Debouncing Switches

- Mechanical switches are one of the most common interfaces to a uC.
- Switch inputs are asynchronous to the uC and are not electrically clean.
- Asynchronous inputs can be handled with a synchronizer (2 FF’s).
- Inputs from a switch are electrically cleansed with a switch debouncer.
- This can also be done with software.
- What is switch bounce?
  - The non-ideal behavior of the contacts that creates multiple electrical transitions for a single user input.

Figure 1: Contact Closure is Anything but Simple
Debouncing Switches

- Falling and rising edge switch bounce from a pushbutton switch

Figure 2: Either Edge Can Bounce
Debouncing Switches

- The problem is that the uC is usually fast enough to see all the transitions
- uC acts on multiple transitions instead of a single one
- The oscilloscope traces showed bounce durations of 10-300us
- our mega128 uC runs at 62.5ns per instruction
- a 10uS bounce (short) is \((1\times10^{-5}/62.5\times10^{-9})\) 160 instructions long!
- a 100uS bounce could be sampled as a valid true or false 100s of times
- results are incorrect behavior as seen by user
Debouncing Switches

- Characteristics of switch bounce:
  - Nearly all switches do it
  - The duration of bouncing and the period of each bounce varies
  - Switches of exactly the same type bounce differently
  - Bounce differs depending on user force and speed
  - Typical bounce frequency is 100us-10ms

![Specifications](image)

**Figure 3:** Specifications for Panasonic EVP-BD6C1A000 pushbutton switch
Debouncing Switches

► One possible solution - Analog filtering
► RC network filters out the rapid changes in switch output
► Choose R and C so input threshold is not crossed while input is still bouncing

RC filter is formed by R1, R2 and C1.
R2 also protects S1 from current spike when S1 is closed.
R3 protects U1 from capacitor dumping current into input pin when power is removed.
Debouncing Switches

- Another solution would be to use a latch (MC14044)
- Logic gates lock change in $2t_{pd}$ using a SPDT switch
- Both switch ($3.69$) and chip ($0.38$) are expensive
- Momentary click switches (AVR board) are ($0.12$)
Debouncing Switches

- **Software solutions**
  - Need to minimize CPU usage and be independent of CPU clock speed
  - Use constant defines in makefile to remove speed dependencies
  - Don’t use interrupt pins, only periodic polling
  - Don’t synchronously scan noisy devices
  - Quickly identify initial switch closure (100mS max)
Debouncing Switches

▶ Count-based software solution

```c
// source: Jack Gansel, "Guide to Debouncing"
// returns '1' once per button push, detects falling edge

uint8_t debounce_pulse() {
    static uint16_t switch_state = (state << 1) | (! bit_is_clear(PIND, 2)) | 0xE000;
    if (state == 0xF000) return 1;
    return 0;
}
```

<table>
<thead>
<tr>
<th>Which pass</th>
<th>Value of state</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>first pass after reset</td>
<td>1110 0000 0000 0001</td>
<td>return 0</td>
</tr>
<tr>
<td>second pass after reset</td>
<td>1110 0000 0000 0011</td>
<td>return 0</td>
</tr>
<tr>
<td>after 12 false passes</td>
<td>1111 1111 1111 1111</td>
<td>return 0</td>
</tr>
<tr>
<td>after 7 true passes</td>
<td>1111 1111 1000 0000</td>
<td>return 0</td>
</tr>
<tr>
<td>after 12 true passes</td>
<td>1111 0000 0000 0000</td>
<td>return 1</td>
</tr>
<tr>
<td>after many true passes</td>
<td>1110 0000 0000 0000</td>
<td>return 0</td>
</tr>
<tr>
<td>after 5 false passes</td>
<td>1110 0000 0001 1111</td>
<td>return 0</td>
</tr>
</tbody>
</table>
Debouncing Switches

Solution based on digital 1st-order recursive low-pass filter

```c
// Acts like RC filter followed by schmitt trigger
// Continuous output like an analog switch
// 0.25 = 0x3F, 0.75 = 0xC0, 1.0 = 0xFF

int8_t debounce_cont(){
    static uint8_t y_old = 0, flag = 0;
    uint8_t temp;

    // Digital filter: y_old = x_new * 0.25 + y_old * 0.75
    temp = (y_old >> 2); // yields y_old / 4
    y_old = y_old - temp; // (y_old * 0.75) by subtraction

    // If button pushed, add 0.25
    if(bit_is_clear(PIND, 2)){ y_old = y_old + 0x3F;}

    // Software schmitt trigger
    if((y_old > 0xF0) && (flag == 0)){ flag = 1; return 1;}
    if((y_old < 0x0F) && (flag == 1)){ flag = 0; return 0;}
    return (-1); // No change from last time
}
```
Debouncing Switches

Figure 4: Behavior of IIR Filter with Schmitt Trigger
Debouncing Switches

- Sometimes we want an output that is continuous for as long as the switch contacts are in their active state. For example, the keys on an electronic keyboard.
- Other times we want a momentary or pulsed output, such as a button that increments the hour alarm on a clock.
- The first count-based debouncer (Gansel) gave a pulsed output.
- The digital filter algorithm gives a continuous output.

Figure 5: Pulsed versus Continuous Output
Debouncing Switches

- How would you convert between types of debouncer output?
- Use a state machine to get a pulsed output from a continuous debouncer.

```c
// state machine returns one pulse for each push and release
static enum button_state_type{IDLE, PUSHED, WAIT} state;
switch (state){
    case (IDLE): output = 0; if (debounce_cont()) { state = PUSHED; } break;
    case (PUSHED): output = 1; state = WAIT; break;
    case (WAIT): output = 0; if (debounce_cont()) { state = IDLE; } break;
    default: break;
} // switch
```

![Diagram showing the state machine](https://via.placeholder.com/150)

- **PUSHOFF**
  - Rising edge: output = 1, output = 0
  - Falling edge: output = 1, output = 0

- **PUSHED**
  - Always: output = 1, output = 0

- **IDLE**
  - Output: 0

- **WAIT**
  - Output: 0

- **debounce_cont**
  - 0 always

- **output**
  - 0 always
Debouncing Switches

- A state machine for continuous output from a pulsed debouncer.
- This scheme requires rising and falling edge detection.

```c
// 2 state state machine returns continuous output
static enum button_state_type {OFF, PUSH} state;
switch (state) {
    case (OFF) : if (debounce_fpulse()) { state = PUSH; } break;
    case (PUSH) : output = 1; if (debounce_rpulse()) { state = OFF; } break;
    default : break;
} // switch
```