Embedded System Design and Modeling

EE382V, Spring 2014

Homework #3

Synthesis and Refinement

Assigned: April 3, 2014
Due: April 17, 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 3.1: Mapping and Exploration

Given the SDF graph from Homework 2, Problem 2.3(a), explore various approaches for an automated and optimized mapping of this graph onto a 2-processor system. Assume that processor are homogeneous and that each actor takes 1 time unit to execute independent of where it is mapped to. Remember that every SDF graph can be converted into an equivalent homogeneous SDF model. Using this SDF to HSDF transformation as a preprocessing step, we can apply mapping algorithms developed for standard task graphs to the equivalent precedence graph of the SDF:

(a) Write down the ILP formulation for the mapping (combined partitioning and scheduling) of the problem graph on 2 processors. Limit the optimization problem to a basic (non-pipelined) schedule of a single iteration of the graph in the time window $0 \leq t \leq T$. List all the inputs to your ILP and all constraints for number of iterations, unique mapping of actors to processors, sequential execution on each processor and sequencing/dependency relations between actors. Formulate an objective to minimize overall latency (time to execute the single iteration of the graph).

Finally, write down one valid (not necessarily optimal) solution of the ILP and its latency. You can either show that your answer for 2.3(f) is a valid solution, or you can feed the ILP into an automated solver to find an optimal mapping. For the latter, as discussed in class, you can use IBM's commercial ILOG CLPEX tool, which we have installed on the LRC machines (accessible via `module load cplex`). A basic CPLEX tutorial is available at:

http://www.yzuda.org/tutorials/CPLEX/CPLEX_ILP_01.html

(b) Apply a list scheduling algorithm that uses the level (longest distance to the sink, i.e. critical path length) of a node in the graph as priority function. Feel free to (optionally) implement the list scheduler in your language of choice and submit your code as solution of the assignment (including a README showing how to compile and run your code). In either case, show the step-by-step operation of the algorithm (state of the priority-sorted ready queue and mapping decisions made in each step), as well as the final graph execution generated by the scheduler as part of your writeup.

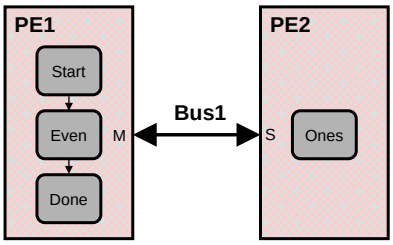
As discussed in class, under the assumptions of this assignment (uniform tasks and processors), a highest-level first (HLF) list scheduler as applied here is the same as Hu's algorithm, which is provably optimal for such problems. Compare your list scheduling result to the ILP result in (a), and confirm that it is a valid and optimal mapping (i.e. either better or the same than the mapping solution you obtained in (a)).

(c) Extra credit: apply either a simulate annealer (SA) or genetic algorithm (GA) to the mapping problem. You can start from one of the many available SA or GA libraries out there, or you can develop your own code for a most basic version from scratch. In either case, you need to think about how to encode the design decisions, how to compute the cost function and how to implement the making and/or checking of valid (random) moves/recombinations. Hint: ILP encodings of optimization problems (decision variables, constraints and cost functions) can be very helpful in providing a corresponding understanding and hence starting point.

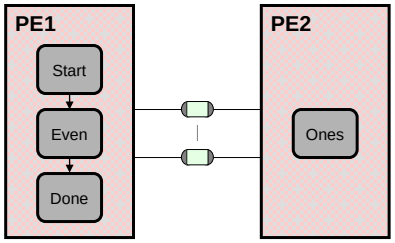
Problem 3.2: Model Refinement

For this problem, we will further refine the parity checker from Homework 1 all the way down to both pin-accurate and transaction-level communication models of its design. You are free to work in your preferred language, i.e. either SpecC or SystemC. You can start from the code for the specification model of the parity checker that you developed for Problem 1.1(d)/1.2(c) in Homework 1 (or the reference solutions provided).

Assume an implementation in which `Start`, `Even` and `Done` 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):

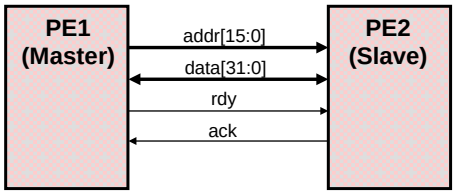


(a) Manually refine the specification model into a computation model where the `Parity` design reflects the partitioning and scheduling of behaviors and variables to PEs:



Insert execution delays of 30/50 time units per word in `Even`/`Ones`. Modify the testbench (`IO`) to print the input-to-output latency of the parity encoder.

(b) Assuming that *Bus1* connecting *PE1* (master) and *PE2* (slave) uses a double-handshake protocol taken from the bus database:



Source code with an implementation of the corresponding *DblHndShkBus* protocol is given in the SpecC bus database:

`$SPECC/share/sce/db/busses/simple/DblHndShkBus.sc`

(ignore all the code enclosed in `#if USE_MAC_TLM` conditional compiler directives). A SystemC version is provided at:

`/home/projects/courses/spring_14/ee382v_17303/DblHndShkBus{.h/.cpp}`

Copy the code to your directory and browse the bus database model to 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).

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 context switches, i.e. wait and waitfor events, what is the expected speedup per bus transaction of transaction-level vs. pin-accurate modeling?

Finally, 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 protocol adapters or channels (inlined/instantiated adapters in the PEs or as channel between PEs, respectively) for PAM- or TLM-level realization of *Bus1* communication.

Document the transformation steps you applied and include listings of your modified source code. Simulate all models to validate their correctness. Explain the quantitative and qualitative composition of and contributions to the simulated delays observed in each model. Report on the differences in lines of code and simulation runtimes/speed between the models. 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. Can you think of any ways for speeding up the simulation (with our without a loss in accuracy compared to the PAM)?