Embedded System Design and Modeling

ECE382N.23, Fall 2022

Homework #2

Design Languages & Architecture Modeling

Assigned: October 13, 2022
Due: ~~October 28, 2022~~ November 4, 2022

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 2.1: Discrete-Event Semantics

For each of the following code examples, what is the value of `myB` at the end of execution and at what simulated time does the program terminate. You are free to run the code on top of the SystemC simulator and observe the program output, but you need to provide an explanation and reasoning of why the program is behaving as it is (e.g. sequence of events happening during simulation):

(a)

```
void M::A(void)
{
  myB = 10;
};

void M::B(void)
{
  myB = 42;
};

SC_MODULE(M)
{
int myB;

  void A(void);
  void B(void);

SC_CTOR(M) {
    SC_THREAD(A);
    SC_THREAD(B);
  }
};
```

(b)

```
void M::A(void) {
  wait(42, SC_NS);
  myB = 10;
};

void M::B(void)
{
  myB = 42;
};

SC_MODULE(M)
{
  int myB;

  void A(void);
  void B(void);

  SC_CTOR(M) {
    SC_THREAD(A);
    SC_THREAD(B);
  }
};
```

(c)

```
void M::A(void) {
  myB = 10;
  wait(42, SC_NS);
};

void M::B(void) {
  wait(10, SC_NS);
  myB = 42;
};

SC_MODULE(M)
{
  int myB;

  void A(void);
  void B(void);

  SC_CTOR(M) {
    SC_THREAD(A);
    SC_THREAD(B);
  }
};
```

(d)

```
1  void M::A(void)
2  {
3    myA = 10;
4    e.notify();
5    myA = 11;
6    e.notify();
7    wait(10, SC_NS);
8  };
9
10 void M::B(void)
11 {
12   wait(e);
13   myB = myA;
14 };
15
16 SC_MODULE(M)
17 {
18   int myA;
19   int myB;
20   sc_event e;
21
22   void A(void);
23   void B(void);
24
25   SC_CTOR(M) {
26     SC_THREAD(A);
27     SC_THREAD(B);
28   }
29 };
```

(e)

```
1  void M::A(void)
2  {
3    myA = 10;
4    e.notify();
5    wait(10, SC_NS);
6    myA = 11;
7    e.notify();
8  };
9
10 void M::B(void)
11 {
12   wait(e);
13   myB = myA;
14 };
15
16 SC_MODULE(M)
17 {
18   int myA;
19   int myB;
20   sc_event e;
21
22   void A(void);
23   void B(void);
24
25   SC_CTOR(M) {
26     SC_THREAD(A);
27     SC_THREAD(B);
28   }
29 };
```

(f) What code has to be inserted at the beginning of M::B (line 12) in (e) to change the output of the program? What must *not* appear there for the program not to deadlock?

Problem 2.2: Architecture Modeling

In this problem, you will use the SystemC C++ library to refine a KPN into a proper architecture model. Follow the SystemC setup guide and tutorial posted on the class page at:

http://www.ece.utexas.edu/~gerstl/ece382n_f22/docs/SystemC_setup.pdf

to setup the environment and get familiar with it. To get you started for this problem, download a private copy of the repository located at:

https://github.com/esalcort/KPN-Refinement

(a) The *kpn-arch* folder contains a *kpn-arch.cpp* file that models a KPN with three processes, *A*, *B*, and *C*, mapped onto an SoC architecture with two processing elements, *PE1* and *PE2*. Since *A* and *B* are mapped to the same *PE1*, we need to implement an OS that switches between the two. The repository that we have given you includes an *os_api.h* file with an incomplete implementation of an OS. Extend the code to complete the implementation. Note that the interface should not be modified, you should only modify the channel implementation (line 30 and below). Additionally, note that *wait()* statements with annotated execution delays have been overloaded in *kpn-arch.cpp* with OS *time_wait()* methods to model preemption. Compile and simulate the model, and validate that the simulation using your OS model implementation executes as expected. How long does it take to simulate the code (in wall-clock time)? What is the total simulated time? Report the simulation log and your changes to *os_api.h*. Assume that "C2" produces the outputs of this model. What is the latency and throughput of the model?

(b) To further refine our KPN, we need a bus model that simulates the communication between *PE1* and *PE2*. For this purpose, we will use a simple hardware bus protocol with address, data and control. A detailed, pin-accurate implementation and a transaction-level model (TLM) of this bus protocol is provided in the *HWBus.h* file. The pin-accurate bus model defines physical layer realizations for bus wires and a protocol-level implementation for master (*MasterHardwareBus*) and slave (*SlaveHardwareBus*) sides. The transaction-level model (*HardwareBusProtocolTLM*) replaces wires and physical layer protocol state machines with plain variables and events to model bus communication semantics and delays. In addition, media access (MAC) channels (named [*Master*|*Slave*]*HardwareBusLinkAccess*) show the methods of how to access the bus.

i. Refine the model from part (a) into a transaction-level model (TLM) at an abstracted level, where *PE1* is the bus master and *PE2* is the bus slave. For this, the bus is modeled by a single instance of the *HardwareBusProtocolTLM* channel in *HWBus.h*. Create a *kpn-TLM.cpp* file where you refine the existing *kpn-arch.cpp* architecture model of the system into a TLM-based communication model. The model refinement is achieved by creating a master driver on *PE1* and a slave driver on *PE2* that service the *write()* or *read()* requests from the application processes by translating them into bus communication (accessing the MAC channels). See Fig. 1 for reference. Note that as part of this process, you may have to add synchronization from slave to master (to let the master know that a slave is ready to accept a transaction), which can be done by instantiating *sc_mutex* or *sc_semaphore* channels as appropriate at this abstract TLM level. Run your simulation and report any changes in simulated time. Have the latency and throughput changed? How long does it take to simulate the code (in wall-clock time) and how does that compare to the simulation time of the architecture model?

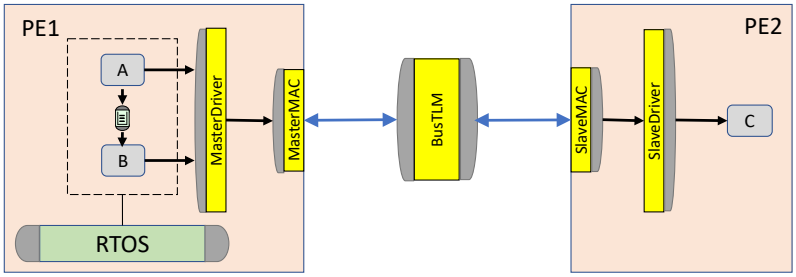

Figure 1. Transaction-level model (TLM)

ii. (Extra credit) Next, we are going to replace the transaction-level model of the system with a pin-accurate model (PAM),. Create a copy of your *kpn-arch.cpp* file and name it *kpn-PAM.cpp* to complete this task. Such a PAM has a similar structure as the TLM, but the TLM bus channel is replaced with instances of physical layers and wires that will transfer the data following the bus protocol state machines. See Fig. 2 for reference. Does your PAM reach the same accuracy (in measured latencies and throughput) as the TLM? Why? Measure the wall-clock simulation time. How does it differ from the TLM? Discuss any trade-offs you find between the PAM and TLM models.

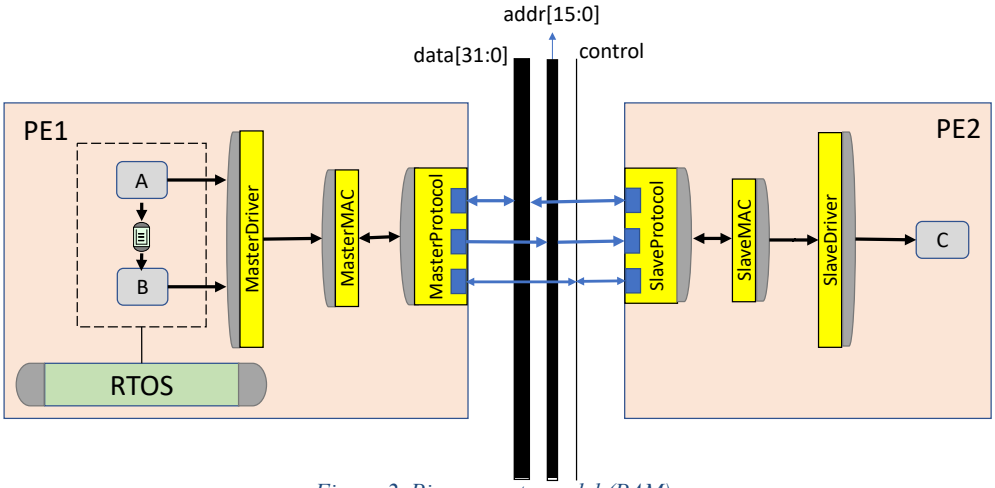

Figure 2. Pin-accurate model (PAM)