Embedded System Design and Modeling

EE382V, Fall 2009

Lab #2

SpecC Conversion

Due: October 1, 2009 in class (3:30pm)

Instructions:

- Please submit your solutions via Blackboard. Submissions should include a single PDF with the lab report and a single Zip or Tar archive with the source 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.

SpecC Model of Design Example

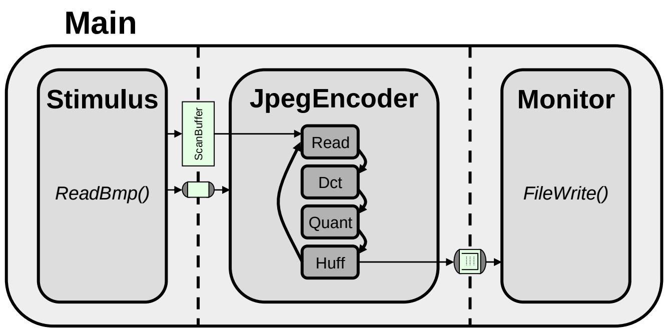
The purpose of this lab is to convert the JPEG Encoder reference code into an initial SpecC model of the digital camera with proper behavioral and structural hierarchy. Starting from the simplified, static code developed as a result of Lab #1:

```
/home/projects/courses/fall_09/ee382v-17220/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 *.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 child behaviors to 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 *.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: it is highly recommended to update the *Makefile* in order to automate the compilation process using the `make` utility.

Briefly report on how the resulting model employs the concepts of and follows the guidelines for granularity, encapsulation, hierarchy, concurrency, and communication. Describe if and how this model could be improved in any of these aspects.