# Embedded Controller Programming II

*I/O Device Programming in C*
*Part 1: Input and Interrupts*

Ken Arnold


051221



## Overview

- Basic Input Devices
- Switch Input
- Matrix Keypad Input
- Multiplexing
- Interrupt Driven I/O





## I/O Types

- Standard Peripherals
  - General Purpose I/O Pins
  - Serial Port, Counter/Timers
  - Memory Mapped I/O
  - Enhanced Peripherals:
    - PWM, A/D, PCA
- Bus Interfaces:
  - SPI/Microwire, $I^2C$, USB, CAN, 1-Wire

3

## Input

- Input (Reading) the I/O Ports
  - Set I/O Line High (1) to use as Input
  - External Circuit Pulls I/O Pin Low for 0
  - Switches Must Be Debounced
- Reading a Matrix Keypad


4

## Switch Inputs

- 8051 Ports have internal pull-ups
  - Connect switch between I/O pin and Gnd
  - Input=1 when Switch is OFF
  - Input=0 when Switch is ON
- Switches Bounce!
  - Typical bounces last 1 to 50 milliseconds
  - Occurs most when contacts close

5

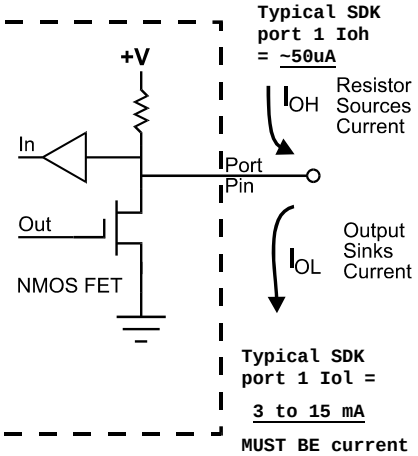
## Port I/O *for the 8051 family*

- Ports are SFRs in Internal Data Memory
- Port is 0xff after Reset (input mode)
- Port 1 is uncommitted - location 0x90
- Bit addressable from bits 0x90 to 0x97
  - LSB address = 0x90, MSB address = 0x97
- SDK Port 1 on DIP plug pins 1..8
  - P1 = 0xff  // sets all bits of Port 1 high

6

# Simplified Port Circuit

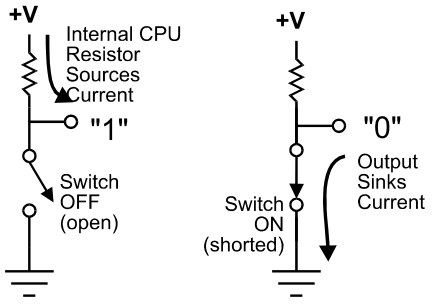
- Modified Open-Drain
  - Constant current pull-up (Current Source)
  - Switch is a pull-down (FET Sinks Current)
- Active Switch sinks more current, so
- Sink current is large compared to source




7

# Switch Input Circuit

- Internal CPU pull-up for sourcing current
  - Switch open
  - Logic level = 1 (high)
- Switch ON pulls down pin, sinking current
  - Switch closed
  - Logic level = 0 (low)

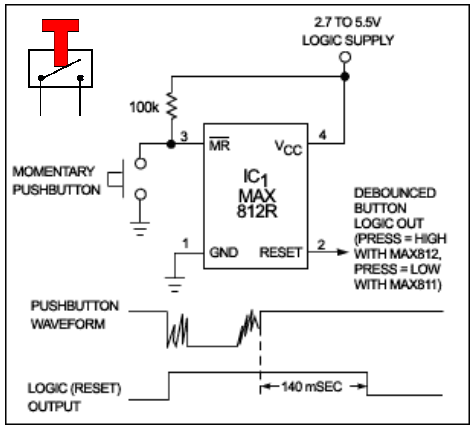



- ***Switch Contacts Bounce!***

8

## Switch Bounce

- Mechanical Contacts Bounce, Due to Elastic Collision
- Can be De-Bounced with Hardware:
- Usually Done in Code




9

## Debouncing with Software

- Look for first transition low (ON)
- Ignore additional switch transitions for debounce period, typically ~50 mS
- If switch is still on, key is down.
- Look for first transition high (OFF)
- Ignore transitions for ~50 mS
- If switch is still high, key is up.

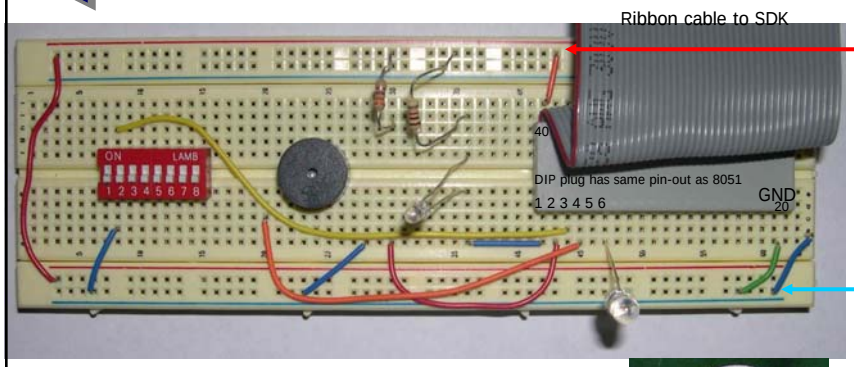
10

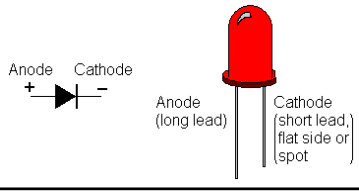
## Switch and Keyboard Input

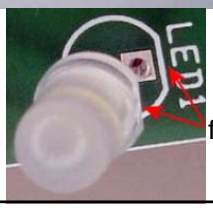
- Switches, Keypads
  - Simple switch inputs
  - Multiplexed Keypads
  - Multiplexed Keyboards
- Intelligent Keyboards
  - PC Style
  - ASCII

11

## Switch & LED

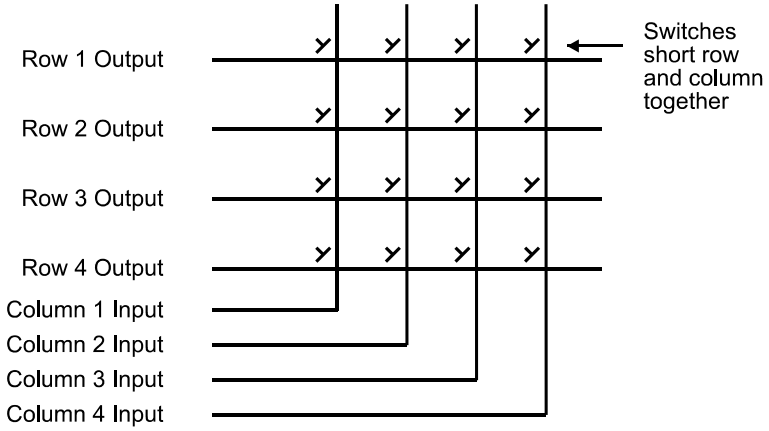



12

## Simple Switch Matrix




13

## Switch Matrix

- Multiplexed row and column:
- Switches organized in Rows/Columns
- Switch Shorts row line to column line
- Walking zero on rows to activate one row at a time
- Check for low level on column inputs to determine which key is pressed

14

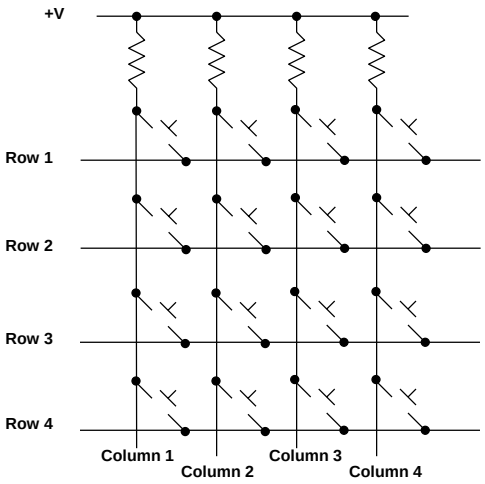
## Switch Matrix

- Switches organized as Row/Column
- Switch shorts row line to column line
- Walking zero on rows to activate one row at a time
- Check for low level on column inputs to determine which key in the current row is pressed

15

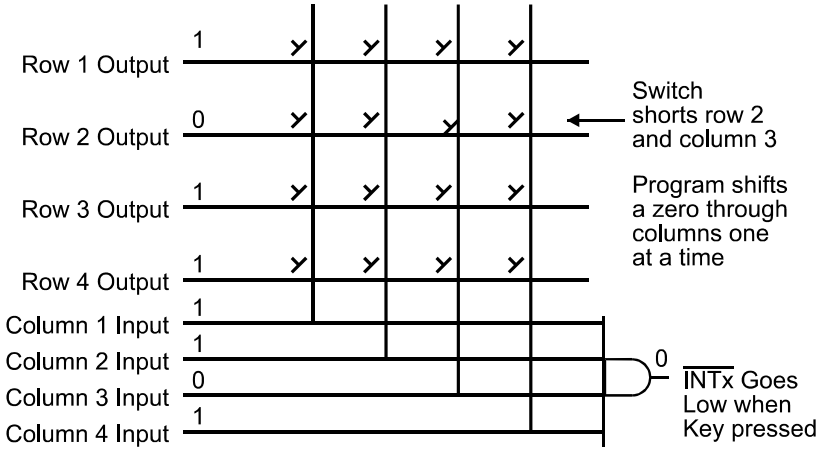
## Matrix Keypad Interface

- Generic Keypad
- Requires Pull-ups
- Not Required for I/O Pins with Built-in Pull-ups



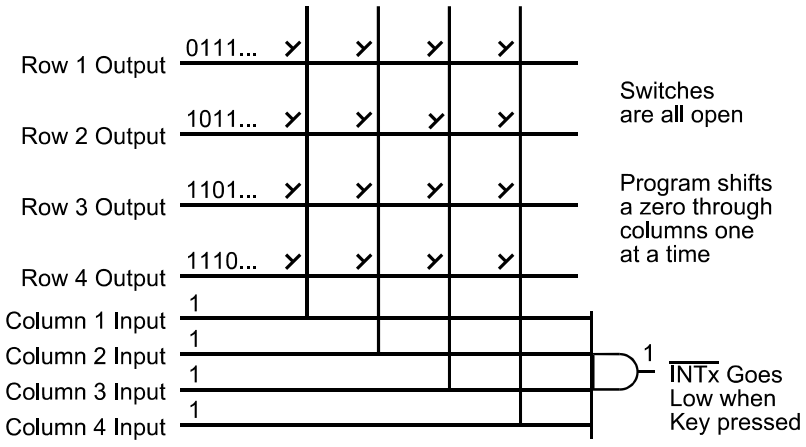

16

## Multiplexed Switch Matrix




17

## Scanning the Switch Matrix



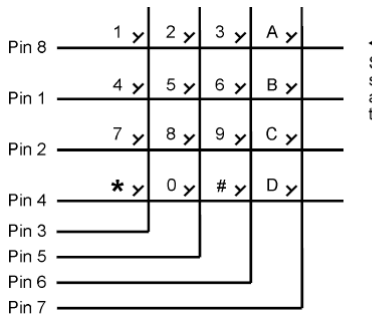

18

## Why Multiplex I/O?

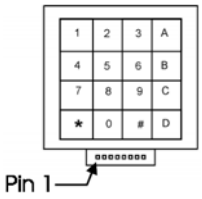
- It saves the limited I/O pins on a uC
- Non-multiplexed requires 1 bit per I/O:
  - 16 switches requires 16 I/O pins
- Multiplexing shares I/O lines:
  - 16 switches require 4+4 = 8 lines
- This can be done for displays as well, due to persistence of vision effect

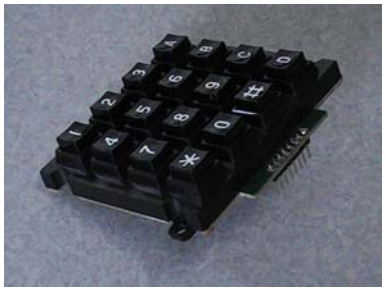
19

## Keypad Connections




Switches short row and column together



Keypad is jameco.com part #169244, with an 8 pin single row male header, with pin spacing = 0.1" soldered to the keypad, so it can be plugged directly into the solderless proto board.

20

## Keypad Connections to CPU:

| CPU Signal | Keypad | CPU Pin # | Keypad Pin # |
|---|---|---|---|
| P1.0 | Row 2 | 1 | 1 |
| P1.1 | Row 3 | 2 | 2 |
| P1.2 | Col 1 | 3 | 3 |
| P1.3 | Row 4 | 4 | 4 |
| P1.4 | Col 2 | 5 | 5 |
| P1.5 | Col 3 | 6 | 6 |
| P1.6 | Col 4 | 7 | 7 |
| P1.7 | Row 1 | 8 | 8 |

Connections are shown for the sample keypad1.c program, which uses a software driven, bit-banged interface.

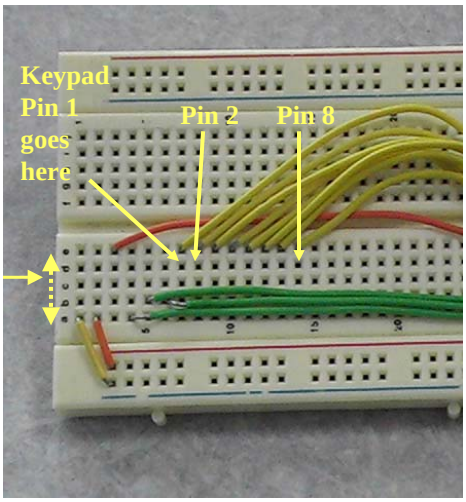
21

## Connections to Keypad

- Keypad wiring
- For sample program:

  keypad1.c

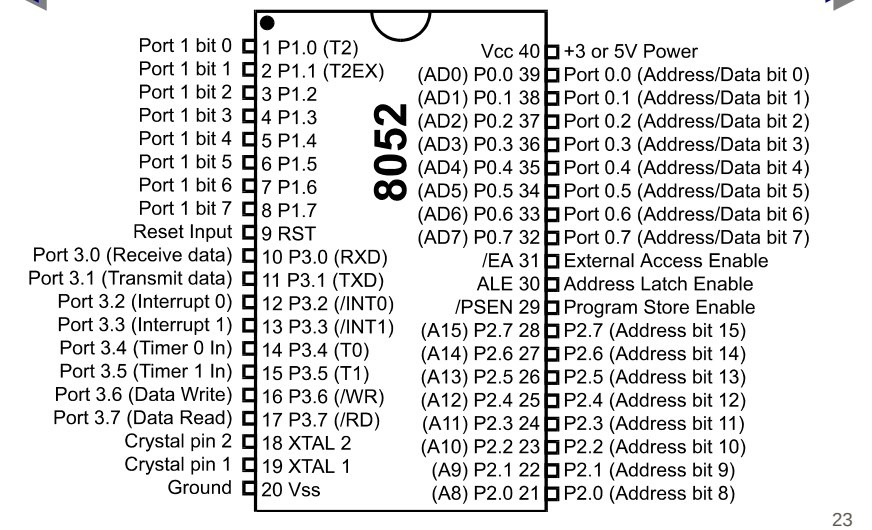
http://hte.com/uconline/ecp/keypad1.exe




Only the 8 yellow wires are needed for this example – the other wires are used for the LCD demonstration.

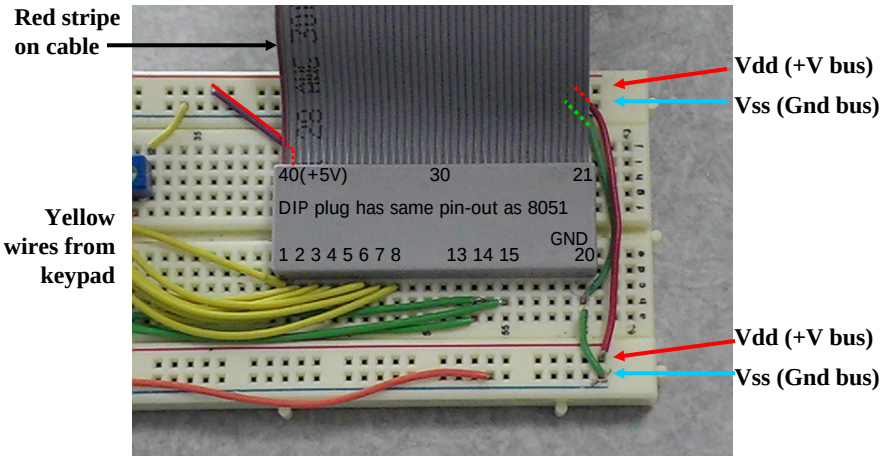
22

# 8052 Pin Definitions



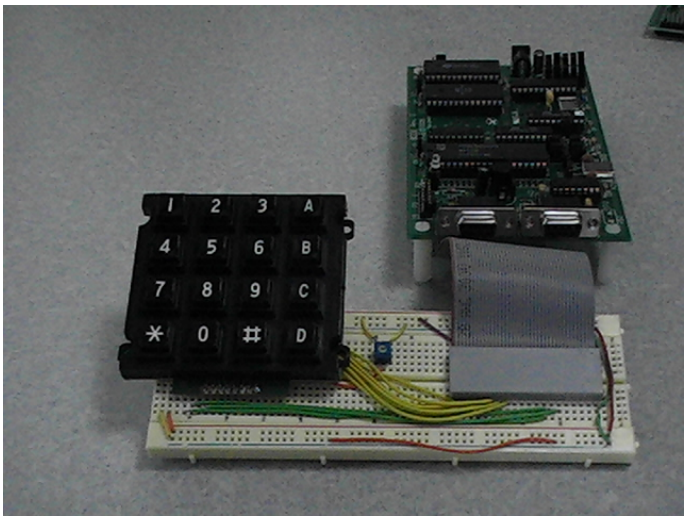

23

# Connections to SDK Cable




24

## SDK Cable to Keypad




25

## Scanning a Keypad

```
unsigned char code row[] = {0xEF, 0xDF, 0xBF, 0x7F };
             /* for strobing each row */
    /* 4 MSBs are row outputs, 4 LSBs are col inputs */

unsigned int bdata keys /* for storing key info */

void keypress (void) interrupt 0 // ISR for /INT0
  {
   unsigned char I;
   keys = 0xFFFF;  // get ready to input key press data
   for (I=0; I <= 3; I++)
    {
     P1 = row[I];  // strobe low one row at a time
     keys = (keys<<4) | (P1&0x0F);
        // read and store info from each column
    }
```

26

## Keyboards - Matrix

- Reduces the number of I/O pins req'd
- Software Driven is lowest cost
  - Increases Software Complexity
  - MUST debounce in software
- Hardware Multiplexed (e.g. 74xx922)
  - Independent, interrupt driven
  - Hardware compatibility issues

27

## Hardware Abstraction

- I/O Driver Characteristics:
  - Hardware Design Specific
  - Processor Architecture Dependent
  - Uses Compiler-Unique Extensions (Non-ANSI)
- Separate and Organize Your Code:
  - ANSI Standard Application Code
  - Hardware Abstraction Layer Isolates ANSI Application Code From non-ANSI code

28

## Standard Port Input

**Hardware Abstraction Example**

**Data Input via Port 1 (8 position DIP switch)**

```
#include <reg51.h>
char 8DIPswitches (void)
  {
    return (P1); /* return value of P1 */
  }
```

**This code should be identified as hardware specific…**

29

## Standard Port I/O

**Data Output via an 8 bit wide Port;**

```
void 8LEDs (char outval )
  {
   P1 = outval;
  }
```

**This code should be the only code that outputs to the LEDs – other ANSI standard C functions can call this function which uses non-ANSI features of the compiler.**

30

## Serial Port Setup

```
// Configure Serial Port & Timer
void setup_serial (void)
    {
     SCON = 0x40; /* 8 bit UART */
     TMOD = 0x21; /* 0x21;
     TH1 = TL1 = 0xF3; /* 1200 baud at 12 Mhz */
     TR1 = 1;  /* Start Timer 1 */
     TI = 1;
    }
```

31

## Serial Port Output

```
void main (void)
  {      // Send Serial Data …
   setup_serial();  // initialize the port
   while (1)
     {
        while (TimerOV==0)   // until timer overflow
        send_serial(*string);  // send string
     }
   }
```

32

# Crude Software Time Delay (bad idea!)

```
/* FROM FILE time.c */
 /*********************************************************
  *              FUNCTION delaymsec                       *
  *  This function is used to generate a delay of from 1  *
  * to 255 milliseconds based on the value of ms passed to *
  * it.  The functional delay is based on the basic count  *
  * equation for c1 and c2 where;                          *
  *                                                        *
  *    delay = approx ms*(9+3*count1+2*count1*count2)+10   *
  *  This delay includes call and return times.            *
  *  To change delay cycles, use equation for inner loop   *
  *  expression;                                           *
  *                                                        *
  *         for (count1=11;count1>0;count1--)              *
  *          for (count2=40;count2>0;count2--);            *
  *                                                        *
  *  For this inner loop, the approximate delay is;        *
  *              cycles = 1+3*count1+2*count1+count2       *
  *                                                        *
  *********************************************************/
```

33

# Crude Time Delay (cont'd)

```
void delaymsec (unsigned char ms)
 {
  unsigned char i,count1,count2;
  for (i=ms; i>0; i--)
   {  /* The delay for this function is ~922 cycles with */
      /* count1=11, count2=40                            */
    for (count1=11;count1>0;count1--)
      for (count2=40;count2>0;count2--);
   }
 }

void delaysec (unsigned char sec)
 {
  unsigned char i,j;
  for (j=0; j<sec; ++j)
    for (i=0; i<4; ++i)
      delaymsec(250);
 }
```

**Delay depends on Compiler options, version, and more... delay = = Unpredictable!!**

34

## The Right Way to Delay

- Use a timer, and its interrupt
- Interrupt on Timer Overflow
- Count the number of overflows in Interrupt Service Routine (ISR)
- Do something when the delay time has expired

35

## What is an Interrupt?

- Imagine a pushbutton that forces the CPU to call a specific program subroutine.
  - When the switch is closed, it effectively CALLs a special subroutine (the Interrupt Service Routine, or ISR).
  - This routine can be called independently of the instruction executing when it occurs.
  - This is literally what is done for many microcontroller interrupt applications.

36

## Hardware Interrupts

- Hardware Interrupts: a “hardware” CALL
  - Asynchronous to program
    - Can occur at any point in program execution
    - Timing is unrelated to main program execution
  - Forces the CPU to call interrupt subroutine
    - **I**nterrupt **S**ervice **R**outine (ISR)
  - Allows timely response to an event
    - Important events are processed on demand

37

## 8051 Interrupt Structure

- Multiple, Nested, Two Priorities
- Fixed Interrupt Vectors
- Interrupt Sources:
  - External: Interrupt /INT0, /INT1
    - Level or Edge Sensitive Programmable
  - Internal:
    - Timer/Counters 0, 1, 2
    - Serial I/O

38

## Interrupt Events

- External Interrupt Sources
  - /INT0, /INT1, Low Level or Falling Edge
- Internal Interrupt Sources
  - Timers 0, 1 and 2 on Overflow
  - Serial Port Transmit Empty or Receive Full
  - Others: A/D conversion complete...
- Reset (sort of…)
  - Non-maskable, forces jump to addr Zero

39

## External Interrupts

- 8051 Has Two:
  - External Interrupt 0 and 1 (/INT0, /INT1)
    - IE0 is interrupt 0 flag, IE1 is interrupt 2 flag
    - /INT0 vector starts at base+003, INT1 base + 13h
  - Level or Edge triggered
- Using external interrupts
  - Global Enable (EA “Enables All” interrupts)
  - Must also be individually enabled
  - Must be configured
    - Priority (IP Register)
    - Edge/level Sensitive: IT0, IT1
  - May need to be debounced

40

## Timer Interrupt

```
#include <reg51.h>
#include <stdio.h>
#define RELOADH 0x3C
#define RELOADL 0xB0
extern unsigned int tick_count;

void timer0 (void) interrupt 1
{
     tick_count++;  // increment global tick count
}
```

41

## Interrupt Functions

```
// external interrupt 0:
//   the /INT0 pin ISR Function

void int0 (void) interrupt 0
{
   LED = ON;  // indicate start of ISR execution
   ea = 1;  // allow higher level ints to occur
   /* ... Handle external event here */
   LED = OFF; // indicate end of ISR execution
}
```

42

## 8xx2 Interrupt Vectors

- Int# Address, Source:
- 0 0003h, IE0 - External Interrupt 0
- 1 000Bh, TF0 - Timer 0 OF
- 2 0013h, IE1 - External Interrupt 1
- 3 001Bh, TF1 - Timer 1 OF
- 4 0023h, TI or RI - Serial Tx/Rx
- 5 002Bh, TF2 - Timer 2 OF

may be others...

43

## Reset and Interrupt Vectors

```
                ; SDK Interrupt Vectors offset by 4000h
base  equ  4000h      ; SDK code in RAM
      org  base       ; addr of user code
      jmp  start      ; Reset entry point
      org  base+3     ; Int vector 0: External INT0
      jmp  int0isr    ; jump to INT0 ISR
      org  base+0bh   ; Int vector 1: Timer0
      jmp  tmr0isr    ; jump to Timer0 ISR
      etc...          ; vectors every 8 bytes
```

44

## Interrupt Enable IE 0A8h

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| EA | - | ET2 | ES | ET1 | EX1 | ET0 | EX0 |

- EA Enable All Interrupts (Global enable)
- ES Enable Serial Port
- ETx Enable Timer x
- EXx Enable External Interrupt x

45

## Interrupt Priority IP 0B8h

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | - | PT2 | PS | PT1 | PX1 | PT0 | PX0 |

- 0==Low Priority, 1==High Priority
- PS Enable Serial Port
- PTx Enable Timer x
- PXx Enable External Interrupt x

46

## Priority (cont'd)

- Within a level, simultaneous interrupts are processed according to priority shown:
  - 1: IE0  External Interrupt 0
  - 2: TF0 Timer Interrupt Flag 0
  - 3: IE1 External Interrupt 1
  - 4: TF1 Timer Interrupt Flag 1
  - 5: RI or TI Transmit/Receive Interrupt
  - 6: TF2 Timer Interrupt Flag 2

47

## Clearing Interrupt Requests

- These are cleared when CPU calls ISR:
  - IE0 and IE1 External IRQ Flag
  - TF0, TF1, TF2 Timer IRQ Flags
- Must be explicitly cleared by program:
  - TI Serial Port Transmit Buffer Empty
  - RI Serial Port Receive Buffer Full
  - ex: CLR TI or CLR RI inside the serial ISR

48

## External Interrupt Requests

- IE0 External Interrupt 0 Flag
- IE1 External Interrupt 1 Flag
  - IE0 in TCON.1, IE1 in TCON.3
  - Detects when INTx input pin goes low
  - IEx flags set when the INTx pin goes low
  - IEx Flags are cleared when ISR is called

49

## Timer Control TCON 088h

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| TF1 | TR1 | TF0 | TR0 | IE1 | IT1 | IE0 | IT0 |

- TFx Timer x Overflow Flag (input)
- TRx Timer x Run Enable (control)
- IEx Ext Int Detect Flag (input)
- ITx Ext Int Edge Sensitive 1=Edge 0=Level

50

## Timer Interrupts

- Timers interrupt on overflow if enabled
  - Count increments every 12 clock cycles
- Mode 1 used for a single 16 bit delay
  - One-shot delay, then it must be reloaded
- Mode 2 reloads automatically, 8 bit delay
  - TLx reloaded with THx when TLx overflows
  - Results in constant interrupt frequency
- Load timer with 2's complement of count

51

## Timer Interrupts

- Clock Tick
  - Tick ISR Keeps Time
  - Constant Interval (e.g. timer mode 2)
- Typical OS Functions:
  - Time of Day Uses Tick Time
  - Fixed Time Delays
  - Scheduler Allocates CPU Time

52

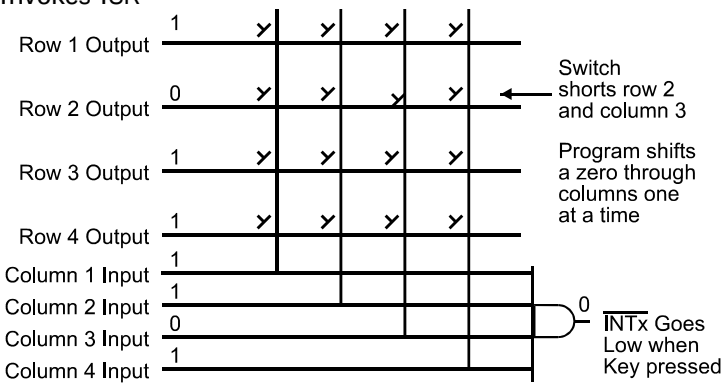
## Watchdog Timers

- Timer Count is Cleared Periodically
- If Not Cleared, Overflow Causes Reset
- Code Detects “Warm Start” Condition
  - “Cold Start” is the Initial Power-On Reset
  - Unique Flag Stored in Volatile Memory
- “Warm Start” Should be Different
  - Graceful Recovery and Resume Operation

53

## Interrupt Keypad Input

- Pressing a key
  - Activates Interrupt
  - Invokes ISR
- ISR Executes
  - Store row/column number
  - Return to main




54

## Program Structure

- Interrupt Driven Programs have 3 parts:
  - Initialization
    - Getting everything into a known state
  - Main Program (not event driven)
    - Performs non-time critical processing
  - Interrupt Service Routines (ISRs)
    - Processes the events

55

## Initialization

- Only happens once (hopefully!)
- Puts all variables, memory, in known state
  - Initialize variables, flags, I/O, IRQs
- Sets up CPU interrupt control system
  - Initialize interrupt priorities, enables
  - Clear pending interrupt requests
- Enables interrupts to occur

56

## Main Program

- Processes non-critical functions
  - The things that can be done at any time
- Anything which is not tied to an event
- When there's nothing better to do:
  - Put CPU in Low Power "Sleep" or Idle Mode
  - Execute software diagnostics
  - Execute hardware diagnostics
  - May just wait in a tight loop

57

## Interrupt Service Routine

- External event causes interrupt request
  - CPU interrupts the running program
  - CPU saves PC on stack and "CALLs" the ISR
- When an interrupt is processed:
  - ISR must save the processor state
  - ISR processes the event
  - ISR restores the processor state
  - ISR returns to the interrupted program

58

## Summary

- Basic I/O
- Switch Input
- Matrix Keypad Input
- Multiplexing I/O
- Interrupt Driven I/O

59

## Homework

- Work on Your Class Project!
- Extra Credit:
  - Count the number of contact closures using Port 1 pins, and display the number, not including bounces
  - Debug keypad1.c – row 1 is inoperative

60