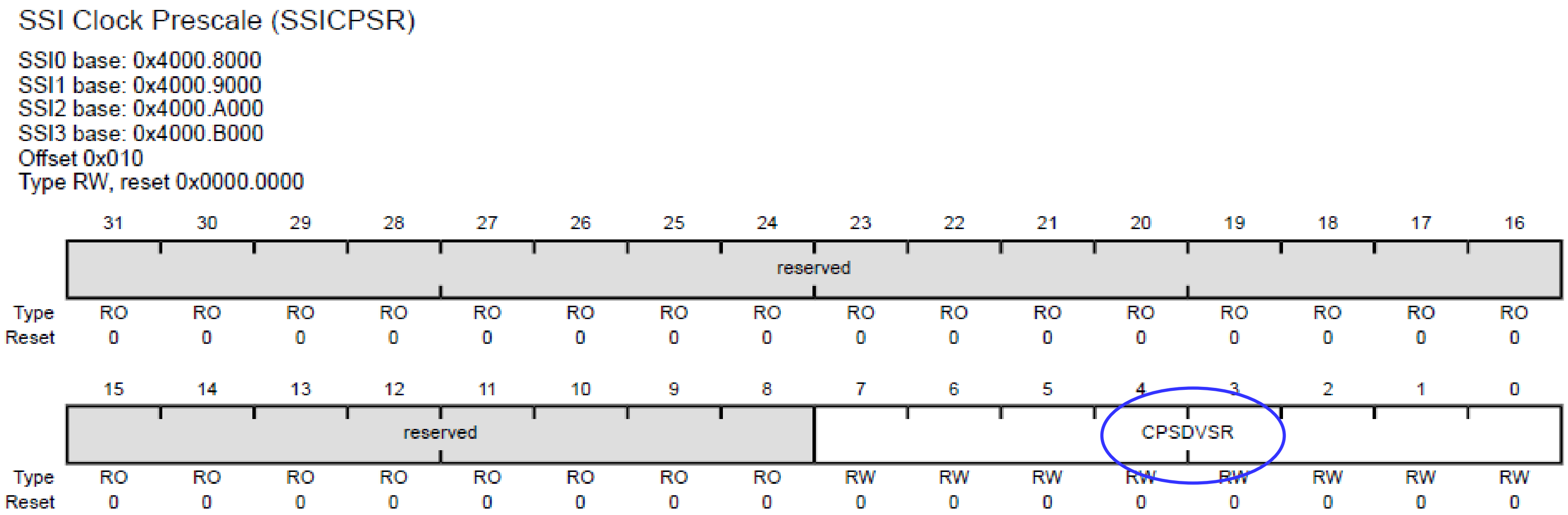
**SCR:** SSI Serial Clock Rate. This bit field is used to generate the Transmit & Receive bit rate of the SSI module.

$$SSI\ Bit\ Rate\ (BR) = \frac{SysClk}{\boldsymbol{CPSDVSR} \times (1 + \boldsymbol{SCR})}$$

**CPSDVSR** = pre-scale value; even value from 2-254 (set in **SSICPSR** reg – *next slide* )
**SCR** = value from 0-255 (8-bit field).

Source: TM4C123GH6PM Datasheet (spms376e), pg 969

# SSI Clock Pre-Scale Register (SSICPSR)



SSI Clock Prescale (SSICPSR)

SSI0 base: 0x4000.8000
SSI1 base: 0x4000.9000
SSI2 base: 0x4000.A000
SSI3 base: 0x4000.B000
Offset 0x010
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 | | | | | | | | CPSDVSR | | | | | | | |
| 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 |

SSI Clock Division Factor (8-bits)

- **CPSDVSR** specifies the division factor to use to derive the SSI*n*Clk from the system clock. The clock is then further divided by a value from 1 to 256, which is *(1 + SCR)*.

$$SSI\ Bit\ Rate\ (BR) = \frac{SysClk}{\boldsymbol{CPSDVSR} \times (1 + \boldsymbol{SCR})}$$

- **CPSDVSR** must be an even number from 2 to 254.
- **SCR** is programmed in the **SSICR0** register *(see previous slide on SSICR0).*

Source: TM4C123GH6PM Datasheet (spms376e), pg 976

# SPI Clock Rate Setting

Example:

At 80 MHz system bus speed, the following would yield a 1 MHz synchronous SPI clock.

$$SSI\ Bit\ Rate\ (BR) = 1MHz = \frac{SysClk}{CPSDVSR \times (1 + SCR)}$$

$$= \frac{80MHz}{2 \times (1 + SCR)}$$

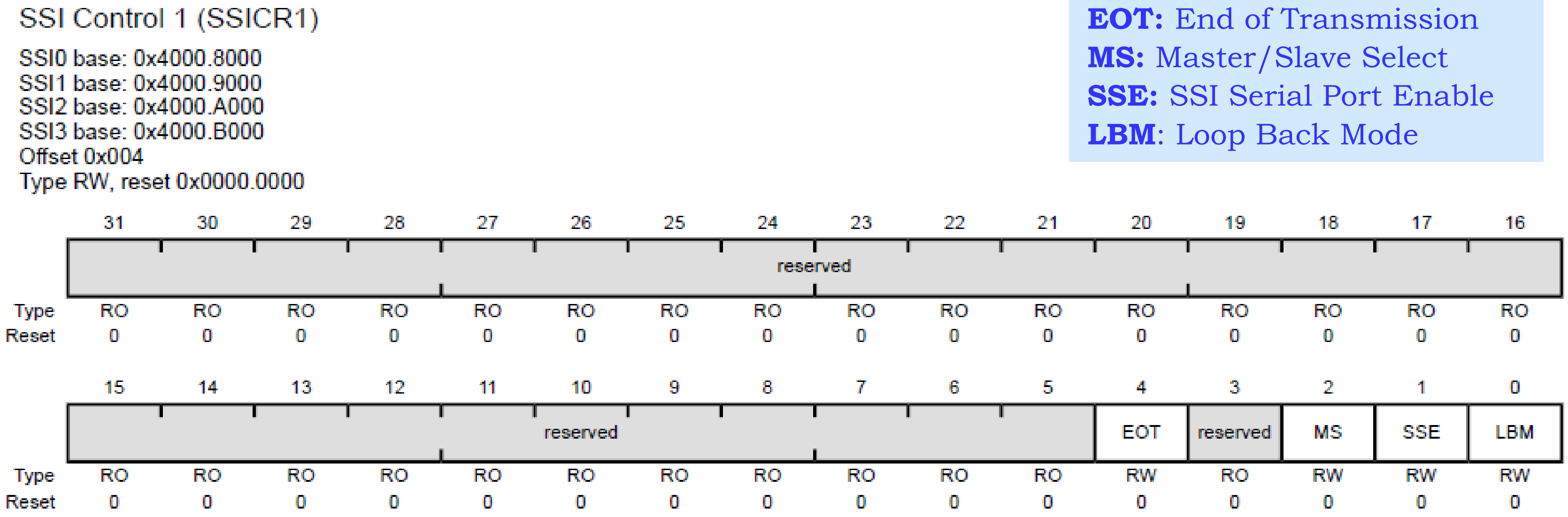
….. (we select *CPSDVSR* = 2)

**CPSDVSR** = 0x02 *(must be even between 2 to 254).*

**SCR** = 39 = 0x27 → write value to SCR field in SSICR0.

# SSI Control Register 1 (SSICR1)

**EOT:** End of Transmission
**MS:** Master/Slave Select
**SSE:** SSI Serial Port Enable
**LBM**: Loop Back Mode



SSI Control 1 (SSICR1)

SSI0 base: 0x4000.8000
SSI1 base: 0x4000.9000
SSI2 base: 0x4000.A000
SSI3 base: 0x4000.B000
Offset 0x004
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 | | | | | | | | | | | EOT | reserved | MS | SSE | LBM |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RW | RO | RW | RW | RW |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**SSICR1** register controls the Master & Slave functionality.

**EOT:** ‘0’ = Tx FIFO half full or less
‘1’ = EOT Intr enabled (TXRIS)
[**EOT** bit valid only in Master mode]

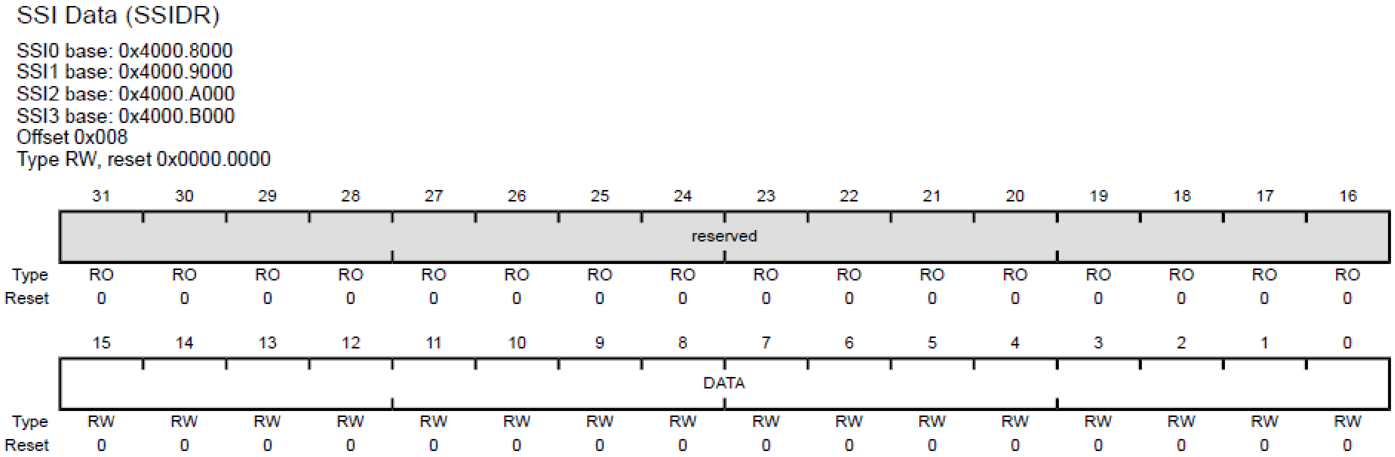
**MS :** ’0’ = SSI as Master
‘1’ = SSI as Slave

**SSE:** ‘0’ = SSI disabled
‘1’ = SSI Enabled

**LBM:** ‘0’ = Normal Mode
‘1’ = Loop Back Mode

Source: TM4C123GH6PM Datasheet (spms376e), pg 971

# SSI Data Register (SSIDR)



SSI Data (SSIDR)

SSI0 base: 0x4000.8000
SSI1 base: 0x4000.9000
SSI2 base: 0x4000.A000
SSI3 base: 0x4000.B000
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 |

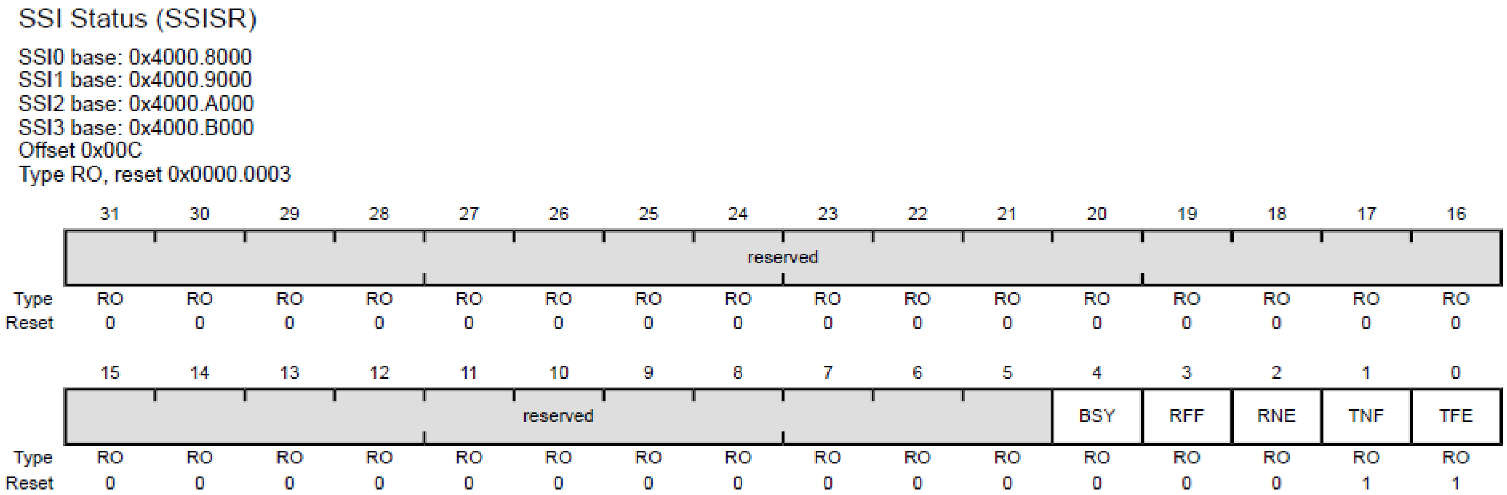
| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | DATA | | | | | | | | | | | | | | | |
| Type | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset | 0 | 0 | 0 | 0 | 0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

Data width from 4 to 16 bits

- **SSIDR** allows the SSI module to send/receive up to 16-bits of data.
- Data size is selected through the **SSICR0** register (4- to 16-bits).
  - During Transmit, if data length is <16 bits, data needs to be right-justified. Unused bits are ignored.
  - During Receive, if data length is <16 bits, data is automatically right-justified.

Source: TM4C123GH6PM Datasheet (spms376e), pg 973

# SSI Status Register (SSISR)



SSI Status (SSISR)

SSI0 base: 0x4000.8000
SSI1 base: 0x4000.9000
SSI2 base: 0x4000.A000
SSI3 base: 0x4000.B000
Offset 0x00C
Type RO, reset 0x0000.0003

| | 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 | | | | | | | | | | | BSY | RFF | RNE | TNF | TFE |
| 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 | 1 | 1 |

- **BSY**: SSI Busy bit

  '0' – SSI is not busy.

  '1' – SSI is currently transmitting/receiving a frame, or Tx FIFO is not empty.

- **RFF**: SSI Receive FIFO Full

  '0' – FIFO is not full; '1' – FIFO is full.

- **TNF**: SSI Transmit FIFO Not Full

  '0' – FIFO is full; '1' – FIFO is not full.

- **RNE**: SSI Receive FIFO Not Empty

  '0' – FIFO is empty; '1' – FIFO is not empty.

- **TFE**: SSI Transmit FIFO Empty

  '0' – FIFO is not empty; '1' – FIFO is empty.

Source: TM4C123GH6PM Datasheet (spms376e), pg 974

# SSI Initialization

SSICR0, SSICR1, SSICPSR Registers

# SSI Data Structure

| | |
|---|---|
| 0x4000.A000 | CR0 |
| 0x4000.A004 | CR1 |
| 0x4000.A008 | DR |
| 0x4000.A00C | SR |
| 0x4000.A010 | CPSR |
| 0x4000.A014 | IM |
| 0x4000.A018 | RIS |
| 0x4000.A01C | MIS |
| 0x4000.A020 | ICR |
| 0x4000.A024 | DMACTL |
| | |
| 0x4000.AFC8 | CC |

[SS0]

```
typedef struct {                  /* SSI0 Structure         */
  __IO uint32_t  CR0;             /* SSI Control 0          */
  __IO uint32_t  CR1;             /* SSI Control 1          */
  __IO uint32_t  DR;              /* SSI Data               */
  __IO uint32_t  SR;              /* SSI Status             */
  __IO uint32_t  CPSR;            /* SSI Clock Prescale     */
  __IO uint32_t  IM;              /* SSI Interrupt Mask     */
  __IO uint32_t  RIS;             /* SSI Raw Interrupt Status      */
  __IO uint32_t  MIS;             /* SSI Masked Interrupt Status   */
  __O  uint32_t  ICR;             /* SSI Interrupt Clear    */
  __IO uint32_t  DMACTL;          /* SSI DMA Control */
  __I  uint32_t  RESERVED[1000];
  __IO uint32_t  CC;              /* SSI Clock Configuration   */
} SSI0_Type;
```

[file: TM4C123GH6PM7.h]

# Accessing the SSI Data Structure

|  |  |
|---|---|
| 0x4000.A000 | CR0 |
| 0x4000.A004 | CR1 |
| 0x4000.A008 | DR |
| 0x4000.A00C | SR |
| 0x4000.A010 | CPSR |
| 0x4000.A014 | IM |
| 0x4000.A018 | RIS |
| 0x4000.A01C | MIS |
| 0x4000.A020 | ICR |
| 0x4000.A024 | DMACTL |
|  |  |
| 0x4000.AFC8 | CC |

[SS0]

```
typedef struct {                  /* SSI0 Structure        */
  __IO uint32_t  CR0;             /* SSI Control 0         */
  __IO uint32_t  CR1;             /* SSI Control 1         */
  __IO uint32_t  DR;              /* SSI Data              */
  __IO uint32_t  SR;              /* SSI Status            */
  __IO uint32_t  CPSR;            /* SSI Clock Prescale    */
  __IO uint32_t  IM;              /* SSI Interrupt Mask    */
  __IO uint32_t  RIS;             /* SSI Raw Interrupt Status     */
  __IO uint32_t  MIS;             /* SSI Masked Interrupt Status  */
  __O  uint32_t  ICR;             /* SSI Interrupt Clear   */
  __IO uint32_t  DMACTL;          /* SSI DMA Control       */
  __I  uint32_t  RESERVED[1000];
  __IO uint32_t  CC;              /* SSI Clock Config      */
} SSI0_Type;

/* macro to cast memory address to a pointer  */
#define SSI0_BASE               0x40008000UL
#define SSI1_BASE               0x40009000UL
#define SSI2_BASE               0x4000A000UL
#define SSI3_BASE               0x4000B000UL
#define SSI0            ((SSI0_Type  *) SSI0_BASE)
#define SSI1            ((SSI0_Type  *) SSI1_BASE)
#define SSI2            ((SSI0_Type  *) SSI2_BASE)
#define SSI3            ((SSI0_Type  *) SSI3_BASE)
```

```
/* accessing SSI registers  */
SSI0->CR0 |= 1UL <<14    /* set bit 14  */
SSI2->CR1 &= ~1UL <<20   /* reset bit 20  */
```

# SPI Initialization [1/2]

Two steps to enable & initialize a **SSI** module:

- Configure GPIO Alternate pin function,
- Configure SSI module.

- Configure GPIO to alternate function:
  - Enable SSI module using the **RCGCSSI** register.
  - Enable the clock to the appropriate GPIO module through **RCGCGPIO**.
  - Set the **GPIO AFSEL** bits for the appropriate pins
  - Configure the **PMC*n*** fields in the **GPIOPCTL** register to assign the SSI signals to the appropriate pins.
  - Program **GPIODEN** register to enable SSI pin's digital function & configure drive strength, drain select and pull-up/pull-down functions.

  [Note: Pull-ups can be used to avoid noise on SSI pins].

# SPI Initialization [2/2]

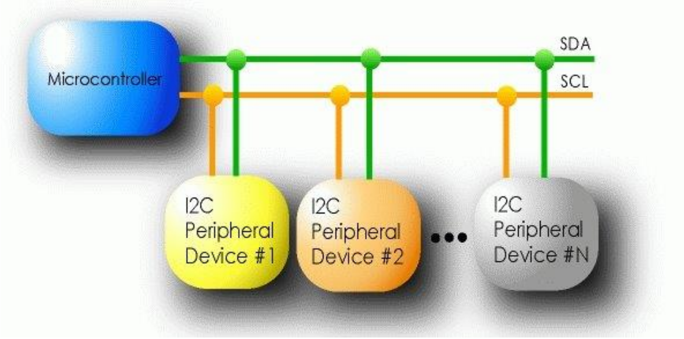
- Configure SSI Module:
  - Check that SSI module is ready through the **PRSSI** register. *(Clock needs to be enabled earlier through RCGCSSI).*
  - Clear **SSE** bit in **SSICR1** register before making any configuration changes.
  - Select whether the SSI is a Master or Slave:
    - Master: **SSICR1, MS** = 0**.**
    - Slave (output enabled): **SSICR1 =** 0x04.
    - Slave (output disabled), **SSICR1 =** 0x0C.
  - Configure SSI clock source through **SSICC** register.
  - Configure the Clock Prescale Divisor by writing to **SSICPSR** register.
  - Write to **SSICR0** register to configure:
    - Serial clock rate (**SCR**)
    - Clock phase/polarity (**SPH** and **SPO**)
    - Protocol mode: Freescale SPI, TI SSF, MICROWIRE (**FRF**)
    - Number of data bits (**DSS**)
  - Enable SSI by setting **SSE** bit in the **SSICR1** register.

# $I^2C$ Protocol

2-Wire Synchronous Serial Protocol

# I$^2$C History & Development

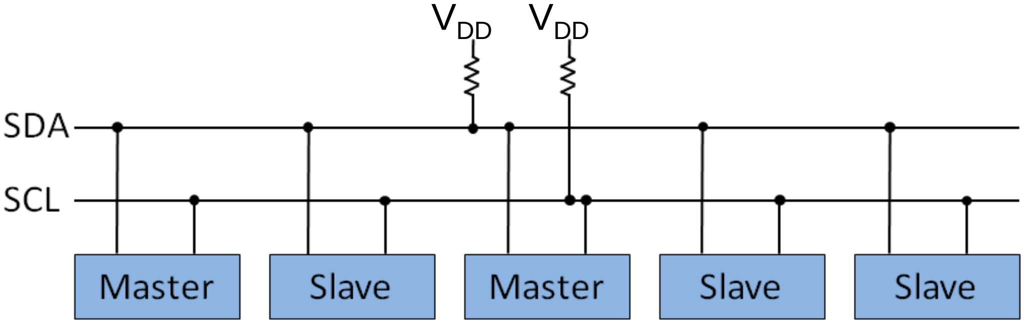
- **I$^2$C** short for **I**nter-**I**ntegrated **C**ircuit serial interface protocol.
- Also written as **I2C** or **IIC**.
- Originally developed at Philips Semiconductors (*now NXP Semiconductor* ) in 1992.
- Objective of **I$^2$C** is to define a communication protocol between microprocessors & peripherals, like
  - Real-Time-Clock (RTC)
  - Analog to Digital Converter (ADC)
  - LCD display,
  - Sensors, …
- Side-note: System Management Bus (**SMBus**), defined by Intel in 1995, is a subset of I$^2$C.



- Version 1.0:
  - Introduced in 1992.
  - "**Standard**" mode: **100 Kbits/s**.
  - "**Fast**" mode: **400 Kbits/s**.
- Version 2.0:
  - Released in 1998.
  - "**High-Speed**" mode: up to **3.4 Mbits/s**
- Version 2.1:
  - Released in 2000.
  - Introduced Clock Stretching.
  - "High-Speed" Timings Relaxed.

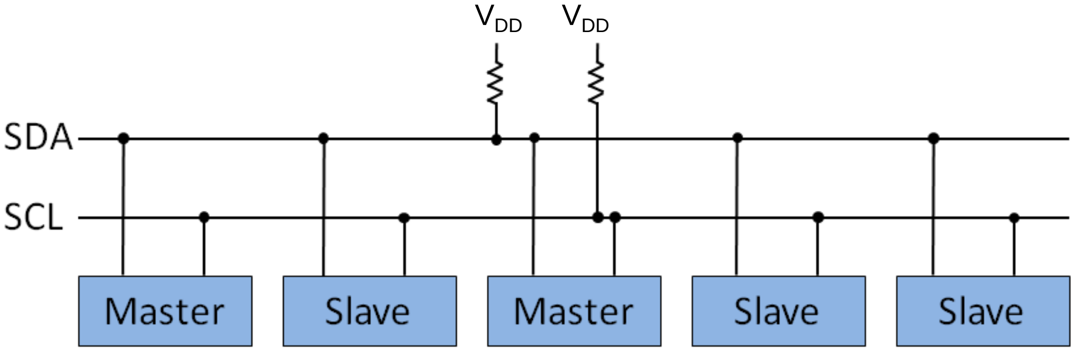
# $I^2C$ Characteristics

- **Synchronous** bi-directional serial bus.
- **Half-duplex** communications.
- Only **2 Wires** needed on $I^2C$ Bus:
  - Serial Data (**SDA**)
  - Serial Clock (**SCL**)
  - Both signals are open-collector/open-drain.
- Multiple Devices Connected On Bus: **Master-Slave** model.
  - 7- or 10-bit addressing
  - Number of slave ($2^7$ or $2^{10}$)
- Clock rate controlled by Master.
  - Supports Various Data Transfer Rates (100Kbps, 400Kbps, 3.4Mbps).



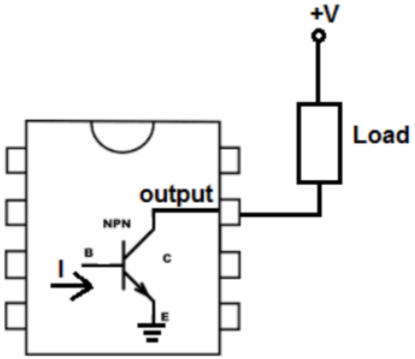
# I$^2$C Interface: SDA & SCL Signals

- **SDA** & **SCL**:
  - **Bi-directional**.
  - Open-collector/Open-drain outputs (requires pull-up resistors).
  - Typically, pull-up at 2.2KΩ (Standard), 1KΩ (Fast), 4.7KΩ (if speed <100Kbps).
- $V_{DD}$ = 5.5V max; $V_{IL}$ = 0.3 $V_{DD}$ ; $V_{IH}$ = 0.7$V_{DD}$.
- Data on **SDA** line must be stable during the 'H' period of **SCL**.
- Data can only change when **SCL** is 'L'.
- One clock pulse per data bit.



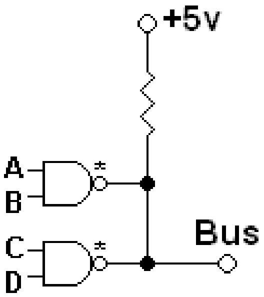
# Open-Collector/Drain Outputs

- In **Open-Collector** configuration,
  - Output signal is applied to the base of a NPN transistor with collector brought out externally.
  - The emitter of the transistor is connected internally to the ground pin.



- If transistor is replaced with a MOSFET, configuration is called **Open-Drain.**

- Open-Collector/Drain outputs may be tied-together.
  - When several open collector outputs are connected together, the common line becomes a "wired AND" (positive-true logic) or "wired OR" (negative-true logic) gate.
  - A "wired AND" behaves like the boolean AND of the two (or more) gates; output is at High if all inputs are in high impedance state, and 0 otherwise.
  - A "wired OR" behaves like the Boolean OR for negative-true logic; output is LOW if any of its inputs are low.



# $I^2C$ Signal Components

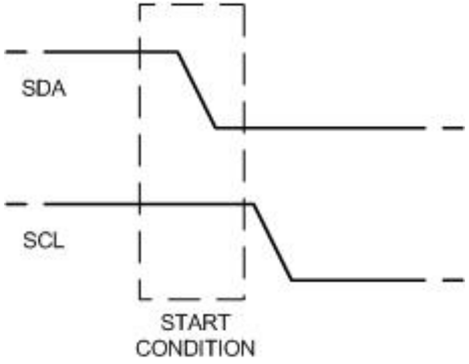
START, REPEATED-START, STOP, ACK/NACK, DATA BLOCK

# I$^2$C Signal Components

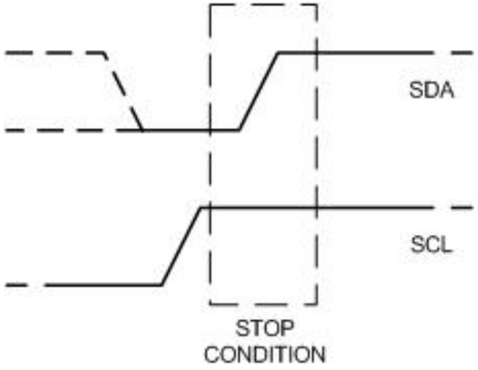
- I$^2$C data transfer consist of the following fundamental signal components:
  - **START** (S)
  - **STOP** (P)
  - **Repeated START** (R)
  - Data
  - Acknowledge (**ACK**, **NACK**).

# I²C START & STOP Conditions

- A **START** condition indicates that a device would like to transfer data onto the I²C bus.
- A device needs to initiate a **START** condition in order to perform a data transfer.
- Represented by the **SDA** going Low when **SCL** is at High.

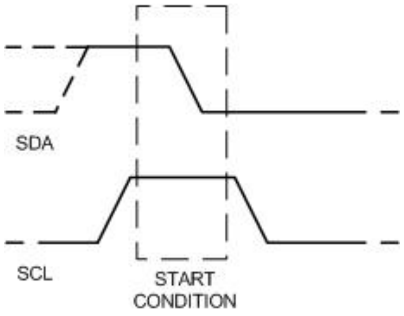


- A **STOP** condition indicates that a device wants to release the bus.
- Once released, other devices may use the bus.
- Represented by the **SDA** going High when **SCL** is at High.

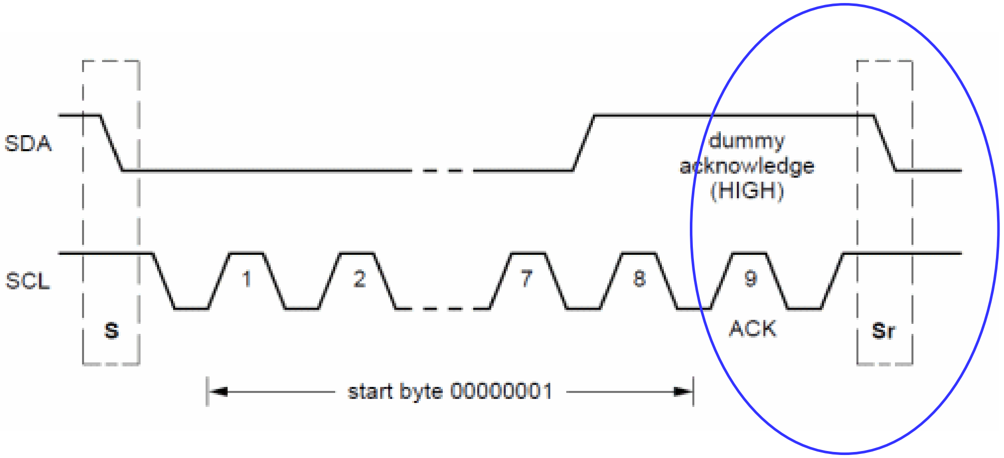


# I$^2$C Repeated START Condition

- A **Repeated START** condition is generated WITHOUT first generating a **STOP**.
- Used by a Master to indicate it wants to switch mode or send more data instead of releasing the bus.



Normal START

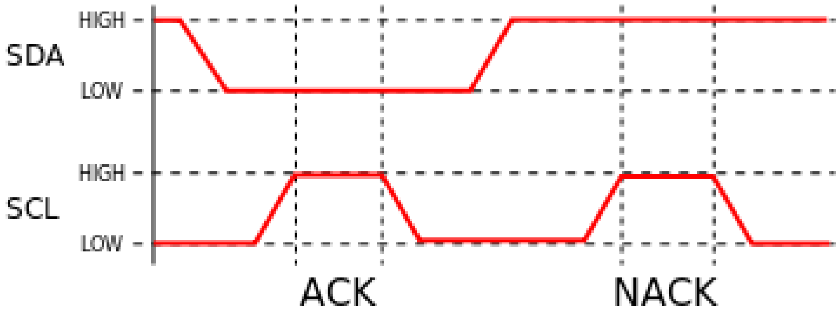


Repeat START

Note: No **STOP** conditions occurs between **START** & **Repeated START**.

# I$^2$C Acknowledge Condition

- Data transfer needs to be acknowledged either positively (**ACK**) or negatively (**NACK**).
- **ACK**: **SDA** go Low during 9th clock pulse of **SCL**.
- **NACK**: **SDA** does not go Low – device allows it to float high (*open-collector output*).

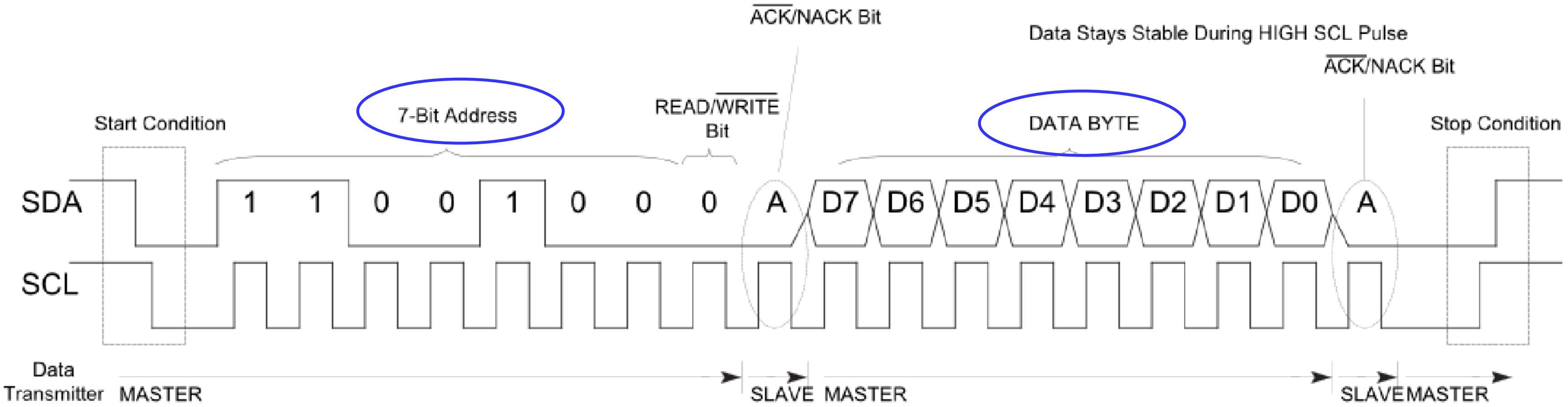


# I$^2$C Data Block

- A data block represents a transfer of 8 bits of information.
- Data is sent on the **SDA** line.
- Data may be a Control Code, Address, or Data bits.
- Data is stable when **SCL** is High.
  - Data is Read when SCL is High

→ (See also slide 39, lines in blue)

- Data on **SDA** line must be stable during ‘H’ period of **SCL**.
- Data can only change when **SCL** is ‘L’.



Note: MSB of data is transmitted first.

# I$^2$C Data Transmission

I$^2$C Frame, Clock Stretching/Synchronization, Arbitration

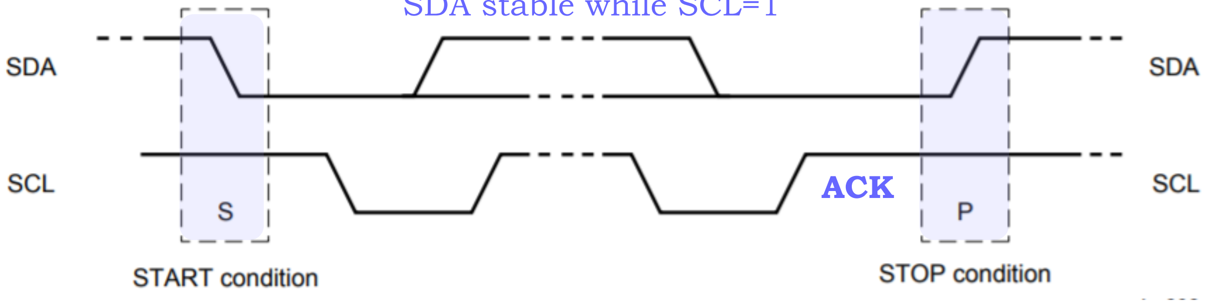
# I$^2$C Data Transmission

- Bus transaction begin with **START** & end with **STOP**. Both are generated by the Master.
- Bus is considered busy after the **START** condition, and is free after the **STOP** condition.
- **ACK** occurs after <u>every</u> byte.
- **STOP**: Transmitter releases SDA; receiver put SDA=0 while SCL=1.

**START**: SDA 1->0 while SCL=1

**STOP**: SDA 0->1 while SCL=1

**Data bits transmitted**: SDA stable while SCL=1



**ACK**

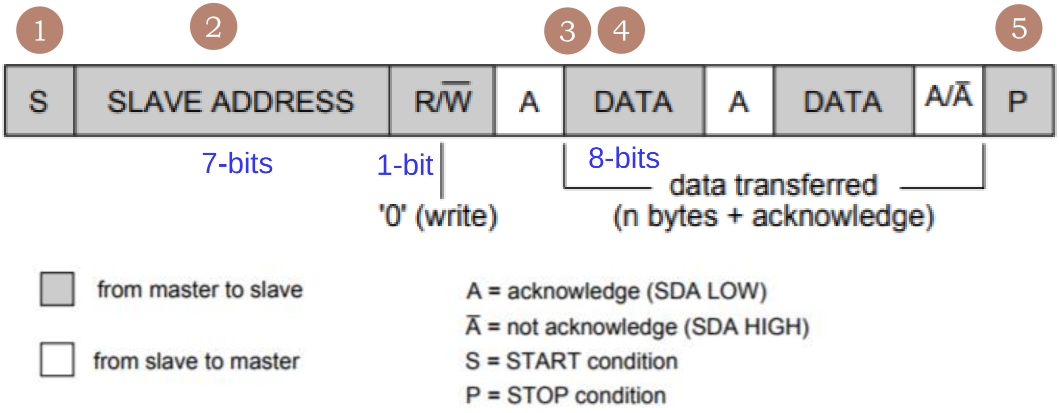
Source: I2C Specifications, V3.0

# I$^2$C Basic Data Frames: Master Write

- Master sending (WRITE) data to Slave.

1. Send **START**
2. Write slave address with last bit as WRITE ACCESS (0)
3. Write sub-address: this is usually the address of the register you want to write to; if not applicable skip to 4.
4. Write data
5. Send **STOP**

Each byte written to the slave device is answered with an ACK if the operation was successful.

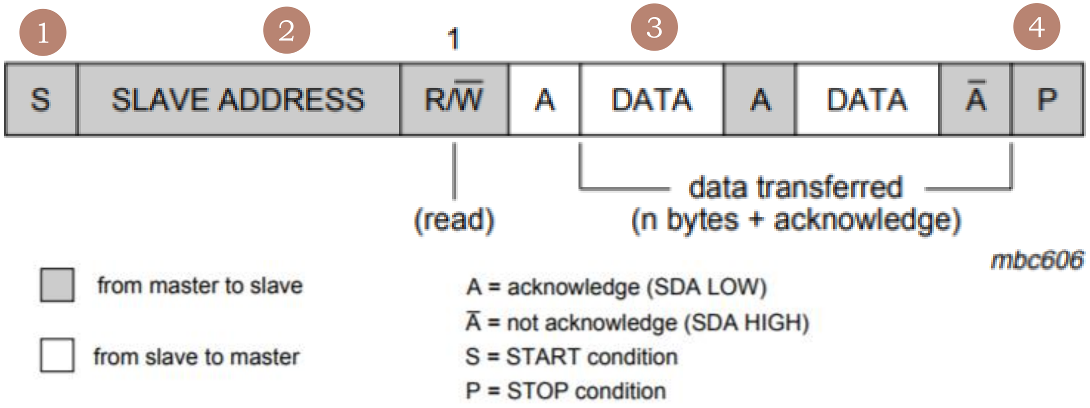


Source: I2C Specifications, V3.0

# $I^2C$ Basic Data Frames: Master Read

- Master receiving (READ) data from Slave.

1. Send **START**
2. Write slave address with last bit as READ ACCESS (1)
3. Read data from Slave, till NACK is received.
4. Send **STOP to slave**.



*<u>Note</u>: All write and read operations (except the last read) are answered with a ACK if successful.*

# I$^2$C Masters and Slaves

- Each device has a 7-bit or 10-bit address using which the data transfers take place.
- **Master:**
  - With 7-bits, a Master can address 128 ($2^7$) other slaves.
  - A Master has a processing element functioning as bus controller or a microcontroller with I$^2$C (Inter Integrated Circuit) bus interface circuit.
  - At any instance, the Master is the one which initiates a data transfer on **SDA** (serial data) line and which transmits the **SCL** (serial clock) pulses.
- **Slave:**
  - Each Slave can also optionally has I$^2$C bus controller and processing element.

# I$^2$C Modes

- I$^2$C has **FOUR** operating modes. Default is Slave mode.
  - **Master-sender**
    - Module issues START and ADDRESS, and then transmits data to the addressed slave device.
  - **Master-receiver**
    - Module issues START and ADDRESS, and receives data from the addressed slave device.
  - **Slave-sender**
    - Another master issues START and ADDRESS of this (Slave) module, which then sends data to the master.
  - **Slave-receiver**
    - Another master issues START and ADDRESS of this (Slave) module, which then receives data from the master.

Note: Some devices may support only the slave modes, like some sensors or LCD displays. These usually arise due to need to keep costs low.

# I$^2$C Clock Stretching

- **Clock stretching** is an optional feature implementation in I$^2$C. Introduced in 3$^{rd}$ I$^2$C version.
- Clock stretching pauses a transaction by holding the **SCL** line LOW. The transaction cannot continue until the line is released HIGH again.
- A Slave may be able to receive bytes of data at a fast rate, but needs more time to store a received byte or prepare another byte to be transmitted.
  - The Slave can then hold (stretch) the **SCL** line LOW after receive and ACK of a byte to force the Master into a **WAIT** state until the Slave is ready for the next byte transfer.

Source: I2C Specifications, V3.0, pg 10

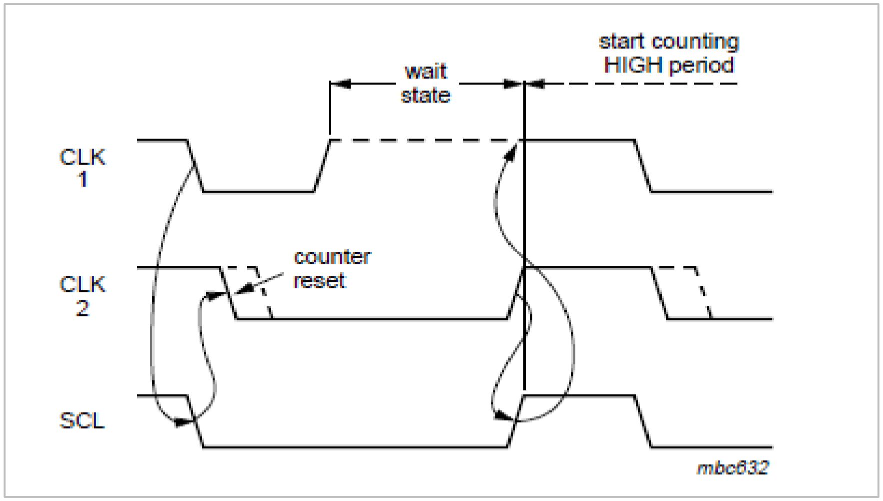
# I$^2$C Clock Synchronization & Arbitration

- If two Masters transmit on an idle I$^2$C bus at the same time, we need a method to decide which Master will take control of the bus and complete its transmission.
- This is done by **Clock Synchronization** (similar to Clock Stretching) & **Arbitration**.

Source: I2C Specifications, V3.0, pg 10

# I$^2$C Clock Synchronization

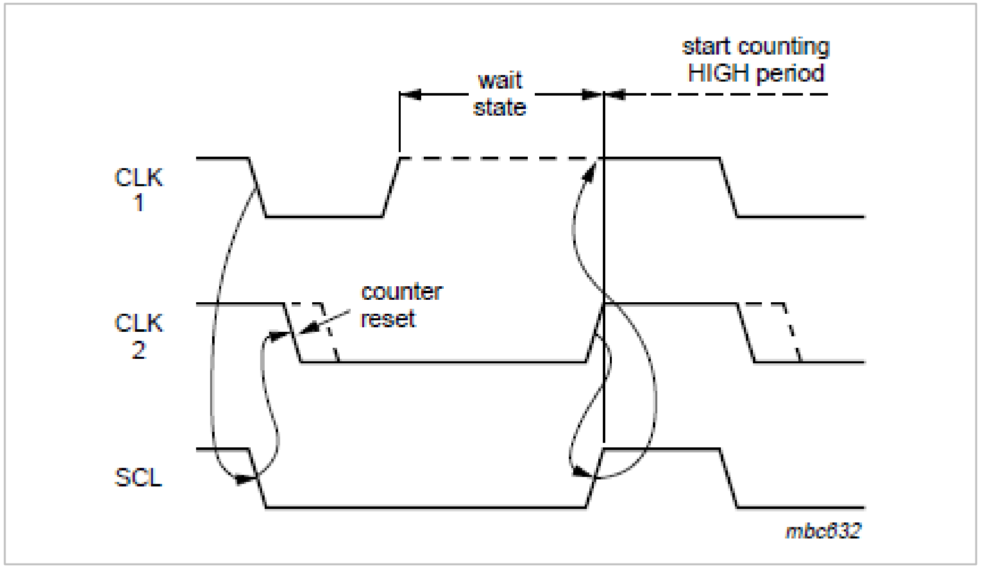
- **Clock synchronization** is performed using the wired-AND (‘*Open-Collector*’) connection to SCL line.
- A ‘H’ to ‘L’ transition on SCL line cause the Masters concerned to start counting off their ‘L’ period.



- Once a Master clock has gone ‘L’, it hold the SCL line ‘L’ until the clock ‘H’ state is reached.
- However, the ‘L’ to ‘H’ transition of this clock may not change the state of the SCL line if another clock is still within its ‘L’ period.
- The SCL line will therefore be held ‘L’ by the master with the longest ‘L’ period. Masters with shorter ‘L’ periods enter a Wait-state during this time.

Source: I2C Specifications, V3.0, pg 10

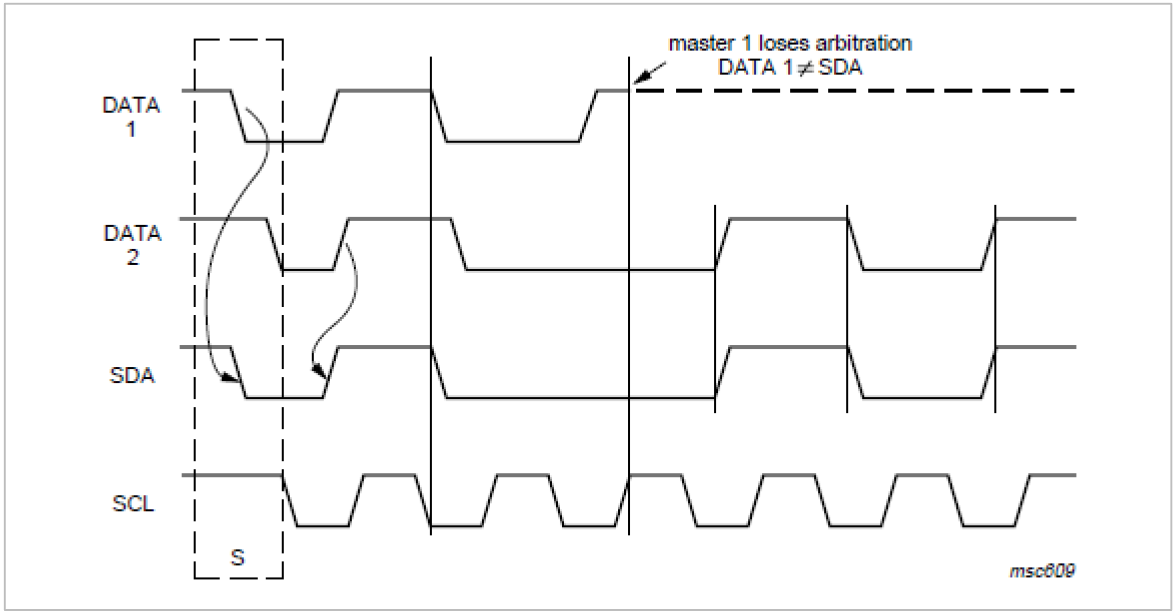
# $I^2C$ Clock Synchronization



- When all Masters have counted off their ‘L’ period, the SCL clock line is released and go HIGH.
- All the masters will start counting their ‘H’ periods. The 1st Master to complete its ‘H’ period will again pull the SCL line ‘L’.
- As a result, a synchronized SCL clock is generated with its ‘L’ period determined by the Master with the longest clock ‘L’ period & its ‘H’ period determined by the Master with the shortest clock ‘H’ period.

Source: I2C Specifications, V3.0, pg 10

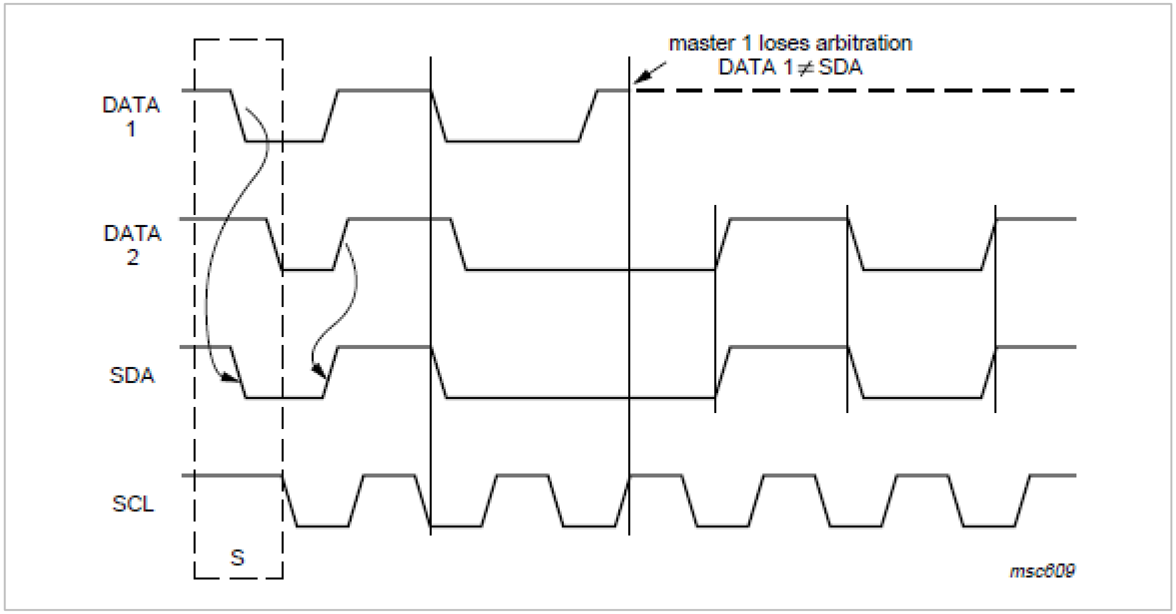
# $I^2C$ Arbitration



- A Master may start a transfer only if the bus is free. Two Masters may generate a START condition within the minimum hold time of the START condition which results in a valid START condition on the bus.
- **Arbitration** is then required to determine which Master will complete its transmission.

Source: I2C Specifications, V3.0, pg 12

# I$^2$C Arbitration



- **Arbitration** proceeds bit by bit.
- During every bit, while SCL is ‘H’, each Master checks to see if the SDA level matches what it has sent. This process may take many bits.
- When a Master tries to send a ‘H’, but detects that the SDA level is ‘L’, the master knows that it has lost the arbitration & turns off its SDA output driver. The other master goes on to complete its transaction.

Source: I2C Specifications, V3.0, pg 12

# 7-bit Address: Reserved Addresses

- For 7-bit Slave Address field, some addresses are reserved.
- **2 groups of 8 addresses (0b0000.xxx & 0b1111.xxx) are reserved.**
- However, if it is known that the reserved address is never going to be used for its intended purpose, a reserved address can be used for a slave address.

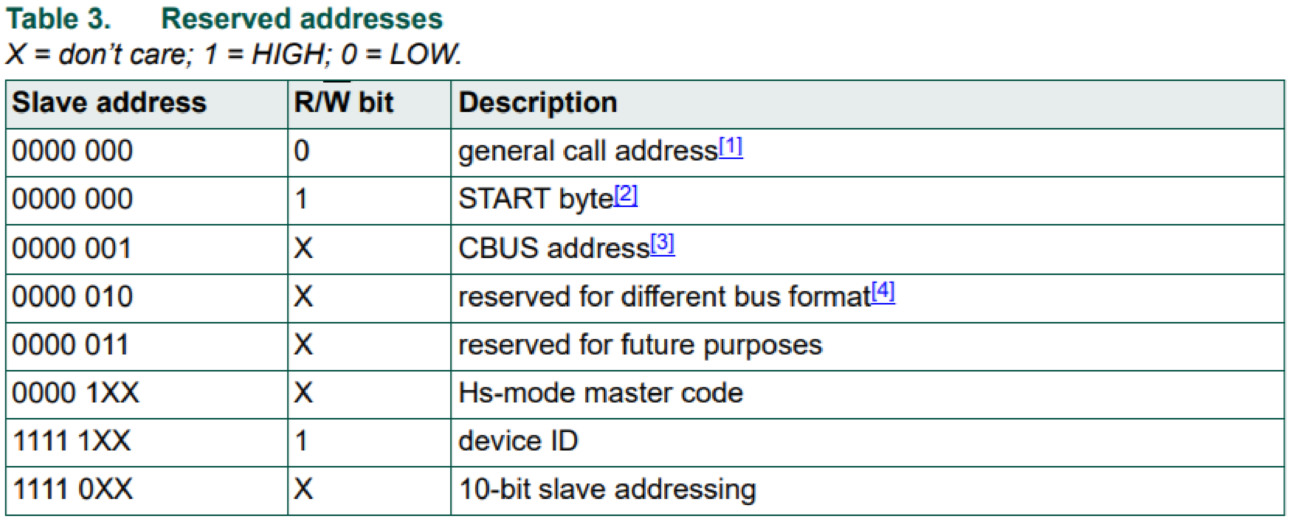


**Table 3. Reserved addresses**

*X = don't care; 1 = HIGH; 0 = LOW.*

| Slave address | R/$\overline{W}$ bit | Description |
|---|---|---|
| 0000 000 | 0 | general call address[1] |
| 0000 000 | 1 | START byte[2] |
| 0000 001 | X | CBUS address[3] |
| 0000 010 | X | reserved for different bus format[4] |
| 0000 011 | X | reserved for future purposes |
| 0000 1XX | X | Hs-mode master code |
| 1111 1XX | 1 | device ID |
| 1111 0XX | X | 10-bit slave addressing |

Source: I2C Specifications

# Disadvantages of I$^2$C

- Time taken by algorithm in the hardware that analyses the bits through I$^2$C in case the slave hardware does not provide for the hardware that supports it.
- Certain ICs support the protocol and certain do not.
- Open-collector/drain drivers at the Master need a pull-up resistor on each line.

# I$^2$C: Summary

- Two bus signals: Data (SDA), Clock (SCL).
- No Chip-enable or chip-select needed to select a Slave device.
- Supports a large number of devices on a bus: 7- or 10-bit address
- Supports multiple masters through bus arbitration.
- Generally a more complex protocol to implement – HW needs to generate START condition, STOP condition, ACK, repeat START condition, DATA bits.
- With multiple bus masters, bus arbitration is carried out automatically.
- Both 7-bit and 10-bit address are supported on the same bus.
- Support for 100 kHz (Standard), 400 kHz (Fast) & 3.4 MHz (High-Speed) bus speeds.
- Each data transfer start with a START condition & end with a STOP condition.
- For each data byte, the receiver must assert either the ACK or the NACK condition to acknowledge or un-acknowledge the data transfer.

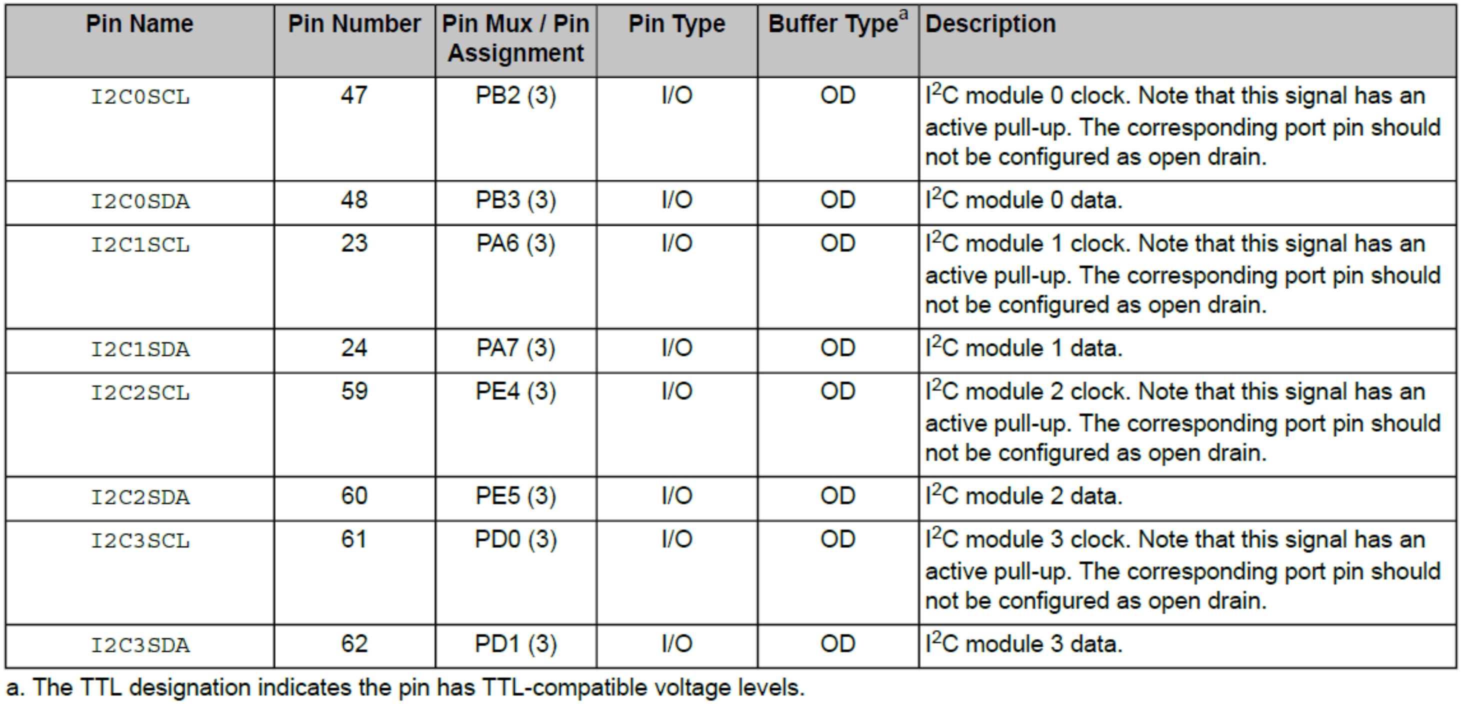
# $I^2C$ Modules in Tiva Launchpad

I2C0, I2C1, I2C2, I2C3 Modules

# $I^2C$ Modules in Tiva LaunchPad

- 4 $I^2C$ modules: **I2C0**, **I2C1**, **I2C2**, **I2C3**.
- Each module can function as Master or Slave simultaneously.
- Supports 4 $I^2C$ modes:
  - Master Transmit
  - Master Receive
  - Slave Transmit
  - Slave Receive
- Supports 4 different speeds:
  - Standard (100 Kbps)
  - Fast-mode (400 Kbps)
  - Fast-mode plus (1 Mbps)
  - High-speed mode (3.33 Mbps).
- Master & Slave Interrupts generation.
- Bus arbitration & clock synchronization.

# I$^2$C Modules in Tiva LaunchPad

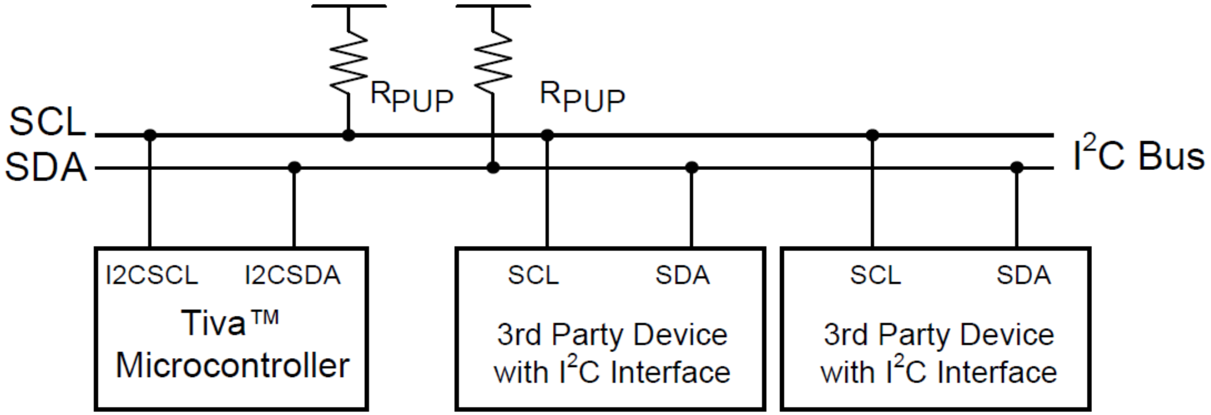


| Pin Name | Pin Number | Pin Mux / Pin Assignment | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|---|
| I2C0SCL | 47 | PB2 (3) | I/O | OD | I$^2$C module 0 clock. Note that this signal has an active pull-up. The corresponding port pin should not be configured as open drain. |
| I2C0SDA | 48 | PB3 (3) | I/O | OD | I$^2$C module 0 data. |
| I2C1SCL | 23 | PA6 (3) | I/O | OD | I$^2$C module 1 clock. Note that this signal has an active pull-up. The corresponding port pin should not be configured as open drain. |
| I2C1SDA | 24 | PA7 (3) | I/O | OD | I$^2$C module 1 data. |
| I2C2SCL | 59 | PE4 (3) | I/O | OD | I$^2$C module 2 clock. Note that this signal has an active pull-up. The corresponding port pin should not be configured as open drain. |
| I2C2SDA | 60 | PE5 (3) | I/O | OD | I$^2$C module 2 data. |
| I2C3SCL | 61 | PD0 (3) | I/O | OD | I$^2$C module 3 clock. Note that this signal has an active pull-up. The corresponding port pin should not be configured as open drain. |
| I2C3SDA | 62 | PD1 (3) | I/O | OD | I$^2$C module 3 data. |

a. The TTL designation indicates the pin has TTL-compatible voltage levels.

OD: Open-Collector

# I²C Modules in Tiva LaunchPad



- For Tiva LaunchPad:
  - SDA must be configured as open-drain/open-collector.
  - SCL must not be configured to be open-drain (although it functions as one).
  - Both SDA & SCL must be connected to supply voltage through tie-up resistors.

# I$^2$C Data Structure & Initialization

I2CMPTR, I2CMCR, I2CDR, I2CMSA, I2CMCS Registers

# I$^2$C Data Structure

[file: TM4C123GH6PM7.h]

| | |
|---|---|
| 0x4002.0000 | MSA |
| 0x4002.0004 | MCS |
| 0x4002.0008 | MDR |
| 0x4002.000C | MTPR |
| 0x4002.0010 | MIMR |
| 0x4002.0014 | MRIS |
| 0x4002.0018 | MMIS |
| 0x4002.001C | MICR |
| 0x4002.0020 | MCR |
| 0x4002.0024 | MCLKOCNT |
| 0x4002.0028 | *(Reserved)* |
| 0x4002.002C | MBMON |
| | *(Reserved)* |
| 0x4002.0038 | MCR2 |
| | *(Reserved)* |
| 0x4002.0800 | SOAR |
| 0x4002.0804 | SCSR |
| 0x4002.0808 | SDR |
| 0x4002.080C | SIMR |
| 0x4002.0810 | SRIS |
| 0x4002.0814 | SMIS |
| 0x4002.0818 | SICR |
| 0x4002.081C | SOAR2 |
| 0x4002.0820 | SACKCTL |
| | *(Reserved)* |
| 0x4002.0FC0 | PP |
| 0x4002.0FC4 | PC |

```
typedef struct {            /* I2C0 Structure           */
  __IO uint32_t  MSA;       /* I2C Master Slave Address */
  union {
    __IO uint32_t  MCS_I2C0_ALT; /* I2C Master Control/Status */
    __IO uint32_t  MCS; };       /* I2C Master Control/Status */
  __IO uint32_t  MDR;            /* I2C Master Data          */
  __IO uint32_t  MTPR;           /* I2C Master Timer Period  */
  __IO uint32_t  MIMR;           /* I2C Master Interrupt Mask */
  __IO uint32_t  MRIS;           /* I2C Master Raw Interrupt Status    */
  __IO uint32_t  MMIS;           /* I2C Master Masked Interrupt Status */
  __O  uint32_t  MICR;           /* I2C Master Interrupt Clear */
  __IO uint32_t  MCR;            /* I2C Master Configuration   */
  __IO uint32_t  MCLKOCNT;       /* I2C Master Clock Low Timeout Count */
  __I  uint32_t  RESERVED;
  __IO uint32_t  MBMON;          /* I2C Master Bus Monitor     */
  __I  uint32_t  RESERVED1[2];
  __IO uint32_t  MCR2;           /* I2C Master Configuration 2 */
  __I  uint32_t  RESERVED2[497];
  __IO uint32_t  SOAR;           /* I2C Slave Own Address      */
  union {
    __IO uint32_t  SCSR_I2C0_ALT;/* I2C Slave Control/Status   */
    __IO uint32_t  SCSR; };      /* I2C Slave Control/Status   */
  __IO uint32_t  SDR;            /* I2C Slave Data                 */
  __IO uint32_t  SIMR;           /* I2C Slave Interrupt Mask       */
  __IO uint32_t  SRIS;           /* I2C Slave Raw Interrupt Status */
  __IO uint32_t  SMIS;           /* I2C Slave Masked Interrupt Status */
  __O  uint32_t  SICR;           /* I2C Slave Interrupt Clear      */
  __IO uint32_t  SOAR2;          /* I2C Slave Own Address 2        */
  __IO uint32_t  SACKCTL;        /* I2C Slave ACK Control          */
  __I  uint32_t  RESERVED3[487];
  __IO uint32_t  PP;             /* I2C Peripheral Properties      */
  __IO uint32_t  PC;             /* I2C Peripheral Configuration   */
} I2C0_Type;
```

# I$^2$C Data Structure

[file: TM4C123GH6PM7.h]

```
/** I2C memory map   **/
#define I2C0_BASE   0x40020000UL
#define I2C1_BASE   0x40021000UL
#define I2C2_BASE   0x40022000UL
#define I2C3_BASE   0x40023000UL

/** I2C declaration   **/
#define I2C0   ((I2C0_Type  *) I2C0_BASE)
#define I2C1   ((I2C0_Type  *) I2C1_BASE)
#define I2C2   ((I2C0_Type  *) I2C2_BASE)
#define I2C3   ((I2C0_Type  *) I2C3_BASE)
```

```
/** accessing I2C registers  **/
I2C0->MDR = 0x14;
I2C0->MCS |= 1UL <<4    /* set bit 4  */
```

# I²C Clock Rate

In the Tiva LaunchPad, the following formula determines the I²C bus speed in Standard and Fast Mode:

$$SCL_{Period} = 2 \times (1 + \boldsymbol{TPR}) \times (SCL_{LP} + SCL_{HP}) \times SysClk\_PRD$$

*(TPR is a value in* ***I2CMTPR*** *register; value from 1 to 127)*

$SCL_{Period} = I^2C\ clock\ period$

$\boldsymbol{TPR} = Timer\ Period$

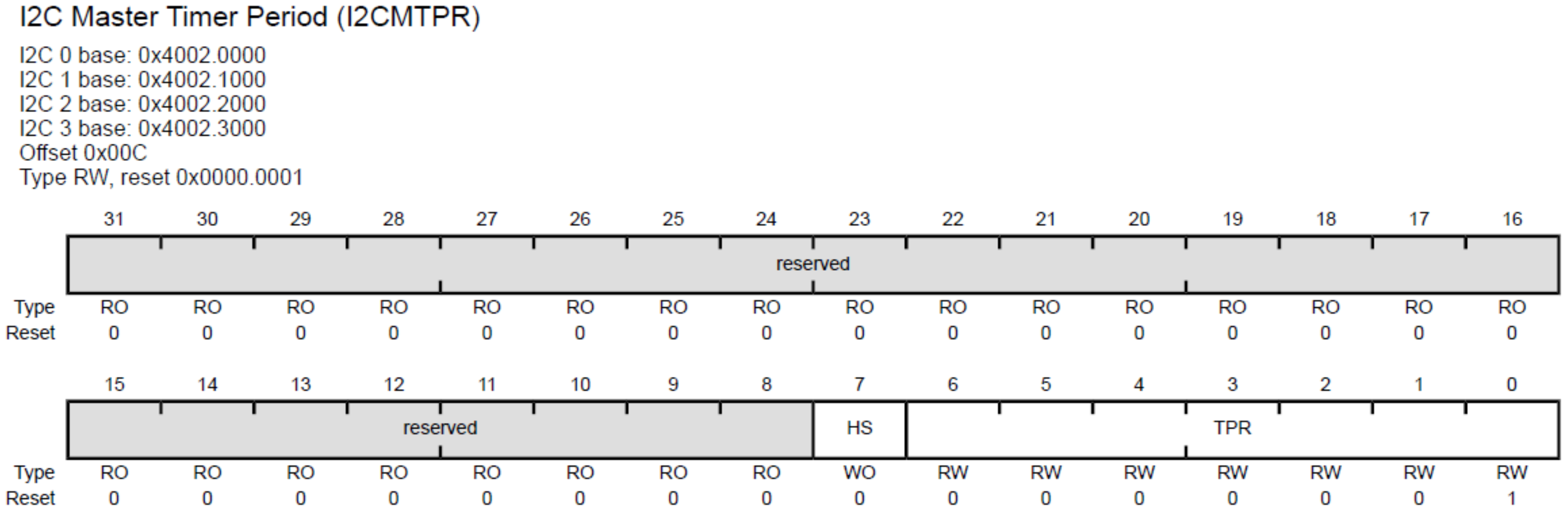
$SCL_{LP} = SCL\ Low\ period\ (= 6\ (fixed\ value))$

$SCL_{HP} = SCL\ High\ period\ (= 4\ (fixed\ value))$

$SysClk_{PRD} = System\ Clock\ period\ in\ ns$

**TPR** value written to the **I2CMTPR** register represents the number of system clock periods in one SCL clock period.

# I$^2$C Master Timer Period Register (I2CMTPR)



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 |

**TPR** = Timer Period, range from 1 to 127 (7-bits). Value computed from equation in previous slide.

**HS:** To set I$^2$C speed.

"0": Standard (100Kbps), Fast-mode (400Kbps), Fast-mode plus (1Mbps).

"1": High-speed (3.34 Mbps)

Source: TM4C123GH6PM Datasheet (spms376e), pg 1026

# Ex 1: I$^2$C Clock Rate

Example: On the Tiva LaunchPad, what value of **TPR** must be programmed to the **I2CMTPR** register for an I$^2$C bus speed of 100 KHz (Standard mode)?

Assume system bus speed to be 20 MHz.

$$SCL_{Period} = 2 \times (1 + \boldsymbol{TPR}) \times (SCL_{LP} + SCL_{HP}) \times SysClk\_PRD$$

$$SCL_{Period} = \frac{1}{100\ KHz} = 2 \times (1 + TPR) \times (6 + 4) \times 50ns$$

$$TPR + 1 = \frac{1}{(100KHz \times 2 \times 10 \times 50ns)} = 10$$

$$TPR = 10 - 1 = 9$$

We load **TPR** = 0x09 & set **HS** = '0' in **I2CMTPR** register to obtain an I$^2$C bus speed of 100KHz (Standard mode).

# Ex 2: I$^2$C Clock Rate

Example: On the Tiva LaunchPad, what value of **TPR** must be programmed to the **I2CMTPR** register for an I$^2$C bus speed of 400 KHz (Fast mode)? Assume system bus speed to be 80 MHz.

$$SCL_{Period} = 2 \times (1 + \boldsymbol{TPR}) \times (SCL_{LP} + SCL_{HP}) \times SysClk\_PRD$$

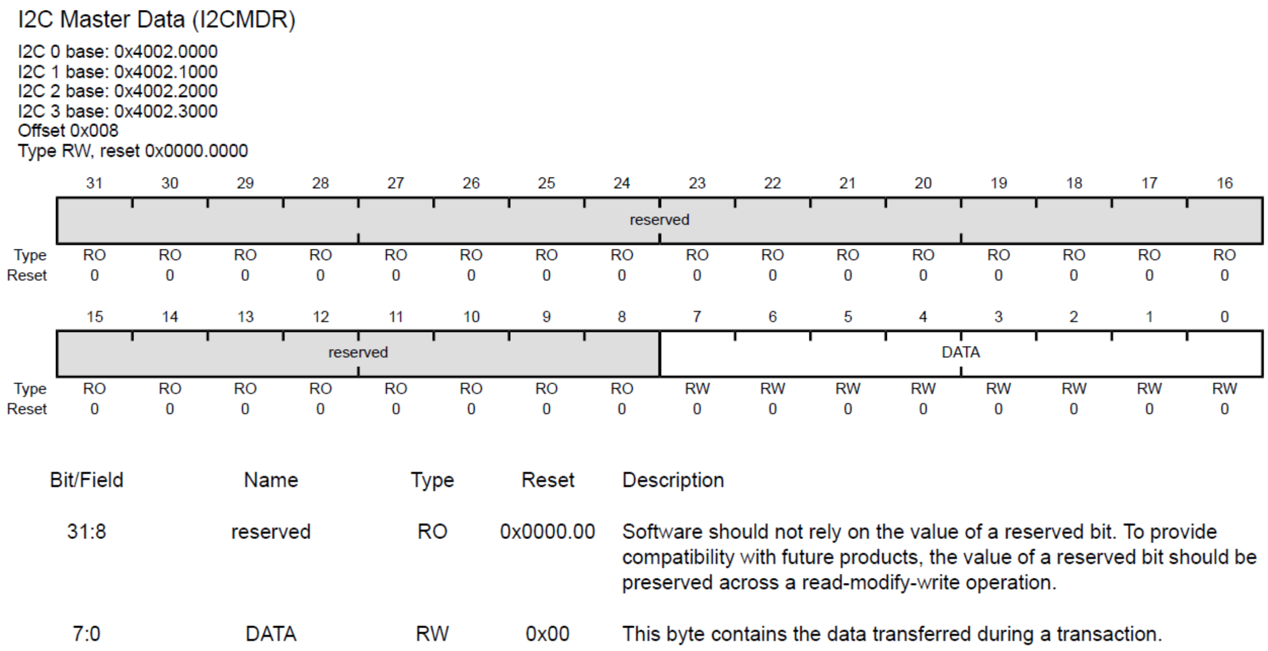
$$SCL_{Period} = \frac{1}{400\ KHz} = 2 \times (1 + TPR) \times (6 + 4) \times 12.5ns$$

$$TPR + 1 = \frac{1}{(400KHz \times 2 \times 10 \times 12.5ns)} = 10$$

$$TPR = 10 - 1 = 9$$

We load **TPR** = 0x09 & set **HS** = ‘0’ in **I2CMTPR** register to obtain an I$^2$C bus speed of 400KHz (Fast Mode).

# I$^2$C Master Data Register (I2CMDR)



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 |

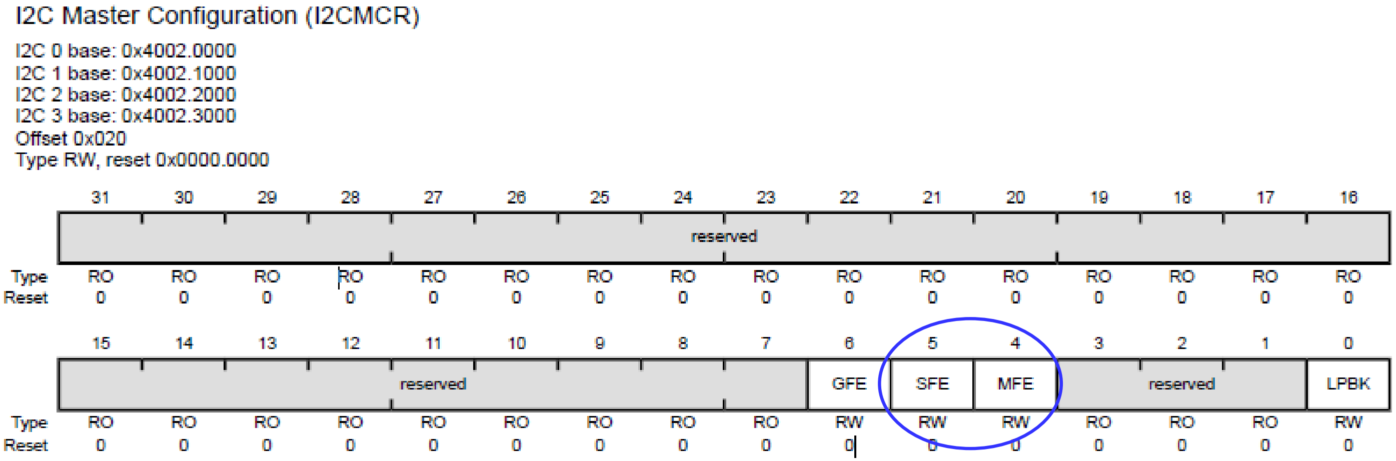
| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | DATA | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset | 0 | 0 |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

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x0000.00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | DATA | RW | 0x00 | This byte contains the data transferred during a transaction. |

**DATA:** Contains data during Master Transmit/Receive cycles.

Source: TM4C123GH6PM Datasheet (spms376e), pg 1025

# I$^2$C Master Configuration Register (I2CMCR)



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

| | 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 |

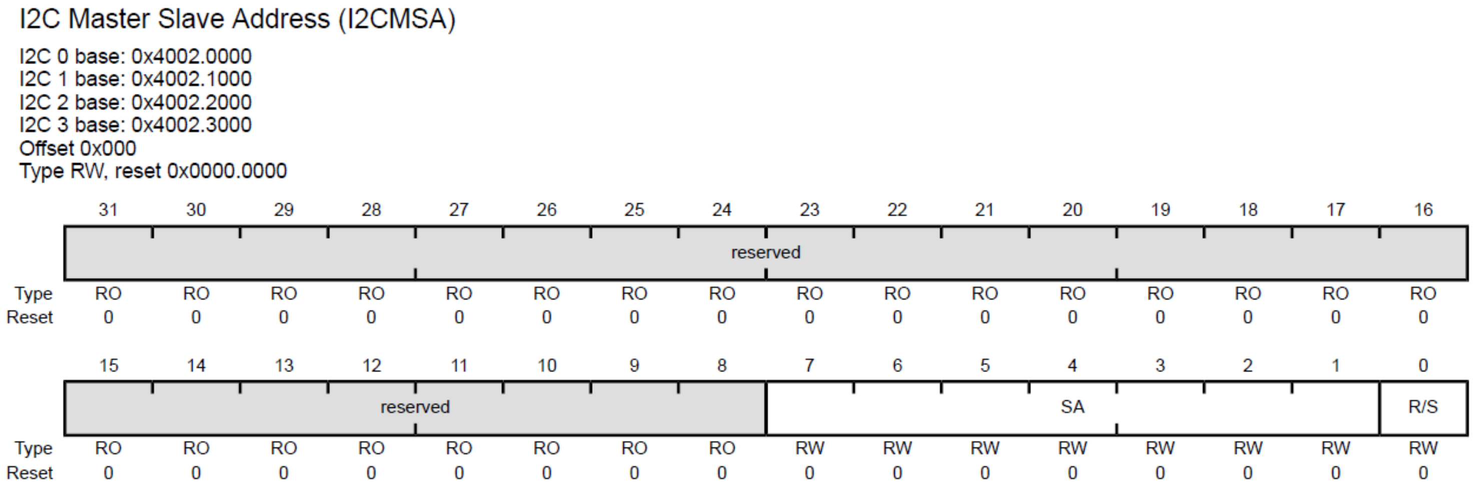
| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | GFE | SFE | MFE | reserved | | | LPBK |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RW | RW | RW | RO | RO | RO | RW |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**MFE:** Master Function Enable.

**SFE**: Slave Function Enable.

Source: TM4C123GH6PM Datasheet (spms376e), pg 1031

# $I^2C$ Master Slave Address Register (I2CMSA)



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 |

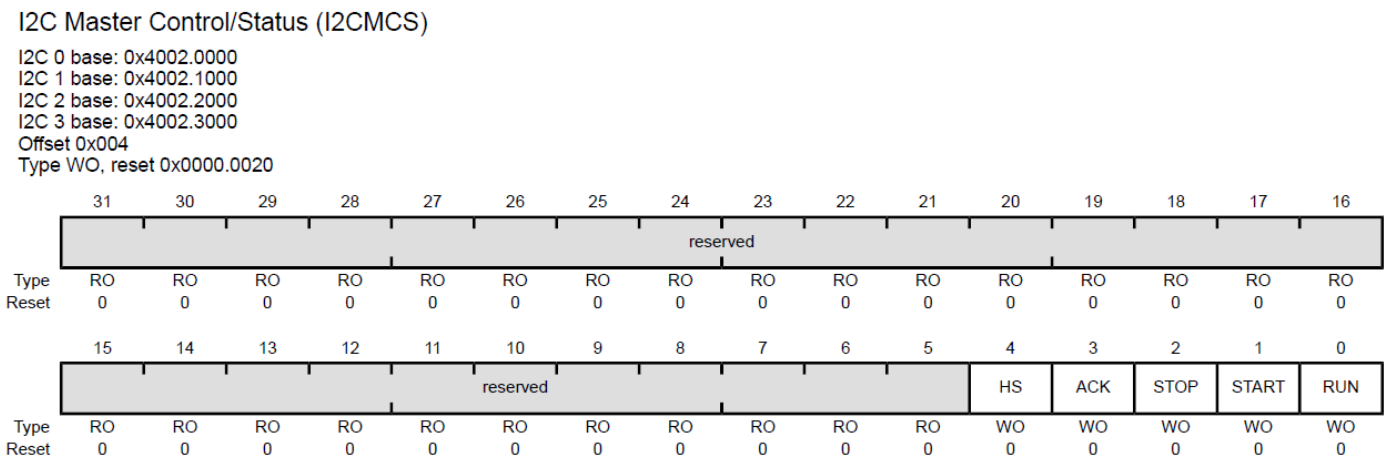
| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | SA | | | | | | | R/S |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset | 0 | 0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

**SA:** 7-bit (A6 to A0) of Slave Address.

**R/S**: ‘1’ – next Master operation is a Receive cycle; ‘0’ – Transmit operation.

Source: TM4C123GH6PM Datasheet (spms376e), pg 1019

# I$^2$C Master Control/Status Register (I2CMCS) - WRITE



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

| | 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 | ACK | STOP | START | RUN |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | WO | WO | WO | WO | WO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**I2CMCS** register behaves differently during READ or WRITE:

- READ: provides status bits to indicate state of I$^2$C module.
- WRITE: configures I$^2$C controller module.

**START:** Setting bit generates a START or REPEATED START condition.

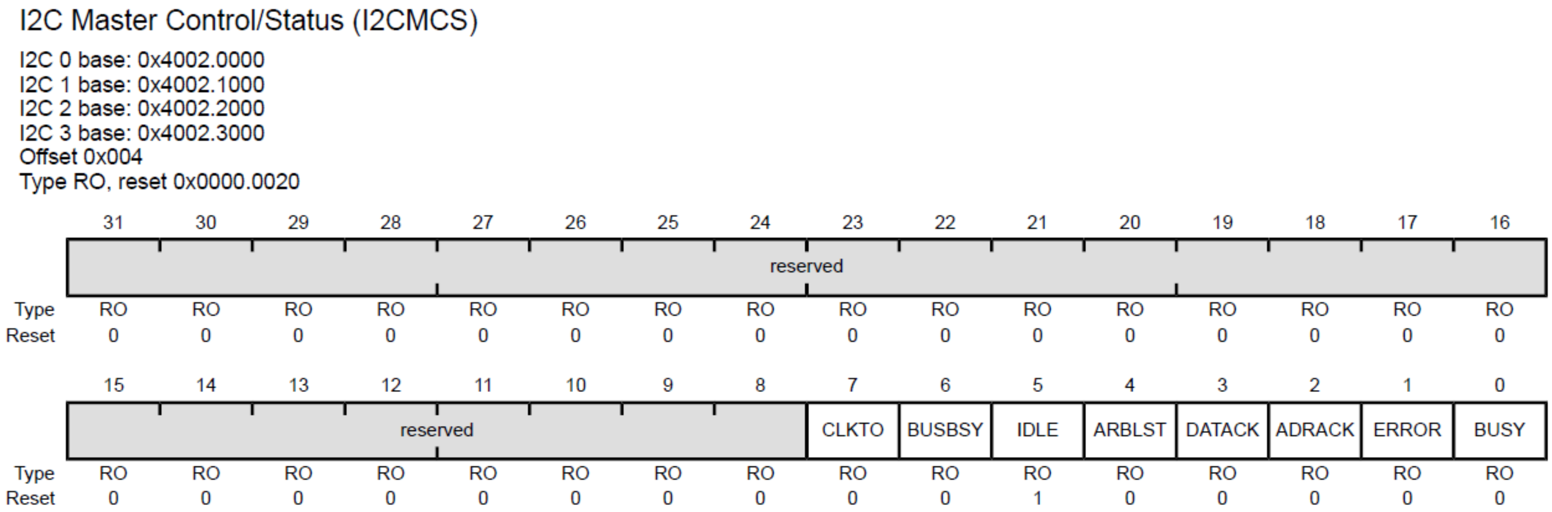
**STOP**: Setting bit generates a STOP condition at end of current I2C transmission.

**HS**: Setting bit enables I2C High-Speed mode.

**ACK**: Setting bit enables ACK bit to be sent automatically by the Master.

Source: TM4C123GH6PM Datasheet (spms376e), pg 1022

# I$^2$C Master Control/Status Register (I2CMCS) - READ



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 |

**BUSY:** ‘1’ – I2C controller is busy.

**ERROR**: ’1’ – error occurred during last operation.

**ARBLST**: ’1’ – I2C controller has lost arbitration.

**BUSBSY**: ’1’ – I2C bus is busy.

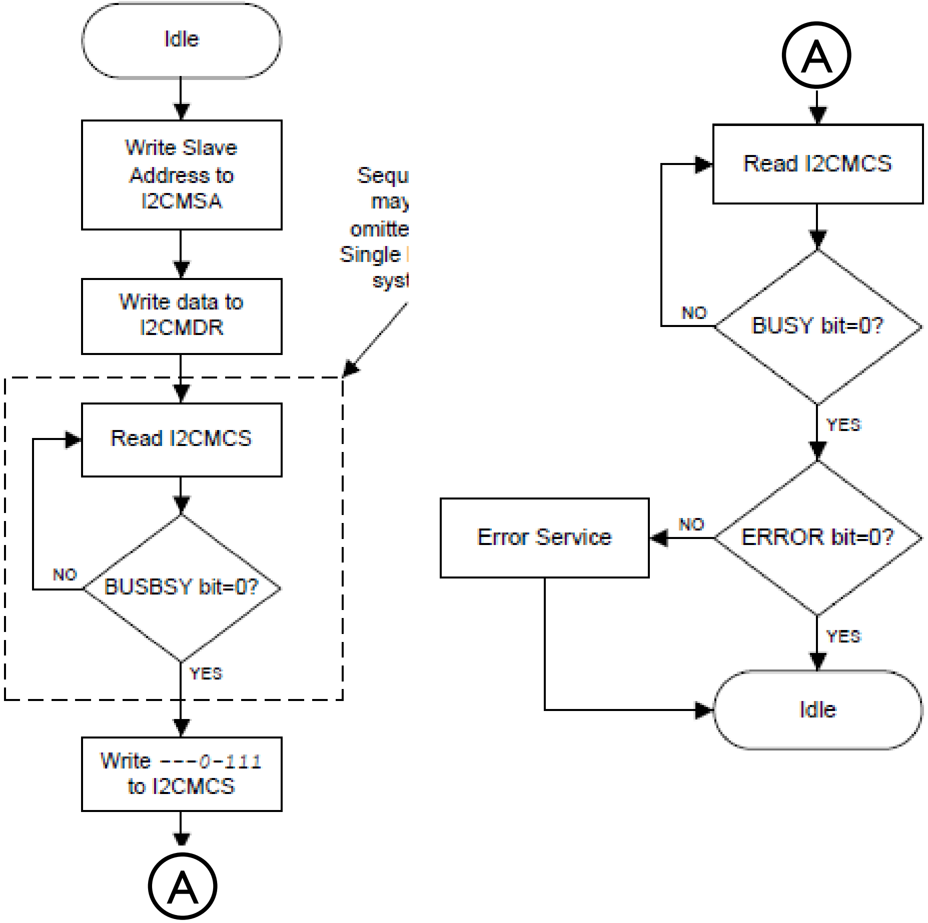
Source: TM4C123GH6PM Datasheet (spms376e), pg 1020

See complete list of I$^2$C registers in datasheet.

# I$^2$C Master *(Transmit Single Byte)*

I$^2$C Single Byte Transmit:

- Write Slave Address & clear R/S in **I2CMSA**.
- Write send data to **I2CMDR**.
- Set bits START=1, STOP=1,ACK=X (0 or 1) in **I2CMCS [Write]**.
- [Data will now be sent by I2C module]
- Check if data has been sent through BUSY flag in **I2CMCS [Read]**.
- Check ERROR bit in **I2CMCS [Read]** if there is transmission error.



# I$^2$C Initialization & Send

Steps to initialize an **I$^2$C** module & Send a byte:

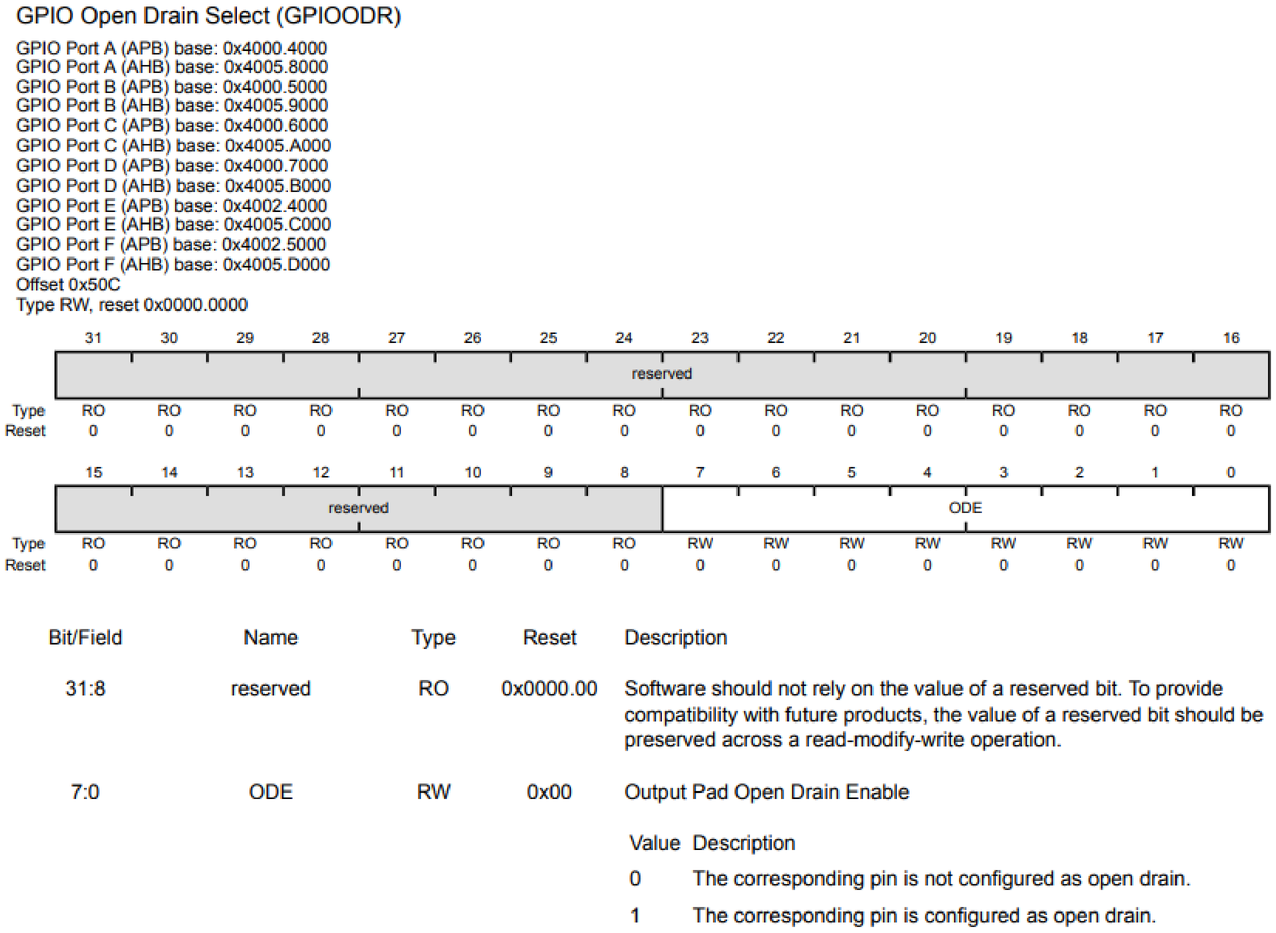
- Configure GPIO to alternate pin function (Steps 1 -4 )
- Configure I$^2$C module (Steps 5 – 6).
- Send byte through I$^2$C module (Steps 7 – 11).

1. Enable the I2C clock using the **RCGCI2C** register in the System Control module *(pg 348 of datasheet)*.
2. Enable the clock to the appropriate GPIO module via the **RCGCGPIO** register in the System Control module *(pg 340)*.
3. In the GPIO module, enable the appropriate pins for their alternate function through **GPIOAFSEL** register *(pg 671)*.
4. Configure the PMC*n* fields in the **GPIOPCTL** register to assign the I2C signals to the appropriate pins.
5. Initialize the I2C Master by setting **MFE** (Master Function Enable] bit in **I2CMCR** register.
6. Set the desired SCL clock speed by writing the **I2CMTPR** register with the correct value.

# I$^2$C Initialization & Send

7. Specify the Slave address of the master and that the next operation is a Transmit (**R/S** bit) by writing the **I2CMSA** register. E.g: With value of 0x0000.0076 in I2CMSA, it sets the slave address to 0x3B.
8. Place data (byte) to be transmitted in the data register by writing the **I2CMDR** register.
9. Initiate a single byte transmit of the data from Master to Slave by writing the **I2CMCS** register with a value of 0x0000.0007 (set **STOP**, **START**, **RUN** bits).
10. Wait until the transmission completes by polling the **I2CMCS** register's **BUSBSY** bit until it has been cleared.
11. Check the ERROR bit in the **I2CMCS** register to confirm the transmit was acknowledged.

# GPIO Open Drain Select Register (GPIO ODR)



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 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x0000.00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | ODE | RW | 0x00 | Output Pad Open Drain Enable |

| Value | Description |
|---|---|
| 0 | The corresponding pin is not configured as open drain. |
| 1 | The corresponding pin is configured as open drain. |

Source: TM4C123GH6PM Datasheet (spms376e), pg 676

# I$^2$C Initialization *(Example)*

```
#define PB_I2C0_SCL 2U   // PB2
#define PB_I2C0_SDA 3U   // PB3

void Port_Init( void ) /* initialize I2C0  */
{
     SYSCTL->RCGCGPIO |= SYSCTL_RCGCGPIO_R1; // enable clock Port B
     SYSCTL->RCGCI2C |= SYSCTL_RCGCI2C_R0;   // enable clock to I2C0 module
     while( 0 == (SYSCTL->PRGPIO & SYSCTL_PRGPIO_R1)){};
     while( 0 == (SYSCTL->PRI2C & SYSCTL_RCGCI2C_R0)){};

     /** I2C0 is mapped to Port B:       **/
     /** I2C0SCL = PB2; I2C0SDA = PB3    **/
     GPIOB->AFSEL |= BIT(PB_I2C0_SCL) |   // enable alt function
                     BIT(PB_I2C0_SDA);
     GPIOB->ODR |= BIT(PB_I2C0_SDA);      // enable open-drain on SDA
     GPIOB->DEN |= BIT(PB_I2C0_SCL) |     // enable output
                   BIT(PB_I2C0_SDA);
     /* disable analog function  */
     GPIOB->AMSEL &= ~(BIT(PB_I2C0_SCL)|BIT(PB_I2C0_SDA));

     /** configure pins for I2C     **/
     GPIOB->PCTL &= ~GPIO_PCTL_PB2_M;       // PB2 mask
     GPIOB->PCTL |= GPIO_PCTL_PB2_I2C0SCL;  // configure to SCL
     GPIOB->PCTL &= ~GPIO_PCTL_PB3_M;       // PB3 mask
     GPIOB->PCTL |= GPIO_PCTL_PB3_I2C0SDA;  // configure to SDA
```

*..... continue in next slide*

# I$^2$C Initialization *(Example)*

*..... from previous slide*

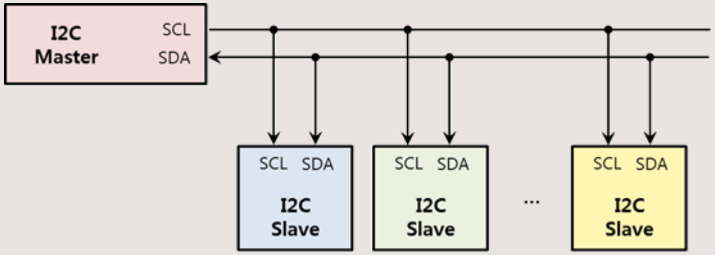
```
     /** enable I2C0 functions                                  **/
     /*   at 100 Kbps, period = 10us                             */
     /*   SCL_PERIOD = 2*(1+TPR)*(6+4)*(1/SystemCoreClock)      */
     /*   10us = 2*(1+TPR)*10*12.5ns  (80MHz sys clk)          */
     /*   TPR = 40-1 = 0x27                                     */
     I2C0->MCR |= I2C_MCR_MFE;             // enable Master function
     I2C0->MTPR = (I2C_MTPR_TPR_M & 0x27); // write TPR value
     I2C0->MTPR &= ~I2C_MTPR_HS;           // HS = 0
} // end Port_Init()
```

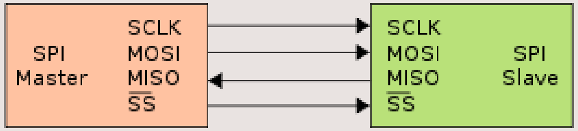
# Serial Communications & Protocols …

Summary

# I$^2$C & SPI Differences

| I$^2$C | SPI |
|---|---|
| Supports multiple Masters. | One Master at a time. |
| 2-wire interface. No CE or CS signal to Slave device. | 4-wire interface |
| Half-duplex | Full-duplex |
| More complex protocol. | Lower SW overhead for data transmission. |
| Support clock stretching (allows slower slaves) | Fixed clock rate. |
| Developed by Phillips.<br>Has official specification. | Developed by Motorola.<br>No official specification (vendor-dependent). |





# Serial Protocols: A Summary

- **UART**:
  - UART link can be achieved with only 3 signals (Tx, Rx, GND) though Control signals are defined (RTS, CTS, DTE, …).
  - Tx & Rx signals need to be criss-crossed between 2 devices.
  - UART is used to implement the RS232 interface.
  - RS232 signals levels can range from ±12V. However UART signals from CPU are usually ±3V.
    - Voltage level shifter is needed. Higher RS232 signals enables data communications over longer distances.
- **SPI**:
  - 4-wire interface: MISO, MOSI, SCLK, SS.
  - MISO & MOSI need to be criss-crossed between Master & Slave.
  - SPI is a synchronous protocol. Master provides the clock; clock need not be precise as long as it is common to all device.
  - SS lines serves as a CS – therefore, multiple SPI peripherals can be supported.
  - Protocol is relatively simpler (compared to $I^2C$) to implement in SW. Can also be implemented a *bit-bang* interface.

# Serial Protocols: A Summary …

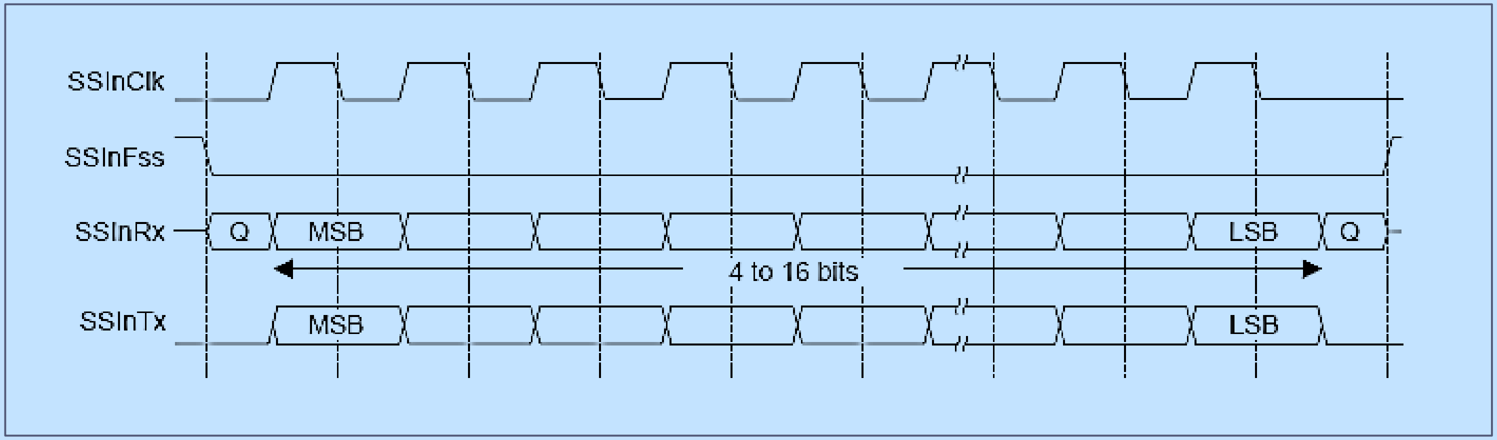
- **I$^2$C**:
  - 3-wire interface: SCK, SDA, GND. Is a synchronous protocol.
  - Like SPI, the Master provides the clock.
  - Unlike SPI, Bus Master can be changed or rotated.
  - SDA goes both ways (on a single line). Hence, it is a half-duplex protocol.
  - Protocol is more complex to implement compared to SPI – due partly to multiple Master & Slaves & switching of Masters. Usually implemented as a state machine.
  - Standard speed is 100 Kbps, Faster speeds at 400 Kbps, 1 Mbps, 3.4 Mbps.
  - Like SPI, the Master sends the Clock => Clock need not be precise.

# Review Questions – I$^2$C, SPI

1. Is the I$^2$C bus asynchronous?
2. Do I$^2$C support multiple Masters?
3. How many I$^2$C slave can be supported in a 7-bit address scheme?
4. What is the defined clock speed for Standard and Fast modes?
5. Does the I$^2$C clock frequency need to be exactly at the defined Standard or Fast modes?
6. What are the possible I$^2$C Modes? Name them and describe them briefly.
7. Explain how Repeated START is used. What situations warrants use of Repeated START?
8. For the Tiva LaunchPad, what value of **TPR** must be programmed to the **I2CMTPR** register to implement an I$^2$C bus speed of 400 KHz (Fast mode)? Assume system bus speed to be 40 MHz.
9. For the Tiva LaunchPad, what value of **TPR** must be programmed to the **I2CMTPR** register to implement an I$^2$C bus speed of 100 KHz (Standard mode)? Assume system bus speed to be 80 MHz.

# Review Questions - I$^2$C, SPI

10. When 2 I$^2$C Masters are trying to send data to the bus, how do they each Master determine if it has control of the bus and thus able to send? *[Arbitration process]*
11. I$^2$C protocol can support 10-bit addresses. How are 10 bit addresses sent on the SDA signal. *(Find out, this is not in the lecture notes).*
12. Which SPI Mode is the following SPI transaction? Reference the Mode in terms of the SPO and SPH bit logic levels.



# Review Questions – I$^2$C, SPI

13. What are the 4 SPI modes? List them and describe their differences.
14. Does each SPI Slave have individual addresses?
15. How many SSI modules do the Tiva LaunchPad has? Name them.
16. Which register and bit is used to enable a SPI module?
17. Which register is used to enable the clock to a SSI module? What would be a reason for implementing such a feature?
18. Which SSI register is used to check if a SSI module's register is ready for Read/Write operations?
19. If you wish to set the SSI module for SPI clock of 4 MHz transmit/receive rates, what values of **CPSDVSR** & **SCR** would you use?
20. What is the base address for the SSI3 module?

# I$^2$C Clock Timing

Source: I2C Specifications, V3.0, pg 40

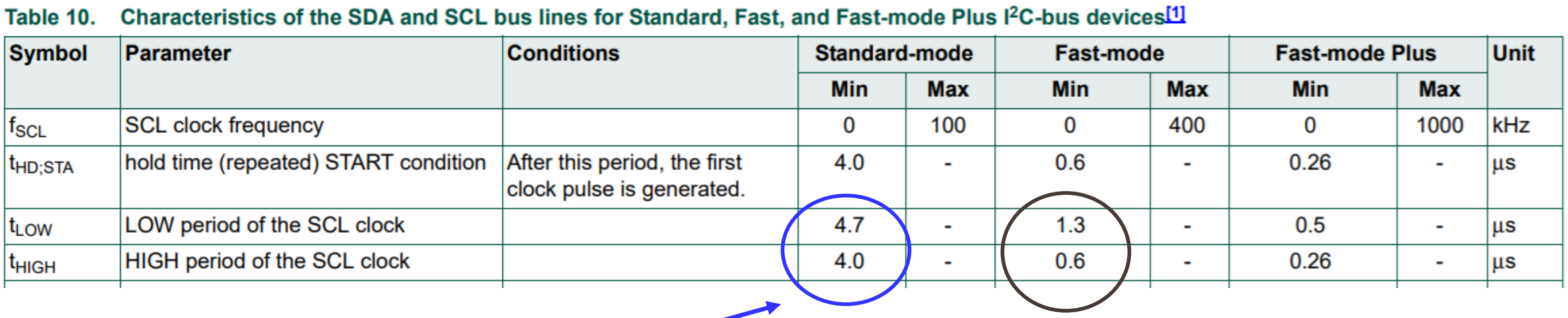


Table 10. Characteristics of the SDA and SCL bus lines for Standard, Fast, and Fast-mode Plus I$^2$C-bus devices[1]

| Symbol | Parameter | Conditions | Standard-mode | | Fast-mode | | Fast-mode Plus | | Unit |
|---|---|---|---|---|---|---|---|---|---|
| | | | Min | Max | Min | Max | Min | Max | |
| $f_{SCL}$ | SCL clock frequency | | 0 | 100 | 0 | 400 | 0 | 1000 | kHz |
| $t_{HD;STA}$ | hold time (repeated) START condition | After this period, the first clock pulse is generated. | 4.0 | - | 0.6 | - | 0.26 | - | µs |
| $t_{LOW}$ | LOW period of the SCL clock | | 4.7 | - | 1.3 | - | 0.5 | - | µs |
| $t_{HIGH}$ | HIGH period of the SCL clock | | 4.0 | - | 0.6 | - | 0.26 | - | µs |

**Standard mode** (100K bps)

- SCL Low: 4.7us/10.0us = 0.47 ratio (min)
- SCL High: 4.0us/10.0us = 0.4 ratio (min)
- Tiva implements ratio of Low : High = 6:4 (which is within I$^2$C specifications)

**Fast mode** (400K bps)

- SCL Low: 1.3us/2.5us = 0.52 ratio (min)
- SCL High: 0.6us/2.5us = 0.24 ratio (min)
- Tiva implements ratio of Low : High = 6:4 (which is within I$^2$C specifications)