```vhdl
--345678901234567890123456789012345678901234567890123456789012345678901234567890
-- $Id: internal_memory.vhd,v 1.2 2004/12/08 22:52:28 tofp Exp $
--*****************************************************************************
--*  INTERNAL_MEMORY.VHD
--*
--*
--*  REVISION HISTORY:
--*    11-Oct-2001 CS  Original coding
--*    31-Oct-2002 CS  Interface has been revised
--*                    Memory is going to be written and read using user defined
--*                    block write and read, respectively.
--*
--*****************************************************************************

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.ALL;

ENTITY internal_memory IS
  PORT (
    clock       : IN  std_logic;
    arstn       : IN  std_logic;
    block_read  : IN  std_logic;
    block_write : IN  std_logic;
    tid         : IN  std_logic_vector (3 DOWNTO 0);
    datai       : IN  std_logic_vector (31 DOWNTO 0);
    datai_valid : IN  std_logic;
    suspend     : IN  std_logic;
    datao       : OUT std_logic_vector (32 DOWNTO 0);
    datao_valid : OUT std_logic
    );
END internal_memory;

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

USE work.my_conversions.ALL;
USE work.my_utilities.ALL;

ARCHITECTURE SYN OF internal_memory IS

  CONSTANT LENGTH512 : std_logic_vector := "0000000001000000000";

  TYPE write_state IS (
    WRSTOP,
    WRINIT,
    WRDATA);

  TYPE read_state IS (
    RDSTOP,
    RDINIT,
    RDFETCH,
    RDDATA,
    RDIDLE,
    RDCLOSE,
    RDEND);

  SIGNAL s_im_address : std_logic_vector (8 DOWNTO 0);
  SIGNAL s_im_we      : std_logic;
  SIGNAL s_im_outenab : std_logic;
  SIGNAL s_im_dio     : std_logic_vector (31 DOWNTO 0);
  SIGNAL s_im_din     : std_logic_vector (31 DOWNTO 0);
```

```
67      SIGNAL s_im_dout     : std_logic_vector (31 DOWNTO 0);
68
69    BEGIN
70
71      s_im_we  <= datai_valid AND block_write;
72      s_im_din <= datai;
73
74      altsyncram_component : altsyncram
75        GENERIC MAP (
76          operation_mode                      => "SINGLE_PORT",
77          width_a                             => 32,
78          widthad_a                           => 9,
79          numwords_a                          => 512,
80          lpm_type                            => "altsyncram",
81          width_byteena_a                     => 1,
82          outdata_reg_a                       => "UNREGISTERED",
83          outdata_aclr_a                      => "NONE",
84          indata_aclr_a                       => "NONE",
85          wrcontrol_aclr_a                    => "NONE",
86          address_aclr_a                      => "NONE",
87          read_during_write_mode_mixed_ports  => "DONT_CARE",
88          lpm_hint                            => "ENABLE_RUNTIME_MOD=NO",
89          intended_device_family              => "Stratix"
90          )
91        PORT MAP (
92          wren_a    => s_im_we,
93          clock0    => clock,
94          address_a => s_im_address,
95          data_a    => s_im_din,
96          q_a       => s_im_dout
97          );
98
99      main : PROCESS (clock, arstn)
100       VARIABLE address_counter : std_logic_vector (8 DOWNTO 0);
101       VARIABLE word_counter    : std_logic_vector (19 DOWNTO 0);
102       VARIABLE block_end       : std_logic;
103       VARIABLE write_present   : write_state;
104       VARIABLE write_next      : write_state;
105       VARIABLE read_present    : read_state;
106       VARIABLE read_next       : read_state;
107       VARIABLE write_enable    : std_logic;
108       VARIABLE read_enable     : std_logic;
109     BEGIN
110       IF (arstn = '0') THEN
111         s_im_address <= (OTHERS => '0');
112         datao_valid  <= '0';
113         datao        <= (OTHERS => '0');
114
115         address_counter := (OTHERS => '0');
116         word_counter    := (OTHERS => '0');
117         block_end       := '0';
118         write_present   := WRSTOP;
119         write_next      := WRSTOP;
120         read_present    := RDSTOP;
121         read_next       := RDSTOP;
122         write_enable    := '0';
123         read_enable     := '0';
124       ELSIF (clock'event AND clock = '1') THEN
125         write_enable := datai_valid;
126         IF (write_present = WRINIT) OR (read_present = RDINIT) THEN
127           address_counter := (OTHERS => '0');
128         ELSIF (write_enable = '1') OR
129           ((read_enable = '1') AND (suspend = '0')) THEN
130           address_counter := inc(address_counter);
131         END IF;
132         s_im_address <= address_counter;
```

```
133
134         CASE write_present IS
135           WHEN WRSTOP =>
136             IF (block_write = '1') THEN
137               write_next := WRINIT;
138             ELSE
139               write_next := WRSTOP;
140             END IF;
141           WHEN WRINIT =>
142             write_next := WRDATA;
143           WHEN WRDATA =>
144             IF (block_write = '0') THEN
145               write_next := WRSTOP;
146             ELSE
147               write_next := WRDATA;
148             END IF;
149         END CASE;
150         write_present := write_next;
151
152         IF (read_present = RDCLOSE) THEN
153           datao <= "10" & LENGTH512 & tid & "01100100";
154         ELSE
155           datao <= '0' & s_im_dout;
156         END IF;
157
158         block_end := word_counter(19);
159         CASE read_present IS
160           WHEN RDSTOP =>
161             word_counter := '0' & LENGTH512;
162             datao_valid  <= '0';
163             read_enable  := '0';
164             IF (block_read = '1') THEN
165               read_next := RDINIT;
166             ELSE
167               read_next := RDSTOP;
168             END IF;
169           WHEN RDINIT =>
170             word_counter := dec(word_counter);
171             datao_valid  <= '0';
172             read_enable  := '1';
173             read_next    := RDFETCH;
174           WHEN RDFETCH =>
175             word_counter := dec(word_counter);
176             datao_valid  <= '0';
177             read_enable  := '1';
178             IF (suspend = '0') THEN
179               read_next := RDDATA;
180             ELSE
181               read_next := RDIDLE;
182             END IF;
183           WHEN RDIDLE =>
184             datao_valid <= '0';
185             read_enable := '1';
186             IF (suspend = '0') THEN
187               read_next := RDDATA;
188             ELSE
189               read_next := RDIDLE;
190             END IF;
191           WHEN RDDATA =>
192             word_counter := dec(word_counter);
193             datao_valid  <= '1';
194             read_enable  := '1';
195             IF (block_end = '1') THEN
196               read_next := RDCLOSE;
197             ELSIF (suspend = '1') THEN
198               read_next := RDIDLE;
```

```
199             ELSE
200               read_next := RDDATA;
201             END IF;
202           WHEN RDCLOSE =>
203             datao_valid <= '1';
204             read_enable := '1';
205             read_next   := RDEND;
206           WHEN RDEND =>
207             datao_valid <= '0';
208             read_enable := '0';
209             IF block_read = '0' THEN
210               read_next := RDSTOP;
211             ELSE
212               read_next := RDEND;
213             END IF;
214         END CASE;
215         read_present := read_next;
216
217       END IF;
218     END PROCESS;
219
220   END SYN;
221
```