uC101: Introduction to Microcontrollers / Interfacing with the real world

Josh Johnson

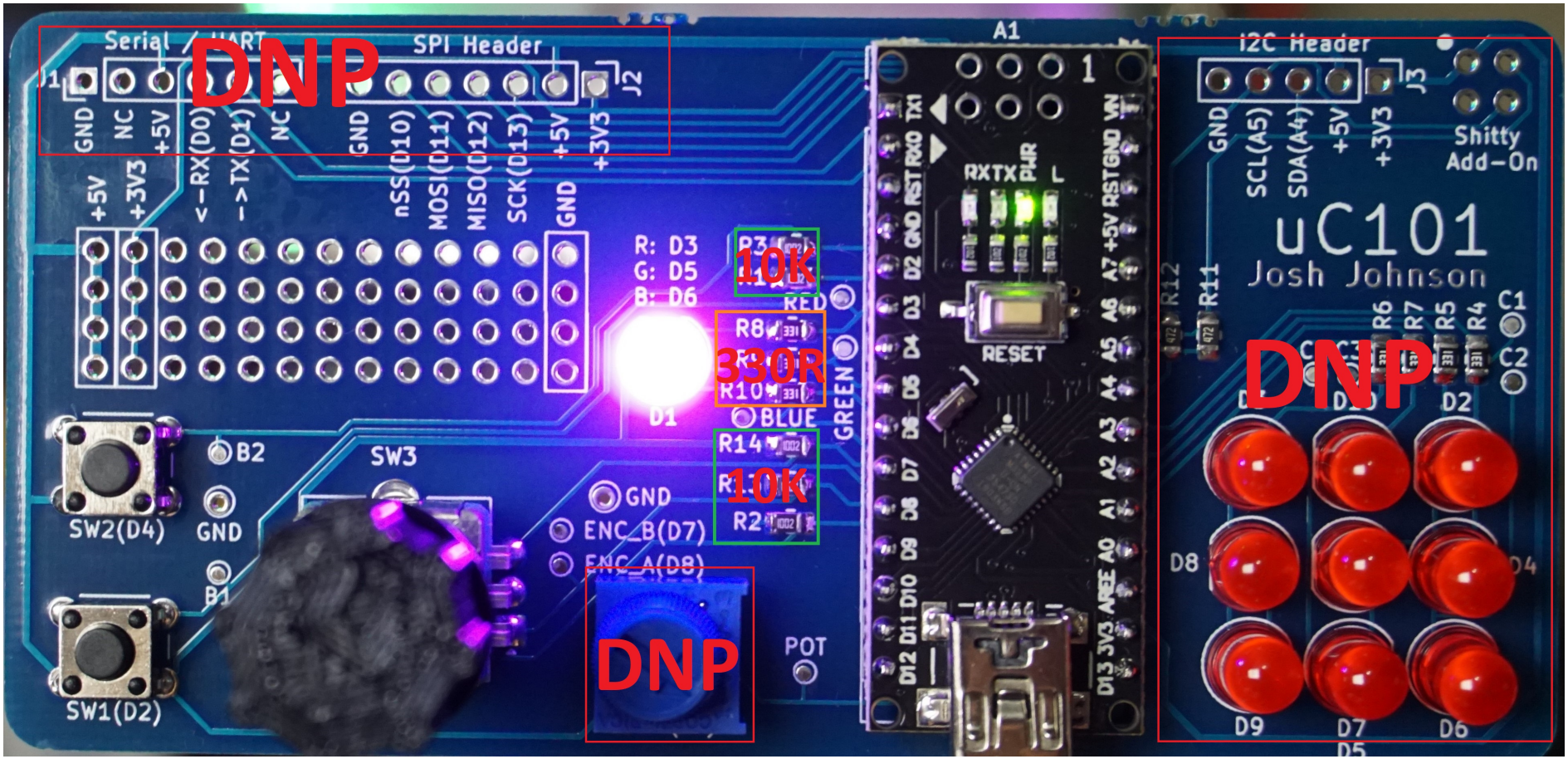
13/5/2019

Overview

- Assembly of Hardware / Installation of Software
- Microcontroller 101
- Tools
- Bit Math
- Demos
  - Blink
  - Button
  - RGB LED (PWM)
  - Rotary Encoder
  - Charlieplexing

Project Files: `github.com/joshajohnson/CBRhardware`

Assembly of Hardware

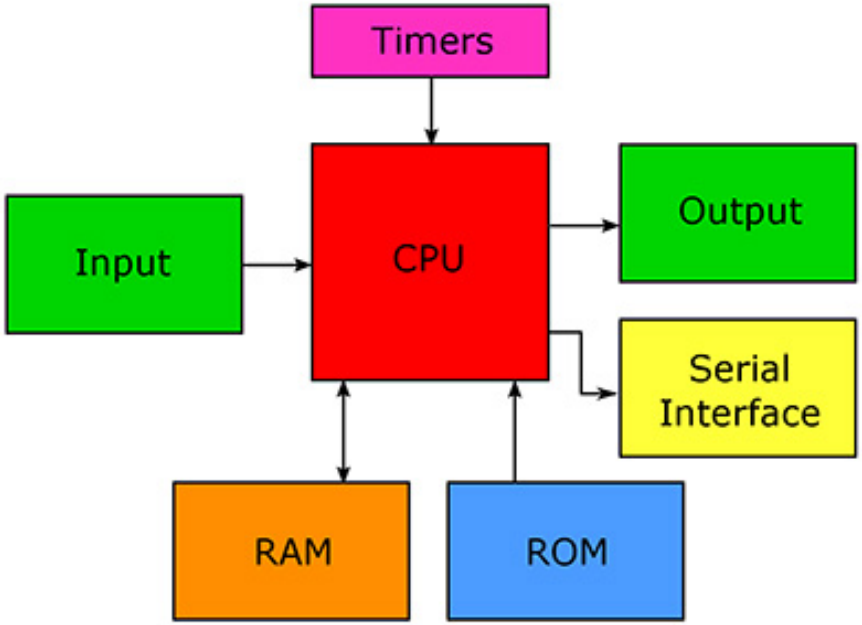


Software Installation!

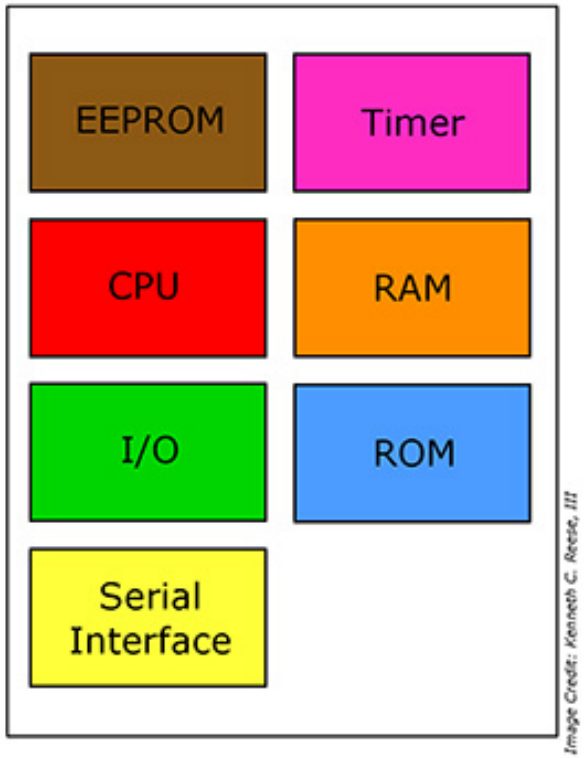
Install Arduino IDE
Copy the uC101Library folder to `Documents/Arduino/libraries`
Open `Firmware/blink/blink.ino`
In Arduino IDE:
`Tools->Board:-> Arduino Nano`
`Tools->Processor:-> ATmega328p`
`Tools->Port:-> $comPort`
Run blink.ino and confirm that the onboard LED is blinking

What is a microcontroller?

Microprocesser: CPU
and several supporting chips.



Microcontroller: CPU
on a single chip.



Microcontroller 101

Common Options

8 bit

- ATtiny
- ATmega (Atmel / Microchip)
- PIC (Microchip)

16 bit

- MSP430 (TI)

32 bit

- STM32 (ST)
- SAM (Atmel/Microchip)
- nRF5x (Nordic Semi)
- ESP8266/32 (Espressif)
- CCxxxx (TI)
- LPCxxxx (NXP)
- PIC32 (Microchip)

32 bit ARM cores

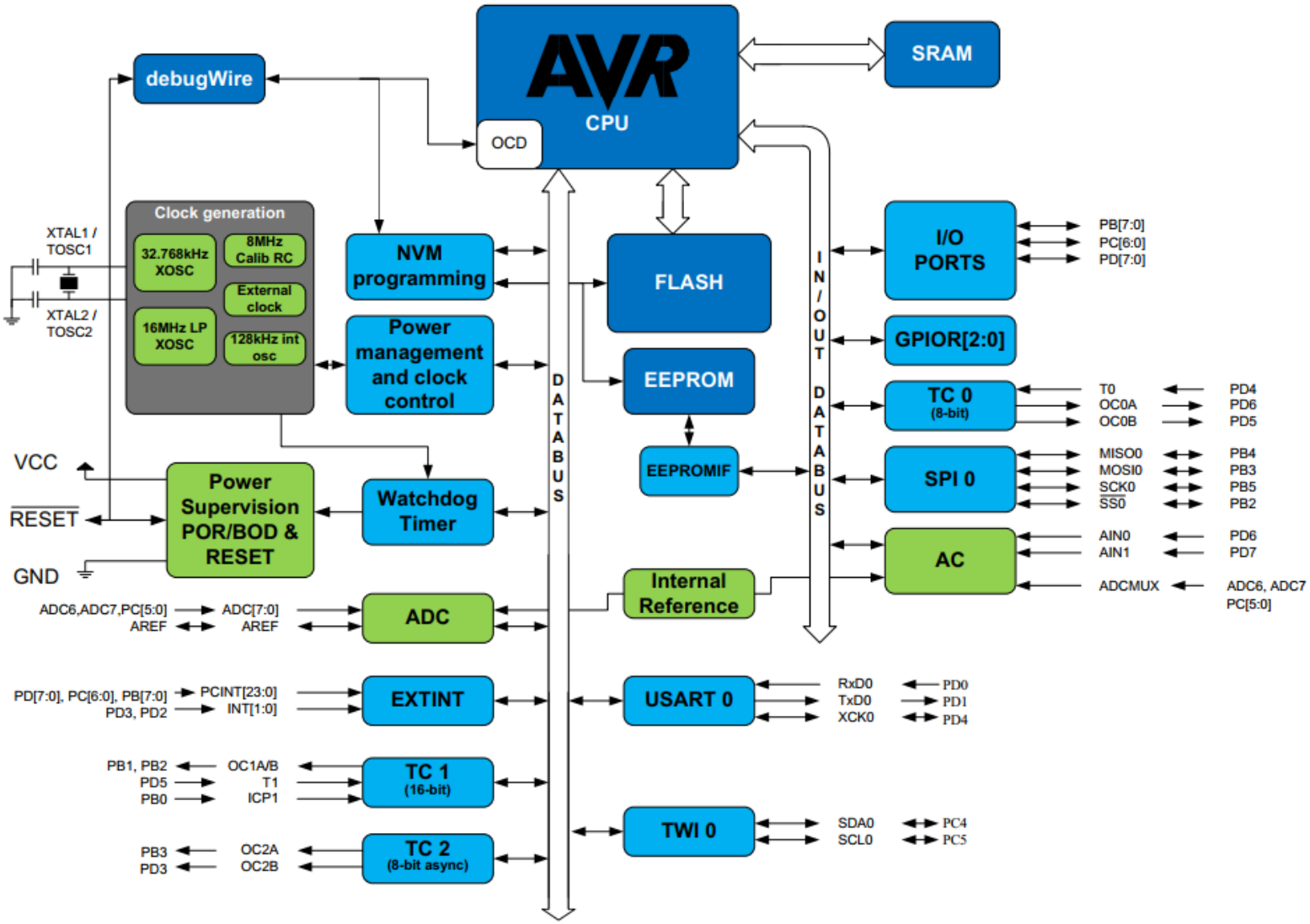
- Cortex-M0/M0+
- Cortex-M1 (FPGA only)
- Cortex-M3
- Cortex-M4 (M3 + DSP + FPU)
- ...

Microcontroller 101

How to choose?

- Compute power
  - 8 bit vs 32 bit
  - DSP / FPU
- Peripherals
  - Wireless
    - WiFi
    - Bluetooth
    - LoRa
    - Cellular
  - USB
  - ADC
  - Ethernet
  - CAN
  - Number of SPI/UART/I2C/Timers

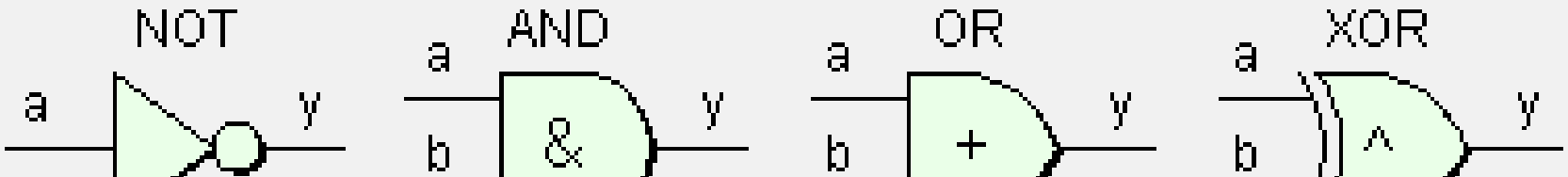
ATmega328p Architecture



Tools

- Oscilloscope
  - + High quality time domain measurements ($\geq$8 bits, $\geq$1 GSPS)
  - - Expensive, short record time
- Logic Analyser
  - + Easy to use UI, long sample time / depth
  - - 1 bit*, low sample rate (25 - 100 - $\geq$500 MSPS)
  - * some logic analysers have limited analog capabilities
- Debugger
  - Program and debug device
- Multimeter
  - Voltage and resistance measurements
- Development Board
  - Whilst not a tool, extremely helpful whilst bringing up firmware
  - Helpful when developing code - the issue isn't (typically) the hardware!

Bitwise Operators



a	y
0	1
1	0

a	b	y
0	0	0
0	1	0
1	0	0
1	1	1

a	b	y
0	0	0
0	1	1
1	0	1
1	1	1

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

```
y = ~a      y = a & b      y = a | b      y = a ^ b
          NOT y = a && b
```

Binary Operations

Binary Numbers

```
Binary:   0  1  0  0  1  0  1  1
Weight: 128 64 32 16  8  4  2  1
Base 2  (Binary): 0b01001011
Base 10 (Decimal): 75
Base 16 (Hexadecimal): 0x4B
```

Bit Shifting

```
a = 5;          // binary: 00000101
b = a << 3;     // binary: 00101000, 40 in decimal
c = b >> 3;     // binary: 00000101, back to 5 like we started
```

Putting it all together

```
x = 1;        // binary: 00000001
x <<= 3;      // binary: 00001000
x |= 3;       // binary: 00001011 - 3 is 11 in binary
x &= 1;       // binary: 00000001
```

Setting Registers

Setting a bit in a register

```
REG = REG | (1 << x);
REG |= (1<<x);
```

Clearing a bit in a register

```
REG = REG & ~(1 << x);
REG &= ~(1<<x);
```

Toggling a bit in a register

```
REG = REG ^ (1 << x);
REG ^= (1<<x);
```

Toggling multiple bits in a register

```
REG = REG ^ ((1 << x) | (1 << y));
REG ^= (1<<x) | (1<<y);
```

AVR specific macro to bitshift

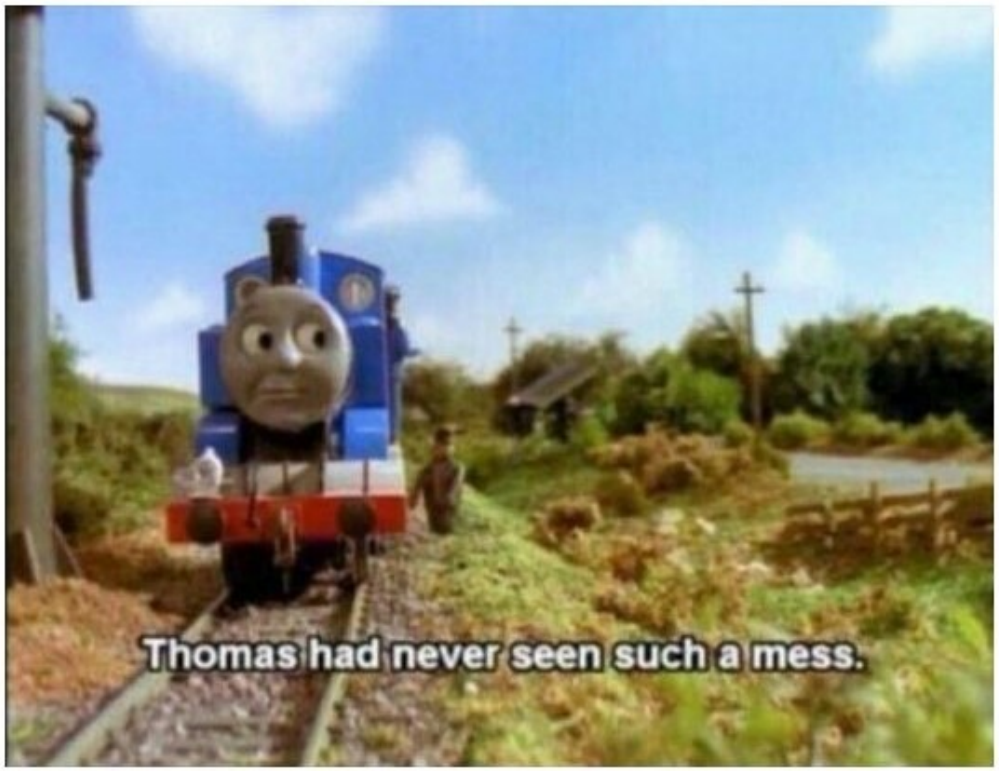
```
(1<<x) == _BV(x)
REG = (1<<x) | (1<<y);
REG = _BV(x) | _BV(y);
```

What is the difference between the above and the first example (other than two variables)?

Demo Time!

Me : *Explain my code to colleague*

Colleague:



- I'm a hardware person, not a software person
- Solder is my preferred programming language
- I know enough to be dangerous, nothing more
- More experienced folks, please jump in and correct me / answer questions I can't / point out bad practices
- I'm here to learn like everyone else

blink.ino

- Default blink
- Register level blink
- Nicer blink
- Even nicer blink
- Blink without delay
- Size comparison
- Speed comparison

button.ino

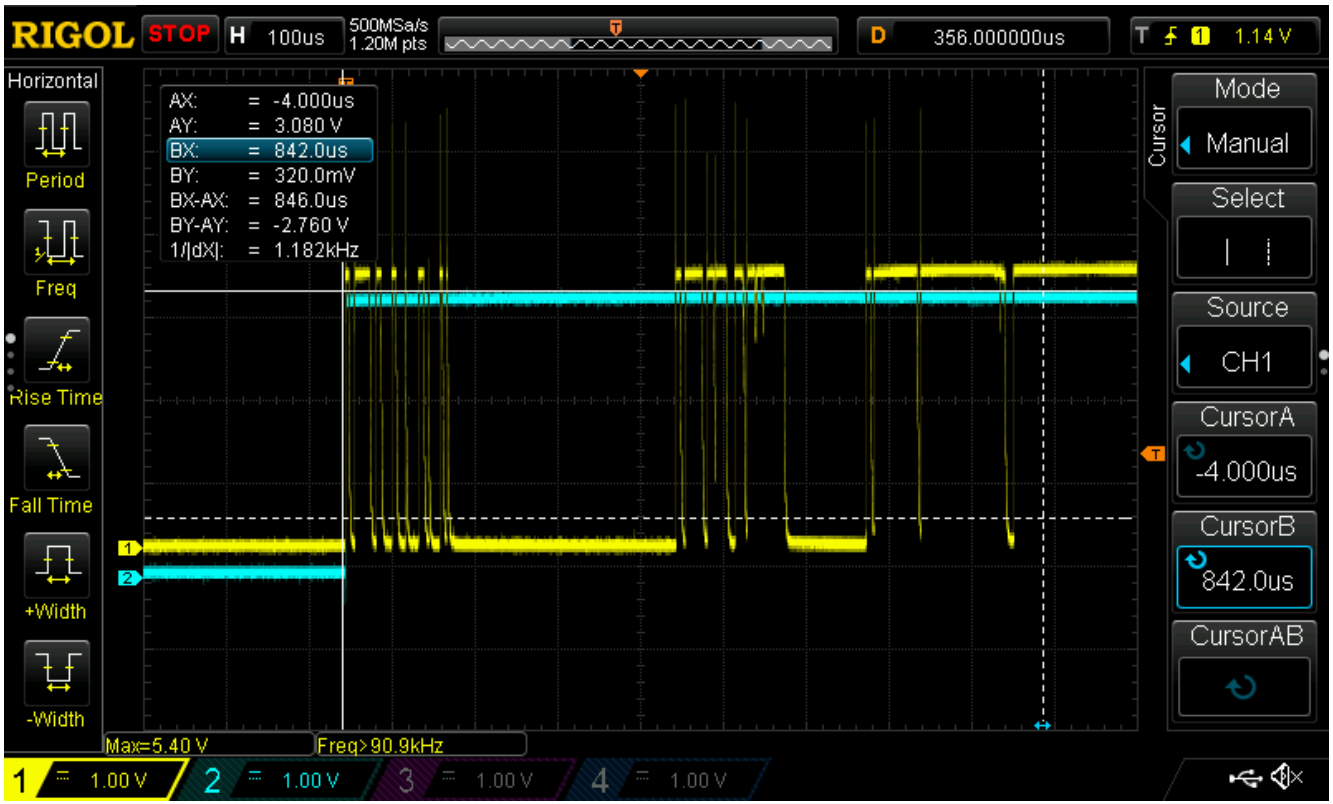
- Toggle on press
- Toggle on press with delay
- Button with interrupt
- Debounced button

Button Bounce

- What is button bounce?
- How to fix it?
  - Hardware - RC Circuit
  - Hardware - RC + Schmitt trigger
  - Software - Button polling

Button Bounce

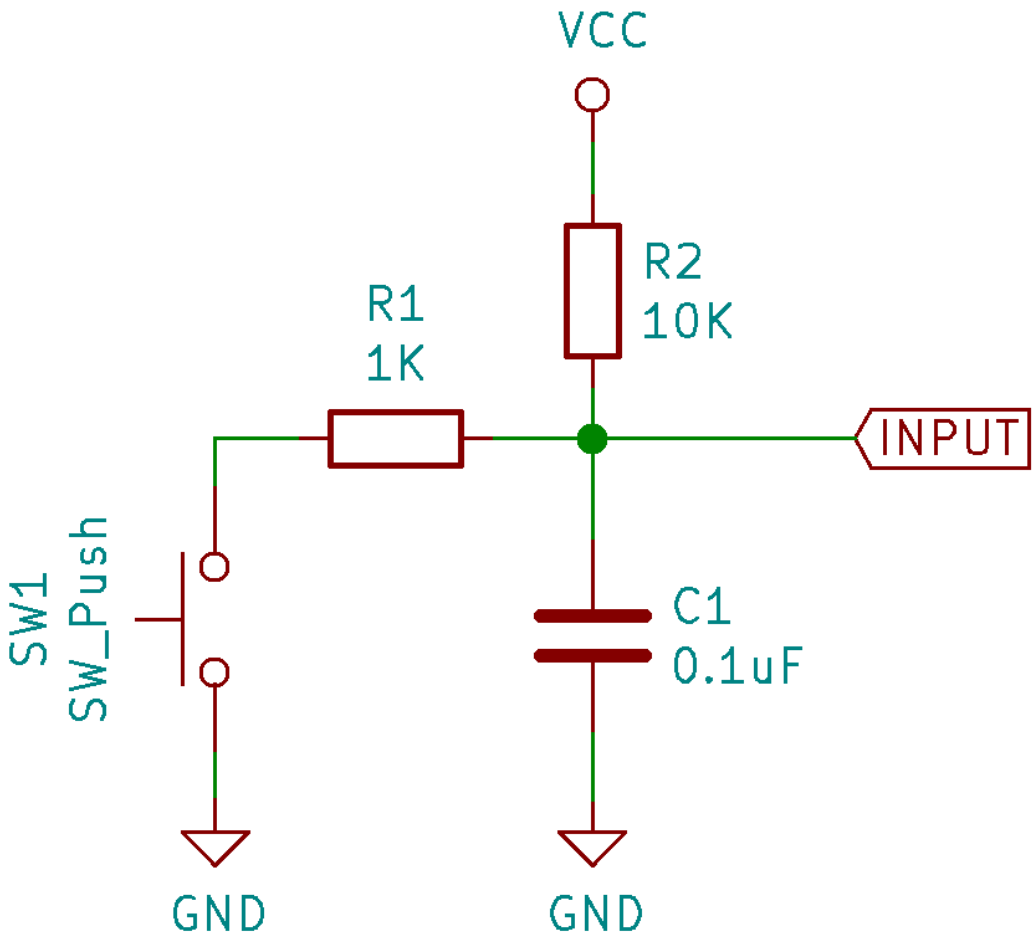
- What is button bounce?



Yellow = actual button, Blue = 'ideal button'

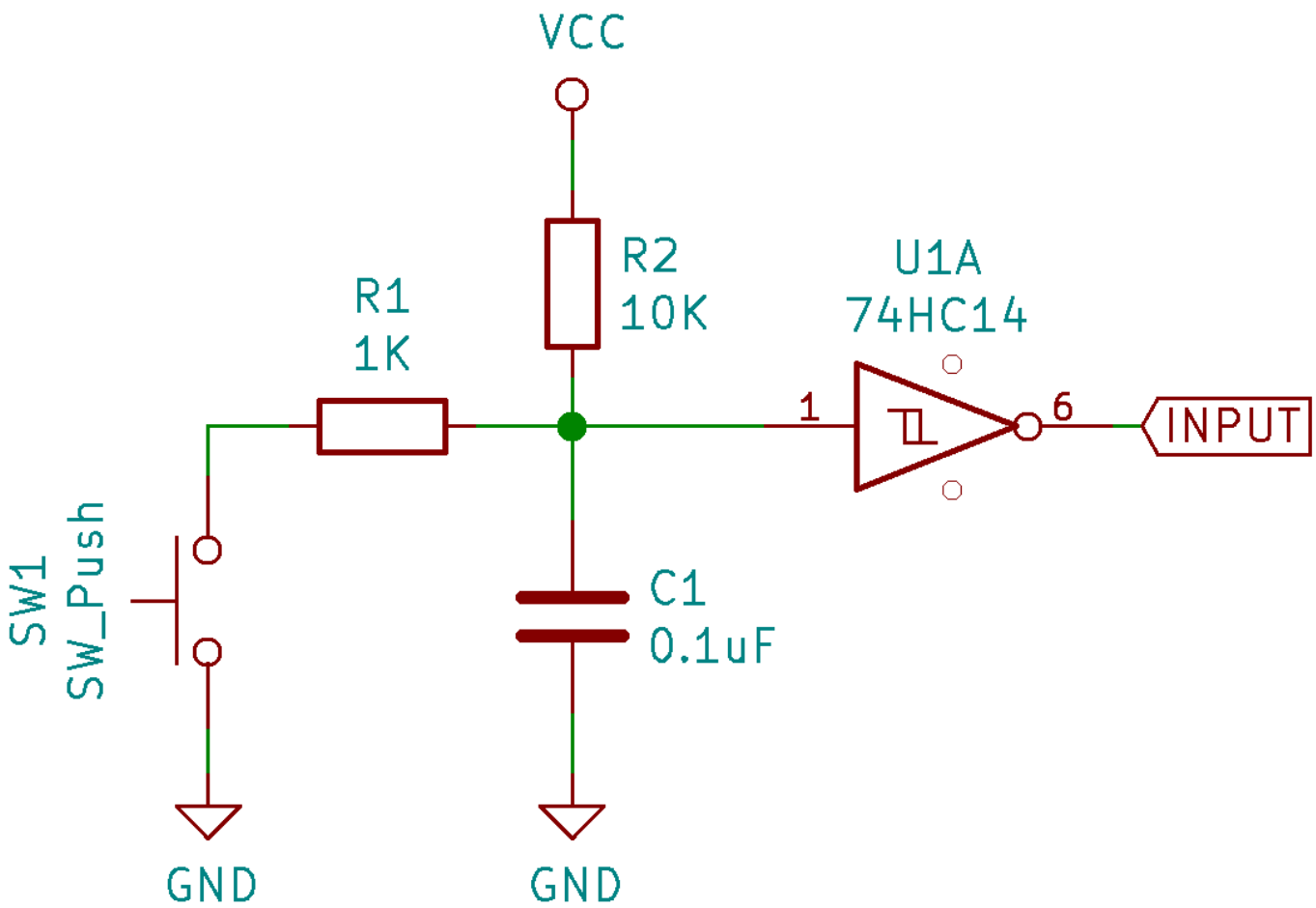
Button Bounce

- Fix 1 - RC Circuit

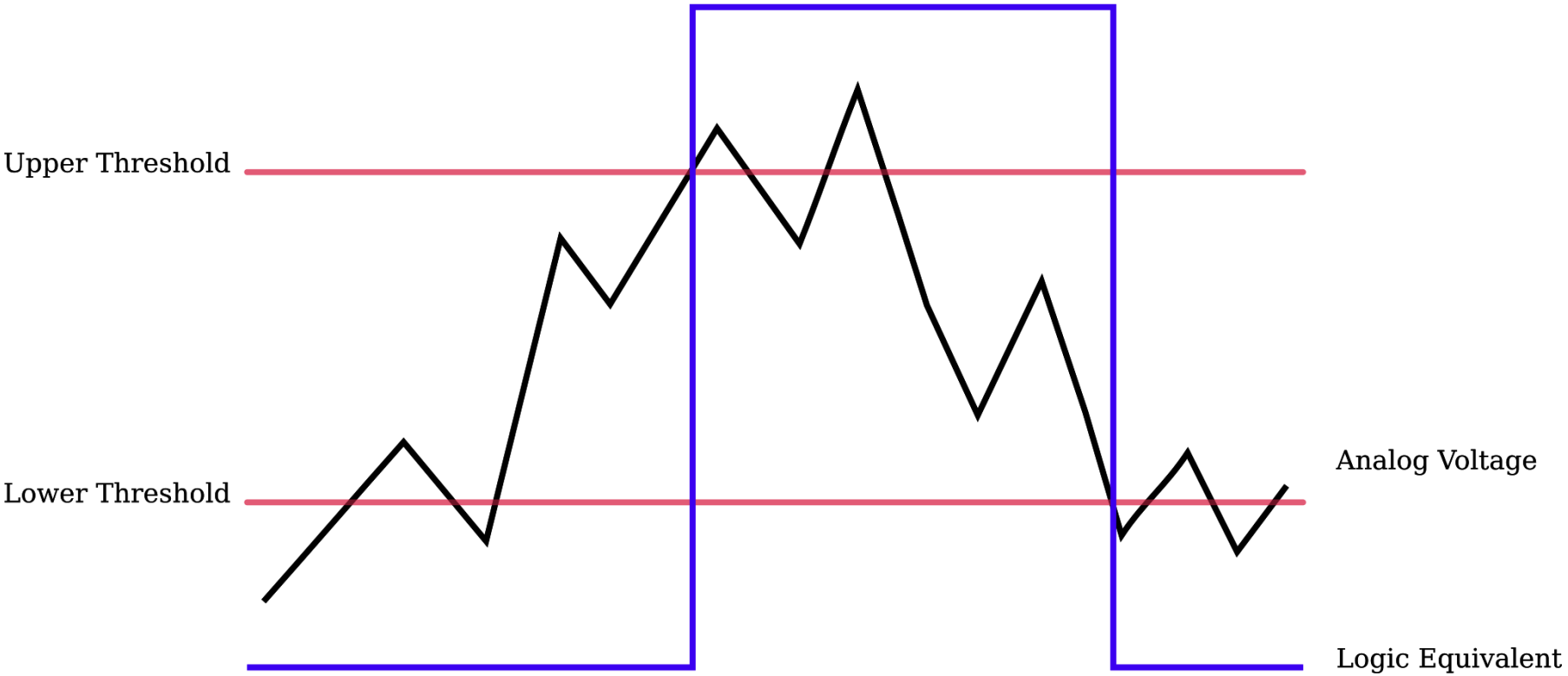


Button Bounce

- Fix 2 - RC Circuit + Schmitt trigger

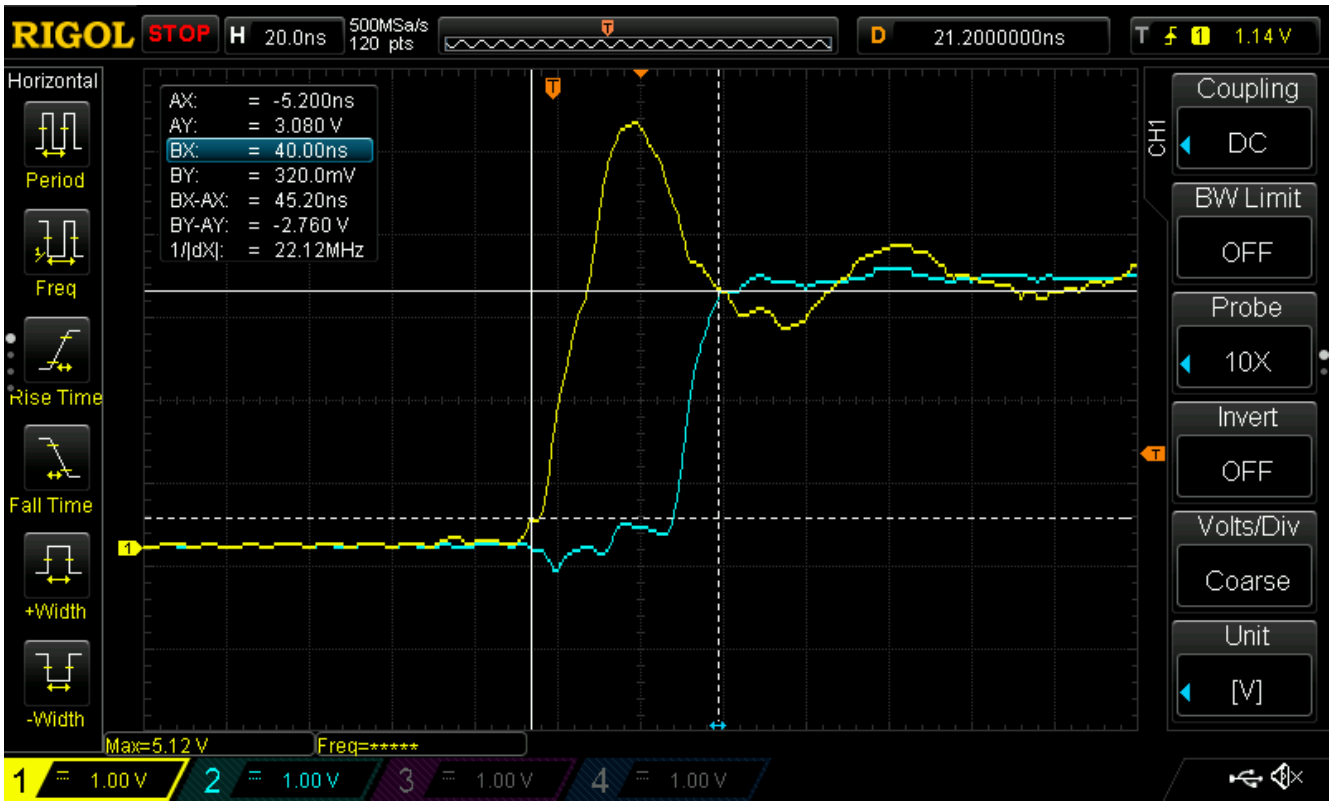


Hysteresis


Upper Threshold
Lower Threshold
Analog Voltage
Logic Equivalent

Button Bounce

- Fix 3 - Button polling (FPGA assignment version)



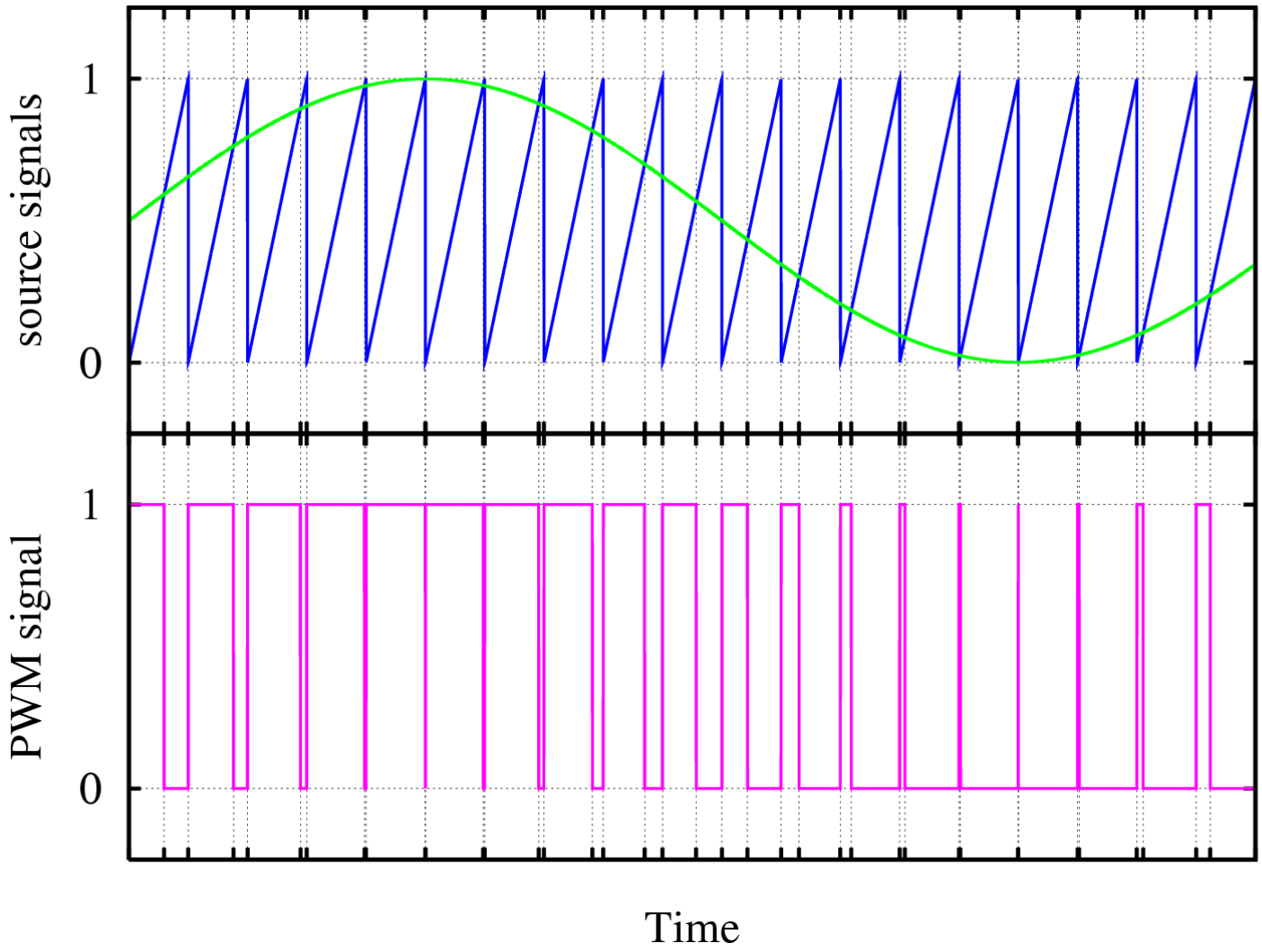
Button Bounce

- Fix 3 - Button polling (uC101 version)

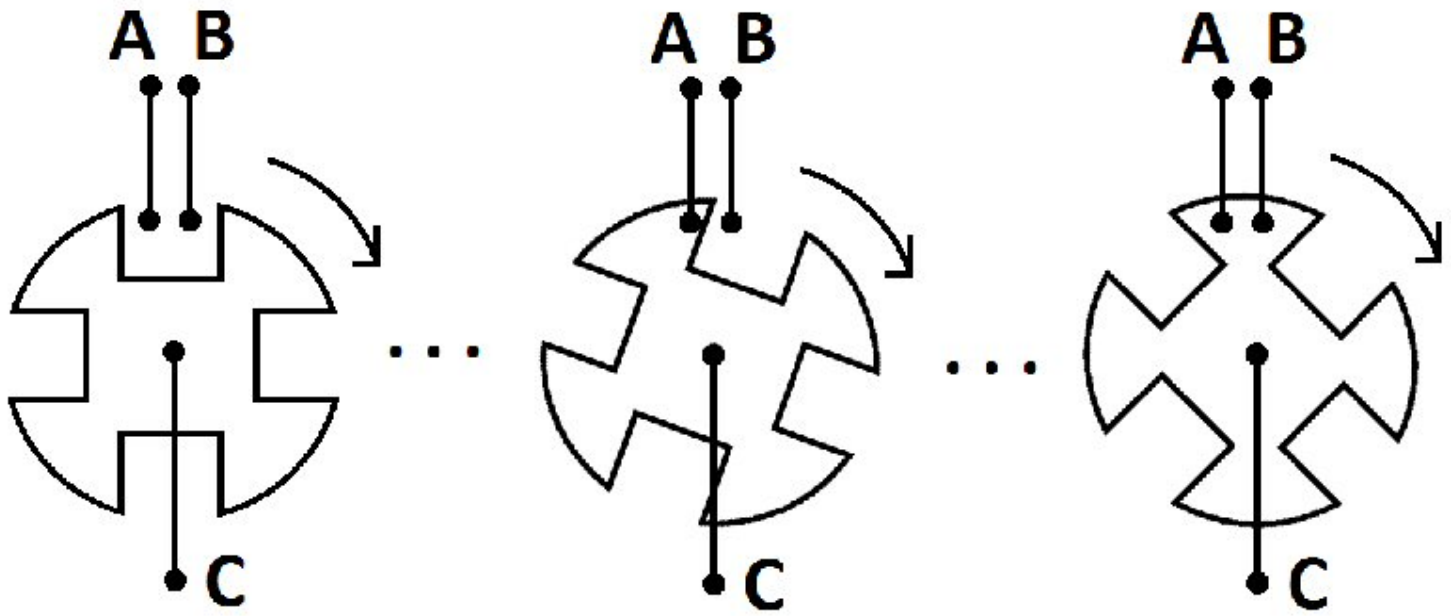
```
// Shift in a new value
history = history << 1;
history |= readPin(button);

// Check for button conditions
(history == 0b11111111) ? on = 1 : on = 0;
(history == 0b00000000) ? off = 1 : off = 0;
(history == 0b01111111) ? rising = 1 : rising = 0;
(history == 0b10000000) ? falling = 1 : falling = 0;
```

Above code called every 1-5ms

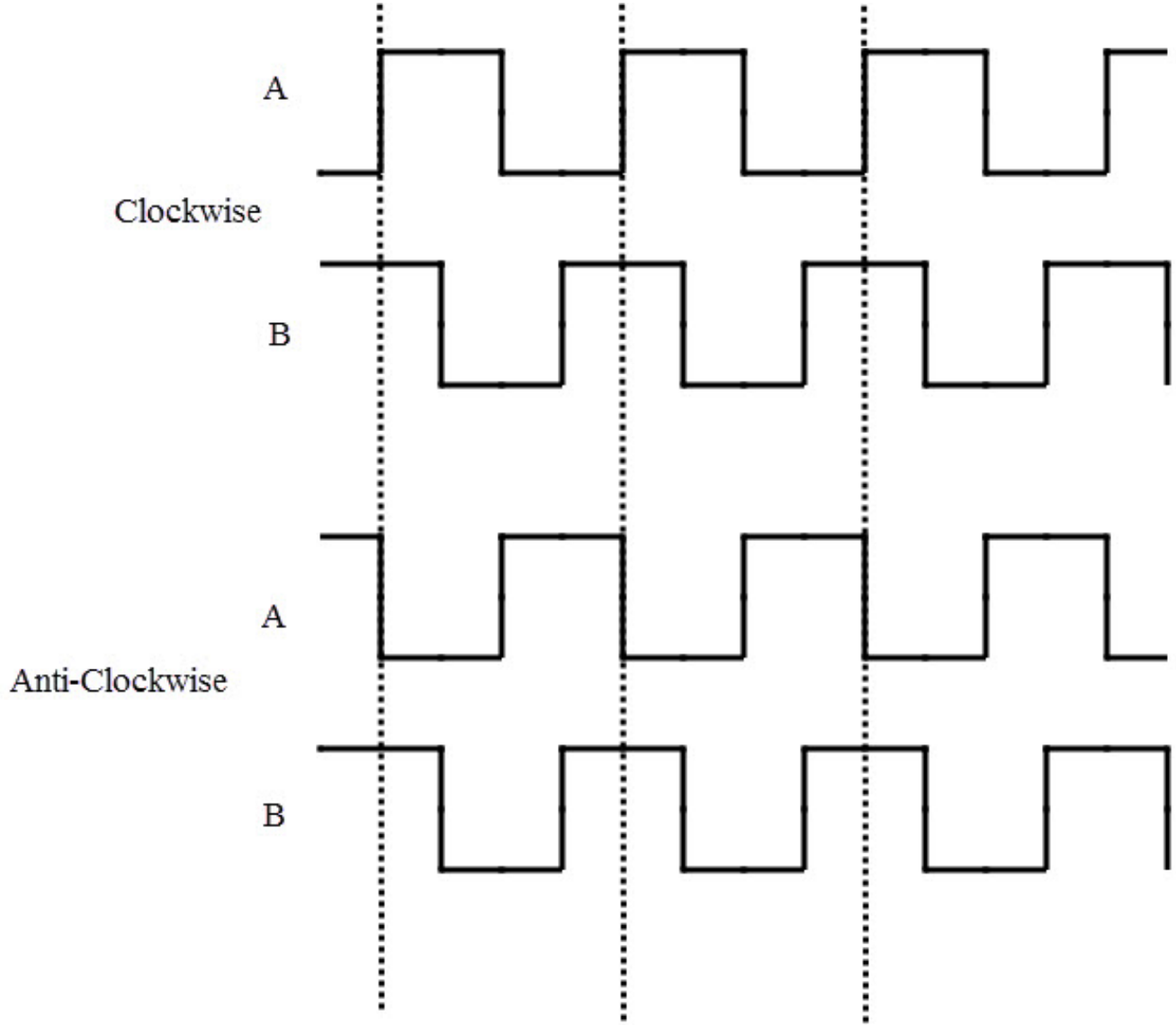
Pulse Width Modulation



Rotary Encoder Internal Operation



Rotary Encoder Output



Rotary Encoder Code

```
(onB && risingA) ? clockwise = 1 : clockwise = 0;
(onB && fallingA) ? antiClockwise = 1 : antiClockwise = 0;
```

or

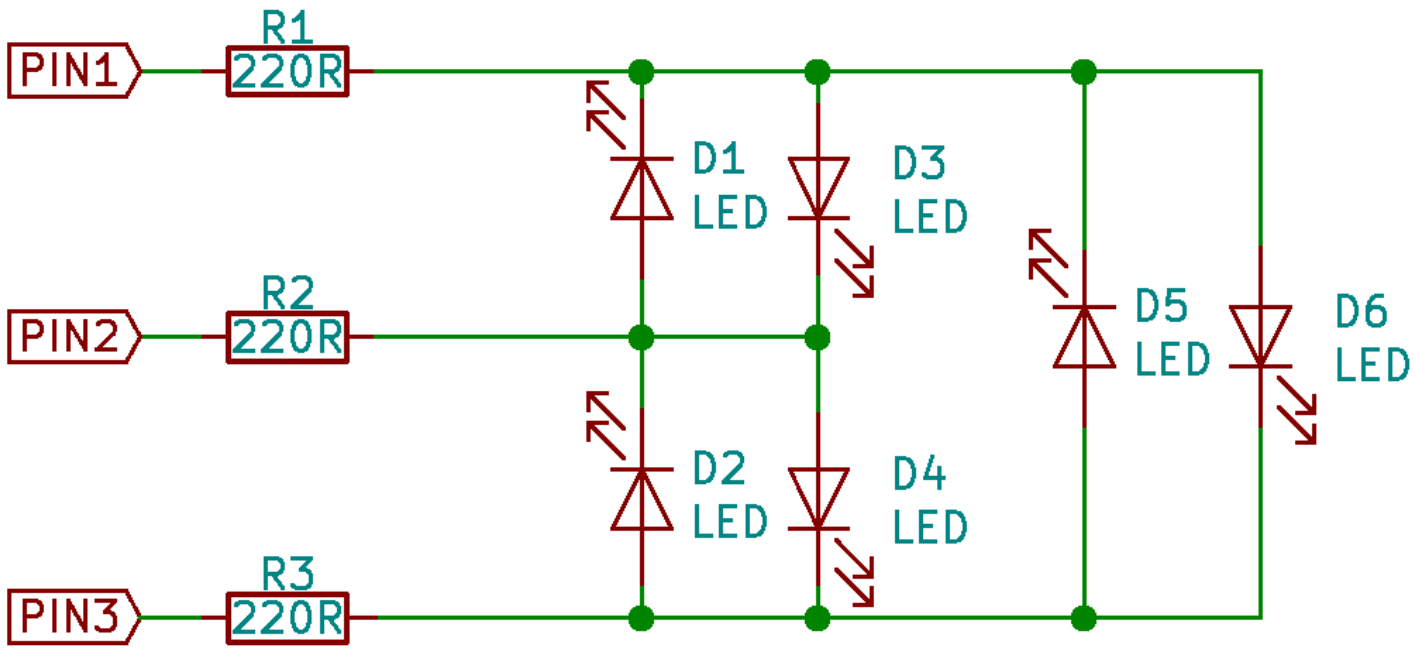
```
(onB && risingA) ? clockwise = 1 : clockwise = 0;
(onA && risingB) ? antiClockwise = 1 : antiClockwise = 0;
```

or

```
(fallingA && onB) ? clockwise = 1 : clockwise = 0;
(fallingA && offB) ? antiClockwise = 1 : antiClockwise = 0;
```

all are functionally identical, however last one can be pin interrupt driven

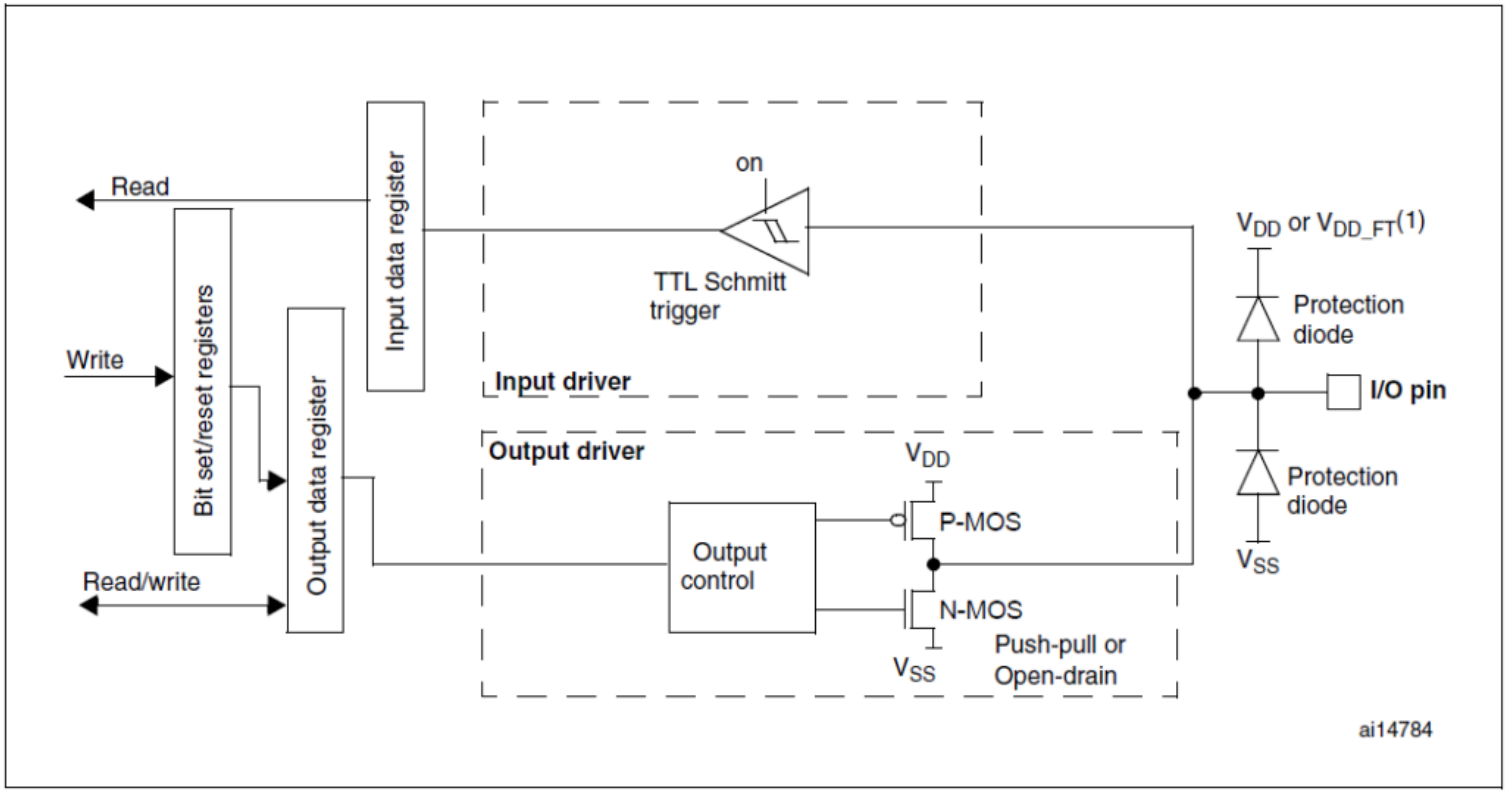
Charlieplexing vs Multiplexing



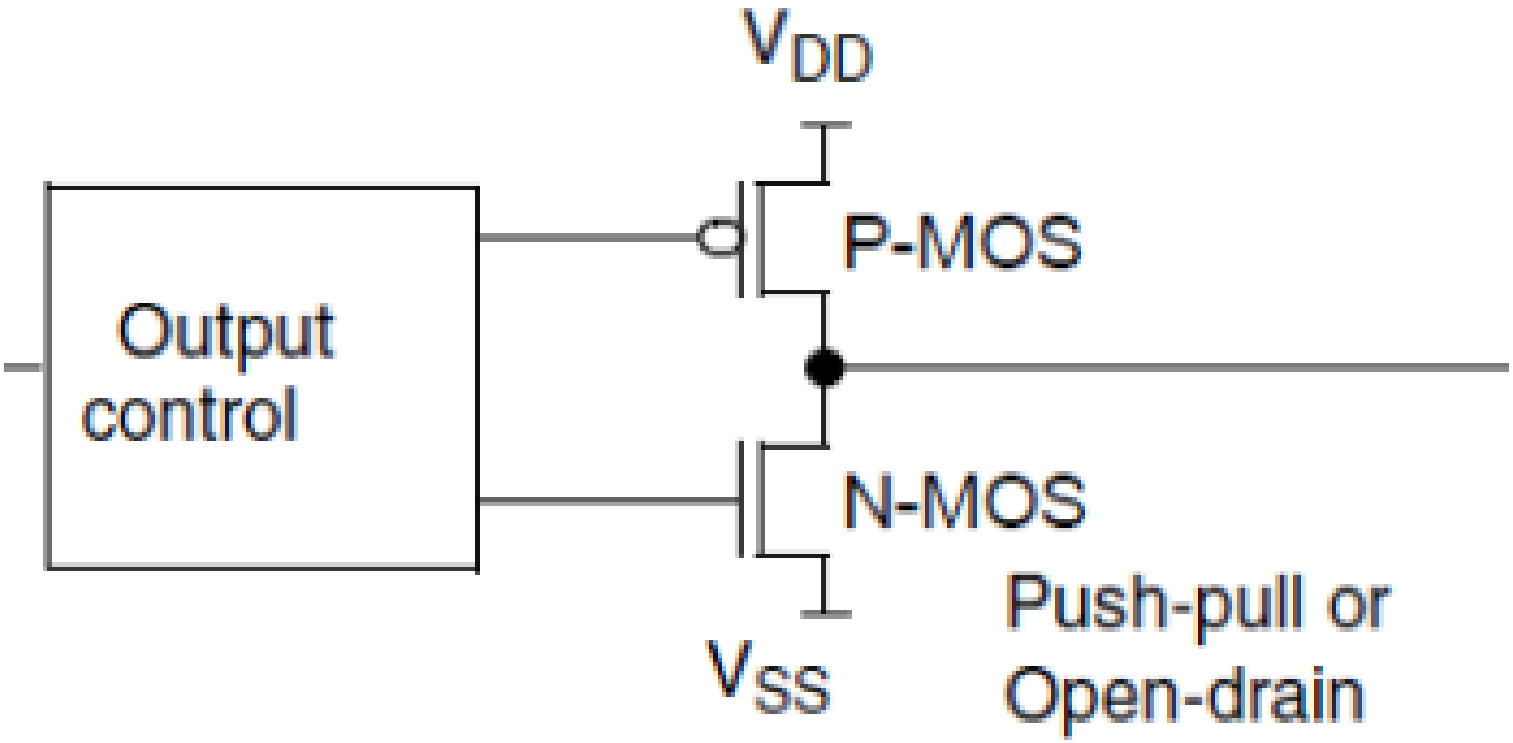
Charlieplexing: numLed $= p^2 - p$

Has to be scanned

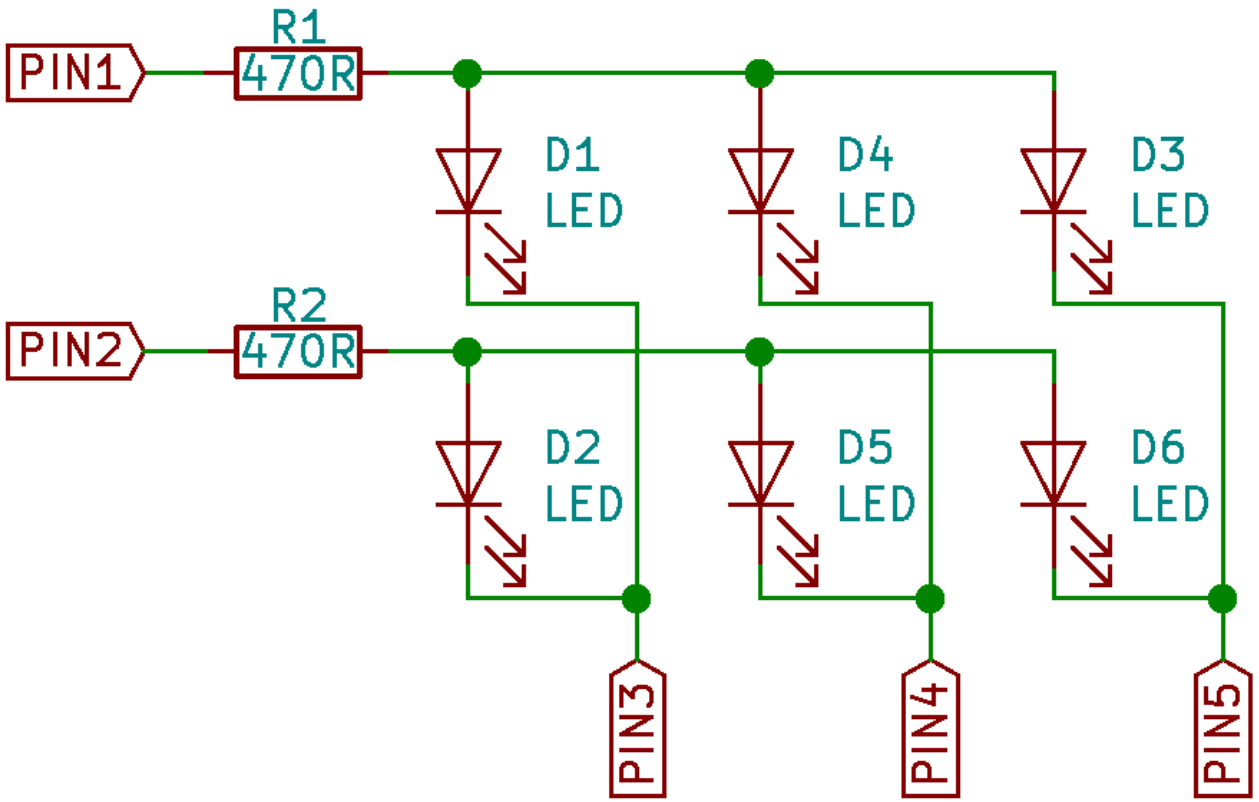
How to tristate a pin



How to tristate a pin



Charlieplexing vs Multiplexing



$$\text{Multiplexing: numLed} = \left\lfloor \frac{p}{2}^2 \right\rfloor$$

Can be continuously driven

The End

Links to resources: `uC101/README.md`

Next month

- Breadboard to Printed Circuit Board
- Mechanical Design Considerations

Month after that

- uC102: Communication Protocol Edition

Say Hello!
BSidesCbr Slack: josh
Twitter: @_joshajohnson
Email: josh@joshajohnson.com

Project Files: `github.com/joshajohnson/CBRhardware`