REAL TIME SYSTEMS

Ecole Centrale de Nantes

LAB 2 REPORT

REPORT BY

Gopi Krishnan Regulan

Ragesh Ramachandran

Ankit Tripathi

December 2017

Goal:

The lab deals with how to trigger processing as a result of time passing (expiration of an Alarm). The concepts like alarm and counter are used. On the TP ECN board, the *SystemCounter* timer is used as interrupt source for alarms. The interrupt is sent every 1ms.

First Application:

The .cpp file and the .oil file of the first application is shown in the following figures. The task *button_scanner* reads the button when it is pushed. The task *t_process* has the priority 3 displays “processing triggered” on odd application and clears the LCD on even executions.

```
int n = 1;

TASK(button_scanner)
{
  ButtonState button = readButton();

  if (button == BUTTON_PUSH) {
    ActivateTask(t_process);
  }

  TerminateTask();
}

TASK(t_process)
{
  if (n%2 == 1) {
    lcd.setCursor(0,0);
    lcd.println("processing triggered");
  }
  else {
    lcd.clear();
  }
  n = n+1;
  TerminateTask();
}
```

Fig.1 First application .cpp file

```
TASK t_process {
      PRIORITY = 3;
      AUTOSTART = FALSE;
      ACTIVATION = 1;
      SCHEDULE = FULL;
};
```

Fig.2 First application .oil file

Second Application:

The second application will use 2 periodic tasks: t1 (priority 2, period 1s) and t2 (priority 1, period 1.5s). t1 toggles LED L0_3 each time it executes and t2 toggles LED L1_4.

Question 8:

Yes, it is periodic, the period is 3 s . Their periodic behavior is like Fig.3:

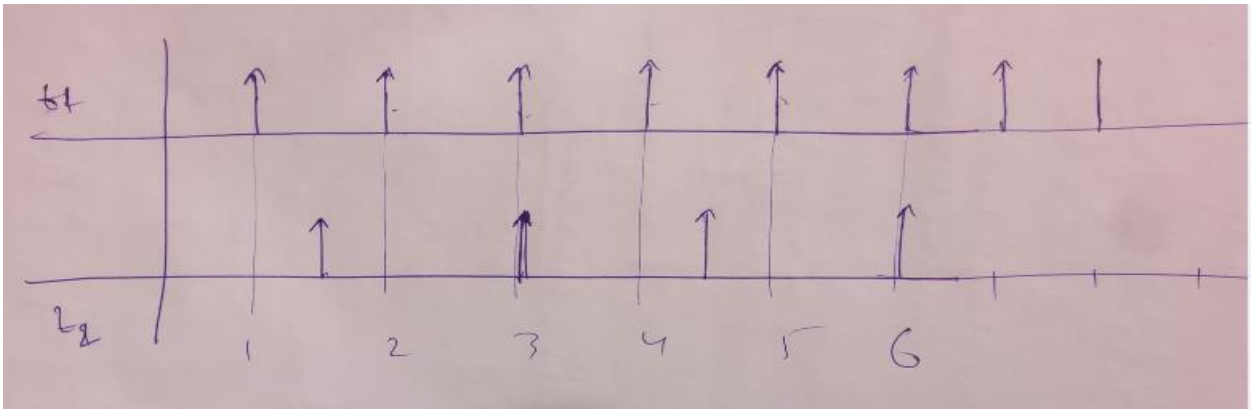

Fig. 3 The periodic behavior of the second application

Question 9:

The application needs a counter *SystemCounter* and 2 alarms *run_t1* and *run_t2*. The maximum *TICKSPERBASE* required is 500 (L.C.M. of 1000 ms, 1500 ms).

Question 10:

The alarms and the counter are configured and it is shown in figure.

```
DeclareTask(t1);
DeclareTask(t2);

TASK(t1)
{

  if (LOW == digitalRead(3)) {
    digitalWrite(3, HIGH);
  }
  else if (HIGH == digitalRead(3)) {
    digitalWrite(3, LOW);
  }

  TerminateTask();
}

TASK(t2)
{

 if (LOW == digitalRead(4)) {
    digitalWrite(4, HIGH);
  }
  else if (HIGH == digitalRead(4)) {
    digitalWrite(4, LOW);
  }
  TerminateTask();
}
```

Fig.4 Second application .cpp file

```
TASK t1 {
    PRIORITY = 2;
    AUTOSTART = TRUE { APPMODE = stdMode; };
    ACTIVATION = 1;
    SCHEDULE = FULL;
  };

  TASK t2 {
    PRIORITY = 1;
    AUTOSTART = TRUE { APPMODE = stdMode; };
    ACTIVATION = 1;
    SCHEDULE = FULL;
  };

  ALARM run_t1 {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
    AUTOSTART = TRUE {
      APPMODE = stdMode;
      ALARMTIME = 2;
   CYCLETIME = 2;

      };
  };

 ALARM run_t2 {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK {
      TASK = t2;
    };
    AUTOSTART = TRUE {
      APPMODE = stdMode;
      ALARMTIME = 3;
      CYCLETIME = 3;
    };
  };

COUNTER SystemCounter {
    MAXALLOWEDVALUE = 6;
    TICKSPERBASE = 500;
    MINCYCLE = 1;
    };
```

Fig.5 Second application .oil file

Third Application:

In the third application, alarms, counters and polling of the push buttons are mixed. This application is a system with 2 push buttons. After starting the system waits. When the button is pressed, the system start a function F that is implemented using a periodic task (period = 1s). To "see" F, uses a blinking LED as in the second application. When the button is pressed again, function F is stopped. When the switch is pressed, the system is shutdown as quickly as possible (ShutdownOS is called).

Question 11:

The application is designed and programmed as shown in figure.

```
TASK(button_scanner_eight)
{
 static int start_stop = 0;
 ButtonState button = readButton8();
 if (button == BUTTON_PUSH) {
    start_stop = start_stop + 1;
 }
 if ( start_stop%2 == 1 ) {
    SetRelAlarm(run_t1, 100,100);
 } else {
    CancelAlarm(run_t1);
 }
}

TASK(button_scanner_nine)
{
  ButtonState button = readButton9();
  if (button == BUTTON_PUSH) {
    digitalWrite(3, LOW);
    ShutdownOS(E_OK);
  }
}

TASK(t1)
{
  if (LOW == digitalRead(3)) {
    digitalWrite(3, HIGH);
  }
  else if (HIGH == digitalRead(3)) {
    digitalWrite(3, LOW);
  }
  TerminateTask();
}
```

Fig.6 Third application – Ques 11 .cpp file

```
  TASK button_scanner_eight {
    PRIORITY = 2;
    AUTOSTART = FALSE;
    ACTIVATION = 1;
    SCHEDULE = FULL;
  };

  ALARM run_button_scanner_eight {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK {
      TASK = button_scanner_eight;
    };
    AUTOSTART = TRUE {
      APPMODE = stdMode;
      ALARMTIME = 100;
      CYCLETIME = 100;
    };
  };

TASK button_scanner_nine {
    PRIORITY = 2;
    AUTOSTART = FALSE;
    ACTIVATION = 1;
    SCHEDULE = FULL;
  };

  ALARM run_button_scanner_nine {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK {
      TASK = button_scanner_nine;
    };
    AUTOSTART = TRUE {
      APPMODE = stdMode;
      ALARMTIME = 100;
      CYCLETIME = 100;
    };
  };

  TASK t1 {
    PRIORITY = 2;
    AUTOSTART = FALSE;
    ACTIVATION = 1;
    SCHEDULE = FULL;
  };

  ALARM run_t1 {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
    AUTOSTART = FALSE ;
  };
```

Fig.7 Third application – Ques 11 .oil file

Question 12:

Requirement changes. Now function F implementation needs an Init code (runs once when the F is started) and a Final code (runs once when F is stopped). There are three basic tasks *t1_init*, *t1_body* and *t1_final* to implement the function. The task *button_scanner_eight* read the button 8 and activates the tasks *t1_init* and *t1_final* according to the conditions. The task *t1_body* is activated by the alarm *run_t1_body*. The .cpp file and the .oil file is shown in figures.

```
TASK(button_scanner_eight)
{
 static int start_stop = 0;
 ButtonState button = readButton8();
 if (button == BUTTON_PUSH) {
    if( start_stop%2 == 0 ){
        ActivateTask(t1_init);
        SetRelAlarm(run_t1_body, 1000,1000);
    }else if( start_stop %2 == 1 ){
        ActivateTask(t1_final);
    }
    start_stop = start_stop + 1;
 }
}

TASK(t1_body)
{
  if (LOW == digitalRead(3))
  {
    digitalWrite(3, HIGH);
  }
  else if (HIGH == digitalRead(3))
  {
    digitalWrite(3, LOW);
  }
  TerminateTask();
}

TASK(t1_init)
{
    lcd.print(" InIt ");
    TerminateTask();
}

TASK(t1_final)
{
    lcd.print(" Final ");
    CancelAlarm(run_t1_body);
    TerminateTask();
}

TASK(button_scanner_nine)
{
  ButtonState button = readButton9();
  if (button == BUTTON_PUSH) {
    digitalWrite(3, LOW);
    lcd.clear();
    lcd.print(" shutting down");
    ShutdownOS(E_OK);
  }
```

Fig.8 Third application – Ques 12 .cpp file

```
TASK t1_body {
  PRIORITY = 2;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};

ALARM run_t1_body {
  COUNTER = SystemCounter;
  ACTION = ACTIVATETASK {
    TASK = t1_body;
  };
  AUTOSTART = FALSE ;
};

TASK t1_init {
  PRIORITY = 2;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};

TASK t1_final {
  PRIORITY = 2;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
```

Fig.9 Third application – Ques 12 .oil file

Question 13:

Only one extended task is used to implement function F. The task *t1* is an extended task and consists of three events. The task *t1* is periodic task activated from the .oil file. The task *button_scanner* gets activated when button P0.8 is pressed and sends *ev_0, ev_1* to task *t1*. Task *t1*, which till now was waiting for events, get the event and clears the event and activates corresponding tasks for the events received. If it is low, it writes high and vice versa. The task *button_scanner_nine* shutdowns the OS.

```
TASK(button_scanner)
{
 static int start_stop = 0;
 ButtonState button8 = readButton8();
 if (button8 == BUTTON_PUSH) {
    if( start_stop%2 == 0 ){
        SetEvent(t1, ev_0);
        SetEvent(t1, ev_1);
    }else {
        SetEvent(t1, ev_2);
    }
    start_stop = start_stop + 1;
 }
}

TASK(t1)
{
  while(1) {
    EventMaskType event_got;
    WaitEvent(ev_1 | ev_2 | ev_0);
    GetEvent(t1, &event_got);
    ClearEvent(event_got);
    if (event_got & ev_0) {
        lcd.print(" InIt ");
    }
    if (event_got & ev_1) {
        lcd.print(" body ");
        SetRelAlarm(run_t1_body, 1000,1000);
    }
    if (event_got & ev_2) {
        lcd.print(" Final ");
        CancelAlarm(run_t1_body);
    }
  }
}

TASK(button_scanner_nine)
{
  ButtonState button9 = readButton9();
  if (button9 == BUTTON_PUSH) {
    digitalWrite(3, LOW);
    lcd.clear();
    lcd.print(" shutting down");
    ShutdownOS(E_OK);
  }
}

TASK(t1_body)
{
  if (LOW == digitalRead(3)) {
    digitalWrite(3, HIGH);
  }
  else if (HIGH == digitalRead(3)) {
    digitalWrite(3, LOW);
  }
  TerminateTask();
}
```

Fig.10 Third application – Ques 11 .cpp file

```
TASK button_scanner_nine {
  PRIORITY = 2;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};

ALARM run_button_scanner_nine {
  COUNTER = SystemCounter;
  ACTION = ACTIVATETASK {
    TASK = button_scanner_nine;
  };
  AUTOSTART = TRUE {
    APPMODE = stdMode;
    ALARMTIME = 100;
    CYCLETIME = 100;
  };
};

TASK t1 {
  PRIORITY = 2;
  AUTOSTART = TRUE {
    APPMODE = stdMode;};
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = ev_0;
  EVENT = ev_1;
  EVENT = ev_2;
};

EVENT ev_0 {
  MASK = AUTO;
};
EVENT ev_1 {
  MASK = AUTO;
};
EVENT ev_2 {
  MASK = AUTO;
};

ALARM run_t1_body {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK {
      TASK = t1_body;
    };
    AUTOSTART = FALSE ;
};

TASK t1_body {
  PRIORITY = 2;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
```

Fig.11 Third application – Ques 11 .oil file

Fourth Application:

Question 14:

In this application, each time P0_8 is pressed, P1_9 must be pressed within 2 seconds. In such case, it prints the time between the two occurrences. Otherwise, an error message is displayed. If 2 or more P0 _8 are got within 3 seconds from the first one they are ignored.

```
TASK(button_scanner)
{
    ButtonState button8 = readButton8();
    if (button8 == BUTTON_PUSH) {
        TickType tick1;
        TickType tick2;
        if ( GetAlarm(alarm_t2, &tick2) == E_OS_NOFUNC ) {
            tick1 = 0;
        } else {
            GetAlarm(alarm_t2, &tick1);
        }
        if ( tick1 == 0 ) {
            SetRelAlarm(alarm_t1,2000,0);
            SetRelAlarm(alarm_t2,3000,0);
        }
    }
    ButtonState button9 = readButton9();
    if (button9 == BUTTON_PUSH) {
        TickType tick11;
        TickType tick21;
        if ( GetAlarm(alarm_t1, &tick11) != E_OS_NOFUNC ) {
            GetAlarm(alarm_t1, &tick21);
            tick21 = 2000 - tick11;
            lcd.clear();
            lcd.print(tick21);
            CancelAlarm(alarm_t1);
        }
    }
    TerminateTask();
}

TASK(t1) {
     lcd.clear();
     lcd.println("error");
     TerminateTask();
}

TASK(t2) {
     TerminateTask();
}
```

Fig.12 Fourth application – Question 14.cpp file

```
ALARM run_button_scanner {
  COUNTER = SystemCounter;
  ACTION = ACTIVATETASK {
    TASK = button_scanner;
  };
  AUTOSTART = TRUE {
    APPMODE = stdMode;
    ALARMTIME = 100;
    CYCLETIME = 100;
  };
};

TASK button_scanner {
  PRIORITY = 2;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};


ALARM alarm_t1{
  COUNTER = SystemCounter;
  ACTION = ACTIVATETASK {
    TASK = t1;
  };
  AUTOSTART = FALSE;
};

ALARM alarm_t2{
  COUNTER = SystemCounter;
  ACTION = ACTIVATETASK {
    TASK = t2;
  };
  AUTOSTART = FALSE;
};

TASK t1 {
  PRIORITY = 2;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};

TASK t2 {
  PRIORITY = 2;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
```

Fig.13 Fourth application – Question 14.oil file

Question 15:

The Gantt diagram of the scenario is shown in figure.

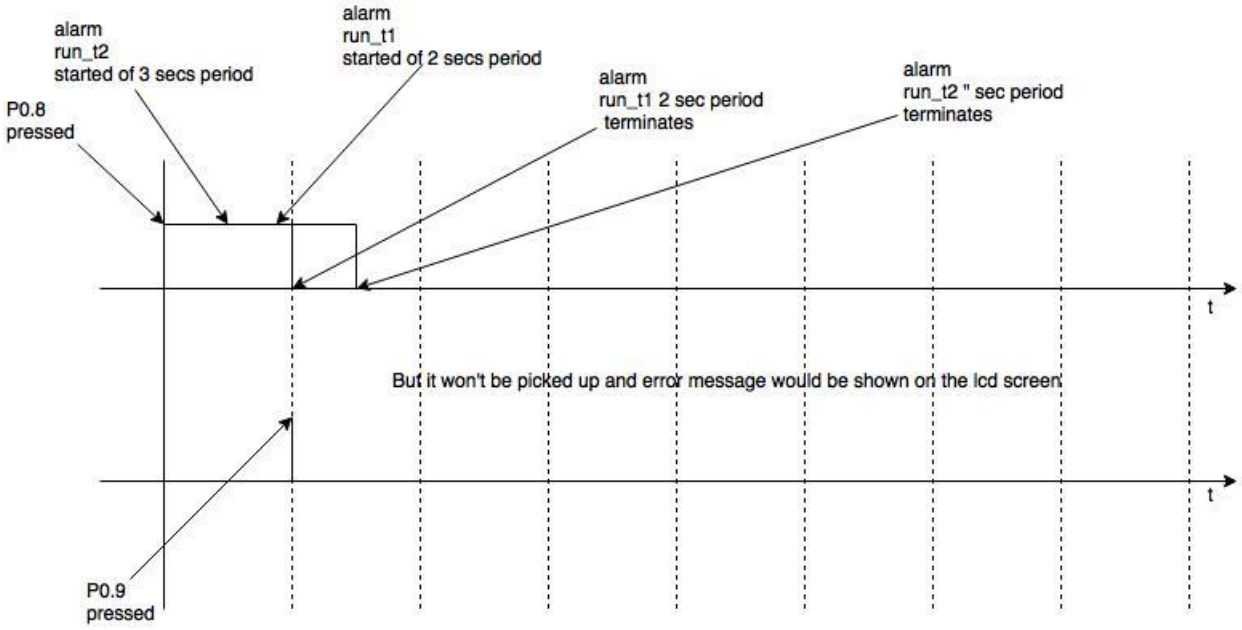

Fig. 14 Gantt diagram for question 15

Fifth application:

The code for the chasing is programmed and given in figure. In the .oil file 4 tasks are activated at the startup which does the chasing of the led's. button_scanner tasks continuously runs with a period of 1 ms. When any button is pushed the state of the current led's are stored and then taken the appropriate action. i.e. in case we want to stop the chase we just save the led states. In case we want to continue the chase we set the alarm again with a lagging time for all 4 led's using previously stored led states. In case we want to change direction of the chase we cancel previous alarms and set new alarms with respective time lag in opposite direction.

```
void continue_led_3(TickType i) {
    switch (i) {
        case 0:   led3=1;  SetRelAlarm(alarm1,100,500);break;
        case 1:   led3=0;  SetRelAlarm(alarm1,500,500);break;
        case 2:   led3=0;  SetRelAlarm(alarm1,400,500);break;
        case 3:   led3=0;  SetRelAlarm(alarm1,300,500);break;
        case 4:   led3=0;  SetRelAlarm(alarm1,200,500);break;
        case 5:   led3=0;  SetRelAlarm(alarm1,100,500);break;
        case 6:   led3=1;  SetRelAlarm(alarm1,500,500);break;
        case 7:   led3=1;  SetRelAlarm(alarm1,400,500);break;
        case 8:   led3=1;  SetRelAlarm(alarm1,300,500);break;
        case 9:   led3=1;  SetRelAlarm(alarm1,200,500);break;
    }
}

void continue_led_4(TickType i) {
    switch (i) {
        case 1:   led4=0;  SetRelAlarm(alarm2,500,500);break;
        case 2:   led4=0;  SetRelAlarm(alarm2,400,500);break;
        case 3:   led4=0;  SetRelAlarm(alarm2,300,500);break;
        case 4:   led4=0;  SetRelAlarm(alarm2,200,500);break;
        case 5:   led4=0;  SetRelAlarm(alarm2,100,500);break;
        case 6:   led4=1;  SetRelAlarm(alarm2,500,500);break;
        case 7:   led4=1;  SetRelAlarm(alarm2,400,500);break;
        case 8:   led4=1;  SetRelAlarm(alarm2,300,500);break;
        case 9:   led4=1;  SetRelAlarm(alarm2,200,500);break;
        case 0:   led4=1;  SetRelAlarm(alarm2,100,500);break;
    }
 }

 void continue_led_5(TickType i) {
    switch (i) {
        case 1:   led5=0;  SetRelAlarm(alarm3,500,500);break;
        case 2:   led5=0;  SetRelAlarm(alarm3,400,500);break;
        case 3:   led5=0;  SetRelAlarm(alarm3,300,500);break;
        case 4:   led5=0;  SetRelAlarm(alarm3,200,500);break;
        case 5:   led5=0;  SetRelAlarm(alarm3,100,500);break;
        case 6:   led5=1;  SetRelAlarm(alarm3,500,500);break;
        case 7:   led5=1;  SetRelAlarm(alarm3,400,500);break;
        case 8:   led5=1;  SetRelAlarm(alarm3,300,500);break;
        case 9:   led5=1;  SetRelAlarm(alarm3,200,500);break;
        case 0:   led5=1;  SetRelAlarm(alarm3,100,500);break;
    }
 }
```

```
void continue_led_6(TickType i) {
    switch (i) {
        case 1:   led6=0;  SetRelAlarm(alarm4,500,500);break;
        case 2:   led6=0;  SetRelAlarm(alarm4,400,500);break;
        case 3:   led6=0;  SetRelAlarm(alarm4,300,500);break;
        case 4:   led6=0;  SetRelAlarm(alarm4,200,500);break;
        case 5:   led6=0;  SetRelAlarm(alarm4,100,500);break;
        case 6:   led6=1;  SetRelAlarm(alarm4,500,500);break;
        case 7:   led6=1;  SetRelAlarm(alarm4,400,500);break;
        case 8:   led6=1;  SetRelAlarm(alarm4,300,500);break;
        case 9:   led6=1;  SetRelAlarm(alarm4,200,500);break;
        case 0:   led6=1;  SetRelAlarm(alarm4,100,500);break;
    }
}

TASK(t1) {
    if (led3%2 == 1) {
        digitalWrite(3, HIGH);
    } else {
        digitalWrite(3, LOW);
    }
    led3++;
    TerminateTask();
}
TASK(t2) {
    if (led4%2 == 1) {
        digitalWrite(4, HIGH);
    } else {
        digitalWrite(4, LOW);
    }
    led4++;
    TerminateTask();
}

TASK(t3) {
    if (led5%2 == 1) {
        digitalWrite(5, HIGH);
    } else {
        digitalWrite(5, LOW);
    }
    led5++;
    TerminateTask();
}

TASK(t4) {
    if (led6%2 == 1) {
        digitalWrite(6, HIGH);
    } else {
        digitalWrite(6, LOW);
    }
    led6++;
    TerminateTask();
}
```

```
TASK(button_scanner) {
    ButtonState button = readButton8();
    if (button == BUTTON_PUSH) {
        GetAlarm(alarm_save_led_state, &tick);
        tick=1200-tick;
        CancelAlarm(alarm1);
        CancelAlarm(alarm2);
        CancelAlarm(alarm3);
        CancelAlarm(alarm4);
    }
   ButtonState button2 = readButton9();
    if (button2 == BUTTON_PUSH) {
       if (chase_direction == 0) {
        continue_led_3(tick%10);
        continue_led_4((tick-1)%10);
        continue_led_5((tick-2)%10);
        continue_led_6((tick-3)%10);
       } else {
        continue_led_3(tick%10);
        continue_led_4((tick+1)%10);
        continue_led_5((tick+2)%10);
        continue_led_6((tick+3)%10);
       }
    }
    ButtonState button3 = readButton10();
    if (button3 == BUTTON_PUSH) {
        CancelAlarm(alarm1);
        CancelAlarm(alarm2);
        CancelAlarm(alarm3);
        CancelAlarm(alarm4);
        digitalWrite(3, LOW);
        digitalWrite(4, LOW);
        digitalWrite(5, LOW);
        digitalWrite(6, LOW);
        led3=0;
        led4=0;
        led5=0;
        led6=0;
        if (chase_direction == 0) {
            SetRelAlarm(alarm4,100,500);
            SetRelAlarm(alarm3,200,500);
            SetRelAlarm(alarm2,300,500);
            SetRelAlarm(alarm1,400,500);
            chase_direction = 1;
        } else{
            SetRelAlarm(alarm1,100,500);
            SetRelAlarm(alarm2,200,500);
            SetRelAlarm(alarm3,300,500);
            SetRelAlarm(alarm4,400,500);
            chase_direction = 0;
        }
    }
}
```

Fig.15 Fifth Application .cpp file

```
ALARM alarm1 {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
    AUTOSTART = TRUE {
      APPMODE = stdMode;
      ALARMTIME = 100;
      CYCLETIME = 500;
    };
};

ALARM alarm2 {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK {
      TASK = t2;
    };
    AUTOSTART = TRUE {
      APPMODE = stdMode;
      ALARMTIME = 200;
      CYCLETIME = 500;
    };
};

ALARM alarm3 {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK {
      TASK = t3;
    };
    AUTOSTART = TRUE {
      APPMODE = stdMode;
      ALARMTIME = 300;
      CYCLETIME = 500;
    };
};

ALARM alarm4 {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK {
      TASK = t4;
    };
    AUTOSTART = TRUE {
      APPMODE = stdMode;
      ALARMTIME = 400;
      CYCLETIME = 500;
    };
};

TASK t1 {
        PRIORITY = 2;
        AUTOSTART = FALSE;
        ACTIVATION = 1;
        SCHEDULE = FULL;
};
```

```
TASK t2 {
        PRIORITY = 2;
        AUTOSTART = FALSE;
        ACTIVATION = 1;
        SCHEDULE = FULL;
};

TASK t3 {
        PRIORITY = 2;
        AUTOSTART = FALSE;
        ACTIVATION = 1;
        SCHEDULE = FULL;
};

TASK t4{
        PRIORITY = 2;
        AUTOSTART = FALSE;
        ACTIVATION = 1;
        SCHEDULE = FULL;
};

TASK button_scanner {
    PRIORITY = 2;
    AUTOSTART = FALSE;
    ACTIVATION = 1;
    SCHEDULE = FULL;
};

ALARM run_button_scanner {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK {
      TASK = button_scanner;
    };
    AUTOSTART = TRUE {
      APPMODE = stdMode;
      ALARMTIME = 1;
      CYCLETIME = 1;
    };
};

TASK save_led_state {
        PRIORITY = 2;
        AUTOSTART = FALSE;
        ACTIVATION = 1;
        SCHEDULE = FULL;
};

ALARM alarm_save_led_state {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK {
      TASK = save_led_state;
    };
    AUTOSTART = TRUE {
      APPMODE = stdMode;
      ALARMTIME = 100;
      CYCLETIME = 10000;
    };
```

Fig.16 Fifth Application .oil file