Embedded System Design and Modeling

EE382V, Fall 2011

Homework #1

Methodologies, Languages

Assigned: September 1, 2011
Due: September 22, 2011

Instructions:

- Please submit your solutions via Blackboard. Submissions should include a single PDF with the writeup and a single Zip or Tar archive for any supplementary files (e.g. source files, which has to be compilable by simply running 'make' and should include a README with instructions for running each model).
- You may discuss the problems with your classmates but make sure to submit your own independent and individual solutions.
- Some questions might not have a clearly correct or wrong answer. In general, grading is based on your arguments and reasoning for arriving at a solution.

Problem 1.1: System Design

During design space exploration as part of the system design process, the target system architecture and its key architectural parameters are decided on. These design decisions have a major influence on the final design quality metrics such as (i) performance, (ii) power, (iii) cost, and (iv) time-to-market:

(a) Briefly discuss how the following target platform styles rate in relation to each other in terms of the metrics listed above:
- A pure software solution on a general-purpose processor
- A general-purpose processor assisted by a custom hardware accelerator/co-processor
- A general-purpose processor and a specialized processor (DSP or ASIP)

(b) Try to sketch a potential simple strategy for exploring the design space for a given application under a given set of constraints/requirements.

Problem 1.2: Languages

(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?

Problem 1.3: Language Concepts

We have discussed the concepts of synthesizability, orthogonality and separation of concerns in languages.

(a) What are the requirements for languages to be synthesizable?

(b) Briefly define what orthogonality is? What is separation of concerns?

(c) Try to show an example other than the ones discussed in class of non-orthogonal concepts or constructs in a language.

(d) Name two concerns that should be separated as they cover orthogonal aspects? Show a short code excerpt that demonstrates a non separated code and code that separates these concerns (in C/C++/SpecC/SystemC or similar).

Problem 1.4: Design Methodology

(a) Compare and contrast a top-down, bottom-up and meet-in-the-middle design methodology.

(b) What type of methodology is a platform-based approach? Describe a platform-based methodology and flow.

(c) Why do we perform computation design before communication design in the SpecC/SCE methodology?

Problem 1.5: SpecC Language and Modeling

The SpecC environment is installed on the ECE LRC Linux servers. Instructions for accessing and setting up the tools are posted on the class website:

http://www.ece.utexas.edu/~gerstl/ee382v_f11/docs/SpecC_setup.pdf

In short, once logged in, you need to load the corresponding module:

```
module load sce
```

The SpecC installation includes a comprehensive set of examples showing the features and use of the language. Examples are found in `$SPECC/examples/simple/`. You can copy them into a working directory:

```
mkdir hw1.5
cd hw1.5
cp $SPECC/examples/simple/* .
```

And then use the provided `Makefile` to compile and simulate all examples:

```
make all
make test
```

It is recommended to inspect the sources of all examples and the included `Makefile` to understand the use of `scc` for the compilation and simulation process, and to experiment with the `scc` command-line usage and with the various `sir_`*`xxx`* tools.

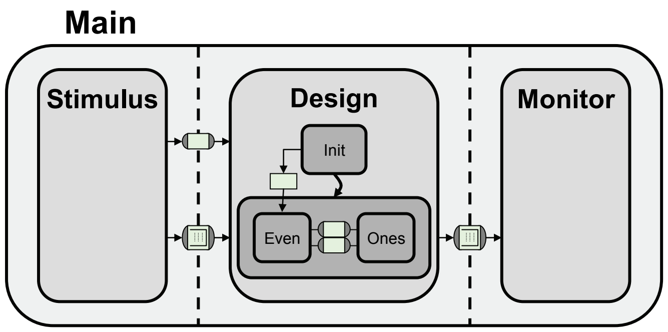
The directory `$SPECC/examples/parity` contains an example of a parity generator written in SpecC. For this assignment, copy the example to a local working directory and follow the instructions in the README file to compile and run it:

(a) Draw the SpecC diagram (graphical notation/representation) of the system and briefly describe its functionality. What is the functionality of the behaviors in the files `ones.sc` and `even.sc`? Could the same functionality be modeled as a single behavior and/or why do you think it might be useful or necessary to have it split into two?

(b) Modify the example to separate computation from communication, i.e. replace all shared variable plus event communication between `ones` and `even` to exclusively use (synchronous) message-passing, i.e. using the `c_double_handshake` channel from the standard SpecC channel library. Simulate the modified code to verify its correctness.

(c) Modify the parity checker into a parity generator/encoder. At its input, the parity generator accepts a stream of 7-bit words (represented as `char` bytes where the MSB is not used) over a `c_queue` channel. At the output, a `c_queue` channel produces the stream of parity-encoded words (MSB is the parity bit). Furthermore, add an initialization behavior before the actual `Parity` generator that waits for an external start message and sets an internal flag to select odd or even parity encoding depending on a mode contained in the start message.

Finally, enclose this design into a typical testbench setup. Modify the top level of the example (`Main` behavior) to describe a proper structure consisting of `Stimulus`, `Design` and `Monitor` behaviors:



The `Stimulus` should feed the mode and a set of input words into the design over the input queue. The `Monitor` should receive the encoded words on the output. Make sure that the testbench cleanly terminates the simulation (via an `exit(0)` system call) when the end of the streams has been reached (e.g. in the `Monitor` after a fixed number of words have been received). The goal is to reuse this same testbench throughout the whole design process. The Monitor should compare the generated output to known-good/golden values. If you have the Stimulus and Monitor read/write the input/output data from/to a file, you can at every step run the design on different test vectors and use the `diff` command to compare the generated output file with a golden reference file.

Problem 1.6: SystemC Modeling

The SystemC environment is installed on the ECE LRC Linux servers. Instructions for accessing and setting up the tools are posted on the class website:

http://www.ece.utexas.edu/~gerstl/ee382v_f11/docs/SystemC_setup.pdf

In short, once logged in, you need to set the $SYSTEMC environment variable (depending on your $SHELL):

`setenv SYSTEMC /usr/local/packages/systemc-2.2.0` ([t]csh)

or

`export SYSTEMC=/usr/local/packages/systemc-2.2.0` ([ba]sh)

The SystemC installation comes with a set of examples, available under `$SYSTEMC/examples`. You can copy an example into a working directory:

```
mkdir hw1.6
cd hw1.6
cp /home/projects/gerstl/pkg/systemc-2.2.0/examples/simple_fifo/* .
```

And then use the provided `Makefile` to compile and simulate the example:

```
make
./simple_fifo
```

Inspect the sources of the example and the included `Makefile` to understand SystemC compilation and simulation process, experiment with the `Makefile` usage and start modifying the example to experiment with different features of the SystemC language:

(a) Briefly explain (in 1-2 sentences) the functionality and structure of the example.

(b) Modify the example to replace the custom `fifo` channel with a corresponding `sc_fifo<char>` channel from the standard SystemC channel library. Simulate the code to verify correctness and submit the modified source code.

(c) Translate the SpecC specification model of the parity checker from Problem 1.5(c) into a corresponding SystemC model at the untimed functional (UTF) level. This is very similar to the process the SpecC compiler performs when translating a SpecC model into a C++ executable for simulation. In the process, the SpecC behavior hierarchy is converted into a matching hierarchy of SystemC modules where each SpecC behavior becomes a SystemC module with exactly one *main* process. As a reference, you can follow the ideas outlined in reference [6] on the class webpage.

(d) Report on your experiences with SpecC and SystemC modeling. Based on your experiences, compare and contrast the languages in terms of ease of use, feature richness, modeling expressiveness, etc. Do the languages meet the goals and requirements for SLDLs outlined in class? How could the languages be improved? This is about hearing your opinions, so be frank and criticize the pros and cons, the good and bad of both SpecC and SystemC.

(e) Extra credit: both SpecC and SystemC are based on a concurrency model that exposes control parallelism. Staying within the philosophies of each language, how would you extend them with constructs to better support explicit modeling of data parallelism, i.e. applying the same computation/function to different parts of a data set (e.g. different parts of an array)?