Embedded System Design and Modeling

EE382N.23, Fall 2017

Homework #3

System Modeling & Refinement

Assigned: October 16, 2017
Due: October 30, 2017

Instructions:

- Please submit your solutions via Canvas. 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: Source-Level Simulation

Given the following C code and its ARM assembly implementation, and assuming execution on an ARM micro-architecture with a typical 5-stage in-order pipeline (consisting of fetch, decode, execute, memory and write-back stages), full data forwarding/bypassing, and static not-taken branch prediction with branches resolved in the execution stage, can you find an annotation of the C code with `waitfor()`statements that accurately replicates the timing of the ARM execution? State all your assumptions.

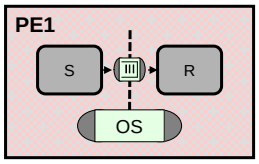
```
int f(int *a, int *b) {
  int i, t;
  int c = 0;

  for(i = 0; i < 100; i++)
  {
    t = a[i];
    if (t) {
      t = b[i];
    }
    c = c + t;
  }
  return c;
}
```

```
; a in R0, b in R1
f       MOV R2,#0       ; c in R2
        MOV R3,#0       ; i in R3
loop    CMP R3,#100     ; i < 100?
        BGE end
        LDR R4,[R0,R3 LSL #2] ; t = a[i]
        CMP R4,#0       ; t == 0?
        BNE skip
        LDR R4,[R1,R3 LSL #2] ; t = b[i]
skip    ADD R2,R2,R4    ; c = c + t
        ADD R3,R3,#1    ; i++
        B   loop
end     MOV R0,R2       ; c in R0
        BX  LR
```

Problem 3.2: Computation and Communication Refinement

For this problem, we will further refine the Producer-Consumer example from Problem 1.1 in Homework 1 all the way down to both pin-accurate and transaction-level models of different design variants. Start from the code for the specification model that you developed for Problem 1.1(e)(4) in Homework 1 (or the reference solutions provided).

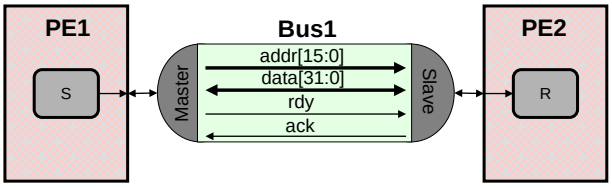
(a) First assume an implementation in which both `S` and `R` behaviors are mapped to the same *PE1*. This will require the two concurrent behaviors to execute and be dynamically scheduled under the control of an operating system (OS). Manually refine the specification model into a scheduled computation model that reflects these design decisions:



Wrap the parallel Producer-Consumer behavior pair into a *PE1* behavior and insert an OS model channel that emulates dynamic scheduling in a round-robin fashion. You can refer to slides 19-21 in Lecture 8 and references [19] and [20] on the class website for more details about OS modeling

Write an OS model that is based on cooperative multitasking, i.e. that provides a *yield()* method to be called by application tasks in order to trigger a task/context switch. Convert parallel S and R application behaviors into tasks that call OS channel methods to wait for OS activation at the beginning of their execution and terminate themselves at the end of their *main()*. Insert *yield()* calls into the loop bodies of S and R behaviors to enable cooperative scheduling. Does it matter where the *yield()* calls are inserted? Why or why not?

(b) Now assume an implementation in which the S behavior is mapped to PE1 and the R behavior to *PE2*, where a single *Bus1* connects *PE1* (bus master) and *PE2* (bus slave). Manually refine the specification model into a communication model that reflects these design decisions:



We will use a simple HW bus protocol with address, data and control wires for all PE-to-PE interconnect and communication. Source code for a pin-accurate model (PAM) of a corresponding *HWBus* protocol channel is available at:

`/home/projects/courses/fall_17/ee382n-23/HWBusPAM.sc`

Wrap the S and R behaviors into *PE1* and *PE2* behaviors, respectively, and insert an instance of the *HWBusPAM* protocol channel to connect the two PEs. Refine all communication between S and R into transactions over *Bus1* that realize equivalent semantics and desired overall application behavior/functionality.

Finally, implement a transaction-level model (TLM) of the *HWBus* channel where the internal communication is not realized via wires but as abstract function calls. Replace the existing *HWBusPAM* instance with an instance of your *HWBusTLM* channel connecting the two PEs. The TLM channel should implement the exact same interface as the PAM, such that no code inside the PE behaviors will have to be touched for this plug-and-play replacement.

What accuracy (in measured latencies) and speedup can your TLM reach compared to the PAM? 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? What is the actual speedup in your simulations? Can you think of any

further ways for speeding up the simulation (with our without a loss in accuracy compared to the PAM)?