Embedded System Design and Modeling

EE382V, Fall 2009

Lab #4

Exploration and Refinement

Due: November 12, 2009 in class (3:30pm)

Instructions:

- Please submit your solutions via Blackboard. Submissions should include a single PDF with a lab report and a single Zip or Tar archive with the source and supplementary files.
- You are allowed to work in teams of up to three people and you are free to switch partners between labs and the project. Please submit one solution per team.

Part 1: System-On-Chip Environment (SCE)

The goal of this first part is to continue the System-On-Chip Environment (SCE) tutorial that was started in the previous lab:

http://www.cecs.uci.edu/~cad/publications/tech-reports/2003/TR-03-41.tutorial.pdf
http://www.ece.utexas.edu/~gerstl/ee382v_f09/docs/SCE_Tutorial_Errata.pdf

Go back to your local working directory for the tutorial demo, launch the SCE GUI and follow the steps of the tutorial in Section 3. Make sure to simulate all the generated models and generate both a TLM and a PAM in the final Communication Synthesis step. You are free to venture into HW and SW synthesis steps (Sections 4 and 5), but those have not been tested and are not required at this point; use at your own risk but if you do, we'd be happy to hear about any successes/failures.

Part 2: Digital Camera Synthesis

The main purpose of this lab is to explore the system design space and synthesize the previously developed specification of the digital camera example down to a MPSoC implementation at the TLM and PAM levels. We start from the SpecC code developed as a result of Lab #3:

`/home/projects/courses/fall_09/ee382v-17220/jpegencoder3.tar.gz`

(a) We are now ready to go through the architecture and scheduling exploration and refinement process. The goal is to find an optimal realization on a system architecture consisting of up to 3 ARM processors:

1. Open SCE in the digital camera directory and load the "digicam.sce" project file created in the first lab (`Project→Open`). Open the "DigicamSpec.sir" specification model added to the project in the previous lab and set the *Design* behavior as the top level.

2. Allocate two custom hardware PEs of *HW_Virtual* type and name them *CCD* and *FLSH*. Those two custom hardware blocks are placeholders for the controllers implementing the I/O with the external CCD sensor and flash memory. As such, map the *ReadBlock* and *WriteBlock* behaviors onto the *CCD* and *FLSH* PEs, respectively. Furthermore, enable the channel view (`Synthesis→Show Channels`) and map the *dctin* and *dataout* input and output queues at the *Design* level into the *CCD* and *FLSH* PEs, respectively. This is necessary because we want to have the two I/O blocks implement the dedicated input and output buffers

associated with the queues. In general, mapping a complex channel into a PE means that the channel will result in a specific implementation being synthesized as part of that PE. Unmapped complex channels, on the other hand, will simply be resolved into their basic elements without any guarantees about a particular buffer realization, for example.

3. Allocate between 1 and 3 processor PEs of *ARM7_TDMI* type and explore possible mapping options of *JpegEncoder* behaviors onto 1, 2, or 3 *ARM*x processors. When allocating ARMs, use the default parameters (100MHz clock frequency). Make sure to reprofile and reanalyze the design every time you change the allocation. Perform architecture refinement for every feasible design alternative.
4. Explore various feasible scheduling strategies for each allocated ARM processor in each architecture alternative. You can choose from static, round-robin and priority-based scheduling (with various task priority assignments) of parallel behaviors mapped to the same ARM. Note that you should not schedule (i.e. select *None* under dynamic scheduling) ARM processors with only one mapped behavior (there is a bug in SCE that prevents dynamic scheduling on more than one processor). Do not schedule any of the hardware units. Perform scheduling refinement for every feasible design alternative.
5. Compile and simulate all generated scheduled architecture models. Record the simulated encoding times for each alternative and plot the design space as points in an encoding time vs. cost (equal to # of processors) graph. What is the best design?

(b) In the next step, we can then go into network exploration and communication synthesis for various promising architecture candidates that we identified in (a). Specifically, we want to synthesize the best designs each with 1, 2 or 3 ARM processors down to a TLM or PAM realization:

6. Open network allocation to define the overall network topology. Busses for each ARM processor in the system should already be pre-allocated and ARM processors should be pre-connected as masters on their respective busses. Connect the *CCD* and *FLSH* hardware PEs as slaves on the same bus as the ARM processor running their direct communication partners (i.e. *Dct* and *Huff* behaviors, respectively). Note that "slave0" connectivity is reserved by the ARM processor itself and should never be used. If there is more than one ARM (and hence more than one bus) in the system, allocate transducer CEs of *T_Custom* type to bridge and connect busses as necessary (where transducers are slaves on each bus they connect to). Note that transducers by default only have one port but additional ports can be created by right-clicking on the transducer name in the Connectivity tab and selecting `Add port…`.
7. Perform network refinement and make sure to select a custom packet size of 256 bytes in the process. An increased packet size will make sure that every image block can be transferred over any transducers in a single packet (64 elements/block times 4 bytes/integer), greatly reducing the overall communication overhead (you can experiment with synthesizing a design with a packet size of 1 and comparing the final encoding times).
8. Assign the link parameters for each channel on each bus. You can freely choose the interrupt/synchronization scheme. However, due to the mux-based architecture of

the ARM/AMBA AHB bus being used, addresses need to be assigned to match the slave connectivity. Specifically, channels served by a particular "slave*N*" have to be assigned a bus address in the range between `0x`***N***`0000000-0x`***N***`fffffff` (otherwise, you will see a deadlock in the PAM simulation).

9. Perform communication refinement to generate both a transaction-level and pin-accurate model of each design. Compile and simulate each model to record the final encoding delays. How much percent communication overhead does each design have?

(c) In a final step, we will create a mixed HW/SW system design with a hardware-accelerated DCT. We will synthesize this design all the way down to binary object code for the software running on top of an uCOS-II real-time operating system (RTOS) in an ARM instruction set simulator (ISS) that is co-simulated with the other digital camera hardware:

1. For software synthesis to work, we need to replace the *c_int64_queues* inside the *JpegEncoder* with untyped *c_double_handshake* channels (software code generation currently does not support queue synthesis). You can do the conversion yourself or use the pre-prepared "jpecencoder3.tar.gz" master solution from above. For the latter, running

```
make clean
make DEFINES=-DSW_SYNTH
```

will compile a version of "digicam.sir" with the required changes.

2. Load the modified digital camera specification "digicam.sir" into SCE, add it to the project and rename it to "DigicamSystemSpec.sir". Select *Design* as top level.

3. Allocate and map *CCD* and *FLSH* PEs as before. Allocate a single *ARM* PE and map the *JpegEncoder* onto the *ARM*. Allocate a custom hardware PE of *HW_Standard* type and map the *Dct* behavior onto it. Compile, simulate, profile and analyze the design. Perform architecture refinement.

4. Apply priority-based scheduling on the *ARM* and assign priorities 1 and 2 to *Quant* and *Huff*, respectively. Do not schedule any of the *CCD*, *FLSH* or *DCT* PEs. Perform scheduling refinement and compile and simulate the design.

5. Open network allocation, rename *Bus0* to *AHB* and connect the *CCD* and *FLSH* as "slave2" and "slave3" to the *AHB*. Connect the *DCT* as bus-mastering hardware ("master1 & slave1") to the *AHB*. The *DCT* needs to be both bus master and slave such that it can communicate with both the *ARM* (which is a "master0") and the *CCD* (which is a "slave2"). Alternatively, you can make the *DCT* a slave-only on the *AHB* and allocate a separate *DblHndShkBus* as a point-to-point connection between *CCD* and *DCT* (adding separate ports as necessary). In either case, perform network refinement and compile and simulate the design to validate that it works correctly.

6. Assign link parameters on the *AHB* bus such that channels connecting to *CCD* map to address `0x200000`***xx***, channels connecting to *FLSH* map to `0x300000`***xx***, and channels between *DCT* and *ARM* map to address `0x100000`***xx***. Assign unique or shared interrupts to each channel (do not choose polling). Perform communication refinement to generate a PAM. Compile and simulate the PAM. How does the performance of the design compare to the ones created under (b)?

7. Due to a bug in the GUI we need to patch the PAM to attach an additional annotation for software synthesis to work. Go to the command line and run the following command:
   ```
   sir_note <pam_name> ARM_7TDMI_Core_20000_0_<arm_name>
                  '_PE_HAL_MODEL="ARM_7TDMI_HAL_20000_0_<arm_name>"'
   ```
   where <*pam_name*> is the design name of the PAM file and <*arm_name*> is the name of the ARM processor assigned during allocation (e.g. *ARM*).

8. Reload the PAM in SCE and select `Synthesis→C Code Generation`. Choose output files "ARM/ARM.c" and "ARM/ARM.h" and run the refinement process.

9. The master "jpegencoder3.tar.gz" tarball comes with an "ARM/Makefile" for cross-compiling the generated ARM source code and linking it against the ARM-ported uCOS-II and other runtime libraries on the LRC machines. If it is not already there, copy that "Makefile" into the ARM subdirectory next to the generated code, compile the code and copy the generated ARM executable into the SCE project directory:
   ```
   cd ARM
   make
   cp userCode ..
   ```

10. In the last step, we need to insert the ISS model for the ARM into the PAM and replace the SCE-generated ARM model with the ISS version. Open the PAM in SCE, select `Edit→Import Design` and choose and import the file "$SPECC/share/sce/db/processors/general/arm7tdmiiss.sir". Locate the *ARM* instance under the top-level *Design*, right-click it and `Change Type` to *ARM_7TDMI_ISS*. Save the design as a new model (`File→Save As…`) in the project directory. Compile and simulate the design. You might see some IRQ messages flying by as the ISS is running but in the end the simulation should stop after some time when the picture is encoded. Compare the final encoding time to the previous PAM simulation result, how much differences are there?

11. Congratulations, we achieved a full-system co-simulation of the actual target software binary together with its surrounding hardware for the complete SoC!