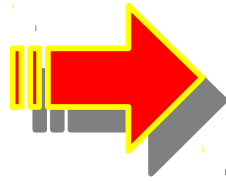


Lab. 10/A

Table of Contents



Step 0 - Brief introduction to μC

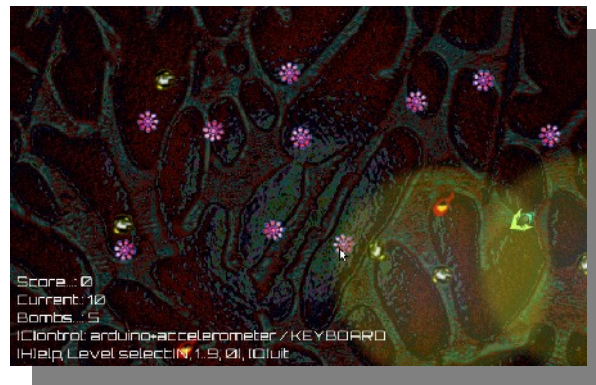


Step 1 - Safe-lock Finite State Machine (FSM)



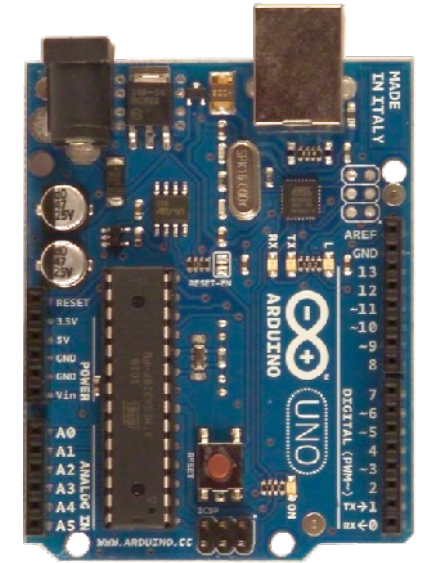
Step 2 - Coin acceptor FSM

Step 3 – Homework



Brief Introduction to Microcontrollers (μC) and Arduino I/O board as a simplified DAQ hardware

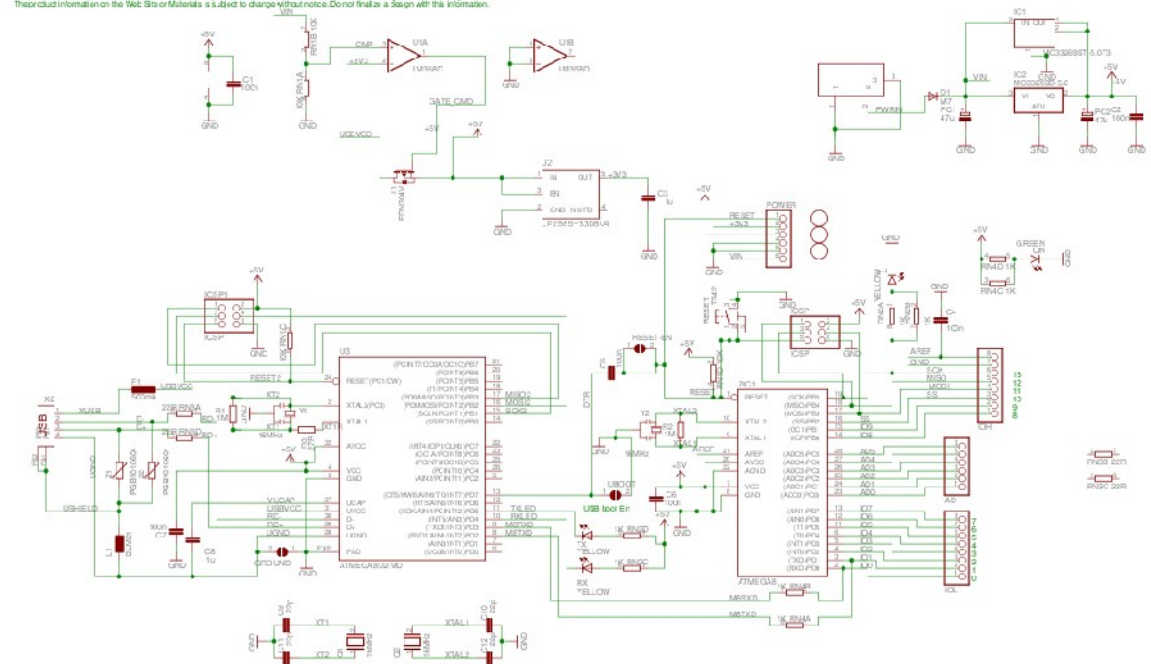
- The Arduino I/O Board accepts **inputs from real-world sensors** such as **switches, potentiometers, light, or sound** sensors, **accelerometers, IR** devices, etc.
- And similarly it can **set values on its pins to drive real-world actuators** such as **LEDs, LCDs, motors, any other device accepting control signals**
- It does this by **converting sensor inputs** into numbers and **communicating them** to a **computer running any program capable of serial communication over a serial data** connection with an **open protocol**.
- Arduino Integrated Development Environment (IDE) is **freely** available and Arduino I/O board is an **open hardware**.



Arduino I/O board

Arduino™ UNO Reference Design

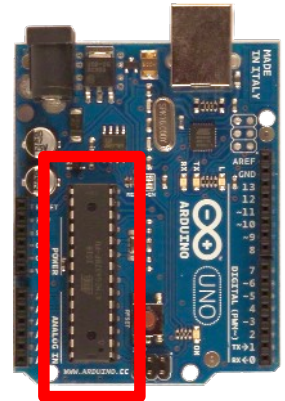
Reference Design & RE PRODUCTION OF THE "ARDUINO UNO" ARE PROVIDED AS IS WITHOUT WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. THE CUSTOMER MUST NOT RELY ON THE SPECIFICATIONS OR CHARACTERISTICS OF ANY HARDWARE OR SOFTWARE "AS IS" UNLESS SPECIFICALLY STATED IN WRITING BY ARDUINO. ARDUINO RESERVES THE RIGHT TO MAKE CHANGES TO THE SPECIFICATIONS OR CHARACTERISTICS OF ANY HARDWARE OR SOFTWARE "AS IS" WITHOUT NOTICE. THE CUSTOMER SHALL BE RESPONSIBLE FOR VERIFYING THE COMPATIBILITY OF ANY HARDWARE OR SOFTWARE "AS IS" WITH THEIR OWN SYSTEMS. THE CUSTOMER SHALL BE RESPONSIBLE FOR VERIFYING THE COMPATIBILITY OF ANY HARDWARE OR SOFTWARE "AS IS" WITH THEIR OWN SYSTEMS. THE CUSTOMER SHALL BE RESPONSIBLE FOR VERIFYING THE COMPATIBILITY OF ANY HARDWARE OR SOFTWARE "AS IS" WITH THEIR OWN SYSTEMS.



Schematics, firmwares and communication protocols are all **open**

Brief Introduction to Microcontrollers (μC) and Arduino I/O board as a simplified DAQ hardware

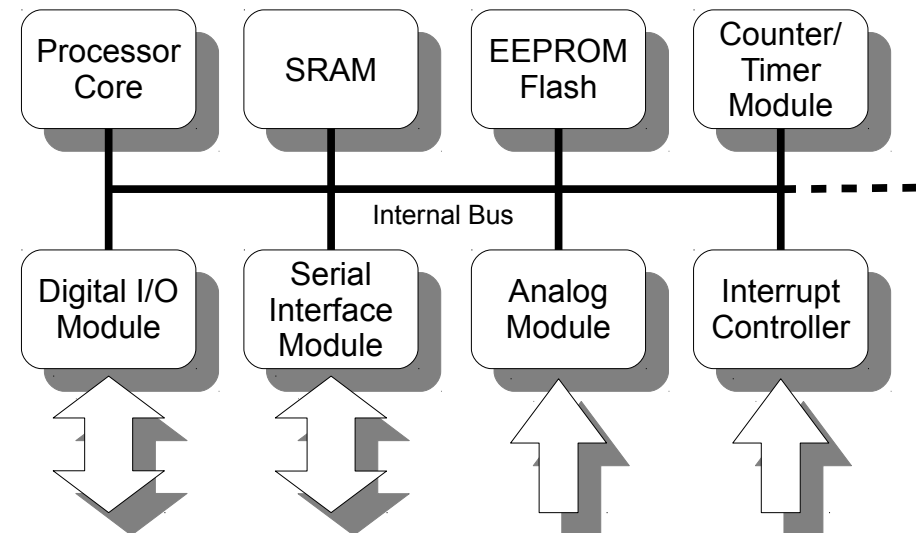
- We said “... It does this by **converting sensor inputs ...**”
- Arduino I/O board has a **micro-controller** which performs all the functions; it is a **tiny computer** with all the needed **peripherals**
- A **micro-controller** is a small **computer** dedicated to a specific field on a single integrated circuit consisting of a **simple CPU**, **RAM**, various **timers**, digital and analog **I/O** etc.



Atmel μC on
Arduino I/O board

- ➔ **Processor core**, runs your C program
- ➔ **SRAM**, is the memory to store data
- ➔ **EEPROM/Flash**, is the memory to store your program
- ➔ **Counter/Timer**, are counting units (e.g. # of seconds elapsed etc.)
- ➔ **Digital I/O**, is for sending/receiving digital information
- ➔ **Serial Interface**, is to communicate with others via various protocols (e.g. USB, RS232, etc)
- ➔ **Analog Module**, to deal with analog signals, A/D converters inside
- ➔ **Interrupt Controller**, a mechanism to detect whether an important thing is happening “outside”

A possible μC
architecture



Brief Introduction to Microcontrollers (μC) and Arduino I/O board as a simplified DAQ hardware

- **Micro-controller** is not a **micro-processor**
 - Micro-processors need other components to work **such as memories, I/O, etc.**
- Micro-controller is an **all-in-one** device
 - Saving **time** and **space** to implement functions
- Micro-controllers are **less powerful** compared to PCs; **however** they are:
 - **Small** – 1 cm x 2-3 cm
 - **Tough** – high/low temperatures, rough environment
 - **Low power** consumers
 - **Integrated** with other **low-level devices** easily **such as sensors, switches, etc.**
 - **Inexpensive** – usually
- Therefore they are **everywhere**:
 - Probably you already carry a few of them all the time **in your pocket**

Brief Introduction to Microcontrollers (μC) and Arduino IDE as a simplified DAQ software

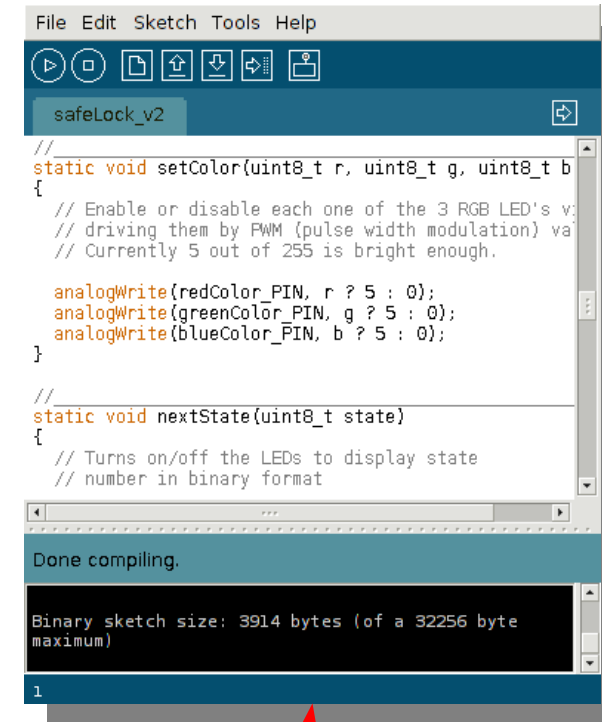
- The Arduino I/O Board accepts **inputs from real-world sensors** such as **switches, potentiometers, light, or sound** sensors, **accelerometers, IR** devices, etc.
- And similarly it can **set values on its pins to drive real-world actuators** such as **LEDs, LCDs, motors, any other device accepting control signals**

```
#define LED0_PIN 11
// Configure general-purpose LED's
pinMode(LED0_PIN, OUTPUT);
digitalWrite(LED0_PIN, LOW);
```

Declare a port as **digital output** and assign **logic 0**

```
#define ACCEL_X_PIN 4
#define ACCEL_Y_PIN 3
#define ACCEL_Z_PIN 2
// Read the 3 accelerometer axes
x = (uint8_t) (analogRead(ACCEL_X_PIN)>>2);
y = (uint8_t) (analogRead(ACCEL_Y_PIN)>>2);
z = (uint8_t) (analogRead(ACCEL_Z_PIN)>>2);
```

Read 10-bit analog value: an **accelerometer** is connected to the **analog port** setting this analog value as a function of gravity vector it experiences



```
File Edit Sketch Tools Help
safeLock_v2
//
static void setColor(uint8_t r, uint8_t g, uint8_t b)
{
  // Enable or disable each one of the 3 RGB LED's v
  // driving them by PWM (pulse width modulation) va
  // Currently 5 out of 255 is bright enough.

  analogWrite(redColor_PIN, r ? 5 : 0);
  analogWrite(greenColor_PIN, g ? 5 : 0);
  analogWrite(blueColor_PIN, b ? 5 : 0);
}

//
static void nextState(uint8_t state)
{
  // Turns on/off the LEDs to display state
  // number in binary format
}
```

Done compiling.

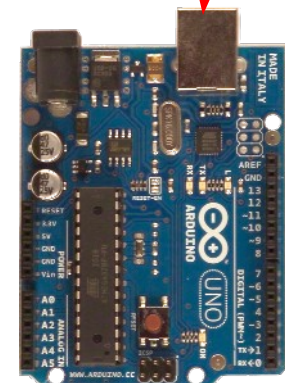
Binary sketch size: 3914 bytes (of a 32256 byte maximum)

1

Arduino IDE

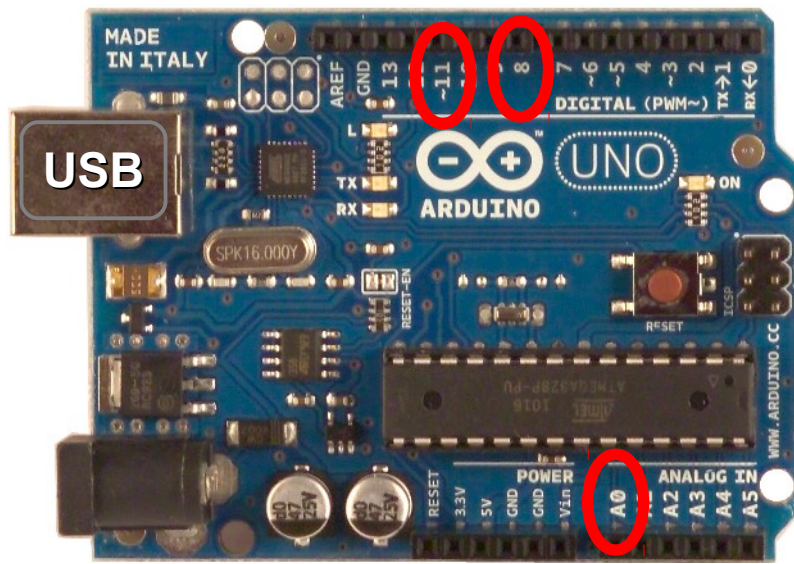


USB cable



Arduino I/O board

Arduino + Gauge Shield Tutorial: Basic idea and code structure



Arduino I/O board



Accelerometer Z	A2	Analog In	0V-3.3V
Potentiometer 0	A1	Analog In	0V-5V
Potentiometer 1	A0	Analog In	0V-5V

Pin correspondence table which comes with the daughter boards (in our case gadget shield)

```
// Potentiometer
#define pot_PIN 0

// Active-low push-button
#define button_PIN 8

// LED display
#define LEDO_PIN 11

// _____
void setup() { // Mandatory function

    // Configure the push-button as input
    pinMode(button_PIN, INPUT);
}

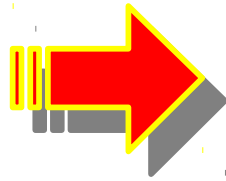
// _____
void loop() { // Mandatory function

    // Read pot value when button is pressed
    if (digitalRead(button_PIN) == LOW) {
        PotVal = analogRead(pot_PIN);
        digitalWrite(LED0_PIN, HIGH);
    } else {
        // Do something else
    }
}
```

Arduino IDE

Lab. 10/A

Table of Contents



Step 0 - Brief introduction to μC

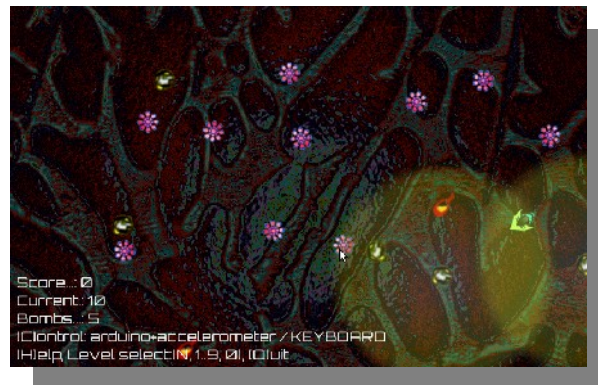


Step 1 - Safe-lock Finite State Machine (FSM)



Step 2 - Coin acceptor FSM

Step 3 – Homework

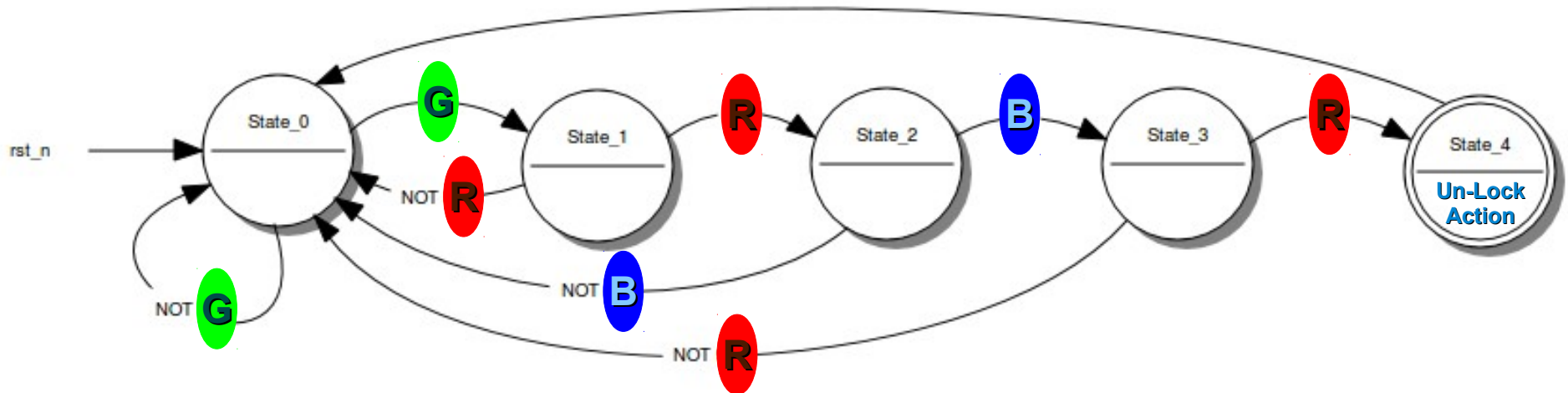




Finite State Machine (FSM)

by a safe lock design example

- FSMs are used to model systems which have a limited number of **states**, **transitions** between these states and the **actions** taken as a result.
- State transition is a function of **the current state AND the input** to the system.



- The FSM above:** value-on-arrow represents what the system senses as an **input**, thus a **transition** from one **state** to another can take place. If there is no input, the system keeps the **current state**.
- The example represents an **acceptor** FSM, parsing the secret **combination** (G, R, B) which, in our example, would un-lock a safe: **green, red, blue** and again **red**. In case an unexpected color is entered, the system returns to the first state: *State_0*. The safe is unlocked, that is, the system reaches the final action (*State_4*), only if the **correct colors** are entered in the **correct order**.
- In the next page, **a possible FSM implementation** in Arduino environment is given.

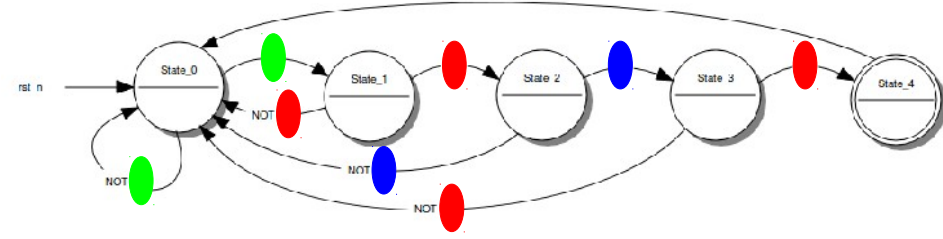


Finite State Machine (FSM)

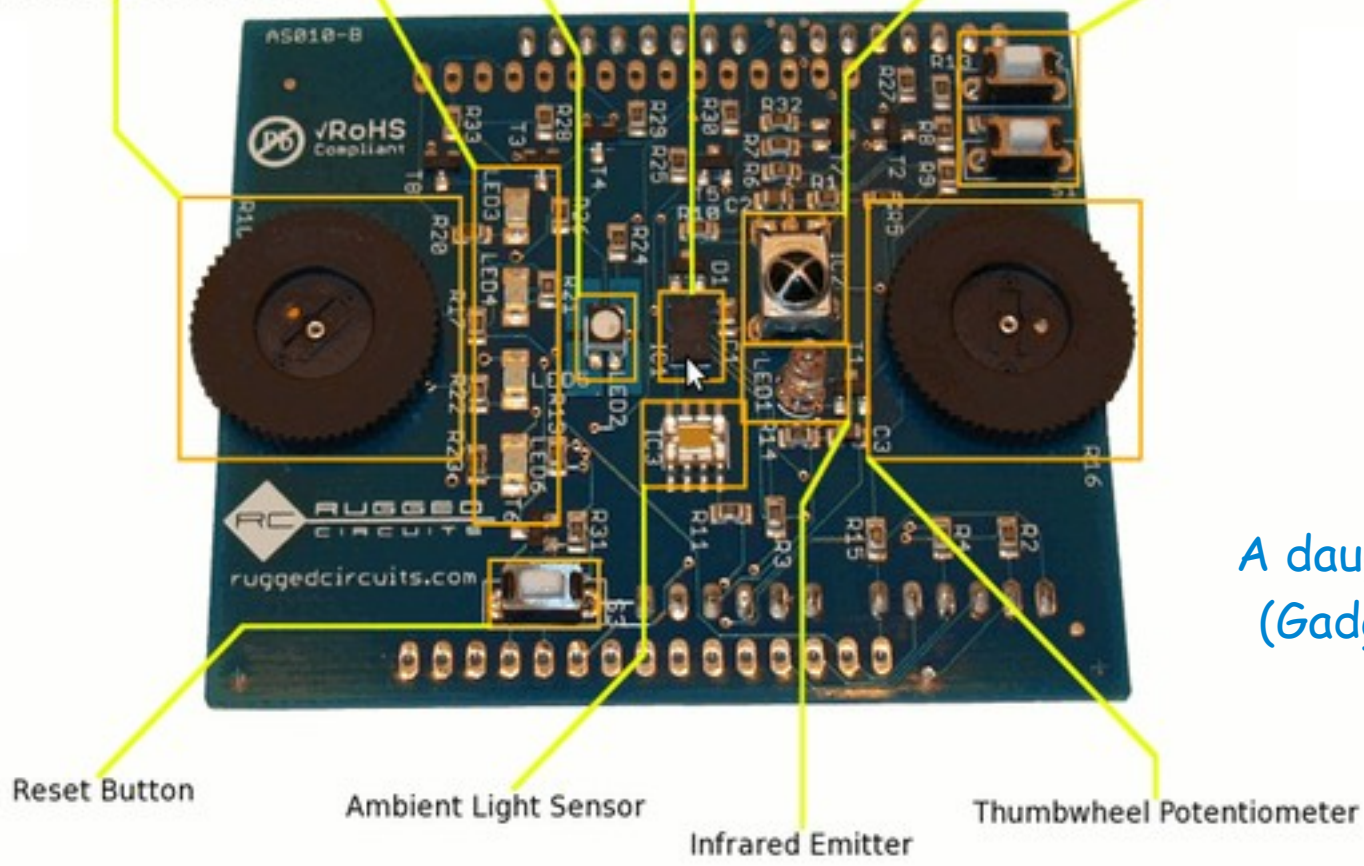
by a safe lock design example

Plan of action:

Use **gadget shield** (daughter board) with **arduino uno** (I/O board).



Color select Thumbwheel Potentiometer
Binary State display Four LED's
Color display RGB LED
Infrared Logic Detector
Color enter Two Pushbutton Switches



A daughter board (Gadget Shield)

Finite State Machine (FSM)

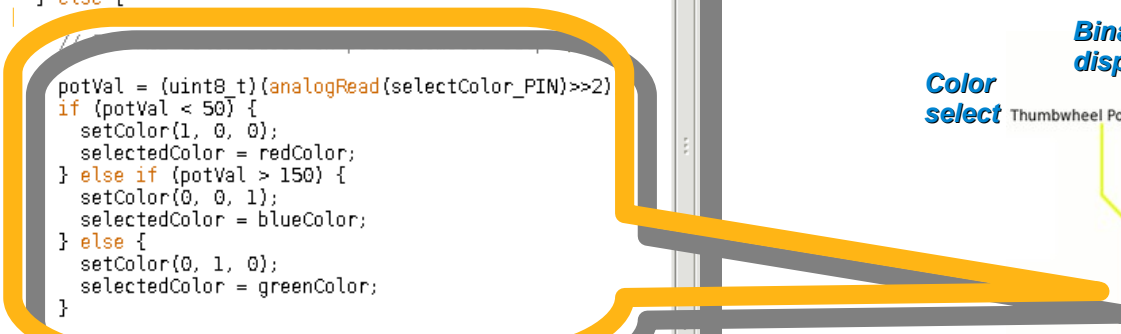
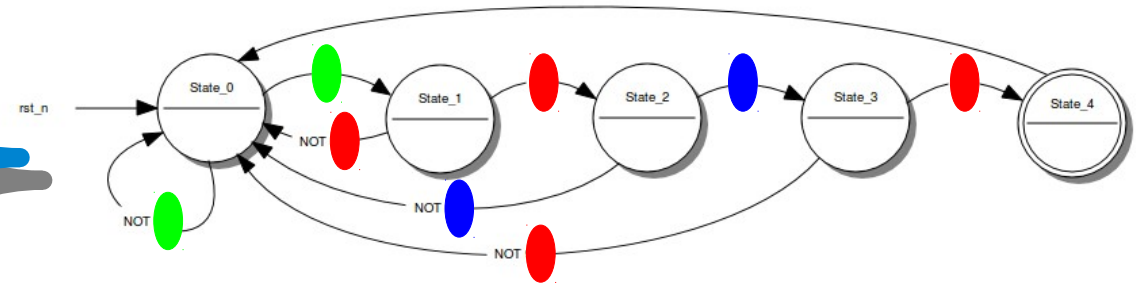
by a safe lock design example

```

File Edit Sketch Tools Help
safeLock_v2
if (digitalRead(insertColor_PIN) == LOW) {
    // Enter the currently selected color
    delay(500);
    enteredColor = selectedColor;

    // Transition the FSM accordingly
    switch(currentState) {
    case State_0:
        // Transition: State_0 -> State_1
        if (enteredColor == greenColor) nextState(State_1);
        else nextState(State_0);
        break;
    case State_1:
        // Transition: State_1 -> State_2
        if (enteredColor == redColor) nextState(State_2);
        else nextState(State_0);
        break;
    case State_2:
        // Transition: State_2 -> State_3
        if (enteredColor == blueColor) nextState(State_3);
        else nextState(State_0);
        break;
    case State_3:
        // Transition: State_3 -> State_4
        if (enteredColor == redColor) {
            nextState(State_4);
            safeUnlockAnimation();
        }
        // Transition: Stage_4 -> State_0
        nextState(State_0);
    } else nextState(State_0);
    break;
    }
} else {
    potVal = (uint8_t){analogRead(selectColor_PIN)>>2};
    if (potVal < 50) {
        setColor(1, 0, 0);
        selectedColor = redColor;
    } else if (potVal > 150) {
        setColor(0, 0, 1);
        selectedColor = blueColor;
    } else {
        setColor(0, 1, 0);
        selectedColor = greenColor;
    }
}
    
```

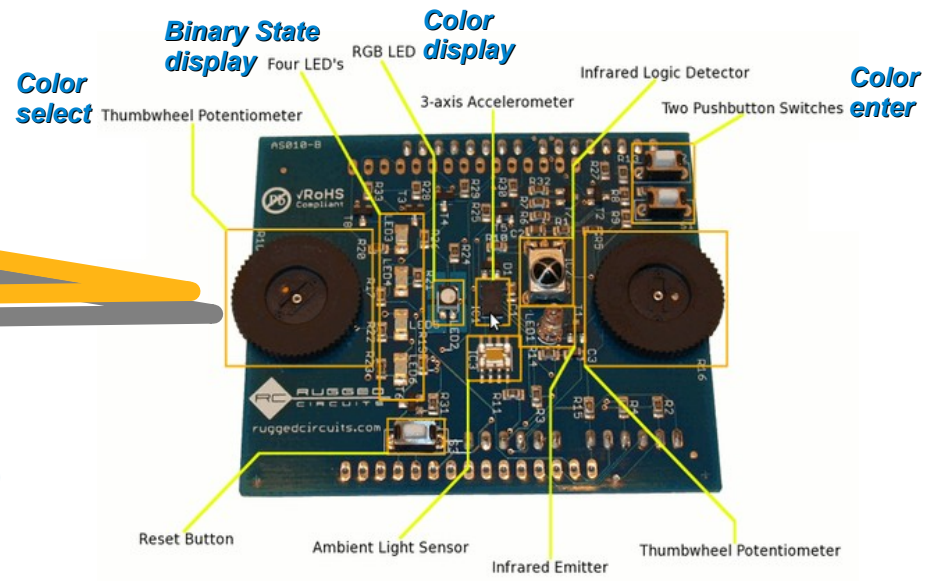
- Implemented in **Arduino IDE**
 - Continuously read potentiometer output
 - Continuously set RGB LED color
- If (the button is pressed) { transition the FSM }**
 - Use **switch/case** for the FSM



```

Done Saving.
Binary sketch size: 3914 bytes (of a 32256 byte maximum)
166
    
```

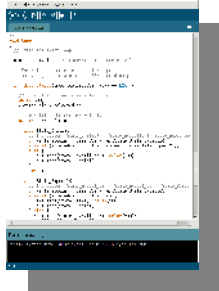



Arduino IDE



Finite State Machine (FSM)

by a safe lock design example

To Do:

- Change directory to “**Lab10a/Step_1**”
- Open the file “**safeLock.pde**” with Arduino IDE 
- Compile () and upload () the I/O board to see it working
 - ➔ Use the **potentiometer to change color** of the RGB LED
 - ➔ Start **serial monitor** () to see the log messages coming from the board
 - ➔ Use the **push button to enter** the selected color to the FSM
 - ➔ Observe the binary display showing the state indexes
- Add another color to the secret combination to make it safer.

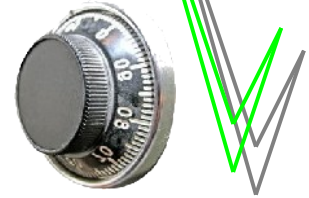
Backup for Lab. 10/A

Q fsm Table da in Contents



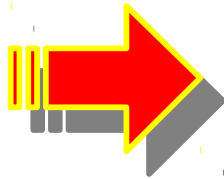
Step 0 - Brief introduction to μC

Step 1 - Safe-lock Finite State Machine (FSM)



Step 2 - Coin acceptor FSM

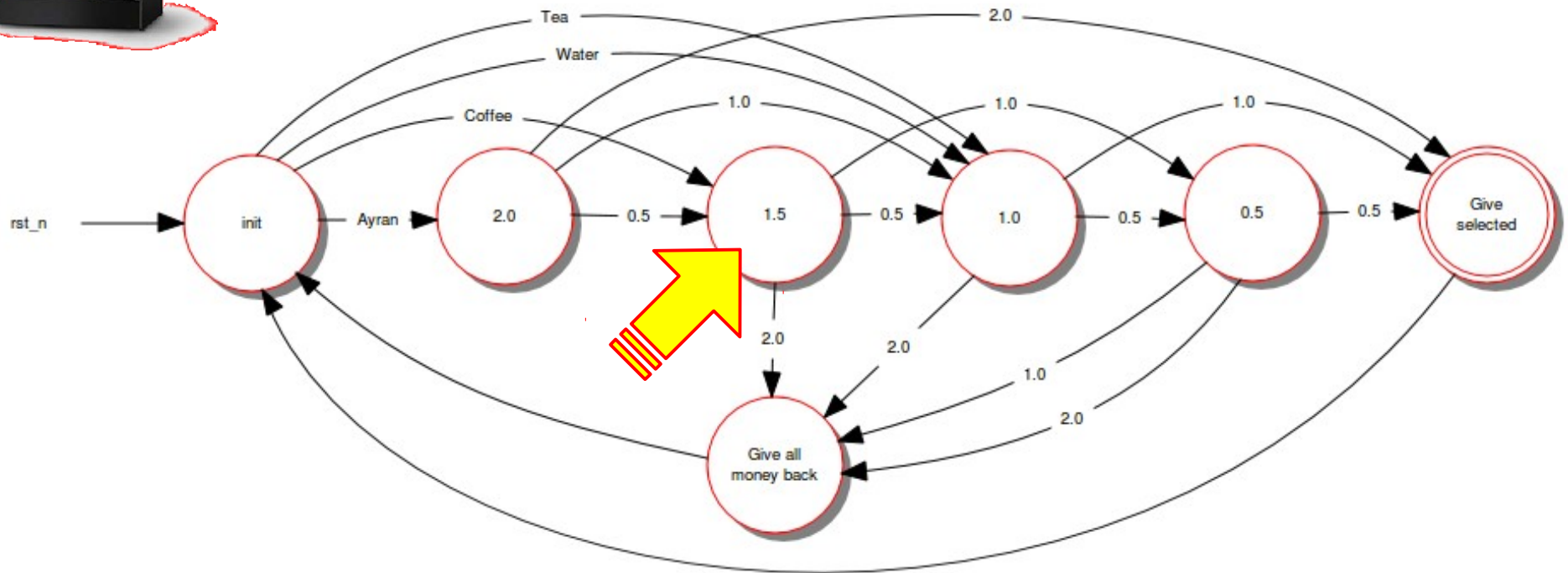
Step 3 - Homework



Finite State Machine (FSM)

by a coin acceptor design example

- FSMs are used to model systems which have a limited number of **states**, **transitions** between these states and the **actions** taken as a result.
- The text on the arrows represent what the system senses as **input**, thus a **transition** from one **state** to another can take place. When there is no input, the system keeps the **current state**.

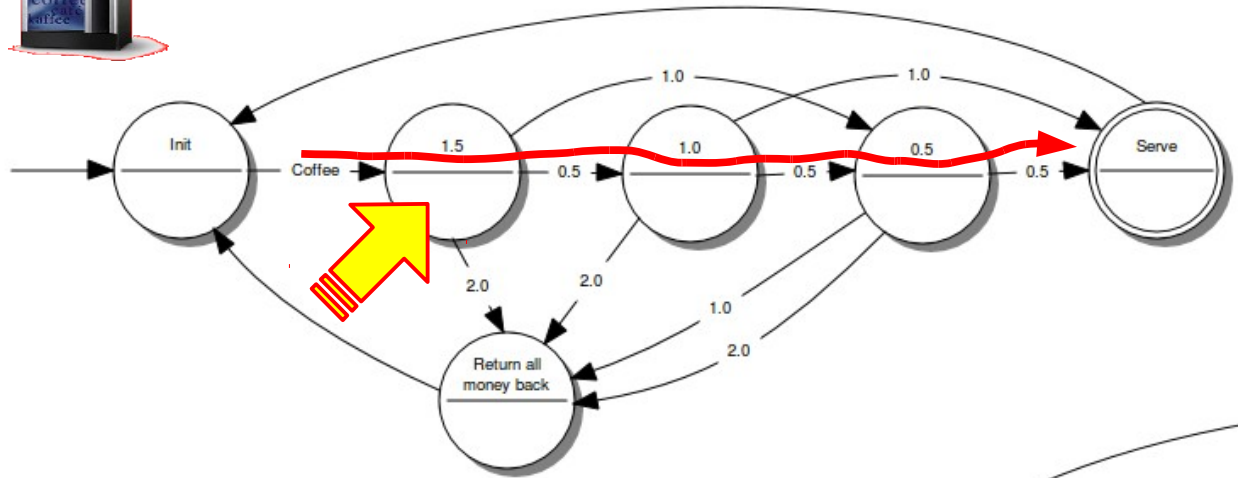


- The example above represents a coin acceptor FSM, expecting the required amount for the selected drink where there are 3 available coins: **2 (red)**, **1 (blue)** and **0.5 (green)**:
 - 1.5** unit for **coffee ristretto**
 - 2.0** unit for **ayran**
 - 1.0** unit both for **water** and **tea**

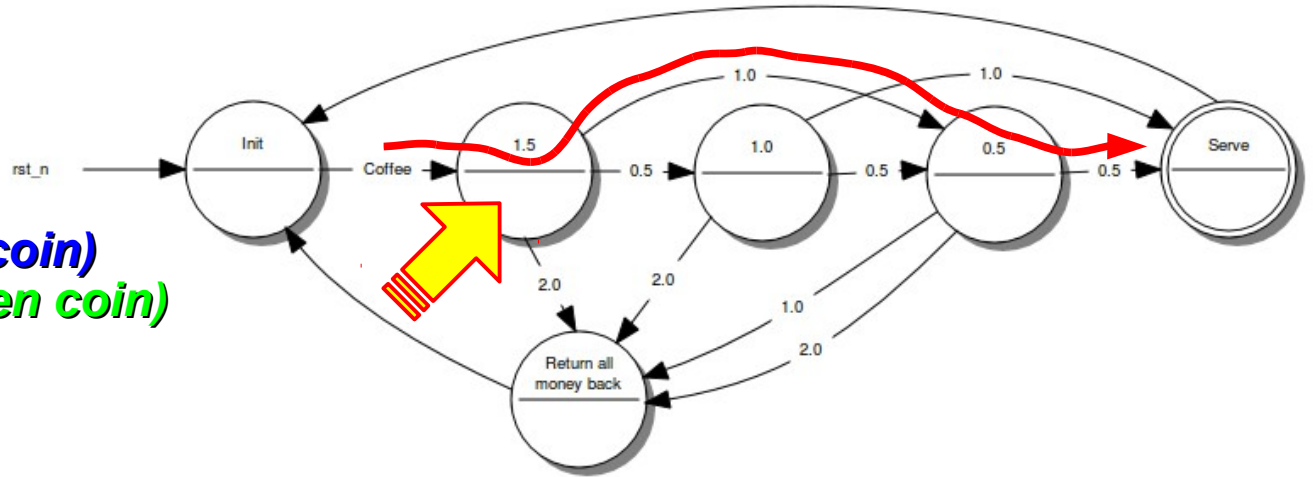


Finite State Machine (FSM)

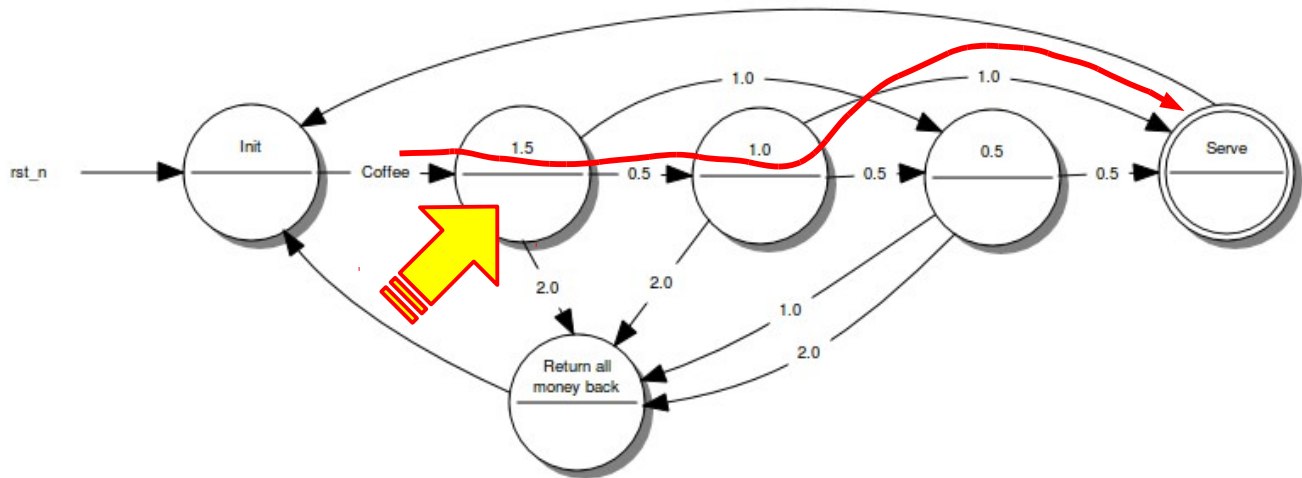
by a coin acceptor design example - Coffee case



Spend:
3 x 0.5 (green coin)



Spend:
1 x 1 (blue coin)
1 x 0.5 (green coin)



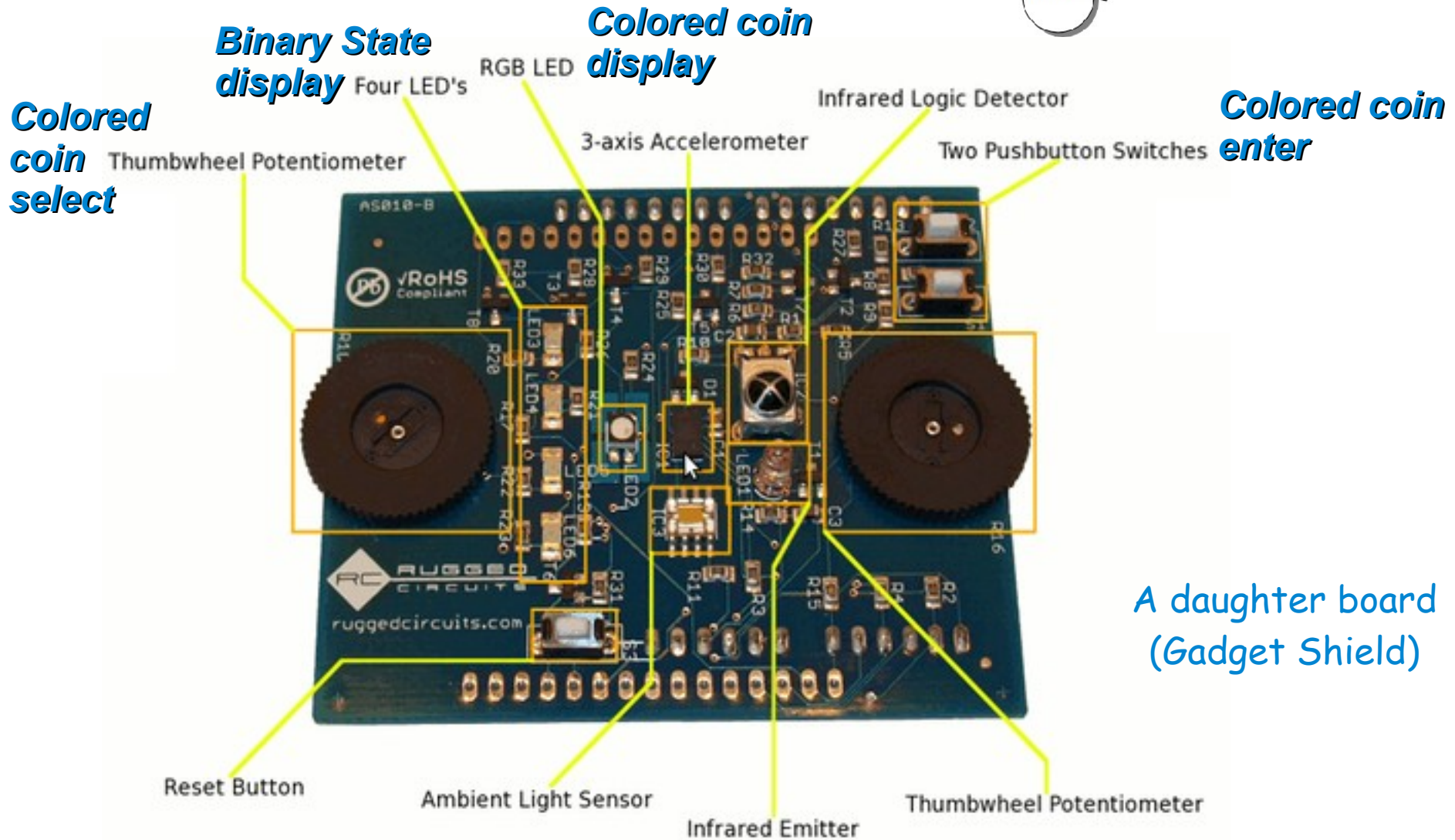
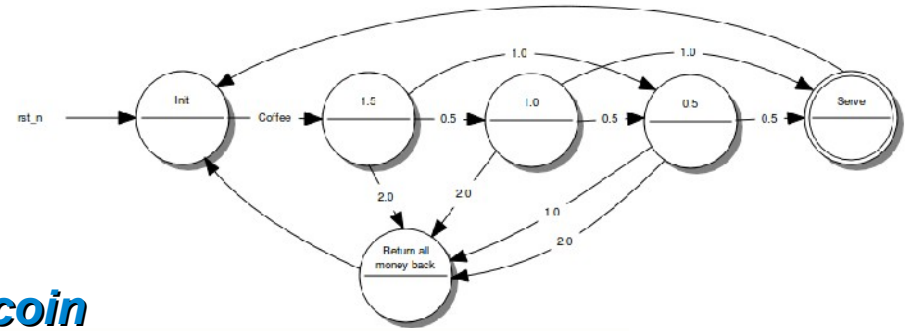
Spend:
1 x 0.5 (green coin)
1 x 1 (blue coin)

Finite State Machine (FSM)

by a coin acceptor design example - Coffee case

Plan of action:

Use **gadget shield (daughter board)** with **arduino uno (I/O board)**.



A daughter board (Gadget Shield)

Finite State Machine (FSM)

Coin acceptor - Coffee case



```
//
void loop()
{
  // Start the event loop

  uint8_t potVal;    // Current potentiometer value

  // Sense the button click as color input
  // and change the state of the FSM accordingly

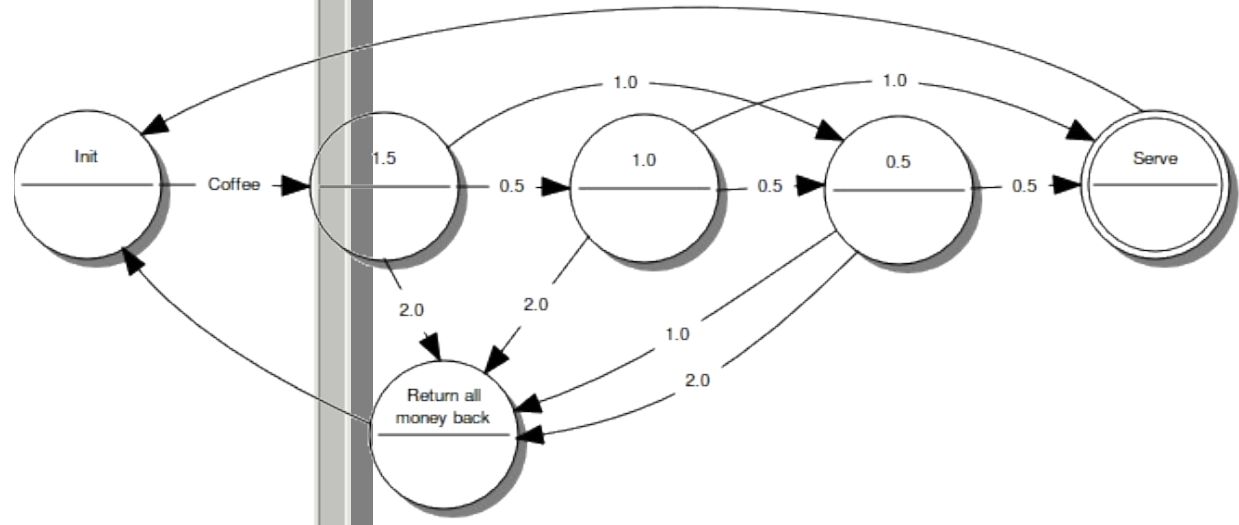
  if (digitalRead(insertColoredCoin_PIN) == LOW) {

    // Enter the currently selected color
    delay(500);
    enteredCoin = selectedCoin;

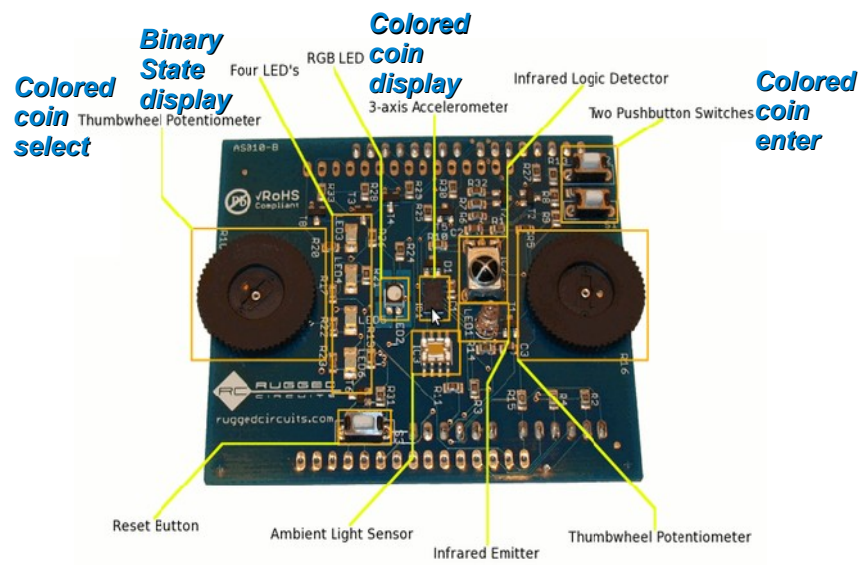
    // Transition the FSM accordingly
    switch(currentState) {

    case State_Initial:
      // Transition: State_Initial -> State_Expect10 || State_Expect05
      if (enteredCoin == greenCoin) nextState(State_Expect10);
      else if (enteredCoin == blueCoin) nextState(State_Expect05);
      else {
        nextState(State_GiveAllBack); delay(500);
        nextState(State_Initial);
      }
      break;

    case State_Expect10:
      // Transition: State_Expect_10 -> State_Expect_05 || State_Serve
      if (enteredCoin == greenCoin) nextState(State_Expect05);
      else if (enteredCoin == blueCoin) {
        nextState(State_Serve); delay(500);
        nextState(State_Initial);
      } else {
        nextState(State_GiveAllBack); delay(500);
        nextState(State_Initial);
      }
    }
  }
}
```



Green coin
 (=0.5)
 entered



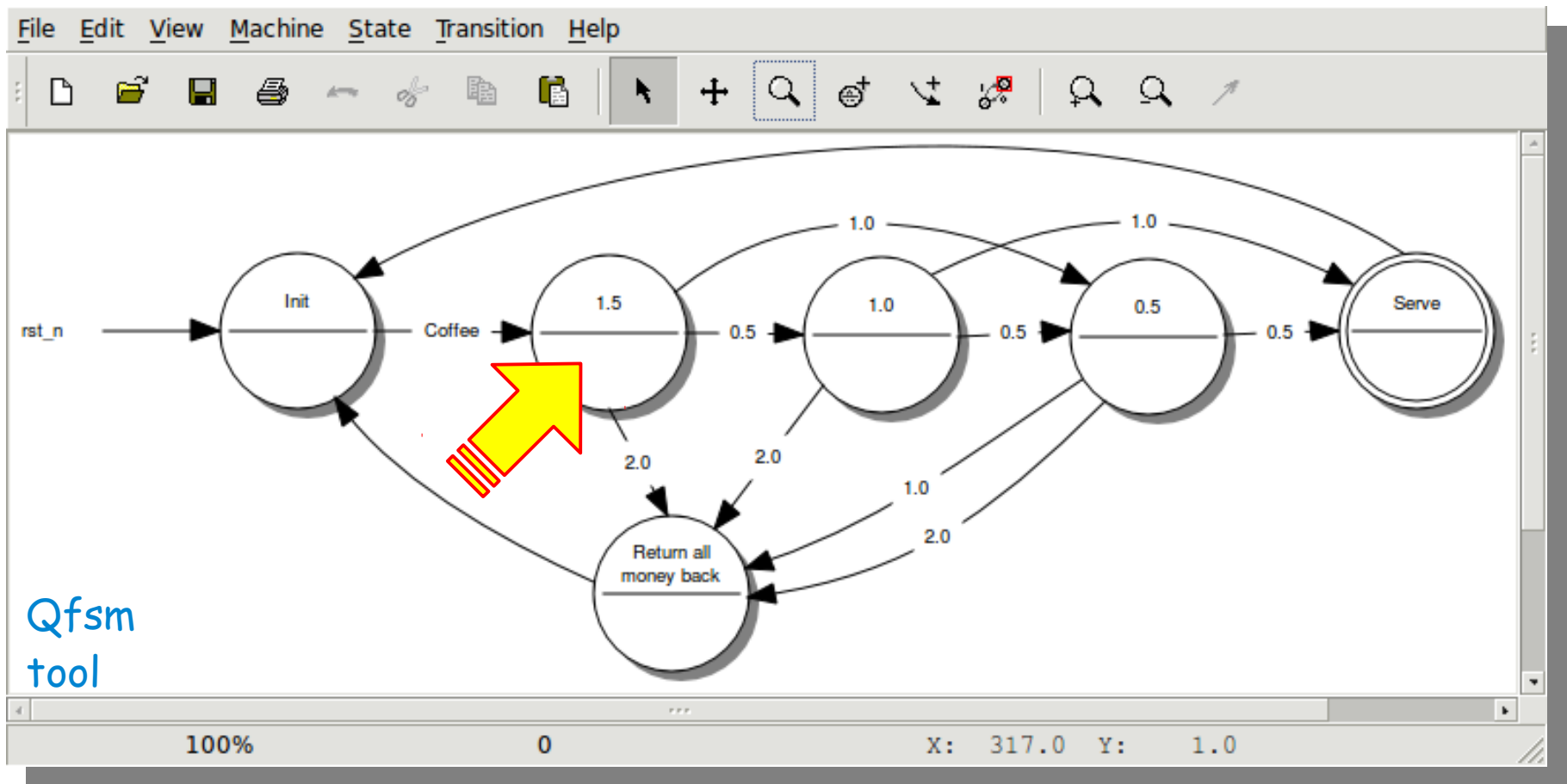
Done uploading.
 Binary sketch size: 4032 bytes (of a 32256 byte maximum)

Arduino
 IDE

Finite State Machine (FSM)

by a coin acceptor design example - Coffee case

- However this FSM has a limitation; let us consider the **coffee case**:
 - ➔ The machine gives what is selected **only if** a client enters the **exact** amount; **otherwise** all the coins are **returned**, terminating the operation
 - ➔ We would like the machine to have the **notion** of “the rest” so that the client is not supposed to enter the exact amount but an amount which can **exceed what is required**
 - ➔ The machine should serve the coffee and only the rest should be returned





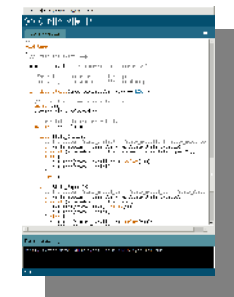
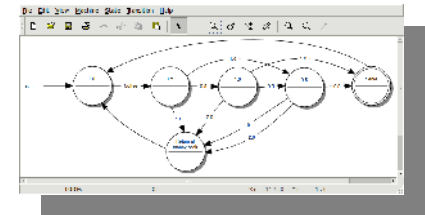
Finite State Machine (FSM)

by a coin acceptor design example



To Do:

- Change directory to “**Lab10a/Step_2**”
- Open the file “**coinAcceptor.fsm**” with **Qfsm**
- Add/remove states and transitions to the FSM so that it **returns the rest** to the user **and delivers** what is selected
 - Use **State > New** menu item to add a new state (or )
 - Use **Transition > New** menu item to add a new transition (or )
 - Use “**Del**” key to remove a selected state and/or a transition
- Open the file “**coinAcceptor.pde**” with **Arduino IDE**
- Modify the implementation accordingly



Lab. 10/A

Table of Contents



Step 0 - Brief introduction to μC

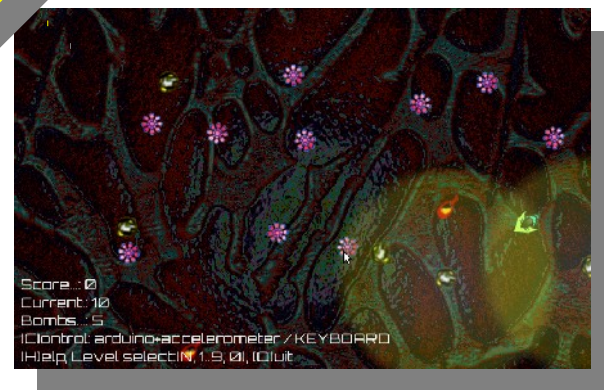
Step 1 - Safe-lock Finite State Machine (FSM)



Step 2 - Coin acceptor FSM



Step 3 – Homework



Optional Homework / A Tilt-to-Dodge Game

The target game is already there with only keyboard controls



- **Accelerometer** connected to **Arduino I/O** board
- **Standalone C/C++ application** reading (x, y, z) accelerometer values **via serial port** (w/o Arduino IDE)
- You do the math: an addictive game !



SIYIRDUINO

- Prerequisites:
 - **Arduino** I/O board
 - An **accelerometer** connected to the Arduino I/O board
 - or
 - **Gadget shield** (a daughter board connected to Arduino)
 - It already has an accelerometer and other devices
 - We used it in the previous examples

Optional Homework / A Tilt-to-Dodge Game

Your task is to implement **arduino+accelerometer** controls

- Download **SIYIRDUINO.tgz** from the web page of the **ISOTDAQ-2011**
- Open **main.cxx** and go the place to be modified (below image)
- Accelerometer outputs are **already** available as the (x, y, z) values in a **string**
- **Parse** the string to control the ship (use keyboard controls as your starting example)

```
File Edit Search Preferences Shell Macro Windows Help
/home/o/Documents/DAQschoolRoma/siyirduino/docir/main.cxx byte 41434 of 53211 L: 1195 C: 24
1168     } else { // Arduino delivers: "xxx.87.yyy.45.zzz.67.xyz\n", where 87, 45
1169         // and 67 are the accelerometer values read-out from the arduino.
1170         // Edit this part: use accelerometer values to control the ship.
1171     /*
1172         Ivmecolcer ayari (Accelerometer calibration):
1173         -----x---y---z-----
1174         duz iken (streight): 79, 86, 110
1175         en sag (right-most): 79, 50, 86
1176         en sol (left-most) : 79, 126, 82
1177         en ileri (forward) : 114, 88, 85
1178         en geri (backward) : 42, 88, 85
1179
1180         Yonelim (allignment): Arduino/Gadget'i duz tuttugumda (holding streight):
1181         masa duzlemi (table surface)          -> soldan saga (left to right)    , y eksenini (y-axis)
1182                                                asagidan yukari (bottom to top), x eksenini (x-axis)
1183         masaya dik (perpendicular to desktop) -> asagidan yukari (bottom to top), z eksenini (z-axis)
1184     */
1185
1186     serialport_read_until(fd, inStr, '\n'); // Read serial port until line end ("\n")
1187     sprintf(tmpStr, "Arduino delivers: %s", inStr); // Create the string to be displayed
1188     printf("%s \n", tmpStr); // Print the created string
1189
1190     // Your optional homework: parse this line and use the (x,y,z) values to control the ship.
1191     // Use the keyboard controls above as your starting point and send your solutions to:
1192     //
1193     //   Ozgur.Cobanoglu@cern.ch
1194     //
1195     // Have fun ![]
1196 }
```

Consol output

```
...
Bad line -> Arduino delivers: .110.xyzyy.83.zzz.109.xyz
Good line -> Arduino delivers: xxx.77.yyy.83.zzz.109.xyz
Arduino delivers: .zzz.110.xyz3.zzz.109.xyz
Arduino delivers: xxx.77.yyy.83.zzz.109.xyz
...
```

Optional Homework / A Tilt-to-Dodge Game

Arduino firmware delivers what the C/C++ executable receives

- Download the **siyirduinoFirmware** for the Arduino I/O board from the web page
- Just compile and **upload** to Arduino I/O board

```
File Edit Sketch Tools Help
siyirduinoFirmware
// Accelerometer device analog input
#define ACCEL_X_PIN 4
#define ACCEL_Y_PIN 3
#define ACCEL_Z_PIN 2

// Accelerometer sensitivity display
#define ACCEL_GSEL_PIN 7

// RGB LED outputs, active high
#define RED_PIN 5
#define GREEN_PIN 6
#define BLUE_PIN 10

// _____
void setup()
{
    // Configure accelerometer g-rating
    pinMode(ACCEL_GSEL_PIN, OUTPUT);
    digitalWrite(ACCEL_GSEL_PIN, LOW); // 1.5g range when low, 6g when high

    // Send data via serial port
    Serial.begin(9600);
}

// _____
static void setRGB(uint8_t r, uint8_t g, uint8_t b)
{
    analogWrite(RED_PIN, r > 16 : 0);
    analogWrite(GREEN_PIN, g > 16 : 0);
    analogWrite(BLUE_PIN, b > 16 : 0);
}

// _____
void loop()
{
    uint8_t x, y, z;

    // Read the 3 accelerometer axes and light the RGB LED color
    x = (uint8_t) (analogRead(ACCEL_X_PIN)>>2);
    y = (uint8_t) (analogRead(ACCEL_Y_PIN)>>2);
    z = (uint8_t) (analogRead(ACCEL_Z_PIN)>>2);
    if ((x>y) && (x>z)) {
        setRGB(1, 0, 0);
    } else if ((y>x) && (y>z)) {
        setRGB(0, 1, 0);
    } else if ((z>y) && (z>x)) {
        setRGB(0, 0, 1);
    } else {
        setRGB(0, 0, 0);
    } // xxx.87.yyy.45.zzz.56.xyz.\n
    Serial.print("xxx"); Serial.print("."); Serial.print(x, DEC); Serial.print(".");
    Serial.print("yyy"); Serial.print("."); Serial.print(y, DEC); Serial.print(".");
    Serial.print("zzz"); Serial.print("."); Serial.print(z, DEC); Serial.print(".");
    Serial.print("xyz"); Serial.println();
    delay(10);
}
```

```
File Edit Sketch Tools Help
siyirduinoFirmware
    analogWrite(GREEN_PIN, g > 16 : 0);
    analogWrite(BLUE_PIN, b > 16 : 0);
}

// _____
void loop()
{
    uint8_t x, y, z;

    // Read the 3 accelerometer axes and light the RGB LED color
    x = (uint8_t) (analogRead(ACCEL_X_PIN)>>2);
    y = (uint8_t) (analogRead(ACCEL_Y_PIN)>>2);
    z = (uint8_t) (analogRead(ACCEL_Z_PIN)>>2);
    if ((x>y) && (x>z)) {
        setRGB(1, 0, 0);
    } else if ((y>x) && (y>z)) {
        setRGB(0, 1, 0);
    } else if ((z>y) && (z>x)) {
        setRGB(0, 0, 1);
    } else {
        setRGB(0, 0, 0);
    } // xxx.87.yyy.45.zzz.56.xyz.\n
    Serial.print("xxx"); Serial.print("."); Serial.print(x, DEC); Serial.print(".");
    Serial.print("yyy"); Serial.print("."); Serial.print(y, DEC); Serial.print(".");
    Serial.print("zzz"); Serial.print("."); Serial.print(z, DEC); Serial.print(".");
    Serial.print("xyz"); Serial.println();
    delay(10);
}
```

Good luck and have fun !..

Lab. 10/A

Table of Contents



Step 0 - Brief introduction to μ C

Step 1 - Safe-lock Finite State Machine (FSM)



Step 2 - Coin acceptor FSM

Step 3 – Homework

