CS/ECE 6780/5780

Al Davis

Today's topics:

•Debouncing switches

•e.g. matrix keypad

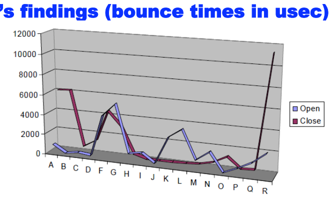
•lab4 issues

Basic Concepts

- Switches are often mechanical
  - move something and
    - » contact is made or broken
  - in either case
    - » metal rebounds
      - causing "hash" oscillations in the observed signal
      - source of massive ISR confusion if you're not careful
- Problem
  - make multiple events look like one event
  - usual solution
    - » hardware debounce
      - extra logic
    - » software debounce
      - focus for this weeks lab
- See "debouncing.pdf" on the class web site
  - figures in the next few slides come from this document
    - » thanks to Jack Ganssle for an interesting read

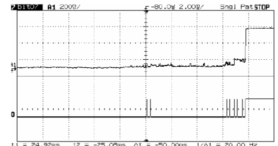
Switch Anatomy

- Lots of types
  - SPST, SPDT, DPDT, and beyond
- How long does a switch bounce
  - varies with switch type and often assymetric w/ open vs. close
    - » typical a few ms but can be as bad as 100's of ms
    - » also varies even for a single switch
      - min to max can vary by 2x or so
- Ganssle's findings (bounce times in usec)



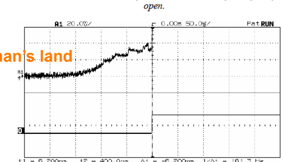
Switches and TTL Sampling Levels

- Aliasing happens in the analog to digital transition



Switch A at 2 msec/div. Note 8 msec of unsettled behavior before it finally decides to open.

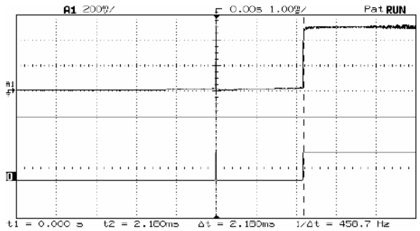
TTL no man's land



Switch C – 50 μsec/div and 200 mV/div.

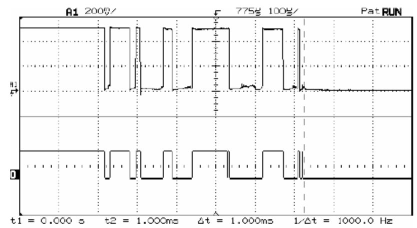
.8 – 2v supposed to be illegal for TTL

Switch G



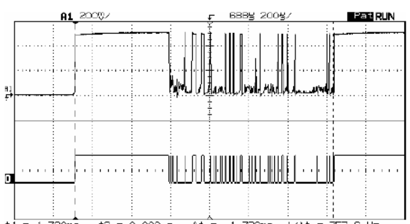
Switch G. One super narrow pulse followed by 2 msec of nothingness. A sure-fire ISR confuser.

Switch O



Switch O, which zaps around enough to confuse dumb debouncers.

Switch Q

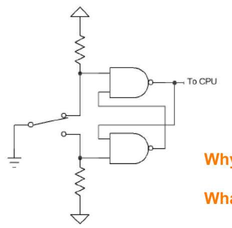


Switch Q – when released, it goes high for 480 μsec before generating 840 μsec of hash, a sure way to blow an interrupt system mad if poorly designed.

Bottom Line

- **In general**
  - **characterize the switches before you use them**
    - » **a thorough test takes a lot of time**
      - **vary how you activate**
      - **take scope traces**
      - **use multiple versions of the same switch**
  - **PCB mounted switches are often better than these somewhat pathological examples**
    - » **but it is wise to check**
  - **weird behavior or intermittent failure**
    - » **suspect your debounce method**

SR Latch HW Debouncer



Why does it work?

What switch property is required?

Downside?

SR Software Equivalent

- **Simplest possible code**
  - **examine both inputs**
    - » one will bounce the other won't
    - » simple loop

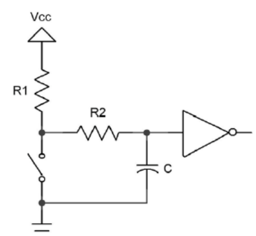
```
if(switch_hi())state=ON;
if(switch_lo())state=OFF;
```

- » problems
  - 2 input capture pins required
  - SPDT switches are more costly and bulky
    - – rarely found on PCB's these days

RC Debouncer

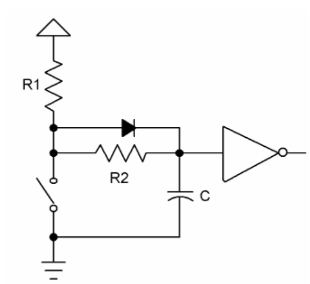
- **Simple**
  - **but hides a lot of complexity**
  - **need to characterize hash time to know desired RC time constant**

What's tricky here?

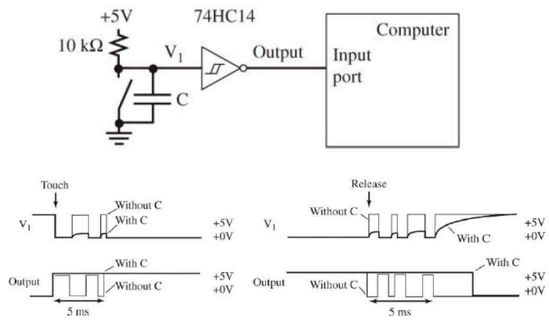


A Better RC Debouncer

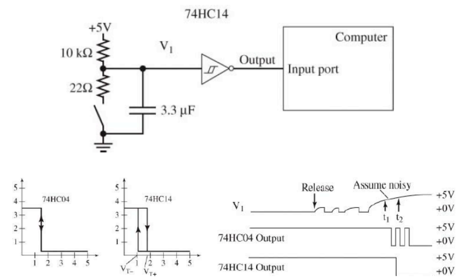
- **Why is this one better?**



Schmitt Trigger Debounce



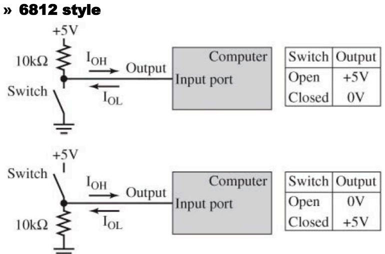
2R Schmitt Debounce



Similar slew decoupling issue but w/ hysteresis

Switch Interfaces

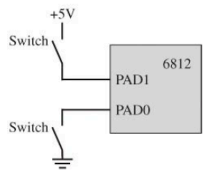
- **HW debouncers make SW's life easier**
  - **but adds to cost**
  - **so let's consider a direct SPST interface w/ SW debounce**
    - » **6812 style**



6812 Ports

- **Ports AD, J, M, P, S and T**
  - **support both internal pull-ups and pull-down resistors**
    - » **note to use port AD as a digital port**
      - **corresponding bits in ATDDIEN must be set**
  - **Port Pull Select Register must be set**
    - » **PPSAD, PPSJ, PPSP, PPSM, PPSS, PPST**
      - **pull-up =0, pull-down=1**
  - **Pull Enable Register**
    - » **PERAD, PERJ, PERP, PERM, PERS, PERT**
      - **enables the pull-up or pull-down function**
  - **Note**
    - » **first set PPSx then PERx**
    - » **if enable happens before select then get signals in possibly the wrong polarity**

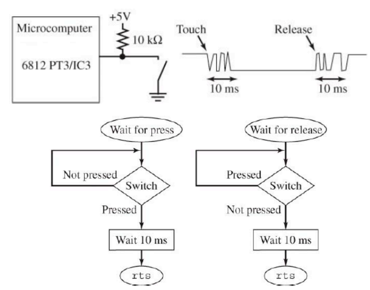
Port AD Initialization Example




```
void PortAD_Init(void){
  ATDDIEN |= 0x03;  // PAD1-0 digital I/O
  DDRAD &= ~0x03;   // PAD1-0 inputs
  PPSAD |=  0x02;   // pull-down on PAD1
  PPSAD &= ~0x01;   // pull-up on PAD0
  PERAD |=  0x03;   // enable pull-up and pull-down
}
```


Software Debounce Model

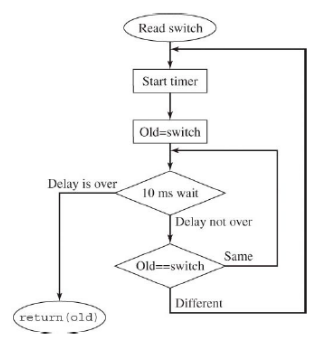
- **Assume bounce time <10ms**



Software Debounce w/ Gadfly Timer

```
void Key_WaitPress(void){
  while(PTT&0x08);    // PT3=0 when pressed
  Timer_Wait10ms(1); // debouncing
}
void Key_WaitRelease(void){
  while((PTT&0x08)==0); // PT3=1 -> released
  Timer_Wait10ms(1);    // debouncing
}
void Key_Init(void){
  Timer_Init();
  DDRT &=~0x08; // PT3 is input
}
```


SW Debounce Version 2



This version returns a new value every time switch position changes

Unified press and release functions

Same <10ms hash assumption

Timer Control & Output Compare

- **Use**
  - **create squarewaves, generate pulses, implement time delays, generate periodic interrupts**
- **6812 has 8 output compare modules**
  - **Each module has**
    - » **external ouput pin (Ocn)**
    - » **flag bit, interrupt mask bit, and 16-bit output compare register**
    - » **force output compare bit (FOCn)**
    - » **two mode bits (OMn OIn)**

Table 15-9. Compare Result Output Action

OMx	OLx	Action
0	0	Timer disconnected from output pin logic
0	1	Toggle OCx output line
1	0	Clear OCx output line to zero
1	1	Set OCx output line to one

MC9S12 reference manual

Output Compare Process Example

- **Basic steps**
  - **read the current 16-bit TCNT**
  - **calculate TCNT+delay**
  - **set output compare register to TCNT+delay**
  - **clear the output compare flag**
  - **wait for the output compare flag to be set**
- **Essentially another SW debounce approach**

Output Compare

```
void Key_Init(void) {
  TIOS  |= 0x20;    // enable OC5 (see Chapter 6)
  TSCR1 = 0x80;     // enable
  TSCR2 = 0x01;     // 500 ns clock
  DDRT &=~0x08;}  // PT3 is input
unsigned char Key_Read(void){
  unsigned char old;
  old = PTT&0x08;      // Current value
  TC5 = TCNT+20000;    // 10ms delay
  TFLG1 = 0x20;        // Clear C5F
  while((TFLG1&0x20)==0){ // 10ms
     if(old!=(PTT&0x08)){ // changed?
       old = PTT&0x08;     // New value
       TC5 = TCNT+20000;}} // restart delay
  return(old); }
```


Debouncing Multiple Switches

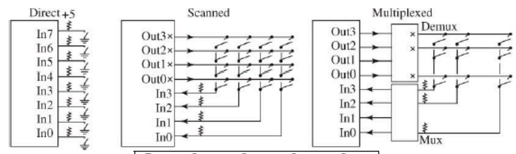
```
#define MAX_CHECKS 10
uint8_t Debounced_State;
uint8_t State[MAX_CHECKS];
uint8_t Index;

void DebounceSwitches(void) {
  uint8_t i,j;
  State[Index] = ReadKeys();
  Index++;
  j = 0xff;
  for(i=0;i<MAX_CHECKS-1;i++) {
    j &= State[i];
  }
  Debounced_State ^= j;
  if(Index >= MAX_CHECKS) { Index = 0; }
}
```

Based on "My favorite software debouncers" by Jack Gannsle.

Interfacing Multiple Keys

- **3 basic methods**
  - **direct – input pin per switch**
    - » **downside is what happens if you have more switches than input pins**
    - » **upside – you can recognize every possible switch combination**
      - **note this doesn't matter in a keyboard where one switch is pressed at a time**
        - **or very few – e.g. Shift, CTL, FN, …**
  - **scanned**
    - » **keys belong to a matrix**
      - **know the row and column and you know which key**
      - **6812 drives one row low at each step (enables the row)**
        - **column values indicate which key in that row was pushed**
  - **multiplexed**
    - » **same idea but uses less pins (e.g. $\log_2 n$)**
      - **put out binary value of the row**
      - **demux generates the one-hot code similar to the scanned mode**
      - **mux on the way back in does the symmetric function**

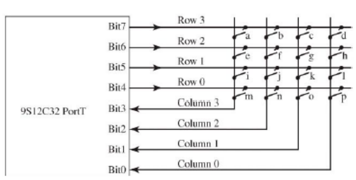
3 Approach View



Row	Out3	Out2	Out1	Out0
3	0	HiZ	HiZ	HiZ
2	HiZ	0	HiZ	HiZ
1	HiZ	HiZ	0	HiZ
0	HiZ	HiZ	HiZ	0

Row	Out3	Out2	Out1	Out0	15	14	...	0
15	1	1	1	1	0	HiZ	...	HiZ
14	1	1	1	0	HiZ	0	...	HiZ
...	...	...	...	...	...	...	...	...
0	0	0	0	0	HiZ	HiZ	...	0

4x4 Scanned Keypad



Row 3	Row 2	Row 1	Row 0	Col 3	Col 2	Col 1	Col 0
0	1	1	1	a	b	c	d
1	0	1	1	e	f	g	h
1	1	0	1	i	j	k	l
1	1	1	1	m	n	o	p

4x4 Keypad

- **Two steps to scan a particular row:**
  - **select row by driving it low**
    - » **other rows stay Hi-Z**
  - **read the columns to discover which key is pressed**
    - » **0→ pressed in this case due to pull-up**
- **Works if**
  - **no key is pressed**
  - **1 key is pressed**
  - **2 keys are pressed**
    - » **note general case would allow up to 4**

4x4 Handler Code

```
const struct Row
{ unsigned char direction;
  unsigned char keycode[4];}
typedef const struct Row RowType;
RowType ScanTab[5]={
{   0x80, "abcd" }, // row 3
{   0x40, "efgh" }, // row 2
{   0x20, "ijkl" }, // row 1
{   0x10, "mnop" }, // row 0
{   0x00, "    " }};
void Key_Init(void){
  DDRT = 0x00; // PT3-PT0 inputs
  PTT = 0;     // PT7-PT4 oc output
  PPST = 0;     // pull-up on PT3-PT0
  PERT = 0x0F;}
```

continued next slide

4x4 Code (cont'd)

```
/* Returns ASCII code for key pressed,
   Num is the number of keys pressed
   both equal zero if no key pressed */
unsigned char Key_Scan(short *Num){
  RowType *pt; unsigned char column,key;
  short j;
  (*Num)=0; key=0;    // default values
  pt=&ScanTab[0];
  while(pt->direction){
    DDRT = pt->direction;  // one output
    column = PTT;   // read columns
    for(j=3; j>=0; j--){
      if((column&0x01)==0){
        key = pt->keycode[j];
        (*Num)++;}
      column>>=1;} // shift into position
    pt++; }
  return key;}
```


Concluding Remarks

- **Controller sits in a sea of I's and O's**
  - **might be a tight connection – e.g. keypad**
    - » **O's say what we care about**
    - » **I's say given what you care about this is what happened**
- **Output compare tied to inputs are useful**
  - **6812 supports them**
- **All switches are not created equal**
  - **need to understand what you're working with**
    - » **then you'll know the debounce strategy**
  - **fortunately the 6812 understands most of this inequality**
    - » **and provides relatively simple & useful interface options**
- **Non-switch interfaces**
  - **analog input values**
    - » **must convert to digital via AD port**
  - **digital inputs – these are the simple ones**