CmpE213 – Digital Systems Design

Learning Objectives

Interrupts, timers, serial prog., and real-time systems

At the end of the following sections, you should be able to:

Chapters 6: Interrupts

1. Define an interrupt and an interrupt service routine (ISR). Explain why an interrupt is sometimes called an unexpected function call. Explain the difference between a process which works in the background and a process which works in the foreground.

2. Identify each of the 5 common 8051 interrupts and where they come from (internal or external, and from which pins or internal device). Identify what event triggers each interrupt (overflowing timer, high-to-low transition on P3.2, etc).

3. Write code which configures individual interrupts by a) enabling them, b) setting their priority, c) setting them for low-level or high-to-low transition activation (external ints only). Identify the above settings for each interrupt given the contents of IE and IP.

4. Explain the purpose of having multiple interrupt priorities. Give examples of situations where multiple priorities are useful. Show the execution sequence (a timing diagram of sorts) of a series of interrupts given their priority levels and when the interrupt signals occur. Given an execution sequence, identify the priority of each interrupt. Explain why reset might be called the highest priority interrupt.

5. Explain how the order interrupts are executed might be effected by the polling sequence. Given two interrupts, possibly of different priorities, that come in at the same time, identify the execution sequence. Identify which interrupt is serviced by the "interrupt n" command in Keil (e.g. Which interrupt is serviced by void blah(void) interrupt 3{...}?).

6. Given an interrupt execution sequence, show the associated timing diagram for the interrupt flags. Show which flags would be set when interrupt-triggering events occur (like a high-to-low transition on INT0). Show which flags would be reset on activation of an ISR. Given a timing diagram for the signals which cause an interrupt (like timer 1 overflows) and the initial states of IE and IP, draw a timing diagram showing interrupt execution sequence and the state of interrupt flags.

7. Explain why an ISR must begin at an interrupt vector location (i.e. why the microcontroller uses interrupt vectors). Identify where in memory an ISR must be placed (e.g. where must the ISR for timer 1 begin?). Given the contents of code memory, identify the code that services each interrupt. Given ASM code for an ISR, show the associated contents of code memory. Explain why one often sees the instruction "LJMP main" at location C:0x0000. Explain what must be done in ASM if your ISR exceeds the 8 bytes allocated in the interrupt vector space. Explain why we don't have to do this when writing code in C.

8. Interpret/write/debug programs using interrupts in either C or ASM. Given a program using an interrupt and the times an interrupt occurs, show the changing contents of memory

as the program executes. Explain why interrupts are declared like "void blah(void)" and why global variables are used to pass results to and from an interrupt (as opposed to something like "char blah(int x)").

9. Explain why registers like the accumulator, the carry flag, and others, might be pushed onto the stack before being used in an interrupt and popped off before exiting. Write ISRs which apply this concept. Explain why the stack must always be initialized in ASM when writing code that uses interrupts.

10. Explain, in detail, how an interrupt is serviced (when flags are polled, in what order, what steps the uP performs when calling or returning from an interrupt, etc). Explain how the uP implements interrupt priorities by changing IE. Manipulate IE to artificially create additional priority levels. Explain what occurs on a RETI and why RETI must always be used to return from an interrupt (as opposed to RET).

11. Define interrupt latency. Given a timing graph showing when an interrupt signal occurs and when an ISR begins, label the interrupt latency. Calculate latency given the code executed by the uP, the time the interrupt signal occurs, the priority of the interrupt, and whether any other interrupts are running. Explain the steps in calculating minimum and maximum interrupt latency. Explain why latency is important.

Chapters 4: Timers

1. Explain the difference between a timer and a counter. Give an overview of how a timer/counter works. Read or set the count of a timer/counter using TH0,TL0 or TH1,TL1. Turn a timer/counter on or off. Identify the overflow flag and explain when it is set. Identify and set the mode of a timer. Explain the operation of the different timer modes. Identify whether the timer operates as a counter or a timer. Explain the use of the GATE bit. Given TMOD and TCON, identify the state of the timers.

2. Interpret/debug/create C and ASM code using counters and timers. This includes setting and/or reading the timer/counter "count", setting/reading/understanding timer mode, understanding when a timer-interrupt occurs, setting and/or interpreting whether the counter/timer functions as a counter or a timer, and setting and/or interpreting use of the GATE bit. Given clock frequency of the microcontroller, write/interpret code which sets a timer to time-out in a specific interval of time (say 42mS). Given a program which uses a timer-interrupt, calculate how often the interrupt is called. Write/interpret an interrupt which manually reloads a 16-bit timer/counter. Write/interpret code which automatically reloads an 8-bit timer/counter (where the counter is reloaded by hardware not software).

3. Explain why problems may occur when reading or writing TH0,TL0 or TH1,TL1 "on the fly" and describe solutions for overcoming this problem. Identify code which may encounter this problem and correct it.

Chapters 5: Serial communication

1. Explain why the 8051 is said to have a 1-byte receive buffer. Explain why there are 2 internal SBUF registers. Explain what is meant by "full-duplex" and specify whether the 8051 is full-duplex or not.

2. Write/interpret/debug C or ASM code to transmit or receive data using the 8051's internal serial interface (UART). This requires understanding how to: a) set timer 1 to supply the appropriate baud rate, b) read or write values from SBUF, c) set the serial port to transmit or receive data, d) set the mode of the port and explain what each mode does.
   Write/interpret/debug code to transmit or receive a "ninth" data bit.
   Write/interpret/debug code which transmits or receives data using an interrupt.

3. Given the states of SCON, PCON, and timer 1, either a) draw a timing diagram of TXD and/or RXD given the data to be sent and/or received, or b) given a timing diagram of TXD and/or RXD find the data sent and/or received.

4. Explain how SM2=SCON.5 can be used to facilitate multi-processor communication.

Real-Time Systems:

1. Define a Real-time system. Explain the need for a system which works in "real-time". Define "Real-time" (hint: it doesn't mean "real-fast"!). Given the rates a system performs a set of tasks and the rate those tasks *should* be performed, identify whether the system operates in real-time. Give examples of real-time systems (besides those we talked about in class).

2. Explain the drawbacks to using the standard serial approach in a real-time environment. Given a simple serial program (code or flowchart), list possible methods to improve the program and make it "real-time".

3. Define or explain the following multi-tasking techniques: Round-Robin, Time-Slice, Scheduler, Priority-based Pre-emptive. Discuss differences between them. Define a Real-time clock. Given a simple program (code or flowchart), identify if the program operates in a serial manner or using the Round-Robin, Time-Slice, etc, technique. Suggest methods to make a program operate in real-time using one of the above techniques (e.g. Here is a flowchart for a serial program. Outline how you would make it into a real-time system using the time-slice technique). Given a particular set of tasks, suggest the best programming technique to perform those tasks and defend your answer.