Embedded System Design and Modeling

EE382V, Spring 2014

Homework #1

Languages

Assigned: January 23, 2014
Due: February 6, 2014

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: 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_s14/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 hw1.1
cd hw1.1
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.

(b) Modify the example to separate computation from communication. Create a new channel that encapsulates basic communication primitives and replace all shared variables and events between `Ones` and `Even` to exclusively use one or more instances of your custom channel. You can look at the example on slide 41 of lecture 2 for guidance. Simulate the modified code to verify its correctness.

(c) Now replace your custom channel with one or more `c_queue` instances out of the SpecC standard channel library. Again, simulate the code to verify correctness. Does it make any difference whether you use a `c_queue` or a `c_double_handshake` channel? Why or why not?

(d) Similarly upgrade the communication between the main design and the testbench (`IO`) to use proper communication channels (pick what you feel is appropriate). In the process, modify the parity checker to separate out all communication with the testbench into additional `Start` and `Done` behaviors that execute before and after the `Ones`/`Even` combination, respectively. Make sure to create a clean SpecC hierarchy that does not mix different types of behavioral compositions in one parent behavior. Again, simulate and make sure everything works correctly. Does your design exhibit any actual concurrency (in the sense of code that can run truly in parallel)? How could the model be changed to expose additional parallelism (just sketch the graphical diagram; specifically think about situations in which a continuous stream of data items need to be run through the parity checker)?

Problem 1.2: 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 hw1.2
cd hw1.2
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 become familiar with SystemC, including its compilation and simulation process. You can use this example to experiment with the `Makefile` usage and as a starting point for developing the code for this assignment:

(a) 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.

(b) Translate the SpecC model of the parity checker from Problem 1.1(c) into a corresponding SystemC model. Aim to be as faithful as possible in replicating SpecC concepts using equivalent SystemC features. As discussed in class, 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 [2] on the class webpage. Make sure your SystemC model compiles and simulates.

(c) Translate the SpecC model of the modified parity checker from Problem.1.1(d) into a corresponding SystemC model. Again, try to be as faithful as possible in replicating SpecC concepts using equivalent SystemC features. How do the two languages compare in expressiveness for being able to capture the behavior of the two parity checker variants? Based on your experiences, what is your personal opinion about the pros and cons of each language?

Problem 1.3: Discrete-Event Semantics

For each of the following code examples, what is the value of `myB` printed at the end of execution and at what simulated time does the program terminate. You are free to run the code on top of the SpecC simulator and observe the program output, but you need to provide an explanation and reasoning of why the program is behaving as it is (e.g. sequence of events happening during simulation):

(a)

```
behavior A(int myB)
{
  void main(void)
  {
    myB = 10;
  }
};

behavior B(int myB)
{
  void main(void)
  {
    myB = 42;
  }
};

behavior Main(void)
{
  int myB;

  A a(myB);
  B b(myB);

  int main(void) {
    par { a; b; }
    printf("%d", myB);
    return 0;
  }
};
```

(b)

```
behavior A(int myB)
{
  void main(void) {
    waitfor 42;
    myB = 10;
  }
};

behavior B(int myB)
{
  void main(void)
  {
    myB = 42;
  }
};

behavior Main(void)
{
  int myB;

  A a(myB);
  B b(myB);

  int main(void) {
    par { a; b; }
    printf("%d", myB);
    return 0;
  }
};
```

(c)

```
behavior A(int myB)
{
  void main(void) {
    myB = 10;
    waitfor 42;
  }
};

behavior B(int myB)
{
  void main(void) {
    waitfor 10;
    myB = 42;
  }
};

behavior Main(void)
{
  int myB;

  A a(myB);
  B b(myB);

  int main(void) {
    par { a; b; }
    printf("%d", myB);
    return 0;
  }
};
```

(d)

```
1  behavior A(int myA, event e)
2  {
3    void main(void) {
4      myA = 10;
5      notify e;
6      myA = 11;
7      notify e;
8      waitfor 10;
9    }
10 };
11
12 behavior B(int myA, int myB, event e)
13 {
14   void main(void) {
15     wait e;
16     myB = myA;
17   }
18 };
19
20 behavior Main(void)
21 {
22   int myA;
23   int myB;
24   event e;
25
26   A a(myA, e);
27   B b(myA, myB, e);
28
29   int main(void) {
30     par { a; b; }
31     printf("%d", myB);
32     return 0;
33   }
34 };
```

(e)

```
behavior A(int myA, event e)
{
  void main(void) {
    myA = 10;
    notify e;
    waitfor 10;
    myA = 11;
    notify e;
  }
};

behavior B(int myA, int myB, event e)
{
  void main(void) {
    wait e;
    myB = myA;
  }
};

behavior Main(void)
{
  int myA;
  int myB;
  event e;

  A a(myA, e);
  B b(myA, myB, e);

  int main(void){
    par { a; b; }
    printf("%d", myB);
    return 0;
  }
};
```

(f) What code has to be inserted at the beginning of behavior `B` (line 15) in (e) to change the output of the program? Give two different options. What must *not* appear there for the program not to deadlock?

(g) Why do SpecC events have a semantic in which they can get lost? Under what condition do SpecC events get lost? What type of channel/communication could not be modeled if delivery would always be guaranteed?