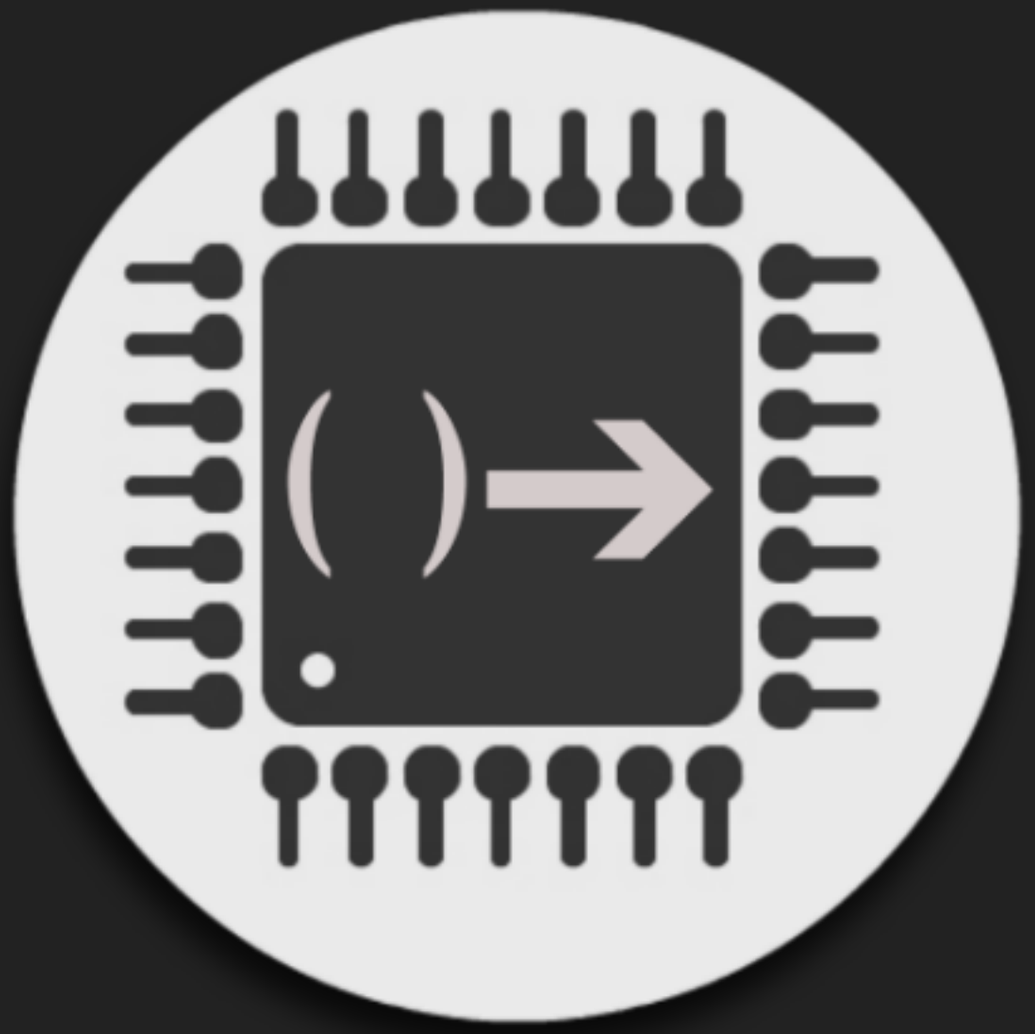
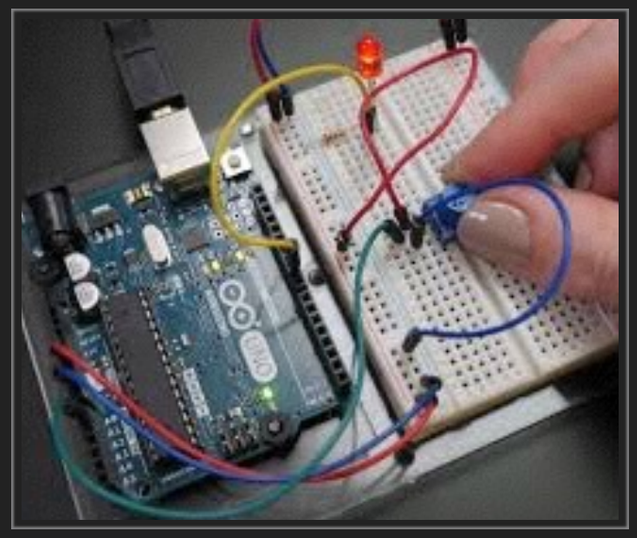
Carl Peto

10th August 2017



SWIFT FOR ARDUINO

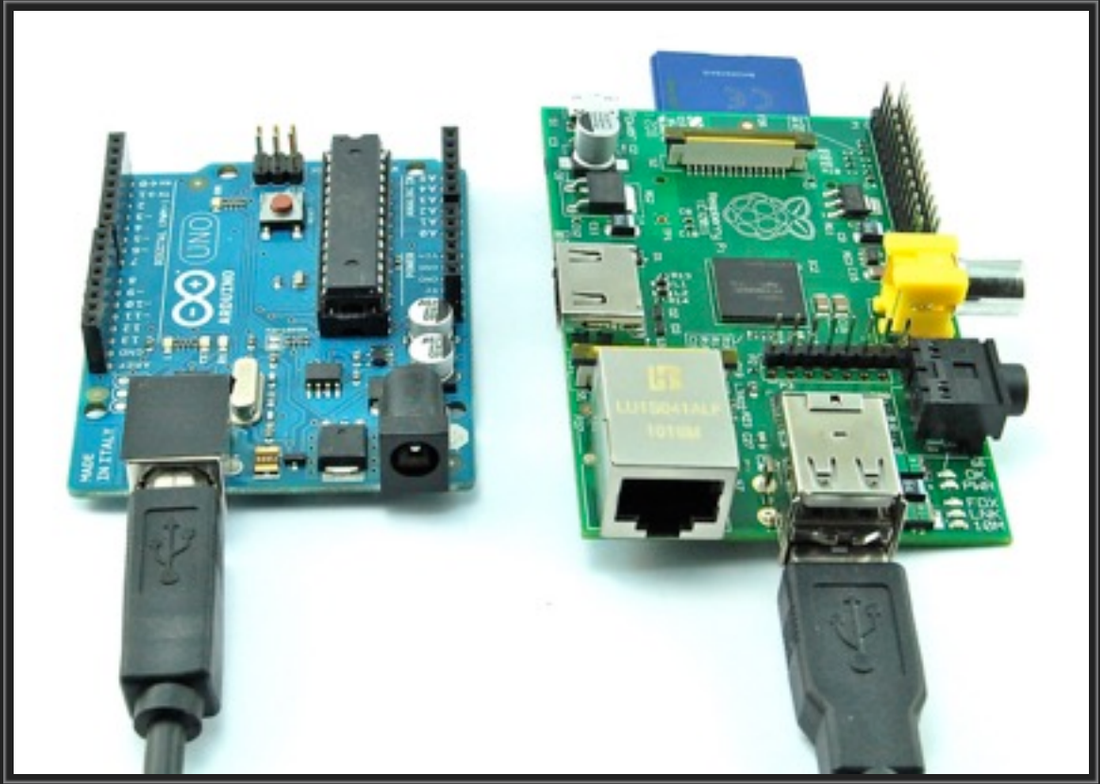
ARDUINO



Microcontrollers are a small, cheap, multi purpose IoT computer in a box, with built in interfaces in the package, all in one chip. They can be bought and programmed in bulk once you have worked out your design. They come in thousands of different variants from many different manufacturers, Atmel, PIC, ARM, etc.

For example, the Atmega 328p chip that is the core of the Arduino UNO has flash memory for the program, a small amount of working RAM, EEPROM for configuration settings, ADC, PWM, I2C, UART, precise timers, interrupts and much more.

ARDUINO



A raspberry pi makes a good webserver and can turn on a vacuum cleaner, a microcontroller can make a **device**. It can control the motors, servos and lights on a hoover in subtle ways, PWM, create waveforms, see your code create a shape on an oscilloscope! Responses with precise microsecond timing. e.g. how do you precisely control the lamps on a set of traffic lights?

Can be cheap (~$1), made to scale, tiny.

- What makes them hard to use?

  Single thread, no memory protection, no operating system, no keyboard or screen. Arduino UNO only has 2kb of memory!!

  Watch every assignment, watch every buffer for overrun.

  The standard IDEs all use C or C++ or simple variants and are relatively unforgiving.

  I found you have to learn everything the hard way.

What is Arduino?

"Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It's intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments.

"Arduino can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors, and other actuators...

"The boards can be built by hand or purchased preassembled... The hardware reference designs (CAD files) are available under an open-source license, you are free to adapt them to your needs. [There are multiple, free, open source compilers and IDEs.]"

Before Arduino, writing microcontroller code was highly esoteric. Arduino have brought the avr-gcc/libc/binutils toolchain and the wiring library to make a C style platform.

BUT it's still terribly easy to create buffer overruns, attempt to use the UART pins, etc.

- Why is Swift a good language for Arduino?

  1. Swift is built to be safe = reduce your mistakes

  2. Type safe, no nulls, you can write framework that limits people's mistakes

  3. Easy to read, expressive, beautiful, exciting, cross platform and built for the future

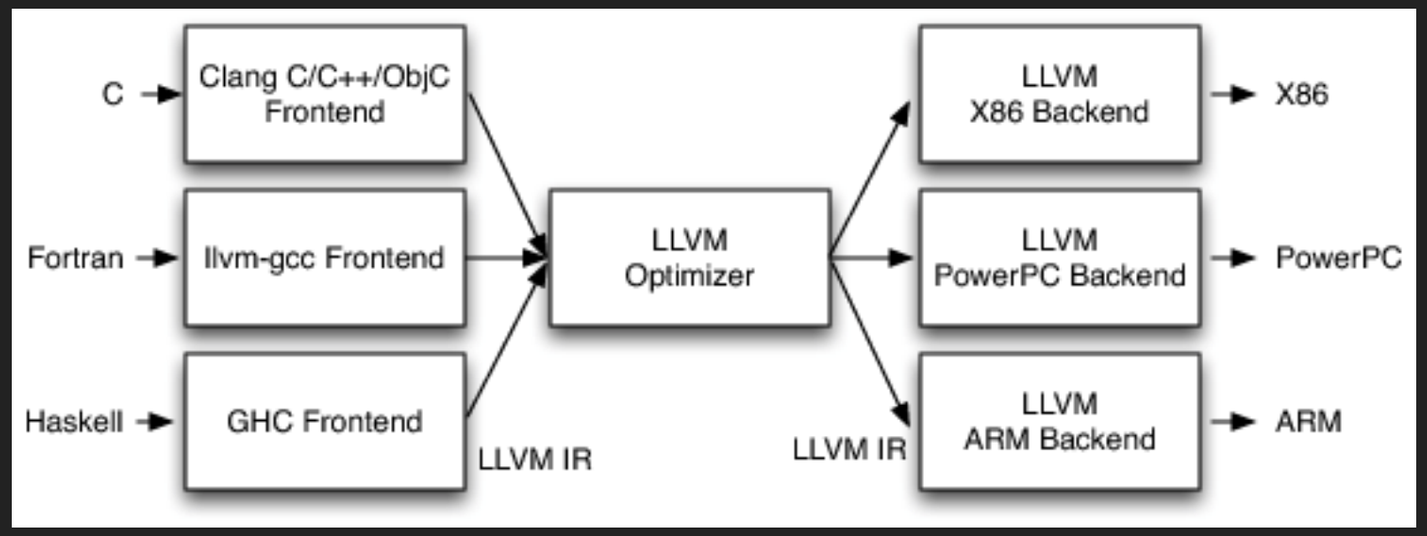
Do an analog read, wait for the results but keep processing, deliver the value as a callback in an interrupt handler? In classic Arduino C that's not easy. In Swift for Arduino...

```
analogRead(pin: .A0)
{ temperatureGuageVoltage in

  SPI.write(temperatureGuageVoltage)

}
```

Swift is built on LLVM.



LLVM parses and compiles “front end” languages like Swift, C, C++, Rust to an intermediate representation. This can exist as one of three states, LLVM IR, bitcode or (most usually) an in memory representation.

There is an experimental back end for AVR.

I hooked them up, got it working, patched the back end, created a basic library and built a simple IDE as a mac app. And it works!

PROJECTS

- Traffic light:

```
import AVR

let red: UInt8 = 3
let amber: UInt8 = 10
let green: UInt8 = 9

pinMode(pin:red, mode:OUTPUT)
pinMode(pin:amber, mode:OUTPUT)
pinMode(pin:green, mode:OUTPUT)

// flash an LED with a certain period for a length of time, both in 100ths of a second
func showFlasher(pin: UInt8, durationCs: UInt16, periodCs: UInt8) {
  // … snip…
}

while(true) {
   // stop = red
   analogWrite(pin: red, value: 255)
   analogWrite(pin: amber, value: 0)
   analogWrite(pin: green, value: 0)

   delay(milliseconds: 5000)

   // prepare to go = flashing amber
   analogWrite(pin: red, value: 0)
   analogWrite(pin: amber, value: 255)
   analogWrite(pin: green, value: 0)
   showFlasher(pin: amber, durationCs: 500, periodCs: 70)

   // go = green
   analogWrite(pin: red, value: 0)
   analogWrite(pin: amber, value: 0)
   analogWrite(pin: green, value: 255)

   delay(milliseconds: 5000)

   // prepare to stop = amber
   analogWrite(pin: red, value: 0)
   analogWrite(pin: amber, value: 255)
   analogWrite(pin: green, value: 0)

   delay(milliseconds: 5000)
}
```

See the actual source code in https://github.com/swiftarduinoexamples/examples

The bit I skipped shows how it's still experimental and how careful you have to be…

```
func showFlasher(pin: UInt8, durationCs: UInt16, periodCs: UInt8) {
  var brightness: UInt8 = 0
  let brightnessStep: UInt8 = 255 / periodCs
  var counter: UInt16 = 0 // counts up in cs until we hit durationCs, then finish
  var waxing: Bool = true

  while counter < durationCs {
    analogWrite(pin: pin, value: brightness)

    if waxing {
      brightness = brightness &+ brightnessStep
    } else {
      brightness = brightness &- brightnessStep
    }

    if brightness > 230 {
      waxing = false
    }

    if brightness < 10 {
      waxing = true
    }

    counter = counter &+ 1
    delay(milliseconds: 10)
  }
}
```

Note: I am explicitly defining int variable types as UInt8 or UInt16 as this makes sense. If you don't do that the swift compiler assumes it's a 64 bit int, which makes no sense and is very wasteful on an Arduino. Also pay attention to overflows…

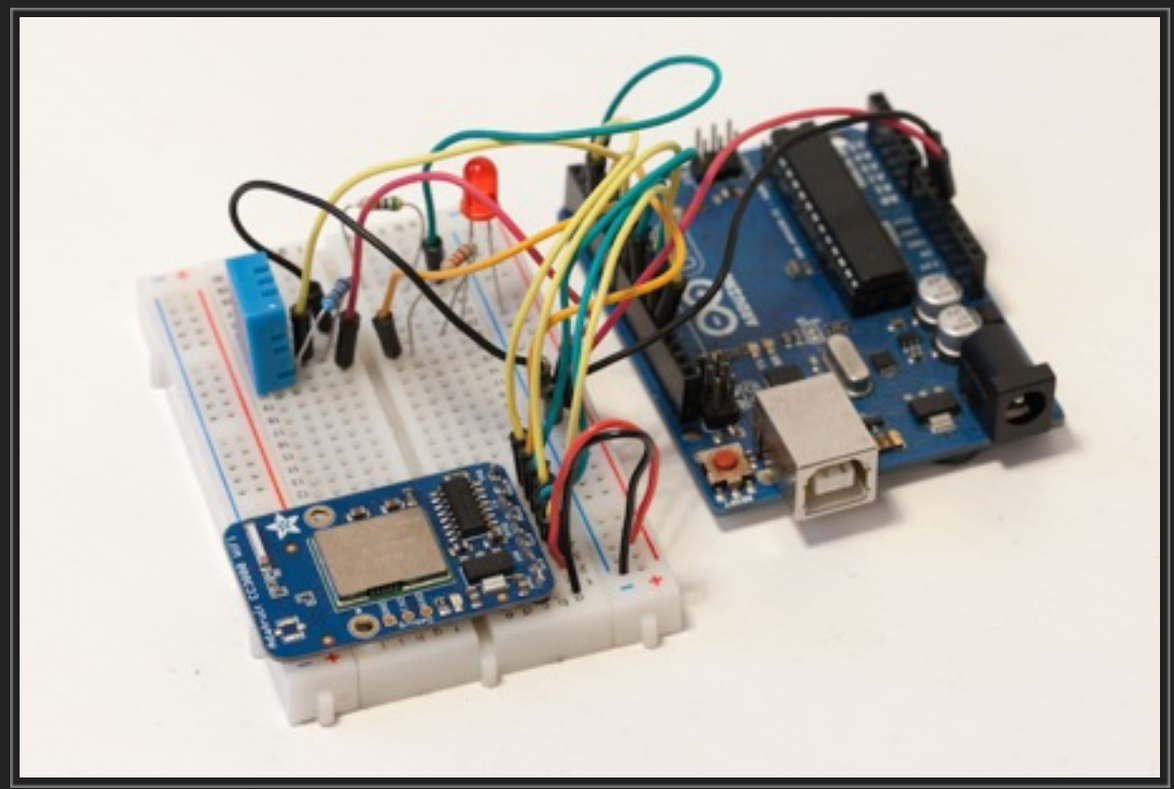
Swift is built as a safer, type safe language, minimising undefined behaviour (see wwdc video this year). When arithmetic would overflow, this is trapped by an abort in normal Swift. That's good on a regular OS, bad on a micro controller. My IDE attempts to warn you about this but at the moment it's not possible to work around these limitations.

Upshot: try to avoid aborts, define types carefully, don't rely on type inference (yet), don't force cast to a smaller int type, unsigned, etc. Use &+ or &- instead of + or -!

Understanding undefined behaviour, Apple WWDC 2017 session 407… https://developer.apple.com/videos/play/wwdc2017/407/

- Flashing LED light:

The standard “hello world” of micro controller code... flash an LED on and off regularly.



I will attempt a live-coding demo!

FIND OUT MORE...

swiftforarduino

swiftforarduino

website: https://swiftforarduino.wixsite.com/home

Acknowledgements:

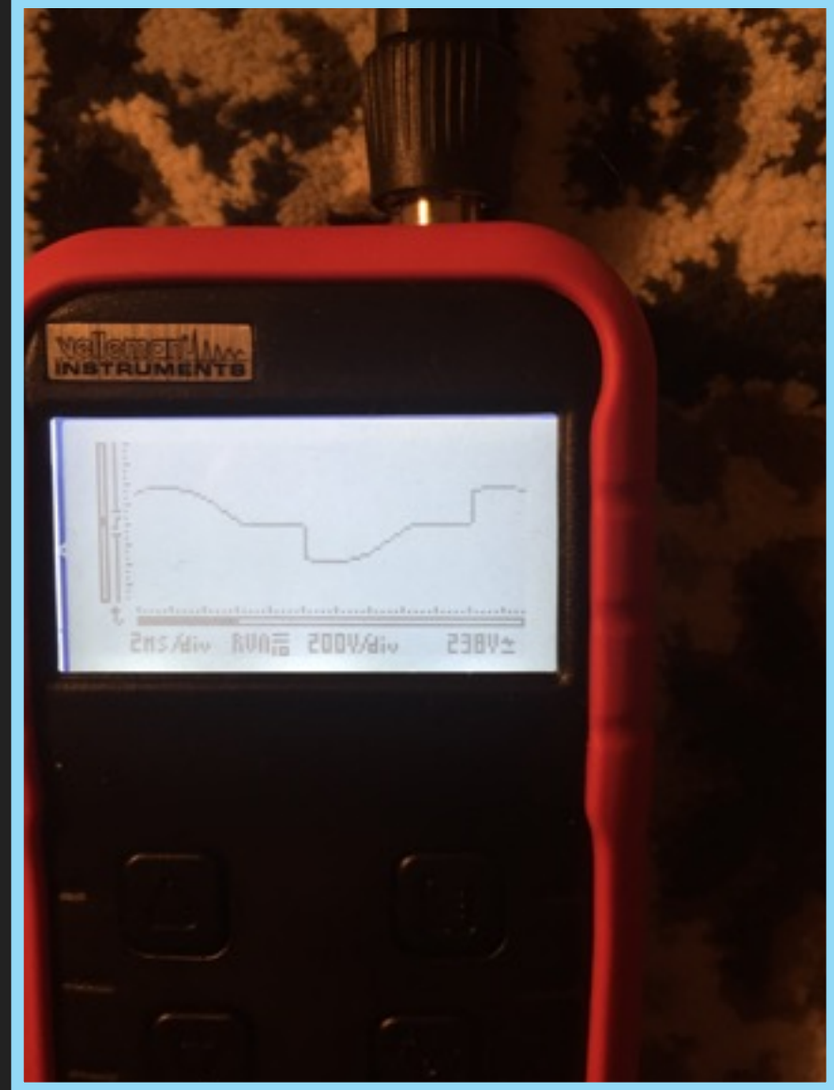
Chris Lattner and many others - LLVM

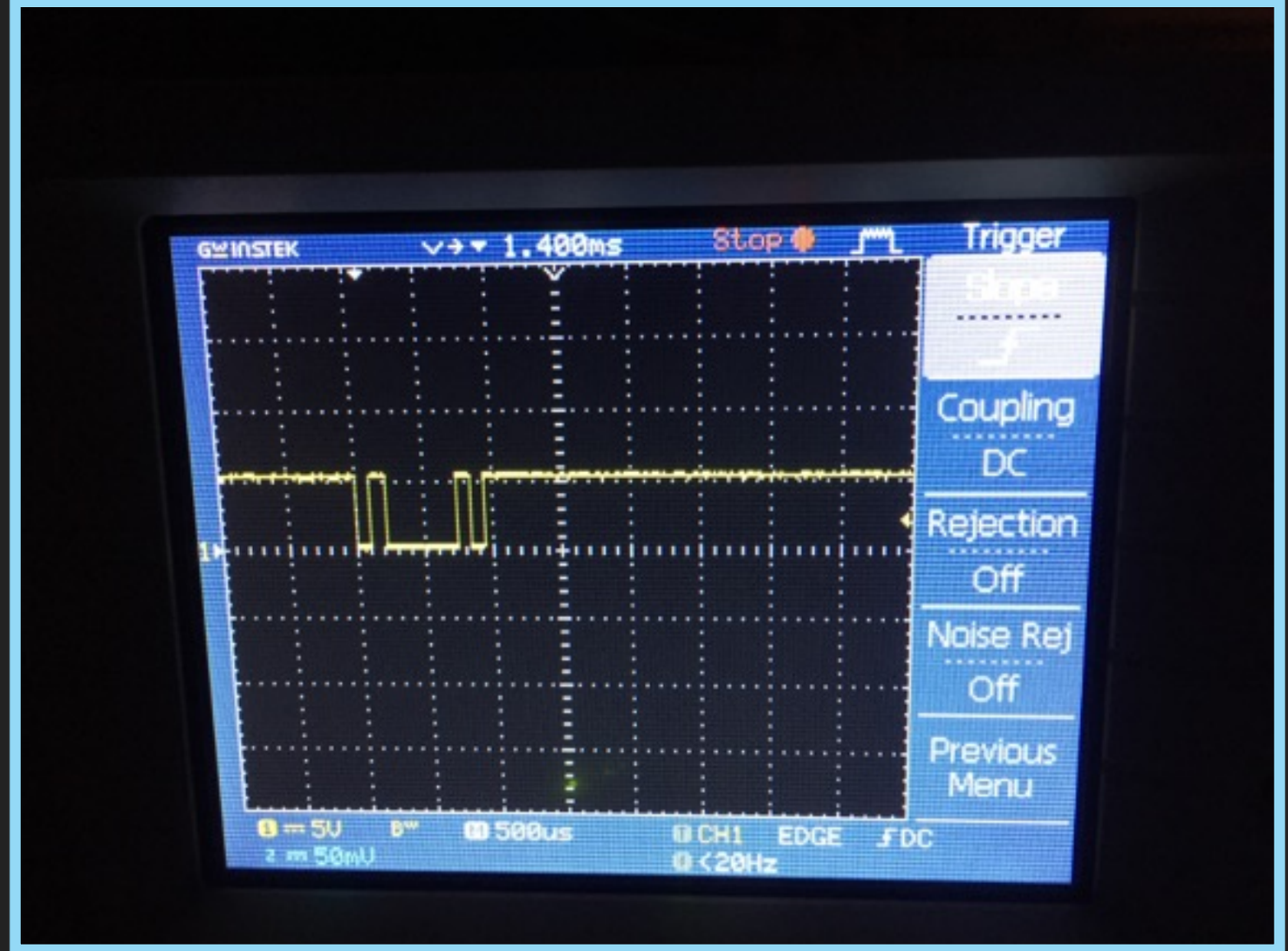
Dylan McKay, Josef Eisl, John Myers,

Eric Weddington, Borja Ferrer - LLVM AVR

Ellie Kay - slides, Facebook page, Twitter,

website, design, logos


INSTRUMENTS
2ms/div
200V/div
238V~


GW INSTEK
1.400ms
Stop
Trigger
Coupling
DC
Rejection
Off
Noise Rej
Off
Previous
Menu
5V
500us
CH1
EDGE
DC
50mV
<20Hz