1. Minimal Sorting Network

A sorting network is an algorithm for sorting items. Sorting process is performed with a sequence of comparison-exchange operations that are executed in a fixed order. Figure 1 shows a sorting network for four items that are a, b, c and d. The numbers within the circles represent states for each comparison-exchange operation.

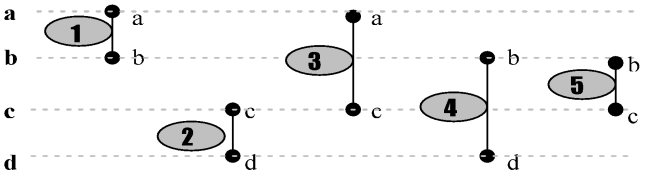


Figure 1 Minimal sorting network for four items.

The items, a, b, c and d to be sorted are given at the left end of the horizontal lines. The first vertical line connecting two upper horizontal lines indicates that items a and b are to be compared and exchanged, if necessary, so that the larger one is always at the bottom. This step and the next consecutive three steps cause the largest and smallest items to be routed down and up, respectively. The fifth step ensures that the remaining two items end up in the correct order and therefore correctly sorted output is obtained at the end of the fifth step. A five-step network is known to be minimal for four items (Koza et.al 1998a).

Sorting networks are independent of their inputs in the sense that they always perform the same fixed sequence of comparison-exchange operations. Sorting networks are also more efficient for sorting small numbers of items than the well-known non-oblivious sorting algorithms such as Quicksort (Koza et.al 1998b).

2. Minimal Sorting Algorithm

To efficiently sort a small number of items, we neither attempt to directly map a general purpose-sorting algorithm into hardware or use a brute force hardware implementation. Instead, the concept of minimal sorting networks is used (Knuth, 1973), where the principle objective of a Minimum Sorting Algorithm (MSA) is to sort items by using the minimal number of pair wise comparison-exchange operations. In the case of 4 items the minimum number of such operations is five. Table 1 summarises the processing steps of the MSA.

Table 1 Example of 4 item sort using MSN

Items	Initial Values	T_1	T_2	T_3	T_4	T_5	Sorted Items
A	7	5		4			4
B	5	7			6	5	5
C	6		4	5		6	6
D	4		6		7		7

As an example, consider the case in which the four items take the following values: a(7), b(5), c(6), d(4). Table 1 illustrates the progress of the search for this example. However, from a hardware perspective this is expensive. Because exchange operations of long integer values requires more hardware components. If the index values that are pointing to the corresponding items are used then the implementation will be independent of item value sizes. This process is summarised in Table 2 for 4 items.

Table 2 Example of 4 item sort using MSN and indexed location

Items	Index Values	T_1	T_2	T_3	T_4	T_5	Sorted Items
7	0	**1**		**3**		3	4
5	1	**0**			**2**	**1**	5
6	2		**3**	**1**		**2**	6
4	3		**2**		**0**	0	7

One way of sorting items using index values can be implemented as given in Figure 2. Here, counters represent fitness values that are the items to be sorted. Each counter has an associated 2-bit address {00, 01, 10, 11}. The contents of counters are loaded into A and B inputs of the comparator in the order given in Figure 1. The main operation in this implementation is the comparison of fitness values A, and B and exchange of corresponding index values, that are represented as a, b, c, and d. If A is greater than B, the output of the comparator is set to true (T), and then the index values of corresponding counters are swapped. F represents false situation and in this case there is no swap. TR is a dummy register that is used during the swap operation of index values. The content of the first value is saved into TR before it is loaded with the new value. The final piece of hardware, state machine, represents the control circuitry responsible for coordinating the entire process (pair wise loading of counter values to the 'm' bit comparator and manipulating the order of the counters' indexes in the registers accordingly). Thus, the contents of registers from a to d following the execution of the sort algorithm indicates the required ascending ordered results.

Although it is not illustrated in Figure 2 the comparator and four m-bit counters are linked by a suitable bus structure. This requires a pair of 4 to 1 multiplexer for each bit of counters, one for A and one for B input of the comparator. It is also possible to use m-bit tri-state bus in place of multiplexers.

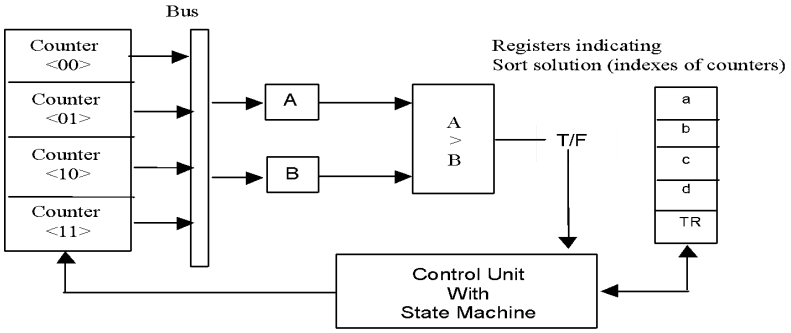


Figure 2 Minimum Sorting Network Hardware

2.1. Parallel Sorter Design

In sorting algorithm, c and d are compared after a and b comparison. The reason for this is the use of a single comparator. By using an additional comparator and a multiplexer within the logic circuit, two individual discrete comparison steps can be performed at the same time instance in parallel. This new sorting algorithm was realized with a new sorter named as Parallel Sorter. With this modification it is possible to have the shortest response time to sort the fitness cases. The processing states for Parallel Sorter can be seen in Figure 3.

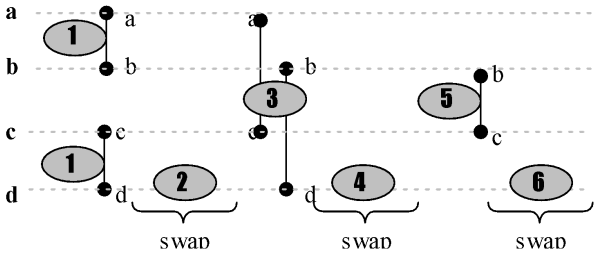


Figure 3 Processing states for Parallel Sorter

As seen in Figure 3 a-b and c-d comparisons (state-1) are performed at the same time in parallel. The comparison results are shown as Comp-X and Comp-Y, respectively. T2, T4 and T6 time instances are used for swap operations according to the comparison results. Swap operation takes place only if the first item is greater than the second. Whether the swap takes place or not, time instances reserved for swap operation are always consumed. Therefore, the total time spent for completing sorting process is always equal to 6 clock cycles. It is also possible to realize a more efficient sorter in terms of time consumption..

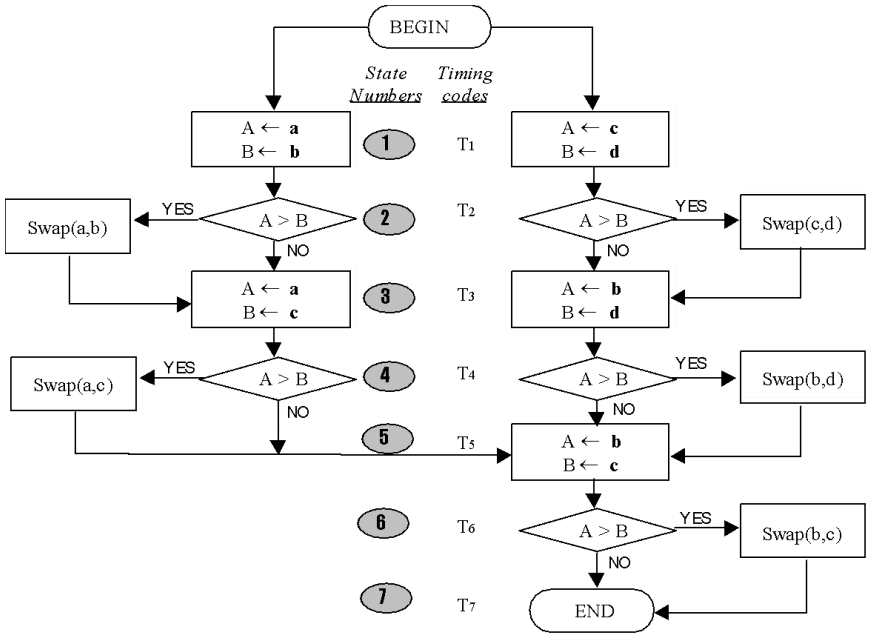


Figure 4 Parallel Sorter

Table 3 Transition table for swap function of parallel sorter

PRESENT STATE	COMP-X INPUT		COMP-Y INPUT		NEXT STATE	NEXT REGISTER VALUES			
	A_x	**B_x**	**A_y**	**B_y**		**a**	**b**	**c**	**d**
1	a	b	c	d	2	B_x	A_x	B_y	A_y
3	a	c	b	d	4	B_x	B_y	A_x	A_y
5			b	c	6		B_y	A_y	

Koza, J.R. (1998a). FPGA98 Evolving Computer Programs using Rapidly Reconfigurable Field-Programable Gate Arrays and Genetic Programming.

Koza, J.R. (1998b). Genetic programming II – automatic discovery of reusable programs. London: The MIT Press

Knuth, D.E. (1973). The art of omputer Programming. Volume 3. Reading, MA: Addison Wesley

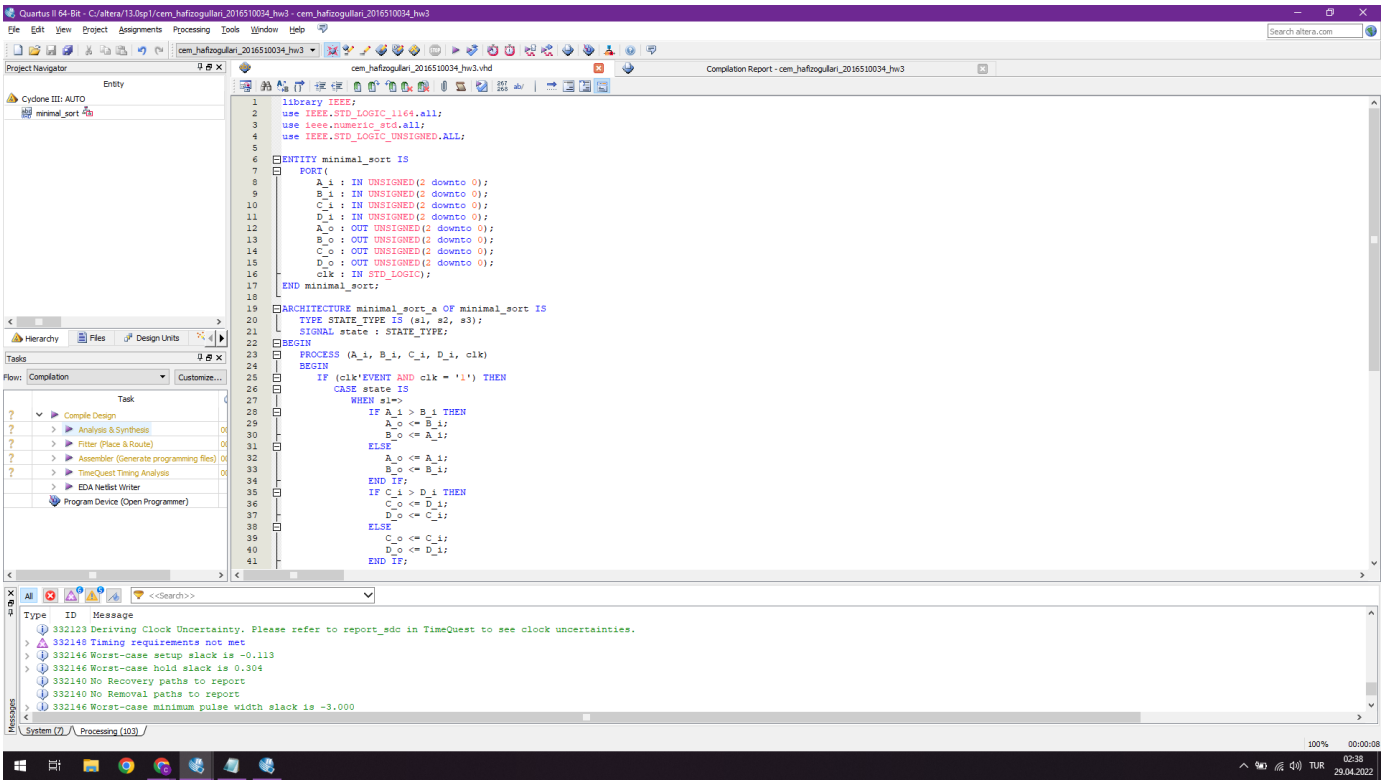
Cem Hafızoğulları

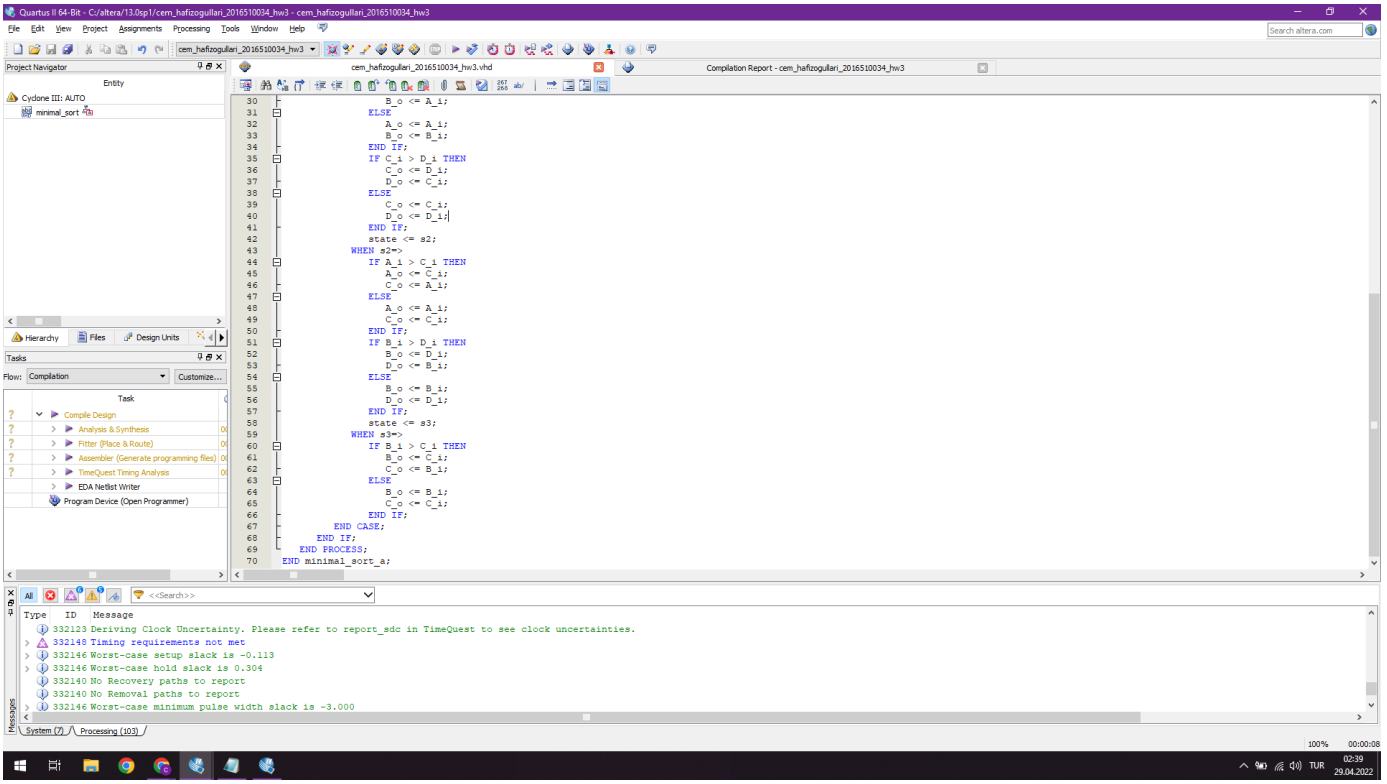
2016510034

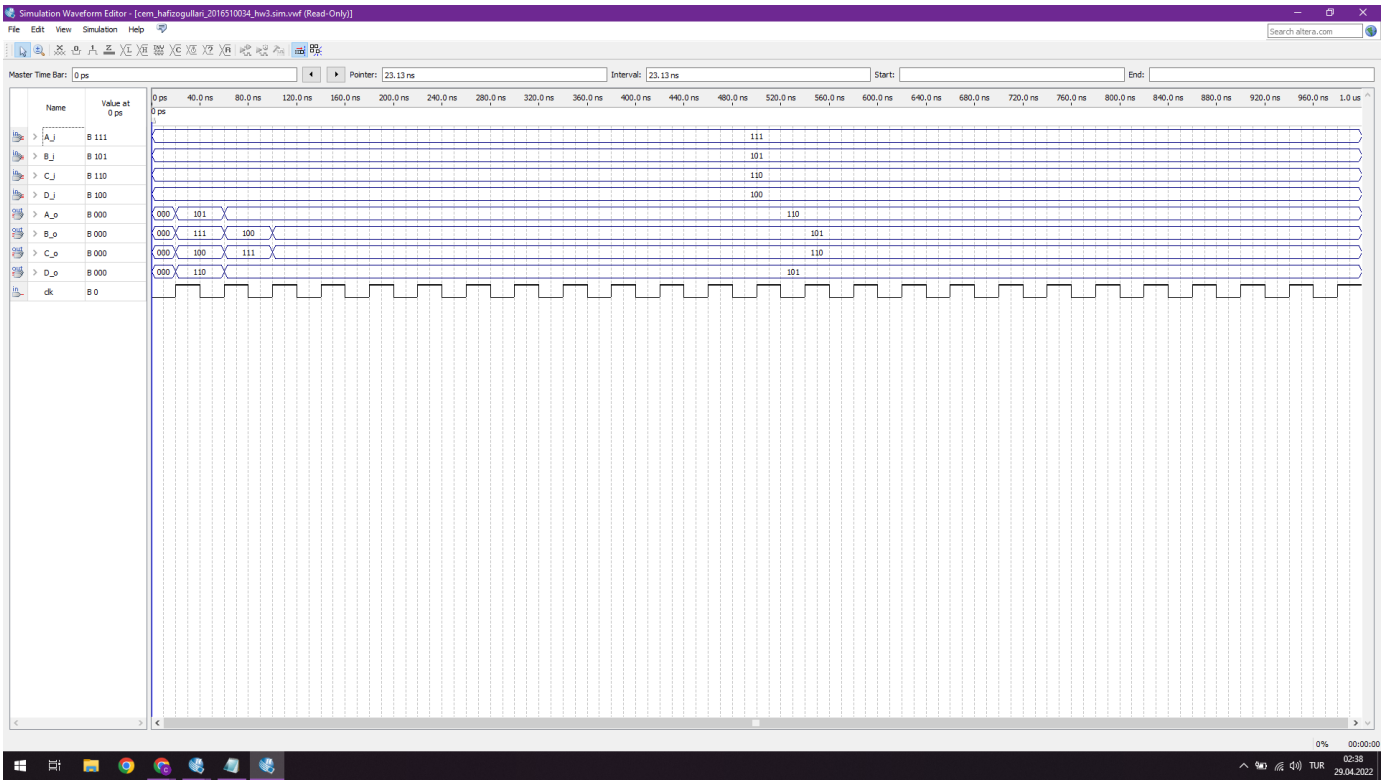
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.numeric_std.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY minimal_sort IS
  PORT(
    A_i : IN UNSIGNED(2 downto 0);
        B_i : IN UNSIGNED(2 downto 0);
        C_i : IN UNSIGNED(2 downto 0);
        D_i : IN UNSIGNED(2 downto 0);
        A_o : OUT UNSIGNED(2 downto 0);
        B_o : OUT UNSIGNED(2 downto 0);
        C_o : OUT UNSIGNED(2 downto 0);
        D_o : OUT UNSIGNED(2 downto 0);
    clk : IN STD_LOGIC);
END minimal_sort;

ARCHITECTURE minimal_sort_a OF minimal_sort IS
  TYPE STATE_TYPE IS (s1, s2, s3);
  SIGNAL state : STATE_TYPE;
BEGIN
  PROCESS (A_i, B_i, C_i, D_i, clk)
```

```
BEGIN
  IF (clk'EVENT AND clk = '1') THEN
    CASE state IS
      WHEN s1=>
        IF A_i > B_i THEN
          A_o <= B_i;
                                    B_o <= A_i;
                                ELSE
                                    A_o <= A_i;
                                    B_o <= B_i;
                                END IF;
        IF C_i > D_i THEN
          C_o <= D_i;
                                    D_o <= C_i;
                                ELSE
                                    C_o <= C_i;
                                    D_o <= D_i;
                                END IF;
                                state <= s2;
      WHEN s2=>
        IF A_i > C_i THEN
          A_o <= C_i;
                                    C_o <= A_i;
                                ELSE
                                    A_o <= A_i;
                                    C_o <= C_i;
                                END IF;
        IF B_i > D_i THEN
          B_o <= D_i;
                                    D_o <= B_i;
```

```
                    ELSE
                        B_o <= B_i;
                        D_o <= D_i;
                    END IF;
                    state <= s3;
        WHEN s3=>
          IF B_i > C_i THEN
            B_o <= C_i;
                        C_o <= B_i;
                    ELSE
                        B_o <= B_i;
                        C_o <= C_i;
                    END IF;
      END CASE;
    END IF;
  END PROCESS;
END minimal_sort_a;
```


Quartus II 64-Bit - C:/altera/13.0sp1/cem_hafizogullari_2016510034_hw3 - cem_hafizogullari_2016510034_hw3
File Edit View Project Assignments Processing Tools Window Help
Project Navigator
Entity
Cyclone III: AUTO
minimal_sort
cem_hafizogullari_2016510034_hw3.vhd
Compilation Report - cem_hafizogullari_2016510034_hw3
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.all;
3 use ieee.numeric_std.all;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 ENTITY minimal_sort IS
7 PORT(
8 A_i : IN UNSIGNED(2 downto 0);
9 B_i : IN UNSIGNED(2 downto 0);
10 C_i : IN UNSIGNED(2 downto 0);
11 D_i : IN UNSIGNED(2 downto 0);
12 A_o : OUT UNSIGNED(2 downto 0);
13 B_o : OUT UNSIGNED(2 downto 0);
14 C_o : OUT UNSIGNED(2 downto 0);
15 D_o : OUT UNSIGNED(2 downto 0);
16 clk : IN STD_LOGIC);
17 END minimal_sort;
18
19 ARCHITECTURE minimal_sort_a OF minimal_sort IS
20 TYPE STATE_TYPE IS (s1, s2, s3);
21 SIGNAL state : STATE_TYPE;
22 BEGIN
23 PROCESS (A_i, B_i, C_i, D_i, clk)
24 BEGIN
25 IF (clk'EVENT AND clk = '1') THEN
26 CASE state IS
27 WHEN s1=>
28 IF A_i > B_i THEN
29 A_o <= B_i;
30 B_o <= A_i;
31 ELSE
32 A_o <= A_i;
33 B_o <= B_i;
34 END IF;
35 IF C_i > D_i THEN
36 C_o <= D_i;
37 D_o <= C_i;
38 ELSE
39 C_o <= C_i;
40 D_o <= D_i;
41 END IF;
Tasks
Flow: Compilation
Task
Compile Design
Analysis & Synthesis
Fitter (Place & Route)
Assembler (Generate programming files)
TimeQuest Timing Analysis
EDA Netlist Writer
Program Device (Open Programmer)
Type ID Message
332123 Deriving Clock Uncertainty. Please refer to report_sdc in TimeQuest to see clock uncertainties.
332148 Timing requirements not met
332146 Worst-case setup slack is -0.113
332146 Worst-case hold slack is 0.304
332140 No Recovery paths to report
332140 No Removal paths to report
332146 Worst-case minimum pulse width slack is -3.000
System (7) Processing (103)
100% 00:00:08
TUR 02:38 29.04.2022


Quartus II 64-Bit - C:/altera/13.0sp1/cem_hafizogullari_2016510034_hw3 - cem_hafizogullari_2016510034_hw3
File Edit View Project Assignments Processing Tools Window Help
Project Navigator
Entity
Cyclone III: AUTO
minimal_sort
cem_hafizogullari_2016510034_hw3.vhd
Compilation Report - cem_hafizogullari_2016510034_hw3
30 B_o <= A_i;
31 ELSE
32 A_o <= A_i;
33 B_o <= B_i;
34 END IF;
35 IF C_i > D_i THEN
36 C_o <= D_i;
37 D_o <= C_i;
38 ELSE
39 C_o <= C_i;
40 D_o <= D_i;
41 END IF;
42 state <= s2;
43 WHEN s2=>
44 IF A_i > C_i THEN
45 A_o <= C_i;
46 C_o <= A_i;
47 ELSE
48 A_o <= A_i;
49 C_o <= C_i;
50 END IF;
51 IF B_i > D_i THEN
52 B_o <= D_i;
53 D_o <= B_i;
54 ELSE
55 B_o <= B_i;
56 D_o <= D_i;
57 END IF;
58 state <= s3;
59 WHEN s3=>
60 IF B_i > C_i THEN
61 B_o <= C_i;
62 C_o <= B_i;
63 ELSE
64 B_o <= B_i;
65 C_o <= C_i;
66 END IF;
67 END CASE;
68 END IF;
69 END PROCESS;
70 END minimal_sort_a;
Tasks
Flow: Compilation
Task
Compile Design
Analysis & Synthesis
Fitter (Place & Route)
Assembler (Generate programming files)
TimeQuest Timing Analysis
EDA Netlist Writer
Program Device (Open Programmer)
Type ID Message
332123 Deriving Clock Uncertainty. Please refer to report_sdc in TimeQuest to see clock uncertainties.
332148 Timing requirements not met
332146 Worst-case setup slack is -0.113
332146 Worst-case hold slack is 0.304
332140 No Recovery paths to report
332140 No Removal paths to report
332146 Worst-case minimum pulse width slack is -3.000
System (7) Processing (103)
100% 00:00:08
TUR 02:39 29.04.2022


Simulation Waveform Editor - [cem_hafizogullari_2016510034_hw3.sim.vwf (Read-Only)]
File Edit View Simulation Help
Search altera.com
Master Time Bar: 0 ps
Pointer: 23.13 ns
Interval: 23.13 ns
Start:
End:
Name
Value at 0 ps
A_i B 111
B_i B 101
C_i B 110
D_i B 100
A_o B 000
B_o B 000
C_o B 000
D_o B 000
clk B 0
0 ps 40.0 ns 80.0 ns 120.0 ns 160.0 ns 200.0 ns 240.0 ns 280.0 ns 320.0 ns 360.0 ns 400.0 ns 440.0 ns 480.0 ns 520.0 ns 560.0 ns 600.0 ns 640.0 ns 680.0 ns 720.0 ns 760.0 ns 800.0 ns 840.0 ns 880.0 ns 920.0 ns 960.0 ns 1.0 us
111
101
110
100
000 101 110
000 111 100 101
000 100 111 110
000 110 101
0%
00:00:00
TUR
02:38
29.04.2022