Embedded System Design and Modeling

EE382V, Fall 2009

Homework #2

Models of Computation, Refinement

Assigned: September 24, 2009
Due: October 8, 2009

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.
- 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: Models of Computation and Languages (10 points)

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 2.2: Process and Dataflow Models of Computation (15 points)

In relation to process network and dataflow type Models of Computation (MoCs), we talked about properties such as determinism and deadlocks. For the following questions, either provide a proof (property holds in all cases) or a counter-example:

(a) Does adding an `empty()` method that reports whether a channel has waiting tokens change the determinism of Kahn Process Networks (KPNs)?

(b) To determine the relative rates for a static schedule of Synchronous Dataflow (SDF) actors, one can derive and solve the system of linear *balance equations*. Can a SDF model have satisfiable balance equations (i.e. be consistent) and not run forever? Can a SDF model violate the balance equations (i.e. be inconsistent) and still run forever?

(c) Can a tree-structured (i.e. acyclic) SDF model ever have deadlocks? Can a static, complete and bounded schedule always be found?

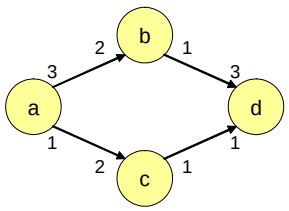
Problem 2.3: Dataflow Synthesis (25 points)

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 2.4: State-Machine Models of Computation (20 points)

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.

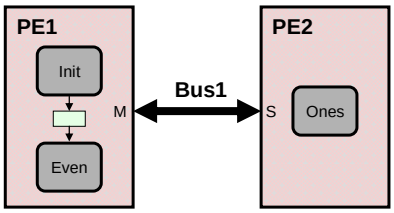
Problem 2.5: System Refinement Flow and Transaction-Level Modeling (30 points)

For this problem, we will implement and refine a modified version of the parity checker from Homework 1, Problem 1.5 all the way down to a Transaction-Level Model (TLM) of its design. An upgraded version of the parity checker example can be found at:

`/home/projects/courses/fall_09/ee382v-17220/parity.tar.gz`

The design has been modified into a parity generator/encoder and converted into a proper specification model. At its input, the parity generator accepts a stream of 7-bit words (represented as `char` bytes where the MSB is not used). At the output, it produces the stream of parity-encoded words (MSB is the parity bit) where an initial start message send during initialization selects odd or even parity encoding. The parity encoder `Design` is then enclosed in a typical testbench setup with `Stimulus` and `Monitor` behaviors that drive and capture the simulation by reading and writing input and output streams from/to files.

Assume an implementation in which `Init` and `Even` behaviors are mapped to *PE1*, the `Ones` behavior is mapped to *PE2*, and everything is statically scheduled. A single *Bus1* connects *PE1* (master) and *PE2* (slave) through a double-handshake protocol from the bus database:



Pin-accurate and transaction-level database models of the bus protocol channel are given in the file `DblHndShkBus.sc`.

(a) Draw the timing diagram of the pin-accurate bus protocol. Draw a similar diagram of the timing of events in the transaction-level model. Assuming that simulation speeds grow linearly with the number of simulated events, what is the expected speedup per bus transaction of transaction-level vs. pin-accurate modeling?

(b) Manually refine the specification model of the parity encoder down to a pin-accurate and transaction-level communication model of the system. Modify the `Design` to reflect the partitioning and scheduling. Insert execution delays of 30/50 time units per word in `Even`/`Ones`. Use and instantiate bus database protocol channels (inlined into the PEs or as-is 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, simulated delays and simulation speeds between the models.