EE 3171 Lecture 17

The Inter-Integrated Circuit Bus

IIC or I2C or I^{2}C or ...

Lecture Overview

- History, Goals, Definitions
- Bus Structure: only 2 lines
- Multi-Master Bus Protocol
    - Technology
    - Bit level, byte level, & packet level conventions
- Complete Packet Transfers – read & write
- Flow Control
- Bus Master Arbitration
- A Few Enhancements

IIC Origins

Originated by Philips Semiconductors in 1982

- Written to Standardize interfacing between ICs
- There have been several revisions to the standard since

Objectives

- short range communication (chip-to-chip)
- low data-rate (100 kbps)
- keep it simple – minimize chip real-estate
- keep it cheap – minimize pins

Synchronous like SPI, but simpler

- The standard predates the growth of SPI

I2C is wonderfully elegant in its simplicity (my opinion)

IIC Introduction

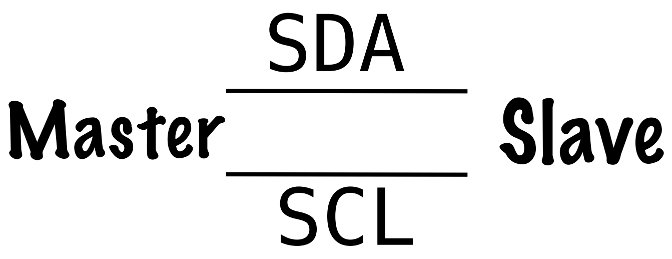
IIC ≡ Inter-Integrated Circuit Bus

Very Simple Protocol:

- Half-Duplex
- Synchronous
- Master-Slave Protocol
  - Actually, a shared “multi-master” protocol

Conceptual Model: Only 2 lines between chips

- `SDA` = Serial Data Line (bidirectional)
- `SCL` = Serial Clock Line (unidirectional)



Multi Master Bus

- Allows multiple masters to operate on the bus
  - Only one at a time, of course ⇒
  - Need an arbitration scheme to decide on current master
- Multi-master bussing
  - Two Master-wannabes could start transmitting simultaneously
  - Must electrically tolerate simultaneous transmitters
  - Assume one bus value is *dominant* over the other
    - if both values are asserted (by different transmitters)
    - then the bus transmits the dominant value
      - Wired-OR ⇒ 1 is dominant over 0
      - Wired-AND ⇒ 0 is dominant over 1

Open Collector Bus

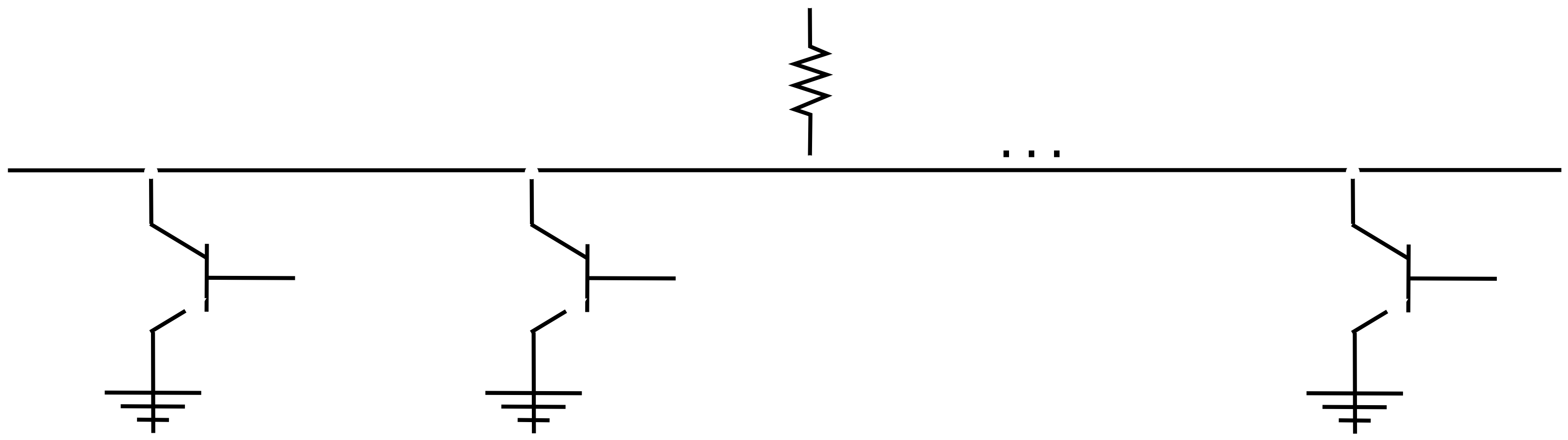
Also called Open Drain.

Allows multiple drivers to disagree without damage

- Quiescent State = 1 (single pull-up resistor)
- Any driver can assert a 0 by shorting the line to ground
- Net effect: Bus value = 1, IFF no drivers are asserting 0

Signal Dominance = Wired-AND

- 0 is the dominant value ⇒ even a single 0 overrides all 1s
- 1 is the recessive value ⇒ exists only if no one asserts 0

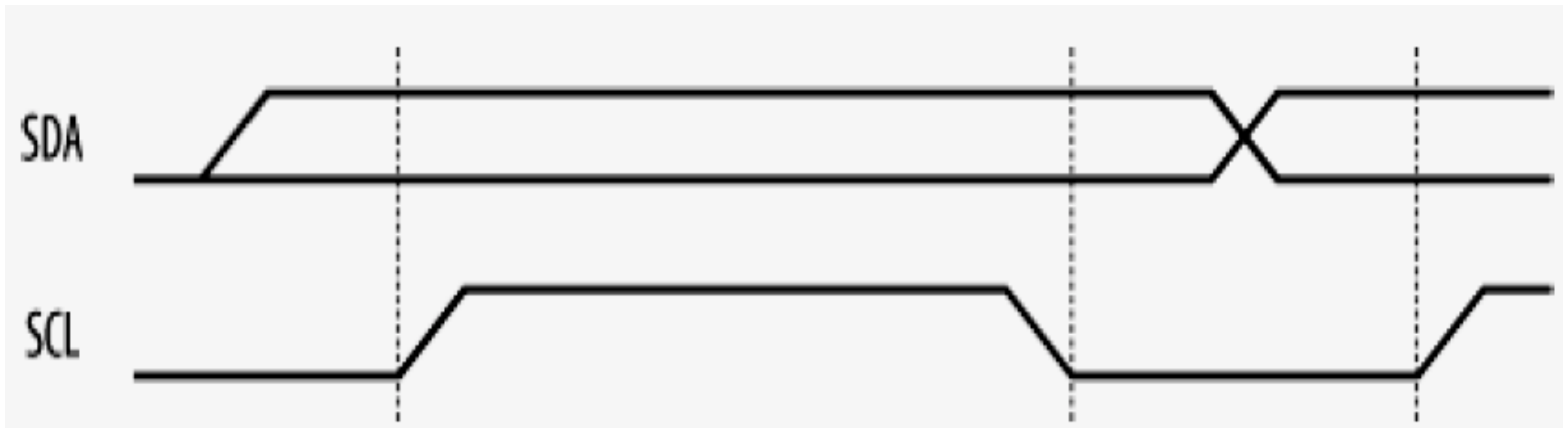


I2C Bit-Signaling Convention

One data bit on **SDA** lasts 1 **SCL** clock period

Rules for Normal Data Bits:

- Transmitter may change **SDA** value only while **SCL** = 0
- Receivers read **SDA** value on rising edge of **SCL**

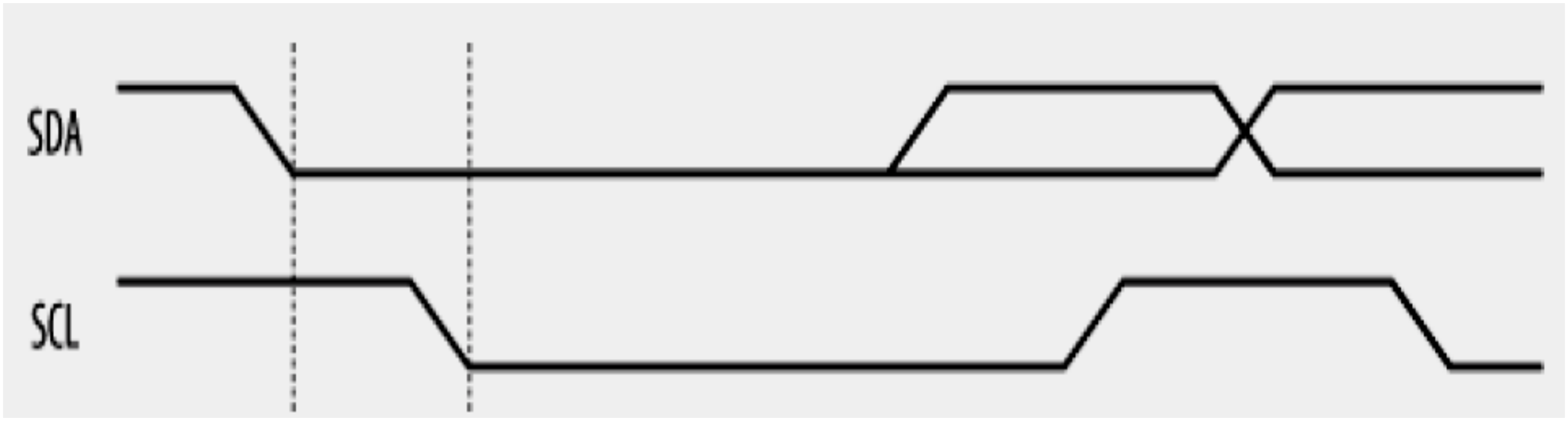


Starting a Data Packet

Quiescent State: both `SDA` and `SCL` = 1

Start Bit (`S`-Bit)

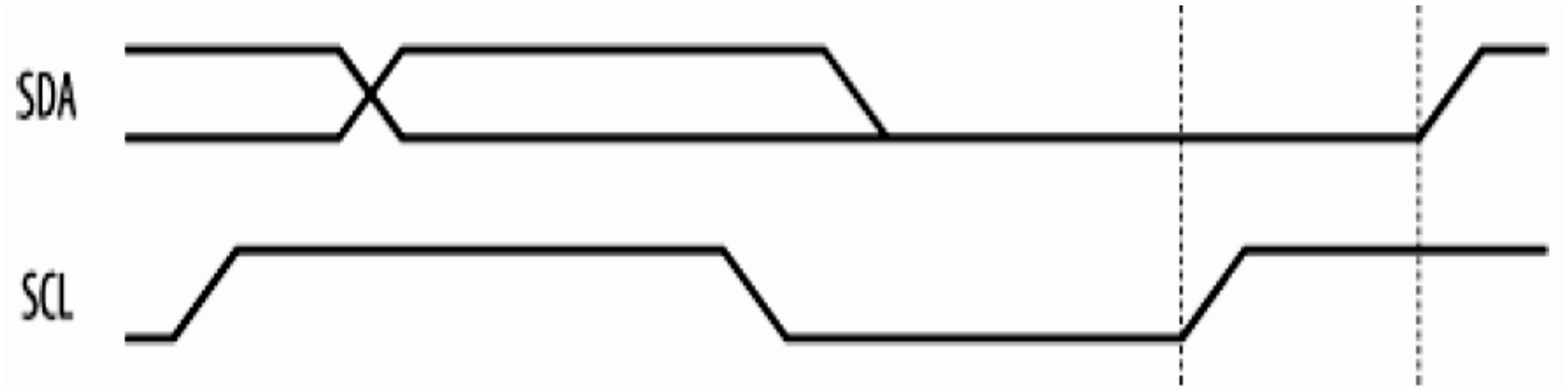
- Master asserts `SDA` = 0 while `SCL` is still 1
  - Violates the rule of changing `SDA` only while `SCL` = 0
  - Distinguishes a start bit from a normal bit
- Master asserts `SCL` = 0
- Master asserts 1st `SDA` bit before next `SCL` rising edge



Terminating a Data Packet

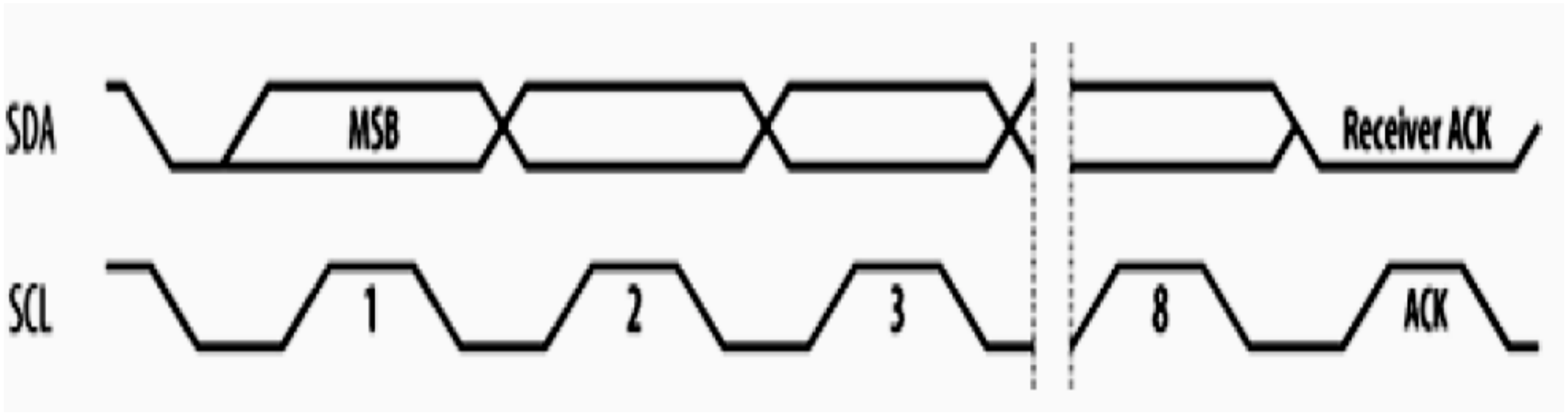
Stop Bit (**P**-Bit)

- Master asserts **SDA** = 0 while **SCL** = 0 (normal behavior)
- Master asserts **SCL** = 1
- Master asserts **SDA** = 1 while **SCL** = 1
  - Violates the rule of changing **SDA** only while **SCL** = 0
  - Distinguishes a stop bit from a normal bit



I2C Byte Signaling Convention

- Absence of any control lines ⇒ need in-channel signaling
- All transfers are in multiples of a Byte
  - Each Byte is transferred MSb first —— That's "Most Significant bit"
  - 9th bit is an Acknowledgement (ACK) bit from the receiver
    - Transmitter releases SDA bus (it floats to 1)
    - Receiver asserts SDA = 0 to acknowledge receipt of byte
    - Master asserts SCL = 1 and reads acknowledgment
  - If receiver sends SDA = 1, that's a "No Ack" (NACK)



Data Packet Structure

Packet Contents:

- Packet Start bit (`S`) Master → Slaves
- Address/Direction Byte Master → Slaves
- One or more Data Bytes Transmitter → Receivers
- Packet Stop bit (`P`) Master → Slaves

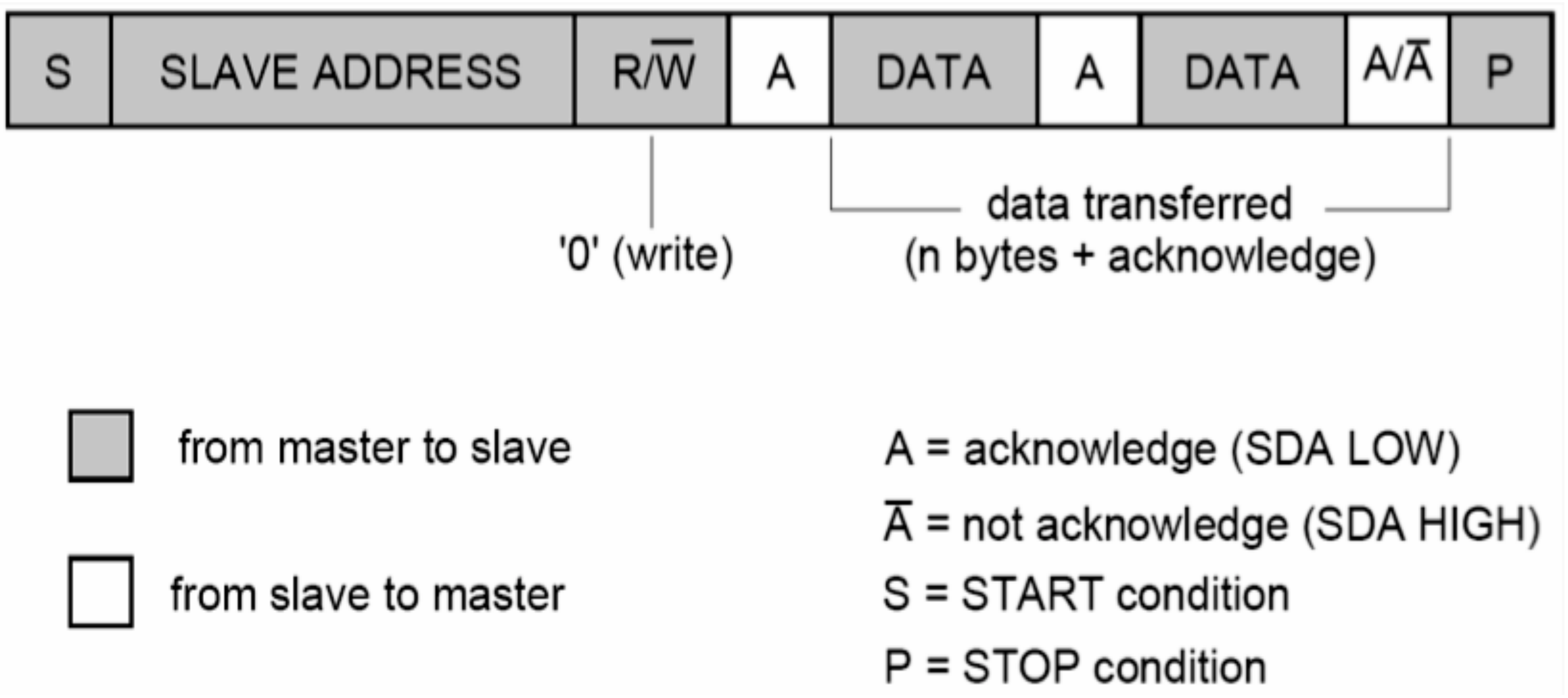
Each Byte is terminated by an `ACK` bit (`A`)

- Flows opposite direction from the byte it ends (Receivers → Transmitter)
- IF `A` = 1, (`NACK`) it usually causes the transmitter to terminate the entire packet at that point

Data Packet Structure

- Address/Direction Byte – Always Asserted by Master
  - First 7 bits = Slave Address
    - 8 addresses are reserved for special uses
      - e.g. `0000,000` = broadcast (All Call)
    - Allows for up-to 120 devices on the bus
  - 8th bit = Direction bit (`R/W*`)
    - 1 ⇒ Master wants to Read from slave
    - 0 ⇒ Master wants to Write to slave(s)
- Data Bytes – Always Asserted by Transmitter
  - Write Direction ⇒ Transmitter = Master
  - Read Direction ⇒ Transmitter = Slave

Write to Slave

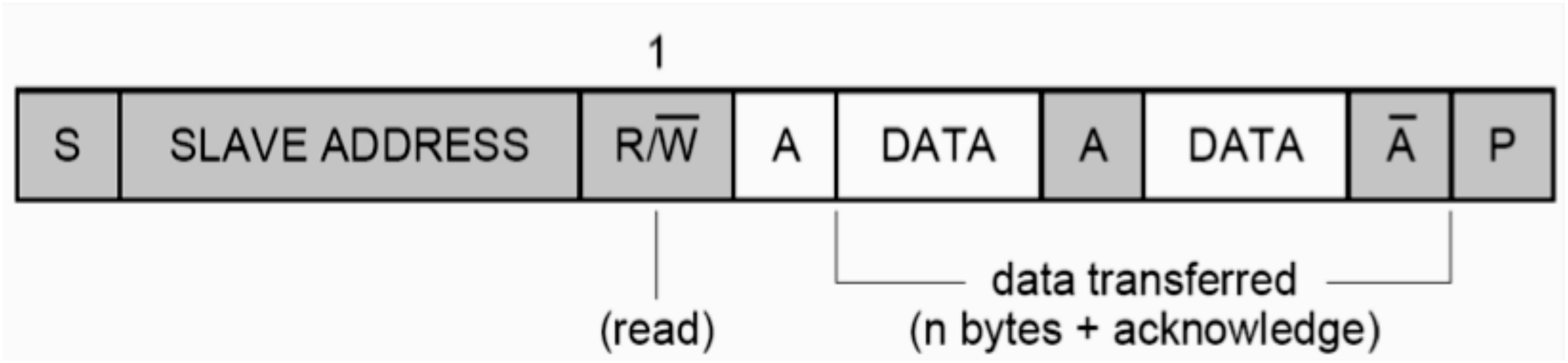


Note on the ACK bit of the final byte:

0 ⇒ ACK ⇒ Normal Receipt ⇒ Normal Termination

1 ⇒ NACK ⇒ Receive Error ⇒ Abort Transmission

Read From Slave



Final Ack bit = **NACK**

Master is telling slave that the packet is done (or aborted)

Slave then relinquishes **SDA**

So Master can send Stop bit (**P**)

Flow Control Using `ACK` Bit

`ACK` = 0 ⇒ Previous byte successfully received

`ACK` = 1 (`NACK`) ⇒ One of the following conditions:

Receiver sees incorrect data or commands in the byte

No such address – no receiver has the transmitted address so no device will `ACK` the address byte

Nobody home – receiver is too busy to receive data or transmit an `ACK`

“No Vacancy” – receiver cannot receive any more data

A master-receiver needs to signal the end of the transfer to the slave-transmitter

Flow Control Using `SCL`

Either Unit may need to temporarily pause the transmission

- Respond to an I2C interrupt
- Move data to/from its I2C shift register (avoid `ROE`/`TOE`)
- etc.

Recall:
ROE=Receive Overrun Error
TOE=Transmit Overrun Error

To pause the packet, simply hold `SCL` = 0

- Either unit can do it, because 0 is dominant
- The other unit then pauses, waiting for `SCL` rising edge
- When ready to continue, release `SCL`
- And normal operation picks up where it left off

Equivalent to pausing the clock in SPI, except in SPI only the master can do that.

A Successful Transfer

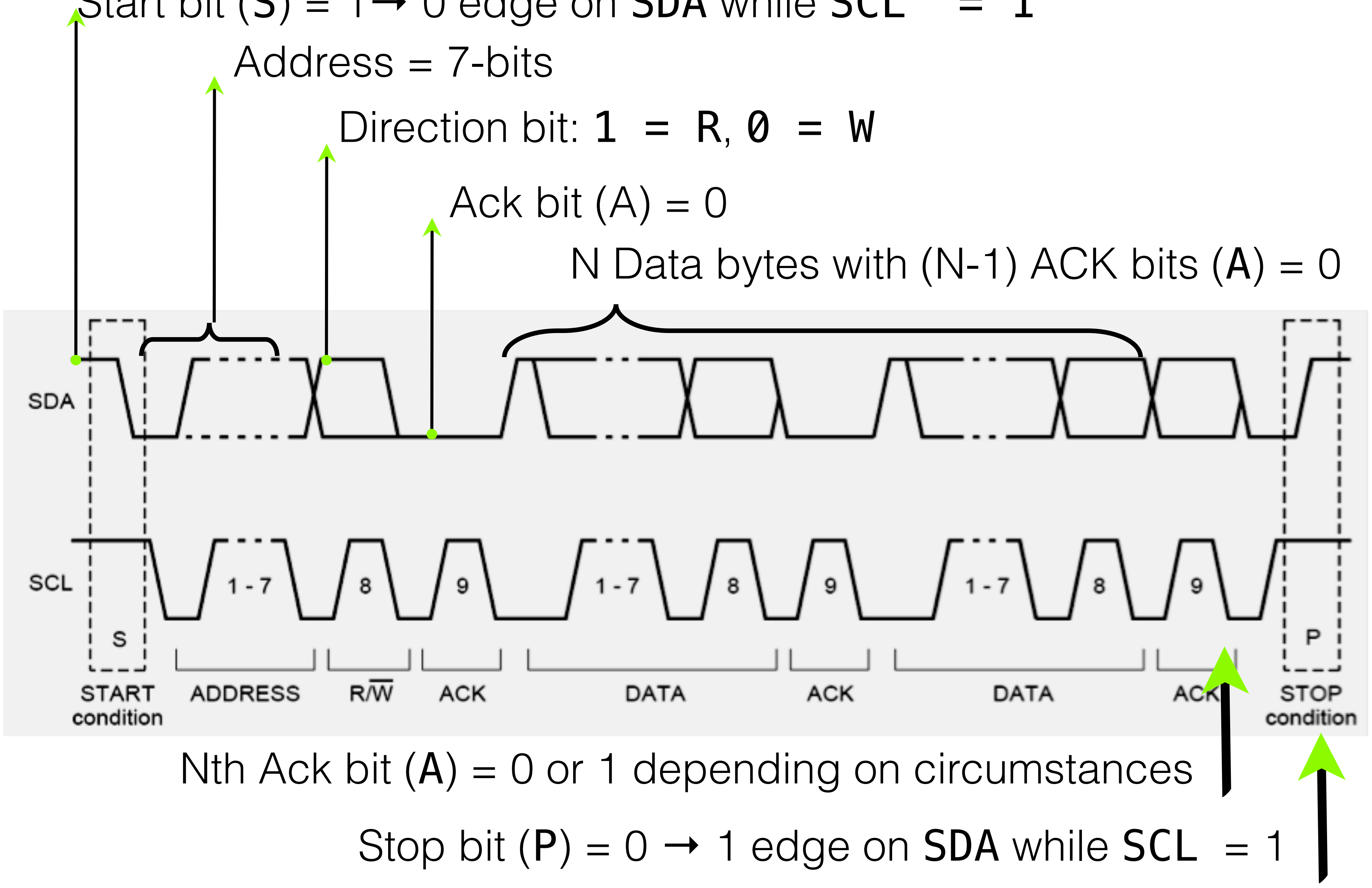
Start bit (**S**) = 1→ 0 edge on **`SDA`** while **`SCL = 1`**

Address = 7-bits

Direction bit: **`1 = R`**, **`0 = W`**

Ack bit (A) = 0

N Data bytes with (N-1) ACK bits (**A**) = 0



Nth Ack bit (**A**) = 0 or 1 depending on circumstances

Stop bit (**P**) = 0 → 1 edge on **`SDA`** while **`SCL`** = 1

Payload Efficiency

To Transfer N Bytes (= 8N bits)

Start bit	= 1 bits
Address byte	= 8 bits
Data Bytes	= 8N bits
Ack bits	= 1 + N bits
Stop bit	= 1 bit

Total Bits = (9N + 11) bits

Payload Efficiency = 8N / (9N + 11)

Example: N = 1 Byte : 8 / (9+11) =

Example: Lim N $\rightarrow \infty$: 8N / 9N = 8 / 9 =

That’s a bit low, but the I2C goal is simplicity, not speed

Bus Master Arbitration

All of the preceding is fine, IF we already have a Master

- But, I2C is a Multi-Master Bus Protocol
- Need to Arbitrate who gets to be Master

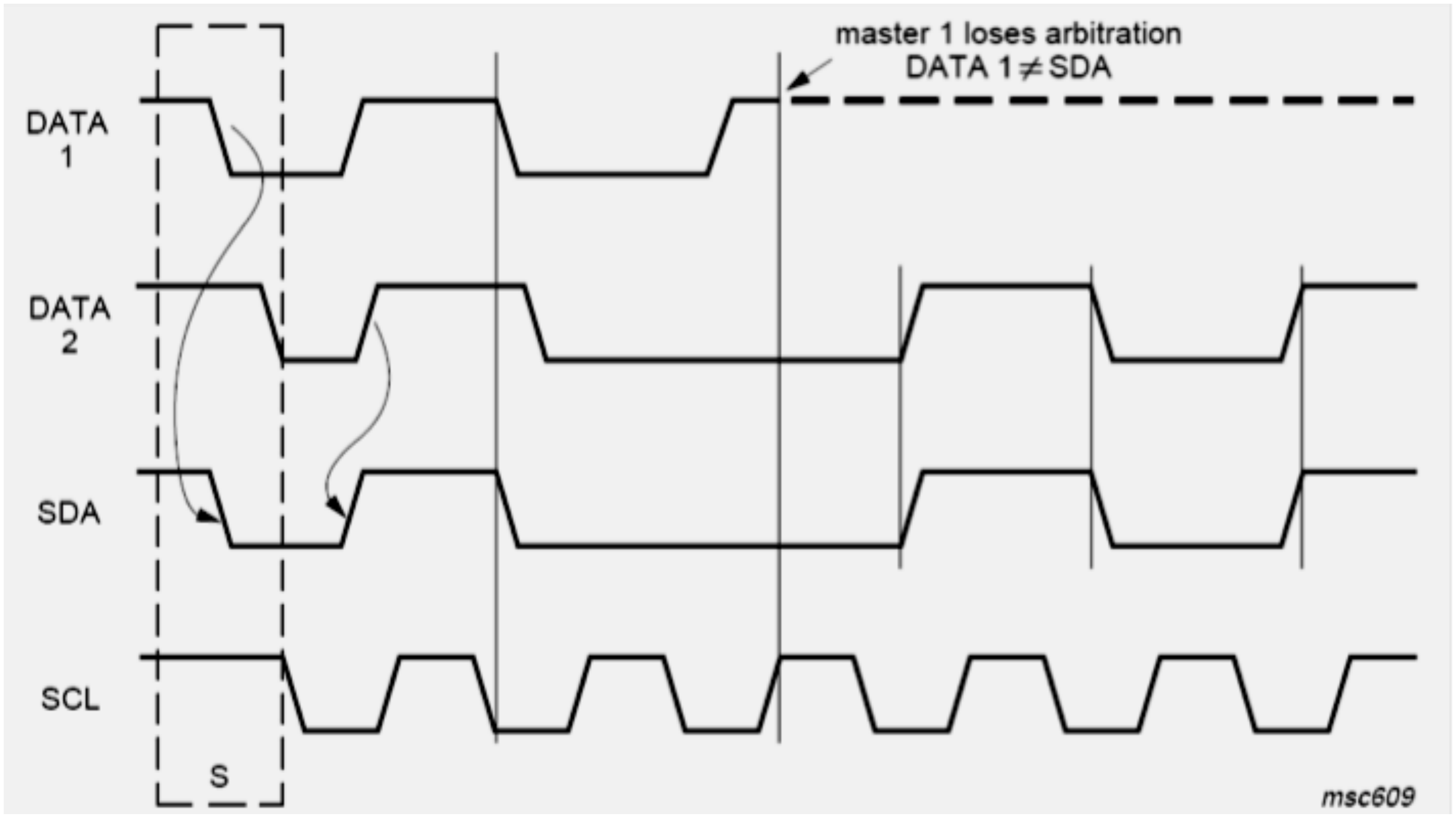
The process is comprised of two stages

- Carrier Sense – Listen to bus before transmitting
- Collision Avoidance – Listen to bus while transmitting

Bus Master Arbitration

- Carrier Sense – Listen to bus before transmitting
  - As soon as one master asserts Start bit or `SCL` all others potential masters back off and wait their turn
- Collision Avoidance – Listen to bus while transmitting
  - If two Masters start transmitting at the same instant
  - Then it slips through carrier detect & both start transmitting
  - But, eventually their bits will disagree
    - The one sending a 0 will see correct value on the bus
    - The one sending a 1 will see wrong value on the bus
    - The one sending a 1 will stop and try again later

Bus Master Arbitration



Example: suppose two masters start at the same instant

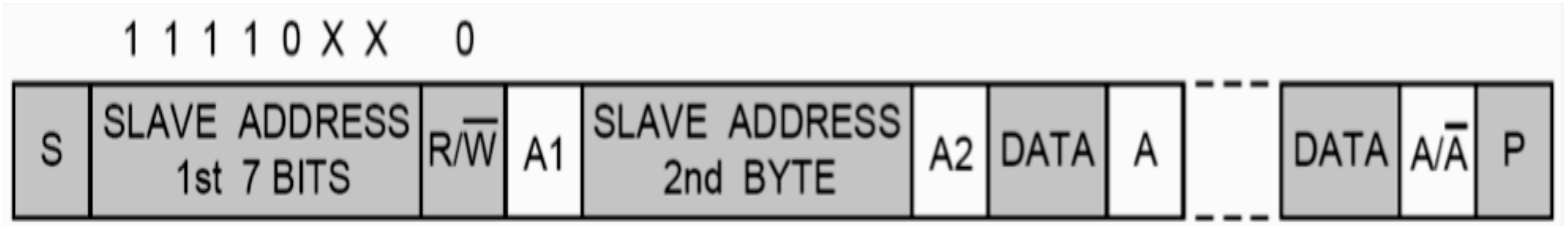
A Few Enhancements

10-bit address:

- 7-bit address only allows 128 – 8 = 120 devices
- 10-bit address allows 1024 – 8 = 1016 devices

Use one of the reserved addresses to flag 10-bit mode

- Address/Direction Byte:
  - If first 5 bits = 11110,
  - Then next 2 bits = 2 MSBs of the address
- Entire next byte = 8 LSBs of address



A Few Enhancements

Higher Speed Transmissions:

Faster modes were added to later revisions of the standard

The older modes were retained

Speeds available

100 kbps	= Standard Mode
400 kbps	= Fast Mode
1.0 Mbps	= Fast Mode Plus
3.4 Mbps	= High Speed Mode

Signaling conventions are somewhat different for the Faster Modes

I2C on the Tiva

The TM4C123GH6PM controller includes I2C modules with the following features:

Devices on the I2C bus can be designated as either a master or a slave

Four I2C modes

Master transmit/receive

Slave transmit/receive

Four transmission speeds:

Standard (100 Kbps) , Fast-mode (400 Kbps), Fast-mode plus (1 Mbps), High-speed mode (3.33 Mbps)

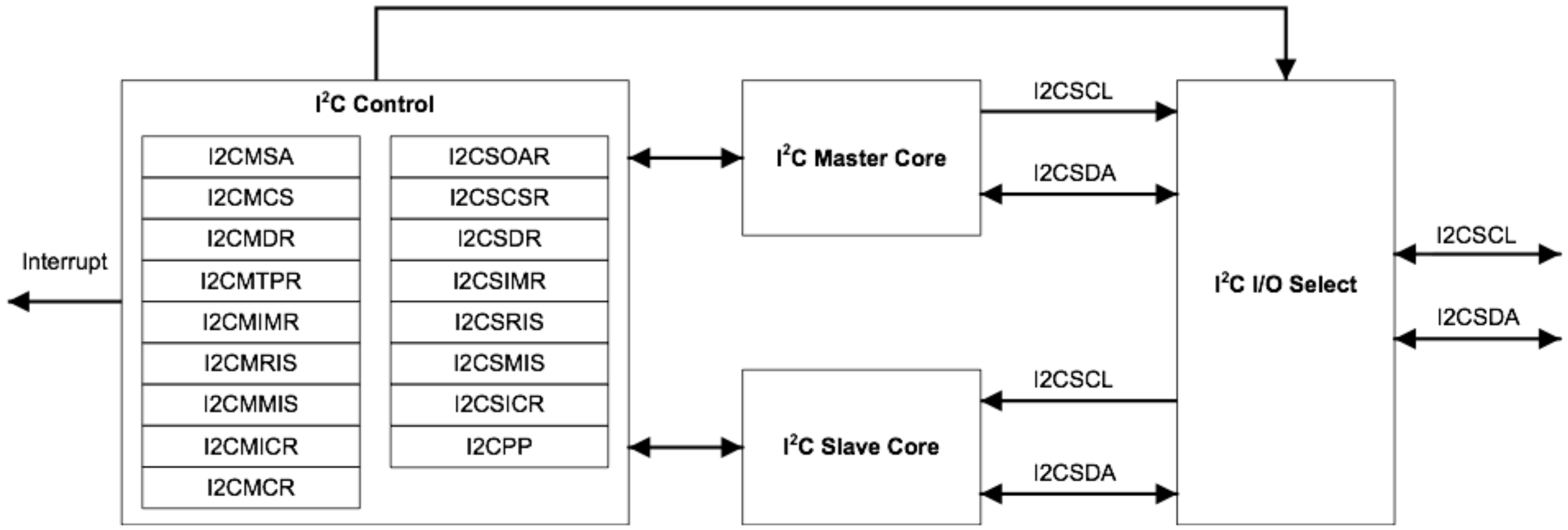
Clock low timeout interrupt

Dual slave address capability

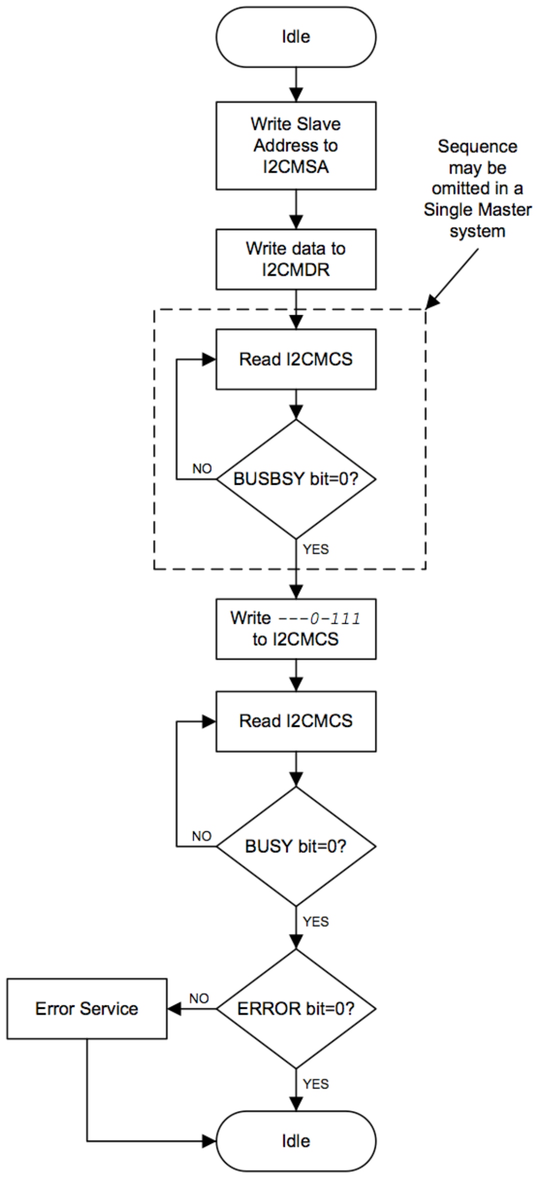
Master and slave interrupt generation

Master with arbitration and clock synchronization, multimaster support, and 7-bit addressing mode

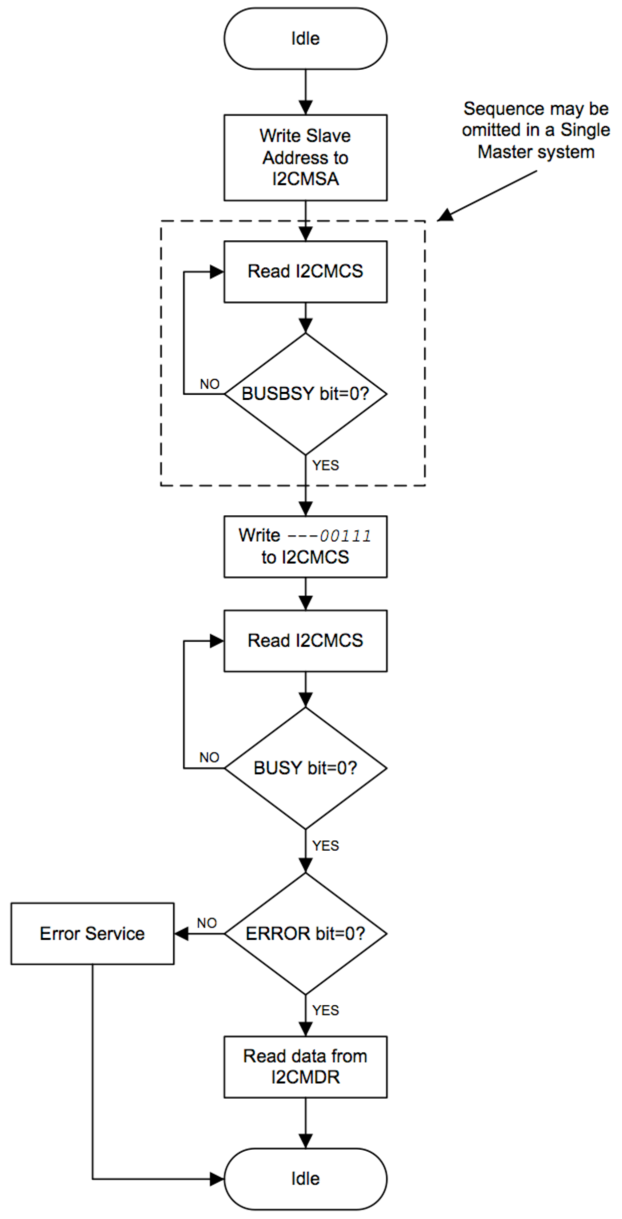
I2C Block Diagram



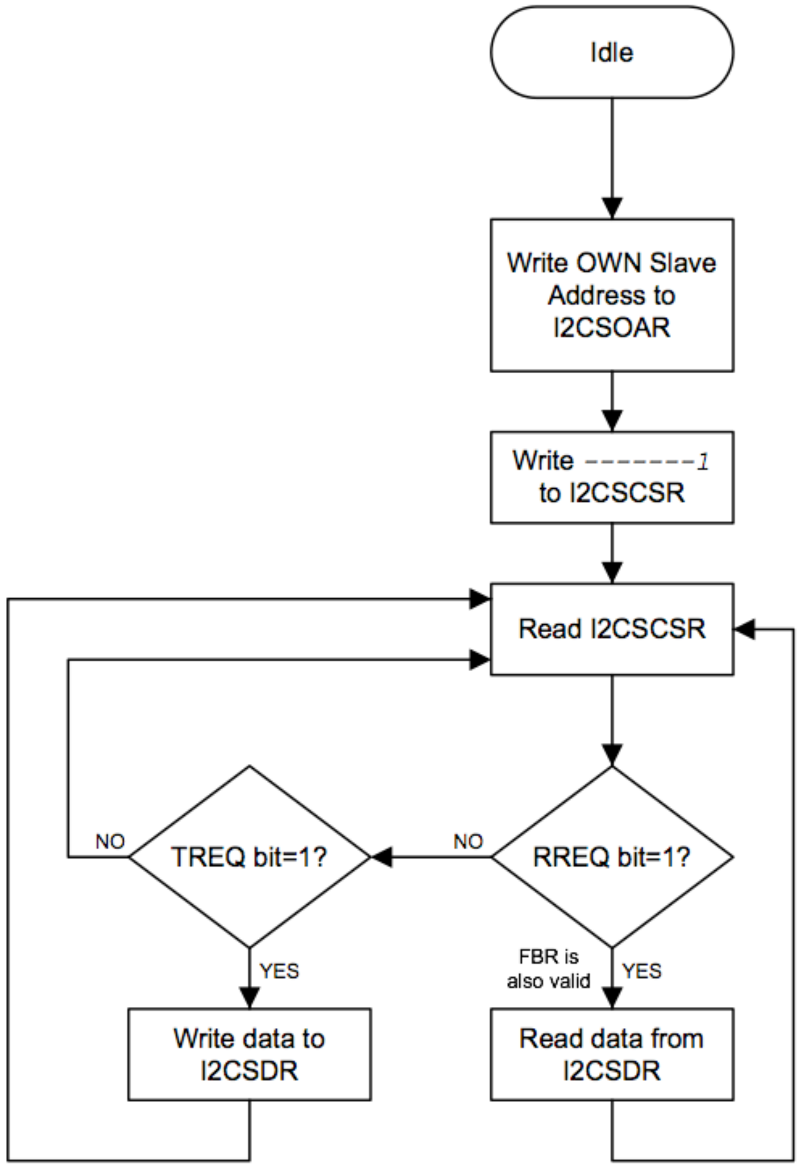
Single Master Transmit



Single Master Receive



Slave Transmit or Receive



Configuring a Master Single-Byte Transmit

The following example shows how to configure the I2C module to transmit a single byte as a master.

1. Enable the I2C clock using the `RCGCI2C` register in the System Control module.

2. Enable the clock to the appropriate GPIO module via the `RCGCGPIO` register in the System Control module.

3. In the GPIO module, enable the appropriate pins for their alternate function using the `GPIOAFSEL` register.

Configuring a Master Single-Byte Transmit

4. Enable the `I2CSDA` pin for open-drain operation.

5. Configure the `PMCn` fields in the `GPIOPCTL` register to assign the I2C signals to the appropriate pins.

6. Initialize the I2C Master by writing the `I2CMCR` register with a value of `0x0000.0010`.

7. Set the desired SCL clock speed of 100 Kbps by writing the `I2CMTPR` register with the correct value. The value written to the `I2CMTPR` register represents the number of system clock periods in one SCL clock period. The TPR value is determined by the following equation:

```
     TPR = (System Clock/(2*(SCL_LP + SCL_HP)*SCL_CLK))−1;
           TPR = (20MHz/(2*(6+4)*100000))−1;
           TPR = 9
```

8. Write the `I2CMTPR` register with the calculated value.

Configuring a Master Single-Byte Transmit

11. Specify the slave address of the master and that the next operation is a Transmit by writing the `I2CMSA` register with a value of `0x0000.0076`. This sets the slave address to `0x3B`.

12. Place data (byte) to be transmitted in the data register by writing the `I2CMDR` register with the desired data.

13. Initiate a single byte transmit of the data from Master to Slave by writing the `I2CMCS` register with a value of `0x0000.0007` (setting the `STOP`, `START`, and `RUN` bits).

14. Wait until the transmission completes by polling the `I2CMCS` register's `BUSBSY` bit until it has been cleared.

15. Check the `ERROR` bit in the `I2CMCS` register to confirm the transmit was acknowledged.

TivaWare for the I2C

The TivaWare libraries simplify all this dramatically.

Initialize the I2C master module with a call to `I2CMasterInitExpClk()`.

> That function sets the bus speed and enables the master module.

Data is transferred by first setting the slave address using `I2CMasterSlaveAddrSet()`.

> That function is also used to define whether the transfer is a send (a write to the slave from the master) or a receive (a read from the slave by the master).

If connected to an I2C bus that has multiple masters, the Tiva I2C master must first call `I2CMasterBusBusy()` before attempting to initiate the desired transaction.

After determining that the bus is not busy, if trying to send data, the user must call the `I2CMasterDataPut()` function.

The transaction can then be initiated on the bus by calling the `I2CMasterControl()` function with any of the following commands:

```
I2C_MASTER_CMD_SINGLE_SEND
I2C_MASTER_CMD_SINGLE_RECEIVE
I2C_MASTER_CMD_BURST_SEND_START
I2C_MASTER_CMD_BURST_RECEIVE_START
```

Any of those commands results in the master arbitrating for the bus, driving the start sequence onto the bus, and sending the slave address and direction bit across the bus. The remainder of the transaction can then be driven using either a polling or interrupt-driven method.

Programming Example

```
//
// Initialize Master and Slave
//
    I2CMasterInitExpClk(I2C0_BASE, SysCtlClockGet(), true);

//
// Specify slave address
//
    I2CMasterSlaveAddrSet(I2C0_BASE, 0x3B, false);

//
// Place the character to be sent in the data register
//
    I2CMasterDataPut(I2C0_BASE, ’Q’);

//
// Initiate send of character from Master to Slave
//
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_SEND);

//
// Delay until transmission completes
//
    while(I2CMasterBusBusy(I2C0_BASE)); // loop
```

Summary

- I2C is a very simple, cheap, easy to build bus
- Designed for simple chip-to-chip communication
    - Only 2 lines
    - Synchronous
- Uses a Wired AND bus technology
    - 0 is the dominant bit
    - Allows multiple transmitters without damage
    - Needed for
        - New Master Arbitration Phase
        - Flow Control using `ACK` bits
        - Flow Control using `SCL` Line

Summary

Packet is Byte Oriented

- Address Byte
- Data Bytes
- `ACK` bits spread between bytes
- Start and Stop bits

New Master Arbitration is based on:

- Carrier Sense: Listen before transmitting
- Collision Avoidance: Listen while transmitting
  - If `SDA` = 0 while sending a 1, then stop & try again later

A Few Enhancements:

- 10-bit addresses – up to 1016 units on the bus
- Faster speeds – up to 3.4 Mbps at this time

Summary

I2C-Bus Specification and User Manual (UM10204), Rev 03, NXP Semiconductors, June 2007, `I2C–spec–uman–2007.pdf` .