Labo n° 6
Microprocessor Architectures [ELEC-H-473]
dsPIC33: simulation 2/2

2015–2016

Purpose

- Understand the time spend in basic computations and other limitations.
- Understand how parameters are passed to a function.
- Understand `string` initialisation in the `dsPIC33`.

Useful Documents

A set of useful documents can be found in the network share "Labo/ELEC-H-473/dsPIC33":
- Introduction to MPLAB
- MPLAB C30 C Compiler User's guide
- dsPIC30F/33F Programmer's Reference Manual
- dsPIC33FJXXXGPX06/X08/X10 Data Sheet
- dsPIC30F/33F CPU reference manual

1 Arithmetics

Open the workspace **"Simul_dsPIC2.mcw"** (File>Open Workspace).

1.1 Basics

Replace, in your current project, the source file by `arithm.c` and compile the project.

Question 1. How are the different additions and multiplications carried out?

Question 2. For each operation, indicate if the result is correct; if it is not, explain the error and the result obtained. Explain the tests (if) based on the "Carry" `C` and on the "Overflow" `OV`.

Question 3. Compare time necessary to execute the multiplications. Are all results correct?

Rules of thumb

- To spare **memory**, use the smallest size variable, according to the maximum value of the variable, of course. It might lead to longer execution time.
- Arithmetical operations are faster with variables of the native size of operands for the ALU (16 bits with the `dsPIC33`).
- Pay attention to Carry and Overflows.
- Do typecasting yourself if you want to be sure of the behaviour of the compiler.

1.1.1 Multiplication, division

Replace the file `arithm.c` by `exemple1.c` and compile the project;

Question 4. Observe the various manners of computing $a \times b/c$: give the precision of the results. Conclude.

Question 5. What are the reasons for those differences? Explain why the multiplication should be done first.

Question 6. Compare the computing time of the operations. Which one is the longest operation?

1.2 Fixed point *vs* Floating point

Replace the file `exemple1.c` by `exemple2.c`; compile the project;

Question 7. Compare, for the 3 methods, the result and the number of clock cycles.

Question 8. Explain how rounding works in the 3rd method.

Question 9. What is, in your opinion, the greatest disadvantage of the second method?
The constants by which `f1` and `f2` are multiplied are chosen this way and:
- we know that `f1` will never exceed 15.
- if `f1` is multiplied by 16, the result will not exceed 255 and can be placed in an `INT8U` (`a` in the program).

Question 10. How would you change the program if `f1` had to be multiplied by 2^4 and `f2` by 2^6? Check that the saving of computation time of calculation is still significant in this case.

Fixed point *vs* Floating point

The great advantage of the float variables is that you have got few risk of overflow (check the largest value of a 32-bit floating number), but the price to pay is an increased computing time if the CPU does not have a FPU. This example shows how you can avoid the use of float variables to save computing time: this program multiplies `f1` and `f2` and extract the integer part to send it to an 8-bit digital-to-analog converter. Three methods are shown.
In the second method, `f1` and `f2` are initially multiplied by a power of 2 (here 2^4), then rounded and placed in char variables. We know that the product is $2^4 \times 2^4 = 2^8$ times too large; it is thus necessary to divide it by 2^8 (i.e. keep the most significant byte)
In the third method, the last division gives a rounded result instead of a truncated on, which is an improvement.

2 Parameter passing

2.1 Passing by value

Replace the file `exemple2.c` by `param1.c` and compile the project.

Question 11. Where are stored the variables of `main()` (a, b, c...)?

Question 12. Where are copied, in the `main()` function, the different parameters of `Add3()` and `Mul2()` before those function are called?

Question 13. In `Add3()` where are stored the local variable `a`? (remark that the name `a` can be reused since it is local to `main()` and `Add3()`, even if it does not improve the legibility of the program).

Question 14. In `Add3()` where are stored the parameters passed to `Add3()`?

Question 15. How `Add3()` and `Mul2()` return their results?

2.2 Passing by reference

Replace the file `param.c` by `param2.c` and compile the project.
In this file, in `function1()`, an array is passed as a parameter.

Question 16. How is this parameter passed?

Question 17. How is the array used in `function1()`?

The function `swapnum()` illustrates how you can pass parameters that are not arrays by reference.

Question 18. What are the meaning of `&a` and `*i`?

Question 19. Explain the mechanism used with `swapnum()`.

3 Character strings

A *character string* is simply an array of `char`. The array size must be equal to the largest string that it shall store. To take into account the variable size of the string, it its terminated by a *null character* (`0x00`). Since a microprocessor can only store binary numbers, letters are coded. The most ancient, popular and compact code is the ASCII code (see Useful documents for a table). A more recent code is Unicode, that may use more than one byte if needed to represent letters in any language[1].

Strings are an opportunity to speak about *constants*, since many strings that you shall use in your programs (like prompts and error messages) are constants that you define when you write your source code. Since they do not change, *constants are stored in program memory.*

The `dsPIC33` is a processor with an Harvard architecture[2]. Program memory is in FLASH technology and stores 24-bit instructions extracted by a special instruction bus. Data memory is a static RAM and stores variables of different sizes extracted via a 16-bit data bus. Thus, accessing constants requires a datapath instruction bus to data bus. In the `dsPIC` this path links the 16 least significant bits of the instruction bus to the data bus. Now what about the addressing mode to access constants. In the `dsPIC`, the trick is called `PSV` (Program Space Visibility) and allows pages of the Program Memory to appear in the upper half (starting at `0x8000`) of the Data Memory (see `dsPIC33` Data Sheet §4.6.3). By default the Page is `0`, so that the address `0x000000` of Program Memory is mapped at `0x8000` of Data Memory.

- Replace the file `param2.c` by `carac.c` and compile the project.
- Put a breakpoint at the first line of code and run the program.
- Open the Program Memory window (View->Program memory) and browse the beginning of the code in the various modes that you can choose by clicking on the buttons at the bottom of the window. In particular you can view the Program Memory seen from the Data Memory point of view by the PSV Mixed and PSV Data.
- Note that the compiler produces a starting code (corresponding to the reset vector) to initialize lots of things in the processor, clear the Data Memory...
- Browse the Program Memory in mode PSV Data until you find the constant strings “hello” and “world”. Note their address in both Program and PSV spaces and length.
- Browse the same addresses in PSV Mixed mode;

Question 20. How is this area seen from the Program point of view? Why is it so.

Observe the initialization of the different variables and constants, and the mechanism of a for loop.

Question 21. What is the difference between using a `INT8U` index and a `INT16U` index?

[1] This lab focuses on the classic ASCII code, for further reference about Unicode and the various possibilities, read the official documentation on the Official Unicode website

[2] This is just a short summary, read the CPU documentation for more exhaustive explanations.