```
1   -- WARNING: Do NOT edit the input and output ports in this file in a text
2   -- editor if you plan to continue editing the block that represents it in
3   -- the Block Editor! File corruption is VERY likely to occur.
4
5   -- Copyright (C) 1991-2010 Altera Corporation
6   -- Your use of Altera Corporation's design tools, logic functions
7   -- and other software and tools, and its AMPP partner logic
8   -- functions, and any output files from any of the foregoing
9   -- (including device programming or simulation files), and any
10  -- associated documentation or information are expressly subject
11  -- to the terms and conditions of the Altera Program License
12  -- Subscription Agreement, Altera MegaCore Function License
13  -- Agreement, or other applicable license agreement, including,
14  -- without limitation, that your use is for the sole purpose of
15  -- programming logic devices manufactured by Altera and sold by
16  -- Altera or its authorized distributors.  Please refer to the
17  -- applicable agreement for further details.
18
19
20  -- Generated by Quartus II Version 9.1 (Build Build 350 03/24/2010)
21  -- Created on Sat Jan 15 11:06:17 2011
22  INCLUDE "lpm_bustri_WORD.inc";
23  INCLUDE "VIDEO/BLITTER/lpm_clshift384.INC";
24  INCLUDE "VIDEO/BLITTER/altsyncram0.INC";
25  INCLUDE "VIDEO/BLITTER/lpm_clshift144.inc";
26
27  --CONSTANT  BL_SKEW_LF = 255;
28
29  --  Title Statement (optional)
30  TITLE "Blitter";
31
32
33  --  Parameters Statement (optional)
34
35  -- {{ALTERA_PARAMETERS_BEGIN}} DO NOT REMOVE THIS LINE!
36  -- {{ALTERA_PARAMETERS_END}} DO NOT REMOVE THIS LINE!
37
38
39  --  Subdesign Section
40
41  SUBDESIGN BLITTER
42  (
43      -- {{ALTERA_IO_BEGIN}} DO NOT REMOVE THIS LINE!
44      nRSTO : INPUT;
45      MAIN_CLK : INPUT;
46      FB_ALE : INPUT;
47      nFB_WR : INPUT;
48      nFB_OE : INPUT;
49      FB_SIZE0 : INPUT;
50      FB_SIZE1 : INPUT;
51      VIDEO_RAM_CTR[15..0] : INPUT;
52      BLITTER_ON : INPUT;
53      FB_ADR[31..0] : INPUT;
54      nFB_CS1 : INPUT;
55      nFB_CS2 : INPUT;
56      nFB_CS3 : INPUT;
57      DDRCLK0 : INPUT;
58      VDP_IN[63..0] : INPUT;
59      BLITTER_DACK[4..0] : INPUT;
60      SR_BLITTER_DACK : INPUT;
61      BLITTER_RUN : OUTPUT;
62      BLITTER_INT : OUTPUT;
63      BLITTER_DOUT[127..0] : OUTPUT;
64      BLITTER_ADR[31..0] : OUTPUT;
65      BLITTER_SIG : OUTPUT;
66      BLITTER_WR : OUTPUT;
67      BLITTER_TA : OUTPUT;
68      FB_AD[31..0] : BIDIR;
69      -- {{ALTERA_IO_END}} DO NOT REMOVE THIS LINE!
70  )
71
72  VARIABLE
73      FB_B[3..0]                  :NODE;
74      FB_16B[1..0]                :NODE;
75      BLITTER_CS                  :NODE;
76      BL_HRAM_CS                  :NODE;
77      BL_HRAM_BE[1..0]            :NODE;
78      BL_HRAM_OUT[15..0]          :NODE;
79      BL_DPRAM_OUT[15..0]         :NODE;
80      BL_SRC_X_INC_CS             :NODE;
81      BL_SRC_X_INC[15..0]         :DFFE;
82      SRC_XINC_NODE[31..0]        :NODE;
83      BL_SRC_Y_INC_CS             :NODE;
84      BL_SRC_Y_INC[15..0]         :DFFE;
85      SRC_YINC_NODE[31..0]        :NODE;
86      BL_ENDMASK1_CS              :NODE;
87      BL_ENDMASK1[15..0]          :DFFE;
88      BL_ENDMASK2_CS              :NODE;
89      BL_ENDMASK2[15..0]          :DFFE;
90      BL_ENDMASK3_CS              :NODE;
91      BL_ENDMASK3[15..0]          :DFFE;
```

```
 92        BL_SRC_ADRH_CS              :NODE;
 93        BL_SRC_ADRL_CS              :NODE;
 94        BL_SRC_ADR[31..0]           :DFFE;
 95        SRC_IADRH_CS                :NODE;
 96        SRC_IADRL_CS                :NODE;
 97        SRC_IADR[31..0]             :DFF;
 98        SRC_IADR_CLR                :NODE;
 99        SIINC                       :NODE;
100        SRC_ADR_NODE[31..0]         :NODE;
101        BL_DST_X_INC_CS             :NODE;
102        BL_DST_X_INC[15..0]         :DFFE;
103        DST_XINC_NODE[31..0]        :NODE;
104        BL_DST_Y_INC_CS             :NODE;
105        BL_DST_Y_INC[15..0]         :DFFE;
106        DST_YINC_NODE[31..0]        :NODE;
107        BL_DST_ADRH_CS              :NODE;
108        BL_DST_ADRL_CS              :NODE;
109        BL_DST_ADR[31..0]           :DFFE;
110        DST_IADRH_CS                :NODE;
111        DST_IADRL_CS                :NODE;
112        DST_IADR[31..0]             :DFF;
113        DST_IADR_CLR                :NODE;
114        DST_ADR_NODE[31..0]         :NODE;
115        DIINC                       :NODE;
116        BL_X_CNT_CS                 :NODE;
117        BL_X_CNT[15..0]             :DFFE;
118        X_CNT_NODE[15..0]           :NODE;
119        BL_Y_CNT_CS                 :NODE;
120        BL_Y_CNT[15..0]             :DFFE;
121        BL_HOP_CS                   :NODE;
122        BL_HOP[7..0]                :DFFE;
123        BL_OP[7..0]                 :DFFE;
124        BL_LN_CS                    :NODE;
125        LN7CLR                      :NODE;
126        BL_LN[7..0]                 :DFFE;
127        BL_SKEW[7..0]               :DFFE;
128    -- barell shifter
129        DIST_RIGHT[7..0]            :NODE;
130        BL_BS_SKEW[7..0]            :NODE;
131        BL_BSIN[383..0]             :NODE;
132        BL_BSOUT[383..0]            :NODE;
133        SHIFT_DIR                   :NODE;
134        BL_SRC_BUF1[127..0]         :DFFE;
135        BL_SRC_BUF2[127..0]         :DFFE;
136        BL_SRC_BUF3[127..0]         :DFFE;
137        BL_DST_BUFRD[127..0]        :DFFE;
138        BL_READ_DST                 :NODE;          -- LATCH SIGNAL DST BUF RD
139        BL_READ_SRC                 :NODE;          -- LATCH SIGNAL SRC BUF
140        SRC_READ                    :NODE;          -- FREIGABE LATCH SIGNAL
141        WREN_B                      :NODE;          -- WR ENA HALFTONE RAM
142        X_INDEX_CS                  :NODE;
143        X_INDEX[15..0]              :DFF;           -- LAUFZEIGER X COUNT
144        X_INDEX_CLR                 :NODE;
145        Y_INDEX_CS                  :NODE;
146        Y_INDEX[15..0]              :DFF;           -- LAUFZEIGER Y COUNT
147        Y_INDEX_CLR                 :NODE;
148        XIINC                       :NODE;          -- INC INDEX SPALTE
149        YIINC                       :NODE;          -- INC INDEX ZEILE
150        ZIINC                       :NODE;          -- INC ADRESSEN ZEILENUMBRUCH
151        FIINC                       :NODE;          -- KORREKTUR ADRESSEN WENN FERTIG
152        HOP_OUT[127..0]             :NODE;
153        OP_OUT[127..0]              :NODE;
154        ENDMASK1_SHIFT[7..0]        :NODE;
155        ENDMASK2_SHIFT[7..0]        :NODE;
156        ENDMASK12_IN[143..0]        :NODE;
157        ENDMASK12_OUT[143..0]       :NODE;
158        ENDMASK23_IN[143..0]        :NODE;
159        ENDMASK23_OUT[143..0]       :NODE;
160        ENDMASK123[127..0]          :NODE;
161        DST_END_LINE_ADR[127..0]    :NODE;
162
163    -- MAIN STATE MACHINE
164        BL_SM                       :MACHINE WITH STATES(START,NEW_LINE,NEW_LINEW,RDSRC1,RDSRC2,RDDST,WRDST,
    TESTZEILENENDE,TESTFERTIG,FERTIG);
165
166    BEGIN
167    -- BYT SELECT 32 BIT
168        FB_B0 = FB_ADR[1..0]==0;                                    -- ADR==0
169        FB_B1 = FB_ADR[1..0]==1                                     -- ADR==1
170              # FB_SIZE1 & !FB_SIZE0 & !FB_ADR1                     -- HIGH WORD
171              # FB_SIZE1 & FB_SIZE0 # !FB_SIZE1 & !FB_SIZE0;       -- LONG UND LINE
172        FB_B2 = FB_ADR[1..0]==2                                     -- ADR==2
173              # FB_SIZE1 & FB_SIZE0 # !FB_SIZE1 & !FB_SIZE0;       -- LONG UND LINE
174        FB_B3 = FB_ADR[1..0]==3                                     -- ADR==3
175              # FB_SIZE1 & !FB_SIZE0 &  FB_ADR1                     -- LOW WORD
176              # FB_SIZE1 & FB_SIZE0 # !FB_SIZE1 & !FB_SIZE0;       -- LONG UND LINE
177    -- BYT SELECT 16 BIT
178        FB_16B0 = FB_ADR[0]==0;                                     -- ADR==0
179        FB_16B1 = FB_ADR[0]==1                                      -- wenn ADR==1
180                # !(!FB_SIZE1 & FB_SIZE0);                          -- or NOT BYT
181    -- BLITTER CS
```

```
182       BLITTER_CS = !nFB_CS1 & FB_ADR[19..7]==H"1F14";          -- FFFF8A00-7F
183       BLITTER_TA = BLITTER_CS;
184   -- REGISTER
185       -- HALFTON RAM
186       BL_HRAM_CS = !nFB_CS1 & FB_ADR[19..5]==H"7C50";                    -- $F8A00-1F.w
187       BL_HRAM_BE1 = BL_HRAM_CS & FB_16B0;
188       BL_HRAM_BE0 = BL_HRAM_CS & FB_16B1;
189       WREN_B = B"0";
190       (BL_DPRAM_OUT[],BL_HRAM_OUT[]) = altsyncram0(FB_ADR[4..1],Y_INDEX[3..0],BL_HRAM_BE[],MAIN_CLK,DDRCLK0,
      FB_AD[31..16],FB_AD[31..16],BL_HRAM_CS & !nFB_WR,WREN_B);
191       -- SRC X INC
192       BL_SRC_X_INC[].CLK = MAIN_CLK;
193       BL_SRC_X_INC[] = FB_AD[31..16];
194       BL_SRC_X_INC_CS = !nFB_CS1 & FB_ADR[19..1]==H"7C510";                  -- $F8A20.w
195       BL_SRC_X_INC[15..8].ENA = BL_SRC_X_INC_CS & !nFB_WR & FB_16B0;
196       BL_SRC_X_INC[7..0].ENA  = BL_SRC_X_INC_CS & !nFB_WR & FB_16B1;
197       SRC_XINC_NODE[] = (H"FFFF0000" & BL_SRC_X_INC15) # (H"0000",BL_SRC_X_INC[]);    -- ERWEITERN AUF 32 BIT
198       -- SRC Y INC
199       BL_SRC_Y_INC[].CLK = MAIN_CLK;
200       BL_SRC_Y_INC[] = FB_AD[31..16];
201       BL_SRC_Y_INC_CS = !nFB_CS1 & FB_ADR[19..1]==H"7C511";                  -- $F8A22.w
202       BL_SRC_Y_INC[15..8].ENA = BL_SRC_Y_INC_CS & !nFB_WR & FB_16B0;
203       BL_SRC_Y_INC[7..0].ENA  = BL_SRC_Y_INC_CS & !nFB_WR & FB_16B1;
204       SRC_YINC_NODE[] = (H"FFFF0000" & BL_SRC_Y_INC15) # (H"0000",BL_SRC_Y_INC[]);    -- ERWEITERN AUF 32 BIT
205       -- SRC ADR HIGH
206       BL_SRC_ADR[].CLK = MAIN_CLK;
207       BL_SRC_ADR[31..16] = FB_AD[31..16];
208       BL_SRC_ADRH_CS = !nFB_CS1 & FB_ADR[19..1]==H"7C512";                   -- $F8A24.w
209       BL_SRC_ADR[31..24].ENA = BL_SRC_ADRH_CS & !nFB_WR & FB_16B0;
210       BL_SRC_ADR[23..16].ENA = BL_SRC_ADRH_CS & !nFB_WR & FB_16B1;
211       -- SRC ADR LOW
212       BL_SRC_ADR[].CLK = MAIN_CLK;
213       BL_SRC_ADR[15..0] = FB_AD[31..16];
214       BL_SRC_ADRL_CS = !nFB_CS1 & FB_ADR[19..1]==H"7C513";                   -- $F8A26.w
215       BL_SRC_ADR[15..8].ENA = BL_SRC_ADRL_CS & !nFB_WR & FB_16B0;
216       BL_SRC_ADR[7..0].ENA  = BL_SRC_ADRL_CS & !nFB_WR & FB_16B1;
217       SRC_IADR[].CLK = DDRCLK0;
218       SRC_IADRH_CS = !nFB_CS1 & FB_ADR[19..1]==H"7C520";                     -- $F8A40.w
219       SRC_IADRL_CS = !nFB_CS1 & FB_ADR[19..1]==H"7C521";                     -- $F8A42.w
220       SRC_IADR_CLR = (BL_SRC_ADRL_CS # BL_SRC_ADRH_CS) & !nFB_WR;            -- LÖSCHEN BEI WRITE
221       SRC_IADR[] = (SRC_IADR[] + (((8 * SRC_XINC_NODE[]) & SIINC) + (((SRC_YINC_NODE[] + ((0,BL_X_CNT[]) - (0
      ,X_INDEX[]) - 8) * SRC_XINC_NODE[])) & ZIINC) - (SRC_YINC_NODE[] & FIINC)) & SRC_READ) & !SRC_IADR_CLR;
222       SRC_ADR_NODE[] = BL_SRC_ADR[] + SRC_IADR[];
223       -- ENDMASK 1
224       BL_ENDMASK1[].CLK = MAIN_CLK;
225       BL_ENDMASK1[] = FB_AD[31..16];
226       BL_ENDMASK1_CS = !nFB_CS1 & FB_ADR[19..1]==H"7C514";                   -- $F8A28.w
227       BL_ENDMASK1[15..8].ENA = BL_ENDMASK1_CS & !nFB_WR & FB_16B0;
228       BL_ENDMASK1[7..0].ENA  = BL_ENDMASK1_CS & !nFB_WR & FB_16B1;
229       -- ENDMASK 2
230       BL_ENDMASK2[].CLK = MAIN_CLK;
231       BL_ENDMASK2[] = FB_AD[31..16];
232       BL_ENDMASK2_CS = !nFB_CS1 & FB_ADR[19..1]==H"7C515";                   -- $F8A2A.w
233       BL_ENDMASK2[15..8].ENA = BL_ENDMASK2_CS & !nFB_WR & FB_16B0;
234       BL_ENDMASK2[7..0].ENA  = BL_ENDMASK2_CS & !nFB_WR & FB_16B1;
235       -- ENDMASK 3
236       BL_ENDMASK3[].CLK = MAIN_CLK;
237       BL_ENDMASK3[] = FB_AD[31..16];
238       BL_ENDMASK3_CS = !nFB_CS1 & FB_ADR[19..1]==H"7C516";                   -- $F8A2C.w
239       BL_ENDMASK3[15..8].ENA = BL_ENDMASK3_CS & !nFB_WR & FB_16B0;
240       BL_ENDMASK3[7..0].ENA  = BL_ENDMASK3_CS & !nFB_WR & FB_16B1;
241       -- DST X INC
242       BL_DST_X_INC[].CLK = MAIN_CLK;
243       BL_DST_X_INC[] = FB_AD[31..16];
244       BL_DST_X_INC_CS = !nFB_CS1 & FB_ADR[19..1]==H"7C517";                  -- $F8A2E.w
245       BL_DST_X_INC[15..8].ENA = BL_DST_X_INC_CS & !nFB_WR & FB_16B0;
246       BL_DST_X_INC[7..0].ENA  = BL_DST_X_INC_CS & !nFB_WR & FB_16B1;
247       DST_XINC_NODE[] = (H"FFFF0000" & BL_DST_X_INC15) # (H"0000",BL_DST_X_INC[]);    -- ERWEITERN AUF 32 BIT
248       -- DST Y INC
249       BL_DST_Y_INC[].CLK = MAIN_CLK;
250       BL_DST_Y_INC[] = FB_AD[31..16];
251       BL_DST_Y_INC_CS = !nFB_CS1 & FB_ADR[19..1]==H"7C518";                  -- $F8A30.w
252       BL_DST_Y_INC[15..8].ENA = BL_DST_Y_INC_CS & !nFB_WR & FB_16B0;
253       BL_DST_Y_INC[7..0].ENA  = BL_DST_Y_INC_CS & !nFB_WR & FB_16B1;
254       DST_YINC_NODE[] = (H"FFFF0000" & BL_DST_Y_INC15) # (H"0000",BL_DST_Y_INC[]);    -- ERWEITERN AUF 32 BIT
255       -- DST ADR HIGH
256       BL_DST_ADR[].CLK = MAIN_CLK;
257       BL_DST_ADR[31..16] = FB_AD[31..16];
258       BL_DST_ADRH_CS = !nFB_CS1 & FB_ADR[19..1]==H"7C519";                   -- $F8A32.w
259       BL_DST_ADR[31..24].ENA = BL_DST_ADRH_CS & !nFB_WR & FB_16B0;
260       BL_DST_ADR[23..16].ENA = BL_DST_ADRH_CS & !nFB_WR & FB_16B1;
261       -- DST ADR LOW
262       BL_DST_ADR[].CLK = MAIN_CLK;
263       BL_DST_ADR[15..0] = FB_AD[31..16];
264       BL_DST_ADRL_CS = !nFB_CS1 & FB_ADR[19..1]==H"7C51A";                   -- $F8A34.w
265       BL_DST_ADR[15..8].ENA = BL_DST_ADRL_CS & !nFB_WR & FB_16B0;
266       BL_DST_ADR[7..0].ENA  = BL_DST_ADRL_CS & !nFB_WR & FB_16B1;
267       DST_IADR[].CLK = DDRCLK0;
268       DST_IADRH_CS = !nFB_CS1 & FB_ADR[19..1]==H"7C522";                     -- $F8A44.w
269       DST_IADRL_CS = !nFB_CS1 & FB_ADR[19..1]==H"7C523";                     -- $F8A46.w
270       DST_IADR_CLR = (BL_DST_ADRL_CS # BL_DST_ADRH_CS) & !nFB_WR;            -- LÖSCHEN BEI WRITE
```

```
271       DST_IADR[] = (DST_IADR[] + ((8 * DST_XINC_NODE[]) & DIINC) + ((DST_YINC_NODE[] + (((0,BL_X_CNT[]) - (0,
      X_INDEX[])) * DST_XINC_NODE[])) & ZIINC) - (DST_YINC_NODE[] & FIINC)) & !DST_IADR_CLR;
272       DST_ADR_NODE[] = BL_DST_ADR[] + DST_IADR[];
273       -- X COUNT
274       BL_X_CNT[].CLK = MAIN_CLK;
275       BL_X_CNT[] = FB_AD[31..16];
276       BL_X_CNT_CS = !nFB_CS1 & FB_ADR[19..1]==H"7C51B";                    -- $F8A36.w
277       BL_X_CNT[15..8].ENA = BL_X_CNT_CS & !nFB_WR & FB_16B0;
278       BL_X_CNT[7..0].ENA  = BL_X_CNT_CS & !nFB_WR & FB_16B1;
279       X_INDEX[].CLK = DDRCLK0;
280       X_INDEX_CS = !nFB_CS1 & FB_ADR[19..1]==H"7C524";                     -- $F8A48.w
281       X_INDEX_CLR = BL_X_CNT_CS & !nFB_WR;                                 -- LÖSCHEN BEI WRITE
282       X_INDEX[] = (X_INDEX[] + (8 & XIINC)) & !X_INDEX_CLR; -- + (BL_X_CNT[] - X_INDEX[]) & ZIINC
283       X_CNT_NODE[] = X_INDEX[] - ((0,DST_ADR_NODE[3..1]) & (X_INDEX[]!=0));-- EFFEKTIV GELESENE
284       -- Y COUNT
285       BL_Y_CNT[].CLK = MAIN_CLK;
286       BL_Y_CNT[] = FB_AD[31..16];
287       BL_Y_CNT_CS = !nFB_CS1 & FB_ADR[19..1]==H"7C51C";                    -- $F8A38.w
288       BL_Y_CNT[15..8].ENA = BL_Y_CNT_CS & !nFB_WR & FB_16B0;
289       BL_Y_CNT[7..0].ENA  = BL_Y_CNT_CS & !nFB_WR & FB_16B1;
290       Y_INDEX[].CLK = DDRCLK0;
291       Y_INDEX_CS = !nFB_CS1 & FB_ADR[19..1]==H"7C525";                     -- $F8A4A.w
292       Y_INDEX_CLR = BL_Y_CNT_CS & !nFB_WR;                                 -- LÖSCHEN BEI WRITE
293       Y_INDEX[] = (Y_INDEX[] + (1 & YIINC)) & !Y_INDEX_CLR;
294       -- HOP LOGIC
295       BL_HOP[].CLK = MAIN_CLK;
296       BL_HOP[] = FB_AD[31..24];
297       BL_HOP_CS = !nFB_CS1 & FB_ADR[19..1]==H"7C51D";                      -- $F8A3A.w
298       BL_HOP[7..0].ENA  = BL_HOP_CS & !nFB_WR & FB_16B0;                   -- $F8A3A
299       -- OP LOGIC
300       BL_OP[].CLK = MAIN_CLK;
301       BL_OP[] = FB_AD[23..16];
302       BL_OP[7..0].ENA  = BL_HOP_CS & !nFB_WR & FB_16B1;                    -- $F8A3B
303       -- LINE NUMBER BYT
304       BL_LN[].CLK = MAIN_CLK;
305       BL_LN[6..0] = FB_AD[30..24];
306       BL_LN7 = FB_AD31 & !LN7CLR;                                          -- BUSY HOG UND SMUDGE
307       BL_LN_CS = !nFB_CS1 & FB_ADR[19..1]==H"7C51E";                       -- $F8A3C.w
308       BL_LN[].ENA  = BL_LN_CS & !nFB_WR & FB_16B0;                         -- $F8A3C
309       BL_LN7.ENA = LN7CLR;
310       -- SKEW BYT
311       BL_SKEW[].CLK = MAIN_CLK;
312       BL_SKEW[] = FB_AD[23..16];
313       BL_SKEW[].ENA  = BL_LN_CS & !nFB_WR & FB_16B1;                       -- $F8A3D
314   --- REGISTER OUT
315       FB_AD[31..16] = lpm_bustri_WORD(
316                           BL_HRAM_CS      & BL_DPRAM_OUT[]
317                         # BL_SRC_X_INC_CS & BL_SRC_X_INC[]
318                         # BL_SRC_Y_INC_CS & BL_SRC_Y_INC[]
319                         # BL_SRC_ADRH_CS  & SRC_ADR_NODE[31..16]
320                         # BL_SRC_ADRL_CS  & SRC_ADR_NODE[15..0]
321                         # BL_ENDMASK1_CS  & BL_ENDMASK1[]
322                         # BL_ENDMASK2_CS  & BL_ENDMASK2[]
323                         # BL_ENDMASK3_CS  & BL_ENDMASK3[]
324                         # BL_DST_X_INC_CS & BL_DST_X_INC[]
325                         # BL_DST_Y_INC_CS & BL_DST_Y_INC[]
326                         # BL_DST_ADRH_CS  & DST_ADR_NODE[31..16]
327                         # BL_DST_ADRL_CS  & DST_ADR_NODE[15..0]
328                         # BL_X_CNT_CS     & (BL_X_CNT[]-X_CNT_NODE[])
329                         # BL_Y_CNT_CS     & (BL_Y_CNT[]-Y_INDEX[])
330                         # BL_HOP_CS       & (BL_HOP[],BL_OP[])
331                         # BL_LN_CS        & (BL_LN[7..4],Y_INDEX[3..0],BL_SKEW[])
332                         # SRC_IADRH_CS    & SRC_IADR[31..16]
333                         # SRC_IADRL_CS    & SRC_IADR[15..0]
334                         # DST_IADRH_CS    & DST_IADR[31..16]
335                         # DST_IADRL_CS    & DST_IADR[15..0]
336                         # X_INDEX_CS      & X_INDEX[]
337                         # Y_INDEX_CS      & Y_INDEX[]
338                         ,BLITTER_CS & !nFB_OE);                                  -- FFFF8A00-7F
339   ----------------------------------------
340   -- SRC BUFFER LADEN
341       BL_SRC_BUF1[].CLK = DDRCLK0;
342       BL_SRC_BUF1[127..64].ENA = BLITTER_DACK1 & BL_READ_SRC;
343       BL_SRC_BUF1[63..0].ENA   = BLITTER_DACK0 & BL_READ_SRC;
344       BL_SRC_BUF1[] = (VDP_IN[],VDP_IN[]);
345       BL_SRC_BUF2[].CLK = DDRCLK0;
346       BL_SRC_BUF2[127..64].ENA = BLITTER_DACK1 & BL_READ_SRC;
347       BL_SRC_BUF2[63..0].ENA   = BLITTER_DACK0 & BL_READ_SRC;
348       BL_SRC_BUF2[] = BL_SRC_BUF1[];
349       BL_SRC_BUF3[].CLK = DDRCLK0;
350       BL_SRC_BUF3[127..64].ENA = BLITTER_DACK1 & BL_READ_SRC;
351       BL_SRC_BUF3[63..0].ENA   = BLITTER_DACK0 & BL_READ_SRC;
352       BL_SRC_BUF3[] = BL_SRC_BUF2[];
353       -- ZUORDNUNG ---------
354       BL_BSIN[255..128] = BL_SRC_BUF2[];
355       IF !BL_SRC_X_INC15 THEN     -- WENN POSITIV NORMALE REIHENFOLGE
356           BL_BSIN[127..0] = BL_SRC_BUF1[];
357           BL_BSIN[383..256] = BL_SRC_BUF3[];
358       ELSE                        -- SONST UMGEKEHRT
359           BL_BSIN[127..0] = BL_SRC_BUF3[];
360           BL_BSIN[383..256] = BL_SRC_BUF1[];
```

```
361       END IF;
362   -- DST BUFFER READ
363       BL_DST_BUFRD[].CLK = DDRCLK0;
364       BL_DST_BUFRD[127..64].ENA = BLITTER_DACK1 & BL_READ_DST;
365       BL_DST_BUFRD[63..0].ENA   = BLITTER_DACK0 & BL_READ_DST;
366       BL_DST_BUFRD[] = (VDP_IN[],VDP_IN[]);
367   -- barell shift ************************************************************************
368   -- SOURCE SHIFT RIGHT = LPM_CSHIFT RIGTH ;SKEW SHIFT: IF FXRS==0 THEN RIGHT ELSE LEFT
369       DIST_RIGHT[] = (16 * ((0,DST_ADR_NODE[3..1]) - (0,SRC_ADR_NODE[3..1]))) + (!BL_SKEW7 &  (0,BL_SKEW[3..0
      ])) - (BL_SKEW7 & (0,BL_SKEW[3..0]));
370       IF DIST_RIGHT[] >= 0 THEN
371           BL_BS_SKEW[] = DIST_RIGHT[];                          -- LPM SHIFT RIGHT
372           SHIFT_DIR = VCC;                                      -- DIR = RIGHT
373       else
374           BL_BS_SKEW[] = 0 - DIST_RIGHT[];                      -- LPM SHIFT LEFT
375           SHIFT_DIR = GND;                                      -- DIR = LEFT
376       end if;
377   -- barell shifter: direction 0=links 1=rechts IN BEZUG AUF ausgabewert!
378       BL_BSOUT[] = lpm_clshift384(BL_BSIN[], SHIFT_DIR , BL_BS_SKEW[]);   -- wir brauchen 128bit
379   -- HOP ************************************************************************************
380       CASE BL_HOP[1..0] IS
381           WHEN H"0" =>
382                       --    12345678901234567890123456789012
383               HOP_OUT[] = H"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF";
384           WHEN H"1" =>
385               HOP_OUT[] = (BL_HRAM_OUT[],BL_HRAM_OUT[],BL_HRAM_OUT[],BL_HRAM_OUT[],BL_HRAM_OUT[],BL_HRAM_OUT
      [],BL_HRAM_OUT[],BL_HRAM_OUT[]);
386           WHEN H"2" =>
387               HOP_OUT[] = BL_BSOUT[255..128];
388           WHEN OTHERS =>
389               HOP_OUT[] = (BL_BSOUT[255..128] & (BL_HRAM_OUT[],BL_HRAM_OUT[],BL_HRAM_OUT[],BL_HRAM_OUT[],
      BL_HRAM_OUT[],BL_HRAM_OUT[],BL_HRAM_OUT[],BL_HRAM_OUT[]));
390       END CASE;
391   -- OP *************************************************************************************
392       CASE BL_OP[3..0] IS
393           WHEN H"0" =>
394               OP_OUT[] = H"0";
395               SRC_READ = B"0";
396           WHEN H"1" =>
397               OP_OUT[] = HOP_OUT[] & BL_DST_BUFRD[];
398               SRC_READ = BL_HOP1 # BL_HOP0;
399           WHEN H"2" =>
400               OP_OUT[] = HOP_OUT[] & !BL_DST_BUFRD[];
401               SRC_READ = BL_HOP1 # BL_HOP0;
402           WHEN H"3" =>
403               OP_OUT[] = HOP_OUT[];
404               SRC_READ = BL_HOP1 # BL_HOP0;
405           WHEN H"4" =>
406               OP_OUT[] = !HOP_OUT[] & BL_DST_BUFRD[];
407               SRC_READ = BL_HOP1 # BL_HOP0;
408           WHEN H"5" =>
409               OP_OUT[] = BL_DST_BUFRD[];
410               SRC_READ = B"0";
411           WHEN H"6" =>
412               OP_OUT[] = HOP_OUT[] $ BL_DST_BUFRD[];
413               SRC_READ = BL_HOP1 # BL_HOP0;
414           WHEN H"7" =>
415               OP_OUT[] = HOP_OUT[] # BL_DST_BUFRD[];
416               SRC_READ = BL_HOP1 # BL_HOP0;
417           WHEN H"8" =>
418               OP_OUT[] = !HOP_OUT[] & !BL_DST_BUFRD[];
419               SRC_READ = BL_HOP1 # BL_HOP0;
420           WHEN H"9" =>
421               OP_OUT[] = !HOP_OUT[] $ BL_DST_BUFRD[];
422               SRC_READ = BL_HOP1 # BL_HOP0;
423           WHEN H"A" =>
424               OP_OUT[] = !BL_DST_BUFRD[];
425               SRC_READ = B"0";
426           WHEN H"B" =>
427               OP_OUT[] = HOP_OUT[] # !BL_DST_BUFRD[];
428               SRC_READ = BL_HOP1 # BL_HOP0;
429           WHEN H"C" =>
430               OP_OUT[] = !HOP_OUT[];
431               SRC_READ = BL_HOP1 # BL_HOP0;
432           WHEN H"D" =>
433               OP_OUT[] = !HOP_OUT[] # BL_DST_BUFRD[];
434               SRC_READ = BL_HOP1 # BL_HOP0;
435           WHEN H"E" =>
436               OP_OUT[] = !HOP_OUT[] # !BL_DST_BUFRD[];
437               SRC_READ = BL_HOP1 # BL_HOP0;
438           WHEN OTHERS =>
439                       --   12345678901234567890123456789012
440               OP_OUT[] = H"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF";
441               SRC_READ = B"0";
442       END CASE;
443   ------------ ENDMASKEN SETZEN
      ********************************************************************************

444               ENDMASK1_SHIFT[3..0] = 0;
445               ENDMASK2_SHIFT[3..0] = 0;
446               IF X_INDEX[]==0 THEN                                  -- ANFANG?
```

```
447                 ENDMASK1_SHIFT[7..4] = 1 + (0,(DST_ADR_NODE[3..1]));                    -- JA ->
    ENDMASK 1 SETZEN
448             ELSE
449                 ENDMASK1_SHIFT[7..4] = 0;                                               --
    NEIN->ENDMASK1  AUF ENDMASK2 SETZEN
450             END IF;
451             DST_END_LINE_ADR[] = DST_ADR_NODE[] + (8 * DST_XINC_NODE[]);
452             IF BL_X_CNT[]<=(X_CNT_NODE[] + 8) THEN                                  -- SCHON
    ZEILENENDE?
453                 ENDMASK2_SHIFT[7..4] = 1 + (0,(DST_END_LINE_ADR[3..1]));            -- JA ENDMASK
    3 SETZEN
454             ELSE
455                 ENDMASK2_SHIFT[7..4] = 0;                                           -- NOCH NICHT
    AKTIV->ENDMASK 3 AUF ENDMASK2 SETZEN
456             END IF;
457 -- ENDMASKEN -- barell shifter 144 bit, direction 0 = links 1 = rechts
458 --               123456789012345678901234567 8
459     ENDMASK12_IN[] = (BL_ENDMASK1[],BL_ENDMASK2[],BL_ENDMASK2[],BL_ENDMASK2[],BL_ENDMASK2[],BL_ENDMASK2[],
    BL_ENDMASK2[],BL_ENDMASK2[],BL_ENDMASK2[]);
460     ENDMASK12_OUT[] = lpm_clshift144(ENDMASK12_IN[],1,ENDMASK1_SHIFT[]);        -- IMMER rechts SCHIEBEN
461     ENDMASK23_IN[] = (BL_ENDMASK2[],BL_ENDMASK2[],BL_ENDMASK2[],BL_ENDMASK2[],BL_ENDMASK2[],BL_ENDMASK2[],
    BL_ENDMASK2[],BL_ENDMASK2[],BL_ENDMASK3[]);
462     ENDMASK23_OUT[] = lpm_clshift144(ENDMASK23_IN[],0,ENDMASK2_SHIFT[]);        -- IMMER LINKS SCHIEBEN
463     ENDMASK123[] = ENDMASK12_OUT[127..0] & ENDMASK23_OUT[143..16];
464     BLITTER_DOUT[] = ((ENDMASK123[] & OP_OUT[]) # (!ENDMASK123[] & BL_DST_BUFRD[]));
465 -- STATE MACHINE
    ********************************************************************************************
466     BLITTER_RUN = BLITTER_ON;
467 -- BLITTER MAIN STATE MACHINE  ---------------------------------------------
468     BL_SM.CLK = DDRCLK0;
469     CASE BL_SM IS
470         WHEN START =>   ------------------------- START
471             IF BLITTER_ON & BL_LN7 & ((BL_X_CNT[] - X_CNT_NODE[])>0) & ((BL_Y_CNT[] - Y_INDEX[]) > 0) THEN
472                 BL_SM = NEW_LINE;
473             ELSE
474                 BL_SM = START;
475             END IF;
476         WHEN NEW_LINE => ----------------------- NEU LINIE
477             X_INDEX_CLR = VCC;                              -- LÖSCHEN
478             BL_SM = RDSRC1;
479         WHEN RDSRC1 =>  ------------------------ READ SRC1
480             IF SRC_READ THEN
481                 BLITTER_ADR[] = (SRC_ADR_NODE[31..4],B"0000");
482                 BLITTER_SIG = VCC;
483                 BL_READ_SRC = VCC;                          -- LATCH UND SB1->SB2
484                 IF BLITTER_DACK0 THEN
485                     SIINC = VCC;                            -- INC SRC ADR
486                     BL_SM = RDSRC2;
487                 ELSE
488                     BL_SM = RDSRC1;
489                 END IF;
490             ELSE
491                 BL_SM = RDDST;
492             END IF;
493         WHEN RDSRC2 =>  ------------------------ READ SRC2
494             IF SRC_READ THEN
495                 BLITTER_ADR[] = (SRC_ADR_NODE[31..4],B"0000");
496                 BLITTER_SIG = VCC;
497                 BL_READ_SRC = VCC;                              -- LATCH UND SB1->SB2
498                 IF BLITTER_DACK0 THEN
499                     SIINC = VCC;                                -- INC SRC ADR
500                     BL_SM = RDDST;
501                 ELSE
502                     BL_SM = RDSRC2;
503                 END IF;
504             ELSE
505                 BL_SM = RDDST;
506             END IF;
507         WHEN RDDST =>   ----------------------- READ DEST
508             BLITTER_ADR[] = (DST_ADR_NODE[31..4],B"0000");
509             BLITTER_SIG = VCC;
510             BL_READ_DST = VCC;
511             IF BLITTER_DACK0 THEN
512                 BL_SM = WRDST;
513             ELSE
514                 BL_SM = RDDST;
515             END IF;
516         WHEN WRDST =>   ------------------- WRITE DEST
517             BLITTER_WR = VCC;
518             BLITTER_SIG = VCC;
519             BLITTER_ADR[] = (DST_ADR_NODE[31..4],B"0000");
520             IF BLITTER_DACK0 THEN
521                 XIINC = VCC;                                -- INC X_INDEX
522                 DIINC = VCC;                                -- INC DEST ADR
523                 BL_SM = TESTZEILENENDE;
524             ELSE
525                 BL_SM = WRDST;
526             END IF;
527         WHEN TESTZEILENENDE =>  ----------------- ZEILENDE?
528             IF BL_X_CNT[]<=(X_CNT_NODE[])    THEN           -- SCHON ZEILENENDE?
529                 YIINC = VCC;                                -- JA -> INC Y-INDEX UND ZEILE SRC UND DEST
```

```
530                 BL_SM = TESTFERTIG;                         -- ->
531             ELSE
532                 BL_SM = RDSRC2;                             -- NEIN NEXT
533             END IF;
534         WHEN TESTFERTIG =>  ---------------------- TEST AUF FERTIG
535             ZIINC = VCC;                                    -- INC ADRESSEN ZEILENUMBRUCH
536             IF Y_INDEX[]>=BL_Y_CNT[] THEN                   -- LETZTE ZEILE?
537                 BL_SM = FERTIG;                             -- JA -->
538             ELSE
539                 BL_SM = NEW_LINE;                           -- NEIN NEXT ->
540             END IF;
541         WHEN FERTIG => --------------------------- FERTIG
542             FIINC = VCC;                                    -- KORREKTUR ADRESSEN
543             BLITTER_INT = VCC;                              -- BLITTER INTERRUPT
544             LN7CLR = VCC;
545             IF BL_LN7==0 THEN                               -- BUSY BIT LÖSCHEN
546                 BL_SM = START;
547             ELSE
548                 BL_SM = FERTIG;
549             END IF;
550         WHEN OTHERS =>
551             BL_SM = FERTIG;
552     END CASE;
553 END;
554
555
```