Interaction Design, Microcontrollers, & Communication

September 10, 2019

Lab Review!

I like, I wish

Cool Frankenlights

1 hour rule

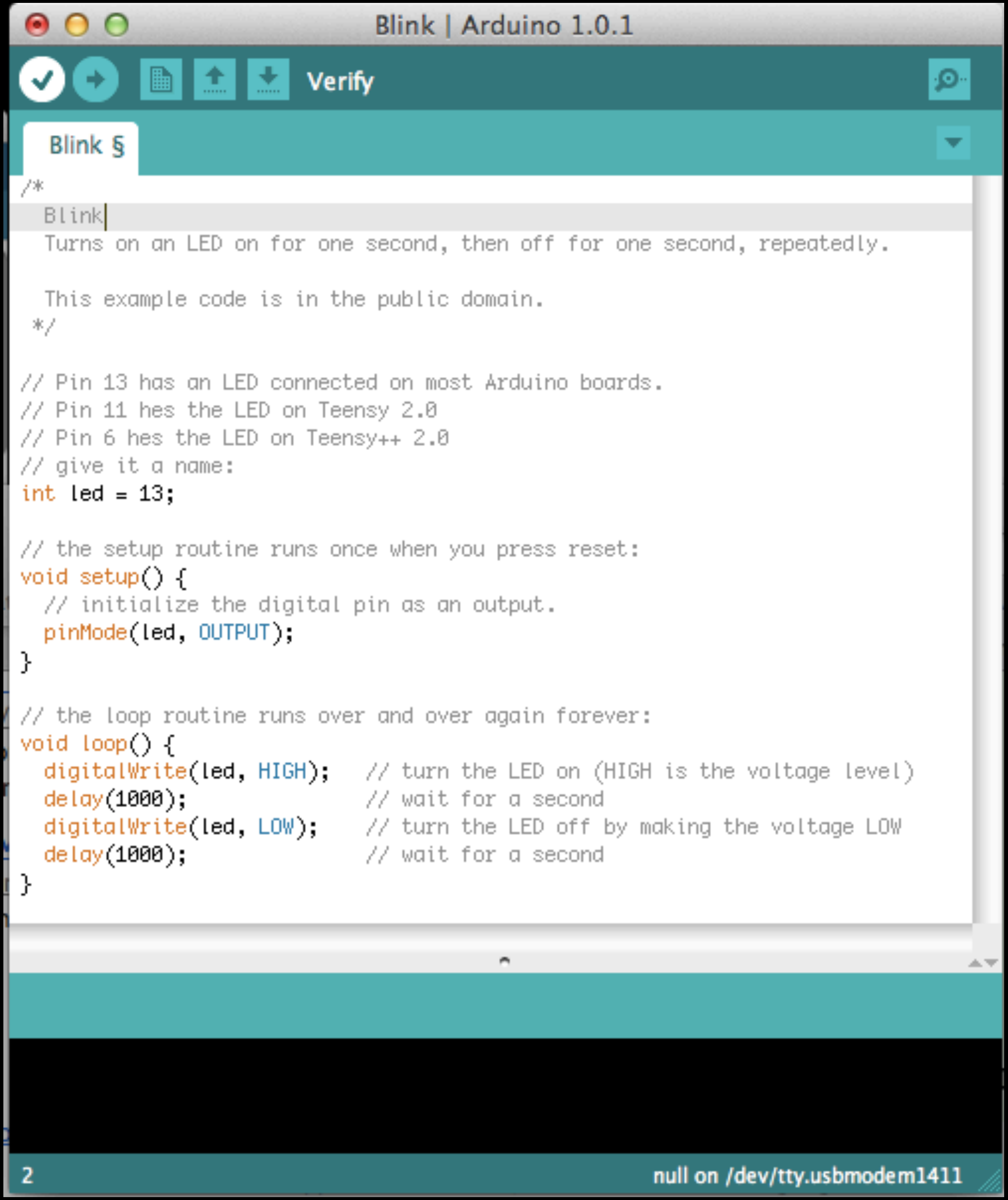
Firmware Programming

Arduino IDE review, questions

Behind the Scenes

Finite State Machines

Modules

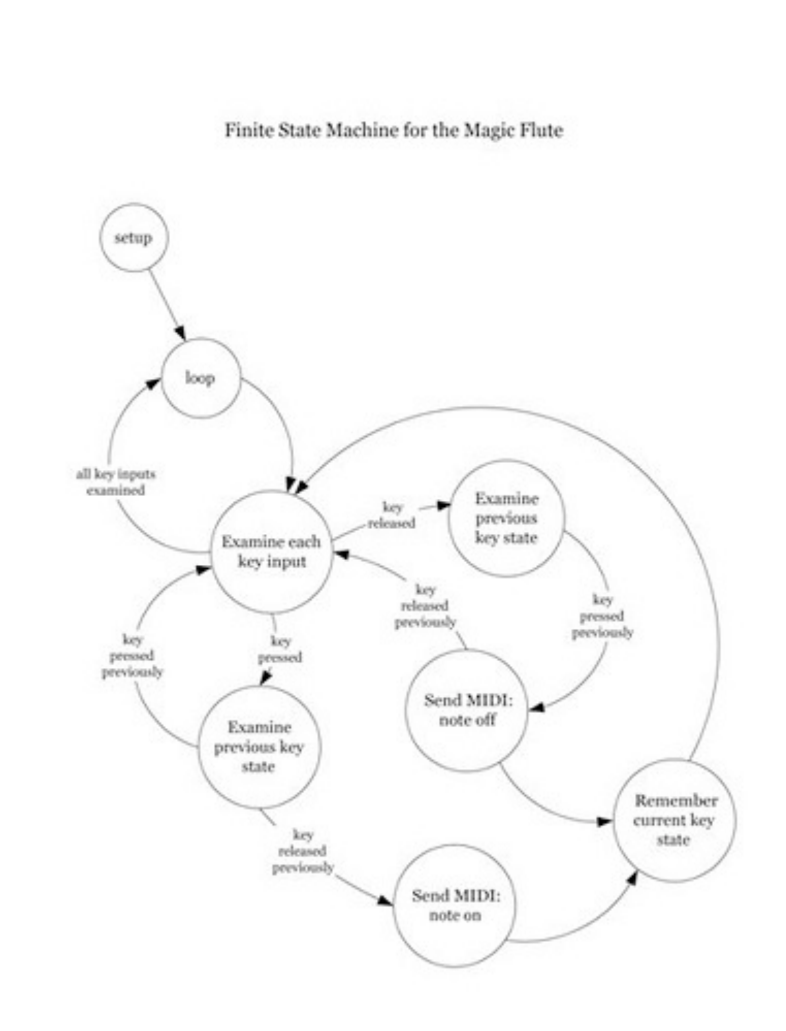


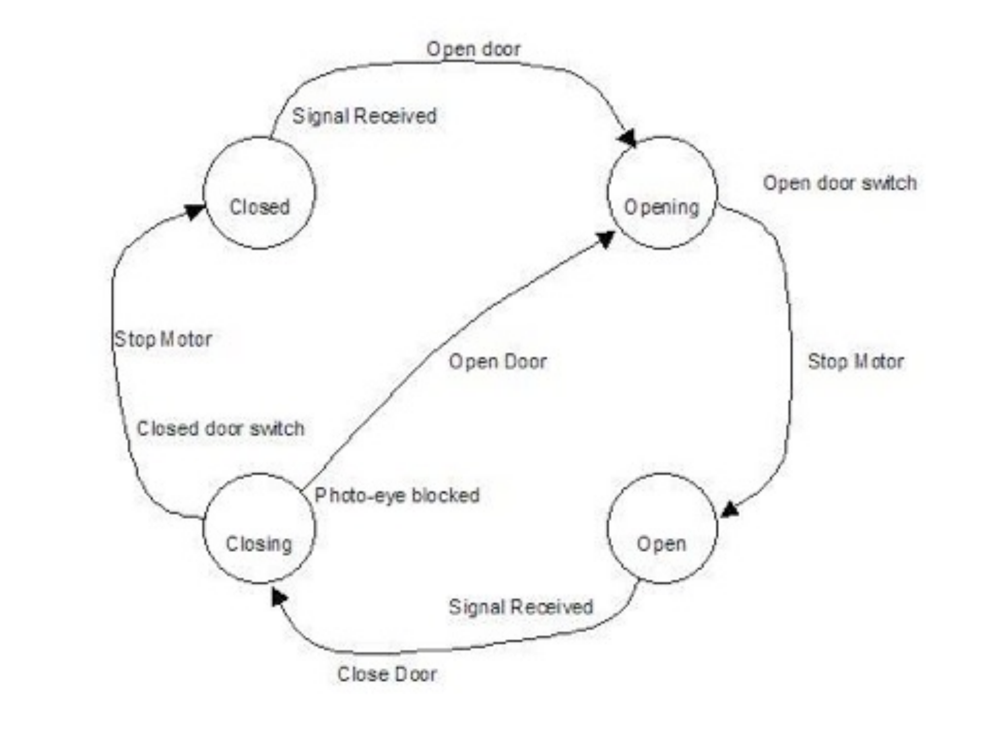
```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// Pin 11 hes the LED on Teensy 2.0
// Pin 6 hes the LED on Teensy++ 2.0
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

```
drwxr-xr-x  21 wendyju  wendyju     714 Apr 11 15:43 .
drwx------  16 wendyju  wendyju     544 Apr 11 15:47 ..
-rw-r--r--   1 wendyju  wendyju     559 Apr 11 15:42 Blink.cpp
-rw-r--r--   1 wendyju  wendyju      13 Apr 11 15:43 Blink.cpp.eep
-rwxr-xr-x   1 wendyju  wendyju   13913 Apr 11 15:43 Blink.cpp.elf
-rw-r--r--   1 wendyju  wendyju    2881 Apr 11 15:43 Blink.cpp.hex
-rw-r--r--   1 wendyju  wendyju    3716 Apr 11 15:42 Blink.cpp.o
-rw-r--r--   1 wendyju  wendyju   17868 Apr 11 15:43 HardwareSerial.cpp.o
-rw-r--r--   1 wendyju  wendyju   31996 Apr 11 15:43 Print.cpp.o
-rw-r--r--   1 wendyju  wendyju   16264 Apr 11 15:43 Tone.cpp.o
-rw-r--r--   1 wendyju  wendyju    5676 Apr 11 15:43 WInterrupts.c.o
-rw-r--r--   1 wendyju  wendyju    7068 Apr 11 15:43 WMath.cpp.o
-rw-r--r--   1 wendyju  wendyju   57548 Apr 11 15:43 WString.cpp.o
-rw-r--r--   1 wendyju  wendyju  184770 Apr 11 15:43 core.a
-rw-r--r--   1 wendyju  wendyju    3168 Apr 11 15:43 main.cpp.o
-rw-r--r--   1 wendyju  wendyju    3288 Apr 11 15:42 pins_arduino.c.o
-rw-r--r--   1 wendyju  wendyju    9392 Apr 11 15:43 wiring.c.o
-rw-r--r--   1 wendyju  wendyju    6776 Apr 11 15:43 wiring_analog.c.o
-rw-r--r--   1 wendyju  wendyju    9256 Apr 11 15:43 wiring_digital.c.o
-rw-r--r--   1 wendyju  wendyju    6812 Apr 11 15:43 wiring_pulse.c.o
-rw-r--r--   1 wendyju  wendyju    5344 Apr 11 15:43 wiring_shift.c.o
```

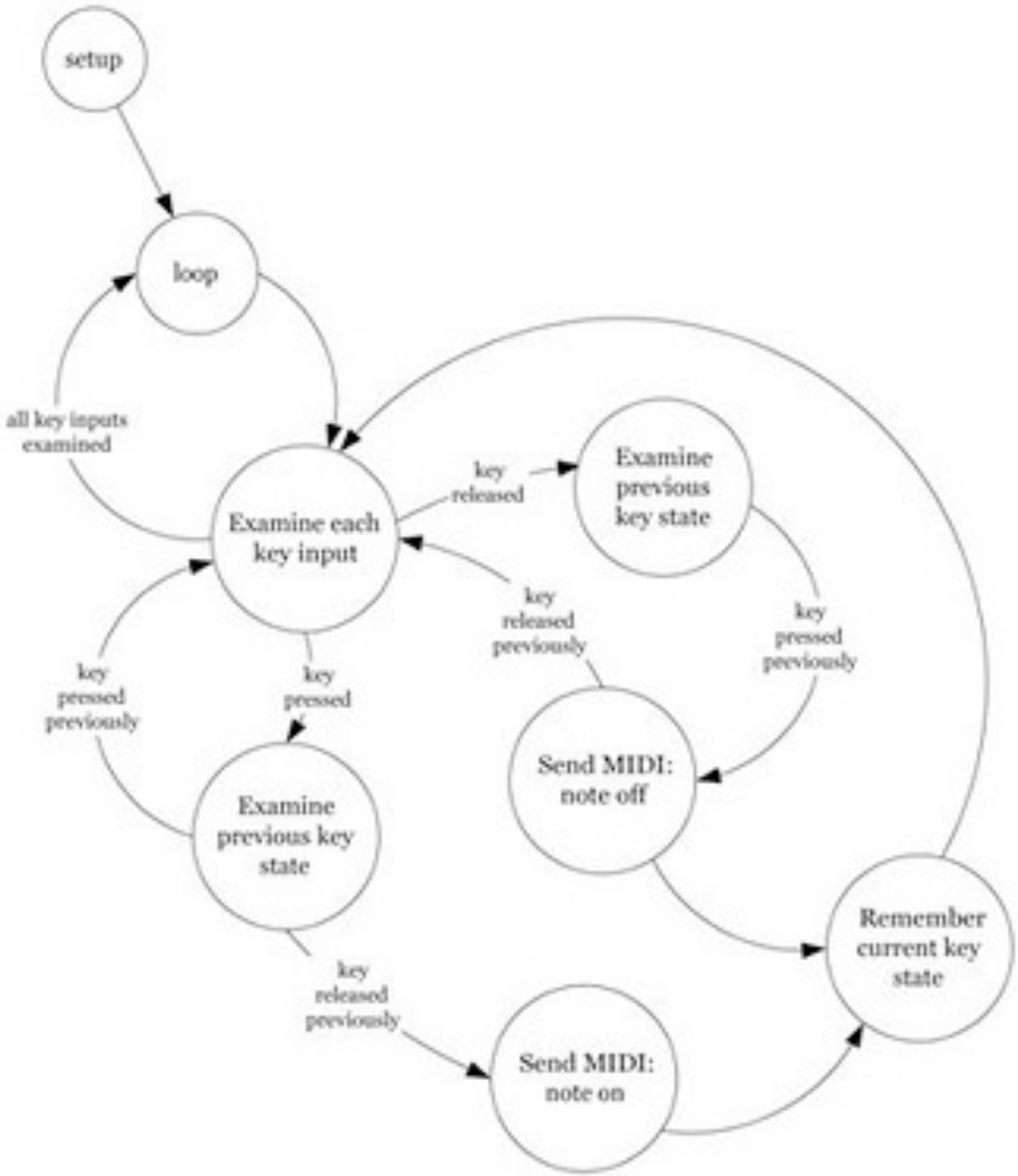
Finite State Machine

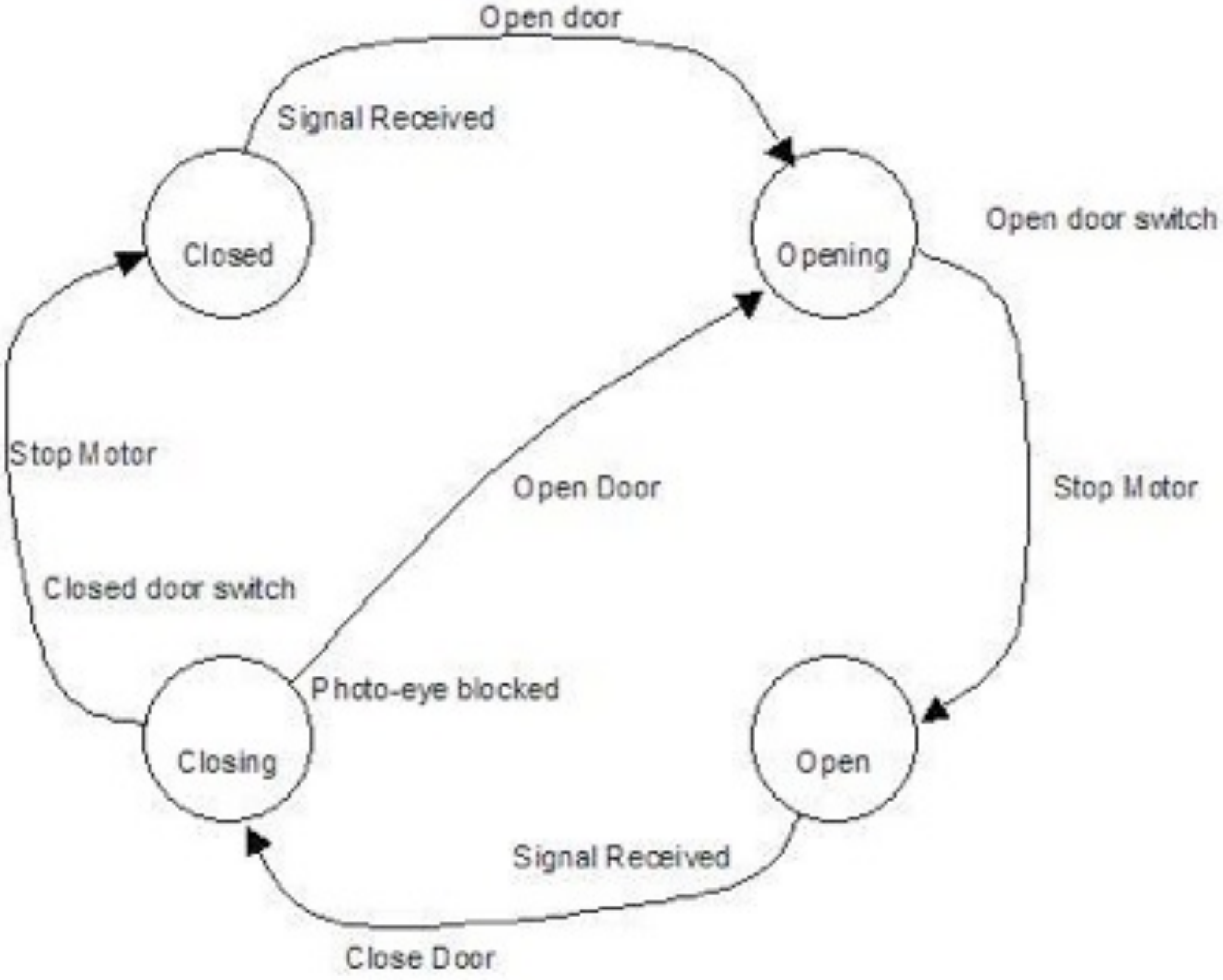
Sketching Interactive Device Behavior





Finite State Machine for the Magic Flute



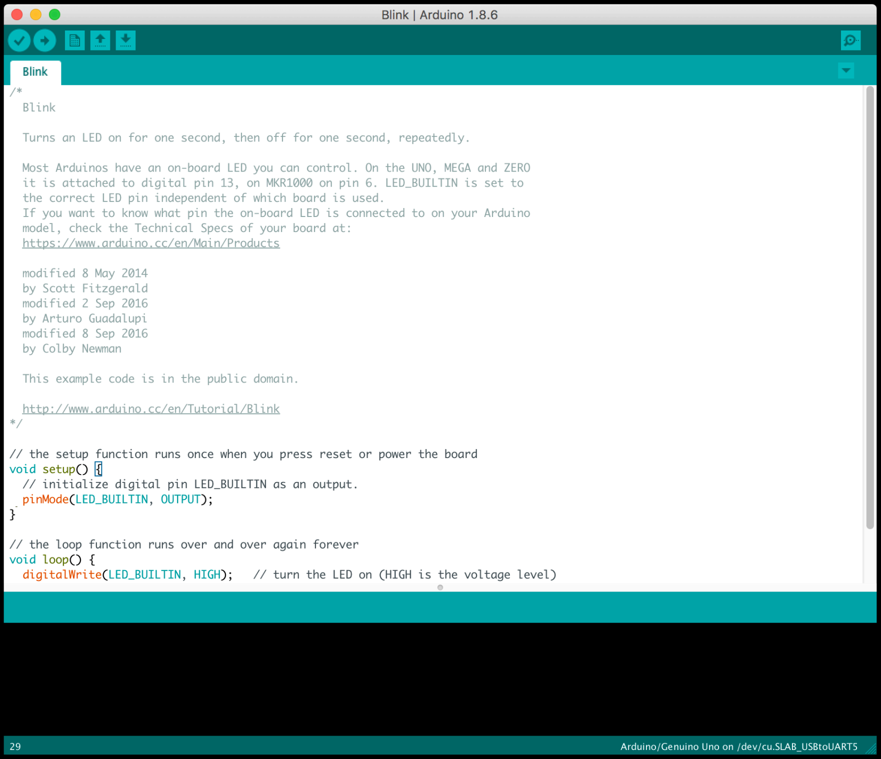

Open door
Signal Received
Closed
Opening
Open door switch
Stop Motor
Open Door
Stop Motor
Closed door switch
Photo-eye blocked
Closing
Open
Signal Received
Close Door

Adapting Found Code

How do we merge two programs?

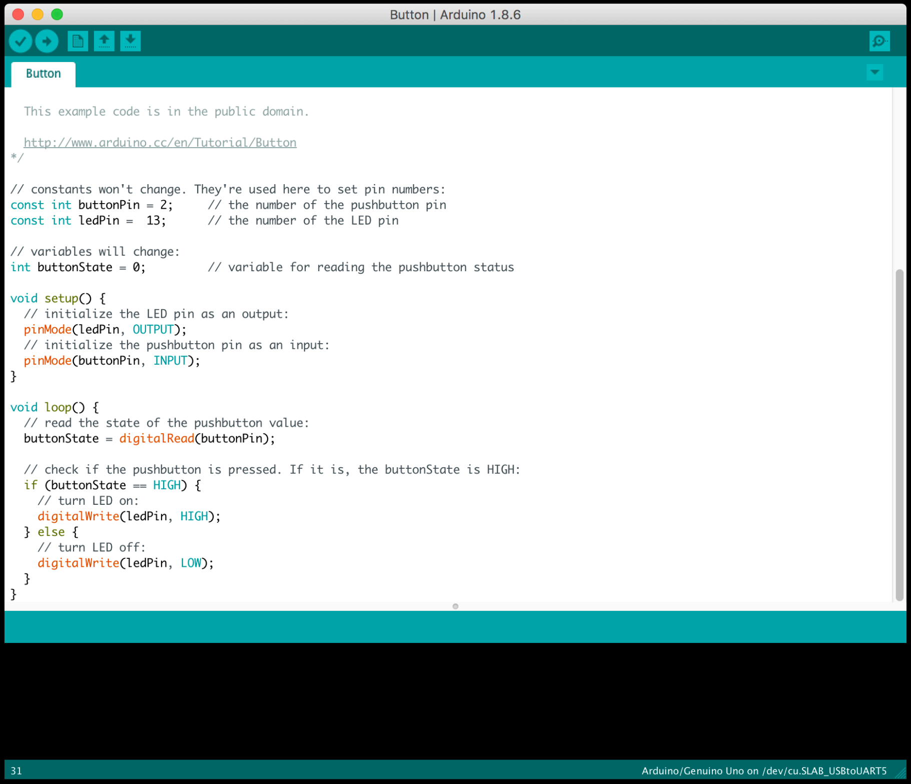
ADAPTING FOUND CODE

How do we merge two programs?



```
/*
  Blink

  Turns an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO
  it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to
  the correct LED pin independent of which board is used.
  If you want to know what pin the on-board LED is connected to on your Arduino
  model, check the Technical Specs of your board at:
  https://www.arduino.cc/en/Main/Products

  modified 8 May 2014
  by Scott Fitzgerald
  modified 2 Sep 2016
  by Arturo Guadalupi
  modified 8 Sep 2016
  by Colby Newman

  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/Blink
*/

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
```

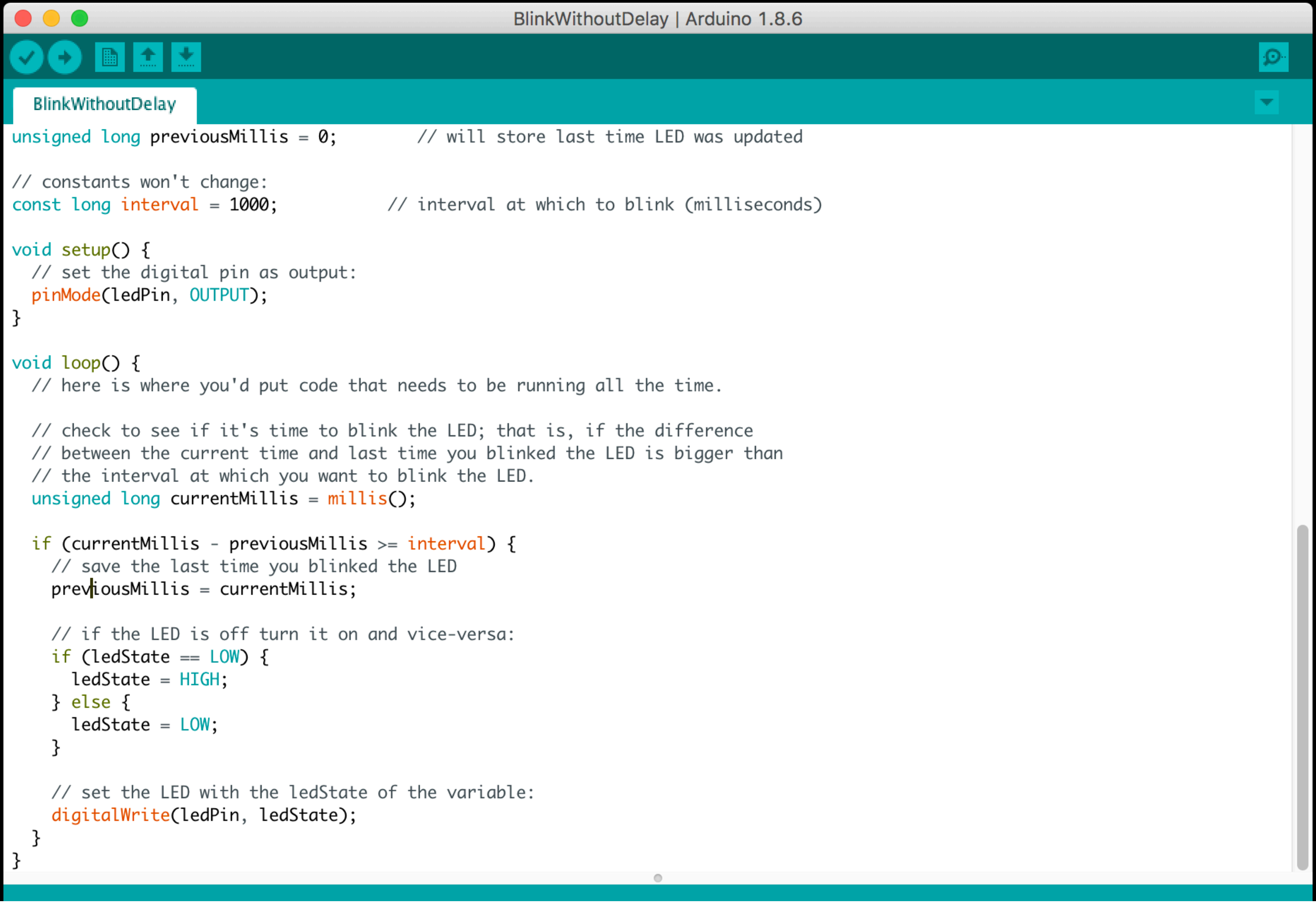
+



```
  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/Button
*/

// constants won't change. They're used here to set pin numbers:
const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin =  13;      // the number of the LED pin

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed. If it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  } else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

Non-blocking Code

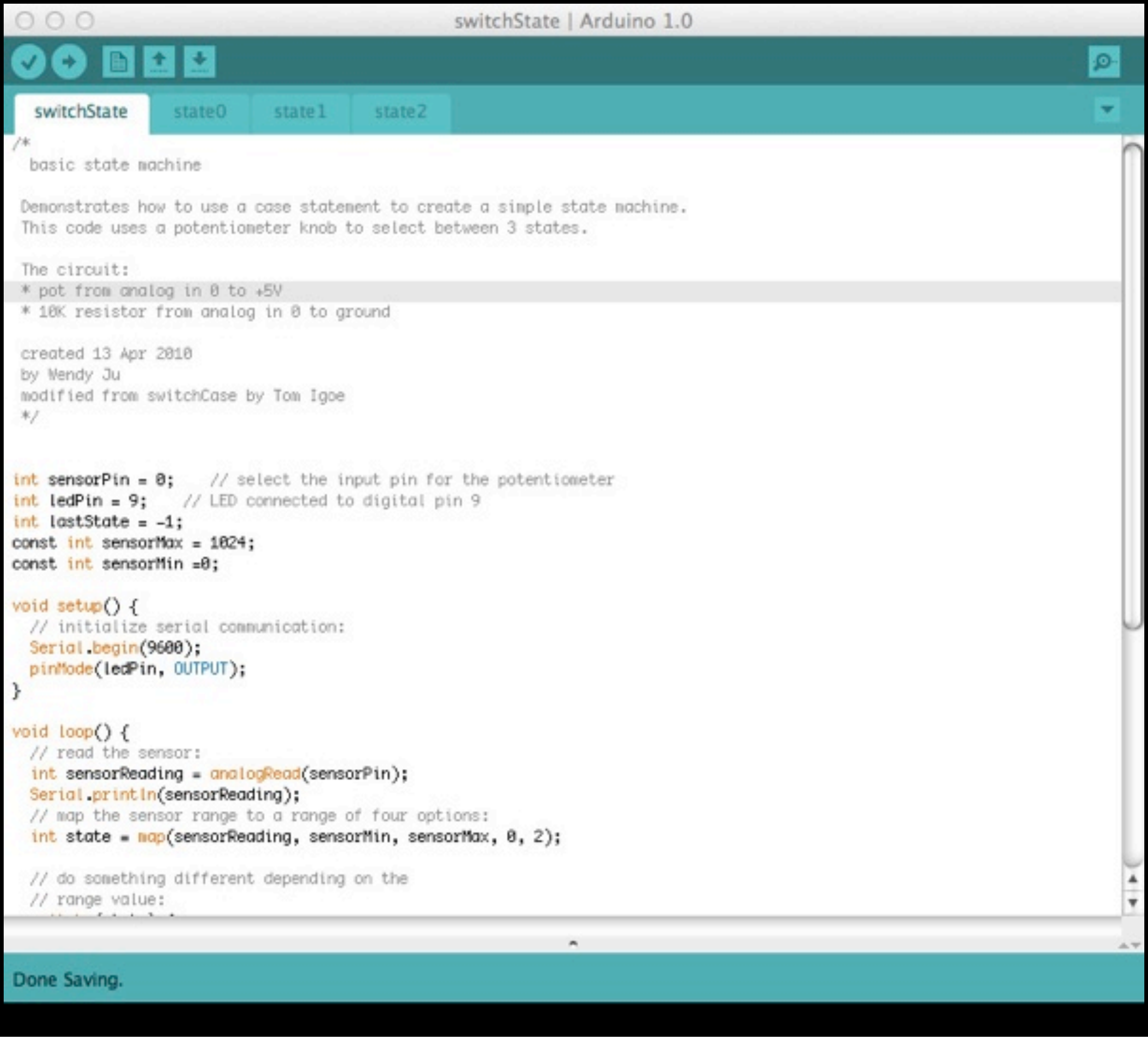
How do we write code that doesn't block execution?



```
unsigned long previousMillis = 0;        // will store last time LED was updated

// constants won't change:
const long interval = 1000;           // interval at which to blink (milliseconds)

void setup() {
  // set the digital pin as output:
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // here is where you'd put code that needs to be running all the time.

  // check to see if it's time to blink the LED; that is, if the difference
  // between the current time and last time you blinked the LED is bigger than
  // the interval at which you want to blink the LED.
  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= interval) {
    // save the last time you blinked the LED
    previousMillis = currentMillis;

    // if the LED is off turn it on and vice-versa:
    if (ledState == LOW) {
      ledState = HIGH;
    } else {
      ledState = LOW;
    }

    // set the LED with the ledState of the variable:
    digitalWrite(ledPin, ledState);
  }
}
```

Modules & Modes

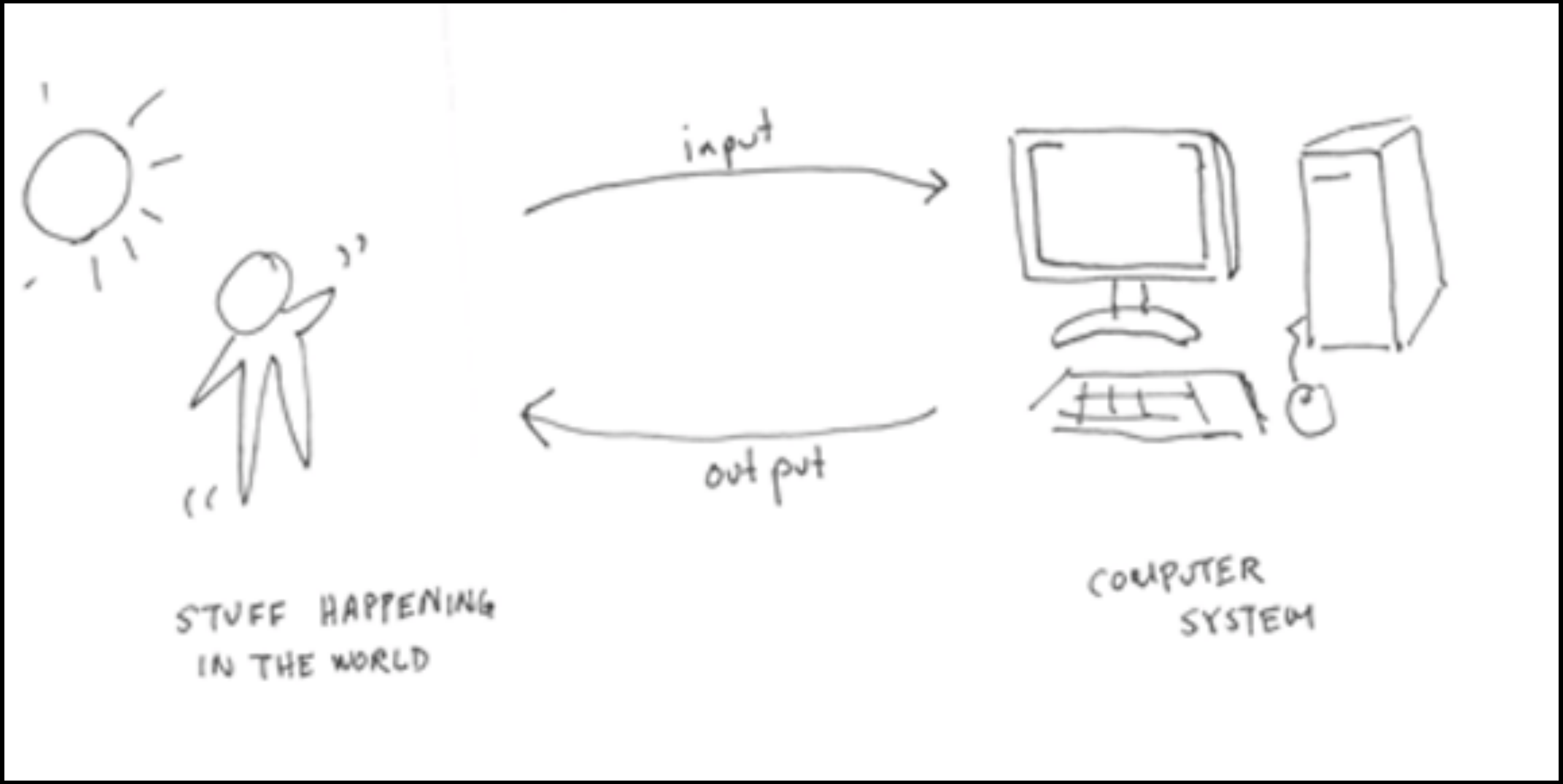
Organizing functions and Behaviors



```
/*
  basic state machine

 Demonstrates how to use a case statement to create a simple state machine.
 This code uses a potentiometer knob to select between 3 states.

 The circuit:
 * pot from analog in 0 to +5V
 * 10K resistor from analog in 0 to ground

 created 13 Apr 2010
 by Wendy Ju
 modified from switchCase by Tom Igoe
 */


int sensorPin = 0;    // select the input pin for the potentiometer
int ledPin = 9;    // LED connected to digital pin 9
int lastState = -1;
const int sensorMax = 1024;
const int sensorMin =0;

void setup() {
  // initialize serial communication:
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // read the sensor:
  int sensorReading = analogRead(sensorPin);
  Serial.println(sensorReading);
  // map the sensor range to a range of four options:
  int state = map(sensorReading, sensorMin, sensorMax, 0, 2);

  // do something different depending on the
  // range value:
```

Done Saving.

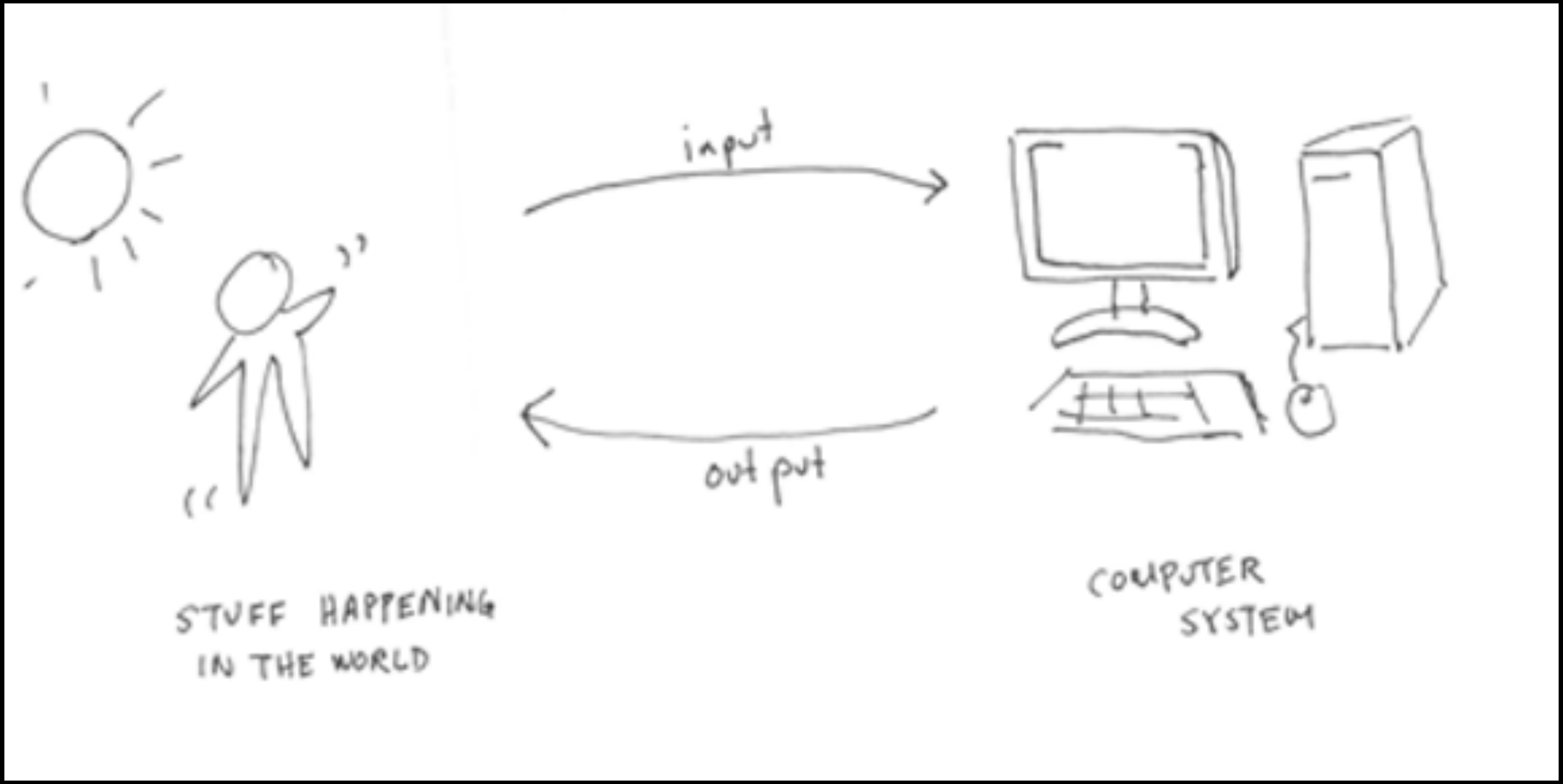
This week's preLab



How do we sketch ideas for interactions?

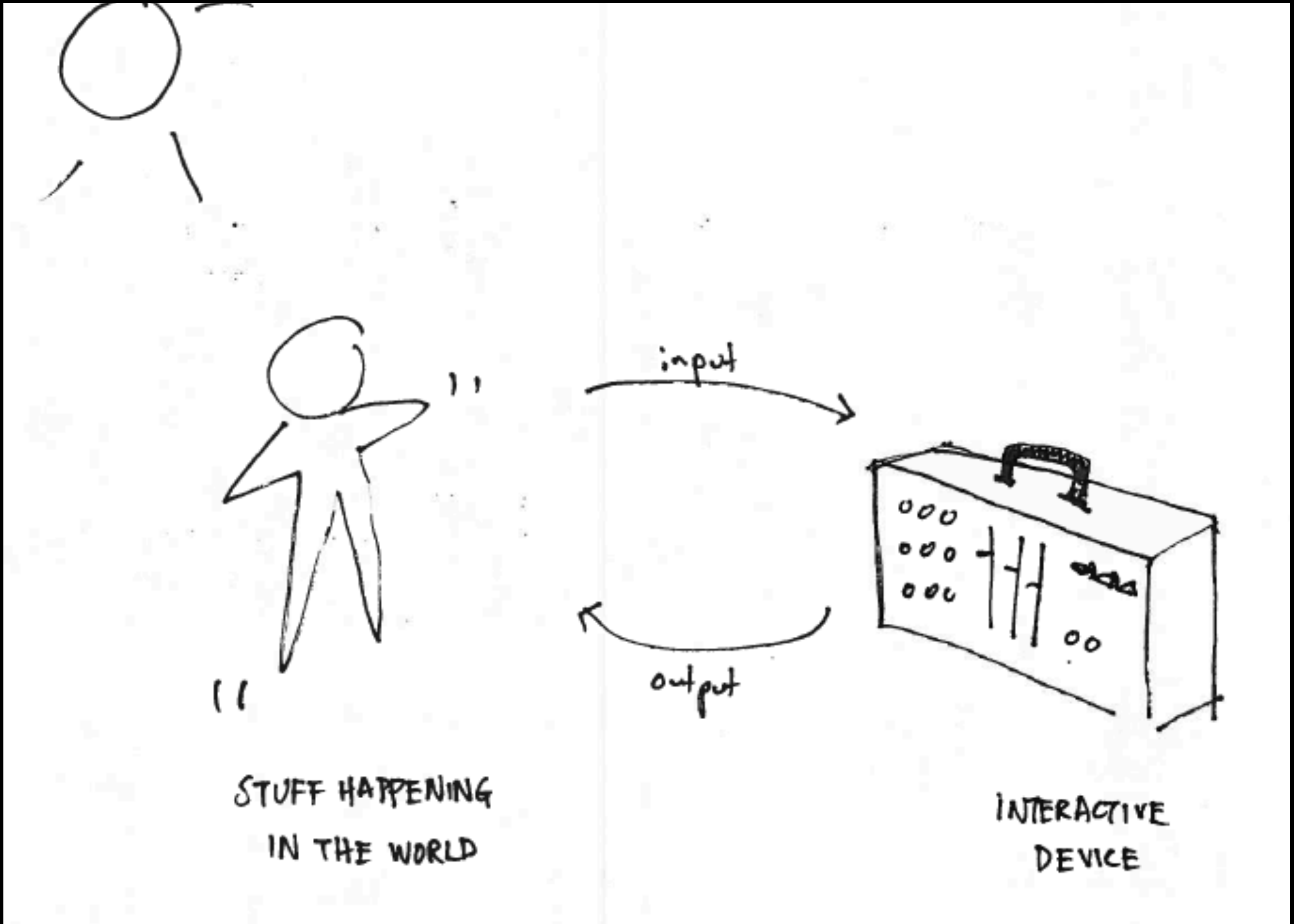
Interacting with Interactive Devices

some sketches



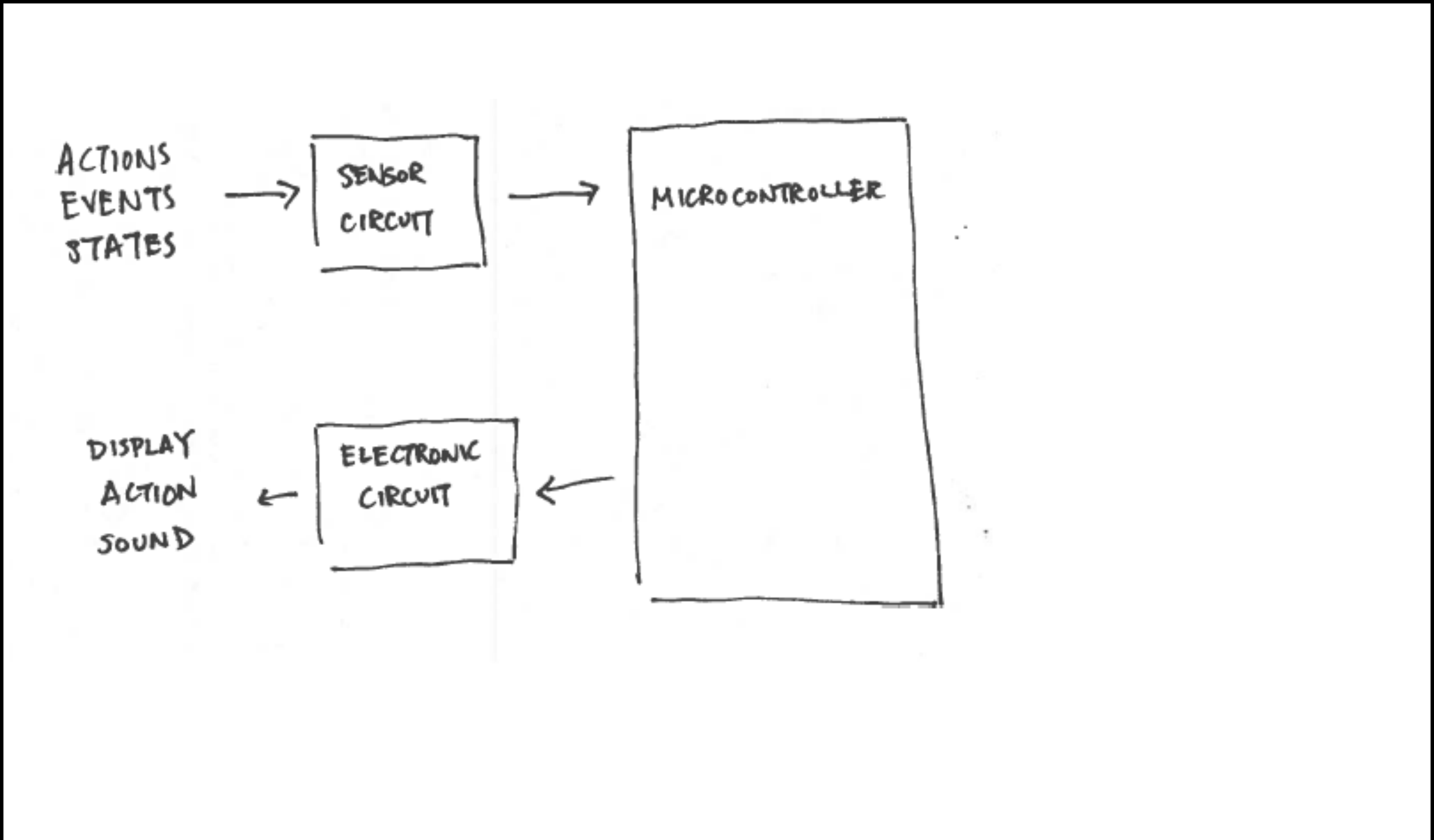
Interacting with Interactive Devices

some sketches



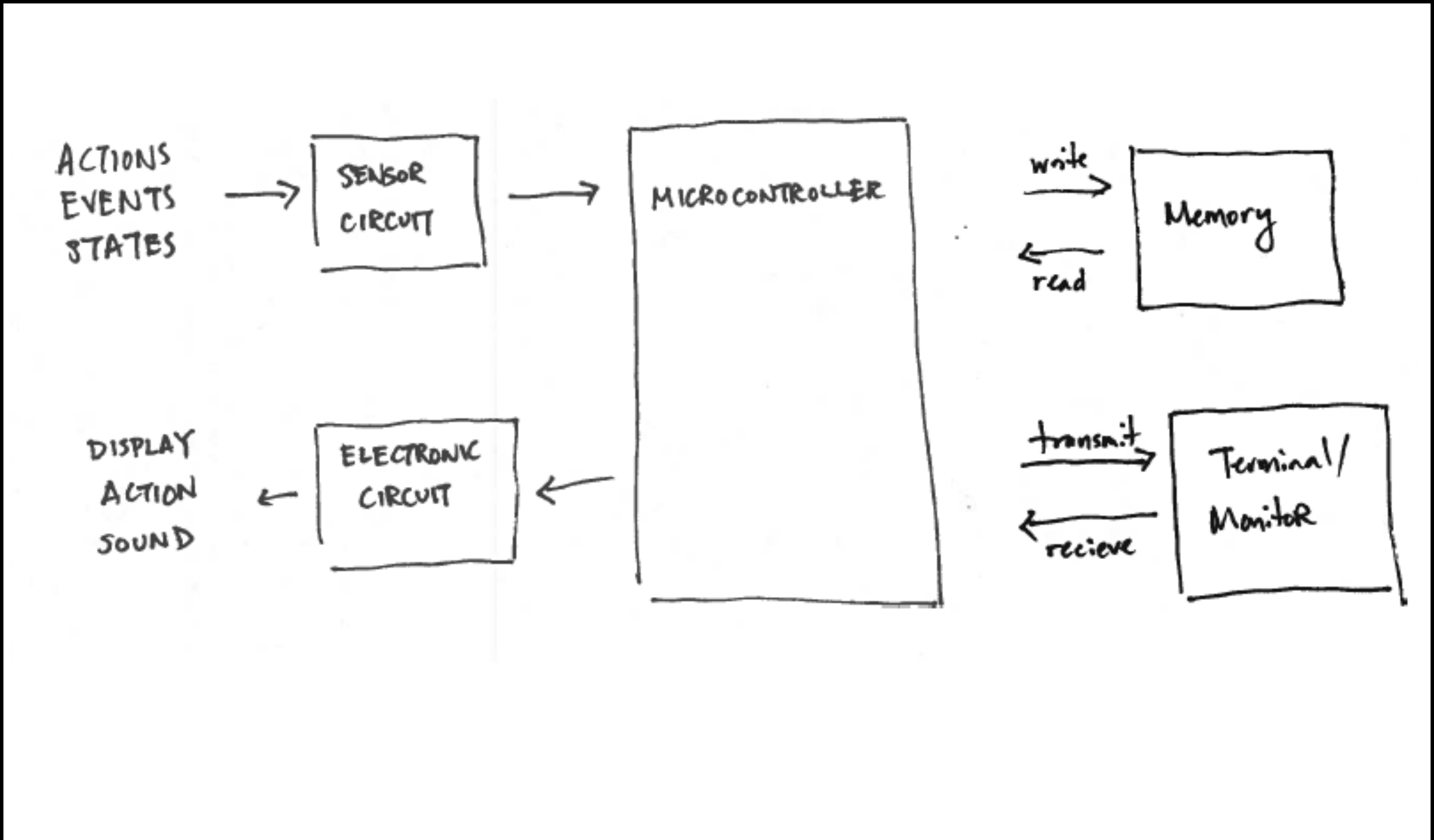
Interacting with Interactive Devices

some sketches



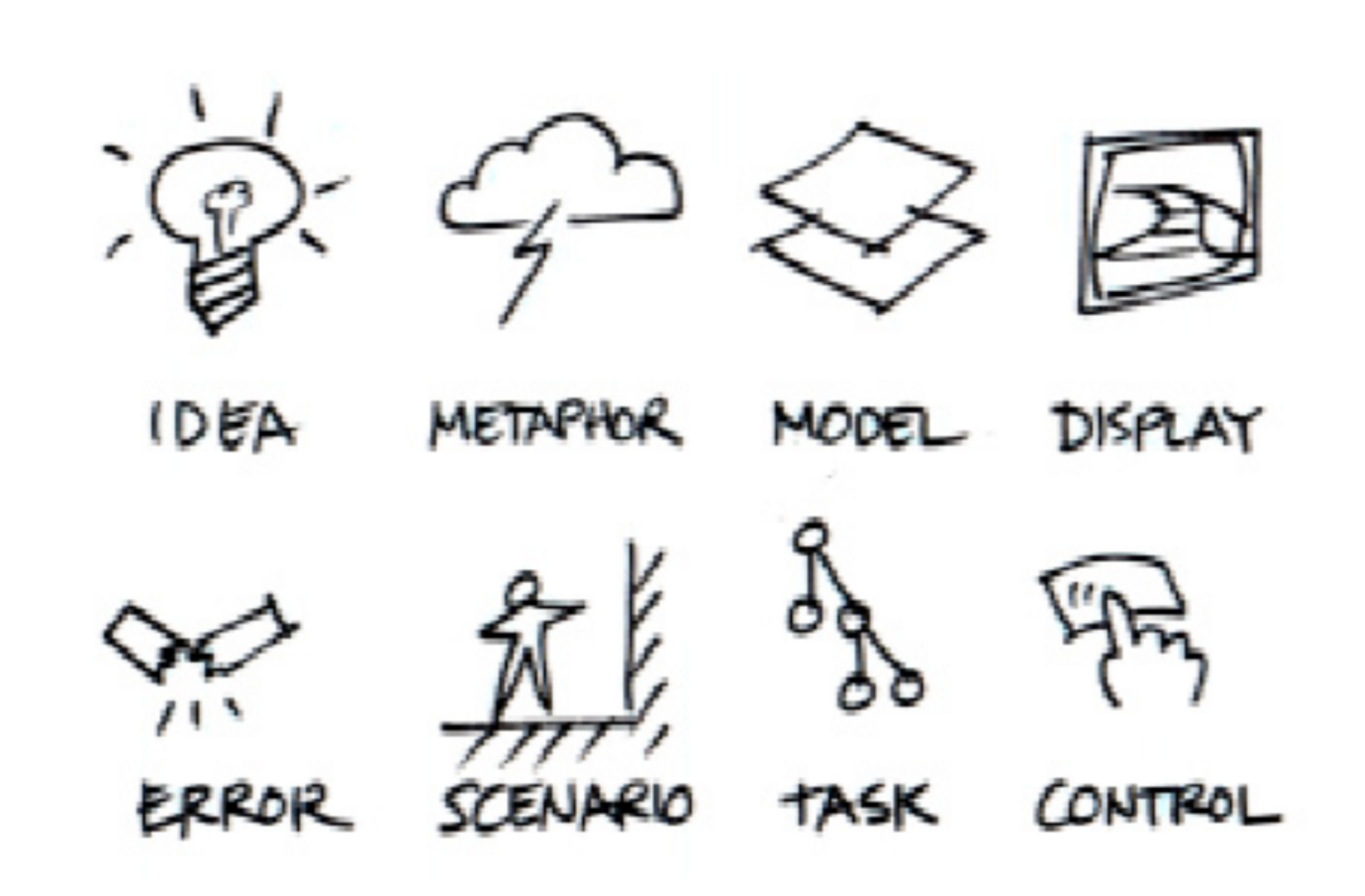
Interacting with Interactive Devices

some sketches



Interacting with Interactive Devices

some sketches

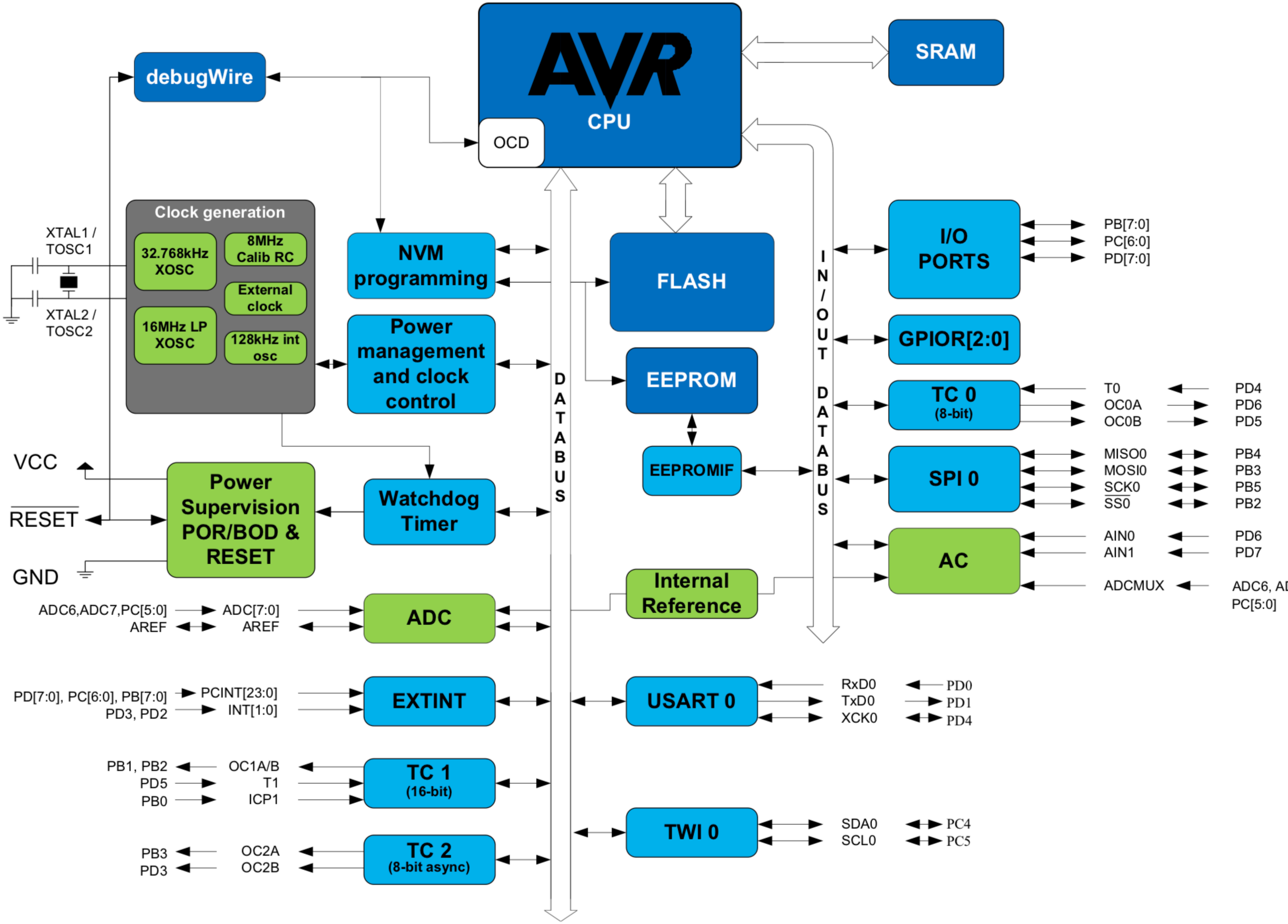


More than you need to know about Microcontrollers

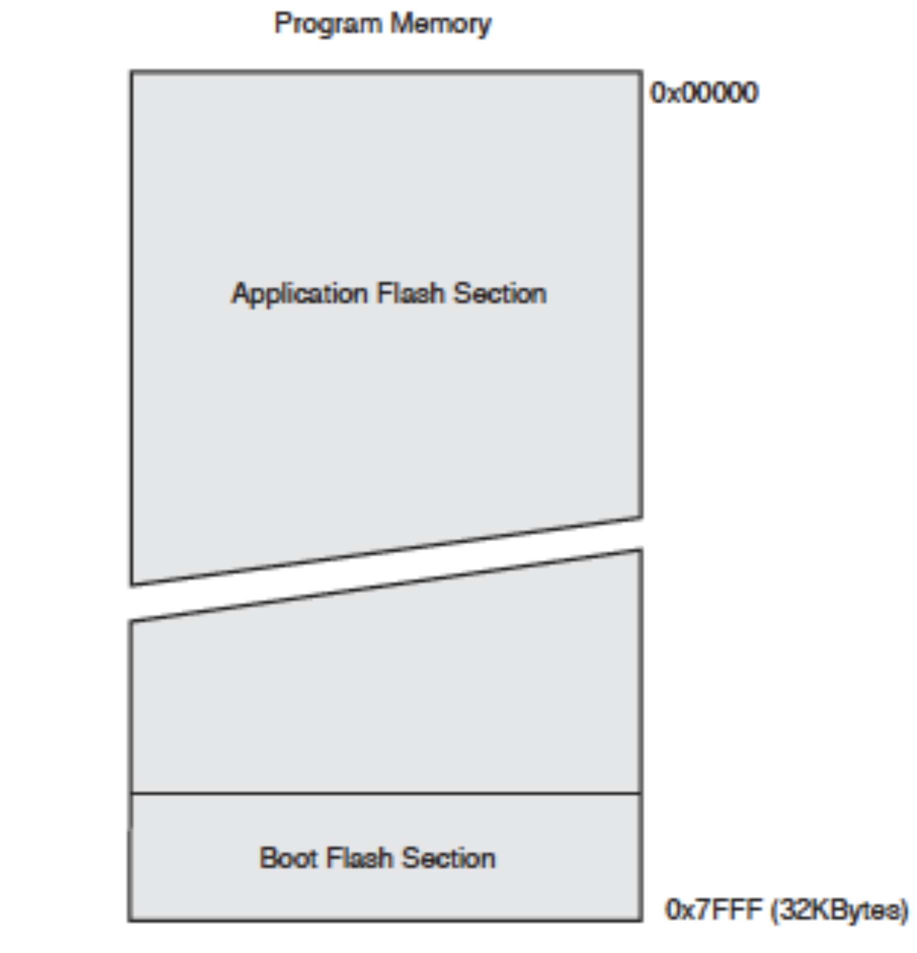
Clock | Program Memory | Data Memory | Registers | Code

4. Block Diagram

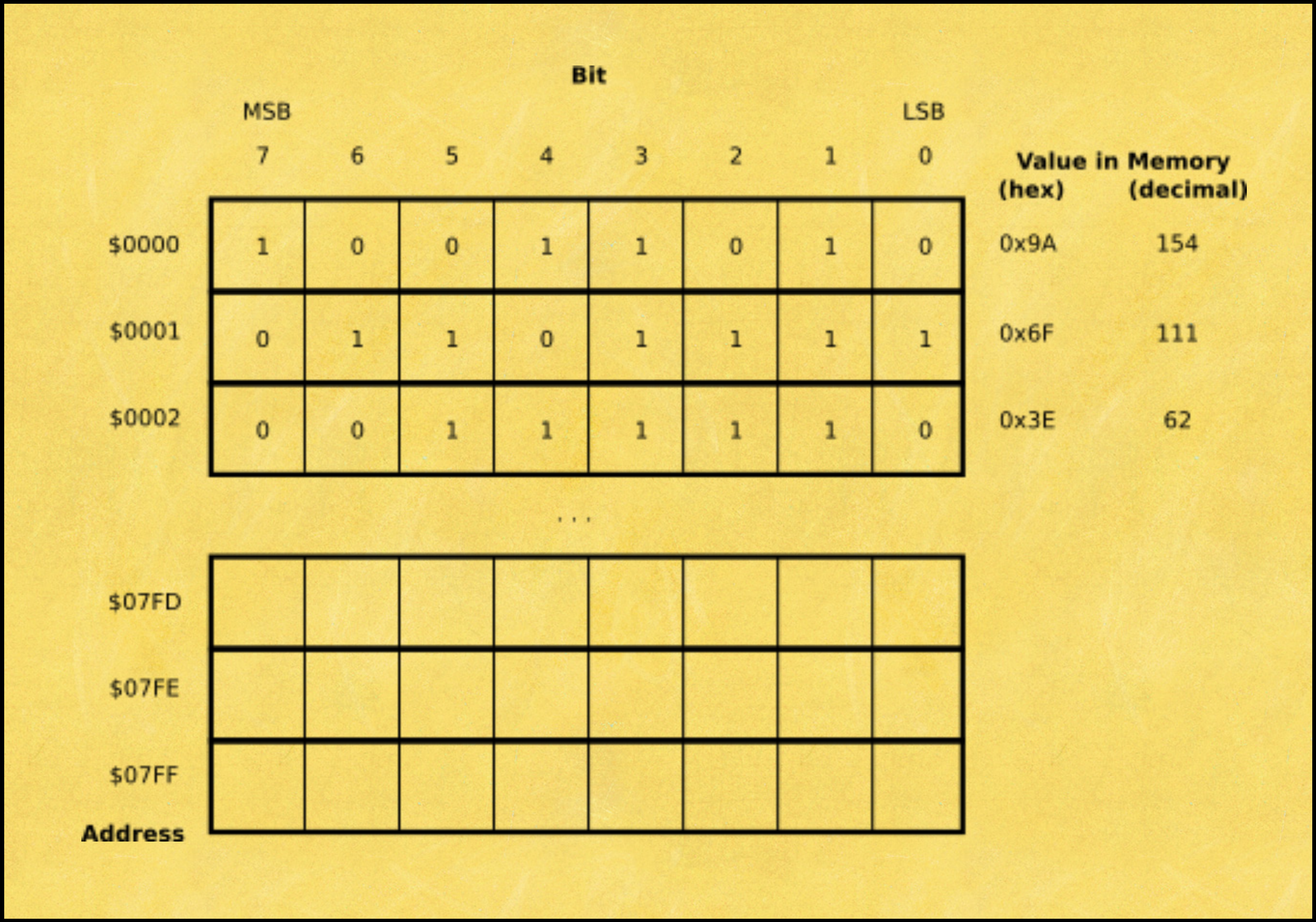
Figure 4-1. Block Diagram



Program Memory



Data Memory



	Bit									
	MSB							LSB	Value in Memory	
Address	7	6	5	4	3	2	1	0	(hex)	(decimal)
$0000	1	0	0	1	1	0	1	0	0x9A	154
$0001	0	1	1	0	1	1	1	1	0x6F	111
$0002	0	0	1	1	1	1	1	0	0x3E	62
				. . .						
$07FD										
$07FE										
$07FF										

Bits and Bytes:

- 1 byte = 8 bits, 256 unique values for each byte
- All the information in the microcontroller is stored in byte-size chunks; we represent each byte of information as a two-digit hexadecimal number.
- 11110011 in binary = 243 in decimal = F3 in hexadecimal
- b11110011 = 0xF3
- Memory addresses are hex, as well, but preceded with $, e.g. $03DF.

IO Registers:

PORT B: (PB7-PB0) 8-bit bi-directional IO

PORT C: (PC 7, 6) 8-bit bi-directional IO

PORT D: (PD7-0) 8-bit bi-directional IO

PORT F: (PF7-4, PF1, PF0): analog inputs to A/D converter (can be used at 8-bit bi-directional IO)

Data Direction Registers (DDR):

Since the IO pins are configurable to be either input or output, the controller needs some place to store the directionality of each bit.

These are stored in the Data Direction Registers. Like all the other registers, the DDRs have 1's and 0's, but its 1's and 0's indicate whether the corresponding port pin is an input (0) or output (1).

Port Features:

Analog to Digital Conversion

Pulse Width Modulation

Timers & Counters

External Interrupts

Serial Peripheral Interface

RX/TX

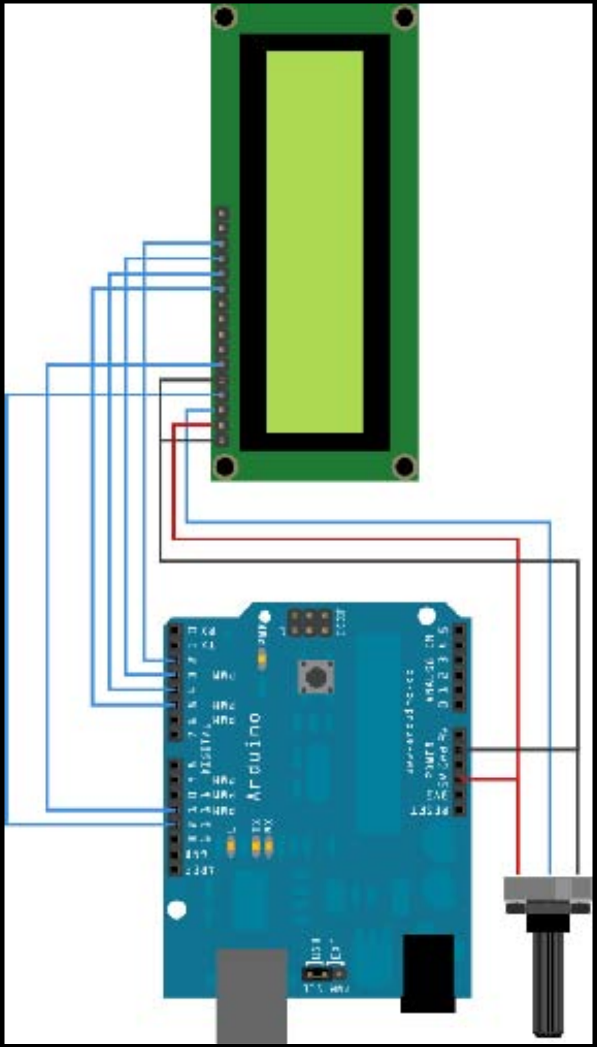
Types of Interface

Parallel

Examples
Graphical LCD
SCSI, Firewire

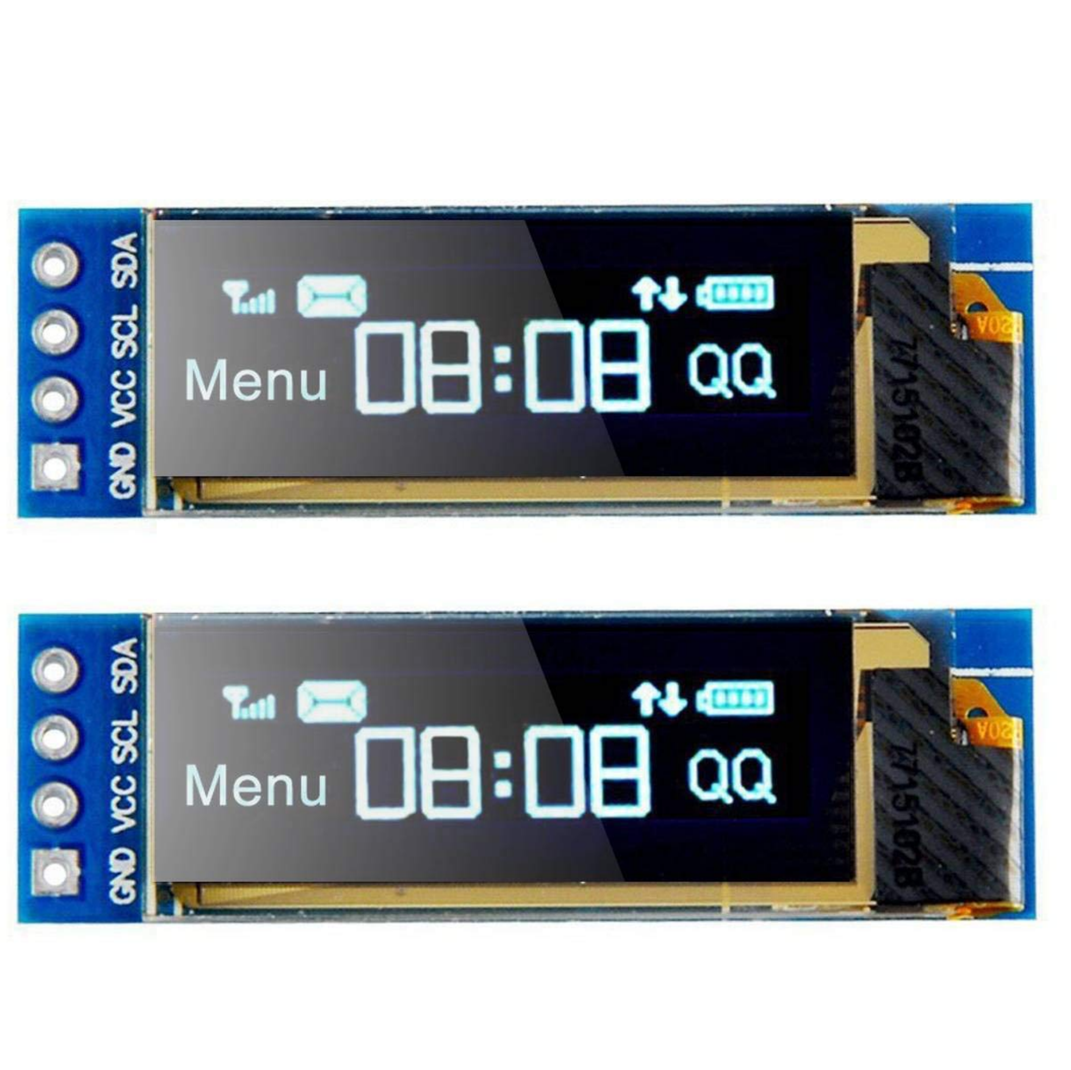
Advantages
Faster in Theory

Drawbacks
Crosstalk
Clock Skew
Wire per Bit



Types of Interface

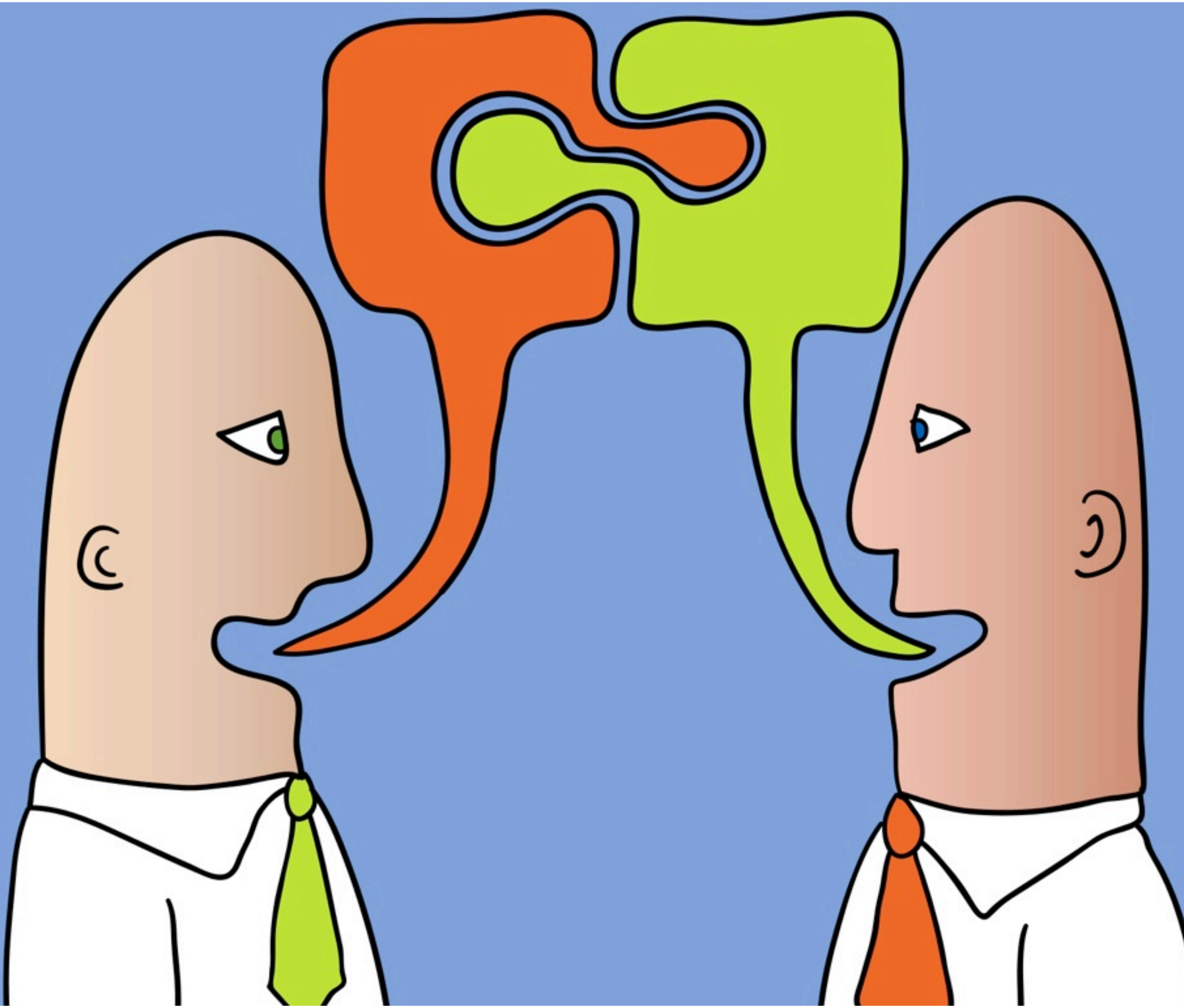
Serial



Examples
USB
SPI and I^{2}C

Advantages
Clock Faster
Fewer Wires

Drawbacks
Overhead of Negotiation



Communication Bus

Chained serial communication

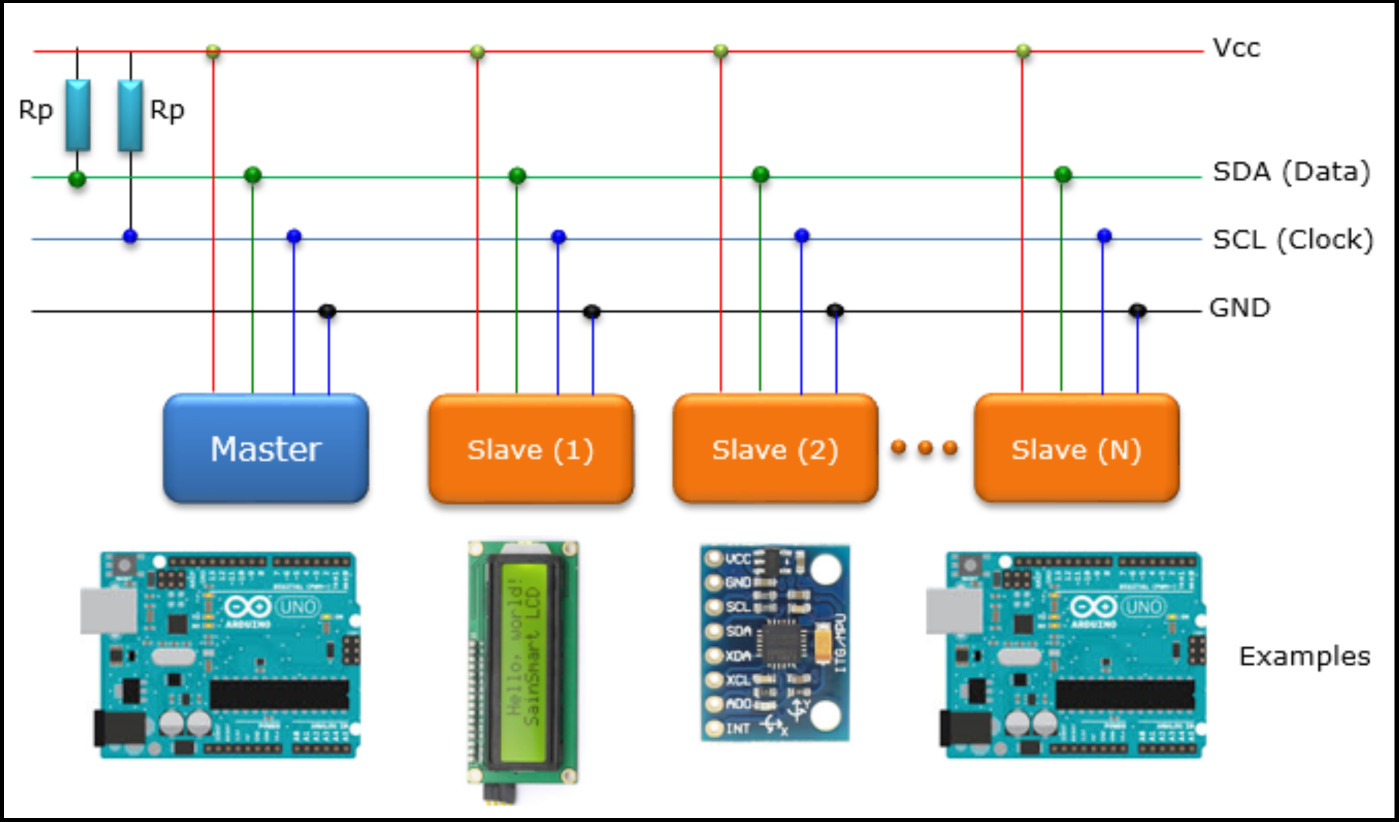


Image from http://www.mbeddedc.com/2017/05/i2c-bus-communication-protocol-tutorial.html

Context of Communication

Conversation - Rules of Conduct

Communication is holding a conversation

Interprocessor communication is peer-to-peer

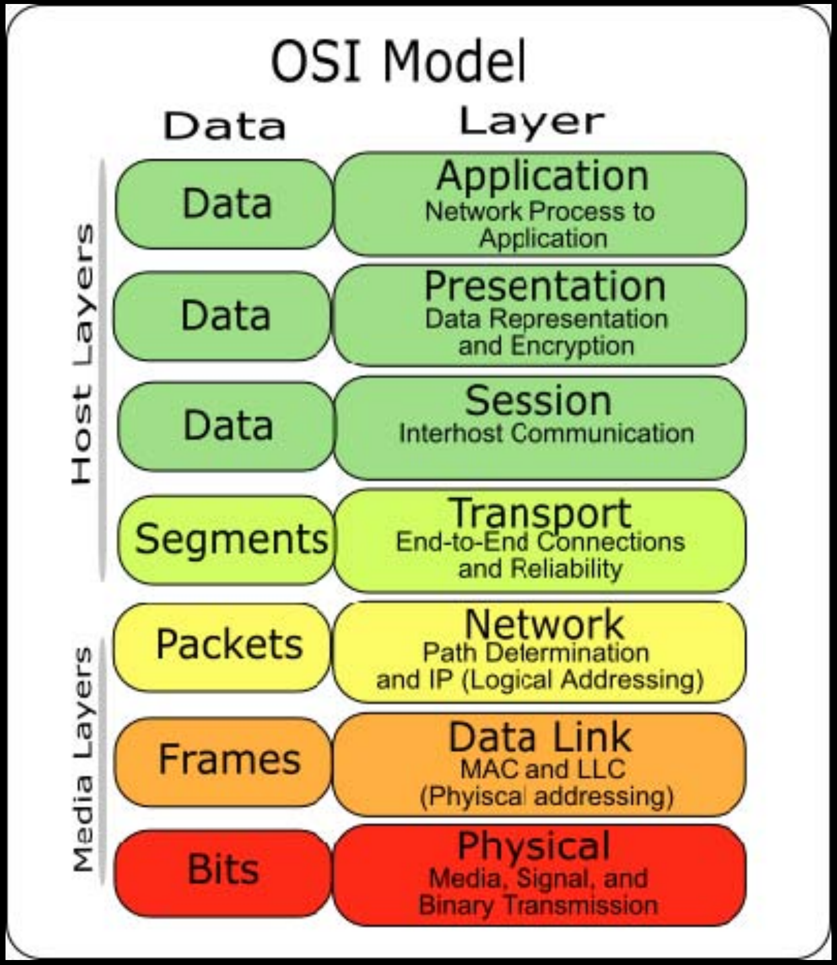
Processor to device conversation is master-slave

A protocol is a set of rules of conduct that we agree to uphold during the conversation

They govern how we start a conversation, who speaks when, how fast, how often, etc.

Context of Communication

Open Systems Interconnection



OSI Model

	Data	Layer
Host Layers	Data	Application Network Process to Application
	Data	Presentation Data Representation and Encryption
	Data	Session Interhost Communication
	Segments	Transport End-to-End Connections and Reliability
Media Layers	Packets	Network Path Determination and IP (Logical Addressing)
	Frames	Data Link MAC and LLC (Phyiscal addressing)
	Bits	Physical Media, Signal, and Binary Transmission

Serial, WiFi, Ethernet

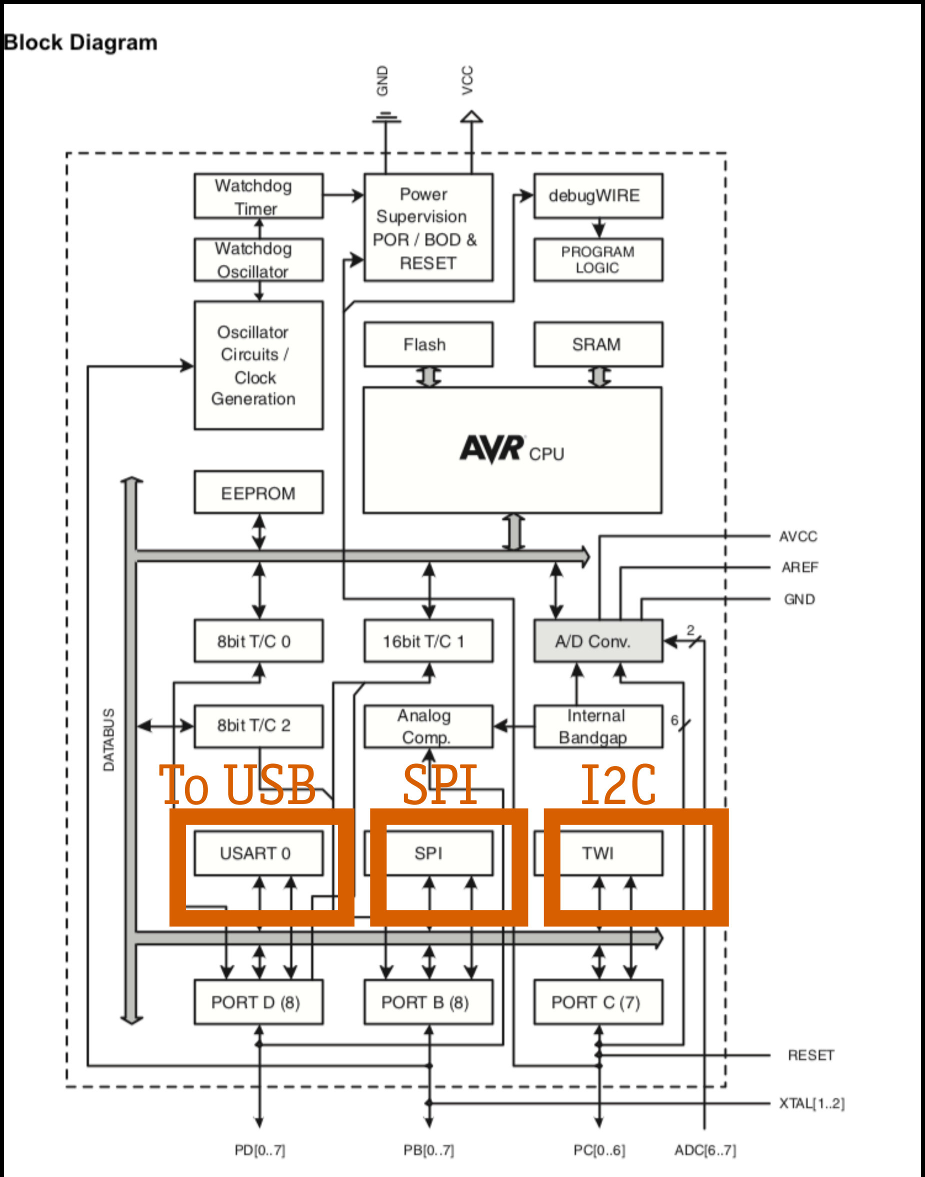
Voltages, Wires, Timing

How does a MCU communicate?

Parallel vs. Serial

Microprocessor
Communication

Atmega 328 supports:
Digital and Analog I/O
Master/Slave SPI interface
2 wire serial interface bus (I2C)
Programmable Serial USART (Universal Synchronous/ Asynchronous Receiver/ Transmitter)


Block Diagram

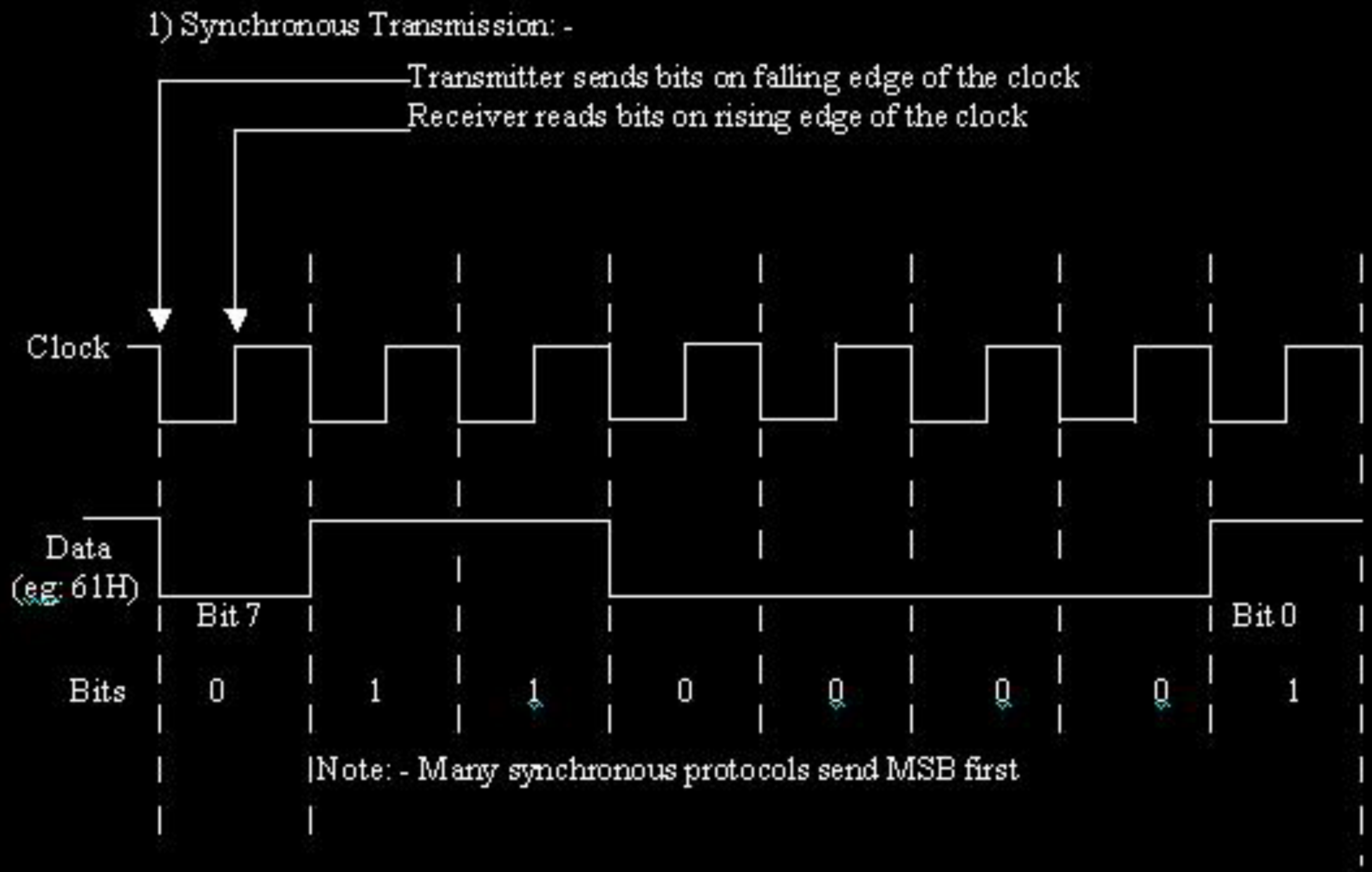
How does a MCU communicate?

Bits & Bytes

1 Bit
  - 0 = LOW
  - 1 = HIGH
- 2 Bits
  - 00 = 0
  - 01 = 1
  - 10 = 2
  - 11 = 3
- 3 Bits
  - 000 = 0
  - 001 = 1
  - 010 = 2
  - 011 = 3
  - 100 = 4
  - 101 = 5
  - 110 = 6
  - 111 = 7

Serial Communication

Synchronous



1) Synchronous Transmission: -

Transmitter sends bits on falling edge of the clock

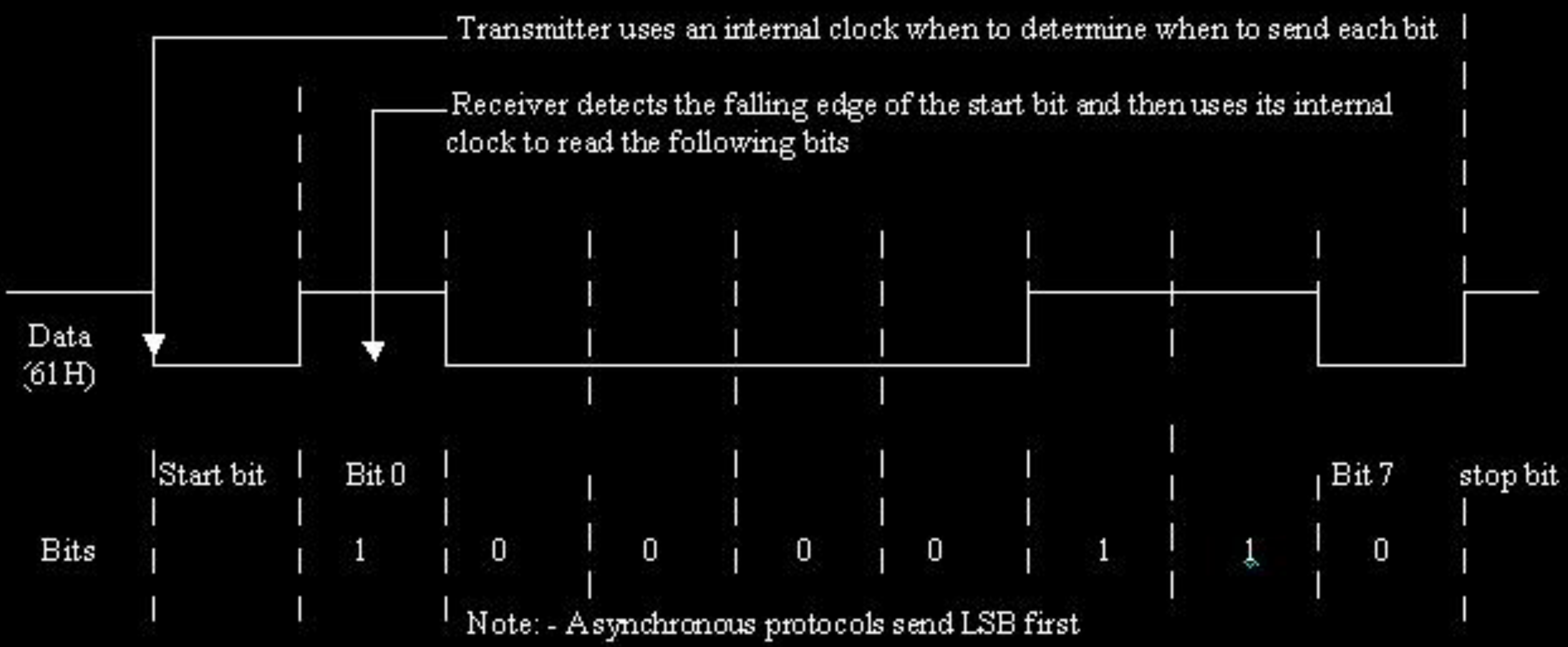
Receiver reads bits on rising edge of the clock

Note: - Many synchronous protocols send MSB first

Serial Communication

Asynchronous

2) Asynchronous Transmission: -



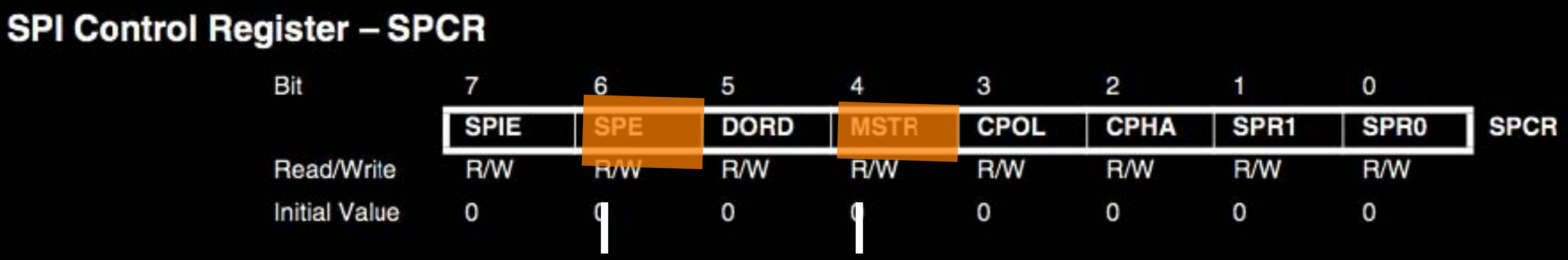
Transmitter uses an internal clock when to determine when to send each bit

Receiver detects the falling edge of the start bit and then uses its internal clock to read the following bits

Note: - Asynchronous protocols send LSB first

Serial Peripheral Interface

Configuration



SPI Control Register – SPCR

Bit	7	6	5	4	3	2	1	0	
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0 1	0	0 1	0	0	0	0	

How do we configure the microcontroller (uC) for SPI?

How do we use SPI to communicate?

Set SPCR = 0101 0000SPE – “Enable SPI mode”MSTR – “I control the clock”