ECE3411 – Fall 2016

Lab 7a-b.

I²C RedBot

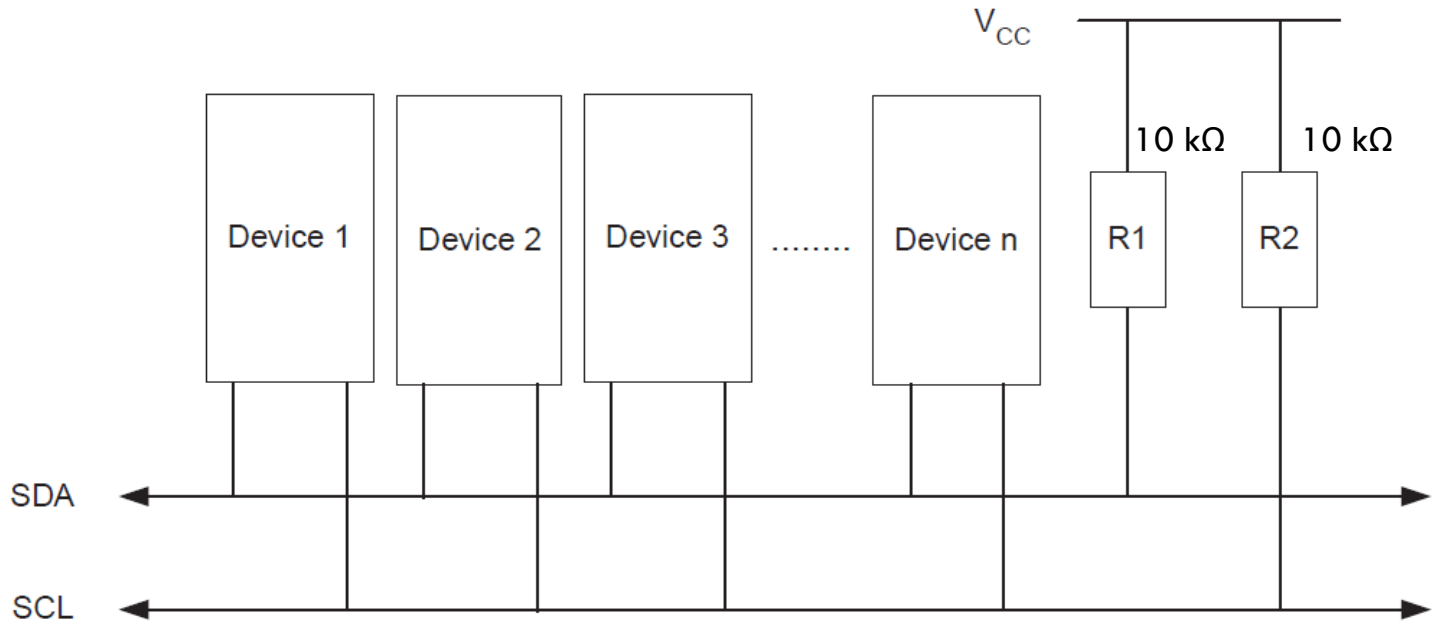
Marten van Dijk, Chenglu
Department of Electrical & Computer Engineering
University of Connecticut
Email: {marten.van_dijk, chenglu.jin}@uconn.edu

Slides of I2C are copied from Lab 7b, ECE3411 – Fall 2015, by Marten van Dijk and Syed Kamran Haider
Some of these slides are extracted or copied from "RedBot Project" offered at Sung Yeul Park in Spring 2016

UCONN

I²C: Inter Integrated Circuit

- Also known as Two Wire Interface (TWI)
- Allows up to 128 different devices to be connected using only two bi-directional bus lines, one for clock (SCL) and one for data (SDA).
- A pull-up resistor (typically 10 kΩ) is needed for each of the TWI bus lines.
- All devices connected to the bus have individual addresses.



I^2C Terminologies

- I^2C (TWI) protocol allows several devices (up to 128) to be connected.
- Each device is identified by a configurable 7-bit address.
- Each device can communicate with any other device
  - The transmitter address the receiver by its 7-bit address.

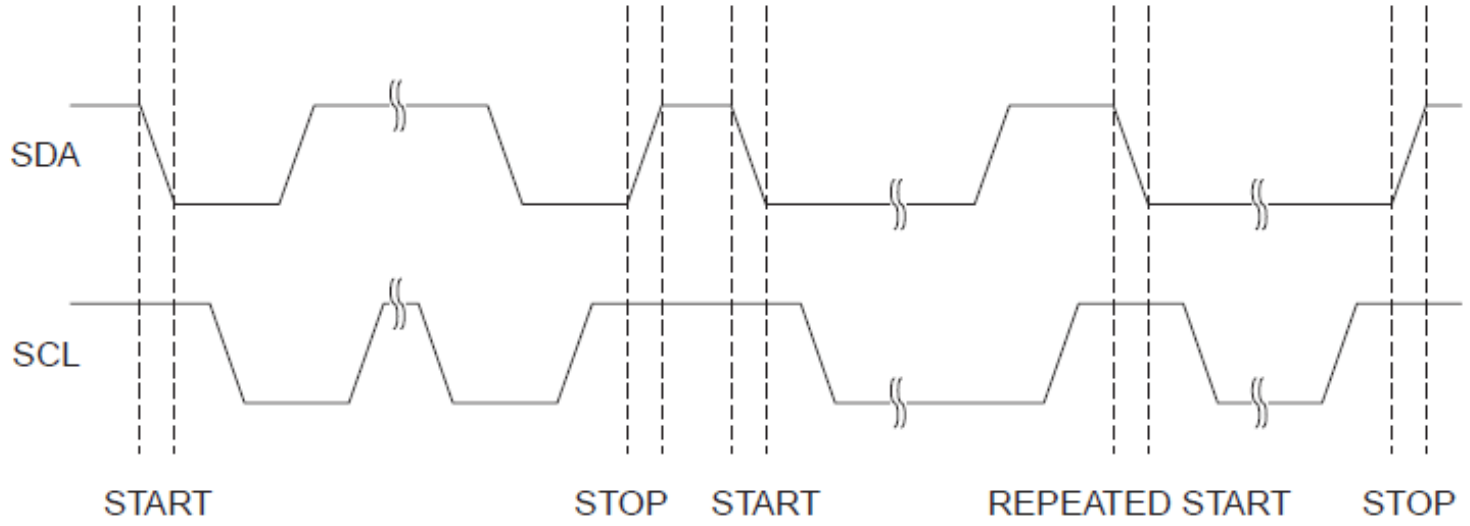
Table 21-1. TWI Terminology

Term	Description
Master	The device that initiates and terminates a transmission. The Master also generates the SCL clock.
Slave	The device addressed by a Master.
Transmitter	The device placing data on the bus.
Receiver	The device reading data from the bus.

I²C START and STOP Conditions

- START and STOP conditions are signaled by changing the level of the SDA line when the SCL line is high.
- When a new START condition is issued between a START and STOP condition, this is referred to as a REPEATED START condition

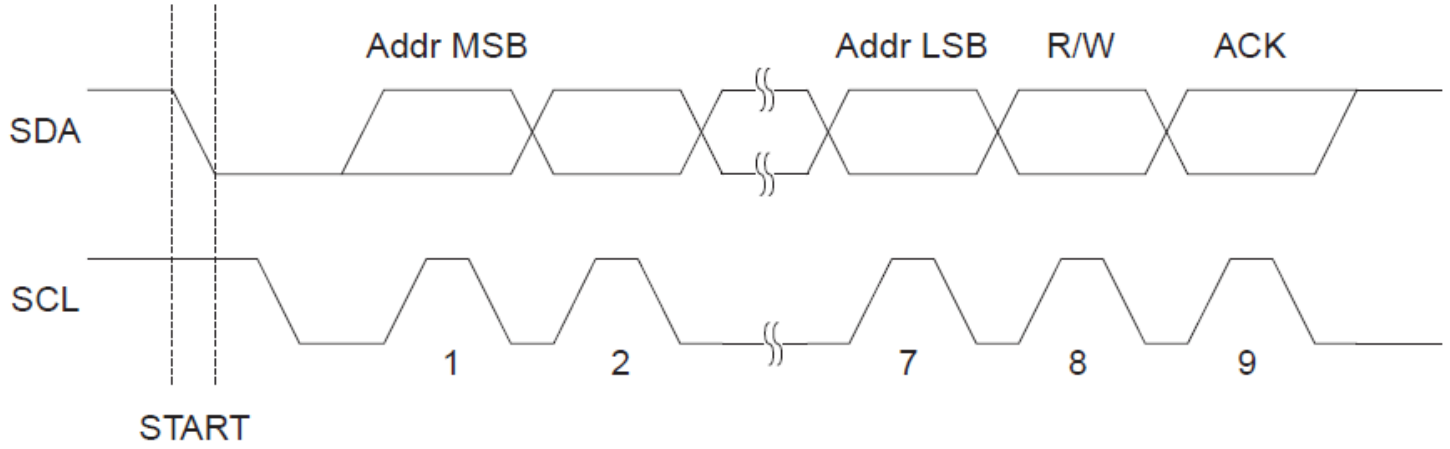
Figure 21-3. START, REPEATED START and STOP conditions



I²C Address Packet Format

- All address packets transmitted on the TWI bus are 9 bits long:
  - 7 address bits, one READ/WRITE control bit and an acknowledge bit.
- When a Slave recognizes that it is being addressed, it should acknowledge by pulling SDA low in the ninth SCL (ACK) cycle.
- The Master can then transmit a STOP condition, or a REPEATED START condition to initiate a new transmission.

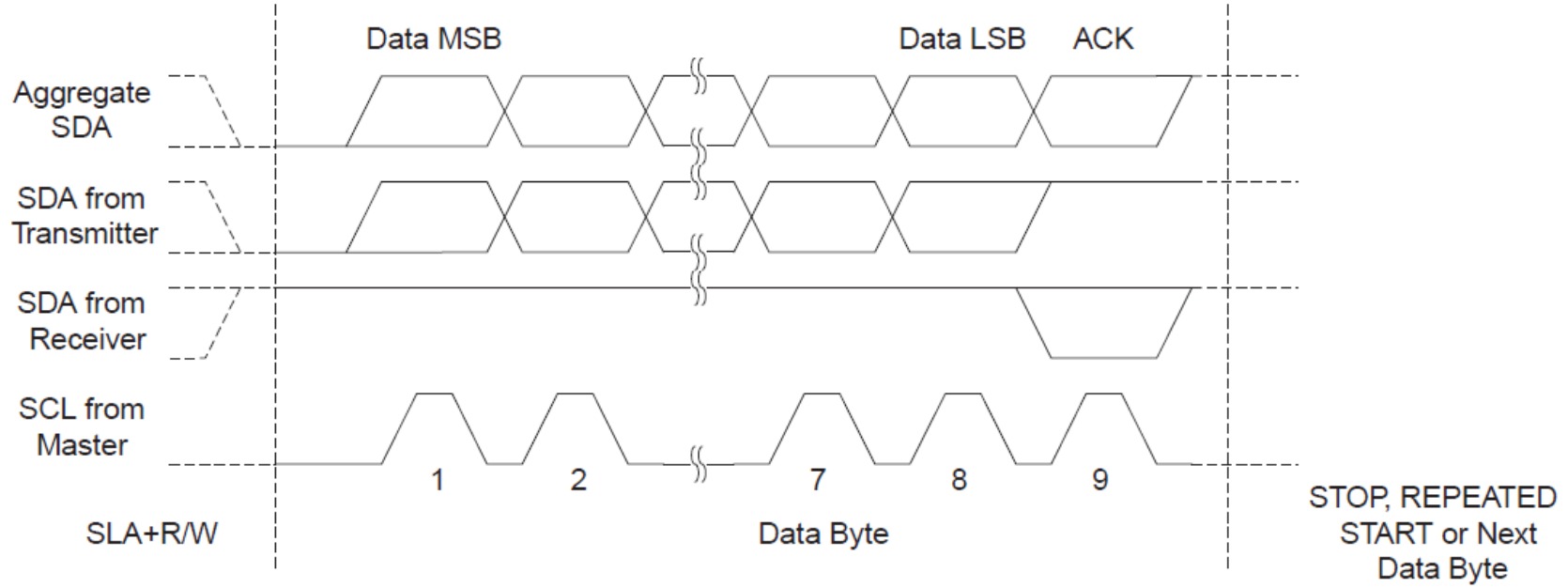
Figure 21-4. Address Packet Format



I²C Data Packet Format

- All data packets transmitted on the TWI bus are 9 bits long:
  - One data byte and one acknowledge bit.
- An Acknowledge (ACK) is signaled by the Receiver pulling the SDA line low during the ninth SCL cycle. If the Receiver leaves the SDA line high, a NACK is signaled.

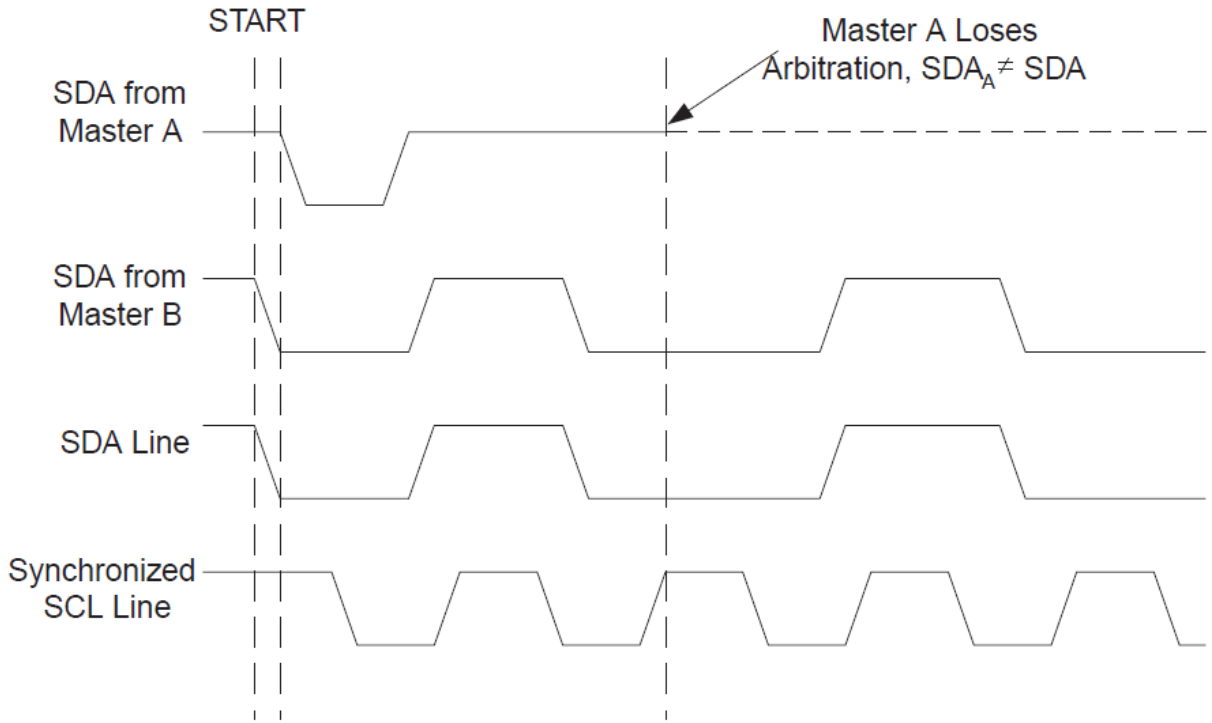
Figure 21-5. Data Packet Format



I^2C Bus Arbitration

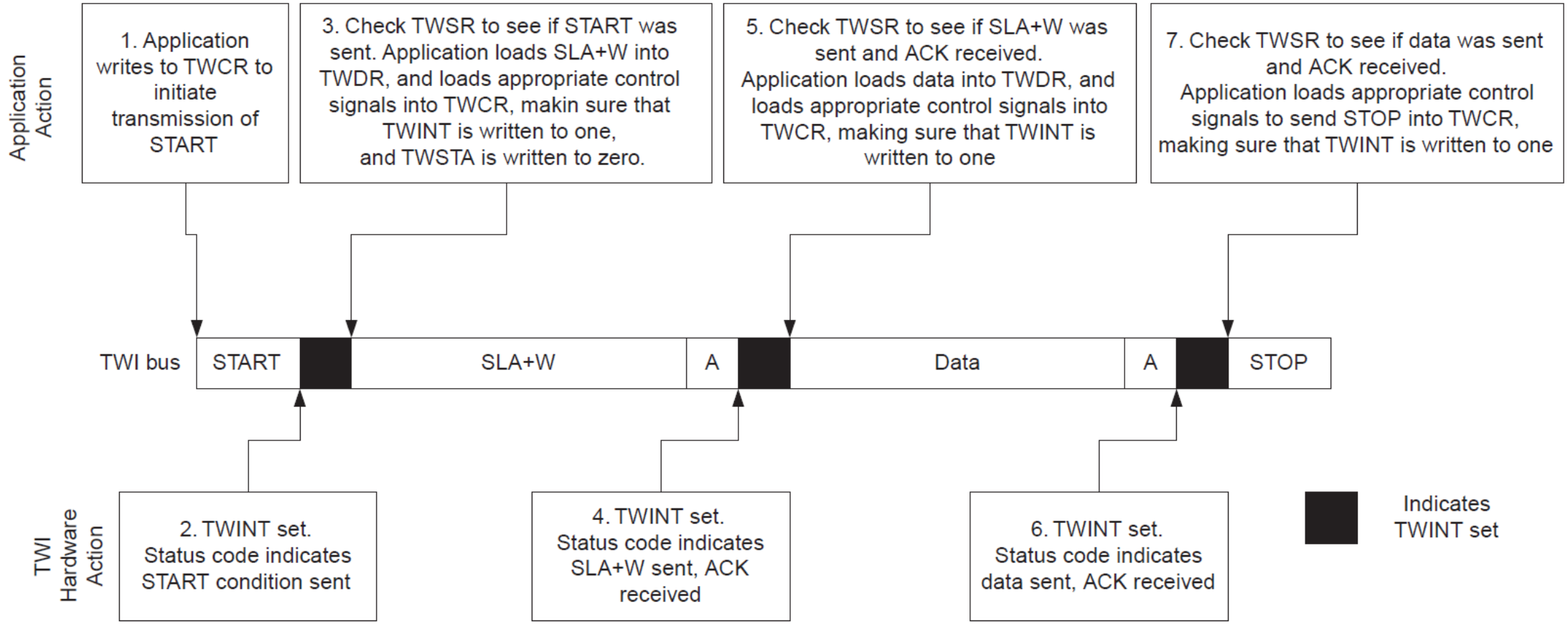
- Arbitration is carried out by all masters continuously monitoring the SDA line after outputting data.
- If the value read from the SDA line does not match the value the Master had output, it has lost the arbitration.

Figure 21-8. Arbitration Between Two Masters



A typical I²C Transmission

Figure 21-10. Interfacing the Application to the TWI in a Typical Transmission



A typical I^2C Transmission Summary

- When the TWI has finished an operation and expects application response, the TWINT Flag is set. The SCL line is pulled low until TWINT is cleared.
- When the TWINT Flag is set, the user must update all TWI Registers with the value relevant for the next TWI bus cycle. As an example, TWDR must be loaded with the value to be transmitted in the next bus cycle.
- After all TWI Register updates and other pending application software tasks have been completed, TWCR is written. When writing TWCR, the TWINT bit should be set.
- Writing a one to TWINT clears the flag. The TWI will then commence executing whatever operation was specified by the TWCR setting.

I²C Transmission Example

```
uint8_t TWI_Master_Transmit(uint8_t Address, uint8_t Data)
{
        TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN);          // Send START condition
        while (!(TWCR & (1<<TWINT)));                    // Wait for TWINT Flag set.
        if ((TWSR & 0xF8) != START)                      // Check value of TWI Status Register.
                ERROR();
        TWDR = (Address << 1) | (WRITE);                 // Load SLA_W (Slave Address & Write) into TWDR Register.
        TWCR = (1<<TWINT) | (1<<TWEN);                   // Clear TWINT bit in TWCR to start transmission of address.
        while (!(TWCR & (1<<TWINT)));                    // Wait for TWINT Flag set.
        if ((TWSR & 0xF8) != MT_SLA_ACK)                 // Check value of TWI Status Register.
                ERROR();
        TWDR = Data;                                     // Load DATA into TWDR Register.
        TWCR = (1<<TWINT) | (1<<TWEN);                   // Clear TWINT bit in TWCR to start transmission of data.
        while (!(TWCR & (1<<TWINT)));                    // Wait for TWINT Flag set.
        if ((TWSR & 0xF8) != MT_DATA_ACK)                // Check value of TWI Status Register.
                ERROR();
        TWCR = (1<<TWINT)|(1<<TWEN)| (1<<TWSTO);         // Transmit STOP condition.
}
```

Note: The code above assumes that several definitions have been made, for example by using include-files.

I²C Reception Example

```
void TWI_Slave_Initialize(uint8_t Address)
{
        TWAR = (Address << 1)|(1);          // Load Slave Address into TWAR Register.
        TWCR = (1<<TWEA)|(1<<TWEN);         // Enable TWI & Acknowledgements.
}
```

```
uint8_t TWI_Slave_Receive(void)
{
        TWCR = (1<<TWEA)|(1<<TWEN);                 // Enable TWI & Acknowledgements.
        while (!(TWCR & (1<<TWINT)));               // Wait for TWINT Flag set (once this slave is addressed)
        if ((TWSR & 0xF8) != 0x60)                  // Check value of TWI Status Register.
                ERROR();
        TWCR = (1<<TWINT) | (1<<TWEN);              // Clear TWINT bit start reception of data.
        while (!(TWCR & (1<<TWINT)));               // Wait for TWINT Flag set.
        if ((TWSR & 0xF8) != 0x80)                  // Check if Data has been received & ACK has been returned
                ERROR();
        TWCR = (1<<TWINT) | (1<<TWEN);              // Clear TWINT bit.
        return TWDR;                                // Read TWDR Register.
}
```

Note: The code above assumes that several definitions have been made, for example by using include-files.

Task1: I^2C Master Slave Communication

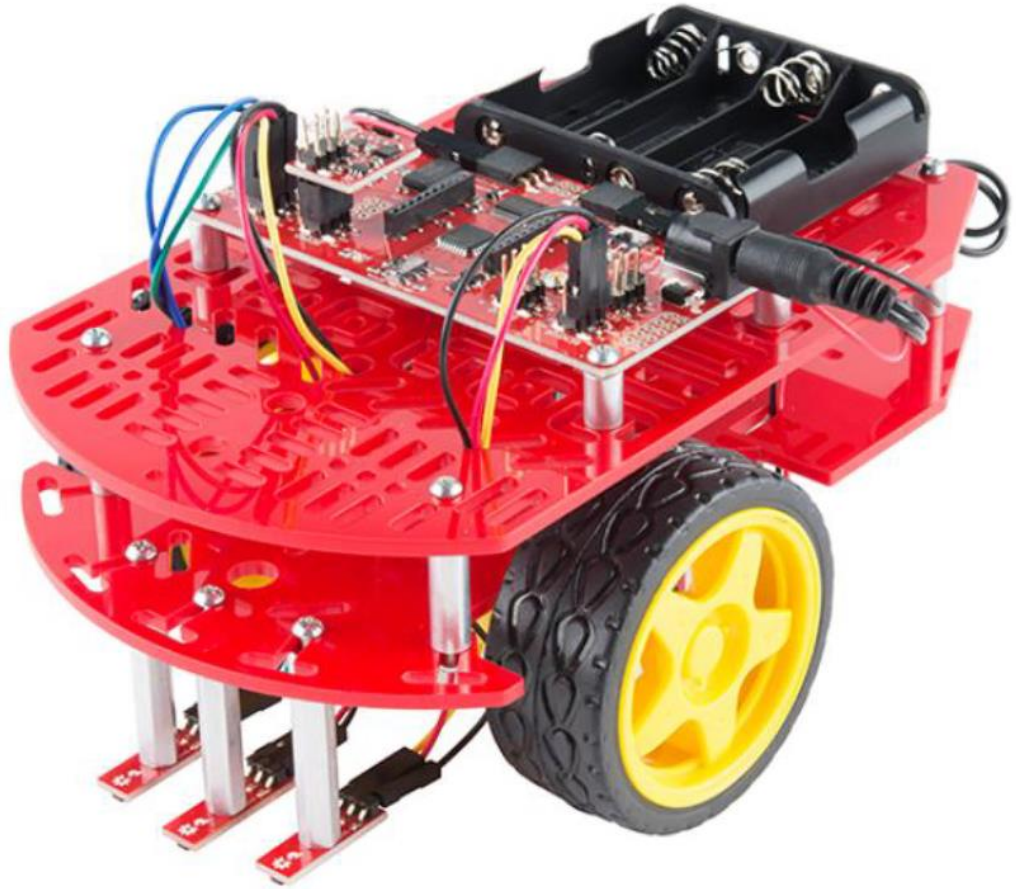
Write a program to send ADC voltage readings to your friend's board over I^2C bus.

- Configure your board as I^2C Master (f_{SCL} = 200kHz) and ask your friend to configure his as I^2C Slave.
- Make proper wire connections of SCK and SDA pins between the two boards.
  Don't forget to put a 10 kΩ pullup resister on each line.
- In Master MCU, read a potentiometer's voltage through ADC every 100ms (only upper 8 bits).
- Transmit Master's voltage value every 100ms.
- For Master, print the transmitted reading on UART.
- For Slave, print the received reading on UART.

Homework: Use I^2C interrupts on both Master and Slave sides for non-blocking I^2C implementation.

RedBot

- Using Sparkfun's RedBot Line Follower kit, you will implement a small robot that follows a line of electrical tape.
- Infrared Sensors are used to sample the desired path in reference to the robot's trajectory.
- Movement is actuated by two PWM controlled H-bridge modules.



- Description: https://www.sparkfun.com/products/13166
- Get started: https://learn.sparkfun.com/tutorials/getting-started-with-the-redbot
- Schematic: https://cdn.sparkfun.com/datasheets/Robotics/RedBot_Mainboard_v14.pdf

Warning: Please do not write anything to EEPROM, since this seems to prevent further programming of the MCU.

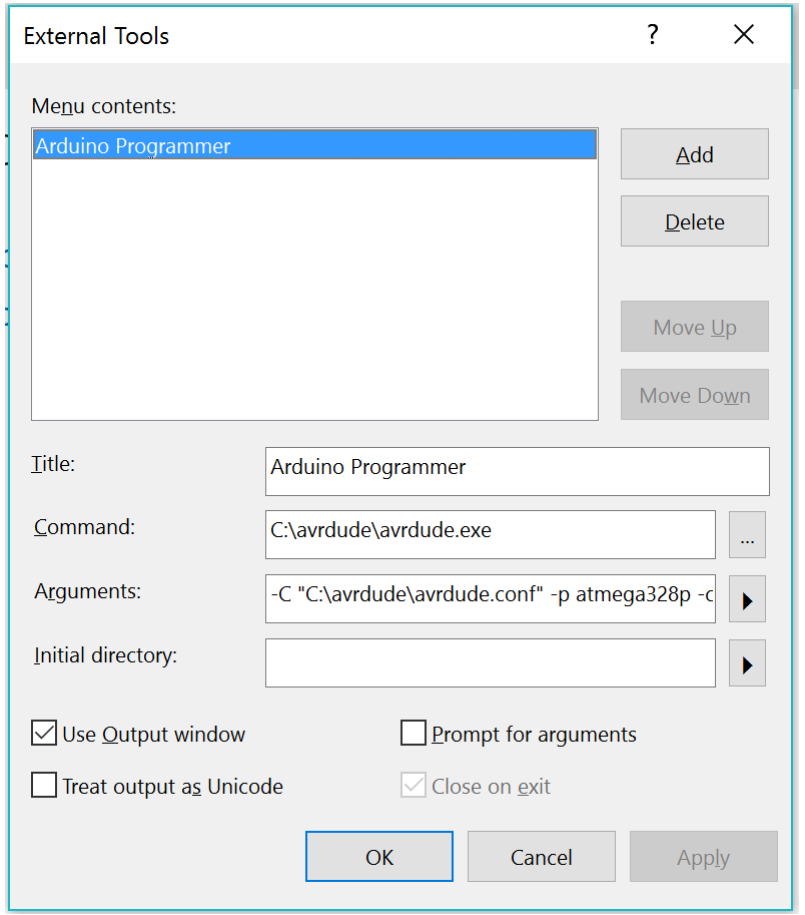
Using Atmel Studio to Program Arduino Board

- Since the board on RedBot is an Arduino Board, we are not able to directly use Atmel Studio to program it. We need to setup an external programmer in Atmel Studio for programming this board.
- Please follow the following steps to setup the external programmer.

1. Download Avrdude from http://mirror.rackdc.com/savannah//avrdude/avrdude-5.11-Patch7610-win32.zip
2. Unzip the downloaded file, rename the directory to ***avrdude***, and copy it into your **C** drive
3. Connect your board to your computer, open Device Manager, check the **COM** port.
4. Open Atmel Studio, go to Tools -> External Tools

Setup External Programmer

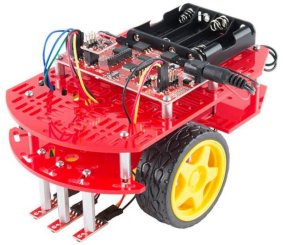
5. Fill the dialog box like this:



6. The Arguments field in the dialog box is

-C "C:\avrdude\avrdude.conf" -p atmega328p -c arduino -P COM9 -b 115200 -U flash:w:"$(ProjectDir)Debug\$(TargetName).hex":i

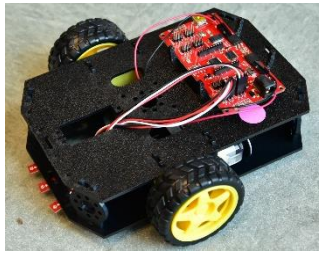
for most of



The Arguments field in the dialog box is

-C "C:\avrdude\avrdude.conf" -p atmega328p -c arduino -P COM9 -b 57600 -U flash:w:"$(ProjectDir)Debug\$(TargetName).hex":i

for most of



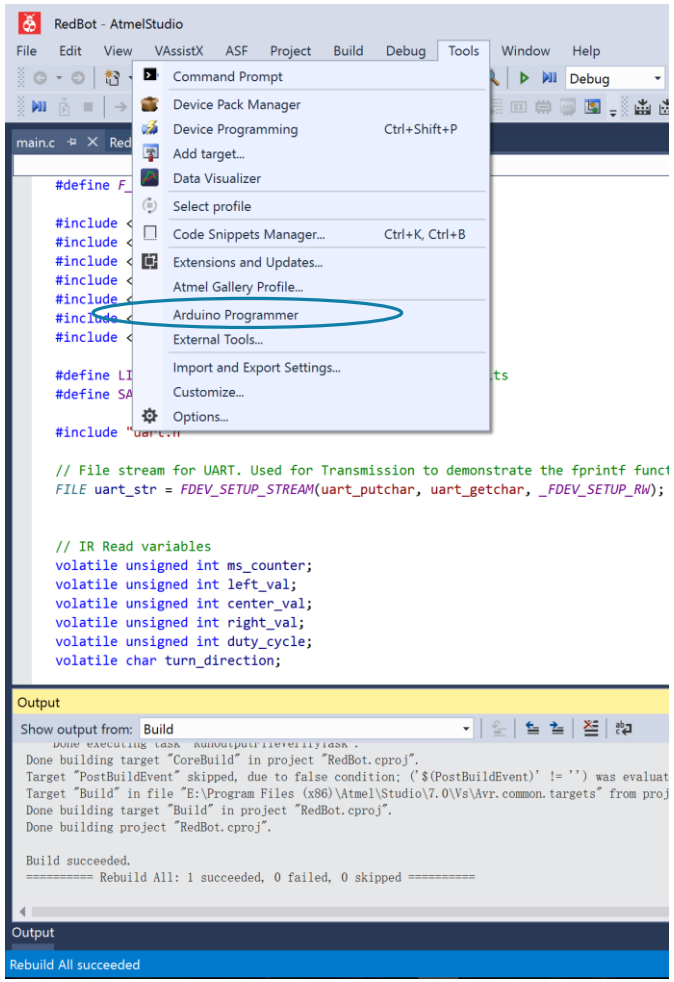
Note: Update your COM number accordingly. If your programmer does not work after setup, change to the other argument. It must be one of these two.

Use External Programmer

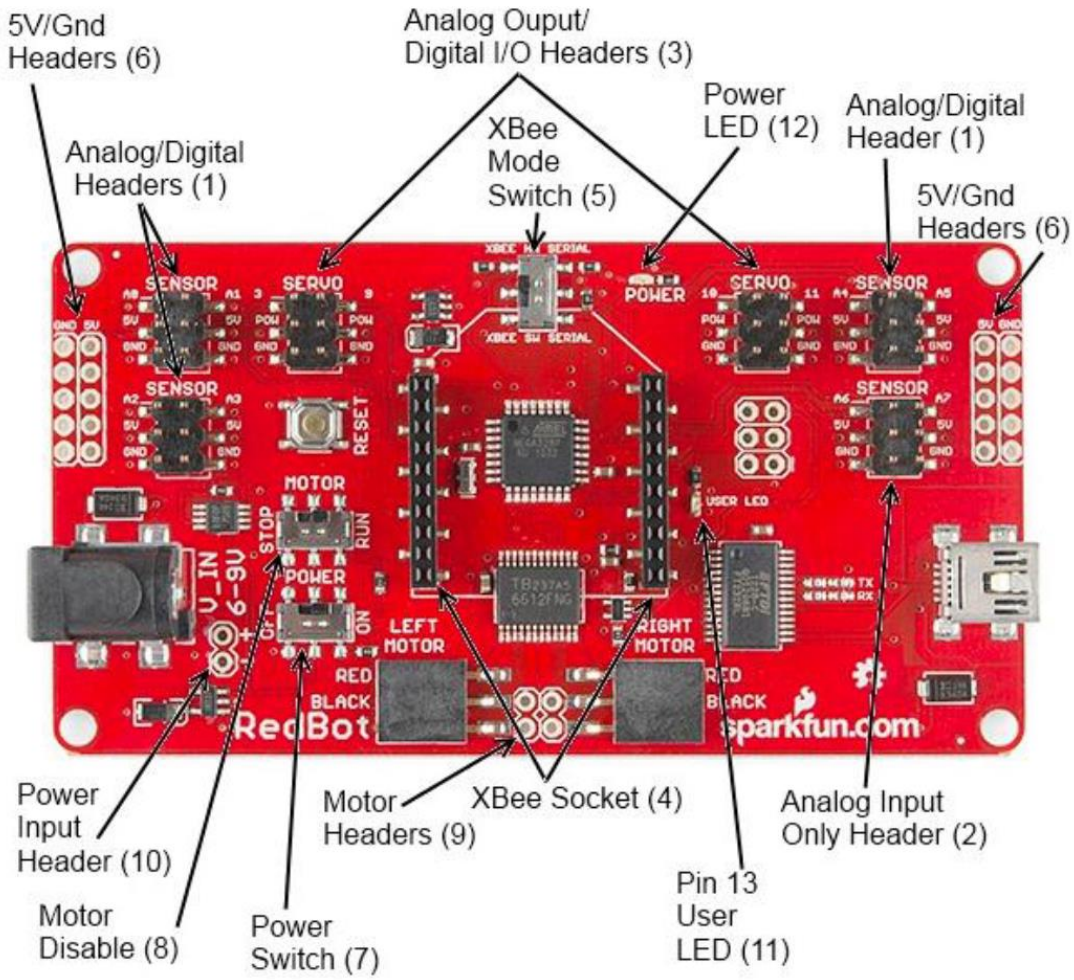
Since the MCU on this board is also ATMega 328P, you just need to create the project and program as usual.

Then first build the project, and click Tools-> Arduino Programmer to program your board.

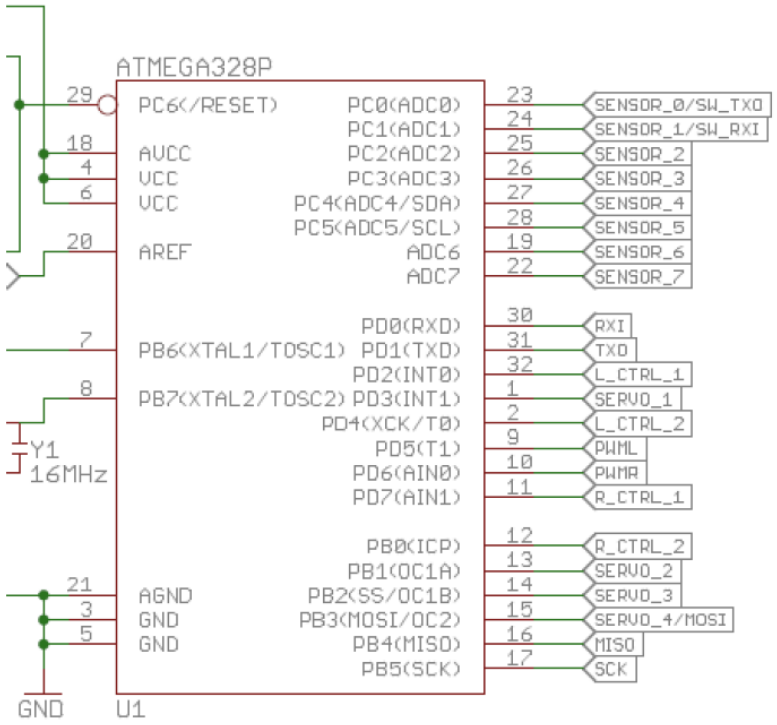
Notice: This Arduino Programmer can only be used to program the board (not to build the solution), so you should always rebuild your solution before you program it.



RedBot Mainboard



ATMega 328P Pin Assignment

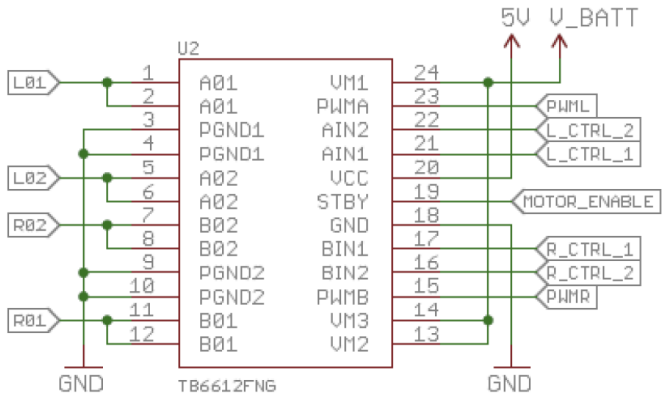


Pin #	Pin Name	Port Name	Ext Circuit
19, 22~29	ADC	PC0~5	ADC input
30, 31	USART	PD0, PO1	XBEE
32, 2	L_CTRL_1/2	PD2, PD4	Left Motor
1, 13~15	SERVO_1/2/3/4	PD3, PB1, PB2, PB3	
9~10	PWML/R	PD5, PD6	
11~12	R_CTRL_1/2	PD7, PB0	Right Motor
29	RESET	PC6	
7,8	CLK	PB6, PB7	
18, 4, 6	AVCC, VCC		
21, 3, 5	AGND, GND		
20	AREF		
16, 17	MISO, SCK	PB4, PB5	6PIN ISP

Line Sensor

- QRE1113: Miniature Reflective Object Sensors
- The sensor works by detecting reflected light coming from its own infrared LED.
- By measuring the amount of reflected infrared light, it can detect transitions from light to dark (lines) or even objects directly in front of it.

Motor Control Mechanism



- TB6612FNG: Dual DC motor driver IC
- Four modes: CW, CCW, Short brake, and stop
- Speed control: PWM duty ratio
- Input1, 2: determine/control Mode
- STBY: motor enable pin

Input				Output		
IN1	IN2	PWM	STBY	OUT1	OUT2	Mode
H	H	H/L	H	L	L	Short brake
L	H	H	H	L	H	CCW
		L	H	L	L	Short brake
H	L	H	H	H	L	CW
		L	H	L	L	Short brake
L	L	H	H	OFF (High impedance)		Stop
H/L	H/L	H/L	L	OFF (High impedance)		Standby

Task 2a: Reading Data from Sensors

- You are required to first initialize ADC and sample three sensors in a round robin fashion.
- Print ADC reads on your screen over UART.
- Test your sensors over a white surface and a black electrical tape, and figure out a proper threshold to distinguish “on tape” and “off tape” states.

Task 2b: Controlling Motors

- Generate the correct command for your motors.
  - Test clockwise and counter-clockwise rotation.
  - Test stop command.
- Generate a PWM signal to control the speed of your motors.

Task 2c: Integration

- Use the data from three sensors to adjust the speed and direction of two motors.
- Test your simple line follower.

Task 3: PID Control

- Improve your line follower by implementing a PID controller in order to make your RedBot move more smoothly.
- Hint: In order to generate an error value in the PID controller, you can first use your thresholds to convert three raw ADC reads to a three-bit value. Then convert this three-bit value into an error value which should be a signed value.
  - E.g. when the left sensor is on the tape and the other two are not on the tape, you first convert it to 0b100, and then convert it to error value -2.
  - E.g. when the left sensor and middle sensor are on the tape, convert it to error -1.
  - This requires you to use fuzzy thresholds, e.g., you can take an average of the sensor values for a white surface and a black tape.
- Hint: The integral of errors can be calculated as a summation of all the errors.
- Hint: The derivative of errors can be calculated as current error – last error.
- Tip: The constant for the I term and D term do not need to be large in comparison with the constant for the P term.

Task 4: Counting the laps

- There is one spot on the track where all the three sensors will sample black tapes.
  - Use this as an indicator of having completed one lap.
  - Whenever the RedBot passes this area, you need to toggle the on-board LED D13, which is connected with PB5.
  - Hint: Don't forget to implement a debounced button to prevent toggling this LED more than once within one lap.

- **You need to demonstrate your RedBot on Dec. 7th during the lab hours.**
- **Submit your lab report on Dec. 7th together with your source code.**