

# James Cook University

## Electrical and Computer Engineering

### EE4306 VHDL Assignment

Occupational Therapy at James Cook University require a small alarm that will bleep to advise a wearer to record the activities/stance/task they are currently undertaking on a small diary. The evidence shows this is more effective at getting a sense of the things people do through a regular day than asking someone to record *all* activity in a diary for 24hrs or so. An assignment designing this was set for CC2510 during first semester.

A modification of this is set as a VHDL assignment for EE4306.

#### **The requirements are:**

The timer is to give a short beep (up to say 2-3 beeps in 3 secs) at a random point in every hour period. The system is to use a watch crystal for a clock (32768 Hz). The audible alarm is to be a piezoelectric speaker, this is to be driven directly from the CPLD in a differential manner. There is to be at least 5 random time checking intervals every minute.

#### **The Task is:**

Write the VHDL code for such a system. It is not necessary to construct any hardware, however the design should fit in either a ispLSI2032 PLCC package IC or an ispLSI2064 TQFP package IC. The clock should be connected to pin 11 and the differential timer outputs (to be directly connected to the piezoelectric speaker) are to be on pin 28 and pin 29.

Submit the complete design before 5pm on Monday 9 September. The assignment is worth 10% of the total course. Zip the complete directory for the VHDL code and all the resulting ispLever project files and email this to [Keith.Kikkert@jcu.edu.au](mailto:Keith.Kikkert@jcu.edu.au) , by the due date.

Ensure that all the submitted items (VHDL code included) are clearly labelled with your name and student number.

C. J. Kikkert  
12 August 2002

## Solution

This description concerns itself with a basic random timer. A possible extension required is to make the time interval other than the 60 minutes, so that a time interval variation to 30, 60, 90 or 120 minutes could easily be allowed for as part of the code for future extensions. Only the fixed time 60 minute interval timer is considered in this description.

A random timer is required, the simplest way to provide a random time is to use a pseudo random sequence generator, to generate a random number and then compare the actual time, with the contents of that random generator. The table below shows the tapping points and sequence length required for a Pseudo Random Generator.

# Registers	Seq. Length	Tapping Points
2	3	2,1
3	7	3,1
4	15	4,1
5	31	5,2
6	63	6,1
7	127	7,1
8	255	8,4,3,2
9	511	9,4
10	1 023	10,3
11	2 047	11,1
15	32 767	15,1
18	262 143	18,7
20	1 048 575	20,3
21	2 097 151	21,2
22	4 191 303	22,1
23	8 388 607	23,5

For at least 5 time checks per minute, and a 60 minute time interval, at least  $5 \times 60 = 300$  time intervals are required. One can count to 512 with a 9 bit counter. A 10 bit counter count to 1023 and will thus allow 5 time checks for a 120 minute period and can be considered as an extension for this project if longer times are required. The pseudo random sequence needs thus to be at least 9 registers long and will have 511 sequences before it repeats.

There are two ways for approaching this problem. Firstly, we can have a binary ratio divider to say 8 seconds, which will then give a maximum of 450 counts, which is more than the required. That then means we have to handle the PRSequence to give a number that is outside the one hour period, detect that and then generate a new sequence. That is done in the first solution presented in detail below.

The second solution is to have all the PRSequence possibilities cover the whole of the hour period. Since there are 511 possible sequences in a 9 register long PR sequence, the required clock for that must be  $3600/511 = 7.04500978$  seconds. We need to obtain this by an integer binary division from the 32 kHz crystal. This is the second solution presented.

### First solution.

If we use an 8 second clock to count to 60 minutes, then 450 counts are required. Thus if the PR generator generates a number above 450 then a new number needs to be generated, since the first number would be more than one hour.

A suitable clock for this system can be a simple 32.768 kHz quartz crystal. By using a 18-bit divider, an 8 second clock period is obtained. This is then used to drive the 9 bit counter, which is compared with the PR sequence.

The block diagram for such a system is shown in Figure 1.

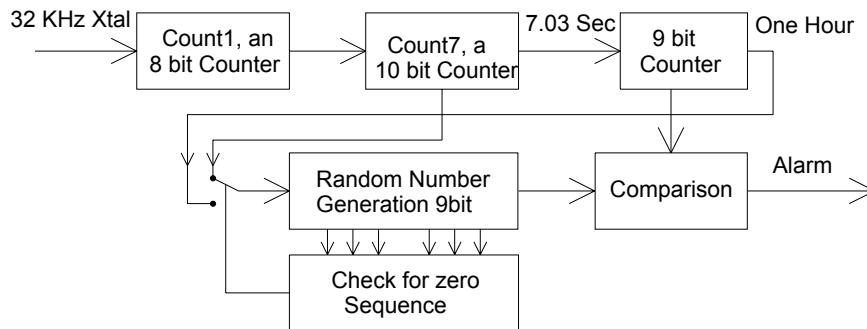


Figure 1. Basic Block Diagram.

In operation, every time the alarm goes off, the random number generator generates a new random number and uses that for the comparison. The alarm output will thus be the clock input for the random number generator. It is important that the random number generator does not change during the timing interval. However if the PRGenerator produces a value greater than 449, resulting in a time greater than on hour, then a new Random number needs to be generated immediately. This is handled by having the clock input to the PRGenerator being either the one Hour clock output from the hour counter or a higher frequency clock such as the 8 second clock.

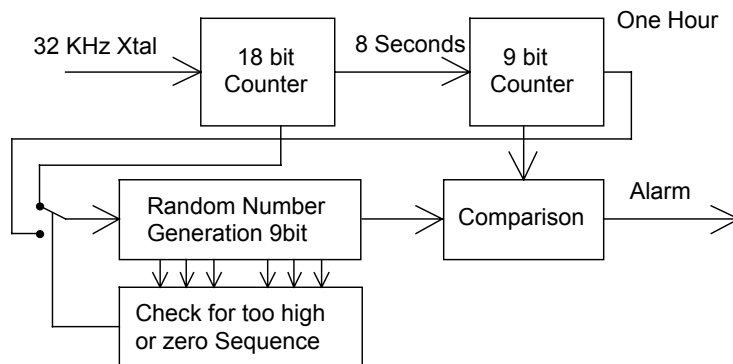


Figure 2. Block Diagram Solution 1.

Figure 2 shows the actual block diagram that is realised in the following VHDL code.

```

-- A prsequence generator and adder for EE4306 assignment 2002 Solution 1
-- Author C. J. Kikkert 21 September 2002

-- Clk is a 32x1024 kHz crystal clock. This is firstly divided to an 8 second clock
-- That 8 second clock is then divided further to produce the one hour pulse, which resets the counter and loads a
new PRSequence value.
-- If the PRSequence is out of range, ie greater than one hour then the next 8 second pulse will generate a new
PRSequence.

```

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
--use ieee.std_logic_arith.all;

```

```

entity Timer is

```

```

    Port (Clk : in std_logic ; -- 32x1024 Hz clock from Xtal
          Count1: inout std_logic_vector (17 downto 0); -- 18 bit counter to go from 32kHz Xtal to 8 seconds.
          CSec8 : inout std_logic_vector (8 downto 0); -- 9 bit counter for comparison with PRS.
          Alarm, Sec8Clk, SeqClk, TwoSec, OnekHz: inout std_logic;
          AlarmSound: out std_logic;
          HourClk: inout std_logic;
          BadStart, MaxCount, MaxSeq: inout std_logic;
          PRSeq: inout std_logic_vector (8 downto 0)
    );

```

```

-- Note Count1, CSec8 etc are normally only signals and are not brought out to pins to save resources.
-- Here we keep them as pins to facilitate testing see signals below.
end;

```

```

architecture TimeArc of Timer is

```

```

-- signal Count1: std_logic_vector (17 downto 0); -- 18 bit counter to go from 32kHz Xtal to 8 seconds.
-- signal CSec8: std_logic_vector (8 downto 0); -- 9 bit counter for comparison with PRS.
-- signal Sec8Clk: std_logic; -- signals for final version use, use Port for debugging
-- signal HourClk: std_logic;
-- signal Second, OnekHz: std_logic;

```

```

begin

```

```

    P_PRSeqClk: process (PRSeq, HourClk, Sec8Clk, MaxSeq)
-- change clock on PRSequence to 8sec clock if error or one hour if OK
    begin
        MaxSeq <= PRSeq(8) and PRSeq(7) and PRSeq(6) and (PRSeq(5) or PRSeq(4) or PRSeq(3) or
PRSeq(2) or PRSeq(1) or PRSeq(0));
        -- MaxSeq if >449, maximum count of 8 sec clock in one hour period.
        -- Binary code 449(dec)= 111000001(bin)
        SeqClk <= (MaxSeq and Sec8Clk) or (not MaxSeq and HourClk); --SeqClk Clock for PRSequence
    end process P_PRSeqClk;

```

```

    P_PRSeq: process
    begin

```

```

        wait until rising_edge (SeqClk);
        BadStart <= PRSeq(8) and PRSeq(7) and PRSeq(6) and PRSeq(5) and PRSeq(4) and PRSeq(3) and
PRSeq(2) and PRSeq(1) and PRSeq(0);
        -- if Seq = "11111111" then it stays at that. Need to change the value, Stuck if BadStart
        if BadStart = '1' then -- PRSeq stuck at 1, reset
            PRSeq <= "000000000"; -- ensure sequence will start set to zero
        else
            PRSeq(8 downto 1) <= PRSeq (7 downto 0);
            -- left shift the bits (note sll only for array of BITS)
            PRSeq(0) <= PRSeq(8) xnor PRSeq(3);
        end if;
    end process P_PRSeq;

```

```

    P_CountoSec8: process -- 18 bit divider to go from 32kHz to 8 seconds.
    begin

```

```

        wait until rising_edge (Clk);
        Count1 <= Count1 + "000000001";
-- Sec8Clk <= Count1(3); -- test for 8 sec clock below. Speeds up hour counters
-- TwoSec <= Count1(2); -- test for 2 sec clock below
-- OnekHz <= Count1(0); -- test for kHz waveforms below.
        Sec8Clk <= Count1(17); -- 8 second clock, also used as a mask for alarm waveform alarm 8 sec
        TwoSec <= Count1(15); -- two second waveform gate with 1 kHz clock to produce alarm tone,
-- 1 Sec ON, 1 sec OFF pulse.

```

```

    OnekHz <= Count1(4);      -- kHz waveforms for gating with 1 sec ON,
                             -- Count1(0) for test, Count1(4) for normal operation
end process P_CountoSec8;

P_CountoHr: process
begin
    wait until rising_edge (Sec8Clk);
    MaxCount <= CSec8(8) and CSec8(7) and CSec8(6) and ( CSec8(5) or CSec8(4) or CSec8(3) or CSec8(2)
    or CSec8(1) or CSec8(0)); --449 or larger
    -- MaxSeq if >449, maximum count of 8 sec clock in one hour period.
    -- Binary code 449(dec)= 111000001(bin)
    if (MaxCount = '1') then
        CSec8 <= "000000001";
    else
        CSec8 <= CSec8 + "000000001";
    end if;
    --
    HourClk <= CSec8(2);      -- for testing only to give a quick response.
    HourClk <= CSec8(8);

end process P_CountoHr;

P_AlarmSound:process (Alarm, TwoSec, OnekHz)
begin
    AlarmSound <= Alarm and TwoSec and OnekHz;
end process P_AlarmSound;

P_Alarm: process          -- turn Alarm off after 8 sec.
begin
    wait until rising_edge (Sec8Clk);
    if (CSec8 = PRSeq) then
        Alarm <= '1';      --turn alarm ON
    else
        Alarm <= '0';
    end if;
end process P_Alarm;

end TimeArc;

```

## Second Solution

For the second solution, the 18 bit counter is split into two counters. The first counter Count1 is a counter that is a multiple of 4 bits long, since for the ispLSI2064 all the 4 flipflops in a GLB have the same clock signal. Note that this does not apply for the ispMach4064 device, which also has 64 flipflops, but uses less current and can operate at a lower supply voltage. The first counter divides by a binary power integer, ie 256 and the second counter divides by an integer.

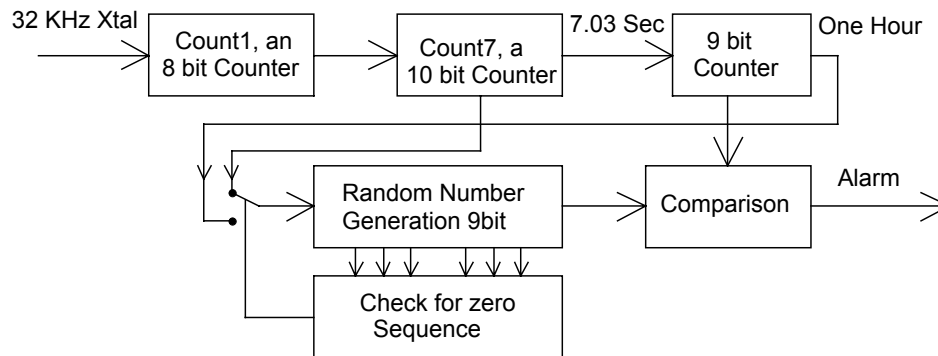


Figure 3. Block Diagram Solution 2.

Since the required clock must be  $3600/511 = 7.04500978$  seconds. We require a 115425.4 division from the 32x1024kHz clock in order to obtain the required 7.045 second output. Since this is not an integer, this solution is not possible. If we do not set the alarm for the first 7 seconds, we can use 512 different sequences, thus requiring a 115200 division ratio, which is an integer ratio and which can be obtained by having a divide by 256 (8 F/F thus two whole GLB's) followed by a 10 bit counter, that counts up to 900. The output of that counter is a 7.03125 second pulse. We must ensure that the PR sequence 00000000 corresponds to the only invalid state of the sequence generator and that if that sequence occurs, the generator is immediately set to some other value.

The code below is for this solution. The code is the testing form, rather than the final form, so that the alarm waveforms can be seen and verified using reasonable length waveforms.

```
-- A radnom timer for EE4306 assignment 2002 Solution 2
-- Author C. J. Kikkert 21 September 2002

-- Clk is a 32x1024 kHz crystal clock. Maximum time = 1 hour.
-- PRSeq length 511 before repeat. Use 512 values in CountH and PRSeq giving 7.03125 seconds per time quantum
-- Sequence 000000 does not exist, so no check first 7 seconds. OK
-- Count1 8 bit counter to divide from 32x1024 kHz to 128 Hz
-- Count7 10 bit counter to count to 900 max. and 7.03125 second pluses
-- CountH 9 bit counter procuce the once per hour output HourOut
-- HourOut is the clock for the PRSeq generator, when the the value of the PRSeq is zero (wrong state) then
-- it is loaded to 010101010 using the 32 kHz clock.

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;          --includes ieee.std_logic_arith.all and allows adding by one.

entity Timer is
    Port (Clk : in std_logic ;                -- 32x1024 Hz clock from Xtal
          Count1: inout std_logic_vector (7 downto 0);    -- 8 bit counter to go from 32kHz Xtal to 128 Hz.
          Count7: inout std_logic_vector (9 downto 0);    -- 10 bit counter to go from 127Hz to 7.03125 seconds,
                                                    -- divide by 900
          CountH : inout std_logic_vector (8 downto 0);   -- 9 bit counter to one Hour for comparison with PRS.
          Alarm : inout std_logic;
          Count7Clk, CountHClk, SeqClk, TwoSec, OnekHz: inout std_logic;
          AlarmSound: out std_logic;
          HourOut: inout std_logic;
          BadStart : inout std_logic;
          PRSeq: inout std_logic_vector (8 downto 0)
    );
end;

architecture TimeArc of Timer is
-- signal Count1: std_logic_vector (17 downto 0);    -- 8 bit counter to go from 32kHz Xtal to 128 Hz.
-- signal CSec8: std_logic_vector (8 downto 0);    -- 9 bit counter for comparison with PRS.
-- signal Sec8Clk: std_logic;
-- signal HourClk: std_logic;
-- signal Second, OnekHz: std_logic;
-- these signals are used in the final realisation once the system has been fully debugged to reduce I/O resources.

begin
    P_PRSeqClk: process (PRSeq, Clk, BadStart, Hourout)    -- process to set PRSequence Clock for fast reset
    begin
        BadStart <= not (PRSeq(8) or PRSeq(7) or PRSeq(6) or PRSeq(5) or PRSeq(4) or PRSeq(3) or PRSeq(2)
            or PRSeq(1) or PRSeq(0));
            -- if Seq = "000000000" then it stays at that. Need to change the value, Stuck if
    BadStart = '1'
    --          SeqClk <= (BadStart and Clk) or (not BadStart and Count7Clk);    -- test fast version
    --          SeqClk <= (BadStart and Clk) or (not BadStart and HourOut);    -- FastClock if Badstart, Hourly
    clockpulse if OK
    end process P_PRSeqClk;
```

```

P_PRSeq: process
begin
    wait until rising_edge (SeqClk);
    if BadStart = '1' then          -- PRSeq stuck at 1, reset
        PRSeq <= "010101010";      -- ensure sequence will start set to zero
    else
        PRSeq(8 downto 1) <= PRSeq (7 downto 0);  -- left shift the bits
    PRSeq(0) <= PRSeq(8) xor PRSeq(3);
    end if;                          -- value 010101010 chosen to give suitable first timer value
end process P_PRSeq;

P_Count1: process                    -- 8 bit divider to go from 32kHz to 128 Hz.
begin
    wait until rising_edge (Clk);     -- Count1 High freq clock to divide 32kHz to 128 Hz
    Count1 <= Count1 + "00000001";
    Count7Clk <= not Count1(2);       -- test for 7 sec clock below. Speeds up hour counters
    OnekHz <= Count1(0);              -- test for kHz waveforms below.
    TwoSec <= Count1(2);              -- test for 2 sec clock below

--    Count7Clk <= not Count1(7);
--    -- 8 second clock, also used as a mask for alarm waveform alarm time 8 seconds
--    OnekHz <= Count1(4);            -- One kHz waveform for gating with 1 sec ON, Count1(0) for
test, Count1(4) for normal operation

end process P_Count1;

P_Count7: process                    -- 10 bit divider to go from 128Hz to 7 seconds, divide by 900.
begin
--    wait until rising_edge (Clk);    -- Test High freq clock to divide fast
    wait until rising_edge (Count7Clk); -- 128Hz Clock
    if (Count7 < "1110000011") then    -- counter is a divide by 900 circuit 899="1110000011"
        Count7 <= Count7 + "0000000001";
    else
        Count7 <= "0000000000";        -- reset counter
    end if;
    CountHClk <= not Count7(0);         -- Test Clock for Hour counter, speeds up hour counters

--    CountHClk <= not Count7(9);      -- Clock for Hour counter, a 7 second clockpulse,
--    -- also used as a mask for alarm waveform alarm time 7 seconds
--    TwoSec <= Count7(7);            -- two second waveform to produce alarm tone
end process P_Count7;

P_CounH: process
begin
--    wait until rising_edge (Clk);    --test
    wait until rising_edge (CountHClk);
    CountH <= CountH + "000000001";
--    HourOut <= not CountH(3);        -- test to speed PRSEQ waveform up
    HourOut <= not CountH(8);         -- Clock for PRSeq generator, one new sequence every hour.
end process P_CounH;

P_AlarmSound:process (Alarm, TwoSec, OnekHz)
begin
    AlarmSound <= Alarm and TwoSec and OnekHz;
end process P_AlarmSound;

P_Alarm: process                    -- turn Alarm off after 7 sec.
begin
    wait until rising_edge (Count7Clk);
    if (CountH = PRSeq) then
        Alarm <= '1';                --turn alarm ON
    else
        Alarm <= '0';
    end if;
end process P_Alarm;

end TimeArc;

```

The file compiles nicely and this file fits inside a ispLSI2064 with 46 macrocells. The partitioning will be slightly different with the final rather than the test design. For this

design, optimising on area actually gave a larger number of macrocells than the fitting for minimum delay. The resulting synthesis and partitioning statistics (delay optimised) are:

Number of Macrocells is 46  
Number of GLBs is 14  
Number of product terms is 150  
Maximum number of GLB levels is 2  
Average number of inputs per GLB is 9.4  
Average number of outputs per GLB is 3.3  
Average number of product terms per GLB is 10.7

Synthesis and partitioning completed successfully

Two parts of the test waveforms are presented. Firstly the waveforms at the start, showing how the PRSequence generator is initialised away from the all zero code. This also starts the alarm for 3.5 seconds, which gives confidence in the system working.



Figure 4. Waveforms at powerup.

The second waveform shows the alarm condition, with the 1kHz signal being pulsed ON and OFF at a one second rate. In practice there are more 1kHz pulses and four two second pulses. The waveforms shown are for the test waveforms only.

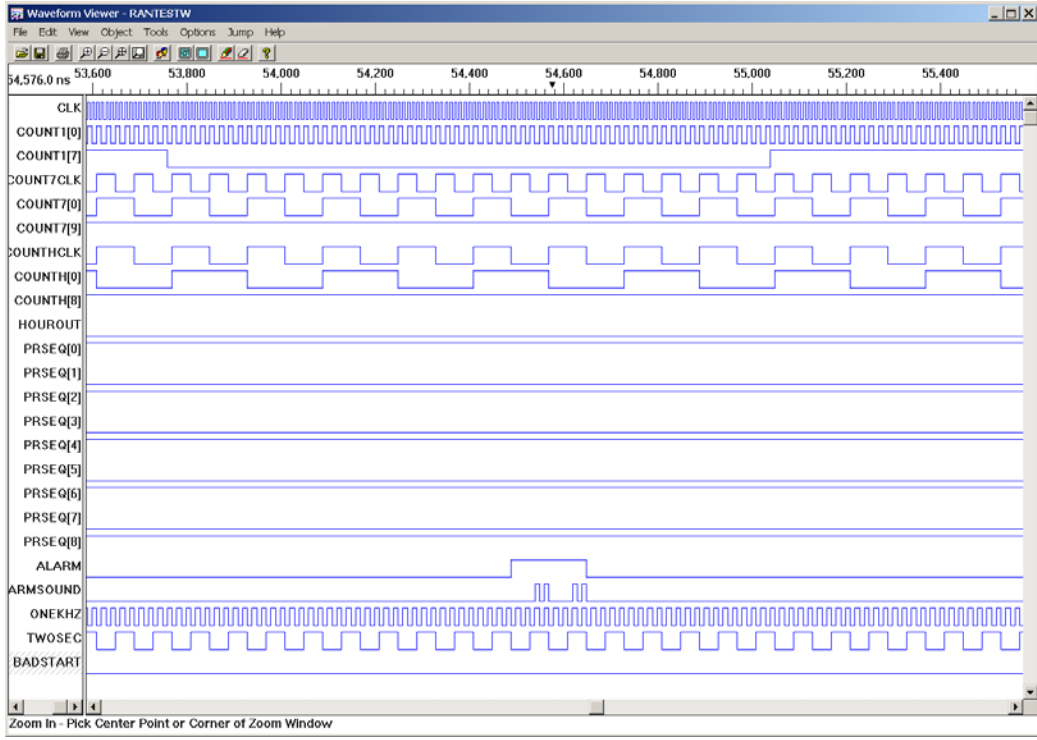


Figure 4. Waveforms at alarm condition