EECS 222A
System-on-Chip Description and Modeling
Fall 2009

Assignment 3

Posted: October 23, 2009
Due: October 30, 2009 at 12pm (noon)

Task: Convert the JPEG encoder application into a SpecC model

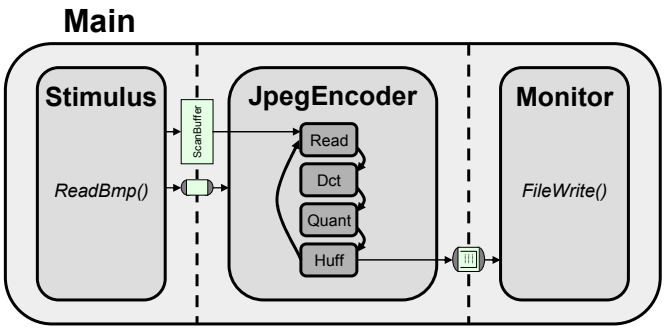
Instructions:

The purpose of this assignment is to convert the JPEG Encoder reference code into an initial SpecC model of the digital camera with proper behavioral and structural hierarchy.
Starting from the simplified, static code developed in the previous Assignment 2, (see reference **`/home/doemer/EECS222A_F09/jpegencoder1.tar.gz`**) we will gradually convert **`<name>.c/.h`** C files into **`<name>.sc/.sir`** SpecC modules. In the process, each module gets translated into one or more SpecC behaviors, which can then be hierarchically imported and composed into an overall design:

1. Convert **`read.c`**, **`dct.c`**, **`quantize.c`**, **`zigzag.c`** and **`huffencode.c`** into corresponding **`.sc`** files. Introduce a single behavior of appropriate name in each file. Let the behavior encapsulate all local variables and functions (i.e. files must not have any variables or functions outside of behaviors). Convert the externally accessible function listed in the corresponding **`.h`** file into the behavior's main method and replace parameters with equivalent behavior ports for external communication. Ensure that behaviors are free of side effects, i.e. that they only communicate with other behaviors through their ports and do not access any global variables outside of their body.
2. Convert **`preshift`**, **`chendct`** and **`bound`** methods in **`dct.sc`** into separate behaviors and transform the **`Dct`** behavior into a sequential composition of these subbehaviors. Connect the child behaviors so that they communicate through variables mapped onto their ports.
3. Introduce a new behavior and file **`huff.sc`** that implements the sequential composition of imported **`Zigzag`** and **`Huffencode`** child behaviors. Connect behavior ports to appropriate external ports or local variables throughout the hierarchy.
4. Convert **`ReadBmp_aux.c`** and **`file.c`** into **`ReadBmp.sc`** and **`file.sc`** files that implement **`Stimulus`** and **`Monitor`** behaviors for the testbench, respectively. The **`Stimulus`** behavior reads the input file into a shared **`ScanBuffer`** port (ReadBmp) and then sends a start signal over a

c_handshake channel. The **Monitor** reads bytes from a **c_queue** interface and writes them into an output file (FileWrite) continuously, one byte at a time until the end-of-file marker is reached.

5. Convert **jpegencoder.c** into a **jpegencoder.sc** file and behavior that first waits for a start signal via a **c_handshake** interface and then executes **ReadBlock**, **Dct**, **Quantize** and **Huff** child behaviors sequentially in a loop. Let child behaviors communicate through variables mapped onto their ports and introduce external ports and mappings as necessary.
6. Introduce a top-level **digicam.sc** file that contains the **Main** behavior implementing a typical testbench setup running **Stimulus**, **JpegEncoder** and **Monitor** subbehaviors concurrently:



The **Stimulus** is connected to the **JpegEncoder** through a shared **ScanBuffer** variable representing the CCD sensor array. In addition, a **c_handshake** channel represents the signal that the camera shutter has been triggered and that encoding of the CCD sensor picture should be started. At the other end, the **Monitor** receives a stream of encoded bytes from the Huffman encoder (**Huffencode**) through a **c_queue** representing the file I/O interface.

7. Remove the **.h** files and compile all **.sc** sources into **.sir** files and check for compile errors. Finally, compile the top-level **digicam.sc** source into an executable and simulate the design. Validate the generated output against the known good data to ensure the design is working correctly.

Note: Because of the multiple files in this design specification, it is highly recommended to update the **Makefile** in order to automate the compilation process using the **make** utility.

If you are unfamiliar with **make** and the **Makefile** and want to avoid handling multiple files, you can create one large specification file instead (**digicam.sc**) which contains all code in a single file.

Deliverables:

Email to **`doemer@uci.edu`** with
(a) Brief description (max. 5 sentences!) of the status of your model, and
(b) Source code attached in a **`.tar.gz`** archive

--
Rainer Doemer (EH3217, x4-9007, doemer@uci.edu)