Debouncing

ENCE361 Embedded Systems 1

Course Coordinator: Ciaran Moore (ciaran.moore@Canterbury.ac.nz)

Lecturer: Le Yang (le.yang@canterbury.ac.nz)

Department of Electrical and Computer Engineering

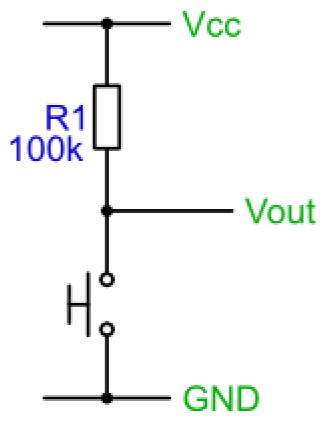
Where we’re going today

- **Switch bounce**
- Hardware debouncing
- Software debouncing
- Homework

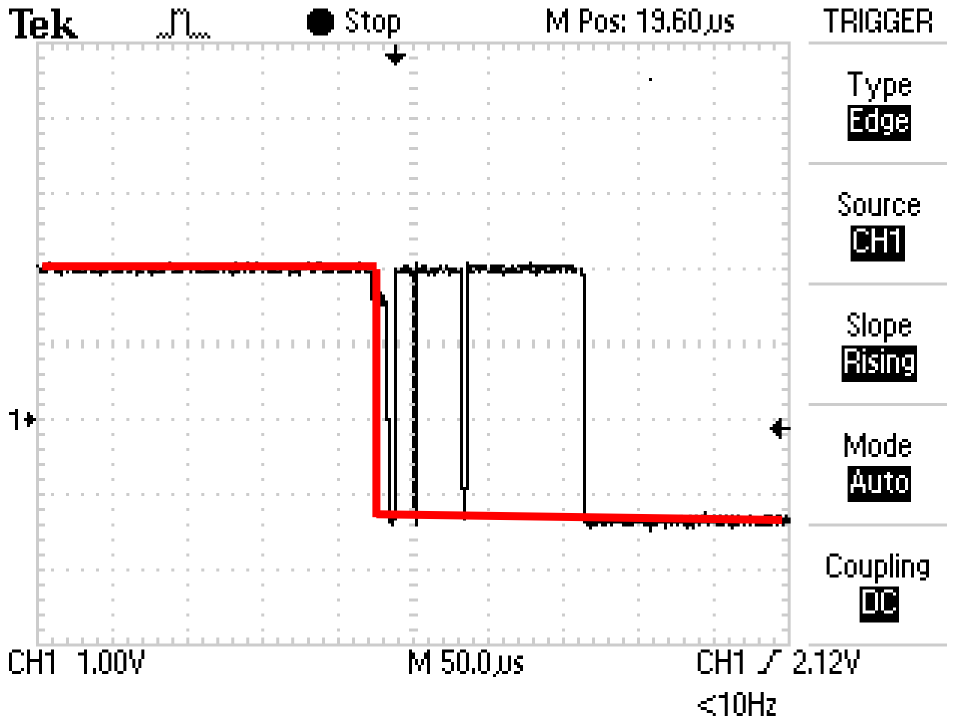
Switch Bounce (1)

- After pushing a button (closing switch), two metal parts repeat connecting and separating multiple times over a short interval before settling down

Simple pull-up switch



V_{out} on an oscilloscope

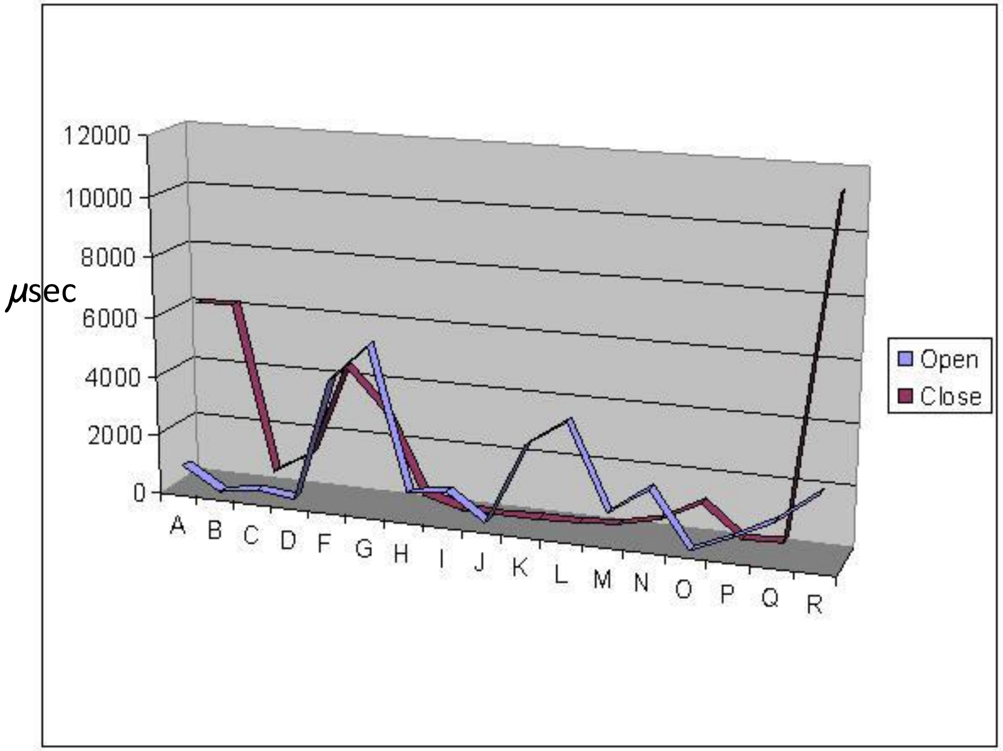


- Switch bounce could also occur when releasing a switch

Figures from http://www.labbookpages.co.uk/electronics/debounce.html

Switch Bounce (2)

- Have to live with (somewhat) unpredictable switch bounce behaviour



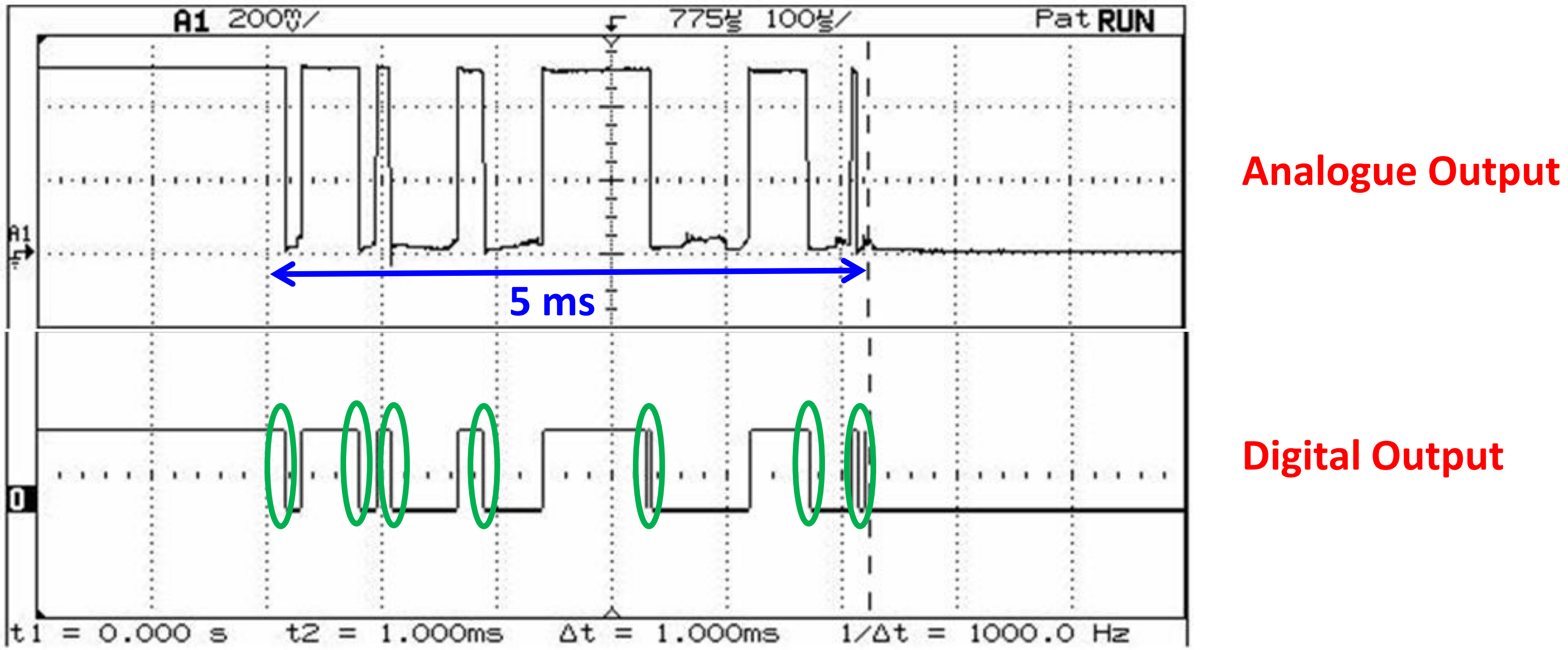
Bounce times in microseconds, for opening and closing each switch (number A to R). Switch E was left out, as its 157 msec bounces would horribly skew the graph.

- Bounce times in μsec when pushing and releasing 20 switches
- Switch E was left out due to its 157 msec bounce time (around 1/6 second!)

Figures from http://www.ganssle.com/item/debouncing-switches-contacts-code.htm

Switch Bounce (3)

- Switch bounce in TV remote control …
  - How many times was the button pushed?



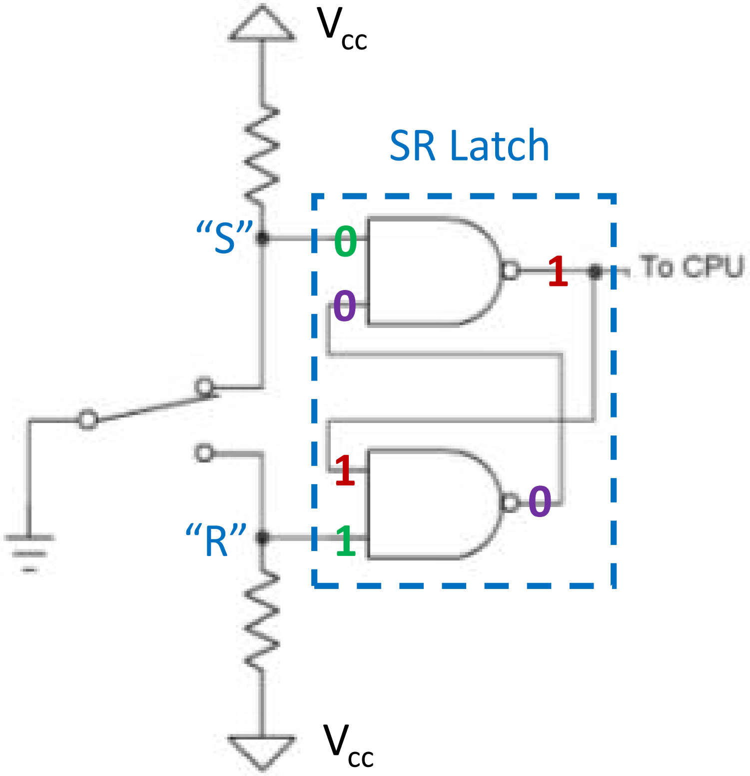
Figures from http://www.ganssle.com/item/debouncing-switches-contacts-code.htm

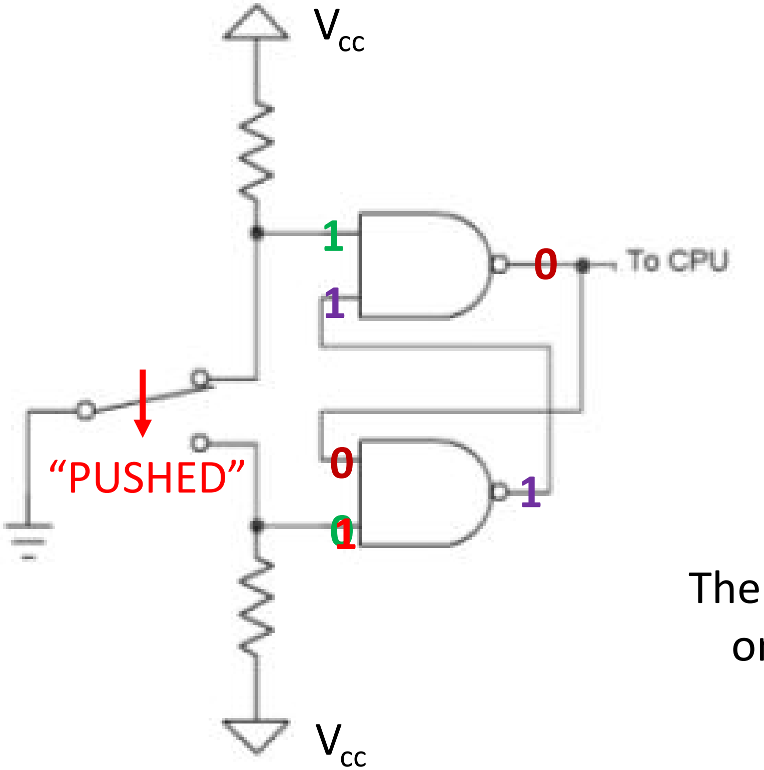
Where we’re going today

- Switch bounce
- **Hardware debouncing**
- Software debouncing
- Homework

Debouncer for Double-Throw Switch (1)

- Debouncing: processing switch signal so that a single press would not be interpreted as multiple presses
- Double-throw switch debouncing using a SR flip-flop



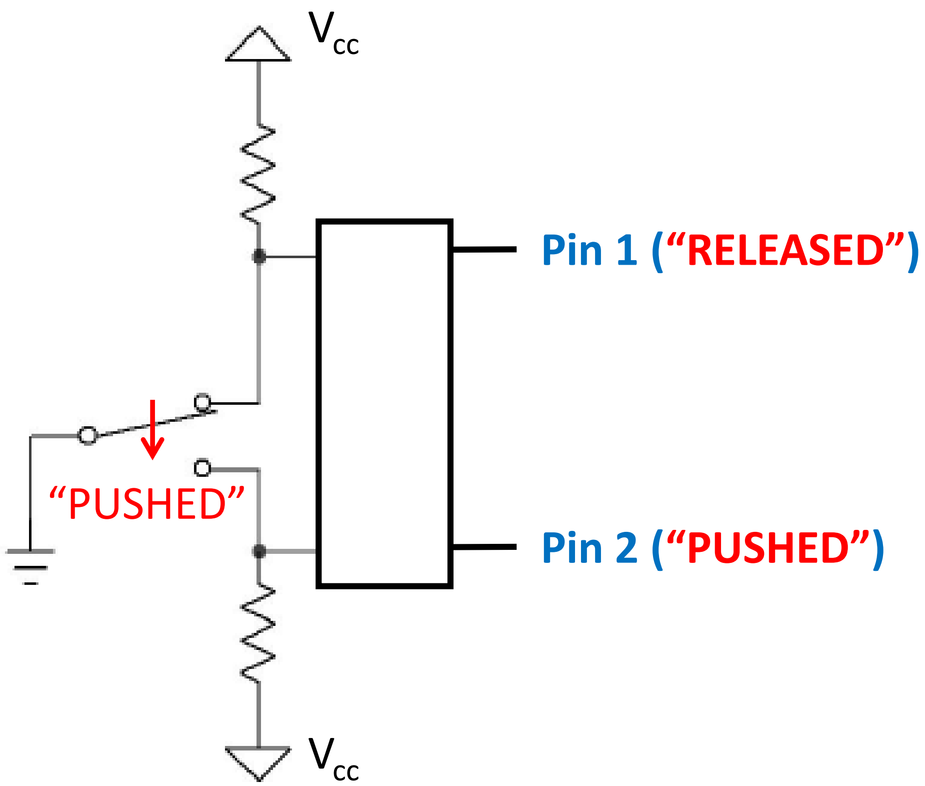


Double-throw switch sold on Amazon

The press is locked once detected

Debouncer for Double-Throw Switch (2)

- Conceptually, SR-based double-throw switch debouncer is equivalent to



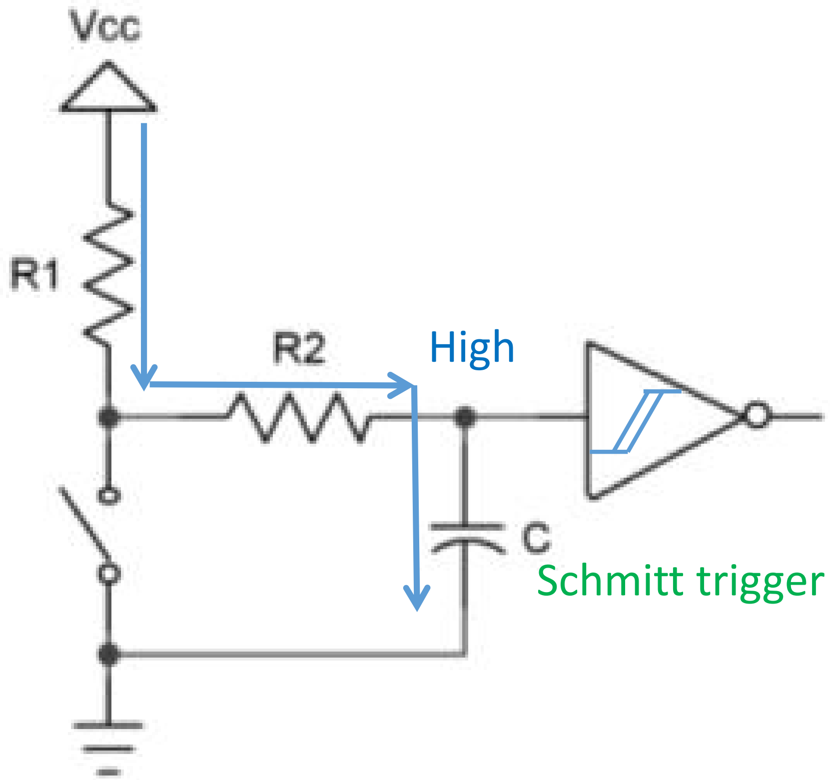
Software version of the debounced double-throw switch

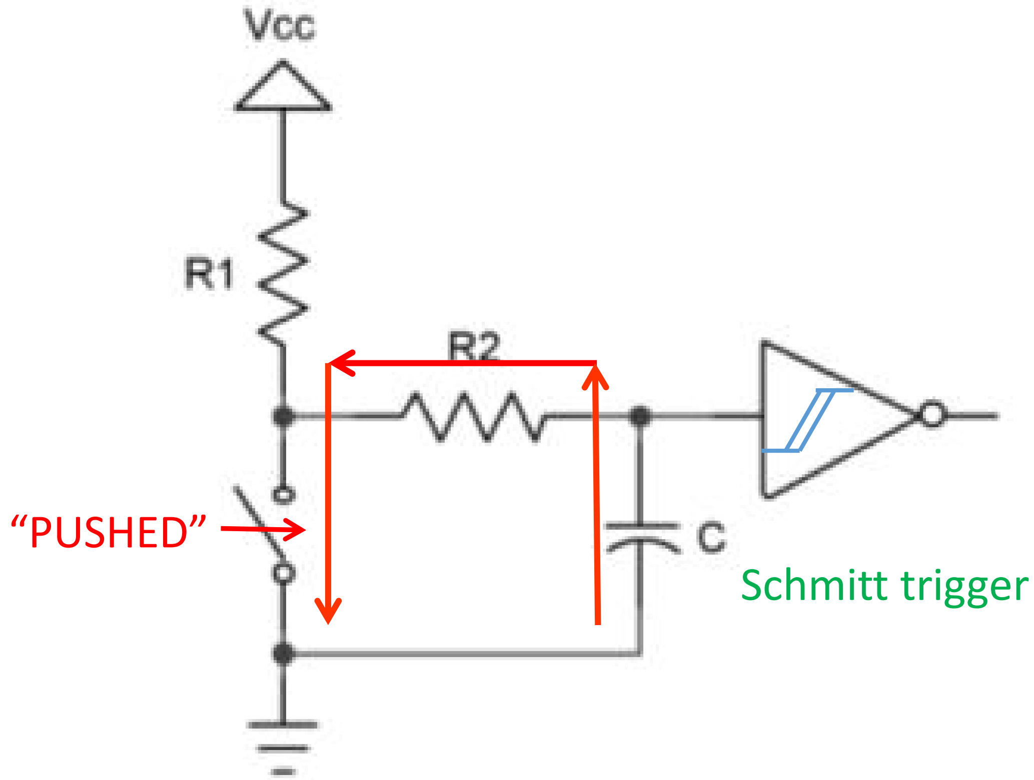
```
if (pin2())    // Detect "PUSH" operation
    state = PUSHED;

if (!pin1())    // Detect "RELEASE" operation
    state = RELEASED;
```

RC Circuit-based Debouncer

- Double-throw switch is bulky and single-throw switches are more common
- Debouncing using RC circuit and Schmitt trigger



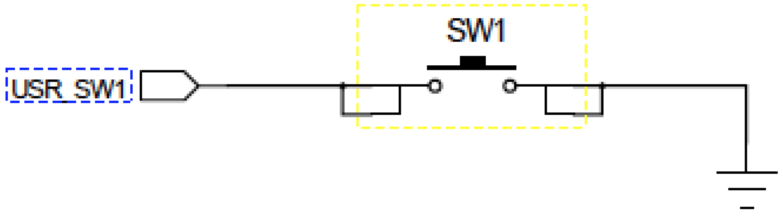


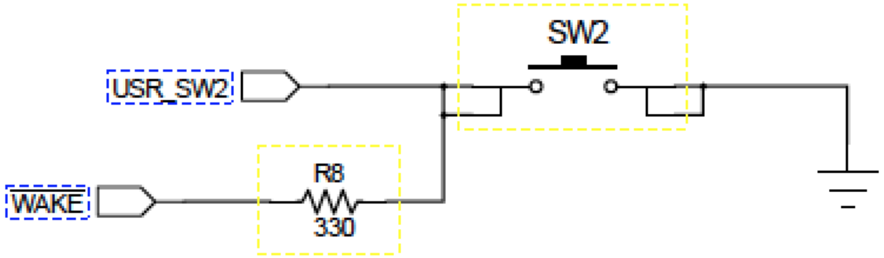
Where we're going today

- Switch bounce
- Hardware debouncing
- **Software debouncing**
- Homework

Switches on Tiva C-Series Launchpad

- Two single-throw switches: SW1 and SW2
- Pushing a button connects a GPIO pin to GND
  - SW1 → PF4
  - SW2 → PF0





- No debouncing circuitry
  - Software debouncing is a must

From Page 20, Tiva C Series TM4C123G Launchpad Users Guide.pdf

Software Debouncing (1)

- Basic idea
  - Switch bounce causes multiple contacting-separation cycles over a short period
  - Voltage settles down after switch bounce

  - N consecutive readings that are the same
    - Indicating that pushing/releasing a button has been completed *reliably*

- How to poll the pin to obtain voltage readings?
  - Pin-change interrupt-based polling
    - Many rapid voltage changes → many interrupts causing MCU to be fully occupied for several ms

  - Regular polling using timers (e.g., SysTick at 100 Hz)
    - Still an interrupt-based approach but with better MCU scheduling

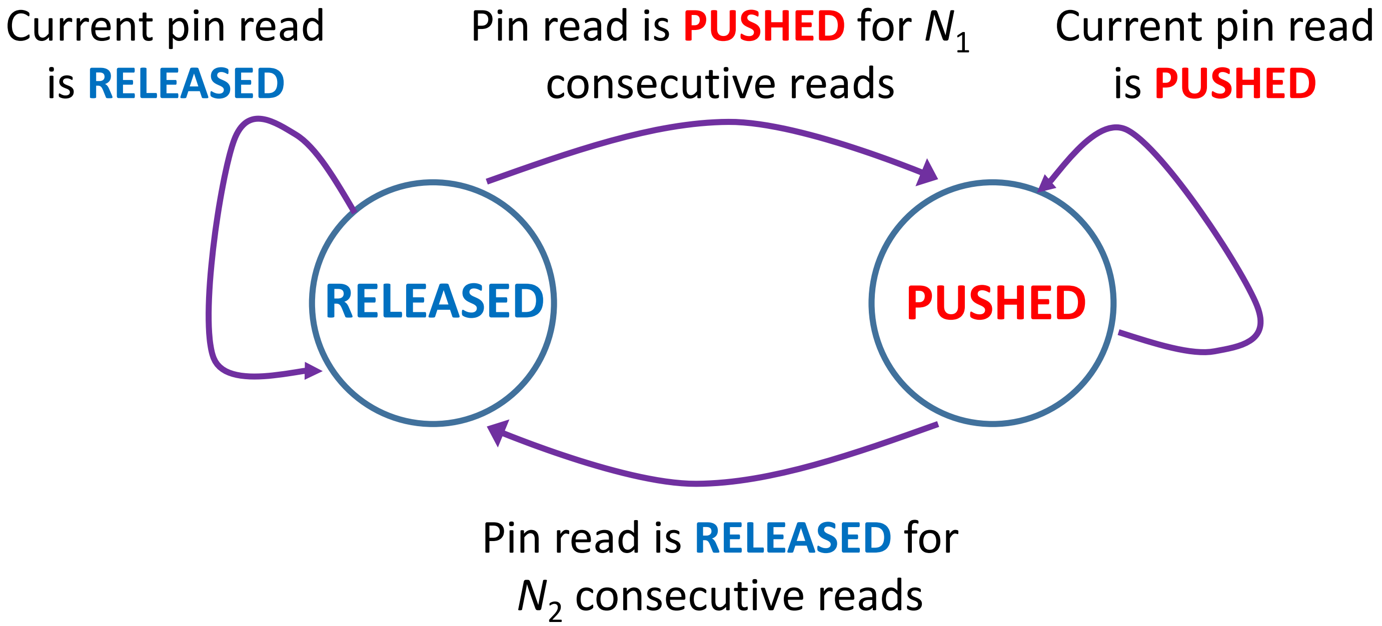
Software Debouncing (2)

- Algorithm
  - Suppose switch button stays at **RELEASED**
  - Poll the pin connecting the switch at a fixed rate
    - If pin read is **PUSHED**
      - If pin read has been **PUSHED** for N_1 consecutive reads
        - Change the button state to PUSHED
    - If pin read is **RELEASED**
      - Stay at **RELEASED**

Sounds complicated? ☺

Finite State Machine (FSM)

- In computer science, finite state automaton (FSA)
- State
  - **PUSHED** and **RELEASED**
- Triggers-based state transition
  - RELEASED → PUSHED
  - RELEASED → RELEASED
  - PUSHED → RELEASED
  - PUSHED → PUSHED



Design Considerations

- Choose **a relatively low value for N_1** for a rapid response to pushing button
- Select **a relatively large value for N_2** to reduce false detections
- Multiple buttons can be polled when servicing each (say SysTick) interrupt
- A state variable that takes the value of **PUSHED** or **RELEASED** can be maintained for each button

Example Code: buttons4.h

```
// buttons4.h
// Support for a set of four specific buttons on the Tiva/Orbit
// The buttons are:  UP and DOWN (on Orbit daughterboard) + RIGHT & LEFT on Tiva board

enum butNames {UP = 0, DOWN, LEFT, RIGHT, NUM_BUTS};
enum butStates {RELEASED = 0, PUSHED, NO_CHANGE};

// UP button
#define UP_BUT_PERIPH  SYSCTL_PERIPH_GPIOE
#define UP_BUT_PORT_BASE  GPIO_PORTE_BASE
#define UP_BUT_PIN  GPIO_PIN_0
#define UP_BUT_NORMAL  false

// DOWN button
#define DOWN_BUT_PERIPH  SYSCTL_PERIPH_GPIOD
#define DOWN_BUT_PORT_BASE  GPIO_PORTD_BASE
#define DOWN_BUT_PIN  GPIO_PIN_2
#define DOWN_BUT_NORMAL  false
```

Example Code: buttons4.c

```
static bool but_state[NUM_BUTS];          // State variables (PUSHED/RELEASED) for 4 buttons
static uint8_t but_count[NUM_BUTS];

void updateButtons (void) {
  bool but_value[NUM_BUTS];
  int i;
  // Poll UP and DOWN buttons
  but_value[UP] = (GPIOPinRead (UP_BUT_PORT_BASE, UP_BUT_PIN) == UP_BUT_PIN);
  but_value[DOWN] = (GPIOPinRead (DOWN_BUT_PORT_BASE, DOWN_BUT_PIN) == DOWN_BUT_PIN);
  for (i = 0; i < NUM_BUTS; i++) {
      if (but_value[i] != but_state[i]) {
          but_count[i]++;                          // Record current pin read if it is different from the pin state
      if (but_count[i] >= NUM_BUT_POLLS) {         //  State transition for button i
          but_state[i] = but_value[i];
          but_count[i] = 0;
      }
    }
    else
      but_count[i] = 0;                            // Stay in the pin state
  }
}
```

Homework

1. With reference to Slide 9, explain how the RC debouncer works and the role of the Schmitt trigger. What is a disadvantage of this hardware method of debouncing?

2. Why are pin-change interrupts for debouncing button pushes **not** recommended?

3. Why are timer generated interrupts recommended for button polling (Slide 12)?

4. Write a modified version of **updateButtons()** which implements the dual count (N_1, N_2) algorithm described on Slide 14.