# Embedded System Design and Modeling

**EE382V, Fall 2011**

## Homework #2

**Models of Computation (MoCs)**

**Assigned:** September 29, 2011
**Due:** October 13, 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 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)

---

**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.

(c) Specifically show a code template for how you would write an SDF actor and an SDF model in SpecC. Discuss in how far the SpecC code is equivalent to the original SDF model, e.g. in terms of being amendable to automated analysis or synthesis by design tools.

---

**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?

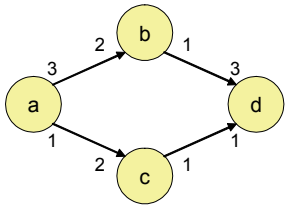
(c) Does Parks' algorithm always find a non-terminating (free of artificial deadlocks) schedule, if it exists? Why or why not?

(d) In class, we mentioned that Parks' algorithm is not guaranteed to find a complete, bounded and non-terminating schedule even if one exists. Show an example of a KPN where such a schedule exists but Parks' algorithm fails to find it. Hint: in the KPN example presented in class, think about token patterns that can happen on the P2→P3 edge.

---

**Problem 2.4: Synchronous Dataflow Synthesis**

For the SDF graph on the right:




(a) Show that the graph is consistent and that it has a valid schedule. What is the repetition vector of the graph?

(b) In class, we discussed the concept of a precedence graph that can be used for scheduling, i.e. the representation of an SDF graph as an equivalent homogeneous SDF (HSDF) model, based on its repetition vector. Show the precedence graph for this example.

(c) List all possible minimal periodic static schedules.

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

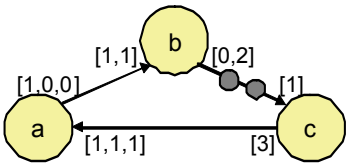
(e) Assume each actor firing executes in one time unit. Find the schedule with the highest throughput (output token rate, i.e. average number of firings of the output actor *d* per time unit). What is the maximum throughput on a single processor?

(f) 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: Cyclo-Static Dataflow**

For the CSDF graph on the right:




(a) Assuming that within each periodic iteration of the graph, the number of firings of each actor must be an integer multiple of the number of phases/firings in the actor's cycle, show that the graph is consistent and that it has a valid repetitive schedule. What is the repetition vector of the graph?

(b) Show the precedence graph for this example.

(c) List all possible minimal periodic static schedules.

(d) Find the periodic single-processor schedule with the lowest buffer usage. What is the minimally required buffer space?

(e) Find single-processor and dual-processor schedules with maximal throughput. What are the buffer requirements in each case?

**Problem 2.6: 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 2.7: Discrete-Event Semantics**

As discussed in class, SpecC includes a channel library that provides higher-level communication primitives on top of the basic SpecC language and its discrete-even execution semantics. As such, channels are written to provide safe communication using basic SpecC events and variables:

(a) 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?

(b) The SpecC channel library includes a `c_handshake` channel that provides safe event delivery. However, with this channel, events can still be overwritten. Write a `c_event_queue` channel that provides a FIFO type of event delivery (for a fixed maximum number of outstanding events). Test your channel to ensure it works under various conditions. Submit the code for both the channel and your testbench.

(c) If you look at the source code of the `c_double_handshake` channel (under `$SPECC/import/c_double_handshake.sc`), what is the semantics of how the "`notify ack; wait ack;`" pair of statements at the end of the *receive()* method behaves, i.e. what is the state of the simulator before and after executing these two statements? Furthermore, what do you think is the purpose of the final "`wait ack;`' statement? Would the channel work as intended without it? Under what conditions would it or would it not? Hint: think about all possible interleavings of *send()* and *receive()* calls (esp. considering that there is a `i_tranceiver` interface).