Embedded System Design and Modeling

EE382V, Fall 2010

Homework #2

Specification and MoCs

Assigned: September 30, 2010
Due: October 14, 2010

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 2.1: Non-determinism

(a) What is nondeterminism?

(b) How might nondeterminism arise? (give one example not discussed in class)

(c) What are the advantages and disadvantages of having nondeterminism in a language or model, i.e. in what circumstances might it be positive/desired or negative/undesired?

(d) Is a SpecC specification model deterministic? If yes, why? If not, list possible sources of non-determinism and how they can be avoided (i.e. how a SpecC model can be made deterministic) (M'08)

Problem 2.2: Models of Computation and Languages

In class, we learned about the different Models of Computation (MoCs): KPN, SDF, FSM(D), HCFSM, PSM.

(a) What is the relationship between MoCs and languages?

(b) Can SpecC support all these MoCs? If so, briefly sketch for each MoC that you think can be supported how you would represent a corresponding model in SpecC.

Problem 2.3: Kahn Process Networks (KPN)

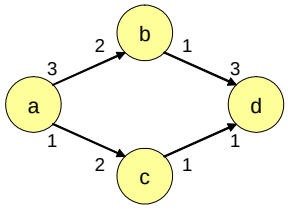
(a) Does Parks' KPN scheduling algorithm always find a bounded schedule if it exists? Why or why not?

(b) Does Parks' algorithm always find a complete KPN schedule, if it exists? Why or why not?

(c) Does Parks' algorithm always find a non-terminating (free of artificial deadlocks) schedule, if it exists? Why or why not?

Problem 2.4: Dataflow Synthesis

For the SDF graph on the right:



(a) Show that the graph consistent and that it has a valid schedule.

(b) List all possible minimal periodic static schedules.

(c) Find the periodic schedule with the lowest token buffer usage. What is the minimum buffer usage?

(d) Assume each actor firing executes in one time unit. Find the schedule with the highest throughput (output token rate, i.e. average number of firings of the output actor *d* per time unit). What is the maximum throughput on a single processor?

(e) Assume the graph is scheduled on two processors where each actor executes in one time unit on either PE and buffers are stored in a shared memory with zero communication overhead. Find a fixed assignment of actors to PEs and a corresponding schedule that maximizes throughput. What is the maximum throughput on two processors?

Problem 2.5: State-Machine Models of Computation

In class, we have discussed the concepts of hierarchy (OR state) and concurrency (AND state) for reducing complexities in a HCFSM (e.g. StateCharts) model. However, both hierarchical and concurrent FSM compositions can be converted into an equivalent plain FSM model:

(a) Derive an expression for the complexity (number of states and number of transitions) of the equivalent plain FSM as a function of the complexity of the OR-composed FSMs.

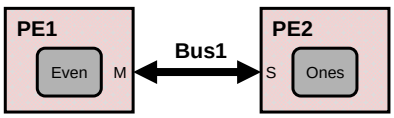
(b) Derive an expression for the complexity (number of states and number of transitions) of the equivalent plain FSM as a function of the complexity of the AND-composed FSMs.

Problem 2.6: Communication Refinement and Modeling

For this problem, we will further refine the parity checker from Homework 1, Problem 1.6 all the way down to both pin-accurate and transaction-level communication models of its design. You can start from the code for the computation model of the parity checker that you developed for Problem 1.6(c) in Homework 1. A reference solution can be found at:

`/home/projects/courses/fall_10/ee382v-16985/parity1.tar.gz`

Assume an implementation in which a single *Bus1* connects *PE1* (master) and *PE2* (slave) through a bus protocol taken from the bus database:



The source code includes an implementation of the *DoubleHandshakeBus* protocol that we will use for *Bus1*. Pin-accurate and transaction-level database models of the bus protocol are given in the file `DblHndShkBus.sc` in the `bus` subdirectory. Browse the bus database model and try to understand its structure. It is easiest to start with the channel *DblHndShkBus* as it shows a demo instantiation of the bus. It first defines the bus wires and a protocol-level (physical) interface each for master (*MasterDblHndShkBus*) and slave (*SlaveDblHndShkBus*) sides, which connect to bus wires.

Finally, media access (MAC) channels (named *(Master|Slave)DblHndShkBusLinkAccess)* show the methods of how to access the bus. The protocol-level interface (both master and slave side) can be exchanged with a single *DblHndShkBusTLM* channel (where the communication is not performed via the wires previously instantiated, but through events as a transaction-level model).

(a) Draw the timing diagram of the pin-accurate model of the bus protocol. Draw a similar diagram of the timing of events in the transaction-level model. Assuming that simulation runtimes grow linearly with the number of simulated events, what is the expected speedup per bus transaction of transaction-level vs. pin-accurate modeling? Is this an efficient transaction-level implementation in terms of simulation performance? Give some suggestions to improve TLM speed. Hint: have a look at references [23,24].

(b) Manually refine the computation model of the parity encoder down to a pin-accurate model (PAM) and a transaction-level model (TLM) of the system. Use and instantiate corresponding bus database protocol adapters or channels (inlined/instantiated adapters in the PEs or as channel between PEs, respectively) for realization of *Bus1* communication. Briefly describe the transformation steps you applied. Simulate all models to validate their correctness. Report on the differences in lines of code and simulation runtimes/speed between the models. Explain the quantitative and qualitative composition of and contributions to the simulated delays observed in each model.

Hint: To compute the lines of code for a SpecC model, you can use the `sir_stats` tool that is part of the SpecC tool set. Also, to obtain simulation runtimes, you can prepend the Unix `time` command in front of the simulation command line. Note, however, that you will have to increase the `time` resolution by averaging over a large number of simulation runs or a larger input test vector file.

Option for extra credit: write a faster and still equally accurate TLM, demonstrate its efficiency and explain why your TLM is better