General Purpose IO Pins (GPIO) 
ATMega128

- can be programmed to be either input or output
  - Port has three registers
    - Data direction register (e.g., DDRE)
      - Write 0/1 to DDR to select as input/output
    - Port input (e.g., PINC)
      - Read = get value, Write = toggle port bits
    - Data register (e.g., PORTB)
      - if DDRX = 1 (Output): 0/1 = Output low/high
      - if DDRX = 0 (Input): 0/1 = Pull-up disabled/enabled
  - By default, input DDRE = 0x00
Taibotics Robot Buttons

- Buttons on the Taibotics Educational Robot
- Button 1 = Port D, 0
- Button 2 = Port D, 1
- Button 3 = Port E, 4
- Button 4 = Port E, 5
- Button 5 = Port E, 7
Buttons

- Button simply makes contact to create a circuit
- Recall pull-up/down circuit to prevent input from floating
- Schematic of a button

- The AtMega has built-in pull-ups
- Internal pull-ups must be enabled
Buttons

- Polling: Need to sample the button state continuously
- Sampling rate
  - How fast does a human press/release a button
  - 0.05 to 0.1s is common
- Our main program must
  - check the status of the button continuously, and
  - update the button state to semantic actions (press, release, click, double click, ...)
Debouncing

● When the button is pressed

● Depends on RF characteristics of circuit
● We can not use a single sample to determine state, otherwise lots of
  false clicks
● Interrupts
  ○ Level triggered
  ○ Edge triggered
Debouncing

- **Hysteresis**
  - Introduce a dead-zone at which no change to the state is signaled or made
  - Example: Robot carrier
  - Standard engineering technique to avoid false alarms due to small noise

- **Buttons**
  - Collect more than one sample
  - Averaging will introduce delay
Button Hysteresis

- Count on states
  - Sample on/off, then increment/decrement count
- Use Min, Max, On, Off limits
- Do not change button status between Off to On

Max = 12
On = 10
Off = 4
Min = 0
Hysteresis

- Tune parameters to deal with different hardware
  - Depends on the level of noise
- General idea can be used to deal with simple noisy/unstable input
- Same idea can be applied to interrupt driven buttons
Buttons

● Respond to
  ○ key press, key release, key click

● Level Interrupts
  ○ If IRQ is generated on ON, then how to detect OFF
  ○ Stay in ISR until release
  ○ Disable interrupt for some time. When to reenable?

● Key repeats
  ○ On for > x ms triggers another key press
Buttons

● Debouncing introduces a delay
● Fast/short key presses may be ignored
● Shortening the debounce may introduce false positives due to noise
● Hardware solutions
  ○ Cross-coupled NAND gates with three button switch
  ○ RF filtering