Embedded System Design and Modeling

EE382V, Fall 2011

Homework #3

Synthesis and Refinement

Assigned:	October 25, 2011
Parts 3.1-3.3 due:	November 3, 2011
Part 3.4 due:	November 17, 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 3.1: Partitioning and Scheduling

In class, we introduced an ILP formulation for multi-processor partitioning and scheduling of synchronous dataflow (SDF) graphs. Both in class and in the previous homework, we also talked about the fact 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 instead develop a simplified ILP formulation for the converted HSDF graphs:

(a) Develop an ILP model for multi-processor partitioning and scheduling of HSDF graphs. Limit the schedule to execution of a single iteration of the graph in the time window $0 \leq t \leq T$, i.e. do not take overlapping/pipelining across iterations into account. Assume each processor has the same cost and each actor takes 1 time unit to execute independent of where it is mapped to. Your ILP formulation should only have two types of binary (i.e. $\in \{0,1\}$) decision variables:

A_{ij}: Actor i mapped to processor j

s_{it}: Actor i starts execution in time t

List all the inputs to your ILP and develop constraints for number of iterations, unique mapping of actors to processors, sequential execution on each processor, and sequencing/dependency relations between actors. Finally, formulate an objective to minimize overall cost and latency (time to execute the single iteration of the graph).

(b) Apply the ILP to the SDF model from Homework 2, Problem 2.4 when scheduled on two processors (as in 2.4(f)). Write down the specific constraints and objective functions for this problem instance. Show that your answer for 2.4(f) is a valid solution to the ILP.

(c) Extra credit: without introducing additional decision variables, how could execution times ≥ 1 time unit that vary from actor to actor (but not for different processors) be taken into account?

Problem 3.2: Bus Taxonomy

We introduced in class an extremely simplified taxonomy of busses, distinguishing between master/slave and node-based communication systems:

(d) What are the system characteristics that suggest when to use a master/slave-based bus system? Briefly reason about the identified characteristics and give one specific example for an architecture (and maybe application).

(e) What are the system characteristics that suggest when to use a node-based bus system? Briefly reason about the identified characteristics and give one specific example for an architecture (and maybe application).

Problem 3.3: Communication Semantics

We discussed synchronous and asynchronous communication primitives at the application level:

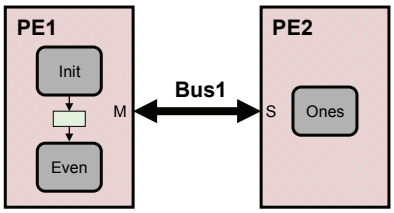
(a) Define in your own words (1-2 sentences) synchronous communication, e.g. by showing and briefly describing the activity graph/message sequence chart (communicating state machines) between a sender and a receiver highlighting synchronization and data transfer. Briefly outline an application example which benefits from synchronous communication as for example implemented with the *c_double_handshake* channel in SpecC.

(b) Define in your own words (1-2 sentences) asynchronous communication, e.g. by showing and briefly describing the activity graph/message sequence chart (communicating state machines) between a sender and a receiver highlighting synchronization and data transfer. Briefly outline an application example which benefits from asynchronous communication as for example implemented with the *c_queue* channel in SpecC.

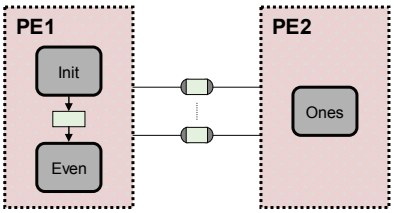
Problem 3.4: Model Refinement

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 either SpecC or SystemC whatever you prefer. You can start from the code for the specification model of the parity checker that you developed for Problem 1.5(c)/1.6(c) in Homework 1. A reference solution is available on Blackboard.

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):

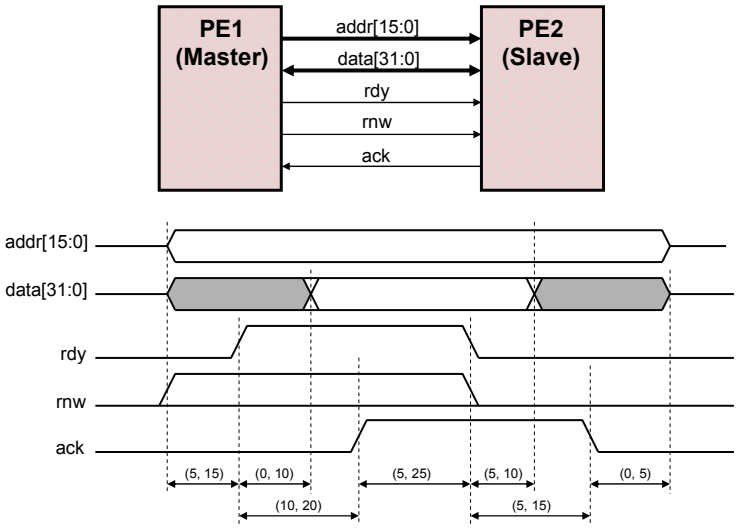


(a) Manually refine the specification model into a computation model where the `Parity` design reflects the partitioning of behaviors and variables to PEs:



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 modified double-handshake protocol according to the following timing diagram. In this protocol, the master signals the type of transaction (read or write) to the slave through an additional *rnw* (read, not write) control wire:



Implement a protocol channel for this bus (with master and slave interfaces and corresponding `masterRead`/`Write` and `slaveServeRead`/`Write` transactions). You can start from the *DblHndShkBus* protocol 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/fall_11/ee382v_17190/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). Make

the necessary modifications to implement both a pin-accurate and a transaction-level model of the bus that match the timing diagram given above. Assume worst case delays.

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.

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?

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.

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: Is this an efficient transaction-level implementation in terms of simulation performance? Give some suggestions to improve TLM speed. Can you write a faster and/or more accurate TLM? Hint: have a look at references [28,29].