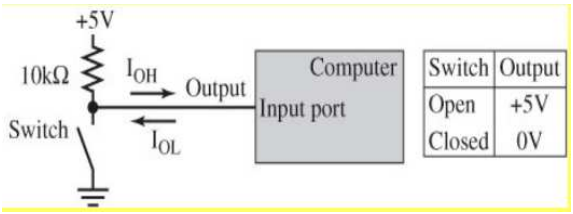
ECE/CS 5780/6780: Embedded System Design

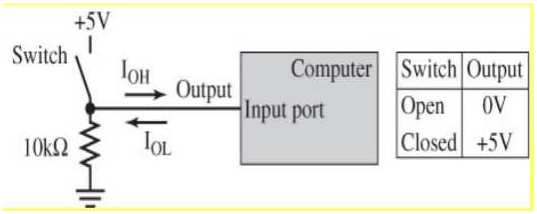
Chris J. Myers

Lecture 5: Debouncing and Matrix Keypads

Interfacing a Switch to a Computer

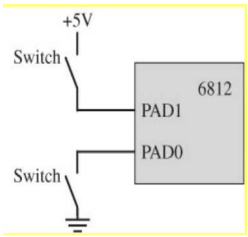


Switch	Output
Open	+5V
Closed	0V



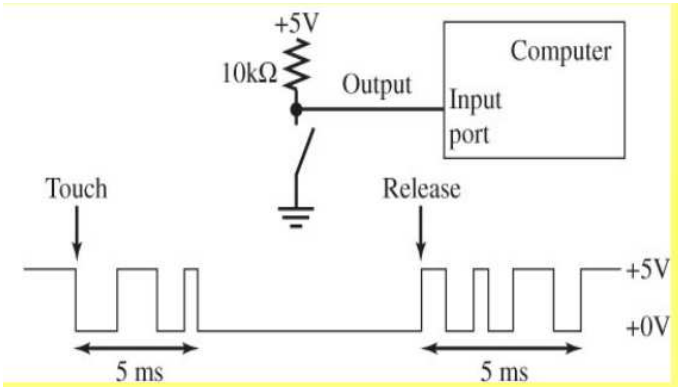
Switch	Output
Open	0V
Closed	+5V

Port AD Initialization Software

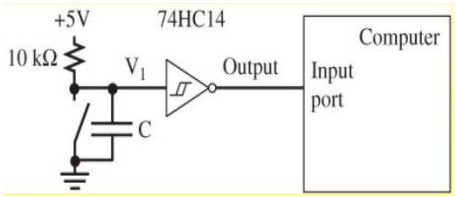


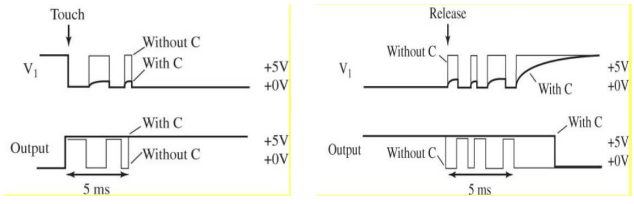

```
void PortAD_Init(void){
  ATDDIEN |= 0x03;  // PAD1-0 digital I/O
  DDRAD &= ~0x03;   // PAD1-0 inputs
  PPSAD |=  0x02;   // pull-down on PAD1
  PPSAD &= ~0x01;   // pull-up on PAD0
  PERAD |=  0x03;   // enable pull-up and pull-down
}
```


Switch Bounce

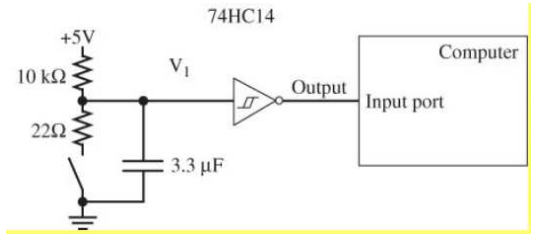


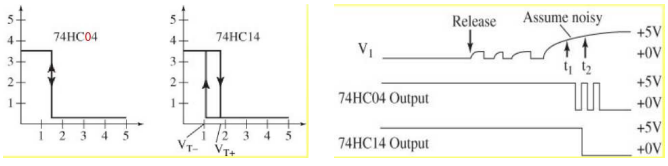
Hardware Debouncing Using a Capacitor



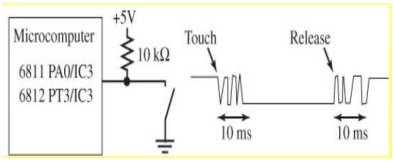


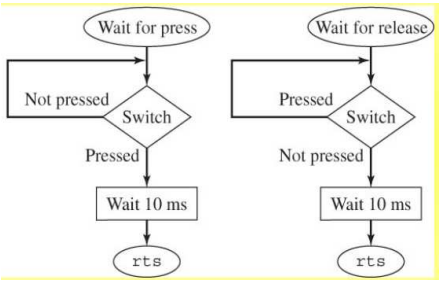
Hardware Debouncing Using a Capacitor (cont)





Software Debouncing

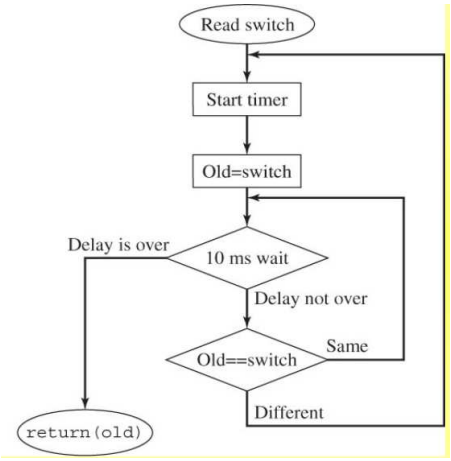




Software Debouncing with Gadfly Synchronization

```
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
}
```

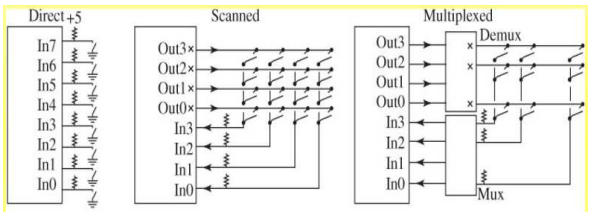

Software Debouncing



Another Approach to Software Debouncing

```
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); }
```

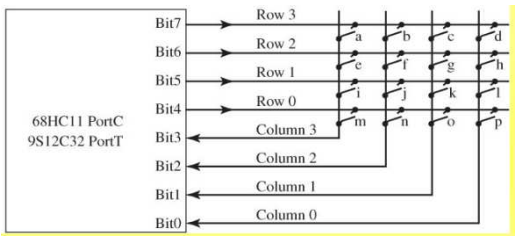

Basic Approaches to Interfacing Multiple Keys



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

4 by 4 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

4 by 4 Scanned Keypad

- There are two steps to scan a particular row:
  1. Select that row by driving low while other rows are not driven.
  2. Read the columns to see if any keys are pressed in that row (0 means key pressed, 1 means not pressed).
- The scanned keyboard operates properly if:
  1. No key is pressed.
  2. Exactly one key is pressed.
  3. Exactly two keys are pressed.

Software for a Matrix Scanned Keypad

```
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;}
```


Software for a Matrix Scanned Keypad (cont)

```
/* 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;}
```