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

Assignment 5

Posted: November 6, 2009
Due: November 13, 2009 at 12pm (noon)

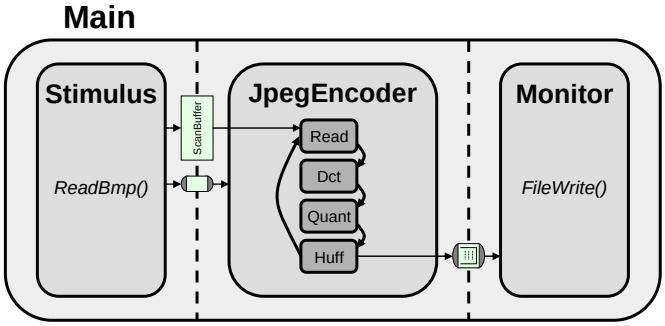
Task: Create a pipelined and synthesizable JPEG Encoder model

Instructions:

The purpose of this assignment is to finalize the SpecC specification model of the digital camera example in order to obtain a clean and parallel/pipelined specification model that conforms to the structure, rules, and guidelines discussed in class and can be explored and synthesized using SCE.

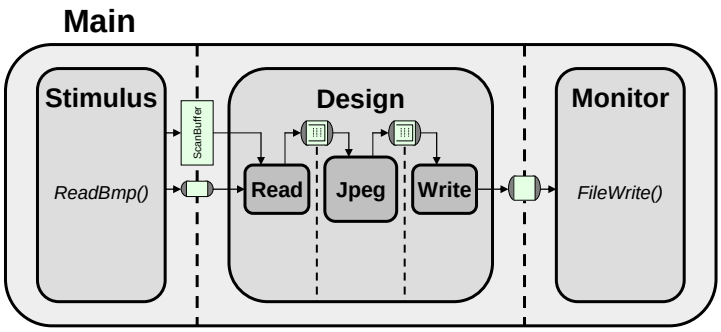
We will start from the SpecC model developed in the previous Assignment 3. As a reference solution, you may use the following model as starting point:

`/home/doemer/EECS222A_F09/jpegencoder2.tar.gz`



1. **Insert timing checks into the test bench:** Update the **`Stimulus`** behavior in **`ReadBmp.sc`** to wait for 1000 time units before sending the start signal to the encoder. Make this start time available to the **`Monitor`** behavior via a new variable **`start_time`** in the test bench. Let the **`Monitor`** in **`file.sc`** print the total delay that the **`jpegencoder`** spends on encoding of a single picture. Compile and simulate the model to ensure that the timing info is printed correctly (the reported delay should be zero at this point).
2. Develop an accurate model of the actual I/O structure for the digital camera which consists of two dedicated I/O units **`read`** (for input) and **`write`** (for output) that run in parallel to the **`jpegencoder`**: Move the **`ReadBlock`** behavior outside of the **`JpegEncoder`**. Move the waiting for

the start signal into **ReadBlock**. Also, modify **ReadBlock** to independently loop over all 180 blocks in a picture and send each block into its outgoing queue after the start signal has been received. Introduce a **WriteBlock** behavior (**write.sc**) that continuously reads bytes from a queue and forwards them into an outgoing double-handshake channel. Introduce an additional level of hierarchy as a **Design** behavior (**design.sc**) that sits between **Monitor** and **Stimulus** and is a parallel composition of **ReadBlock**, **JpegEncoder** and **WriteBlock** instances communicating via **c_queue** channels. The input queue should have space for 1 block of data, the output queue should be 512 bytes in size. This all should result in the following structure:



3. Convert the **JpegEncoder** block into a pipelined parallel model: Remove the **ReadBlock** instance (as discussed above) and change the top-level **JpegEncoder** execution into a single par statement in which the three remaining child behaviors communicate via **c_typed_queue** channels of size 1 data block. Examples for use of typed queues are available in:

   /home/doemer/EECS222A_F09/queue.sc
   $SPECC/examples/sync/c_bit64_queue.sc
   $SPECC/examples/sync/typed_queue.sc

4. Modify **Dct**, **Quantize** and **Huff** to infinitely work on continuous streams of input and output data over **c_int64_queue** channels: Change the sequential sub-composition inside **Dct** and **Huff** behaviors into a **fsm** behavior that runs child behaviors sequentially in an endless loop. Introduce an additional level of hierarchy in **quantize.sc** as a behavior **Quant** that runs **Quantize** in an endlessly looping FSM. Replace the top-level **Quantize** instance in **JpegEncoder** with **Quant**.
5. Compile and simulate the design model in SCE to validate its correctness.
6. Create a hierarchy chart that shows the entire hierarchy and connectivity of your design. Print this chart in color (!) as a PS file and convert it to PDF (in the shell, use **ps2pdf digicam.ps**). Your figure should match the

reference chart shown on the last page of these instructions (except for being in color!).

Deliverables:

Email to **doemer@uci.edu** with

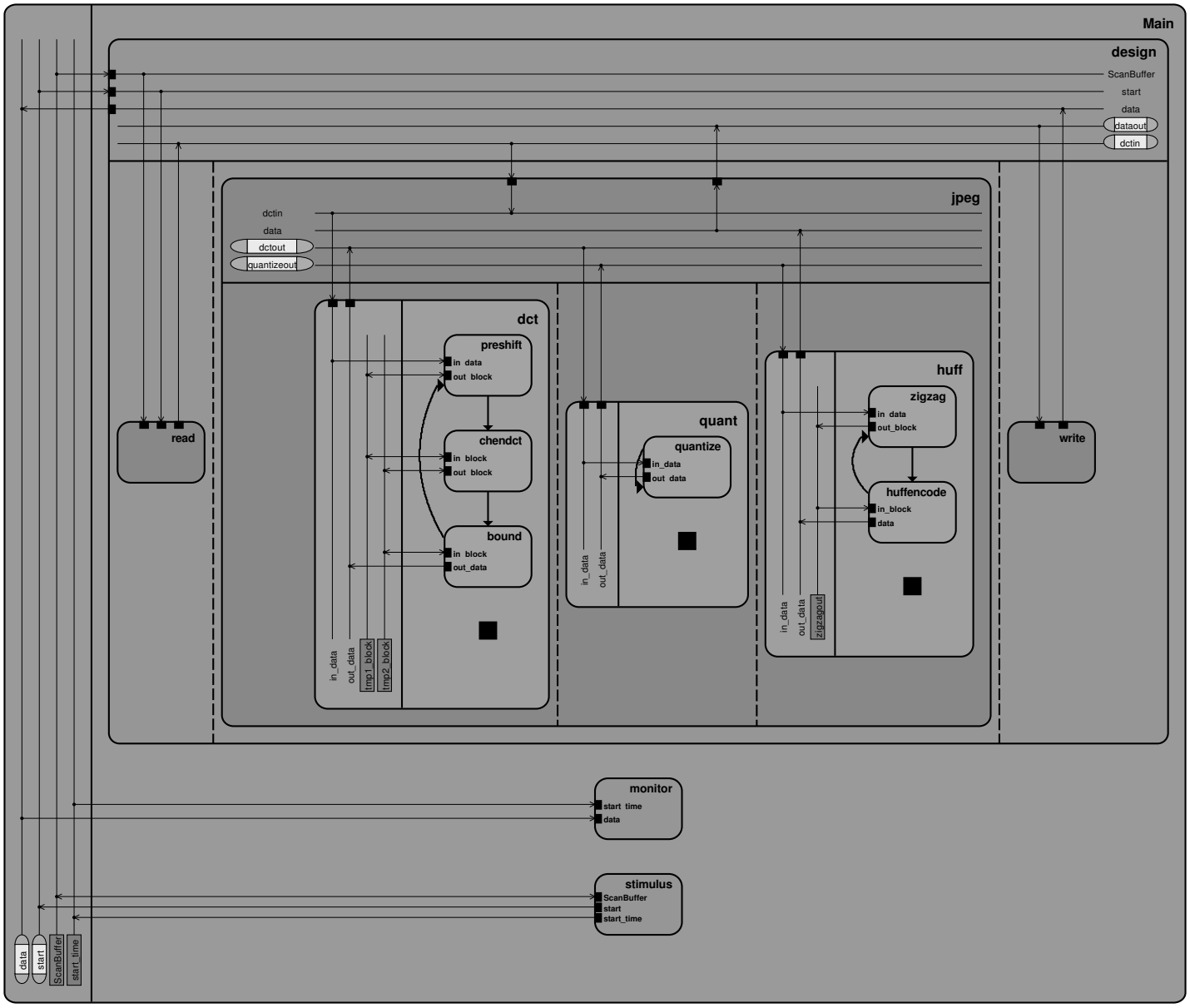
(a) Brief description (max. 5 sentences!) of the status of your model,

(b) Source code attached in a **.tar.gz** archive

(c) Your hierarchy chart **digicam.pdf**.

--

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


Main
design
ScanBuffer
start
data
dataout
dctin
jpeg
dctin
data
dctout
quantizeout
dct
preshift
in data
out block
chendct
in block
out block
bound
in block
out_data
in_data
out_data
tmp1_block
tmp2_block
quant
quantize
in_data
out data
in_data
out_data
huff
zigzag
in data
out_block
huffencode
in_block
data
in_data
out_data
zigzagout
read
write
monitor
start time
data
stimulus
ScanBuffer
start
start_time
data
start
ScanBuffer
start_time