POLITECNICO DI TORINO

OSES LAB #4 – resources and task organization

Department of CONTROL AND COMPUTER ENGINEERING (DAUIN)
Master’s degree in Mechatronic Engineering
2023/2024
Operating systems for embedded systems– Prof. Violante
Laboratory 04
Wednesday, 22 December 2023

Student:
s320296 DI MARTINO GIULIO

Exercise #1

A periodic task S samples analog input voltage A0 every 100ms. On every activation, S inserts the sampled value X into a queue Q, which can hold at most K elements. S must check for queue overflows and report them by writing a message on the serial console. Moreover, S must set the variable *int error* to *1* if X<10 or X>1013, and to *0* otherwise.

A periodic task B runs every 500ms. On every activation, B offloads all samples currently in Q and calculates their minimum N and maximum M. B must set the variable *int alarm* to *1* if M- N > 500, and to *0* otherwise.

A periodic task V runs every 125ms. It must control the LED connected to GPIO pin 13 according to the value of *error* and *alarm*, as follows:

- If *error* is *1*, the LED must blink “fast” (4Hz).
- If *error* is 0 and *alarm* is 1, the LED must blink “slowly” (1Hz).
- If both *error* and *alarm* are *0*, the LED must be off.

C-Code

```
#include "tpl_os.h"
#include "Arduino.h"
#include "tpl_com.h"

DeclareAlarm(a125msec);
DeclareAlarm(a500msec);
DeclareAlarm(a100msec);

#define K 5
int circularBuffer[K];
static int itemCount = 0;
static int alarm=0;
static int error=0;
#define Res 1

void setup()
{
  pinMode(A0,INPUT);
  Serial.begin(115200);

}

TASK(TaskS) {
  static int sensorValue;
  Serial.print("\n[taskS]\n");
  Serial.print("itemCount: ");
  Serial.print(itemCount);
  Serial.print("\t");

  sensorValue = analogRead(A0);

  if(itemCount < K){
    #ifdef Res
      GetResource(Res);
    #endif
    circularBuffer[itemCount] = sensorValue;
    Serial.print("A0: ");
    Serial.print(sensorValue);
    Serial.print("\t");
    itemCount++;
    if(sensorValue<10 || sensorValue>1013){
       error=1;
    }
    else{
       error=0;
    }
    Serial.print("error_instant:");
    Serial.print(error);
    Serial.print("\n");
    #ifdef Res
      ReleaseResource(Res);
    #endif
  }
  else{
    Serial.print("error_buffer");
  }

}

TASK(TaskB)
{

  static int i;
  static int M=0;
  static int N=0;
  int size;


  Serial.print("[taskB]\n");
  size=itemCount;

  M=circularBuffer[0]; // massimo
  N=circularBuffer[0]; // minimo

  Serial.print("\nMax_value");
  Serial.print(M);
  Serial.print("\t");
  Serial.print("Min_value");
  Serial.print(N);

  for(i=0;i<size;i++){
  #ifdef Res
    GetResource(Res);
  #endif
    Serial.print("\ndata:");
    Serial.print(circularBuffer[i]);
    if (circularBuffer[i]>M)
    {
      M=circularBuffer[i];
    }
    if (circularBuffer[i]<=N)
    {
      N=circularBuffer[i];
    }
    itemCount--;
    Serial.print("\n count:");
    Serial.print(itemCount);
    Serial.print("\nMax_value");
    Serial.print(M);
    Serial.print("\t");
    Serial.print("Min_value");
    Serial.print(N);
  }

  Serial.print("\nGeneral: Max_value");
  Serial.print(M);
  Serial.print("\t");
  Serial.print("Min_value");
  Serial.print(N);
  Serial.print("\n");
  Serial.print("item:");
  Serial.print(itemCount);



  if (M-N>500)
    alarm=0;
  else
```

```
    alarm=1;

  Serial.print("\talarm: ");
  Serial.print(alarm);

  #ifdef Res
    ReleaseResource(Res);
  #endif
}

TASK(TaskV)
{
  Serial.print("\n[taskV]\n");

  static unsigned int blink = 0;
  Serial.print("error:");
  Serial.print(error);
  Serial.print("\t");
  Serial.print("alarm:");
  Serial.print(alarm);
  Serial.print("\n");
  #ifdef Res
    GetResource(Res);
  #endif
  if(alarm == 1 && error == 0 ) {
    Serial.print("LOW\t");
    blink++;
    if(blink & 1) digitalWrite(13, HIGH); //odd
    else digitalWrite(13, LOW);        //even

  }
  else if(error==0 && alarm==0) {
    Serial.print("OFF\t");
    digitalWrite(13, LOW);
  }
  else if(error == 1) {
    Serial.print("FAST\t");
    blink++;
    if (blink%4 == 0){
      if(blink%8 == 0) digitalWrite(13, HIGH); //odd
      else digitalWrite(13, LOW);        //even
    }
  }
 #ifdef Res
   ReleaseResource(Res);
 #endif
  Serial.print("\n");
}
```

I set a resource when the there is a shared variable like `circularBuffer` for example in the task S: in this way there is any lost of data

```
TASK(TaskS) {
  static int sensorValue;
  Serial.print("\n[taskS]\n");
  Serial.print("itemCount: ");
  Serial.print(itemCount);
  Serial.print("\t");

  sensorValue = analogRead(A0);

  if(itemCount < K){
    #ifdef Res
      GetResource(Res);
    #endif
    circularBuffer[itemCount] = sensorValue;
    Serial.print("A0: ");
    Serial.print(sensorValue);
    Serial.print("\t");
    itemCount++;



    if(sensorValue<10 || sensorValue>1013){
       error=1;
    }
    else{
       error=0;
    }
    Serial.print("error_instant:");
    Serial.print(error);
    Serial.print("\n");
    #ifdef Res
      ReleaseResource(Res);
    #endif
  }
  else{
    Serial.print("error_buffer");
  }

}
```

In this code the shared variables between the Task are: `circularBuffer; itemCount; alarm; error;` these variables are set as general variable.

OIL_FILE:

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

CPU test {
  OS config {
    STATUS = STANDARD;
    BUILD = TRUE {
      TRAMPOLINE_BASE_PATH = "../../../..";
      APP_NAME = "lab4";
      APP_SRC = "lab4.cpp";
      CPPCOMPILER = "avr-g++";
      COMPILER = "avr-gcc";
      LINKER = "avr-gcc";
      ASSEMBLER = "avr-gcc";
      COPIER = "avr-objcopy";
      SYSTEM = PYTHON;
      LIBRARY = serial;

    };

  };

  APPMODE stdAppmode {};

  RESOURCE Res{
    RESOURCEPROPERTY = STANDARD;
  };



 ALARM a100sec {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK { TASK = TaskS; };
    AUTOSTART = TRUE { APPMODE = stdAppmode; ALARMTIME = 98; CYCLETIME = 98; APPMODE = stdAppmode;
};
  };

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

 ALARM a125msec{
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK{TASK = TaskV;};
    AUTOSTART = TRUE{APPMODE = stdAppmode; ALARMTIME = 122;CYCLETIME = 122; APPMODE = stdAppmode;};
  };

  TASK TaskS {
    PRIORITY = 3;
    AUTOSTART = TRUE {APPMODE = stdAppmode; };
    ACTIVATION = 1;
    SCHEDULE = FULL;
    RESOURCE = Res;
  };

 TASK TaskV {
    PRIORITY = 2;
    AUTOSTART = FALSE;
    ACTIVATION = 1;
    SCHEDULE = FULL;
    RESOURCE = Res;
  };


TASK TaskB {
    PRIORITY = 1;
    AUTOSTART = FALSE;
    ACTIVATION = 1;
    SCHEDULE = FULL;
    RESOURCE = Res;
  };

};
```

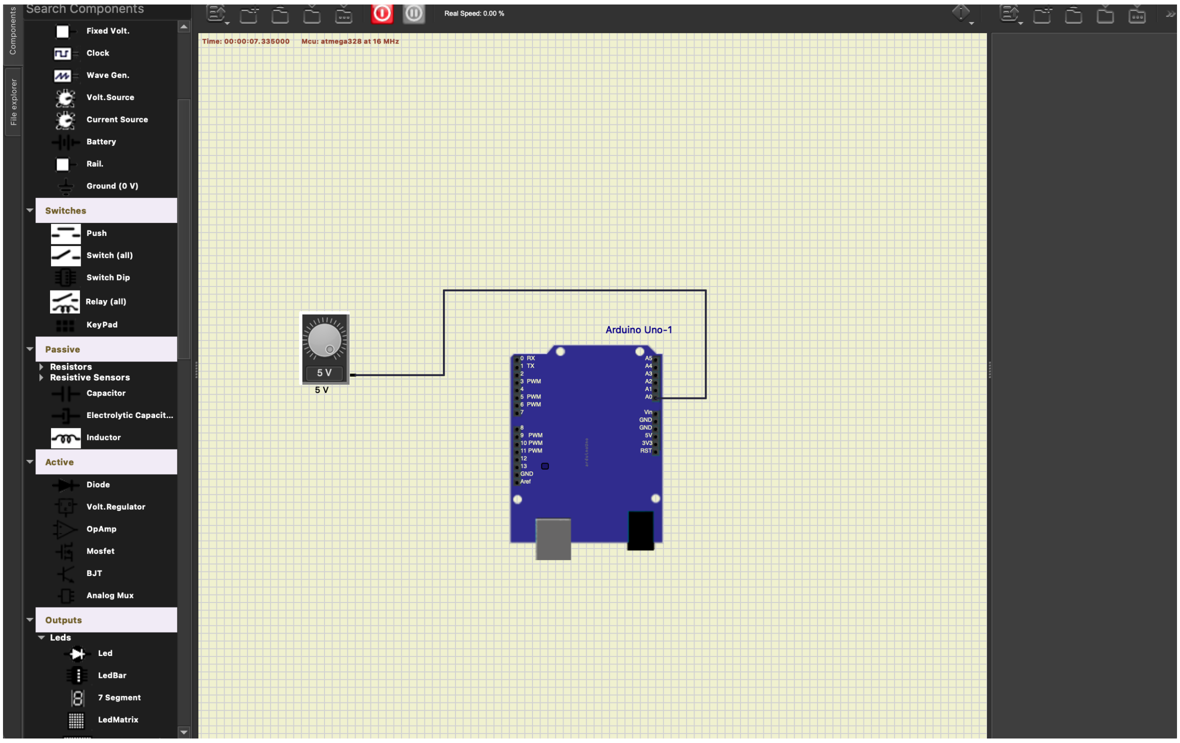
I set the three tasks in this way: task S with priority 3 because it has the less execution time; than the task V and with the less priority the task B.

Only the task S start at the initial time (`AUTOSTART = TRUE {APPMODE = stdAppmode; };`)

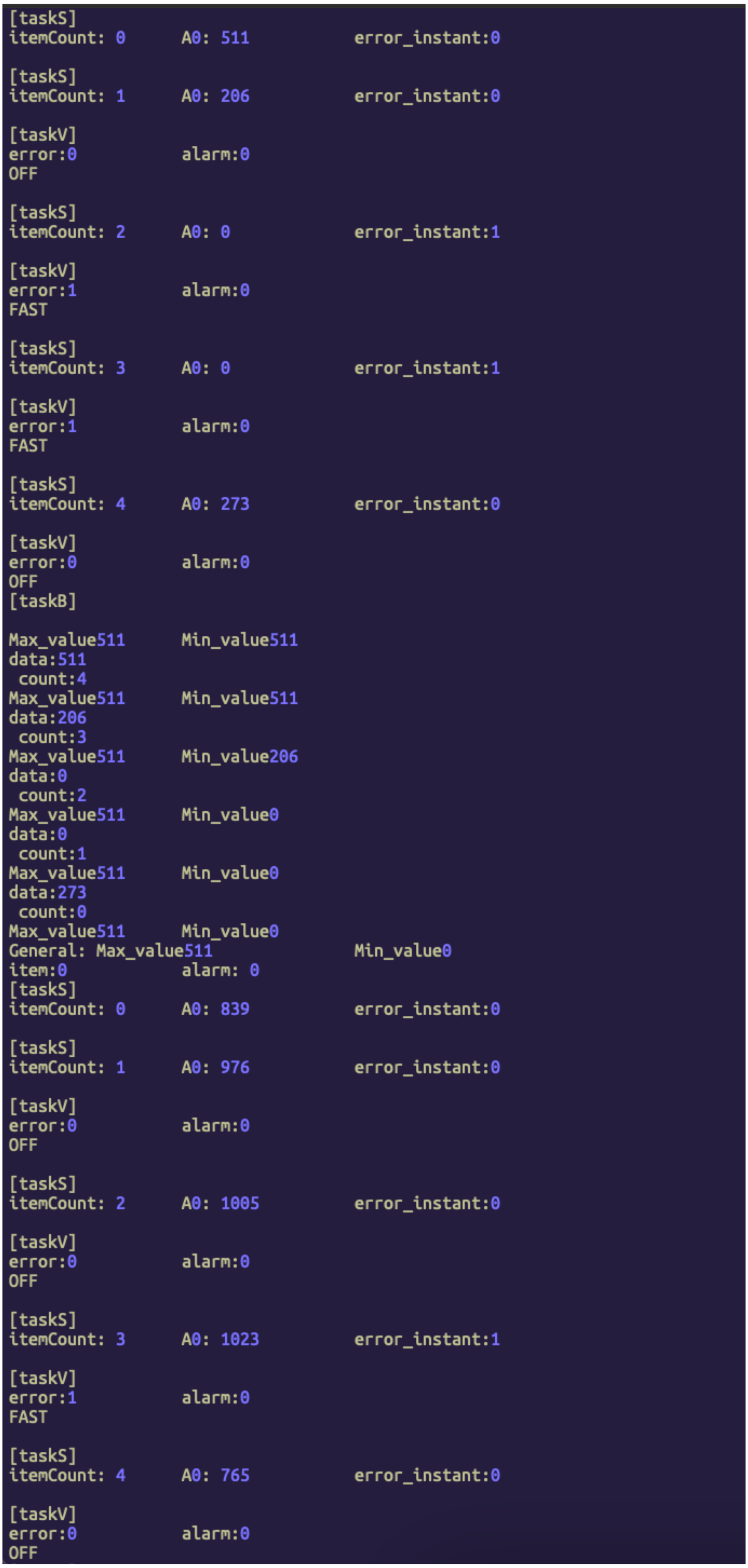
All the tasks have the resource Res.

```
  RESOURCE Res{
    RESOURCEPROPERTY = STANDARD;
  };
```

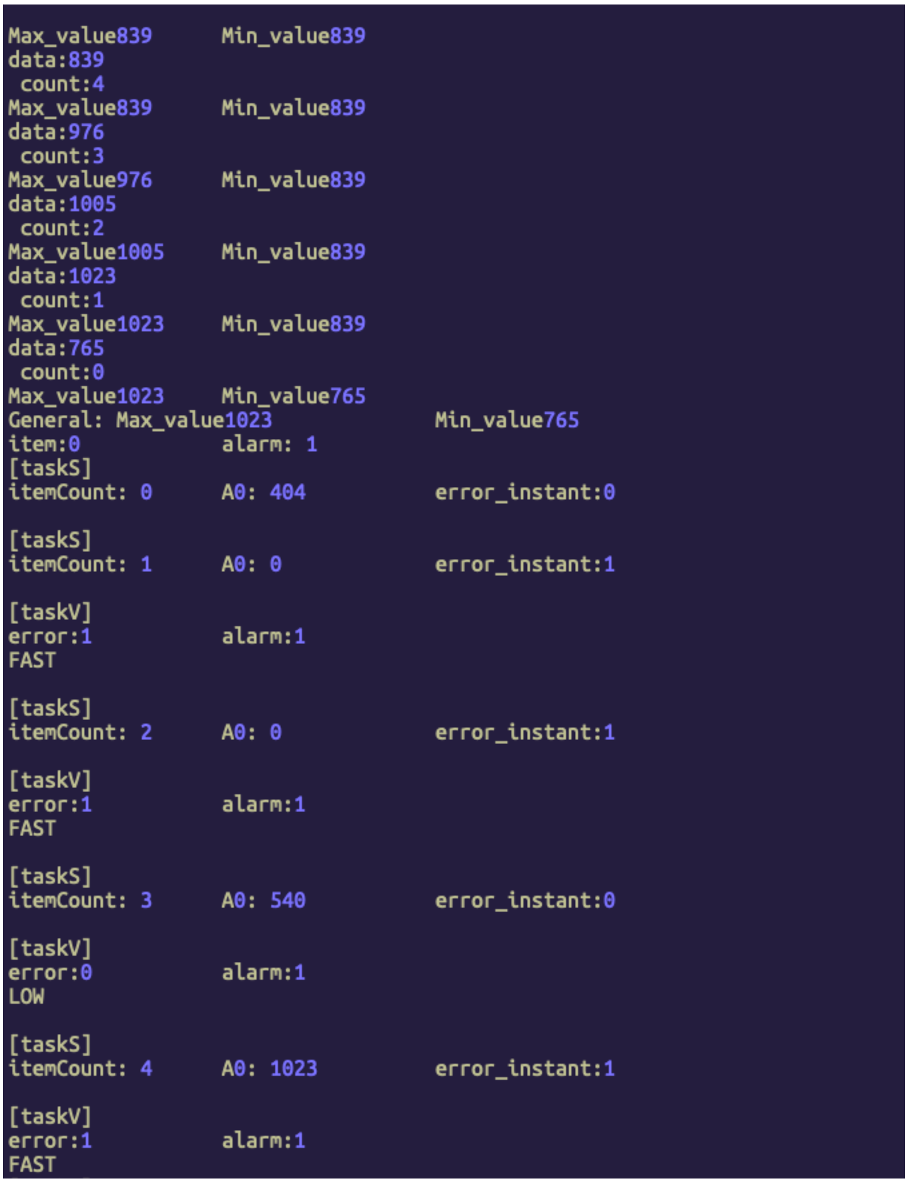
I simulate this code into simulIDE and the result is:



The Serial Monitor is the follow:



```
[taskS]
itemCount: 0      A0: 511           error_instant:0

[taskS]
itemCount: 1      A0: 206           error_instant:0

[taskV]
error:0           alarm:0
OFF

[taskS]
itemCount: 2      A0: 0             error_instant:1

[taskV]
error:1           alarm:0
FAST

[taskS]
itemCount: 3      A0: 0             error_instant:1

[taskV]
error:1           alarm:0
FAST

[taskS]
itemCount: 4      A0: 273           error_instant:0

[taskV]
error:0           alarm:0
OFF
[taskB]

Max_value511      Min_value511
data:511
 count:4
Max_value511      Min_value511
data:206
 count:3
Max_value511      Min_value206
data:0
 count:2
Max_value511      Min_value0
data:0
 count:1
Max_value511      Min_value0
data:273
 count:0
Max_value511      Min_value0
General: Max_value511               Min_value0
item:0            alarm: 0
[taskS]
itemCount: 0      A0: 839           error_instant:0

[taskS]
itemCount: 1      A0: 976           error_instant:0

[taskV]
error:0           alarm:0
OFF

[taskS]
itemCount: 2      A0: 1005          error_instant:0

[taskV]
error:0           alarm:0
OFF

[taskS]
itemCount: 3      A0: 1023          error_instant:1

[taskV]
error:1           alarm:0
FAST

[taskS]
itemCount: 4      A0: 765           error_instant:0

[taskV]
error:0           alarm:0
OFF
```



```
Max_value839      Min_value839
data:839
 count:4
Max_value839      Min_value839
data:976
 count:3
Max_value976      Min_value839
data:1005
 count:2
Max_value1005     Min_value839
data:1023
 count:1
Max_value1023     Min_value839
data:765
 count:0
Max_value1023     Min_value765
General: Max_value1023              Min_value765
item:0            alarm: 1
[taskS]
itemCount: 0      A0: 404           error_instant:0

[taskS]
itemCount: 1      A0: 0             error_instant:1

[taskV]
error:1           alarm:1
FAST

[taskS]
itemCount: 2      A0: 0             error_instant:1

[taskV]
error:1           alarm:1
FAST

[taskS]
itemCount: 3      A0: 540           error_instant:0

[taskV]
error:0           alarm:1
LOW

[taskS]
itemCount: 4      A0: 1023          error_instant:1

[taskV]
error:1           alarm:1
FAST
```

To the status of the LED, I print: FAST, LOW and OFF

The minimum value of K that allows S to run without overflowing Q would depend on the rate at which TaskS samples the analog input voltage A0 and how frequently TaskB offloads the samples. If TaskS runs every 100ms and TaskB runs every 500ms, the minimum value of K could be calculated based on the maximum number of samples TaskS can insert into Q before TaskB offloads them. Given TaskS runs every 100ms and TaskB every 500ms, TaskB needs to clear the buffer before TaskS overflows it. Thus, the minimum value of K would be 5, as TaskS inserts a sample every 100ms, and within 500ms, it would insert 5 samples.

The software's ability to detect significant changes in A0, indicated by the condition M-N > 500, relies heavily on the synchronization between the sampling (TaskS) and processing (TaskB) tasks. Should a substantial alteration in A0 occur between TaskS's sampling and TaskB's processing phases, there exists a potential gap where such changes may go unnoticed.
This vulnerability arises when A0 undergoes a significant transition right after TaskS completes sampling but before TaskB initiates processing. In this scenario, the current execution cycle might not capture the newly occurring significant change in A0.

Exercise #2

Start from the code developed for Exercise #1 and merge tasks S and B into one single task W, so that the system now consists of tasks W and V.

C-CODE

```
#include "tpl_os.h"
#include "Arduino.h"
#include "tpl_com.h"

DeclareAlarm(a125msec);
DeclareAlarm(a100msec);

#define K 5
static int alarm=0;
static int error=0;
#define Res 1

void setup()
{
  pinMode(A0,INPUT);
  Serial.begin(115200); //115200 bps, 8N1

}

TASK(TaskW) {
  static int i;
  static int M=0;
  static int N=0;
  static int circularBuffer[K];
  static int sensorValue;
  static int itemCount = 0;
  int size;

  Serial.print("\n[taskW]\n");


  Serial.print("itemCount: ");
  Serial.print(itemCount);
  Serial.print("\t");
  sensorValue = analogRead(A0);

  if(itemCount < K){
  #ifdef Res
    GetResource(Res);
  #endif
    circularBuffer[itemCount] = sensorValue;
    Serial.print("A0: ");
    Serial.print(sensorValue);
    Serial.print("\t");
    itemCount++;


    if(sensorValue<10 || sensorValue>1013){
    error=1;
    }else{
    error=0;
    }
    Serial.print("error_instant:");
    Serial.print(error);
    Serial.print("\n");

  }
  else if (itemCount>K){
    Serial.print("error_buffer");
  }

  // Serial.print("[taskB]\n");

  // static int a=0;

  else if (itemCount==K){
    size=itemCount;

    M=circularBuffer[0]; // max
    N=circularBuffer[0]; // min

    Serial.print("\nMax_value");
    Serial.print(M);
    Serial.print("\t");
    Serial.print("Min_value");
    Serial.print(N);

    for(i=0;i<size;i++){


      Serial.print("\ndata:");
      Serial.print(circularBuffer[i]);
      if (circularBuffer[i]>M)
      {
        M=circularBuffer[i];
      }
      if (circularBuffer[i]<=N)
      {
        N=circularBuffer[i];
      }
      itemCount--;
      Serial.print("\n count:");
      Serial.print(itemCount);
      Serial.print("\nMax_value");
      Serial.print(M);
      Serial.print("\t");
      Serial.print("Min_value");
      Serial.print(N);
    }

    Serial.print("\nGeneral: Max_value");
    Serial.print(M);
    Serial.print("\t");
    Serial.print("Min_value");
    Serial.print(N);
    Serial.print("\n");
    Serial.print("item:");
    Serial.print(itemCount);



  if (M-N>500)
    alarm=0;
  else
    alarm=1;

  Serial.print("\talarm: ");
  Serial.print(alarm);
  #ifdef Res
   ReleaseResource(Res);
  #endif
  }
}

TASK(TaskV)
{
  Serial.print("[taskV]\n");

  static unsigned int blink = 0;
  #ifdef Res
    GetResource(Res);
  #endif
  Serial.print("error:");
  Serial.print(error);
  Serial.print("\t");
  Serial.print("alarm:");
  Serial.print(alarm);
  Serial.print("\n");
  GetResource(Res);
  if(alarm == 1 && error == 0 ) {
    Serial.print("LOW\t");
    blink++;
    if(blink & 1) digitalWrite(13, HIGH); //odd
    else digitalWrite(13, LOW);        //even
  }
  else if(error==0 && alarm==0) {
    Serial.print("OFF\t");
    digitalWrite(13, LOW);
  }
  else if(error == 1) {
    Serial.print("FAST\t");
    blink++;
    if (blink%4 == 0){
      if(blink%8 == 0) digitalWrite(13, HIGH); //odd
      else digitalWrite(13, LOW);        //even
    }
  }
  #ifdef Res
    ReleaseResource(Res);
  #endif

  Serial.print("\n");
}
```

In this code I use the resource only one time in the task W, because the task W shared with the task V the alarm and the error.

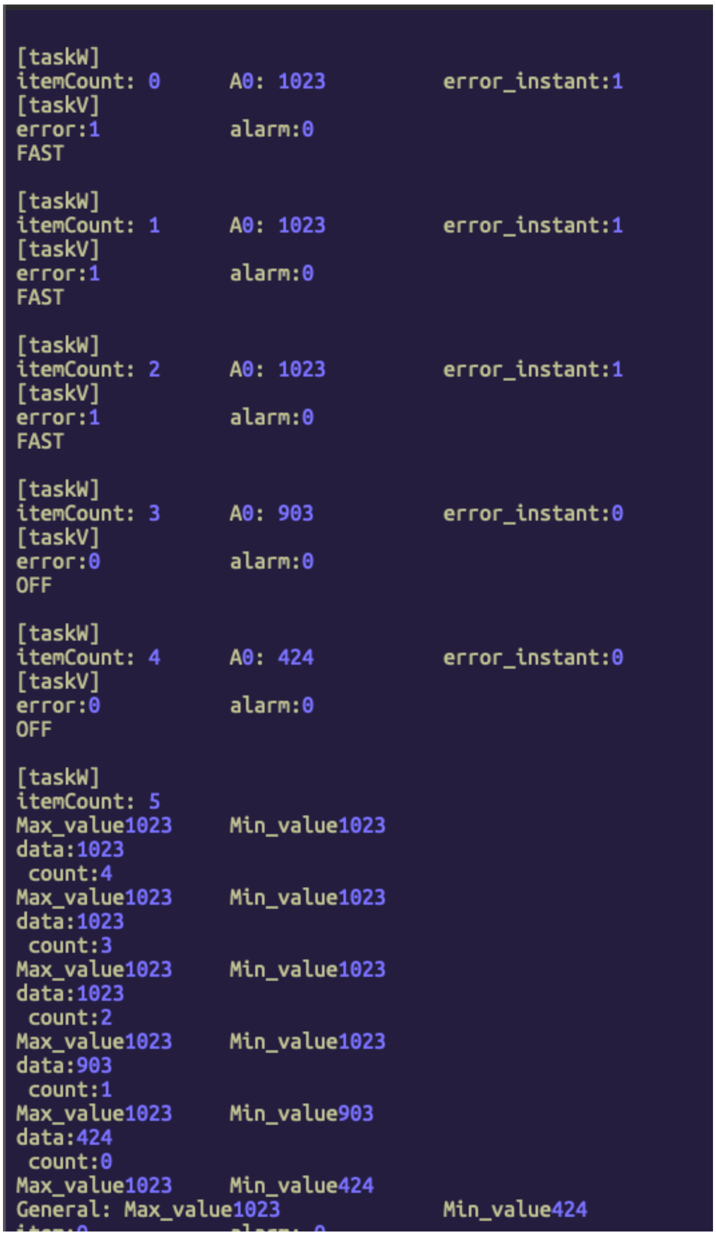
The only shared variable between the two task are alarm and error I set that variable as general variable:

```
static int alarm=0;
static int error=0;
```

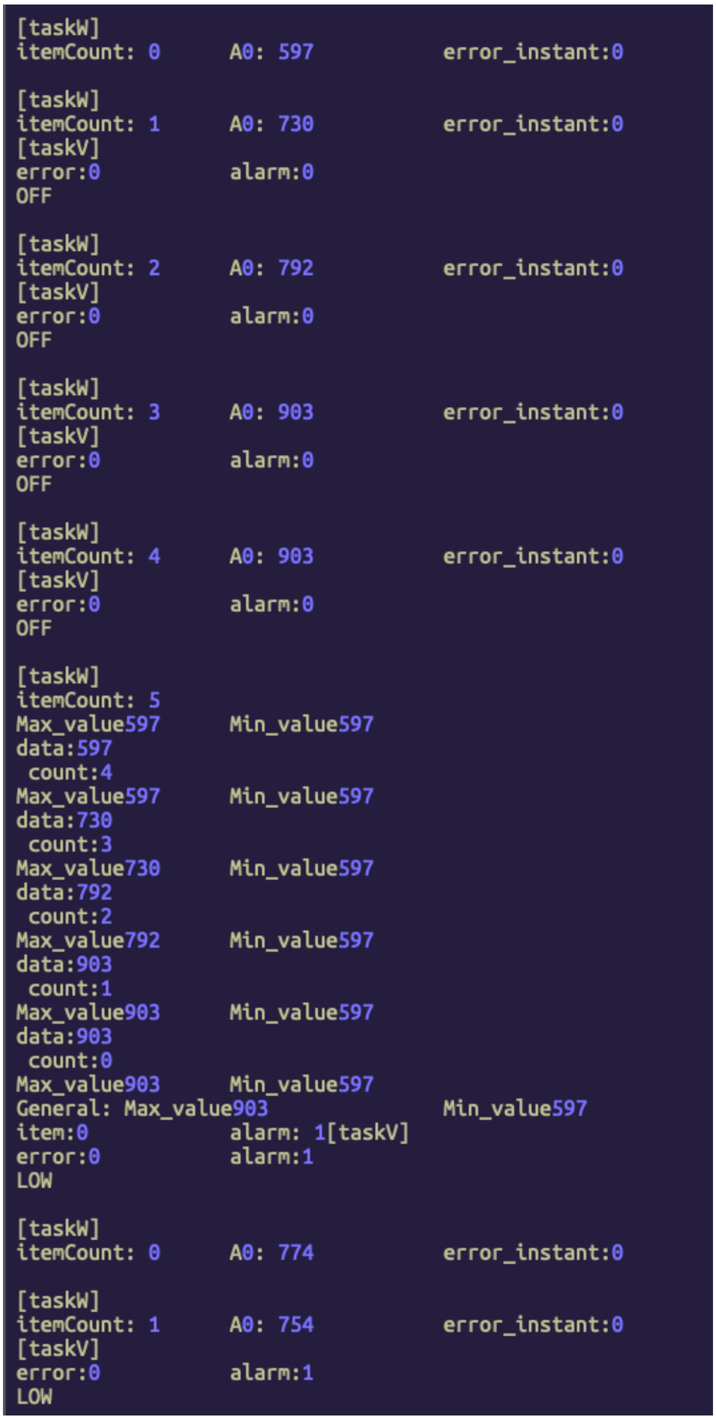
OIL-FILE

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

CPU test {
  OS config {
    STATUS = STANDARD;
    BUILD = TRUE {
      TRAMPOLINE_BASE_PATH = "../../../..";
      APP_NAME = "lab4_2";
      APP_SRC = "lab4.2.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 {};

  RESOURCE Sem{
  RESOURCEPROPERTY = STANDARD;
   };





  ALARM a100ms {
    COUNTER= SystemCounter;
    ACTION = ACTIVATETASK { TASK = TaskW;};
    AUTOSTART = TRUE { ALARMTIME = 98; CYCLETIME = 98;APPMODE = stdAppmode; };
  };



ALARM a125ms {
    COUNTER= SystemCounter;
    ACTION = ACTIVATETASK {TASK =TaskV; };
    AUTOSTART = TRUE {ALARMTIME = 122;CYCLETIME = 122;APPMODE = stdAppmode; };
  };
  TASK TaskW {
    PRIORITY = 2;
    AUTOSTART = TRUE { APPMODE = stdAppmode;};
    ACTIVATION = 1;
    SCHEDULE = FULL;
    RESOURCE= Sem;
  };

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

};
```

The priority of the task W is 2, bigger than the task V, and the period is equal to the Task S (100 ms)



```
[taskW]
itemCount: 0      A0: 1023          error_instant:1
[taskV]
error:1           alarm:0
FAST

[taskW]
itemCount: 1      A0: 1023          error_instant:1
[taskV]
error:1           alarm:0
FAST

[taskW]
itemCount: 2      A0: 1023          error_instant:1
[taskV]
error:1           alarm:0
FAST

[taskW]
itemCount: 3      A0: 903           error_instant:0
[taskV]
error:0           alarm:0
OFF

[taskW]
itemCount: 4      A0: 424           error_instant:0
[taskV]
error:0           alarm:0
OFF

[taskW]
itemCount: 5
Max_value1023     Min_value1023
data:1023
 count:4
Max_value1023     Min_value1023
data:1023
 count:3
Max_value1023     Min_value1023
data:1023
 count:2
Max_value1023     Min_value1023
data:903
 count:1
Max_value1023     Min_value903
data:424
 count:0
Max_value1023     Min_value424
General: Max_value1023              Min_value424
item:0            alarm: 0
```



```
[taskW]
itemCount: 0      A0: 597           error_instant:0

[taskW]
itemCount: 1      A0: 730           error_instant:0
[taskV]
error:0           alarm:0
OFF

[taskW]
itemCount: 2      A0: 792           error_instant:0
[taskV]
error:0           alarm:0
OFF

[taskW]
itemCount: 3      A0: 903           error_instant:0
[taskV]
error:0           alarm:0
OFF

[taskW]
itemCount: 4      A0: 903           error_instant:0
[taskV]
error:0           alarm:0
OFF

[taskW]
itemCount: 5
Max_value597      Min_value597
data:597
 count:4
Max_value597      Min_value597
data:730
 count:3
Max_value730      Min_value597
data:792
 count:2
Max_value792      Min_value597
data:903
 count:1
Max_value903      Min_value597
data:903
 count:0
Max_value903      Min_value597
General: Max_value903               Min_value597
item:0            alarm: 1[taskV]
error:0           alarm:1
LOW

[taskW]
itemCount: 0      A0: 774           error_instant:0

[taskW]
itemCount: 1      A0: 754           error_instant:0
[taskV]
error:0           alarm:1
LOW
```

So, the code works properly. Like the first exercise.

It’s possible to merge tasks W and V into a single task Z and doing everything in a single task

```
#include "tpl_os.h"
#include "Arduino.h"
#include "tpl_com.h"

DeclareAlarm(a25msec);

#define K 5
#define Res 1

void setup()
{
  pinMode(A0,INPUT);
  Serial.begin(115200);

}

TASK(TaskZ) {
    static unsigned int blink = 0;
    int circularBuffer[K];
    static int itemCount = 0;
    static int alarm=0;
    static int error=0;
    static int count=0;




    if (count%4==0)
    {
        Serial.print("\n[taskS]\n");
        // static int X=0;


        Serial.print("itemCount: ");
        Serial.print(itemCount);
        Serial.print("\t");
        int sensorValue = analogRead(A0);

        if(itemCount < K){

            circularBuffer[itemCount] = sensorValue;
            Serial.print("A0: ");
            Serial.print(sensorValue);
            Serial.print("\t");
            itemCount++;


            if(sensorValue<10 || sensorValue>1013){
            error=1;
            }else{
            error=0;
            }
            Serial.print("error_instant:");
            Serial.print(error);
            Serial.print("\n");

        }
        else{
            Serial.print("error_buffer");
        }
    }
    if (count%20==0)
    {
        Serial.print("[taskB]\n");
        // static int mesure=0;
        static int i;
        static int M=0;
        static int N=0;
        // static int a=0;

        int size;

        size=itemCount;

        M=circularBuffer[0]; // massimo
        N=circularBuffer[0]; // minimo

        Serial.print("\nMax_value");
        Serial.print(M);
        Serial.print("\t");
        Serial.print("Min_value");
        Serial.print(N);

        for(i=0;i<size;i++){
            Serial.print("\ndata:");
            Serial.print(circularBuffer[i]);
            if (circularBuffer[i]>M)
            {
            M=circularBuffer[i];
            }
            if (circularBuffer[i]<=N)
            {
            N=circularBuffer[i];
            }
            itemCount--;
            Serial.print("\n count:");
            Serial.print(itemCount);
            Serial.print("\nMax_value");
            Serial.print(M);
            Serial.print("\t");
            Serial.print("Min_value");
            Serial.print(N);
        }

        Serial.print("\nGeneral: Max_value");
        Serial.print(M);
        Serial.print("\t");
        Serial.print("Min_value");
        Serial.print(N);
        Serial.print("\n");
        Serial.print("item:");
        Serial.print(itemCount);



        if (M-N>500)
            alarm=0;
        else
            alarm=1;

        Serial.print("\talarm: ");
        Serial.print(alarm);
    }


    if(count%5==0)
    {

        Serial.print("[taskV]\n");

        Serial.print("error:");
        Serial.print(error);
        Serial.print("\t");
        Serial.print("alarm:");
        Serial.print(alarm);
        Serial.print("\n");

        if(alarm == 1 && error == 0 ) {
            Serial.print("LOW\t");
            blink++;
            if(blink & 1) digitalWrite(13, HIGH); //odd
            else digitalWrite(13, LOW);        //even

        }
        else if(error==0 && alarm==0) {
            Serial.print("OFF\t");
            digitalWrite(13, LOW);
        }
        else if(error == 1) {
            Serial.print("FAST\t");
            blink++;
            if (blink%4 == 0){
            if(blink%8 == 0) digitalWrite(13, HIGH); //odd
            else digitalWrite(13, LOW);        //even
            }
        }

        Serial.print("\n");

    }
    count++;
}
```

Oil-file:

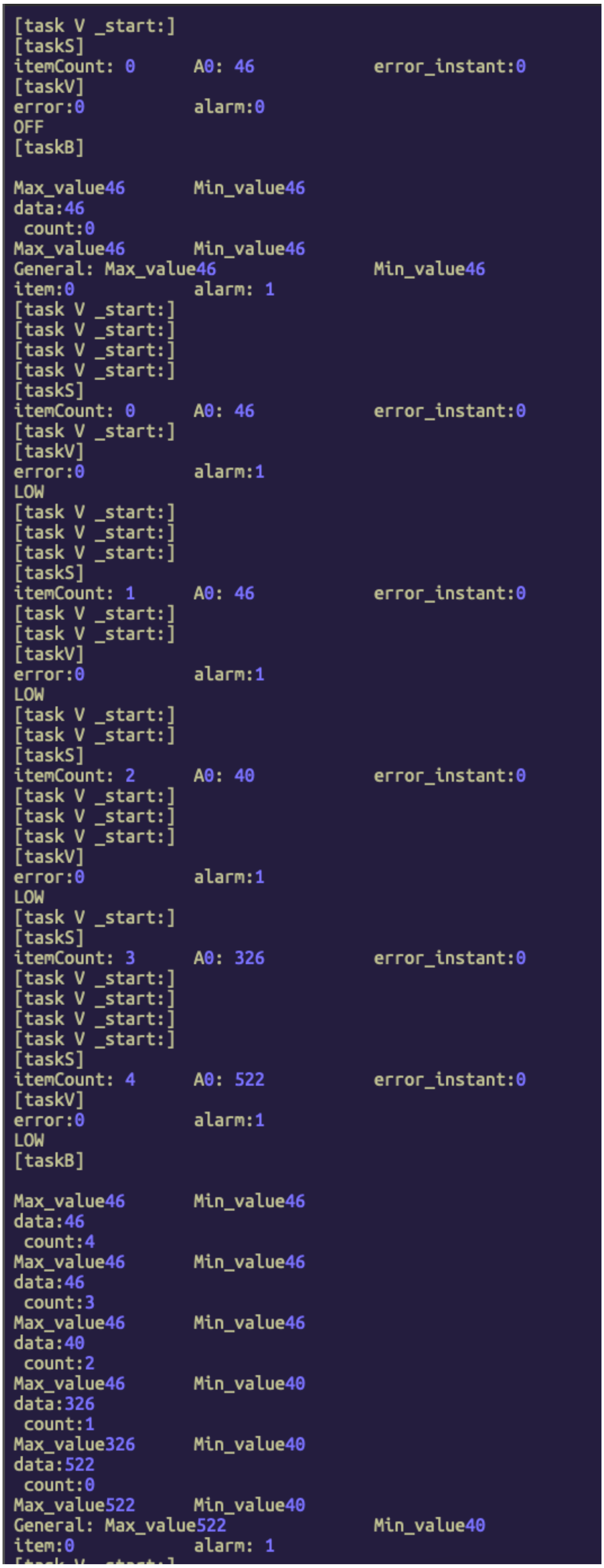
```
OIL_VERSION = "2.5" : "test" ;
```

```
CPU test {
  OS config {
    STATUS = STANDARD;
    BUILD = TRUE {
      TRAMPOLINE_BASE_PATH = "../../../..";
      APP_NAME = "lab43";
      APP_SRC = "lab4.3.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 {};

  RESOURCE Sem{
  RESOURCEPROPERTY = STANDARD;
   };





  ALARM a25ms {
    COUNTER= SystemCounter;
    ACTION = ACTIVATETASK {
      TASK = TaskZ;
    };
    AUTOSTART = TRUE {
      ALARMTIME = 24;CYCLETIME = 24;APPMODE = stdAppmode; };
  };




  TASK TaskZ {
    PRIORITY = 1;
    AUTOSTART = TRUE { APPMODE = stdAppmode;};
    ACTIVATION = 1;
    SCHEDULE = FULL;
    RESOURCE= Sem;
  };


};
```

I use one single task with period=25 ms in this way with some “if-condition” I can emulate the same behavior as the exercise 1

In this case I don’t use any resource because there aren’t shared resource

Combining tasks W and V into a unified task Z might simplify code organization by consolidating functionalities within a single entity. This consolidation, however, might introduce complexity, making the code harder to manage and understand. Additionally, the task's increased complexity could impact CPU utilization, potentially monopolizing the CPU for longer durations and affecting the responsiveness of other tasks within the system. Thus, while consolidating tasks can offer organizational benefits, it requires a delicate balance between simplicity and complexity to maintain code readability and ensure efficient CPU utilization across the system.



```
[task V _start:]
[taskS]
itemCount: 0      A0: 46            error_instant:0
[taskV]
error:0           alarm:0
OFF
[taskB]

Max_value46       Min_value46
data:46
 count:0
Max_value46       Min_value46
General: Max_value46                Min_value46
item:0            alarm: 1
[task V _start:]
[task V _start:]
[task V _start:]
[task V _start:]
[taskS]
itemCount: 0      A0: 46            error_instant:0
[task V _start:]
[taskV]
error:0           alarm:1
LOW
[task V _start:]
[task V _start:]
[task V _start:]
[taskS]
itemCount: 1      A0: 46            error_instant:0
[task V _start:]
[task V _start:]
[taskV]
error:0           alarm:1
LOW
[task V _start:]
[task V _start:]
[taskS]
itemCount: 2      A0: 40            error_instant:0
[task V _start:]
[task V _start:]
[task V _start:]
[taskV]
error:0           alarm:1
LOW
[task V _start:]
[taskS]
itemCount: 3      A0: 326           error_instant:0
[task V _start:]
[task V _start:]
[task V _start:]
[task V _start:]
[taskS]
itemCount: 4      A0: 522           error_instant:0
[taskV]
error:0           alarm:1
LOW
[taskB]

Max_value46       Min_value46
data:46
 count:4
Max_value46       Min_value46
data:46
 count:3
Max_value46       Min_value46
data:40
 count:2
Max_value46       Min_value40
data:326
 count:1
Max_value326      Min_value40
data:522
 count:0
Max_value522      Min_value40
General: Max_value522               Min_value40
item:0            alarm: 1
[task V _start:]
```