CprE 588 – Embedded Computer Systems

Homework #1

Assigned: February 5

Due: February 15

Directions:

- Please submit this assignment by the due date via WebCT.
- Submissions should be in the form of 1) a PDF file with the writeup and 2) a ZIP file containing any additional necessary material.
- You must submit individual work, although you may collaborate with classmates on the problems. Please acknowledge any such collaboration when submitting.
- Engineering Distance Education students are provided with an automatic one week extension to the due date.
- Some of these questions do not have a strictly correct answer. You will be graded based on how well-formed your arguments are.

1) Embedded System Design Issues

(a) Why are sequential programming languages (e.g. C/C++/Java) considered to be insufficient for embedded system specification and design?

(b) Why are hardware design languages (e.g. VHDL/Verilog) considered to be insufficient for embedded system specification and design?

(c) Describe a scenario where a FIFO queue would be a smart choice for a communication structure between two computational elements.

(d) Under what conditions might a shared memory communication structure's efficiency be limited?

(e) Can all programmatic behavior be modeled using a Finite State Machine (FSM) representation? If your answer is ‘no’, provide a counter-example. If your answer is ‘yes’, describe a behavior that may not be efficiently modeled using an FSM.

2) General-Purpose versus Application-Specific Processing

Vahid and Givargis define a general-purpose processor as a “programmable device that is suitable for a variety of applications to maximize the number of devices sold”. Likewise, they define an application-specific instruction-set processor as a “programmable processor optimized for a particular class of applications having common characteristics”.

Evaluate the accuracy and usefulness of these definitions. Use as specific examples the Intel Core i7 (link), the ARM Cortex-A8 (link), and the TI TMS320C67x (link) processors.

3) System Design

Think about an embedded system that you are familiar with and answer the following questions.

(a) Briefly describe the basic functionality of this system.

(b) Suppose your job is to design this system using the “Platform” approach, with the following components:

- One General-Purpose Processor (GPP)
- An Application Specific Integrated Circuit (ASIC), for some specialized functionality in the system
- A DDR RAM module
- An optional set of communication controllers for external communication, such as USB, RS232, etc. (as needed)
- Two buses: one 64-bit wide bus used for memory transfers between the GPP and the RAM module, and a second 32-bit bus used for communications between the rest of the components and the GPP

Draw a block diagram that shows a possible architecture of this platform.

(c) Identify one function or task (or possibly many) in your system, and map it to a particular component. Briefly explain.

(d) Suppose your company moves to an “IP Assembly” design approach, allowing the use of IP cores from a database. Consider the function and component you identified in part c) above. Describe a possible modification to your implementation that results from the flexibility of an IP assembly approach.

(e) Suppose your company sets up a full design environment to support a “Synthesis” approach. Consider the function and component you identified in part c) above. Describe a possible modification to your implementation that results from the flexibility of a synthesis approach.

4) Digital Camera Example

See attached file *hw1.zip* which contains a partially-completed software specification of the digital camera example as described in class. Specifically, in the *Q4* folder, you should find the following C source files:

- *ccd.c* – emulates the Charge Coupled Device (CCD) on a digital camera by reading image data from a specified file
- *ccdpp.c* – performs the zero-bias adjustment on the input data
- *cntrl.c* – controls the JPEG compression operation
- *codec.c* – performs the Discrete Cosine Transform (DCT) on the input image
- *main.c* – initializes the models and simulates the camera functionality
- *uat.c* – emulates an output data transfer by writing compressed image data to a specified file

(a) Compile and run the digital camera model using the included *Makefile*. Provide the output *uat_out.txt* file with your submission.

(b) Profile the executable using the “gprof” command. Identify where the code is spending most of its time. Briefly summarize the relevant output of the profiler.

(c) Modify the digital camera model such that is can handle arbitrary 8-bit PGM images. Compile and run the model on the supplied *camera256.pgm* image. Provide the encoded output file with your submission.

(d) How much compression is achieved for both the *image.txt* and *camera256.pgm* images? Use the output result files from parts a) and c) in order to estimate how much the total image storage requirements have been reduced.

(e) Based on the encoded output file from part c), how much additional compression could be achieved through the use of Huffman coding? This question has a precise answer but a general analysis of Huffman coding as applied to the output file from part c) is sufficient.

5) Introduction to SpecC

The purpose of this question is to introduce you to the SpecC compiler. The SpecC compiler is installed on the department Linux servers; you can find more information on how to access the compiler at the class website (link). See folder *Q5* in attached file *hw1.zip* which contains the specification model of a parity calculation example written in SpecC.

(a) Briefly describe the basic functionality of this system. What is the computation described in files *ones.sc* and *even.sc*?

(b) Compile the parity calculation model using the included *Makefile*. What is the output after running the compiled code for each of the following inputs: {208, 723, 24, 35}?

(c) Modify the source such that it now calculates the longest string of consecutive zeros in the input data. Attach the modified source files in a ZIP file to be submitted with your writeup.