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 week's 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 asymmetric 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
  - TTL no man's land
  - .8 – 2v supposed to be illegal for TTL
Switch G

Switch G: One super narrow pulse followed by a wave of nothingness. A sure-fire ISR confusor.

Switch O

Switch O, which zigzags enough to confuse dumb debounceers.

Switch Q

Switch Q—when released, it goes high for 400 µsec before generating 240 µsec of lurch. A sure way to blow an inner-space module if poorly designed.

Bottom Line

• In general
  • characterize the switches before you use them
    • 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 with 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 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, PERB, 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(PTTA0x08); // PT3=0 when pressed
    Timer_Wait10ms(); // debouncing
}
void Key_WaitRelease(void){
    while(PTTA0x08); // PT3=1 -> released
    Timer_Wait10ms(); // 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 output pin (Ocn)
    - flag bit, interrupt mask bit, and 16-bit output compare register
    - force output compare bit (FOcn)
    - two mode bits (OMn Olm)

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Timer disconnected from output pin logic</td>
</tr>
<tr>
<td>1</td>
<td>Toggle Ocn output line</td>
</tr>
<tr>
<td>1</td>
<td>Clear Ocn output line to zero</td>
</tr>
<tr>
<td>1</td>
<td>Set Ocn output line to one</td>
</tr>
</tbody>
</table>

**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**

```c
void Key_Init(void)
{
    TIDS = 0x20;  // enable DC5 (see Chapter 6)
    TS8CM = 0x80;  // enable
    TS8CN = 0x01;  // 500 ms clock
    EDTO &= 0x08;  // PT3 is input
    unsigned char Key_Read(void)
    {
        unsigned char old;
        old = PTTb0x08;  // Current value
        TC5 = TC5+20000;  // 10ms delay
        TFLO = 0x20;  // Clear CSP
        while((TFLGI&0x20)==0)  // 10ms
            if(old!=(PTTb0x08)) // changed?
                old = PTTb0x08;  // New value
                TC5 = TC5+20000;  // restart delay
            return(old);
    }
```

---

**Debouncing Multiple Switches**

```c
#define MAX_CHECKS 10
int Debounced_State;
int Debounced_State[MAX_CHECKS];
int Index;

void DebounceSwitches(void) {
    unsigned int I;
    State[0] = ReadKeys();
    Index++;
    if(Index==MAX_CHECKS-1){
        I = 0;
        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 Gaemle.
Interfacing Multiple Keys

3 basic methods
- **direct** – input pin per switch
  - downside: 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, Ctrl, F1, ...
- **scanned**
  - keys belong to a matrix
    - know 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

4x4 Scanned 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
    - 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

```c
const struct Row
{
    unsigned char direction;
    unsigned char keycode[4];
} RowType;
RowType ScanTab[6]={
    { 0x00, "abcd" }, // row 3
    { 0x40, "efgh" }, // row 2
    { 0x20, "ijkl" }, // row 1
    { 0x10, "mnop" }, // row 0
    { 0x00, " " },
};
void Key_Init(void)
{
    D0RT = 0x00; // PT3-PT0 inputs
    PTT = 0; // PT7-PT4 oc output
    PRRT = 0; // pull-up on PT3-PT0
    PERN = 0x0F;
}
```

continued next slide

4x4 Code (cont'd)

```c
/* 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
    pr=45ScanTable; 
    while(pt=pr+direction) { 
        D0RT = pt->direction; // one output 
        column = PT7; // read column 
        for(j=3; j>0; j--) { 
            if(column==PT0) { 
                key = pt->keycode[0];
            } 
            else { 
                column+=4; // 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