APPENDIX
VERILOG, MATLAB and PYTHON CODES

DESIGN AND IMPLEMENTATION OF A CUSTOM PROCESSOR OPTIMIZED FOR IMAGE PROCESSING

EN3030
Circuits and Systems Design

GROUP MEMBERS

150001C : G.Abarajithan
150172A : T.T.Fonseka
150689N : W.M.R.R.Wickramasinghe
150707V : C.Wimalasuriya

Contents

REPOSITORIES

github.com/BlazeCode2/ABRUTECH_processor_automatic

github.com/BlazeCode2/ABRUTECH_processor_manual

github.com/BlazeCode2/ABRUTECH_cache

github.com/Jester-2-6/ABRUTECH_graphic_equalizer

Chapter 1

Verilog Codes

1.1 System Modules

1.1.1 State Machine

```
1  //File name : state_machine.v
2  //This module controls the control signals according to the
3  //instructions fetched from IRAM.
4
5  `include "define.v"
6  `include "opcode_define.v"
7  module state_machine(
8        clock,
9        MIDR,
10       TOG,
11       ACI,
12       AWM,
13       MEM,
14       ALU,
15       PRM_param,
16       PRM,
17       OPR,
18       ADR,
19       PCI,
20       STATE,
21       start,
22       status
23       );
24
25 input       clock ,start;
26 input [7:0] MIDR;
27
28 output reg  [4:0] ACI=5'd0;
29 output reg  [3:0] PRM_param=4'd0,ADR=4'd0;
30 output reg  [2:0] AWM=3'd0,MEM=3'd0,OPR=3'd0,ALU=3'd0;
31 output reg  [1:0] PRM=2'd0;
32 output reg  PCI=0,TOG=0;
33 output reg  status=0;
34 output reg  [7:0] STATE = 8'd0;
35
```

```
36
37  always @(negedge clock)
38  begin
39    case(STATE)
40      `END:
41        begin
42          PCI  <=    0;
43          ACI  <=   `aci_none;
44          AWM  <=   `awm_AC  ;
45          MEM  <=   `mem_none;
46          ALU  <=   `alu_none;
47          PRM  <=   `prm_jmp;
48          OPR  <=   `opr_pc;
49          ADR  <=   `adr_none;
50          TOG  <=    0;
51          PRM_param  <=  `jmp_jump;
52          if(start)    //means negative edge of start button
53            begin
54              STATE <= `FETCH;
55              status<=1;
56            end
57          else  status<=0;
58        end
59      `FETCH:
60        begin
61          PCI  <=    0;
62          ACI  <=   `aci_none;
63          AWM  <=   `awm_AC  ;
64          MEM  <=   `mem_none;
65          ALU  <=   `alu_none;
66          PRM  <=   `prm_none;
67          OPR  <=   `opr_none;
68          ADR  <=   `adr_none;
69          TOG  <=    0;
70          STATE<=   `FETCH_2;
71        end
72      `FETCH_2:
73        begin
74          PCI  <=    1;
75          ACI  <=   `aci_none;
76          AWM  <=   `awm_AC  ;
77          MEM  <=   `mem_midr_m_ci;
78          ALU  <=   `alu_none;
79          PRM  <=   `prm_none;
80          OPR  <=   `opr_none;
81          ADR  <=   `adr_none;
82          TOG  <=    0;
83          STATE<=   `FETCH_3;
84        end
85      `FETCH_3:
86        begin
```

```
87          PCI  <=    0;
88          ACI  <=   `aci_none;
89          AWM  <=   `awm_AC  ;
90          MEM  <=   `mem_none;
91          ALU  <=   `alu_none;
92          PRM  <=   `prm_none;
93          OPR  <=   `opr_none;
94          ADR  <=   `adr_none;
95          TOG  <=    0;
96          STATE<=    {MIDR[7:4],4'd0};
97          PRM_param <= MIDR[3:0];
98        end
99
100     `LODK:
101       begin
102         PCI  <=    1;
103         ACI  <=   `aci_none;
104         AWM  <=   `awm_AC  ;
105         MEM  <=   `mem_midr_m_ci;
106         ALU  <=   `alu_none;
107         PRM  <=   `prm_none;
108         OPR  <=   `opr_none;
109         ADR  <=   `adr_none;
110         TOG  <=    0;
111         STATE<=   `LODK_2;
112       end
113     `LODK_2:
114       begin
115         PCI  <=    0;
116         ACI  <=   `aci_AC;
117         AWM  <=   `awm_MIDR;
118         MEM  <=   `mem_none;
119         ALU  <=   `alu_none;
120         PRM  <=   `prm_none;
121         OPR  <=   `opr_none;
122         ADR  <=   `adr_none;
123         TOG  <=    0;
124         STATE<=   `FETCH_2;
125       end
126     `LADD:
127       begin
128         PCI  <=    1;
129         ACI  <=   `aci_none;
130         AWM  <=   `awm_AC  ;
131         MEM  <=   `mem_midr_m_ci;
132         ALU  <=   `alu_none;
133         PRM  <=   `prm_none;
134         OPR  <=   `opr_none;
135         ADR  <=   `adr_none;
136         TOG  <=    0;
137         STATE<=   `LADD_2;
```

```
138         end
139
140       `LADD_2:
141         begin
142           PCI  <=    0;
143           ACI  <=   `aci_none;
144           AWM  <=   `awm_MIDR  ;
145           MEM  <=   `mem_none;
146           ALU  <=   `alu_none;
147           PRM  <=   `prm_none;
148           OPR  <=   `opr_none;
149           ADR  <=   `adr_first2;
150           TOG  <=    0;
151           STATE<=   `LADD_3;
152         end
153
154       `LADD_3:
155         begin
156           PCI  <=    1;
157           ACI  <=   `aci_none;
158           AWM  <=   `awm_AC  ;
159           MEM  <=   `mem_midr_m_ci;
160           ALU  <=   `alu_none;
161           PRM  <=   `prm_none;
162           OPR  <=   `opr_none;
163           ADR  <=   `adr_none;
164           TOG  <=    0;
165           STATE<=   `LADD_4;
166         end
167
168       `LADD_4:
169         begin
170           PCI  <=    0;
171           ACI  <=   `aci_none;
172           AWM  <=   `awm_MIDR  ;
173           MEM  <=   `mem_none;
174           ALU  <=   `alu_none;
175           PRM  <=   `prm_none;
176           OPR  <=   `opr_none;
177           ADR  <=   `adr_mid8;
178           TOG  <=    0;
179           STATE<=   `LADD_5;
180         end
181
182       `LADD_5:
183         begin
184           PCI  <=    1;
185           ACI  <=   `aci_none;
186           AWM  <=   `awm_AC  ;
187           MEM  <=   `mem_midr_m_ci;
188           ALU  <=   `alu_none;
```

```
189           PRM  <=   `prm_none;
190           OPR  <=   `opr_none;
191           ADR  <=   `adr_none;
192           TOG  <=    0;
193           STATE<=   `LADD_6;
194         end
195
196       `LADD_6:
197         begin
198           PCI  <=    0;
199           ACI  <=   `aci_none;
200           AWM  <=   `awm_MIDR  ;
201           MEM  <=   `mem_none;
202           ALU  <=   `alu_none;
203           PRM  <=   `prm_none;
204           OPR  <=   `opr_none;
205           ADR  <=   `adr_last8;
206           TOG  <=    0;
207           STATE<=   `LADD_7;
208         end
209
210       `LADD_7:
211         begin
212           PCI  <=    0;
213           ACI  <=   `aci_none;
214           AWM  <=   `awm_AC  ;
215           MEM  <=   `mem_none;
216           ALU  <=   `alu_none;
217           PRM  <=   `prm_adr;
218           OPR  <=   `opr_none;
219           ADR  <=   `adr_none;
220           TOG  <=    0;
221           STATE<=   `FETCH_2;
222         end
223       `LOAD:
224         begin
225           PCI  <=    0;
226           ACI  <=   `aci_none;
227           AWM  <=   `awm_AC  ;
228           MEM  <=   `mem_none;
229           ALU  <=   `alu_none;
230           PRM  <=   `prm_adr;
231           OPR  <=   `opr_none;
232             ADR  <=   `adr_none;
233           TOG  <=    0;
234           STATE<=   `LOAD_2;
235         end
236
237       `LOAD_2:
238         begin
239           PCI  <=    0;
```

```
240           ACI  <=   `aci_none;
241           AWM  <=   `awm_AC  ;
242           MEM  <=   `mem_none;
243           ALU  <=   `alu_none;
244           PRM  <=   `prm_none;
245           OPR  <=   `opr_none;
246           ADR  <=   `adr_none;
247           TOG  <=    0;
248           STATE<=   `LOAD_3;
249         end
250
251       `LOAD_3:
252         begin
253           PCI  <=    0;
254           ACI  <=   `aci_none;
255           AWM  <=   `awm_AC  ;
256           MEM  <=   `mem_mddr_m_ci;
257           ALU  <=   `alu_none;
258           PRM  <=   `prm_none;
259           OPR  <=   `opr_none;
260           ADR  <=   `adr_none;
261           TOG  <=    0;
262           STATE<=   `FETCH_2;
263         end
264
265       `STAC:
266         begin
267           PCI  <=    0;
268           ACI  <=   `aci_MDDR;
269           AWM  <=   `awm_AC  ;
270           MEM  <=   `mem_dm_write;
271           ALU  <=   `alu_none;
272           PRM  <=   `prm_adr;
273           OPR  <=   `opr_none;
274           ADR  <=   `adr_none;
275           TOG  <=    0;
276           STATE<=   `FETCH_2;
277         end
278       `COPY:
279         begin
280           PCI  <=    1;
281           ACI  <=   `aci_none;
282           AWM  <=   `awm_AC  ;
283           MEM  <=   `mem_midr_m_ci;
284           ALU  <=   `alu_none;
285           PRM  <=   `prm_none;
286           OPR  <=   `opr_none;
287           ADR  <=   `adr_none;
288           TOG  <=    0;
289           STATE<=   `COPY_2;
290         end
```

```
291       `COPY_2:
292         begin
293           PCI  <=    0;
294           ACI  <=   `aci_none;
295           AWM  <=   `awm_AC  ;
296           MEM  <=   `mem_none;
297           ALU  <=   `alu_none;
298           PRM  <=   `prm_none;
299           OPR  <=   `opr_aci_awm;
300           ADR  <=   `adr_none;
301           TOG  <=    0;
302           STATE<=   `FETCH_2;
303         end
304       `RSET:
305         begin
306           PCI  <=    1;
307           ACI  <=   `aci_none;
308           AWM  <=   `awm_AC  ;
309           MEM  <=   `mem_midr_m_ci;
310           ALU  <=   `alu_none;
311           PRM  <=   `prm_none;
312           OPR  <=   `opr_none;
313           ADR  <=   `adr_none;
314           TOG  <=    0;
315           STATE<=   `RSET_2;
316         end
317       `RSET_2:
318         begin
319           PCI  <=    0;
320           ACI  <=   `aci_none;
321           AWM  <=   `awm_AC  ;
322           MEM  <=   `mem_none;
323           ALU  <=   `alu_none;
324           PRM  <=   `prm_none;
325           OPR  <=   `opr_rst;
326           ADR  <=   `adr_none;
327           TOG  <=    0;
328           STATE<=   `FETCH_2;
329         end
330       `JUMP:
331         begin
332           PCI  <=    1;
333           ACI  <=   `aci_none;
334           AWM  <=   `awm_AC  ;
335           MEM  <=   `mem_midr_m_ci;
336           ALU  <=   `alu_none;
337           PRM  <=   `prm_none;
338           OPR  <=   `opr_none;
339           ADR  <=   `adr_none;
340           TOG  <=    0;
341           STATE<=   `JUMP_2;
```

```
342         end
343       `JUMP_2:
344         begin
345           PCI  <=    0;
346           ACI  <=   `aci_none;
347           AWM  <=   `awm_AC  ;
348           MEM  <=   `mem_none;
349           ALU  <=   `alu_none;
350           PRM  <=   `prm_jmp;
351           OPR  <=   `opr_pc;
352           ADR  <=   `adr_none;
353           TOG  <=    0;
354           STATE<=   `FETCH;
355         end
356       `INCR:
357         begin
358           PCI  <=    1;
359           ACI  <=   `aci_none;
360           AWM  <=   `awm_AC  ;
361           MEM  <=   `mem_midr_m_ci;
362           ALU  <=   `alu_none;
363           PRM  <=   `prm_none;
364           OPR  <=   `opr_none;
365           ADR  <=   `adr_none;
366           TOG  <=    0;
367           STATE<=   `INCR_2;
368         end
369
370       `INCR_2:
371         begin
372           PCI  <=    0;
373           ACI  <=   `aci_none;
374           AWM  <=   `awm_AC  ;
375           MEM  <=   `mem_none;
376           ALU  <=   `alu_none;
377           PRM  <=   `prm_none;
378           OPR  <=   `opr_inc;
379           ADR  <=   `adr_none;
380           TOG  <=    0;
381           STATE<=   `FETCH_2;
382         end
383
384       `DECR:
385         begin
386           PCI  <=    1;
387           ACI  <=   `aci_none;
388           AWM  <=   `awm_AC  ;
389           MEM  <=   `mem_midr_m_ci;
390           ALU  <=   `alu_none;
391           PRM  <=   `prm_none;
392           OPR  <=   `opr_none;
```

```
393           ADR  <=   `adr_none;
394           TOG  <=    0;
395           STATE<=   `DECR_2;
396         end
397
398         `DECR_2:
399           begin
400             PCI  <=    0;
401             ACI  <=   `aci_none;
402             AWM  <=   `awm_AC  ;
403             MEM  <=   `mem_none;
404             ALU  <=   `alu_none;
405             PRM  <=   `prm_none;
406             OPR  <=   `opr_dec;
407             ADR  <=   `adr_none;
408             TOG  <=    0;
409             STATE<=   `FETCH_2;
410           end
411
412         `ADD:
413           begin
414             PCI  <=    0;
415             ACI  <=   `aci_none;
416             AWM  <=   `awm_AC  ;
417             MEM  <=   `mem_none;
418             ALU  <=   `alu_add;
419             PRM  <=   `prm_add_sub;
420             OPR  <=   `opr_none;
421             ADR  <=   `adr_none;
422             TOG  <=    0;
423             STATE<=   `FETCH_2;
424           end
425
426         `SUBT:
427           begin
428             PCI  <=    0;
429             ACI  <=   `aci_none;
430             AWM  <=   `awm_AC  ;
431             MEM  <=   `mem_none;
432             ALU  <=   `alu_sub;
433             PRM  <=   `prm_add_sub;
434             OPR  <=   `opr_none;
435             ADR  <=   `adr_none;
436             TOG  <=    0;
437             STATE<=   `FETCH_2;
438           end
439
440         `DIV:
441           begin
442             PCI  <=    1;
443             ACI  <=   `aci_none;
```

```
444             AWM  <=   `awm_AC  ;
445             MEM  <=   `mem_midr_m_ci;
446             ALU  <=   `alu_none;
447             PRM  <=   `prm_none;
448             OPR  <=   `opr_none;
449             ADR  <=   `adr_none;
450             TOG  <=    0;
451             STATE<=   `DIV_2;
452           end
453         `DIV_2:
454           begin
455             PCI  <=    0;
456             ACI  <=   `aci_none;
457             AWM  <=   `awm_MIDR  ;
458             MEM  <=   `mem_none;
459             ALU  <=   `alu_div;
460             PRM  <=   `prm_none;
461             OPR  <=   `opr_none;
462             ADR  <=   `adr_none;
463             TOG  <=    0;
464             STATE<=   `FETCH_2;
465           end
466         `MUL:
467           begin
468             PCI  <=    1;
469             ACI  <=   `aci_none;
470             AWM  <=   `awm_AC  ;
471             MEM  <=   `mem_midr_m_ci;
472             ALU  <=   `alu_none;
473             PRM  <=   `prm_none;
474             OPR  <=   `opr_none;
475             ADR  <=   `adr_none;
476             TOG  <=    0;
477             STATE<=   `MUL_2;
478           end
479         `MUL_2:
480           begin
481             PCI  <=    0;
482             ACI  <=   `aci_none;
483             AWM  <=   `awm_MIDR  ;
484             MEM  <=   `mem_none;
485             ALU  <=   `alu_mul;
486             PRM  <=   `prm_none;
487             OPR  <=   `opr_none;
488             ADR  <=   `adr_none;
489             TOG  <=    0;
490             STATE<=   `FETCH_2;
491           end
492         `TOGL:
493           begin
494             PCI  <=    0;
```

```
495             ACI  <=   `aci_none;
496             AWM  <=   `awm_AC  ;
497             MEM  <=   `mem_none;
498             ALU  <=   `alu_none;
499             PRM  <=   `prm_none;
500             OPR  <=   `opr_none;
501             ADR  <=   `adr_none;
502             TOG  <=    1;
503             STATE<=   `FETCH_2;
504           end
505         `NOOP:
506           begin
507             PCI  <=    0;
508             ACI  <=   `aci_none;
509             AWM  <=   `awm_AC  ;
510             MEM  <=   `mem_none;
511             ALU  <=   `alu_none;
512             PRM  <=   `prm_none;
513             OPR  <=   `opr_none;
514             ADR  <=   `adr_none;
515             TOG  <=    0;
516             STATE<=   `FETCH_2;
517           end
518         default: STATE <= `END;
519     endcase
520   end
521
522   endmodule
```

1.1.2 Automatic Controller

```
//File name : Automatic_controller.v
//This module is responsible for switching between modes of operations
↪  automatically.

module Automatic_controller(
      clk,
      mode,
      ram_mode,
      p_start,
      tx_start,
      receive_status,
      processor_status,
      tx_status,
      reset,
      idle_mode
);

input clk,receive_status,processor_status,tx_status,reset,idle_mode;

output reg [1:0] mode = 2'b01;
output reg ram_mode   = 1'b1;
output reg tx_start   = 1'b0;
output reg p_start    = 1'b0;

parameter LOAD_INS = 3'b000;
parameter LOAD_DAT = 3'b001;
parameter PROCESS  = 3'b010;
parameter TRANSMIT = 3'b011;
parameter IDLE_INS = 3'b100;
parameter IDLE_DAT = 3'b101;

reg [2:0] STATE = 3'b000;

reg a = 1'b0;
reg b = 1'b0;
reg c = 1'b0;
reg d = 1'b0;
reg e = 1'b0;
reg f = 1'b0;
reg g = 1'b0;
reg h = 1'b0;
reg i = 1'b0;
reg j = 1'b0;

always @(negedge clk)
  case(STATE)
    IDLE_DAT:
      begin
        i        <= reset;
        j        <= i;
```

```
50          g        <= idle_mode;
51          h        <= g;
52          ram_mode <= 1'b0;
53          if(~g & h)
54            begin
55              STATE <= LOAD_DAT;
56              mode     <= 2'b01;
57            end
58          else if(~i & j)
59            begin
60              STATE    <= IDLE_INS;
61              p_start  <= 1'b0;
62              mode     <= 2'b00;
63            end
64        end
65      IDLE_INS:
66        begin
67          g        <= idle_mode;
68          h        <= g;
69          i        <= reset;
70          j        <= i;
71          ram_mode <= 1'b1;
72          if(~g & h)
73            begin
74              STATE <= LOAD_DAT;
75              mode     <= 2'b01;
76            end
77          else if(~i & j)
78            begin
79              STATE    <= IDLE_DAT;
80              p_start  <= 1'b0;
81              mode     <= 2'b00;
82            end
83        end
84      LOAD_INS:
85        begin
86          a        <= receive_status;
87          b        <= a;
88          c        <= 1'b0;
89          d        <= 1'b0;
90          e        <= 1'b0;
91          f        <= 1'b0;
92          mode     <= 2'b01;
93          ram_mode <= 1'b1;
94          tx_start <= 1'b0;
95          p_start  <= 1'b0;
96          if(a & ~b)
97            STATE <= LOAD_DAT;
98        end
99      LOAD_DAT:
100       begin
```

```
101         a         <= receive_status;
102         b         <= a;
103         c         <= 1'b0;
104         d         <= 1'b0;
105         e         <= 1'b0;
106         f         <= 1'b0;
107         g         <= idle_mode;
108         h         <= g;
109         ram_mode  <= 1'b0;
110         tx_start  <= 1'b0;
111         if(~reset)
112           STATE <= LOAD_INS;
113         else if(a & ~b)
114           begin
115             STATE    <= PROCESS;
116             p_start  <= 1'b1;
117             mode     <= 2'b10;
118           end
119         else if(~g & h)
120           begin
121             STATE    <= IDLE_DAT;
122             p_start  <= 1'b0;
123             mode     <= 2'b00;
124           end
125         else
126           begin
127             p_start  <= 1'b0;
128             mode     <= 2'b01;
129           end
130       end
131     PROCESS:
132       begin
133         c <= processor_status;
134         d <= c;
135         e         <= 1'b0;
136         f         <= 1'b0;
137         ram_mode  <= 1'b0;
138         tx_start  <= 1'b0;
139         p_start   <= 1'b0;
140         if (~c & d)
141           begin
142             STATE <= TRANSMIT;
143             tx_start <= 1'b1;
144             mode     <= 2'b11;
145           end
146         else
147           begin
148             tx_start <= 1'b0;
149             mode     <= 2'b10;
150           end
151       end
```

```
152       TRANSMIT:
153         begin
154           e <= tx_status;
155           f <= e;
156           c         <= 1'b0;
157           d         <= 1'b0;
158           ram_mode <= 1'b0;
159           tx_start <= 1'b0;
160           p_start  <= 1'b0;
161           if (e & ~f)
162             begin
163               STATE <= 2'b01;
164               mode     <= 2'b01;
165             end
166           else mode     <= 2'b11;
167         end
168       default:
169         begin
170           a         <= 1'b0;
171           b         <= 1'b0;
172           c         <= 1'b0;
173           d         <= 1'b0;
174           e         <= 1'b0;
175           f         <= 1'b0;
176           g         <= 1'b0;
177           h         <= 1'b0;
178           i         <= 1'b0;
179           j         <= 1'b0;
180           mode     <= 2'b01;
181           ram_mode <= 1'b1;
182           tx_start <= 1'b0;
183           p_start  <= 1'b0;
184           STATE    <= LOAD_INS;
185         end
186     endcase
187   endmodule
```

1.1.3 Processor Module

```
1  //File name : Processor.v
2  //This module is the heart of the project, the processor.
3  //Contains many general/special purpose registers and state machine.
4
5  module processor(
6              clock,
7              D_address,
8              D_din,
9              D_dout,
10             D_wen,
11             p_enable,
12             I_dout,
13             I_address,
14             STATE,
15             status,
16             ADR_to_Tx);
17
18 input          clock,p_enable;
19 input [7:0]    D_dout;
20 input [7:0]    I_dout;
21
22
23 output [17:0] D_address;
24 output [17:0] ADR_to_Tx;
25 output [7:0]  D_din;
26 output        D_wen;
27 output [7:0]  I_address;
28 output [7:0]  STATE;
29 output        status;
30
31
32 //wiring
33
34 wire [7:0] AC_to_AWM;
35 wire [7:0] K0_to_AWM;
36 wire [7:0] K1_to_AWM;
37 wire [7:0] G0_to_AWM;
38 wire [7:0] G1_to_AWM;
39 wire [7:0] G2_to_AWM;
40 wire [7:0] MDDR_out;
41 wire [7:0] MIDR_out;
42 wire [7:0] A_BUS;
43
44 wire [7:0] OPR_to_INC;
45 wire [7:0] OPR_to_DEC;
46 wire [7:0] OPR_to_ACI;
47 wire [7:0] OPR_to_AWM;
48 wire [7:0] OPR_to_din_PC;
49 wire [7:0] OPR_to_RST;
50
```

```
51  wire       SM_to_ADR_TOG;
52  wire [4:0] SM_to_ACI;
53  wire [2:0] SM_to_AWM;
54  wire [2:0] SM_to_MEM;
55  wire [2:0] SM_to_ALU;
56  wire [3:0] SM_to_PRM_param;
57  wire [1:0] SM_to_PRM_sel;
58  wire [2:0] SM_to_OPR;
59  wire [3:0] SM_to_ADR;
60  wire       SM_to_PC;
61
62  wire ACI_to_MDDR;
63  wire ACI_to_MDDR_Cin2;
64  wire ACI_to_K0_Cin;
65  wire ACI_to_K1_Cin;
66  wire ACI_to_G_SHF;
67  wire ACI_to_AC;
68
69  wire [3:0] PRM_to_ADR;
70  wire [3:0] PRM_to_JMP;
71  wire [2:0] PRM_to_AWM;
72
73  wire INC_to_ART;
74  wire INC_to_ARG;
75  wire INC_to_AWT;
76  wire INC_to_AWG;
77  wire INC_to_AC;
78  wire INC_to_K0;
79  wire INC_to_K1;
80
81
82  wire DEC_to_ART;
83  wire DEC_to_ARG;
84  wire DEC_to_AWT;
85  wire DEC_to_AWG;
86  wire DEC_to_AC;
87  wire DEC_to_K0;
88  wire DEC_to_K1;
89  wire AC_Z;
90
91
92  wire [8:0]  ART_to_ADR;
93  wire [8:0]  ARG_to_ADR;
94  wire [8:0]  AWT_to_ADR;
95  wire [8:0]  AWG_to_ADR;
96  wire        INC_to_ADR;
97  wire        DEC_to_ADR;
98  wire [17:0] ADR_to_DMEM;
99  wire        ADR_to_JMP_TOG;
100 wire        ADR_to_AWG_cin;
101 wire        ADR_to_AWT_cin;
```

```
102 wire         ADR_to_ARG_cin;
103 wire         ADR_to_ART_cin;
104 wire         ADR_to_ARG_ref_cin;
105 wire         ADR_to_ART_ref_cin;
106 wire [8:0]   ADR_to_ART_data;
107 wire [8:0]   ADR_to_ARG_data;
108 wire [8:0]   ADR_to_AWT_data;
109 wire [8:0]   ADR_to_AWG_data;
110
111 wire ARG_to_JMP_Z;
112 wire ART_to_JMP_Z;
113 wire K0_to_JMP_Z;
114 wire K1_to_JMP_Z;
115
116 wire        JMP_to_PC_Cin;
117 wire [7:0] PC_IMEM;
118
119 wire RST_to_AC;
120 wire RST_to_ADR;
121 wire RST_to_ART;
122 wire RST_to_ARG;
123 wire RST_to_AWT;
124 wire RST_to_AWG;
125 wire RST_to_K0;
126 wire RST_to_K1;
127
128 wire [7:0] IMEM_to_MIDR;
129 wire [7:0] DMEM_to_MDDR;
130
131
132 assign DMEM_to_MDDR=D_dout;
133 assign IMEM_to_MIDR=I_dout;
134 assign I_address=PC_IMEM;
135 assign D_address=ADR_to_DMEM;
136 assign D_wen=SM_to_MEM[2];
137 assign D_din=MDDR_out;
138
139 //STATE MACHINE
140
141 state_machine SM(
142       .clock(clock),
143       .MIDR(MIDR_out),
144       .TOG(SM_to_ADR_TOG),
145       .ACI(SM_to_ACI),
146       .AWM(SM_to_AWM),
147       .MEM(SM_to_MEM),
148       .ALU(SM_to_ALU),
149       .PRM_param(SM_to_PRM_param),
150       .PRM(SM_to_PRM_sel),
151       .OPR(SM_to_OPR),
152       .ADR(SM_to_ADR),
```

```
153         .PCI(SM_to_PC),
154         .STATE(STATE),
155         .start(p_enable),
156         .status(status));
157 
158 
159 //PRM ROUTER
160 
161 PRM prm_router(
162         .PARAM(SM_to_PRM_param),
163         .select(SM_to_PRM_sel),
164         .to_ADR(PRM_to_ADR),
165         .to_JMP(PRM_to_JMP),
166         .to_AWM(PRM_to_AWM));
167 
168 //ACI DECODER
169 
170 ACI_decoder ACI(
171         .A_sel(SM_to_ACI | OPR_to_ACI[4:0]),
172         .MDDR(ACI_to_MDDR_Cin2),
173         .K0(ACI_to_K0_Cin),
174         .K1(ACI_to_K1_Cin),
175         .G(ACI_to_G_SHF),
176         .AC(ACI_to_AC));
177 
178 //AWM MULTIPLEXER
179 
180 AWM_mux AWM_mux(
181         .data0x(AC_to_AWM[7:0]),     //AC
182         .data1x(MDDR_out),           //MDDR
183         .data2x(K0_to_AWM),       //K0
184         .data3x(K1_to_AWM),       //K1
185         .data4x(G0_to_AWM),       //G0
186         .data5x(G1_to_AWM),       //G1
187         .data6x(G2_to_AWM),       //G2
188         .data7x(MIDR_out),         //MIDR
189         .sel(OPR_to_AWM[2:0] | SM_to_AWM | PRM_to_AWM),
190         .result(A_BUS));           //A_Bus
191 
192 //INC DECODER
193 
194 INC_DEC_RST INC(       //instantiate increment
195         .I_sel(OPR_to_INC),
196         .ADR(INC_to_ADR),
197         .ART(INC_to_ART),
198         .ARG(INC_to_ARG),
199         .AWT(INC_to_AWT),
200         .AWG(INC_to_AWG),
201         .AC(INC_to_AC),
202         .K0(INC_to_K0),
203         .K1(INC_to_K1));
```

```
204
205
206 //ALU
207
208 ALU ALU(
209       .select(SM_to_ALU),
210       .A_bus(A_BUS),
211       .Z_out(AC_Z),            //Z
212       .AC(AC_to_AWM),          //AC
213       .cin_AC(ACI_to_AC),
214       .inc_AC(INC_to_AC),      //inc_AC
215       .dec_AC(DEC_to_AC),     //dec_AC
216       .clk(clock),
217       .rst(RST_to_AC));
218
219 //ADDRESS MAKER
220
221 ADR_maker ADR(
222        .AWREF(ADR_to_Tx),
223       .in_Clock(clock),
224       .A(A_BUS),
225       .ART(ART_to_ADR),
226       .ARG(ARG_to_ADR),
227       .AWT(AWT_to_ADR),
228       .AWG(AWG_to_ADR),
229       .inc(INC_to_ADR),
230       .dec(DEC_to_ADR),
231       .TOG_inc(SM_to_ADR_TOG),
232       .TOG(ADR_to_JMP_TOG),
233       .SEL(PRM_to_ADR | SM_to_ADR),
234       .reset(RST_to_ADR),
235       .d_out(ADR_to_DMEM),
236       .d_to_ART(ADR_to_ART_data),
237       .d_to_ARG(ADR_to_ARG_data),
238       .d_to_AWT(ADR_to_AWT_data),
239       .d_to_AWG(ADR_to_AWG_data),
240       .cin_ART(ADR_to_ART_cin),
241       .cin_ARG(ADR_to_ARG_cin),
242       .cin_AWT(ADR_to_AWT_cin),
243       .cin_AWG(ADR_to_AWG_cin),
244       .cin_ART_ref(ADR_to_ART_ref_cin),
245       .cin_ARG_ref(ADR_to_ARG_ref_cin));
246
247 //JUMP MULTIPLEXER
248
249 JMP_mux JMP(
250       .JMP_sel(PRM_to_JMP),
251       .AC_Z(AC_Z),
252       .ZT(ADR_to_JMP_TOG),
253       .ZRG(ARG_to_JMP_Z),
254       .ZRT(ART_to_JMP_Z),
```

```
255         .ZK0(K0_to_JMP_Z),
256         .ZK1(K1_to_JMP_Z),
257         .J(JMP_to_PC_Cin));
258 
259 //OPR
260 
261 OPR OPR(
262         .MIDR(MIDR_out),          //output of MIDR goes in here
263         .select(SM_to_OPR),       //control signals of OPR
264         .ACI(OPR_to_ACI),         //output MIDR value to ACI input
265         .AWM(OPR_to_AWM),         //output MIDR value to AWM input
266         .INC(OPR_to_INC),         //output MIDR value to INC input
267         .DEC(OPR_to_DEC),         //output MIDR value to DEC input
268         .din_PC(OPR_to_din_PC),      //output MIDR value to din_PC input
269         .RST(OPR_to_RST));         //output MIDR value to RST input
270 
271 //RESET DECODER
272 
273 INC_DEC_RST RST(
274         .I_sel(OPR_to_RST),
275         .ADR(RST_to_ADR),
276         .ART(RST_to_ART),
277         .ARG(RST_to_ARG),
278         .AWT(RST_to_AWT),
279         .AWG(RST_to_AWG),
280         .AC(RST_to_AC),
281         .K0(RST_to_K0),
282         .K1(RST_to_K1));
283 
284 //DECREMENT DECODER
285 
286 INC_DEC_RST DEC(
287         .I_sel(OPR_to_DEC),
288         .ADR(DEC_to_ADR),
289         .ART(DEC_to_ART),
290         .ARG(DEC_to_ARG),
291         .AWT(DEC_to_AWT),
292         .AWG(DEC_to_AWG),
293         .AC(DEC_to_AC),
294         .K0(DEC_to_K0),
295         .K1(DEC_to_K1));
296 
297 
298 
299 //registers
300 
301 //PC
302 
303 reg_PC PC(
304         .clk(clock),
305         .cin(JMP_to_PC_Cin),
```

```
306         .inc(SM_to_PC),
307         .d_in(OPR_to_din_PC),
308         .d_out(PC_IMEM));
309 
310 //MIDR
311 
312 reg_DATA MIDR(
313         .clk(clock),
314         .d_in1(8'd0),
315         .d_in2(IMEM_to_MIDR),
316         .c_in1(0),
317         .c_in2(SM_to_MEM[0]),
318         .d_out(MIDR_out));
319 
320 //MDDR
321 
322 reg_DATA MDDR(
323         .clk(clock),
324         .d_in1(DMEM_to_MDDR),
325         .d_in2(A_BUS),
326         .c_in1(SM_to_MEM[1]),
327         .c_in2(ACI_to_MDDR_Cin2),
328         .d_out(MDDR_out));
329 
330 //ART
331 
332 reg_ARG_ART ART(
333         .clk(clock),
334         .inc(INC_to_ART),
335         .dec(DEC_to_ART),
336         .reset(RST_to_ART),
337         .Z_OUT(ART_to_JMP_Z),         //ZRT
338         .d_out(ART_to_ADR),
339         .cin(ADR_to_ART_cin),
340         .cin_ref(ADR_to_ART_ref_cin),
341         .d_from_ADR(ADR_to_ART_data));
342 
343 //ARG
344 
345 reg_ARG_ART ARG(
346         .clk(clock),
347         .inc(INC_to_ARG),
348         .dec(DEC_to_ARG),
349         .reset(RST_to_ARG),
350         .Z_OUT(ARG_to_JMP_Z),       //ZRG
351         .d_out(ARG_to_ADR),
352         .cin(ADR_to_ARG_cin),
353         .cin_ref(ADR_to_ARG_ref_cin),
354         .d_from_ADR(ADR_to_ARG_data));
355 
356 //AWT
```

```
357
358 reg_AWG_AWT AWT(
359       .clk(clock),
360       .inc(INC_to_AWT),           //inc_AWT
361       .dec(DEC_to_AWT),            //dec_AWT
362       .reset(RST_to_AWT),
363       .d_out(AWT_to_ADR),          //AWT out
364       .d_from_ADR(ADR_to_AWT_data),
365       .cin(ADR_to_AWT_cin));
366
367 //AWG
368
369 reg_AWG_AWT AWG(
370       .clk(clock),
371       .inc(INC_to_AWG),             //inc_AWG
372       .dec(DEC_to_AWG),          //dec_AWG
373       .reset(RST_to_AWG),
374       .d_out(AWG_to_ADR),          //AWG out
375       .d_from_ADR(ADR_to_AWG_data),
376       .cin(ADR_to_AWG_cin));
377
378 //K0
379
380 reg_K K0(
381         .din(A_BUS),
382       .dout(K0_to_AWM),
383       .clk(clock),
384       .write(ACI_to_K0_Cin),
385       .rst(RST_to_K0),
386       .inc(INC_to_K0),
387       .dec(DEC_to_K0),
388       .k_Z(K0_to_JMP_Z));
389
390 //K1
391
392 reg_K K1(
393         .din(A_BUS),
394       .dout(K1_to_AWM),
395       .clk(clock),
396       .write(ACI_to_K1_Cin),
397       .rst(RST_to_K1),
398       .inc(INC_to_K1),
399       .dec(DEC_to_K1),
400       .k_Z(K1_to_JMP_Z));
401
402 //G
403
404 reg_G G(
405         .din(A_BUS),
406       .G0_out(G0_to_AWM),
407       .G1_out(G1_to_AWM),
```

```
408           .G2_out(G2_to_AWM),
409         .clk(clock),
410         .shift(ACI_to_G_SHF));
411
412 endmodule
```

1.1.4 Top Level Module (System)

```
1  //File name : top_level_module.v
2  //This module is used to gather all of the sub modules into one project
   ↪  module.
3
4  module
   ↪  top_level_module(hex7,hex6,hex5,hex4,hex3,hex2,hex1,hex0,user_addr_control,
   ↪  write_done,
5                 in_Clock, rx_serial,
6                 manual_clk,clk_mode, tx_serial, retrieve_done, tx_active,
7                 processor_status,iram_output,
8                 test_wire,
9                 reset_processor,idle_button
10                );
11 input idle_button;
12 input reset_processor;         //Reset Processor to load intructions again
13 input in_Clock;                //50Mhz clock(original clock)
14 input rx_serial;               //receiving serial line
15 input manual_clk;              //key0    manual clock button
16 input clk_mode;                //key1    clock mode selection button to select
   ↪  10MHz|1Hz|manual|25MHz
17 input [17:0]user_addr_control; //address bus to 18 switches
18
19 output write_done;             //LEDR0 to indicate Memory storage from Rx is
   ↪  done
20 output tx_serial;              //transmission serial line
21 output retrieve_done;        //LEDR2 to indicate transmission from Tx is done
22 output tx_active;          //LEDR1 to indicate Tx status(lit means
   ↪  transmitting)
23 output processor_status;      //LEDG7 indicates processor status(busy or not)
24 output [6:0]hex7;          //seven segment display(SSD) indicates current mode
   ↪  of operation
25 output [6:0]hex6;           //seven segment display(SSD) indicates Selected
   ↪  ram D_ram|I_ram
26 output [6:0]hex5;           //2 (SSDs) indicates Current state of state
   ↪  machine
27 output [6:0]hex4;
28 output [6:0]hex3;           //seven segment display(SSD) indicates current
   ↪  clock speed when processing 10MHz|1Hz|manual|25MHz
29 output [6:0]hex2;           //3 SSDs indicates the current dram memory output
30 output [6:0]hex1;
31 output [6:0]hex0;
32 output [7:0]iram_output;       //also wired to RLED17-RLED10
33 output [6:0]test_wire;       //GLED3-GLED0 => 4LSB of AWT | GLED6-GLED4 =>
   ↪  3LSB of AWG
34
35
36
37 //processor related wires
38
39 wire        processor_en;    //processor starts when this is low
```

```
40  wire [7:0]  SM_STATE;         //Current state of the state machine (goes to 2
    ↪  SSDs)
41  wire [17:0] ADR_p_Tx;         //bus sending the address, the transmitter use
    ↪  as reference
42
43  //Tx related wires
44
45  wire        tx_tick_wire;
46  wire        retrieve_enable;     //to activate the Tx through data retreiver
47  wire[17:0] retriver_EXSM_addr;
48  wire        retrieve_image;      //wire giving signal to start transmitting
    ↪  image to pc
49
50  //Rx related wires
51
52  wire         rx_tick_wire ;
53  wire [7:0]  uart_to_writer_data;//wire connecting Rx byte and writer data_in
54  wire [7:0]  writer_to_ram_data; //wire connecting D_ram_din and writer data_in
55  wire [17:0] writer_to_ram_addr; //wire carrying the address to be written , to
    ↪  D_ram
56
57  //Debounced button outputs
58
59  wire        manual_clk_debounced;
60  wire        clk_mode_debounced;
61  wire        reset_processor_debounced;
62  wire        idle_button_debounced;
63
64  //Dram related wires
65
66  wire        ram_write_enable;  //wire connecting EXSM and write enable of
    ↪  D_ram
67  wire [17:0] dram_address_bus;  //wire connecting EXSM and address bus of D_ram
68  wire [7:0]  dram_EXSM_din;     //wire connecting EXSM and din of D_ram
69  wire [7:0]  dram_dout;         //wire connected to dout of D_ram
70
71  //Iram related wires
72
73  wire [7:0]  iram_p_dout;       //I_ram dout connects only to processor
74  wire [7:0]  iram_EXSM_din;     //wire connecting EXSM and din of D_ram
75  wire [7:0]  iram_address_bus;  //wire connecting EXSM and address bus of I_ram
76  wire        iram_write_enable; //wire connecting EXSM and write enable of
    ↪  I_ram
77
78  //Memory router related wires
79
80  wire        writer_EXSM_wen;   //For writer to access wen of Dram/Iram
81  wire        p_EXSM_wen;        //For processor to access wen of Dram
82  wire [7:0]  p_EXSM_din;        //For processor to access din of Dram
83  wire [17:0] p_EXSM_address;    //For processor to access address of Dram
84  wire [7:0]  p_EXSM_ins_address;//For processor to access address of Iram
```

```
85  wire [17:0] mem_size;          //wire sending the size of memory locations to
    ↪  be written by writer 255 or 262143
86
87
88  //clock related wires
89
90  wire      clk;                 //this wire goes to everything
91  reg [1:0] CLOCK_MODE=2'd0;     //state of this decide the clock given when
    ↪  processing 10MHz|1Hz|manual|25MHz
92
93  //mode of operations and ram selection
94
95  wire [1:0] STATE;              //state of this decide the mode of operation
    ↪  IDLE|Rx|Process|Tx
96  wire       DATA_INS;           //state of this decide the Ram to be used when
    ↪  writing Dram|Iram
97  wire[6:0] hex;                 //erase (not used(a dummy wire))
98
99
100
101
102 //SSD to display current mode of operation/Ram using/clock mode
103
104 decoder mode_out(
105       .din({2'd0,STATE}),
106       .dout(hex7));
107
108 decoder d_or_i_disp(
109       .din({3'd0,DATA_INS}),
110       .dout(hex6));
111
112 decoder clk_mode_disp(
113       .din({2'd0,CLOCK_MODE}),
114       .dout(hex3));
115
116 //processor instantiation
117
118
119 processor Processor(
120       .clock(clk),
121       .D_address(p_EXSM_address),
122       .D_din(p_EXSM_din),
123       .D_dout(dram_dout),
124       .D_wen(p_EXSM_wen),
125       .p_enable(processor_en),
126       .I_dout(iram_p_dout),
127       .I_address(p_EXSM_ins_address),
128       .status(processor_status),//processor busy or not
129       .STATE(SM_STATE),        //current state of state machine
130       .ADR_to_Tx(ADR_p_Tx));   //reference address to Tx
131
```

```
132
133 //Module connecting Rx and Memory
134
135 Data_writer writer(
136       .Rx_tick(rx_tick_wire),
137       .Din(uart_to_writer_data),
138       .Dout(writer_to_ram_data),
139       .Addr(writer_to_ram_addr),
140       .fin(write_done),
141       .clk(clk),
142       .Wen(writer_EXSM_wen),
143       .memory_size(mem_size));
144
145 //Uart receiver
146
147 uart_rx reciever(
148       .clk(clk),
149       .i_Rx_Serial(rx_serial),
150       .o_Rx_DV(rx_tick_wire),
151       .o_Rx_Byte(uart_to_writer_data));
152
153 //Module to convert 8 bit number to 3 digit decimal number
154
155 bi2bcd bcd(
156       .din(dram_dout),
157       .dout2(hex2),
158       .dout1(hex1),
159       .dout0(hex0));
160
161 bi2bcd SM_st(
162       .din(SM_STATE),
163       .dout2(hex),
164       .dout1(hex5),
165       .dout0(hex4));
166
167
168 //18bit address Dram   ** here it's instantiated to respond to neg edge on clk
    ↪  **
169
170 dram_512 data_ram(
171       .data(dram_EXSM_din),
172       .address(dram_address_bus),
173       .q(dram_dout),
174       .clock(~clk),            //made ram negative edge sensitive
175       .wren(ram_write_enable));
176
177 //8bit address Iram   ** here it's instantiated to respond to neg edge on clk
    ↪  **
178
179 iram IRAM(
180       .address(iram_address_bus),
```

```
181         .clock(~clk),           //made ram negative edge sensitive
182         .data(iram_EXSM_din),
183         .wren(iram_write_enable),
184         .q(iram_p_dout));
185 
186 //Module connecting Memory and uart Tx
187 
188 Data_retriever retriever(
189         .start(retrieve_image), //start transmition
190         .Tx_tick_from_tx(tx_tick_wire),
191         .addr(retriver_EXSM_addr),
192         .fin(retrieve_done),    //to indicate transmission done
193         .wen_Tx(retrieve_enable),
194         .clk(clk),
195         .end_add(ADR_p_Tx));    //reference address to transmit
196 
197 //Uart Tx
198 
199 uart_tx transmitter(
200         .i_Clock(clk),
201         .i_Tx_DV(retrieve_enable),
202         .i_Tx_Byte(dram_dout),
203         .o_Tx_Serial(tx_serial),
204         .o_Tx_Done(tx_tick_wire),
205         .o_Tx_Active(tx_active));
206 
207 
208 //Memory router to map memory ports to required users in different modes of
    ↪  operations
209 
210 memory_router EXSM(
211         .wen(ram_write_enable),.address(dram_address_bus),.din(dram_EXSM_din),
        ↪  //Dram related ports
212         .i_wen(iram_write_enable),.i_add(iram_address_bus),.i_din(iram_EXSM_din),
        ↪  //Iram related ports
213         .p_wen(p_EXSM_wen),.p_address(p_EXSM_address),.p_din(p_EXSM_din),.p_ins_address(p_EXSM
        ↪  //processor related ports
214         .memory_size_select(mem_size),.rx_wen(writer_EXSM_wen),.rx_address(writer_to_ram_addr)
        ↪  //Rx related ports
215         .tx_address(retriver_EXSM_addr),
        ↪  //Tx related ports
216         .user_address(user_addr_control),
        ↪  //user access related ports
217         .mode(STATE),.d_or_i(DATA_INS) );
        ↪  //mode of operation/ram select related ports
218 
219 //Automatic_controller
220 
221 Automatic_controller Auto(
222         .clk(clk),
223         .mode(STATE),
```

```
224         .ram_mode(DATA_INS),
225         .p_start(processor_en),
226         .tx_start(retrieve_image),
227         .receive_status(write_done),
228         .processor_status(processor_status),
229         .tx_status(retrieve_done),
230         .reset(reset_processor_debounced),
231         .idle_mode(idle_button_debounced));
232 
233 
234 //clock mux
235 
236 clock_control clk_cntrl(
237         .in_clock(in_Clock),    //50MHz clock in
238         .sel(CLOCK_MODE),
239         .mode(STATE),
240         .manual(manual_clk_debounced), //debounced manual clock wire in
241         .out_clock(clk)         //10MHz,1Hz,Manual,25MHz
242         );
243 
244 
245 //Debouncing the push buttons (uses 50MHz clock)
246 
247 debouncer   idler(
248         .button_in(idle_button),
249         .button_out(idle_button_debounced),
250         .clk(in_Clock));
251 
252 debouncer   rst(
253         .button_in(reset_processor),
254         .button_out(reset_processor_debounced),
255         .clk(in_Clock));
256 
257 debouncer   clock_mode_button(
258         .button_in(clk_mode),
259         .button_out(clk_mode_debounced),
260         .clk(in_Clock));
261 
262 debouncer   manual_clock(
263         .button_in(manual_clk),
264         .button_out(manual_clk_debounced),
265         .clk(in_Clock));
266 
267 
268 
269 
270 assign iram_output=iram_p_dout;
271 assign test_wire[6:4]=ADR_p_Tx[11:9];
272 assign test_wire[3:0]=ADR_p_Tx[3:0];
273 
274 
```

```
275
276 //update clock mode per button pressed in correct mode(any(but affects only in
    ↪  processor mode))
277
278 always @(negedge clk_mode_debounced)
279     begin
280       CLOCK_MODE<=CLOCK_MODE+1;
281     end
282
283 endmodule
```

1.2 Controllers: MUX, DEMUX, Decoders, Routers

1.2.1 OPR: Operand Router

```
1  //File name : OPR.v
2  //This module operands to required modules.
3
4  `include "define.v"
5
6  module OPR(MIDR,select,ACI,AWM,INC,DEC,din_PC,RST);
7
8  input      [7:0] MIDR;
9  input      [2:0] select;
10 output reg [7:0] ACI=8'd0;
11 output reg [7:0] AWM=8'd0;
12 output reg [7:0] INC=8'd0;
13 output reg [7:0] DEC=8'd0;
14 output reg [7:0] din_PC=8'd0;
15 output reg [7:0] RST=8'd0;
16
17 always @(*)
18   begin
19     case(select)
20       `opr_none:
21          begin
22            ACI    <= 8'd0;
23            AWM    <= 8'd0;
24            INC    <= 8'd0;
25            DEC    <= 8'd0;
26            din_PC <= 8'd0;
27            RST    <= 8'd0;
28          end
29       `opr_awm:
30          begin
31            ACI    <= 8'd0;
32            AWM    <= MIDR;
33            INC    <= 8'd0;
34         DEC    <= 8'd0;
35            din_PC <= 8'd0;
36            RST    <= 8'd0;
37          end
38       `opr_aci_awm:
39          begin
40            ACI    <= {3'd0,MIDR[4:0]};
41            AWM    <= {5'd0,MIDR[7:5]};
42            INC    <= 8'd0;
43         DEC    <= 8'd0;
44            din_PC <= 8'd0;
45            RST    <= 8'd0;
46          end
47       `opr_inc:
48          begin
```

```
49              ACI    <= 8'd0;
50              AWM    <= 8'd0;
51              INC    <= MIDR;
52            DEC    <= 8'd0;
53              din_PC <= 8'd0;
54              RST    <= 8'd0;
55            end
56         `opr_dec:
57            begin
58              ACI    <= 8'd0;
59              AWM    <= 8'd0;
60              INC    <= 8'd0;
61            DEC    <= MIDR;
62              din_PC <= 8'd0;
63              RST    <= 8'd0;
64            end
65         `opr_pc:
66            begin
67              ACI    <= 8'd0;
68              AWM    <= 8'd0;
69              INC    <= 8'd0;
70            DEC    <= 8'd0;
71              din_PC <= MIDR;
72              RST    <= 8'd0;
73            end
74         `opr_rst:
75            begin
76              ACI    <= 8'd0;
77              AWM    <= 8'd0;
78              INC    <= 8'd0;
79            DEC    <= 8'd0;
80              din_PC <= 8'd0;
81              RST    <= MIDR;
82            end
83         default:
84            begin
85              ACI    <= 8'd0;
86              AWM    <= 8'd0;
87              INC    <= 8'd0;
88            DEC    <= 8'd0;
89              din_PC <= 8'd0;
90              RST    <= 8'd0;
91            end
92       endcase
93     end
94
95  endmodule
```

1.2.2 PRM: Parameter Router

```
//File name : PRM.v
//This module routes the parameter portion of the opcode to required modules.

`include "define.v"

module PRM(
    PARAM,
    select,
    to_ADR,
    to_JMP,
    to_AWM);

input      [3:0] PARAM;
input      [1:0] select;
output reg [3:0] to_ADR=4'd0;
output reg [3:0] to_JMP=4'd0;
output reg [2:0] to_AWM=2'd0;

always @(*)
  begin
    case(select)
      `prm_none:
          begin
            to_ADR   <= 4'd0;
            to_JMP   <= 4'd0;
            to_AWM   <= 3'd0;
          end
    `prm_adr:
          begin
            to_ADR   <= PARAM;
            to_JMP   <= 4'd0;
            to_AWM   <= 3'd0;
          end
    `prm_jmp:
          begin
            to_ADR   <= 4'd0;
            to_JMP   <= PARAM;
            to_AWM   <= 3'd0;
          end
    `prm_add_sub:
          begin
            to_ADR   <= 4'd0;
            to_JMP   <= 4'd0;
            to_AWM   <= PARAM[2:0];
          end
       default:
          begin
            to_ADR   <= 4'd0;
            to_JMP   <= 4'd0;
            to_AWM   <= 3'd0;
```

```
51              end
52        endcase
53      end
54
55  endmodule
```

1.2.3 ACI Decoder

```
1  //File name : ACI_decoder.v
2  //This module controls the Cin of MDDR,K0,K1,G,AC registers.
3
4  //A bus to registers write => control instructions
5  module ACI_decoder(A_sel,MDDR,K0,K1,G,AC);
6
7  input [4:0] A_sel;
8
9  output   MDDR,K0,K1,G,AC;
10
11 assign AC   = A_sel[0];
12 assign MDDR = A_sel[1];
13 assign K0   = A_sel[2];
14 assign K1   = A_sel[3];
15 assign G    = A_sel[4];
16
17
18 endmodule
```

1.2.4 AWM Mux

```
1  //File name : AWM_mux.v
2  //This module multiplexes the output of registers
3  //AC,MDDR,K0,K1,G0,G1,G2,MIDR onto the A bus.
4
5
6  // megafunction wizard: %LPM_MUX%
7  // GENERATION: STANDARD
8  // VERSION: WM1.0
9  // MODULE: LPM_MUX
10
11 // ============================================================
12 // File Name: AWM_mux.v
13 // Megafunction Name(s):
14 //       LPM_MUX
15 //
16 // Simulation Library Files(s):
17 //       lpm
18 // ============================================================
19 // ************************************************************
20 // THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
21 //
22 // 15.0.0 Build 145 04/22/2015 SJ Full Version
23 // ************************************************************
24
25
26 //Copyright (C) 1991-2015 Altera Corporation. All rights reserved.
27 //Your use of Altera Corporation's design tools, logic functions
28 //and other software and tools, and its AMPP partner logic
29 //functions, and any output files from any of the foregoing
30 //(including device programming or simulation files), and any
31 //associated documentation or information are expressly subject
32 //to the terms and conditions of the Altera Program License
33 //Subscription Agreement, the Altera Quartus II License Agreement,
34 //the Altera MegaCore Function License Agreement, or other
35 //applicable license agreement, including, without limitation,
36 //that your use is for the sole purpose of programming logic
37 //devices manufactured by Altera and sold by Altera or its
38 //authorized distributors.  Please refer to the applicable
39 //agreement for further details.
40
41
42 // synopsys translate_off
43 `timescale 1 ps / 1 ps
44 // synopsys translate_on
45 module AWM_mux (
46   data0x,        //AC
47   data1x,       //MDDR
48   data2x,      //K0
49   data3x,      //K1
50   data4x,      //G0
```

```
51    data5x,       //G1
52    data6x,       //G2
53    data7x,       //MIDR
54    sel,
55    result);    //A_Bus
56
57    input  [7:0]  data0x;
58    input  [7:0]  data1x;
59    input  [7:0]  data2x;
60    input  [7:0]  data3x;
61    input  [7:0]  data4x;
62    input  [7:0]  data5x;
63    input  [7:0]  data6x;
64    input  [7:0]  data7x;
65    input  [2:0]  sel;
66    output  [7:0]  result;
67
68    wire [7:0] sub_wire9;
69    wire [7:0] sub_wire8 = data7x[7:0];
70    wire [7:0] sub_wire7 = data6x[7:0];
71    wire [7:0] sub_wire6 = data5x[7:0];
72    wire [7:0] sub_wire5 = data4x[7:0];
73    wire [7:0] sub_wire4 = data3x[7:0];
74    wire [7:0] sub_wire3 = data2x[7:0];
75    wire [7:0] sub_wire2 = data1x[7:0];
76    wire [7:0] sub_wire0 = data0x[7:0];
77    wire [63:0] sub_wire1 = {sub_wire8, sub_wire7, sub_wire6, sub_wire5,
      ↪  sub_wire4, sub_wire3, sub_wire2, sub_wire0};
78    wire [7:0] result = sub_wire9[7:0];
79
80    lpm_mux  LPM_MUX_component (
81          .data (sub_wire1),
82          .sel (sel),
83          .result (sub_wire9)
84          // synopsys translate_off
85          ,
86          .aclr (),
87          .clken (),
88          .clock ()
89          // synopsys translate_on
90          );
91    defparam
92      LPM_MUX_component.lpm_size = 8,
93      LPM_MUX_component.lpm_type = "LPM_MUX",
94      LPM_MUX_component.lpm_width = 8,
95      LPM_MUX_component.lpm_widths = 3;
96
97
98  endmodule
99
100 // ============================================================
```

```
101 // CNX file retrieval info
102 // ============================================================
103 // Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone IV E"
104 // Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
105 // Retrieval info: PRIVATE: new_diagram STRING "1"
106 // Retrieval info: LIBRARY: lpm lpm.lpm_components.all
107 // Retrieval info: CONSTANT: LPM_SIZE NUMERIC "8"
108 // Retrieval info: CONSTANT: LPM_TYPE STRING "LPM_MUX"
109 // Retrieval info: CONSTANT: LPM_WIDTH NUMERIC "8"
110 // Retrieval info: CONSTANT: LPM_WIDTHS NUMERIC "3"
111 // Retrieval info: USED_PORT: data0x 0 0 8 0 INPUT NODEFVAL "data0x[7..0]"
112 // Retrieval info: USED_PORT: data1x 0 0 8 0 INPUT NODEFVAL "data1x[7..0]"
113 // Retrieval info: USED_PORT: data2x 0 0 8 0 INPUT NODEFVAL "data2x[7..0]"
114 // Retrieval info: USED_PORT: data3x 0 0 8 0 INPUT NODEFVAL "data3x[7..0]"
115 // Retrieval info: USED_PORT: data4x 0 0 8 0 INPUT NODEFVAL "data4x[7..0]"
116 // Retrieval info: USED_PORT: data5x 0 0 8 0 INPUT NODEFVAL "data5x[7..0]"
117 // Retrieval info: USED_PORT: data6x 0 0 8 0 INPUT NODEFVAL "data6x[7..0]"
118 // Retrieval info: USED_PORT: data7x 0 0 8 0 INPUT NODEFVAL "data7x[7..0]"
119 // Retrieval info: USED_PORT: result 0 0 8 0 OUTPUT NODEFVAL "result[7..0]"
120 // Retrieval info: USED_PORT: sel 0 0 3 0 INPUT NODEFVAL "sel[2..0]"
121 // Retrieval info: CONNECT: @data 0 0 8 0 data0x 0 0 8 0
122 // Retrieval info: CONNECT: @data 0 0 8 8 data1x 0 0 8 0
123 // Retrieval info: CONNECT: @data 0 0 8 16 data2x 0 0 8 0
124 // Retrieval info: CONNECT: @data 0 0 8 24 data3x 0 0 8 0
125 // Retrieval info: CONNECT: @data 0 0 8 32 data4x 0 0 8 0
126 // Retrieval info: CONNECT: @data 0 0 8 40 data5x 0 0 8 0
127 // Retrieval info: CONNECT: @data 0 0 8 48 data6x 0 0 8 0
128 // Retrieval info: CONNECT: @data 0 0 8 56 data7x 0 0 8 0
129 // Retrieval info: CONNECT: @sel 0 0 3 0 sel 0 0 3 0
130 // Retrieval info: CONNECT: result 0 0 8 0 @result 0 0 8 0
131 // Retrieval info: GEN_FILE: TYPE_NORMAL AWM_mux.v TRUE
132 // Retrieval info: GEN_FILE: TYPE_NORMAL AWM_mux.inc FALSE
133 // Retrieval info: GEN_FILE: TYPE_NORMAL AWM_mux.cmp FALSE
134 // Retrieval info: GEN_FILE: TYPE_NORMAL AWM_mux.bsf FALSE
135 // Retrieval info: GEN_FILE: TYPE_NORMAL AWM_mux_inst.v FALSE
136 // Retrieval info: GEN_FILE: TYPE_NORMAL AWM_mux_bb.v TRUE
137 // Retrieval info: LIB_FILE: lpm
```

1.2.5 INC, DEC, RST Decoders

```
1  //File name : INC_DEC_RST.v
2  //This module used to give control signals to the INCrement,DECrement and
   ↪  ReSeT pins
3  //of registers MDAR,ART,ARG,AWT,AWG,AC,K0,K1 in 3 different module
   ↪  instantiations respectively.
4
5  // [INC_decoder] inc signals for registers
6  module INC_DEC_RST(I_sel,ADR,ART,ARG,AWT,AWG,AC,K0,K1);
7
8  input [7:0] I_sel;
9  output ADR,ART,ARG,AWT,AWG,AC,K0,K1;
10
11 assign ADR = I_sel[0];
12 assign ART = I_sel[1];
13 assign ARG = I_sel[2];
14 assign AWT = I_sel[3];
15 assign AWG = I_sel[4];
16 assign AC  = I_sel[5];
17 assign K0  = I_sel[6];
18 assign K1  = I_sel[7];
19
20 endmodule
```

1.3 Registers

1.3.1 Shift Register Bank

```
1  //File name : reg_G.v
2  //This module is the collection of 3 registers to form a shift register.
3
4  module reg_G (din, G0_out, G1_out,
5                G2_out, clk, shift);
6
7  input clk, shift;
8  input[7:0] din;
9  output reg [7:0] G0_out = 8'd0, G1_out = 8'd0, G2_out = 8'd0;
10
11
12 always @(posedge clk)
13   begin
14     if (shift == 1'b1)
15      begin
16       G2_out <= G1_out;
17       G1_out <= G0_out;
18       G0_out <= din;
19      end
20   end
21
22 endmodule // reg_G0
```

1.3.2 Loop Registers

```
//File name : reg_K.v
//This module is used to instantiate K0,K1 the loop registers.
//can be used as general purpose registers as well.

module reg_K (din, dout, clk, write,rst,inc,dec,k_Z);

input clk, write, rst, inc, dec;
input [7:0] din;
output reg [7:0] dout = 8'd0;
output reg k_Z=1;

reg [7:0] k_ref = 8'd0;
reg tog=0;

always @(dout,k_ref)
  begin
    if(k_ref == dout) k_Z <=1;
    else              k_Z <=0;
  end

always @ ( posedge clk )
  begin
    if (rst ==1 )
      begin
        tog   <= 0;
        dout  <= 8'd0;
        k_ref <= 8'd0;
      end
    else if (write == 1)
      begin
        dout <= din;
        if ( tog == 0)
          begin
            k_ref <= din;
            tog   <= 1;
          end
      end
    else if (inc == 1) dout <= dout + 1;
    else if (dec == 1) dout <= dout - 1;
  end

endmodule // reg_Kdin, dout, clk, writeinput clk, write;
```

1.3.3 AR Registers

```
1  //File name : reg_ARG_ART.v
2  //This module used to instantiate AR registers which manipulate the address
   ↪  used to
3  //read from DRAM.
4
5  module reg_ARG_ART (clk, inc, dec, reset, Z_OUT, d_out, cin,
   ↪  cin_ref,d_from_ADR);
6
7  input       clk, inc, dec, reset, cin, cin_ref;
8  input [8:0] d_from_ADR;
9
10 output reg       Z_OUT = 1;
11 output reg [8:0] d_out = 9'd0;
12
13 reg [8:0] ref=9'd0;
14
15 always @ (posedge clk)
16   begin
17      if(inc)         d_out <= d_out+1;
18      else if(dec)    d_out <= d_out-1;
19      else if(reset)  d_out <= 9'd0;
20      else if (cin) d_out <= d_from_ADR;
21      else if (cin_ref) ref <= d_from_ADR;
22   end
23
24 always @ (d_out)
25   begin
26     if(d_out == ref)   Z_OUT <= 1;
27     else               Z_OUT <= 0;
28   end
29
30 endmodule
```

1.3.4 AW Registers

```
//File name : reg_AWG_AWT.v
//This module is used to instantiate AW registers which are used to manipulate
//address used to write to MDDR.

module reg_AWG_AWT (clk, inc, dec, reset, d_out,d_from_ADR,cin);

input clk, inc, dec, reset,cin;
input [8:0] d_from_ADR;
output reg [8:0] d_out = 9'd0;

always @ (posedge clk)
begin

    if(inc)          d_out <= d_out+1;
    else if(dec)     d_out <= d_out-1;
    else if(reset)   d_out <= 9'd0;
   else if (cin)    d_out <= d_from_ADR;

end

endmodule
```

1.3.5 Data Registers

```
1  //File name : reg_DATA.v
2  //This module is used to instantiate MDDR and MIDR registers.
3
4  module reg_DATA (clk, d_in1, d_in2, c_in1, c_in2, d_out);
5
6  input wire[7:0] d_in1, d_in2;
7  output reg[7:0] d_out=8'd0;
8
9  input c_in1, c_in2, clk;
10
11 always @ (posedge clk)
12 begin
13   if      (c_in1) d_out <= d_in1;
14   else if (c_in2) d_out <= d_in2;
15   else            d_out <= d_out;
16 end
17
18 endmodule
```

1.3.6 Program Counter

```
1  //File name : reg_PC.v
2  //This module is the PC register.
3
4  module reg_PC (clk, cin, inc,d_in, d_out);
5
6  input           clk, cin, inc;
7  input wire[7:0] d_in;
8  output reg[7:0] d_out = 8'd0;
9
10
11 always @ ( posedge clk )
12 begin
13   if(cin)       d_out <= d_in;
14   else if(inc)  d_out <= d_out + 1;
15   else          d_out <= d_out;
16 end
17
18
19 endmodule
```

1.4 Special Modules

1.4.1 ADR Maker

```
1  //File name : ADR_maker.v
2  //This module is responsible for handling MDAR register and matrix
   ↪  manipulation.
3
4
5  `include "define.v"
6  module ADR_maker (AWREF,in_Clock, A, ART, ARG, AWT, AWG, inc,
7            dec, TOG_inc, SEL, reset, d_out,TOG,d_to_ART,
              ↪  d_to_ARG,d_to_AWT,d_to_AWG,
8            cin_ART, cin_ARG,cin_AWT,cin_AWG,cin_ART_ref, cin_ARG_ref);
9
10 input           in_Clock, inc, dec,TOG_inc, reset;
11 input wire[3:0] SEL;
12 input wire[7:0] A;
13 input wire[8:0] ART, ARG,AWT,AWG;
14
15 output reg TOG=0;
16 output reg[17:0] d_out=18'd0;  //MDAR
17 output reg[17:0] AWREF=18'd0;
18 output reg[8:0] d_to_ART=9'd0, d_to_ARG=9'd0,d_to_AWT=9'd0,d_to_AWG=9'd0;
19 output reg cin_ART=0, cin_ARG=0,cin_AWT=0,cin_AWG=0,cin_ART_ref=0,
   ↪  cin_ARG_ref=0;
20
21 reg [17:0] TEMP_MDAR=18'd0;
22 wire  clk;
23 assign clk=in_Clock;
24
25 always @ (posedge clk)        //Control signals given in posedge, we check
   ↪  them in negedge
26 begin
27   if(TOG == 0)  AWREF <= {AWG, AWT};
28   else         AWREF <= {AWT, AWG};
29
30   if(TOG_inc)    TOG <= ~TOG;    //Toggle
31
32   if(inc)        d_out <= d_out+1;
33   else if(dec)    d_out <= d_out-1;
34   else if(reset)    d_out <= 18'd0;
35   else
36   begin
37     case (SEL)
38     `adr_matrix_r:            //MDAR is formed by read registers
39       begin
40         if(TOG == 0)  d_out <= {ARG, ART};
41         else         d_out <= {ART, ARG};
42         cin_ART     <= 0;
43         cin_ARG     <= 0;
44         cin_ARG_ref <= 0;
```

```
45          cin_ART_ref <= 0;
46          cin_AWG     <= 0;
47          cin_AWT     <= 0;
48        end
49
50      `adr_matrix_w:            //MDAR is formed by write registers
51        begin
52          if(TOG == 0)  d_out <= {AWG, AWT};
53          else        d_out <= {AWT, AWG};
54          cin_ART     <= 0;
55          cin_ARG     <= 0;
56          cin_ARG_ref <= 0;
57          cin_ART_ref <= 0;
58          cin_AWG     <= 0;
59          cin_AWT     <= 0;
60        end
61
62      `adr_last8:              //MDAR, last 8 is filled by A
63        begin
64          TEMP_MDAR[7:0]   <= A;
65          cin_ART     <= 0;
66          cin_ARG     <= 0;
67          cin_ARG_ref <= 0;
68          cin_ART_ref <= 0;
69          cin_AWG     <= 0;
70          cin_AWT     <= 0;
71        end
72      `adr_mid8:               //MDAR, mid 8 is filled by A
73        begin
74          TEMP_MDAR[15:8]  <= A;
75          cin_ART     <= 0;
76          cin_ARG     <= 0;
77          cin_ARG_ref <= 0;
78          cin_ART_ref <= 0;
79          cin_AWG     <= 0;
80          cin_AWT     <= 0;
81        end
82      `adr_first2:             //MDAR, first 8 is filled by A
83        begin
84          TEMP_MDAR[17:16] <= A[1:0];
85          cin_ART     <= 0;
86          cin_ARG     <= 0;
87          cin_ARG_ref <= 0;
88          cin_ART_ref <= 0;
89          cin_AWG     <= 0;
90          cin_AWT     <= 0;
91        end
92      `adr_to_mdar:
93        begin
94          d_out   <= TEMP_MDAR;
95          cin_ART     <= 0;
```

```
96          cin_ARG     <= 0;
97          cin_ARG_ref <= 0;
98          cin_ART_ref <= 0;
99          cin_AWG     <= 0;
100         cin_AWT     <= 0;
101       end
102     `adr_to_ar:
103         begin
104           cin_ART     <= 1;
105           cin_ARG     <= 1;
106           cin_ARG_ref <= 0;
107           cin_ART_ref <= 0;
108           cin_AWG     <= 0;
109           cin_AWT     <= 0;
110           if(TOG == 0)
111             begin
112               d_to_ARG <= TEMP_MDAR[17:9];
113               d_to_ART <= TEMP_MDAR[8:0];
114             end
115           else
116             begin
117               d_to_ART <= TEMP_MDAR[17:9];
118               d_to_ARG <= TEMP_MDAR[8:0];
119             end
120         end
121     `adr_to_aw:
122         begin
123           cin_ART     <= 0;
124           cin_ARG     <= 0;
125           cin_ARG_ref <= 0;
126           cin_ART_ref <= 0;
127           cin_AWG     <= 1;
128           cin_AWT     <= 1;
129           if(TOG == 0)
130             begin
131               d_to_AWG <= TEMP_MDAR[17:9];
132               d_to_AWT <= TEMP_MDAR[8:0];
133             end
134           else
135             begin
136               d_to_AWT <= TEMP_MDAR[17:9];
137               d_to_AWG <= TEMP_MDAR[8:0];
138             end
139         end
140     `adr_to_ar_ref:
141         begin
142           cin_ART     <= 0;
143           cin_ARG     <= 0;
144           cin_ARG_ref <= 1;
145           cin_ART_ref <= 1;
146           cin_AWG     <= 0;
```

```
147             cin_AWT     <= 0;
148             if(TOG == 0)
149               begin
150                 d_to_ARG <= TEMP_MDAR[17:9];
151                 d_to_ART <= TEMP_MDAR[8:0];
152               end
153             else
154               begin
155                 d_to_ART <= TEMP_MDAR[17:9];
156                 d_to_ARG <= TEMP_MDAR[8:0];
157               end
158           end
159         `adr_none:
160           begin
161             cin_ART     <= 0;
162             cin_ARG     <= 0;
163             cin_ARG_ref <= 0;
164             cin_ART_ref <= 0;
165             cin_AWG     <= 0;
166             cin_AWT     <= 0;
167           end
168         default:
169           begin
170             cin_ART     <= 0;
171             cin_ARG     <= 0;
172             cin_ARG_ref <= 0;
173             cin_ART_ref <= 0;
174             cin_AWG     <= 0;
175             cin_AWT     <= 0;
176           end
177       endcase
178     end
179 end
180 
181 
182 endmodule
```

1.4.2 Jump Decider

```
//File name : JMP_mux.v
//This module is used to manipulate jump operations.

`include "define.v"
//Jump selector. Outputs J (1 bit) connected to the input of the INS module.
module JMP_mux(JMP_sel,AC_Z,ZT,ZRG,ZRT,ZK0,ZK1,J);

input [3:0] JMP_sel;
input       AC_Z,ZT,ZRG,ZRT,ZK0,ZK1;

output reg  J;

always @ (*)
begin
  case (JMP_sel)
    `jmp_none:    J<= 1'd0;
    `jmp_jump:    J<= 1'd1;
    `jmp_jmpz:    J<= AC_Z;
    `jmp_jpnz:    J<= ~AC_Z;
    `jmp_jzt  :    J<= ZT;
    `jmp_jnrg:    J<= ~ZRG;
    `jmp_jnrt:    J<= ~ZRT;
    `jmp_jnk0:     J<= ~ZK0;
    `jmp_jnk1:    J<= ~ZK1;
      default:    J<= 1'd0;
  endcase
end
endmodule
```

1.4.3 ALU

```
//File name : ALU.v
//This module is the Arithmatic and Logic Unit of the processor.
//it also contains the AC register.

`include "define.v"

module ALU(select,A_bus,Z_out,AC,inc_AC,cin_AC,dec_AC,clk,rst);

input       clk,inc_AC,dec_AC,cin_AC,rst;
input [2:0] select;
input [7:0] A_bus;

output reg [11:0] AC=12'd0;
output reg        Z_out=1;

always @(AC)
  begin
    if(AC==12'b0) Z_out<=1; else Z_out<=0;
  end

always @(posedge clk)
  begin
    if (cin_AC) AC <= {4'd0,A_bus};
    else
      begin
        case(select)
          `alu_none:
            begin
              if (inc_AC==1)     AC <= AC+1;
              else if(dec_AC==1) AC <= AC-1;
              else if(rst==1)    AC <= 12'd0;
            end
          `alu_add :  AC <= AC+{4'd0,A_bus};
          `alu_sub :
            begin
              if(AC>{4'd0,A_bus}) AC <= AC-{4'd0,A_bus};
              else                AC <= {4'd0,A_bus}-AC;
            end
          `alu_div :  AC <= (AC+{5'd0,A_bus[7:1]})/{4'd0,A_bus};
          `alu_mul :  AC <= AC * A_bus;
          default  :  AC <= AC;
        endcase
      end
  end

endmodule
```

1.5 Data Memory

1.5.1 DATA MEMORY

```
1  //File name : dram.v
2  //This module is the IP module of the DRAM of size 65536.
3
4  // megafunction wizard: %RAM: 1-PORT%
5  // GENERATION: STANDARD
6  // VERSION: WM1.0
7  // MODULE: altsyncram
8
9  // ============================================================
10 // File Name: dram.v
11 // Megafunction Name(s):
12 //       altsyncram
13 //
14 // Simulation Library Files(s):
15 //       altera_mf
16 // ============================================================
17 // ************************************************************
18 // THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
19 //
20 // 15.0.0 Build 145 04/22/2015 SJ Full Version
21 // ************************************************************
22
23
24 //Copyright (C) 1991-2015 Altera Corporation. All rights reserved.
25 //Your use of Altera Corporation's design tools, logic functions
26 //and other software and tools, and its AMPP partner logic
27 //functions, and any output files from any of the foregoing
28 //(including device programming or simulation files), and any
29 //associated documentation or information are expressly subject
30 //to the terms and conditions of the Altera Program License
31 //Subscription Agreement, the Altera Quartus II License Agreement,
32 //the Altera MegaCore Function License Agreement, or other
33 //applicable license agreement, including, without limitation,
34 //that your use is for the sole purpose of programming logic
35 //devices manufactured by Altera and sold by Altera or its
36 //authorized distributors.  Please refer to the applicable
37 //agreement for further details.
38
39
40 // synopsys translate_off
41 `timescale 1 ps / 1 ps
42 // synopsys translate_on
43 module dram (
44   address,
45   clock,
46   data,
47   wren,
48   q);
```

```
49
50    input  [15:0]  address;
51    input    clock;
52    input  [7:0]  data;
53    input    wren;
54    output  [7:0]  q;
55  `ifndef ALTERA_RESERVED_QIS
56  // synopsys translate_off
57  `endif
58    tri1    clock;
59  `ifndef ALTERA_RESERVED_QIS
60  // synopsys translate_on
61  `endif
62
63    wire [7:0] sub_wire0;
64    wire [7:0] q = sub_wire0[7:0];
65
66    altsyncram  altsyncram_component (
67          .address_a (address),
68          .clock0 (clock),
69          .data_a (data),
70          .wren_a (wren),
71          .q_a (sub_wire0),
72          .aclr0 (1'b0),
73          .aclr1 (1'b0),
74          .address_b (1'b1),
75          .addressstall_a (1'b0),
76          .addressstall_b (1'b0),
77          .byteena_a (1'b1),
78          .byteena_b (1'b1),
79          .clock1 (1'b1),
80          .clocken0 (1'b1),
81          .clocken1 (1'b1),
82          .clocken2 (1'b1),
83          .clocken3 (1'b1),
84          .data_b (1'b1),
85          .eccstatus (),
86          .q_b (),
87          .rden_a (1'b1),
88          .rden_b (1'b1),
89          .wren_b (1'b0));
90    defparam
91      altsyncram_component.clock_enable_input_a = "BYPASS",
92      altsyncram_component.clock_enable_output_a = "BYPASS",
93      altsyncram_component.intended_device_family = "Cyclone IV E",
94      altsyncram_component.lpm_hint = "ENABLE_RUNTIME_MOD=NO",
95      altsyncram_component.lpm_type = "altsyncram",
96      altsyncram_component.numwords_a = 65536,
97      altsyncram_component.operation_mode = "SINGLE_PORT",
98      altsyncram_component.outdata_aclr_a = "NONE",
99      altsyncram_component.outdata_reg_a = "CLOCK0",
```

```
100     altsyncram_component.power_up_uninitialized = "FALSE",
101     altsyncram_component.read_during_write_mode_port_a =
        ↪  "NEW_DATA_NO_NBE_READ",
102     altsyncram_component.widthad_a = 16,
103     altsyncram_component.width_a = 8,
104     altsyncram_component.width_byteena_a = 1;
105
106
107 endmodule
108
109 // ============================================================
110 // CNX file retrieval info
111 // ============================================================
112 // Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
113 // Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
114 // Retrieval info: PRIVATE: AclrByte NUMERIC "0"
115 // Retrieval info: PRIVATE: AclrData NUMERIC "0"
116 // Retrieval info: PRIVATE: AclrOutput NUMERIC "0"
117 // Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
118 // Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
119 // Retrieval info: PRIVATE: BlankMemory NUMERIC "1"
120 // Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
121 // Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
122 // Retrieval info: PRIVATE: Clken NUMERIC "0"
123 // Retrieval info: PRIVATE: DataBusSeparated NUMERIC "1"
124 // Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
125 // Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
126 // Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
127 // Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone IV E"
128 // Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
129 // Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
130 // Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
131 // Retrieval info: PRIVATE: MIFfilename STRING ""
132 // Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "65536"
133 // Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
134 // Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_A NUMERIC "3"
135 // Retrieval info: PRIVATE: RegAddr NUMERIC "1"
136 // Retrieval info: PRIVATE: RegData NUMERIC "1"
137 // Retrieval info: PRIVATE: RegOutput NUMERIC "1"
138 // Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
139 // Retrieval info: PRIVATE: SingleClock NUMERIC "1"
140 // Retrieval info: PRIVATE: UseDQRAM NUMERIC "1"
141 // Retrieval info: PRIVATE: WRCONTROL_ACLR_A NUMERIC "0"
142 // Retrieval info: PRIVATE: WidthAddr NUMERIC "16"
143 // Retrieval info: PRIVATE: WidthData NUMERIC "8"
144 // Retrieval info: PRIVATE: rden NUMERIC "0"
145 // Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
146 // Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
147 // Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING "BYPASS"
148 // Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone IV E"
149 // Retrieval info: CONSTANT: LPM_HINT STRING "ENABLE_RUNTIME_MOD=NO"
```

```
// Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
// Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "65536"
// Retrieval info: CONSTANT: OPERATION_MODE STRING "SINGLE_PORT"
// Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
// Retrieval info: CONSTANT: OUTDATA_REG_A STRING "CLOCK0"
// Retrieval info: CONSTANT: POWER_UP_UNINITIALIZED STRING "FALSE"
// Retrieval info: CONSTANT: READ_DURING_WRITE_MODE_PORT_A STRING
    ↪  "NEW_DATA_NO_NBE_READ"
// Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "16"
// Retrieval info: CONSTANT: WIDTH_A NUMERIC "8"
// Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
// Retrieval info: USED_PORT: address 0 0 16 0 INPUT NODEFVAL "address[15..0]"
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
// Retrieval info: USED_PORT: data 0 0 8 0 INPUT NODEFVAL "data[7..0]"
// Retrieval info: USED_PORT: q 0 0 8 0 OUTPUT NODEFVAL "q[7..0]"
// Retrieval info: USED_PORT: wren 0 0 0 0 INPUT NODEFVAL "wren"
// Retrieval info: CONNECT: @address_a 0 0 16 0 address 0 0 16 0
// Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: @data_a 0 0 8 0 data 0 0 8 0
// Retrieval info: CONNECT: @wren_a 0 0 0 0 wren 0 0 0 0
// Retrieval info: CONNECT: q 0 0 8 0 @q_a 0 0 8 0
// Retrieval info: GEN_FILE: TYPE_NORMAL dram.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL dram.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL dram.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL dram.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL dram_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL dram_bb.v TRUE
// Retrieval info: LIB_FILE: altera_mf
```

1.5.2 DATA MEMORY 512

```
1  //File name : dram_512.v
2  //This module is the DRAM used in the project of size 512x512,
3  //built using 4 RAMs of size 256x256.
4
5  module dram_512(address,clock,data,wren,q);
6
7  input  [17:0] address;
8  input         clock;
9  input  [7:0]  data;
10 input         wren;
11 output[7:0]  q;
12
13 wire [1:0] selector;
14 wire [7:0] q1,q2,q3,q4;
15 wire wren1,wren2,wren3,wren4;
16
17 parameter DM1=2'b00;
18 parameter DM2=2'b01;
19 parameter DM3=2'b10;
20 parameter DM4=2'b11;
21
22 assign selector[1:0]=address[17:16];
23
24
25 dram
   ↪  d1(.address(address[15:0]),.clock(clock),.data(data),.wren(wren1),.q(q1));
26 dram
   ↪  d2(.address(address[15:0]),.clock(clock),.data(data),.wren(wren2),.q(q2));
27 dram
   ↪  d3(.address(address[15:0]),.clock(clock),.data(data),.wren(wren3),.q(q3));
28 dram
   ↪  d4(.address(address[15:0]),.clock(clock),.data(data),.wren(wren4),.q(q4));
29 dram_out_mux
   ↪  dram_out_mux(.data0x(q1),.data1x(q2),.data2x(q3),.data3x(q4),.sel(selector),.result(q));
30 dram_wen_sel_decoder dwsd(
31   .data(selector),
32   .enable(wren),
33   .eq0(wren1),
34   .eq1(wren2),
35   .eq2(wren3),
36   .eq3(wren4));
37
38 endmodule
```

1.5.3 Memory Router

```
1  //File name : memory_router.v
2  //This module is the core that share the resources (Wen,address
   ↪  bus,d_in,d_out)
3  //of IRAM and DRAM to corresponding sources in different modes of operations.
4
5  module memory_router(wen,address,din,
6                       i_wen,i_add,i_din,
7                       p_wen,p_address,p_din,p_ins_address,
8                       memory_size_select,rx_wen,rx_address,rx_din,
9                       tx_address,
10                      user_address,
11                      mode,d_or_i);
12
13 //iram access ports
14
15 output       i_wen;
16 output [7:0] i_add;
17 output [7:0] i_din;
18
19
20 //dram accsess ports
21
22
23 output        wen;
24 output [7:0]  din;
25 output [17:0] address;
26
27 //Tx ports
28
29 input  [17:0] tx_address;
30
31 //Rx ports
32
33 input             rx_wen;
34 input      [17:0] rx_address;
35 input      [7:0]  rx_din;
36 output reg [17:0] memory_size_select=18'd262143;
37
38 //processor ports
39
40 input         p_wen;
41 input  [17:0] p_address;
42 input  [7:0]  p_din;
43 input  [7:0]  p_ins_address;
44
45
46 //user ports
47
48 input [17:0] user_address;
49
```

```
50  //mode selector for 4 modes
51
52  input [1:0] mode;
53
54  //instruction mode or data mode
55
56  input d_or_i;
57
58
59
60  //splitter wiring
61
62  wire d_out_wen;
63  wire [7:0] d_out_din;
64  wire [17:0] d_out_d_address;
65  wire [17:0] i_out_i_address;
66  wire [17:0] i_out_i_user_address;
67  wire [17:0] d_out_d_user_address;
68
69  D_Wen_mux D_Wen_mux (
70              .data0(0),
71              .data1(d_out_wen),
72              .data2(p_wen),
73              .data3(0),
74              .sel(mode),
75              .result(wen));
76
77  D_Address_mux  D_Address_mux (
78              .data0x(d_out_d_user_address),
79              .data1x(d_out_d_address),
80              .data2x(p_address),
81              .data3x(tx_address),
82              .sel(mode),
83              .result(address));
84
85  I_Address_mux  I_Address_mux (
86              .data0x(i_out_i_user_address[7:0]),
87              .data1x(i_out_i_address),
88              .data2x(p_ins_address),
89              .data3x(8'd0),
90              .sel(mode),
91              .result(i_add));
92
93
94  din_mux din_mux (
95              .data0x(8'd0),
96              .data1x(d_out_din),
97              .data2x(p_din),
98              .data3x(8'd0),
99              .sel(mode),
100             .result(din));
```

```
101
102
103 splitter #(.bit_width(1)) wen_split (
104             .in(rx_wen),
105             .d_out(d_out_wen),
106             .i_out(i_wen),
107             .enable(((~mode[1])&(mode[0]))), //in Rx operation mode
108             .selector(d_or_i));
109
110 splitter #(.bit_width(8)) din_split (
111             .in(rx_din),
112             .d_out(d_out_din),
113             .i_out(i_din),
114             .enable(((~mode[1])&(mode[0]))), //in Rx operation mode
115             .selector(d_or_i));
116
117 splitter #(.bit_width(18)) address_split (
118             .in(rx_address),
119             .d_out(d_out_d_address),
120             .i_out(i_out_i_address),
121             .enable(((~mode[1])&(mode[0]))),  //in Rx operation mode
122             .selector(d_or_i));
123
124 splitter #(.bit_width(18)) user_address_split (
125             .in(user_address),
126             .d_out(d_out_d_user_address),
127             .i_out(i_out_i_user_address),
128             .enable(((~mode[1])&(~mode[0]))), //in IDLE operation mode
129             .selector(d_or_i));
130
131
132 always @(d_or_i)
133   begin
134     case(d_or_i)
135       0:memory_size_select<=18'd262143;
136       1:memory_size_select<=18'd255;
137       default:memory_size_select<=18'd262143;
138     endcase
139   end
140
141 endmodule
```

1.5.4 Data Memory Address Mux

```
1  //File name : D_Address_mux.v
2  //This module used to share the address bus with the different sources
3  //that use it in different modes of operations.Used inside the memory_router.v
   ↪  module.
4
5  // megafunction wizard: %LPM_MUX%
6  // GENERATION: STANDARD
7  // VERSION: WM1.0
8  // MODULE: LPM_MUX
9
10 // ============================================================
11 // File Name: D_Address_mux.v
12 // Megafunction Name(s):
13 //       LPM_MUX
14 //
15 // Simulation Library Files(s):
16 //       lpm
17 // ============================================================
18 // ************************************************************
19 // THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
20 //
21 // 15.0.0 Build 145 04/22/2015 SJ Full Version
22 // ************************************************************
23
24
25 //Copyright (C) 1991-2015 Altera Corporation. All rights reserved.
26 //Your use of Altera Corporation's design tools, logic functions
27 //and other software and tools, and its AMPP partner logic
28 //functions, and any output files from any of the foregoing
29 //(including device programming or simulation files), and any
30 //associated documentation or information are expressly subject
31 //to the terms and conditions of the Altera Program License
32 //Subscription Agreement, the Altera Quartus II License Agreement,
33 //the Altera MegaCore Function License Agreement, or other
34 //applicable license agreement, including, without limitation,
35 //that your use is for the sole purpose of programming logic
36 //devices manufactured by Altera and sold by Altera or its
37 //authorized distributors.  Please refer to the applicable
38 //agreement for further details.
39
40
41 // synopsys translate_off
42 `timescale 1 ps / 1 ps
43 // synopsys translate_on
44 module D_Address_mux (
45   data0x,
46   data1x,
47   data2x,
48   data3x,
49   sel,
```

```
50    result);
51
52    input  [17:0]  data0x;
53    input  [17:0]  data1x;
54    input  [17:0]  data2x;
55    input  [17:0]  data3x;
56    input  [1:0]  sel;
57    output  [17:0]  result;
58
59    wire [17:0] sub_wire5;
60    wire [17:0] sub_wire4 = data3x[17:0];
61    wire [17:0] sub_wire3 = data2x[17:0];
62    wire [17:0] sub_wire2 = data1x[17:0];
63    wire [17:0] sub_wire0 = data0x[17:0];
64    wire [71:0] sub_wire1 = {sub_wire4, sub_wire3, sub_wire2, sub_wire0};
65    wire [17:0] result = sub_wire5[17:0];
66
67    lpm_mux  LPM_MUX_component (
68          .data (sub_wire1),
69          .sel (sel),
70          .result (sub_wire5)
71          // synopsys translate_off
72          ,
73          .aclr (),
74          .clken (),
75          .clock ()
76          // synopsys translate_on
77          );
78    defparam
79      LPM_MUX_component.lpm_size = 4,
80      LPM_MUX_component.lpm_type = "LPM_MUX",
81      LPM_MUX_component.lpm_width = 18,
82      LPM_MUX_component.lpm_widths = 2;
83
84
85  endmodule
86
87  // ============================================================
88  // CNX file retrieval info
89  // ============================================================
90  // Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone IV E"
91  // Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
92  // Retrieval info: PRIVATE: new_diagram STRING "1"
93  // Retrieval info: LIBRARY: lpm lpm.lpm_components.all
94  // Retrieval info: CONSTANT: LPM_SIZE NUMERIC "4"
95  // Retrieval info: CONSTANT: LPM_TYPE STRING "LPM_MUX"
96  // Retrieval info: CONSTANT: LPM_WIDTH NUMERIC "18"
97  // Retrieval info: CONSTANT: LPM_WIDTHS NUMERIC "2"
98  // Retrieval info: USED_PORT: data0x 0 0 18 0 INPUT NODEFVAL "data0x[17..0]"
99  // Retrieval info: USED_PORT: data1x 0 0 18 0 INPUT NODEFVAL "data1x[17..0]"
100 // Retrieval info: USED_PORT: data2x 0 0 18 0 INPUT NODEFVAL "data2x[17..0]"
```

```
// Retrieval info: USED_PORT: data3x 0 0 18 0 INPUT NODEFVAL "data3x[17..0]"
// Retrieval info: USED_PORT: result 0 0 18 0 OUTPUT NODEFVAL "result[17..0]"
// Retrieval info: USED_PORT: sel 0 0 2 0 INPUT NODEFVAL "sel[1..0]"
// Retrieval info: CONNECT: @data 0 0 18 0 data0x 0 0 18 0
// Retrieval info: CONNECT: @data 0 0 18 18 data1x 0 0 18 0
// Retrieval info: CONNECT: @data 0 0 18 36 data2x 0 0 18 0
// Retrieval info: CONNECT: @data 0 0 18 54 data3x 0 0 18 0
// Retrieval info: CONNECT: @sel 0 0 2 0 sel 0 0 2 0
// Retrieval info: CONNECT: result 0 0 18 0 @result 0 0 18 0
// Retrieval info: GEN_FILE: TYPE_NORMAL D_Address_mux.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL D_Address_mux.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL D_Address_mux.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL D_Address_mux.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL D_Address_mux_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL D_Address_mux_bb.v TRUE
// Retrieval info: LIB_FILE: lpm
```

1.5.5 Data Memory Write Enable Mux

```
//File name : D_Wen_mux.v
//This module used to share the write enable wire with the different sources
//that use it in different modes of operations.Used inside the memory_router.v
↪  module.

// megafunction wizard: %LPM_MUX%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: LPM_MUX

// ============================================================
// File Name: D_Wen_mux.v
// Megafunction Name(s):
//       LPM_MUX
//
// Simulation Library Files(s):
//       lpm
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 15.0.0 Build 145 04/22/2015 SJ Full Version
// ************************************************************


//Copyright (C) 1991-2015 Altera Corporation. All rights reserved.
//Your use of Altera Corporation's design tools, logic functions
//and other software and tools, and its AMPP partner logic
//functions, and any output files from any of the foregoing
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Altera Program License
//Subscription Agreement, the Altera Quartus II License Agreement,
//the Altera MegaCore Function License Agreement, or other
//applicable license agreement, including, without limitation,
//that your use is for the sole purpose of programming logic
//devices manufactured by Altera and sold by Altera or its
//authorized distributors.  Please refer to the applicable
//agreement for further details.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module D_Wen_mux (
  data0,
  data1,
  data2,
  data3,
  sel,
```

```
50    result);
51
52    input    data0;
53    input    data1;
54    input    data2;
55    input    data3;
56    input  [1:0]  sel;
57    output    result;
58
59    wire [0:0] sub_wire5;
60    wire  sub_wire4 = data3;
61    wire  sub_wire3 = data2;
62    wire  sub_wire2 = data1;
63    wire  sub_wire0 = data0;
64    wire [3:0] sub_wire1 = {sub_wire4, sub_wire3, sub_wire2, sub_wire0};
65    wire [0:0] sub_wire6 = sub_wire5[0:0];
66    wire  result = sub_wire6;
67
68    lpm_mux  LPM_MUX_component (
69          .data (sub_wire1),
70          .sel (sel),
71          .result (sub_wire5)
72          // synopsys translate_off
73          ,
74          .aclr (),
75          .clken (),
76          .clock ()
77          // synopsys translate_on
78          );
79    defparam
80      LPM_MUX_component.lpm_size = 4,
81      LPM_MUX_component.lpm_type = "LPM_MUX",
82      LPM_MUX_component.lpm_width = 1,
83      LPM_MUX_component.lpm_widths = 2;
84
85
86  endmodule
87
88  // ============================================================
89  // CNX file retrieval info
90  // ============================================================
91  // Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone IV E"
92  // Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
93  // Retrieval info: PRIVATE: new_diagram STRING "1"
94  // Retrieval info: LIBRARY: lpm lpm.lpm_components.all
95  // Retrieval info: CONSTANT: LPM_SIZE NUMERIC "4"
96  // Retrieval info: CONSTANT: LPM_TYPE STRING "LPM_MUX"
97  // Retrieval info: CONSTANT: LPM_WIDTH NUMERIC "1"
98  // Retrieval info: CONSTANT: LPM_WIDTHS NUMERIC "2"
99  // Retrieval info: USED_PORT: data0 0 0 0 0 INPUT NODEFVAL "data0"
100 // Retrieval info: USED_PORT: data1 0 0 0 0 INPUT NODEFVAL "data1"
```

```
101 // Retrieval info: USED_PORT: data2 0 0 0 0 INPUT NODEFVAL "data2"
102 // Retrieval info: USED_PORT: data3 0 0 0 0 INPUT NODEFVAL "data3"
103 // Retrieval info: USED_PORT: result 0 0 0 0 OUTPUT NODEFVAL "result"
104 // Retrieval info: USED_PORT: sel 0 0 2 0 INPUT NODEFVAL "sel[1..0]"
105 // Retrieval info: CONNECT: @data 0 0 1 0 data0 0 0 0 0
106 // Retrieval info: CONNECT: @data 0 0 1 1 data1 0 0 0 0
107 // Retrieval info: CONNECT: @data 0 0 1 2 data2 0 0 0 0
108 // Retrieval info: CONNECT: @data 0 0 1 3 data3 0 0 0 0
109 // Retrieval info: CONNECT: @sel 0 0 2 0 sel 0 0 2 0
110 // Retrieval info: CONNECT: result 0 0 0 0 @result 0 0 1 0
111 // Retrieval info: GEN_FILE: TYPE_NORMAL D_Wen_mux.v TRUE
112 // Retrieval info: GEN_FILE: TYPE_NORMAL D_Wen_mux.inc FALSE
113 // Retrieval info: GEN_FILE: TYPE_NORMAL D_Wen_mux.cmp FALSE
114 // Retrieval info: GEN_FILE: TYPE_NORMAL D_Wen_mux.bsf FALSE
115 // Retrieval info: GEN_FILE: TYPE_NORMAL D_Wen_mux_inst.v FALSE
116 // Retrieval info: GEN_FILE: TYPE_NORMAL D_Wen_mux_bb.v TRUE
117 // Retrieval info: LIB_FILE: lpm
```

1.5.6 Data Memory Input Data Mux

```
1  //File name : din_mux.v
2  //This is instantiated in the memory router module to share din of the DRAM.
3
4  // megafunction wizard: %LPM_MUX%
5  // GENERATION: STANDARD
6  // VERSION: WM1.0
7  // MODULE: LPM_MUX
8
9  // ============================================================
10 // File Name: din_mux.v
11 // Megafunction Name(s):
12 //       LPM_MUX
13 //
14 // Simulation Library Files(s):
15 //       lpm
16 // ============================================================
17 // ************************************************************
18 // THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
19 //
20 // 15.0.0 Build 145 04/22/2015 SJ Full Version
21 // ************************************************************
22
23
24 //Copyright (C) 1991-2015 Altera Corporation. All rights reserved.
25 //Your use of Altera Corporation's design tools, logic functions
26 //and other software and tools, and its AMPP partner logic
27 //functions, and any output files from any of the foregoing
28 //(including device programming or simulation files), and any
29 //associated documentation or information are expressly subject
30 //to the terms and conditions of the Altera Program License
31 //Subscription Agreement, the Altera Quartus II License Agreement,
32 //the Altera MegaCore Function License Agreement, or other
33 //applicable license agreement, including, without limitation,
34 //that your use is for the sole purpose of programming logic
35 //devices manufactured by Altera and sold by Altera or its
36 //authorized distributors.  Please refer to the applicable
37 //agreement for further details.
38
39
40 // synopsys translate_off
41 `timescale 1 ps / 1 ps
42 // synopsys translate_on
43 module din_mux (
44   data0x,
45   data1x,
46   data2x,
47   data3x,
48   sel,
49   result);
50
```

```
51    input  [7:0]  data0x;
52    input  [7:0]  data1x;
53    input  [7:0]  data2x;
54    input  [7:0]  data3x;
55    input  [1:0]  sel;
56    output  [7:0]  result;
57
58    wire [7:0] sub_wire5;
59    wire [7:0] sub_wire4 = data3x[7:0];
60    wire [7:0] sub_wire3 = data2x[7:0];
61    wire [7:0] sub_wire2 = data1x[7:0];
62    wire [7:0] sub_wire0 = data0x[7:0];
63    wire [31:0] sub_wire1 = {sub_wire4, sub_wire3, sub_wire2, sub_wire0};
64    wire [7:0] result = sub_wire5[7:0];
65
66    lpm_mux  LPM_MUX_component (
67          .data (sub_wire1),
68          .sel (sel),
69          .result (sub_wire5)
70          // synopsys translate_off
71          ,
72          .aclr (),
73          .clken (),
74          .clock ()
75          // synopsys translate_on
76          );
77    defparam
78      LPM_MUX_component.lpm_size = 4,
79      LPM_MUX_component.lpm_type = "LPM_MUX",
80      LPM_MUX_component.lpm_width = 8,
81      LPM_MUX_component.lpm_widths = 2;
82
83
84  endmodule
85
86  // ============================================================
87  // CNX file retrieval info
88  // ============================================================
89  // Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone IV E"
90  // Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
91  // Retrieval info: PRIVATE: new_diagram STRING "1"
92  // Retrieval info: LIBRARY: lpm lpm.lpm_components.all
93  // Retrieval info: CONSTANT: LPM_SIZE NUMERIC "4"
94  // Retrieval info: CONSTANT: LPM_TYPE STRING "LPM_MUX"
95  // Retrieval info: CONSTANT: LPM_WIDTH NUMERIC "8"
96  // Retrieval info: CONSTANT: LPM_WIDTHS NUMERIC "2"
97  // Retrieval info: USED_PORT: data0x 0 0 8 0 INPUT NODEFVAL "data0x[7..0]"
98  // Retrieval info: USED_PORT: data1x 0 0 8 0 INPUT NODEFVAL "data1x[7..0]"
99  // Retrieval info: USED_PORT: data2x 0 0 8 0 INPUT NODEFVAL "data2x[7..0]"
100 // Retrieval info: USED_PORT: data3x 0 0 8 0 INPUT NODEFVAL "data3x[7..0]"
101 // Retrieval info: USED_PORT: result 0 0 8 0 OUTPUT NODEFVAL "result[7..0]"
```

```
// Retrieval info: USED_PORT: sel 0 0 2 0 INPUT NODEFVAL "sel[1..0]"
// Retrieval info: CONNECT: @data 0 0 8 0 data0x 0 0 8 0
// Retrieval info: CONNECT: @data 0 0 8 8 data1x 0 0 8 0
// Retrieval info: CONNECT: @data 0 0 8 16 data2x 0 0 8 0
// Retrieval info: CONNECT: @data 0 0 8 24 data3x 0 0 8 0
// Retrieval info: CONNECT: @sel 0 0 2 0 sel 0 0 2 0
// Retrieval info: CONNECT: result 0 0 8 0 @result 0 0 8 0
// Retrieval info: GEN_FILE: TYPE_NORMAL din_mux.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL din_mux.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL din_mux.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL din_mux.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL din_mux_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL din_mux_bb.v TRUE
// Retrieval info: LIB_FILE: lpm
```

1.5.7 Data Memory Output Data Mux

```
//File name : dram_out_mux.v
//This module is used to choose an output from 4 256x256 RAMS instantiated
//in the dram_512 module.

// megafunction wizard: %LPM_MUX%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: LPM_MUX

// ============================================================
// File Name: dram_out_mux.v
// Megafunction Name(s):
//       LPM_MUX
//
// Simulation Library Files(s):
//       lpm
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 15.0.0 Build 145 04/22/2015 SJ Full Version
// ************************************************************


//Copyright (C) 1991-2015 Altera Corporation. All rights reserved.
//Your use of Altera Corporation's design tools, logic functions
//and other software and tools, and its AMPP partner logic
//functions, and any output files from any of the foregoing
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Altera Program License
//Subscription Agreement, the Altera Quartus II License Agreement,
//the Altera MegaCore Function License Agreement, or other
//applicable license agreement, including, without limitation,
//that your use is for the sole purpose of programming logic
//devices manufactured by Altera and sold by Altera or its
//authorized distributors.  Please refer to the applicable
//agreement for further details.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module dram_out_mux (
  data0x,
  data1x,
  data2x,
  data3x,
  sel,
  result);
```

```
51
52    input  [7:0]  data0x;
53    input  [7:0]  data1x;
54    input  [7:0]  data2x;
55    input  [7:0]  data3x;
56    input  [1:0]  sel;
57    output  [7:0]  result;
58
59    wire [7:0] sub_wire5;
60    wire [7:0] sub_wire4 = data3x[7:0];
61    wire [7:0] sub_wire3 = data2x[7:0];
62    wire [7:0] sub_wire2 = data1x[7:0];
63    wire [7:0] sub_wire0 = data0x[7:0];
64    wire [31:0] sub_wire1 = {sub_wire4, sub_wire3, sub_wire2, sub_wire0};
65    wire [7:0] result = sub_wire5[7:0];
66
67    lpm_mux  LPM_MUX_component (
68          .data (sub_wire1),
69          .sel (sel),
70          .result (sub_wire5)
71          // synopsys translate_off
72          ,
73          .aclr (),
74          .clken (),
75          .clock ()
76          // synopsys translate_on
77          );
78    defparam
79      LPM_MUX_component.lpm_size = 4,
80      LPM_MUX_component.lpm_type = "LPM_MUX",
81      LPM_MUX_component.lpm_width = 8,
82      LPM_MUX_component.lpm_widths = 2;
83
84
85  endmodule
86
87  // ============================================================
88  // CNX file retrieval info
89  // ============================================================
90  // Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone IV E"
91  // Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "1"
92  // Retrieval info: PRIVATE: new_diagram STRING "1"
93  // Retrieval info: LIBRARY: lpm lpm.lpm_components.all
94  // Retrieval info: CONSTANT: LPM_SIZE NUMERIC "4"
95  // Retrieval info: CONSTANT: LPM_TYPE STRING "LPM_MUX"
96  // Retrieval info: CONSTANT: LPM_WIDTH NUMERIC "8"
97  // Retrieval info: CONSTANT: LPM_WIDTHS NUMERIC "2"
98  // Retrieval info: USED_PORT: data0x 0 0 8 0 INPUT NODEFVAL "data0x[7..0]"
99  // Retrieval info: USED_PORT: data1x 0 0 8 0 INPUT NODEFVAL "data1x[7..0]"
100 // Retrieval info: USED_PORT: data2x 0 0 8 0 INPUT NODEFVAL "data2x[7..0]"
101 // Retrieval info: USED_PORT: data3x 0 0 8 0 INPUT NODEFVAL "data3x[7..0]"
```

```
// Retrieval info: USED_PORT: result 0 0 8 0 OUTPUT NODEFVAL "result[7..0]"
// Retrieval info: USED_PORT: sel 0 0 2 0 INPUT NODEFVAL "sel[1..0]"
// Retrieval info: CONNECT: @data 0 0 8 0 data0x 0 0 8 0
// Retrieval info: CONNECT: @data 0 0 8 8 data1x 0 0 8 0
// Retrieval info: CONNECT: @data 0 0 8 16 data2x 0 0 8 0
// Retrieval info: CONNECT: @data 0 0 8 24 data3x 0 0 8 0
// Retrieval info: CONNECT: @sel 0 0 2 0 sel 0 0 2 0
// Retrieval info: CONNECT: result 0 0 8 0 @result 0 0 8 0
// Retrieval info: GEN_FILE: TYPE_NORMAL dram_out_mux.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL dram_out_mux.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL dram_out_mux.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL dram_out_mux.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL dram_out_mux_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL dram_out_mux_bb.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL dram_out_mux_syn.v TRUE
// Retrieval info: LIB_FILE: lpm
```

1.5.8 Data Memory Write Enable Decoder

```
//File name : dram_wen_sel_decoder.v
//This module is used to demultiplex the wen signal given to 512x512 RAM to
//4 256x256 RAMS.
//Instantiated in the dram_512 module.

// megafunction wizard: %LPM_DECODE%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: LPM_DECODE

// ============================================================
// File Name: dram_wen_sel_decoder.v
// Megafunction Name(s):
//       LPM_DECODE
//
// Simulation Library Files(s):
//       lpm
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 15.0.0 Build 145 04/22/2015 SJ Full Version
// ************************************************************


//Copyright (C) 1991-2015 Altera Corporation. All rights reserved.
//Your use of Altera Corporation's design tools, logic functions
//and other software and tools, and its AMPP partner logic
//functions, and any output files from any of the foregoing
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Altera Program License
//Subscription Agreement, the Altera Quartus II License Agreement,
//the Altera MegaCore Function License Agreement, or other
//applicable license agreement, including, without limitation,
//that your use is for the sole purpose of programming logic
//devices manufactured by Altera and sold by Altera or its
//authorized distributors.  Please refer to the applicable
//agreement for further details.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module dram_wen_sel_decoder (
  data,
  enable,
  eq0,
  eq1,
  eq2,
```

```
51    eq3);
52
53    input  [1:0]  data;
54    input    enable;
55    output    eq0;
56    output    eq1;
57    output    eq2;
58    output    eq3;
59
60    wire [3:0] sub_wire0;
61    wire [3:3] sub_wire4 = sub_wire0[3:3];
62    wire [2:2] sub_wire3 = sub_wire0[2:2];
63    wire [1:1] sub_wire2 = sub_wire0[1:1];
64    wire [0:0] sub_wire1 = sub_wire0[0:0];
65    wire  eq0 = sub_wire1;
66    wire  eq1 = sub_wire2;
67    wire  eq2 = sub_wire3;
68    wire  eq3 = sub_wire4;
69
70    lpm_decode  LPM_DECODE_component (
71          .data (data),
72          .enable (enable),
73          .eq (sub_wire0)
74          // synopsys translate_off
75          ,
76          .aclr (),
77          .clken (),
78          .clock ()
79          // synopsys translate_on
80          );
81    defparam
82      LPM_DECODE_component.lpm_decodes = 4,
83      LPM_DECODE_component.lpm_type = "LPM_DECODE",
84      LPM_DECODE_component.lpm_width = 2;
85
86
87  endmodule
88
89  // ============================================================
90  // CNX file retrieval info
91  // ============================================================
92  // Retrieval info: PRIVATE: BaseDec NUMERIC "1"
93  // Retrieval info: PRIVATE: EnableInput NUMERIC "1"
94  // Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone IV E"
95  // Retrieval info: PRIVATE: LPM_PIPELINE NUMERIC "0"
96  // Retrieval info: PRIVATE: Latency NUMERIC "0"
97  // Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
98  // Retrieval info: PRIVATE: aclr NUMERIC "0"
99  // Retrieval info: PRIVATE: clken NUMERIC "0"
100 // Retrieval info: PRIVATE: eq0 NUMERIC "1"
101 // Retrieval info: PRIVATE: eq1 NUMERIC "1"
```

```
// Retrieval info: PRIVATE: eq2 NUMERIC "1"
// Retrieval info: PRIVATE: eq3 NUMERIC "1"
// Retrieval info: PRIVATE: nBit NUMERIC "2"
// Retrieval info: PRIVATE: new_diagram STRING "1"
// Retrieval info: LIBRARY: lpm lpm.lpm_components.all
// Retrieval info: CONSTANT: LPM_DECODES NUMERIC "4"
// Retrieval info: CONSTANT: LPM_TYPE STRING "LPM_DECODE"
// Retrieval info: CONSTANT: LPM_WIDTH NUMERIC "2"
// Retrieval info: USED_PORT: @eq 0 0 4 0 OUTPUT NODEFVAL "@eq[3..0]"
// Retrieval info: USED_PORT: data 0 0 2 0 INPUT NODEFVAL "data[1..0]"
// Retrieval info: USED_PORT: enable 0 0 0 0 INPUT NODEFVAL "enable"
// Retrieval info: USED_PORT: eq0 0 0 0 0 OUTPUT NODEFVAL "eq0"
// Retrieval info: USED_PORT: eq1 0 0 0 0 OUTPUT NODEFVAL "eq1"
// Retrieval info: USED_PORT: eq2 0 0 0 0 OUTPUT NODEFVAL "eq2"
// Retrieval info: USED_PORT: eq3 0 0 0 0 OUTPUT NODEFVAL "eq3"
// Retrieval info: CONNECT: @data 0 0 2 0 data 0 0 2 0
// Retrieval info: CONNECT: @enable 0 0 0 0 enable 0 0 0 0
// Retrieval info: CONNECT: eq0 0 0 0 0 @eq 0 0 1 0
// Retrieval info: CONNECT: eq1 0 0 0 0 @eq 0 0 1 1
// Retrieval info: CONNECT: eq2 0 0 0 0 @eq 0 0 1 2
// Retrieval info: CONNECT: eq3 0 0 0 0 @eq 0 0 1 3
// Retrieval info: GEN_FILE: TYPE_NORMAL dram_wen_sel_decoder.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL dram_wen_sel_decoder.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL dram_wen_sel_decoder.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL dram_wen_sel_decoder.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL dram_wen_sel_decoder_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL dram_wen_sel_decoder_bb.v TRUE
// Retrieval info: LIB_FILE: lpm
```

1.6 Instruction Memory

1.6.1 Instruction Memory Address Mux

```
1  //File name : I_Address_mux.v
2  //This module is used to share the address bus of IRAM with different sources
3  //in different modes of operations.
4  //Instantiated in the memory router module.
5
6  // megafunction wizard: %LPM_MUX%
7  // GENERATION: STANDARD
8  // VERSION: WM1.0
9  // MODULE: LPM_MUX
10
11 // ============================================================
12 // File Name: I_Address_mux.v
13 // Megafunction Name(s):
14 //       LPM_MUX
15 //
16 // Simulation Library Files(s):
17 //       lpm
18 // ============================================================
19 // ************************************************************
20 // THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
21 //
22 // 15.0.0 Build 145 04/22/2015 SJ Full Version
23 // ************************************************************
24
25
26 //Copyright (C) 1991-2015 Altera Corporation. All rights reserved.
27 //Your use of Altera Corporation's design tools, logic functions
28 //and other software and tools, and its AMPP partner logic
29 //functions, and any output files from any of the foregoing
30 //(including device programming or simulation files), and any
31 //associated documentation or information are expressly subject
32 //to the terms and conditions of the Altera Program License
33 //Subscription Agreement, the Altera Quartus II License Agreement,
34 //the Altera MegaCore Function License Agreement, or other
35 //applicable license agreement, including, without limitation,
36 //that your use is for the sole purpose of programming logic
37 //devices manufactured by Altera and sold by Altera or its
38 //authorized distributors.  Please refer to the applicable
39 //agreement for further details.
40
41
42 // synopsys translate_off
43 `timescale 1 ps / 1 ps
44 // synopsys translate_on
45 module I_Address_mux (
46   data0x,
47   data1x,
48   data2x,
```

```
49    data3x,
50    sel,
51    result);
52
53    input  [7:0]  data0x;
54    input  [7:0]  data1x;
55    input  [7:0]  data2x;
56    input  [7:0]  data3x;
57    input  [1:0]  sel;
58    output  [7:0]  result;
59
60    wire [7:0] sub_wire5;
61    wire [7:0] sub_wire4 = data3x[7:0];
62    wire [7:0] sub_wire3 = data2x[7:0];
63    wire [7:0] sub_wire2 = data1x[7:0];
64    wire [7:0] sub_wire0 = data0x[7:0];
65    wire [31:0] sub_wire1 = {sub_wire4, sub_wire3, sub_wire2, sub_wire0};
66    wire [7:0] result = sub_wire5[7:0];
67
68    lpm_mux  LPM_MUX_component (
69          .data (sub_wire1),
70          .sel (sel),
71          .result (sub_wire5)
72          // synopsys translate_off
73          ,
74          .aclr (),
75          .clken (),
76          .clock ()
77          // synopsys translate_on
78          );
79    defparam
80      LPM_MUX_component.lpm_size = 4,
81      LPM_MUX_component.lpm_type = "LPM_MUX",
82      LPM_MUX_component.lpm_width = 8,
83      LPM_MUX_component.lpm_widths = 2;
84
85
86  endmodule
87
88  // ============================================================
89  // CNX file retrieval info
90  // ============================================================
91  // Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone IV E"
92  // Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
93  // Retrieval info: PRIVATE: new_diagram STRING "1"
94  // Retrieval info: LIBRARY: lpm lpm.lpm_components.all
95  // Retrieval info: CONSTANT: LPM_SIZE NUMERIC "4"
96  // Retrieval info: CONSTANT: LPM_TYPE STRING "LPM_MUX"
97  // Retrieval info: CONSTANT: LPM_WIDTH NUMERIC "8"
98  // Retrieval info: CONSTANT: LPM_WIDTHS NUMERIC "2"
99  // Retrieval info: USED_PORT: data0x 0 0 8 0 INPUT NODEFVAL "data0x[7..0]"
```

```
100 // Retrieval info: USED_PORT: data1x 0 0 8 0 INPUT NODEFVAL "data1x[7..0]"
101 // Retrieval info: USED_PORT: data2x 0 0 8 0 INPUT NODEFVAL "data2x[7..0]"
102 // Retrieval info: USED_PORT: data3x 0 0 8 0 INPUT NODEFVAL "data3x[7..0]"
103 // Retrieval info: USED_PORT: result 0 0 8 0 OUTPUT NODEFVAL "result[7..0]"
104 // Retrieval info: USED_PORT: sel 0 0 2 0 INPUT NODEFVAL "sel[1..0]"
105 // Retrieval info: CONNECT: @data 0 0 8 0 data0x 0 0 8 0
106 // Retrieval info: CONNECT: @data 0 0 8 8 data1x 0 0 8 0
107 // Retrieval info: CONNECT: @data 0 0 8 16 data2x 0 0 8 0
108 // Retrieval info: CONNECT: @data 0 0 8 24 data3x 0 0 8 0
109 // Retrieval info: CONNECT: @sel 0 0 2 0 sel 0 0 2 0
110 // Retrieval info: CONNECT: result 0 0 8 0 @result 0 0 8 0
111 // Retrieval info: GEN_FILE: TYPE_NORMAL I_Address_mux.v TRUE
112 // Retrieval info: GEN_FILE: TYPE_NORMAL I_Address_mux.inc FALSE
113 // Retrieval info: GEN_FILE: TYPE_NORMAL I_Address_mux.cmp FALSE
114 // Retrieval info: GEN_FILE: TYPE_NORMAL I_Address_mux.bsf FALSE
115 // Retrieval info: GEN_FILE: TYPE_NORMAL I_Address_mux_inst.v FALSE
116 // Retrieval info: GEN_FILE: TYPE_NORMAL I_Address_mux_bb.v TRUE
117 // Retrieval info: LIB_FILE: lpm
```

1.6.2 Instruction Memory

```
//File name : iram.v
//This module is used as the IRAM of size 256.

// megafunction wizard: %RAM: 1-PORT%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: altsyncram

// ============================================================
// File Name: iram.v
// Megafunction Name(s):
//       altsyncram
//
// Simulation Library Files(s):
//       altera_mf
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 15.0.0 Build 145 04/22/2015 SJ Full Version
// ************************************************************


//Copyright (C) 1991-2015 Altera Corporation. All rights reserved.
//Your use of Altera Corporation's design tools, logic functions
//and other software and tools, and its AMPP partner logic
//functions, and any output files from any of the foregoing
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Altera Program License
//Subscription Agreement, the Altera Quartus II License Agreement,
//the Altera MegaCore Function License Agreement, or other
//applicable license agreement, including, without limitation,
//that your use is for the sole purpose of programming logic
//devices manufactured by Altera and sold by Altera or its
//authorized distributors.  Please refer to the applicable
//agreement for further details.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module iram (
  address,
  clock,
  data,
  wren,
  q);

  input  [7:0]  address;
```

```
51    input     clock;
52    input   [7:0]  data;
53    input     wren;
54    output  [7:0]  q;
55  `ifndef ALTERA_RESERVED_QIS
56  // synopsys translate_off
57  `endif
58    tri1     clock;
59  `ifndef ALTERA_RESERVED_QIS
60  // synopsys translate_on
61  `endif
62
63    wire [7:0] sub_wire0;
64    wire [7:0] q = sub_wire0[7:0];
65
66    altsyncram  altsyncram_component (
67          .address_a (address),
68          .clock0 (clock),
69          .data_a (data),
70          .wren_a (wren),
71          .q_a (sub_wire0),
72          .aclr0 (1'b0),
73          .aclr1 (1'b0),
74          .address_b (1'b1),
75          .addressstall_a (1'b0),
76          .addressstall_b (1'b0),
77          .byteena_a (1'b1),
78          .byteena_b (1'b1),
79          .clock1 (1'b1),
80          .clocken0 (1'b1),
81          .clocken1 (1'b1),
82          .clocken2 (1'b1),
83          .clocken3 (1'b1),
84          .data_b (1'b1),
85          .eccstatus (),
86          .q_b (),
87          .rden_a (1'b1),
88          .rden_b (1'b1),
89          .wren_b (1'b0));
90    defparam
91      altsyncram_component.clock_enable_input_a = "BYPASS",
92      altsyncram_component.clock_enable_output_a = "BYPASS",
93      altsyncram_component.intended_device_family = "Cyclone IV E",
94      altsyncram_component.lpm_hint = "ENABLE_RUNTIME_MOD=NO",
95      altsyncram_component.lpm_type = "altsyncram",
96      altsyncram_component.numwords_a = 256,
97      altsyncram_component.operation_mode = "SINGLE_PORT",
98      altsyncram_component.outdata_aclr_a = "NONE",
99      altsyncram_component.outdata_reg_a = "CLOCK0",
100     altsyncram_component.power_up_uninitialized = "FALSE",
```

```
101     altsyncram_component.read_during_write_mode_port_a =
        ↪  "NEW_DATA_NO_NBE_READ",
102     altsyncram_component.widthad_a = 8,
103     altsyncram_component.width_a = 8,
104     altsyncram_component.width_byteena_a = 1;
105
106
107 endmodule
108
109 // ============================================================
110 // CNX file retrieval info
111 // ============================================================
112 // Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
113 // Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
114 // Retrieval info: PRIVATE: AclrByte NUMERIC "0"
115 // Retrieval info: PRIVATE: AclrData NUMERIC "0"
116 // Retrieval info: PRIVATE: AclrOutput NUMERIC "0"
117 // Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
118 // Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
119 // Retrieval info: PRIVATE: BlankMemory NUMERIC "1"
120 // Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
121 // Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
122 // Retrieval info: PRIVATE: Clken NUMERIC "0"
123 // Retrieval info: PRIVATE: DataBusSeparated NUMERIC "1"
124 // Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
125 // Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
126 // Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
127 // Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone IV E"
128 // Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
129 // Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
130 // Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
131 // Retrieval info: PRIVATE: MIFfilename STRING ""
132 // Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "256"
133 // Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
134 // Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_A NUMERIC "3"
135 // Retrieval info: PRIVATE: RegAddr NUMERIC "1"
136 // Retrieval info: PRIVATE: RegData NUMERIC "1"
137 // Retrieval info: PRIVATE: RegOutput NUMERIC "1"
138 // Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
139 // Retrieval info: PRIVATE: SingleClock NUMERIC "1"
140 // Retrieval info: PRIVATE: UseDQRAM NUMERIC "1"
141 // Retrieval info: PRIVATE: WRCONTROL_ACLR_A NUMERIC "0"
142 // Retrieval info: PRIVATE: WidthAddr NUMERIC "8"
143 // Retrieval info: PRIVATE: WidthData NUMERIC "8"
144 // Retrieval info: PRIVATE: rden NUMERIC "0"
145 // Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
146 // Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
147 // Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING "BYPASS"
148 // Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone IV E"
149 // Retrieval info: CONSTANT: LPM_HINT STRING "ENABLE_RUNTIME_MOD=NO"
150 // Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
```

```
151 // Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "256"
152 // Retrieval info: CONSTANT: OPERATION_MODE STRING "SINGLE_PORT"
153 // Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
154 // Retrieval info: CONSTANT: OUTDATA_REG_A STRING "CLOCK0"
155 // Retrieval info: CONSTANT: POWER_UP_UNINITIALIZED STRING "FALSE"
156 // Retrieval info: CONSTANT: READ_DURING_WRITE_MODE_PORT_A STRING
     ↪  "NEW_DATA_NO_NBE_READ"
157 // Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "8"
158 // Retrieval info: CONSTANT: WIDTH_A NUMERIC "8"
159 // Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
160 // Retrieval info: USED_PORT: address 0 0 8 0 INPUT NODEFVAL "address[7..0]"
161 // Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
162 // Retrieval info: USED_PORT: data 0 0 8 0 INPUT NODEFVAL "data[7..0]"
163 // Retrieval info: USED_PORT: q 0 0 8 0 OUTPUT NODEFVAL "q[7..0]"
164 // Retrieval info: USED_PORT: wren 0 0 0 0 INPUT NODEFVAL "wren"
165 // Retrieval info: CONNECT: @address_a 0 0 8 0 address 0 0 8 0
166 // Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
167 // Retrieval info: CONNECT: @data_a 0 0 8 0 data 0 0 8 0
168 // Retrieval info: CONNECT: @wren_a 0 0 0 0 wren 0 0 0 0
169 // Retrieval info: CONNECT: q 0 0 8 0 @q_a 0 0 8 0
170 // Retrieval info: GEN_FILE: TYPE_NORMAL iram.v TRUE
171 // Retrieval info: GEN_FILE: TYPE_NORMAL iram.inc FALSE
172 // Retrieval info: GEN_FILE: TYPE_NORMAL iram.cmp FALSE
173 // Retrieval info: GEN_FILE: TYPE_NORMAL iram.bsf FALSE
174 // Retrieval info: GEN_FILE: TYPE_NORMAL iram_inst.v FALSE
175 // Retrieval info: GEN_FILE: TYPE_NORMAL iram_bb.v TRUE
176 // Retrieval info: LIB_FILE: altera_mf
```

1.7 Communication

1.7.1 UART: RX

```
//File name : uart_rx.v
//This module is the UART receiver which forms serial data received form
//computer to bytes.


module uart_rx
  //#(parameter CLKS_PER_BIT)
  (
   input        clk,
   input        i_Rx_Serial,
   output       o_Rx_DV,
   output [7:0] o_Rx_Byte
   );

  parameter c_CLKS_PER_BIT    = 87;
  parameter s_IDLE         = 3'b000;
  parameter s_RX_START_BIT = 3'b001;
  parameter s_RX_DATA_BITS = 3'b010;
  parameter s_RX_STOP_BIT  = 3'b011;
  parameter s_CLEANUP      = 3'b100;

  reg           r_Rx_Data_R = 1'b1;
  reg           r_Rx_Data   = 1'b1;

  reg [7:0]     r_Clock_Count = 0;
  reg [2:0]     r_Bit_Index   = 0; //8 bits total
  reg [7:0]     r_Rx_Byte     = 0;
  reg           r_Rx_DV       = 0;
  reg [2:0]     r_SM_Main     = 0;

  // Purpose: Double-register the incoming data.
  // This allows it to be used in the UART RX Clock Domain.
  // (It removes problems caused by metastability)
  //assign clk = in_Clock;

  always @(posedge clk)
    begin
      r_Rx_Data_R <= i_Rx_Serial;
      r_Rx_Data   <= r_Rx_Data_R;
    end


  // Purpose: Control RX state machine
  always @(posedge clk)
    begin

      case (r_SM_Main)
        s_IDLE :
```

```
49            begin
50              r_Rx_DV       <= 1'b0;
51              r_Clock_Count <= 0;
52              r_Bit_Index   <= 0;
53
54              if (r_Rx_Data == 1'b0)          // Start bit detected
55                r_SM_Main <= s_RX_START_BIT;
56              else
57                r_SM_Main <= s_IDLE;
58            end
59
60          // Check middle of start bit to make sure it's still low
61          s_RX_START_BIT :
62            begin
63              if (r_Clock_Count == (87-1)/2)
64                begin
65                  if (r_Rx_Data == 1'b0)
66                    begin
67                      r_Clock_Count <= 0;  // reset counter, found the middle
68                      r_SM_Main     <= s_RX_DATA_BITS;
69                    end
70                  else
71                    r_SM_Main <= s_IDLE;
72                end
73              else
74                begin
75                  r_Clock_Count <= r_Clock_Count + 1;
76                  r_SM_Main     <= s_RX_START_BIT;
77                end
78            end // case: s_RX_START_BIT
79
80
81          // Wait CLKS_PER_BIT-1 clock cycles to sample serial data
82          s_RX_DATA_BITS :
83            begin
84              if (r_Clock_Count < 87-1)
85                begin
86                  r_Clock_Count <= r_Clock_Count + 1;
87                  r_SM_Main     <= s_RX_DATA_BITS;
88                end
89              else
90                begin
91                  r_Clock_Count          <= 0;
92                  r_Rx_Byte[r_Bit_Index] <= r_Rx_Data;
93
94                  // Check if we have received all bits
95                  if (r_Bit_Index < 7)
96                    begin
97                      r_Bit_Index <= r_Bit_Index + 1;
98                      r_SM_Main   <= s_RX_DATA_BITS;
99                    end
```

```
100                 else
101                   begin
102                     r_Bit_Index <= 0;
103                     r_SM_Main   <= s_RX_STOP_BIT;
104                   end
105               end
106           end // case: s_RX_DATA_BITS
107
108
109         // Receive Stop bit.  Stop bit = 1
110         s_RX_STOP_BIT :
111           begin
112             // Wait CLKS_PER_BIT-1 clock cycles for Stop bit to finish
113             if (r_Clock_Count < 87-1)
114               begin
115                 r_Clock_Count <= r_Clock_Count + 1;
116                 r_SM_Main     <= s_RX_STOP_BIT;
117               end
118             else
119               begin
120                 r_Rx_DV       <= 1'b1;
121                 r_Clock_Count <= 0;
122                 r_SM_Main     <= s_CLEANUP;
123               end
124           end // case: s_RX_STOP_BIT
125
126
127         // Stay here 1 clock
128         s_CLEANUP :
129           begin
130             r_SM_Main <= s_IDLE;
131             r_Rx_DV   <= 1'b0;
132           end
133
134
135         default :
136           r_SM_Main <= s_IDLE;
137
138       endcase
139     end
140
141   assign o_Rx_DV   = r_Rx_DV;
142   assign o_Rx_Byte = r_Rx_Byte;
143
144 endmodule // uart_rx
145
146 //Transmitter
147
148 //
149 //module uart_tx
150 //  (
```

```
151 //   input       i_Clock,
152 //   input       i_Tx_DV,
153 //   input [7:0] i_Tx_Byte,
154 //   output      o_Tx_Active,
155 //   output reg  o_Tx_Serial,
156 //   output      o_Tx_Done
157 //   );
158 //
159 //  parameter s_IDLE         = 3'b000;
160 //  parameter s_TX_START_BIT = 3'b001;
161 //  parameter s_TX_DATA_BITS = 3'b010;
162 //  parameter s_TX_STOP_BIT  = 3'b011;
163 //  parameter s_CLEANUP      = 3'b100;
164 //
165 //  reg [2:0]    r_SM_Main     = 0;
166 //  reg [7:0]    r_Clock_Count = 0;
167 //  reg [2:0]    r_Bit_Index   = 0;
168 //  reg [7:0]    r_Tx_Data     = 0;
169 //  reg          r_Tx_Done     = 0;
170 //  reg          r_Tx_Active   = 0;
171 //
172 //  always @(posedge i_Clock)
173 //    begin
174 //
175 //      case (r_SM_Main)
176 //        s_IDLE :
177 //          begin
178 //            o_Tx_Serial   <= 1'b1;         // Drive Line High for Idle
179 //            r_Tx_Done     <= 1'b0;
180 //            r_Clock_Count <= 0;
181 //            r_Bit_Index   <= 0;
182 //
183 //            if (i_Tx_DV == 1'b1)
184 //              begin
185 //                r_Tx_Active <= 1'b1;
186 //                r_Tx_Data   <= i_Tx_Byte;
187 //                r_SM_Main   <= s_TX_START_BIT;
188 //              end
189 //            else
190 //              r_SM_Main <= s_IDLE;
191 //          end // case: s_IDLE
192 //
193 //
194 //        // Send out Start Bit. Start bit = 0
195 //        s_TX_START_BIT :
196 //          begin
197 //            o_Tx_Serial <= 1'b0;
198 //
199 //            // Wait CLKS_PER_BIT-1 clock cycles for start bit to finish
200 //            if (r_Clock_Count < 87-1)
201 //              begin
```

```
202 //                r_Clock_Count <= r_Clock_Count + 1;
203 //                r_SM_Main     <= s_TX_START_BIT;
204 //              end
205 //            else
206 //              begin
207 //                r_Clock_Count <= 0;
208 //                r_SM_Main     <= s_TX_DATA_BITS;
209 //              end
210 //          end // case: s_TX_START_BIT
211 //
212 //
213 //        // Wait CLKS_PER_BIT-1 clock cycles for data bits to finish
214 //        s_TX_DATA_BITS :
215 //          begin
216 //            o_Tx_Serial <= r_Tx_Data[r_Bit_Index];
217 //
218 //            if (r_Clock_Count < 87-1)
219 //              begin
220 //                r_Clock_Count <= r_Clock_Count + 1;
221 //                r_SM_Main     <= s_TX_DATA_BITS;
222 //              end
223 //            else
224 //              begin
225 //                r_Clock_Count <= 0;
226 //
227 //                // Check if we have sent out all bits
228 //                if (r_Bit_Index < 7)
229 //                  begin
230 //                    r_Bit_Index <= r_Bit_Index + 1;
231 //                    r_SM_Main   <= s_TX_DATA_BITS;
232 //                  end
233 //                else
234 //                  begin
235 //                    r_Bit_Index <= 0;
236 //                    r_SM_Main   <= s_TX_STOP_BIT;
237 //                  end
238 //              end
239 //          end // case: s_TX_DATA_BITS
240 //
241 //
242 //        // Send out Stop bit.  Stop bit = 1
243 //        s_TX_STOP_BIT :
244 //          begin
245 //            o_Tx_Serial <= 1'b1;
246 //
247 //            // Wait CLKS_PER_BIT-1 clock cycles for Stop bit to finish
248 //            if (r_Clock_Count < 87-1)
249 //              begin
250 //                r_Clock_Count <= r_Clock_Count + 1;
251 //                r_SM_Main     <= s_TX_STOP_BIT;
252 //              end
```

```
253 //             else
254 //               begin
255 //                 r_Tx_Done     <= 1'b1;
256 //                 r_Clock_Count <= 0;
257 //                 r_SM_Main     <= s_CLEANUP;
258 //                 r_Tx_Active   <= 1'b0;
259 //               end
260 //           end // case: s_Tx_STOP_BIT
261 //
262 //
263 //         // Stay here 1 clock
264 //         s_CLEANUP :
265 //           begin
266 //             r_Tx_Done <= 1'b1;
267 //             r_SM_Main <= s_IDLE;
268 //           end
269 //
270 //
271 //         default :
272 //           r_SM_Main <= s_IDLE;
273 //
274 //       endcase
275 //     end
276 //
277 //   assign o_Tx_Active = r_Tx_Active;
278 //   assign o_Tx_Done   = r_Tx_Done;
279 //
280 //endmodule
```

1.7.2 UART: TX

```
//File name : uart_tx.v
//This module is the UART tranmitter which transmit a byte to
//the computer serially.

module uart_tx
  (
   input       i_Clock,
   input       i_Tx_DV,
   input [7:0] i_Tx_Byte,
   output      o_Tx_Active,
   output reg  o_Tx_Serial,
   output      o_Tx_Done
   );

  parameter s_IDLE         = 3'b000;
  parameter s_TX_START_BIT = 3'b001;
  parameter s_TX_DATA_BITS = 3'b010;
  parameter s_TX_STOP_BIT  = 3'b011;
  parameter s_CLEANUP      = 3'b100;
  parameter s_CLEANUP2     = 3'b101;
  parameter s_CLEANUP3     = 3'b110;
  parameter s_TX_START_BIT_0=3'b111;
  parameter CLKS_PER_BIT   = 87;

  reg [2:0]    r_SM_Main     = 0;
  reg [7:0]    r_Clock_Count = 0;
  reg [2:0]    r_Bit_Index   = 0;
  reg [7:0]    r_Tx_Data     = 0;
  reg          r_Tx_Done     = 0;
  reg          r_Tx_Active   = 0;

  always @(posedge i_Clock)
    begin

      case (r_SM_Main)
        s_IDLE :
          begin
            o_Tx_Serial   <= 1'b1;         // Drive Line High for Idle
            r_Tx_Done     <= 1'b0;
            r_Clock_Count <= 0;
            r_Bit_Index   <= 0;

            if (i_Tx_DV == 1'b1)
              begin
                r_Tx_Active <= 1'b1;
                r_SM_Main   <= s_TX_START_BIT_0;
              end
            else
              r_SM_Main <= s_IDLE;
          end // case: s_IDLE
```

```
51
52
53          // Send out Start Bit. Start bit = 0
54        s_TX_START_BIT_0:
55          begin
56                  r_Tx_Data   <= i_Tx_Byte;
57                  r_SM_Main   <= s_TX_START_BIT;
58         end
59          s_TX_START_BIT :
60            begin
61              o_Tx_Serial <= 1'b0;
62
63              // Wait CLKS_PER_BIT-1 clock cycles for start bit to finish
64              if (r_Clock_Count < CLKS_PER_BIT-1)
65                begin
66                  r_Clock_Count <= r_Clock_Count + 1;
67                  r_SM_Main     <= s_TX_START_BIT;
68                end
69              else
70                begin
71                  r_Clock_Count <= 0;
72                  r_SM_Main     <= s_TX_DATA_BITS;
73                end
74            end // case: s_TX_START_BIT
75
76
77          // Wait CLKS_PER_BIT-1 clock cycles for data bits to finish
78          s_TX_DATA_BITS :
79            begin
80              o_Tx_Serial <= r_Tx_Data[r_Bit_Index];
81
82              if (r_Clock_Count < CLKS_PER_BIT-1)
83                begin
84                  r_Clock_Count <= r_Clock_Count + 1;
85                  r_SM_Main     <= s_TX_DATA_BITS;
86                end
87              else
88                begin
89                  r_Clock_Count <= 0;
90
91                  // Check if we have sent out all bits
92                  if (r_Bit_Index < 7)
93                    begin
94                      r_Bit_Index <= r_Bit_Index + 1;
95                      r_SM_Main   <= s_TX_DATA_BITS;
96                    end
97                  else
98                    begin
99                      r_Bit_Index <= 0;
100                     r_SM_Main   <= s_TX_STOP_BIT;
101                   end
```

```
102               end
103           end // case: s_TX_DATA_BITS
104
105
106         // Send out Stop bit.  Stop bit = 1
107         s_TX_STOP_BIT :
108           begin
109             o_Tx_Serial <= 1'b1;
110
111             // Wait CLKS_PER_BIT-1 clock cycles for Stop bit to finish
112             if (r_Clock_Count < CLKS_PER_BIT-1)
113               begin
114                 r_Clock_Count <= r_Clock_Count + 1;
115                 r_SM_Main     <= s_TX_STOP_BIT;
116               end
117             else
118               begin
119                 r_Tx_Done     <= 1'b1;
120                 r_Clock_Count <= 0;
121                 r_SM_Main     <= s_CLEANUP;
122                 r_Tx_Active   <= 1'b0;
123               end
124           end // case: s_Tx_STOP_BIT
125
126
127         // Stay here 1 clock
128         s_CLEANUP :
129           begin
130             r_Tx_Done <= 1'b0; //this tick is only high for one clock cycle
131             r_SM_Main <= s_CLEANUP2;
132           end
133
134         s_CLEANUP2 :
135           begin
136             r_SM_Main <= s_CLEANUP3;
137           end
138
139         s_CLEANUP3 :
140           begin
141             r_SM_Main <= s_IDLE;
142           end
143
144         default :
145           r_SM_Main <= s_IDLE;
146
147       endcase
148     end
149
150   assign o_Tx_Active = r_Tx_Active;
151   assign o_Tx_Done   = r_Tx_Done;
152
```

```
153 endmodule
```

1.7.3 TX Controller

```
//File name : Tx_modifier.v
//This module modifies the signals between UART transmitter and data_retreiver module
//inorder to crop the image/data sent back to the computer.

module Tx_modifier(
          Tx_tick_retreiver,
          wen_retreiver,
          Tx_tick_Tx,
          wen_Tx,
          end_address,
          address);


input        wen_retreiver;
input        Tx_tick_Tx;
input [17:0] address;
input [17:0] end_address;

output wen_Tx;
output Tx_tick_retreiver;


reg mux_out=0;

//hijack transmit signals when needed

two_way_mux for_wen(
  .data0(wen_retreiver),
  .data1(0),
  .sel(mux_out),
  .result(wen_Tx));

two_way_mux for_tx_tick(
  .data0(Tx_tick_Tx),
  .data1(1),
  .sel(mux_out),
  .result(Tx_tick_retreiver));


always @ (address)
   begin
     if((address[17:9] <= end_address[17:9]) & (address[8:0] <=end_address[8:0])) mux_out<=0; //if the address is in range maintain normal transmition
     else mux_out<=1;  //if address is out of range hijack the 2 wires connecting Tx and retreiver
   end

endmodule
```

1.7.4 Data Retriever

```
1  //File name : Data_retreiver.v
2  //This module used to transmit the processed data one by one to the UART
   ↪  transmitter.
3
4  module Data_retriever(clk,addr,wen_Tx,fin,start,Tx_tick_from_tx,end_add);
5
6  input        clk,Tx_tick_from_tx,start;
7  input [17:0] end_add;
8
9  output reg [17:0] addr=18'b0;
10 output            wen_Tx;
11 output reg        fin=0;
12
13 reg       wen=0;
14 reg [1:0] STATE=2'b00;
15 reg       a=1;
16 reg       b=1;
17 wire      Tx_tick;
18
19 parameter IDLE=2'b0;
20 parameter TRANSMITTING=2'b01;
21 parameter DONE=2'b10;
22
23
24 Tx_modifier  Tx_modifier(
25           .Tx_tick_retreiver(Tx_tick),
26           .wen_retreiver(wen),
27           .Tx_tick_Tx(Tx_tick_from_tx),
28           .wen_Tx(wen_Tx),
29           .end_address(end_add), //change this inorder to SEND different sizes
             ↪  of images   .put "end_add"/18'b111111111111111111 in brackets
30           .address(addr)
31           );
32
33
34
35 always @(posedge clk)
36  case(STATE)
37    IDLE:
38     begin
39       addr<=18'b0;
40       if (start)
41         begin
42           fin<=0;
43           STATE<=TRANSMITTING;
44         end
45     end
46    TRANSMITTING:
47     begin
48       wen<=1;
```

```
49          if (addr==18'd262143 )
50            begin
51              STATE<=DONE;
52            end
53          else if(Tx_tick==1)
54            begin
55              addr<=addr+1;
56            end
57       end
58      DONE:
59          if(Tx_tick==1)
60            begin
61              STATE<=IDLE;
62              fin<=1;
63              wen<=0;
64            end
65      default: STATE<=IDLE;
66   endcase
67
68  endmodule
```

1.7.5 Data Writer

```
//File name : Data_writer.v
//This module used to store incoming data from UART receiver onto IRAM/DRAM.

module Data_writer(clk,Rx_tick,Din,Wen,Addr,Dout,fin,memory_size);

input        Rx_tick,clk;
input [7:0]  Din;
input [17:0] memory_size;

output reg [17:0] Addr=18'b0;
output reg [7:0]  Dout;
output reg        Wen=1'b0;
output reg        fin=0;

reg       flag = 0;
reg [1:0] STATE=2'b0;

parameter IDLE=2'b0;
parameter STORING1=2'b01;
parameter STORING2=2'b10;
parameter DONE=2'b11;

always @(posedge clk)
begin
  case(STATE)
    IDLE:
      if(Rx_tick==1)
        begin
          fin<=0;
          Wen<=1;
          Dout<=Din;
          STATE<=STORING2;
        end
    STORING1:
      if(Rx_tick==1)
        begin
          Wen<=1;
          Dout<=Din;
          Addr<=Addr+1;
          STATE<=STORING2;
        end
    STORING2:
      begin
        Wen<=0;
        if(Addr==memory_size) STATE<=DONE;
        else                  STATE<=STORING1;
      end
    DONE:
      begin
      Addr<=0;
```

```
51          fin<=1;
52          Wen<=0;
53          STATE<=IDLE;
54          end
55        default:STATE<=IDLE;
56      endcase
57
58  end
59
60  endmodule
```

1.8 Other Modules

1.8.1 Binary to BCD Converter

```
1  //File name : bi2bcd.v
2  //This module is used to decode 8 bit binary number inorder to feed to
3  //3 seven segment display units.
4
5  module bi2bcd(din,dout2,dout1,dout0);
6
7
8  input  [7:0] din;
9  output [6:0] dout0;
10 output [6:0] dout1;
11 output [6:0] dout2;
12
13 reg [3:0]  counter=3'b0;
14 reg [19:0] shifter=20'd0;
15
16 decoder d0(.din(shifter[11:8]),.dout(dout0));
17 decoder d1(.din(shifter[15:12]),.dout(dout1));
18 decoder d2(.din(shifter[19:16]),.dout(dout2));
19
20 always @(din)
21 begin
22   shifter[7:0]=din;
23   shifter[19:8]=12'b0;
24   for (counter=4'd0;counter<4'd8;counter=counter+1) begin
25     if(shifter[11:8]>4'd4)
26       begin
27         shifter[11:8]=shifter[11:8]+4'd3;
28       end
29     if(shifter[15:12]>4'd4)
30       begin
31         shifter[15:12]=shifter[15:12]+4'd3;
32       end
33     if(shifter[19:16]>4'd4)
34       begin
35         shifter[19:16]=shifter[19:16]+4'd3;
36       end
37     shifter=shifter<<1;
38   end
39 end
40
41
42
43 endmodule
```

1.8.2 Clock Control

```
//File name : clock_control.v
//This module drives the project with different clocks of choice.

module clock_control(
          in_clock, //50MHz
          sel,
          mode,
          manual,
          out_clock  //10MHz,1Hz,Manual,25MHz
);

input       in_clock;
input       manual;
input [1:0] mode,sel;

output      out_clock;

reg   [1:0] select=2'd0;

wire _1MHz,_10MHz,_25MHz;

parameter _10mhz=2'b00;
parameter _1hz=2'b01;
parameter _manual=2'b10;
parameter _25mhz=2'b11;

//Selects the needed clock to give out

four_way_mux four_way_mux(
       .data0(_10MHz),
      .data1(_1Hz),
      .data2(manual),
      .data3(_25MHz),
      .sel(select),
      .result(out_clock));

//Convert 10MHz clock to 1Hz clock

clock_divider _10MHz_to_1Hz(
      .inclk(_10MHz),
      .ena(1),
      .clk(_1Hz));

//Convert 50MHz clock to 10MHz clock

pll _50MHz_to_10MHz(
  .inclk0(in_clock),
  .c0(_10MHz));

//Convert 50MHz clock to 25MHz clock
```

```
51
52  pll_25mhz _50MHz_to_25MHz(
53    .inclk0(in_clock),
54    .c0(_25MHz));
55
56  always @(in_clock)
57    begin
58       if(mode==2'b10) //processor mode
59             begin
60             case (sel)
61                 _10mhz :  select<=2'b00;
62                 _1hz   :  select<=2'b01;
63                 _manual:  select<=2'b10;
64                 _25mhz :  select<=2'b11;
65                 default:  select<=2'b00;
66             endcase
67           end
68         else select<=2'd0; //any other mode use 10MHz clock
69    end
70
71  endmodule
```

1.8.3 Clock Divider

```
//File name : clock_divider.v
//This module is used to convert a 10MHz clock to give out a 1Hz clock pulse.
//Instantiated inside the clock_control module.

module clock_divider(inclk,ena,clk);

  parameter maxcount=23'd5000000;// input 10MHz clock and output 1Hz clk

  input inclk;
  input ena;
  output reg clk=1;

  reg [22:0] count=23'd0;

  always @ (posedge inclk )
    begin
    if (ena)
      begin
        if (count==maxcount)
          begin
          clk=~clk;
          count=23'd0;
          end
        else
          begin
          count=count+1;
          end
      end
    else
      begin
      clk=0;
      end
    end

endmodule
```

1.8.4 Debouncer

```
//File name : debouner.v
//This module is used to debounce the push buttons in the FPGA board.

module debouncer(button_in,clk,button_out);

input clk,button_in;
output reg button_out=1;

reg [3:0] counter=4'd0;

always @(posedge clk)
begin
  if (button_in==0)
    begin
      counter<=counter+1;
      if (counter==4'b1111) button_out<=0;
    end
  else
    begin
      counter<=4'd0;
      button_out<=1;
    end
end

endmodule
```

1.8.5 Decoder

```
//File name : decoder.v
//This module is used to decode a 4 bit pattern to display
//a digit on a single seven segment display.
//Instantiated in the bi2bcd module.

module decoder(din,dout);

input [3:0] din;
output reg [6:0] dout;



always @(din)
case(din)
  4'd0:dout<=7'b1000000;
  4'd1:dout<=7'b1111001;
  4'd2:dout<=7'b0100100;
  4'd3:dout<=7'b0110000;
  4'd4:dout<=7'b0011001;
  4'd5:dout<=7'b0010010;
  4'd6:dout<=7'b0000010;
  4'd7:dout<=7'b1111000;
  4'd8:dout<=7'b0000000;
  4'd9:dout<=7'b0011000;
endcase

endmodule
```

1.8.6 Four Way Mux

```
//File name : four_way_mux.v
//This module is instantiated in the clock controll module
//inorder to multiplex from 4 available clocks.

// megafunction wizard: %LPM_MUX%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: LPM_MUX

// ============================================================
// File Name: four_way_mux.v
// Megafunction Name(s):
//       LPM_MUX
//
// Simulation Library Files(s):
//       lpm
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 15.0.0 Build 145 04/22/2015 SJ Full Version
// ************************************************************


//Copyright (C) 1991-2015 Altera Corporation. All rights reserved.
//Your use of Altera Corporation's design tools, logic functions
//and other software and tools, and its AMPP partner logic
//functions, and any output files from any of the foregoing
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Altera Program License
//Subscription Agreement, the Altera Quartus II License Agreement,
//the Altera MegaCore Function License Agreement, or other
//applicable license agreement, including, without limitation,
//that your use is for the sole purpose of programming logic
//devices manufactured by Altera and sold by Altera or its
//authorized distributors.  Please refer to the applicable
//agreement for further details.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module four_way_mux (
  data0,
  data1,
  data2,
  data3,
  sel,
  result);
```

```
51
52    input    data0;
53    input    data1;
54    input    data2;
55    input    data3;
56    input  [1:0]  sel;
57    output    result;
58
59    wire [0:0] sub_wire5;
60    wire  sub_wire4 = data3;
61    wire  sub_wire3 = data2;
62    wire  sub_wire2 = data1;
63    wire  sub_wire0 = data0;
64    wire [3:0] sub_wire1 = {sub_wire4, sub_wire3, sub_wire2, sub_wire0};
65    wire [0:0] sub_wire6 = sub_wire5[0:0];
66    wire  result = sub_wire6;
67
68    lpm_mux  LPM_MUX_component (
69          .data (sub_wire1),
70          .sel (sel),
71          .result (sub_wire5)
72          // synopsys translate_off
73          ,
74          .aclr (),
75          .clken (),
76          .clock ()
77          // synopsys translate_on
78          );
79    defparam
80      LPM_MUX_component.lpm_size = 4,
81      LPM_MUX_component.lpm_type = "LPM_MUX",
82      LPM_MUX_component.lpm_width = 1,
83      LPM_MUX_component.lpm_widths = 2;
84
85
86  endmodule
87
88  // ============================================================
89  // CNX file retrieval info
90  // ============================================================
91  // Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone IV E"
92  // Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
93  // Retrieval info: PRIVATE: new_diagram STRING "1"
94  // Retrieval info: LIBRARY: lpm lpm.lpm_components.all
95  // Retrieval info: CONSTANT: LPM_SIZE NUMERIC "4"
96  // Retrieval info: CONSTANT: LPM_TYPE STRING "LPM_MUX"
97  // Retrieval info: CONSTANT: LPM_WIDTH NUMERIC "1"
98  // Retrieval info: CONSTANT: LPM_WIDTHS NUMERIC "2"
99  // Retrieval info: USED_PORT: data0 0 0 0 0 INPUT NODEFVAL "data0"
100 // Retrieval info: USED_PORT: data1 0 0 0 0 INPUT NODEFVAL "data1"
101 // Retrieval info: USED_PORT: data2 0 0 0 0 INPUT NODEFVAL "data2"
```

```
// Retrieval info: USED_PORT: data3 0 0 0 0 INPUT NODEFVAL "data3"
// Retrieval info: USED_PORT: result 0 0 0 0 OUTPUT NODEFVAL "result"
// Retrieval info: USED_PORT: sel 0 0 2 0 INPUT NODEFVAL "sel[1..0]"
// Retrieval info: CONNECT: @data 0 0 1 0 data0 0 0 0 0
// Retrieval info: CONNECT: @data 0 0 1 1 data1 0 0 0 0
// Retrieval info: CONNECT: @data 0 0 1 2 data2 0 0 0 0
// Retrieval info: CONNECT: @data 0 0 1 3 data3 0 0 0 0
// Retrieval info: CONNECT: @sel 0 0 2 0 sel 0 0 2 0
// Retrieval info: CONNECT: result 0 0 0 0 @result 0 0 1 0
// Retrieval info: GEN_FILE: TYPE_NORMAL four_way_mux.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL four_way_mux.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL four_way_mux.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL four_way_mux.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL four_way_mux_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL four_way_mux_bb.v TRUE
// Retrieval info: LIB_FILE: lpm
```

1.8.7 Key Splitter

```
//File name : key_split.v
//This module is used to split the signal of a push button to do different
↪  channels
//in different modes of operations.

module key_split(in,Tx_out,p_out,i_d_out,enable,selector);

input in;
input enable;
input [1:0] selector;
output reg Tx_out=1,p_out=1,i_d_out=1;

parameter  DIRECT_TO_IDLE =2'b00;
parameter  DIRECT_TO_Rx   =2'b01;
parameter  DIRECT_TO_P    =2'b10;
parameter  DIRECT_TO_Tx   =2'b11;

always @(in,selector,enable)
  if (enable==1)
    begin
      case(selector)
        DIRECT_TO_IDLE:
          begin
            Tx_out  <= 1;
            p_out   <= 1;
            i_d_out <= in;
          end
        DIRECT_TO_Rx:
          begin
            Tx_out  <= 1;
            p_out   <= 1;
            i_d_out <= in;
          end
        DIRECT_TO_P:
          begin
            Tx_out  <= 1;
            p_out   <= in;
            i_d_out <= 1;
          end
        DIRECT_TO_Tx:
          begin
            Tx_out  <= in;
            p_out   <= 1;
            i_d_out <= 1;
          end
        default:
          begin
            Tx_out  <= 1;
            p_out   <= 1;
            i_d_out <= 1;
```

```
50              end
51          endcase
52        end
53      else
54        begin
55          Tx_out  <= 1;
56          p_out   <= 1;
57          i_d_out <= 1;
58        end
59  endmodule
```

1.8.8 PLL for Clock Control

```
//File name : pll.v
//This module is used to convert a 50MHz clock to 10MHz clock.
//instantiated in the clock control module.

// megafunction wizard: %ALTPLL%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: altpll

// ============================================================
// File Name: pll.v
// Megafunction Name(s):
//       altpll
//
// Simulation Library Files(s):
//       altera_mf
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 17.1.0 Build 590 10/25/2017 SJ Lite Edition
// ************************************************************


//Copyright (C) 2017  Intel Corporation. All rights reserved.
//Your use of Intel Corporation's design tools, logic functions
//and other software and tools, and its AMPP partner logic
//functions, and any output files from any of the foregoing
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Intel Program License
//Subscription Agreement, the Intel Quartus Prime License Agreement,
//the Intel FPGA IP License Agreement, or other applicable license
//agreement, including, without limitation, that your use is for
//the sole purpose of programming logic devices manufactured by
//Intel and sold by Intel or its authorized distributors.  Please
//refer to the applicable agreement for further details.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module pll (
  areset,
  inclk0,
  c0);

  input    areset;
  input    inclk0;
  output    c0;
```

```
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
  tri0    areset;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
`endif

  wire [0:0] sub_wire2 = 1'h0;
  wire [4:0] sub_wire3;
  wire  sub_wire0 = inclk0;
  wire [1:0] sub_wire1 = {sub_wire2, sub_wire0};
  wire [0:0] sub_wire4 = sub_wire3[0:0];
  wire  c0 = sub_wire4;

  altpll  altpll_component (
        .areset (areset),
        .inclk (sub_wire1),
        .clk (sub_wire3),
        .activeclock (),
        .clkbad (),
        .clkena ({6{1'b1}}),
        .clkloss (),
        .clkswitch (1'b0),
        .configupdate (1'b0),
        .enable0 (),
        .enable1 (),
        .extclk (),
        .extclkena ({4{1'b1}}),
        .fbin (1'b1),
        .fbmimicbidir (),
        .fbout (),
        .fref (),
        .icdrclk (),
        .locked (),
        .pfdena (1'b1),
        .phasecounterselect ({4{1'b1}}),
        .phasedone (),
        .phasestep (1'b1),
        .phaseupdown (1'b1),
        .pllena (1'b1),
        .scanaclr (1'b0),
        .scanclk (1'b0),
        .scanclkena (1'b1),
        .scandata (1'b0),
        .scandataout (),
        .scandone (),
        .scanread (1'b0),
        .scanwrite (1'b0),
        .sclkout0 (),
        .sclkout1 (),
```

```
102         .vcooverrange (),
103         .vcounderrange ());
104     defparam
105       altpll_component.bandwidth_type = "AUTO",
106       altpll_component.clk0_divide_by = 5,
107       altpll_component.clk0_duty_cycle = 50,
108       altpll_component.clk0_multiply_by = 1,
109       altpll_component.clk0_phase_shift = "0",
110       altpll_component.compensate_clock = "CLK0",
111       altpll_component.inclk0_input_frequency = 20000,
112       altpll_component.intended_device_family = "Cyclone IV E",
113       altpll_component.lpm_hint = "CBX_MODULE_PREFIX=pll",
114       altpll_component.lpm_type = "altpll",
115       altpll_component.operation_mode = "NORMAL",
116       altpll_component.pll_type = "AUTO",
117       altpll_component.port_activeclock = "PORT_UNUSED",
118       altpll_component.port_areset = "PORT_USED",
119       altpll_component.port_clkbad0 = "PORT_UNUSED",
120       altpll_component.port_clkbad1 = "PORT_UNUSED",
121       altpll_component.port_clkloss = "PORT_UNUSED",
122       altpll_component.port_clkswitch = "PORT_UNUSED",
123       altpll_component.port_configupdate = "PORT_UNUSED",
124       altpll_component.port_fbin = "PORT_UNUSED",
125       altpll_component.port_inclk0 = "PORT_USED",
126       altpll_component.port_inclk1 = "PORT_UNUSED",
127       altpll_component.port_locked = "PORT_UNUSED",
128       altpll_component.port_pfdena = "PORT_UNUSED",
129       altpll_component.port_phasecounterselect = "PORT_UNUSED",
130       altpll_component.port_phasedone = "PORT_UNUSED",
131       altpll_component.port_phasestep = "PORT_UNUSED",
132       altpll_component.port_phaseupdown = "PORT_UNUSED",
133       altpll_component.port_pllena = "PORT_UNUSED",
134       altpll_component.port_scanaclr = "PORT_UNUSED",
135       altpll_component.port_scanclk = "PORT_UNUSED",
136       altpll_component.port_scanclkena = "PORT_UNUSED",
137       altpll_component.port_scandata = "PORT_UNUSED",
138       altpll_component.port_scandataout = "PORT_UNUSED",
139       altpll_component.port_scandone = "PORT_UNUSED",
140       altpll_component.port_scanread = "PORT_UNUSED",
141       altpll_component.port_scanwrite = "PORT_UNUSED",
142       altpll_component.port_clk0 = "PORT_USED",
143       altpll_component.port_clk1 = "PORT_UNUSED",
144       altpll_component.port_clk2 = "PORT_UNUSED",
145       altpll_component.port_clk3 = "PORT_UNUSED",
146       altpll_component.port_clk4 = "PORT_UNUSED",
147       altpll_component.port_clk5 = "PORT_UNUSED",
148       altpll_component.port_clkena0 = "PORT_UNUSED",
149       altpll_component.port_clkena1 = "PORT_UNUSED",
150       altpll_component.port_clkena2 = "PORT_UNUSED",
151       altpll_component.port_clkena3 = "PORT_UNUSED",
152       altpll_component.port_clkena4 = "PORT_UNUSED",
```

```
153     altpll_component.port_clkena5 = "PORT_UNUSED",
154     altpll_component.port_extclk0 = "PORT_UNUSED",
155     altpll_component.port_extclk1 = "PORT_UNUSED",
156     altpll_component.port_extclk2 = "PORT_UNUSED",
157     altpll_component.port_extclk3 = "PORT_UNUSED",
158     altpll_component.width_clock = 5;
159
160
161 endmodule
162
163 // ============================================================
164 // CNX file retrieval info
165 // ============================================================
166 // Retrieval info: PRIVATE: ACTIVECLK_CHECK STRING "0"
167 // Retrieval info: PRIVATE: BANDWIDTH STRING "1.000"
168 // Retrieval info: PRIVATE: BANDWIDTH_FEATURE_ENABLED STRING "1"
169 // Retrieval info: PRIVATE: BANDWIDTH_FREQ_UNIT STRING "MHz"
170 // Retrieval info: PRIVATE: BANDWIDTH_PRESET STRING "Low"
171 // Retrieval info: PRIVATE: BANDWIDTH_USE_AUTO STRING "1"
172 // Retrieval info: PRIVATE: BANDWIDTH_USE_PRESET STRING "0"
173 // Retrieval info: PRIVATE: CLKBAD_SWITCHOVER_CHECK STRING "0"
174 // Retrieval info: PRIVATE: CLKLOSS_CHECK STRING "0"
175 // Retrieval info: PRIVATE: CLKSWITCH_CHECK STRING "0"
176 // Retrieval info: PRIVATE: CNX_NO_COMPENSATE_RADIO STRING "0"
177 // Retrieval info: PRIVATE: CREATE_CLKBAD_CHECK STRING "0"
178 // Retrieval info: PRIVATE: CREATE_INCLK1_CHECK STRING "0"
179 // Retrieval info: PRIVATE: CUR_DEDICATED_CLK STRING "c0"
180 // Retrieval info: PRIVATE: CUR_FBIN_CLK STRING "c0"
181 // Retrieval info: PRIVATE: DEVICE_SPEED_GRADE STRING "Any"
182 // Retrieval info: PRIVATE: DIV_FACTOR0 NUMERIC "5"
183 // Retrieval info: PRIVATE: DUTY_CYCLE0 STRING "50.00000000"
184 // Retrieval info: PRIVATE: EFF_OUTPUT_FREQ_VALUE0 STRING "10.000000"
185 // Retrieval info: PRIVATE: EXPLICIT_SWITCHOVER_COUNTER STRING "0"
186 // Retrieval info: PRIVATE: EXT_FEEDBACK_RADIO STRING "0"
187 // Retrieval info: PRIVATE: GLOCKED_COUNTER_EDIT_CHANGED STRING "1"
188 // Retrieval info: PRIVATE: GLOCKED_FEATURE_ENABLED STRING "0"
189 // Retrieval info: PRIVATE: GLOCKED_MODE_CHECK STRING "0"
190 // Retrieval info: PRIVATE: GLOCK_COUNTER_EDIT NUMERIC "1048575"
191 // Retrieval info: PRIVATE: HAS_MANUAL_SWITCHOVER STRING "1"
192 // Retrieval info: PRIVATE: INCLK0_FREQ_EDIT STRING "50.000"
193 // Retrieval info: PRIVATE: INCLK0_FREQ_UNIT_COMBO STRING "MHz"
194 // Retrieval info: PRIVATE: INCLK1_FREQ_EDIT STRING "100.000"
195 // Retrieval info: PRIVATE: INCLK1_FREQ_EDIT_CHANGED STRING "1"
196 // Retrieval info: PRIVATE: INCLK1_FREQ_UNIT_CHANGED STRING "1"
197 // Retrieval info: PRIVATE: INCLK1_FREQ_UNIT_COMBO STRING "MHz"
198 // Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone IV E"
199 // Retrieval info: PRIVATE: INT_FEEDBACK__MODE_RADIO STRING "1"
200 // Retrieval info: PRIVATE: LOCKED_OUTPUT_CHECK STRING "0"
201 // Retrieval info: PRIVATE: LONG_SCAN_RADIO STRING "1"
202 // Retrieval info: PRIVATE: LVDS_MODE_DATA_RATE STRING "Not Available"
203 // Retrieval info: PRIVATE: LVDS_MODE_DATA_RATE_DIRTY NUMERIC "0"
```

```
// Retrieval info: PRIVATE: LVDS_PHASE_SHIFT_UNIT0 STRING "deg"
// Retrieval info: PRIVATE: MIG_DEVICE_SPEED_GRADE STRING "Any"
// Retrieval info: PRIVATE: MIRROR_CLK0 STRING "0"
// Retrieval info: PRIVATE: MULT_FACTOR0 NUMERIC "1"
// Retrieval info: PRIVATE: NORMAL_MODE_RADIO STRING "1"
// Retrieval info: PRIVATE: OUTPUT_FREQ0 STRING "100.00000000"
// Retrieval info: PRIVATE: OUTPUT_FREQ_MODE0 STRING "0"
// Retrieval info: PRIVATE: OUTPUT_FREQ_UNIT0 STRING "MHz"
// Retrieval info: PRIVATE: PHASE_RECONFIG_FEATURE_ENABLED STRING "1"
// Retrieval info: PRIVATE: PHASE_RECONFIG_INPUTS_CHECK STRING "0"
// Retrieval info: PRIVATE: PHASE_SHIFT0 STRING "0.00000000"
// Retrieval info: PRIVATE: PHASE_SHIFT_STEP_ENABLED_CHECK STRING "0"
// Retrieval info: PRIVATE: PHASE_SHIFT_UNIT0 STRING "deg"
// Retrieval info: PRIVATE: PLL_ADVANCED_PARAM_CHECK STRING "0"
// Retrieval info: PRIVATE: PLL_ARESET_CHECK STRING "1"
// Retrieval info: PRIVATE: PLL_AUTOPLL_CHECK NUMERIC "1"
// Retrieval info: PRIVATE: PLL_ENHPLL_CHECK NUMERIC "0"
// Retrieval info: PRIVATE: PLL_FASTPLL_CHECK NUMERIC "0"
// Retrieval info: PRIVATE: PLL_FBMIMIC_CHECK STRING "0"
// Retrieval info: PRIVATE: PLL_LVDS_PLL_CHECK NUMERIC "0"
// Retrieval info: PRIVATE: PLL_PFDENA_CHECK STRING "0"
// Retrieval info: PRIVATE: PLL_TARGET_HARCOPY_CHECK NUMERIC "0"
// Retrieval info: PRIVATE: PRIMARY_CLK_COMBO STRING "inclk0"
// Retrieval info: PRIVATE: RECONFIG_FILE STRING "pll.mif"
// Retrieval info: PRIVATE: SACN_INPUTS_CHECK STRING "0"
// Retrieval info: PRIVATE: SCAN_FEATURE_ENABLED STRING "1"
// Retrieval info: PRIVATE: SELF_RESET_LOCK_LOSS STRING "0"
// Retrieval info: PRIVATE: SHORT_SCAN_RADIO STRING "0"
// Retrieval info: PRIVATE: SPREAD_FEATURE_ENABLED STRING "0"
// Retrieval info: PRIVATE: SPREAD_FREQ STRING "50.000"
// Retrieval info: PRIVATE: SPREAD_FREQ_UNIT STRING "KHz"
// Retrieval info: PRIVATE: SPREAD_PERCENT STRING "0.500"
// Retrieval info: PRIVATE: SPREAD_USE STRING "0"
// Retrieval info: PRIVATE: SRC_SYNCH_COMP_RADIO STRING "0"
// Retrieval info: PRIVATE: STICKY_CLK0 STRING "1"
// Retrieval info: PRIVATE: SWITCHOVER_COUNT_EDIT NUMERIC "1"
// Retrieval info: PRIVATE: SWITCHOVER_FEATURE_ENABLED STRING "1"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
// Retrieval info: PRIVATE: USE_CLK0 STRING "1"
// Retrieval info: PRIVATE: USE_CLKENA0 STRING "0"
// Retrieval info: PRIVATE: USE_MIL_SPEED_GRADE NUMERIC "0"
// Retrieval info: PRIVATE: ZERO_DELAY_RADIO STRING "0"
// Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
// Retrieval info: CONSTANT: BANDWIDTH_TYPE STRING "AUTO"
// Retrieval info: CONSTANT: CLK0_DIVIDE_BY NUMERIC "5"
// Retrieval info: CONSTANT: CLK0_DUTY_CYCLE NUMERIC "50"
// Retrieval info: CONSTANT: CLK0_MULTIPLY_BY NUMERIC "1"
// Retrieval info: CONSTANT: CLK0_PHASE_SHIFT STRING "0"
// Retrieval info: CONSTANT: COMPENSATE_CLOCK STRING "CLK0"
// Retrieval info: CONSTANT: INCLK0_INPUT_FREQUENCY NUMERIC "20000"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone IV E"
```

```
// Retrieval info: CONSTANT: LPM_TYPE STRING "altpll"
// Retrieval info: CONSTANT: OPERATION_MODE STRING "NORMAL"
// Retrieval info: CONSTANT: PLL_TYPE STRING "AUTO"
// Retrieval info: CONSTANT: PORT_ACTIVECLOCK STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_ARESET STRING "PORT_USED"
// Retrieval info: CONSTANT: PORT_CLKBAD0 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_CLKBAD1 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_CLKLOSS STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_CLKSWITCH STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_CONFIGUPDATE STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_FBIN STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_INCLK0 STRING "PORT_USED"
// Retrieval info: CONSTANT: PORT_INCLK1 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_LOCKED STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_PFDENA STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_PHASECOUNTERSELECT STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_PHASEDONE STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_PHASESTEP STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_PHASEUPDOWN STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_PLLENA STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANACLR STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANCLK STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANCLKENA STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANDATA STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANDATAOUT STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANDONE STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANREAD STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANWRITE STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clk0 STRING "PORT_USED"
// Retrieval info: CONSTANT: PORT_clk1 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clk2 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clk3 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clk4 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clk5 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clkena0 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clkena1 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clkena2 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clkena3 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clkena4 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clkena5 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_extclk0 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_extclk1 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_extclk2 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_extclk3 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: WIDTH_CLOCK NUMERIC "5"
// Retrieval info: USED_PORT: @clk 0 0 5 0 OUTPUT_CLK_EXT VCC "@clk[4..0]"
// Retrieval info: USED_PORT: areset 0 0 0 0 INPUT GND "areset"
// Retrieval info: USED_PORT: c0 0 0 0 0 OUTPUT_CLK_EXT VCC "c0"
// Retrieval info: USED_PORT: inclk0 0 0 0 0 INPUT_CLK_EXT GND "inclk0"
// Retrieval info: CONNECT: @areset 0 0 0 0 areset 0 0 0 0
// Retrieval info: CONNECT: @inclk 0 0 1 1 GND 0 0 0 0
```

```
306 // Retrieval info: CONNECT: @inclk 0 0 1 0 inclk0 0 0 0 0
307 // Retrieval info: CONNECT: c0 0 0 0 0 @clk 0 0 1 0
308 // Retrieval info: GEN_FILE: TYPE_NORMAL pll.v TRUE
309 // Retrieval info: GEN_FILE: TYPE_NORMAL pll.ppf TRUE
310 // Retrieval info: GEN_FILE: TYPE_NORMAL pll.inc FALSE
311 // Retrieval info: GEN_FILE: TYPE_NORMAL pll.cmp FALSE
312 // Retrieval info: GEN_FILE: TYPE_NORMAL pll.bsf FALSE
313 // Retrieval info: GEN_FILE: TYPE_NORMAL pll_inst.v FALSE
314 // Retrieval info: GEN_FILE: TYPE_NORMAL pll_bb.v TRUE
315 // Retrieval info: LIB_FILE: altera_mf
316 // Retrieval info: CBX_MODULE_PREFIX: ON
```

1.8.9 25 MHz PLL

```
//File name : pll_25mhz.v
//This module is used to convert a 50MHz clock pulse to a 25MHz clock pulse.
//Instantiated in the clock controller module.

// megafunction wizard: %ALTPLL%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: altpll

// ============================================================
// File Name: pll_25mhz.v
// Megafunction Name(s):
//       altpll
//
// Simulation Library Files(s):
//       altera_mf
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 15.0.0 Build 145 04/22/2015 SJ Full Version
// ************************************************************


//Copyright (C) 1991-2015 Altera Corporation. All rights reserved.
//Your use of Altera Corporation's design tools, logic functions
//and other software and tools, and its AMPP partner logic
//functions, and any output files from any of the foregoing
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Altera Program License
//Subscription Agreement, the Altera Quartus II License Agreement,
//the Altera MegaCore Function License Agreement, or other
//applicable license agreement, including, without limitation,
//that your use is for the sole purpose of programming logic
//devices manufactured by Altera and sold by Altera or its
//authorized distributors.  Please refer to the applicable
//agreement for further details.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module pll_25mhz (
  areset,
  inclk0,
  c0,
  locked);

  input    areset;
```

```
51    input    inclk0;
52    output    c0;
53    output    locked;
54  `ifndef ALTERA_RESERVED_QIS
55  // synopsys translate_off
56  `endif
57    tri0    areset;
58  `ifndef ALTERA_RESERVED_QIS
59  // synopsys translate_on
60  `endif
61
62    wire [0:0] sub_wire2 = 1'h0;
63    wire [4:0] sub_wire3;
64    wire  sub_wire5;
65    wire  sub_wire0 = inclk0;
66    wire [1:0] sub_wire1 = {sub_wire2, sub_wire0};
67    wire [0:0] sub_wire4 = sub_wire3[0:0];
68    wire  c0 = sub_wire4;
69    wire  locked = sub_wire5;
70
71    altpll  altpll_component (
72          .areset (areset),
73          .inclk (sub_wire1),
74          .clk (sub_wire3),
75          .locked (sub_wire5),
76          .activeclock (),
77          .clkbad (),
78          .clkena ({6{1'b1}}),
79          .clkloss (),
80          .clkswitch (1'b0),
81          .configupdate (1'b0),
82          .enable0 (),
83          .enable1 (),
84          .extclk (),
85          .extclkena ({4{1'b1}}),
86          .fbin (1'b1),
87          .fbmimicbidir (),
88          .fbout (),
89          .fref (),
90          .icdrclk (),
91          .pfdena (1'b1),
92          .phasecounterselect ({4{1'b1}}),
93          .phasedone (),
94          .phasestep (1'b1),
95          .phaseupdown (1'b1),
96          .pllena (1'b1),
97          .scanaclr (1'b0),
98          .scanclk (1'b0),
99          .scanclkena (1'b1),
100         .scandata (1'b0),
101         .scandataout (),
```

```
102         .scandone (),
103         .scanread (1'b0),
104         .scanwrite (1'b0),
105         .sclkout0 (),
106         .sclkout1 (),
107         .vcooverrange (),
108         .vcounderrange ());
109     defparam
110       altpll_component.bandwidth_type = "AUTO",
111       altpll_component.clk0_divide_by = 2,
112       altpll_component.clk0_duty_cycle = 50,
113       altpll_component.clk0_multiply_by = 1,
114       altpll_component.clk0_phase_shift = "0",
115       altpll_component.compensate_clock = "CLK0",
116       altpll_component.inclk0_input_frequency = 20000,
117       altpll_component.intended_device_family = "Cyclone IV E",
118       altpll_component.lpm_hint = "CBX_MODULE_PREFIX=pll_25mhz",
119       altpll_component.lpm_type = "altpll",
120       altpll_component.operation_mode = "NORMAL",
121       altpll_component.pll_type = "AUTO",
122       altpll_component.port_activeclock = "PORT_UNUSED",
123       altpll_component.port_areset = "PORT_USED",
124       altpll_component.port_clkbad0 = "PORT_UNUSED",
125       altpll_component.port_clkbad1 = "PORT_UNUSED",
126       altpll_component.port_clkloss = "PORT_UNUSED",
127       altpll_component.port_clkswitch = "PORT_UNUSED",
128       altpll_component.port_configupdate = "PORT_UNUSED",
129       altpll_component.port_fbin = "PORT_UNUSED",
130       altpll_component.port_inclk0 = "PORT_USED",
131       altpll_component.port_inclk1 = "PORT_UNUSED",
132       altpll_component.port_locked = "PORT_USED",
133       altpll_component.port_pfdena = "PORT_UNUSED",
134       altpll_component.port_phasecounterselect = "PORT_UNUSED",
135       altpll_component.port_phasedone = "PORT_UNUSED",
136       altpll_component.port_phasestep = "PORT_UNUSED",
137       altpll_component.port_phaseupdown = "PORT_UNUSED",
138       altpll_component.port_pllena = "PORT_UNUSED",
139       altpll_component.port_scanaclr = "PORT_UNUSED",
140       altpll_component.port_scanclk = "PORT_UNUSED",
141       altpll_component.port_scanclkena = "PORT_UNUSED",
142       altpll_component.port_scandata = "PORT_UNUSED",
143       altpll_component.port_scandataout = "PORT_UNUSED",
144       altpll_component.port_scandone = "PORT_UNUSED",
145       altpll_component.port_scanread = "PORT_UNUSED",
146       altpll_component.port_scanwrite = "PORT_UNUSED",
147       altpll_component.port_clk0 = "PORT_USED",
148       altpll_component.port_clk1 = "PORT_UNUSED",
149       altpll_component.port_clk2 = "PORT_UNUSED",
150       altpll_component.port_clk3 = "PORT_UNUSED",
151       altpll_component.port_clk4 = "PORT_UNUSED",
152       altpll_component.port_clk5 = "PORT_UNUSED",
```

```
153     altpll_component.port_clkena0 = "PORT_UNUSED",
154     altpll_component.port_clkena1 = "PORT_UNUSED",
155     altpll_component.port_clkena2 = "PORT_UNUSED",
156     altpll_component.port_clkena3 = "PORT_UNUSED",
157     altpll_component.port_clkena4 = "PORT_UNUSED",
158     altpll_component.port_clkena5 = "PORT_UNUSED",
159     altpll_component.port_extclk0 = "PORT_UNUSED",
160     altpll_component.port_extclk1 = "PORT_UNUSED",
161     altpll_component.port_extclk2 = "PORT_UNUSED",
162     altpll_component.port_extclk3 = "PORT_UNUSED",
163     altpll_component.self_reset_on_loss_lock = "OFF",
164     altpll_component.width_clock = 5;
165
166
167 endmodule
168
169 // ============================================================
170 // CNX file retrieval info
171 // ============================================================
172 // Retrieval info: PRIVATE: ACTIVECLK_CHECK STRING "0"
173 // Retrieval info: PRIVATE: BANDWIDTH STRING "1.000"
174 // Retrieval info: PRIVATE: BANDWIDTH_FEATURE_ENABLED STRING "1"
175 // Retrieval info: PRIVATE: BANDWIDTH_FREQ_UNIT STRING "MHz"
176 // Retrieval info: PRIVATE: BANDWIDTH_PRESET STRING "Low"
177 // Retrieval info: PRIVATE: BANDWIDTH_USE_AUTO STRING "1"
178 // Retrieval info: PRIVATE: BANDWIDTH_USE_PRESET STRING "0"
179 // Retrieval info: PRIVATE: CLKBAD_SWITCHOVER_CHECK STRING "0"
180 // Retrieval info: PRIVATE: CLKLOSS_CHECK STRING "0"
181 // Retrieval info: PRIVATE: CLKSWITCH_CHECK STRING "0"
182 // Retrieval info: PRIVATE: CNX_NO_COMPENSATE_RADIO STRING "0"
183 // Retrieval info: PRIVATE: CREATE_CLKBAD_CHECK STRING "0"
184 // Retrieval info: PRIVATE: CREATE_INCLK1_CHECK STRING "0"
185 // Retrieval info: PRIVATE: CUR_DEDICATED_CLK STRING "c0"
186 // Retrieval info: PRIVATE: CUR_FBIN_CLK STRING "c0"
187 // Retrieval info: PRIVATE: DEVICE_SPEED_GRADE STRING "Any"
188 // Retrieval info: PRIVATE: DIV_FACTOR0 NUMERIC "1"
189 // Retrieval info: PRIVATE: DUTY_CYCLE0 STRING "50.00000000"
190 // Retrieval info: PRIVATE: EFF_OUTPUT_FREQ_VALUE0 STRING "25.000000"
191 // Retrieval info: PRIVATE: EXPLICIT_SWITCHOVER_COUNTER STRING "0"
192 // Retrieval info: PRIVATE: EXT_FEEDBACK_RADIO STRING "0"
193 // Retrieval info: PRIVATE: GLOCKED_COUNTER_EDIT_CHANGED STRING "1"
194 // Retrieval info: PRIVATE: GLOCKED_FEATURE_ENABLED STRING "0"
195 // Retrieval info: PRIVATE: GLOCKED_MODE_CHECK STRING "0"
196 // Retrieval info: PRIVATE: GLOCK_COUNTER_EDIT NUMERIC "1048575"
197 // Retrieval info: PRIVATE: HAS_MANUAL_SWITCHOVER STRING "1"
198 // Retrieval info: PRIVATE: INCLK0_FREQ_EDIT STRING "50.000"
199 // Retrieval info: PRIVATE: INCLK0_FREQ_UNIT_COMBO STRING "MHz"
200 // Retrieval info: PRIVATE: INCLK1_FREQ_EDIT STRING "100.000"
201 // Retrieval info: PRIVATE: INCLK1_FREQ_EDIT_CHANGED STRING "1"
202 // Retrieval info: PRIVATE: INCLK1_FREQ_UNIT_CHANGED STRING "1"
203 // Retrieval info: PRIVATE: INCLK1_FREQ_UNIT_COMBO STRING "MHz"
```

```
204 // Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone IV E"
205 // Retrieval info: PRIVATE: INT_FEEDBACK__MODE_RADIO STRING "1"
206 // Retrieval info: PRIVATE: LOCKED_OUTPUT_CHECK STRING "1"
207 // Retrieval info: PRIVATE: LONG_SCAN_RADIO STRING "1"
208 // Retrieval info: PRIVATE: LVDS_MODE_DATA_RATE STRING "Not Available"
209 // Retrieval info: PRIVATE: LVDS_MODE_DATA_RATE_DIRTY NUMERIC "0"
210 // Retrieval info: PRIVATE: LVDS_PHASE_SHIFT_UNIT0 STRING "deg"
211 // Retrieval info: PRIVATE: MIG_DEVICE_SPEED_GRADE STRING "Any"
212 // Retrieval info: PRIVATE: MIRROR_CLK0 STRING "0"
213 // Retrieval info: PRIVATE: MULT_FACTOR0 NUMERIC "1"
214 // Retrieval info: PRIVATE: NORMAL_MODE_RADIO STRING "1"
215 // Retrieval info: PRIVATE: OUTPUT_FREQ0 STRING "25.00000000"
216 // Retrieval info: PRIVATE: OUTPUT_FREQ_MODE0 STRING "1"
217 // Retrieval info: PRIVATE: OUTPUT_FREQ_UNIT0 STRING "MHz"
218 // Retrieval info: PRIVATE: PHASE_RECONFIG_FEATURE_ENABLED STRING "1"
219 // Retrieval info: PRIVATE: PHASE_RECONFIG_INPUTS_CHECK STRING "0"
220 // Retrieval info: PRIVATE: PHASE_SHIFT0 STRING "0.00000000"
221 // Retrieval info: PRIVATE: PHASE_SHIFT_STEP_ENABLED_CHECK STRING "0"
222 // Retrieval info: PRIVATE: PHASE_SHIFT_UNIT0 STRING "deg"
223 // Retrieval info: PRIVATE: PLL_ADVANCED_PARAM_CHECK STRING "0"
224 // Retrieval info: PRIVATE: PLL_ARESET_CHECK STRING "1"
225 // Retrieval info: PRIVATE: PLL_AUTOPLL_CHECK NUMERIC "1"
226 // Retrieval info: PRIVATE: PLL_ENHPLL_CHECK NUMERIC "0"
227 // Retrieval info: PRIVATE: PLL_FASTPLL_CHECK NUMERIC "0"
228 // Retrieval info: PRIVATE: PLL_FBMIMIC_CHECK STRING "0"
229 // Retrieval info: PRIVATE: PLL_LVDS_PLL_CHECK NUMERIC "0"
230 // Retrieval info: PRIVATE: PLL_PFDENA_CHECK STRING "0"
231 // Retrieval info: PRIVATE: PLL_TARGET_HARCOPY_CHECK NUMERIC "0"
232 // Retrieval info: PRIVATE: PRIMARY_CLK_COMBO STRING "inclk0"
233 // Retrieval info: PRIVATE: RECONFIG_FILE STRING "pll_25mhz.mif"
234 // Retrieval info: PRIVATE: SACN_INPUTS_CHECK STRING "0"
235 // Retrieval info: PRIVATE: SCAN_FEATURE_ENABLED STRING "1"
236 // Retrieval info: PRIVATE: SELF_RESET_LOCK_LOSS STRING "0"
237 // Retrieval info: PRIVATE: SHORT_SCAN_RADIO STRING "0"
238 // Retrieval info: PRIVATE: SPREAD_FEATURE_ENABLED STRING "0"
239 // Retrieval info: PRIVATE: SPREAD_FREQ STRING "50.000"
240 // Retrieval info: PRIVATE: SPREAD_FREQ_UNIT STRING "KHz"
241 // Retrieval info: PRIVATE: SPREAD_PERCENT STRING "0.500"
242 // Retrieval info: PRIVATE: SPREAD_USE STRING "0"
243 // Retrieval info: PRIVATE: SRC_SYNCH_COMP_RADIO STRING "0"
244 // Retrieval info: PRIVATE: STICKY_CLK0 STRING "1"
245 // Retrieval info: PRIVATE: SWITCHOVER_COUNT_EDIT NUMERIC "1"
246 // Retrieval info: PRIVATE: SWITCHOVER_FEATURE_ENABLED STRING "1"
247 // Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
248 // Retrieval info: PRIVATE: USE_CLK0 STRING "1"
249 // Retrieval info: PRIVATE: USE_CLKENA0 STRING "0"
250 // Retrieval info: PRIVATE: USE_MIL_SPEED_GRADE NUMERIC "0"
251 // Retrieval info: PRIVATE: ZERO_DELAY_RADIO STRING "0"
252 // Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
253 // Retrieval info: CONSTANT: BANDWIDTH_TYPE STRING "AUTO"
254 // Retrieval info: CONSTANT: CLK0_DIVIDE_BY NUMERIC "2"
```

```
// Retrieval info: CONSTANT: CLK0_DUTY_CYCLE NUMERIC "50"
// Retrieval info: CONSTANT: CLK0_MULTIPLY_BY NUMERIC "1"
// Retrieval info: CONSTANT: CLK0_PHASE_SHIFT STRING "0"
// Retrieval info: CONSTANT: COMPENSATE_CLOCK STRING "CLK0"
// Retrieval info: CONSTANT: INCLK0_INPUT_FREQUENCY NUMERIC "20000"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone IV E"
// Retrieval info: CONSTANT: LPM_TYPE STRING "altpll"
// Retrieval info: CONSTANT: OPERATION_MODE STRING "NORMAL"
// Retrieval info: CONSTANT: PLL_TYPE STRING "AUTO"
// Retrieval info: CONSTANT: PORT_ACTIVECLOCK STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_ARESET STRING "PORT_USED"
// Retrieval info: CONSTANT: PORT_CLKBAD0 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_CLKBAD1 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_CLKLOSS STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_CLKSWITCH STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_CONFIGUPDATE STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_FBIN STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_INCLK0 STRING "PORT_USED"
// Retrieval info: CONSTANT: PORT_INCLK1 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_LOCKED STRING "PORT_USED"
// Retrieval info: CONSTANT: PORT_PFDENA STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_PHASECOUNTERSELECT STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_PHASEDONE STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_PHASESTEP STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_PHASEUPDOWN STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_PLLENA STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANACLR STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANCLK STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANCLKENA STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANDATA STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANDATAOUT STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANDONE STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANREAD STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_SCANWRITE STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clk0 STRING "PORT_USED"
// Retrieval info: CONSTANT: PORT_clk1 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clk2 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clk3 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clk4 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clk5 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clkena0 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clkena1 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clkena2 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clkena3 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clkena4 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_clkena5 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_extclk0 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_extclk1 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_extclk2 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: PORT_extclk3 STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: SELF_RESET_ON_LOSS_LOCK STRING "OFF"
```

```
306 // Retrieval info: CONSTANT: WIDTH_CLOCK NUMERIC "5"
307 // Retrieval info: USED_PORT: @clk 0 0 5 0 OUTPUT_CLK_EXT VCC "@clk[4..0]"
308 // Retrieval info: USED_PORT: areset 0 0 0 0 INPUT GND "areset"
309 // Retrieval info: USED_PORT: c0 0 0 0 0 OUTPUT_CLK_EXT VCC "c0"
310 // Retrieval info: USED_PORT: inclk0 0 0 0 0 INPUT_CLK_EXT GND "inclk0"
311 // Retrieval info: USED_PORT: locked 0 0 0 0 OUTPUT GND "locked"
312 // Retrieval info: CONNECT: @areset 0 0 0 0 areset 0 0 0 0
313 // Retrieval info: CONNECT: @inclk 0 0 1 1 GND 0 0 0 0
314 // Retrieval info: CONNECT: @inclk 0 0 1 0 inclk0 0 0 0 0
315 // Retrieval info: CONNECT: c0 0 0 0 0 @clk 0 0 1 0
316 // Retrieval info: CONNECT: locked 0 0 0 0 @locked 0 0 0 0
317 // Retrieval info: GEN_FILE: TYPE_NORMAL pll_25mhz.v TRUE
318 // Retrieval info: GEN_FILE: TYPE_NORMAL pll_25mhz.ppf TRUE
319 // Retrieval info: GEN_FILE: TYPE_NORMAL pll_25mhz.inc FALSE
320 // Retrieval info: GEN_FILE: TYPE_NORMAL pll_25mhz.cmp FALSE
321 // Retrieval info: GEN_FILE: TYPE_NORMAL pll_25mhz.bsf FALSE
322 // Retrieval info: GEN_FILE: TYPE_NORMAL pll_25mhz_inst.v FALSE
323 // Retrieval info: GEN_FILE: TYPE_NORMAL pll_25mhz_bb.v TRUE
324 // Retrieval info: LIB_FILE: altera_mf
325 // Retrieval info: CBX_MODULE_PREFIX: ON
```

1.8.10 Splitter

```
//File name : splitter_2.v

module splitter_2 #(parameter bit_width=1) (in,d_out,i_out,enable,selector);

input [bit_width-1:0] in;
input enable;
input selector;    // [D or I]
output reg [bit_width-1:0] d_out,i_out;

parameter  DIRECT_TO_D=0;
parameter  DIRECT_TO_I=1;

always @(in,selector,enable)
  if (enable==1)
    begin
      case(selector)
        DIRECT_TO_D:
          begin
            d_out<=in;
            i_out<=1;
          end
        DIRECT_TO_I:
          begin
            i_out<=in;
            d_out<=1;
          end
        default:
          begin
            i_out<=1;
            d_out<=1;
          end
      endcase
    end
  else
    begin
      d_out<=1;
      i_out<=1;
    end
endmodule
```

1.8.11 Two Way Mux

```
1  //File name : two_way_mux.v
2  //A two input multiplexer
3
4  // megafunction wizard: %LPM_MUX%
5  // GENERATION: STANDARD
6  // VERSION: WM1.0
7  // MODULE: LPM_MUX
8
9  // ============================================================
10 // File Name: two_way_mux.v
11 // Megafunction Name(s):
12 //       LPM_MUX
13 //
14 // Simulation Library Files(s):
15 //       lpm
16 // ============================================================
17 // ************************************************************
18 // THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
19 //
20 // 15.0.0 Build 145 04/22/2015 SJ Full Version
21 // ************************************************************
22
23
24 //Copyright (C) 1991-2015 Altera Corporation. All rights reserved.
25 //Your use of Altera Corporation's design tools, logic functions
26 //and other software and tools, and its AMPP partner logic
27 //functions, and any output files from any of the foregoing
28 //(including device programming or simulation files), and any
29 //associated documentation or information are expressly subject
30 //to the terms and conditions of the Altera Program License
31 //Subscription Agreement, the Altera Quartus II License Agreement,
32 //the Altera MegaCore Function License Agreement, or other
33 //applicable license agreement, including, without limitation,
34 //that your use is for the sole purpose of programming logic
35 //devices manufactured by Altera and sold by Altera or its
36 //authorized distributors.  Please refer to the applicable
37 //agreement for further details.
38
39
40 // synopsys translate_off
41 `timescale 1 ps / 1 ps
42 // synopsys translate_on
43 module two_way_mux (
44   data0,
45   data1,
46   sel,
47   result);
48
49   input    data0;
50   input    data1;
```

```
51    input    sel;
52    output    result;
53
54    wire [0:0] sub_wire5;
55    wire  sub_wire2 = data1;
56    wire  sub_wire0 = data0;
57    wire [1:0] sub_wire1 = {sub_wire2, sub_wire0};
58    wire  sub_wire3 = sel;
59    wire  sub_wire4 = sub_wire3;
60    wire [0:0] sub_wire6 = sub_wire5[0:0];
61    wire  result = sub_wire6;
62
63    lpm_mux  LPM_MUX_component (
64          .data (sub_wire1),
65          .sel (sub_wire4),
66          .result (sub_wire5)
67          // synopsys translate_off
68          ,
69          .aclr (),
70          .clken (),
71          .clock ()
72          // synopsys translate_on
73          );
74    defparam
75      LPM_MUX_component.lpm_size = 2,
76      LPM_MUX_component.lpm_type = "LPM_MUX",
77      LPM_MUX_component.lpm_width = 1,
78      LPM_MUX_component.lpm_widths = 1;
79
80
81  endmodule
82
83  // ============================================================
84  // CNX file retrieval info
85  // ============================================================
86  // Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone IV E"
87  // Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
88  // Retrieval info: PRIVATE: new_diagram STRING "1"
89  // Retrieval info: LIBRARY: lpm lpm.lpm_components.all
90  // Retrieval info: CONSTANT: LPM_SIZE NUMERIC "2"
91  // Retrieval info: CONSTANT: LPM_TYPE STRING "LPM_MUX"
92  // Retrieval info: CONSTANT: LPM_WIDTH NUMERIC "1"
93  // Retrieval info: CONSTANT: LPM_WIDTHS NUMERIC "1"
94  // Retrieval info: USED_PORT: data0 0 0 0 0 INPUT NODEFVAL "data0"
95  // Retrieval info: USED_PORT: data1 0 0 0 0 INPUT NODEFVAL "data1"
96  // Retrieval info: USED_PORT: result 0 0 0 0 OUTPUT NODEFVAL "result"
97  // Retrieval info: USED_PORT: sel 0 0 0 0 INPUT NODEFVAL "sel"
98  // Retrieval info: CONNECT: @data 0 0 1 0 data0 0 0 0 0
99  // Retrieval info: CONNECT: @data 0 0 1 1 data1 0 0 0 0
100 // Retrieval info: CONNECT: @sel 0 0 1 0 sel 0 0 0 0
101 // Retrieval info: CONNECT: result 0 0 0 0 @result 0 0 1 0
```

```
102 // Retrieval info: GEN_FILE: TYPE_NORMAL two_way_mux.v TRUE
103 // Retrieval info: GEN_FILE: TYPE_NORMAL two_way_mux.inc FALSE
104 // Retrieval info: GEN_FILE: TYPE_NORMAL two_way_mux.cmp FALSE
105 // Retrieval info: GEN_FILE: TYPE_NORMAL two_way_mux.bsf FALSE
106 // Retrieval info: GEN_FILE: TYPE_NORMAL two_way_mux_inst.v FALSE
107 // Retrieval info: GEN_FILE: TYPE_NORMAL two_way_mux_bb.v TRUE
108 // Retrieval info: LIB_FILE: lpm
```

1.9 Definition Files

1.9.1 Keyword Definitions

```
1  //File name : define.v
2  //This contains the numbers assigned for the parameters used in
3  //the modules mentioned.
4
5  // Control Signals
6  // use
7
8
9  // [ACI_decoder] To write from A-bus to registers' cin
10
11 `define aci_none      5'b00000
12 `define aci_AC        5'b00001
13 `define aci_MDDR      5'b00010
14 `define aci_K0        5'b00100
15 `define aci_K1        5'b01000
16 `define aci_G        5'b10000
17 `define aci_all      5'b11111
18
19 // [AWM_mux] To write from registers to A-bus (A mux selection bits)
20
21 `define awm_AC        3'd0
22 `define awm_MDDR      3'd1
23 `define awm_K0        3'd2
24 `define awm_K1        3'd3
25 `define awm_G0        3'd4
26 `define awm_G1        3'd5
27 `define awm_G2        3'd6
28 `define awm_MIDR      3'd7
29
30 // [INC_decoder] inc signals for registers
31
32 `define inc_none      3'd0
33 `define inc_ADR        3'd1
34 `define inc_ART        3'd2
35 `define inc_ARG        3'd3
36 `define inc_AWT        3'd4
37 `define inc_AWG        3'd5
38 `define inc_AC        3'd6
39 `define inc_MDAR      3'd7
40
41 // [DEC_decoder] dec signals for registers
42
43 `define dec_none      3'd0
44 `define dec_ADR        3'd1
45 `define dec_ART        3'd2
46 `define dec_ARG        3'd3
47 `define dec_AWT        3'd4
48 `define dec_AWG        3'd5
```

```
`define dec_AC        3'd6
`define dec_MDAR      3'd7

// [ALU] ALU selection bits

`define alu_none      3'd0
`define alu_add        3'd1
`define alu_sub        3'd2
`define alu_div        3'd3
`define alu_mul        3'd4


// [ADR_maker] Selection bits for ADR_maker

`define adr_none      4'd0
`define adr_matrix_r    4'd1
`define adr_matrix_w    4'd2
`define adr_last8      4'd3
`define adr_mid8      4'd4
`define adr_first2    4'd5
`define adr_to_mdar    4'd6
`define adr_to_ar     4'd7
`define adr_to_aw     4'd8
`define adr_to_ar_ref  4'd9

// [JMP_encoder] Jump signals


`define jmp_none      4'd0
`define jmp_jump      4'd1
`define jmp_jmpz      4'd2
`define jmp_jpnz      4'd3
`define jmp_jzt      4'd4
`define jmp_jnrg      4'd5
`define jmp_jnrt      4'd6
`define jmp_jnk0      4'd7
`define jmp_jnk1      4'd8

// [OPR_decoder] To give operations controls

`define opr_none      3'd0
`define opr_aci_awm        3'd1
`define opr_awm          3'd2
`define opr_inc        3'd3
`define opr_pc        3'd4
`define opr_rst        3'd5
`define opr_dec             3'd6

// [RST_decoder] To give reset controls

`define rst_none      3'd0
```

```
100 `define rst_ART         3'd1
101 `define rst_ARG         3'd2
102 `define rst_AWT         3'd3
103 `define rst_AWG         3'd4
104 `define rst_MDAR       3'd5
105 `define rst_all         3'd6
106
107 //memory signals
108
109 `define mem_none           3'b000
110 `define mem_dm_write       3'b100
111 `define mem_mddr_m_ci      3'b010
112 `define mem_midr_m_ci      3'b001
113
114 //parameter router
115
116 `define prm_none           2'd0
117 `define prm_jmp            2'd1
118 `define prm_adr            2'd2
119 `define prm_add_sub        2'd3
```

1.9.2 Opcode Definitions

```
//File name : opcode_define.v
//This module contains the values assigned to the parameters
//in the state machine's STATEs.

// Opcodes and their binary values
`define END  0
`define NOOP  16
`define FETCH  1
`define FETCH_2  2
`define FETCH_3  3
`define LODK  48
`define LODK_2  49
`define LADD  64
`define LADD_2  65
`define LADD_3  66
`define LADD_4  67
`define LADD_5  68
`define LADD_6  69
`define LADD_7  70
`define LOAD  80
`define LOAD_2  81
`define LOAD_3  82
`define STAC  96
`define COPY  112
`define COPY_2  113
`define RSET  128
`define RSET_2  129
`define JUMP  144
`define JUMP_2  145
`define INCR  160
`define INCR_2  161
`define DECR  176
`define DECR_2  177
`define ADD  192
`define SUBT  208
`define DIV  224
`define DIV_2  225
`define MUL  32
`define MUL_2  33
`define TOGL  240
```

Chapter 2

MATLAB Codes

2.1 Transmission and Reception

```
clc,clear all,close all;
%% ========================DEFINE SERIAL==================================

fpga = serial('COM17');
fpga.InputBufferSize = 10000000;
fpga.OutputBufferSize = 10000000;
fpga.BaudRate = 115200;

%% ========================LOAD INSTRUCTIONS==============================
option_1=questdlg('Load Instructions ?');
ins_list = {    'div4ds',...
                'DIVdownsample',...
                'downBy3',...
                'downBy5',...
                'bilinear_upsample',...
                'NN_upsample',...
                'GaussianSmoothing',...
                'EdgeDetectVert',...
                'Custom_filter',...
                'PrimeFinder',...
                'Fibbonacci_Sequence',...
                'Pascal_triangle',...
                'factorial'};

ins_display_list = {'Downsample by 2 : Fast',...
                'Downsample by 2 : Initial',...
                'Downsample by 3',...
                'Downsample by 5',...
                'Upsample : Bilinear Interpolation',...
                'Upsample : Near Neighbour Interpolation',...
                'Gaussian Smoothing',...
                'Edge Detector',...
                'Custom Filter',...
                'Prime Finder',...
                'Fibonacci Sequence',...
                'Pascal Triangle',...
                'Factorial'};
```

```
38
39  if(strcmp(option_1,'Yes'))
40      [indx,tf] =
        ↪  listdlg('ListString',ins_display_list,'SelectionMode','single','Name','Algorithms',
        ↪  180],'PromptString','Select an Algorithm');
41      if (tf==1)
42          binary_file=sprintf('D:\\downsampling_processor_fpga\\Finalized
            ↪  Projects\\Project Final_Auto\\Compiler
            ↪  3.0\\bin_%s.txt',char(ins_list(indx)));
43          file=fopen(binary_file);
44          [instructions ins_amount]=fscanf(file,'%i');
45          ins_array=zeros(1,256);
46          ins_array(1:ins_amount)=instructions;
47          ins_array=uint8(ins_array);
48          fclose(file);
49          fprintf('Loading instructions to IRAM.\n');
50          pause(1);
51
52
53          fclose(instrfind);
54          fopen(fpga);
55          fpga.Timeout = 5;
56          fwrite(fpga,ins_array);
57          fclose(instrfind)
58          clc;
59          fprintf('Instructions Loaded.\n');
60          pause(0.5);
61      end
62  end
63
64
65  %% ============================LOAD DATA==================================
66
67
68  if(indx<10)
69      option_2=questdlg('Select the type of image to
        ↪  use.','','RGB','Grayscale','Cancel','Grayscale');
70
71      if(isempty(option_2) | strcmp(option_2,'Cancel'))
72          fprintf('\nProgram Terminated.\n');
73          return ;
74      end
75      im_list = {    'iron-man-3',...
76                     'Team',...
77                     'Landscape',...
78                     'Puppy',...
79                     'Flower',...
80                     'Group',...
81                     'Waterfall',...
82                     'iron-man-3-256'};
83
```

```
84      im_display_list = {'Iron Man 512x512',...
85                         'Team',...
86                         'Landscape',...
87                         'Labrador Puppy',...
88                         'Flower',...
89                         'Group',...
90                         'Waterfall',...
91                         'Iron Man 256x256'};
92      [im_indx,tf] =
        ↪  listdlg('ListString',im_display_list,'SelectionMode','single','Name','Images','ListS
        ↪  150],'PromptString','Select an Image');
93      if (tf==1)
94          image_file=sprintf('%s.png',char(im_list(im_indx)));
95      else
96          image_file=sprintf('iron-man-3.png');
97      end
98      im_in = imread(image_file);      %change image name here
99
100
101
102     if(strcmp(option_2,'Grayscale')| indx==8)
103         if(size(im_in,3) == 3)
104             im_in = rgb2gray(im_in);       %convert to grayscale
105         end
106     end
107
108     iter=size(im_in,3);                                %number of iterations
        ↪  to send
109
110     imwrite(im_in,'D:\downsampling_processor_fpga\Finalized Projects\Project
        ↪  Final_Auto\Processor output\Im_in.png');
111     clc;
112     if(indx~=8)
113         for i=1:iter
114             clc;
115             fprintf('Transmitting Image layer %i into DRAM.........\n',i);
116             im_array = im_in(:,:,i); %select layer to send.
117             im_array = im_array(:);
118             im_array = uint8(im_array);
119
120             fclose(instrfind);
121             fopen(fpga);
122             fpga.Timeout = 30;
123             fwrite(fpga,im_array);
124 %             fclose(instrfind)
125             clc;
126             fprintf('Image layer %i loaded. \n\nWaiting to be processed.\n',i)
127             clc;
128             if(indx==1 | indx==2)
129                 wait_time=10;
130             elseif(indx==3 |indx==4)
```

```
131                 wait_time=8;
132             else
133                 wait_time=30;
134             end
135             fprintf('Receiving processed Image layer %i.\n',i);
136 %             fclose(instrfind);
137 %             fopen(fpga);
138             fpga.Timeout = wait_time;                   % Timeout period in
                ↪  seconds (10 for div by 4 else 30)
139 
140             im_received = fread(fpga);
141             fclose(instrfind);
142             clc;
143             fprintf('Received Image layer %i.\n',i);
144 
145             received_size = ceil(sqrt(numel(im_received)));
146             im_out(:,:,i)=reshape(im_received,[received_size,received_size]);
147             pause(1.5);
148             clc;
149 
150         end
151         fprintf('\nReceived full Image.\n',i);
152         im_out=uint8(im_out);
153 %           figure('NumberTitle', 'off', 'Name', 'Image
    ↪  in'),imshow(im_in),title(sprintf('Original Image.  (Size : 512 x 512)'));
154 %           figure('NumberTitle', 'off', 'Name', 'Image
    ↪  out'),imshow(im_out),title(sprintf('Processed Image.   (Size : %i x
    ↪  %i)',received_size,received_size));
155   figure('NumberTitle', 'off', 'Name', 'Image
    ↪  in'),imshow(im_in),title(sprintf('Original Image.  (Size : 512 x
    ↪  512)'),'FontSize',8);
156         figure('NumberTitle', 'off', 'Name', 'Image
            ↪  out'),imshow(im_out),title([ins_display_list(indx),sprintf('
            ↪  (Size : %i x %i)',received_size,received_size)],'FontSize',8);
157 imwrite(im_out,'D:\downsampling_processor_fpga\Finalized Projects\Project
    ↪  Final_Auto\Processor output\Im_out.png');
158 
159     else
160     for i=1:2
161         clc;
162         fprintf('Phase %i.\n Transmitting Image into DRAM.........\n',i);
163         im_array = im_in(:);
164         im_array = uint8(im_array);
165 
166         fclose(instrfind);
167         fopen(fpga);
168         fpga.Timeout = 30;
169         fwrite(fpga,im_array);
170 %           fclose(instrfind)
171         clc;
172         fprintf('Phase %i.\n\nImage loaded. \n\nWaiting to be processed.\n',i)
```

```
173         clc;
174
175 %         fclose(instrfind);
176 %         fopen(fpga);
177         fpga.Timeout = 30;                  % Timeout period in seconds (10
            ↪  for div by 4 else 30)
178         fprintf('Phase %i.\n\nReceiving processed Image.\n',i);
179         im_received = fread(fpga);
180         fclose(instrfind);
181         clc;
182         fprintf('Phase %i.\n\nReceived Image.\n',i);
183
184         received_size = ceil(sqrt(numel(im_received)));
185         im_out(:,:,i)=reshape(im_received,[received_size,received_size]);
186         pause(1.5);
187         clc;
188     end
189     fprintf('\nReceived full Image.\n',i);
190     im_out=im_out.^2;
191     im_out=sqrt(sum(im_out,3));
192     im_out=uint8(im_out);
193     figure('NumberTitle', 'off', 'Name', 'Image
        ↪  in'),imshow(im_in),title('Original Image','FontSize',8);
194     figure('NumberTitle', 'off', 'Name', 'Image
        ↪  out'),imshow(im_out),title('Edge detection','FontSize',8);
195     imwrite(im_out,'D:\downsampling_processor_fpga\Finalized Projects\Project
        ↪  Final_Auto\Processor output\Im_out.png');
196     end
197
198 %Prime Finder
199
200 elseif(indx==10)
201     fprintf('Writing dummy data.....\n');
202     fclose(instrfind);
203     fopen(fpga);
204     fpga.Timeout = 30;
205 %     fwrite(fpga,randi(255,512*512,1));
206 fwrite(fpga,zeros(512*512,1));
207 %     fclose(instrfind)
208     clc;
209     fprintf('Receiving processed Data.....\n');
210 %     fclose(instrfind);
211 %     fopen(fpga);
212     fpga.Timeout = 5;                   % Timeout period in seconds (10 for div
        ↪  by 4 else 30)
213     im_received = fread(fpga);
214     fclose(instrfind);
215     clc;
216     fprintf('Received Data.\n\n');
217     pause(1);
218     Memory_Index=[0:numel(im_received)-1]';
```

```
219     Data = im_received(:);
220     T=table(Memory_Index,Data);
221      figure('NumberTitle', 'off', 'Name', 'Prime Numbers')
222     uitable('Data',T{:,:},'ColumnName',{' Memory Location ',' Data '},...
223     'RowName',T.Properties.RowNames,'Units', 'Normalized', 'Position',[0, 0,
        ↪  1, 1]);
224
225 %Pascal Triangle
226
227 elseif(indx==12)
228     fprintf('Resetting DRAM.....\n');
229     fclose(instrfind);
230     fopen(fpga);
231     fpga.Timeout = 30;
232     fwrite(fpga,zeros(512*512,1));
233 %     fclose(instrfind)
234     clc;
235     fprintf('Receiving Pascal Triangle Coefficients.....\n');
236 %     fclose(instrfind);
237 %     fopen(fpga);
238     fpga.Timeout = 4;                   % Timeout period in seconds (10 for div
        ↪  by 4 else 30)
239     im_received = fread(fpga);
240     fclose(instrfind);
241     clc;
242     fprintf('Received Data.\n\n');
243     pause(1);
244     pascal=['Pascal Triangle \n\n'];
245     im_out=reshape(im_received,23,11);
246     im_out=im_out';
247     for i=1:11
248         for j=1:23
249            if(im_out(i,j)==0)
250                pascal=[pascal,' '];
251            else
252                pascal=[pascal,int2str(im_out(i,j))];
253            end
254         end
255         pascal=[pascal,'\n'];
256     end
257     fprintf(pascal);
258
259  %Fibonacci
260
261 elseif(indx==11)
262     fprintf('Writing dummy data.....\n');
263     fclose(instrfind);
264     fopen(fpga);
265     fpga.Timeout = 30;
266     fwrite(fpga,randi(255,512*512,1));
267 %     fclose(instrfind)
```

```
268     clc;
269     fprintf('Receiving Fibonacci Sequence.....\n');
270 %     fclose(instrfind);
271 %     fopen(fpga);
272     fpga.Timeout = 4;                   % Timeout period in seconds (10 for div
        ↪  by 4 else 30)
273     im_received = fread(fpga);
274     fclose(instrfind);
275     clc;
276     fprintf('Received Data.\n\n');
277     pause(1);
278     Memory_Index=[0:numel(im_received)-1]';
279     Data = im_received(:);
280     T=table(Memory_Index,Data);
281     figure('NumberTitle', 'off', 'Name', 'Fibonacci Series')
282     uitable('Data',T{:,:},'ColumnName',{' Memory Location ',' Data '},...
283     'RowName',T.Properties.RowNames,'Units', 'Normalized', 'Position',[0, 0,
        ↪  1, 1]);
284
285 %Factorial
286
287 elseif(indx==13)
288     fprintf('Writing dummy data.....\n');
289     fclose(instrfind);
290     fopen(fpga);
291     fpga.Timeout = 30;
292     fwrite(fpga,randi(255,512*512,1));
293 %     fclose(instrfind)
294     clc;
295     fprintf('Receiving Factorial Sequence.....\n');
296 %     fclose(instrfind);
297 %     fopen(fpga);
298     fpga.Timeout = 4;                   % Timeout period in seconds (10 for div
        ↪  by 4 else 30)
299     im_received = fread(fpga);
300     fclose(instrfind);
301     clc;
302     fprintf('Received Data.\n\n');
303     pause(1);
304     Memory_Index=[0:numel(im_received)-1]';
305     Data = im_received(:);
306     T=table(Memory_Index,Data);
307     figure('NumberTitle', 'off', 'Name', 'Factorial Series')
308     uitable('Data',T{:,:},'ColumnName',{' Memory Location ',' Data '},...
309     'RowName',T.Properties.RowNames,'Units', 'Normalized', 'Position',[0, 0,
        ↪  1, 1]);
310
311 end
```

2.2 Error Analysis

```
function SSD = error_analyse_sanke(im_in,im_out,factor);
im_in=double(im_in);
im_out=double(im_out);
ML_down_sampled=im_in;

for k=1:log2(factor) %factor x2,x4,x8
    for j=1:2:511
        for i=1:2:511

            ↪ ML_down_sampled((i+1)/2,(j+1)/2,:)=round((ML_down_sampled(i,j,:)+ML_down_sa
        end
    end
end
[l w h]=size(im_out);
cropped=ML_down_sampled(1:l,1:w,:);
figure,imshow(uint8(cropped));
difference=abs(cropped-im_out);
max(difference);
difference_sqred=difference.^2;
SSD=sum(difference_sqred(:));
fprintf("\nSSD = %f\n",SSD);
% figure,heatmap(sum(difference,3)),title('Heat map');

end
```

Chapter 3

Python Codes

3.1 Compiler

```
# Filename: compile.py
# Compiler program
# Author: Aba

from read_isa import *
import binascii
import pandas as pd
import numpy as np
import os

def compile(fname):
    isa, isa_dict = read_isa()


    program_fName = fname
    program_file  = open(program_fName)

    reg_inc = {'MDAR':0, 'ART':1, 'ARG':2, 'AWT':3, 'AWG':4, 'AC':5, 'KO':6,
    ↪  'K1':7}
    reg_from   = {'AC':0, 'MDDR':1, 'KO':2, 'K1':3,
                  'GO':4, 'G1':5, 'G2':6, 'MIDR':7}
    reg_to   = {'AC':0,'MDDR':1, 'KO':2, 'K1':3,
                'GO':4}

    parameters = { 'LOAD':{'FROM_ADR':0, 'FROM_MAT':1},
                   'LADD':{'TO_MDAR':6, 'TO_AR':7, 'TO_AW':8, 'TO_AR_REF':9,
                   ↪  'TO_AW_REF':10},
                   'STAC':{'TO_ADR':0, 'TO_MAT':2},

                   ↪  'JUMP':{'J':1,'Z_AC':2,'NZ_AC':3,'Z_TOG':4,'NZ_ARG':5,'NZ_ART':6,'NZ
            }

    binary_name = 'binary.txt'
    binary_txt = open(binary_name, "w")

    lineNo = 0;
    loop = {}
```

```
35
36      last_opcode = ""
37
38      isError = False
39
40      program = []
41      program_binary = []
42
43      for line in program_file:
44
45          line = line.strip()
46
47          if(len(line) == 0):
48              continue;
49
50          if(line[0][0] == '$'):                          #loop reference
51              words = line.split()
52              loop[words[0][1:].upper()] = lineNo
53              del words[0]
54              line = ''.join(words)
55
56          if(len(line) == 0):
57              continue;
58
59          line = line.replace(' ', '').replace('\t', '')
60
61
62          if(line[0] == '#'):                             #comment
63              continue
64          words = line.split('#')
65
66          if(len(words) == 0):
67              continue
68
69          word = words[0].upper()                         #Word is operand or opcode
            ↪  or opcode:parameter
70
71
72          if (word[0] == '['):                            #Operand
73              lineNo += 1
74              operand_type = isa['Op'][last_opcode]
75              word = word.replace('[', '').replace(']', '').strip()
76
77              if(operand_type == 'A'):
78                  program_binary.append(word)
79                  program.append(word)
80                  continue
81
82              elif(operand_type == 'I'):
83                  operands = word.replace(' ', '').split(',')
84                  binary_operand = 0
```

```
85
86                  if(len(operands) > 0 and operands[0] == 'ALL'):
87                      program_binary.append(str(255))
88                      program.append('ALL')
89                      continue
90
91                  for given_reg in operands:
92                      if(given_reg in reg_inc):
93                          binary_operand += 2**reg_inc[given_reg]
94                      else:
95                          print ('''Error: Word '%s' in line %s. Expected a
96                                 register name which can be incremented,
    ↪  decremented or reset''' % (word, lineNo))
97                          isError = True
98                          break
99                  if(isError):
100                     break
101
102                 program_binary.append(str(binary_operand))
103                 program.append(word)
104                 continue
105
106             elif(operand_type == 'K'):
107                 if(word.isnumeric):
108                     if(int(word) < 256):
109                         program_binary.append(word)
110                         program.append(word)
111                         continue
112                 print ('''Error: Word '%s' in line %s. Expected a
113                            number < 256''' % (word, lineNo))
114                 isError = True
115                 break
116             elif(operand_type == 'RR'):
117                 if(word in reg_from):
118                     program_binary.append(str(reg_from[word]))
119                     program.append(word)
120                     continue
121                 else:
122                     print ('''Error: Word '%s' in line %s. Expected a
123                            register which can be read into A bus''' % (word,
    ↪  lineNo))
124                     isError = True
125                     break
126
127             elif(operand_type == 'RW'):
128
129                 try:
130                     if('->' in word):
131                         operands = word.replace(' ', '').split('->')
132                         from_reg = operands[0]
133                         to_reg_list = operands[1].split(',')
```

```
134                     elif('<-' in word):
135                         operands = word.replace(' ', '').split('<-')
136                         from_reg = operands[1]
137                         to_reg_list = operands[0].split(',')
138                     else:
139                         print ("Invalid COPY operands in word '%s' in line
                            ↪  %s"% (word, lineNo))
140                         isError = True
141                         break
142                 except:
143                     print ('''Error: Word '%s' in line %s. At least two
                        ↪  registers should be specified''' % (word, lineNo))
144                     isError = True
145                     break
146 
147                 to_reg_binary_sum = 0
148 
149                 if(to_reg_list == ['ALL']):
150                     binary_operand = reg_from[from_reg]*32 + 31
151                     program_binary.append(str(binary_operand))
152                     program.append([from_reg, to_reg_list])
153                     continue;
154 
155                 elif(from_reg in reg_from and len(to_reg_list) != 0):
156                     try:
157                         for to_reg in to_reg_list:
158                             to_reg_binary_sum += 2**reg_to[to_reg]
159                     except:
160                         print ('''Error: Word '%s' in line %s. Expected a
161                            register which can be written into A bus
162                            and a register that can write into A bus''' %
    ↪  (word, lineNo))
163                         isError = True
164                         break
165 
166                     binary_operand = reg_from[from_reg]*32 + to_reg_binary_sum
167                     program_binary.append(str(binary_operand))
168                     program.append([from_reg, to_reg_list])
169                     continue
170 
171                 else:
172                     print ('''Error: Word '%s' in line %s. Expected a
173                            register which can be written into A bus
174                            and a register that can write into A bus''' %
    ↪  (word, lineNo))
175                     isError = True
176                     break
177 
178             elif(operand_type == '3A'):
179                 addr = -1
180 
```

```
181
182                 if(',' in word):
183                     coords = word.split(',')
184
185                     if(len(coords) == 2):
186                         first_half = coords[0]
187                         second_half = coords[1]
188
189                         if(first_half.isnumeric and second_half.isnumeric):
190                             first_half = int(first_half) % 512
191                             second_half = int(second_half) % 512
192
193                             addr = first_half * 512 + second_half
194
195                 elif(word.isnumeric):
196                     addr = int(word)
197
198                 if(addr >= 0 and addr < 512*512 ):
199                     first8 = int(addr/2**16)
200                     reminder = addr % (2**16)
201
202                     mid8 = int(reminder/2**8)
203                     last8 = reminder % (2**8)
204
205                     program_binary.extend([first8, mid8, last8])
206                     program.append(addr)
207                     program.append('-')
208                     program.append('-')
209                     lineNo += 2
210                     continue
211
212                 print ('''Error: Word '%s' in line %s. Expected a
213                             number < 512*512''' % (word, lineNo))
214                 isError = True
215                 break
216
217         word_split = word.split(':')
218
219         opcode = word_split[0]
220
221         if (opcode in isa_dict['BIN'].keys()):        #Opcode
222             lineNo += 1
223             last_opcode = opcode;
224
225             param = '-'
226             param_binary = 0
227
228             if(len(word_split) > 1):
229                 param = word_split[1]
230
231                 if(param == '-'):
```

```
232                     if    (opcode == 'LOAD'):
233                         param = 'FROM_MAT'
234                     elif (opcode == 'STAC'):
235                         param = 'TO_MAT'
236                     elif (opcode == 'LADD'):
237                         param = 'TO_MDR'
238 
239                 try:
240                     if(opcode in ['ADD', 'SUBT']):
241                         param_binary = reg_from[param]
242                     else:
243                         param_binary = parameters[opcode][param]
244                 except:
245                     print("Operand-Parameter mismatch in line %s, word %s with
                        ↪  operand %s, parametr %s"
246                           %(lineNo, word, opcode, param))
247                     isError = True;
248                     break;
249 
250             opcode_binary = isa['BIN'][opcode]
251             output_binary = int(opcode_binary) + int(param_binary)
252 
253 
254             program_binary.append(output_binary)
255             program.append([opcode, param])
256             continue
257 
258         if(isError == True):
259             print("Error Found in line %s in word '%s' " %(lineNo, word))
260 
261     if(lineNo - 1 > 256):
262         print ("Error: Program is more than 256 bytes long. Cannot be stored
            ↪  in memory")
263         isError = True
264 
265     for i in range(len(program_binary)):
266         word = program_binary[i]
267         if(type(word) == int ):
268             binary_txt.write(str(word) + '\n')
269         elif(word.isnumeric()):
270             binary_txt.write(word + '\n')
271         else:
272             if(word in loop.keys()):
273                 binary_txt.write(str(loop[word]) + '\n')
274                 program[i] = loop[word]
275             else:
276                 print ('''Error: loop reference not found
277                           for word '%s' in line %s''' % (word, i+1))
278                 isError = True
279 
280     program_file.close()
```

```
281     binary_txt.close()
282 
283     binary_txt = open(binary_name, 'ab')            #To remove the last
        ↪  newline
284     binary_txt.seek(-2, os.SEEK_END)
285     binary_txt.truncate()
286     binary_txt.close()
287 
288     if(not isError):
289         print("Compilation Successful, with no errors\n")
290 
291     return program
```

3.2 Simulator

```
# Filename  : processor.py
# The Simulator Program
# Author  : Aba

import cv2
from compile import compile
import numpy as np

im_in = cv2.imread('iron-man-31.png', cv2.IMREAD_GRAYSCALE)

d_mem = np.reshape(im_in, [512*512,], order = 'F').tolist()
#d_mem = np.zeros([512*512], dtype = np.uint8).tolist()

reg = {'PC': 0, 'MIDR': 0, 'MDAR': 0, 'MDDR': 0, 'ART': 0, 'ARG': 0
        , 'AWT': 0, 'AWG': 0, 'AC': 0, 'KO': 0, 'K1': 0, 'GO': 0, 'G1': 0,
        'G2': 0, 'Z': 0, 'ZT': 0, 'ZRG': 0, 'ZRT': 0, 'ZKO':0, 'ZK1':0
        , 'ref_ART':0, 'ref_ARG':0, 'ref_AWT':0, 'ref_AWG':0
        , 'ref_KO':0, 'ref_K1':0, 'f_KO':1, 'f_K1': 1}

reg_bits = {'PC': 8, 'MIDR': 8, 'MDAR': 18, 'MDDR': 8, 'ART': 9, 'ARG': 9
        , 'AWT': 9, 'AWG': 9, 'Z': 0, 'ZT': 1, 'ZRG': 1, 'ZRT': 1
        , 'AC': 12, 'KO': 8, 'K1': 8, 'GO': 8, 'G1': 8, 'G2': 8
        , 'ref_ART':9, 'ref_ARG':9, 'ref_AWT':9, 'ref_AWG':9
        , 'ref_KO':1, 'ref_K1':1, 'ZKO':1, 'ZK1':1}

no_of_breaks = {}

def imshow(arg):
    if(arg == 'in'):
        cv2.imshow('Input Image', im_in)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    elif(arg != 'both'):

        im_out = np.transpose(np.reshape(np.array(d_mem, dtype = np.uint8),
        ↪  arg))

        cv2.imshow('Output Image', im_out)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
    else: #Print both
        if(reg['ZT'] == 0):
            last_address = reg['AWG'], reg['AWT']
        else:
            last_address = reg['AWT'], reg['AWG']

        im_out = np.transpose(np.reshape(np.array(d_mem, dtype = np.uint8),
                                        [512,512])[0:last_address[0],
                                        ↪  0:last_address[1]])
```

```
49          #im_out = np.transpose(np.reshape(np.array(d_mem, dtype = np.uint8),
            ↪  [512,512]))
50
51          cv2.imshow('Input Image', im_in)
52          cv2.imshow('Output Image', im_out)
53          cv2.waitKey(0)
54          cv2.destroyAllWindows()
55
56  def INPC():
57      reg['PC'] = reg['PC'] + 1
58
59      if(reg['PC'] < len(i_mem)):
60          reg['MIDR'] = i_mem[reg['PC']]
61
62  def updateZ():
63      if(reg['AC'] == 0):
64          reg['Z'] = 1
65      else:
66          reg['Z'] = 0
67
68  def updateZRG():
69      if(reg['ARG'] == reg['ref_ARG']):
70          reg['ZRG'] = 1
71      else:
72          reg['ZRG'] = 0
73
74  def updateZRT():
75      if(reg['ART'] == reg['ref_ART']):
76          reg['ZRT'] = 1
77      else:
78          reg['ZRT'] = 0
79
80  def updateZK0():
81      if(reg['K0'] ==  reg['ref_K0']):
82          reg['ZK0'] = 1
83      else:
84          reg['ZK0'] = 0
85
86  def updateZK1():
87      if(reg['K1'] == reg['ref_K1']):
88          reg['ZK1'] = 1
89      else:
90          reg['ZK1'] = 0
91
92  def updateREG(reg_name):
93      maxVal = 2**reg_bits[reg_name]
94      reg[reg_name]  =  reg[reg_name] % maxVal
95
96  def TOGL():
97      if(reg['ZT'] == 0):
98          reg['ZT'] = 1
```

```
99      else:
100         reg['ZT'] = 0
101     INPC()
102
103 def LOAD():
104     if(param == 'FROM_ADR'):
105         pass
106     elif(param == 'FROM_MAT'):
107         if(reg['ZT'] == 0):
108             reg['MDAR'] = reg['ARG']*512 + reg['ART']
109         else:
110             reg['MDAR'] = reg['ART']*512 + reg['ARG']
111     else:
112         print("ERROR. Parameter mismatch in LOAD")
113
114     reg['MDDR'] = d_mem[int(reg['MDAR'])]
115     INPC()
116
117 def LADD():
118     INPC()
119     addr = reg['MIDR']
120     INPC()
121     INPC()
122     INPC()
123
124     first_half = int(addr/512)
125     second_half = addr % 512
126
127     if(param == 'TO_MDAR'):
128         reg['MDAR'] = addr
129     elif(param == 'TO_AR'):
130         if(reg['ZT'] == 0):
131             reg['ARG'] = first_half
132             reg['ART'] = second_half
133         else:
134             reg['ART'] = first_half
135             reg['ARG'] = second_half
136     elif(param == 'TO_AW'):
137         if(reg['ZT'] == 0):
138             reg['AWG'] = first_half
139             reg['AWT'] = second_half
140         else:
141             reg['AWT'] = first_half
142             reg['AWG'] = second_half
143     elif(param == 'TO_AR_REF'):
144         if(reg['ZT'] == 0):
145             reg['ref_ARG'] = first_half
146             reg['ref_ART'] = second_half
147         else:
148             reg['ref_ART'] = first_half
149             reg['ref_ARG'] = second_half
```

```
150     else:
151         print("Parameter error in LADD")
152 
153     updateZRG();
154     updateZRT();
155 
156 def LODK():
157     INPC()
158     reg['AC'] = int(reg['MIDR'])
159     INPC()
160 
161     updateZ()
162 
163 def STAC():
164     if(param == 'TO_ADR'):
165         pass
166     elif(param == 'TO_MAT'):
167         if(reg['ZT'] == 0):
168             reg['MDAR'] = reg['AWG']*512 + reg['AWT']
169         else:
170             reg['MDAR'] = reg['AWT']*512 + reg['AWG']
171     else:
172         print("ERROR. Parameter mismatch in STAC")
173 
174     reg['MDDR'] = reg['AC']
175     updateREG('AC')
176     d_mem[int(reg['MDAR'])] = reg['MDDR']
177     INPC()
178 
179 def COPY():
180     INPC()
181     operand = reg['MIDR']
182     INPC()
183 
184     from_reg = operand[0]
185     to_reg_list = operand[1]
186 
187     if(to_reg_list == ['ALL']):
188         for register in reg_bits.keys():
189             if(reg_bits[register] == 8 and register not in ['PC', 'MIDR']):
190                 reg[register] = reg[from_reg]
191 
192                 updateREG(register)
193 
194                 if(register == 'K0' and reg['f_K0'] == 1):
195                     reg['ref_K0'] = reg[from_reg]
196                     reg['f_K0'] = 0
197                 if(register == 'K1' and reg['f_K1'] == 1):
198                     reg['ref_K1'] = reg[from_reg]
199                     reg['f_K1'] = 0
200 
```

```
201
202     else:
203         for to_reg in to_reg_list:
204             if(to_reg == 'G0'):
205                 reg['G2'] = reg['G1']
206                 reg['G1'] = reg['G0']
207                 reg['G0'] = int(reg[from_reg])
208                 updateREG('G0')
209             else:
210                 reg[to_reg] = reg[from_reg]
211                 updateREG(to_reg)
212
213                 if(to_reg == 'K0' and reg['f_K0'] == 1):
214                     reg['ref_K0'] = reg[from_reg]
215                     reg['f_K0'] = 0
216                 if(to_reg == 'K1' and reg['f_K1'] == 1):
217                     reg['ref_K1'] = reg[from_reg]
218                     reg['f_K1'] = 0
219
220
221 def JUMP():
222     INPC()
223     addr = int(reg['MIDR'])
224     INPC()
225
226     updateZ()
227     updateZRG()
228     updateZRT()
229     updateZK0()
230     updateZK1()
231
232     j        = (param == 'J')
233     j_ZAC    = (param == 'Z_AC'      and reg['Z']    == 1)
234     j_NZAC   = (param == 'NZ_AC'     and reg['Z']    == 0)
235     j_ZT     = (param == 'Z_TOG'     and reg['ZT']   == 1)
236     j_NZRT   = (param == 'NZ_ART'    and reg['ZRT']  == 0)
237     j_NZRG   = (param == 'NZ_ARG'    and reg['ZRG']  == 0)
238     j_NZK0   = (param == 'NZ_K0'     and reg['ZK0']  == 0)
239     j_NZK1   = (param == 'NZ_K1'     and reg['ZK1']  == 0)
240
241     j_now = j or j_ZAC or j_NZAC or j_ZT or j_NZRT or j_NZRG or j_NZK0 or
        ↪  j_NZK1
242
243     if(j_now):
244         reg['PC'] = addr
245         reg['MIDR'] = d_mem[reg['PC']]
246
247
248 def INCR():
249     INPC()
250     operand = reg['MIDR']
```

```
251     INPC()
252     operands = operand.replace(' ', '').split(',')
253 
254 
255     for register in operands:
256         reg[register] = reg[register] + 1
257         updateREG(register)
258 
259 def DECR():
260     INPC()
261     operand = reg['MIDR']
262     INPC()
263 
264     operands = operand.replace(' ', '').split(',')
265 
266     for register in operands:
267         reg[register] = reg[register] - 1
268         updateREG(register)
269 
270 def RSET():
271     INPC()
272     operand = reg['MIDR']
273     INPC()
274 
275     if(operand == 'ALL'):
276         for register in reg_bits.keys():
277             if(reg_bits[register] != 1 and register not in ['PC', 'MIDR']):
278                 reg[register] = 0
279                 updateREG(register)
280                 updateZ()
281                 updateZRG()
282                 updateZRT()
283         reg['f_K0'] == 1
284         reg['f_K1'] == 1
285 
286     else:
287         operands = operand.replace(' ', '').split(',')
288         for register in operands:
289             reg[register] = 0
290 
291             if(register == 'K0' or register == 'K1'):
292                 reg['f_'+ register] = 1
293 
294 def ADD():
295     INPC()
296     reg['AC'] = reg['AC'] + reg[param]
297 
298     updateREG('AC')
299 
300 def SUBT():
301     INPC()
```

```
302
303     reg['AC'] = abs(reg['AC'] - reg[param])
304
305     updateREG('AC')
306
307 def DIV():
308     INPC()
309     operand = reg['MIDR']
310     INPC()
311
312     reg['AC'] = int(round(float(reg['AC']) / float(operand),0))
313
314     updateREG('AC')
315
316 def MUL():
317     INPC()
318     operand = reg['MIDR']
319     INPC()
320
321     reg['AC'] = reg['AC'] * int(operand)
322
323     updateREG('AC')
324
325 def NOOP():
326     INPC()
327
328 def END():
329     print('END reached')
330
331
332 instructions = {'TOGL': TOGL, 'LOAD': LOAD, 'LADD': LADD,
333                 'LODK' : LODK, 'STAC': STAC, 'JUMP' : JUMP, 'ADD': ADD,
334                 'INCR': INCR,  'DECR': DECR, 'DIV': DIV, 'MUL': MUL, 'SUBT':
                    ↪  SUBT,
335                 'RSET': RSET, 'COPY': COPY, 'NOOP': NOOP, 'END':END}
336
337 clocks       = {'TOGL': 2, 'LOAD': 2, 'LADD': 9,
338                 'LODK' : 3, 'STAC': 2, 'JUMP' : 4, 'ADD': 2,
339                 'INCR': 3,  'DECR': 3, 'DIV': 3, 'MUL': 3, 'SUBT': 2,
340                 'RSET': 3, 'COPY': 3, 'NOOP': 2, 'END':1}
341
342
343 i_mem = ''
344 param = '';
345
346 total_ins = 0;
347 total_clocks = 3;
348
349 def process(fname, step = False, debug_compiler = False):
350
351     global reg, i_mem, param, d_mem, total_ins, total_clocks
```

```
352
353     i_mem = compile(fname + '.txt')
354     reg['MIDR'] = i_mem[0]
355     isError = False
356
357     while reg['PC'] < len(i_mem):
358         if(step):
359
360             for attribute, value in reg.items():
361                 print('{} : {}'.format(attribute, value))
362             x = input('Step at %i. Enter: CONTINUE, "s": STOP: IMSHOW --' %
363                         (reg['PC']))
364             print(d_mem[0:10])
365             if(x == 's'):
366                 break
367             elif(x == 'i'):
368                 imshow('both')
369         if(i_mem[reg['PC']] == 'END'):
370             break
371
372         if(debug_compiler):
373             opcode_str, param = i_mem[reg['PC']];
374             opcode = instructions[opcode_str]
375
376             opcode()
377
378         else:
379
380             try:
381                 opcode_str, param = i_mem[reg['PC']];
382                 opcode = instructions[opcode_str]
383
384                 opcode()
385
386                 total_ins += 1;
387                 total_clocks += clocks[opcode_str]
388             except:
389                 isError = True;
390                 print('Error at line: ', reg['PC'] + 1, i_mem[reg['PC']],
                    ↪  i_mem[reg['PC']-1])
391                 break
392
393     if(not isError):
394         print('Simulation completed\n')
395         try:
396             print('Total Instructions: ', total_ins)
397             print('Total Clocks: ', total_clocks)
398
399             imshow('both')
400
401             #print(reg['AWG'], reg['AWT'])
```

```
402             except:
403                 print('Error displaying image')
404             #print(d_mem[0:256])
```

3.3 Module to Parse Excel Sheet

```
# Filename  : read_isa.py
# Module to Parse the Excel Sheet into a Dictionary
# Author  : Aba

import pandas as pd
import numpy as np
import sys
sys.path.append('../')

file_name = 'Instruction Set.xlsx'

isa_file = pd.ExcelFile(file_name)

def read_isa():
    isa_df = isa_file.parse('ISA')
    isa_df = isa_df[isa_df['Op'].notnull()]
    isa_df = isa_df[isa_df['BIN'].notnull()]
    isa_df = isa_df.fillna(0)
    isa_df = isa_df[['OPCODE', 'BIN', 'Op']]
    isa_df = isa_df.set_index('OPCODE')
    isa_df[['BIN']] = isa_df[['BIN']].astype(np.uint16)
    isa_df[['BIN']] = isa_df[['BIN']].astype(np.str)

    isa_dict = isa_df.to_dict()

    return isa_df, isa_dict

def read_ins():
    ins_df = isa_file.parse('u-ins')

    ins_df = ins_df[ins_df['u-ins'].notnull()]
    ins_df = ins_df.fillna(0)
    ins_df = ins_df[['u-ins', 'N']]
    ins_df = ins_df.set_index('u-ins')
    ins_df[['N']] = ins_df[['N']].astype(np.uint16)
    ins_df[['N']] = ins_df[['N']].astype(np.str)

    ins_dict = ins_df.to_dict()

    return ins_df, ins_dict
```

3.4 Module to Build Verilog File

```
# Filename  : make_define.py
# Module to Create the Verilog File
# Author  : Aba

from read_isa import *

def make_define():
    ins_df, ins_dict = read_ins()

    v_file = open('opcode_define.v', 'w')

    v_file.write ("// Opcodes and their binary values\n")

    #opcodes = isa_dict['BIN'].keys()
    u_ins = ins_dict['N'].keys()

    for u_in in u_ins:
        binary = ins_dict['N'][u_in]
        v_file.write("`" +"define "+ u_in.upper() + "\t" + str(binary) + '\n')

    print ("Opcode Define file updated successfully\n")
    v_file.close();
```

3.5 User Interface

```
# Filename  : program.py
# Main Compiler-Simulator program that provides User Interface
# Author  : Aba

from compile import *
from processor import *
from make_define import *

print('''
------------------------------------------------------------------------
Greetings!

This program does the following actions for a program
written for the version 5.6 of the CART / ABRUTECH custom matrix-manipulating
processor.

1. Reading the ISA specfied in Excel file
2. Generating Opcode_define Verilog file
3. Syntax-Checking your program written in human language
4. Compiling / Assembling your program into the binary text file
5. Simulating your program by executing the same exact steps of the processor

''')

while(True):
    fname = input('Input file name: ')
    option = input('''Do you wish to simulate? [y/n]: ''')
    try:
        make_define()
        if(fname[-4:] == '.txt'):
            fname = fname[:-4]

        if(fname == ''):
            fname = 'div4ds'

        if (option == 'y'):
            option2 = input('Do you want to simulate step by step? [y/n]: ')
            if(option2 == 'y'):
                process(fname, True)
            else:
                process(fname)
        else:
            compile(fname + '.txt')

        exit_option = input('''\nDo you wish to exit? [y/n]: ''')

        if(exit_option == 'y'):
            input('Thank you for evaluating our processor. Press enter to
            ↪  exit.')
```

```
49              break
50
51      except FileNotFoundError:
52          print('The file you mentioned is not found in the compile directory.
            ↪  Try again')
53
54      print('--------------------------------------------------------------\n')
```