**ALARM CLOCK PROJECT**

**Write RTL and verify components, integrate and test the Alarm Clock specified.**

**RTL-aclk\_controller**

**module** aclk\_controller**(input** clk**,**reset**,**one\_second**,**alarm\_button**,**time\_button**,**

**input** **[**3**:**0**]**key**,**

**output** load\_new\_c**,**show\_new\_time**,**show\_a**,**load\_new\_a**,**shift**,**reset\_count**);**

**parameter** SHOW\_TIME**=**3'd0**,**

KEY\_STORED**=**3'd1**,**

KEY\_WAITED**=**3'd2**,**

KEY\_ENTRY**=**3'd3**,**

SET\_ALARM\_TIME**=**3'd4**,**

SET\_CURRENT\_TIME**=**3'd5**,**

SHOW\_ALARM**=**3'd6**;**

**reg** **[**2**:**0**]**pstate**,**next\_state**;**

**reg** **[**3**:**0**]**count1**,**count2**;**

**reg** timeout**;**

//reg load\_new\_c\_reg,show\_new\_time\_reg,show\_a\_reg,load\_new\_a\_reg,shift\_reg;

**assign** load\_new\_c **=** **(**pstate**==**SET\_CURRENT\_TIME**)?**1**:**0**;**

**assign** load\_new\_a **=** **(**pstate**==** SET\_ALARM\_TIME**)?**1**:**0**;**

**assign** show\_new\_time**=** **(**pstate**==**KEY\_STORED **||** pstate**==**KEY\_WAITED **||** pstate**==**KEY\_ENTRY **||** pstate**==**SET\_ALARM\_TIME **||** pstate**==**SET\_CURRENT\_TIME **)?**1**:**0**;**

**assign** show\_a**=(**pstate**==**SHOW\_ALARM**)?**1**:**0**;**

**assign** reset\_count**=**0**;**

**assign** shift**=** **(**pstate**==**KEY\_STORED**)?**1**:**0**;**

//SEQUENTIAL

**always** **@(posedge** clk **or** **posedge** reset**)**

**begin**

**if(**reset**==**1**)**

**{**pstate**,**next\_state**}=**0**;**

**else**

pstate**<=**next\_state**;**

**end**

//10 Second TIMER

**always** **@(posedge** one\_second **or** **posedge** reset**)**

**begin**

**if(**reset**==**1**)**

count1**=**0**;**

**else** **if(**pstate**!=**KEY\_WAITED**)**

count1**=**0**;**

**else** **if(**count1**==**9**)**

count1**=**0**;**

**else**

count1**=**count1**+**1**;**

**end**

**always** **@(posedge** one\_second **or** **posedge** reset**)**

**begin**

**if(**reset**==**1**)**

count2**=**0**;**

**else** **if(**pstate**!=**KEY\_ENTRY**)**

count2**=**0**;**

**else** **if(**count2**==**9**)**

count2**=**0**;**

**else**

count2**=**count2**+**1**;**

**end**

**assign** timeout **=** **(**count1**==**9 **||** count2**==**9**)?**0**:**1**;**

//NEXT\_STATE LOGIC

**always** **@(**pstate**,**one\_second**,**alarm\_button**,**time\_button**,**key**,**timeout**)**

**begin**

**case** **(**pstate**)**

SHOW\_TIME**:**

**if(**key**!=**10**)**

next\_state**=**KEY\_STORED**;**

**else** **if(**alarm\_button**==**1**)**

next\_state**=**SHOW\_ALARM**;**

**else**

next\_state**=**SHOW\_TIME**;**

KEY\_STORED**:**

**if(**key**!=**10**)**

next\_state**=**KEY\_WAITED**;**

**else**

next\_state**=**SHOW\_TIME**;**

KEY\_WAITED**:**

**if(**key**==**10**)**

next\_state**=**KEY\_ENTRY**;**

**else** **if(**timeout**==**0**)**

next\_state**=**SHOW\_TIME**;**

**else**

next\_state**=**KEY\_WAITED**;**

KEY\_ENTRY**:**

**if(**key**!=**10**)**

next\_state**=**KEY\_STORED**;**

**else** **if(**alarm\_button**==**1**)**

next\_state**=**SET\_ALARM\_TIME**;**

**else** **if(**time\_button**==**1**)**

next\_state**=**SET\_CURRENT\_TIME**;**

**else** **if(**timeout**==**0**)**

next\_state**=**SHOW\_TIME**;**

**else**

next\_state**=**KEY\_ENTRY**;**

SET\_ALARM\_TIME**:**

next\_state**=**SHOW\_TIME**;**

SET\_CURRENT\_TIME**:**

next\_state**=**SHOW\_TIME**;**

SHOW\_ALARM**:**

**if(**alarm\_button**==**1**)**

next\_state**=**SHOW\_ALARM**;**

**else**

next\_state**=**SHOW\_TIME**;**

**endcase**

**end**

**endmodule**

**TB-aclk\_controller**

**module** tb\_aclk\_controller**();**

**reg** clk**,**reset**,**one\_second**,**alarm\_button**,**time\_button**;**

**reg** **[**3**:**0**]**key**;**

**wire** load\_new\_c**,**show\_new\_time**,**show\_a**,**load\_new\_a**,**shift**,**reset\_count**;**

**wire** **[**3**:**0**]** key\_buffer\_ls\_min**,**

key\_buffer\_ms\_min**,**

key\_buffer\_ls\_hr**,**

key\_buffer\_ms\_hr**;**

aclk\_controller DUT**(.**clk**(**clk**),.**reset**(**reset**),.**one\_second**(**one\_second**),.**alarm\_button**(**alarm\_button**),.**time\_button**(**time\_button**),**

**.**reset\_count**(**reset\_count**),.**key**(**key**),.**load\_new\_c**(**load\_new\_c**),.**show\_new\_time**(**show\_new\_time**),.**show\_a**(**show\_a**),**

**.**load\_new\_a**(**load\_new\_a**),.**shift**(**shift**)** **);**

keyreg DUT2**(**

**.**reset**(**reset**),**

**.**clock**(**clk**),**

**.**shift**(**shift**),**

**.**key**(**key**),**

**.**key\_buffer\_ls\_min**(**key\_buffer\_ls\_min**),**

**.**key\_buffer\_ms\_min**(**key\_buffer\_ms\_min**),**

**.**key\_buffer\_ls\_hr**(**key\_buffer\_ls\_hr**),**

**.**key\_buffer\_ms\_hr**(**key\_buffer\_ms\_hr**)**

**);**

**initial**

clk**=**0**;**

**always** **#**5 clk**=~**clk**;**

**initial**

one\_second**=**0**;**

**always** **#**20 one\_second**=~**one\_second**;**

**task** initialize**;**

**begin**

**{**one\_second**,**alarm\_button**,**time\_button**}=**0**;**

key**=**10**;**

**end**

**endtask**

**initial**

**begin**

**@(negedge** clk**);**

initialize**;**

**@(negedge** clk**);**

reset**=**1**;**

**@(negedge** clk**);**

reset**=**0**;**

//FIRST

**@(negedge** clk**);**

key**=**1**;** //PRESS

//KEY\_STORED

**@(negedge** clk**);**

////KEY\_WAITED

**#**480 //CHECKING KEY\_WAITEED TIMEOUT

**@(negedge** clk**);**

reset**=**1**;**

**@(negedge** clk**);**

reset**=**0**;**

/\*---------------------------------------------------------------------\*/

//FIRST

**@(negedge** clk**);**

key**=**1**;** //PRESS

//KEY\_STORED

**@(negedge** clk**);**

////KEY\_WAITED

**@(negedge** clk**);**

key**=**10**;** //RELEASE

//KEY\_ENTRY

**#**480 //CHECKING KEY ENTRY TIMEOUT

**@(negedge** clk**);**

reset**=**1**;**

**@(negedge** clk**);**

reset**=**0**;**

/\*---------------------------------------------------------------------\*/

//FIRST

**@(negedge** clk**);**

key**=**1**;** //PRESS

//KEY\_STORED

**@(negedge** clk**);**

////KEY\_WAITED

**@(negedge** clk**);**

key**=**10**;** //RELEASE

//KEY\_ENTRY

**@(negedge** clk**);**

key**=**7**;**

//KEY\_STORED

**@(negedge** clk**);**

**@(negedge** clk**);**

key**=**10**;** //RELEASE

//SECOND

**@(negedge** clk**);**

key**=**3**;** //PRESS

//KEY\_STORED

**@(negedge** clk**);**

////KEY\_WAITED

**@(negedge** clk**);**

key**=**10**;** //RELEASE

//KEY\_ENTRY

**@(negedge** clk**);**

key**=**2**;**

//KEYSTORED

**@(negedge** clk**);**

**@(negedge** clk**);**

key**=**10**;** //FINAL RELEASE

**#**50**;**

**@(negedge** clk**);**

time\_button**=**1**;**

**@(negedge** clk**);**

time\_button**=**0**;**

**@(negedge** clk**);**

reset**=**1**;**

**@(negedge** clk**);**

reset**=**0**;**

**#**100**;**

**end**

**endmodule**

**RTL-LCD DRIVER**

**module** lcd\_driver **(**alarm\_time**,**

current\_time**,**

show\_alarm**,**

show\_new\_time**,**

key**,**display\_time**,**

sound\_alarm**);**

//Define input and output ports direction

**input** **[**3**:**0**]** key**;**

**input** **[**3**:**0**]**alarm\_time**;**

**input** **[**3**:**0**]**current\_time**;**

**input** show\_alarm**;**

**input** show\_new\_time**;**

**output** **reg** **[**7**:**0**]**display\_time**;**

**output** **reg** sound\_alarm**;**

//Define the internal signals

**reg** **[**3**:**0**]**display\_value **;**

//Define the Parameter constants to represent LCD numbers

**parameter** ZERO **=** 8'h30**;**

**parameter** ONE **=** 8'h31**;**

**parameter** TWO **=** 8'h32**;**

**parameter** THREE **=** 8'h33**;**

**parameter** FOUR **=** 8'h34**;**

**parameter** FIVE **=** 8'h35**;**

**parameter** SIX **=** 8'h36**;**

**parameter** SEVEN **=** 8'h37**;**

**parameter** EIGHT **=** 8'h38**;**

**parameter** NINE **=** 8'h39**;**

**parameter** ERROR **=** 8'h3A**;**

**assign** sound\_alarm**=(**alarm\_time**==**current\_time**)?**1**:**0**;**

**always** **@** **(**alarm\_time **or** current\_time **or** show\_alarm **or** show\_new\_time **or** key**)**

**begin**

//Displays the key\_time,alarm\_time or current\_time as per the control signals

**if(**show\_alarm**==**1 **&&** show\_new\_time**==**0**)**

display\_value**=**alarm\_time**;**

**else** **if(**show\_alarm**==**0 **&&** show\_new\_time**==**1**)**

display\_value**=**key**;**

**else** **if(**show\_alarm**==**0 **&&** show\_new\_time**==**0**)**

display\_value**=**current\_time**;**

**else**

display\_value**=**current\_time**;**

**end**

//Decoder logic

**always** **@** **(**display\_value**)**

**begin**

// For number stored in display\_value register, load display\_time register with LCD equivalent

**case** **(**display\_value**)**

0**:** display\_time**=**ZERO**;**

1**:** display\_time**=**ONE**;**

2**:** display\_time**=**TWO**;**

3**:** display\_time**=**THREE**;**

4**:** display\_time**=**FOUR**;**

5**:** display\_time**=**FIVE**;**

6**:** display\_time**=**SIX**;**

7**:** display\_time**=**SEVEN**;**

8**:** display\_time**=**EIGHT**;**

9**:** display\_time**=**NINE**;**

**default** **:** display\_time **=** ERROR**;**

**endcase**

**end**

**endmodule**

**TB-LCD DRIVER**

**module** lcd\_driver\_tb**();**

**reg** **[**3**:**0**]** key**;**

**reg** **[**3**:**0**]**alarm\_time**;**

**reg** **[**3**:**0**]**current\_time**;**

**reg** show\_alarm**;**

**reg** show\_new\_time**;**

**wire** **[**7**:**0**]**display\_time**;**

**wire** sound\_alarm**;**

lcd\_driver DUT**(**

**.**alarm\_time**(**alarm\_time**),**

**.**current\_time**(**current\_time**),**

**.**show\_alarm**(**show\_alarm**),**

**.**show\_new\_time**(**show\_new\_time**),**

**.**key**(**key**),**

**.**display\_time**(**display\_time**),**

**.**sound\_alarm**(**sound\_alarm**)** **);**

**initial**

**begin**

**{**key**,**alarm\_time**,**current\_time**,**show\_alarm**,**show\_new\_time**}=**0**;**

**#**5 current\_time**=**4'b0011**;**

**#**5 key**=**4'b0101**;** show\_new\_time**=**1**;**

**#**5 alarm\_time**=**4'b0011**;** show\_alarm**=**1**;**show\_new\_time**=**0**;**

**#**5 show\_alarm**=**0**;**

**end**

**initial**

$monitor**(**"Inputs key=%b,alarm\_time=%b,current\_time=%b,show\_alarm=%b,show\_new=%b --- Outputs display\_time=%b,sound\_a=%b"**,**

key**,**alarm\_time**,**current\_time**,**show\_alarm**,**show\_new\_time**,**display\_time**,**sound\_alarm**);**

**endmodule**

**RTL - LCD DRIVER 4LCDS**

**module** lcd\_driver\_4 **(** alarm\_time\_ms\_hr**,**

alarm\_time\_ls\_hr**,**

alarm\_time\_ms\_min**,**

alarm\_time\_ls\_min**,**

current\_time\_ms\_hr**,**

current\_time\_ls\_hr**,**

current\_time\_ms\_min**,**

current\_time\_ls\_min**,**

key\_ms\_hr**,**

key\_ls\_hr**,**

key\_ms\_min**,**

key\_ls\_min**,**

show\_a**,**

show\_current\_time**,**

display\_ms\_hr**,**

display\_ls\_hr**,**

display\_ms\_min**,**

display\_ls\_min**,**

sound\_a**);**

// Define input and output port directions

**input** **[**3**:**0**]** alarm\_time\_ms\_hr**,**

alarm\_time\_ls\_hr**,**

alarm\_time\_ms\_min**,**

alarm\_time\_ls\_min**,**

current\_time\_ms\_hr**,**

current\_time\_ls\_hr**,**

current\_time\_ms\_min**,**

current\_time\_ls\_min**,**

key\_ms\_hr**,**

key\_ls\_hr**,**

key\_ms\_min**,**

key\_ls\_min**;**

**output** **[**7**:**0**]** display\_ms\_hr**,**

display\_ls\_hr**,**

display\_ms\_min**,**

display\_ls\_min**;**

**input** show\_a**,**show\_current\_time**;**

**output** sound\_a**;**

**wire** sound\_alarm1**,**sound\_alarm2**,**sound\_alarm3**,**sound\_alarm4**;**

// Assert sound\_a when all 4 digits matches

//assign sound\_alarm1=(alarm\_time\_ms\_hr==current\_time\_ms\_hr)?1:0;

//assign sound\_alarm2=(alarm\_time\_ls\_hr==current\_time\_ls\_hr)?1:0;

//assign sound\_alarm3=(alarm\_time\_ms\_min==current\_time\_ms\_min)?1:0;

//assign sound\_alarm4=(alarm\_time\_ls\_min==current\_time\_ls\_min)?1:0;

//assign sound\_a=(sound\_alarm1 && sound\_alarm2 && sound\_alarm3 && sound\_alarm4)?1:0;

//Instantiate lcd\_driver as MS\_HR\_display

lcd\_driver MS\_HR **(.**alarm\_time**(**alarm\_time\_ms\_hr**),**

**.**current\_time**(**current\_time\_ms\_hr**),**

**.**key**(**key\_ms\_hr**),**

**.**show\_alarm**(**show\_a**),**

**.**show\_new\_time**(**show\_current\_time**),**

**.**display\_time**(**display\_ms\_hr**),**

**.**sound\_alarm**(**sound\_alarm1**));**

//Instantiate lcd\_driver as LS\_HR\_display

lcd\_driver LS\_HR **(.**alarm\_time**(**alarm\_time\_ls\_hr**),**

**.**current\_time**(**current\_time\_ls\_hr**),**

**.**key**(**key\_ls\_hr**),**

**.**show\_alarm**(**show\_a**),**

**.**show\_new\_time**(**show\_current\_time**),**

**.**display\_time**(**display\_ls\_hr**),**

**.**sound\_alarm**(**sound\_alarm2**));**

//Instantiate lcd\_driver as MS\_MIN\_display

lcd\_driver MS\_MIN **(.**alarm\_time**(**alarm\_time\_ms\_min**),**

**.**current\_time**(**current\_time\_ms\_min**),**

**.**key**(**key\_ms\_min**),**

**.**show\_alarm**(**show\_a**),**

**.**show\_new\_time**(**show\_current\_time**),**

**.**display\_time**(**display\_ms\_min**),**

**.**sound\_alarm**(**sound\_alarm3**));**

//Instantiate lcd\_driver as LS\_MIN\_display

lcd\_driver LS\_MIN **(.**alarm\_time**(**alarm\_time\_ls\_min**),**

**.**current\_time**(**current\_time\_ls\_min**),**

**.**key**(**key\_ls\_min**),**

**.**show\_alarm**(**show\_a**),**

**.**show\_new\_time**(**show\_current\_time**),**

**.**display\_time**(**display\_ls\_min**),**

**.**sound\_alarm**(**sound\_alarm4**));**

**assign** sound\_a**=(**sound\_alarm1 **&&** sound\_alarm2 **&&** sound\_alarm3 **&&** sound\_alarm4**)?**1**:**0**;**

**endmodule**

**RTL - ACLK-TIMING GENERTAOR**

**module** aclk\_timegen**(**clk**,**reset**,**reset\_count**,**fast\_watch**,**one\_minute**,**one\_second**);**

**input** clk**,**reset**,**fast\_watch**,**reset\_count**;**

**output** one\_minute**,**one\_second**;**

**reg** **[**15**:**0**]**count\_reg**;**

**reg** one\_minute\_reg**,**one\_second\_reg**;**

//one second

**always** **@(posedge** clk **or** **posedge** reset**)**

**begin**

**if(**reset**==**1**)**

**begin**

count\_reg**<=**0**;**

one\_second\_reg**<=**0**;**

**end**

**else** **if(**reset\_count**==**1**)**

**begin**

count\_reg**<=**0**;**

one\_second\_reg**<=**0**;**

**end**

**else** **if(**count\_reg**%**256**==**0 **&&** count\_reg**!=**0**)**

**begin**

one\_second\_reg**<=**1**;**

**if(**count\_reg**!=**15360**)**

count\_reg**<=**count\_reg**+**1**;**

**end**

**else**

**begin**

count\_reg**<=**count\_reg**+**1**;**

one\_second\_reg**<=**0**;**

**end**

**end**

//one minute

**always** **@(posedge** clk **or** **posedge** reset**)**

**begin**

**if(**reset**==**1 **||** reset\_count**==**1**)**

**begin**

count\_reg**<=**0**;**

one\_minute\_reg**<=**0**;**

**end**

**else** **if(**reset\_count**==**1**)**

**begin**

count\_reg**<=**0**;**

one\_minute\_reg**<=**0**;**

**end**

**else** **if(**count\_reg**==**15360**)**

**begin**

one\_minute\_reg**<=**1**;**

count\_reg**<=**0**;**

**end**

**else**

one\_minute\_reg**<=**0**;**

**end**

**assign** one\_second**=**one\_second\_reg**;**

**assign** one\_minute**=(**fast\_watch**==**1**)?**one\_second\_reg**:**one\_minute\_reg**;**

**endmodule**

**TB - ACLK-TIMING GENERTAOR**

**module** tb\_aclk\_timegen**();**

**reg** clk**,**reset**,**fast\_watch**,**reset\_count**;**

**wire** one\_minute**,**one\_second**;**

**initial**

clk**=**0**;**

**always** **#**5 clk**=~**clk**;**

aclk\_timegen DUT**(.**clk**(**clk**),.**reset**(**reset**),.**reset\_count**(**reset\_count**),.**fast\_watch**(**fast\_watch**),.**one\_minute**(**one\_minute**),.**one\_second**(**one\_second**));**

**task** initialize**;**

**{**reset**,**fast\_watch**,**reset\_count**}=**0**;**

**endtask**

**initial**

**begin**

initialize**;**

**@(negedge** clk**);**

reset**=**1**;**

**@(negedge** clk**);**

reset**=**0**;**

**#**16000**;**

/\*

@(negedge clk);

fast\_watch=1;

#1000;

@(negedge clk);

fast\_watch=0;

#100;

\*/

**end**

**endmodule;**

**RTL – KEY REG**

**module** keyreg**(**reset**,**

clock**,**

shift**,**

key**,**

key\_buffer\_ls\_min**,**

key\_buffer\_ms\_min**,**

key\_buffer\_ls\_hr**,**

key\_buffer\_ms\_hr**);**

// Define input and output port direction

**input** reset**,**clock**,**shift**;**

**input** **[**3**:**0**]**key**;**

**output** **[**3**:**0**]** key\_buffer\_ls\_min**,**

key\_buffer\_ms\_min**,**

key\_buffer\_ls\_hr**,**

key\_buffer\_ms\_hr**;**

**reg** **[**3**:**0**]** key\_buffer\_ls\_min\_reg**,**

key\_buffer\_ms\_min\_reg**,**

key\_buffer\_ls\_hr\_reg**,**

key\_buffer\_ms\_hr\_reg**;**

**always** **@(posedge** clock **or** **posedge** reset**)**

**begin**

// For asynchronous reset, reset the key\_buffer output register to 1'b0

**if(**reset**==**1**)**

**begin**

key\_buffer\_ls\_min\_reg**<=**0**;**

key\_buffer\_ms\_min\_reg**<=**0**;**

key\_buffer\_ls\_hr\_reg**<=**0**;**

key\_buffer\_ms\_hr\_reg**<=**0**;**

**end**

**else** **if(**shift**==**1 **&&** key**!=**10**)**

**begin**

key\_buffer\_ls\_min\_reg**<=**key**;**

key\_buffer\_ms\_min\_reg**<=**key\_buffer\_ls\_min\_reg**;**

key\_buffer\_ls\_hr\_reg**<=**key\_buffer\_ms\_min\_reg**;**

key\_buffer\_ms\_hr\_reg**<=**key\_buffer\_ls\_hr\_reg**;**

**end**

// Else if there is a shift, perform left shift from LS\_MIN to MS\_HR

**end**

**assign** key\_buffer\_ls\_min**=**key\_buffer\_ls\_min\_reg**;**

**assign** key\_buffer\_ms\_min**=**key\_buffer\_ms\_min\_reg**;**

**assign** key\_buffer\_ls\_hr**=**key\_buffer\_ls\_hr\_reg**;**

**assign** key\_buffer\_ms\_hr**=**key\_buffer\_ms\_hr\_reg**;**

**endmodule**

**RTL - COUNTER**

**module** counter **(**clk**,**

reset**,**

one\_minute**,**

load\_new\_c**,**

new\_current\_time\_ms\_hr**,**

new\_current\_time\_ms\_min**,**

new\_current\_time\_ls\_hr**,**

new\_current\_time\_ls\_min**,**

current\_time\_ms\_hr**,**

current\_time\_ms\_min**,**

current\_time\_ls\_hr**,**

current\_time\_ls\_min**);**

// Define input and output port directions

**input** clk**,**reset**,**one\_minute**,**load\_new\_c**;**

**input** **[**3**:**0**]** new\_current\_time\_ms\_hr**,**

new\_current\_time\_ms\_min**,**

new\_current\_time\_ls\_hr**,**

new\_current\_time\_ls\_min**;**

**output** **[**3**:**0**]** current\_time\_ms\_hr**,**

current\_time\_ms\_min**,**

current\_time\_ls\_hr**,**

current\_time\_ls\_min**;**

// Define register to store current time

**reg** **[**3**:**0**]** current\_time\_ms\_hr\_reg**,**

current\_time\_ms\_min\_reg**,**

current\_time\_ls\_hr\_reg**,**

current\_time\_ls\_min\_reg**;**

// Lodable Binary up synchronous Counter logic /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Write an always block with asynchronous reset

**always@(** **posedge** clk **or** **posedge** reset**)**

**begin**

// Check for reset signal and upon reset load the current\_time register with default value (1'b0)

**if(**reset**==**1**)**

**{**current\_time\_ms\_hr\_reg**,**

current\_time\_ms\_min\_reg**,**

current\_time\_ls\_hr\_reg**,**

current\_time\_ls\_min\_reg**}=**0**;**

// Else if there is no reset, then check for load\_new\_c signal and load new\_current\_time to current\_time register

**else** **if(**load\_new\_c**==**1**)**

**begin**

current\_time\_ms\_hr\_reg**<=**new\_current\_time\_ms\_hr**;**

current\_time\_ms\_min\_reg**<=**new\_current\_time\_ms\_min**;**

current\_time\_ls\_hr\_reg**<=**new\_current\_time\_ls\_hr**;**

current\_time\_ls\_min\_reg**<=**new\_current\_time\_ls\_min**;**

**end** // 0 0 0 9 -> 00:10

// Else if there is no load\_new\_c signal, then check for one\_minute signal and implement the counting algorithm

**else** **if(**one\_minute**==**1**)**

**begin**

**if(**current\_time\_ms\_hr\_reg**==**2 **&&** current\_time\_ls\_hr\_reg**==**3 **&&**

current\_time\_ms\_min\_reg**==**5 **&&** current\_time\_ls\_min\_reg**==**9**)**

**begin**

current\_time\_ms\_hr\_reg**<=**0**;**

current\_time\_ls\_hr\_reg**<=**0**;**

current\_time\_ms\_min\_reg**<=**0**;**

current\_time\_ls\_min\_reg**<=**0**;**

**end**

**else** **if(**current\_time\_ms\_min\_reg**==**5 **&&** current\_time\_ls\_min\_reg**==**9**)**

**begin**

**if(**current\_time\_ls\_hr\_reg**==**9**)**

**begin**

current\_time\_ms\_hr\_reg**<=**current\_time\_ms\_hr\_reg**+**1**;**

current\_time\_ls\_hr\_reg**<=**0**;**

current\_time\_ms\_min\_reg**<=**0**;**

current\_time\_ls\_min\_reg**<=**0**;**

**end**

**else**

**begin**

current\_time\_ls\_hr\_reg**<=**current\_time\_ls\_hr\_reg**+**1**;**

current\_time\_ms\_min\_reg**<=**0**;**

current\_time\_ls\_min\_reg**<=**0**;**

**end**

**end**

**else** **if(**current\_time\_ls\_min\_reg**==**9**)**

**begin**

current\_time\_ms\_min\_reg**<=**current\_time\_ms\_min\_reg**+**1**;**

current\_time\_ls\_min\_reg**<=**0**;**

**end**

**else**

current\_time\_ls\_min\_reg**<=**current\_time\_ls\_min\_reg**+**1**;**

**end**

// Check for the corner case

// If the current\_time is 23:59, then the next current\_time will be 00:00

// Else check if the current\_time is 09:59, then the next current\_time will be 10:00

// Else check if the time represented by minutes is 59, Increment the LS\_HR by 1 and set MS\_MIN and LS\_MIN to 1'b0

// Else check if the LS\_MIN is equal to 9, Increment the MS\_MIN by 1 and set MS\_MIN to 1'b0

// Else just increment the LS\_MIN by 1

**end**

**assign** current\_time\_ms\_hr**=**current\_time\_ms\_hr\_reg**;**

**assign** current\_time\_ms\_min**=**current\_time\_ms\_min\_reg**;**

**assign** current\_time\_ls\_hr**=**current\_time\_ls\_hr\_reg**;**

**assign** current\_time\_ls\_min**=**current\_time\_ls\_min\_reg**;**

**endmodule**

**TB – COUNTER**

**module** tb\_counter**();**

**reg** clk**,**reset**,**one\_minute**,**load\_new\_c**;**

**reg** **[**3**:**0**]** new\_current\_time\_ms\_hr**,**

new\_current\_time\_ms\_min**,**

new\_current\_time\_ls\_hr**,**

new\_current\_time\_ls\_min**;**

**wire** **[**3**:**0**]** current\_time\_ms\_hr**,**

current\_time\_ms\_min**,**

current\_time\_ls\_hr**,**

current\_time\_ls\_min**;**

**initial**

clk**=**0**;**

**always** **#**5 clk**=~**clk**;**

counter DUT**(** **.**clk**(**clk**),**

**.**reset**(**reset**),**

**.**one\_minute**(**one\_minute**),**

**.**load\_new\_c**(**load\_new\_c**),**

**.**new\_current\_time\_ms\_hr**(**new\_current\_time\_ms\_hr**),**

**.**new\_current\_time\_ms\_min**(**new\_current\_time\_ms\_min**),**

**.**new\_current\_time\_ls\_hr**(**new\_current\_time\_ls\_hr**),**

**.**new\_current\_time\_ls\_min**(**new\_current\_time\_ls\_min**),**

**.**current\_time\_ms\_hr**(**current\_time\_ms\_hr**),**

**.**current\_time\_ms\_min**(**current\_time\_ms\_min**),**

**.**current\_time\_ls\_hr**(**current\_time\_ls\_hr**),**

**.**current\_time\_ls\_min**(**current\_time\_ls\_min**)**

**);**

**task** initialize**();**

**begin**

**{**reset**,**one\_minute**,**load\_new\_c**}=**0**;**

**{**new\_current\_time\_ms\_hr**,**

new\_current\_time\_ms\_min**,**

new\_current\_time\_ls\_hr**,**

new\_current\_time\_ls\_min**}=**0**;**

**end**

**endtask**

**initial**

**begin**

initialize**;**

**@(negedge** clk**);**

reset**=**1**;**

**@(negedge** clk**);**

reset**=**0**;**load\_new\_c**=**1**;**one\_minute**=**1**;**

new\_current\_time\_ms\_hr**=**2**;**

new\_current\_time\_ls\_hr**=**3**;**

new\_current\_time\_ms\_min**=**4**;**

new\_current\_time\_ls\_min**=**5**;**

**@(negedge** clk**);**

load\_new\_c**=**0**;**

**#**10000**;**

**end**

**endmodule**

**RTL – ALARM REG**

**module** alarm\_reg **(**new\_alarm\_ms\_hr**,**

new\_alarm\_ls\_hr**,**

new\_alarm\_ms\_min**,**

new\_alarm\_ls\_min**,**

load\_new\_alarm**,**

clock**,**

reset**,**

alarm\_time\_ms\_hr**,**

alarm\_time\_ls\_hr**,**

alarm\_time\_ms\_min**,**

alarm\_time\_ls\_min **);**

// Define input and output port directions

**input** **[**3**:**0**]**new\_alarm\_ms\_hr**,**

new\_alarm\_ls\_hr**,**

new\_alarm\_ms\_min**,**

new\_alarm\_ls\_min**;**

**input** load\_new\_alarm**,**clock**,**reset**;**

**output** **reg** **[**3**:**0**]**alarm\_time\_ms\_hr**,**

alarm\_time\_ls\_hr**,**

alarm\_time\_ms\_min**,**

alarm\_time\_ls\_min**;**

**always** **@** **(posedge** clock **or** **posedge** reset**)**

**begin**

// Upon reset, store reset value(1'b0) to the alarm\_time registers

**if(**reset**==**1**)**

**begin**

**{**alarm\_time\_ms\_hr**,**

alarm\_time\_ls\_hr**,**

alarm\_time\_ms\_min**,**

alarm\_time\_ls\_min**}=**0**;**

**end**

// Else if no reset, check for load\_new\_alarm signal and load new\_alarm time to alarm\_time registers

**else** **if(**load\_new\_alarm**==**1**)**

**begin**

alarm\_time\_ms\_hr**<=**new\_alarm\_ms\_hr**;**

alarm\_time\_ls\_hr**<=**new\_alarm\_ls\_hr**;**

alarm\_time\_ms\_min**<=**new\_alarm\_ms\_min**;**

alarm\_time\_ls\_min**<=**new\_alarm\_ls\_min**;**

**end**

**end**

**endmodule**

**TB - ALARM REG**

**module** alarm\_reg\_tb **();**

// Define input and output port directions

**reg** **[**3**:**0**]**new\_alarm\_ms\_hr**,**

new\_alarm\_ls\_hr**,**

new\_alarm\_ms\_min**,**

new\_alarm\_ls\_min**;**

**reg** load\_new\_alarm**,**clock**,**reset**;**

**wire** **[**3**:**0**]**alarm\_time\_ms\_hr**,**

alarm\_time\_ls\_hr**,**

alarm\_time\_ms\_min**,**

alarm\_time\_ls\_min**;**

**initial**

clock**=**0**;**

**always** **#**5 clock**=~**clock**;**

alarm\_reg DUT**(.**new\_alarm\_ms\_hr**(**new\_alarm\_ms\_hr**),**

**.**new\_alarm\_ls\_hr**(**new\_alarm\_ls\_hr**),**

**.**new\_alarm\_ms\_min**(**new\_alarm\_ms\_min**),**

**.**new\_alarm\_ls\_min**(**new\_alarm\_ls\_min**),**

**.**load\_new\_alarm**(**load\_new\_alarm**),**

**.**clock**(**clock**),**

**.**reset**(**reset**),**

**.**alarm\_time\_ms\_hr**(**alarm\_time\_ms\_hr**),**

**.**alarm\_time\_ls\_hr**(**alarm\_time\_ls\_hr**),**

**.**alarm\_time\_ms\_min**(**alarm\_time\_ms\_min**),**

**.**alarm\_time\_ls\_min**(**alarm\_time\_ls\_min**));**

**initial**

**begin**

**@(negedge** clock**);**

load\_new\_alarm**=**0**;**reset**=**1**;**

**@(negedge** clock**);**

load\_new\_alarm**=**1**;**reset**=**0**;**

new\_alarm\_ms\_hr**=**4'b0000**;**

new\_alarm\_ls\_hr**=**4'b0101**;**

new\_alarm\_ms\_min**=**4'b0011**;**

new\_alarm\_ls\_min**=**4'b0010**;**

**@(negedge** clock**);**

load\_new\_alarm**=**0**;**reset**=**0**;**

**@(negedge** clock**);**

**end**

**initial**

$monitor**(**"TIME IS %d%d:%d%d HH:MM"**,**new\_alarm\_ms\_hr**,**new\_alarm\_ls\_hr**,**new\_alarm\_ms\_min**,**new\_alarm\_ls\_min**);**

**endmodule**

**RTL – ALARM CLOCK TOP MODULE**

**module** alarm\_clock\_top**(**clock**,**

key**,**

reset**,**

time\_button**,**

alarm\_button**,**

fast\_watch**,**

ms\_hour**,**

ls\_hour**,**

ms\_minute**,**

ls\_minute**,**

alarm\_sound**);**

// Define port directions for the signals

**input** clock**,**reset**,**time\_button**,**alarm\_button**,**fast\_watch**;**

**input** **[**3**:**0**]**key**;**

**output** alarm\_sound**;**

**output** **[**7**:**0**]** ms\_hour**,**

ls\_hour**,**

ms\_minute**,**

ls\_minute**;**

//Define the Interconnecting internal wires

//TIMING GENERTOR

**wire** one\_minute**,**one\_second**,**reset\_count**;**

//ALARM CONTROLLER

**wire** load\_new\_c**,**show\_new\_time**,**show\_a**,**load\_new\_a**,**shift**;**

//KEY REG

**wire** **[**3**:**0**]** key\_buffer\_ls\_min**,**

key\_buffer\_ms\_min**,**

key\_buffer\_ls\_hr**,**

key\_buffer\_ms\_hr**;**

//ALARM,CURRENT TIME REGS

**wire** **[**3**:**0**]** alarm\_time\_ms\_hr**,**

alarm\_time\_ls\_hr**,**

alarm\_time\_ms\_min**,**

alarm\_time\_ls\_min**,**

current\_time\_ms\_hr**,**

current\_time\_ls\_hr**,**

current\_time\_ms\_min**,**

current\_time\_ls\_min**;**

//Instantiate lower sub-modules. Use interconnect(Internal) signals for connecting the sub modules

// Instantiate the timing generator module

aclk\_timegen TG**(.**clk**(**clock**),.**reset**(**reset**),.**reset\_count**(**reset\_count**),.**fast\_watch**(**fast\_watch**),.**one\_minute**(**one\_minute**),.**one\_second**(**one\_second**));**

//FSM

aclk\_controller CTRL**(.**clk**(**clock**),.**reset**(**reset**),.**one\_second**(**one\_second**),.**alarm\_button**(**alarm\_button**),.**time\_button**(**time\_button**),**

**.**key**(**key**),.**load\_new\_c**(**load\_new\_c**),.**show\_new\_time**(**show\_new\_time**),.**show\_a**(**show\_a**),.**load\_new\_a**(**load\_new\_a**),**

**.**shift**(**shift**),.**reset\_count**(**reset\_count**)** **);**

// Instantiate the counter module

counter CNT**(** **.**clk**(**clock**),**

**.**reset**(**reset**),**

**.**one\_minute**(**one\_minute**),**

**.**load\_new\_c**(**load\_new\_c**),**

**.**new\_current\_time\_ms\_hr**(**key\_buffer\_ms\_hr**),**

**.**new\_current\_time\_ms\_min**(**key\_buffer\_ms\_min**),**

**.**new\_current\_time\_ls\_hr**(**key\_buffer\_ls\_hr**),**

**.**new\_current\_time\_ls\_min**(**key\_buffer\_ls\_min**),**

**.**current\_time\_ms\_hr**(**current\_time\_ms\_hr**),**

**.**current\_time\_ms\_min**(**current\_time\_ms\_min**),**

**.**current\_time\_ls\_hr**(**current\_time\_ls\_hr**),**

**.**current\_time\_ls\_min**(**current\_time\_ls\_min**)** **);**

// Instantiate the key register module

keyreg KEYREG**(.**reset**(**reset**),**

**.**clock**(**clock**),**

**.**shift**(**shift**),**

**.**key**(**key**),**

**.**key\_buffer\_ls\_min**(**key\_buffer\_ls\_min**),**

**.**key\_buffer\_ms\_min**(**key\_buffer\_ms\_min**),**

**.**key\_buffer\_ls\_hr**(**key\_buffer\_ls\_hr**),**

**.**key\_buffer\_ms\_hr**(**key\_buffer\_ms\_hr**)** **);**

// Instantiate the alarm register module

alarm\_reg A\_REG**(.**new\_alarm\_ms\_hr**(**key\_buffer\_ms\_hr**),**

**.**new\_alarm\_ls\_hr**(**key\_buffer\_ls\_hr**),**

**.**new\_alarm\_ms\_min**(**key\_buffer\_ms\_min**),**

**.**new\_alarm\_ls\_min**(**key\_buffer\_ls\_min**),**

**.**load\_new\_alarm**(**load\_new\_a**),**

**.**clock**(**clock**),**

**.**reset**(**reset**),**

**.**alarm\_time\_ms\_hr**(**alarm\_time\_ms\_hr**),**

**.**alarm\_time\_ls\_hr**(**alarm\_time\_ls\_hr**),**

**.**alarm\_time\_ms\_min**(**alarm\_time\_ms\_min**),**

**.**alarm\_time\_ls\_min**(**alarm\_time\_ls\_min**)** **);**

// Instantiate the lcd\_driver\_4 module

lcd\_driver\_4 LCD**(** **.**alarm\_time\_ms\_hr**(**alarm\_time\_ms\_hr**),**

**.**alarm\_time\_ls\_hr**(**alarm\_time\_ls\_hr**),**

**.**alarm\_time\_ms\_min**(**alarm\_time\_ms\_min**),**

**.**alarm\_time\_ls\_min**(**alarm\_time\_ls\_min**),**

**.**current\_time\_ms\_hr**(**current\_time\_ms\_hr**),**

**.**current\_time\_ls\_hr**(**current\_time\_ls\_hr**),**

**.**current\_time\_ms\_min**(**current\_time\_ms\_min**),**

**.**current\_time\_ls\_min**(**current\_time\_ls\_min**),**

**.**key\_ms\_hr**(**key\_buffer\_ms\_hr**),**

**.**key\_ls\_hr**(**key\_buffer\_ls\_hr**),**

**.**key\_ms\_min**(**key\_buffer\_ms\_min**),**

**.**key\_ls\_min**(**key\_buffer\_ls\_min**),**

**.**show\_a**(**show\_a**),**

**.**show\_current\_time**(**show\_new\_time**),**

**.**display\_ms\_hr**(**ms\_hour**),**

**.**display\_ls\_hr**(**ls\_hour**),**

**.**display\_ms\_min**(**ms\_minute**),**

**.**display\_ls\_min**(**ls\_minute**),**

**.**sound\_a**(**alarm\_sound**)**

**);**

**endmodule**

**TB – ALARM CLOCK TOP MODULE**

`timescale 1ms**/**1ns

**module** tb\_alarm\_clock**();**

**reg** clk**,**

reset**,**

fast\_watch**,**

alarm\_button**,**

time\_button**;**

**reg** **[**3**:**0**]** key**;**

**wire** **[**7**:**0**]** display\_ms\_hr**,**

display\_ms\_min**,**

display\_ls\_hr**,**

display\_ls\_min**;**

**wire** sound\_alarm**;**

**parameter** cycle **=** 3.90625**;**

alarm\_clock\_top DUV**(.**clock**(**clk**),**

**.**reset**(**reset**),**

**.**fast\_watch**(**fast\_watch**),**

**.**alarm\_button**(**alarm\_button**),**

**.**time\_button**(**time\_button**),**

**.**key**(**key**),**

**.**alarm\_sound**(**sound\_alarm**),**

**.**ms\_hour**(**display\_ms\_hr**),**

**.**ls\_hour**(**display\_ls\_hr**),**

**.**ms\_minute**(**display\_ms\_min**),**

**.**ls\_minute**(**display\_ls\_min**));**

**task** initialize**;**

**begin**

**{**alarm\_button**,**time\_button**,**key**,**fast\_watch**}=**0**;**

**end**

**endtask**

//Clock generation logic

**initial**

**begin**

clk **=** 1'b0**;**

**forever**

**#(**cycle**/**2**)** clk **=** **~**clk**;**

**end**

//

//Stimulus logic

**initial**

**begin**

initialize**;**

//Hard reset the design

reset **=** 1**;**

**#**10**;**

reset **=** 0**;**

//Set fastwatch to 1 to make counting faster

fast\_watch **=** 1**;**

//Set key time to current time :11:23

key **=** 1**;**

**repeat(**3**)**

**@(negedge** clk**);**

key **=** 10**;**

**@(negedge** clk**);**

key **=** 1**;**

**repeat(**3**)**

**@(negedge** clk**);**

key **=** 10**;**

**@(negedge** clk**);**

key **=** 2**;**

**repeat(**3**)**

**@(negedge** clk**);**

key **=** 10**;**

**@(negedge** clk**);**

key **=** 3**;**

**repeat(**3**)**

**@(negedge** clk**);**

key **=** 10**;**

**@(negedge** clk**);**

time\_button **=** 1**;**

**@(negedge** clk**);**

time\_button **=** 0**;**

//Set key time to alarm time :11:30

key **=** 1**;**

**repeat(**3**)**

**@(negedge** clk**);**

key **=** 10**;**

**@(negedge** clk**);**

key **=** 1**;**

**repeat(**3**)**

**@(negedge** clk**);**

key **=** 10**;**

**@(negedge** clk**);**

key **=** 3**;**

**repeat(**3**)**

**@(negedge** clk**);**

key **=** 10**;**

**@(negedge** clk**);**

key **=** 0**;**

**repeat(**3**)**

**@(negedge** clk**);**

key **=** 10**;**

**@(negedge** clk**);**

alarm\_button **=** 1**;**

**@(negedge** clk**);**

alarm\_button **=** 0**;**

**#(**7**\***256**\***2**);**//7 -> 7minutes ->7seconds->7 \*256 clock cycles ->7\*256\*2(Time period of clock)

//Time out for Alarm clock

//key = 7;

**repeat(**4**\***2564**)** //Wait for minimum 10second pulses i.e (10\*256) clock cycles

**@(negedge** clk**);**

$finish**;**

**end**

**initial**

$monitor**(**$time**,**"\-ns\t MAVEN SILICON : \tDISPLAY\_MS\_HR =%H >>> DISPLAY\_LS\_HR =%H>>> DISPLAY\_MS\_MIN =%H>>> DISPLAY\_LS\_MIN=%H"**,**display\_ms\_hr**[**3**:**0**],**display\_ls\_hr**[**3**:**0**],**display\_ms\_min**[**3**:**0**],**display\_ls\_min**[**3**:**0**]);**

**endmodule**

**Terminal Output**

**#** 0**-**ns MAVEN SILICON **:** DISPLAY\_MS\_HR **=**0 **>>>** DISPLAY\_LS\_HR **=**0**>>>** DISPLAY\_MS\_MIN **=**0**>>>** DISPLAY\_LS\_MIN**=**0

**#** 18**-**ns MAVEN SILICON **:** DISPLAY\_MS\_HR **=**0 **>>>** DISPLAY\_LS\_HR **=**0**>>>** DISPLAY\_MS\_MIN **=**0**>>>** DISPLAY\_LS\_MIN**=**1

**#** 29**-**ns MAVEN SILICON **:** DISPLAY\_MS\_HR **=**0 **>>>** DISPLAY\_LS\_HR **=**0**>>>** DISPLAY\_MS\_MIN **=**1**>>>** DISPLAY\_LS\_MIN**=**1

**#** 45**-**ns MAVEN SILICON **:** DISPLAY\_MS\_HR **=**0 **>>>** DISPLAY\_LS\_HR **=**1**>>>** DISPLAY\_MS\_MIN **=**1**>>>** DISPLAY\_LS\_MIN**=**2

**#** 61**-**ns MAVEN SILICON **:** DISPLAY\_MS\_HR **=**1 **>>>** DISPLAY\_LS\_HR **=**1**>>>** DISPLAY\_MS\_MIN **=**2**>>>** DISPLAY\_LS\_MIN**=**3

**#** 84**-**ns MAVEN SILICON **:** DISPLAY\_MS\_HR **=**1 **>>>** DISPLAY\_LS\_HR **=**2**>>>** DISPLAY\_MS\_MIN **=**3**>>>** DISPLAY\_LS\_MIN**=**1

**#** 96**-**ns MAVEN SILICON **:** DISPLAY\_MS\_HR **=**2 **>>>** DISPLAY\_LS\_HR **=**3**>>>** DISPLAY\_MS\_MIN **=**1**>>>** DISPLAY\_LS\_MIN**=**1

**#** 111**-**ns MAVEN SILICON **:** DISPLAY\_MS\_HR **=**3 **>>>** DISPLAY\_LS\_HR **=**1**>>>** DISPLAY\_MS\_MIN **=**1**>>>** DISPLAY\_LS\_MIN**=**3

**#** 127**-**ns MAVEN SILICON **:** DISPLAY\_MS\_HR **=**1 **>>>** DISPLAY\_LS\_HR **=**1**>>>** DISPLAY\_MS\_MIN **=**3**>>>** DISPLAY\_LS\_MIN**=**0

**#** 143**-**ns MAVEN SILICON **:** DISPLAY\_MS\_HR **=**1 **>>>** DISPLAY\_LS\_HR **=**1**>>>** DISPLAY\_MS\_MIN **=**2**>>>** DISPLAY\_LS\_MIN**=**3

run

**#** 1018**-**ns MAVEN SILICON **:** DISPLAY\_MS\_HR **=**1 **>>>** DISPLAY\_LS\_HR **=**1**>>>** DISPLAY\_MS\_MIN **=**2**>>>** DISPLAY\_LS\_MIN**=**4

run

**#** 2018**-**ns MAVEN SILICON **:** DISPLAY\_MS\_HR **=**1 **>>>** DISPLAY\_LS\_HR **=**1**>>>** DISPLAY\_MS\_MIN **=**2**>>>** DISPLAY\_LS\_MIN**=**5

run

**#** 3018**-**ns MAVEN SILICON **:** DISPLAY\_MS\_HR **=**1 **>>>** DISPLAY\_LS\_HR **=**1**>>>** DISPLAY\_MS\_MIN **=**2**>>>** DISPLAY\_LS\_MIN**=**6

run

**#** 4018**-**ns MAVEN SILICON **:** DISPLAY\_MS\_HR **=**1 **>>>** DISPLAY\_LS\_HR **=**1**>>>** DISPLAY\_MS\_MIN **=**2**>>>** DISPLAY\_LS\_MIN**=**7

run

**#** 5018**-**ns MAVEN SILICON **:** DISPLAY\_MS\_HR **=**1 **>>>** DISPLAY\_LS\_HR **=**1**>>>** DISPLAY\_MS\_MIN **=**2**>>>** DISPLAY\_LS\_MIN**=**8

run

**#** 6018**-**ns MAVEN SILICON **:** DISPLAY\_MS\_HR **=**1 **>>>** DISPLAY\_LS\_HR **=**1**>>>** DISPLAY\_MS\_MIN **=**2**>>>** DISPLAY\_LS\_MIN**=**9

run

**#** 7018**-**ns MAVEN SILICON **:** DISPLAY\_MS\_HR **=**1 **>>>** DISPLAY\_LS\_HR **=**1**>>>** DISPLAY\_MS\_MIN **=**3**>>>** DISPLAY\_LS\_MIN**=**0

run

**#** 8018**-**ns MAVEN SILICON **:** DISPLAY\_MS\_HR **=**1 **>>>** DISPLAY\_LS\_HR **=**1**>>>** DISPLAY\_MS\_MIN **=**3**>>>** DISPLAY\_LS\_MIN**=**1

run

**#** 9018**-**ns MAVEN SILICON **:** DISPLAY\_MS\_HR **=**1 **>>>** DISPLAY\_LS\_HR **=**1**>>>** DISPLAY\_MS\_MIN **=**3**>>>** DISPLAY\_LS\_MIN**=**2

run

**#** 10018**-**ns MAVEN SILICON **:** DISPLAY\_MS\_HR **=**1 **>>>** DISPLAY\_LS\_HR **=**1**>>>** DISPLAY\_MS\_MIN **=**3**>>>** DISPLAY\_LS\_MIN**=**3

run

**#** 11018**-**ns MAVEN SILICON **:** DISPLAY\_MS\_HR **=**1 **>>>** DISPLAY\_LS\_HR **=**1**>>>** DISPLAY\_MS\_MIN **=**3**>>>** DISPLAY\_LS\_MIN**=**4

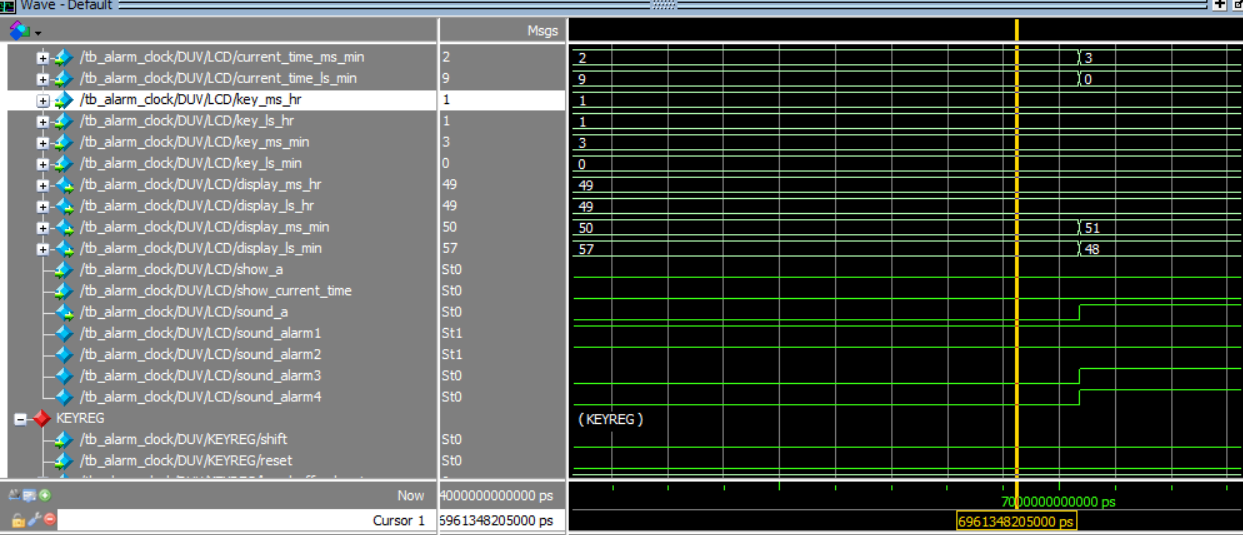
run

**#** 12018**-**ns MAVEN SILICON **:** DISPLAY\_MS\_HR **=**1 **>>>** DISPLAY\_LS\_HR **=**1**>>>** DISPLAY\_MS\_MIN **=**3**>>>** DISPLAY\_LS\_MIN**=**5

run

**#** 13018**-**ns MAVEN SILICON **:** DISPLAY\_MS\_HR **=**1 **>>>** DISPLAY\_LS\_HR **=**1**>>>** DISPLAY\_MS\_MIN **=**3**>>>** DISPLAY\_LS\_MIN**=**6

**Waveform**

****