- Including the file `xc.h` gives our program access to variables, data types, and constants that significantly simplify programming by allowing us to access SFRs easily from C code without needing to specify addresses directly.
- The included file `pic32mx/include/proc/p32mx795f512h.h` contains variable declarations, like TRISF, that allow us to read from and write to the SFRs. We have several options for manipulating these SFRs. For TRISF, for example, we can directly assign the bits with `TRISF=0x3`, or we can use bitwise operations like `&` and `|`. Many SFRs have associated CLR, SET, and INV registers which can be used to efficiently clear, set, or invert certain bits. Finally, particular bits or groups of bits can be accessed using bit fields. For example, we access bit 3 of TRISF using TRISFbits.TRISF3. The names of the SFRs and bit fields follow the names in the Data Sheet (particularly the Memory Organization section) and Reference Manual.
- All programs are linked with `pic32mx/lib/proc/32MX795F512H/crt0_mips32r2.o` to produce the final `.hex` file. This C run-time startup code executes first, doing things like initializing global variables in RAM, before jumping to the `main` function. Other linked object code includes `processor.o`, with the VAs of the SFRs.
- Upon reset, the PIC32 jumps to the boot flash address 0xBFC00000. For a PIC32 with a bootloader, the `crt0_mips32r2` of the bootloader is installed at this address. When the bootloader completes, it jumps to an address where the bootloader has previously installed a bootloaded executable.
- When the bootloader was installed with a device programmer, the programmer set the Device Configuration Registers. In addition to loading or running executables, the bootloader enables the prefetch cache module and minimizes the number of CPU wait cycles for instructions to load from flash.
- A bootloaded program is linked with a custom linker script, like `NU32bootloaded.ld`, to make sure the flash addresses for the instructions do not conflict with the bootloader's, and to make sure that the program is placed at the address where the bootloader jumps.

3.10 Exercises

1. Convert the following virtual addresses to physical addresses, and indicate whether the address is cacheable or not, and whether it resides in RAM, flash, SFRs, or boot flash. (a) 0x80000020. (b) 0xA0000020. (c) 0xBF800001. (d) 0x9FC00111. (e) 0x9D001000.
2. Look at the linker script used with programs for the NU32. Where does the bootloader install your program in virtual memory? (Hint: look at the `_RESET_ADDR`.)
3. Refer to the Memory Organization section of the Data Sheet and Figure 2.1.
   a. Referring to the Data Sheet, indicate which bits, 0-31, can be used as input/outputs for each of Ports B through G. For the PIC32MX795F512H in Figure 2.1, indicate which pin corresponds to bit 0 of port E (this is referred to as RE0).

   b. The SFR INTCON refers to "interrupt control." Which bits, 0-31, of this SFR are unimplemented? Of the bits that are implemented, give the numbers of the bits and their names.
4. Modify `simplePIC.c` so that both lights are on or off at the same time, instead of opposite each other. Turn in only the code that changed.
5. Modify `simplePIC.c` so that the function `delay` takes an `int cycles` as an argument. The `for` loop in `delay` executes `cycles` times, not a fixed value of 1,000,000. Then modify `main` so that the first time it calls `delay`, it passes a value equal to `MAXCYCLES`. The next time it calls `delay` with a value decreased by `DELTACYCLES`, and so on, until the value is less than zero, at which time it resets the value to `MAXCYCLES`. Use `#define` to define the constants `MAXCYCLES` as 1,000,000 and `DELTACYCLES` as 100,000. Turn in your code.
6. Give the VAs and reset values of the following SFRs. (a) I2C3CON. (b) TRISC.
7. The `processor.o` file linked with your `simplePIC` project is much larger than your final `.hex` file. Explain how that is possible.
8. The building of a typical PIC32 program makes use of a number of files in the XC32 compiler distribution. Let us look at a few of them.
   a. Look at the assembly startup code `pic32-libs/libpic32/startup/crt0.S`. Although we are not studying assembly code, the comments help you understand what the startup code does. Based on the comments, you can see that this code clears the RAM addresses where uninitialized global variables are stored, for example. Find and list the line(s) of code that call the user's `main` function when the C runtime startup completes.
   b. Using the command `xc32-nm -n processor.o`, give the names and addresses of the five SFRs with the highest addresses.
   c. Open the file `p32mx795f512h.h` and go to the declaration of the SFR SPI2STAT and its associated bit field data type __SPI2STATbits_t. How many bit fields are defined? What are their names and sizes? Do these coincide with the Data Sheet?
9. Give three C commands, using TRISDSET, TRISDCLR, and TRISDINV, that set bits 2 and 3 of TRISD to 1, clear bits 1 and 5, and flip bits 0 and 4.

Further Reading

MPLAB XC32 C/C++ compiler user's guide. (2012). Microchip Technology Inc.
MPLAB XC32 linker and utilities user guide. (2013). Microchip Technology Inc.
PIC32 family reference manual. Section 32: Configuration. (2013). Microchip Technology Inc.