Lecture 24:
Interrupts in C

Today’s Goals

- Write interrupt service routines in C
- Assembly code inside a C subroutine
- Set vector addresses in C

Using Interrupts in C

Same issues are involved in assembly programming

- Stack pointer
  - Initialize the stack pointer.
- Configure and enable the device.
  - This involved setting bits in control registers, and is not significantly different from what we did at previous lectures.
- Enable global interrupts.
  - There is no C instruction to do this. We will need an assembly subroutine to do this.
- Write an interrupt service routine.
  - We need to differentiate between a callable subroutine and an ISR.
- Clear the interrupt device's flag bit in the ISR.
  - The flag bit should be cleared in the ISR just like in assembly code.
- Set the vector address.
  - We will discuss this with the CodeWarrior linker.

Initialize the Stack Pointer

- No instruction in C to load the stack pointer register.
  - Assembly subroutines require the stack initialized.
  - But subroutine cannot be used before the initialization of the stack pointer.
- Then, how do we use assembly code without using a subroutine?
- Inline assembly
- Note: Some IDEs like CodeWarrior will automatically include “Startup” code that initializes the stack and then jumps to the ‘main’ function.

Enabling Global Interrupts

- No way to do this in C programming level.
- This means calling the CLI assembly instruction.
- We can do this using inline assembly or as an assembly subroutine.
- Below, we’ll set up an assembly subroutine to do this.

Writing an ISR in C

Also Clearing Flag Bits

- This is one of the easier steps.
- ISR’s do not interact with the code that was running just before and after the ISR runs
  - There is no input parameter to the ISR and no return value from the ISR.
  - Therefore, “`void`” is used as both the input and output parameter. Below is a sample ISR.
- Notes:
  - The “`interrupt`” descriptor tells the compiler to end the subroutine with RTI instead of RTS.
  - C won’t know how to clear the flag bit, so this must be done manually.
  - Do not end with a “`return`” instruction.

Setting the Vector Address

- There are many ways for a development system to manage how the vector addresses are initialized.
  - We will cover a relatively easy, straightforward method.
- The linker parameter file (.prm)
  - This tells the linker several things.
    - Where RAM is located / Where ROM is located / Where vector addresses are located
  - This file begins as a generic file for a specific processor and may be customized by the user.
- Customization of .prm
  - The main customization is adding vector addresses. The following line shows an entry in a .prm file for the S12.
  - This method is sometimes useful in evaluation boards since the user has direct control of the vector addresses location. These are often directed to RAM locations, so for our Dragon12+ board, the line above would need to be

Questions?

Wrap-up

What we’ve learned

What to Come