Georgia Institute of Technology

I/O Software Interface, Outputs, and Inputs

(Module 2)

Vertically Integrated Projects (VIP) Program

Contents

- I/O Software Interface
  - Configuring registers
  - The header file
  - Subsystem interfaces
  - Toggling an output
- Outputs
  - Types
    - Digital
    - PWM
  - Setting output pins
  - Turning on and off an LED
- Inputs
  - Types
    - Digital
    - Analog
  - A switch for a digital input
    - Modified software architecture
    - Button as interrupt signal
    - Debouncing digital inputs
  - Analog inputs
    - Analog to digital conversion

I/O Software Interface

- Configuring registers
  - Registers are memory-mapped
    - i.e. each register may be accessed through an address
  - Based on bit-wise operations and Boolean algebra
    - Setting the third bit in the register to 1

```
register = register | (1 << 3);
register |= (1 << 3);             // more compact
```

    - Setting the third bit in the register to 0

```
register &= ~(1 << 3);
```

I/O Software Interface

- The Header File
  - Many times provided by processor or compiler vendor
  - Defines constants naming raw register addresses
    - Example, in LPC13xx.h

```
typedef struct{
__IO uint32_t DATA;
uint32_t RESERVEDO[4095];   // 12 bits of the address bus
                            // are used for bit masking
                            // (See manual 7-4.1)
__IO uint32_t DIR           // direction set for output
__IO uint32_t IS            // interrupt sense
__IO uint32_t IBE           // interrupt in both edges
...
} LPC_GPIO_TypeDef;

#define LPC_AHB_BASE       (0X50000000UL)
#define LPC_GPIO0_BASE     (LPC_AHB_BASE + 0x00000)
#define LPCGPIO01          ((LPC_GPIO_TypeDef  *) LPCGPIO01_BASE)
```

I/O Software Interface

- The Header File
  - Example of "define" statements

```
#define LED_SET_DIRECTION (P1DIR)
#define LED_REGISTER      (P1OUT)
#define LED_BIT           (1 << 3)
```

  - Purpose
    - To configure processor-independent output subsystems

```
LED_SET_DIRECTION |= LED_BIT;   // set the output
LED_REGISTER |= LED_BIT;        // turn on LED
LED_REGISTER &= ~LED_BIT;       // turn off LED
```

    - Allows handling of different devices and hardware upgrades

```
// ioMapping.h
#if COMPILING_FOR_V1
#include "ioMapping_v1.h"
#elif COMPILING_FOR_V2
#include "ioMapping_v2.h"
#else
#error "No I/O map selected. What is your target?"
#endif
```

I/O Software Interface

- Subsystem Interfaces
  1) An I/O Write function
     - Defines the state of a pin (HIGH or LOW) at a given port
     - Makes use of less code space, but makes use of more RAM

```
IOWrite(port, pin, state);
```

  2) Two functions with equivalent effect: I/O Set and I/O Clear
     - Sets or clears the state of a pin at a given port
     - Makes use of less RAM, but may require more code space

```
IOSet(port, pin);
IOClear(port, pin);
```

  3) Another alternative: I/O Toggle
     - Switches the state of a pin at a given port
     - Employs a comparable number of processing cycles than the I/O Set – I/O Clear combination

```
IOToggle(port, pin);
```

I/O Software Interface

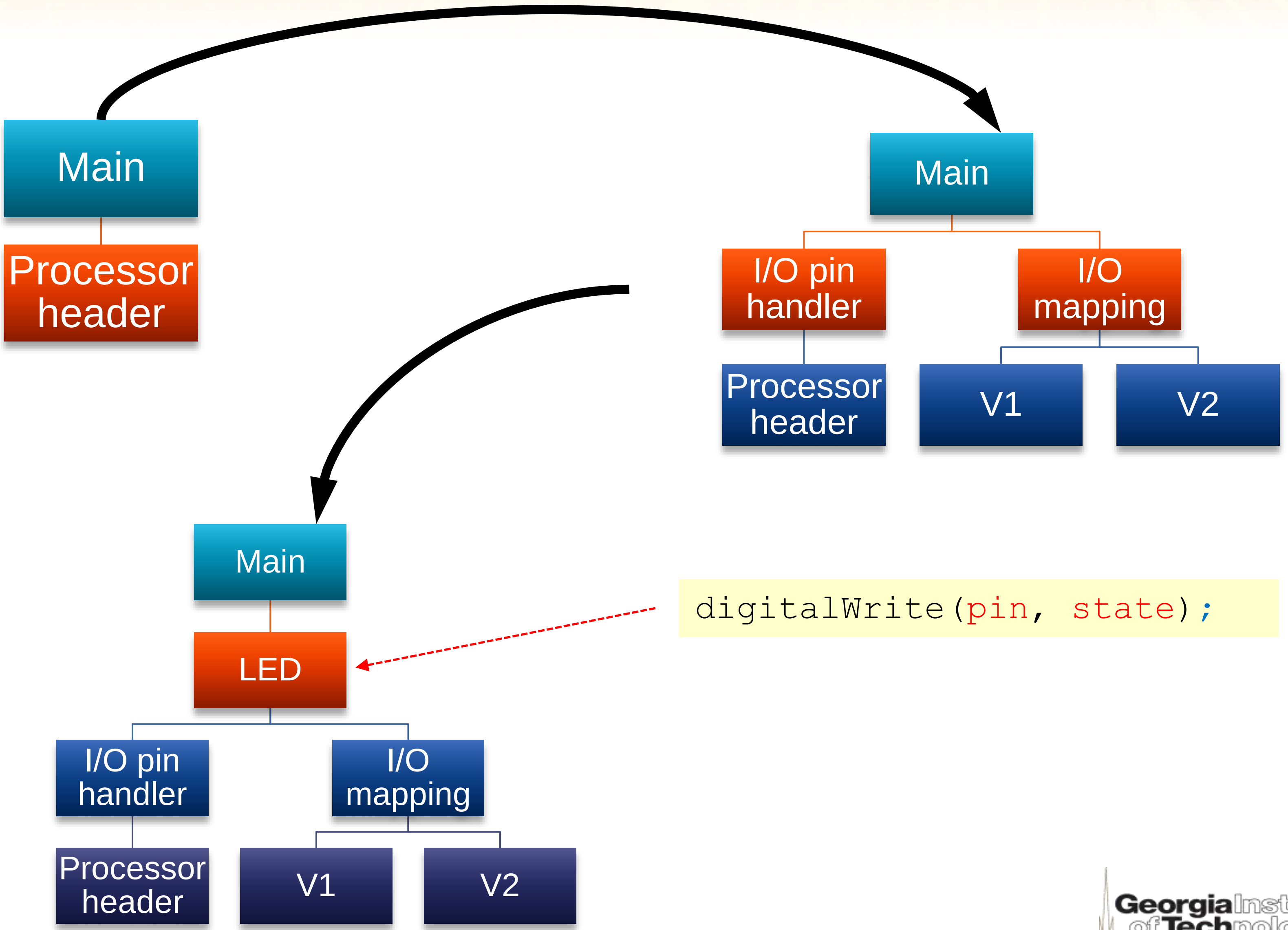
- Toggling an output
  - Option with I/O Write

```
void main(){
    IOSetDir(LED_PORT, LED_PIN, OUTPUT);
    while (1) { // spin forever
        IOWrite(LED_PORT, LED_PIN, HIGH);
        DelayMs(DELAY_TIME);
        IOWrite(LED_PORT, LED_PIN, LOW);
        DelayMs(DELAY_TIME);
    }
}
```

  - Option with I/O Toggle

```
void main(){
    IOSetDir(LED_PORT, LED_PIN, OUTPUT);
    while (1) { // spin forever
        IOToggle(LED_PORT, LED_PIN);
        DelayMs(DELAY_TIME);
    }
}
```

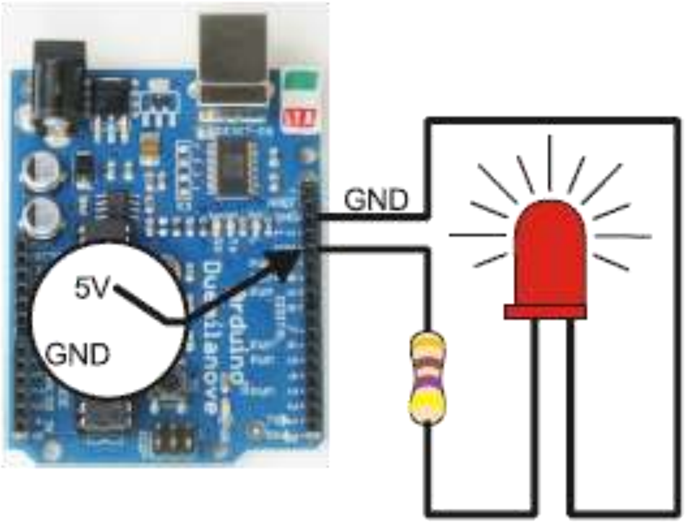
I/O Software Interface



Outputs: Types

- Digital
  - Voltage = 5 V
    - Digital: 1
    - Boolean: TRUE
    - Level: HIGH
  - Voltage = 0 V
    - Digital: 0
    - Boolean: FALSE
    - Level: LOW

```
digitalWrite(13, HIGH);
```




```
digitalWrite(13, LOW);
```

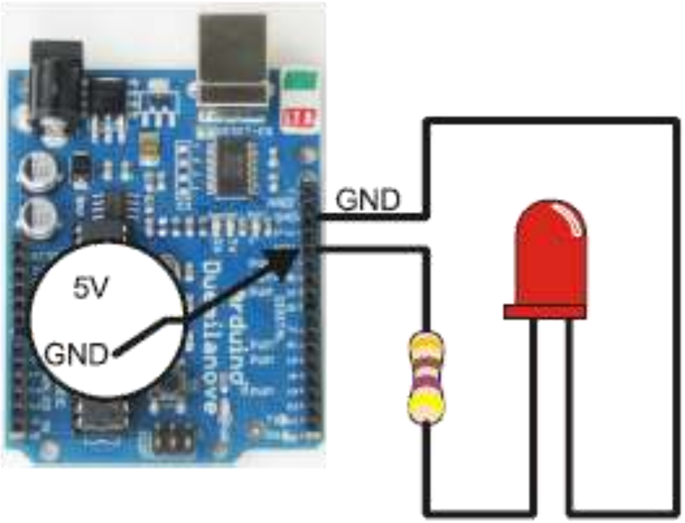


Image from http://learn.parallax.com/node/176

Outputs

- Pulse width modulation (PWM)
  - Produces the effect of a analog output (Fig 1, red) by changing the width (sometimes also the polarity) of a train of pulses (Fig 1, blue)
  - The train of pulses (Fig 2, pink) is obtained by modulating its duty cycle
    - A triangular or sawtooth signal (Fig 2, blue)
    - A carrier or modulating signal (Fig 2, green)
  - Disadvantage: introduces harmonic components to electrical systems

```
analogWrite(pin, value);
```

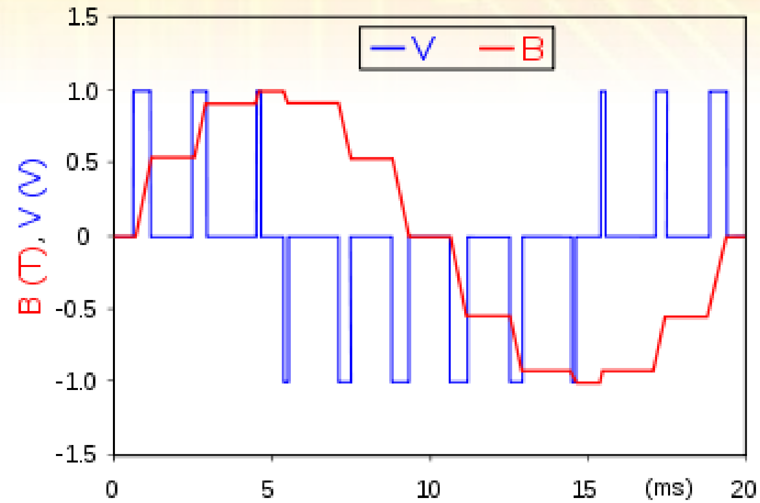


Fig 1

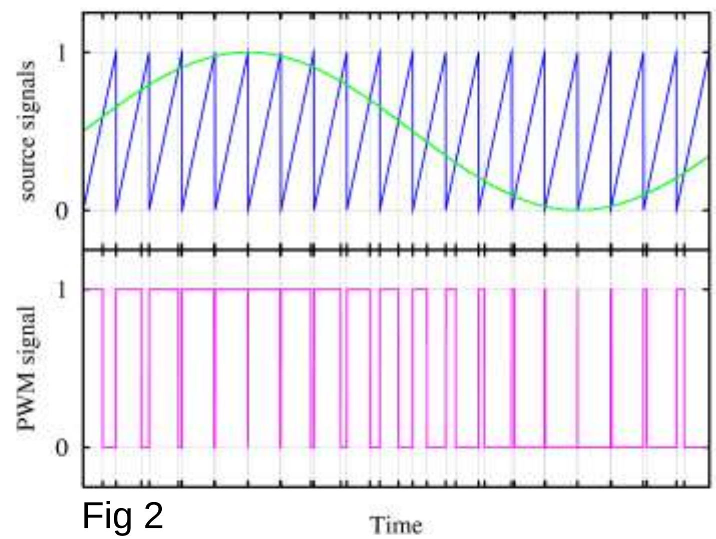
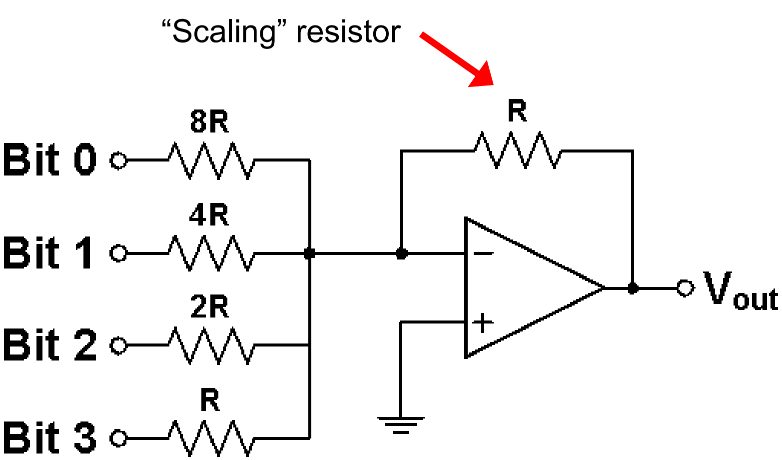


Fig 2

Output: An alternative to PWM

- What if there is no PWM on your board?
  - Then, you can employ:
    - Digital outputs
    - Breadboard
    - Op-amp circuit



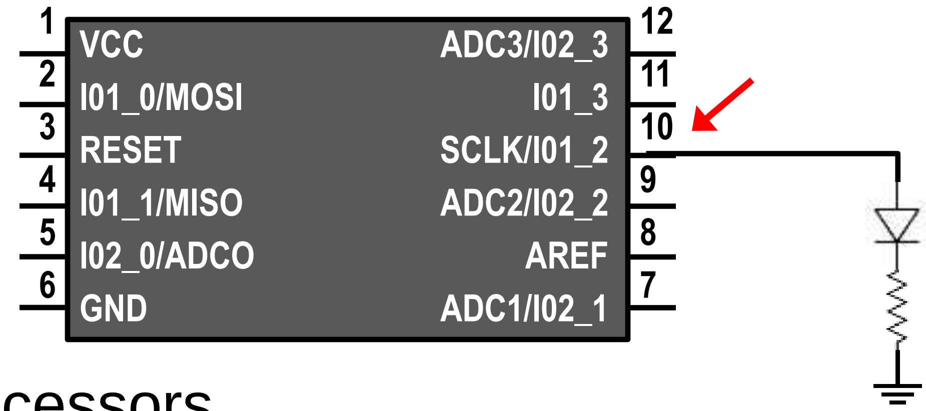
Digital Input Code				Analog Output Voltage
Bit 3	Bit 2	Bit 1	Bit 0	(V)
0	0	0	0	0.000
0	0	0	1	-0.625
0	0	1	0	-1.250
0	0	1	1	-1.875
0	1	0	0	-2.500
0	1	0	1	-3.125
0	1	1	0	-3.750
0	1	1	1	-4.375
1	0	0	0	-5.000
1	0	0	1	-5.625
1	0	1	0	-6.250
1	0	1	1	-6.875
1	1	0	0	-7.500
1	1	0	1	-8.125
1	1	1	0	-8.750
1	1	1	1	-9.375

$$V_{OUT} = -V_{ref}\left(\frac{1}{1}Bit3 + \frac{1}{2}Bit2 + \frac{1}{4}Bit1 + \frac{1}{8}Bit0\right)$$

Georgia Institute of Technology

Outputs

- Setting output pins (s/w side)
  - Set pin 10 (SCLK/I01_2) to be an output:



  - Example processors
    - LPC13XX

```
LPC_GPIO01->DIR |= (1 << 2);
```

    - MSP430

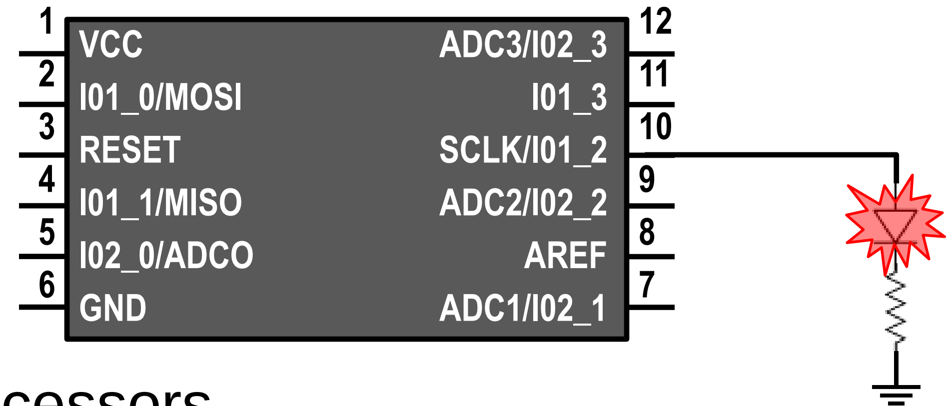
```
P1DIR |= (1 << 2);
```

    - ATtiny

```
DDRB |= (1 << 2);
```

Outputs

- Turning on the LED
  - Set I01_2 to HIGH:



  - Example processors
    - LPC13XX

```
LPC_GPIO01->DATA |= (1 << 2);
```

    - MSP430

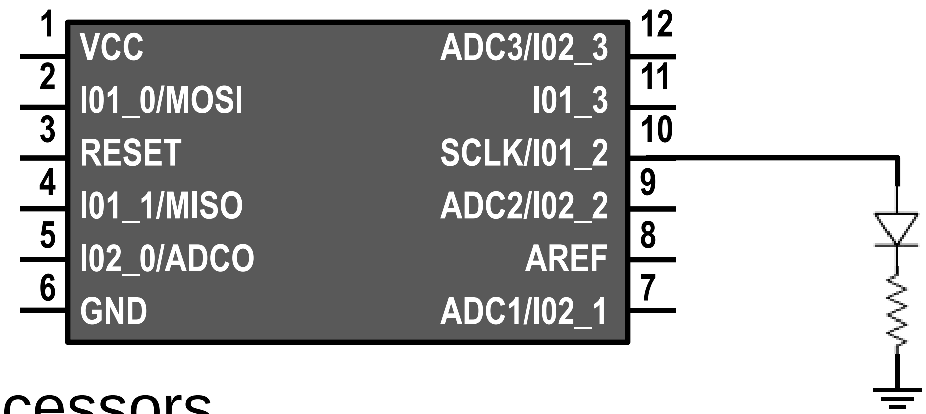
```
P1OUT |= BIT2;
```

    - ATtiny

```
PORTB |= 0x4;
```

Outputs

- Turning off the LED
  - Set I01_2 to LOW:



  - Example processors
    - LPC13XX

```
LPC_GPIO01->DATA &= ~(1 << 2);
```

    - MSP430

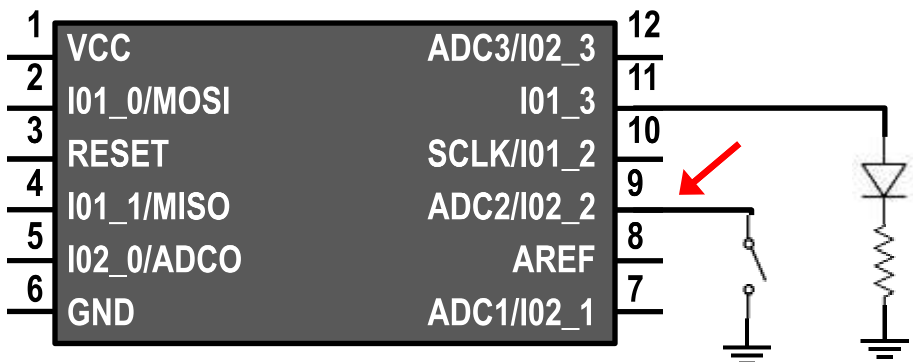
```
P1OUT &= ~(BIT2);
```

    - ATtiny

```
PORTB &= ~0x4;
```

Inputs

- Including a switch for digital input
  - Set pin 9 to be an input and pin 11 to be an output:



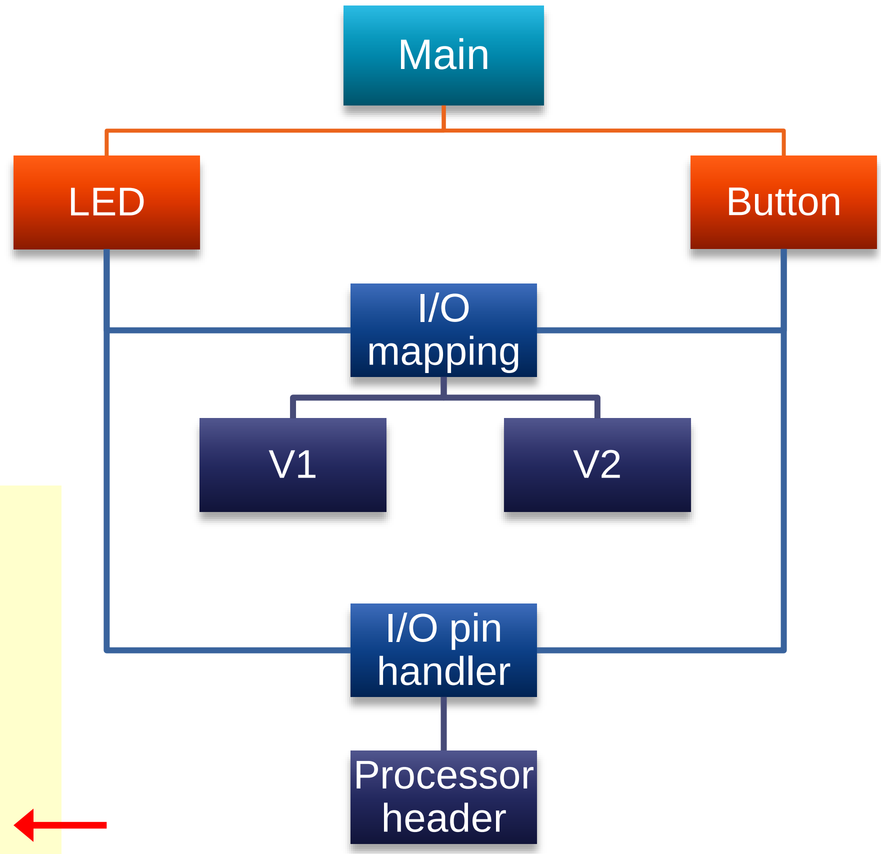
  - Setup procedure
    - Include the input pin in the header file
    - Set the pin as an input if necessary
    - Configure pin to be pull-up (5 V when open) if necessary.
  - Behavior
    - The switch will connect pin 9 to ground when closed

Georgia Institute of Technology

Inputs

- Higher level software architecture
  - Includes a button subsystem
  - A façade simplifies the button subsystem interface
  - Button reuses:
    - I/O pin handler
    - I/O mapping header file
  - Implementation

```
main:
    initialize LED
    initialize button
loop:
    if button pressed, turn LED off
    else toggle LED
    do nothing for a period of time
    repeat
```



Can you see why interrupts are useful?

Georgia Institute of Technology

Inputs

- Button as interrupt signal
  Configuring the button pin as an interrupt

  - Pin interrupt setting is separate from input setting

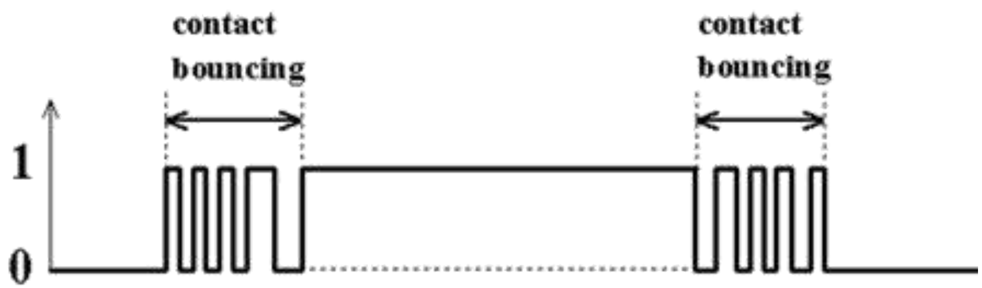
  - Adds three functions to the I/O software interface
    - `IOConfigureInterrupt(`*`port`*`, `*`pin`*`, `*`trigger type`*`, `*`trigger state`*`)`
    - `IOInterruptEnable(`*`port`*`, `*`pin`*`)`
    - `IOInterruptDisable(`*`port`*`, `*`pin`*`)`

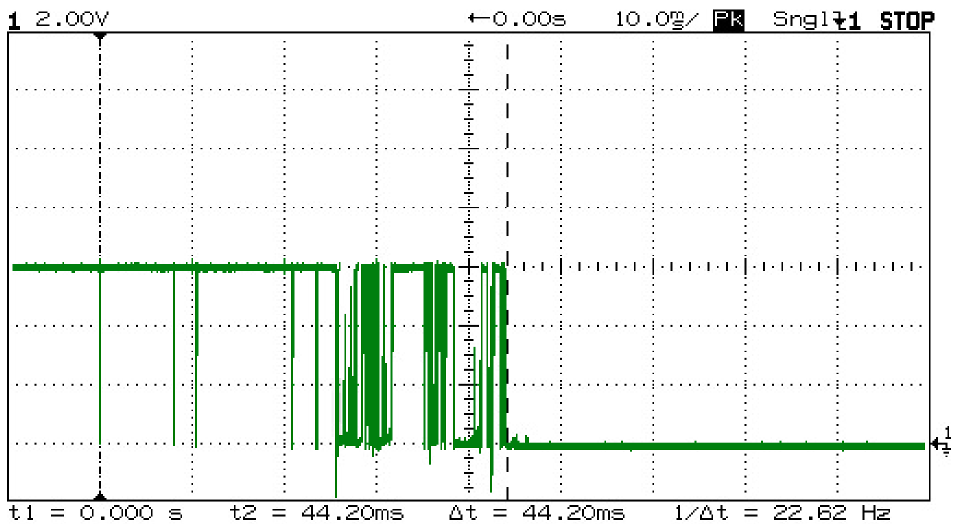
  - Configuration might also be per-bank or per pin
    - I/O pins with individual interrupt allow for modular and uncoupled software design

  - Interrupts will be treated in a later module. For now, they introduce the challenge of dealing with bouncing digital input signals

Inputs

- Bouncing digital inputs
  - Causes
    - Mechanical
    - Electrical
  - Consequence
    - Defective falling and rising edges
  - Input devices (e.g. switches) may have datasheets describing bouncing characteristics





Georgia Institute of Technology

Inputs

- Debouncing digital inputs
  - Manages defective signal edges
  - Makes use of multiple readings (data samples)
  - After several consistent samples, notify the system about input state change
  - Example pseudo-code

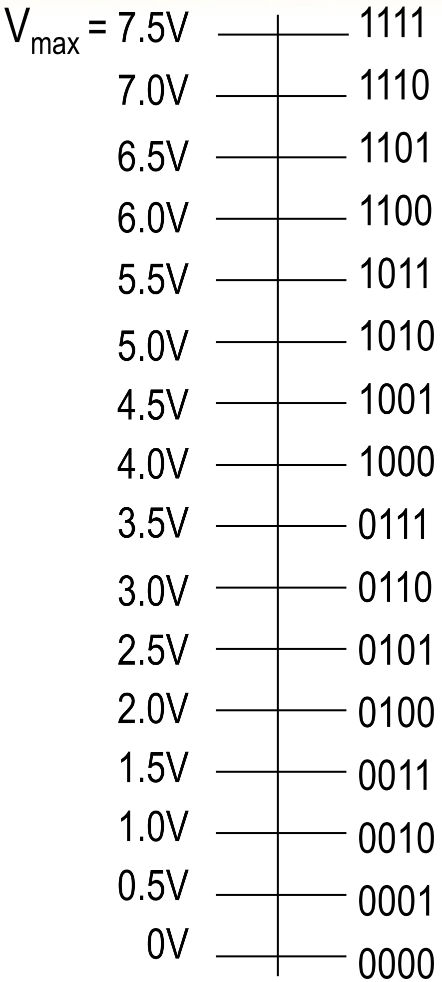
```
main loop:
  if time to read button,
    read button
    if button is released
      set button set to false
      set delay period
    if time to toggle the LED
      toggle LED
    repeat
```

```
read button:
  if raw data equals debounced value
    reset the counter
  else
    decrement the counter
    if counter is zero,
      set button value to raw data
      set changed to true
      reset the counter
```


Georgia Institute of Technology

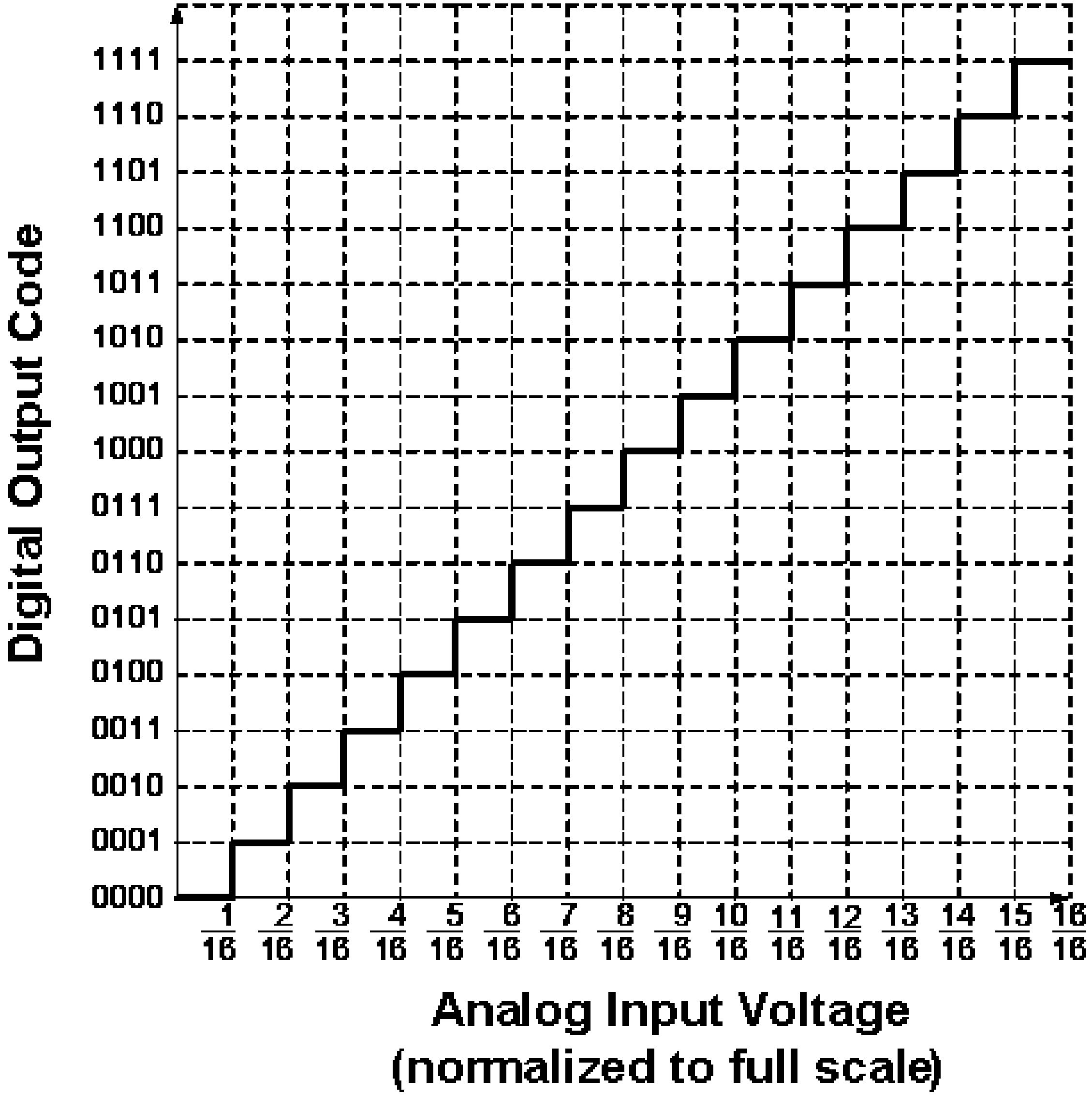
Inputs

- Analog inputs
  - Voltages
    - Minimum 0 V
    - Maximum Vmax
  - Digital encoding
    - a/Vmax = d/M
      - a: analog value
      - Vmax: maximum input voltage
      - d: digital encoding
      - M: steps in digital scale
        - $M = 2^n-1$
      - n: number of bits in digital encoding
    - Resolution: largest voltage change required to shift one bit



Inputs

- Analog Inputs
  - Ideal Transfer Curve of a 4-bit ADC

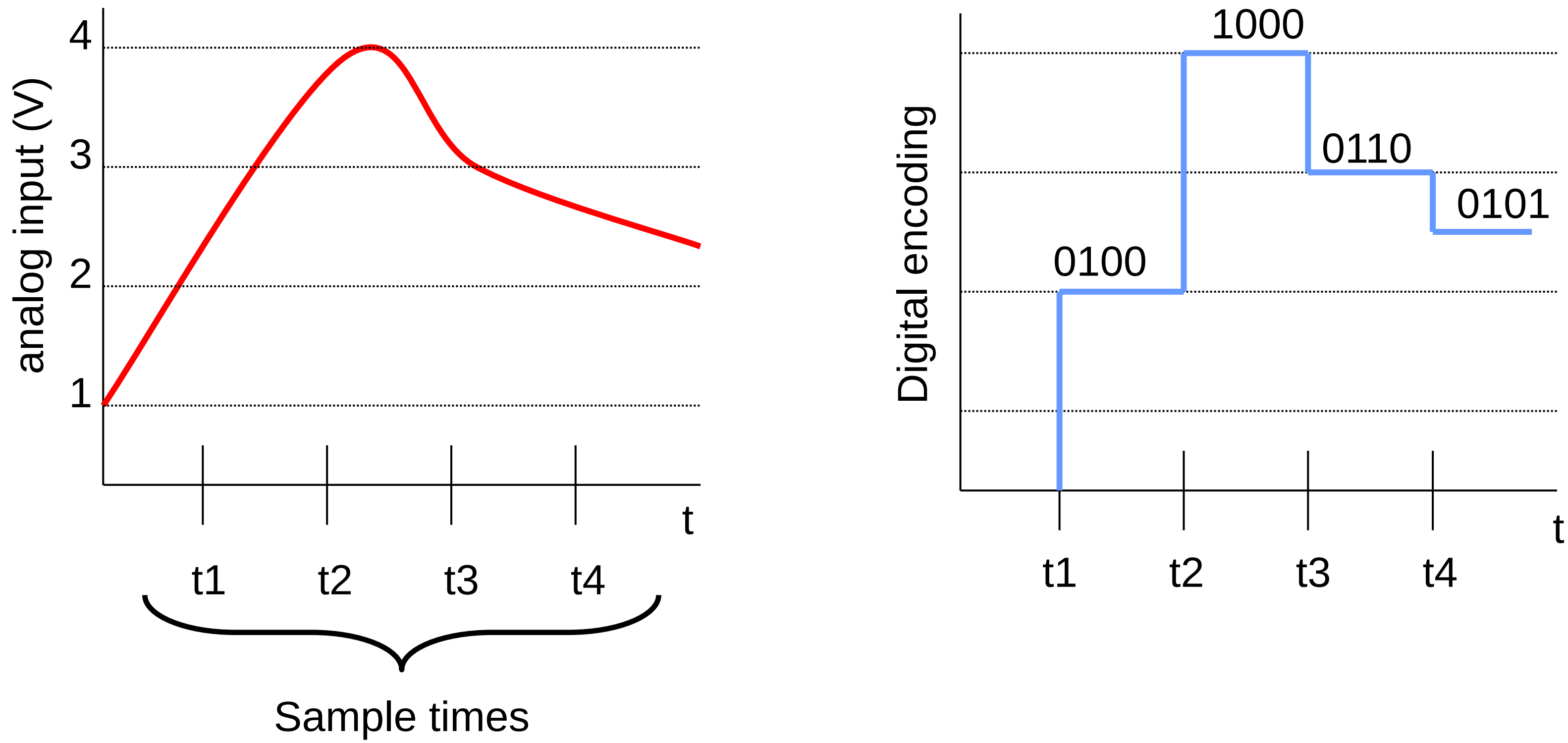


From the Communications Museum of Macao

Georgia Institute of Technology

Inputs

- Analog to digital conversion
  - Example



In the Next Modules…

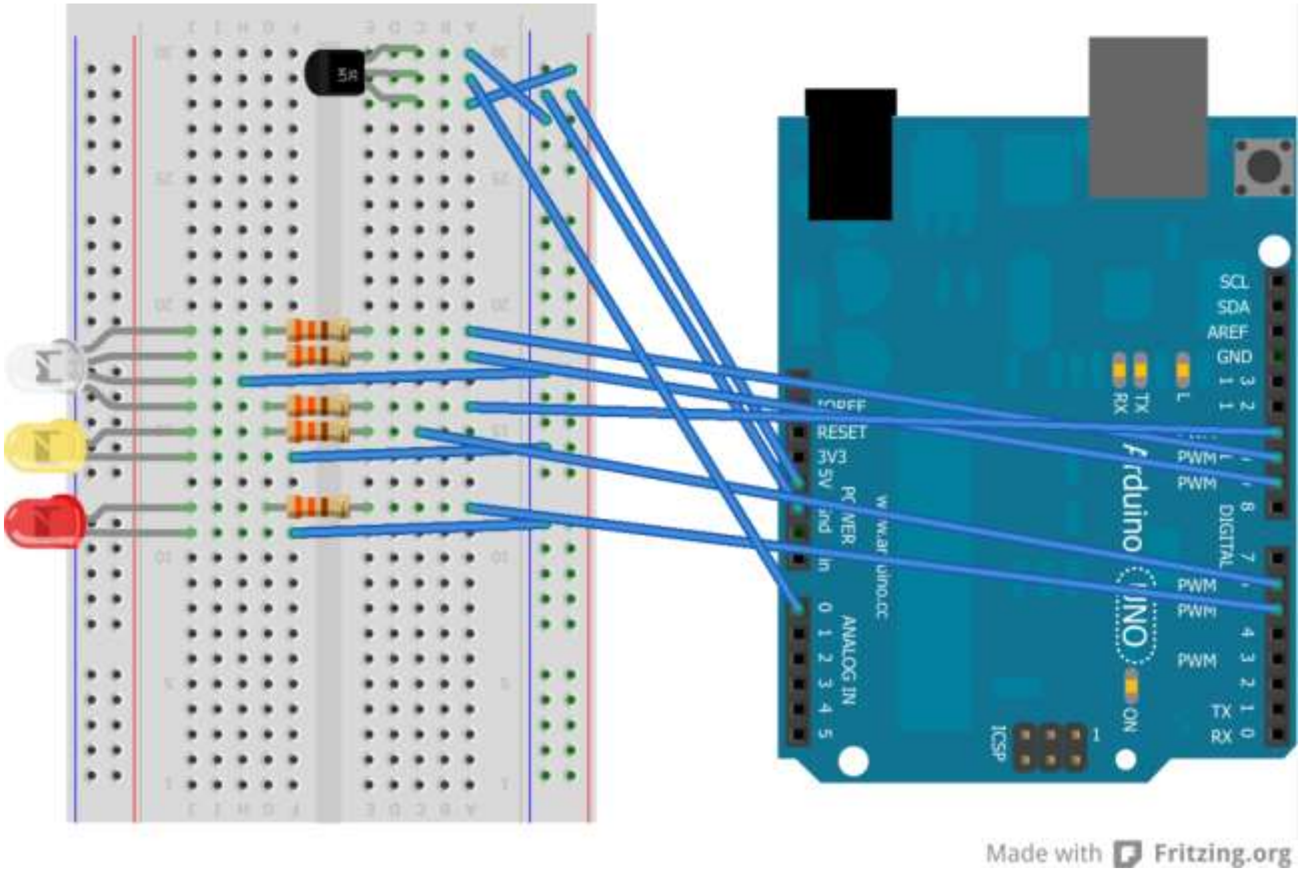
- Handling Uncertainty
  - Alternating LED activation
    - Dependency injection
  - Clocks and timers

- Scheduling
  - Communication between tasks
  - State machines
  - Interrupts
  - Watchdog

- Communication with peripherals
- Managing resource scarcity
- Reducing power consumption (from the s/w side)

Fuzzy Temperature Indicator

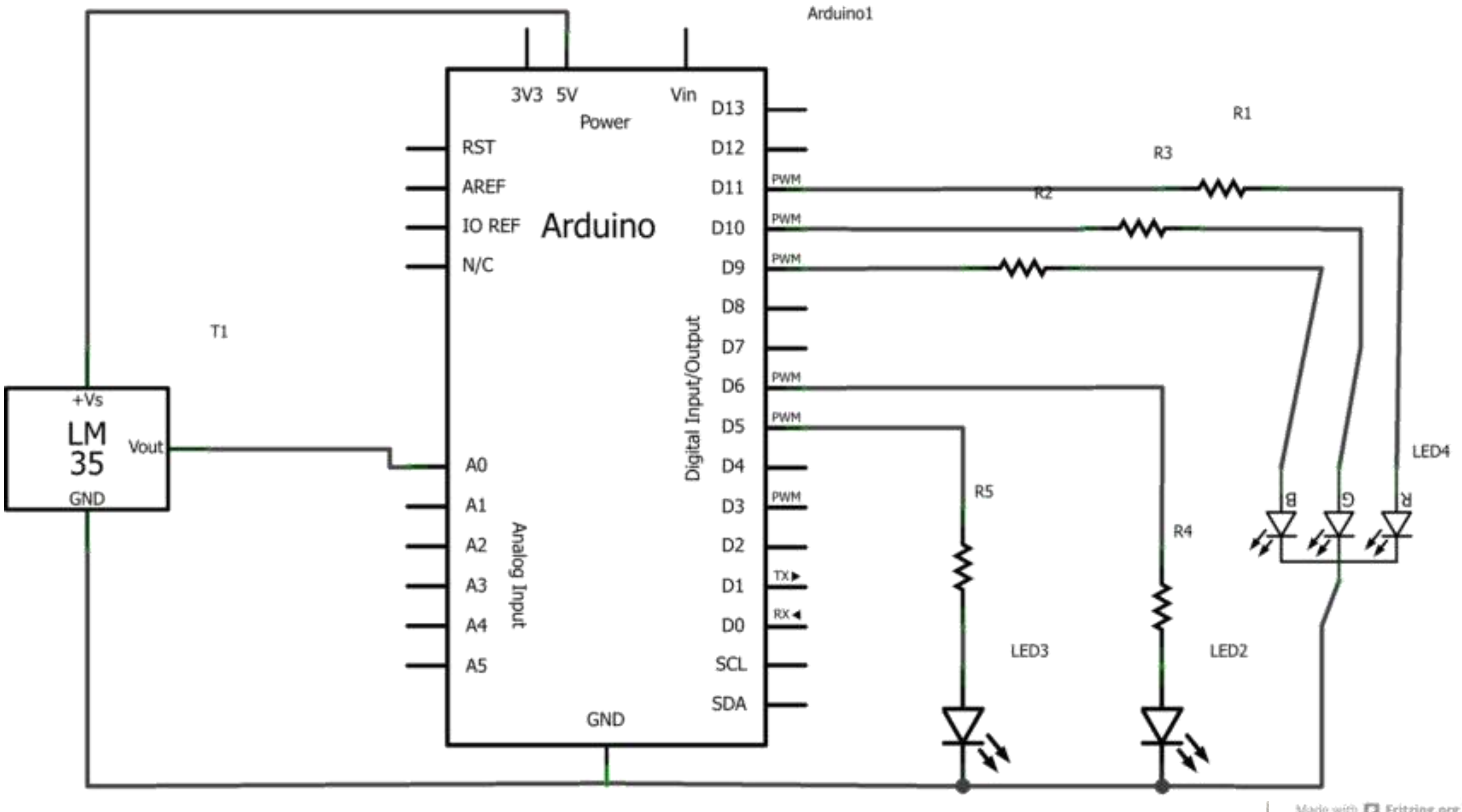
- Circuit on the bread board in Fritzing



Georgia Institute of Technology

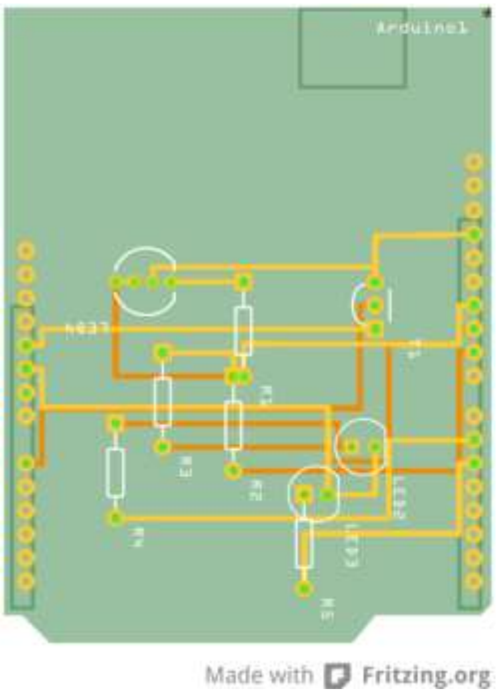
Fuzzy Temperature Indicator

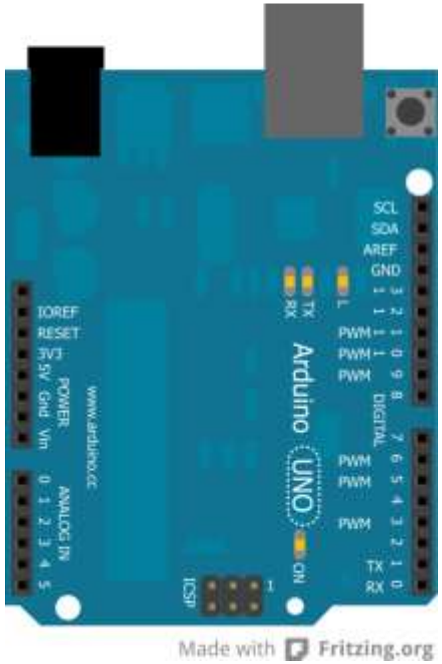
- Schematic in Fritzing

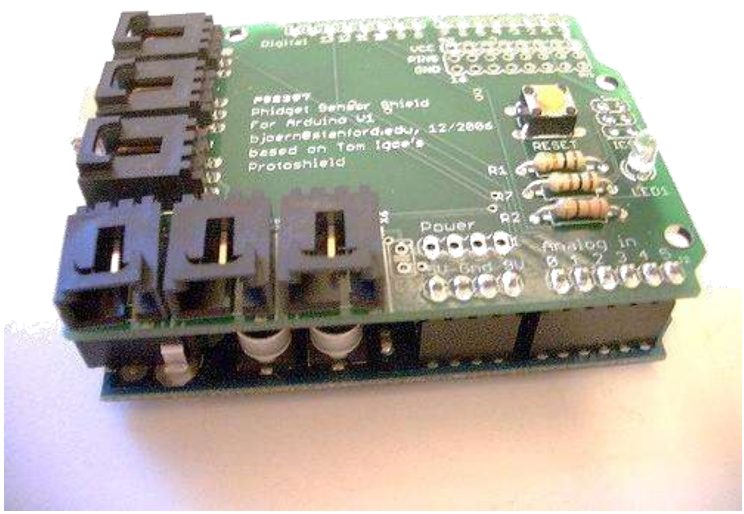


Fuzzy Temperature Indicator

- Printed circuit board manufacturing from Fritzing







Fuzzy Temperature Indicator

- Pseudo-code

```
variables declaration
pin setup
main:
for i<98
  take sensor voltage value
  convert to °F and °C
  wait for a small time
calculate mean temperatures
setRGB LED
if averageF < 64
  blink red and board LEDs
elseif averageF > 70
  blink yellow and board LEDs
else
  blink board LED
```

```
setRGB:
variables declaration
trapMF value for RED
trapMF value for GREEN
trapMF value for BLUE
set PWM for RED GREEN and BLUE
```

```
trapMF:
variable declaration
output=constraint(map(arguments1))
  -constraint(map(arguments2))
```