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.

What is switch bounce?
  - The non-ideal behavior of the contacts that creates multiple electrical transitions for a single user input.
Debouncing Switches

- Falling and rising edge switch bounce from a pushbutton switch

![Graphs showing switch bounce](image-url)
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 \((1 \times 10^{-5}/62.5 \times 10^{-9})\) 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

<table>
<thead>
<tr>
<th>Specifications for Panasonic EVP-BD6C1A000 pushbutton switch</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td><strong>Snap action / Push-on type SPST</strong></td>
</tr>
<tr>
<td>Rating</td>
<td>10μA 2V DC to 50mA 12V DC (Resistive load)</td>
</tr>
<tr>
<td>Contact Resistance</td>
<td>500mΩ max.</td>
</tr>
<tr>
<td>Insulation Resistance</td>
<td>100MΩ min. (at 100V DC)</td>
</tr>
<tr>
<td>Dielectric Withstanding Voltage</td>
<td>250V AC for 1 minute</td>
</tr>
<tr>
<td>Bouncing</td>
<td>10ms max. (ON, OFF)</td>
</tr>
</tbody>
</table>
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

![Diagram of RC network with S1 and U1](image)

- **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.

- **U1** is a schmitt trigger inverter similar to a 74HC14.
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
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;  //yold*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

- Behavior of the digital filter debouncer with schmitt trigger

First Order Digital Filter Debouncer Behavior

- $y_{old}$
- Upper threshold
- Lower threshold

Output values:

- 13F 63 147 117 16F 93 111 16F 93 174 117 194 194 D1 209 1D2 220 1D4 234 1D6 239 1D8 243 1D1 246 1D1 249 1D1 252 1D1 255 1D1 258 1D1 261 1D1 264 1D1 267 1D1 270 1D1 273 1D1 276 1D1 279 1D1 282 1D1 285 1D1 288 1D1 291 1D1 294 1D1 297 1D1 300 1D1 303 1D1 306 1D1 309 1D1 312 1D1 315 1D1 318 1D1 321 1D1 324 1D1 327 1D1 330 1D1 333 1D1 336 1D1 339 1D1 342 1D1 345 1D1 348 1D1 351 1D1 354 1D1 357 1D1 360 1D1 363 1D1 366 1D1 369 1D1 372 1D1 375 1D1 378 1D1 381 1D1 384 1D1 387 1D1 390 1D1 393 1D1 396 1D1 399 1D1 402 1D1 405 1D1 408 1D1 411 1D1 414 1D1 417 1D1 420 1D1 423 1D1 426 1D1 429 1D1 432 1D1 435 1D1 438 1D1 441 1D1 444 1D1 447 1D1 450 1D1 453 1D1 456 1D1 459 1D1 462 1D1 465 1D1 468 1D1 471 1D1 474 1D1 477 1D1 480 1D1 483 1D1 486 1D1 489 1D1 492 1D1 495 1D1 498 1D1 501 1D1 504 1D1 507 1D1 510 1D1 513 1D1 516 1D1 519 1D1 522 1D1 525 1D1 528 1D1 531 1D1 534 1D1 537 1D1 540 1D1 543 1D1 546 1D1 549 1D1 552 1D1 555 1D1 558 1D1 561 1D1 564 1D1 567 1D1 570 1D1 573 1D1 576 1D1 579 1D1 582 1D1 585 1D1 588 1D1 591 1D1 594 1D1 597 1D1 600 1D1 603 1D1 606 1D1 609 1D1 612 1D1 615 1D1 618 1D1 621 1D1 624 1D1 627 1D1 630 1D1 633 1D1 636 1D1 639 1D1 642 1D1 645 1D1 648 1D1 651 1D1 654 1D1 657 1D1 660 1D1 663 1D1 666 1D1 669 1D1 672 1D1 675 1D1 678 1D1 681 1D1 684 1D1 687 1D1 690 1D1 693 1D1 696 1D1 699 1D1 702 1D1 705 1D1 708 1D1 711 1D1 714 1D1 717 1D1 720 1D1 723 1D1 726 1D1 729 1D1 732 1D1 735 1D1 738 1D1 741 1D1 744 1D1 747 1D1 750 1D1 753 1D1 756 1D1 759 1D1 762 1D1 765 1D1 768 1D1 771 1D1 774 1D1 777 1D1 780 1D1 783 1D1 786 1D1 789 1D1 792 1D1 795 1D1 798 1D1 801 1D1 804 1D1 807 1D1 810 1D1 813 1D1 816 1D1 819 1D1 822 1D1 825 1D1 828 1D1 831 1D1 834 1D1 837 1D1 840 1D1 843 1D1 846 1D1 849 1D1 852 1D1 855 1D1 858 1D1 861 1D1 864 1D1 867 1D1 870 1D1 873 1D1 876 1D1 879 1D1 882 1D1 885 1D1 888 1D1 891 1D1 894 1D1 897 1D1 900 1D1 903 1D1 906 1D1 909 1D1 912 1D1 915 1D1 918 1D1 921 1D1 924 1D1 927 1D1 930 1D1 933 1D1 936 1D1 939 1D1 942 1D1 945 1D1 948 1D1 951 1D1 954 1D1 957 1D1 960 1D1 963 1D1 966 1D1 969 1D1 972 1D1 975 1D1 978 1D1 981 1D1 984 1D1 987 1D1 990 1D1 993 1D1 996 1D1 999 1D1
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.

![Diagram of debouncing types](image-url)
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
```
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
```