Inter-Integrated Circuit Bus

IIC   I2C   TWI

I2C Bus

Synchronous, multi-master, multi-slave, packet switched, single ended serial bus

Developed by Philips in the early 1980's (prior to SPI)

Intended for on-board communications between devices

First(?) commercially viable bus for connecting low-speed peripherals found in control systems (factory floor, monitoring systems, simple sensors, etc.)

Objectives included multiple masters and slaves with single two-wire interface (SCL and SDA)

Lots of users: TI, ADI, NXP, On semi, ST, Intersil, etc; mostly for low-speed sensors and actuators, and sending parameters for audio and display/video devices

References: i2c.info/i2c-bus-specification; www.i2c-bus.org; wiki page

I2C

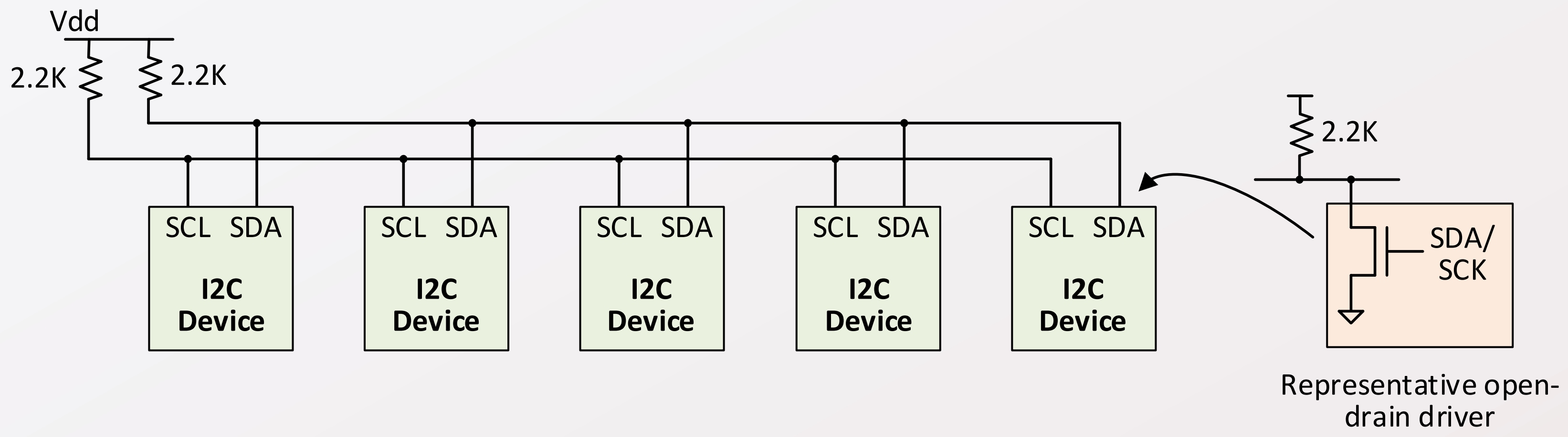
I2C uses two bidirectional, synchronous signals: SCK and SDA

Any network node (device) can be master or slave at any given time

Both signals are open-drain (they can drive low, but must float high). Why?

Pull-up is typically 2K – 10K, but can be anything.

If Vdd is 3.3V, Rp is 10K, and total signal Cs is 100pF, what is rise time? Max Freq?



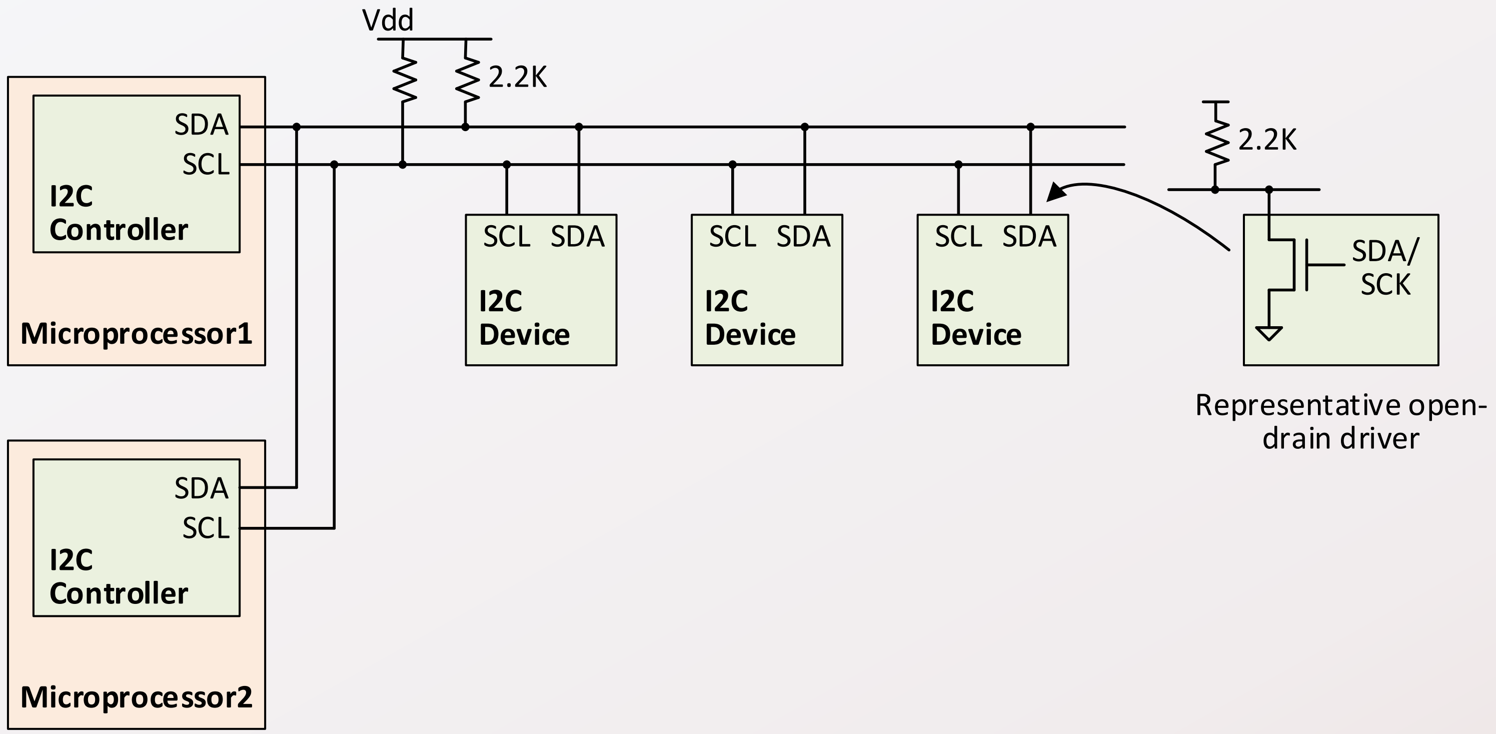
What limits Max Freq, and what drives the limiting parameters?

I2C

In a typical “real world” situation, 1 or maybe 2 masters and N slaves

Each master must check SDA and SCL lines before driving

Both lines ____, OK to drive. Both lines ____, bus already in use

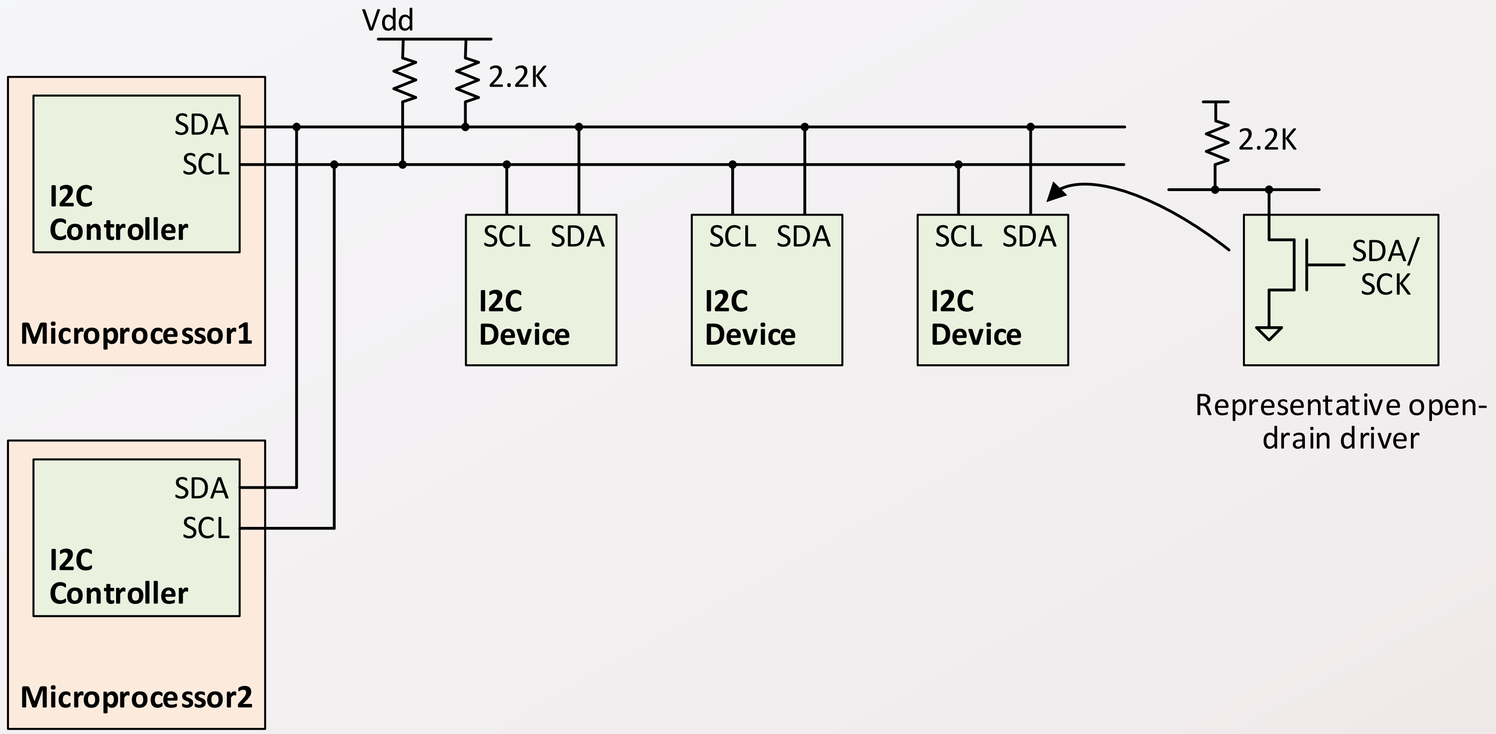


Arbitration

If bus is busy, next master can poll, or simply wait and try again “next time”

What if two masters start simultaneously?

Both must constantly monitor SCL and SDA. If either detects lines are low when should be pulled high, transaction aborted



Signals

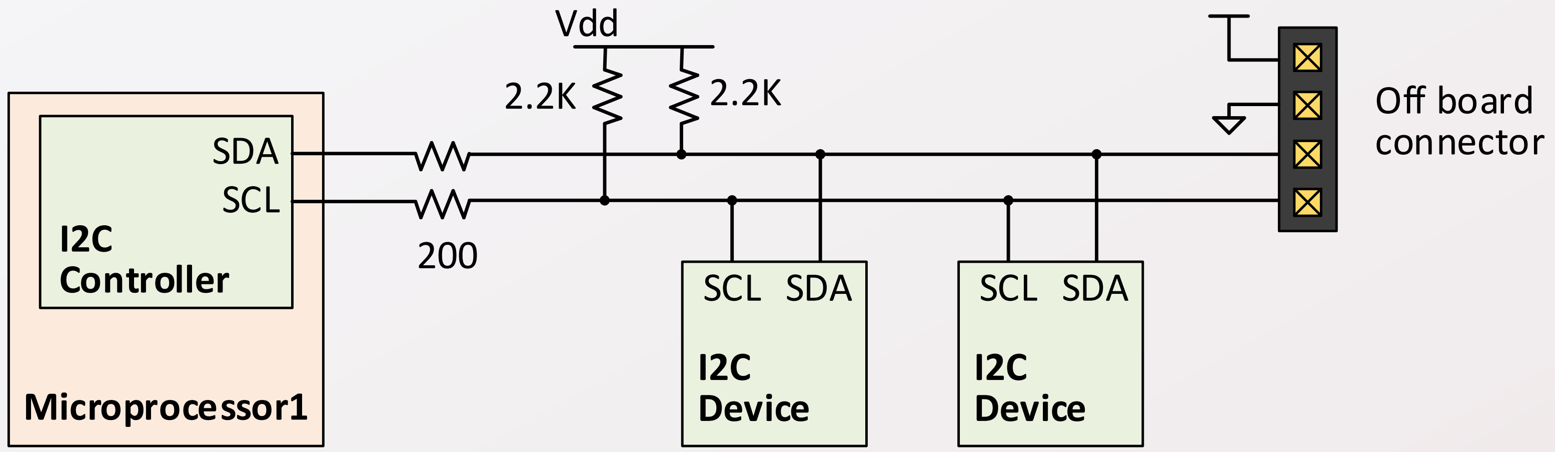
Most I2C busses use 3.3V or 5V Vdd, but there is no requirement

Most busses use around 2-2K for pull-up, but there is no requirement

If signals go off board, series R is common (why?) Any problems?

Note slave can drive to ground, but Master cannot (why?) Problem? Benefit?

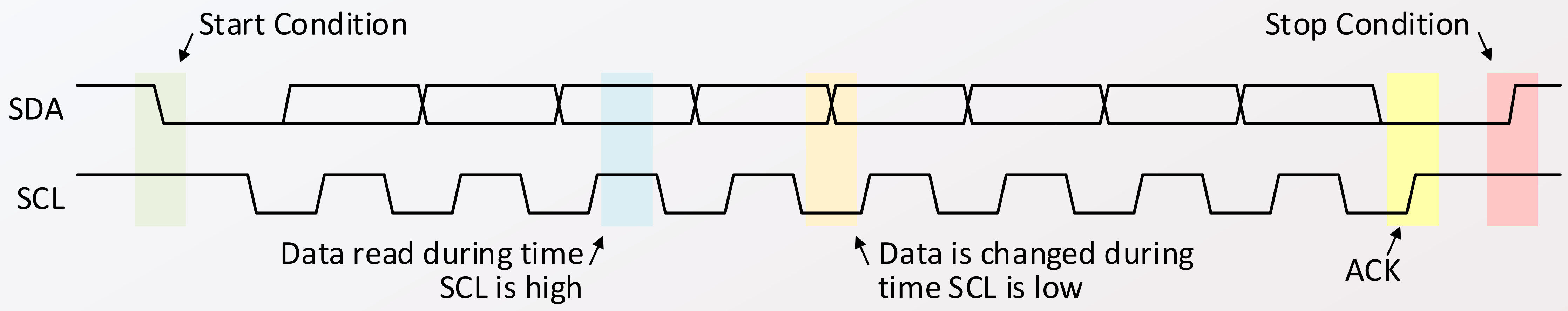
What if off-board port uses different Vdd?



I2C Packets

I2C packets are composed of one or more 8-bit bytes. Each packet begins with a start condition and ends with a stop condition. Every byte must have an ACK bit.

SDA can only change when SCL is low. SDA read when SCL is high.


Start Condition
Stop Condition
SDA
SCL
Data read during time SCL is high
Data is changed during time SCL is low
ACK

After a start condition, the bus is busy and cannot be used by another master until a stop condition is detected.

Clock is not tightly specified; slave must use clock to send and receive data

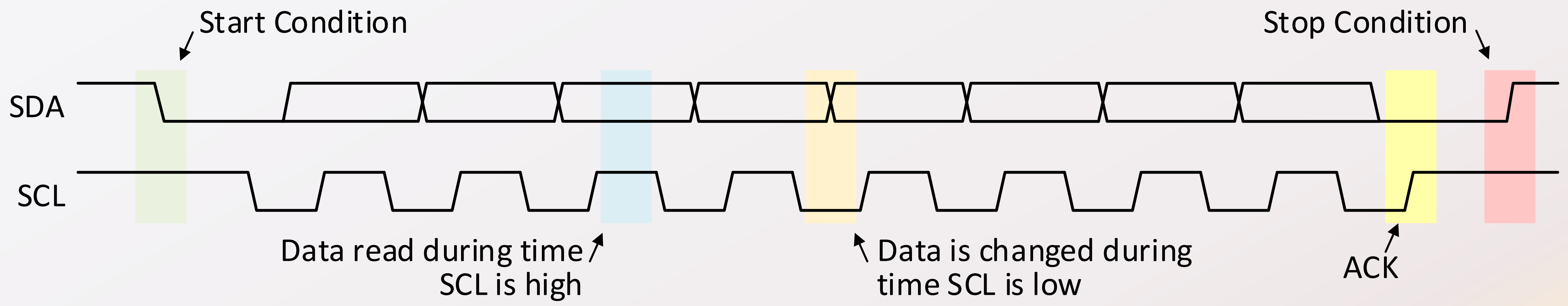
I2C Clock

Standard I2C clock mode is 100KHz; fast mode is 400KHz; high speed is 3.4MHz

Clock is not tightly specified; slave must use clock to send and receive data

The master always generates SCL

Most I2C controllers sample SDA and SCK with a higher frequency clock and use a state machine to track changes

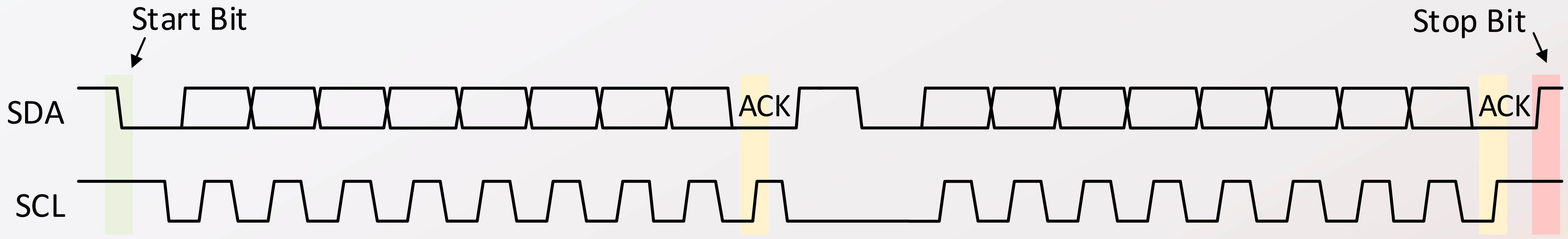

Start Condition
Stop Condition
SDA
SCL
Data read during time SCL is high
Data is changed during time SCL is low
ACK

I2C Packets

There is no limit on how many bytes can be transferred per packet.

Each byte must be followed by an ACK. If No ACK (NACK) after write, the slave cannot accept more data and master must wait for ACK or timeout. If NACK after read, there is no more data to read.

For multi-byte packets, no stop bit is issued between bytes, but ACKs are still required.



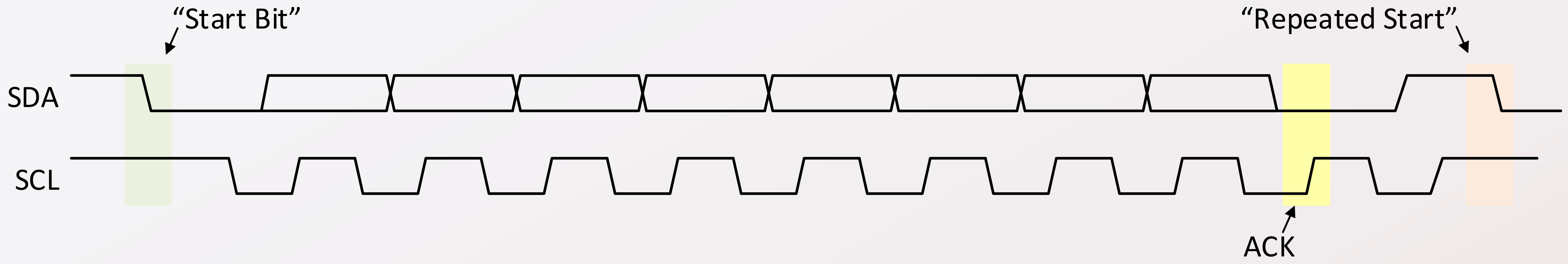
I2C Packets

After the ACK bit…

- Another byte can be transmitted

- A stop bit can be sent to release the I2C bus

- A repeated start can be sent to start a new packet without releasing the bus



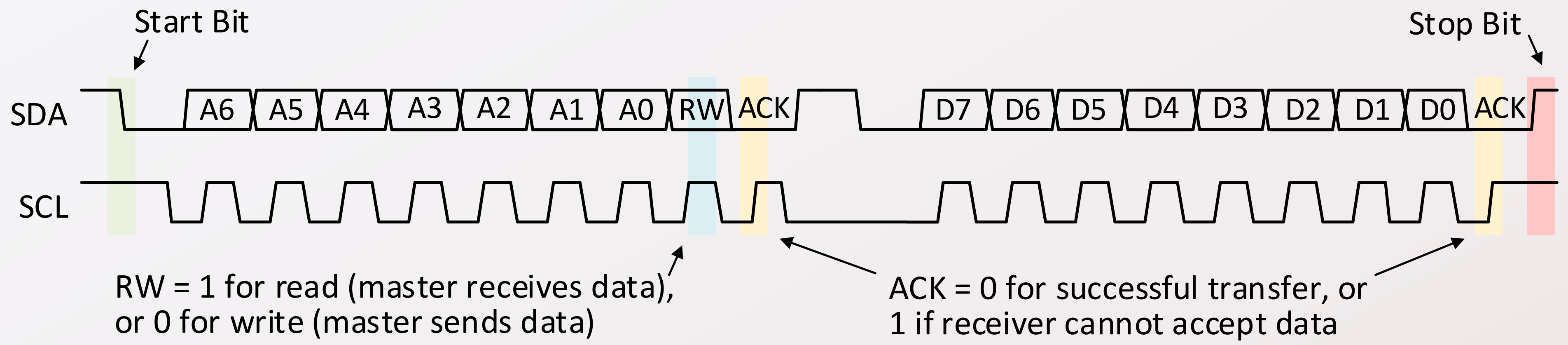
Packet Contents

Every I2C packet starts with a 7-bit address and a read/write bit (1 for read).

17 out of the possible 128 seven-bit addresses are reserved, leaving 112 valid addresses (see following table)

Addresses are sent out MSB first. After the address, the R/~W bit is sent.

Then, one or more data bytes are sent. Every byte must be ACK'ed, stop bit must be used to terminate packet and release bus (or a repeated start can be issued)


Start Bit
Stop Bit
SDA
SCL
A6
A5
A4
A3
A2
A1
A0
RW
ACK
D7
D6
D5
D4
D3
D2
D1
D0
ACK
RW = 1 for read (master receives data), or 0 for write (master sends data)
ACK = 0 for successful transfer, or 1 if receiver cannot accept data

I2C Reserved Addresses

Address	Function	Description
0000 000 0	General Call	Broadcast to all slaves; slaves can ACK if relevant
0000 000 1	Start Byte	Used by processors “bit banging” I2C port to identify start packet
0000 001 X	CBUS address	Older deprecated bus that reused I2C pins. No longer used.
0000 010 X	Reserved	
0000 011 X	Reserved	
0000 1XX X	High Speed Master Code	Signifies a masters intention to send high-speed packets
1111 0XX X	10-bit Slave Addressing	Signifies a 10-bit address (instead of standard 7) follows (uses XX bits + next byte)
1111 1XX X	Reserved	

Note for 10-bit addressing, devices with 7-bit addresses simply ignore packets starting with 11110

I2C Addressing

Slave devices must constantly monitor bus, watching for their address (or 0000 000 if applicable)

Slave devices must use SCL to receive data; SCL timing is not strictly enforced

Slaves may have several readable and/or writable registers; it is typical that the initial payload byte(s) by master identifies which register to read or write.

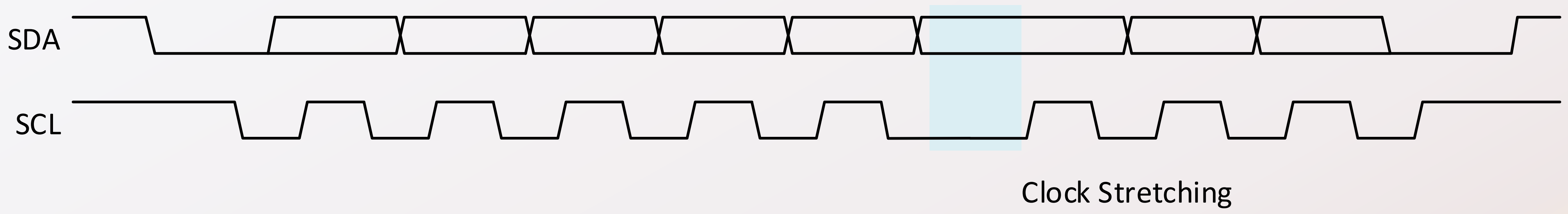
Many I2C slaves use an auto-increment feature so consecutive data registers can be read or written without sending a new address

Clock Stretching

SDA changes when SCL is low. The Master is required to check SCL at the end of each bit-change time to make sure SCL has transitioned to a ‘1’ before proceeding.

A slave can hold SCL low if it is unable to read and deal with the bit in the allotted time. There is no limit on how long SCL can be held low.

This is known as “clock stretching”, and it gives slaves the ability to slow down the master. It is rarely used, and it can significantly decrease overall bus bandwidth.



I2C relative advantages

- Master generates clock used by slaves, so a precision clock is not needed. Period, phase and duty cycle can vary widely, provided the minimum clock period requirement is met
- Multi-slaves on same bus – only two pins needed (good for 8-pin MCUs)
- Open-drain for simple arbitration logic
- No restrictions on packet sizes
- No special transceivers are needed

I2C relative disadvantages

- Shared bus – any node can hang the bus by holding SDA or SCK low
- Limited speed
- Unique addresses required for all slaves. With only 7 bits of address (or 10 for newer systems), and 1000’s of devices, how is this implemented?
- Addresses and ACKs cut into available bandwidth

So… is I2C a good choice?

SMBus and PMBus

SMBus (System Management Bus) is a stricter version of I2C developed by Intel in the mid 1990s. It is used for managing intra-PC resources, like turning on and off power, battery management, sensors (like temperature and voltage), bus configuration, etc.

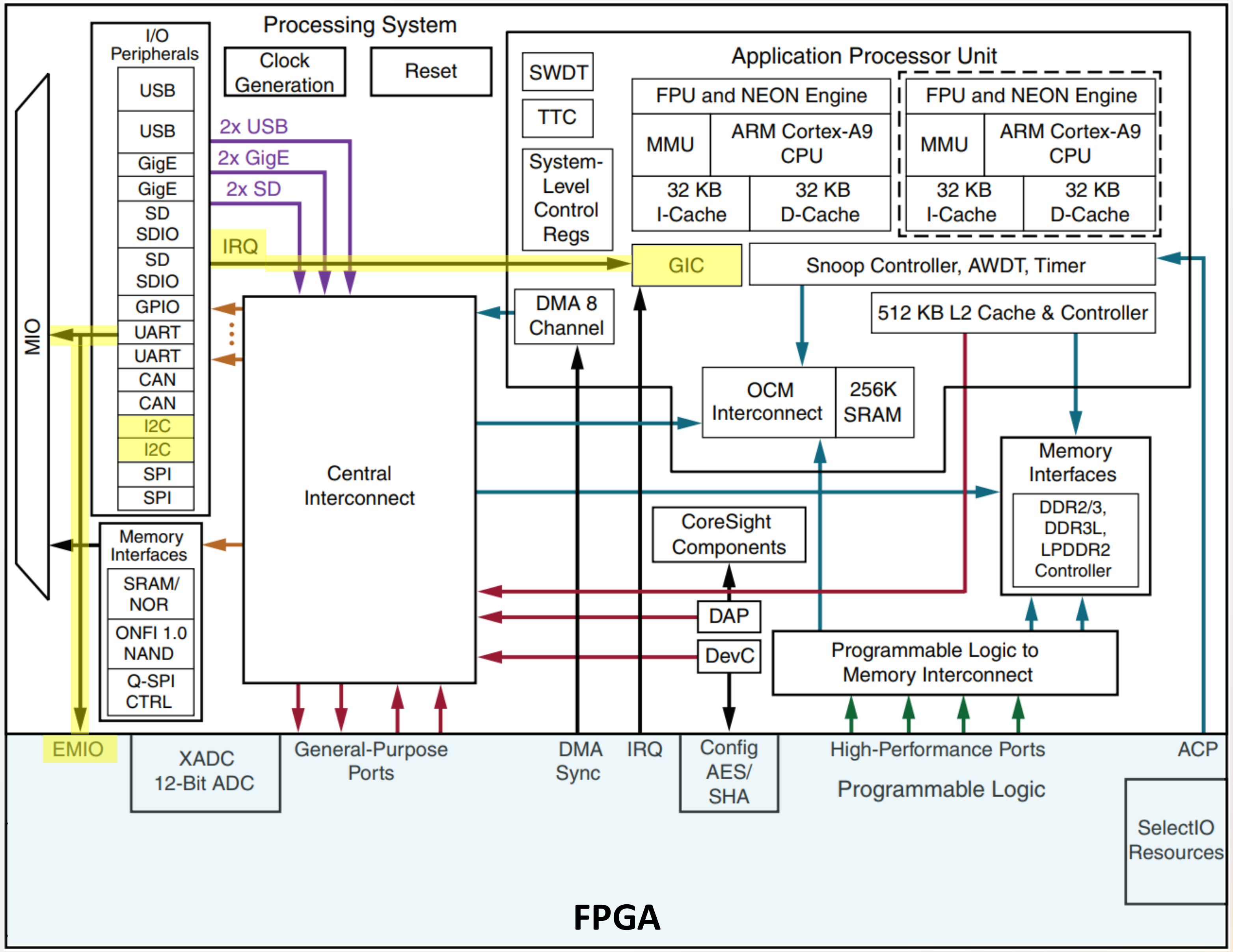
PMBus (Power Management Bus) is a variant of SMBus that specifically targets power supplies.

Both busses are similar, and differ from I2C only slightly (ACKs are required for all writes, time-outs are strictly controlled, error checking is built in)

I2C on ZYNQ

Two independent I2C controllers

Can use MIO pins or EMIO (via FPGA) pins

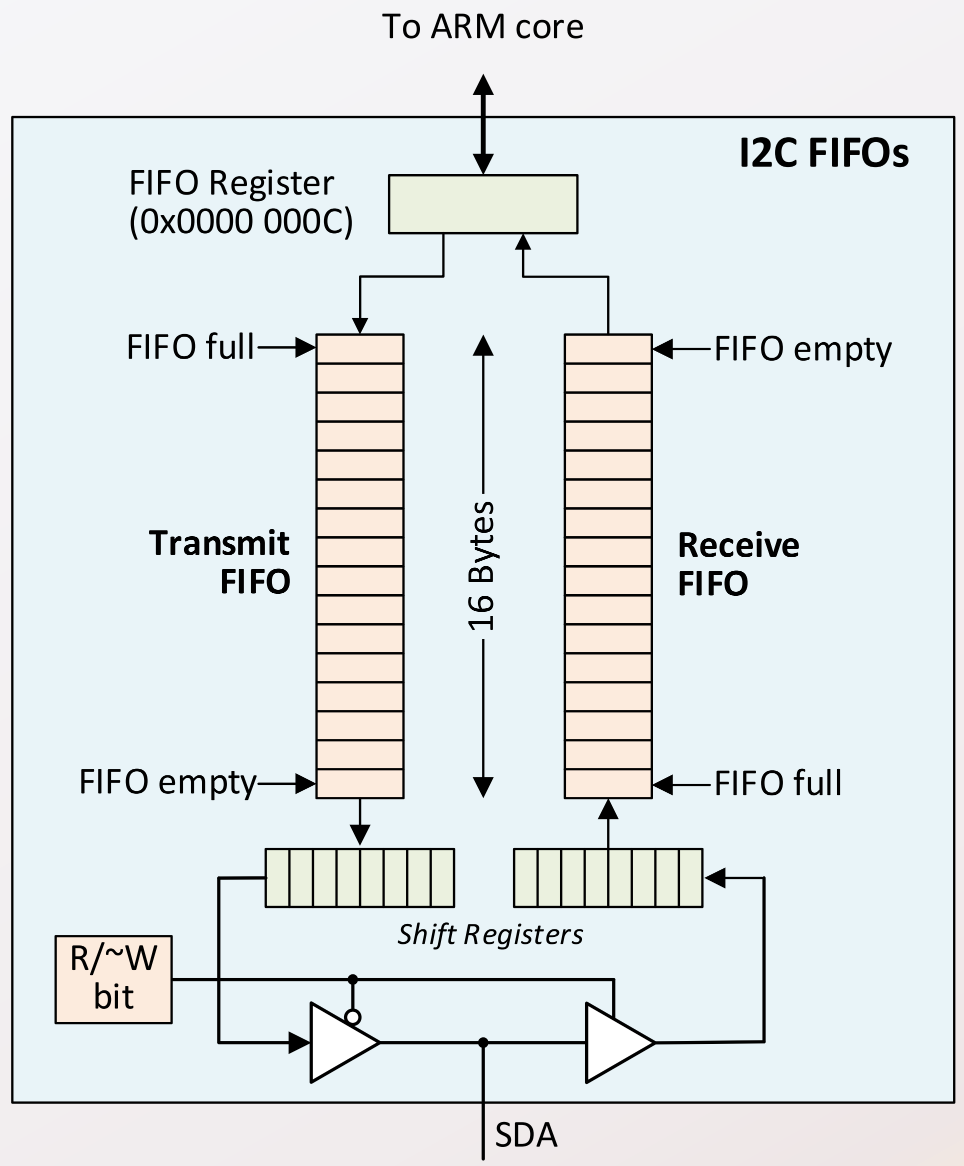


Memory-mapped peripheral using AXI bus

I2C Configuration and Status Registers			
Name	**Function**	**Address**	**Bits**
CR (Configuration Register)	I2C Configuration (clk setup, enables, overall setups)	0x 0000 0000	16
SR (Status Register)	Status (Read and write status)	0x 0000 0004	9
ADDR (I2C Address Register)	Address	0x 0000 0008	10
DATA (I2C Data Register)	Read and write data values	0x 0000 000C	8
ISR (Interrupt Status Reg.)	FIFO, read and write, and timeout status	0x 0000 0010	10
TRANS_SIZE (Xfer Size Reg.)	Number of bytes to send, received, or expected	0x 0000 0014	8
SLV_PAUSE (Pause Reg.)	Length of optional pause interval	0x 0000 0018	4
TIME_OUT (Time out Register)	Set timeout period before interrupt generated	0x 0000 001C	8
IMR (Interrupt Mask Register)	Interrupt Mask bits	0x 0000 0020	10
IER (Interrupt Enable Reg.)	Interrupt Enable bits	0x 0000 0024	10
IDR (Interrupt Disable Reg.)	Interrupt Disable bits	0x 0000 0028	10

ZYNQ's I2C FIFOs

TXD and RXD path have 16-byte FIFOs



ZYNQ’s I2C Status Register

I2C Status Register Bits		
Name	**Function (definitions for ‘1’ bit)**	**Bit#**
BA	Bus Active	8
RXOVF	Rx FIFO Full and new byte recieved	7
TXDV	Tx FIFO has more entries	6
RXDV	Rx FIFO has valid data	3
RXRW	Read/Write mode received	2

ZYNQ I2C Configuration

Bits in I2C Configuration Register		
Name	**Function**	**Bit#**
CR_DIV_A	2-bit divisor for stage A clock divider	15:14
CR_DIV_B	6-bit divisor for stage B clock divider	13:8
CLR_FIFO	Clears FIFO and transfer size register (auto cleared if set)	6
SLVMON	Enable slave monitor mode	5
HOLD	Hold SCL low	4
ACKEN	Enable acknowledge	3
NEA	Address mode (1 for 7 bit, 0 for 10 bit)	2
MS	Overall mode (1 for master, 0 for slave)	1
RD_WR	Transfer direction (master mode only. 1: Rx, 0: Tx)	0

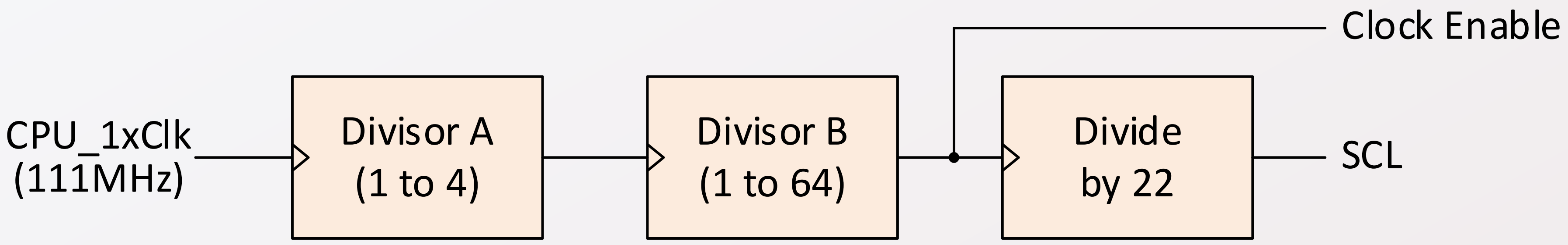
ZYNQ I2C Configuration

I2C Interrupt Functions		
Name	**Function**	**Bit#**
ARB_LOST	Arbitration lost; cycle must delay	9
RX_UNF	Rx FIFO underflow	7
TX_OVR	Tx FIFO overflow	6
RX_OVR	Rx FIFO overflow	5
SLV_RDY	Monitored slave ready (ACK received)	4
IXR_TO	Transfer timeout (SCLK low too long)	3
IXR_NACK	Transfer timeout (NACK received)	2
DATA_MAS	More data for Master to write	1
COMP	Transfer Complete	0

Baud Generator

SCL = 111MHz / (Divisor A * Divisor B * 22)

where Divisor A and B are defined by bits in configuration register



Configuring the I2C controller

The I2C controller is a “protected resource”, so access must be unlocked.

“System Level Control Registers” control access to most on-board peripherals.

Access is unlocked by writing “DF0D” to address 0xF800 0002
(this is fixed by chip designers)

After unlocking, I2C system can be reset and then configured

Configuring the SPI controller

```
#define MOD_RESET_BASEADDR 0xF8000000

void SPI_reset() {
    uint32_t register_value;
    *((uint32_t *) MOD_RESET_BASEADDR+0x8/4) = 0x0000DF0D;          // unlock the SLCRs
    *((uint32_t *) MOD_RESET_BASEADDR + 0x00000224/4) = 0x3;        // Reset I2C
    *((uint32_t *) MOD_RESET_BASEADDR + 0x00000224/4) = 0;          // Release the reset
return;
}
```

Configuring ZYNQ’s I2C Controller (setup Configuration Register)

1. Set Clock Divisor A to 0 and Divisor B to 50 to generate a 100KHz SCL;
2. Initialize the FIFO to all zeros by setting the Clear FIFO bit (bit6 = 1);
3. Set slave monitor mode to normal operation (bit5 = 0);
4. Do not hold the bus when transaction completed (bit4 = 0);
5. Enable ACK (bit3 = 1);
6. Set addressing mode normal 7-bit (bit2 = 1);
7. Set overall mode to Master (bit1 = 0);
8. Set direction as master receiver (bit0 = 0). After the I2C controller has been configured, you can create a C function to read the bus. The function should perform the following steps:
9. Write the desired number of bytes (2) to be read to the TRANS_SIZE register;
10. Write the address of the temperature sensor (1001000) to the ADDR register;
11. Poll on the RXDV bit from the I2C_Status Register (bit3);
12. Read data from the FIFO (DATA register) and decrement the read data count (TRANS_SIZE) until data count = 0

Blackboard’s Temperature Module

The Blackboard includes an LM75BDP temperature module from NXP

The LM75BDP is an I2C temperature module that includes:

- a band-gap temperature sensor
- a 12-bit analog-to-digital, sigma-delta converter
- an I2C controller

The LSM9DS1 is connected to I2C0 through the MIO interface

The I2C address is 1001000

Temperature Module

The temperature module returns two bytes, with the upper 11 bits containing the 2’s complement temperature value.

The 10 magnitude bits cover a range of -55C to +125C. The 10-bit value must be scaled by multiplying it by .125 to convert to a temperature value. (Shift!)