Arduino and multitasking

How to start writing multi tasking programs in Arduino from scratch

Jan Rouvroye

The problem

- It is hard to start programming from scratch
  - Especially when building on existing examples from internet
- Programs become unstructured (spaghetti programming)
  - hard to improve or add functionality
  - hard to debug
  - responsive interaction becomes a problem because loop() becomes too slow

A solution

- Use a more structured approach for developing the software by working step-by-step using finite state machines as core of the program
- A 'state' is the condition of a thing at a specific time.
- Finite state machine: an abstract machine that can be in one of a finite number of states

  (wikipedia http://en.wikipedia.org/wiki/Finite-state_machine).
  - Only one state at a time (current state).
  - Change from one state to another when initiated by a triggering event or condition (transition).
  - Defined by a list of states, and triggering condition for each transition.

Why is it useful

- Divides the program in smaller parts that can be programmed (almost) independently:
  for each state you need to consider what needs to happen when
  - the program enters a new state
  - the program is/stays in a state
  - the program leaves a state
- Provides a clear structure -> easier to build further upon
- Current state is always known -> simpler debugging
- Multiple state-machine processes can be combined in one program ("multi-tasking") -> better (faster) interaction
- Almost self documenting when implemented well

What we use here: multitasking framework by Loe Feijs

Feijs, L.M.G. (2013). *Multi-tasking and Arduino : why and how?*. In L.L. Chen, T. Djajadiningrat, L.M.G. Feijs, S. Fraser, J. Hu, S. Kyffin & D. Steffen (Eds.), Conference Paper : Design and semantics of form and movement. 8th International Conference on Design and Semantics of Form and Movement (DeSForM 2013), 22-25 September 2013, Wuxi, China, (pp. 119-127).

Download from: http://purl.tue.nl/601467275212099.pdf

Core principle of this multitasking approach

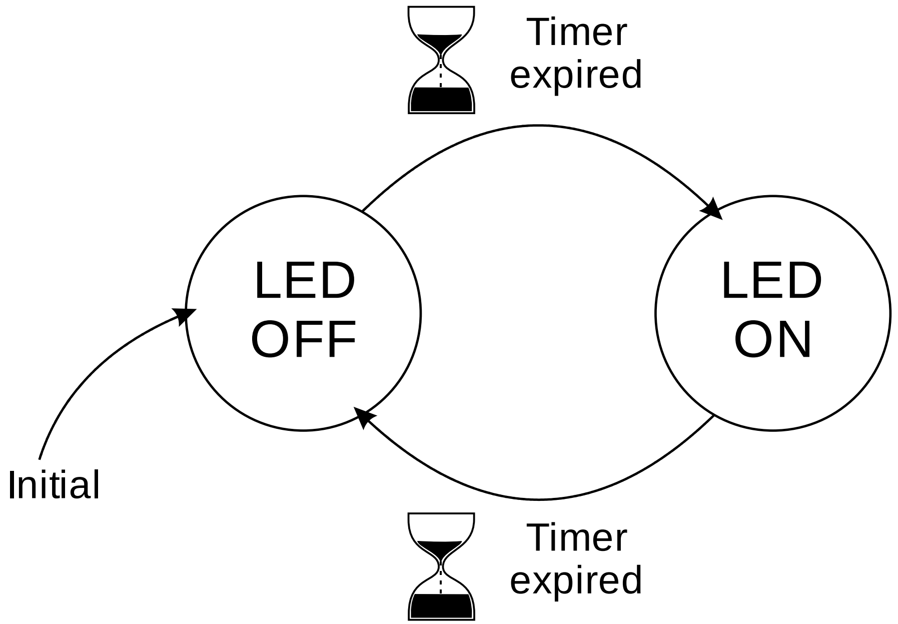
- Run loop() at fixed (short) intervals
- So relation between time passed and number of passes of the loop function is known
  (principle is frequently used in plc machine control computers)
- This means we can use counters for timing! No need to check time using millis()
- Use step function for dealing with sensors and actuators each time of passing through loop()

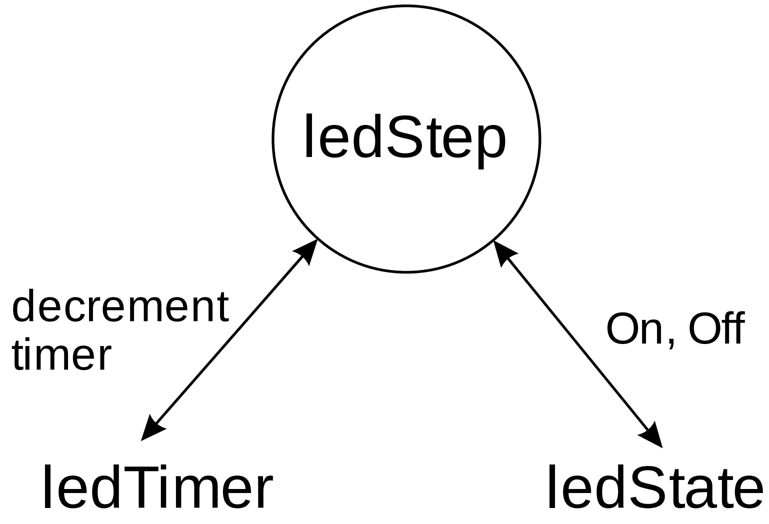
Example 1: Blink

- Blink: We want to program a LED to blink on/off at certain time intervals.

- States? Transitions? Processes?
  - One process: ledStep() which handles the led
  - States: led can be on or off
  - Transitions: when timer expired change to other state

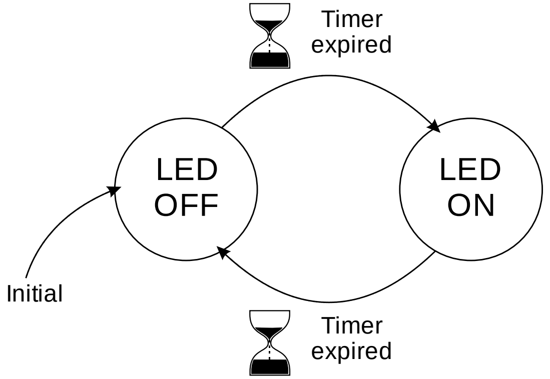
State transition and data flow diagram





What to do related to each state

	LED OFF	LED ON
When entering	- switch off led - initialize timer	- switch on led - initialize timer
When in state	- decrease timer - when timer expires change state to LED ON	- decrease timer - when timer expires change state to LED OFF
When leaving	-	-



Programming state machines

- Use counters that count number of passes of the loop for timing
- Program state machine using switch() construction

```
switch (var)
{
 case label1:
 // statements for label1
 break;

 case label2:
 // statements for label2
 break;
 …
 default:
 // statements for other labels (optional)
}
```

Execute part of program depending on value of variable ‘var’

If the value of ‘var’ = = labeli: execute the statements until next break;

Optional (may or may not be included) for other values of ‘var’ execute the following statements

loop()

```
void loop()
 {
   ledStep();

   delay(10); // loop once every 10 ms (if no other processes delay execution)
 }
```

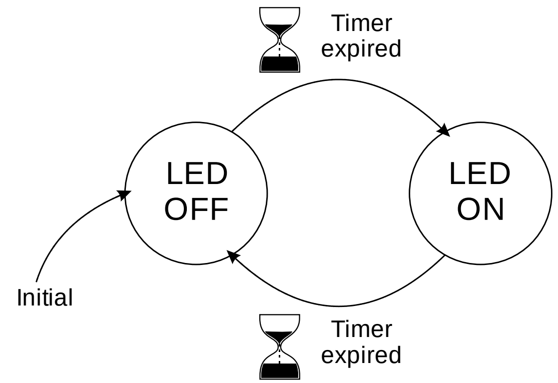
A function is a block of code designed to perform a particular task

This function will perform the process for the led state machine.
It runs every time the loop runs, so every 10 ms

ledStep()

```
void ledStep() {
  switch(ledState) {
   case LED_ON:
     digitalWrite(LED_PIN, HIGH);
     if (ledTimer > 0) // stay in LED_ON state
       ledTimer = ledTimer - 1;
     else { // change to LED_OFF state
       ledState = LED_OFF;
       ledTimer = LED_OFF_TIME;   // initialize timer for LED_OFF state
     }
     break;

   case LED_OFF:
     digitalWrite(LED_PIN, LOW);
     if (ledTimer > 0) // stay in LED_OFF state
       ledTimer = ledTimer - 1;
     else { // change to LED_ON state
       ledState = LED_ON;
       ledTimer = LED_ON_TIME;   // initialize timer for LED_ON state
     }
     break;
  }
}
```



Each time passing the loop ledTimer will decrease by 1; if it becomes 0 state will change and ledTimer will be re-initialized

Initializations and setup()

I prefer using pre-compiler statements

```
#define LED_PIN 11   // pin where led is connected
#define LED_OFF 0
#define LED_ON 1
#define LED_OFF_TIME 250          // number of intervals of 10 ms when the led is off
#define LED_ON_TIME 10            // number of intervals of 10 ms when the led is on

int ledState; // variable indicating the state of the led, either LED_OFF or LED_ON
int ledTimer; // counter for the number of intervals of 10 ms when timing is needed

void setup()
{
  pinMode(LED_PIN, OUTPUT);
  ledState = LED_OFF;                  // initial state is LED_OFF
  ledTimer = LED_ON_TIME;
}
```

Global scope for ledState and ledTimer because they are used in several functions

Complete program

```
/* This program blinks a led without the use of the delay function
 * Multitasking framework used is from paper by Loe Feijs entitled Multi-tasking
 * and Arduino: Why and How?  Published in Chen, L.-L., T. Djajadiningrat,
 * L. Feijs, S. Fraser, J. Hu, S. Kyffin and D. Steffen, Eds. (2013).
  * Design and semantics of form and movement. 8th International Conference
 *  on   Design and Semantics of Form and Movement (DeSForM 2013).
 * ISBN 978-90-386-34623,  Wuxi, Philips
 *
 * Software coding Jan Rouvroye for workshop multitasking Arduino.
  * Last edit: May 23, 2014
 */

#define LED_PIN 11            // pin where led is connected
#define LED_OFF 0
#define LED_ON 1
 #define LED_OFF_TIME 250        // number of intervals of 10 ms when the led is off
#define LED_ON_TIME 10          // number of intervals of 10 ms when the led is on

int ledState; // variable indicating the state of the led, either LED_OFF or LED_ON
int ledTimer; // counter for the number of intervals of 10 ms when timing is needed

void setup() {
 pinMode(LED_PIN, OUTPUT);
 ledState = LED_OF;          // initial state is LED_ON

 ledTimer = LED_OFF_TIME;
 }
```

```
void loop() {
 ledStep();

  delay(10); // loop once every 10 ms (if no other processes delay execution)
}


void ledStep() {
  switch(ledState) {
   case LED_ON:
     digitalWrite(LED_PIN, HIGH);
     if (ledTimer > 0) // stay in LED_ON state
       ledTimer = ledTimer - 1;
     else { // change to LED_OFF state
       ledState = LED_OFF;
       ledTimer = LED_OFF_TIME;   // initialize timer for
LED_OFF state
     }
     break;

   case LED_OFF:
     digitalWrite(LED_PIN, LOW);
     if (ledTimer > 0) // stay in LED_OFF state
       ledTimer = ledTimer - 1;
     else { // change to LED_ON state
       ledState = LED_ON;
       ledTimer = LED_ON_TIME;   // initialize timer for LED_ON
state
     }
     break;
  }
}
```

Example 2: hotel corridor light

- We want to create a light that:
  - Is off by default (energy saving) and switches to full brightness after a button has been pressed
  - Stays on full brightness for a certain time period
  - Then slowly fades to off so if needed the user has time to press the button again
  - When the button is pressed during fading the light goes to full brightness once more and the timer is reset
  - We want to include debouncing for the button

Processes

- One process for detecting whether the button is pressed (including debouncing): buttonStep()

- One process for controlling the led: ledStep()

- buttonStep() communicates to ledStep() whether valid button press has been detected

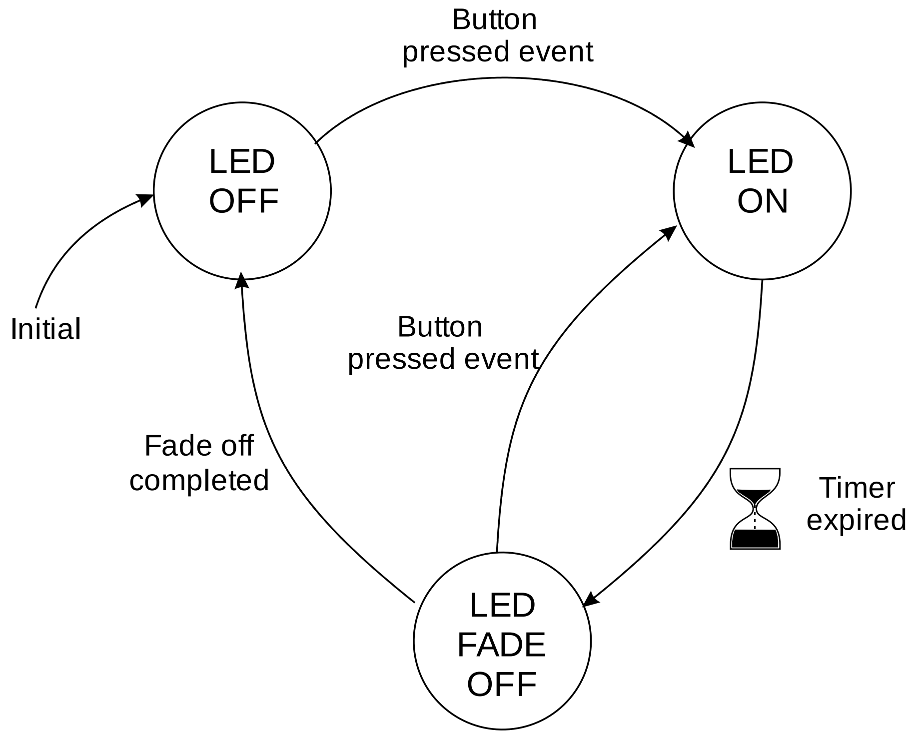
- Both processes will be implemented as finite state machines

Led process: states, transitions?

- Led process

	LED OFF	LED ON	LED FADING
When entering	- switch off led	- switch on led - initialize onTimer	- initialize fadeTimer
When in state	- if buttonEvent = = true change state to LED ON	- decrease onTimer - when onTimer expires change state to LED FADING	- when timer expires change fadeValue, re-initialize fadeTimer - if buttonEvent = = true change state to LED ON - if fadeValue == 0 change state to LED OFF
When leaving	-	-	-

Led process: state transition diagram

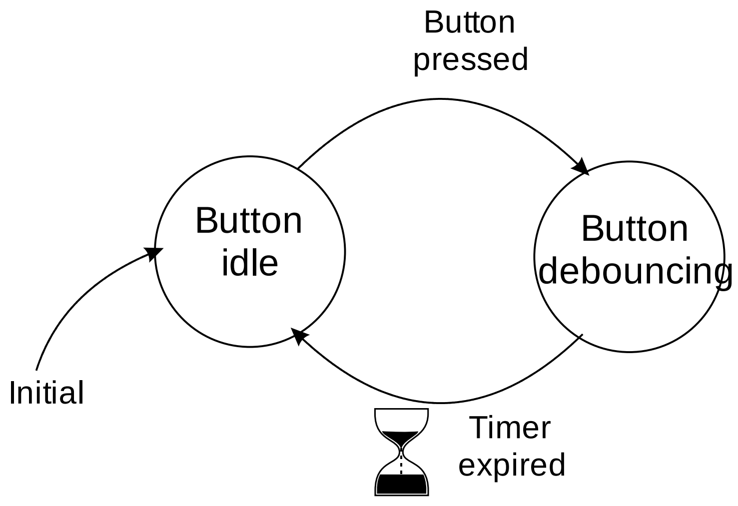


Button process: states, transitions?

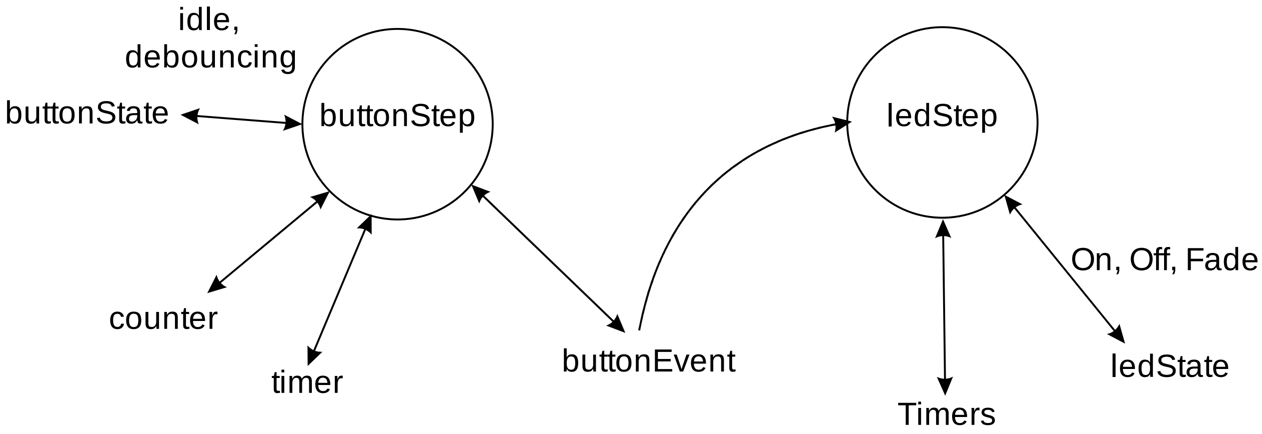
- Button process

	IDLE	DEBOUNCING
When entering	-	- initialize counter - initialize timer
When in state	- if button is pressed change state to DEBOUNCING	- decrease timer - if button is pressed increase counter - when timer expires set buttonEvent according to evaluation result ,change state to IDLE
When leaving	-	-

Button process: state transition diagram



Data flow diagram



The program: loop()

```
void loop() {
 buttonstep();
 ledStep();

 //printDebuginfo();

 delay(10); // loop once every 10 ms (if no other processes delay execution)
}
```

These functions will perform the process for the button respectively led state machine.
They run every time the loop runs, so every 10 ms

This function will print information for debugging to the serial port. Commented out during regular use because it slows down loop() too much while printing which results in timing issues

The program: ledStep()

```
void ledStep() {
 // handles the led depending on the led state
 static int onTimer; // timer for led on full brightness
 static int fadeTimer; // timer for fading
 static int currentPwm = 0; // pwm settng for led (0-255)

 switch(ledState) {
   case STATE_LED_OFF:
     currentPwm = 0;
     if (buttonEvent == true) {
       ledState = STATE_LED_ON;
       onTimer = ONTIMER_SETTING;
     }
   break;
   case STATE_LED_ON:
     currentPwm = 255;
     if (onTimer > 0)
       onTimer = onTimer - 1;
     else {
       ledState = STATE_LED_FADEOFF;
       fadeTimer = FADETIMER_SETTING;
     }
   break;
```

Value will be written to led pin at the end of ledStep()

… … continued on next slide … …

The program: ledStep() continued

```
void ledStep() {
 // handles the led depending on the led state
 static int onTimer; // timer for led on full brightness
 static int fadeTimer; // timer for fading
 static int currentPwm = 0; // pwm setting for led (0-255)

 switch(ledState) {
   case STATE_LED_OFF: … … break;
   case STATE_LED_ON: … … break;

   case STATE_LED_FADEOFF:
     if (fadeTimer > 0)
       fadeTimer = fadeTimer - 1;
     else {
       fadeTimer = FADETIMER_SETTING;
       currentPwm = currentPwm - FADESTEP;
       if (currentPwm < 0) {
         currentPwm = 0;
         ledState = STATE_LED_OFF;
       }
     }
     if (buttonEvent == true) {
       ledState = STATE_LED_ON;
       onTimer = ONTIMER_SETTING;
     }
   break; }
 analogWrite(LED_PIN, currentPwm);
}
```

See previous slide

If timer expires: decrease pwm value
Value will be written to led pin at the end of ledStep()

The program: buttonStep()

```
void buttonstep() {
 // reads button, performs debouncing and sets buttonEvent indicator accordingly
 int buttonValue;
 static int buttonState = STATE_BUTTON_IDLE;
 static int buttonPressedcounter, debounceTimer;

 buttonValue = digitalRead(BUTTON_PIN);

 buttonEvent = false;
 switch(buttonState) {
   case STATE_BUTTON_IDLE:
     if (buttonValue == LOW) //remember LOW means button is being pressed {
       buttonPressedcounter = 1;
       buttonState = STATE_BUTTON_DEBOUNCING;
       debounceTimer = DEBOUNCE_TIMER_SETTING;
     }
   break;
   case STATE_BUTTON_DEBOUNCING:
     if (buttonValue == LOW)
       buttonPressedcounter++;

     debounceTimer--;
     if (debounceTimer < 0) // timer expired {
       buttonState = STATE_BUTTON_IDLE;
       if (buttonPressedcounter > DEBOUNCE_TIMER_SETTING/2)
       // button is assumed to be pressed when at least half of the time of the debouncing period it was
pressed
         buttonEvent = true;
     }
   break;
 }
}
```

First time button press detected from within idle state.

If button press detected within debouncing interval: increase count

After timer has expired: indicate whether press is valid

The program: initializations and setup()

```
#define LED_PIN 11 // pin where led is connected
#define BUTTON_PIN 12 // pin where button is connected
// states for led
#define STATE_LED_OFF 0
#define STATE_LED_ON 1
#define STATE_LED_FADEOFF 2
// states for button
#define STATE_BUTTON_IDLE 0
#define STATE_BUTTON_DEBOUNCING 1
// other settings
#define DEBOUNCE_TIMER_SETTING 15 // timer setting for button debouncing
#define ONTIMER_SETTING 500 // timer setting for led on
#define FADETIMER_SETTING 10 // timer setting for fade off to dimmed (number of timesteps before next
fade step)
#define FADESTEP 1 // pwm setting for dimmed state

int ledState; // variable indicating the state of the led
boolean buttonEvent = false; // indicator buttonpress has been registered after debouncing (true or
false)

void setup() {
 Serial.begin(9600); // for debugging

 pinMode(LED_PIN, OUTPUT);
 ledState = STATE_LED_OFF;
 analogWrite(LED_PIN, 0);

 pinMode(BUTTON_PIN, INPUT_PULLUP);
}
```

The program: printDebuginfo()

```
void printDebuginfo() {
 Serial.print("buttonEvent: ");
 Serial.print(buttonEvent);
 Serial.print("\t");
 Serial.print("\t");
 Serial.print("ledState: ");
 Serial.print(ledState);
 Serial.print("\t");
 Serial.println();
}
```

Print relevant information for debugging to the serial port, one line each time the function runs. Separation is done using tabs (“\t”)

Exercise

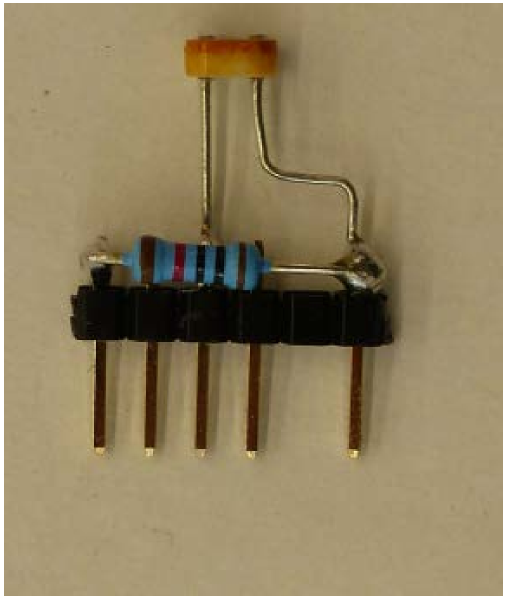
Make a better hotel corridor light that:

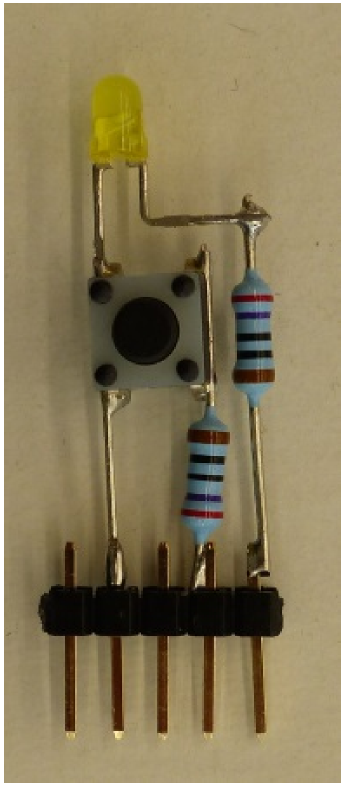
- Is off when the environment has a sufficient light level
- When the environment is dark the light fades on to a dimmed setting
- The light switches to full brightness after a button has been pressed at any time
- Stays on full brightness for a certain time period
- Then slowly fades to off to the setting determined by the light level in the environment so if needed the user has time to press the button again
- We want to include debouncing for the button

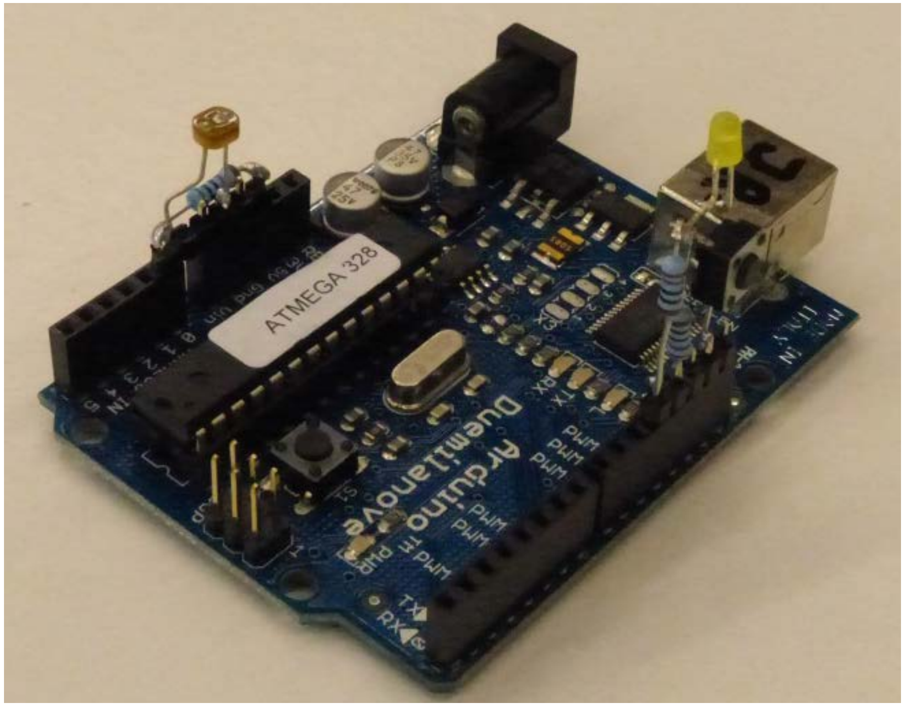
Hardware pictures

Hardware:
- Light: LED connected to digital pin 11 and GND
- button connected to pin 12 and GND

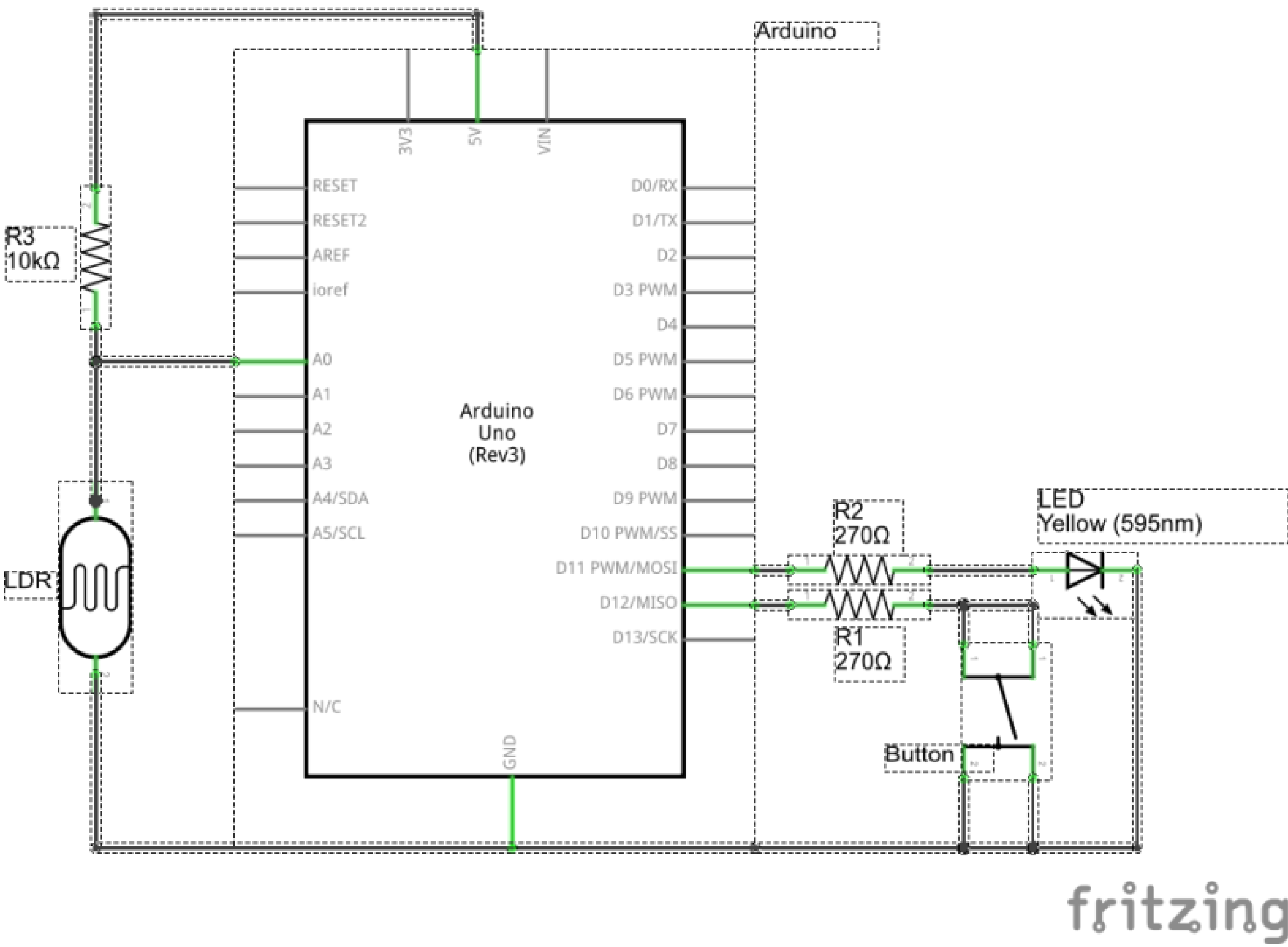
Hardware:
- Sensor: LDR
connected to pin A0 using
voltage divider

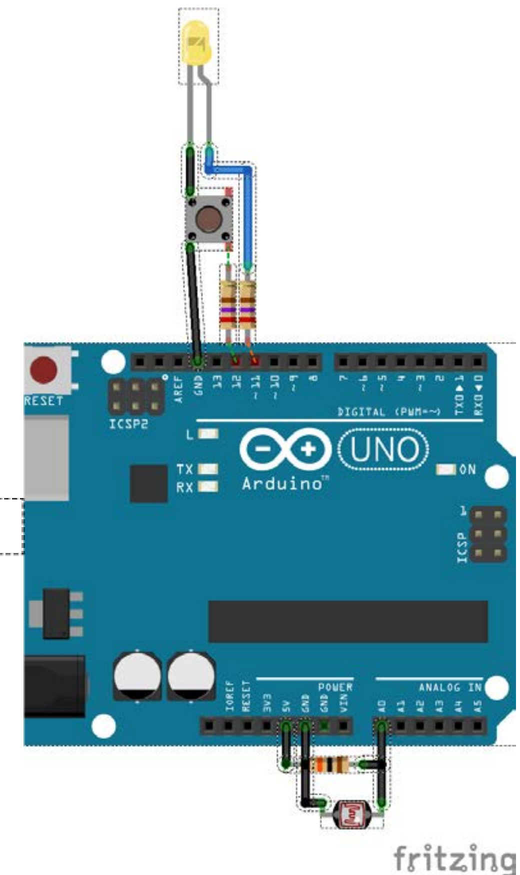






Hardware schematics





Steps

1. Define processes and determine how they interact
2. For each process define states and transitions
3. Make state transition and data flow diagrams
4. For each state determine what to do when
   - Entering state
   - Staying in the state
   - Leaving the state
5. Now you can start programming

Programming

1. Start with developing the loop() function
2. Program the processes in separate functions
   - Program the state machines by focusing on each state separately.
   - Don't forget initializations for the next state when changing state.
3. Program declarations, initializations, definitions on the fly when you need them
4. It may be worthwhile to include a function to create output for debugging.

Tips and tricks

- In order to react to slow environmental changes only, use a moving average for example the exponentially weighted moving average:

$$avg(t = 0) = value(t = 0)$$
$$avg(t + \Delta t) = \alpha * avg(t) + (1 - \alpha) * value(t)$$
$$0 \ < \alpha < 1$$

- Programming this:

```
static float ldrAvg = (float)analogRead(LDR_PIN); // initialize average with current value

ldrValue = analogRead(LDR_PIN);

ldrAvg = alfa*(float)ldrValue + (1-alfa)*ldrAvg;
```