Lecture 6: Debouncing and Matrix Keypads
Interfacing a Switch to a Computer

- **Switch**
- **10kΩ**
- **Input port**
- **Computer**
- **Output**
- **Switch**
- **IOH**
- **IOL**

<table>
<thead>
<tr>
<th>Switch</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open</td>
<td>+5V</td>
</tr>
<tr>
<td>Closed</td>
<td>0V</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Switch</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open</td>
<td>0V</td>
</tr>
<tr>
<td>Closed</td>
<td>+5V</td>
</tr>
</tbody>
</table>
Port Pull Configuration

Port Pull Select Register (PPSAD, PPSJ, PPSP, PPSM, PPSS, PPST) select pull-up (0) or pull-down (1) for the port. Pull Enable Register (PERAD, PERJ, PERP, PERM, PERS, PERT) enables the pull-up or pull-down function. You should first set the PPSx register and then the PERx register to avoid glitches.
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
}
Switch Bounce

The diagram shows a basic circuit involving a switch, resistor, and computer input port. The switch is connected to a +5V source through a 10kΩ resistor. When the switch is touched, it completes the circuit, causing the output to go high (+5V). The output remains high for 5 ms before returning to its initial state, which is +0V. This process is repeated when the switch is released, with another 5 ms of high output before returning to +0V. The circuit is designed to prevent switch bounce, ensuring stable output signals to the computer.
Hardware Debouncing Using a Capacitor

The diagram shows a circuit with a 74HC14 integrated circuit connected to a computer input port. A 10 kΩ resistor is connected to a 5V power supply, and a capacitor C is used to debounce the signal. The waveforms illustrate the voltage levels (V1) for different states: touch, without C, with C, release without C, and output with and without C.
Hardware Debouncing Using a Capacitor (cont)
Software Debouncing

Diagram:
- Microcomputer input: 6812 PT3/IC3
- 10 kΩ resistor
- Input voltage +5V
- Switch state
  - Touch: 10 ms
  - Release: 10 ms

Flowchart:
- Wait for press
  - Not pressed
    - Switch: Pressed
      - Wait 10 ms
      - rts
  - Pressed
    - Wait 10 ms
    - rts
- Wait for release
  - Pressed
    - Switch: Not pressed
      - Wait 10 ms
      - rts
  - Not pressed
    - Wait 10 ms
    - rts
void Key_WaitPress(void){
    while(P TT & 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
}
Software Debouncing

1. Read switch
2. Start timer
3. Old = switch
4. Delay is over
   - 10 ms wait
   - Delay not over
     - Old == switch
       - Same
       - Different
5. return(old)
Output Compare Basics

Used to create squarewaves, generate pulses, implement time delays, and execute periodic interrupts.
6812 has 8 output compare modules.
Each module has:
- An external output pin (OCn).
- A flag bit.
- Force compare control bit (FOCn).
- Two control bits (OMn, OLn).
- An interrupt mask bit.
- 16-bit output compare register.
Output Compare Process Example

Read the current 16-bit TCNT.
Calculate TCNT+delay.
Set the 16-bit output compare register to TCNT+delay.
Clear the output compare flag.
Wait for the output compare flag to be set.
Another Approach to Software Debouncing

```c
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); }
```
#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.
Basic Approaches to Interfacing Multiple Keys

<table>
<thead>
<tr>
<th>Row</th>
<th>Out3</th>
<th>Out2</th>
<th>Out1</th>
<th>Out0</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>HiZ</td>
<td>HiZ</td>
<td>HiZ</td>
</tr>
<tr>
<td>2</td>
<td>HiZ</td>
<td>0</td>
<td>HiZ</td>
<td>HiZ</td>
</tr>
<tr>
<td>1</td>
<td>HiZ</td>
<td>HiZ</td>
<td>0</td>
<td>HiZ</td>
</tr>
<tr>
<td>0</td>
<td>HiZ</td>
<td>HiZ</td>
<td>HiZ</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Row</th>
<th>Out3</th>
<th>Out2</th>
<th>Out1</th>
<th>Out0</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

...
4 by 4 Scanned Keypad

<table>
<thead>
<tr>
<th>Row 3</th>
<th>Row 2</th>
<th>Row 1</th>
<th>Row 0</th>
<th>Col 3</th>
<th>Col 2</th>
<th>Col 1</th>
<th>Col 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>m</td>
<td>n</td>
<td>o</td>
<td>p</td>
</tr>
</tbody>
</table>
There are two steps to scan a particular row:
   Select that row by driving low while other rows are not driven.
   Read the columns to see if any keys are pressed in that row
   (0 means key pressed, 1 means not pressed).

The scanned keypad operates properly if:
   No key is pressed.
   Exactly one key is pressed.
   Exactly two keys are pressed.
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;}
/* 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++; }               // shift into position
  return key;}