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.

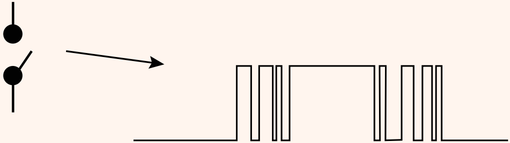


Figure 1: Contact Closure is Anything but Simple

Debouncing Switches

- Falling and rising edge switch bounce from a pushbutton switch

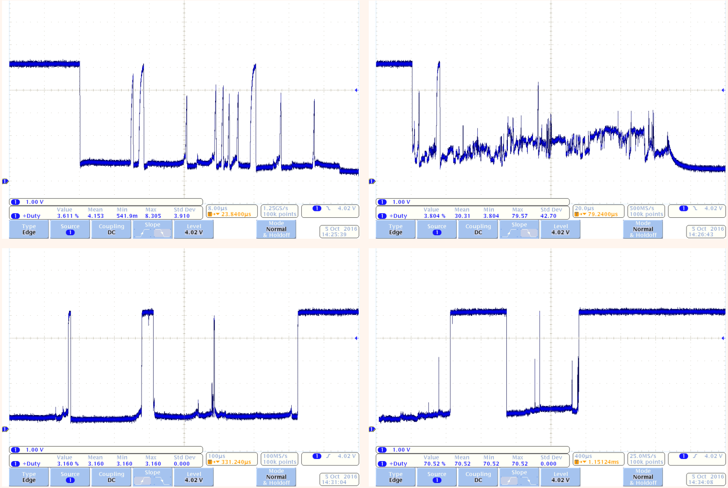


Figure 2: Either Edge Can Bounce

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 (1x10-5/62.5x10-9) 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

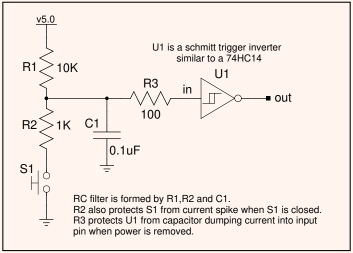
■Specifications

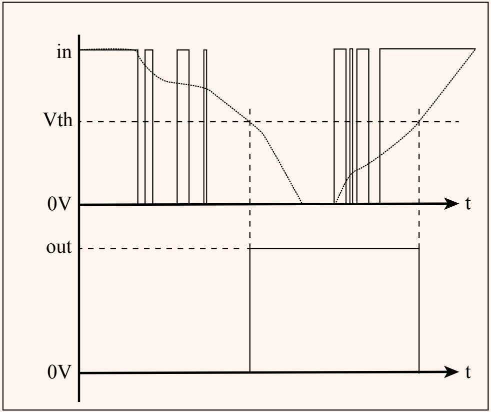
Type		Snap action / Push-on type SPST
Electrical	Rating	$10\mu A$ 2V DC to 50mA 12V DC (Resistive load)
	Contact Resistance	500mΩ max.
	Insulation Resistance	100MΩ min. (at 100V DC)
	Dielectric Withstanding Voltage	250V AC for 1 minute
	Bouncing	10ms max. (ON, OFF)

Figure 3: Specifications for Panasonic EVP-BD6C1A000 pushbutton switch

Debouncing Switches

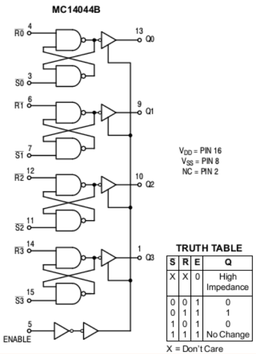
- One possible solution - Analog filtering
- RC network filters out the rapid changes in switch output
- Choose R and C so input threshold is not crossed while input is still bouncing





Debouncing Switches

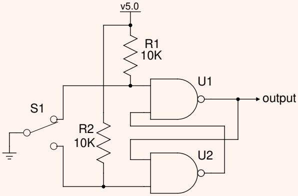
- Another solution would be to use a latch (MC14044)
- Logic gates lock change in $2t_{pd}$ using a SPDT switch
- Both switch (\$3.69) and chip (\$0.38) are expensive
- Momentary click switches (AVR board) are (\$0.12)



TRUTH TABLE

S	R	E	Q
X	X	0	High Impedance
0	0	1	0
0	1	1	1
1	0	1	0
1	1	1	No Change

X = Don't Care



usage model

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

```
//source: Jack Gansel, "Guide to Debouncing"
//returns '1' once per button push, detects falling edge
uint8_t debounce_pulse() {
  static uint16_t state
  state = (state << 1) | (! bit_is_clear(PIND, 2)) | 0xE000;
  if (state == 0xF000) return 1;
  return 0;
}
```

Which pass	Value of state	Return value
first pass after reset	1110 0000 0000 0001	return 0
second pass after reset	1110 0000 0000 0011	return 0
after 12 false passes	1111 1111 1111 1111	return 0
after 7 true passes	1111 1111 1000 0000	return 0
after 12 true passes	1111 0000 0000 0000	return 1
after many true passes	1110 0000 0000 0000	return 0
after 5 false passes	1110 0000 0001 1111	return 0

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

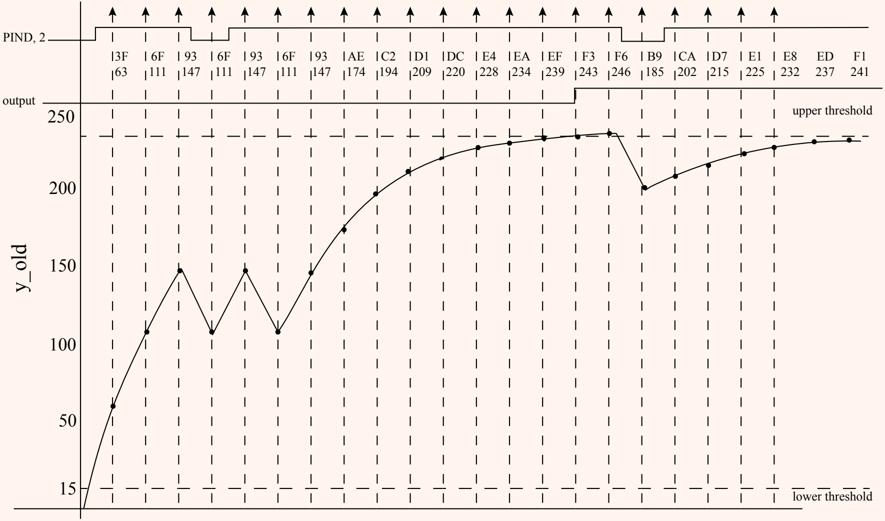


Figure 4: Behavior of IIR Filter with Schmitt Trigger

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.

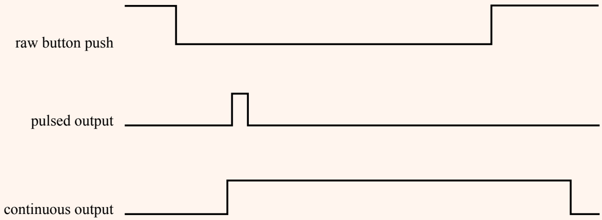
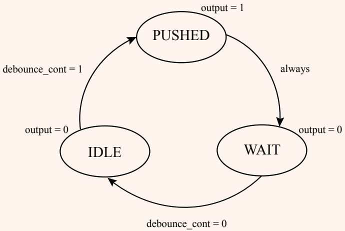


Figure 5: Pulsed versus Continuous Output

Debouncing Switches

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

```
//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.

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