Lab 1: Introduction to PSoC Creator



Alexis Adie and Madison Mastroberte

ELC 411-01: Embedded Systems

Submitted:

September 15th 2017

Design Assignment 1 - Embedded ‘C’ Programming Basics

Alexis Adie and Madison Mastroberte

Department of Electrical and Computer Engineering

The College of New Jersey

2000 Pennington Road, Ewing, NJ 08618, USA

(adiea1, mastrom7)@tcnj.edu

# Introduction

The lab was designed to expose students to the implementation of PSoC devices, specifically the PSoC 5LP. Utilizing the code design process in PSoC Creator, students were able to discover topics such as various data types (signed and unsigned), register addressing, pointers, character strings, numerical representation, and operators (bitwise and boolean). By providing a general outline of a c program and were given functionals to generate. Overall, the lab helped introduce students to the environment and hardware that PSoC Creator utilizes, while providing a glimpse into the backend processes of the IDE.

# II. Methodology

# *Pre-lab*

In the first portion of the lab, students were required to produce code capable of accessing a register via its address and subcomponents. Students were given a manual with registers, so that students could find the partitions of the address. These partitions were: variant, implementor, constant, partno, and revision. The secondary portion of the pre-lab was to sort a character string with a bubble sort. The output would be the characters in ascii order, including spaces and capital letters. Once, the code was completed, a PSoC board was connected and PSoC creator was placed into debug mode with compiler optimization off.

*B. Accessing the Register*

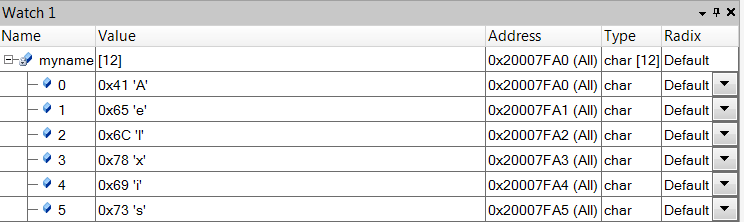
To test the capabilities of the code, the team utilized the debugging features in PSoC Creator. First, variables were placed onto a Watch List, which allows users to see the backend outputs for each variable of interest. Another debugging feature that was useful is the F10 function. This key allows the compiler to run the program on one line of code at a time, observing the variable values at each stage. As seen in Table 2.1, the values of each register change after the code is implemented.

**Table 2.1: Register Code Step-Through Showing the Values of Each Variable**

|  |  |  |  |
| --- | --- | --- | --- |
| Register | Address | Initial Value | Final Value |
| Variant | 0x20007FD8 | 0x00000121 | 0x00000002 |
| Implementor | 0x20007FDC | 0x0000036F | 0x00000041 |
| Constant | 0x20007FD4 | 0x00000163 | 0x0000000F |
| Partno | 0x20007FD0 | 0x000003EC | 0x00000C23 |
| Revision | 0x20007FCC | 0x00000265 | 0x00000001 |

*C. Sorting a Character String Using Bubble Sort*

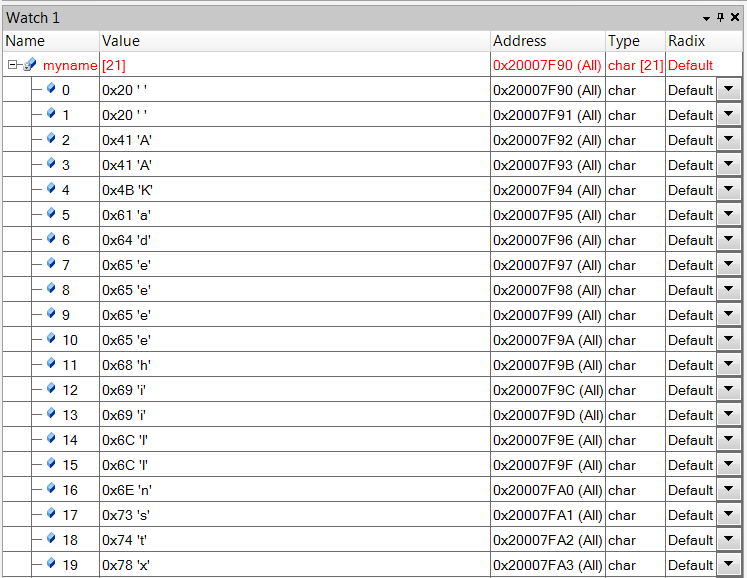
Next, the character sorting program was tested in the debugger. The character string to be tested was one of the experimenter’s full name “Alexis Kathleen Adie.” In order to ensure proper function of the bubble sort, the team observed the “myname” variable for the first few loops as seen in Figure 2.1.



**Figure 2.1: Bubble Sort Functioning Test Showing Value of “myname”**

As shown in the figure above the ‘e’ was compared to ‘l’, and since it had a higher code it was swapped.

After the program was completed, the final result was found to be “ AAKadeeehiillnstx” which confirmed that the bubble sort functioned correctly (Figure 2.2).



**Figure 2.2: The Value of “myname” After the Bubble Sort**

# III. Discussion

**In ‘C’ how would you extract the value of a register field occupying bits [5:3] from a register stored in a variable regval ?**

A pointer would be needed to be “pointed” to a specific address in order to read and extract values from the designated bits. Variables would need to be initialized to store the extracted data values. After storing the data, the results must be masked in order to cut any unnecessary bits. The C code is included below:

//Assume macro is holding address //named REG

//Declare variables and pointer

uint32\_t regval;

uint32\_t final;

uint32\_t volatile \*my\_reg\_ptr;

//The pointer points to the address

my\_reg\_ptr = ( uint32\_t \*) REG;

//Store value of address into regval

regval = \*my\_reg\_ptr;

//Mask values desired

final = (regval >> 5) & 0x38;

**How would you modify regval to insert a new value into these bits, based on the integer value stored in a variable fieldval, where we know that the value in fieldval is in the range 0...7.**

According to the *Arm Information Website,* an array can be forced to a specific address. The variable (in this specific case fieldval) would be declared in its own file. Once compiled, the file will contain data with the desired bits. The ARM scatter-loading mechanism could then be used to place its value into the address. Scatter-loading allows for the program to utilize any gaps between existing registers.

**Why does the space character (s) get sorted to the start of the array?**

The space character is sorted to the beginning of the array because it has a lower ascii character code than the other letters, for example a space character is 0x20 versus the capital A character which is 0x41. The second lowest are the capital letters and the largest codes are the lower case letters. These numbers can be observed in the ASCII table.

**Briefly explain the function of the register**

**PRT6\_DR.**

According to the PSoC Tutorial, if a port pin is not connected to the global bus and the drive mode of the pin is not HighZ or HighZ Analog, the state of the pin can be controlled by the PRTxDR register, which includes PRT6\_DR.

# IV. Conclusion

The purpose of this lab was to introduce students to the use of Cypress’ PSoC Creator to run code to access a register address and sort a character string. By doing so, the team gained useful insight into the capabilities of the ‘C’ programming language. The lab also allowed students to begin to utilize the PSoC 5LP and understand the interactions between hardware and coding. This particular lab showed students important debugging tools that give backend insight as to how C and embedded systems work.

V. References

ARM Information Center. Accessed September 14, 2017.

Sundaram, Meenakshi R. *PSoC 1: Getting Started With GPIO*. PDF. Cypress

VI. Appendix

/\*=======================================

|  |
| --- |
| Lexi Adie and Madie Mastroberte |
| ELC 411 |
| Dr. Pearlstein |
| Design Assignment 1 |
| \*======================================\*/ |
|  |
| #include <stdio.h> |
| #include <string.h> |
| #include <ctype.h> |
| #include <stdint.h> |
|  |
| // Macro which holds the register address |
| #define NVIC\_CPUID\_BASE\_ADDR (0xe000ed00) |
|  |
| int main(void) |
| { |
|  |
| // Declares the 32 bit unsigned integers |
| uint32\_t implementer; |
| uint32\_t variant; |
| uint32\_t partno; |
| uint32\_t constant; |
| uint32\_t revision; |
| uint32\_t reg\_val; |
|  |
| // Declares a pointer to a volatile unsigned integer |
| uint32\_t volatile \*my\_reg\_ptr; |
|  |
| // Puts the value of the pointer into reg val |
| reg\_val = \*my\_reg\_ptr; |
|  |
| // Access and Extract Register |
|  |
| //Take values of registers and stores them into corresponding variables |
|  |
| implementer = (reg\_val >> 24) & 0xFF; |
| variant = (reg\_val >> 20) & 0xF; |
| constant = (reg\_val >> 16) & 0xF; |
| partno = (reg\_val >> 04) & 0xFFF; |
| revision = (reg\_val >> 00) & 0xF;  //Manipulate Character String  //Bubble sort    char myname[] = "Alexis Kathleen Adie";    int size = sizeof(myname)/sizeof(char);  for (int i = 0; i < size-1; i++)  {  if (myname[i] > myname[i+1])  {  // Compares the letter to the one before it and flips the two if  // the letter has a higher code  char temp = myname[i+1];  myname[i+1] = myname[i];  myname[i] = temp;  // Resets  i = -1;  }  }  } |

|  |  |
| --- | --- |
|  |  |