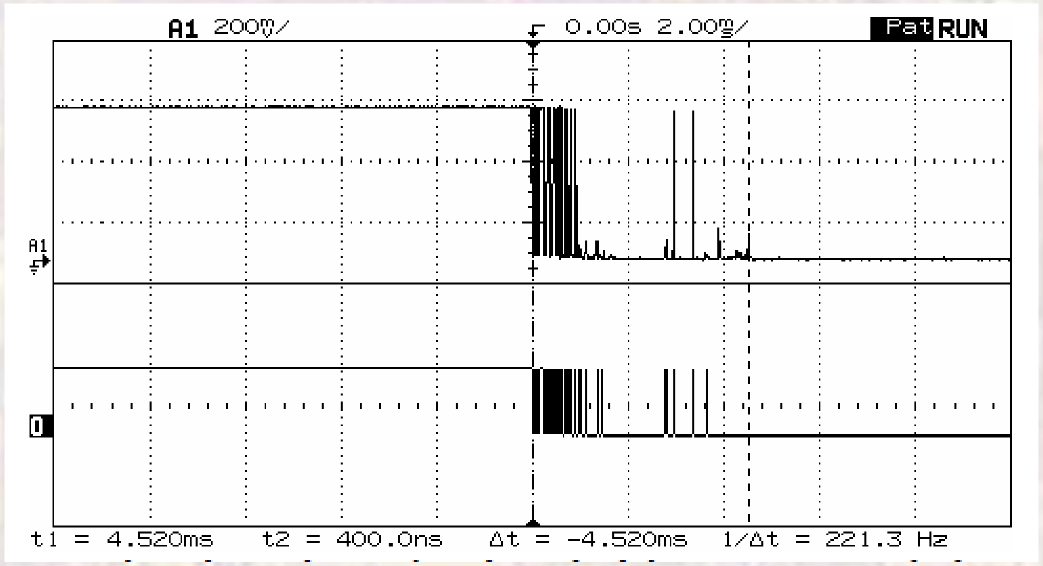
Pin Debounce

Last updated – 2/14/20

Debounce

- When a button is pressed (or released) it often bounces
  - This causes the pin associated with the button to oscillate between 0 and 1



src: The Ganssle Group

Debounce

- There are hardware and software solutions
  - This problem is very complex
  - Hardware solutions can be made very robust – but may not be practical (or available) on our board
  - Software solutions are not 100% effective
  - We want to asynchronously check a pin
  - Any solution we choose has some failure mechanism
  - Note: typically the bouncing is resolved in less than a milli-second

Debounce

- Simple software based debounce solution
  - We want to asynchronously check a pin
  - Any solution we choose has some failure mechanism
  - Typically the bouncing is resolved in less than a milli-second

  - We can check the pin, wait a few milli-seconds and check again
    - If the pin is different we may be bouncing – do not update the value
    - If the pin is the same we know we are not bouncing – “valid”

  - Keep track of the current pin value
    - Update the value only if the new “valid” pin value is different than the old “current” pin value

Debounce

- get pin value - debounced

Output:
updated pin value via pointer

Input:
Pointer to pin register

Input:
Mask for pin bit

```
void check_pin(uint8_t * pin_val_ptr, const volatile uint8_t* pin_reg, uint8_t pin_mask){
    // Check the input two times separated by 5ms to debounce a pin
    // pin_val_ptr - pointer to the value of the pin
    // pin_reg - pointer to pin register, pin_mask - mask for the desired pin
    // ex: check p6.6 and store in variable my_pin_val
    //     check_pin(&my_pin_val, &P6->IN, 0x40)

    // *** assumes default frequency of ~3MHz ***
```

Example: check pin P6.2 and store the value in my_pin_val

`check_pin(&my_pin_val, &P6->IN, 0x04);`

Debounce

- get pin value - debounced

```c
void check_pin(uint8_t * pin_val_ptr, const volatile uint8_t* pin_reg, uint8_t pin_mask){
    // Check the input two times separated by 5ms to debounce a pin
    // pin_val_ptr - pointer to the value of the pin
    // pin_reg - pointer to pin register, pin_mask - mask for the desired pin
    // ex: check p6.6 and store in variable my_pin_val
    //     check_pin(&my_pin_val, &P6->IN, 0x40)

    // *** assumes default frequency of ~3MHz ***

    // temporary variables
    uint8_t pin_val_a;
    uint8_t pin_val_b;

    // first check
    pin_val_a = *pin_reg & pin_mask; // get input pin value

    // delay for debouncing  (5ms)
    __delay_cycles(5*(3000000/1000)); // change this for different clock frequencies

    // second check
    pin_val_b = *pin_reg & pin_mask; // get input pin value

    // test for changes
    if (pin_val_a == pin_val_b){
        *pin_val_ptr = pin_val_a && 1; // save new pin value
    }
    else{
        ; // keep current pin value
    } // end if

    return;
} // end check_pin
```

Debounce

- check_pin() limitation

  - It is possible that the pin could be changed (and start bouncing) during the 5ms waiting period

  AND

  - The second check catches the bounce in the original position – leading to a decision of a stable pin and missing the change

  - Solution – add a second delay and a third check and require all three checks to match before updating the pin value – not necessary for our purposes