UC Irvine
ICS Technical Reports

Title
Design of a JPEG encoder using SpecC methodology

Permalink
https://escholarship.org/uc/item/7q05h8kq

Authors
Yin, Hanyu
Du, Haitao
Lee, Tzu-Chia
et al.

Publication Date
2000-07-24

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

Z
699
C3
no. 00-23

Notice: This Material
may be protected
by Copyright Law
(Title 17 U.S.C.)

Design of a JPEG Encoder using SpecC Methodology

Hanyu Yin
Haitao Du
Tzu-Chia Lee
Daniel D. Gajski

July 24, 2000

Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425, USA
(949) 824-8059

{hyin, hdu, tzuchial, gajski}@ics.uci.edu
http://www.ics.uci.edu/~cad

Contents

1 Introduction **1**

2 Introduction of JPEG **2**

3 Specification Model **2**
3.1 Test Bench: Read.Bmp File and Write.Bmp File . . . 3
3.2 Blocks in the JPEG encoder . . . 3
3.2.1 HandleData Block . . . 3
3.2.2 DCT Block . . . 3
3.2.3 Quantization Block . . . 3
3.2.4 HuffmanEncoder Block . . . 3

4 Architecture Model **3**
4.1 HW/SW Partition . . . 3
4.2 Channel Partition . . . 4

5 Communication Model **4**
5.1 Bus Protocol . . . 5
5.1.1 ColdFire Master Bus . . . 5
5.1.2 DCT Bus . . . 6
5.2 Transducer Design . . . 8
5.2.1 Transducer FSMD for data transfer from DCT to ColdFire . . . 8
5.2.2 Transducer FSMD for data transfer from ColdFire to DCT . . . 8

6 Implementation Model of DCT Block in JPEG Encoder **9**

7 Conclusions **10**

References **10**

A Specification Model **12**
A.1 JPEG Encoder . . . 12
A.1.1 const.sc . . . 12
A.1.2 global.sc . . . 12
A.1.3 chann.sc . . . 15
A.1.4 handle.sc . . . 16
A.1.5 dct.sc . . . 22
A.1.6 quant.sc . . . 25
A.1.7 huff.sc . . . 26
A.1.8 jpeg.sc . . . 29
A.2 Testbench . . . 30
A.2.1 io.sc . . . 30
A.2.2 tb.sc . . . 34

B Architecture Model **36**
B.1 SpecC Code before Channel Partition . . . 36
B.1.1 sw.sc . . . 36
B.1.2 hw.sc . . . 37
B.1.3 jpeg.sc . . . 38
B.2 SepcC Code after Channel Partition . . . 38
B.2.1 sw.sc . . . 38
B.2.2 hw.sc . . . 40

B.2.3 jpeg.sc 41
B.2.4 bus.sh 41
B.2.5 bus.sc 41

C Communication Model **43**
C.1 SpecC Code for the Communication Model 43
C.1.1 sw.sc 43
C.1.2 hw.sc 44
C.1.3 jpeg.sc 46
C.1.4 bus.sh 46
C.1.5 bus.sc 46
C.1.6 transducer.sc 51

D Implementation Model (DCT only) **54**
D.1 SpecC Code for the Implementation Model of the HW 54
D.1.1 bus.sc 54
D.1.2 transducer.sc 60
D.1.3 hw.sc 62
D.1.4 dct.sc (Behavioral RTL) 63
D.1.5 dct.sc (Behavioral RTL with known Data Path and bound variables) 82

List of Figures

1 The SpecC methodology . . . 1
2 Block diagram of the JPEG encoder . . . 2
3 Specification model of the JPEG encoder . . . 2
4 Model after behavior partitioning . . . 4
5 Model after channel partitioning . . . 4
6 ColdFire master bus timing diagram . . . 5
7 FSMD for the ColdFire normal read transfer . . . 6
8 FSMD for the ColdFire normal write transfer . . . 6
9 RTL structural diagram of DCT . . . 6
10 DCT bus operation timing diagram . . . 7
11 FSMD for DCT bus transfer . . . 7
12 Transducer between ColdFire and DCT . . . 8
13 FSMD for the transducer . . . 8
14 ColdFire reading DCT . . . 9
15 ColdFire writing DCT . . . 9
16 Superstate FSMD of HW behavior . . . 10
17 Superstate FSMD of the receive hdata part . . . 10
18 Superstate FSMD of the dct process part . . . 10
19 Superstate FSMD of the send ddata part . . . 10

Design of a JPEG Encoding System using SpecC Methodology

H. Yin, H. Du, T. Lee, D.D. Gajski
Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425, USA

Abstract

This report describes the design of a JPEG encoder, using the SpecC system level design methodology developed at CAD lab, UC Irvine. We first begin with an executable specification model in SpecC, and then refine the specification model into a architecture model which accurately reflects the system architecture. Based on the architecture model, a communication model where the communication protocols between the system components are defined are developed. Finally, we refine the communication model of the DCT block into the implementation model, which is the lowest level of abstraction in the SpecC methodology. This Project is a result of a course "System Tools" at Information and Computer Science Department, UC Irvine.

1 Introduction

The goal of this project is to verify SpecC methodology by applying it to the design of a JPEG encoder. In this project, the JPEG encoder was described in four models which represent four different levels of abstraction in the SpecC methodology. Figure 1 shows the SpecC methodology.

The specification model is a pure behavior description. In our project, the SepcC specification model was derived from the C description of the JPEG encoder without introducing any implementation detail. The communication between the behavioral blocks are implemented by using global variables rather than channels since up to now no concurrency and synchronization characteristic is specified.

The architecture model is an refined model from the specification model with the partition of hardware and software. The concurrency and synchronization relationships resulting from this partitioning are explicitly described by substituting the global variables with the channels, which will finally be encapsulated into a global bus. The communication between the software blocks running on the same processor is still implemented by using the global variables.

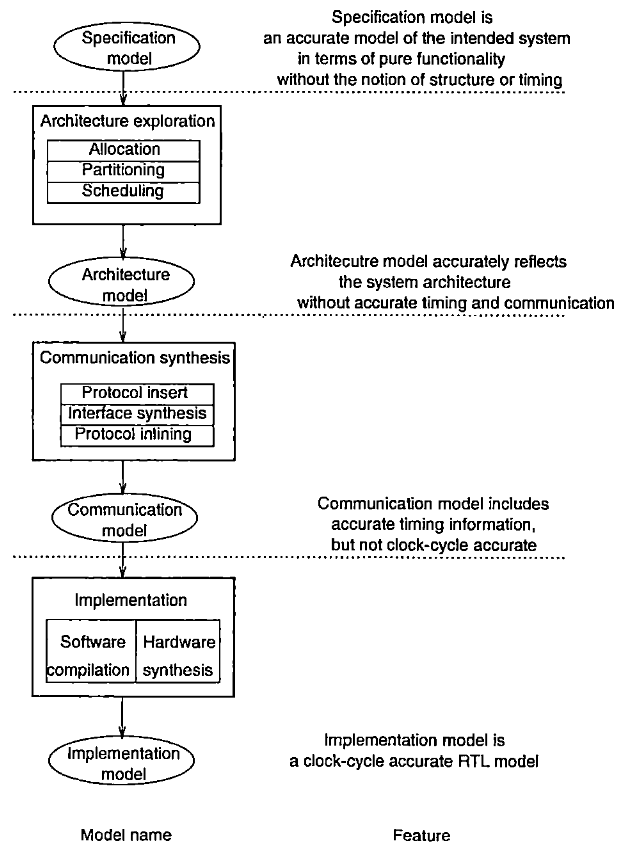

Figure 1: The SpecC methodology

The communication model is the same as the architecture model in that the blocks are mapped to the same components. However,the protocol description for the inter-block communication is refined into the timing-accurate description. This timing-accurate model can be described without knowing the internal structure of the hardware if the I/O protocols between these communicating blocks are clearly defined. But if the protocol is not explicitly given for a block,

the RTL description must be analyzed to generate the corresponding I/O protocol. what's more, if the two protocols for the two communicating blocks are not compatible, a transducer has to be inserted.

Finally, the hardware block, which is DCT block in our design, is refined into the cycle-accurate description, which is the lowest level of abstraction in the SpecC methodology.

The rest of the report is organized as follows: Section 2 describes the JPEG encoder algorithm. Section 3 describes the JPEG specification model. In Section 4, the translation from the specification model to the architecture model is shown. In Section 5, a refined communication model of the JPEG encoder is given. In Section 6, the DCT block of the JPEG encoder is refined into a clock-cycle accurate RTL model. We conclude this report in Section 7.

2 Introduction of JPEG

JPEG is an standard for image compression. It applies to either the full-color images or the gray-scale images of the natural, real-world scenes. Today, parts of JPEG are already available as software-only packages or together with specific hardware support.

There are four modes of the operations in the JPEG standard: the sequential discrete cosine transform (DCT)-based mode, the progressive DCT-based mode, the lossless mode and the hierarchical mode. Our design employs the first mode, the sequential DCT-based mode, which is the simplest and the most commonly used mode.

Based on the sequential DCT-based mode, the JPEG encoder is divided into four functional blocks: the image fragmentation block, the DCT block, the quantization block and the entropy coding block. The corresponding block diagram in Figure 2 illustrates the communication relationship between these blocks.

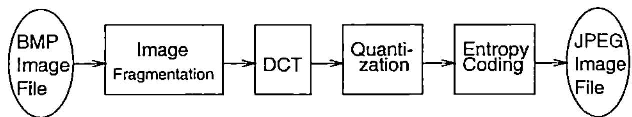

Figure 2: Block diagram of the JPEG encoder

In the image fragmentation functional block, an image is divided into the non-overlapping data blocks, each of which contains an 8*8 matrix of pixels.

In the DCT functional block, each data block is transformed into a frequency representation. There are two commonly used DCT algorithms for this translation process: the standard DCT and the ChenDCT [DCT]. The ChenDCT algorithm is employed in our design.

In the quantization block, the DCT output coefficients are quantized.

Finally, in the entropy coding block, the AC coefficients are encoded by using a predictive coder and the DC coefficients are encoded by using a run-length coder. Then the Huffman coding algorithm is employed to generate the JPEG image.

3 Specification Model

The SpecC specification model models the functionality of the system without introducing any unnecessary implementation details. No timing characteristic is modeled for the computation and communication behaviors. When simulated, the specification model is assumed to be executed in zero time. The specification model is as Figure 3.

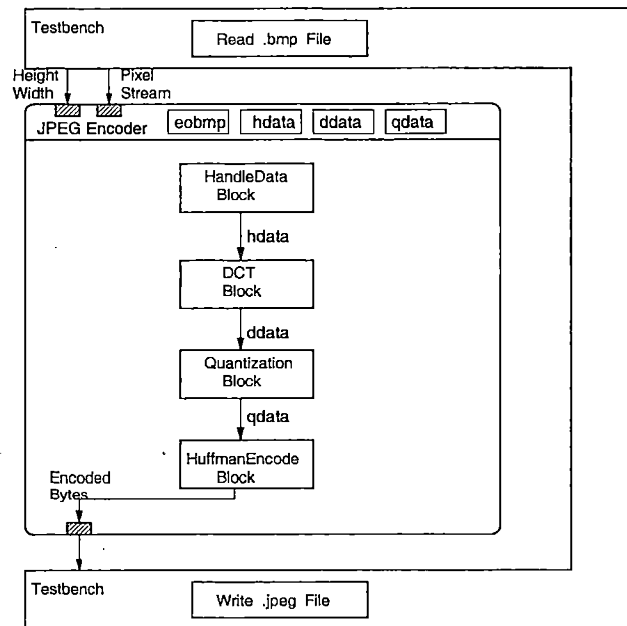

Figure 3: Specification model of the JPEG encoder

The specification model for the JPEG encoder is illustrated in Figure 3 in which there are three top-level behaviors: Read.Bmp File, Write.Bmp File, and JPEG Encoder. Right now, the Height&Width, Pixel stream and Encoded Byte are channels for the communication between these top-level behaviors. The JPEG Encoder is further divided into four sub behaviors: HandleData, DCT, Quantization, and HuffmanEncode. There are three global variables

(hdata,ddata,qdata) are introduced for the data exchange between these sub behaviors since these sub behaviors are sequentially executed. Another global variable (eobmp) is used to indicate the end of the data transfer for the current image. The eobmp is public to all the four behaviors in the JPEG Encoder. The functionality of each behavior is described in the following subsections. The complete SpecC model is given in Appendix A.

Before going to the functionality descriptions, we make one note here: channels are used in the SpecC specification model only if some concurrency and synchronization relationships are explicitly defined. However, later implementation may be different depending on the selected architecture.

3.1 Test Bench: Read.Bmp File and Write.Bmp File

Read.Bmp File and Write.Bmp File can be modeled as the testbench for the JPEG Encoder. First, the height (H) and width (W) of the image are passed from the Read.Bmp File to the JPEG Encoder, which uses them to determine the number of the iterations for the block transfer (H*W). Then serials of the image pixel streams (8-bit wide) are passed to the JPEG Encoder where the image is packeted into blocks and then processed block by block. Each block consists of 8*8 bytes of image information. The processed image is then sent out byte by byte to the Write.Bmp file block where the final JPG image file is generated.

3.2 Blocks in the JPEG encoder

The JPEG encoder includes four blocks: HandleData, DCT, Quantization and HuffmanEncoder.

3.2.1 HandleData Block

The first block, the HandleData block, reads the inputs H, W and pixel stream from the Read.Bmp File, calculates the number of the iterations, groups the pixel stream into 8*8 pixel matrix (MCUs) and sends the MCUs to the DCT block.

3.2.2 DCT Block

The second block, the DCT block, reads the MCUs passed in from the HandleData block, preshifts the MCUs, performs the Chen forward DCT algorithm on them, and sends the result (still 8*8 matrix called transformed MCUs) to the Quantization block.

3.2.3 Quantization Block

The third block, the Quantization block, uses a quantization table to quantize each element of the resulting MCUs passed in from the DCT block and sends the result to the HuffmanEncoder block.

3.2.4 HuffmanEncoder Block

The last block, the HuffmanEncoder block, performs a Huffman entropy-encoding and a run-length-encoding (RLE) on the successive bytes in the incoming MCUs. Each byte is transformed into a bit-sequence (often smaller than the 8 bits of the input byte). The sequence of encoded bits is then packed into bytes and written to the Write.Bmp File block.

4 Architecture Model

The architecture model is an refinement of the specification model in that the behaviors are partitioned into the hardware behaviors and the software behaviors, which can execute concurrently. Because of this, channels are used in the architecture model to describe the concurrency and synchronization relationship between these concurrent behaviors. The use of channels implies that the final implementation of the communication relationship would be the buses. So at the end of the architecture modeling, these channels are encapsulated into one bus. All the communication that go through the channels will be implemented in this bus. Global variables are still used for the communication between the behavioral blocks running on the same processor.

In the architecture model, neither the software blocks nor the hardware blocks are timing-accurate. The communication between concurrent blocks is also not timing-accurate.

In the SpecC design methodology, two steps must be performed to get the final architecture model: HW/SW partition and Channel partition, both of which are described in the following two subsections.

4.1 HW/SW Partition

To get the better trade-off between the performance and cost, HW/SW partition is performed, which involves the estimation of the different partitions. Based on the estimation [CPGC], the JPEG Encoder in our project is partitioned into two parts: 1) the software model, which includes three blocks: HandleData, Quantization, and HuffmanEncode; 2) the hardware model, which includes the block DCT. Even if the

DCT block is called the hardware block, its model is still in the behavioral description in the specC architecture model. The refined model after behavior partitioning is illustrated in Figure 4.

The software model is implemented on the selected processor. In our design, the ColdFire processor is employed. The hardware model can be synthesized into a custom hardware. The channels are employed to model the communication between the software block and the hardware block because concurrency and synchronization relationship can be efficiently described in channels in SpecC. In our model, these channels includes two data channels: chdata, cddata, and one control channel: ceobmp. The functionality of these channels is to convey the corresponding data and control information. Behavior OEOBmp, IEOBmp, and channel ceobmp are used to transfer eobmp. Behavior OHData, IHData, and channel chdata are used to transfer hdata. Behavior IDData, ODData, and channel cddata are used to transfer ddata. The SpecC source code is given in Appendix B.

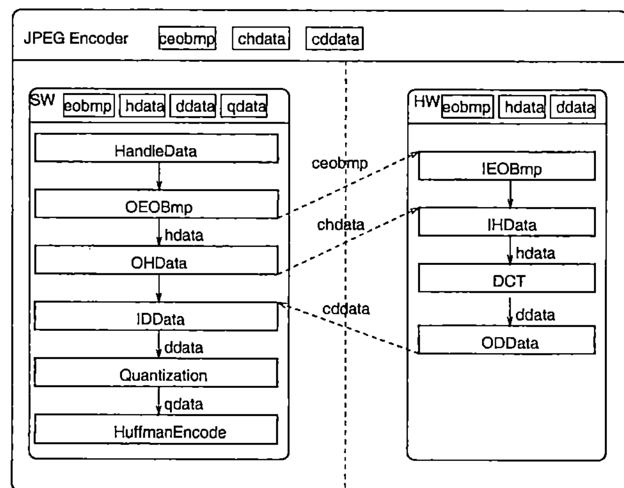

Figure 4: Model after behavior partitioning

4.2 Channel Partition

The goal of the channel partitioning is to group and encapsulate all the channels existing between the two communicating blocks into one bus. The bus is also a type of channel in SpecC, and it implies that the future implementation would be the wired buses. There is some difference between the channel and the bus. That is: channels connect the concurrent behaviors while buses connect the corresponding components into which these behaviors are mapped. Only one bus can exist between two components. The refined architecture model after the channel partitioning is given in Appendix B.

In our model, the channel ceobmp, chdata, and cddata are encapsulated into one global channel Bus. All the implementations of these channels are also encapsulated into this bus. The bus can refer the corresponding channel for the communication between two behavior blocks. The SpecC source code for the bus channel is shown in Appendix B.

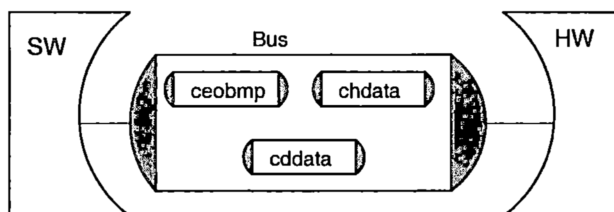

Figure 5: Model after channel partitioning

5 Communication Model

The communication model is the same as the architecture model in that all blocks are mapped to the same components. However, the bus model between the two communicating blocks is timing-accurate but not necessarily to be cycle-accurate. To do this, the time-accurate bus protocols for both of the communicating blocks should be defined. What's more, if these two protocols are not compatible, a transducer should be inserted. Finally, the implementation of the bus communication is inlined into the two communicating components. For the software component, this implementation is represented by an Assembly language I/O procedure which is called a bus driver. For the hardware component, this implementation is a time-accurate behavior description such as a Super FSMD in which each state represents a set of operations that are executed in one or more clock cycles. This time-accurate description will later be refined into a clock-accurate FSMD model.

In our model, the ColdFire protocol is employed for the software part to input/output the data. However, no protocol is defined for the hardware (DCT) part so we need to analyze the DCT datapath to generate the I/O protocol. As we will see, the protocol for the ColdFire and the one for the DCT are not compatible, so a transducer is inserted. The bus protocols for the ColdFire and the DCT as well as the transducer description are described in the following subsections.

5.1 Bus Protocol

5.1.1 ColdFire Master Bus

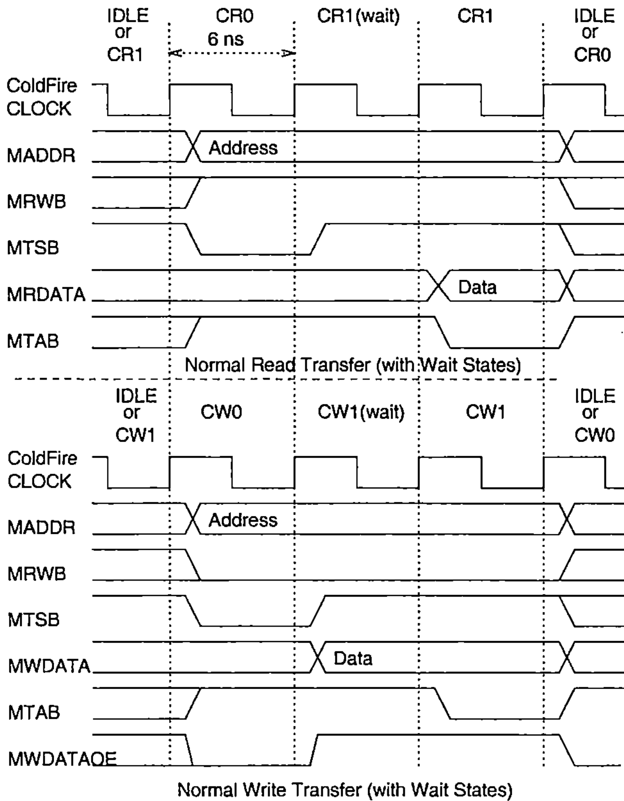

Figure 6: ColdFire master bus timing diagram

The communication between the ColdFire processor and an external device is based on the ColdFire Master Bus protocol. A subset of the master bus signals used in our model are listed below.

Master Address Bus (MADDR[31:0]). If these output signals are used, the memory space of the external device is implicitly mapped into the memory space in the ColdFire processor. So the ColdFire processor can explicitly specify the target address of the external device during the I/O process.

Master Read/Write (MRWB). This output signal indicates a data read/write operation for the current bus communication. A high level '1' indicates a read and a low level '0' indicates a write.

Master Transfer Start (MTSB). This output signal indicates the start of a bus transfer when it is asserted to be '0'.

Master Read/Write Data Bus (MRDATA[31:0] and MWDATA[31:0]). These input/output bus signals provide the datapaths for the data I/O. We can choose to use 8, 16, 32 bits of the data bus per data bus transfer.

Master Transfer Acknowledge (MTAB). This input signal is asserted to '0' by the slave to indicate the successful completion of a bus transfer. It is sampled by the ColdFire Processor at the end of each clock cycle. If it is not asserted, the ColdFire inserts one or more wait states.

Master Write Data Output Enable (MWDATAOE). when asserted to '1', this output signal indicates that the ColdFire is driving the master write data bus. This is used to control optional bidirectional data bus three-state drivers.

Besides these, there is a special bus INTC coming into the ColdFire from outside. Please note that this bus is not in the ColdFire Protocol.

Figure 6 shows the timing diagram for the master bus read (with wait states) and write (with wait states) I/O operations. Figure 7 shows the corresponding FSMD for the ColdFire normal read transfer (with wait states) and Figure 8 shows the ColdFire normal write transfer (with wait state).

The ColdFire master bus protocol is a two-cycle process which include the data transfer start cycle and the data transfer acknowledge cycle. However, one or more wait cycles may be inserted if the acknowledgment of the slave cannot arrive in two ColdFire clock cycles. At first (when there is no I/O request), the ColdFire is in the IDLE state. Once an I/O instruction is decoded and executed, a bus I/O is activated and the ColdFire starts the data transfer. Please note in the usual data transfer, the state before the current data transfer can be an IDLE state or the last state for the previous data transfer. This is shown in the timing diagram Figure 6.

For the ColdFire read I/O, in the first clock cycle (CR0), the address (MADDR) and control information (MTSB, MRWB) are driven onto the bus. In the next cycle (CR1), the address MADDR remains on the bus. The MRWB is still asserted. But MTSB is deasserted to '1'. At this time, one wait states may be inserted if the slave is not ready to send the data. In this case, the slave will keep (desserting) MTAB as '1'. The ColdFire samples the MTAB at the end of the current clock cycle. Because it finds that the MTAB is deasserted ('1'), the ColdFire will insert one wait state (CR1 wait). In other words, the ColdFire will stay one more clock cycle at this state (CR1). In the next cycle (the ColdFire is still at state CR1), if the slave is ready to send the data, it will assert the MTAB to '0'. At the end of this clock cycle, the ColdFire samples the MTAB. It finds that the MTAB is asserted to '0', so at the end of this clock cycle, the data at the data bus

MRDATA is latch into the ColdFire data buffer. If there are more I/O requests suspending, the ColdFire will immediately go back to the state CR0 and start the next data I/O. Otherwise, if there is no more I/O requests, the ColdFire will go to the IDLE state.

For the ColdFire write I/O, in the first clock cycle (CW0), the ColdFire asserts the MTSB to '0', drives the address to address bus MADDR, asserts the MRWB to '0' indicating a write I/O. The MWDATAOE is kept deasserted as '0' indicating that data will be not ready on the MWDATA bus until the next clock cycle. In the next clock cycle (CW1), the data is driven onto the data bus by the ColdFire. Also in this clock cycle, the address remains at the MADDR. The MRWB is still asserted as '0'. But the MTSB is deasserted to '1', and the MWDATAOE is asserted to '1' indicating that data is now on the data bus MWDATA. If the slave is not ready to latch the data, the MTAB will not be asserted, that is, the MTAB is '1' in this clock cycle(CW1). At the end of the current clock cycle (CW1), the ColdFire samples the MTAB, and if the MTAB is still deasserted ('1'), one more wait state (CW1 wait) will be inserted. In the next clock cycle (now the ColdFire is still in the state CW1), if the slave is ready to latch the data (which is not shown in the timing diagram), it will assert the MTAB to '0'. At the end of this clock cycle (CW1), the ColdFire samples the MTAB, and it finds that the MTAB is asserted to '0', so it removes the data from MWDATA and jumps out of the wait state (CW1). If there is more I/O requests suspending, the ColdFire will immediately go to the state CW0 and starts a new data transfer. Otherwise, it will go to the IDLE state.

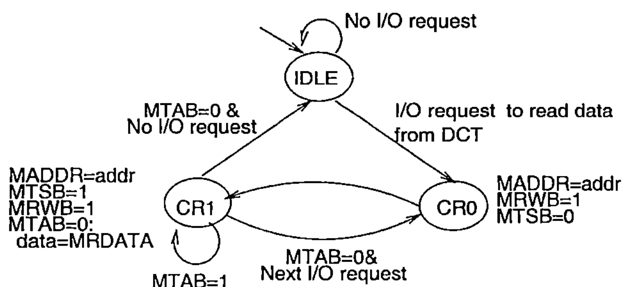

Figure 7: FSMD for the ColdFire normal read transfer

Figure 7 shows the corresponding FSMD for the ColdFire normal read transfer (with wait states).

Figure 8 shows the corresponding FSMD for the ColdFire normal write transfer (with wait states).

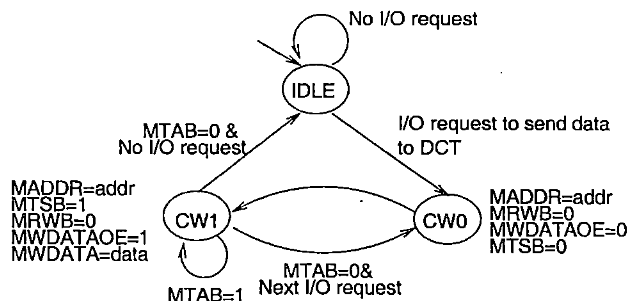

Figure 8: FSMD for the ColdFire normal write transfer

5.1.2 DCT Bus

In our design, the internal structure of the DCT block need to be analyzed to generate the I/O protocol because no protocol is explicitly defined for the DCT. Figure 9 shows the RTL structural diagram of the DCT block.

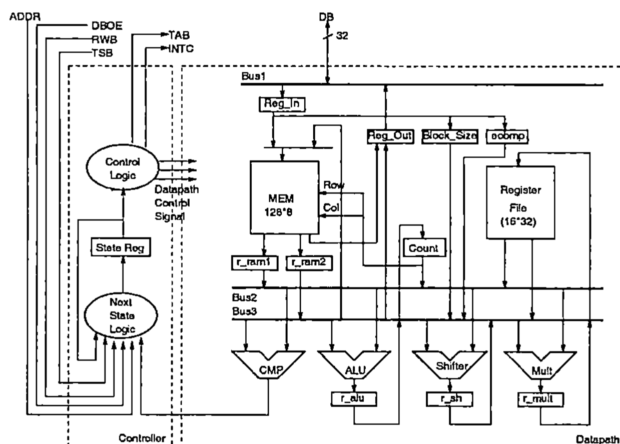

Figure 9: RTL structural diagram of DCT

The bus signals used for the DCT I/O are listed below.

Address bus (ADDR). This input address signal is decoded by the DCT to indicate the corresponding source/destination for data I/O.

Read/Write (RWB). This input signal indicates the data transfer direction in the current bus cycle. A high level indicates a cycle to write to the DCT and a low level indicates a cycle to read from the DCT.

Transfer Start (TSB). This input signal indicates the start of a bus I/O when it is asserted to be '0'.

Data Bus (DB[31:0]). These input/output 32-bit bi-directional signals provide the data paths for the DCT I/O.

Data Bus Data Enable (DBOE). This input signal indicates that the data on the DB[31:0] is ready for latching when it is asserted to be '1'.

Transfer Acknowledge (TAB). This output signal acknowledges the successful completion of the current bus I/O when asserted to be '0'.

Besides these, there is a output interrupt signal interrupt INTC, which is used to interrupt the ColdFire processor. Please note that this signal is not in the DCT bus protocol.

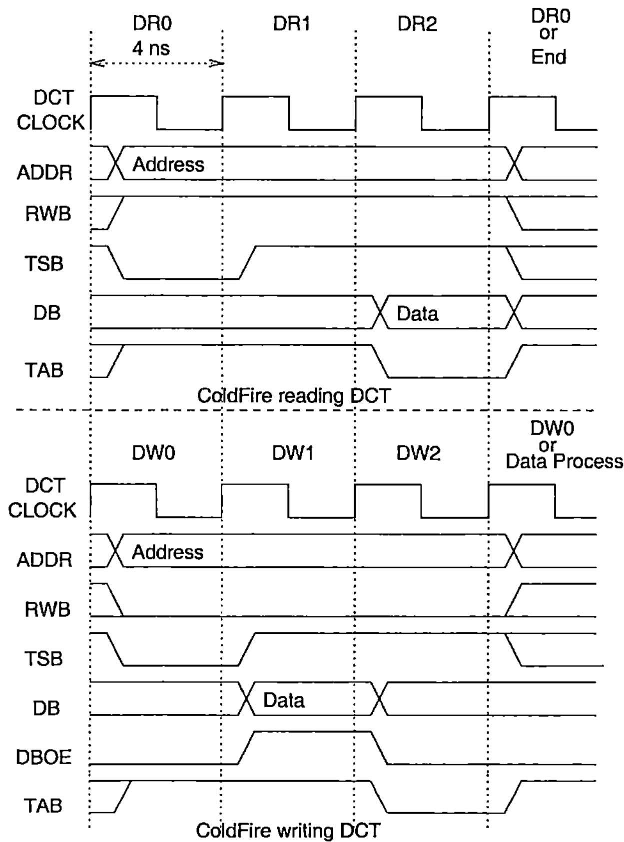

Figure 10: DCT bus operation timing diagram

The DCT clock cycle is assumed to be 4 ns. Figure 10 shows the timing diagram for the DCT bus I/Os. We can see that the DCT bus protocol is a three-cycle process.

When the DCT is read (after the data is processed in the DCT), it will keep TAB deasserted ('1') in the first clock cycle (DR0). The DCT will check the input signal TSB at the end of the first clock cycle (DR0) to see if the TSB is asserted ('0'). If the TSB is not asserted to '0', the DCT will insert a wait state. It means that the DCT will stay at DR0 for one more DCT clock cycle. If the TSB is asserted to be '0', in

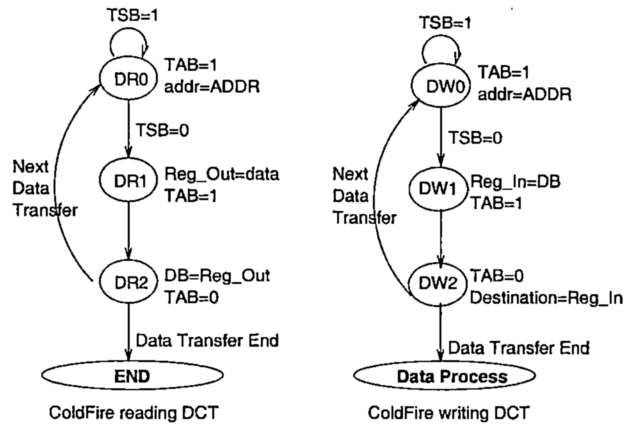

Figure 11: FSMD for DCT bus transfer

the next clock cycle (DR1), the decoded ADDR signals will be used as the memory source address to pull the data out of the MEM into the default output register Reg_Out. In this clock cycle (DR1), the TAB is kept deasserted as '1'. In the third clock cycle (DR2), the data in the Reg_Out is driven onto the DB[31:0]. At the same time (DR2), the TAB is asserted to '0' to acknowledge the successful completion of the current bus I/O.

When the DCT is written, it will keep the TAB deasserted ('1') in the first clock cycle (DW0). The DCT will check the TSB at the end of the first clock cycle (DW0) to see if the TSB is asserted to '0'. If the TSB is not asserted to '0', the DCT will insert a wait state. It means that the DCT will stay at DW0 for one more clock cycle. If the TSB is asserted to '0', the DCT will jump out of the wait state(DW0) in the next clock cycle. The decoded ADDR signals will be accepted as the valid memory destination address, and will be used later in the third clock cycle (DW2). In the clock cycle (DW1) after the DCT jump out of the wait state, the data on the DB[31:0] is latched into the default input register Reg_In. In the third clock cycle (DW2), the data in the Reg_In is sent to the destination. At the same time (DW2), the TAB is asserted to be '0' to acknowledge the successful completion of the current bus I/O. The next state can be either the following data transfer start state (DW0) if the data transfer is not ended, or the data processing start state (Data Processing) if the data transfer is ended.

Figure 11 shows the corresponding FSMD for the DCT bus transfer.

5.2 Transducer Design

The functionality of a transducer includes 1) re-packet the data to the type which can be recognized by the other communicating block. 2) adjust the data arrival and leaving time such that data is safely transfered. So at least, there should be a storage unit in a transducer.

A transducer is inserted into our model since the ColdFire protocol and the DCT protocol are not compatible. The datapath and the corresponding control units are shown in Figure 12. The signals for the communication between the ColdFire and the DCT are shown in this figure.

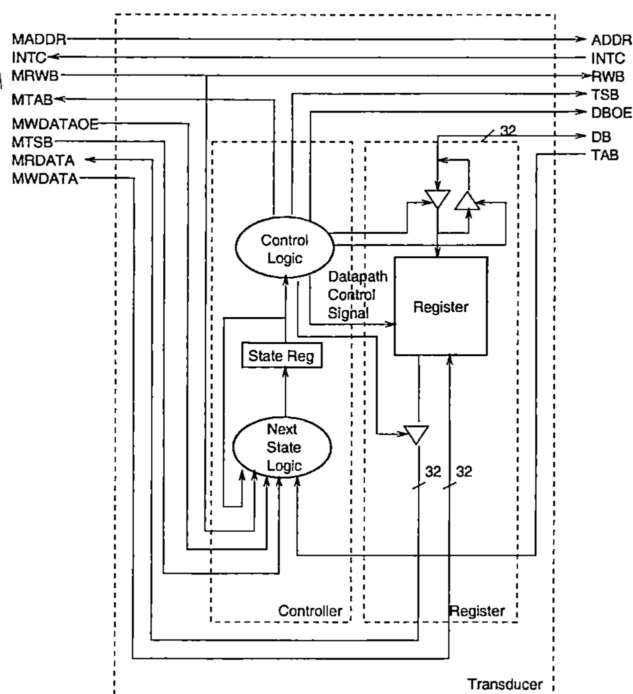

Figure 12: Transducer between ColdFire and DCT

Figure 13 shows the FSMD for the transducer which illustrates the communication process between the ColdFire and the DCT.

The Figure 14 is the corresponding timing diagram for data transfer from the DCT to the ColdFire.

The Figure 15 is the corresponding timing diagram for data transfer from the ColdFire to the DCT. Please note that in this timing diagram, TW2 can be removed if the rising of the MWDATAOE is faster enough such that it is sampled to be '1' at the end of the TW1.

The transducer is always waiting for an data I/O request. Thus In T0, the MTAB is kept '1' for deassertion. The transducer is waiting for the assertion signal of '0' from the MTSB, which indicates the start of a bus I/O. If the MTSB is asserted to be '0', the next

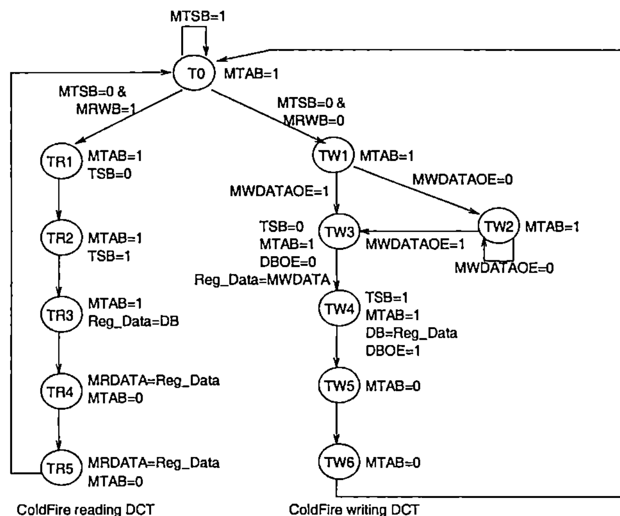

Figure 13: FSMD for the transducer

state might be TR1 or TW1 depending on the value of the MRWB (1: Read, 0: Write). The FSMD for the data transfer from DCT to ColdFire and the one from ColdFire to DCT are described in the following two subsections.

5.2.1 Transducer FSMD for data transfer from DCT to ColdFire

In TR1, the MTAB is not asserted (that is '1'), but the TSB is asserted to be '0', telling the DCT that a new bus I/O begins. In TR2, the DCT loads the out-going (to ColdFire) data into the Reg_Out from the MEM. The MTAB is still not asserted at this state (TR2). During TR3, the DCT drives the data from Reg_Out onto the DCT data bus (DB[31:0]). In TR4, the data is loaded into the Register in the transducer and driven onto the ColdFire's master read data bus (MRDATA[31:0]). At the same time (TR4), the MTAB is asserted to '0'. The data remains on the data bus and the MTAB is still asserted '0' in TR5 because the clock rate of the transducer is faster than that of the ColdFire. After that, the transducer goes back to the wait state (T0) for the next data I/O.

5.2.2 Transducer FSMD for data transfer from ColdFire to DCT

In TW1, the MTAB is not asserted (that is '1'). At the end of TW1, the transducer samples the MWDATAOE and latches the current value on the MWDATA[31:0]. If the MWDATAOE is asserted to '1', the transducer goes to TW3. If MWDATAOE is not

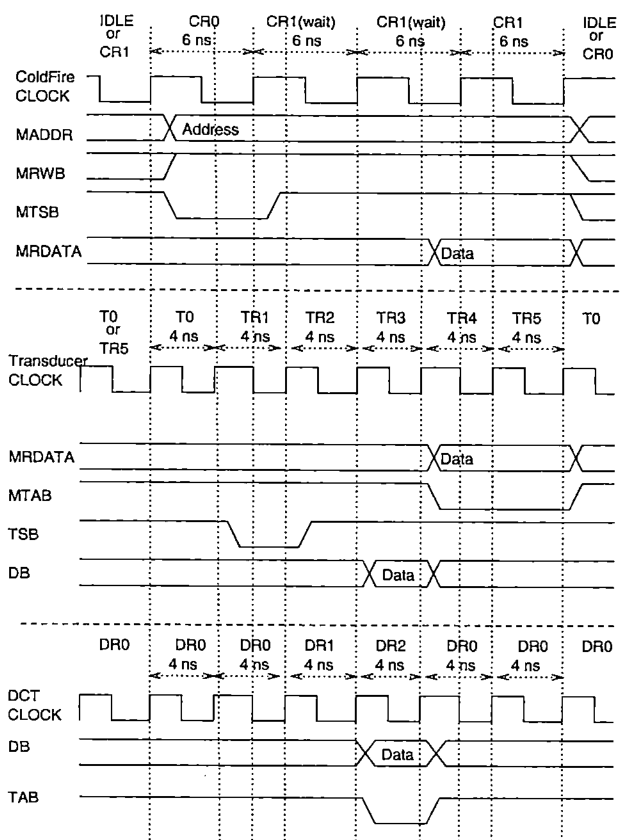

Figure 14: ColdFire reading DCT

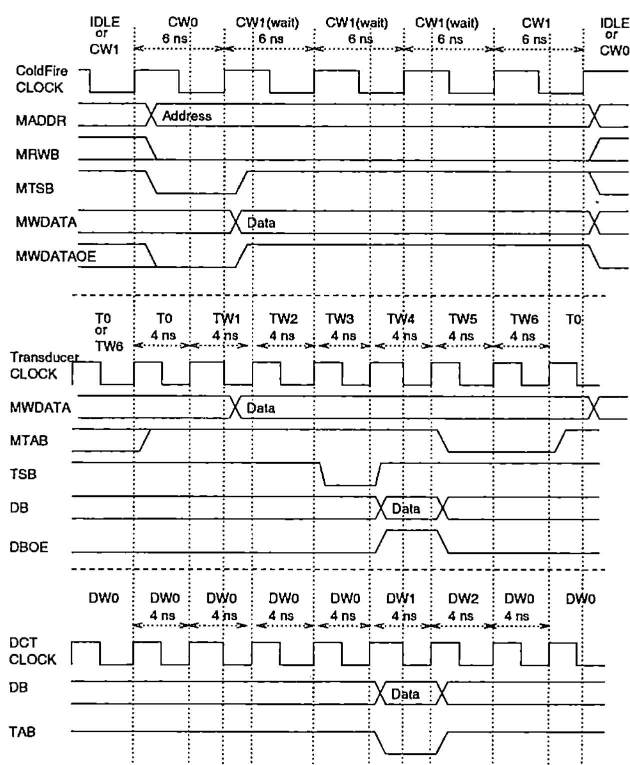

Figure 15: ColdFire writing DCT

asserted (that is '0'), the transducer goes to TW2. In TW2, the transducer keeps sampling the MWDATAOE and latches the current value on the MWDATA[31:0] until the MWDATAOE is asserted to '1', then it goes to TW3. In TW3, the transducer asserts the TSB to '0', telling the DCT that a new bus writing I/O begins. During TW4, the transducer drives the data onto the DCT data bus (DB[31:0]). Concurrently, the DCT asserts the transfer acknowledge (TAB) to '0'. In TW4, the DCT latches data from the data bus into the Reg_In. In TW5, the data is sent to the destination. At the same time, the transducer asserts the MTAB to '0' indicating the ColdFire the completion of this bus I/O. The MTAB remains '0' in TW6 to ensure that the ColdFire samples the correct value of the MTAB.

6 Implementation Model of DCT Block in JPEG Encoder

In our design, the DCT block is the only block which is refined into the Implementation model (RTL model), and which will later be synthesized into a custom hardware.

The implementation model of the DCT block can be globally divided into five functional parts: receive eobmp, receive blocksize, receive hdata, dct process, and send ddata. As is illustrated in Figure 16.

The recieve eobmp, receive blocksize, and receive hdata get data from SW behavior. Since the first two parts only receive one data each, the FSMD for these two parts is the same as the FSMD of ColdFire writing DCT in Figure 11. The receive hdata part gets the 8*8 matrix from SW behavior, and Figure 17 shows the superstate FSMD of the receive hdata part. Here we introduce a count 'i', which is campared with 64(blocksize) each time. RB2 is a super state which represents the FSMD of ColdFire writing DCT in Figure 11.

The dct process part processes the 8*8 matrix which is sent from SW behavior. Figure 18 shows the superstate FSMD of the dct process part. There are three super states: Preshift DCT, ChenDCT, and Bound DCT. Preshift DCT preshifts the 8*8 matrix in the range of -128 to 127. It has five states. ChenDCT implements the Chen forward dct. It has 147 states. Bound DCT bounds the output 8*8 matrix of ChenDCT in the range of -1023 to 1023. It

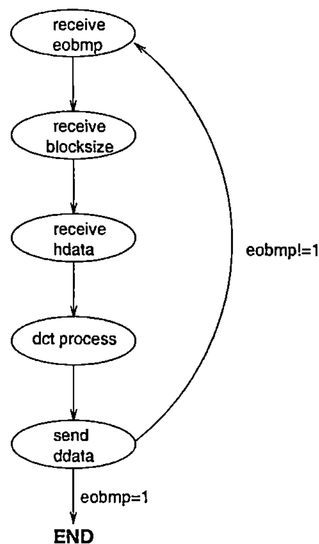

Figure 16: Superstate FSMD of HW behavior

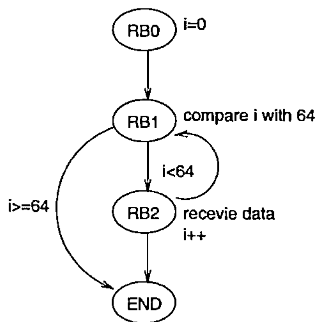

Figure 17: Superstate FSMD of the receive hdata part

has seven states. The SpecC source code is given in Appendix D in two styles of Behavioral RTL (For difference, Please see the Appendix) .

The send ddata part sends the 8*8 output matrix to SW behavior, and Figure 19 shows the superstate FSMD of the send ddata part. Here we introduce a count 'i', which is campared with 64(blocksize) each time. SB2 is a super state which represents the FSMD of ColdFire reading DCT in Figure 11.

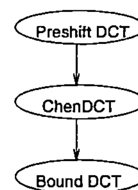

Figure 18: Superstate FSMD of the dct process part

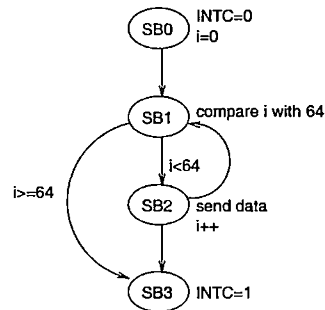

Figure 19: Superstate FSMD of the send ddata part

7 Conclusions

In this report, we apply the SpecC system-level design methodology into the design of a JPEG encoder. Four models are constructed based on this methodology: the specification model, the architecture model, the communication model, and the RTL (behavioral) model. For each model, we demonstrated the mechanism of refining the previous (higher level) model into the current (lower level) model. What's more, during the communication synthesis, we generated the I/O protocol for the DCT hardware block, and derived a traducer for the communication of the ColdFire and the DCT. The timing diagram shows that these two protocols match well with the translation of the traducer. We verified our SpecC models by running the testbench we mentioned in the specification model. The correct output demonstrates the correctness of our models

References

[1] [DCT] V. Bhaskaran, K. Konstantinides, *Image and Video Compression Standards*, Second Edition, Kluwer Academic Publishers, 1997.

[2] [GZDGZ] D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, S. Zhao, *SpecC: Specification Language and Methodology*, Kluwer Academic Publishers, March, 2000.

[3] [CPCG] L. Cai, J. Peng, C. Chang, A. Gerstlauer, H. Li, A. Selka, C. Siska, L. Sun, S. Zhao, D. Gajski, *Design of a JPEG Encoding System*, University of California, Irvine, Technical Report ICS-99-xx, September 1999.

[4] [DZG] R. Dömer, J. Zhu, D. Gajski, *The SpecC Language Reference Manual*, University of Cal-

ifornia, Irvine, Technical Report ICS-TR-99-11, February 1999.

[5] [Gerstlauer]A. Gerstlauer, *Architecture Exploration*, University of California, Irvine, Technical Report ICS-00-xx, May 2000

[6] [ColdFire] Motorola, Inc., Semiconductors Products Sector, DSP Division, *ColdFire 2/2M Intergrated Micro-processor User's Manual, 1998*

A Specification Model

This section contains the SpecC code for the specification model of the JPEG encoder.

A.1 JPEG Encoder

A.1.1 const.sc

```
   /************     JPEG encode      *******/
   /************     const.sh         *******/
   /************     Hanyu Yin        *******/
   /************     05/11/2000       *******/
 5
   #ifndef __CONST_
   #define __CONST_

   #define          BLOCKSIZE        64
10
   /***********************************************************************
   ** JPEG output stream
   ***********************************************************************/
   #define          M_APP0           0xe0
15 #define          M_COM            0xfe
   #define          M_DAC            0xcc
   #define          M_DHP            0xde
   #define          M_DHT            0xc4
   #define          M_DNL            0xdc
20 #define          M_DQT            0xdb
   #define          M_DRI            0xdd
   #define          M_EOI            0xd9
   #define          M_EXP            0xdf
   #define          M_JPG            0xc8
25 #define          M_RST0           0xd0
   #define          M_SOI            0xd8
   #define          M_SOS            0xda
   #define          M_SOF0           0xc0

30 #endif
```

A.1.2 global.sc

```
   /************     Global Function  for JPEG encode          *******/
   /************     global.sc        *******/
   /************     Hanyu Yin        *******/
   /************     05/11/2000       *******/
 5
   #ifndef __GLOBAL_
   #define __GLOBAL_

   #include <stdio.h>
10 #include <stdlib.h>

   import "chann";

   #include "const.sh"
15
   #define          XHUFF            struct huffman_standard_structure
   #define          EHUFF            struct huffman_encoder

   XHUFF {
20          int bits[36];
            int huffval[257];
   };

   EHUFF {
```

```
25          int ehufco[257];
            int ehufsi[257];
   };

   static int QuantizationMatrix[] = {
30 16, 11, 10, 16, 24, 40, 51, 61,
   12, 12, 14, 19, 26, 58, 60, 55,
   14, 13, 16, 24, 40, 57, 69, 56,
   14, 17, 22, 29, 51, 87, 80, 62,
   18, 22, 37, 56, 68, 109, 103, 77,
35 24, 35, 55, 64, 81, 104, 113, 92,
   49, 64, 78, 87, 103, 121, 120, 101,
   72, 92, 95, 98, 112, 100, 103, 99};

   int IZigzagIndex[] =
40          {0,  1,  8, 16,  9,  2,  3, 10,
            17, 24, 32, 25, 18, 11,  4,  5,
            12, 19, 26, 33, 40, 48, 41, 34,
            27, 20, 13,  6,  7, 14, 21, 28,
            35, 42, 49, 56, 57, 50, 43, 36,
45          29, 22, 15, 23, 30, 37, 44, 51,
            58, 59, 52, 45, 38, 31, 39, 46,
            53, 60, 61, 54, 47, 55, 62, 63};

   static XHUFF *Xhuff=0;
50 static EHUFF *Ehuff=0;

   static XHUFF ACXhuff, DCXhuff;
   static EHUFF ACEhuff, DCEhuff;

55 int LastDC;

   // Error messages
   void error(const char *Format, const char* Name)
   {
60   fprintf(stderr, Format, Name);
     exit(1);
   }


65 FILE* openStdout()
   {
     return stdout;
   }

70
   /**********************************************************************
   **              I/O routines
   **********************************************************************/

75 int WriteWord(int code, iBlckSendByte Data_Ch)
   {
            Data_Ch.send((char)(code>>8));
            Data_Ch.send((char)(code & 0xff));
            return 2;
80 }

   int WriteByte(int code, iBlckSendByte Data_Ch)
   {
            Data_Ch.send((char)code);
85          return 0;
   }

   int WriteBits(int n, int code, iBlckSendByte Data_Ch)
   {
90          static unsigned char write_byte = 0;
```

```
        static left_bits = 8;
        int p;

        unsigned lmask[] = {
95              0x0000,
                0x0001, 0x0003, 0x0007, 0x000f,
                0x001f, 0x003f, 0x007f, 0x00ff,
                0x01ff, 0x03ff, 0x07ff, 0x0fff,
                0x1fff, 0x3fff, 0x7fff, 0xffff
100     };

        // synchronize buffer value
        if (n < 0) {
                if (left_bits < 8) {
105                     n = left_bits;
                        Data_Ch.send(write_byte);
                        if (write_byte == 0xff) {
                                Data_Ch.send(0);
                        }
110                     write_byte = 0;
                        left_bits = 8;
                }
                else    n = 0;

115             return n;
        }

        code &= lmask[n];
        p = n - left_bits;
120
        if (n == left_bits) {
                write_byte |= code;
                Data_Ch.send(write_byte);
                if (write_byte == 0xff) {
125                     Data_Ch.send(0);
                }

                write_byte = 0;
                left_bits = 8;
130     }
        else if (n > left_bits) {
                write_byte |= (code >> p);
                Data_Ch.send(write_byte);
                if (write_byte == 0xff) {
135                     Data_Ch.send(0);
                }

                if (p > 8) {
                        write_byte = (0xff & (code >> (p - 8)));
140                     Data_Ch.send(write_byte);
                        if (write_byte == 0xff) {
                                Data_Ch.send(0);
                        }
                        p -= 8;
145             }

                write_byte = (code & lmask[p]) << (8 - p);
                left_bits = 8 - p;
        }
150     else {
                write_byte |= (code << -p);
                left_bits -= n;
        }

155     return n;
    }
```

```
    void UseDCHuffman()
    {
160   Xhuff = &DCXhuff;
      Ehuff = &DCEhuff;
    }

    void UseACHuffman()
165 {
      Xhuff = &ACXhuff;
      Ehuff = &ACEhuff;
    }

170 #endif
```

A.1.3 chann.sc

```
    #ifndef __CHANNEL_
    #define __CHANNEL_

    typedef char     BYTE ;
5
    interface iBlckSendByte {
            void send(BYTE val);
    };

10  interface iBlckRecvByte {
            BYTE receive(void);
    };

    interface iBlckSendInt {
15          void send(int val);
    };

    interface iBlckRecvInt {
            int receive(void);
20  };

    interface iBlckSendBlock {
            void send(int val[64]);
    };
25
    interface iBlckRecvBlock {
            void receive(int val[64]);
    };

30  channel cSyncByte(void) implements iBlckSendByte, iBlckRecvByte
    {
    BYTE message;             // temporary buffer
    bool valid=false;         // semaphore
    event sent, received;     // semaphore event
35
    // blocking send
    void send(BYTE val){
            message = val;
            valid=true;
40          notify(sent);
            if(valid)
                    wait(received);
    }

45  // blocking receive
    BYTE receive(void){
            BYTE local_message;

            if(!valid)
50                  wait(sent);
```

```
        local_message = message;
        valid=false;
        notify(received);
        return local_message;
55 }
   };

   channel cSyncInt(void) implements iBlckSendInt, iBlckRecvInt
   {
60 int message;               // temporary buffer
   bool valid=false;          // semaphore
   event sent, received;      // semaphore event

   //blocking send
65 void send(int val){
        message = val;
        valid=true;
        notify(sent);
        if(valid)
70              wait(received);
   }

   //blocking receive
   int receive(void){
75      int local_message;

        if(!valid)
                wait(sent);
        local_message = message;
80      valid=false;
        notify(received);
        return local_message;
   }
   };
85
   channel cSyncBlock(void) implements iBlckSendBlock, iBlckRecvBlock
   {
   int i;                     // temporary count
   int message[64];           // temporary buffer
90 bool valid=false;          // semaphore
   event sent, received;      // semaphore event

   // blocking send
   void send(int val[64]){
95      for(i=0; i<64; i++)
                message[i] = val[i];
        valid=true;
        notify(sent);
        if(valid)
100             wait(received);
   }

   // blocking receive
   void receive(int val[64]){
105     if(!valid)
                wait(sent);
        for(i=0; i<64; i++)
                val[i] = message[i];
        valid=false;
110     notify(received);
   }
   };
   #endif
```

A.1.4 handle.sc

```
/************   JPEG encode    *******/
```

```
/************   Handledata.sc   *******/
/************   Hanyu Yin       *******/
/************   05/11/2000      *******/

#ifndef _HANDLE_
#define _HANDLE_

import "global";

#include "const.sh"

/* Default huffman table */
static int LuminanceDCBits[] = {
0x00, 0x01, 0x05, 0x01, 0x01, 0x01, 0x01, 0x01,
0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};

static int LuminanceDCValues[] = {
0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b};

static int LuminanceACBits[] = {
0x00, 0x02, 0x01, 0x03, 0x03, 0x02, 0x04, 0x03,
0x05, 0x05, 0x04, 0x04, 0x00, 0x00, 0x01, 0x7d};

static int LuminanceACValues[] = {
0x01, 0x02, 0x03, 0x00, 0x04, 0x11, 0x05, 0x12,
0x21, 0x31, 0x41, 0x06, 0x13, 0x51, 0x61, 0x07,
0x22, 0x71, 0x14, 0x32, 0x81, 0x91, 0xa1, 0x08,
0x23, 0x42, 0xb1, 0xc1, 0x15, 0x52, 0xd1, 0xf0,
0x24, 0x33, 0x62, 0x72, 0x82, 0x09, 0x0a, 0x16,
0x17, 0x18, 0x19, 0x1a, 0x25, 0x26, 0x27, 0x28,
0x29, 0x2a, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39,
0x3a, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49,
0x4a, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59,
0x5a, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69,
0x6a, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79,
0x7a, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89,
0x8a, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, 0x98,
0x99, 0x9a, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7,
0xa8, 0xa9, 0xaa, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6,
0xb7, 0xb8, 0xb9, 0xba, 0xc2, 0xc3, 0xc4, 0xc5,
0xc6, 0xc7, 0xc8, 0xc9, 0xca, 0xd2, 0xd3, 0xd4,
0xd5, 0xd6, 0xd7, 0xd8, 0xd9, 0xda, 0xe1, 0xe2,
0xe3, 0xe4, 0xe5, 0xe6, 0xe7, 0xe8, 0xe9, 0xea,
0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, 0xf8,
0xf9, 0xfa};

static int huffcode[257];
static int huffsize[257];
static int lastp = 0;

void WriteHuffman(iBlckSendByte Data_Ch)
{
  int i,accum;

  if (Xhuff)
    {
      for(accum=0,i=1;i<=16;i++)
        {
          WriteByte(Xhuff->bits[i], Data_Ch);
          accum += Xhuff->bits[i];
        }
      for(i=0;i<accum;i++)
        {
          WriteByte(Xhuff->huffval[i], Data_Ch);
        }
    }
```

```
    else
      {
70      printf("Null␣Huffman␣table␣found.\n");
      }
  }

  void WriteMarker(int m, iBlckSendByte Data_Ch)
75 {
        Data_Ch.send(0xff);
        Data_Ch.send((char)m);
  }

80 void WriteAPP0(iBlckSendByte Data_Ch)
  {
        WriteMarker(M_APP0, Data_Ch);

        /* length */
85      WriteWord(16, Data_Ch);

        /* identifier */
        WriteByte('J', Data_Ch);
        WriteByte('F', Data_Ch);
90      WriteByte('I', Data_Ch);
        WriteByte('F', Data_Ch);
        WriteByte(0, Data_Ch);

        /* version */
95      WriteWord(0x0102, Data_Ch);

        /* units */
        WriteByte(2, Data_Ch);

100     /* Xdensity */
        WriteWord(0x001d, Data_Ch);

        /* Ydensity */
        WriteWord(0x001d, Data_Ch);
105
        /* Xthumbnail, Ythumbnail */
        WriteWord(0x0000, Data_Ch);
  }

110 void WriteSOF(int ImageHeight, int ImageWidth, iBlckSendByte Data_Ch)
  {

        WriteMarker(M_SOF0, Data_Ch);

115     /* Lf: frame header length */
        WriteWord(11, Data_Ch);

        /* P: precision */
        WriteByte(8, Data_Ch);
120
        /* Y: # lines, X: # samples/line */
        WriteWord(ImageHeight, Data_Ch);
        WriteWord(ImageWidth, Data_Ch);

125     /* Nf: number of component in frame */
        WriteByte(1, Data_Ch);

        /* Frame component specification */
        /* Ci: component identifier */
130     WriteByte(1, Data_Ch);
        /* Hi: horizontal sampling factor, Vi: vertical sampling factor */
        WriteByte(0x11, Data_Ch);
        /* Tqi: quantization table destination selector */
```

```
        WriteByte(0, Data_Ch);
135 }

    void WriteDQT(iBlckSendByte Data_Ch)
    {
            int i;
140
            WriteMarker(M_DQT, Data_Ch);

            /* Lq */
            WriteWord(67, Data_Ch);
145
            /* define each quantization table */
            WriteByte(0, Data_Ch);

            for (i=0; i<64; i++) {
150                 WriteByte(QuantizationMatrix[IZigzagIndex[i]], Data_Ch);
            }
    }

    void WriteDHT(iBlckSendByte Data_Ch)
155 {
            WriteMarker(M_DHT, Data_Ch);

            WriteWord(0x4+0x20+0xc+0xa2, Data_Ch);

160         /* Write DC Huffman */
            /* Tc, Th */
            WriteByte(0, Data_Ch);
            UseDCHuffman();
            WriteHuffman(Data_Ch);
165
            /* Write AC Huffman */
            /* Tc, Th */
            WriteByte(0x10, Data_Ch);
            UseACHuffman();
170         WriteHuffman(Data_Ch);
    }


    void WriteSOS(iBlckSendByte Data_Ch)
175 {
            WriteMarker(M_SOS, Data_Ch);

            /* Ls: scan header length */
            WriteWord(8, Data_Ch);
180
            /* Ns: number of components in scan */
            WriteByte(1, Data_Ch);

            /* Csk: scan component selector */
185         WriteByte(1, Data_Ch);

            /* Tdk, Tak: DC/AC entropy coding table selector */
            WriteByte(0, Data_Ch);

190         /* Ss: start of spectral selection or predictor selection */
            WriteByte(0, Data_Ch);

            /* Se: end of spectral selection */
            WriteByte(63, Data_Ch);
195
            /* Ah, Al: successssive approximation bit position high/low */
            WriteByte(0, Data_Ch);
    }
```

```
200
    static void SizeTable()
    {
      int i,j,p;

205   for(p=0,i=1;i<17;i++)
        {
          for(j=1;j<=Xhuff->bits[i];j++)
            {
              huffsize[p++] = i;
210         }
        }
      huffsize[p] = 0;
      lastp = p;
    }
215
    static void CodeTable()
    {
      int p,code,size;

220   p=0;
      code=0;
      size = huffsize[0];
      while(1)
        {
225       do
            {
              huffcode[p++] = code++;
            }
          while((huffsize[p]==size)&&(p<257)); /* Overflow Detection */
230       if (!huffsize[p]) /* All finished. */
            {
              break;
            }
          do                  /* Shift next code to expand prefix. */
235         {
              code <<= 1;
              size++;
            }
          while(huffsize[p] != size);
240     }
    }

    static void OrderCodes()
    {
245   int index,p;

      for(p=0;p<lastp;p++)
        {
          index = Xhuff->huffval[p];
250       Ehuff->ehufco[index] = huffcode[p];
          Ehuff->ehufsi[index] = huffsize[p];
        }
    }

255 void SpecifiedHuffman(int *bts,int *hvls)
    {
            int i;
            int accum;

260         for(accum=0,i=0;i<16;i++)
            {
                    accum+= bts[i];
                    Xhuff->bits[i+1] = bts[i];      /* Shift offset for internal specs.*/
            }
265         for(i=0;i<accum;i++)
```

```
        {
                Xhuff->huffval[i] = hvls[i];
        }
        SizeTable();                              /*From Xhuff to Ehuff */
270     CodeTable();
        OrderCodes();
    }

    void JpegDefaultHuffman()
275 {
        UseDCHuffman();
        SpecifiedHuffman(LuminanceDCBits, LuminanceDCValues);
        UseACHuffman();
        SpecifiedHuffman(LuminanceACBits, LuminanceACValues);
280 }

    int MDUWide, MDUHigh;
    int mduWide=0, mduHigh=0;
    FILE *fptr;
285 unsigned char *stripe;

    behavior HandleData(iBlckRecvInt Header_Ch, iBlckRecvByte Pixel_Ch,
            iBlckSendByte Data_Ch, out int HData[64], out int eobmp)
    {
290         unsigned char temp;
            int i, j;
            int ImageHeight, ImageWidth;

        void main(void) {
295
          if ((mduWide==0)&&(mduHigh==0))
          {
            LastDC = 0;

300         ImageWidth = Header_Ch.receive();
            ImageHeight = Header_Ch.receive();

            MDUWide = (ImageWidth+7) >> 3;
            MDUHigh = (ImageHeight+7) >> 3;
305
            JpegDefaultHuffman();

            WriteMarker(M_SOI, Data_Ch);
            WriteAPP0(Data_Ch);
310         WriteSOF(ImageHeight, ImageWidth,  Data_Ch);
            WriteDQT(Data_Ch);
            WriteDHT(Data_Ch);
            WriteSOS(Data_Ch);

315         stripe = (unsigned char *) calloc(64*MDUWide, sizeof(char));
            fptr=fopen("data.txt", "w");
          }

          if ((mduWide==0)&&(mduHigh<MDUHigh))
320       {
            // Read a stripe from testbench
            for (i=0; i<8; i++) {
                    for (j=0; j<MDUWide*8; j++) {
                            if ((j<ImageWidth) && (mduHigh*8+i<ImageHeight)) {
325                                 temp = Pixel_Ch.receive();
                                    stripe[i*MDUWide*8+j] = temp;
                            }
                            else {
                                    stripe[i*MDUWide*8+j]
330                                 = stripe[i*MDUWide*8+ImageWidth-1];
                            }
```

```
                        }
                }
335             for (i=0; i<64*MDUWide; i++)
                {
                        fprintf(fptr, "%d ", stripe[i]);
                        fprintf(fptr,"\n");
                }
340             mduHigh++;
        }

        if(mduWide<MDUWide)
        {
345             // fetch a block and send it to DCT
                for(i=0; i<8; i++) {
                        for(j=0; j<8; j++) {
                                HData[i*8+j] = stripe[i*MDUWide*8+mduWide*8+j];
                        }
350             }
                mduWide++;
        }

        if((mduWide==MDUWide)&&(mduHigh==MDUHigh))
355     {
                eobmp = 1;
                fclose(fptr);
        }

360     if(mduWide==MDUWide)
        {
                mduWide = 0;
        }
    }
365 };

#endif
```

A.1.5 dct.sc

```
/************  JPEG encode  *******/
/************  DCT.sc       *******/
/************  Hanyu Yin    *******/
/************  05/11/2000   *******/
5
#ifndef __DCT_
#define __DCT_

#include "const.sh"
10
import "global";
import "chann";

/*************************************************************************
15 * DCT
*************************************************************************/

#define NO_MULTIPLY

20 #ifdef NO_MULTIPLY
#define LS(r,s) ((r) << (s))
#define RS(r,s) ((r) >> (s))       /* Caution with rounding... */
#else
#define LS(r,s) ((r) * (1 << (s)))
25 #define RS(r,s) ((r) / (1 << (s))) /* Correct rounding */
#endif

/* Cos constants */
```

```
   #define c1d4 362L
30 #define c1d8 473L
   #define c3d8 196L
   #define c1d16 502L
   #define c3d16 426L
   #define c5d16 284L
35 #define c7d16 100L
   #define MSCALE(expr)  RS((expr),9)

   /*
     VECTOR_DEFINITION makes the temporary variables vectors.
40   Useful for machines with small register spaces.
   */
   #ifdef VECTOR_DEFINITION
   #define a0 a[0]
   #define a1 a[1]
45 #define a2 a[2]
   #define a3 a[3]
   #define b0 b[0]
   #define b1 b[1]
   #define b2 b[2]
50 #define b3 b[3]
   #define c0 c[0]
   #define c1 c[1]
   #define c2 c[2]
   #define c3 c[3]
55 #endif

   /*START*/
   /*BFUNC
   ChenDCT() implements the Chen forward dct. Note that there are two
60 input vectors that represent x=input, and y=output, and must be
   defined (and storage allocated) before this routine is called
   EFUNC*/

   void ChenDct(int *x,int *y)
65 {
     register int i;
     register int *aptr,*bptr;
   #ifdef VECTOR_DEFINITION
     register int a[4];
70   register int b[4];
     register int c[4];
   #else
     register int a0,a1,a2,a3;
     register int b0,b1,b2,b3;
75   register int c0,c1,c2,c3;
   #endif

     /* Loop over columns */

80   for(i=0;i<8;i++)
       {
         aptr = x+i;
         bptr = aptr+56;

85       a0 = LS((*aptr+*bptr),2);
         c3 = LS((*aptr-*bptr),2);
         aptr += 8;
         bptr -= 8;
         a1 = LS((*aptr+*bptr),2);
90       c2 = LS((*aptr-*bptr),2);
         aptr += 8;
         bptr -= 8;
         a2 = LS((*aptr+*bptr),2);
         c1 = LS((*aptr-*bptr),2);
```

```
95        aptr += 8;
          bptr -= 8;
          a3 = LS((*aptr+*bptr),2);
          c0 = LS((*aptr-*bptr),2);

100       b0 = a0+a3;
          b1 = a1+a2;
          b2 = a1-a2;
          b3 = a0-a3;

105       aptr = y+i;

          *aptr = MSCALE(c1d4*(b0+b1));
          aptr[32] = MSCALE(c1d4*(b0-b1));

110       aptr[16] = MSCALE((c3d8*b2)+(c1d8*b3));
          aptr[48] = MSCALE((c3d8*b3)-(c1d8*b2));

          b0 = MSCALE(c1d4*(c2-c1));
          b1 = MSCALE(c1d4*(c2+c1));
115
          a0 = c0+b0;
          a1 = c0-b0;
          a2 = c3-b1;
          a3 = c3+b1;
120
          aptr[8] = MSCALE((c7d16*a0)+(c1d16*a3));
          aptr[24] = MSCALE((c3d16*a2)-(c5d16*a1));
          aptr[40] = MSCALE((c3d16*a1)+(c5d16*a2));
          aptr[56] = MSCALE((c7d16*a3)-(c1d16*a0));
125     }

      for(i=0;i<8;i++)
        {         /* Loop over rows */
          aptr = y+LS(i,3);
130       bptr = aptr+7;

          c3 = RS((*(aptr)-*(bptr)),1);
          a0 = RS((*(aptr++)+*(bptr--)),1);
          c2 = RS((*(aptr)-*(bptr)),1);
135       a1 = RS((*(aptr++)+*(bptr--)),1);
          c1 = RS((*(aptr)-*(bptr)),1);
          a2 = RS((*(aptr++)+*(bptr--)),1);
          c0 = RS((*(aptr)-*(bptr)),1);
          a3 = RS((*(aptr)+*(bptr)),1);
140
          b0 = a0+a3;
          b1 = a1+a2;
          b2 = a1-a2;
          b3 = a0-a3;
145
          aptr = y+LS(i,3);

          *aptr = MSCALE(c1d4*(b0+b1));
          aptr[4] = MSCALE(c1d4*(b0-b1));
150       aptr[2] = MSCALE((c3d8*b2)+(c1d8*b3));
          aptr[6] = MSCALE((c3d8*b3)-(c1d8*b2));

          b0 = MSCALE(c1d4*(c2-c1));
          b1 = MSCALE(c1d4*(c2+c1));
155
          a0 = c0+b0;
          a1 = c0-b0;
          a2 = c3-b1;
          a3 = c3+b1;
160
```

```
        aptr[1] = MSCALE((c7d16*a0)+(c1d16*a3));
        aptr[3] = MSCALE((c3d16*a2)-(c5d16*a1));
        aptr[5] = MSCALE((c3d16*a1)+(c5d16*a2));
        aptr[7] = MSCALE((c7d16*a3)-(c1d16*a0));
165     }

      /* We have an additional factor of 8 in the Chen algorithm. */

      for(i=0, aptr=y; i<64; i++,aptr++)
170     *aptr = (((*aptr<0) ? (*aptr-4) : (*aptr+4))/8);
    }

    void PreshiftDctMatrix(int *matrix, int shift)
    {
175         int i;
            for(i=0; i<BLOCKSIZE; i++)
                    matrix[i] -= shift;
    }

180 void BoundDctMatrix(int *matrix, int bound)
    {
            int i;
            for (i=0; i<BLOCKSIZE; i++) {
                    if (matrix[i] < -bound)             matrix[i] = -bound;
185                 else if (matrix[i] > bound)         matrix[i] = bound;
            }
    }

    behavior DCT(in int HData[64], out int DData[64])
190 {
            int DCTBound=1023;
            int DCTShift=128;

        void main(void) {
195         PreshiftDctMatrix(HData, DCTShift);
            ChenDct(HData, DData);
            BoundDctMatrix(DData, DCTBound);
        }
    };
200
    #endif
```

A.1.6 quant.sc

```
    /************   JPEG encode        *******/
    /************   quantization.sc   *******/
    /************   Hanyu Yin          *******/
    /************   05/11/2000         *******/
 5
    #ifndef _QUANT_
    #define _QUANT_

    #include "const.sh"
10
    import "global";

    void Quantize(int *matrix, int *qmatrix)
    {
15          int i, m, q;
            for (i=0; i<BLOCKSIZE; i++) {
                    m = matrix[i];
                    q = qmatrix[i];
                    if (m > 0) {
20                          matrix[i] = (m + q/2) / q;
                    }
                    else {
                            matrix[i] = (m - q/2) / q;
```

```
         2,  4,  7, 13, 16, 26, 29, 42,
         3,  8, 12, 17, 25, 30, 41, 43,
         9, 11, 18, 24, 31, 40, 44, 53,
        10, 19, 23, 32, 39, 45, 52, 54,
50      20, 22, 33, 38, 46, 51, 55, 60,
        21, 34, 37, 47, 50, 56, 59, 61,
        35, 36, 48, 49, 57, 58, 62, 63};

void PrintTable(int *table)
55 {
  int i,j;

  for(i=0;i<16;i++)
    {
60    for(j=0;j<16;j++)
        {
          printf("%2x ",*(table++));
        }
      printf("\n");
65  }
}

void PrintHuffman()
{
70  int i;

  if (Xhuff)
    {
      printf("Bits: [length:number]\n");
75    for(i=1;i<9;i++)
        {
          printf("[%d:%d]",i,Xhuff->bits[i]);
        }
      printf("\n");
80    for(i=9;i<17;i++)
        {
          printf("[%d:%d]",i,Xhuff->bits[i]);
        }
      printf("\n");
85
      printf("Huffval:\n");
      PrintTable(Xhuff->huffval);
    }
  if (Ehuff)
90  {
      printf("Ehufco:\n");
      PrintTable(Ehuff->ehufco);
      printf("Ehufsi:\n");
      PrintTable(Ehuff->ehufsi);
95  }
}

void EncodeHuffman(int value, iBlckSendByte Data_Ch)
{
100     if (Ehuff->ehufsi[value]) {
                WriteBits(Ehuff->ehufsi[value], Ehuff->ehufco[value], Data_Ch);
        }
        else {
                printf("Null Code for [%d] Encountered:\n", value);
105             printf("*** Dumping Huffman Table ***\n");
                PrintHuffman();
                printf("***\n");
                exit(-1);
        }
110 }
```

```
    void ZigzagMatrix(int *imatrix, int *omatrix)
    {
            int i, z;
115
            for (i=0; i<BLOCKSIZE; i++) {
                    z = ZigzagIndex[i];
                    omatrix[z] = imatrix[i];
            }
120 }




125 void EncodeDC(int coef, iBlckSendByte Data_Ch)
    {
            int s, diff, cofac;

            UseDCHuffman();
130
            diff = coef - LastDC;
            LastDC = coef;
            cofac = abs(diff);
            if (cofac < 256) {
135                 s = csize[cofac];
            }
            else {
                    cofac = cofac >> 8;
                    s = csize[cofac] + 8;
140         }

            EncodeHuffman(s, Data_Ch);
            if (diff < 0) {
                    diff--;
145         }

            WriteBits(s, diff, Data_Ch);
    }

150 void EncodeAC(int *matrix, iBlckSendByte Data_Ch)
    {
            int i, k, r, ssss, cofac;

            UseACHuffman();
155
            for (k=r=0; ++k<BLOCKSIZE; ) {
                    cofac = abs(matrix[k]);
                    if (cofac < 256)        ssss = csize[cofac];
                    else {
160                         cofac = cofac >> 8;
                            ssss = csize[cofac] + 8;
                    }

                    if (matrix[k] == 0) {
165                         if (k == BLOCKSIZE-1) {
                                    EncodeHuffman(0, Data_Ch);
                                    break;
                            }
                            r++;
170                 }
                    else {
                            while (r > 15) {
                                    EncodeHuffman(240, Data_Ch);
                                    r -= 16;
175                         }
                            i = 16 * r + ssss;
                            r = 0;
```

```
                                EncodeHuffman(i, Data_Ch);
                                if (matrix[k] < 0) WriteBits(ssss, matrix[k]-1, Data_Ch);
180                             else WriteBits(ssss, matrix[k], Data_Ch);
                        }
                }
        }

185 behavior HuffmanEncode(in int QData[64], iBlckSendByte Data_Ch)
    {
        int output[BLOCKSIZE];

        void main(void) {
190         ZigzagMatrix(QData, output);
            EncodeDC(output[0], Data_Ch);
            EncodeAC(output, Data_Ch);
        }
    };
195
#endif
```

A.1.8 jpeg.sc

```
import "handle";
import "dct";
import "quant";
import "huff";
5
#include "const.sh"

behavior Jpeg(iBlckRecvInt header_ch, iBlckRecvByte pixel_ch,
        iBlckSendByte data_ch)
10 {
        int      block_no ;
        int      hdata[64];
        int      ddata[64];
        int      qdata[64];
15      int      eobmp;
        HandleData      handledata(header_ch, pixel_ch, data_ch,
                                   hdata, eobmp);
        DCT             dct(hdata, ddata);
        Quantization    quantization(ddata, qdata);
20      HuffmanEncode   huffmanencode(qdata, data_ch);

    void main(void) {
        printf("********************\n");
        printf("JPEG_Encoder_Begin...\n");
25      printf("********************\n");
        block_no = 0 ;
        eobmp = 0;
        do
        {
30              block_no ++ ;
                printf("Processing_Block_%dth...\n", block_no);

                handledata.main();
                dct.main();
35              quantization.main();
                huffmanencode.main();
        }while(eobmp!=1);
        WriteBits(-1, 0, data_ch);
        WriteMarker(M_EOI, data_ch);
40      printf("******************\n");
        printf("JPEG_Encoder_End...\n");
        printf("******************\n");
    }
};
```

A.2 Testbench

A.2.1 io.sc

```
//------------------------------------------------------------------------
// Input/Output behaviors for testbench
//------------------------------------------------------------------------
//
5 // 05/17/99  A. Gerstlauer
//   * Fixed a bug in BMP file reading: scan lines are aligned to
//     LONG boundaries.
//
// 05/10/99  A. Gerstlauer
10 //

#ifndef __IO_
#define __IO_

15 import "global";

//------------------------------------------------------------------------
// Input stimuli generator

20 // Type definitions for BMP file format

typedef short WORD;
typedef long DWORD;

25 typedef struct tagBITMAPFILEHEADER {
        WORD bfType;
        DWORD bfSize;
        WORD bfReserved1;
        WORD bfReserved2;
30      DWORD bfOffBits;
} BITMAPFILEHEADER;

typedef struct tagBITMAPINFOHEADER {
        /* BITMAP core header info -> OS/2 */
35      DWORD biSize;
        DWORD biWidth;
        DWORD biHeight;
        WORD biPlanes;
        WORD biBitCount;
40
        /* BITMAP info -> Windows 3.1 */
        DWORD biCompression;
        DWORD biSizeImage;
        DWORD biXPelsPerMeter;
45      DWORD biYPelsPerMeter;
        DWORD biClrUsed;
        DWORD biClrImportant;
} BITMAPINFOHEADER;

50 typedef struct tagRGBTRIPLE {
        BYTE B, G, R;
} RGBTRIPLE;


55 // Input behavior
// Read BMP file and send data to JPEG behavior.
// NOTE: Only able to handle certain BMP files: grayscale, non-compressed...
behavior Input (in char* ifname,
                iBlckSendInt Header,
60              iBlckSendByte Pixel)
{
  FILE* ifp;
```

```
BITMAPFILEHEADER BmpFileHeader;
BITMAPINFOHEADER BmpInfoHeader;
RGBTRIPLE *BmpColors;
int BmpScanWidth, BmpScanHeight;

int ReadRevWord()
{
  int c;
  c = fgetc(ifp);
  c |= fgetc(ifp) << 8;

  return c;
}

int ReadWord()
{
  int c;
  c = fgetc(ifp) << 8;
  c |= fgetc(ifp);

  return c;
}

int ReadByte()
{
  return fgetc(ifp);
}

long ReadRevDWord()
{
  long c;
  c = fgetc(ifp);
  c |= fgetc(ifp) << 8;
  c |= fgetc(ifp) << 16;
  c |= fgetc(ifp) << 24;

  return c;
}

long ReadDWord()
{
  long c;
  c = fgetc(ifp) << 24;
  c |= fgetc(ifp) << 16;
  c |= fgetc(ifp) << 8;
  c |= fgetc(ifp);

  return c;
}

int  IsBmpFile()
{
  int t = ('B'<<8) | 'M';
  int c;
  c = ReadWord();
  fseek(ifp, -2, 1);

  return t == c;
}

void ReadBmpHeader()
{
  int i, count;

  if (!IsBmpFile()) {
    error("This file is not compatible with BMP format.\n", 0);
```

```
    }
130
    /* BMP file header */
    BmpFileHeader.bfType = ReadWord();
    BmpFileHeader.bfSize = ReadRevDWord();
    BmpFileHeader.bfReserved1 = ReadRevWord();
135 BmpFileHeader.bfReserved2 = ReadRevWord();
    BmpFileHeader.bfOffBits = ReadRevDWord();

    /* BMP core info */
    BmpInfoHeader.biSize = ReadRevDWord();
140 BmpInfoHeader.biWidth = ReadRevDWord();
    BmpInfoHeader.biHeight = ReadRevDWord();
    BmpInfoHeader.biPlanes = ReadRevWord();
    BmpInfoHeader.biBitCount = ReadRevWord();

145 if (BmpInfoHeader.biSize > 12) {
      BmpInfoHeader.biCompression = ReadRevDWord();
      BmpInfoHeader.biSizeImage = ReadRevDWord();
      BmpInfoHeader.biXPelsPerMeter = ReadRevDWord();
      BmpInfoHeader.biYPelsPerMeter = ReadRevDWord();
150   BmpInfoHeader.biClrUsed = ReadRevDWord();
      BmpInfoHeader.biClrImportant = ReadRevDWord();

      /* read RGBQUAD */
      count = BmpFileHeader.bfOffBits - ftell(ifp);
155   count >>= 2;

      BmpColors = (RGBTRIPLE*) calloc(sizeof(RGBTRIPLE), count);

      for (i=0; i<count; i++) {
160     BmpColors[i].B = ReadByte();
        BmpColors[i].G = ReadByte();
        BmpColors[i].R = ReadByte();
        ReadByte();
      }
165 }
    else {
      /* read RGBTRIPLE */
      count = BmpFileHeader.bfOffBits - ftell(ifp);
      count /= 3;
170
      BmpColors = (RGBTRIPLE*) calloc(sizeof(RGBTRIPLE), count);

      for (i=0; i<count; i++) {
        BmpColors[i].B = ReadByte();
175     BmpColors[i].G = ReadByte();
        BmpColors[i].R = ReadByte();
      }
    }

180 /* BMP scan line is aligned by LONG boundary */
    if (BmpInfoHeader.biBitCount == 24) {
      BmpScanWidth = ((BmpInfoHeader.biWidth*3 + 3) >> 2) << 2;
      BmpScanHeight = BmpInfoHeader.biHeight;
    }
185 else {
      BmpScanWidth = ((BmpInfoHeader.biWidth + 3) >> 2) << 2;
      BmpScanHeight = BmpInfoHeader.biHeight;
    }
  }
190

  void main(void)
  {
    int c, r;
```

```
195      char buf;

         // Open file
         ifp = fopen(ifname, "rb");
         if (!ifp) {
200        error("Cannot open input file %s\n", ifname);
         }

         // Read BMP file header
         ReadBmpHeader();
205
         // Send header (size) information
         Header.send (BmpInfoHeader.biWidth);
         Header.send (BmpInfoHeader.biHeight);

210      // Loop over rows
         for (r = BmpInfoHeader.biHeight - 1; r >= 0; r--)
         {
           // Position file pointer to corresponding row
           fseek (ifp, BmpFileHeader.bfOffBits + r * BmpScanWidth, 0);
215
           // Loop over columns
           for (c = 0; c < BmpInfoHeader.biWidth; c++) {
             // Read pixel
             if (ferror(ifp) || (fread(&buf, 1, 1, ifp) != 1)) {
220            error("Error reading data from file %s\n", ifname);
             }

             // Send off
             Pixel.send(buf);
225        }
         }

         fclose (ifp);
       }
230 };


   //-----------------------------------------------------------------------
   // Output monitor.
235
   // End of image marker
   #define MARKER       0xFF
   #define MARK_EOI     0xD9

240
   // Output behavior
   // Listen to JPEG output and write it into a file.
   behavior Output (in char* ofname,
                    iBlckRecvByte Data)
245 {
     FILE* ofp;

     void main(void)
     {
250    bool running, marker;
       unsigned char buf;

       // Open file
       if (ofname) {
255      ofp = fopen(ofname, "wb");
         if (!ofp) {
           error("Cannot open output file %s\n", ofname);
         }
       }
260    else {
```

```
      ofp = openStdout();
    }

    // Read JPEG data, write to file
265 running = true;
    marker = false;
    while (running)
    {
      buf = Data.receive();
270
      if ((fwrite(&buf, 1, 1, ofp) != 1) || ferror(ofp)) {
        error("Error writing to file %s\n", ofname);
      }

275   // Is it a marker?
      if (marker) {
        // End of image?
        running = (buf != MARK_EOI);
        marker = false;
280   }
      else {
        marker = (buf == MARKER);
      }

285 }

    fclose (ofp);
  }
};
290
#endif
```

A.2.2 tb.sc

```
//---------------------------------------------------------------------
// JPEG Testbench
//---------------------------------------------------------------------
//
5 // 05/10/99 A. Gerstlauer
//

import "io";
import "jpeg";
10
behavior Main
{
  char* ifname;
  char* ofname;
15
  // Channels
  cSyncInt  header;
  cSyncByte pixel;
  cSyncByte data;
20
  Input   input(ifname, header, pixel);
  Jpeg    jpeg(header, pixel, data);
  Output  output(ofname, data);

25
  int main (int argc, char** argv)
  {
    // Command line arguments
    if (argc < 2) {
30    error("Usage: %s infile [ outfile ]\n", argv[0]);
    }
    ifname = argv[1];
    if (argc >= 3) {
```

```
        ofname = argv[2];
35    } else {
        ofname = 0;
      }

      // And now run the stuff ...
40    par {
        input.main();
        jpeg.main();
        output.main();
      }
45
      return 0;
    }

  };
```

B Architecture Model

This section contains the SpecC code for the architecture model of the JPEG encoder. Compared to the specification model, the major refinement of the model is limited to the top level structure. The testbench and the lower levels of the hierarchy are unchanged. Therefor, only those behaviors that have changed are listed here.

B.1 SpecC Code before Channel Partition

B.1.1 sw.sc

```
import "handle";
import "quant";
import "huff";

5 #include "const.sh"


behavior BOHData(in int hdata[64], iBlckSendBlock CHData)
{
10        void main ( void )
          {
                  // send item hdata over channel
                  CHData.send ( hdata ) ;
          }
15 };

behavior BOEOBmp(in int eobmp, iBlckSendInt CEOBmp)
{
          void main ( void )
20        {
                  // send item eobmp over channel
                  CEOBmp.send ( eobmp ) ;
          }
};
25
behavior BIDData(out int ddata[64], iBlckRecvBlock CDData)
{
          void main ( void )
          {
30                // receive item ddata over channel
                  CDData.receive ( ddata ) ;
          }
};

35 behavior SW(iBlckRecvInt header_ch, iBlckRecvByte pixel_ch,
          iBlckSendByte data_ch,
          iBlckSendBlock CHData,
          iBlckSendInt CEOBmp,
          iBlckRecvBlock CDData)
40 {
          int     hdata[64];
          int     ddata[64];
          int     qdata[64];
          int     eobmp;
45        int     block_no ; //for display
          int     i ;

          HandleData      handledata(header_ch, pixel_ch, data_ch,
                                     hdata, eobmp);
50        Quantization    quantization(ddata, qdata);
          HuffmanEncode   huffmanencode(qdata, data_ch);

          BOEOBmp         OEOBmp(eobmp, CEOBmp);
          BOHData         OHData(hdata, CHData);
```

```
55          BIDData             IDData(ddata, CDData);

      void main(void) {

          eobmp = 0 ;
60        block_no = 0 ;

          do
          {
                  block_no ++ ;
65                printf("Processing␣Block␣%dth...\n", block_no);

                  handledata.main();       // original behavior

                  //send data from ColdFire to DCT
70                OEOBmp.main() ;          // send eobmp ouput
                  OHData.main() ;          // send hdata ouput

                  //Receive data from DCT
                  IDData.main() ;          // receive ddata ouput
75
                  quantization.main();     // original behavior
                  huffmanencode.main();    // original behavior

          }while(eobmp!=1);  // end of while
80
          WriteBits(-1, 0, data_ch);
          WriteMarker(M_EOI, data_ch);

      }   // end of main
85 };  // end of behavior
```

B.1.2 hw.sc

```
import "dct";

#include "const.sh"

5 behavior BIHData(out int hdata[64], iBlckRecvBlock CHData)
{
        void main ( void )
        {
                CHData.receive ( hdata ) ;
10      }
};

behavior BIEOBmp(out int eobmp, iBlckRecvInt CEOBmp)
{
15      void main ( void )
        {
                eobmp = CEOBmp.receive() ;
        }
};
20
behavior BODData(in int ddata[64], iBlckSendBlock CDData)
{
        void main ( void )
        {
25              CDData.send ( ddata ) ;
        }
};

behavior HW(    iBlckRecvBlock CHData,
30              iBlckRecvInt CEOBmp,
                iBlckSendBlock CDData)
{
        int     hdata[64];
```

```
        int      ddata[64];
35      int      eobmp;

        DCT      dct(hdata, ddata);
        BIHData IHData(hdata, CHData);
        BIEOBmp IEOBmp(eobmp, CEOBmp);
40      BODData ODData(ddata, CDData);

    void main(void) {
        do
        {
45              //Receive data from ColdFire
                IEOBmp.main();  // receive eobmp output
                IHData.main();  // receive hdata output

                dct.main();     // original behavior
50
                //send data from DCT to ColdFire
                ODData.main();  // send ddata output
        }while(eobmp!=1) ;  // end of while
    }  // end of main
55 };  // end of behavior
```

B.1.3 jpeg.sc

```
import "sw";
import "hw";

#include "const.sh"
5
behavior Jpeg(iBlckRecvInt header_ch, iBlckRecvByte pixel_ch,
        iBlckSendByte data_ch)
{

10      cSyncBlock       chdata, cddata;
        cSyncInt         ceobmp;

        SW               sw(header_ch, pixel_ch, data_ch,
                                chdata, ceobmp, cddata);
15      HW               hw(chdata, ceobmp, cddata);

    void main(void) {
        printf("*********************\n");
        printf("JPEG_Encoder_Begin...\n");
20      printf("*********************\n");
        par
        {

                sw.main();
25              hw.main();
        }
        printf("*******************\n");
        printf("JPEG_Encoder_End...\n");
        printf("*******************\n");
30  }
};
```

B.2 SepcC Code after Channel Partition

B.2.1 sw.sc

```
import "handle";
import "quant";
import "huff";
import "bus";
5
#include "const.sh"
```

```
   #include "bus.sh"

   behavior BOHData(in int hdata[64], iBus bus)
10 {
        void main ( void )
        {
                // send item hdata over channel
                bus.send ( hdata, DCT_DATAIN ) ;
15      }
   };

   behavior BOEOBmp(in int eobmp, iBus bus)
   {
20      void main ( void )
        {
                // send item eobmp over channel
                bus.send ( &eobmp, DCT_EOBMP ) ;
        }
25 };

   behavior BIDData(out int ddata[64], iBus bus)
   {
        void main ( void )
30      {
                // receive item ddata over channel
                bus.receive ( ddata, DCT_DATAOUT ) ;
        }
   };
35
   behavior SW(iBlckRecvInt header_ch, iBlckRecvByte pixel_ch,
        iBlckSendByte data_ch,
        iBus bus)
   {
40      int     hdata[64];
        int     ddata[64];
        int     qdata[64];
        int     eobmp;
        int     block_no ; //for display
45      int     i ;

        HandleData      handledata(header_ch, pixel_ch, data_ch,
                                   hdata, eobmp);
        Quantization    quantization(ddata, qdata);
50      HuffmanEncode   huffmanencode(qdata, data_ch);

        BOEOBmp         OEOBmp(eobmp, bus);
        BOHData         OHData(hdata, bus);
        BIDData         IDData(ddata, bus);
55
     void main(void) {

        eobmp = 0 ;
        block_no = 0 ;
60
        do
        {
                block_no ++ ;
                printf("Processing_Block_%dth...\n", block_no);
65
                handledata.main();      // original behavior

                //send data from ColdFire to DCT
                OEOBmp.main() ;         // send eobmp ouput
70              OHData.main() ;         // send hdata ouput

                //Receive data from DCT
```

```
                IDData.main() ;            // receive ddata ouput
75
                quantization.main();       // original behavior
                huffmanencode.main();      // original behavior

        }while(eobmp!=1);   // end of while

80      WriteBits(-1, 0, data_ch);
        WriteMarker(M_EOI, data_ch);

    }   // end of main
};  // end of behavior
```

B.2.2 hw.sc

```
import "dct";
import "bus";

#include "const.sh"
5#include "bus.sh"

behavior BIHData(out int hdata[64], iBus bus)
{
        void main ( void )
10      {
                bus.receive ( hdata, DCT_DATAIN ) ;
        }
};

15 behavior BIEOBmp(out int eobmp, iBus bus)
{
        void main ( void )
        {
                bus.receive( &eobmp, DCT_EOBMP ) ;
20      }
};

behavior BODData(in int ddata[64], iBus bus)
{
25      void main ( void )
        {
                bus.send ( ddata, DCT_DATAOUT ) ;
        }
};
30
behavior HW ( iBus bus )
{
        int     hdata[64];
        int     ddata[64];
35      int     eobmp;

        DCT     dct(hdata, ddata);
        BIHData IHData(hdata, bus);
        BIEOBmp IEOBmp(eobmp, bus);
40      BODData ODData( ddata, bus);

    void main(void) {
        do
        {
45              //Receive data from ColdFire
                IEOBmp.main();  // receive eobmp output
                IHData.main();  // receive hdata output

                dct.main();     // original behavior
50
                //send data from DCT to ColdFire
                ODData.main();  // send ddata output
```

```
        }while(eobmp!=1) ;   // end of while
    }  // end of main
55 };  // end of behavior
```

B.2.3 jpeg.sc

```
import "sw";
import "hw";
import "bus";

5 #include "const.sh"

behavior Jpeg(iBlckRecvInt header_ch, iBlckRecvByte pixel_ch,
        iBlckSendByte data_ch)
{
10
        Bus              bus();
        SW               sw(header_ch, pixel_ch, data_ch,
                                 bus);
        HW               hw(bus);
15
    void main(void) {
        printf("*********************\n");
        printf("JPEG Encoder Begin...\n");
        printf("*********************\n");
20      par
        {

                sw.main();
                hw.main();
25      }
        printf("*******************\n");
        printf("JPEG Encoder End...\n");
        printf("*******************\n");
    }
30 };
```

B.2.4 bus.sh

```
//channel index in the bus

#define DCT_BLOCKSIZE   20
#define DCT_DATAIN      30
5 #define DCT_DATAOUT   40
#define DCT_EOBMP       50
```

B.2.5 bus.sc

```
#ifndef __BUS_
#define __BUS_

import "chann";
5 #include "bus.sh"

interface iBus
{
        void send(int* data, int channel_index);
10
        void receive(int* data, int channel_index);
};

channel Bus()
15      implements iBus
{
    cSyncInt       ceobmp;
    cSyncBlock     chdata;
    cSyncBlock     cddata;
20
```

```
    void send(int* data, int channel_index)
    {
          switch(channel_index)
          {
25                case DCT_EOBMP:   return ceobmp.send(*data);
                  case DCT_DATAIN:  return chdata.send(data);
                  case DCT_DATAOUT: return cddata.send(data);
          }
    }
30
    void receive(int *data, int channel_index)
    {
          switch(channel_index)
          {
35                case DCT_EOBMP:
                        *data = ceobmp.receive();
                        return;
                  case DCT_DATAIN:
                        chdata.receive(data);
40                      return;
                  case DCT_DATAOUT:
                        cddata.receive(data);
                        return;
          }
45  }
  };
  #endif
```

C Communication Model

This section contains the SpecC code for the communication model of the JPEG encoder.

C.1 SpecC Code for the Communication Model

C.1.1 sw.sc

```
import "bus";
import "handle";
import "quant";
import "huff";
5
#include "const.sh"
#include "bus.sh"

behavior BOHData(in int hdata[64], iMasterBus bus)
10 {
        void main ( void )
        {
                // send item hdata over bus
                bus.sendBlock(hdata, DCT_DATAIN);
15      }
};

behavior BOEOBmp(in int eobmp, iMasterBus bus)
{
20      void main ( void )
        {
                bus.sendInt(eobmp, DCT_EOBMP);
        }
};
25
behavior BOBlockSize(in int blocksize, iMasterBus bus)
{
        void main ( void )
        {
30              bus.sendInt(blocksize, DCT_BLOCKSIZE);
        }
};

behavior BIDData(out int ddata[64], iMasterBus bus)
35 {
        void main ( void )
        {
                bus.recvBlock(ddata, DCT_DATAOUT ) ;
        }
40 };

behavior SW(iBlckRecvInt header_ch, iBlckRecvByte pixel_ch,
        iBlckSendByte data_ch,
        //Signals to the Interface
45      out bit[31:0] MWDATA,
        in bit[31:0] MRDATA,
        iOSignal MTSB,
        iISignal MTAB,
        iOSignal MWDATAOE,
50      //Signals directly to the HW
        out bit[31:0] MADDR,
        iOSignal MRWB,
        iISignal INTC
         )
55 {
        // master bus interface
        cMasterBus      bus(MWDATA, MRDATA, MTSB, MTAB, MWDATAOE,
```

```
                                MADDR, MRWB, INTC);
60        int      hdata[64];
          int      ddata[64];
          int      qdata[64];
          int      eobmp;
          int      blcksize;
65        int      block_no ; //for display
          int      i ;

          HandleData       handledata(header_ch, pixel_ch, data_ch,
                           hdata, eobmp);
70        Quantization     quantization(ddata, qdata);
          HuffmanEncode    huffmanencode(qdata, data_ch);

          BOEOBmp          OEOBmp(eobmp, bus);
          BOBlockSize      OBlockSize(blcksize, bus);
75        BOHData          OHData(hdata, bus);
          BIDData          IDData(ddata, bus);

      void main(void) {

80        eobmp = 0;
          block_no = 0 ;
          blcksize = 64 ;

          do
85        {
                  block_no ++ ;
                  printf("Processing Block %dth...\n", block_no);

                  handledata.main();        // original behavoir
90
                  //send data from ColdFire to DCT
                  OEOBmp.main() ;           // send eobmp ouput
                  OBlockSize.main() ;       // send blocksize output
                  OHData.main() ;           // send ddata ouput
95
                  //Receive data from DCT
                  IDData.main() ;           // receive ddata ouput

                  quantization.main();      // original behavior
100               huffmanencode.main();     // original behavior

          }while(eobmp!=1); // end of while

          WriteBits(-1, 0, data_ch);
105       WriteMarker(M_EOI, data_ch);

      } // end of main
  }; // end of behavior
```

C.1.2 hw.sc

```
  import "dct";
  import "bus";

  #include "const.sh"
 5#include "bus.sh"

  behavior BIHData(out int hdata[64], iSlaveBus bus)
  {
          void main ( void )
10        {
                  bus.recvBlock ( hdata, DCT_DATAIN ) ;
          }
  };
```

```
15 behavior BIBlockSize(out int blocksize, iSlaveBus bus)
   {
           void main ( void )
           {
                   bus.recvInt ( &blocksize, DCT_BLOCKSIZE ) ;
20         }
   };

   behavior BIEOBmp(out int eobmp, iSlaveBus bus)
   {
25         void main ( void )
           {
                   bus.recvInt ( &eobmp, DCT_EOBMP ) ;
           }
   };
30
   behavior BODData(in int ddata[64], iSlaveBus bus)
   {
           void main ( void )
           {
35                 bus.sendBlock ( ddata, DCT_DATAOUT ) ;
           }
   };

   behavior BOEnd(in int eobmp, iSlaveBus bus)
40 {
           void main ( void )
           {
                   if ( eobmp == 1 )
                           bus.sendEnd() ;
45         }
   };

   behavior HW(    inout bit[31:0] DB,
                   iISignal DBOE,
50                 iISignal TSB,
                   iOSignal TAB,
                   in bit[31:0] MADDR,
                   iISignal MRWB,
                   iOSignal EXEEND,
55                 iOSignal INTC)
   {
           // slave bus interface
           cSlaveBus bus(DB, DBOE, TSB, TAB, MADDR, MRWB, EXEEND, INTC);

60         int     hdata[64];
           int     ddata[64];
           int     eobmp;
           int     blocksize;

65         DCT     dct(hdata, ddata);
           BIHData IHData(hdata, bus);
           BIEOBmp IEOBmp(eobmp, bus);
           BIBlockSize IBlockSize(blocksize, bus);
           BODData ODData(ddata, bus);
70         BOEnd   OEnd(eobmp, bus);

       void main(void) {
           do
           {
75                 //Receive data from ColdFire
                   IEOBmp.main() ;          // receive eobmp
                   IBlockSize.main() ;      // receive blocksize
                   IHData.main() ;          // receive hdata
```

```
80                  dct.main() ;              // original behavior

                    //send data from DCT to ColdFire
                    ODData.main() ;           // send ddata

85                  //send end signal from DCT to transducer
                    OEnd.main() ;             // send end signal
            }while(eobmp!=1);  // end of while
        }   // end of main
    };  // end of behavior
```

C.1.3 jpeg.sc

```
import "sw";
import "hw";
import "transducer";

5 #include "const.sh"

behavior Jpeg(iBlckRecvInt header_ch, iBlckRecvByte pixel_ch,
        iBlckSendByte data_ch)
{
10
        bit[31:0] MWDATA ;        // coldfire write data bus
        bit[31:0] MRDATA ;        // coldfire read data bus
        bit[31:0] MADDR ;         // address
        bit[31:0] DB ;            // dct data bus
15      cSignal MTSB, MTAB, MWDATAOE ;  // coldfire-transducer control
        cSignal TSB, TAB, EXEEND, DBOE ;        // transducer-dct control
        cSignal MRWB, INTC ;            // coldfire-dct control

        SW              sw(header_ch, pixel_ch, data_ch,
20                          MWDATA, MRDATA, MTSB, MTAB, MWDATAOE,
                            MADDR, MRWB, INTC);
        HW              hw(DB, DBOE, TSB, TAB,
                            MADDR, MRWB, EXEEND, INTC);
        INTERFACE       inter(MWDATA, MRDATA, MTSB, MTAB, MWDATAOE,
25                          DB, DBOE, TSB, TAB, MRWB, EXEEND);

    void main(void) {
        MWDATAOE.assign(0) ;
        DBOE.assign(0) ;
30      printf("*********************\n");
        printf("JPEG Encoder Begin...\n");
        printf("*********************\n");
        par
        {
35              sw.main();
                inter.main();
                hw.main();
        }
        printf("******************\n");
40      printf("JPEG Encoder End...\n");
        printf("******************\n");
    }
};
```

C.1.4 bus.sh

```
//channel index in the bus

#define DCT_BLOCKSIZE    20
#define DCT_DATAIN       30
5 #define DCT_DATAOUT    40
#define DCT_EOBMP        50
```

C.1.5 bus.sc

```
   #ifndef __BUS_
   #define __BUS_

   #include "bus.sh"
 5 import "global";

   //Signal channel for representation of control signal
   interface iOSignal
   {
10         void assign ( int v ) ;
   };

   interface iISignal
   {
15         int val() ;
           void waitval ( int v ) ;
   };

   channel cSignal()
20         implements iISignal , iOSignal
   {
           int value=1;
           event ev;

25         void assign ( int v )    // assign a value
           {
                   value = v ;
                   notify ( ev ) ;
           }
30
           int val()         // return a value
           {
                   return value ;
           }
35
           void waitval ( int v )  // wait for a value
           {
                   while ( value != v )
                           wait ( ev ) ;
40         }
   };

   //master bus channle
   interface iMasterBusProtocol
45 {
           bit[31:0] read(bit[31:0] addr);
           void write(bit[31:0] addr, bit[31:0] data);
   };

50 // master bus side of protocol
   channel cMasterBusProtocol(     out bit[31:0] MWDATA,
                                   in bit[31:0] MRDATA,
                                   iOSignal MTSB,
                                   iISignal MTAB,
55                                 iOSignal MWDATAOE,
                                   out bit[31:0] MADDR,
                                   iOSignal MRWB
                                )
           implements iMasterBusProtocol
60 {
           bit[31:0] read(bit[31:0] addr)
           {
                   int data;
                   int val;
65
                   // first clock cycle CR0
```

```
            MADDR = addr ;              // assign address lines
            MRWB assign (1) ;           // assert read control
            MTSB. assign (0) ;          // assert the start of bus transfer
70          waitfor ( 6 ) ;

            // second clock cycle CR1
            val = MTAB. val () ;        // sample acknowledge line
            while(val==1)
75          {
                    MADDR = addr ;  // assign address lines
                    MTSB. assign (1) ; // deassert the start of bus transfer
                    MRWB assign (1) ; // assert read control
                    waitfor (6);
80                  val = MTAB. val (); // sample acknowledge line
            }

            data = MRDATA ;             // sample data bus
            waitfor (6);
85
            return data;
        }

        void write(bit[31:0] addr, bit[31:0] data)
90      {
            int val ;

            // first clock cycle CW0
            MADDR = addr ;              // assign address lines
95          MRWB assign (0) ;           // assert write control
            MWDATAOE. assign (0) ;      // deassert write data available
            MTSB. assign (0) ;          // assert the start of bus transfer
            waitfor ( 6 ) ;

100         // second clock cycle CW1
            val = MTAB. val () ;        // sample acknowledge line
            while(val==1)
            {
                    MADDR = addr ;  // assign address lines
105                 MTSB. assign (1) ; // deassert the start of bus transfer
                    MRWB assign (0) ; // assert write control
                    MWDATAOE assign (1); // assert write data available
                    MWDATA = data ; // drive data outputs
                    waitfor (6) ;
110                 val = MTAB. val () ; // sample acknowledge line
            }

            waitfor (6);
        }
115 };

    interface iMasterBus
    {
        void sendInt(int data, int channel_index);
120     void sendBlock(int data[64], int channel_index);

        void recvInt(int* data, int channel_index);
        void recvBlock(int data[64], int channle_index);

125 };

    channel cMasterBus (    out bit[31:0] MWDATA,
                            in bit[31:0] MRDATA,
                            iOSignal MTSB,
130                         iISignal MTAB,
                            iOSignal MWDATAOE,
                            out bit[31:0] MADDR,
```

```
                            iOSignal MRWB,
                            iISignal INTC
135                         )
          implements iMasterBus
   {

          cMasterBusProtocol        protocol (MWDATA, MRDATA, MTSB, MTAB, MWDATAOE,
140                                           MADDR, MRWB ) ;
          int i ;

          void sendInt(int data, int addr)
          {
145               // data transfer
                  protocol.write ( addr, data ) ;
          }

          void recvInt(int *data, int addr)
150       {
                  // high-level synchronization, wait for ready signal
                  INTC.waitval ( 0 ) ;

                  // data transfer
155               *data = protocol.read ( addr ) ;
          }

          void sendBlock(int data[64], int addr)
          {
160               // data transfer
                  for ( i=0; i<64; i++ )
                          protocol.write ( addr, data[i] ) ;
          }

165       void recvBlock(int data[64], int addr)
          {
                  // high-level synchronization, wait for ready signal
                  INTC.waitval ( 0 ) ;

170               // data transfer
                  for ( i=0; i<64; i++ )
                          data[i] = protocol.read ( addr ) ;
          }
   };
175
   //slave bus channel
   interface iSlaveBusProtocol
   {
          bit[31:0] receive(bit[31:0] addr);
180       void send(bit[31:0] addr, bit[31:0] data);
   };

   //slave bus side of protocol
   channel cSlaveBusProtocol(     inout bit[31:0] DB,
185                               iISignal DBOE,
                                  iISignal TSB,
                                  iOSignal TAB,
                                  in bit[31:0] MADDR,
                                  iISignal MRWB,
190                               iOSignal EXEEND
                           )
          implements iSlaveBusProtocol
   {

195       // coldfire writing dct
          bit[31:0] receive(bit[31:0] addr)
          {
                  int Reg_In ;
```

```
200             //DW0, idle state, waiting for new bus transfer
                TSB.waitval(0) ;
                TAB.assign(1) ;
                // exception
                if(addr!=MADDR)
205             {
                        printf("Internal Error!\n");
                        exit(0);
                }
                waitfor(4) ;
210
                //DW1,
                DBOE.waitval(1) ;
                Reg_In = DB ;
                TAB.assign(1) ;
215             waitfor(4);

                //DW2,
                TAB.assign(0) ;
                waitfor(4) ;
220             return Reg_In ;
        }

        // coldfire reading dct
        void send(bit[31:0] addr, bit[31:0] data)
225     {
                int Reg_Out;

                //DR0, idle state, waiting for new bus transfer
                TSB.waitval(0) ;
230             TAB.assign(1) ;
                // exception
                if(addr!=MADDR)
                {
                        printf("Internal Error!\n");
235                     exit(0);
                }
                waitfor(4);

                //DR1,
240             Reg_Out = data ;
                TAB.assign(1) ;
                waitfor(4);

                //DR2
245             DB = Reg_Out ;
                TAB.assign(0) ;
                waitfor(4);
                return;
        }
250 };

interface iSlaveBus
{
        void sendBlock(int data[64], int addr);
255
        void recvInt(int *data, int addr);

        void recvBlock(int data[64], int addr);

260     void sendEnd();
};

channel cSlaveBus (      inout bit[31:0] DB,
                         iISignal DBOE,
```

```
                              iISignal TSB,
                              iOSignal TAB,
                              in bit[31:0] MADDR,
                              iISignal MRWB,
                              iOSignal EXEEND,
                              iOSignal INTC
                              )
        implements iSlaveBus
{
        cSlaveBusProtocol        protocol ( DB, DBOE, TSB, TAB, MADDR, MRWB, EXEEND ) ;
        int i ;

        void recvInt(int *data, int addr)
        {
                // data transfer
                *data = protocol.receive( addr ) ;
        }

        void sendBlock(int data[64], int addr)
        {
                // high-level synchronization, wait for ready signal
                INTC.assign ( 0 ) ;
                // data transfer
                for ( i=0; i<64; i++ )
                        protocol.send ( addr, data[i] ) ;
                INTC.assign ( 1 ) ;
        }

        void recvBlock(int data[64], int addr)
        {
                // data transfer
                for ( i=0; i<64; i++ )
                        data[i] = protocol.receive ( addr ) ;
        }

        void sendEnd()
        {
                EXEEND.assign(0) ;
        }
};

#endif
```

C.1.6 transducer.sc

```
import "global";
import "bus";

#ifndef _INTERFACE_
#define _INTERFACE_

behavior INTERFACE(
                in bit[31:0] MWDATA,
                out bit[31:0] MRDATA,
                iISignal MTSB,
                iOSignal MTAB,
                iISignal MWDATAOE,
                inout bit[31:0]  DB,
                iOSignal DBOE,
                iOSignal TSB,
                iISignal TAB,
                iISignal MRWB,
                iISignal EXE_END
                  )
{
        int Reg_Data ;
        int val ;
```

```
    void main(void) {
25      //T0
        t1: MTAB.assign(1) ;
        TSB.assign(1) ;
        do
        {
30              // check if it's the end of bmp
                val = EXE_END.val() ;
                if(val==0) // End of bmp, exit
                        return;

35              waitfor(4);
                // check the value of MTSB
                val = MTSB.val() ;
                if(val==0)  // begin a new bus transfer
                        break ;
40      }while(1);

        val = MRWB.val() ;
        if(val==1)
        {
45              //TR1
                MTAB.assign(1) ;
                TSB.assign(0) ;
                waitfor(4);

50              //TR2
                MTAB.assign(1) ;
                TSB.assign(1) ;
                waitfor(4) ;

55              //TR3
                TAB.waitval(0) ;
                MTAB.assign(1) ;
                Reg_Data = DB ;
                waitfor(4) ;
60
                //TR4
                MRDATA = Reg_Data ;
                MTAB.assign(0) ;
                waitfor(4) ;
65
                //TR5
                MRDATA = Reg_Data ;
                MTAB.assign(0) ;
                waitfor(4) ;
70
                goto t1 ;
        }
        else
        {
75              //TW1
                MTAB.assign(1) ;
                waitfor(4);

                //TW2
80              val = MWDATAOE.val() ;
                while(val==0)
                {
                        MTAB.assign(1) ;
                        waitfor(4) ;
85                      val = MWDATAOE.val() ;
                }

                //TW3
```

```
                    TSB.assign(0);
90                  MTAB.assign(1);
                    DBOE.assign(0);
                    Reg_Data = MWDATA;
                    waitfor(4);

95                  //TW4
                    TSB.assign(1);
                    MTAB.assign(1);
                    DB = Reg_Data;
                    DBOE.assign(1);
100                 waitfor(4);

                    //WS5
                    MTAB.assign(0);
                    waitfor(4);
105
                    //WS6
                    MTAB.assign(0);
                    waitfor(4);
                    goto t1;
110             }
        }
    };

    #endif
115
```

D Implementation Model (DCT only)

This section contains the SpecC code for the implementation model of the JPEG encoder. Only the HW part of the JPEG encoder was implemented.

D.1 SpecC Code for the Implementation Model of the HW

D.1.1 bus.sc

```
#ifndef __BUS__
#define __BUS__

#include "bus.sh"
import "global";

//Signal channel for representation of control signal
interface iOSignal
{
        void assign ( int v ) ;
};

interface iISignal
{
        int val() ;
        void waitval ( int v ) ;
};

channel cSignal()
        implements iISignal , iOSignal
{
        int value=1;
        event ev;

        void assign ( int v )    // assign a value
        {
                value = v ;
                notify ( ev ) ;
        }

        int val()          // return a value
        {
                return value ;
        }

        void waitval ( int v )  // wait for a value
        {
                while ( value != v )
                        wait ( ev ) ;
        }
};

//master bus channle
interface iMasterBusProtocol
{
        bit[31:0] read(bit[31:0] addr);
        void write(bit[31:0] addr, bit[31:0] data);
};

// master bus side of protocol
channel cMasterBusProtocol(      out bit[31:0] MWDATA,
                                 in bit[31:0] MRDATA,
                                 iOSignal MTSB,
                                 iISignal MTAB,
                                 iOSignal MWDATAOE,
                                 out bit[31:0] MADDR,
```

```
                                        iOSignal MRWB
                                )
        implements iMasterBusProtocol
60 {
        bit[31:0] read(bit[31:0] addr)
        {
                int data;
                int val;
65
                // first clock cycle CR0
                MADDR = addr ;              //  assign address lines
                MTSB.assign(0) ;            //  assert the start of bus transfer
                MRWB.assign(1) ;            //  assert read control
70              waitfor ( 6 ) ;

                // second clock cycle CR1
                val = MTAB.val() ;          //  sample acknowledge line
                while(val==1)
75              {
                        MADDR = addr ;   //  assign address lines
                        MTSB.assign(1) ; //  deassert the start of bus transfer
                        MRWB.assign(1) ; //  assert read control
                        waitfor(6);
80                      val = MTAB.val(); //  sample acknowledge line
                }

                data = MRDATA ;             //  sample data bus
                waitfor(6);
85
                //exit(0);
                return data;
        }

90      void write(bit[31:0] addr, bit[31:0] data)
        {
                int val ;

                // first clock cycle CW0
95              MADDR = addr ;              //  assign address lines
                MTSB.assign(0) ;            //  assert the start of bus transfer
                MRWB.assign(0) ;            //  assert write control
                MWDATAOE.assign(0) ;        //  deassert write data available
                waitfor ( 6 ) ;
100
                // second clock cycle CW1
                val = MTAB.val() ;          //  sample acknowledge line
                while(val==1)
                {
105                     MADDR = addr ;   //  assign address lines
                        MTSB.assign(1) ; //  deassert the start of bus transfer
                        MRWB.assign(0) ; //  assert write control
                        MWDATAOE.assign(1); //  assert write data available
                        MWDATA = data ;  //  drive data outputs
110                     waitfor(6) ;
                        val = MTAB.val() ; // sample acknowledge line
                }

                waitfor(6);
115     }
   };

   interface iMasterBus
   {
120     void sendInt(int data, int channel_index);
        void sendBlock(int data[64], int channel_index);
```

```
        void recvInt(int* data, int channel_index);
        void recvBlock(int data[64], int channle_index);
125
    };

    channel cMasterBus (     out bit[31:0] MWDATA,
                             in bit[31:0] MRDATA,
130                          iOSignal MTSB,
                             iISignal MTAB,
                             iOSignal MWDATAOE,
                             out bit[31:0] MADDR,
                             iOSignal MRWB,
135                          iISignal INTC
                             )
        implements iMasterBus
    {

140     cMasterBusProtocol        protocol (MWDATA, MRDATA, MTSB, MTAB, MWDATAOE,
                                            MADDR, MRWB ) ;
        int i ;

        void sendInt(int data, int addr)
145     {
                // data transfer
                protocol.write ( addr, data ) ;
        }

150     void recvInt(int *data, int addr)
        {
                // high-level synchronization, wait for ready signal
                INTC.waitval ( 0 ) ;
                waitfor(6) ;
155
                // data transfer
                *data = protocol.read ( addr ) ;
        }

160     void sendBlock(int data[64], int addr)
        {
                // data transfer
                for ( i=0; i<64; i++)
                        protocol.write ( addr, data[i] ) ;
165     }

        void recvBlock(int data[64], int addr)
        {
                // high-level synchronization, wait for ready signal
170             INTC.waitval ( 0 ) ;
                waitfor(6) ;

                // data transfer
                for ( i=0; i<64; i++)
175                     data[i] = protocol.read ( addr ) ;
        }
    };

    //slave bus channel
180 interface iSlaveBusProtocol
    {
        bit[31:0] receive(bit[31:0] addr);
        void send(bit[31:0] addr, bit[31:0] data);
    };
185
    //slave bus side of protocol
    channel cSlaveBusProtocol(      inout bit[31:0] DB,
                                    iISignal DBOE,
```

```
                              iISignal TSB,
190                           iOSignal TAB,
                              in bit[31:0] MADDR,
                              iISignal MRWB,
                              iOSignal EXEEND
                        )
195     implements iSlaveBusProtocol
    {

        typedef  enum { DW0, DW1, DW2, DW_End } dw_state ;
        typedef  enum { DR0, DR1, DR2, DR_End } dr_state ;
200
        // coldfire  writing  dct
        bit[31:0] receive(bit[31:0] addr)
        {
            int Reg_In ;
205         dw_state state ;

            state = DW0 ;

            while ( state != DW_End )
210         {
                switch ( state )
                {
                    case DW0 :
                        //DW0, idle state, waiting for new bus transfer
215                     TSB.waitval(0) ;
                        TAB.assign(1) ;
                        // exception
                        if(addr!=MADDR)
                        {
220                             printf("Internal_Error!\n");
                                exit(0);
                        }
                        state = DW1 ;
                        break ;
225
                    case DW1 :
                        DBOE.waitval(1) ;
                        Reg_In = DB ;
                        TAB.assign(1) ;
230                     state = DW2 ;
                        break ;

                    case DW2 :
                        TAB.assign(0) ;
235                     state = DW_End ;
                        break ;

                } // end of switch
                waitfor(4);
240         } // end of while
            return Reg_In ;
        } // end of receive

        // coldfire  reading  dct
245     void send(bit[31:0] addr, bit[31:0] data)
        {
            int Reg_Out ;
            dr_state state ;

250         state = DR0 ;

            while ( state != DR_End )
            {
                switch ( state )
```

```
255                 {
                        case DR0:
                            //DR0, idle state, waiting for new bus transfer
                            TSB.waitval(0) ;
                            TAB.assign(1) ;
260                         // exception
                            if(addr!=MADDR)
                            {
                                    printf("Internal␣Error!\n");
                                    exit(0);
265                         }
                            state = DR1 ;
                            break ;

                        case DR1 :
270                         Reg_Out = data ;
                            TAB.assign(1) ;
                            state = DR2 ;
                            break;

275                     case DR2 :
                            DB = Reg_Out ;
                            TAB.assign(0) ;
                            state = DR_End ;
                            break ;
280                 } // end of switch
                    waitfor(4) ;
                } // end of while
            } // end of send
    };
285
    interface iSlaveBus
    {
            void sendBlock(int data[64], int addr);

290         void recvInt(int *data, int addr);

            void recvBlock(int data[64], int addr);

            void sendEnd();
295 };

    channel cSlaveBus (      inout bit[31:0] DB,
                             iISignal DBOE,
                             iISignal TSB,
300                          iOSignal TAB,
                             in bit[31:0] MADDR,
                             iISignal MRWB,
                             iOSignal EXEEND,
                             iOSignal INTC
305                          )
            implements iSlaveBus
    {
            cSlaveBusProtocol        protocol ( DB, DBOE, TSB, TAB, MADDR, MRWB, EXEEND ) ;

310         typedef enum { SB0, SB1, SB2, SB3, SB_END } sb_state ;
            typedef enum { RB0, RB1, RB2, RB_END } rb_state ;


            void recvInt(int *data, int addr)
315         {
                    // data transfer
                    *data = protocol.receive( addr ) ;
            }

320         void sendBlock(int data[64], int addr)
```

```
    {
        int i ;
        sb_state state ;

325     state = SB0 ;

        while(state != SB_END)
        {
            switch(state)
330         {
                case SB0:    //  assert ready, start loop
                    // high-level synchronization , wait for ready signal
                    INTC.assign ( 0 ) ;
                    i = 0 ;
335                 state = SB1 ;
                    break ;

                case SB1:    // loop header
                    if(i<64)
340                         state = SB2 ;
                    else
                            state = SB3 ;
                    break ;

345             case SB2:
                    // data transfer
                    protocol.send ( addr, data[i] ) ;
                    i ++ ;
                    state = SB1 ;
350                 break ;

                case SB3:
                    INTC.assign ( 1 ) ;
                    state = SB_END ;
355                 break ;
            } // end of switch
            waitfor ( 4 ) ;
        } // end of while
    } // end of sendBlock
360
    void recvBlock(int data[64], int addr)
    {
        int i ;
        rb_state state ;
365
        state = RB0 ;

        while(state != RB_END)
        {
370         switch(state)
            {
                case RB0:    // start loop
                    i = 0 ;
                    state = RB1 ;
375                 break ;

                case RB1:    // loop header
                    if(i<64)
                            state = RB2 ;
380                 else
                            state = RB_END ;
                    break ;

                case RB2:
385                 // data transfer
                    data[i] = protocol.receive ( addr ) ;
```

```
                        i ++ ;
                        state = RB1 ;
                        break ;
390
                } // end of switch
                waitfor ( 4 ) ;
            } // end of while data transfer
        }
395
        void sendEnd()
        {
                EXEEND.assign(0) ;
        }
400 };

#endif
```

D.1.2 transducer.sc

```
import "global";
import "bus";

#ifndef _INTERFACE_
5 #define _INTERFACE_

behavior INTERFACE(
                in bit[31:0] MWDATA,
                out bit[31:0] MRDATA,
10              iISignal MTSB,
                iOSignal MTAB,
                iISignal MWDATAOE,
                inout bit[31:0]  DB,
                iOSignal DBOE,
15              iOSignal TSB,
                iISignal TAB,
                iISignal MRWB,
                iISignal EXE_END
                  )
20 {
        int Reg_Data ;
        int val ;
    void main(void) {

25      //T0
        t1: MTAB.assign(1) ;
        TSB.assign(1) ;
        do
        {
30              // check if it's the end of bmp
                val = EXE_END.val() ;
                if(val==0) // End of bmp, exit
                        return;

35              waitfor(4);
                // check the value of MTSB
                val = MTSB.val() ;
                if(val==0)  // begin a new bus transfer
                        break ;
40      }while(1);

        val = MRWB.val() ;
        if(val==1)
        {
45              //TR1
                MTAB.assign(1) ;
                TSB.assign(0) ;
                waitfor(4);
```

```
50                  //TR2
                    MTAB.assign(1) ;
                    TSB.assign(1) ;
                    waitfor(4) ;

55                  //TR3
                    TAB.waitval(0) ;
                    Reg_Data = DB ;
                    waitfor(4) ;

60                  //TR4
                    MRDATA = Reg_Data ;
                    MTAB.assign(0) ;
                    waitfor(4) ;

65                  //TR5
                    MRDATA = Reg_Data ;
                    MTAB.assign(0) ;
                    waitfor(4) ;

70                  goto t1 ;
            }
            else
            {
                    //TW1
75                  MTAB.assign(1) ;
                    waitfor(4);

                    //TW2
                    val = MWDATAOE.val() ;
80                  while(val==0)
                    {
                            MTAB.assign(1) ;
                            waitfor(4) ;
                            val = MWDATAOE.val() ;
85                  }

                    //TW3
                    TSB.assign(0) ;
                    MTAB.assign(1) ;
90                  DBOE.assign(0) ;
                    Reg_Data = MWDATA ;
                    waitfor(4) ;

                    //TW4
95                  TSB.assign(1) ;
                    MTAB.assign(1) ;
                    DBOE.assign(1) ;
                    DB = Reg_Data ;
                    waitfor(4) ;
100
                    //WS5
                    MTAB.assign(0) ;
                    waitfor(4) ;

105                 //WS6
                    MTAB.assign(0) ;
                    waitfor(4) ;
                    goto t1 ;
            }
110     }
    };

    #endif
```

D.1.3 hw.sc

```
import "dct";
import "bus";

#include "const.sh"
#include "bus.sh"

behavior BIHData(out int hdata[64], iSlaveBus bus)
{
        void main ( void )
        {
                bus.recvBlock ( hdata, DCT_DATAIN ) ;
        }
};

behavior BIBlockSize(out int blocksize, iSlaveBus bus)
{
        void main ( void )
        {
                bus.recvInt ( &blocksize, DCT_BLOCKSIZE ) ;
        }
};

behavior BIEOBmp(out int eobmp, iSlaveBus bus)
{
        void main ( void )
        {
                bus.recvInt ( &eobmp, DCT_EOBMP ) ;
        }
};

behavior BODData(in int ddata[64], iSlaveBus bus)
{
        void main ( void )
        {
                bus.sendBlock ( ddata, DCT_DATAOUT ) ;
        }
};

behavior BOEnd(in int eobmp, iSlaveBus bus)
{
        void main ( void )
        {
                bus.sendEnd() ;
        }
};

behavior HW(    inout bit[31:0] DB,
                iISignal DBOE,
                iISignal TSB,
                iOSignal TAB,
                in bit[31:0] MADDR,
                iISignal MRWB,
                iOSignal EXEEND,
                iOSignal INTC)
{
        // slave bus interface
        cSlaveBus bus(DB, DBOE, TSB, TAB, MADDR, MRWB, EXEEND, INTC);

        int     hdata[64];
        int     ddata[64];
        int     eobmp;
        int     blocksize;
        int     i ;
```

```
65          DCT       dct(hdata, ddata);
            BIHData IHData(hdata, bus);
            BIEOBmp IEOBmp(eobmp, bus);
            BIBlockSize IBlockSize(blocksize, bus);
            BODData ODData(ddata, bus);
70          BOEnd    OEnd(eobmp, bus);

        void main(void) {
            do
            {
75                  //Receive data from ColdFire
                    IEOBmp.main() ;          // receive eobmp
                    IBlockSize.main() ;      // receive blocksize
                    IHData.main() ;          // receive hdata

80                  dct.main() ;             // original behavior

                    //send data from DCT to ColdFire
                    ODData.main() ;          // send ddata
            }while(eobmp!=1);  // end of while
85
            //send end signal from DCT to transducer
            OEnd.main() ;            // send end signal

        }   // end of main
90 };  // end of behavior
```

D.1.4 dct.sc (Behavioral RTL)

This section contains the SpecC code for a general behavioral RTL view of the DCT behavior. It specifies the operations performed in each clock cycle without explicitly modelling the units in the components's data path. It's close to the original, sequential C code and easy to perform verification.

Please note, complex C expressions are not allowed inside the state and we have to sperate each complex expression into several simple 3-address C expressions. For this purpose, temporary variables (representing registers) are introduced.

Though the data path is not explicitly modelled, some constraints still has to be applied, such as the number of different resources that we can use for scheduling operations into clock cycles. In this design, we assume there is one ALU, one shifter, one multiplier , one divider, and several registers in the DCT custom hardware.

```
#include "const.sh"

import "global";
import "chann";
5
#define LS(r,s) ((r) << (s))
#define RS(r,s) ((r) >> (s))

behavior DCT(in int HData[64], out int DData[64])
10 {
        typedef enum {
        // PreShift
        PD_S1, PD_S2, PD_S3, PD_S4, PD_S5,

15      // Chen DCT
        CDCT_S1, CDCT_S2, CDCT_S3, CDCT_S4, CDCT_S5, CDCT_S6,
        CDCT_S7, CDCT_S8,CDCT_S9, CDCT_S10, CDCT_S11, CDCT_S12, CDCT_S13,
        CDCT_S14, CDCT_S15, CDCT_S16, CDCT_S17, CDCT_S18, CDCT_S19,
        CDCT_S20, CDCT_S21, CDCT_S22, CDCT_S23, CDCT_S24, CDCT_S25,
20      CDCT_S26, CDCT_S27, CDCT_S28,CDCT_S29, CDCT_S30, CDCT_S31,
        CDCT_S32, CDCT_S33, CDCT_S34, CDCT_S35, CDCT_S36, CDCT_S37,
        CDCT_S38, CDCT_S39, CDCT_S40, CDCT_S41, CDCT_S42, CDCT_S43,
        CDCT_S44, CDCT_S45, CDCT_S46, CDCT_S47, CDCT_S48, CDCT_S49,
        CDCT_S50, CDCT_S51, CDCT_S52, CDCT_S53, CDCT_S54, CDCT_S55,
25      CDCT_S56, CDCT_S57, CDCT_S58,CDCT_S59, CDCT_S60, CDCT_S61,
        CDCT_S62, CDCT_S63, CDCT_S64, CDCT_S65, CDCT_S66, CDCT_S67,
```

```
        CDCT_S68, CDCT_S69, CDCT_S70, CDCT_S71, CDCT_S72, CDCT_S73,
        CDCT_S74, CDCT_S75, CDCT_S76, CDCT_S77, CDCT_S78, CDCT_S79,
        CDCT_S80, CDCT_S81, CDCT_S82, CDCT_S83, CDCT_S84, CDCT_S85,
30      CDCT_S86, CDCT_S87, CDCT_S88,CDCT_S89, CDCT_S90, CDCT_S91,
        CDCT_S92, CDCT_S93, CDCT_S94, CDCT_S95, CDCT_S96, CDCT_S97,
        CDCT_S98, CDCT_S99, CDCT_S100, CDCT_S101, CDCT_S102, CDCT_S103,
        CDCT_S104, CDCT_S105, CDCT_S106, CDCT_S107, CDCT_S108, CDCT_S109,
        CDCT_S110, CDCT_S111, CDCT_S112, CDCT_S113, CDCT_S114, CDCT_S115,
35      CDCT_S116, CDCT_S117, CDCT_S118,CDCT_S119, CDCT_S120, CDCT_S121,
        CDCT_S122, CDCT_S123, CDCT_S124,

        // Bound
        BD_S0,BD_S1, BD_S2, BD_S3, BD_S4, BD_S5, BD_S6, BD_S7, BD_S8, BD_S9,
40
        // Bmp ends
        Blck_End
        } State_value ;

45      int     i ; // counter
        int     temp1, temp2, temp3, temp4 ;  // temporary variable

        int     *aptr, * bptr ;
        int a0, a1, a2, a3 ;
50      int b0, b1, b2, b3 ;
        int c0, c1, c2, c3 ;

        void main ( void )
        {
55          State_value state ;
            i = 0 ;

            state = PD_S1 ;

60          while ( state != Blck_End )
            {
                switch ( state )
                {
                        /////////////////////////////////////////////
65                      //      state PD_S1
                        /////////////////////////////////////////////
                        case PD_S1 : // loop head, begin PreshiftDct
                                if ( i < BLOCKSIZE)
                                        state = PD_S2;
70                              else
                                        state = PD_S5;
                                break ;

                        /////////////////////////////////////////////
75                      //      state PD_S2
                        /////////////////////////////////////////////
                        case PD_S2 :  // load item from memory
                                temp1 = HData[i];
                                state = PD_S3;
80                              break ;

                        /////////////////////////////////////////////
                        //      state PD_S3
                        /////////////////////////////////////////////
85                      case PD_S3 :
                                temp2 = temp1 - 128;
                                state = PD_S4;
                                break ;

90                      /////////////////////////////////////////////
                        //      state PD_S4
                        /////////////////////////////////////////////
```

```
        case PD_S4 :
                HData[i] = temp2 ;
95              i = i+1;
                state = PD_S1;
                break ;

        ////////////////////////////////////////////
100     //      state PD_S5
        ////////////////////////////////////////////
        case PD_S5 :    // loop end, go to ChenDct
                i = 0;
                state =CDCT_S1;
105             break ;

        ////////////////////////////////////////////
        //      state CDCT_S1
        ////////////////////////////////////////////
110     case CDCT_S1 :  // loop head, begin ChenDct
                if (i<8)
                        state =CDCT_S2;
                else
                        state =CDCT_S62;
115             break ;

        ////////////////////////////////////////////
        //      state CDCT_S2
        ////////////////////////////////////////////
120     case CDCT_S2 :
                aptr = HData + i ;
                state =CDCT_S3;
                break ;

125     ////////////////////////////////////////////
        //      state CDCT_S3
        ////////////////////////////////////////////
        case CDCT_S3 :
                bptr = aptr + 56 ;
130             state =CDCT_S4;
                break ;

        ////////////////////////////////////////////
        //      state CDCT_S4
135     ////////////////////////////////////////////
        case CDCT_S4 :
                temp1 = *aptr;
                temp2 = *bptr;
                state = CDCT_S5;
140             break ;

        ////////////////////////////////////////////
        //      state CDCT_S5
        ////////////////////////////////////////////
145     case CDCT_S5 :
                temp3 = temp1 + temp2 ;
                state = CDCT_S6;
                break ;

150     ////////////////////////////////////////////
        //      state CDCT_S6
        ////////////////////////////////////////////
        case CDCT_S6 :
                a0 = LS(temp3,2);
155             temp3 = temp1 - temp2 ;
                state = CDCT_S7;
                break ;
```

```
160         /////////////////////////////////////////////
            //        state  CDCT_S7
            /////////////////////////////////////////////
            case CDCT_S7 :
                    c3 = LS(temp3,2);
                    state = CDCT_S8;
165                 break ;

            /////////////////////////////////////////////
            //        state  CDCT_S8
            /////////////////////////////////////////////
170         case CDCT_S8 :
                    aptr = aptr + 8;
                    state = CDCT_S9;
                    break ;

175         /////////////////////////////////////////////
            //        state  CDCT_S9
            /////////////////////////////////////////////
            case CDCT_S9 :
                    bptr = bptr - 8;
180                 state = CDCT_S10;
                    break ;

            /////////////////////////////////////////////
            //        state  CDCT_S10
185         /////////////////////////////////////////////
            case CDCT_S10 :
                    temp1 = *aptr;
                    temp2 = *bptr;
                    state = CDCT_S11;
190                 break;

            /////////////////////////////////////////////
            //        state  CDCT_S11
            /////////////////////////////////////////////
195         case CDCT_S11 :
                    temp3 = temp1 + temp2;
                    state = CDCT_S12;
                    break;

200         /////////////////////////////////////////////
            //        state  CDCT_S12
            /////////////////////////////////////////////
            case CDCT_S12 :
                    a1 = LS(temp3, 2) ;
205                 temp3 = temp1 - temp2;
                    state = CDCT_S13;
                    break;

            /////////////////////////////////////////////
210         //        state  CDCT_S13
            /////////////////////////////////////////////
            case CDCT_S13 :
                    c2 = LS(temp3,2);
                    state = CDCT_S14;
215                 break ;

            /////////////////////////////////////////////
            //        state  CDCT_S14
            /////////////////////////////////////////////
220         case CDCT_S14 :
                    aptr = aptr + 8 ;
                    state = CDCT_S15;
                    break;
```

```
//////////////////////////////////////////////
//        state CDCT_S15
//////////////////////////////////////////////
case CDCT_S15 :
        bptr = bptr - 8 ;
        state = CDCT_S16;
        break;

//////////////////////////////////////////////
//        state CDCT_S16
//////////////////////////////////////////////
case CDCT_S16 :
        temp1 = *aptr;
        temp2 = *bptr;
        state = CDCT_S17;
        break;

//////////////////////////////////////////////
//        state CDCT_S17
//////////////////////////////////////////////
case CDCT_S17 :
        temp3 = temp1+temp2;
        state = CDCT_S18;
        break;

//////////////////////////////////////////////
//        state CDCT_S18
//////////////////////////////////////////////
case CDCT_S18 :
        a2 = LS(temp3,2);
        temp3 = temp1-temp2;
        state = CDCT_S19;
        break;

//////////////////////////////////////////////
//        state CDCT_S19
//////////////////////////////////////////////
case CDCT_S19 :
        c1 = LS(temp3,2);
        state = CDCT_S20;
        break;

//////////////////////////////////////////////
//        state CDCT_S20
//////////////////////////////////////////////
case CDCT_S20 :
        aptr = aptr + 8 ;
        state = CDCT_S21 ;
        break;

//////////////////////////////////////////////
//        state CDCT_S21
//////////////////////////////////////////////
case CDCT_S21 :
        bptr = bptr - 8 ;
        state = CDCT_S22;
        break;

//////////////////////////////////////////////
//        state CDCT_S22
//////////////////////////////////////////////
case CDCT_S22 :
        temp1 = *aptr ;
        temp2 = *bptr ;
        state = CDCT_S23;
        break;
```

```
295     ////////////////////////////////////////////
        //        state CDCT_S23
        ////////////////////////////////////////////
        case CDCT_S23 :
                temp3 = temp1+temp2;
                state = CDCT_S24;
                break;

300     ////////////////////////////////////////////
        //        state CDCT_S24
        ////////////////////////////////////////////
        case CDCT_S24 :
                a3 = LS(temp3,2);
305             temp3 = temp1-temp2;
                state = CDCT_S25;
                break;

        ////////////////////////////////////////////
310     //        state CDCT_S25
        ////////////////////////////////////////////
        case CDCT_S25 :
                c0 = LS(temp3,2);
                state = CDCT_S26;
315             break;

        ////////////////////////////////////////////
        //        state CDCT_S26
        ////////////////////////////////////////////
320     case CDCT_S26 :
                b0 = a0 + a3 ;
                state = CDCT_S27;
                break;

325     ////////////////////////////////////////////
        //        state CDCT_S27
        ////////////////////////////////////////////
        case CDCT_S27 :
                b1 = a1 + a2 ;
330             state = CDCT_S28;
                break;

        ////////////////////////////////////////////
        //        state CDCT_S28
335     ////////////////////////////////////////////
        case CDCT_S28 :
                b2 = a1 - a2 ;
                state = CDCT_S29 ;
                break;
340
        ////////////////////////////////////////////
        //        state CDCT_S29
        ////////////////////////////////////////////
        case CDCT_S29 :
345             b3 = a0 - a3 ;
                state = CDCT_S30 ;
                break;

        ////////////////////////////////////////////
350     //        state CDCT_S30
        ////////////////////////////////////////////
        case CDCT_S30 :
                aptr = DData + i ;
                state = CDCT_S31;
355             break;
```

```
//////////////////////////////////////////////
//      state CDCT_S31
//////////////////////////////////////////////
case CDCT_S31 :
        temp1 = b0 + b1 ;
        state = CDCT_S32 ;
        break;

//////////////////////////////////////////////
//      state CDCT_S32
//////////////////////////////////////////////
case CDCT_S32 :
        temp2 = 362 * temp1 ;
        temp1 = b0 - b1 ;
        state = CDCT_S33;
        break;

//////////////////////////////////////////////
//      state CDCT_S33
//////////////////////////////////////////////
case CDCT_S33 :
        temp3 = RS(temp2, 9) ;
        temp2 = 362 * temp1 ;
        state = CDCT_S34 ;
        break;

//////////////////////////////////////////////
//      state CDCT_S34
//////////////////////////////////////////////
case CDCT_S34 :
        *aptr = temp3 ;
        temp3 = RS(temp2, 9) ;
        state = CDCT_S35 ;
        break;

//////////////////////////////////////////////
//      state CDCT_S35
//////////////////////////////////////////////
case CDCT_S35 :
        aptr[32] = temp3 ;
        state = CDCT_S36 ;
        break;

//////////////////////////////////////////////
//      state CDCT_S36
//////////////////////////////////////////////
case CDCT_S36 :
        temp1 = 196 * b2 ;
        state = CDCT_S37 ;
        break;

//////////////////////////////////////////////
//      state CDCT_S37
//////////////////////////////////////////////
case CDCT_S37 :
        temp2 = 473 * b3 ;
        state = CDCT_S38 ;
        break;

//////////////////////////////////////////////
//      state CDCT_S38
//////////////////////////////////////////////
case CDCT_S38 :
        temp3 = temp1 + temp2 ;
        temp1 = 196 * b3 ;
        state = CDCT_S39 ;
```

```
                break;
425
        /////////////////////////////////////////////
        //          state  CDCT_S39
        /////////////////////////////////////////////
        case CDCT_S39 :
                temp4 = RS(temp3, 9) ;
430             temp2 = 473 * b2 ;
                state = CDCT_S40 ;
                break ;

        /////////////////////////////////////////////
435     //          state  CDCT_S40
        /////////////////////////////////////////////
        case CDCT_S40 :
                aptr[16] = temp4 ;
                temp3 = temp1 - temp2 ;
440             state = CDCT_S41 ;
                break;

        /////////////////////////////////////////////
        //          state  CDCT_S41
445     /////////////////////////////////////////////
        case CDCT_S41 :
                temp4 = RS(temp3, 9) ;
                state = CDCT_S42 ;
                break;
450
        /////////////////////////////////////////////
        //          state  CDCT_S42
        /////////////////////////////////////////////
        case CDCT_S42 :
455             aptr[48] = temp4 ;
                state = CDCT_S43;
                break;

        /////////////////////////////////////////////
460     //          state  CDCT_S43
        /////////////////////////////////////////////
        case CDCT_S43 :
                temp1 = c2 - c1 ;
                state = CDCT_S44 ;
465             break;

        /////////////////////////////////////////////
        //          state  CDCT_S44
        /////////////////////////////////////////////
470     case CDCT_S44 :
                temp2 = temp1 * 362 ;
                temp1 = c2 + c1 ;
                state = CDCT_S45 ;
                break;
475
        /////////////////////////////////////////////
        //          state  CDCT_S45
        /////////////////////////////////////////////
        case CDCT_S45 :
480             b0 = RS(temp2, 9) ;
                temp2 = temp1 * 362 ;
                state = CDCT_S46 ;
                break;

485     /////////////////////////////////////////////
        //          state  CDCT_S46
        /////////////////////////////////////////////
        case CDCT_S46 :
```

```
                 b1 = RS(temp2, 9) ;
490              state = CDCT_S47 ;
                 break;

     ////////////////////////////////////////////
     //      state CDCT_S47
495  ////////////////////////////////////////////
     case CDCT_S47 :
                 a0 = c0 + b0 ;
                 state = CDCT_S48 ;
                 break;
500
     ////////////////////////////////////////////
     //      state CDCT_S48
     ////////////////////////////////////////////
     case CDCT_S48 :
505              a1 = c0 - b0 ;
                 state = CDCT_S49 ;
                 break;

     ////////////////////////////////////////////
510  //      state CDCT_S49
     ////////////////////////////////////////////
     case CDCT_S49 :
                 a2 = c3 - b1 ;
                 state = CDCT_S50 ;
515              break;

     ////////////////////////////////////////////
     //      state CDCT_S50
     ////////////////////////////////////////////
520  case CDCT_S50 :
                 a3 = c3 + b1 ;
                 state = CDCT_S51 ;
                 break;

525  ////////////////////////////////////////////
     //      state CDCT_S51
     ////////////////////////////////////////////
     case CDCT_S51 :
                 temp1 = 100 * a0 ;
530              state = CDCT_S52;
                 break;

     ////////////////////////////////////////////
     //      state CDCT_S52
535  ////////////////////////////////////////////
     case CDCT_S52 :
                 temp2 = 502 * a3 ;
                 state = CDCT_S53 ;
                 break;
540
     ////////////////////////////////////////////
     //      state CDCT_S53
     ////////////////////////////////////////////
     case CDCT_S53 :
545              temp3 = temp1 + temp2 ;
                 temp1 = 426 * a2;
                 state = CDCT_S54 ;
                 break;

550  ////////////////////////////////////////////
     //      state CDCT_S54
     ////////////////////////////////////////////
     case CDCT_S54 :
                 temp4 = RS(temp3, 9) ;
```

```
555                     temp2 = 284 * a1 ;
                        state = CDCT_S55 ;
                        break;

                ////////////////////////////////////////////
560             //      state CDCT_S55
                ////////////////////////////////////////////
                case CDCT_S55 :
                        aptr[8] = temp4 ;
                        temp3 = temp1 - temp2 ;
565                     temp1 = 426 * a1 ;
                        state = CDCT_S56 ;
                        break;

                ////////////////////////////////////////////
570             //      state CDCT_S56
                ////////////////////////////////////////////
                case CDCT_S56 :
                        temp4 = RS(temp3, 9) ;
                        temp2 = 284 * a2 ;
575                     state = CDCT_S57 ;
                        break;

                ////////////////////////////////////////////
                //      state CDCT_S57
580             ////////////////////////////////////////////
                case CDCT_S57 :
                        aptr[24] = temp4 ;
                        temp3 = temp1 + temp2 ;
                        temp1 = 100 * a3 ;
585                     state = CDCT_S58 ;
                        break;

                ////////////////////////////////////////////
                //      state CDCT_S58
590             ////////////////////////////////////////////
                case CDCT_S58 :
                        temp4 = RS(temp3, 9) ;
                        temp2 = 502 * a0 ;
                        state = CDCT_S59 ;
595                     break;

                ////////////////////////////////////////////
                //      state CDCT_S59
                ////////////////////////////////////////////
600             case CDCT_S59 :
                        aptr[40] = temp4 ;
                        temp3 = temp1 - temp2 ;
                        state = CDCT_S60 ;
                        break;
605
                ////////////////////////////////////////////
                //      state CDCT_S60
                ////////////////////////////////////////////
                case CDCT_S60 :
610                     temp4 = RS(temp3, 9) ;
                        state = CDCT_S61 ;
                        break;

                ////////////////////////////////////////////
615             //      state CDCT_S61
                ////////////////////////////////////////////
                case CDCT_S61 :
                        aptr[56] = temp4 ;
                        i ++ ;
620                     state = CDCT_S1 ;
```

```
               break ;

          /////////////////////////////////////////////
          //        state  CDCT_S62
625       /////////////////////////////////////////////
          case CDCT_S62 : // loop  initialization
               i = 0;
               state = CDCT_S63 ;
               break;
630
          /////////////////////////////////////////////
          //        state  CDCT_S63
          /////////////////////////////////////////////
          case CDCT_S63 : // loop  head
635            if ( i<8)
                         state =CDCT_S64 ;
               else
                         state =CDCT_S117 ;
               break;
640
          /////////////////////////////////////////////
          //        state  CDCT_S64
          /////////////////////////////////////////////
          case CDCT_S64 :
645            temp1 = LS(i , 3) ;
               state = CDCT_S65 ;
               break;

          /////////////////////////////////////////////
650       //        state  CDCT_S65
          /////////////////////////////////////////////
          case CDCT_S65 :
               aptr = DData + temp1 ;
               state = CDCT_S66 ;
655            break;

          /////////////////////////////////////////////
          //        state  CDCT_S66
          /////////////////////////////////////////////
660       case CDCT_S66 :
               bptr = aptr + 7 ;
               state = CDCT_S67 ;
               break;

665       /////////////////////////////////////////////
          //        state  CDCT_S67
          /////////////////////////////////////////////
          case CDCT_S67 :
               temp1 = *aptr ;
670            temp2 = *bptr ;
               aptr = aptr + 1 ;
               state = CDCT_S68 ;
               break;

675       /////////////////////////////////////////////
          //        state  CDCT_S68
          /////////////////////////////////////////////
          case CDCT_S68 :
               temp3 = temp1 - temp2 ;
680            state = CDCT_S69 ;
               break;

          /////////////////////////////////////////////
          //        state  CDCT_S69
685       /////////////////////////////////////////////
          case CDCT_S69 :
```

```
                c3 = RS(temp3, 1) ;
                temp3 = temp1 + temp2 ;
                state = CDCT_S70 ;
690             break;

        ////////////////////////////////////////////
        //      state CDCT_S70
        ////////////////////////////////////////////
695     case CDCT_S70 :
                a0 = RS(temp3, 1) ;
                bptr = bptr - 1 ;
                state = CDCT_S71 ;
                break;
700
        ////////////////////////////////////////////
        //      state CDCT_S71
        ////////////////////////////////////////////
        case CDCT_S71 :
705             temp1 = *aptr ;
                temp2 = *bptr ;
                aptr = aptr + 1 ;
                state = CDCT_S72 ;
                break;
710
        ////////////////////////////////////////////
        //      state CDCT_S72
        ////////////////////////////////////////////
        case CDCT_S72 :
715             temp3 = temp1 - temp2 ;
                state = CDCT_S73 ;
                break;

        ////////////////////////////////////////////
720     //      state CDCT_S73
        ////////////////////////////////////////////
        case CDCT_S73 :
                c2 = RS(temp3, 1) ;
                temp3 = temp1 + temp2 ;
725             state = CDCT_S74 ;
                break;

        ////////////////////////////////////////////
        //      state CDCT_S74
730     ////////////////////////////////////////////
        case CDCT_S74 :
                a1 = RS(temp3, 1) ;
                bptr = bptr - 1 ;
                state = CDCT_S75 ;
735             break;

        ////////////////////////////////////////////
        //      state CDCT_S75
        ////////////////////////////////////////////
740     case CDCT_S75 :
                temp1 = *aptr ;
                temp2 = *bptr ;
                aptr = aptr + 1 ;
                state = CDCT_S76 ;
745             break;

        ////////////////////////////////////////////
        //      state CDCT_S76
        ////////////////////////////////////////////
750     case CDCT_S76 :
                temp3 = temp1 - temp2 ;
                state = CDCT_S77 ;
```

```
                break;
755     ////////////////////////////////////////////
        //      state CDCT_S77
        ////////////////////////////////////////////
        case CDCT_S77 :
                c1 = RS(temp3, 1) ;
760             temp3 = temp1 + temp2 ;
                state = CDCT_S78 ;
                break;

        ////////////////////////////////////////////
765     //      state CDCT_S78
        ////////////////////////////////////////////
        case CDCT_S78 :
                a2 = RS(temp3, 1) ;
                bptr = bptr - 1 ;
770             state = CDCT_S79 ;
                break;

        ////////////////////////////////////////////
        //      state CDCT_S79
775     ////////////////////////////////////////////
        case CDCT_S79 :
                temp1 = *aptr ;
                temp2 = *bptr ;
                state = CDCT_S80 ;
780             break;

        ////////////////////////////////////////////
        //      state CDCT_S80
        ////////////////////////////////////////////
785     case CDCT_S80 :
                temp3 = temp1 - temp2 ;
                state = CDCT_S81 ;
                break;

790     ////////////////////////////////////////////
        //      state CDCT_S81
        ////////////////////////////////////////////
        case CDCT_S81 :
                c0 = RS(temp3, 1) ;
795             temp3 = temp1 + temp2 ;
                state = CDCT_S82 ;
                break;

        ////////////////////////////////////////////
800     //      state CDCT_S82
        ////////////////////////////////////////////
        case CDCT_S82 :
                a3 = RS(temp3, 1) ;
                state = CDCT_S83 ;
805             break;

        ////////////////////////////////////////////
        //      state CDCT_S83
        ////////////////////////////////////////////
810     case CDCT_S83 :
                b0 = a0 + a3 ;
                state = CDCT_S84 ;
                break;

815     ////////////////////////////////////////////
        //      state CDCT_S84
        ////////////////////////////////////////////
        case CDCT_S84 :
```

```
820             b1 = a1 + a2 ;
                state = CDCT_S85 ;
                break;

        /////////////////////////////////////////////
        //          state CDCT_S85
825     /////////////////////////////////////////////
        case CDCT_S85 :
                b2 = a1 - a2 ;
                state = CDCT_S86 ;
                break;
830
        /////////////////////////////////////////////
        //          state CDCT_S86
        /////////////////////////////////////////////
        case CDCT_S86 :
835             b3 = a0 - a3 ;
                temp1 = LS(i, 3) ;
                state = CDCT_S87;
                break;

840     /////////////////////////////////////////////
        //          state CDCT_S87
        /////////////////////////////////////////////
        case CDCT_S87 :
                aptr = DData + temp1 ;
845             state = CDCT_S88 ;
                break;

        /////////////////////////////////////////////
        //          state CDCT_S88
850     /////////////////////////////////////////////
        case CDCT_S88 :
                temp1 = b0 + b1 ;
                state = CDCT_S89 ;
                break;
855
        /////////////////////////////////////////////
        //          state CDCT_S89
        /////////////////////////////////////////////
        case CDCT_S89 :
860             temp2 = 362 * temp1 ;
                temp1 = b0 - b1 ;
                state = CDCT_S90 ;
                break;

865     /////////////////////////////////////////////
        //          state CDCT_S90
        /////////////////////////////////////////////
        case CDCT_S90 :
                temp3 = RS(temp2, 9) ;
870             temp2 = 362 * temp1 ;
                state = CDCT_S91 ;
                break;

        /////////////////////////////////////////////
875     //          state CDCT_S91
        /////////////////////////////////////////////
        case CDCT_S91 :
                *aptr = temp3 ;
                temp3 = RS(temp2, 9) ;
880             temp1 = 196 * b2 ;
                state = CDCT_S92 ;
                break;

        /////////////////////////////////////////////
```

```
885         //       state CDCT_S92
            ////////////////////////////////////////////
            case CDCT_S92 :
                    aptr[4] = temp3 ;
                    temp2 = 473 * b3 ;
890                 state = CDCT_S93 ;
                    break;

            ////////////////////////////////////////////
            //       state CDCT_S93
895         ////////////////////////////////////////////
            case CDCT_S93 :
                    temp3 = temp1 + temp2 ;
                    temp1 = 196 * b3 ;
                    state = CDCT_S94 ;
900                 break;

            ////////////////////////////////////////////
            //       state CDCT_S94
            ////////////////////////////////////////////
905         case CDCT_S94 :
                    temp4 = RS(temp3, 9) ;
                    temp2 = 473 * b2 ;
                    state = CDCT_S95 ;
                    break;
910
            ////////////////////////////////////////////
            //       state CDCT_S95
            ////////////////////////////////////////////
            case CDCT_S95 :
915                 aptr[2] = temp4 ;
                    temp3 = temp1 - temp2 ;
                    state = CDCT_S96 ;
                    break;

920         ////////////////////////////////////////////
            //       state CDCT_S96
            ////////////////////////////////////////////
            case CDCT_S96 :
                    temp4 = RS(temp3, 9) ;
925                 state = CDCT_S97 ;
                    break;

            ////////////////////////////////////////////
            //       state CDCT_S97
930         ////////////////////////////////////////////
            case CDCT_S97 :
                    aptr[6] = temp4 ;
                    state = CDCT_S98 ;
                    break;
935
            ////////////////////////////////////////////
            //       state CDCT_S98
            ////////////////////////////////////////////
            case CDCT_S98 :
940                 temp1 = c2 - c1 ;
                    state = CDCT_S99 ;
                    break;

            ////////////////////////////////////////////
945         //       state CDCT_S99
            ////////////////////////////////////////////
            case CDCT_S99 :
                    temp2 = temp1 * 362 ;
                    temp1 = c2 + c1 ;
950                 state = CDCT_S100 ;
```

```
                break;

        ///////////////////////////////////////////
        //      state CDCT_S100
955     ///////////////////////////////////////////
        case CDCT_S100 :
                b0 = RS(temp2, 9) ;
                temp2 = temp1 * 362 ;
                state = CDCT_S101 ;
960             break;

        ///////////////////////////////////////////
        //      state CDCT_S101
        ///////////////////////////////////////////
965     case CDCT_S101 :
                b1 = RS(temp2, 9) ;
                state = CDCT_S102 ;
                break;

970     ///////////////////////////////////////////
        //      state CDCT_S102
        ///////////////////////////////////////////
        case CDCT_S102 :
                a0 = c0 + b0 ;
975             state = CDCT_S103 ;
                break;

        ///////////////////////////////////////////
        //      state CDCT_S103
980     ///////////////////////////////////////////
        case CDCT_S103 :
                a1 = c0 - b0 ;
                state = CDCT_S104 ;
                break;
985
        ///////////////////////////////////////////
        //      state CDCT_S104
        ///////////////////////////////////////////
        case CDCT_S104 :
990             a2 = c3 - b1 ;
                state = CDCT_S105 ;
                break;

        ///////////////////////////////////////////
995     //      state CDCT_S105
        ///////////////////////////////////////////
        case CDCT_S105 :
                a3 = c3 + b1 ;
                state = CDCT_S106 ;
1000            break;

        ///////////////////////////////////////////
        //      state CDCT_S106
        ///////////////////////////////////////////
1005    case CDCT_S106 :
                temp1 = 100 * a0 ;
                state = CDCT_S107 ;
                break;

1010    ///////////////////////////////////////////
        //      state CDCT_S107
        ///////////////////////////////////////////
        case CDCT_S107 :
                temp2 = 502 * a3 ;
1015            state = CDCT_S108 ;
                break;
```

```
1020    ////////////////////////////////////////////
        //      state CDCT_S108
        ////////////////////////////////////////////
        case CDCT_S108 :
                temp3 = temp1 + temp2 ;
                temp1 = 426 * a2 ;
                state = CDCT_S109 ;
1025            break;

        ////////////////////////////////////////////
        //      state CDCT_S109
        ////////////////////////////////////////////
1030    case CDCT_S109 :
                temp4 = RS(temp3, 9) ;
                temp2 = 284 * a1 ;
                state = CDCT_S110 ;
                break;
1035
        ////////////////////////////////////////////
        //      state CDCT_S110
        ////////////////////////////////////////////
        case CDCT_S110 :
1040            aptr[1] = temp4 ;
                temp3 = temp1 - temp2 ;
                temp1 = 426 * a1 ;
                state = CDCT_S111 ;
                break;
1045
        ////////////////////////////////////////////
        //      state CDCT_S111
        ////////////////////////////////////////////
        case CDCT_S111 :
1050            temp4 = RS(temp3, 9) ;
                temp2 = 284 * a2 ;
                state = CDCT_S112 ;
                break;

1055    ////////////////////////////////////////////
        //      state CDCT_S112
        ////////////////////////////////////////////
        case CDCT_S112 :
                aptr[3] = temp4 ;
1060            temp3 = temp1 + temp2 ;
                temp1 = 100 * a3 ;
                state = CDCT_S113 ;
                break;

1065    ////////////////////////////////////////////
        //      state CDCT_S113
        ////////////////////////////////////////////
        case CDCT_S113 :
                temp4 = RS(temp3, 9) ;
1070            temp2 = 502 * a0 ;
                state = CDCT_S114 ;
                break;

        ////////////////////////////////////////////
1075    //      state CDCT_S114
        ////////////////////////////////////////////
        case CDCT_S114 :
                aptr[5] = temp4 ;
                temp3 = temp1 - temp2 ;
1080            state = CDCT_S115 ;
                break;
```

```
1085        /////////////////////////////////////////////
            //        state CDCT_S115
            /////////////////////////////////////////////
            case CDCT_S115 :
                    temp4 = RS(temp3, 9) ;
                    state = CDCT_S116 ;
                    break;
1090
            /////////////////////////////////////////////
            //        state CDCT_S116
            /////////////////////////////////////////////
            case CDCT_S116 :
1095                aptr[7] = temp4 ;
                    i = i + 1 ;
                    state = CDCT_S63 ;
                    break;

1100        /////////////////////////////////////////////
            //        state CDCT_S117
            /////////////////////////////////////////////
            case CDCT_S117 :  // loop initialization
                    i = 0;
1105                aptr = DData ;
                    state = CDCT_S118 ;
                    break;

            /////////////////////////////////////////////
1110        //        state CDCT_S118
            /////////////////////////////////////////////
            case CDCT_S118 :  // loop header
                    if(i<64)
                            state = CDCT_S119 ;
1115                else
                            state = BD_S0;
                    break;

            /////////////////////////////////////////////
1120        //        state CDCT_S119
            /////////////////////////////////////////////
            case CDCT_S119 :
                    temp1 = *aptr ;
                    state = CDCT_S120 ;
1125                break;

            /////////////////////////////////////////////
            //        state CDCT_S120
            /////////////////////////////////////////////
1130        case CDCT_S120 :
                    if(temp1<0)
                            state = CDCT_S121 ;
                    else
                            state = CDCT_S122 ;
1135                break;

            /////////////////////////////////////////////
            //        state CDCT_S121
            /////////////////////////////////////////////
1140        case CDCT_S121 :
                    temp2 = temp1 - 4 ;
                    state = CDCT_S123 ;
                    break;

1145        /////////////////////////////////////////////
            //        state CDCT_S122
            /////////////////////////////////////////////
            case CDCT_S122 :
```

```
1150            temp2 = temp1 + 4 ;
                state = CDCT_S123 ;
                break;

        ////////////////////////////////////////////
1155    //        state CDCT_S123
        ////////////////////////////////////////////
        case CDCT_S123 :
                temp3 = temp2 / 8 ;
                state = CDCT_S124 ;
1160            break;

        ////////////////////////////////////////////
        //        state CDCT_S124
        ////////////////////////////////////////////
1165    case CDCT_S124 :
                *aptr = temp3 ;
                aptr = aptr + 1 ;
                i = i+1;
                state = CDCT_S118 ;
1170            break;

        ////////////////////////////////////////////
        //        state BD_S0
        ////////////////////////////////////////////
1175    case BD_S0 :
                i = 0;
                state= BD_S1;
                break;

1180    ////////////////////////////////////////////
        //        state BD_S1
        ////////////////////////////////////////////
        case BD_S1 :
                if (i < 64)
1185                    state = BD_S2;
                else
                        state = BD_S9;
                break;

1190    ////////////////////////////////////////////
        //        state BD_S2
        ////////////////////////////////////////////
        case BD_S2 :
                temp1 = DData[i] ;
1195            state = BD_S3;
                break;

        ////////////////////////////////////////////
        //        state BD_S3
1200    ////////////////////////////////////////////
        case BD_S3 :
                if (temp1 < -1023)  state = BD_S4;
                else state = BD_S5;
                break;

1205    ////////////////////////////////////////////
        //        state BD_S4
        ////////////////////////////////////////////
        case BD_S4 :
                temp1 = -1023;
1210            state = BD_S7;
                break;

        ////////////////////////////////////////////
        //        state BD_S5
```

```
1215                            ////////////////////////////////////////////////
                                case BD_S5 :
                                        if (templ > 1023)  state = BD_S6;
                                        else state = BD_S8;
                                        break;
1220
                                ////////////////////////////////////////////////
                                //      state BD_S6
                                ////////////////////////////////////////////////
                                case BD_S6 :
1225                                    templ = 1023 ;
                                        state = BD_S7 ;
                                        break;

                                ////////////////////////////////////////////////
1230                            //      state BD_S7
                                ////////////////////////////////////////////////
                                case BD_S7 :
                                        DData[i] = templ ;
                                        state = BD_S8 ;
1235                                    break;

                                ////////////////////////////////////////////////
                                //      state BD_S8
                                ////////////////////////////////////////////////
1240                            case BD_S8 :
                                        i = i + 1 ;
                                        state = BD_S1 ;
                                        break;

1245                            ////////////////////////////////////////////////
                                //      state BD_S9
                                ////////////////////////////////////////////////
                                case BD_S9 :
                                        i = 0 ;
1250                                    state = Blck_End ;
                                        break;
                        }//end of switch
                        waitfor(4) ;
                }//end of while
1255        }//end of main
    };//end of behavior
```

D.1.5 dct.sc (Behavioral RTL with known Data Path and bound variables)

This section contains the SpecC code for the behavioral RTL view of the DCT behavior under the constraints of a certain architecture (Figure 9).

Compared to the general behavioral RTL view of the DCT behavior, it's closer to the custom hardware, though it's still in the C format. Each variable in the code is mapped to a register or memory in the system. Since the data path is better known, more stages (clock cycles) have to be introduced to correctly model it.

```
    #include "const.sh"

    import "global";
    import "chann";
  5
    #define LS(r,s) ((r) << (s))
    #define RS(r,s) ((r) >> (s))

    behavior DCT(in int HData[64], out int DData[64])
 10 {
            typedef enum {
            // PreShift
            PD_S1, PD_S2, PD_S3, PD_S4, PD_S5,
```

```
15      // Chen DCT
        CDCT_S1, CDCT_S2, CDCT_S3, CDCT_S4, CDCT_S5, CDCT_S6,
        CDCT_S7, CDCT_S8,CDCT_S9, CDCT_S10, CDCT_S11, CDCT_S12, CDCT_S13,
        CDCT_S14, CDCT_S15, CDCT_S16, CDCT_S17, CDCT_S18, CDCT_S19,
        CDCT_S20, CDCT_S21, CDCT_S22, CDCT_S23, CDCT_S24, CDCT_S25,
20      CDCT_S26, CDCT_S27, CDCT_S28,CDCT_S29, CDCT_S30, CDCT_S31,
        CDCT_S32, CDCT_S33, CDCT_S34, CDCT_S35, CDCT_S36, CDCT_S37,
        CDCT_S38, CDCT_S39, CDCT_S40, CDCT_S41, CDCT_S42, CDCT_S43,
        CDCT_S44, CDCT_S45, CDCT_S46, CDCT_S47, CDCT_S48, CDCT_S49,
        CDCT_S50, CDCT_S51, CDCT_S52, CDCT_S53, CDCT_S54, CDCT_S55,
25      CDCT_S56, CDCT_S57, CDCT_S58,CDCT_S59, CDCT_S60, CDCT_S61,
        CDCT_S62, CDCT_S63, CDCT_S64, CDCT_S65, CDCT_S66, CDCT_S67,
        CDCT_S68, CDCT_S69, CDCT_S70, CDCT_S71, CDCT_S72, CDCT_S73,
        CDCT_S74, CDCT_S75, CDCT_S76, CDCT_S77, CDCT_S78, CDCT_S79,
        CDCT_S80, CDCT_S81, CDCT_S82, CDCT_S83, CDCT_S84, CDCT_S85,
30      CDCT_S86, CDCT_S87, CDCT_S88,CDCT_S89, CDCT_S90, CDCT_S91,
        CDCT_S92, CDCT_S93, CDCT_S94, CDCT_S95, CDCT_S96, CDCT_S97,
        CDCT_S98, CDCT_S99, CDCT_S100, CDCT_S101, CDCT_S102, CDCT_S103,
        CDCT_S104, CDCT_S105, CDCT_S106, CDCT_S107, CDCT_S108, CDCT_S109,
        CDCT_S110, CDCT_S111, CDCT_S112, CDCT_S113, CDCT_S114, CDCT_S115,
35      CDCT_S116, CDCT_S117, CDCT_S118,CDCT_S119, CDCT_S120, CDCT_S121,
        CDCT_S122, CDCT_S123, CDCT_S124, CDCT_S125, CDCT_S126, CDCT_S127,
        CDCT_S128, CDCT_S129, CDCT_S130, CDCT_S131, CDCT_S132, CDCT_S133,
        CDCT_S134, CDCT_S135, CDCT_S136, CDCT_S137, CDCT_S138, CDCT_S139,
        CDCT_S140, CDCT_S141, CDCT_S142, CDCT_S143, CDCT_S144, CDCT_S145,
40      CDCT_S146, CDCT_S147,

        // Bound
        BD_S0,BD_S1, BD_S2, BD_S3, BD_S4, BD_S5, BD_S6, BD_S7,

45      // Bmp ends
        Blck_End
        } State_value ;

        int     i, j, k;
50
        int     r_ram1, r_ram2 ;
        int     r_alu, r_sh, r_mul, r_div ;
        int     rf[32] ;

55      void main ( void )
        {
            State_value state ;
            i = 0 ;

60          state = PD_S1 ;

            while ( state != Blck_End )
            {
                switch ( state )
65              {
                        ////////////////////////////////////////////
                        //      state PD_S1
                        ////////////////////////////////////////////
                        case PD_S1 :
70                              if ( i < 64)
                                        state = PD_S2;
                                else
                                        state = PD_S5;
                                break ;
75                      ////////////////////////////////////////////
                        //      state PD_S2
                        ////////////////////////////////////////////
                        case PD_S2 :
                                r_ram1 = HData[i];
```

```
80                  state = PD_S3;
                    break ;
          ////////////////////////////////////////////
          //        state PD_S3
          ////////////////////////////////////////////
85        case PD_S3 :
                    r_alu = r_ram1 - 128;
                    state = PD_S4;
                    break ;
          ////////////////////////////////////////////
90        //        state PD_S4
          ////////////////////////////////////////////
          case PD_S4 :
                    HData[i] = r_alu;
                    i = i+1;
95                  state = PD_S1;
                    break ;
          ////////////////////////////////////////////
          //        state PD_S5
          ////////////////////////////////////////////
100       case PD_S5 :
                    i = 0;
                    state =CDCT_S1;
                    break ;
          ////////////////////////////////////////////
105       //        state CDCT_S1
          ////////////////////////////////////////////
          case CDCT_S1 :
                    if (i<8)
                              state =CDCT_S2;
110                 else
                              state =CDCT_S72;
                    break ;

          ////////////////////////////////////////////
115       //        state CDCT_S2
          ////////////////////////////////////////////
          case CDCT_S2 :
                    rf[0]=i;
                    r_alu = i+56;
120                 state =CDCT_S3;
                    break ;

          ////////////////////////////////////////////
          //        state CDCT_S3
125       ////////////////////////////////////////////
          case CDCT_S3 :
                    rf[1]=r_alu;
                    state =CDCT_S4;
                    break ;
130


          ////////////////////////////////////////////
          //        state CDCT_S4
          ////////////////////////////////////////////
135       case CDCT_S4 :
                    r_ram1 = HData[rf[0]];
                    r_ram2 = HData[rf[1]];
                    state = CDCT_S5;
                    break ;
140       ////////////////////////////////////////////
          //        state CDCT_S5
          ////////////////////////////////////////////
          case CDCT_S5 :
                    r_alu = r_ram1+r_ram2;
145                 state = CDCT_S6;
```

```
            break ;

    /////////////////////////////////////////////
    //          state CDCT_S6
150 /////////////////////////////////////////////
    case CDCT_S6 :
            r_sh  = LS(r_alu ,2);
            r_alu  = r_ram1-r_ram2;
            state  = CDCT_S7;
155         break ;

    /////////////////////////////////////////////
    //          state CDCT_S7
    /////////////////////////////////////////////
160 case CDCT_S7 :
            rf[2]=r_sh ;
            r_sh  = LS(r_alu ,2);
            state  = CDCT_S8;
            break ;
165
    /////////////////////////////////////////////
    //          state CDCT_S8
    /////////////////////////////////////////////
    case CDCT_S8 :
170         rf[13]=r_sh ;
            state  = CDCT_S9;
            break ;
    /////////////////////////////////////////////
    //          state CDCT_S9
175 /////////////////////////////////////////////
    case CDCT_S9 :
            rf[0]=rf[0]+8;
            state  = CDCT_S10;
            break ;
180 /////////////////////////////////////////////
    //          state CDCT_S10
    /////////////////////////////////////////////
    case CDCT_S10 :
            rf[1]=rf[1]-8;
185         state  = CDCT_S11;
            break ;

    ////////////////////////////////////////////////-
    ////////////////////////////////////////////////-
190 ////////////////////////////////////////////////-

    /////////////////////////////////////////////
    //          state CDCT_S11
    /////////////////////////////////////////////
195 case CDCT_S11 :
            r_ram1  = HData[rf[0]];
            r_ram2  = HData[rf[1]];
            state  = CDCT_S12;
            break;
200
    /////////////////////////////////////////////
    //          state CDCT_S12
    /////////////////////////////////////////////
    case CDCT_S12 :
205         r_alu  = r_ram1+r_ram2;
            state  = CDCT_S13;
            break;

    /////////////////////////////////////////////
210 //          state CDCT_S13
    /////////////////////////////////////////////
```

```
        case CDCT_S13 :
                r_sh = LS(r_alu,2);
                r_alu = r_ram1-r_ram2;
                state = CDCT_S14;
                break;

        /////////////////////////////////////////////
        //      state CDCT_S14
        /////////////////////////////////////////////
        case CDCT_S14 :
                rf[3]=r_sh;
                r_sh = LS(r_alu,2);
                state = CDCT_S15;
                break ;
        /////////////////////////////////////////////
        //      state CDCT_S15
        /////////////////////////////////////////////
        case CDCT_S15 :
                rf[12]=r_sh;
                state = CDCT_S16;
                break;

        /////////////////////////////////////////////
        //      state CDCT_S16
        /////////////////////////////////////////////
        case CDCT_S16 :
                rf[0]=rf[0]+8;
                state = CDCT_S17;
                break;

        /////////////////////////////////////////////
        //      state CDCT_S17
        /////////////////////////////////////////////
        case CDCT_S17 :
                rf[1]=rf[1]-8;
                state = CDCT_S18;
                break;

        /////////////////////////////////////////////
        /////////////////////////////////////////////
        /////////////////////////////////////////////
        //      state CDCT_S18
        /////////////////////////////////////////////
        case CDCT_S18 :
                r_ram1 = HData[rf[0]];
                r_ram2 = HData[rf[1]];
                state = CDCT_S19;
                break;

        /////////////////////////////////////////////
        //      state CDCT_S19
        /////////////////////////////////////////////
        case CDCT_S19 :
                r_alu = r_ram1+r_ram2;
                state = CDCT_S20;
                break;

        /////////////////////////////////////////////
        //      state CDCT_S20
        /////////////////////////////////////////////
        case CDCT_S20 :
                r_sh = LS(r_alu,2);
                r_alu = r_ram1-r_ram2;
                state = CDCT_S21;
                break;
```

```
////////////////////////////////////////////
//        state CDCT_S21
////////////////////////////////////////////
case CDCT_S21 :
        rf[4]=r_sh;
        r_sh = LS(r_alu,2);
        state = CDCT_S22;
        break;

////////////////////////////////////////////
//        state CDCT_S22
////////////////////////////////////////////
case CDCT_S22 :
        rf[11]=r_sh;
        state = CDCT_S23;
        break;

////////////////////////////////////////////
//        state CDCT_S23
////////////////////////////////////////////
case CDCT_S23 :
        rf[0]=rf[0]+8;
        state = CDCT_S24;
        break;

////////////////////////////////////////////
//        state CDCT_S24
////////////////////////////////////////////
case CDCT_S24 :
        rf[1] = rf[1] - 8 ;
        state = CDCT_S25;
        break;

////////////////////////////////////////////
////////////////////////////////////////////
////////////////////////////////////////////
//        state CDCT_S25
////////////////////////////////////////////
case CDCT_S25 :
        r_ram1 = HData[rf[0]];
        r_ram2 = HData[rf[1]];
        state = CDCT_S26;
        break;

////////////////////////////////////////////
//        state CDCT_S26
////////////////////////////////////////////
case CDCT_S26 :
        r_alu = r_ram1+r_ram2;
        state = CDCT_S27;
        break;

////////////////////////////////////////////
//        state CDCT_S27
////////////////////////////////////////////
case CDCT_S27 :
        r_sh = LS(r_alu,2);
        r_alu = r_ram1-r_ram2;
        state = CDCT_S28;
        break;

////////////////////////////////////////////
//        state CDCT_S28
////////////////////////////////////////////
case CDCT_S28 :
        rf[5]=r_sh;
```

```
345             r_sh = LS(r_alu,2);
                state = CDCT_S29;
                break;

            ////////////////////////////////////////////
            //      state CDCT_S29
350         ////////////////////////////////////////////
            case CDCT_S29 :
                rf[10]=r_sh;
                r_alu = rf[2]+rf[5];
                state = CDCT_S30;
355             break;

            ////////////////////////////////////////////
            //      state CDCT_S30
            ////////////////////////////////////////////
360         case CDCT_S30 :
                rf[6]=r_alu;
                r_alu = rf[3]+rf[4];
                state = CDCT_S31;
                break;
365
            ////////////////////////////////////////////
            //      state CDCT_S31
            ////////////////////////////////////////////
            case CDCT_S31 :
370             rf[7]=r_alu;
                r_alu = rf[3]-rf[4];
                state = CDCT_S32;
                break;
            ////////////////////////////////////////////
375         //      state CDCT_S32
            ////////////////////////////////////////////
            case CDCT_S32 :
                rf[8]=r_alu;
                r_alu = rf[2]-rf[5];
380             state = CDCT_S33;
                break;
            ////////////////////////////////////////////
            //      state CDCT_S33
            ////////////////////////////////////////////
385         case CDCT_S33 :
                rf[9]=r_alu;
                state = CDCT_S34;
                break;
            ////////////////////////////////////////////
390         //      state CDCT_S34
            ////////////////////////////////////////////
            case CDCT_S34 :
                rf[0]=i;
                state = CDCT_S35;
395             break;

            ////////////////////////////////////////////
            //      state CDCT_S35
            ////////////////////////////////////////////
400         case CDCT_S35 :
                r_alu = rf[6]+rf[7];
                state = CDCT_S36;
                break;

405         ////////////////////////////////////////////
            //      state CDCT_S36
            ////////////////////////////////////////////
            case CDCT_S36 :
                r_mul = 362*r_alu;
```

```
410                     r_alu = rf[6]-rf[7];
                        state = CDCT_S37;
                        break;

                ////////////////////////////////////////////
415             //      state CDCT_S37
                ////////////////////////////////////////////
                case CDCT_S37 :
                        r_sh = RS(r_mul,9);
                        r_mul = 362*r_alu;
420                     state = CDCT_S38;
                        break;

                ////////////////////////////////////////////
                //      state CDCT_S38
425             ////////////////////////////////////////////
                case CDCT_S38 :
                        DData[rf[0]] = r_sh;
                        r_sh = RS(r_mul,9);
                        state = CDCT_S39;
430                     break;

                ////////////////////////////////////////////
                //      state CDCT_S39
                ////////////////////////////////////////////
435             case CDCT_S39 :
                        r_alu = i+32;
                        state = CDCT_S40;
                        break;

440             ////////////////////////////////////////////
                //      state CDCT_S40
                ////////////////////////////////////////////
                case CDCT_S40 :
                        DData[r_alu] =r_sh;
445                     state = CDCT_S41;
                        break;

                ////////////////////////////////////////////
                //      state CDCT_S41
450             ////////////////////////////////////////////
                case CDCT_S41 :
                        r_mul = 196*rf[8];
                        state = CDCT_S42;
                        break;
455
                ////////////////////////////////////////////
                //      state CDCT_S42
                ////////////////////////////////////////////
                case CDCT_S42 :
460                     rf[15]=r_mul;
                        r_mul = 473*rf[9];
                        state = CDCT_S43;
                        break;

465             ////////////////////////////////////////////
                //      state CDCT_S43
                ////////////////////////////////////////////
                case CDCT_S43 :
                        r_alu = r_mul+rf[15];
470                     r_mul = 196*rf[9];
                        state = CDCT_S44;
                        break;

                ////////////////////////////////////////////
475             //      state CDCT_S44
```

```
////////////////////////////////////////////
case CDCT_S44 :
        r_sh = RS(r_alu,9);
        rf[14]=r_mul;
        r_mul = 473*rf[8];
        state = CDCT_S45;
        break ;

////////////////////////////////////////////
//      state CDCT_S45
////////////////////////////////////////////
case CDCT_S45 :
        rf[15]=r_sh;
        r_alu = rf[14]-r_mul;
        state = CDCT_S46;
        break;

////////////////////////////////////////////
//      state CDCT_S46
////////////////////////////////////////////
case CDCT_S46 :
        r_sh = RS(r_alu,9);
        r_alu = i+16;
        state = CDCT_S47;
        break;


////////////////////////////////////////////
//      state CDCT_S47
////////////////////////////////////////////
case CDCT_S47 :
        DData[r_alu] =rf[15];
        r_alu = i+48;
        state = CDCT_S48;
        break;

////////////////////////////////////////////
//      state CDCT_S48
////////////////////////////////////////////
case CDCT_S48 :
        DData[r_alu]=r_sh;
        state = CDCT_S49;
        break;

////////////////////////////////////////////
//      state CDCT_S49
////////////////////////////////////////////
case CDCT_S49 :
        r_alu = rf[12]-rf[11];
        state = CDCT_S50;
        break;

////////////////////////////////////////////
//      state CDCT_S50
////////////////////////////////////////////
case CDCT_S50 :
        r_mul = r_alu*362;
        r_alu = rf[12]+rf[11];
        state = CDCT_S51;
        break;

////////////////////////////////////////////
//      state CDCT_S51
////////////////////////////////////////////
case CDCT_S51 :
        r_sh = RS(r_mul, 9);
```

```
            r_mul = r_alu*362;
            state = CDCT_S52;
            break;

    /////////////////////////////////////////////
    //       state CDCT_S52
    /////////////////////////////////////////////
    case CDCT_S52 :
            rf[6]=r_sh;
            r_sh = RS(r_mul, 9);
            state = CDCT_S53;
            break;

    /////////////////////////////////////////////
    //       state CDCT_S53
    /////////////////////////////////////////////
    case CDCT_S53 :
            rf[7]=r_sh;
            r_alu = rf[10]+rf[6];
            state = CDCT_S54;
            break;

    /////////////////////////////////////////////
    //       state CDCT_S54
    /////////////////////////////////////////////
    case CDCT_S54 :
            rf[2] = r_alu ;
            r_alu = rf[10]-rf[6];
            state = CDCT_S55;
            break;

    /////////////////////////////////////////////
    //       state CDCT_S55
    /////////////////////////////////////////////
    case CDCT_S55 :
            rf[3] = r_alu ;
            r_alu = rf[13]-rf[7];
            state = CDCT_S56;
            break;

    /////////////////////////////////////////////
    //       state CDCT_S56
    /////////////////////////////////////////////
    case CDCT_S56 :
            rf[4] = r_alu ;
            r_alu = rf[13]+rf[7];
            state = CDCT_S57;
            break;

    /////////////////////////////////////////////
    //       state CDCT_S57
    /////////////////////////////////////////////
    case CDCT_S57 :
            rf[5] = r_alu;
            r_mul = 100*rf[2];
            state = CDCT_S58;
            break;
    /////////////////////////////////////////////
    /////////////////////////////////////////////
    /////////////////////////////////////////////

    /////////////////////////////////////////////
    //       state CDCT_S58
    /////////////////////////////////////////////
    case CDCT_S58 :
            rf[15] = r_mul ;
```

```
                r_mul = 502*rf[5];
                state = CDCT_S59;
610             break;

        //////////////////////////////////////////////
        //      state CDCT_S59
        //////////////////////////////////////////////
615     case CDCT_S59 :
                r_alu = r_mul+rf[15];
                r_mul = 426*rf[4];
                state = CDCT_S60;
                break;
620
        //////////////////////////////////////////////
        //      state CDCT_S60
        //////////////////////////////////////////////
        case CDCT_S60 :
625             r_sh = RS(r_alu,9);
                rf[14] = r_mul ;
                r_mul = 284*rf[3];
                state = CDCT_S61;
                break;
630
        //////////////////////////////////////////////
        //      state CDCT_S61
        //////////////////////////////////////////////
        case CDCT_S61 :
635             rf[15] = r_sh;
                r_alu = rf[14]-r_mul;
                state = CDCT_S62;
                break;

640     //////////////////////////////////////////////
        //      state CDCT_S62
        //////////////////////////////////////////////
        case CDCT_S62 :
                r_sh = RS(r_alu,9);
645             r_alu = i+8 ;
                state = CDCT_S63;
                break;


650     //////////////////////////////////////////////
        //      state CDCT_S63
        //////////////////////////////////////////////
        case CDCT_S63 :
                DData[r_alu] =rf[15];
655             r_alu = i+24;
                state = CDCT_S64;
                break;

        //////////////////////////////////////////////
660     //      state CDCT_S64
        //////////////////////////////////////////////
        case CDCT_S64 :
                DData[r_alu] = r_sh;
                r_mul = 426*rf[3];
665             state = CDCT_S65;
                break;
        //////////////////////////////////////////////
        //////////////////////////////////////////////
        //////////////////////////////////////////////
670     //////////////////////////////////////////////
        //      state CDCT_S65
        //////////////////////////////////////////////
        case CDCT_S65 :
```

```
675             rf[15] = r_mul ;
                r_mul = 284*rf[4];
                state = CDCT_S66;
                break;

680     ////////////////////////////////////////////
        //      state CDCT_S66
        ////////////////////////////////////////////
        case CDCT_S66 :
                r_alu = r_mul+rf[15];
685             r_mul = 100*rf[5];
                state = CDCT_S67;
                break;

        ////////////////////////////////////////////
690     //      state CDCT_S67
        ////////////////////////////////////////////
        case CDCT_S67 :
                r_sh = RS(r_alu,9);
                rf[14] = r_mul ;
695             r_mul = 502*rf[2];
                state = CDCT_S68;
                break;

        ////////////////////////////////////////////
700     //      state CDCT_S68
        ////////////////////////////////////////////
        case CDCT_S68 :
                rf[15] = r_sh;
                r_alu = rf[14]-r_mul;
705             state = CDCT_S69;
                break;

        ////////////////////////////////////////////
        //      state CDCT_S69
710     ////////////////////////////////////////////
        case CDCT_S69 :
                r_sh = RS(r_alu,9);
                r_alu = i+40 ;
                state = CDCT_S70;
715             break;


        ////////////////////////////////////////////
        //      state CDCT_S70
720     ////////////////////////////////////////////
        case CDCT_S70 :
                DData[r_alu] =rf[15];
                r_alu = i+56;
                state = CDCT_S71;
725             break;

        ////////////////////////////////////////////
        //      state CDCT_S71
        ////////////////////////////////////////////
730     case CDCT_S71 :
                DData[r_alu]=r_sh;
                i = i+1;
                state = CDCT_S1;
                break;
735
        ////////////////////////////////////////////
        //      state CDCT_S72
        ////////////////////////////////////////////
        case CDCT_S72 :
                i = 0;
```

```
740                       state = CDCT_S73;
                          break;

                  /////////////////////////////////////////////
                  //         state  CDCT_S73
745               /////////////////////////////////////////////
                  case CDCT_S73 :
                          if ( i<8)
                                  state =CDCT_S74;
                          else
750                               state =CDCT_S139;
                          break;

                  /////////////////////////////////////////////
                  //         state  CDCT_S74
755               /////////////////////////////////////////////
                  case CDCT_S74 :
                          r_sh = LS(i,3);
                          state = CDCT_S75;
                          break;
760
                  /////////////////////////////////////////////
                  //         state  CDCT_S75
                  /////////////////////////////////////////////
                  case CDCT_S75 :
765                       j = r_sh;
                          r_alu = r_sh+7;
                          state = CDCT_S76;
                          break;

770               /////////////////////////////////////////////
                  //         state  CDCT_S76
                  /////////////////////////////////////////////
                  case CDCT_S76 :
                          k = r_alu;
775                       state = CDCT_S77;
                          break;

                  /////////////////////////////////////////////
                  //         state  CDCT_S77
780               /////////////////////////////////////////////
                  case CDCT_S77 :
                          r_ram1 = DData[j];
                          r_ram2 = DData[k];
                          state = CDCT_S78;
785                       break;

                  /////////////////////////////////////////////
                  //         state  CDCT_S78
                  /////////////////////////////////////////////
790               case CDCT_S78 :
                          r_alu = r_ram1-r_ram2;
                          state = CDCT_S79;
                          break;

795               /////////////////////////////////////////////
                  //         state  CDCT_S79
                  /////////////////////////////////////////////
                  case CDCT_S79 :
                          r_sh = RS(r_alu,1);
800                       r_alu = r_ram1 + r_ram2 ;
                          j = j+1;
                          k = k-1;
                          state = CDCT_S80;
                          break;
805
```

```
            ////////////////////////////////////////////
            //        state CDCT_S80
            ////////////////////////////////////////////
            case CDCT_S80 :
810                 rf[13] = r_sh;
                    r_sh = RS(r_alu,1);
                    state = CDCT_S81;
                    break;

815         ////////////////////////////////////////////
            //        state CDCT_S81
            ////////////////////////////////////////////
            case CDCT_S81 :
                    rf[2]=r_sh ;
820                 state = CDCT_S82;
                    break;
            ////////////////////////////////////////////
            ////////////////////////////////////////////

825         ////////////////////////////////////////////
            //        state CDCT_S82
            ////////////////////////////////////////////
            case CDCT_S82 :
                    r_ram1 = DData[j];
830                 r_ram2 = DData[k];
                    state = CDCT_S83;
                    break;

            ////////////////////////////////////////////
835         //        state CDCT_S83
            ////////////////////////////////////////////
            case CDCT_S83 :
                    r_alu = r_ram1-r_ram2;
                    state = CDCT_S84;
840                 break;

            ////////////////////////////////////////////
            //        state CDCT_S84
            ////////////////////////////////////////////
845         case CDCT_S84 :
                    r_sh = RS(r_alu, 1) ;
                    r_alu = r_ram1+r_ram2;
                    j = j+1;
                    k = k-1;
850                 state = CDCT_S85;
                    break;

            ////////////////////////////////////////////
            //        state CDCT_S85
855         ////////////////////////////////////////////
            case CDCT_S85 :
                    rf[12] = r_sh;
                    r_sh = RS(r_alu,1);
                    state = CDCT_S86;
860                 break;

            ////////////////////////////////////////////
            //        state CDCT_S86
            ////////////////////////////////////////////
865         case CDCT_S86 :
                    rf[3] = r_sh;
                    state = CDCT_S87;
                    break;

870         ////////////////////////////////////////////
            ////////////////////////////////////////////
```

```
//        state CDCT_S87
/////////////////////////////////////////////
case CDCT_S87 :
        r_ram1 = DData[j];
        r_ram2 = DData[k];
        state = CDCT_S88;
        break;

/////////////////////////////////////////////
//        state CDCT_S88
/////////////////////////////////////////////
case CDCT_S88 :
        r_alu = r_ram1-r_ram2;
        state = CDCT_S89;
        break;

/////////////////////////////////////////////
//        state CDCT_S89
/////////////////////////////////////////////
case CDCT_S89 :
        r_sh = RS(r_alu, 1);
        r_alu = r_ram1+r_ram2;
        j = j+1;
        k = k-1;
        state = CDCT_S90;
        break;

/////////////////////////////////////////////
//        state CDCT_S90
/////////////////////////////////////////////
case CDCT_S90 :
        rf[11] = r_sh;
        r_sh = RS(r_alu,1);
        state = CDCT_S91;
        break;

/////////////////////////////////////////////
//        state CDCT_S91
/////////////////////////////////////////////
case CDCT_S91 :
        rf[4] = r_sh;
        state = CDCT_S92;
        break;

/////////////////////////////////////////////
//        state CDCT_S92
/////////////////////////////////////////////
case CDCT_S92 :
        r_ram1 = DData[j];
        r_ram2 = DData[k];
        state = CDCT_S93;
        break;

/////////////////////////////////////////////
//        state CDCT_S93
/////////////////////////////////////////////
case CDCT_S93 :
        r_alu = r_ram1-r_ram2;
        state = CDCT_S94;
        break;

/////////////////////////////////////////////
//        state CDCT_S94
/////////////////////////////////////////////
case CDCT_S94 :
        r_sh = RS(r_alu, 1);
```

```
                r_alu = r_ram1+r_ram2;
                j = j+1;
940             k = k-1;
                state = CDCT_S95;
                break;

        ////////////////////////////////////////////
945     //      state CDCT_S95
        ////////////////////////////////////////////
        case CDCT_S95 :
                rf[10] = r_sh;
                r_sh = RS(r_alu,1);
950             state = CDCT_S96;
                break;

        ////////////////////////////////////////////
        //      state CDCT_S96
955     ////////////////////////////////////////////
        case CDCT_S96 :
                rf[5] = r_sh;
                r_alu = rf[2] + r_sh;
                state = CDCT_S97;
960             break;

        ////////////////////////////////////////////
        //      state CDCT_S97
        ////////////////////////////////////////////
965     case CDCT_S97 :
                rf[6] = r_alu;
                r_alu = rf[3]+rf[4];
                state = CDCT_S98;
                break;
970

        ////////////////////////////////////////////
        //      state CDCT_S98
        ////////////////////////////////////////////
975     case CDCT_S98 :
                rf[7] = r_alu;
                r_alu = rf[3]-rf[4];
                state = CDCT_S99;
                break;
980
        ////////////////////////////////////////////
        //      state CDCT_S99
        ////////////////////////////////////////////
        case CDCT_S99 :
985             rf[8] = r_alu;
                r_alu = rf[2]-rf[5];
                state = CDCT_S100;
                break;

990
        ////////////////////////////////////////////
        //      state CDCT_S100
        ////////////////////////////////////////////
        case CDCT_S100 :
995             rf[9] = r_alu;
                r_sh = LS(i,3);
                state = CDCT_S101;
                break;

1000    ////////////////////////////////////////////
        //      state CDCT_S101
        ////////////////////////////////////////////
        case CDCT_S101 :
```

```
1005                rf[0] = r_sh;
                    state = CDCT_S102;
                    break;

          ////////////////////////////////////////////
          //        state CDCT_S102
1010      ////////////////////////////////////////////
          case CDCT_S102 :
                    r_alu = rf[6]+rf[7];
                    state = CDCT_S103;
                    break;
1015
          ////////////////////////////////////////////
          //        state CDCT_S103
          ////////////////////////////////////////////
          case CDCT_S103:
1020                r_mul = 362 * r_alu ;
                    r_alu = rf[6]-rf[7];
                    state = CDCT_S104;
                    break;

1025      ////////////////////////////////////////////
          //        state CDCT_S104
          ////////////////////////////////////////////
          case CDCT_S104 :
                    r_sh = RS(r_mul,9);
1030                r_mul = 362*r_alu;
                    state = CDCT_S105;
                    break;


1035      ////////////////////////////////////////////
          //        state CDCT_S105
          ////////////////////////////////////////////
          case CDCT_S105 :
                    DData[rf[0]] = r_sh ;
1040                r_sh = RS(r_mul,9);
                    state = CDCT_S106;
                    break;

          ////////////////////////////////////////////
1045      //        state CDCT_S106
          ////////////////////////////////////////////
          case CDCT_S106 :
                    r_alu = rf[0]+4;
                    state = CDCT_S107;
1050                break;

          ////////////////////////////////////////////
          //        state CDCT_S107
          ////////////////////////////////////////////
1055      case CDCT_S107 :
                    DData[r_alu]=r_sh;
                    state = CDCT_S108;
                    break;

1060      ////////////////////////////////////////////
          //        state CDCT_S108
          ////////////////////////////////////////////
          case CDCT_S108 :
                    r_mul = 196*rf[8];
1065                state = CDCT_S109;
                    break;

          ////////////////////////////////////////////
          //        state CDCT_S109
```

```
1070        /////////////////////////////////////////////
            case CDCT_S109 :
                    rf[15] = r_mul ;
                    r_mul = 473*rf[9];
                    state = CDCT_S110;
1075                break;

            /////////////////////////////////////////////
            //      state CDCT_S110
            /////////////////////////////////////////////
1080        case CDCT_S110 :
                    r_alu = r_mul+rf[15];
                    r_mul = 196*rf[9];
                    state = CDCT_S111;
                    break;
1085
            /////////////////////////////////////////////
            //      state CDCT_S111
            /////////////////////////////////////////////
            case CDCT_S111 :
1090                r_sh = RS(r_alu,9);
                    rf[14] = r_mul ;
                    r_mul = 473*rf[8];
                    state = CDCT_S112;
                    break;
1095
            /////////////////////////////////////////////
            //      state CDCT_S112
            /////////////////////////////////////////////
            case CDCT_S112 :
1100                rf[15] = r_sh;
                    r_alu = rf[14]-r_mul;
                    state = CDCT_S113;
                    break;

1105        /////////////////////////////////////////////
            //      state CDCT_S113
            /////////////////////////////////////////////
            case CDCT_S113 :
                    r_sh = RS(r_alu, 9) ;
1110                r_alu = rf[0]+2;
                    state = CDCT_S114;
                    break;


1115        /////////////////////////////////////////////
            //      state CDCT_S114
            /////////////////////////////////////////////
            case CDCT_S114 :
                    DData[r_alu] =rf[15];
1120                r_alu = rf[0]+6;
                    state = CDCT_S115;
                    break;

            /////////////////////////////////////////////
1125        //      state CDCT_S115
            /////////////////////////////////////////////
            case CDCT_S115 :
                    DData[r_alu]=r_sh;
                    state = CDCT_S116;
1130                break;


            /////////////////////////////////////////////
            //      state CDCT_S116
1135        /////////////////////////////////////////////
```

```
        case CDCT_S116 :
                r_alu = rf[12]-rf[11];
                state = CDCT_S117;
                break;

        ////////////////////////////////////////////
        //        state CDCT_S117
        ////////////////////////////////////////////
        case CDCT_S117 :
                r_mul = r_alu * 362 ;
                r_alu = rf[12]+rf[11];
                state = CDCT_S118;
                break;

        ////////////////////////////////////////////
        //        state CDCT_S118
        ////////////////////////////////////////////
        case CDCT_S118 :
                r_sh = RS(r_mul, 9);
                r_mul = r_alu*362;
                state = CDCT_S119;
                break;

        ////////////////////////////////////////////
        //        state CDCT_S119
        ////////////////////////////////////////////
        case CDCT_S119 :
                rf[6] = r_sh ;
                r_sh = RS(r_mul, 9);
                state = CDCT_S120;
                break;

        ////////////////////////////////////////////
        //        state CDCT_S120
        ////////////////////////////////////////////
        case CDCT_S120 :
                rf[7]=r_sh;
                r_alu = rf[10]+rf[6];
                state = CDCT_S121;
                break;

        ////////////////////////////////////////////
        //        state CDCT_S121
        ////////////////////////////////////////////
        case CDCT_S121 :
                rf[2] = r_alu ;
                r_alu = rf[10]-rf[6];
                state = CDCT_S122;
                break;

        ////////////////////////////////////////////
        //        state CDCT_S122
        ////////////////////////////////////////////
        case CDCT_S122 :
                rf[3] = r_alu ;
                r_alu = rf[13]-rf[7];
                state = CDCT_S123;
                break;

        ////////////////////////////////////////////
        //        state CDCT_S123
        ////////////////////////////////////////////
        case CDCT_S123 :
                rf[4] = r_alu ;
                r_alu = rf[13]+rf[7];
                state = CDCT_S124;
```

```
                break;

        ////////////////////////////////////////////
1205    //          state  CDCT_S124
        ////////////////////////////////////////////
        case CDCT_S124 :
                rf[5]=r_alu;
                r_mul = 100*rf[2];
1210            state = CDCT_S125;
                break;

        ////////////////////////////////////////////
        //          state  CDCT_S125
1215    ////////////////////////////////////////////
        case CDCT_S125 :
                rf[15] = r_mul ;
                r_mul = 502*rf[5];
                state = CDCT_S126;
1220            break;

        ////////////////////////////////////////////
        //          state  CDCT_S126
        ////////////////////////////////////////////
1225    case CDCT_S126 :
                r_alu = r_mul+rf[15];
                r_mul = 426*rf[4];
                state = CDCT_S127;
                break;
1230
        ////////////////////////////////////////////
        //          state  CDCT_S127
        ////////////////////////////////////////////
        case CDCT_S127 :
1235            r_sh = RS(r_alu,9);
                rf[14] = r_mul ;
                r_mul = 284*rf[3];
                state = CDCT_S128;
                break;
1240
        ////////////////////////////////////////////
        //          state  CDCT_S128
        ////////////////////////////////////////////
        case CDCT_S128 :
1245            rf[15] = r_sh;
                r_alu = rf[14]-r_mul;
                state = CDCT_S129;
                break;

1250    ////////////////////////////////////////////
        //          state  CDCT_S129
        ////////////////////////////////////////////
        case CDCT_S129 :
                r_sh = RS(r_alu,9);
1255            r_alu = rf[0] + 1 ;
                state = CDCT_S130;
                break;

        ////////////////////////////////////////////
1260    //          state  CDCT_S130
        ////////////////////////////////////////////
        case CDCT_S130 :
                DData[r_alu] =rf[15];
                r_alu = rf[0]+3;
1265            state = CDCT_S131;
                break;
```

```
1270  ////////////////////////////////////////////
      //        state  CDCT_S131
      ////////////////////////////////////////////
      case CDCT_S131 :
              DData[ r_alu]=r_sh ;
              r_mul = 426*rf [3];
1275          state = CDCT_S132;
              break;
      ////////////////////////////////////////////
      ////////////////////////////////////////////
      ////////////////////////////////////////////
1280  ////////////////////////////////////////////
      //        state  CDCT_S132
      ////////////////////////////////////////////
      case CDCT_S132 :
              rf [15] = r_mul ;
1285          r_mul = 284*rf [4];
              state = CDCT_S133;
              break;

      ////////////////////////////////////////////
1290  //        state  CDCT_S133
      ////////////////////////////////////////////
      case CDCT_S133 :
              r_alu = r_mul+rf [15];
              r_mul = 100*rf [5];
1295          state = CDCT_S134;
              break;

      ////////////////////////////////////////////
      //        state  CDCT_S134
1300  ////////////////////////////////////////////
      case CDCT_S134 :
              r_sh = RS( r_alu ,9);
              rf [14] = r_mul ;
              r_mul = 502*rf [2];
1305          state = CDCT_S135;
              break;

      ////////////////////////////////////////////
      //        state  CDCT_S135
1310  ////////////////////////////////////////////
      case CDCT_S135 :
              rf [15] = r_sh ;
              r_alu = rf [14]-r_mul;
              state = CDCT_S136;
1315          break;

      ////////////////////////////////////////////
      //        state  CDCT_S136
      ////////////////////////////////////////////
1320  case CDCT_S136 :
              r_sh = RS( r_alu , 9) ;
              r_alu = rf [0]+5;
              state = CDCT_S137;
              break;
1325
      ////////////////////////////////////////////
      //        state  CDCT_S137
      ////////////////////////////////////////////
      case CDCT_S137 :
1330          DData[ r_alu] =rf [15];
              r_alu = rf [0]+7;
              state = CDCT_S138;
              break;
```

```
1335 //////////////////////////////////////////////
     //        state CDCT_S138
     //////////////////////////////////////////////
     case CDCT_S138 :
             DData[r_alu]=r_sh;
             i = i+1;
1340         state = CDCT_S73;
             break;

     //////////////////////////////////////////////
     //        state CDCT_S139
1345 //////////////////////////////////////////////
     case CDCT_S139 :
             i = 0;
             state = CDCT_S140;
             break;
1350
     //////////////////////////////////////////////
     //        state CDCT_S140
     //////////////////////////////////////////////
     case CDCT_S140 :
1355         if(i<64)
                     state = CDCT_S141;
             else
                     state = BD_S0;
             break;
1360
     //////////////////////////////////////////////
     //        state CDCT_S141
     //////////////////////////////////////////////
     case CDCT_S141 :
1365         r_ram1 = DData[i];
             state = CDCT_S142;
             break;


1370 //////////////////////////////////////////////
     //        state CDCT_S142
     //////////////////////////////////////////////
     case CDCT_S142 :
             if(r_ram1<0)
1375                 state = CDCT_S146;
             else
                     state = CDCT_S147;
             break;

1380 //////////////////////////////////////////////
     //        state CDCT_S146
     //////////////////////////////////////////////
     case CDCT_S146 :
             r_alu = r_ram1-4;
1385         state = CDCT_S143;
             break;

     //////////////////////////////////////////////
     //        state CDCT_S147
1390 //////////////////////////////////////////////
     case CDCT_S147 :
             r_alu = r_ram1+4;
             state = CDCT_S143;
             break;
1395
     //////////////////////////////////////////////

     //        state CDCT_S143
     //////////////////////////////////////////////
```

```
1400            case CDCT_S143 :
                        r_div = r_alu/8;
                        state = CDCT_S144;
                        break;

1405            ////////////////////////////////////////////
                //      state CDCT_S144
                ////////////////////////////////////////////
                case CDCT_S144 :
                        DData[i]=r_div;
1410                    i = i+1;
                        state = CDCT_S140;
                        break;

                ////////////////////////////////////////////
1415            //      state BD_S0
                ////////////////////////////////////////////
                case BD_S0 :
                        i = 0;
                        state= BD_S1;
1420                    break;

                ////////////////////////////////////////////
                //      state BD_S1
                ////////////////////////////////////////////
1425            case BD_S1 :
                        if (i < 64)
                                state = BD_S2;
                        else
                                state = CDCT_S145;
1430                    break;

                ////////////////////////////////////////////
                //      state BD_S2
                ////////////////////////////////////////////
1435            case BD_S2 :
                        r_ram1 = DData[i];
                        state = BD_S3;
                        break;

1440            ////////////////////////////////////////////
                //      state BD_S3
                ////////////////////////////////////////////
                case BD_S3 :
                        if (r_ram1 < -1023)  state = BD_S4;
1445                    else state = BD_S5;
                        break;

                ////////////////////////////////////////////
                //      state BD_S4
1450            ////////////////////////////////////////////
                case BD_S4 :
                        DData[i] = -1023;
                        i = i+1;
                        state = BD_S1;
1455                    break;

                ////////////////////////////////////////////
                //      state BD_S5
                ////////////////////////////////////////////
1460            case BD_S5 :
                        if (r_ram1 > 1023)  state = BD_S6;
                        else state = BD_S7;
                        break;

1465            ////////////////////////////////////////////
```

```
                                    //        state BD_S6
                                    ////////////////////////////////////////////
                                    case BD_S6 :
                                            DData[i] = 1023;
1470                                        state = BD_S7;
                                            break;

                                    ////////////////////////////////////////////
                                    //        state BD_S7
1475                                ////////////////////////////////////////////
                                    case BD_S7 :
                                            i = i+1;
                                            state = BD_S1;
                                            break;
1480
                                    ////////////////////////////////////////////
                                    //        state CDCT_S145
                                    ////////////////////////////////////////////
                                    case CDCT_S145 :
1485                                        i = 0;
                                            state = Blck_End ;
                                            break;
                          }//end of switch
                          waitfor(4) ;
1490                 }//end of while
            }//end of main
    };//end of behavior
```