CpE 213 Assignment 8
8051 External Interrupts
Due Nov. 25 2003 at 11 am

The purpose of this exercise is to explore the 8051's interrupt system by experimenting with the two external interrupts Int0 and Int1. Both of these are supported reasonably well by Keil's debugger.

The following C51 program for the 8051 uses external interrupt 0 to implement the clock for a simple toggle flip flop (TFF). The TFF is negative edge sensitive and toggles whenever a negative edge appears at Int0. The TFF Q output is implemented by Port 0 bit 0. The background task continuously reads P2, scales it up by a factor of 5, and outputs the result to P1. Although this is simple on purpose so that we can focus on the external interrupts, we can pretend that it is an arbitrarily complicated (and long) program, probably an external function, and possibly one for which we don't even have the source code.

```
#include <reg51.h>
sbit P0_0= P0^0;
void Ex0Isr(void) interrupt 0 {
 P0_0= ~P0_0;
 }

void main (void) {

 IT0= 1; //enable negative edge triggered Int0
 EX0= 1; //enable external interrupt 0
 EA=1;   //enable global interrupts
 while (1) P1= 5*P2;

}
```

The program consists of two functions: Ex0Isr and main. P0_0 is declared to be P0.0. Ex0Isr is the interrupt service routine (ISR) for Int0. As such, it can take no parameters and returns nothing. The function is tagged with *interrupt 0* in order to use the RETI instruction to return and also to generate the interrupt vector at location 3 (the vector location for Int0). Main enables edge triggering on Int0 by setting IT0, sets the enable bit for Int0 (EX0), and sets the global interrupt enable bit (EA). EA must be set in order for any interrupts to be acknowledged at all. If that bit is cleared, the processor will ignore all interrupt requests. EX0 must also be set in order for signals from Int0 to be acknowledged. After setting up the interrupt system, Main simply executes the background task as an infinite loop. Any negative edges which show up at the Int0 pin (shared with port 3 bit 2) will result in Ex0Isr being called and P0.0 being toggled. Of course edges at Int0 aren't synchronized to the 8051's clock and so can occur at any time relative to the execution of Main's while loop. This means that the time between a negative edge at Int0 and the corresponding change in P0.0 is variable since the interrupt isn't serviced until the current instruction is finished. In other words, our TFF has a variable (and random) propagation delay!

You can control the simulator's Int0 (and Int1) bits by opening the Peripherals/Interrupt window and select Int0# (pronounced Int0 not). EA, IT0, and EX0 will be checked if you have run the program previously. You can watch P0 bits by opening the Peripherals/IO Ports/Port 0 window. You can run the program (press Go!) and click on the IE0 box to cause Int0 interrupts and watch P0.0 toggle. Note that the Interrupt System window also shows the other five interrupt sources for a standard 8051.

Deliverable

Write a brief report that includes the answers to the following problems. Include a listing with assembly code of your program from part 3.

1. Examine the assembly code for the example program and calculate what you expect the worst-case interrupt latency to be. Express your answer in both clock cycles and instruction cycles. Show your calculations.
2. Use the example program to measure the worst case latency for external interrupt 0. This is defined as the longest time between a falling edge at Int0 and the corresponding edge of P0.0. Software engineers call this *latency.* Hardware engineers call it the clock to Q propagation delay. Use the debugger to measure the latency as well as you can within limits of the simulator. Compare your measured latency to your calculated worst-case latency. Describe the procedure used to measure the worst-case latency.
3. Modify the program to include a negative edge-triggered D flip flop with P0.1 as the D input and P0.2 as the Q output. Use Int1 as the clock input. A negative edge at Int1 should cause an interrupt. Q should be set equal to D in the interrupt service routine for Int1. Test your program using the debugger and describe your test procedure.
4. Will the previous modification affect your answer in parts 1 and 2? How?
5. Suppose it is important for the TFF to respond as quickly as possible. Describe and make suitable modifications to the program to accomplish this. Keep in mind that modification of the background task is usually not an option. Limit your modifications to the interrupt service code.