EEL 4742 – Embedded Systems
Module 7 – I2C

Hadi M Kamali

Department of Electrical and Computer Engineering (**ECE**)
University of Central Florida

Office Location/phone: HEC435 – (407) 823-0764
webpage: https://www.ece.ucf.edu/~kamali/
e-mail: kamali@ucf.edu

HAVEN Research Group
https://haven.ece.ucf.edu/

HARDWARE VERIFICATION ENGINEERING ASSURANCE

UNIVERSITY OF CENTRAL FLORIDA

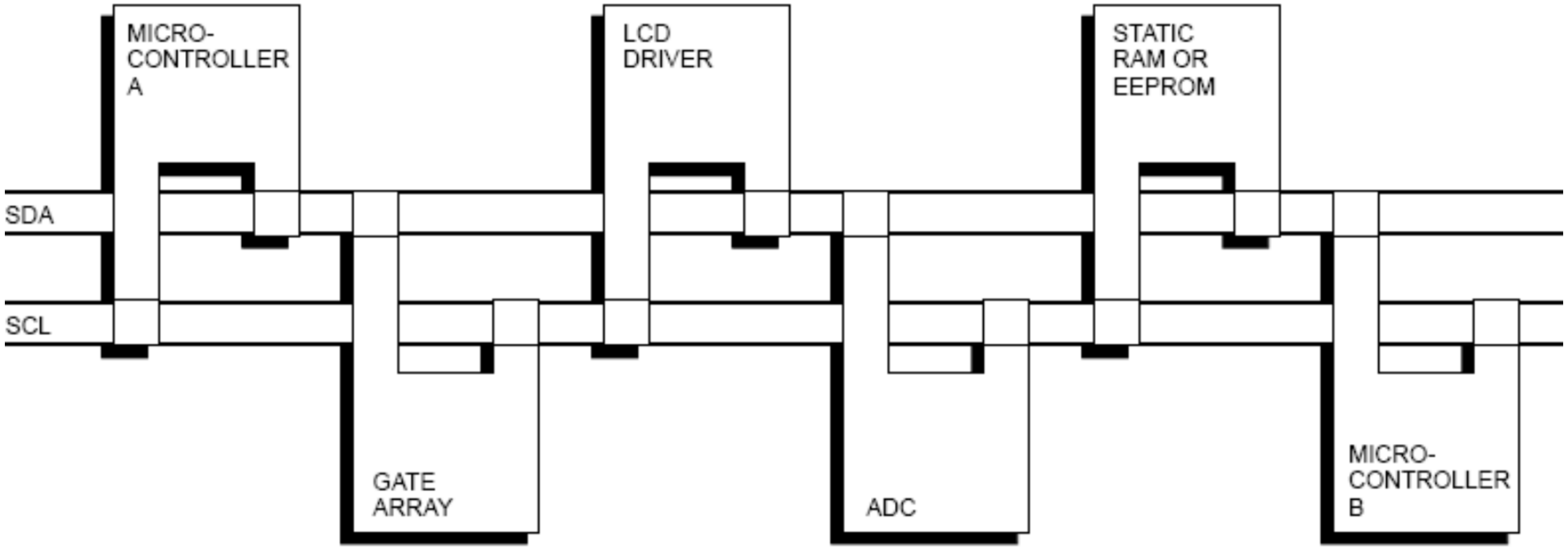
What is I2C

- Inter-Integrated Circuit (I^2C) *Also called* ***Two Wire Interface (TWI)***
  - Synchronous serial communication protocol developed by Philips (now NXP)
  - Communication between multiple devices (such as microcontrollers, sensors, memory chips, etc.)

BUS *topology and has two wires*

Serial data

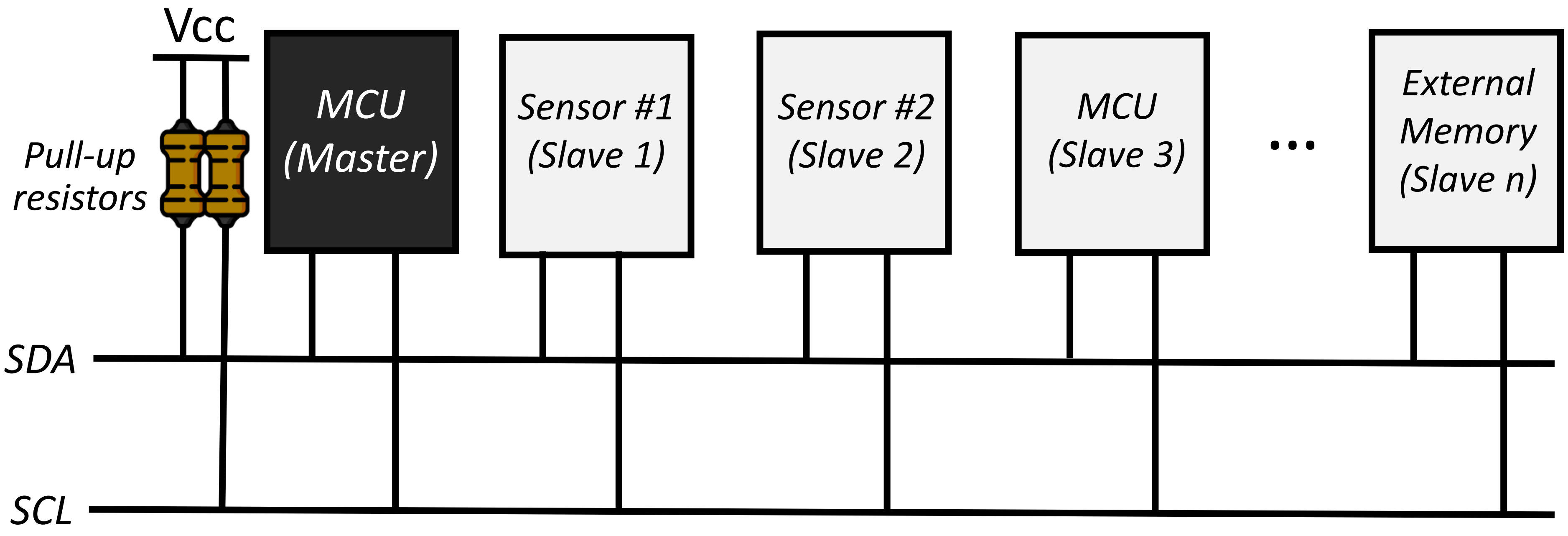
Serial clock



- I2C is SYNCHRONOUS
  - There is a shared clock reference between devices
  - Each side of communication use this clock source for transceiving data

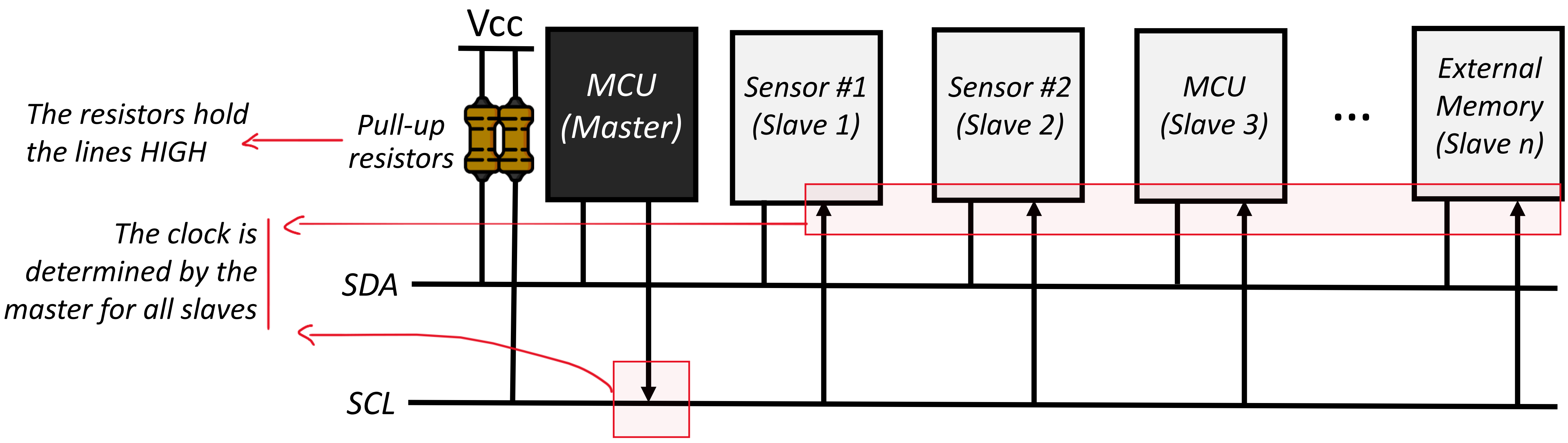
I2C Communication

- Inter-Integrated Circuit (I^2C) *Also called **Two Wire Interface (TWI)***
  - It is master-slave communication protocol.
    - The master initiates communication and controls the clock (SCL line).
    - The slave is the device being communicated with by the master.
      - (Because it is bus-based) Multiple slave devices can be connected to the same bus.



I2C Communication

- Inter-Integrated Circuit (I^2C) *Also called **Two Wire Interface (TWI)***
  - It is master-slave communication protocol.
    - The master initiates communication and controls the clock (SCL line). → *Could be for write or read operation!*
    - The slave is the device being communicated with by the master.
      - (Because it is bus-based) Multiple slave devices can be connected to the same bus.

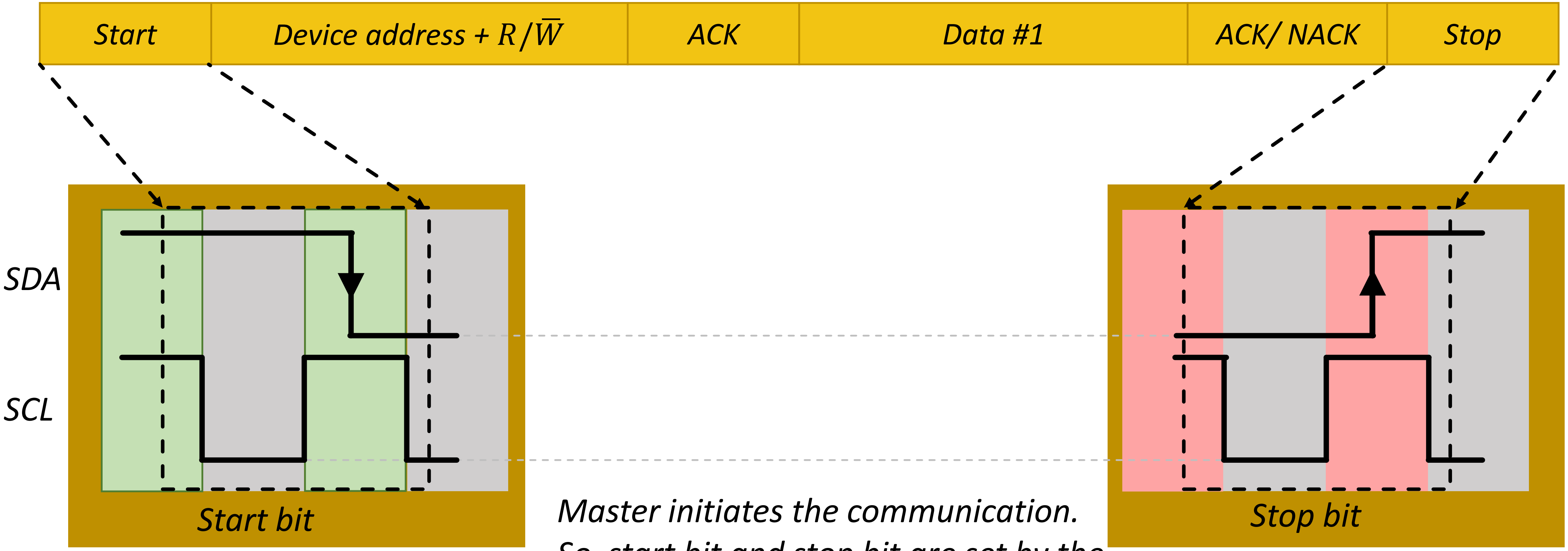


I2C Master-Slave Relationships

- We have primary devices vs. secondary devices
  - Primary transmitters/receivers
  - Secondary transmitters/receivers
  - Suppose microcontroller A wants to send information to microcontroller B
    - A (primary) addresses B (secondary)
    - A (primary transmitter), sends data to B (secondary receiver)
    - A terminates the transfer
  - Suppose microcontroller A wants to receive information from microcontroller B
    - A (primary) addresses B (secondary)
    - A (primary transmitter), receives data from B (secondary receiver)
    - A terminates the transfer
- The primary (microcontroller A) generates the timing and terminates the transfer

Data Packet in I2C

- **Start and Stop Bit**

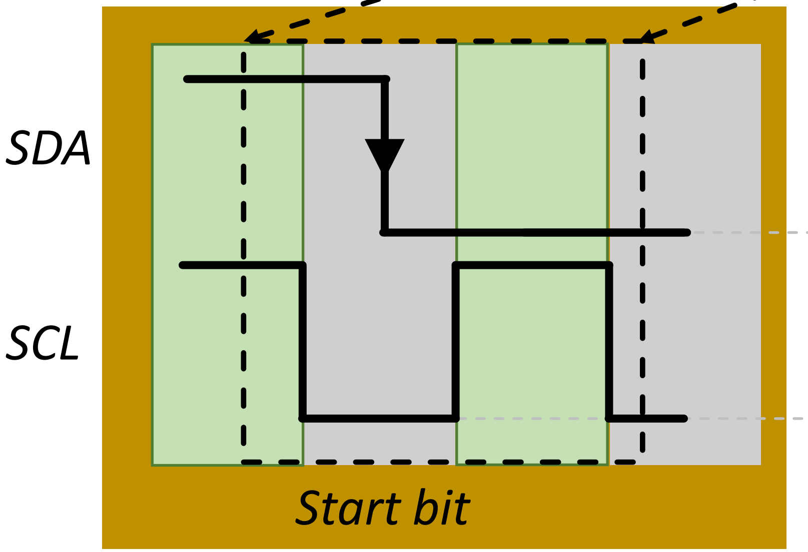


Master initiates the communication. So, start bit and stop bit are set by the master to signal the beginning and ending of the data transmission.

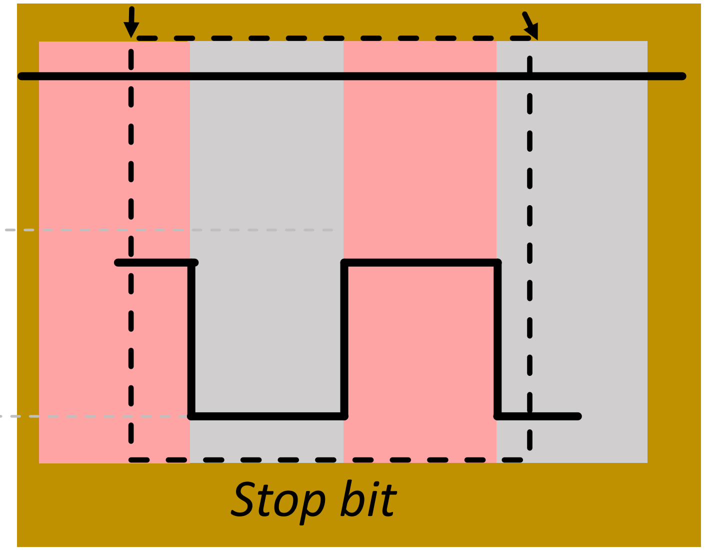
Data Packet in I2C

- Ack and Nack signaling

Start	Device address + $R/\overline{W}$	ACK	Data #1	ACK/ NACK	Stop



Either master/slave can send ACK/NACK bit. Normally, the SDA wire is pulled-high due to the pull-up resistors. During ACK, the line is pulled low. And during NACK, the line is not pulled low.



Data Packet in I2C

- Addressing

Start	Device address + $R/\overline{W}$	ACK	Data #1	ACK/ NACK	Stop

7-bit device address							$R/\overline{W}$
7	6	5	4	3	2	1	0

8-bit device data							
7	6	5	4	3	2	1	0

- 7- or 10-bit address

  hard coded into the chip

- Peripherals often have fixed and programmable address portions
- Addresses starting with 0000 or 1111 have special functions
  - 0000000 Is a General Call Address – E.g. system reset
  - 1111XXX Address Extension (for 10-bit address)
    - 1111111 Address Extension – Next Bytes are the Actual Address

Data Packet in I2C

- Addressing

Start	Device address + $R/\overline{W}$	ACK	Data #1	ACK/ NACK	Stop

7-bit device address							$R/\overline{W}$
7	6	5	4	3	2	1	0

8-bit device data							
7	6	5	4	3	2	1	0

- 7- or 10-bit address

hard coded into the chip

- Peripherals often have fixed and programmable address portions

[redacted]

(Q) Is there any way to use two chips with the same I2C Address?

Data Packet in I2C

- Addressing

Start	Device address + $R/\overline{W}$	ACK	Data #1	ACK/ NACK	Stop

7-bit device address							$R/\overline{W}$
7	6	5	4	3	2	1	0

8-bit device data							
7	6	5	4	3	2	1	0

- 7- or 10-bit address
- Peripherals often have fixed and programmable address portions

hard coded into the chip

- Addresses starting with 0000 or 1111 have special functions
  - 0000000 Is a General Call Address – E.g. system reset
  - 1111XXX Address Extension
    - 1111111 Address Extension – Next Bytes are the Actual Address

(Q) Is there any way to use two chips with the same I2C Address?

(A) Yes, we can use {chip select, enable bits, or/and clock gating} (if available)

Data Packet in I2C

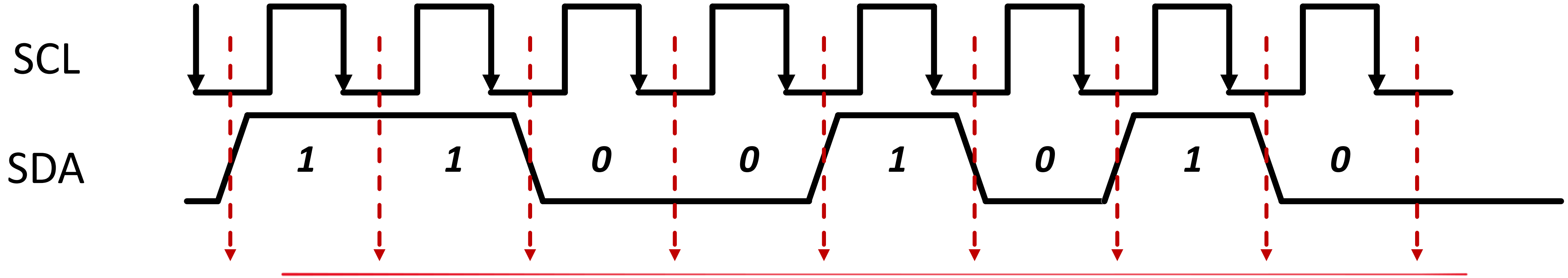
- Data transfer

Start	Device address + $R/\overline{W}$	ACK	Data #1	ACK/ NACK	Stop

7- or 10-bit address

7-bit device address							$R/\overline{W}$
7	6	5	4	3	2	1	0

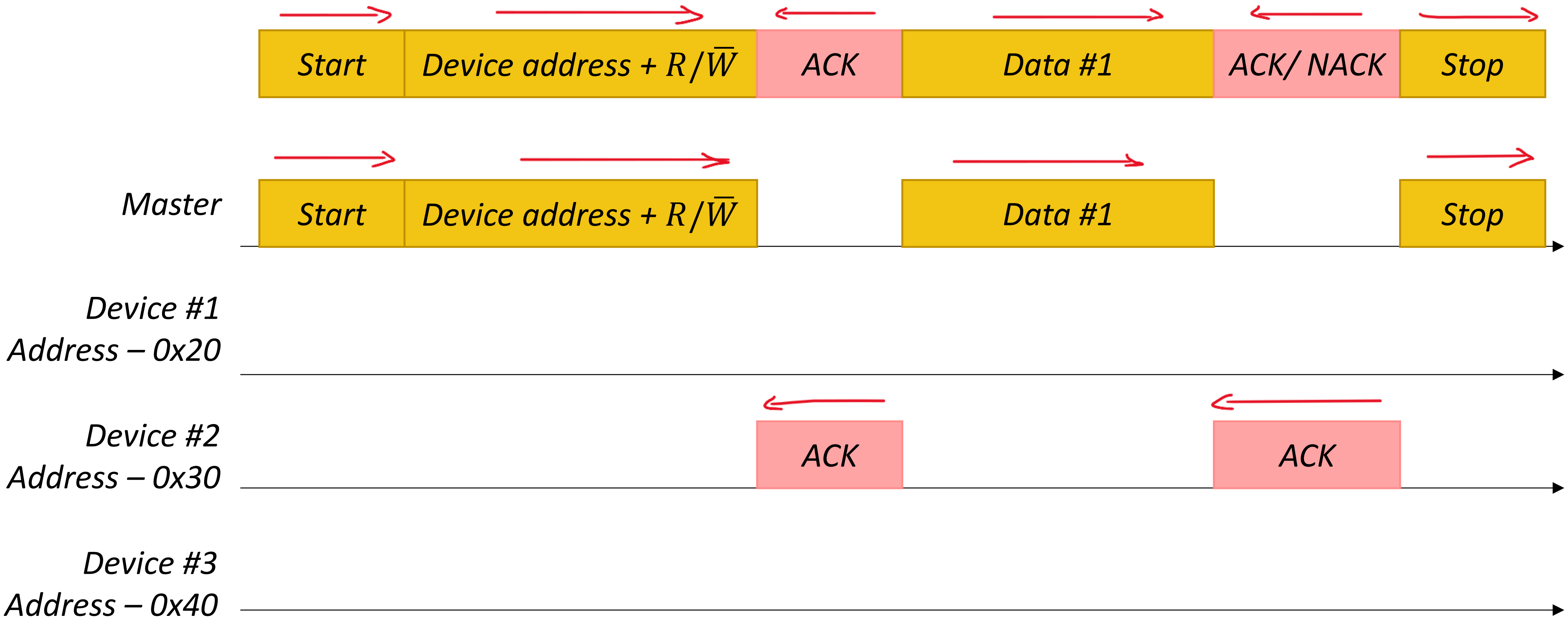
8-bit device data							
7	6	5	4	3	2	1	0



Changes in SDA is allowed when SCL is LOW. SDA must be stable when SCL is HIGH.

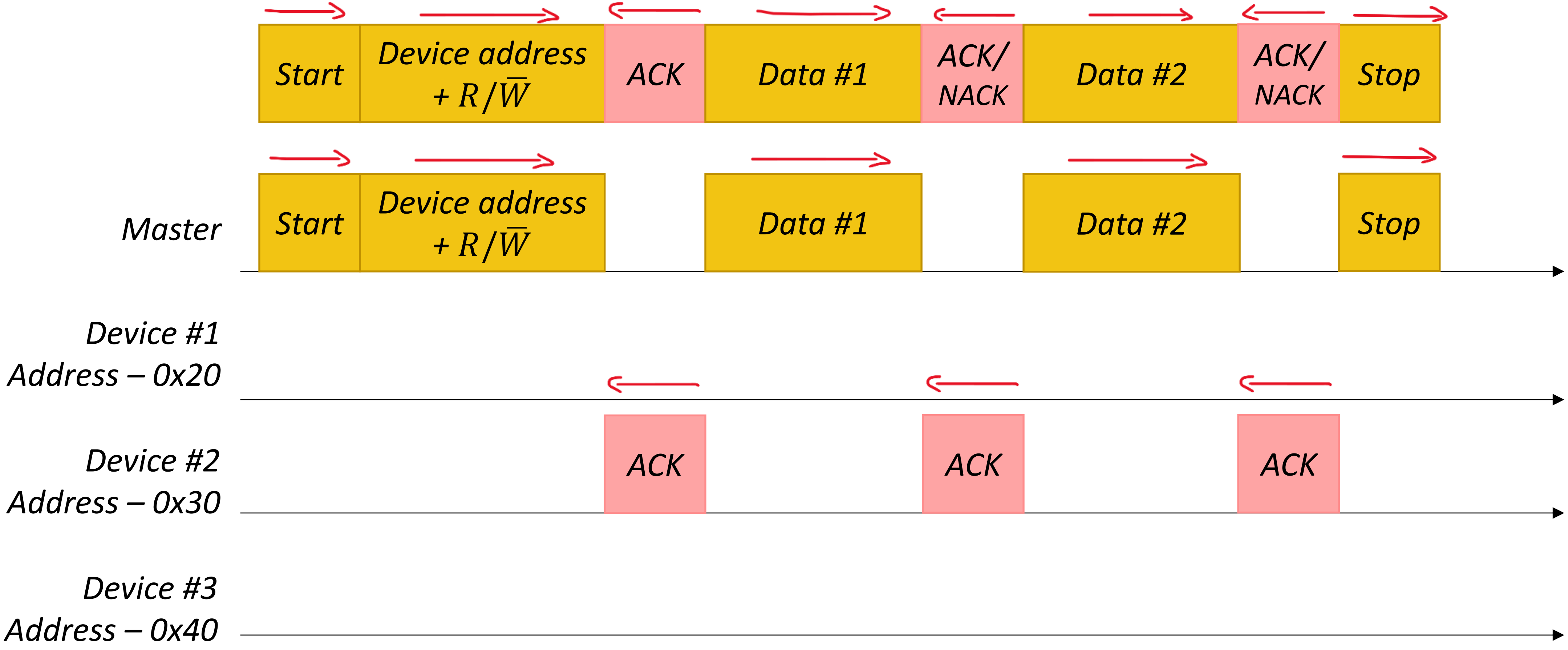
Example of Data Transfer in I2C

- The master writing ONE byte to a device at address 0x30



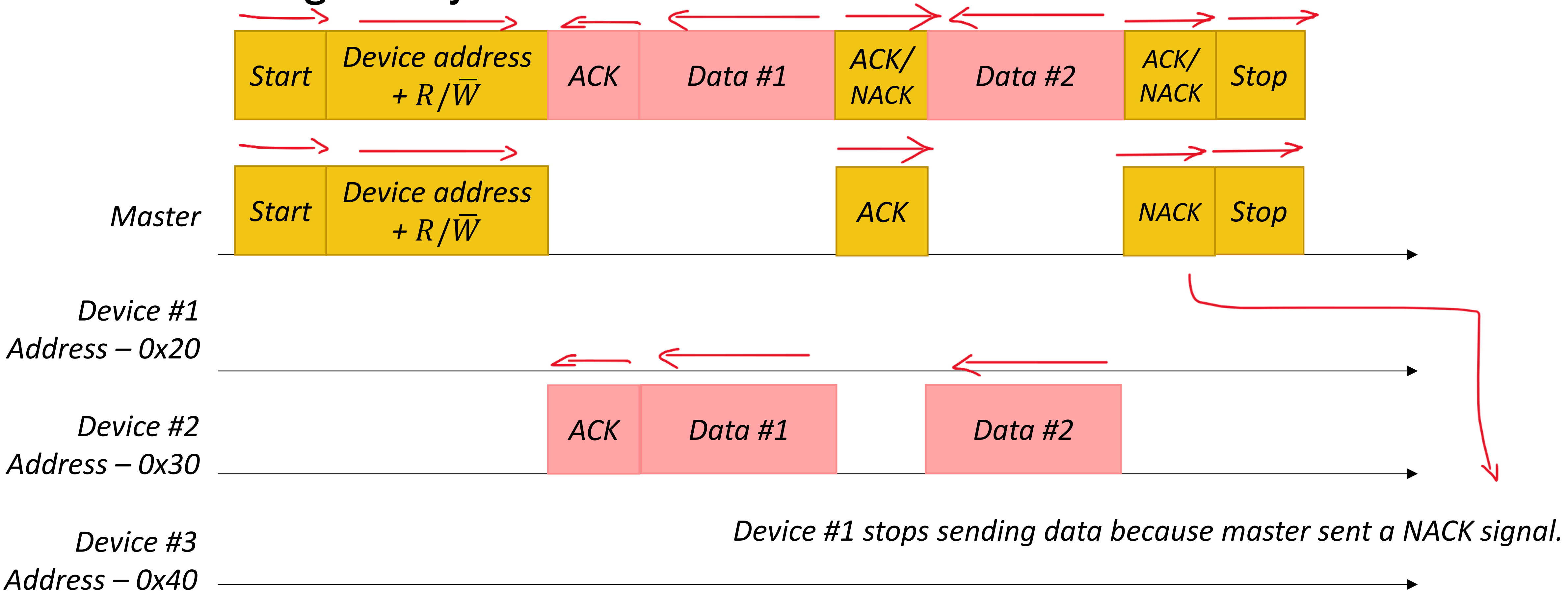
Example of Data Transfer in I2C

- The master writing TWO bytes to a device at address 0x30



Example of Data Transfer in I2C

- The master reading TWO bytes from a device at address 0x30



I2C Modes

- Modes, each designating a maximum clock frequency
  - both the master and the device(s) should support

- Modes
  - Standard Mode → up to 100 KHz
  - Fast Mode → up to 400 KHz
  - Fast Mode Plus → up to 1 MHz
  - High Speed Mode → up to 3.4 MHz
  - Ultra Fast Mode →up to 5 MHz (unidirectional)

Most MCUs and I2C devices support these modes!

More reliable

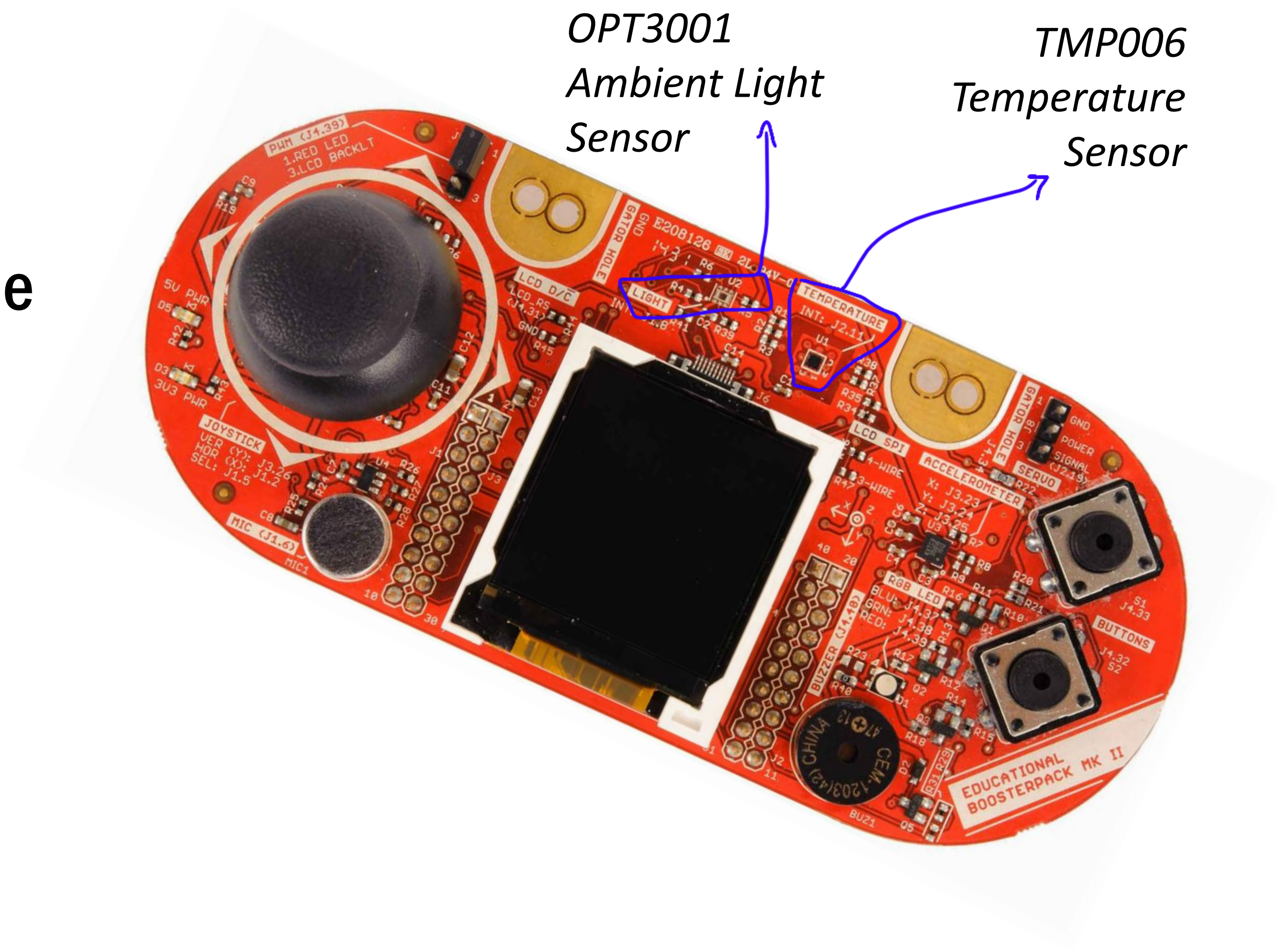
Less reliable

Usages of I2C

- Mostly for low-throughput systems
  - The operational frequency of I2C is low
  - Length of bus limited to a few meter
    - Extender can be used (with data loss and performance degradation)
- Configuration of modules
  - Configuring SDRAM
- Sensors (receiving status) and small actuators (transmitting control)

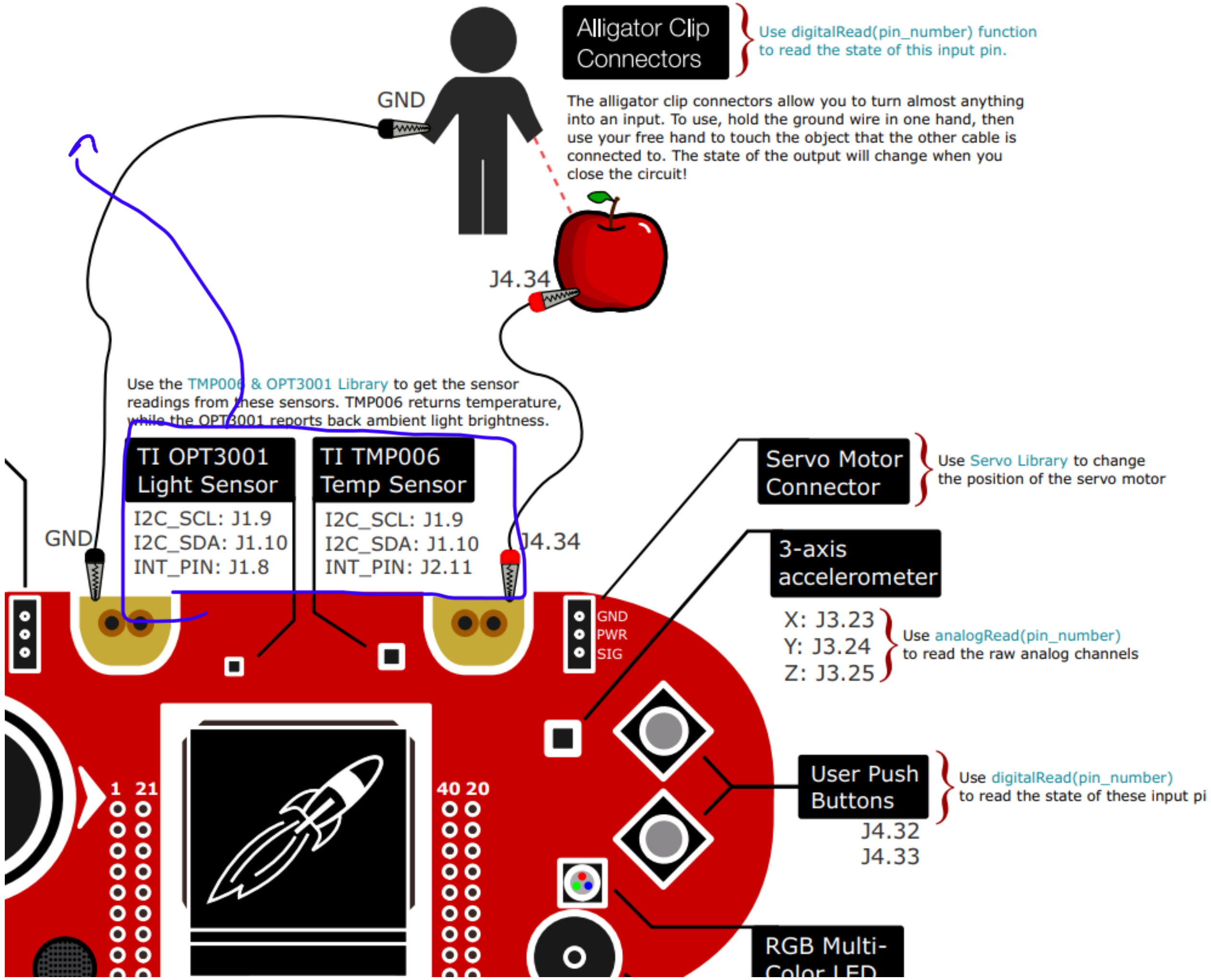
Light Sensor through I2C

- TI OPT3001 Ambient Light Sensor (ALS)
- The OPT3001 ALS is a light sensor in the TI Educational BoosterPack plugin module
- The sensor is connected to the MCU via the I2C bus.
  - The address of the sensor is provided in the datasheet of the light sensor.



Light Sensor through I2C

- TI OPT3001 Ambient Light Sensor (ALS)
- The OPT3001 ALS is a light sensor in the TI Educational BoosterPack plugin module
- The sensor is connected to the MCU via the I2C bus.
  - The address of the sensor is provided in the datasheet of the light sensor.

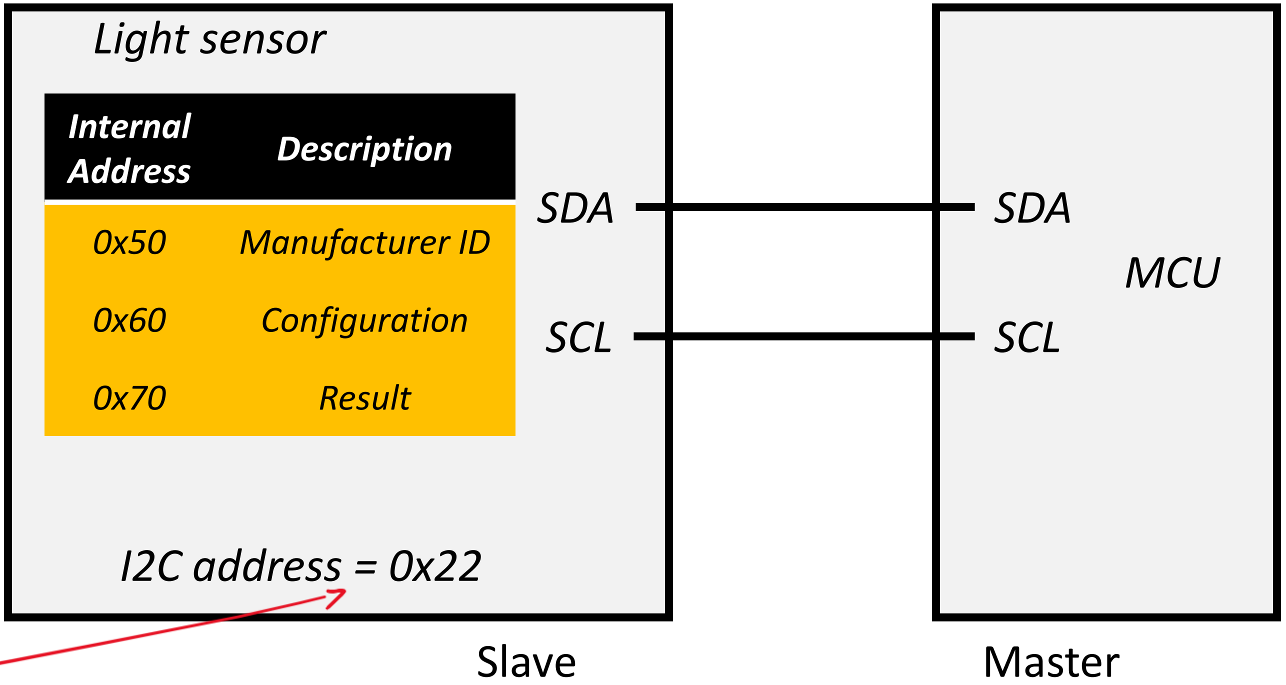


Light Sensor through I2C

- TI OPT3001 Ambient Light Sensor (ALS)

- The OPT3001 ALS is a light sensor in the TI Educational BoosterPack plugin module

- The sensor is connected to the MCU via the I2C bus.
  - The address of the sensor is provided in the datasheet of the light sensor.



Internal Address	Description
0x50	Manufacturer ID
0x60	Configuration
0x70	Result

*The light sensors have **internal registers**. The master can write to (or read from) the registers. The result register is read-only and has the light sensor reading.*

Light Sensor – Internal Register

- Result register (16bit)

E3	E2	E1	E0	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0
Exponent				*Fractional (Component)*											

The most recent light to digital conversion

E3	E2	E1	E0	Full-Range lux (lux)	Lux per LSB (lux)
0	0	0	0	40.95	0.01
0	0	0	1	81.9	0.02
0	0	1	0	163.8	0.04
0	0	1	1	327.6	0.08
0	1	0	0	655.2	0.16
0	1	0	1	1310.4	0.32
0	1	1	0	2620.8	0.64
0	1	1	1	5241.6	1.28
1	0	0	0	10483.2	2.56
1	0	0	1	20966.4	5.12
1	0	1	0	41932.8	10.24
1	0	1	1	83865.6	20.48

Brightness = (Lux per LSB).($R_{[11:0]}$)

<u>*Example.*</u>

E = 0101 & R = 111100001010

***Brightness** = 0.32 x 3850 = 1232 lux*

Direct sunlight:	100,000 to 120,000 lux
Overcast day:	1,000 to 10,000 lux
Office lighting:	300 to 500 lux
Moonlight:	0.1 lux
Twilight:	10 lux
Street lighting at night:	10-20 lux

Max per $E_{3..0}$ = (2^{12} -1).(Lux per LSB)

Light Sensor – Internal Register

- Configuration register (16bit)

RN3	RN2	RN1	RN0	CT	R10	R9	8	7	6	5	4	3	2	1	0
Sets the exponent				CT	Mode								ME		

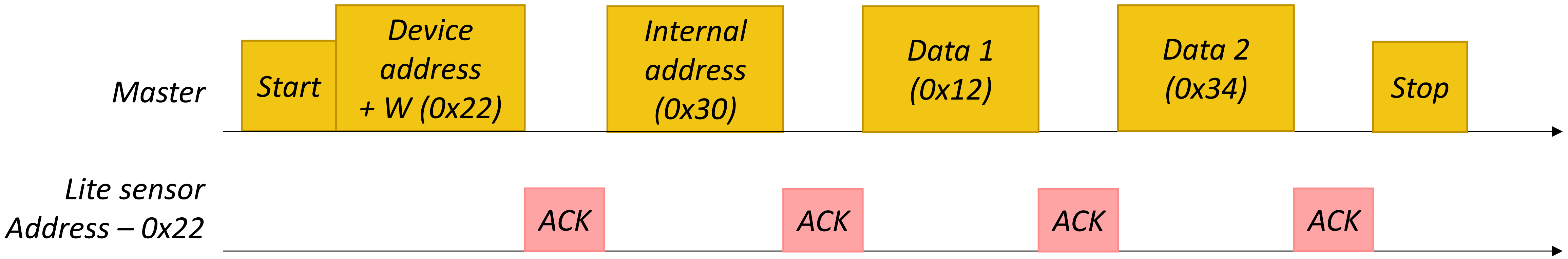
E3	E2	E1	E0	Full-Range lux (lux)	Lux per LSB (lux)
0	0	0	0	40.95	0.01
0	0	0	1	81.9	0.02
0	0	1	0	163.8	0.04
0	0	1	1	327.6	0.08
0	1	0	0	655.2	0.16
0	1	0	1	1310.4	0.32
0	1	1	0	2620.8	0.64
0	1	1	1	5241.6	1.28
1	0	0	0	10483.2	2.56
1	0	0	1	20966.4	5.12
1	0	1	0	41932.8	10.24
1	0	1	1	83865.6	20.48

- ***RN3:0 – (Range Number Field)***
  - *at 1100b: The range is automatically chosen by the sensor (powers up in 1100b - automatic full-scale setting mode).*
- ***CT – Conversion time (Length of time for conversion)***
  - *0 – 100 ms*
  - *1 – 800 ms*
- ***Mode – Mode of operation***
  - *00 – Shutdown (default)*
  - *01 – Single-shot*
  - *10,11 – Continuous conversion*
- ***ME – Mask exponent bits***
  - *when it is set, the exponent in the result register is 0000b.*

Disabling exponent field

Writing data to an internal register

- I2C address for the ALS is 0x22.
- Internal register address is 0x30.
- The data is 0x1234.

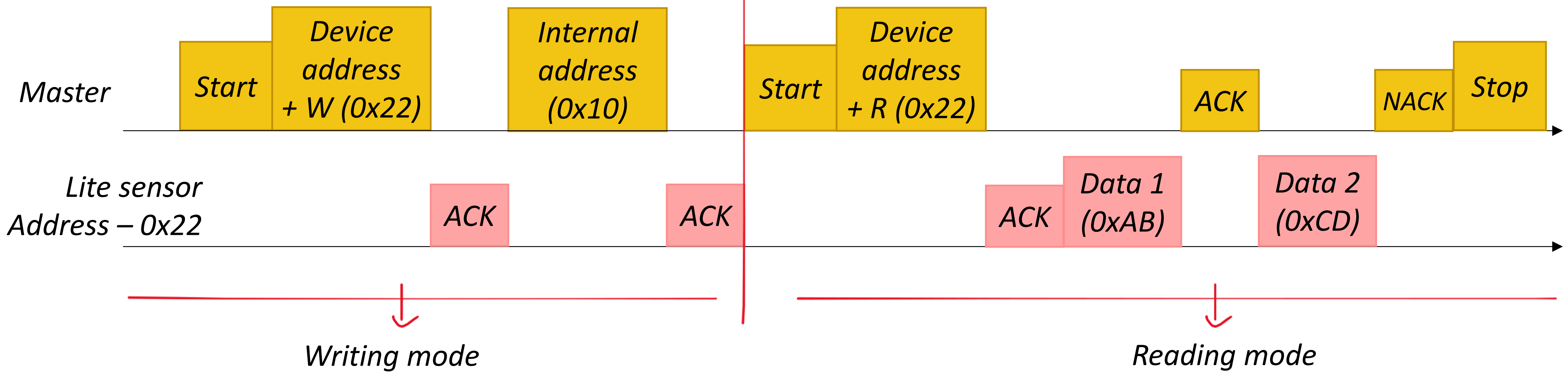


Reading data from an internal register

- I2C address for the ALS is 0x22.
- Internal register address is 0x10.
- The data is 0xABCD.

*Recurring start is '**starting again without sending the stop bit**'.*

If a stop bit is sent, then some other master can take control of the bus.



eUSCI Module - Revisiting

- Enhanced Universal Serial Communication Interface (eUSCI)
  - Supports multiple serial communication protocols
    - e.g., UART
    - e.g., serial peripheral interface (SPI)
    - e.g., inter-integrated Circuit (I2C)

- In MSP430FR6989
  - There are two implementations of eUSCI_A
    - eUSCI_A0 and eUSCI_A1
  - There are two implementations of eUSCI_A
    - eUSCI_B0 and eUSCI_B1

eUSCI_A supports

- *UART*
- *SPI*

eUSCI_B supports

- ***I2C***
- *SPI*

*The boosterpack I2C bus is connected to the eUSCI_B1 module via **pins P4.0** and **P4.1**.*
Refer to the boosterpack and launchpad user guide for detailed information on all the connected pins.

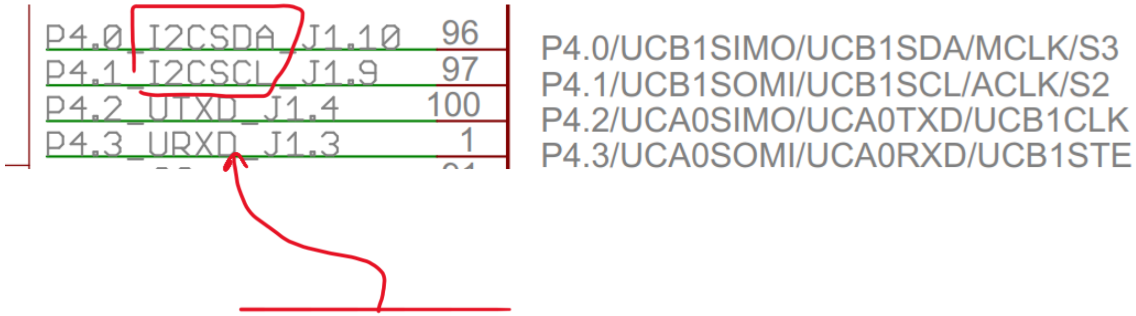
eUSCI Module - Revisiting

- Enhanced Universal Serial Communication Interface (eUSCI)
  - Supports multiple serial communication protocols
    - e.g., UART
    - e.g., serial peripheral interface (SPI)
    - e.g., inter-integrated Circuit (I2C)

eUSCI_A supports
- *UART*
- *SPI*

eUSCI_B supports
- ***<u>I2C</u>***
- *SPI*

- In MSP430FR6989
  - There are two implementations of eUSCI_A
    - eUSCI_A0 and eUSCI_A1
  - There are two implementations of eUSCI_A
    - eUSCI_B0 and eUSCI_B1



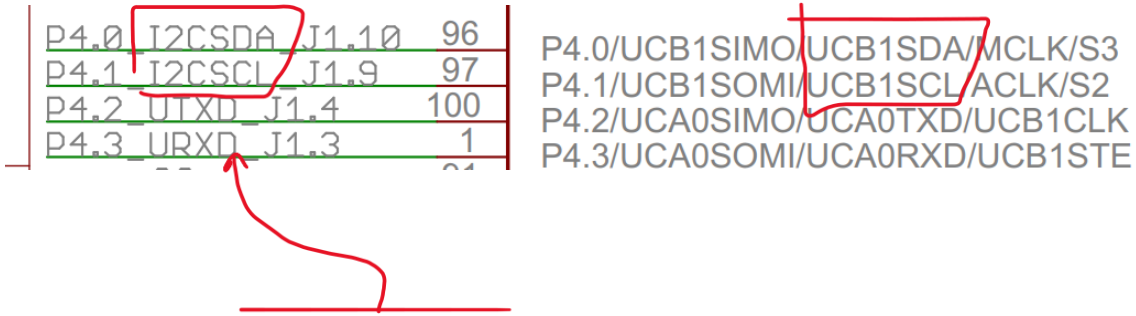
*The boosterpack I2C bus is connected to the eUSCI_B1 module via **pins P4.0** and **P4.1**.*
Refer to the boosterpack and launchpad user guide for detailed information on all the connected pins.

eUSCI Module - Revisiting

- Enhanced Universal Serial Communication Interface (eUSCI)
  - Supports multiple serial communication protocols
    - e.g., UART
    - e.g., serial peripheral interface (SPI)
    - e.g., inter-integrated Circuit (I2C)

PIN NAME (P4.x)	x	FUNCTION	CONTROL BITS AND SIGNALS (1)			
			P4DIR.x	P4SEL1.x	P4SEL0.x	LCDSz
P4.0/UCB1SIMO/UCB1SDA/MCLK/ Sz	0	P4.0 (I/O)	I: 0; O: 1	0	0	0
		N/A	0	0	1	0
		Internally tied to DVSS	1			
		UCB1SIMO/UCB1SDA	X (2)	1	0	0
		N/A	0	1	1	0
		MCLK	1			
		Sz (3)	X	X	X	1
P4.1/UCB1SOMI/UCB1SCL/ACLK/ Sz	1	P4.1 (I/O)	I: 0; O: 1	0	0	0
		N/A	0	0	1	0
		Internally tied to DVSS	1			
		UCB1SOMI/UCB1SCL	X (2)	1	0	0
		N/A	0	1	1	0
		ACLK	1			
		Sz (3)	X	X	X	1

- In MSP430FR6989
  - There are two implementations of eUSCI_A
    - eUSCI_A0 and eUSCI_A1
  - There are two implementations of eUSCI_A
    - eUSCI_B0 and eUSCI_B1



*The boosterpack I2C bus is connected to the eUSCI_B1 module via **pins P4.0** and **P4.1**.*
Refer to the boosterpack and launchpad user guide for detailed information on all the connected pins.

eUSCI Module - Revisiting

- Enhanced Universal Serial Communication Interface (eUSCI)
  - Supports multiple serial communication protocols
    - e.g., UART
    - e.g., serial peripheral interface (SPI)
    - e.g., inter-integrated Circuit (I2C)

PIN NAME (P4.x)	x	FUNCTION	CONTROL BITS AND SIGNALS [1]			
			P4DIR.x	P4SEL1.x	P4SEL0.x	LCDSz
P4.0/UCB1SIMO/UCB1SDA/MCLK/Sz	0	P4.0 (I/O)	I: 0; O: 1	0	0	0
		N/A	0	0	1	0
		Internally tied to DVSS	1			
		UCB1SIMO/UCB1SDA	X [2]	1	0	0
		N/A	0	1	1	0
		MCLK	1			
		Sz [3]	X	X	X	1
P4.1/UCB1SOMI/UCB1SCL/ACLK/Sz	1	P4.1 (I/O)	I: 0; O: 1	0	0	0
		N/A	0	0	1	0
		Internally tied to DVSS	1			
		UCB1SOMI/UCB1SCL	X [2]	1	0	0
		N/A	0	1	1	0
		ACLK	1			
		Sz [3]	X	X	X	1

- In MSP430FR6989
  - There are two implementations of eUSCI_A
    - eUSCI_A0 and eUSCI_A1
  - There are two implementations of eUSCI_A
    - eUSCI_B0 and eUSCI_B1

```
// Configure pins to I2C functionality
// (UCB1SDA same as P4.0) (UCB1SCL same as P4.1)
// (P4SEL1=11, P4SEL0=00) (P4DIR=xx)
P4SEL1 |= (BIT1|BIT0);
P4SEL0 &= ~(BIT1|BIT0);
```

*The boosterpack I2C bus is connected to the eUSCI_B1 module via **pins P4.0** and **P4.1**.*
Refer to the boosterpack and launchpad user guide for detailed information on all the connected pins.

I2C Programming

- **Initialization of I2C**

eUSCI_B module (I2C) in reset mode to safely configure the registers

Enabling I2C through pins using P4 (from GPIO to I2C mode.)

Enabling the master mode

Clock divider (e.g., SMLK/8 for here)

Once the configuration is done, the reset for the eUSCI_B will be disabled.

```
// Configure eUSCI in I2C leader mode
void Initialize_I2C(void) {
   // Enter reset state before the configuration starts...
   UCB1CTLW0 |= UCSWRST;

   // Divert pins to I2C functionality
   P4SEL1 |= (BIT1|BIT0);
   P4SEL0 &= ~(BIT1|BIT0);

   // Keep all the default values except the fields below...
   // (UCMode 3:I2C) (Master Mode) (UCSSEL 1:ACLK, 2,3:SMCLK)
   UCB1CTLW0 |= UCMODE_3 | UCMST | UCSSEL_3;

   // Clock divider = 8 (SMCLK @ 1.048 MHz / 8 = 131 KHz)
   UCB1BRW = 8;

   // Exit the reset mode
   UCB1CTLW0 &= ~UCSWRST;
}
```

I2C Programming

- **Reading and Writing (TI OPT3001 light sensor)**

Reading and wiring a word from/to I2C address/register
(refer to Lab Manual Appendix)

```
// Read a word (2 bytes) from I2C (address, register)
int i2c_read_word(unsigned char i2c_address, unsigned char i2c_reg, unsigned int * data)

// Write a word (2 bytes) to I2C (address, register)
int i2c_write_word(unsigned char i2c_address, unsigned char i2c_reg, unsigned int data)
```

Initialization of I2C (slide 23 here)
Initialization of UART (M6 – Slide 36)

```
void main(){
        unsigned int data_rd, data_wr;
        Initialize_I2C();
        Initialize_UART();

         // The variable data_w is passed by reference
        i2c_read_word(0x22, 0x50, &data_rd);
        // Print to serial port
        uart_write_uint16(data_rd);

         // The variable to be written into I2C address
        data_wr = 0xABCD;
        // The variable data_wr is passed by value
        i2c_write_word(0x22, 0x60, data_wr);
}
```

I2C Registers

- Some of the I2C addresses and register addresses are examples.

- More details
  - OPT3001 Datasheet http://www.ti.com/lit/ds/symlink/opt3001.pdf
  - BoosterPack Datasheet http://www.ti.com/lit/ug/slau599a/slau599a.pdf

Thank You!

Questions?

Email: kamali@ucf.edu
UCF HEC 435 (407) 823 – 0764
https://www.ece.ucf.edu/~kamali/

HAVEN Research Group
https://haven.ece.ucf.edu/