POLITECNICO DI TORINO

# OSES LAB #1 – task and alarms

Department of CONTROL AND COMPUTER ENGINEERING (DAUIN)
Master Degree in Mechatronic Engineering
2023/2024
Operating systems for embedded systems– Prof. Violante
Laboratory 02
Wednesday, 16 November, 2023

Students :
s320296 DI MARTINO GIULIO

## Exercise #1

In the exercise 1 we shall set three task C that samples a switch S connected to GPIO pin 12 and analog input voltage A0 every 100ms; this task sends an int messages to task M encodes as Bits 0... 9 represent the value of A0 expressed as an integer in the range [0, 1024[. Bit 12, if set, indicates the switch S has been pressed continuously for at least 1s. The task M runs every 500ms and perform the following action.

- If the switch S has been pressed continuously for at least 1s, the current value of A0 becomes the new reference value R.
- If no reference value has been set yet, the LED must be on continuously.
- If a reference value R has been set and the absolute value X of the difference between the current value of A0 and R is less than 100, the LED must be off.
- If X is at least 100, but less than 200, the LED must blink slowly.
- If X is 200 or more, the LED must blink fast.

The .oil code is the follow :
lab01_ex01.oil

```
OIL_VERSION = "2.5" : "test";



CPU test {
  OS config {
    STATUS = STANDARD;
    BUILD = TRUE {
      TRAMPOLINE_BASE_PATH = "/Users/giuliodimartino/Desktop/trampoline";
      APP_NAME = "lab01";
      APP_SRC = "lab02.cpp";
      CPPCOMPILER = "avr-g++";
      COMPILER = "avr-gcc";
      LINKER = "avr-gcc";
      ASSEMBLER = "avr-gcc";
      COPIER = "avr-objcopy";
      SYSTEM = PYTHON;
      LIBRARY = serial;
    };
    SYSTEM_CALL = TRUE;
  };

  APPMODE stdAppmode {};

MESSAGE Message_send_taskC {
  MESSAGEPROPERTY = SEND_STATIC_INTERNAL{
     CDATATYPE = "int";
  };
};


  MESSAGE Message_send_taskM {
  MESSAGEPROPERTY = SEND_STATIC_INTERNAL{
     CDATATYPE = "int";
  };
};


  MESSAGE Message_rec_taskM{
    MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL{
      SENDINGMESSAGE = Message_send_taskC;
    };
  };


  MESSAGE Message_rec_taskV{
    MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL {
      SENDINGMESSAGE = Message_send_taskM;
    };
  };



  ALARM a100msec {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK { TASK = TaskC; };
    AUTOSTART = TRUE { APPMODE = stdAppmode; ALARMTIME =98 ; CYCLETIME =98; };
  };

  ALARM a500msec {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK { TASK = TaskM; };
    AUTOSTART = TRUE { APPMODE = stdAppmode; ALARMTIME = 488; CYCLETIME = 488; };
  };

 ALARM aLED
  {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK{TASK = TaskV; };
    AUTOSTART = TRUE{APPMODE = stdAppmode;ALARMTIME = 122;CYCLETIME = 122; };
  };


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

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

  TASK TaskV {
    PRIORITY = 3;
    AUTOSTART = TRUE { APPMODE = stdAppmode; };
    ACTIVATION = 1;
    SCHEDULE = FULL;
  };

  };
```

The oil code is composed by:

```
OIL_VERSION = "2.5" : "test";

CPU test {
  OS config {
    STATUS = STANDARD;
    BUILD = TRUE {
      TRAMPOLINE_BASE_PATH = "/Users/giuliodimartino/Desktop/trampoline";
      APP_NAME = "lab01";
      APP_SRC = "lab02.cpp";
      CPPCOMPILER = "avr-g++";
      COMPILER = "avr-gcc";
      LINKER = "avr-gcc";
      ASSEMBLER = "avr-gcc";
      COPIER = "avr-objcopy";
      SYSTEM = PYTHON;
      LIBRARY = serial;
    };
```

This part of the code is useful because we define all the specification that are needed for the program.

TASK DEFINE:

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

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

  TASK TaskV {
    PRIORITY = 3;
    AUTOSTART = TRUE { APPMODE = stdAppmode; };
    ACTIVATION = 1;
    SCHEDULE = FULL;
  };
```

The task V has PRIORITY = 3, this is bigger respect to the task C and task M(greater it is the number, greater is the priority). That because the task V that is the task that turn on or off the led is important and it shall be executed before the other tasks; instead, the task C that samples the signal from ArduinoUno is more important than the task M (because task C sends the message to the task M)
AUTOSTART =TRUE so the Task V is placed in the ready list upon started the Operating System, if AUTOSTART = FALSE the task isn't placed in the ready list, so it needs an external system call.
ACTIVATION = 1 means that we all the task in the ready list; SCHEDULE = FULL means the scheduler is fully preemptible.
If the AUTOSTART = FALSE means that the task is not instantiate at the beginning of the execution of the operating system, so the task is not in the ready list, so it needs an external system call.

ALARM DEFINE:

```
ALARM a100msec {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK { TASK = TaskC; };
    AUTOSTART = TRUE { APPMODE = stdAppmode; ALARMTIME =98 ; CYCLETIME =98; };
  };
```

```
ALARM a500msec {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK { TASK = TaskM; };
    AUTOSTART = TRUE { APPMODE = stdAppmode; ALARMTIME = 488; CYCLETIME = 488; };
  };
```

```
ALARM aLED
  {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK{TASK = TaskV;};
    AUTOSTART = TRUE{APPMODE = stdAppmode;ALARMTIME = 122;CYCLETIME = 122;};
  };
```

The time in the alarm instruction is in this configuration because a 16Mhz Arduino board performs a tick every 1024 µs, to have the same task execution time we had to modify the ALARMTIME and CYCLETIME with the following proportion:

$$1024\colon 1000 = time_{desired}\colon X$$

so, we find an alarmtime of 98 and uncycletime of 98 for task C, an alarmtime of 488 and uncycletime of 488 for task M and to set the Led we set another alarm (aLED) that perform the task V with a frequency of 4 Hz.

MESSAGE DEFINE:

```
MESSAGE Message_send_taskC {
  MESSAGEPROPERTY = SEND_STATIC_INTERNAL{
     CDATATYPE = "uint16";
  };
};


  MESSAGE Message_send_taskM {
  MESSAGEPROPERTY = SEND_STATIC_INTERNAL{
     CDATATYPE = "uint16";
  };
};


  MESSAGE Message_rec_taskM{
    MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL{
      SENDINGMESSAGE = Message_send_taskC;
    };
  };


  MESSAGE Message_rec_taskV{
    MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL {
      SENDINGMESSAGE = Message_send_taskM;
    };
  };
```

In this part of the code, we set two messages: from the task C to the task M, from task M to task V. I set all the message to UNQUEUED type: only the last value is stored. Each new value overrides the previous one.

## The C code

```
#include <time.h>
#include <stdio.h>
#include <math.h>
#include "tpl_os.h"
#include "Arduino.h"
#include "tpl_com.h"


DeclareAlarm(a500msec);
DeclareAlarm(a100msec);
DeclareAlarm(aLED);

DeclareMessage(Message_send_taskC);
DeclareMessage(Message_rec_taskM);
DeclareMessage(Message_send_taskM);
DeclareMessage(Message_rec_taskV);

void setup()
{
    StartOS(OSDEFAULTAPPMODE);
    pinMode(13, OUTPUT);
    pinMode(12, INPUT_PULLUP);
    Serial.begin(115200);
}

TASK(TaskC)
{

    static int numero=0;
    static int output=0;
    static int contatore=0;

    numero = analogRead(A0);
    if(digitalRead(12)== HIGH){
        contatore++;
    }else if (digitalRead(12)== LOW){
        contatore=0;
    }

    if(contatore<10){

        output=numero;// switch off
    }else{

        output=numero+4096;
        contatore=0;
    }

        SendMessage(Message_send_taskC,&output);


    TerminateTask();
}

TASK(TaskM){
    static int count =0;
    static int input=0;
    static int R=0;
    static int X=0;
    static int messaggio=3;

    if(ReceiveMessage(Message_rec_taskM, &input) != E_OK) {
      Serial.print("Task M: Message from C is not received\n");
    }

    if (input>=1024){
        R=input-4096;
        count=1;
    }
    else if(count==1){
        X=abs(R-input);
        if (X<100) {
            messaggio=0;
        }else if(X>=100 && X<200){
            messaggio=1;
        }else if(X>200){
            messaggio=2;
        }
    }


    SendMessage(Message_send_taskM, &messaggio);

    Serial.print("Voltage = ");
    Serial.print(input);
    Serial.print("\t R = ");
    Serial.print(R);
    Serial.print("\t mode = ");
    Serial.println(messaggio);



    TerminateTask();
}


TASK(TaskV)
{
    static unsigned int blink = 0;
    static int data = 0;

 if(ReceiveMessage(Message_rec_taskV, &data) != E_OK) {
      Serial.print("Task V: Message from M not received\n");
    }
    if(data == 0) {
      digitalWrite(13, LOW);
    } else if(data == 1) {
      blink++;
      if (blink%4 == 0){
        if(blink%8 == 0) digitalWrite(13, LOW); //odd
        else digitalWrite(13, HIGH);        //even
      }
    } else if(data == 2) {
      blink++;
      if(blink & 1) digitalWrite(13, HIGH); //odd
      else digitalWrite(13, LOW);        //even
    } else if(data == 3) {
      digitalWrite(13, HIGH);
    }
    fflush(stdout);
    TerminateTask();
}
```

TASK C:

```
TASK(TaskC)
{

    static int numero=0;
    static int output=0;
    static int contatore=0;

    numero = analogRead(A0);
    if(digitalRead(12)== HIGH){
        contatore++;
    }else if (digitalRead(12)== LOW){
        contatore=0;
    }

    if(contatore<10){

        output=numero;
    }else{
        S=4096; // value of the Switch
        output=numero+S;
        contatore=0;
    }

        SendMessage(Message_send_taskC,&output);


    TerminateTask();
}
```

In the task C I check if the switch is on or off; if the switch is on I add 4096 (that is the 12th bit ) to the value in bit of the voltage from the voltage generator; to see if the Switch is on or off I define the digitalRead and a counter that count if the switch is on at least for 1 sec at the end I send the output to another task with SendMessage command.

TASK M:

```
TASK(TaskM){
    static int count =0;
    static int input=0;
    static int R=0;
    static int X=0;
    static int messaggio=3;

    if(ReceiveMessage(Message_rec_taskM, &input) != E_OK) {
      Serial.print("Task M: Message from C is not received\n");
    }

    if (input>=1024){
        R=input-4096;
        count=1;
    }
    else if(count==1){
        X=abs(R-input);
        if (X<100) {
            messaggio=0;
        }else if(X>=100 && X<200){
            messaggio=1;
        }else if(X>200){
            messaggio=2;
        }
    }


    SendMessage(Message_send_taskM, &messaggio);

    Serial.print("Output_taskC = ");
    Serial.print(input);
    Serial.print("\t R = ");
    Serial.print(R);
    Serial.print("\t mode = ");
    Serial.println(messaggio);



    TerminateTask();
}
```

In this task I check if the message was send correctly and follow the instruction of the lab I define same parameter like R and X after that I send the message:" messaggio" to the task V . in the end of this task I print some parameter to see if the code works properly.

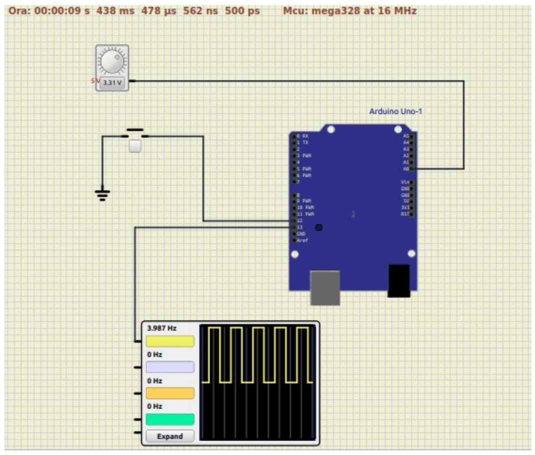
```
TASK(TaskV)
{
    static unsigned int blink = 0;
    static int data = 0;

 if(ReceiveMessage(Message_rec_taskV, &data) != E_OK) {
      Serial.print("Task V: Message from M not received\n");
    }
    if(data == 0) {
      digitalWrite(13, LOW);
    } else if(data == 1) {
      blink++;
      if((blink/4)%2 == 0)
            digitalWrite(13, HIGH);
      else
            digitalWrite(13, LOW);

    } else if(data == 2) {
      blink++;
      if(blink & 1)
            digitalWrite(13, HIGH);
      else
            digitalWrite(13, LOW);
    } else if(data == 3) {
      digitalWrite(13, HIGH);
    }
    fflush(stdout);
    TerminateTask();
}
```

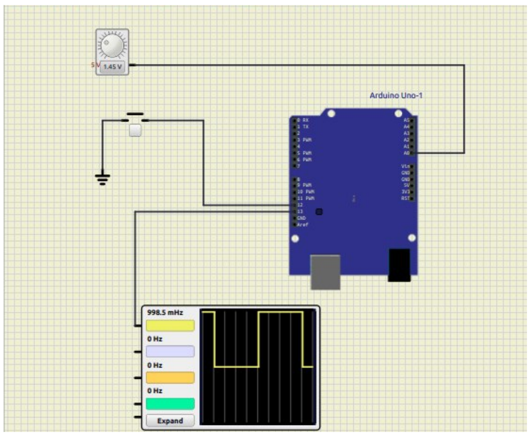
The last task is the taskV in I use this task to set the LED in the oil file I set an alarm of 4 Hz for this task but

- if the data =1 the led shall blink slowly (1 Hz); I divide the time into 4 and I check if blink variable is odd or even in this way I obtain the 1 Hz, the blink variable increase at each iterations.
- If data=2 so the Led shall blink at 4 Hz: the variable blink is incremented at each run, and then the bitwise AND operator (&) with the number 1 (which in binary is 00000001)

determines if the last bit of blink is 1 or 0. If it's 1, blink & 1 will return true and the LED will turn on (digitalWrite(13, HIGH)); otherwise, if it's 0, the LED will turn off (digitalWrite(13, LOW)).

I generate the lab01.hex, putting this file into SimulIDE I obtain the turn on and off of the pin 13, with different behavior in different conditions.





Calculate the worst-case latency between a variation of A0 and the corresponding change of state of the LED. To do that I calculate with Simulide I set the time very slow to see when the frequency change it behavior; I calculate that the latency is almost 595 ms that is the defference between when I change the voltage and I see a change on the oscilloscope

## Exercise #2

If we have that M is activate only when there is a message for it to precess. The oil file change because there isn't the periodic task so there is not the alarm a500 ms :

```
OIL_VERSION = "2.5" : "test";



CPU test {
  OS config {
    STATUS = STANDARD;
    BUILD = TRUE {
      TRAMPOLINE_BASE_PATH = "/Users/giuliodimartino/Desktop/trampoline";
      APP_NAME = "lab01";
      APP_SRC = "lab02.cpp";
      CPPCOMPILER = "avr-g++";
      COMPILER = "avr-gcc";
      LINKER = "avr-gcc";
      ASSEMBLER = "avr-gcc";
      COPIER = "avr-objcopy";
      SYSTEM = PYTHON;
      LIBRARY = serial;
    };
    SYSTEM_CALL = TRUE;
  };

  APPMODE stdAppmode {};

MESSAGE Message_send_taskC {
  MESSAGEPROPERTY = SEND_STATIC_INTERNAL{
     CDATATYPE = "uint16";
  NOTIFICATION=ACTIVATETASK{
       TASK = TaskV;
       };
  };
};


  MESSAGE Message_send_taskM {
  MESSAGEPROPERTY = SEND_STATIC_INTERNAL{
     CDATATYPE = "uint16";
  };
};


  MESSAGE Message_rec_taskM{
    MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL{
      SENDINGMESSAGE = Message_send_taskC;
    };
  };


  MESSAGE Message_rec_taskV{
    MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL {
      SENDINGMESSAGE = Message_send_taskM;
    };
  };



  ALARM a100msec {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK { TASK = TaskC; };
    AUTOSTART = TRUE { APPMODE = stdAppmode; ALARMTIME =98 ; CYCLETIME =98; };
  };

 ALARM aLED
  {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK{TASK = TaskV; };
    AUTOSTART = TRUE{APPMODE = stdAppmode;ALARMTIME = 122;CYCLETIME = 122; };
  };


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

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

  TASK TaskV {
    PRIORITY = 3;
    AUTOSTART = TRUE { APPMODE = stdAppmode; };
    ACTIVATION = 1;
    SCHEDULE = FULL;
  };

  };
```

In this case the latency is almost 250. I repeat the same process that make in the exercise 1. The latency in less than before because we don’t have the alarm in the task M so the task C read the input and immediately send that message to task M that process the message and send it to the task V that turn on and off the led. In this case else we don’t miss any data in the process.