



Chapter 7.1

Additional Design Example

Version: 2023/12/26

Contents



Design of a Wristwatch

Memory Timing Models

A Universal Asynchronous Receiver Transmitter

1 Design of a Wristwatch

Multifunction wristwatch

3 modes:

- Time-keeping
- Alarm
- Stopwatch

3 Buttons: **B1**, **B2**, **B3**

- **B1**: mode change:
Time → Alarm → Stopwatch → Time
- **B2**: depend on the mode
- **B3**: depend on the mode

1.1 Specifications

Operation in **time** mode:

- hh:mm:ss A/P
- **B3**: shut off the alarm
- **B2**: Time mode → Set Hours → Set Minutes → Time mode
- **B3**: +1 in Set Hours/Minutes

Operation in **alarm** mode:

- hh:mm A/P
- **B2**: Alarm mode → Set Alarm Hours → Set Alarm Minutes → Alarm mode
- **B3**: +1 in Set Alarm Hours/Minutes
- **B3**: set/reset in Alarm state
- Ring for 50 seconds (shut off by B3)

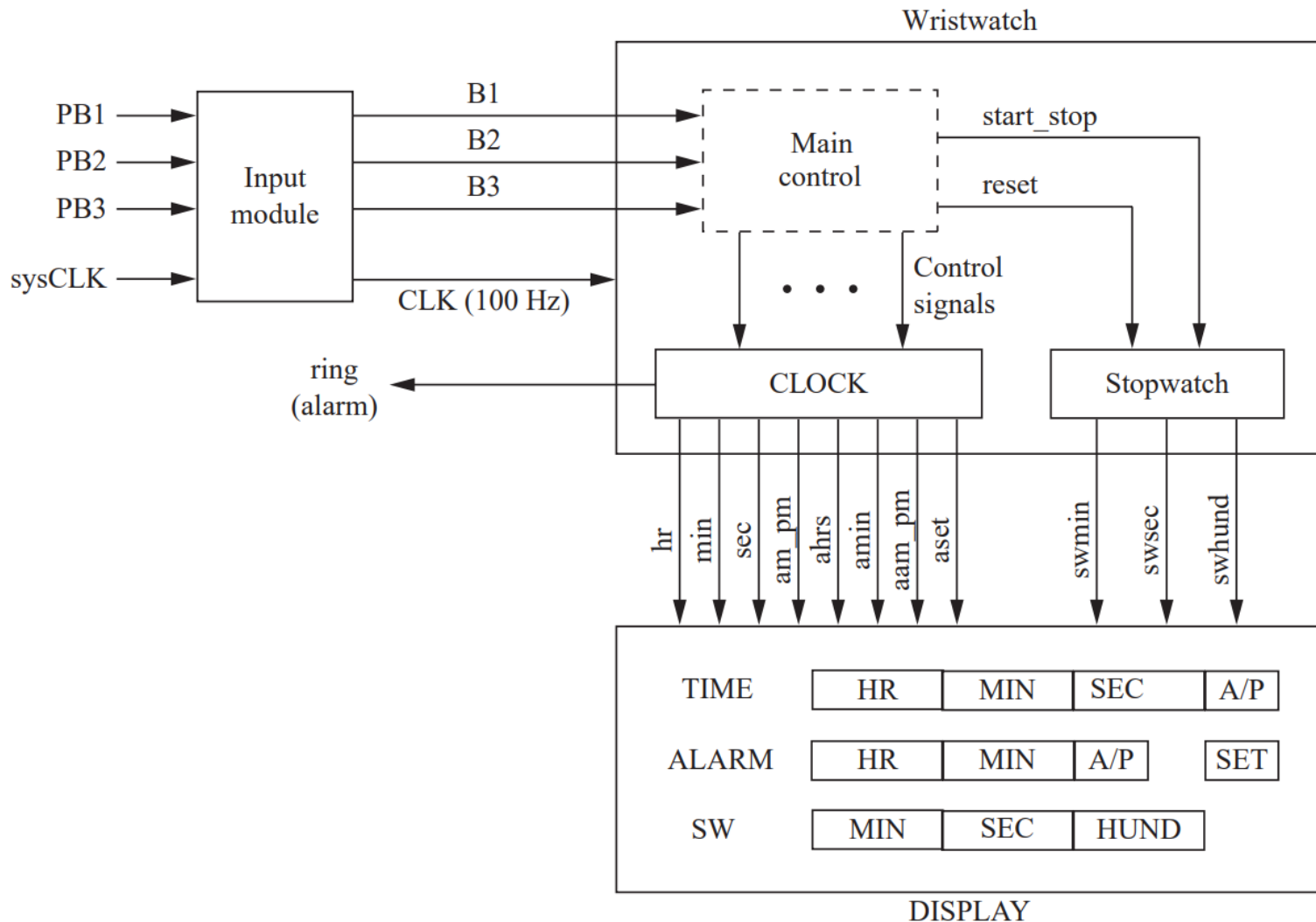
1.1 Specifications

Operation in **stopwatch** mode:

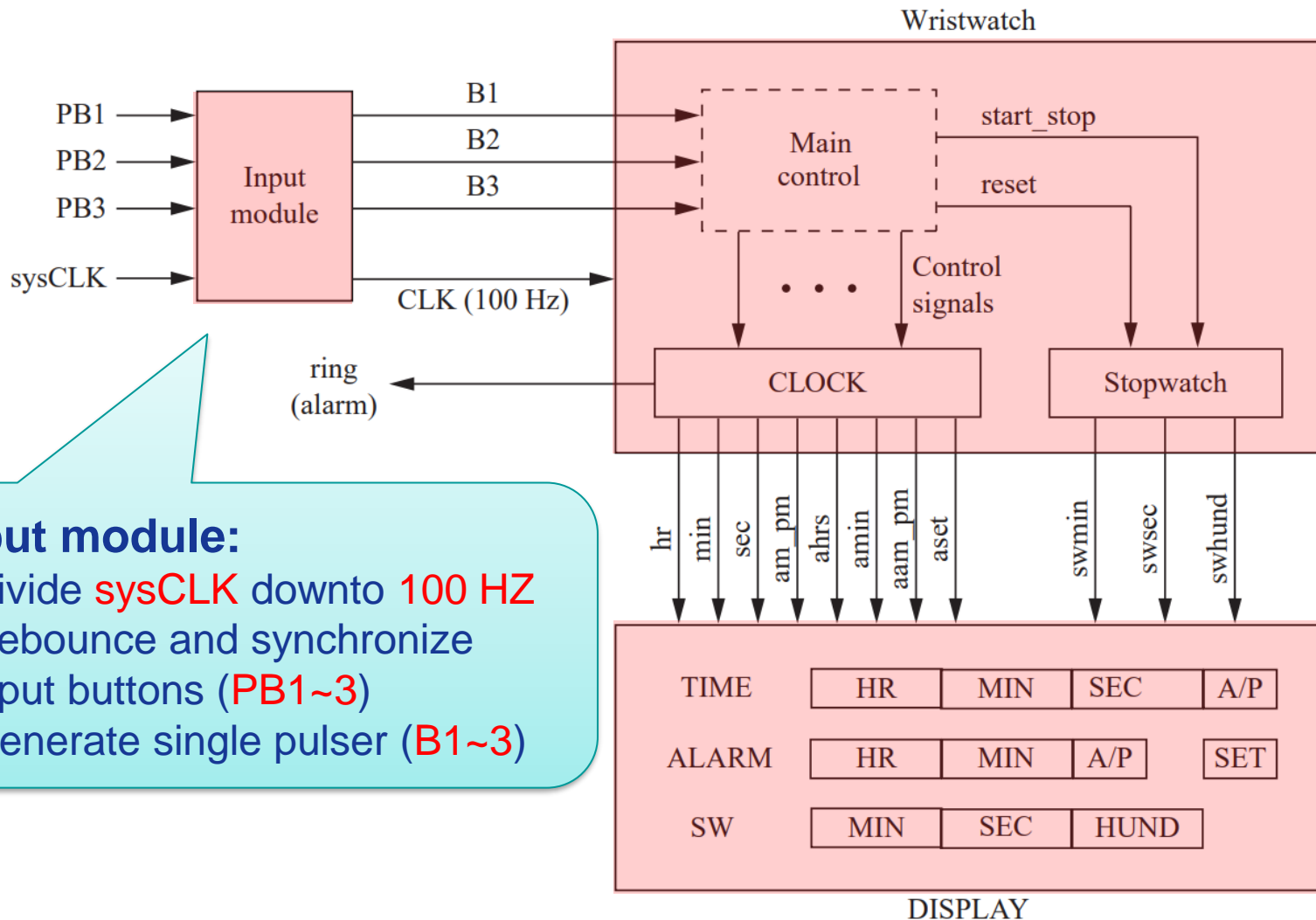
- mm:ss.cc  cc is 1% second
- **B2**: start → stop → restart
- **B3**: reset

1.1 Specifications

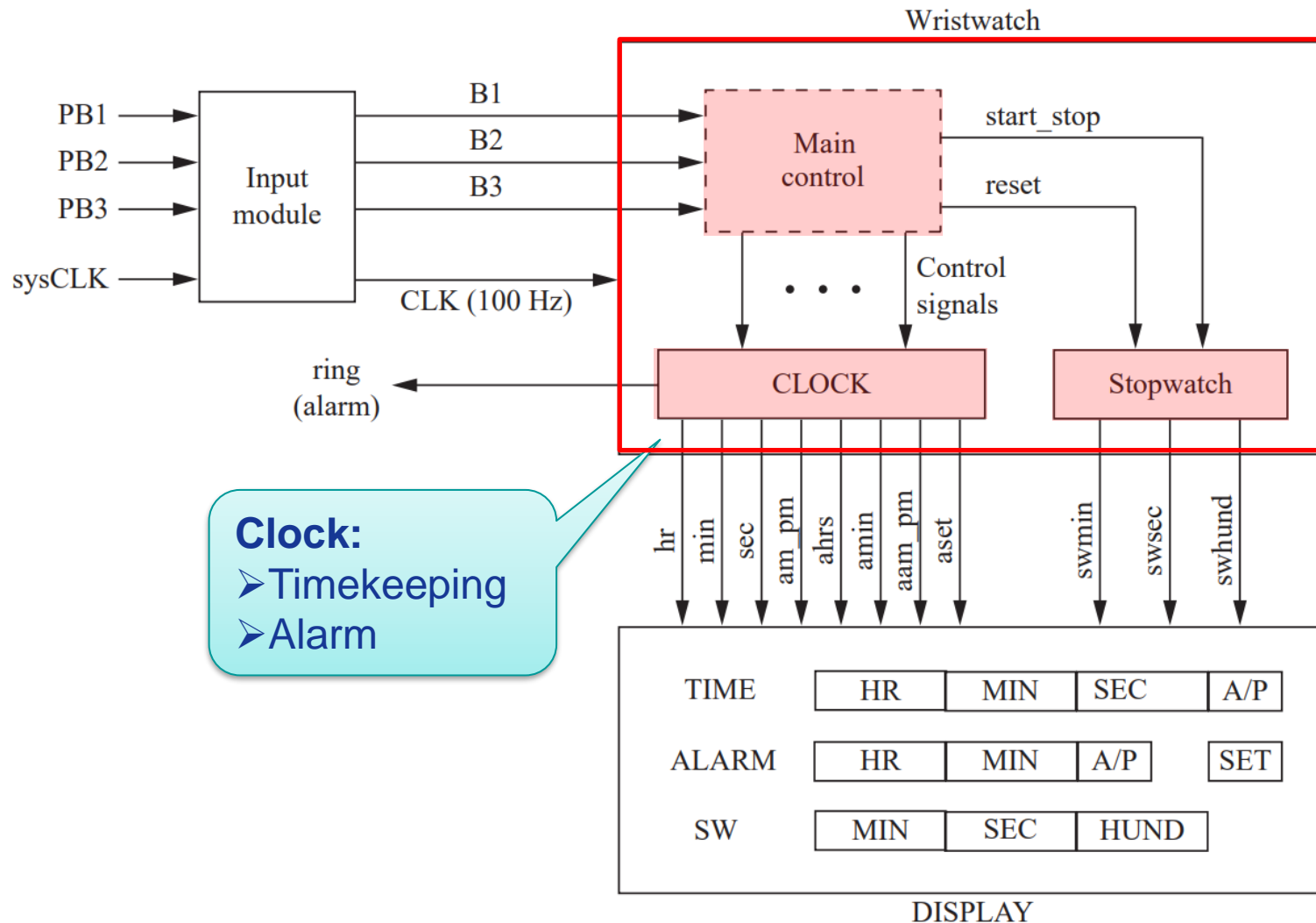
Block Diagram of Wristwatch Design



1.1 Specifications

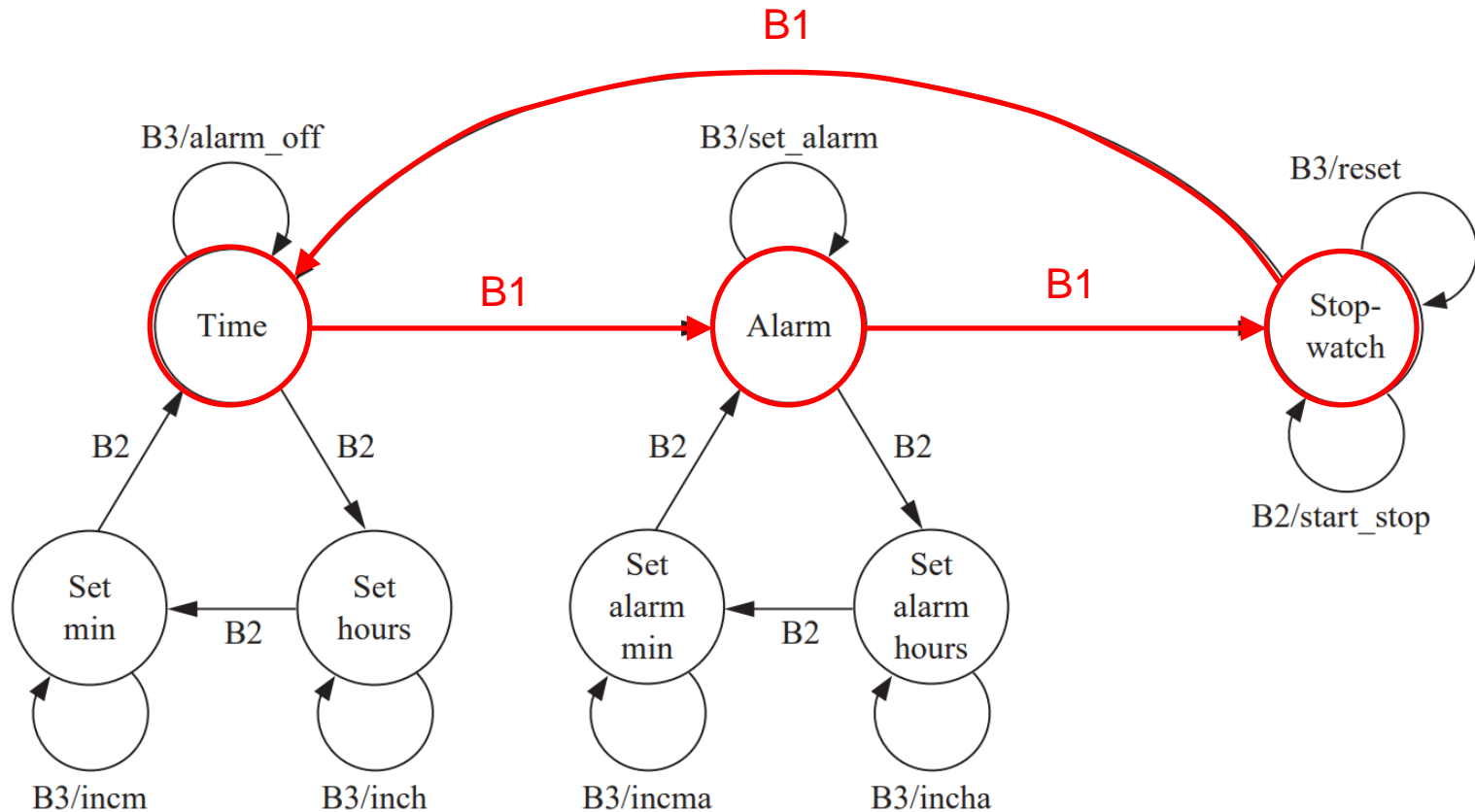


1.1 Specifications



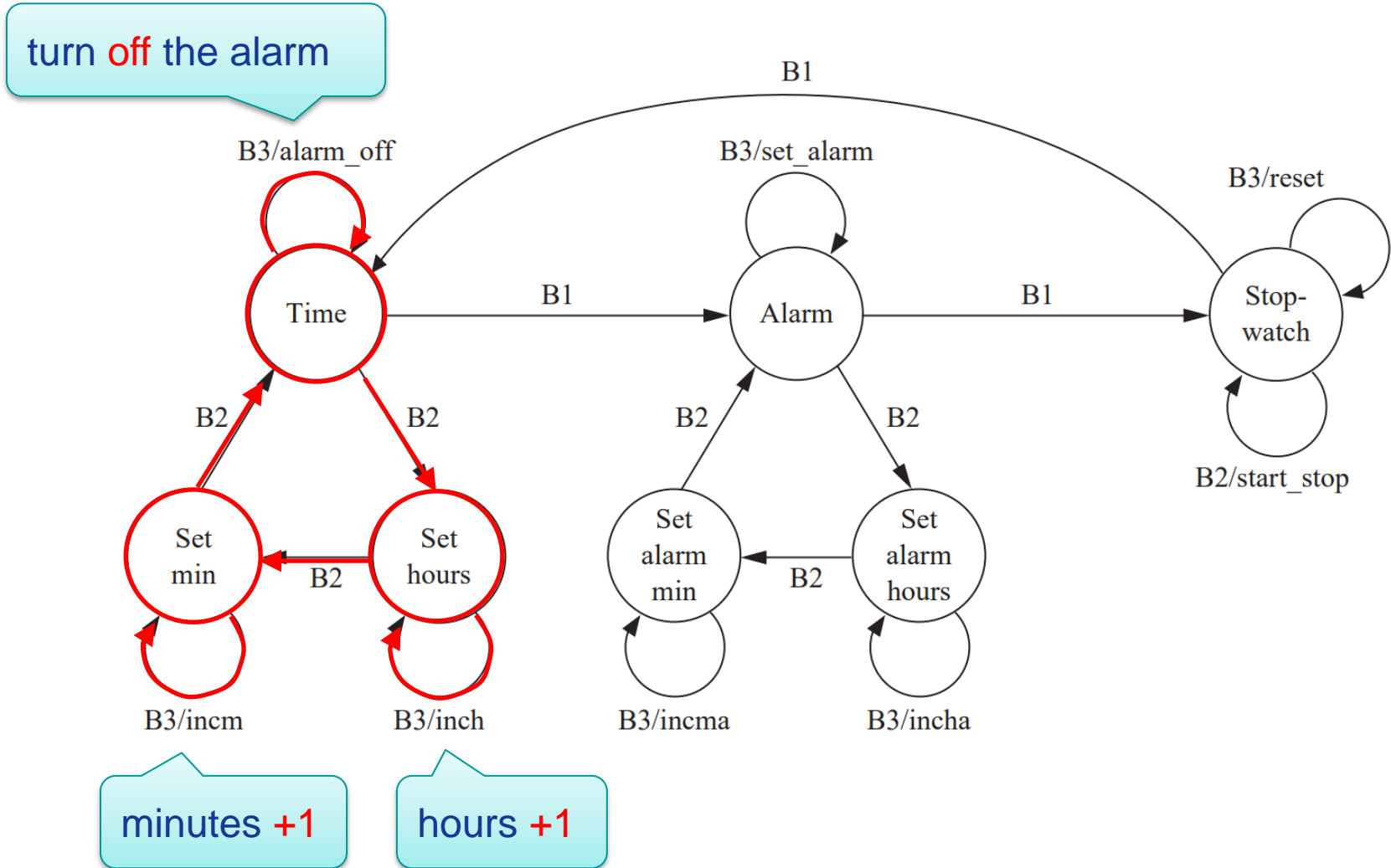
1.1 Specifications

State Graph of Wristwatch Module



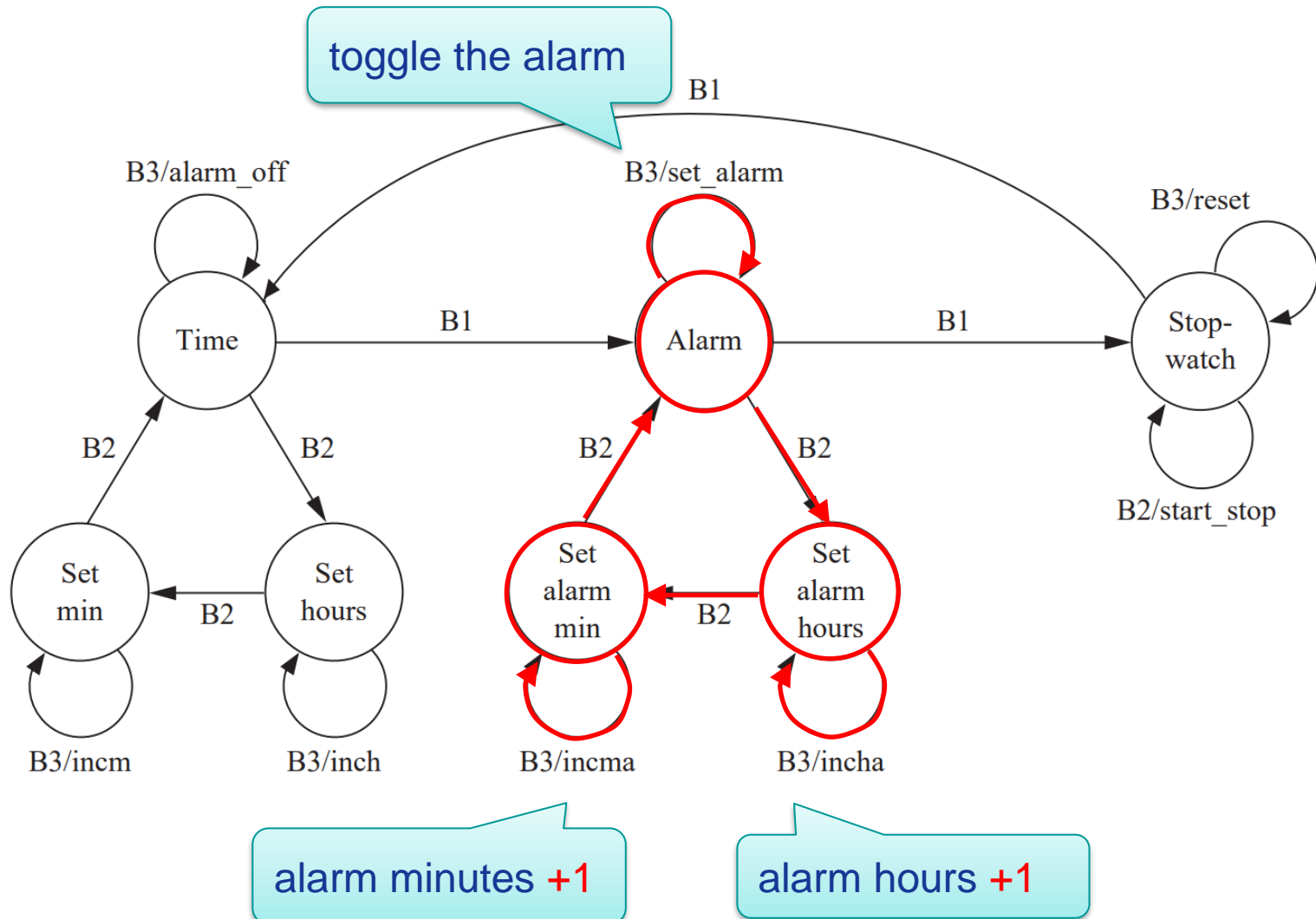
1.1 Specifications

State Graph of Wristwatch Module



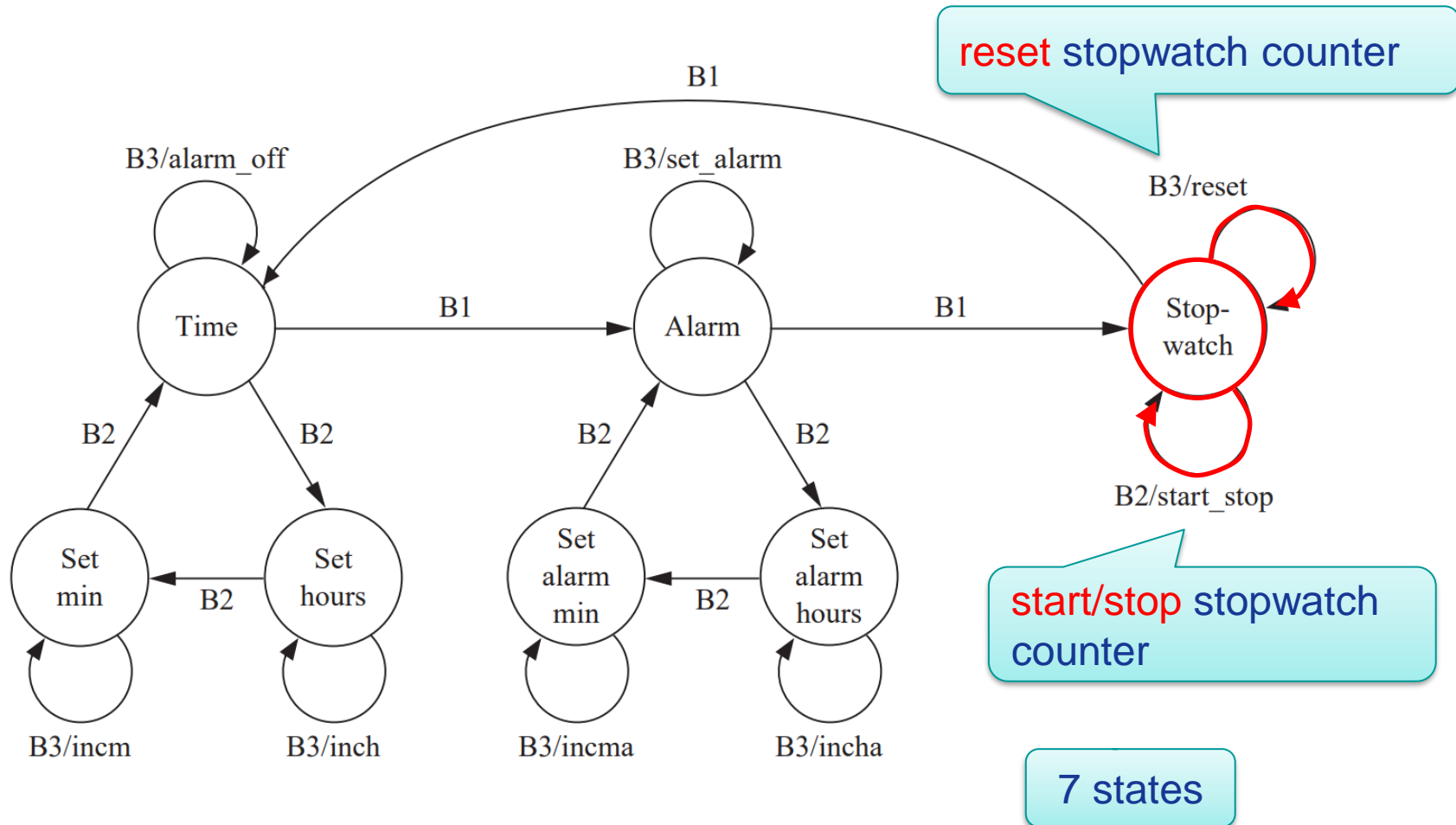
1.1 Specifications

State Graph of Wristwatch Module



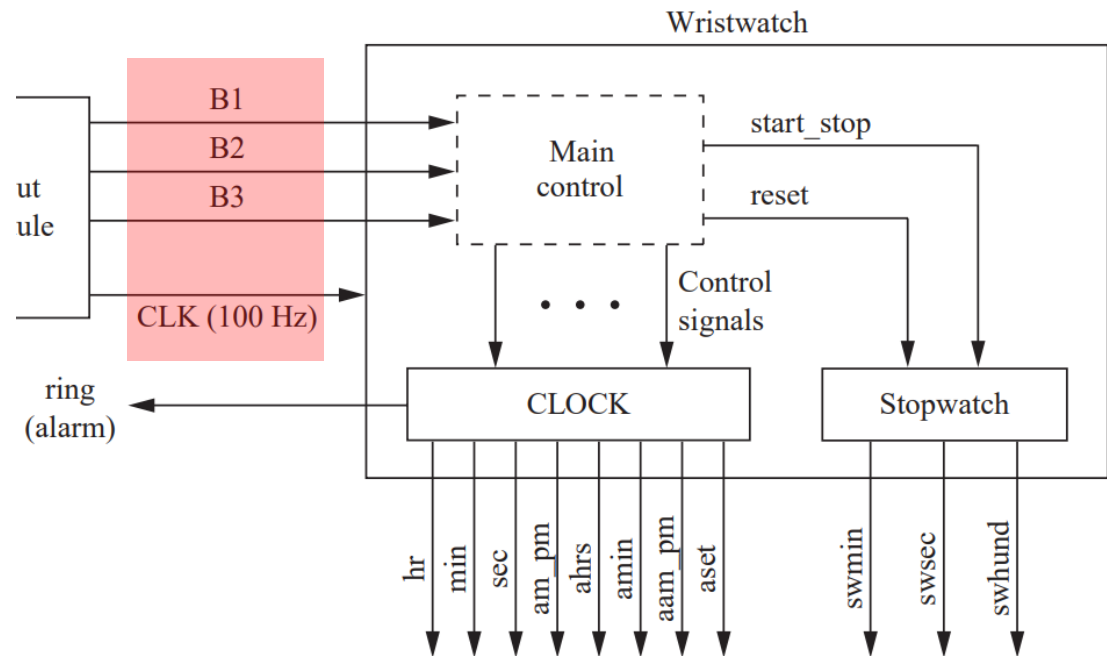
1.1 Specifications

State Graph of Wristwatch Module



1.2 Design Implementation

VHDL Code for the Wristwatch Module

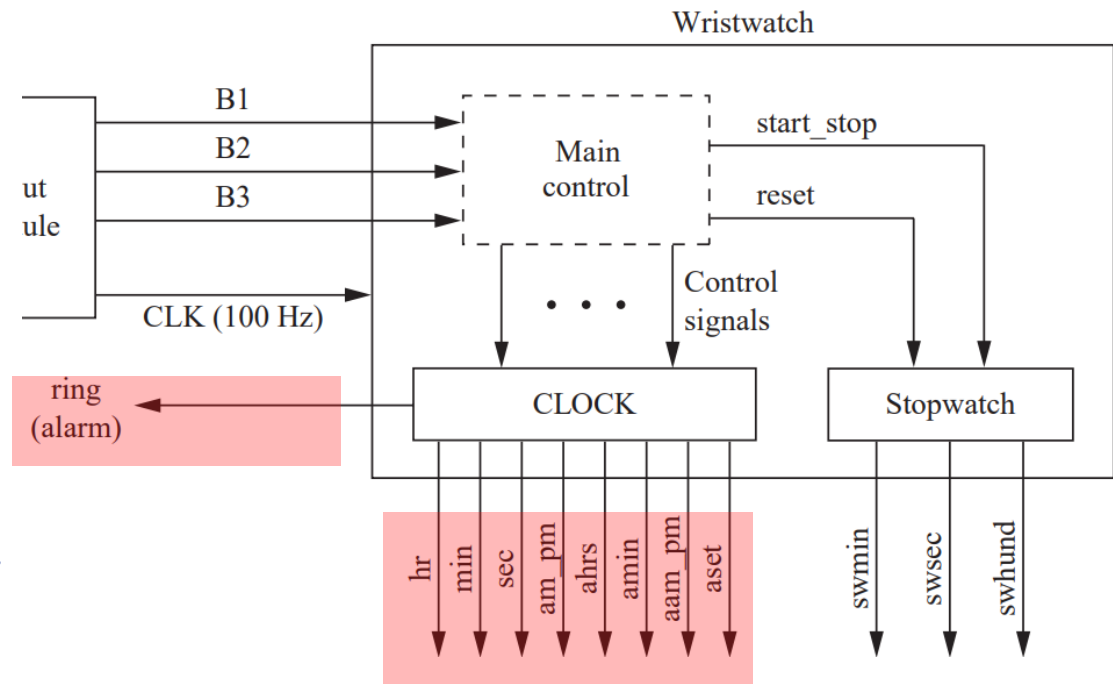


```
library IEEE;
use IEEE.numeric_bit.all;

entity wristwatch is
    port (B1, B2, B3, clk: in bit;
          am_pm, aam_pm, ring, alarm_set: inout bit;
          hours, ahour, minutes, aminutes, seconds: inout unsigned(7
downnto 0);
          swhundreths, swseconds, swminutes: out unsigned(7 downto 0));
end wristwatch;
```

1.2 Design Implementation

VHDL Code for the Wristwatch Module



```
library IEEE;
use IEEE.numeric_bit.all;
```

```
entity wristwatch is
```

```
    port(B1, B2, B3, clk: in bit;
```

```
          am_pm, aam_pm, ring, alarm_set: inout bit;
```

```
          hours, ahour, minutes, aminutes, seconds: inout unsigned(7
```

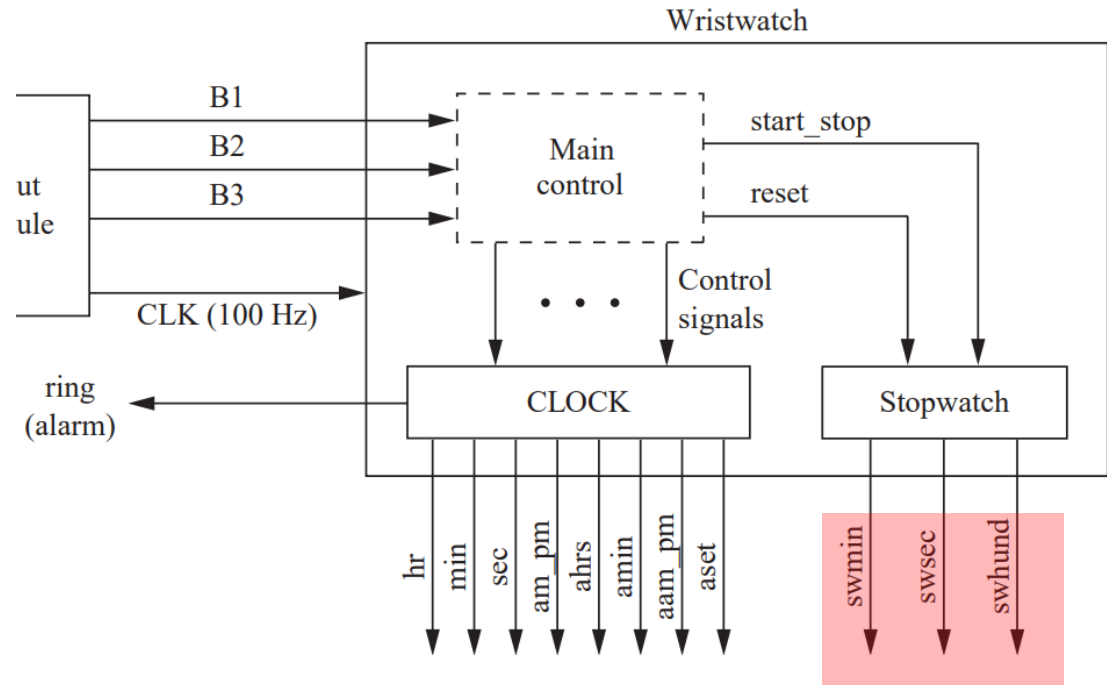
```
          downto 0));
```

```
          swhundreths, swseconds, swminutes: out unsigned(7 downto 0));
```

```
end wristwatch;
```

1.2 Design Implementation

VHDL Code for the Wristwatch Module



```
library IEEE;
use IEEE.numeric_bit.all;

entity wristwatch is
    port(B1, B2, B3, clk: in bit;
         am_pm, aam_pm, ring, alarm_set: inout bit;
         hours, ahour, minutes, aminutes, seconds: inout unsigned(7
downto 0);
         swhundreths, swseconds, swminutes: out unsigned(7 downto 0));
end wristwatch;
```

1.2 Design Implementation

```
architecture wristwatch1 of wristwatch is
```

```
    component clock is
```

```
        port(clk, inch, incm, incha, incma, set_alarm, alarm_off; in bit;  
            hours, ahours, minutes, aminutes, seconds: inout unsigned(7  
downto 0));
```

```
            am_pm, aam_pm, ring, alarm_set: inout bit);
```

```
    end component;
```

```
    component stopwatch is
```

```
        port(clk, reset, start_stop: in bit;
```

```
            swhundreths, swseconds, swminutes: out unsigned(7 downto 0));
```

```
    end component;
```

```
    type st_type is (timel, set_min, set_hours, alarm, set_alarm_hrs,  
                    set_alarm_min, stop_watch);
```

```
    signal state, nextstate: st_type;
```

```
    signal inch, incm, alarm_off, set_alarm, incha, incma, start_stop,  
        reset: bit;
```

```
begin
```

```
    ... ..
```

```
end wristwatch1;
```



```
component clock is
```

```
    port(clk, inch, incm, incha, incma, set_alarm, alarm_off; in bit;  
        hours, ahours, minutes, aminutes, seconds: inout unsigned(7  
downto 0);
```

```
        am_pm, aam_pm, ring, alarm_set: inout bit);
```

```
end component;
```

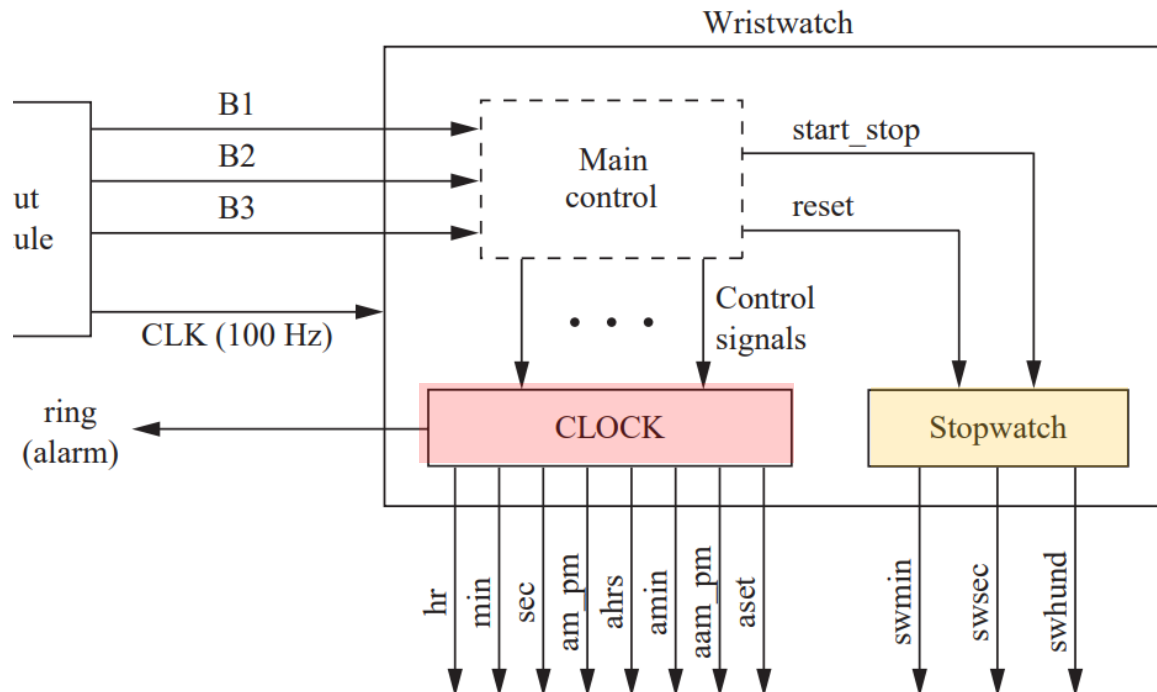
described later

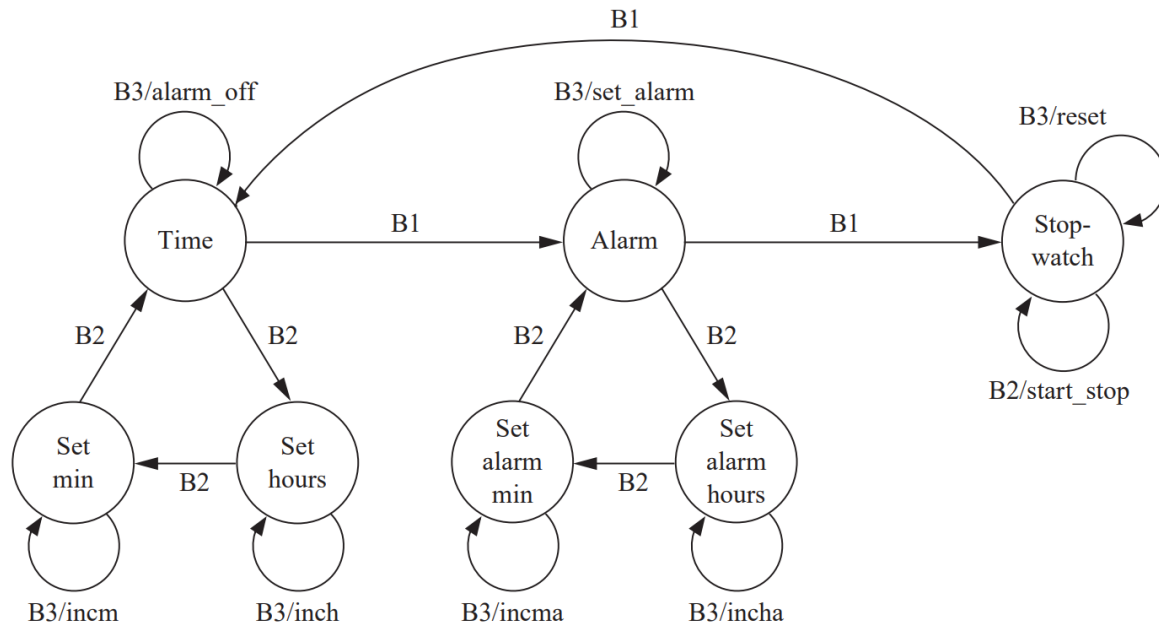
```
component stopwatch is
```

```
    port(clk, reset, start_stop: in bit;
```

```
        swhundreths, swseconds, swminutes: out unsigned(7 downto 0));
```

```
end component;
```





```

type st_type is (timel, set_min, set_hours, alarm, set_alarm_hrs,
                  set_alarm_min, stop_watch);
signal state, nextstate: st_type;
signal inch, incm, alarm_off, set_alarm, incha, incma, start_stop,
        reset: bit;

```

```

begin

```

```

    ... ..

```

```

end wristwatch1;

```

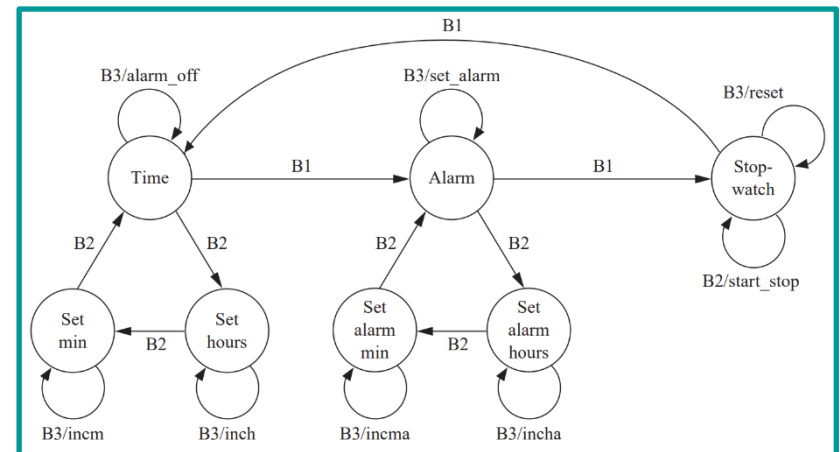
- States of Main control
- Control signals to **CLOCK & STOPWATCH**

1.2 Design Implementation

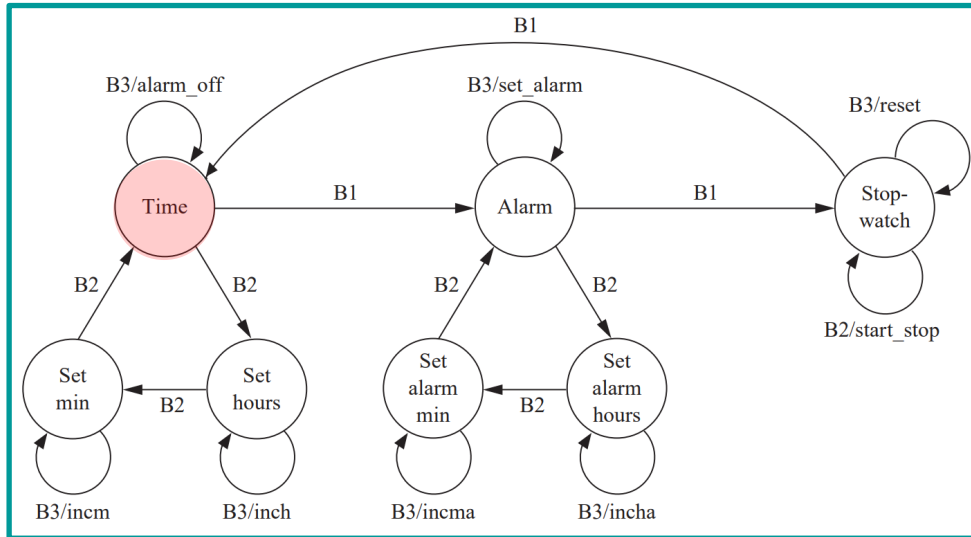
```
begin  
  clock1: clock port map(clk, inch, incm, incha, incma, set_alarm,  
                           alarm_off, hours, ahours, minutes, aminutes,  
                           seconds, am_pm, aam_pm, ring, alarm_set);  
  
  stopwatch1: stopwatch port map(clk, reset, start_stop, swhundreths,  
                                   swseconds, swminutes);
```

```
process(state, B1, B2, B3)  
begin  
  ... ..  
end process;
```

```
process(clk)  
begin  
  if clk'event and clk = '1' then  
    state <= nextstate;  
  end if;  
end process;  
end wristwatch1;
```



1.2 Design Implementation



begin

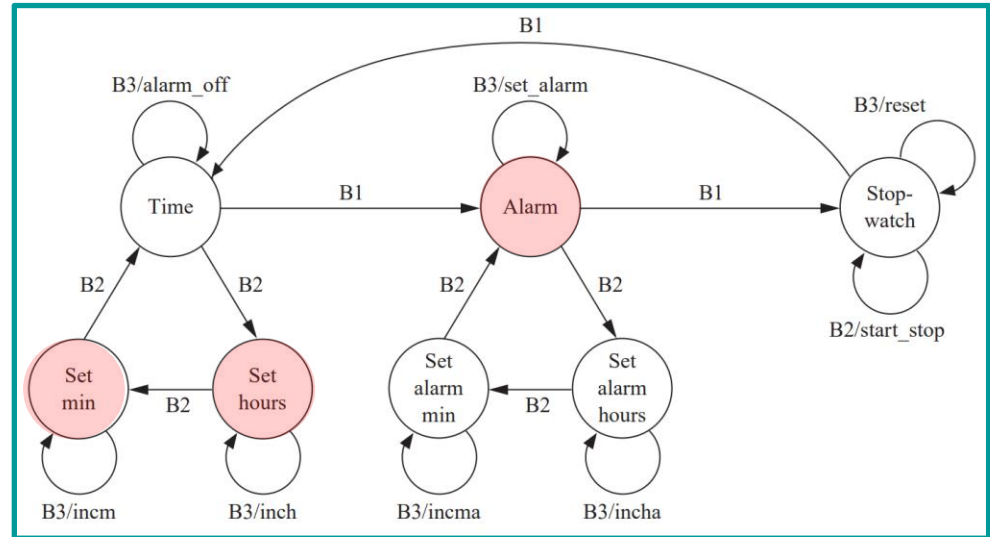
```
alarm_off <= '0'; inch <= '0'; incm <= '0'; set_alarm <= '0'; incha <= '0';  
incma <= '0'; start_stop <= '0'; reset <= '0';
```

case state is

when timel =>

```
if B1 = '1' then nextstate <= alarm;  
elsif B2 = '1' then nextstate <= set_hours;  
else nextstate <= timel;  
end if;  
if B3 = '1' then alarm_off <= '1';  
end if;
```

1.2 Design Implementation

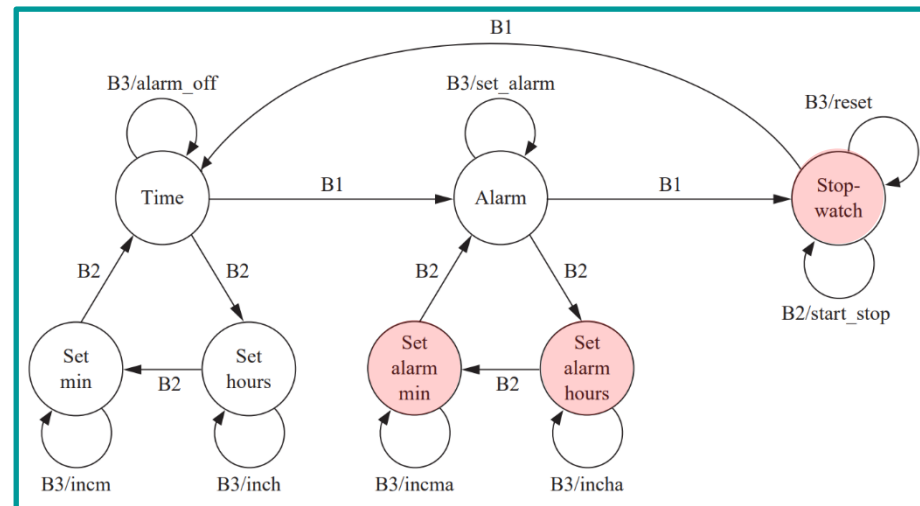


```

when set_hours =>
    if B3 = '1' then inch <= '1'; nextstate <= set_hours;
    else nextstate <= set_hours;
    end if;
    if B2 = '1' then nextstate <= set_min;
    end if;
when set_min =>
    if B3 = '1' then incm <= '1'; nextstate <= set_min;
    else nextstate <= set_min;
    end if;
    if B2 = '1' then nextstate <= time1;
    end if;
when alarm =>
    if B1 = '1' then nextstate <= stop_watch;
    elsif B2 = '1' then nextstate <= set_alarm_hrs;
    
```

1.2 Design Implementation

```
else nextstate <= alarm;
end if;
if B3 = '1' then set_alarm <= '1'; nextstate <= alarm;
end if;
when set_alarm_hrs =>
  if B2 = '1' then nextstate <= set_alarm_min;
  else nextstate <= set_alarm_hrs;
  end if;
  if B3 = '1' then incha <= '1';
  end if;
when set_alarm_min =>
  if B2 = '1' then nextstate <= alarm;
  else nextstate <= set_alarm_min;
  end if;
  if B3 = '1' then incma <= '1';
  end if;
when stop_watch =>
  if B1 = '1' then nextstate <= time1;
  else nextstate <= stop_watch;
  end if;
  if B2 = '1' then start_stop <= '1';
  end if;
  if B3 = '1' then reset <= '1';
  end if;
end case;
end process;
```



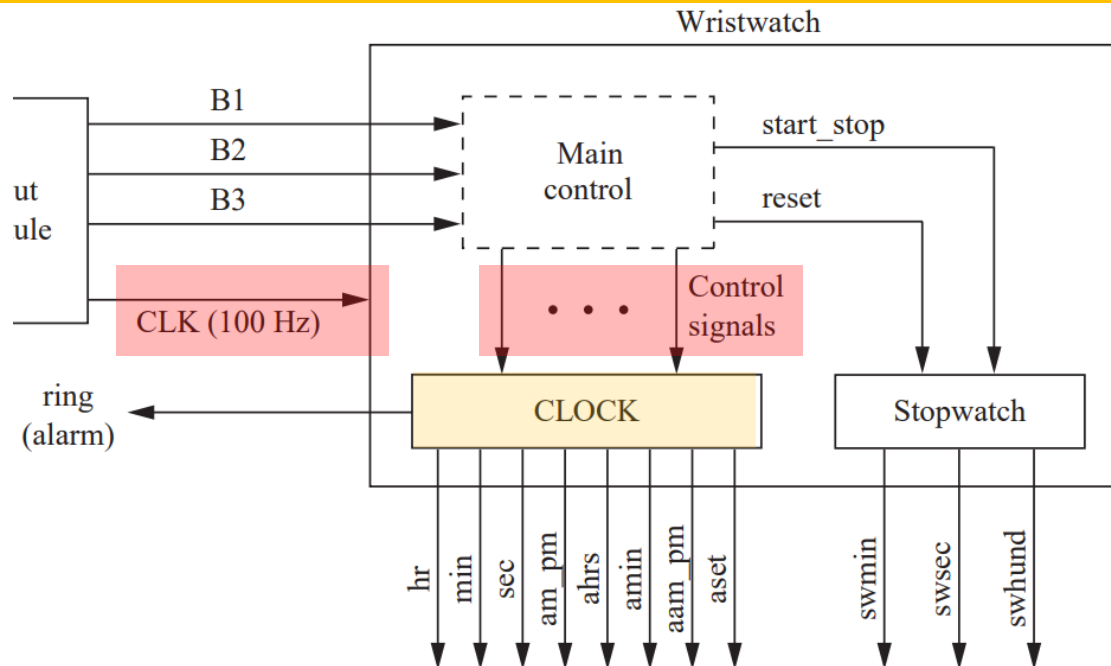
1.2 Design Implementation

Clock Module

```
library IEEE;  
use IEEE.numeric_bit.all;
```

CLK + Control signals from Main control

```
entity clock is  
  port(clk, inch, incm, incha, incma, set_alarm, alarm_off: in bit;  
        hours, ahours, minutes, aminutes, seconds: inout unsigned(7 downto 0);  
        am_pm, aam_pm, ring, alarm_set: inout bit);  
end clock;
```

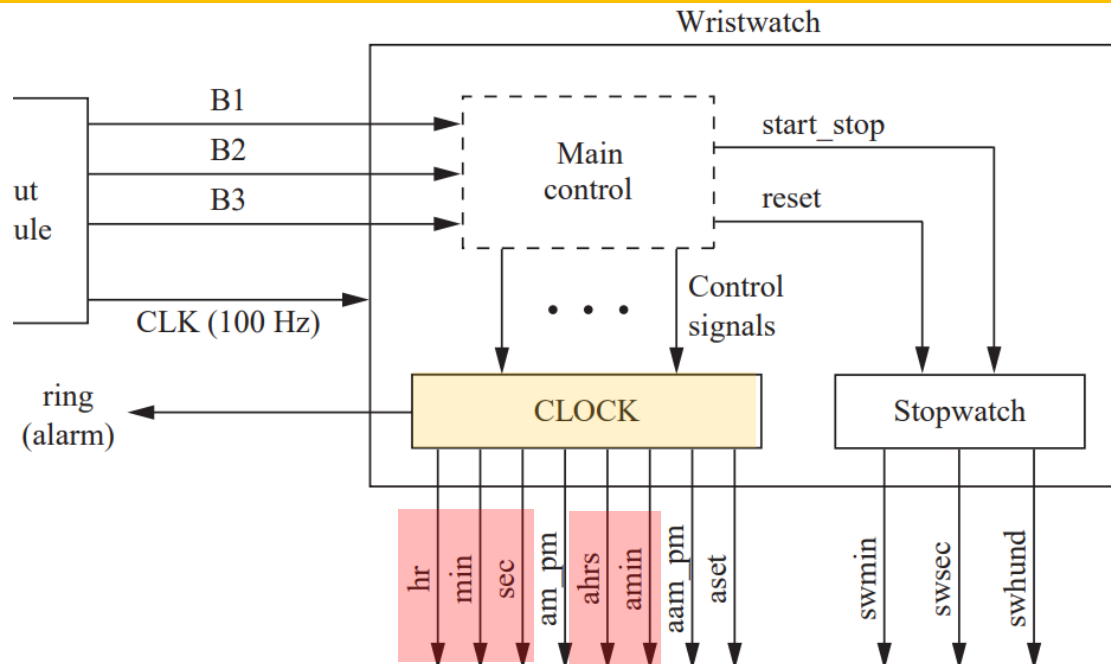


1.2 Design Implementation

Clock Module

```
library IEEE;
use IEEE.numeric_bit.all;

entity clock is
  port(clk, inch, incm, incha, incma, set_alarm, alarm_off: in bit;
       hours, ahours, minutes, aminutes, seconds: inout unsigned(7 downto 0);
       am_pm, aam_pm, ring, alarm_set: inout bit);
end clock;
```

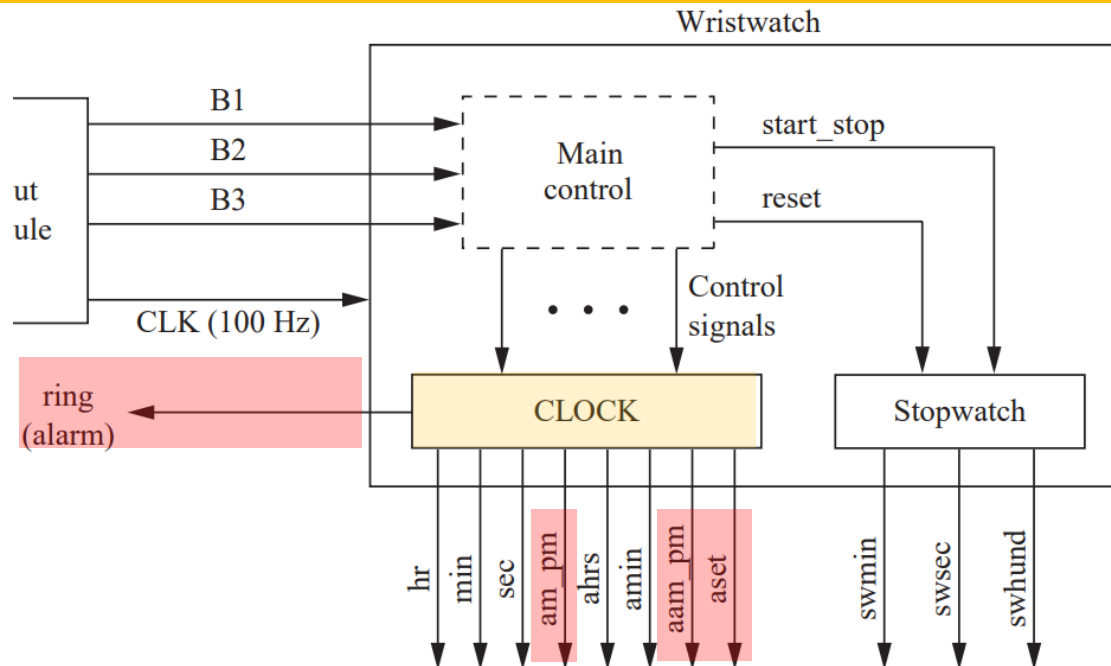


1.2 Design Implementation

Clock Module

```
library IEEE;
use IEEE.numeric_bit.all;

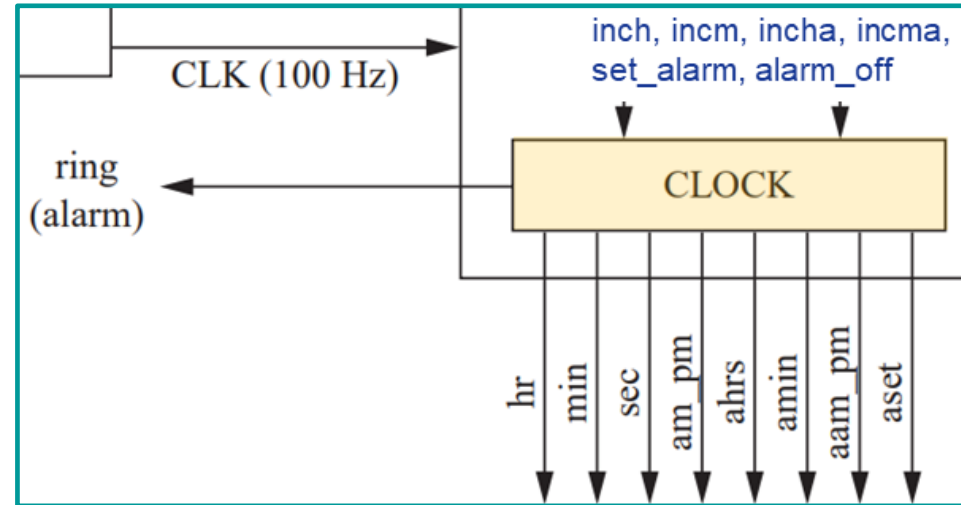
entity clock is
  port(clk, inch, incm, incha, incma, set_alarm, alarm_off: in bit;
        hours, ahours, minutes, aminutes, seconds: inout unsigned(7 downto 0);
        am_pm, aam_pm, ring, alarm_set: inout bit);
end clock;
```



1.2 Design Implementation

Clock Module

Counters 0~59, 1~12



architecture clock1 of clock is

component CTR_59 is

```
port(clk, inc, reset: in bit; dout: out unsigned(7 downto 0);
      t59: out bit);
```

end component;

component CTR_12 is

```
port(clk, inc: in bit; dout: out unsigned(7 downto 0); am_pm: inout bit);
end component;
```

```
signal s59, m59, inchr, incmin, c99: bit;
```

```
signal alarm_ring_time: integer range 0 to 50;
```

```
signal div100: integer range 0 to 99;
```

```
begin
```

```
... ..
```

```
end clock1;
```

sec, min

hrs, am_pm

1.2 Design Implementation

Clock Module

```
architecture clock1 of clock is
  component CTR_59 is
    port(clk, inc, reset: in bit; dout: out unsigned(7 downto 0);
          t59: out bit);
  end component;
  component CTR_12 is
    port(clk, inc: in bit; t12: out bit; am_pm: inout bit);
  end component;
  signal s59, m59, inchr, incmin, c99: bit;
  signal alarm_ring_time: integer range 0 to 50;
  signal div100: integer range 0 to 99;
begin
```

S59 = 1 when seconds = 59 ...

The alarm will ring for 50 s

-
-
-
-
-
-

1.2 Design Implementation

begin

```
sec1: ctr_59 port map(clk, c99, '0', seconds, s59);  
min1: ctr_59 port map(clk, incmin, '0', minutes, m59);  
hrs1: ctr_12 port map(clk, inchr, hours, am_pm);
```

Counters that
keep track of time

```
incmin <= (s59 and c99) or incm;  
inchr <= (m59 and s59 and c99) or inch;
```

```
alarm_min: ctr_59 port map(clk, incma, '0', aminutes, open);  
alarm_hr: ctr_12 port map(clk, incha, ahours, aam_pm);
```

```
c99 <= '1' when div100 = 99 else '0';
```

```
process(clk)
```

```
begin
```

```
  if clk'event and clk = '1' then
```

```
    if c99 = '1' then div100 <= 0;    -- divide by 100 counter
```

```
    else div100 <= div100 + 1;
```

```
    end if;
```

```
    if set_alarm = '1' then
```

```
      alarm_set <= not alarm_set;
```

```
    end if;
```

•

•

•

•

•

•

Counters for alarm

1.2 Design Implementation

c99 increments the
sec1 counter

begin

```
sec1: ctr_59 port map(clk, c99, '0', seconds, s59);  
min1: ctr_59 port map(clk, incmin, '0', minutes, m59);  
hrs1: ctr_12 port map(clk, inchr, hours, am_pm);  
incmin <= (s59 and c99) or incm;  
inchr <= (m59 and s59 and c99) or inch;  
alarm_min: ctr_59 port map(clk, incma, '0', aminutes, open);  
alarm_hr: ctr_12 port map(clk, incha, ahours, aam_pm);  
c99 <= '1' when div100 = 99 else '0';
```

process(clk)

begin

```
if clk'event and clk = '1' then  
  if c99 = '1' then div100 <= 0;  
  else div100 <= div100 + 1;  
  end if;  
  if set_alarm = '1' then  
    alarm_set <= not alarm_set;  
  end if;
```

When divide-by-100 counter is
99, it outputs a signal c99


-- divide by 100 counter

clk is 100Hz

1.2 Design Implementation

- Sec1 is a divide-by-60 counter
- When Sec1 reaches 59, it outputs s59


begin



```
sec1: ctr_59 port map(clk, c99, '0', seconds, s59);  
min1: ctr_59 port map(clk, incmin, '0', minutes, m59);  
hrs1: ctr_12 port map(clk, inchr, hours, am_pm);  
incmin <= (s59 and c99) or incm;  
inchr <= (m59 and s59 and c99) or inch;  
alarm_min: ctr_59 port map(clk, incma, '0', aminutes, open);  
alarm_hr: ctr_12 port map(clk, incha, ahours, aam_pm);  
c99 <= '1' when div100 = 99 else '0';
```

1.2 Design Implementation

begin

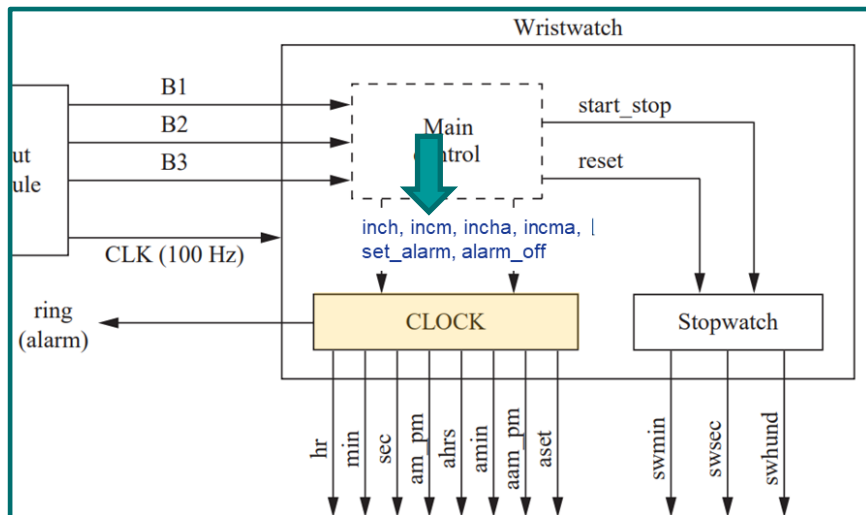


```
sec1: ctr_59 port map(clk, c99, '0', seconds, s59);
min1: ctr_59 port map(clk, incmin, '0', minutes, m59);
hrs1: ctr_12 port map(clk, inchr, hours, am_pm);
incmin <= (s59 and c99) or incm;
inchr <= (m59 and s59 and c99) or inch;
alarm_min: ctr_59 port map(clk, incma, '0', aminutes, open);
alarm_hr: ctr_12 port map(clk, incha, ahours, aam_pm);
c99 <= '1' when div100 = 99 else '0';
...
```

incmin is used to denote the condition when minutes has to be incremented

min1 +1


- when **s59** and **c99** are both 1, or
- when **incm = 1** in **set_minutes** state



1.2 Design Implementation

When **min1** reaches 59, it outputs a signal **m59**

begin



```
sec1: ctr_59 port map(clk, c99, '0', seconds, \;  
min1: ctr_59 port map(clk, incmin, '0', minutes, m59);  
hrs1: ctr_12 port map(clk, inchr, hours, am_pm);  
incmin <= (s59 and c99) or incm;  
inchr <= (m59 and s59 and c99) or inch;  
alarm_min: ctr_59 port map(clk, incma, '0', aminutes, open);  
alarm_hr: ctr_12 port map(clk, incha, ahours, aam_pm);  
c99 <= '1' when div100 = 99 else '0';  
...
```

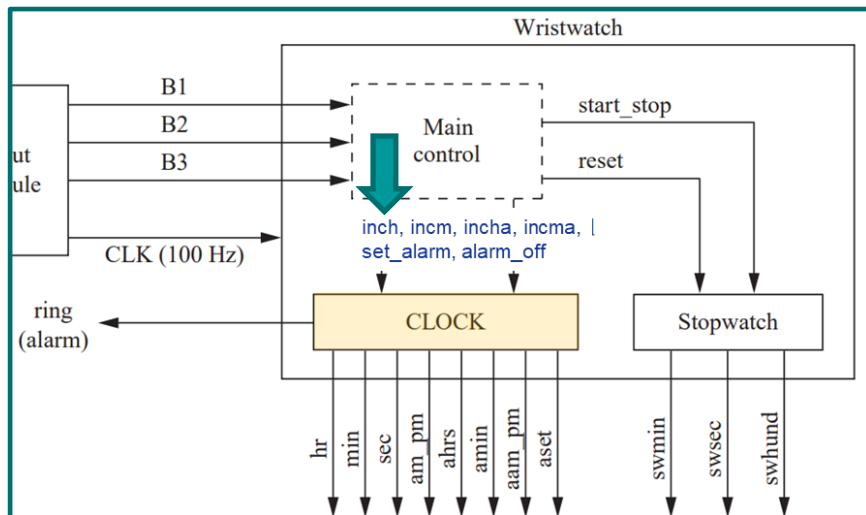

1.2 Design Implementation

begin

```
sec1: ctr_59 port map(clk, c99, '0', seconds, s59);  
min1: ctr_59 port map(clk, incmin, '0', minutes, m59);  
hrs1: ctr_12 port map(clk, inchr, hours, am_pm);  
incmin <= (s59 and c99) or incm;  
inchr <= (m59 and s59 and c99) or inch;  
alarm_min: ctr_59 port map(clk, incma, '0', aminutes, open);  
alarm_hr: ctr_12 port map(clk, incha, ahours, aam_pm);  
c99 <= '1' when div100 = 99 else '0';  
...
```

hrs1 +1

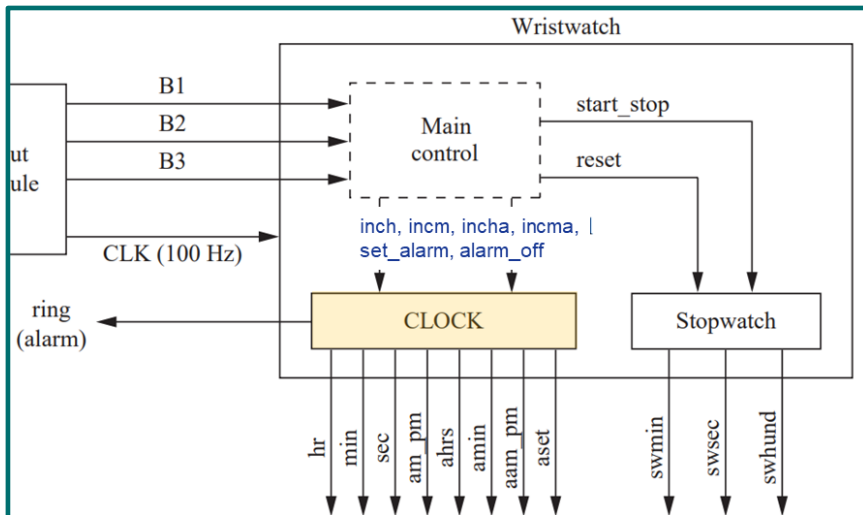
- when **m59 = s59 = c99 = 1**, or
- when **inch = 1** in **set_hours** state



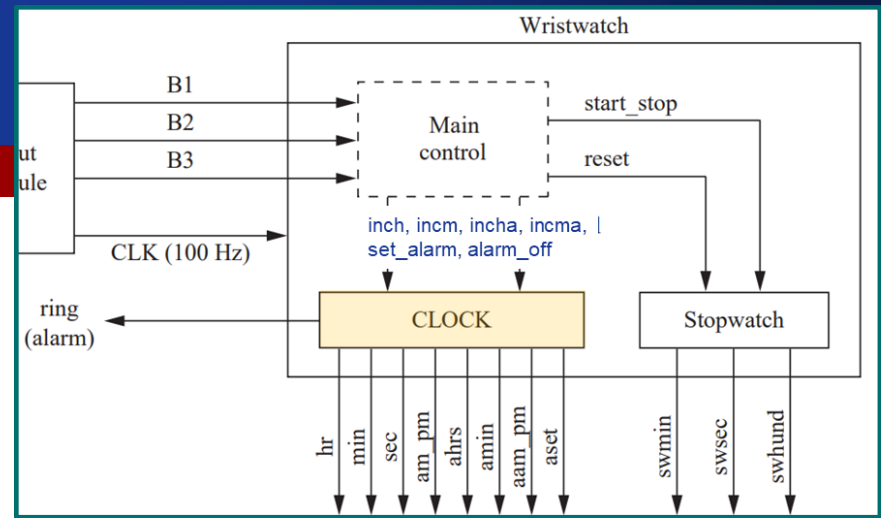
1.2 Design Implementation

begin

```
sec1: ctr_59 port map(clk, c99, '0', seconds, s59);
min1: ctr_59 port map(clk, incmin, '0', minutes, m59);
hrs1: ctr_12 port map(clk, inchr, hours, am_pm);
incmin <= (s59 and c99) or incm;
inchr <= (m59 and s59 and c99) or inch;
alarm_min: ctr_59 port map(clk, incma, '0', minutes, open);
alarm_hr: ctr_12 port map(clk, incha, hours, am_pm);
c99 <= '1' when div100 = 99 else '0';
```



- **hrs1** counts the hours
- It also toggles am_pm flip-flop when time changes from **11:59:59:99** to **12:00:00:00**



```
process(clk)
begin
```

```
  if clk'event and clk = '1' then
    if c99 = '1' then div100 <= 0;    -- divide by 100 counter
    else div100 <= div100 + 1;
    end if;
```

```
    if set_alarm = '1' then
      alarm_set <= not alarm_set;
    end if;
```

alarm_set flip-flop is toggled
when set_alarm = '1'

```
    if ((minutes = aminutes) and (hours = ahours) and (am_pm = aam_pm)) and
      seconds = 0 and alarm_set = '1' then
      ring <= '1';
```

```
    end if;
```

```
    if ring = '1' and c99 = '1' then
      alarm_ring_time <= alarm_ring_time + 1;
```

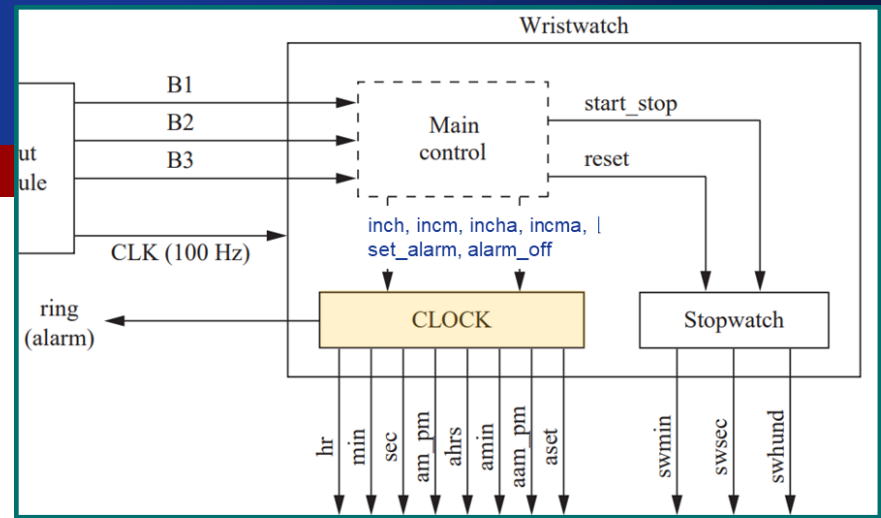
```
    end if;
```

```
    if alarm_ring_time = 50 or alarm_off = '1' then
      ring <= '0'; alarm_ring_time <= 0;
```

```
    end if;
```

```
  end if;
```

```
end process;
```



```
process(clk)
begin
```

```
  if clk'event and clk = '1' then
    if c99 = '1' then div100 <= 0;    -- divide by 100 counter
    else div100 <= div100 + 1;
    end if;
```

```
    if set_alarm = '1' then
      alarm_set <= not alarm_set;
    end if;
```

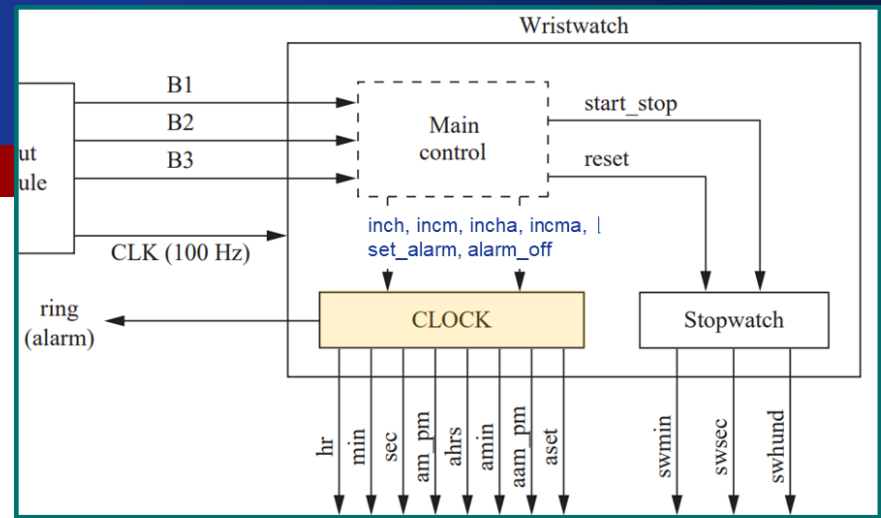
ring flip-flop is set to 1 when

- alarm setting matches the time counters and
- alarm is set

```
    if ((minutes = aminutes) and (hours = ahours) and (am_pm = aam_pm)) and
      seconds = 0 and alarm_set = '1' then
      ring <= '1';
    end if;
```

```
    if ring = '1' and c99 = '1' then
      alarm_ring_time <= alarm_ring_time + 1;
    end if;
    if alarm_ring_time = 50 or alarm_off = '1' then
      ring <= '0'; alarm_ring_time <= 0;
    end if;
```

```
  end if;
end process;
```



```
process(clk)
begin
```

```
  if clk'event and clk = '1' then
    if c99 = '1' then div100 <= 0;    -- divide by 100 counter
    else div100 <= div100 + 1;
    end if;
```

```
  end if;
```

```
  if set_alarm = '1' then
```

Alarm_ring_time counts seconds when the alarm is ringing

```
    and (am_pm = aam_pm)) and
```

```
    ring <= '1';
```

```
  end if;
```

```
  if ring = '1' and c99 = '1' then
```

```
    alarm_ring_time <= alarm_ring_time + 1;
```

```
  end if;
```

```
  if alarm_ring_time = 50 or alarm_off = '1' then
```

```
    ring <= '0'; alarm_ring_time <= 0;
```

```
  end if;
```

```
end if;
```

```
end process;
```

ring flip-flop is cleared

➤ after 50 seconds or

➤ when **alarm_off** signal is received

1.2 Design Implementation

Stopwatch Module

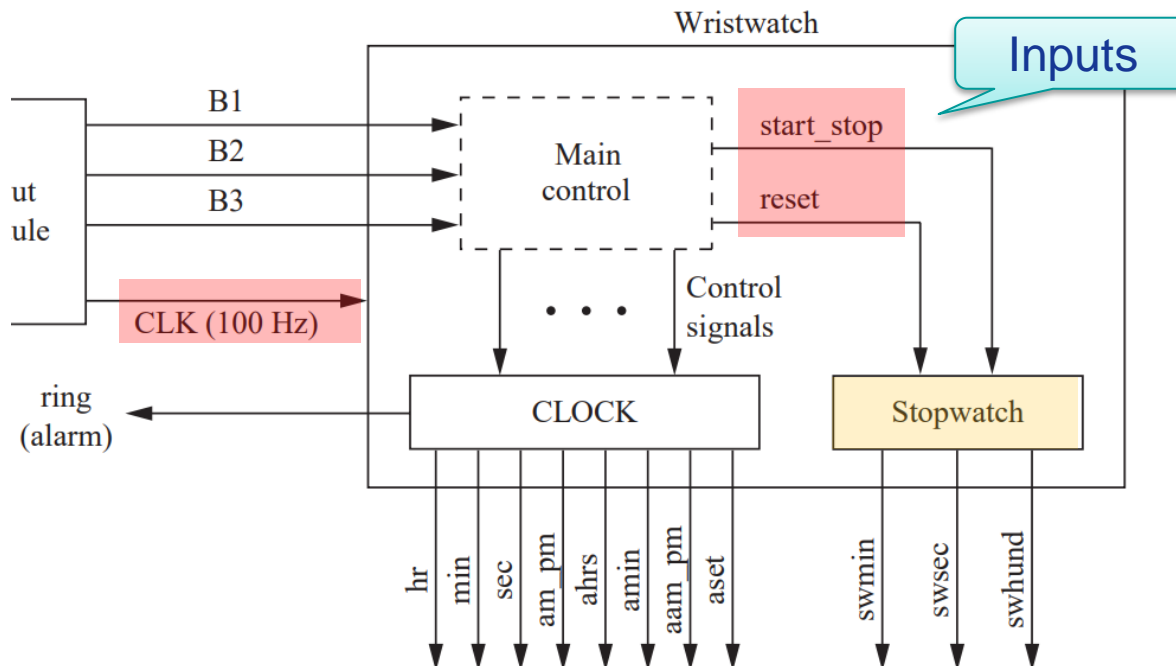
```
library IEEE;  
use IEEE.numeric_bit.all;
```

```
entity stopwatch is
```

```
  port(clk, reset, start_stop: in bit;
```

```
        swhundreths, swseconds, swminutes: out unsigned(7 downto 0));
```

```
end stopwatch;
```



1.2 Design Implementation

Stopwatch Module

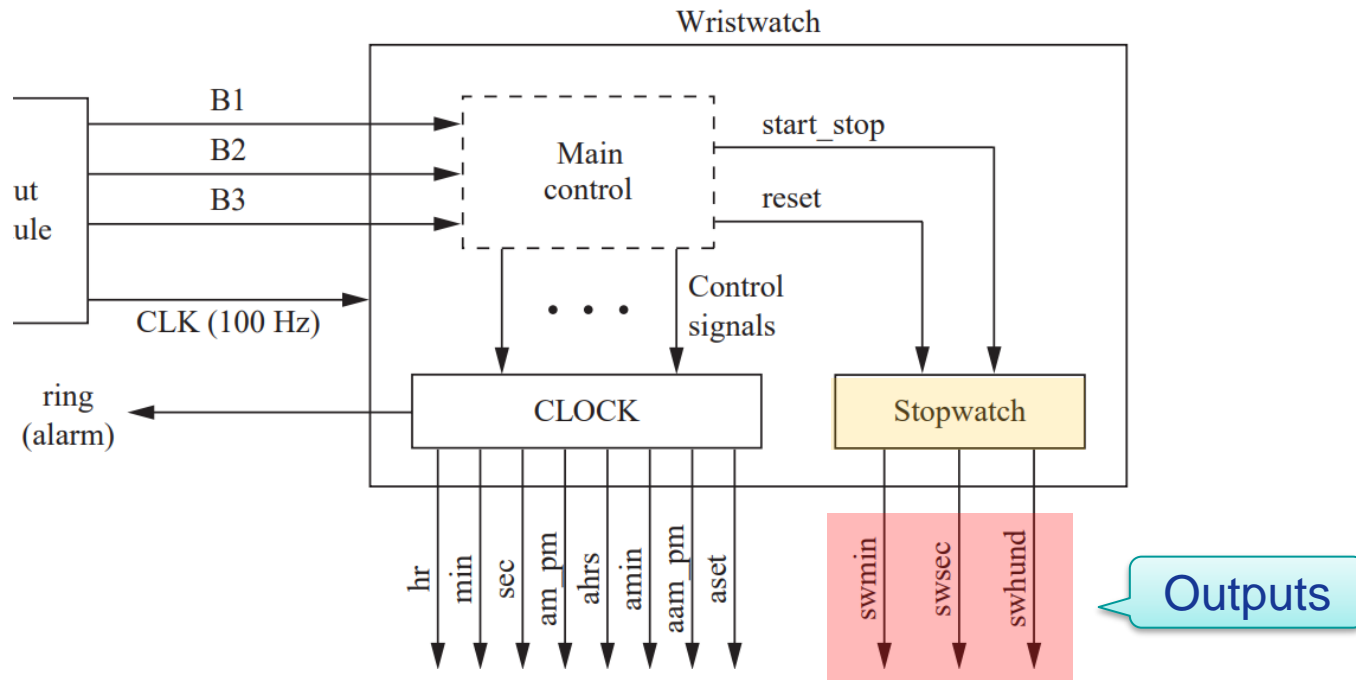
```
library IEEE;  
use IEEE.numeric_bit.all;
```

```
entity stopwatch is
```

```
    port(clk, reset, start_stop: in bit;
```

```
          swhundreths, swseconds, swminutes: out unsigned(7 downto 0));
```

```
end stopwatch;
```



1.2 Design Implementation

Stopwatch Module

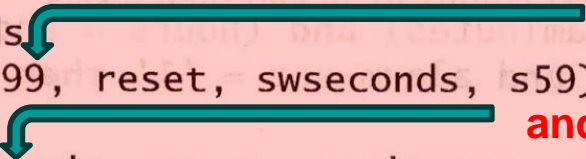
```
library IEEE;
use IEEE.numeric_bit.all;

entity stopwatch is
    port(clk, reset, start_stop: in bit;
          swhundreths, swseconds, swminutes: out unsigned(7 downto 0));
end stopwatch;

architecture stopwatch1 of stopwatch is
    component CTR_59 is
        port(clk, inc, reset: in bit; dout: out unsigned(7 downto 0); t59: out bit);
    end component;
    component CTR_99 is
        port(clk, inc, reset: in bit; dout: out unsigned(7 downto 0); t99: out bit);
    end component;
    signal swc99, s59, counting, swincmin: bit;
```


1.2 Design Implementation

```
begin
  ctr2: ctr_99 port map(clk, counting, reset, swhundreths, swc99);
  --counts hundreths of seconds
  sec2: ctr_59 port map(clk, swc99, reset, swseconds, s59);
  --counts seconds
  min2: ctr_59 port map(clk, swincmin, reset, swminutes, open);
  --counts minutes
  swincmin <= s59 and swc99;
  process(clk)
  begin
    if clk'event and clk = '1' then
      if start_stop = '1' then
        counting <= not counting;
      end if;
    end if;
  end process;
end stopwatch1;
```



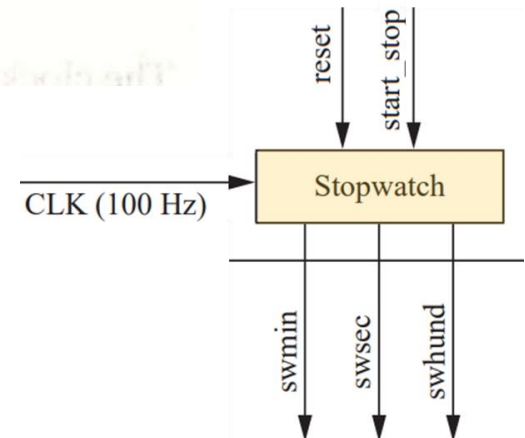
The idea is similar to **CLOCK** module

1.2 Design Implementation



```
begin
  ctr2: ctr_99 port map(clk, counting, reset, swhundreths, swc99);
  --counts hundreths of seconds
  sec2: ctr_59 port map(clk, swc99, reset, swseconds, s59);
  --counts seconds
  min2: ctr_59 port map(clk, swincmin, reset, swminutes, open);
  --counts minutes
  swincmin <= s59 and swc99;
  process(clk)
  begin
    if clk'event and clk = '1' then
      if start_stop = '1' then
        counting <= not counting;
      end if;
    end if;
  end process;
end stopwatch1;
```

When a **start_stop** signal is received, the counting flip-flop is toggled



1.2 Design Implementation

Divide-by-100 Counter

```
library IEEE;
use IEEE.numeric_bit.all;
--divide by 100 BCD counter
entity CTR_99 is
    port(clk, inc, reset: in bit; dout: out unsigned(7 downto 0); t99: out bit);
end CTR_99;

architecture count99 of CTR_99 is
    signal dig1, dig0: unsigned(3 downto 0);
begin
    process(clk)
    begin
        if clk'event and clk = '1' then
            if reset = '1' then dig0 <= "0000"; dig1 <= "0000";
            else
                if inc = '1' then
                    if dig0 = 9 then dig0 <= "0000";
                    if dig1 = 9 then dig1 <= "0000";
                    else dig1 <= dig1 + 1;
                    end if;
                else dig0 <= dig0 + 1;
                end if;
            end if;
        end if;
    end process;
    t99 <= '1' when (dig1 = 9 and dig0 = 9) else '0';
    dout <= dig1 & dig0;
end count99;
```

Two-digit BCD counter

1.2 Design Implementation

Divide-by-60 Counter

```
library IEEE;
use IEEE.numeric_bit.all;
--this counter counts seconds or minutes 0 to 59
entity CTR_59 is
    port(clk, inc, reset: in bit; dout: out unsigned(7 downto 0); t59: out bit);
end CTR_59;

architecture count59 of CTR_59 is
    signal dig1, dig0: unsigned(3 downto 0);
begin
    process(clk)
    begin
        if clk'event and clk = '1' then
            if reset = '1' then dig0 <= "0000"; dig1 <= "0000";
            else
                if inc = '1' then
                    if dig0 = 9 then dig0 <= "0000";
                    if dig1 = 5 then dig1 <= "0000";
                    else dig1 <= dig1 + '1';
                    end if;
                else dig0 <= dig0 + 1;
                end if;
            end if;
        end if;
    end process;
    t59 <= '1' when (dig1 = 5 and dig0 = 9) else '0';
    dout <= dig1 & dig0;
end count59;
```


1.2 Design Implementation

Hours Counter

```
library IEEE;
use IEEE.numeric_bit.all;
--this counter counts hours 1 to 12 and toggles am_pm
entity CTR_12 is
    port(clk, inc: in bit; dout: out unsigned(7 downto 0); am_pm: inout bit);
end CTR_12;

architecture count12 of CTR_12 is
    signal dig0: unsigned(3 downto 0);
    signal dig1: bit;
begin
    process(clk)
    begin
        if clk'event and clk = '1' then
            if inc = '1' then
                if dig1 = '1' and dig0 = 2 then
                    dig1 <= '0'; dig0 <= "0001";
                else
                    if dig0 = 9 then dig0 <= "0000"; dig1 <= '1';
                    else dig0 <= dig0 + 1;
                    end if;
                    if dig1 = '1' and dig0 = 1 then am_pm <= not am_pm;
                    end if;
                end if;
            end if;
        end if;
    end process;
    dout <= "000" & dig1 & dig0;
end count12;
```

Hours counter counts to 12 and then changes to 1 the next time inc = '1'

Hours counter toggles **am_pm** when the count changes from 11 to 12

1.3 Testing the Wristwatch

```
library IEEE;
use IEEE.numeric_bit.all;

entity testww is -- test bench for wristwatch
    port(hours, ahours, minutes, aminutes, seconds,
          swhundreths, swseconds, swminutes: inout unsigned(7 downto 0);
          am_pm, aam_pm, ring, alarm_set: inout bit);
end testww;

architecture testww1 of testww is
    component wristwatch is
        port(B1, B2, B3, clk: in bit;
             am_pm, aam_pm, ring, alarm_set: inout bit;
             hours, ahours, minutes, aminutes, seconds: inout unsigned(7 downto 0);
             swhundreths, swseconds, swminutes: out unsigned(7 downto 0));
    end component;
    signal B1, B2, B3, clk: bit;
```


Wristwatch as a component of **testww**

1.3 Testing the Wristwatch

```
library IEEE;
use IEEE.numeric_bit.all;

entity testww is -- test bench for wristwatch
  port(hours, ahours, minutes, aminutes, seconds,
        swhundreths, swseconds, swminutes: inout unsigned(7 downto 0);
        am_pm, aam_pm, ring, alarm_set: inout bit);
end testww;

architecture testww1 of testww is
  component wristwatch is
    port(B1, B2, B3, clk: in bit;
         am_pm, aam_pm, ring, alarm_set: inout bit;
         hours, ahours, minutes, aminutes, seconds: inout unsigned(7 downto 0);
         swhundreths, swseconds, swminutes: out unsigned(7 downto 0));
  end component;
  signal B1, B2, B3, clk: bit;
```




1.3 Testing the Wristwatch

```
library IEEE;
use IEEE.numeric_bit.all;

entity testww is -- test bench for wristwatch
  port(hours, ahours, minutes, aminutes, seconds,
        swhundreths, swseconds, swminutes: inout unsigned(7 downto 0);
        am_pm, aam_pm, ring, alarm_set: inout bit);
end testww;

architecture testww1 of testww is
  component wristwatch is
    port(B1, B2, B3, clk: in bit;
         am_pm, aam_pm, ring, alarm_set: inout bit;
         hours, ahours, minutes, aminutes, seconds: inout unsigned(7 downto 0);
         swhundreths, swseconds, swminutes: out unsigned(7 downto 0));
  end component;
  signal B1, B2, B3, clk: bit;
```




1.3 Testing the Wristwatch

```
library IEEE;
use IEEE.numeric_bit.all;

entity testww is -- test bench for wristwatch
    port(hours, ahours, minutes, aminutes, seconds,
          swhundreths, swseconds, swminutes: inout unsigned(7 downto 0);
          am_pm, aam_pm, ring, alarm_set: inout bit);
end testww;

architecture testww1 of testww is
    component wristwatch is
        port(B1, B2, B3, clk: in bit;
              am_pm, aam_pm, ring, alarm_set: inout bit;
              hours, ahours, minutes, aminutes, seconds: inout unsigned(7 downto 0);
              swhundreths, swseconds, swminutes: out unsigned(7 downto 0));
    end component;
    signal B1, B2, B3, clk: bit;
```



1.3 Testing the Wristwatch

```
library IEEE;
use IEEE.numeric_bit.all;

entity testww is -- test bench for wristwatch
    port(hours, ahours, minutes, aminutes, seconds,
          swhundreths, swseconds, swminutes: inout unsigned(7 downto 0);
          am_pm, aam_pm, ring, alarm_set: inout bit);
end testww;

architecture testww1 of testww is
    component wristwatch is
        port(B1, B2, B3, clk: in bit;
             am_pm, aam_pm, ring, alarm_set: inout bit;
             hours, ahours, minutes, aminutes, seconds: inout unsigned(7 downto 0);
             swhundreths, swseconds, swminutes: out unsigned(7 downto 0));
    end component;
    signal B1, B2, B3, clk: bit;
```

1.3 Testing the Wristwatch

begin

```
wristwatch1: wristwatch port map(B1, B2, B3, clk, am_pm, aam_pm, ring,  
                                alarm_set, hours, ahours, minutes, aminutes,  
                                seconds, swhundreths, swseconds, swminutes);  
clk <= not clk after 5 ms; -- generate 100 hz clock
```

process

procedure wait1 -- waits for N1 clocks

(N1: in integer)

variable count: integer;

begin

count := N1;

while count /= 0 **loop**

wait until clk'event and clk = '1';

count := count - 1;

wait until clk'event and clk = '0';


end loop;

end procedure wait1;

- Instantiation of wristwatch
- Generate a 100Hz clock

1.3 Testing the Wristwatch

```
begin
  wristwatch1: wristwatch port map(B1, B2, B3, clk, am_pm, aam_pm, ring,
    alarm_set, hours, ahours, minutes, aminutes,
    seconds, swhundreths, swseconds, swminutes);
  clk <= not clk after 5 ms; -- generate 100 hz clock
  process
    procedure wait1 -- waits for N1 clocks
      (N1: in integer) is
        variable count: integer;
      begin
        count := N1;
        while count /= 0 loop
          wait until clk'event and clk = '1';
          count := count - 1;
          wait until clk'event and clk = '0';
        end loop;
      end procedure wait1;
```



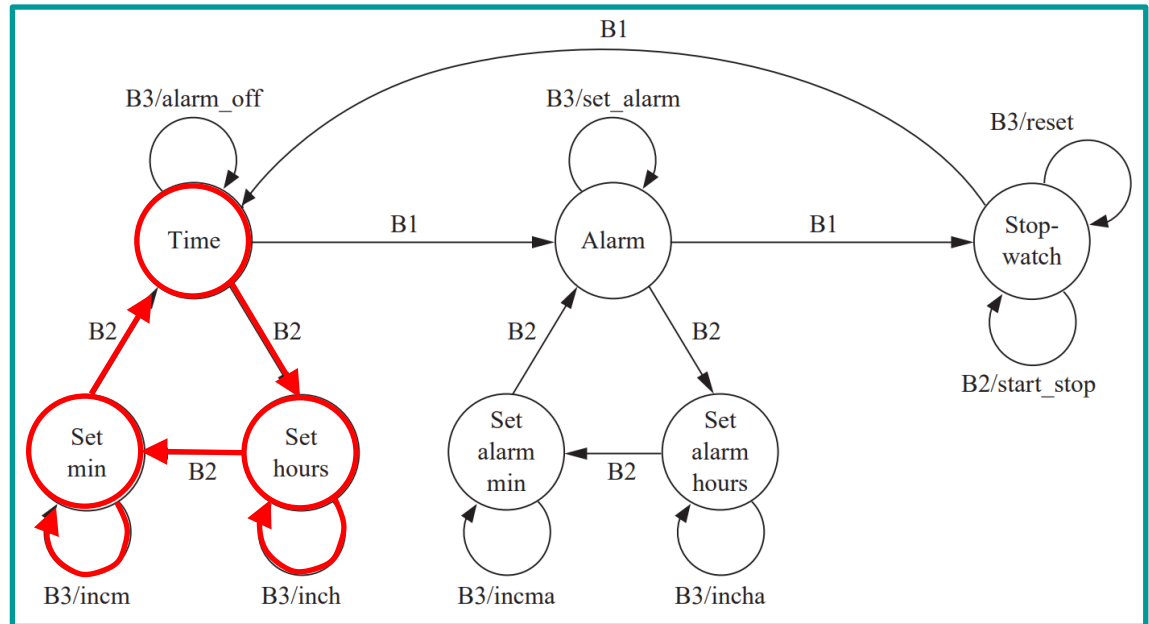
Procedure: wait for $N1 \cdot 10\text{ms}$

1.3 Testing the Wristwatch

```
procedure push    -- simulates pushing a button N times
  (signal button: out bit; N: in integer) is
begin
  for i in 1 to N loop
    button <= '1';
    wait1(1);
    button <= '0';
    wait1(120);    -- wait 1200 ms between pushes
  end loop;
end procedure push;
```

Procedure

1.3 Testing the Wristwatch

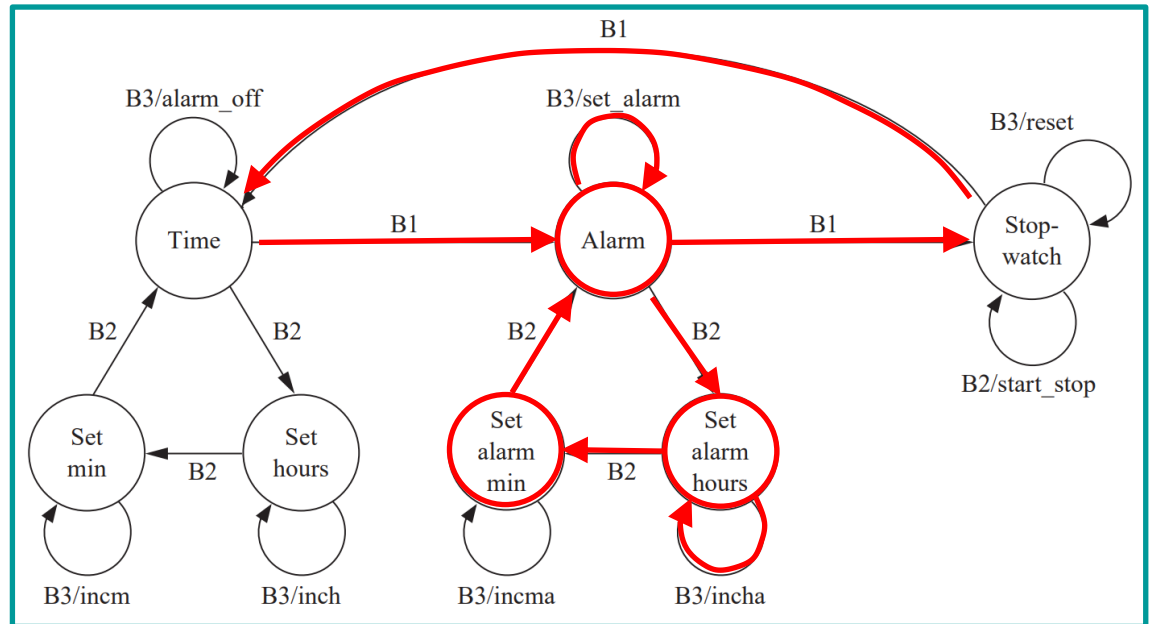


begin

57?

```
wait1(10);    -- set time to 11:58 pm
push(b2, 1);  push(b3, 23); push(b2, 1); push(b3, 57); push(b2, 1);
report "time should be 11:58 P.M.";
push(b1, 1);  -- set alarm to 12:00 am
push(b2, 1);  push(b3, 24); push(b2, 2); push(b3, 1); push(b1, 2);
report "alarm should be set to 12:00 A.M.";
```

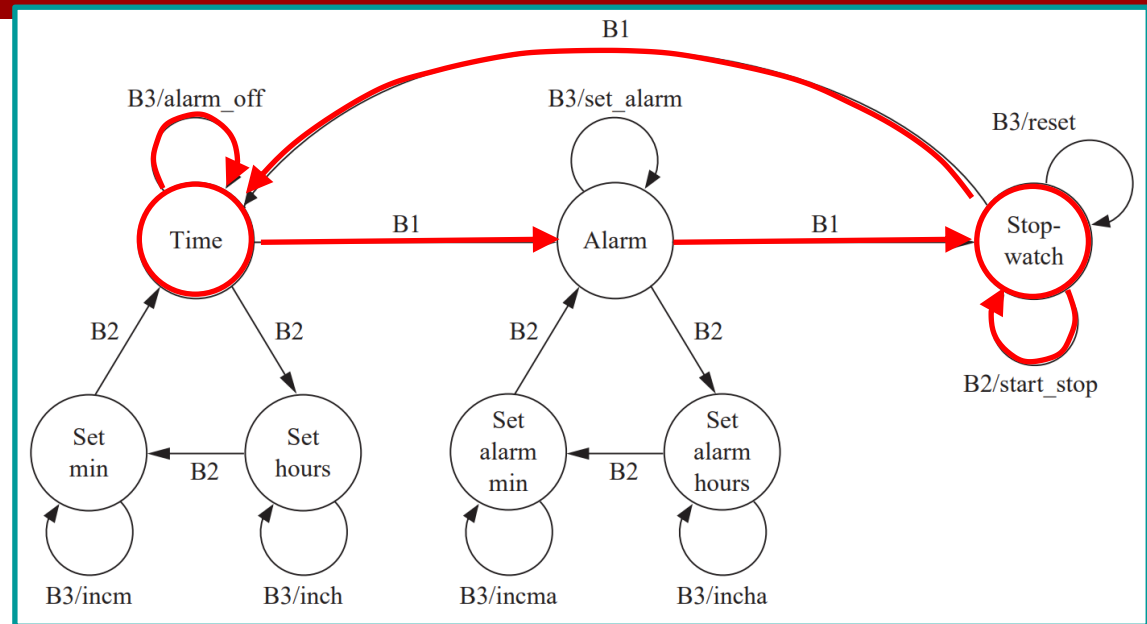
1.3 Testing the Wristwatch



begin

```
wait1(10);    -- set time to 11:58 pm
push(b2, 1);  push(b3, 23); push(b2, 1); push(b3, 57); push(b2, 1);
report "time should be 11:58 P.M.";
push(b1, 1);   -- set alarm to 12:00 am
push(b2, 1);  push(b3, 24); push(b2, 2); push(b3, 1); push(b1, 2);
report "alarm should be set to 12:00 A.M.";
```

1.3 Testing the Wristwatch

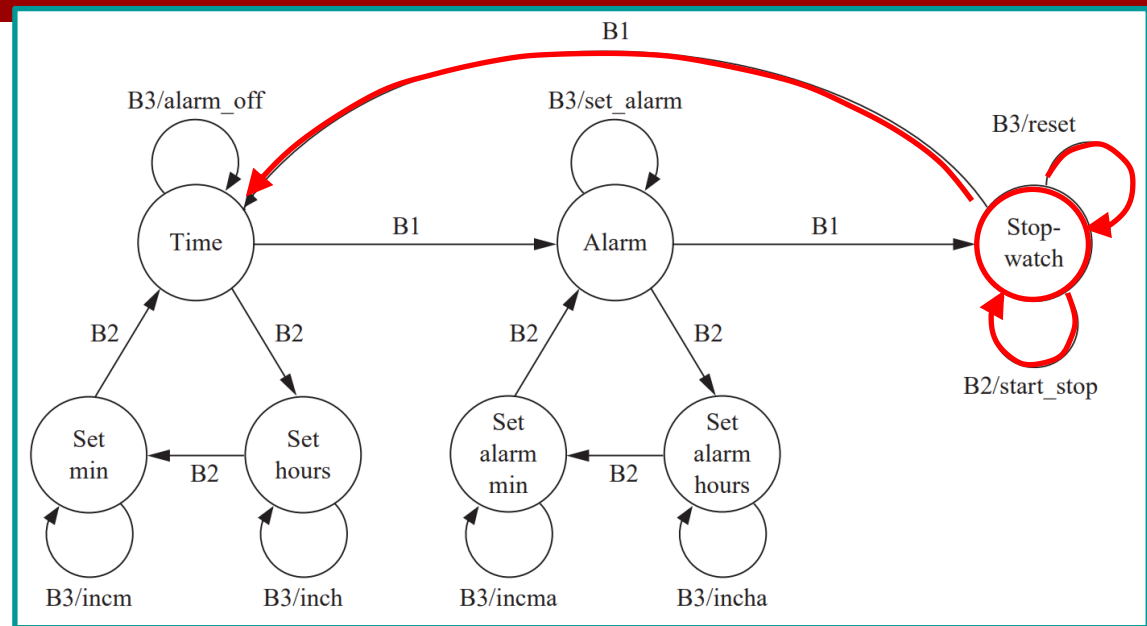


```

wait until hours = "00010010" and seconds = "00000101";
push(b3, 1);    -- turn alarm off at 12 hours and 5 seconds
push(b1, 2);    -- run stopwatch, go to time mode, go back to stopwatch
push(b2, 1); wait1(120); push(b1, 1); wait1(1000); push(b1, 2);
wait until swminutes = "00000001" and swseconds = "00000010";
    --stop stopwatch after 1 min. and 2 sec., then reset
report "stopwatch should read 1 min. 2 sec.";
push(b2, 1); push(b3, 1); push(b1, 1);
wait;
end process;
end testww1;
  
```

12:00:05

1.3 Testing the Wristwatch



```

wait until hours = "00010010" and seconds = "00000101";
push(b3, 1);    -- turn alarm off at 12 hours and 5 seconds
push(b1, 2);    -- run stopwatch, go to time mode, go back to stopwatch
push(b2, 1); wait1(120); push(b1, 1); wait1(1000); push(b1, 2);
wait until swminutes = "00000001" and swseconds = "00000010";
    --stop stopwatch after 1 min. and 2 sec., then reset
report "stopwatch should read 1 min. 2 sec.";
push(b2, 1); push(b3, 1); push(b1, 1);
wait;
end process;
end testww1;
    
```

stop stopwatch; reset stopwatch; go to time mode

1.3 Testing the Wristwatch

```
vsim -t 1ms testww -- set simulator resolution to 1 ms
add list -hex hours minutes seconds am_pm
          ahours aminutes aam_pm ring
add list b1 b2 b3 wristwatch1/state
add list -hex swminutes swseconds -notrigger swhundredths
run 300000 ms
```