

Arduino concurrent programming with LOccam

Gilbert Dhuime

Introduction

Recently, real-time OS efficient kernels (RTOS) have been made available to amateurs who use the Arduino micro controller family.

These are adaptations of FreeRTOS [Ser], which is a professional RTOS, to some microcontrollers of this family.

It is obvious that some robotics applications who are being developed cannot be implemented without recourse to processes associated with some RTOS.

The coming of these adaptations has filled an expectation. Unfortunately, for the most part, the tools provided by an RTOS are relatively technical and require, to be used, some training.

To overcome these difficulties, our knowledge of the Occam [Tho95] language prompted us to formulate a “light” version of it.

LOccam (OCCAM LIGHT) is a C written macros language based on FreeRTOS. A program in LOccam is made up of sequential processes (analogous to C functions) communicating through channels thus resuming the main ideas of the OCCAM language.

The base constructions of LOccam are few and are easily mastered.

The current version of LOccam, LOccam2, differs from the previous one by important changes that brings it closer to the original syntax of Occam especially for alternation processes.

The public in view is made up of amateurs who have a good practice of the C language on Arduino microcontrollers.

No knowledge of concurrent programming is required.

The concepts of the language are studied in detail and many examples are provided that should allow to quickly have the mastery of the latter.

A robot project is finally studied to illustrate, in a realistic setting, an example of a robot program in LOccam using sensors and actuators.

The current implementation of LOccam is based on the adaptation of FreeRTOS due to Bill Greimann [Gre] on the ATmega2560 and Due microcontrollers of the Arduino family.

The OCCAM language has evolved into OCCAM-pi and is maintained by the Kent University which also provides a very good compiler: Kroc. (Kent retargetable Occam Compiler). All information on Kroc can be found at:

<https://github.com/concurrency/kroc>

Moreover one will be able to find a detailed documentation on OCCAM-pi in my manual (in french) [Dhu15] .

Saconin et Breuil
23 Mai 2018

Install LOccam

0.1 Install LOccam

0.1.1 Arduino tools

The software is developed as part of the Arduino IDE that can be downloaded from :

<https://www.arduino.cc/en/Main/Software>

The microcontroller recognized by LOccam2 is either an Arduino ATmega2560 or an Arduino Due.

0.1.2 Install LOccam

Go to the URL: **<http://gilbert.dhuime.pagesperso-orange.fr>**

In the section Programmation concurrente pour Arduino avec LOccam do :

Install LOccam2

Click the link **LOccam2** , download the file LOccam2.tar.gz and unzip it .

Finally copy the file in the **libraries** folder of your Arduino IDE.

Install FreeRTOS

Click the link **FreeRTOS _AVR** if you are using the ATmega2560 controller or

click the link **FreeRTOS _ARM** if you are using the Due controller.

Unzip the file and copy it to the **libraries** folder of your Arduino IDE .

Edit LOccam2.h

If you work with a Mega2560 microcontroller there is nothing to do.

If you are working with a microcontroller Due you have to edit LOccam2.h.

Set comments **# include <FreeRTOS _AVR.h>**

Remove comments from **# include <FreeRTOS _ARM.h>**

Syntax and format of programs

A program LOccam is a program written in C constituting of declaration of process and channels followed by system functions that addresses the RTOS all in the frame imposed by the IDE of the Arduino.
It can include declarations of functions of Arduino's libraries.

0.2 Structure of a LOccam program

Although a LOccam program is written in C and its structure is thus perfectly defined it seems good to remind you of its specificities in the particular setting of the Arduino IDE.

We can distinguish three parts in it.

First part

It is reserved for declarations `#include` and `#define`.

In particular any program must include `#include <LOccam2.h>`.

If structure declarations are needed they are done in this part.

Second part

It consists of declarations of functions and specific declarations to LOccam like `CHAN`, `PROC`, `ALT`, `PALT`, etc.

Third part

This part is constituted by the `setup ()` of the Arduino software.

It includes Arduino-specific system functions like `Serial.begin ()` and specific system functions to LOccam like `NEWCHAN ()`, `PAR ()`, `START ()` etc ...

Note: The `loop ()` part of the Arduino is ignored.

0.2.1 Example

The `par1.ino` program launches two processes `proc0` and `proc1` in parallel.
These processes write 0s and 1s that interleave on the screen and then finish.

Processes

0.3 The processes

A LOccam process is a sequential process that runs in parallel with other processes with which it communicates through channels.

A process does share **no** information with other processes, be it data or code . From the syntactic point of view, a process is a **C** function.

0.3.1 The process declaration

signature: `type proc_id (Parameters).`

type: `PROC | ALT | PALT | SKIP | STOP`

proc_id: identifier

Parameters: `(void * pointer)` Allows the passage of parameters to the process

Example:

```
PROC my_process (CHAN * in)
```

```
//my_process PROC type accepts a read channel
```

0.3.2 The different types of processes

As it has been seen LOccam processes are sequential processes.

Some of them have specific roles.

PROC is the most general type of sequential processes.

ALT and **PALT** are the types associated with alternation processes.

These processes, by their ability to handle multiple channel inputs, allow to implement complex control structures.

The **SKIP** and **STOP** processes, whose principals are essentially theoretical, can be used for debugging.

A special chapter is devoted to the **TIMER** process.

0.3.3 SKIP, STOP

SKIP is a process that does not execute any actions and ends.

In fact SKIP only executes the `EXIT()` function (described here).

STOP once launched is a process that does nothing and does not end. We can consider STOP as a broken process.

Examples:

```
SKIP;
statement1; // statement1 can execute
..... ;
STOP;
statement2; // statement2 can not be executed
..... ;
```

0.3.4 The processes-specific system functions

These functions are only invoked by processes.

WAIT

A process that executes the **WAIT** () function is suspended for an evaluation time in milli seconds.

Example: `WAIT (1000);` // suspend the process for one second

EXIT

A terminating process executes the **EXIT** () function as its last statement.

0.3.5 RTOS specific system functions

These functions are RTOS functions associated with process management.

They are included in the `setup ()` of the Arduino.

PAR

This function declares a new process at RTOS and possibly indicates it that a parameter must be passed to it when it is launched.

Usually the parameters are relative to a channel, a channel array and more in general, a structure involving channels.

signature: `void PAR (process_id, void *parameters)`

Examples:

```
PAR (my_process, NULL);
Declare my_process. No parameters are passed
PAR (my_process, &channel);
Declare my_process. A channel parameter is passed
```

START

This function requires the RTOS to run in parallel all the processes previously set by the **PAR** function

Let's take a look at the program `par1.ino` presented in the previous chapter.

```

void setup() {
    Serial.begin (9600);
    Serial.flush();

    PAR (proc0, NULL); // declare proc0 to RTOS without a parameter
    PAR (proc1, NULL); // declare proc1 to RTOS without parameter
    START () ; // make proc0 and proc1 running in parallel
}

\end {verbatim}

```

PAR replicate

If several processes execute exactly the same code it is useless to declare as many PARs as processes. Their common code is then shared. Example:

```

// The text of the processes "my_process" is declared only once
// nbr_proc copies of my_process are run in parallel.

for (i = 0; i <nbr_proc; i ++) PAR (my_process, NULL);
Start ();

```

Important note

The own resources of a process are expressed by its local variables.

For processes that do not share code it is good to declare them **static** to mend the allocations on the stack.

In the case of a replicated PAR it is essential to **do not declare them static** because there must be as many separate copies of local variables (on the stack) as there are processes in the replicated PAR .

In the example par1.ino the local variables are declared static.

In the following example of par2.ino the local variables are **not** declared static.

Passing a process identifier in parameter

In the case of a PAR replicate an identifier (an integer) must be passed as a parameter so that the actions of the different processes are discriminated.

In the following example if the identifier is 0 the associated process the green led is flashing if not (it is 1) and the associated process causes the red led to flash.

```

// par2.ino
// Two leds are blinking 5 times each.
// Only one text is declared.

#include <L0ccam2.h>

```

```

const int redled = 3      ;
const int greenled = 2   ;

int procid[2]            ; // array of identifier of the processes

PROC process(int *pid){
    int i, id, led        ;

    id = *pid              ;
    if(!id) led = redled   ;
    else led = greenled    ;

    for(i=0;i<5;i++){
        digitalWrite(led,LOW) ;
        WAIT(1000)             ;
        digitalWrite(led,HIGH) ;
        WAIT(1000)             ;
    }

    Serial.print( id)        ;
    Serial.print(" BY ....\n") ;
    EXIT()                   ;
}

void setup() {
    int i                    ;

    Serial.begin(9600)        ;
    pinMode(redled,OUTPUT)    ;
    pinMode(greenled,OUTPUT)  ;

    for(i=0;i<2;i++) procid[i] = i ; // i = identifier of process i

    for(i=0;i<2;i++) PAR(process,procid+i) ; // replicated PAR
    START()                   ;
}

void loop(){}

```

Channels

0.4 Channels

Channels are the only way by which two processes communicate.

Communication is uni directional.

A process writes in a channel and a second process reads in the same channel.

This communication is done according to the "rendez vous" protocol .

The first of the two processes which is ready to read (or write) to a channel is suspended.

When the second process comes the transaction is done and the two processes continue to run their respective codes. In this version of LOccam the type of data transported by the channels is the type **int**.

Warning:

On the Arduino ATmega2560 the int type is coded on 16 bits. On the Arduino DUE the int type is 32 bit coded.

0.4.1 The declaration of channels

A channel has the CHAN type

```
CHAN channel0, channel1; // declare two channels: channel0 and channel1
CHAN channels [20]; // declare an array of 20 channels
```

0.4.2 The referral of the Channels

Once declared, a channel must be referenced to the RTOS.

This referral is performed by the system function NEWCHAN() in the setup() of the Arduino.

signature: NEWCHAN(CHAN *)

Examples

```
Newchan (&Channel0); // Channel0 is referenced
```

```
for (i = 0; i <20; i ++) NEWCHAN(&channels[i]);
// All the channels are referenced
```

0.4.3 Reading and writing channels

The functions `readCHAN()` and `writeCHAN()` allow reading (resp. writing) in a channel by the processes.

```
signature: int readCHAN (CHAN *) // reading a channel returns an int.
signature: void writeCHAN (CHAN *, int) // write an int in a channel
```

Examples:

```
// int x;
x = readCHAN(&channel0); // the channel0 reading is stored in x
//
x = 310;
writeCHAN(&channels[2], x); // write 310 in channels [2]
```

A good notation for channels

When a process manages a channel, a pointer to it is passed in parameter. It is good practice to name the parameter identifier either **in** (read management) or **out** (write management).

Example:

Are two processes P, Q communicating through the channel `mcanal`

```
CHAN mcanal ; // declare the channel
```

```
PROC P (CHAN * in) { // mcanal is read by P
  \\ execute code
}
```

```
PROC Q (CHAN * out) { // mcanal is written by Q
  \\ execute code
}
```

```
void setup () {
  Newchan (&mcanal); // reference mcanal

  PAR (P, &mcanal); // mcanal is passed in parameter to P
  PAR (Q, &mcanal); // mcanal is passed in parameter to Q
  Start (); // make P and Q running
}
```

A first example

In this very simple example, two processes `proc0` and `proc1` exchange the value of an integer through the channel `mcanal`.

To force `proc1` to boot first (while it is playing) a `WAIT (2)` is executed by `proc0` (`proc1` will be suspended until `proc0` has written).

Thus we can visualize the implementation of the "rendez vous" in the transfer protocol.

```

// channel.ino

#include <L0ccam2.h>

CHAN mcanal;

PROC proc0 (CHAN * out) {// written in mcanal

    WAIT(2); // suspended 2 milliseconds
    Serial.println ("\ nP0 emet 123");
    writeCHAN (out, 123);
    Serial.print ("\ nP0: BY ...");
    EXIT();
}

PROC proc1 (CHAN * in) {// reads in channel
    static int received;

    Serial.print ("P1 first:");
    received = readCHAN (in);

    Serial.print ("\ nP1 has received:");
    Serial.print (received);
    Serial.print ("\ nP1: BY ...");
    EXIT();
}

void setup() {

    Serial.begin(9600);
    Serial.flush();

    NEWCHAN (&mcanal);

    PAR (proc0, &mcanal);
    PAR (proc1, &mcanal);
    START();

}

void loop () {
}

```

0.4.4 tokenring.ino

The token ring is a very popular structure in distributed computing. Here 5 processes, organized in ring, circulate a token (an integer).

To do this the identifier process i reads the value of the token in the index channel $(i + 4) \% 5$ which connects it to the process that precede it in the ring, increments it, then writes the new value of the token in the direction of the successor process in the ring by writing in the index channel i .

The process of index 0, process0, launches the token first.

As a result, its text differs from that of other processes.

The token performs 2 turns in the ring.

Note that the parameters passed to the processes in this example must take into account the indices of the incoming and outgoing channels (the channels are declared globally as an array).

Likewise an identifier `procId` is needed for indexing processes that share the same code (whose index is > 0).

To do this the `PARAM` structure is created to meet these needs.

```
// tokenring2.ino

#include <L0ccam2.h>

typedef struct sp {
  CHAN * pred; // pointer to the incoming channel
  CHAN * succ; // pointer to the outgoing channel
  int procId; // process identifier
} PARAM;

CHAN channel[5]; // the ring is constituted of 5 channels
PARAM[5]; // array of parameters

PROC process0 (PARAM * param) { // write first in channel 0
  int token, pid, sender;
  CHAN * precedes, * success; // makes the program easier to read

  precede = param-> pred; // pointer to the channel that is read
  success = param-> succ; // pointer to the channel that is written

  pid = param-> procId;
  sender = (pid + 4) % 5;

  token = 0;
  writeCHAN(successor, token);
  token = readCHAN(precede);

  ++ token;
  writeCHAN (successor, token);
  token = readCHAN (precede);
  Serial.print ("process:");
  Serial.print (pid);
```

```

    Serial.print ("received:");
    Serial.print (token);
    Serial.print ("from:");
    Serial.println (sender);
    //
    Serial.println ("BY ...");
    EXIT();
}

```

```

PROC process (PARAM * param) {
    int pid, sender;
    int token, tower;

    CHAN * precedes, * success;

    pid = param-> procId;
    precede = param-> pred;
    success = param-> succ;
    sender = (pid + 4)% 5;
    turn = 0;
    do{
        token = readCHAN (precede);
        Serial.print ("process:");
        Serial.print (pid);
        Serial.print ("received:");
        Serial.print (token);
        Serial.print ("from:");
        Serial.println (sender);
        token ++;
        tower ++;
        writeCHAN (successor, token);
    } while (turn <2); // each process see the token 2 times
    EXIT();
} // process

```

```

void setup() {
    int i;

    Serial.begin (9600);

    for (i = 0; i <5; i ++) NEWCHAN (&channel[i]);
    for (i = 0; i <5; i ++) {
        parameter[i].succ = &channel[i];           // initialize the fields of the structure
        parameter[i].pred = &channel[(i + 4)% 5]; // output channel for process i
        parameter[i].procId = i;                   // input channel for process i
        parameter[i].procId = i;                   // process identifier
    }
}

```

14

```
PAR(process0, &parameter [0]) ;  
for (i = 1; i <5; i ++) PAR (process, &parameter[i]); // PAR replicated  
Start() ;  
}  
void loop(){}  

```

Alternation processes

Alternation processes ALT are processes that have the ability to manage multiple channels as inputs. If several input channels of such a process have been written then only one is chosen to be read provided it is set ready . Following the reading of the chosen channel a function associated to that channel is executed.

To each input channel of a ALT process is associated a guard.
An input channel is said ready if it has been written and if the associated guard is set to TRUE.
By default the guards are set to TRUE but their value can be changed by specific instructions (see below).
The ready channel that was written first is chosen by a ALT process.

In robotics the inputs of an alternation process can be connected by channels to processes responsible for sensor management.

The function associated with the reading of a channel is devoted to the analysis of the received value, after which it writes a control code in a channel connected to a process responsible for the actions to be performed about the environment of the robot like motors, direction, etc ...
As a result, alternation processes are tools of choice for implementing a structure of control in a robot.

0.5 Alternation processes

In LOccam an alternation process has the ALT type. Its main task is to manage an ALTR structure whose role is to test if an input channel is ready.

signature: ALT process_id (void * parameter)

Examples:

```
ALT my_process (CHAN * tcanal)
```

tcanals is the identifier of a channel array

```
//
```

```
ALT my_process (MyStructure * str)
```

str is the identifier of a structure containing a channel array

The parameters of a ALT process is a channel array identifier or a structure identifier containing **ALL** the channels (read and write) managed by the process.

0.5.1 The ALTR structure

The ALTR structure is defined to ensure the management of the input channels of an alternation process.

However, an alternation process can also manage writing channels, but the management of these is not within the scope of the ALTR structure.

Once declared, the ALTR structure must be recognized from the RTOS by the system function NEWALTR().

The first parameter of NEWALTR() is a pointer to an ALTR structure.

The second parameter (unsigned) gives the number of channels **supported in reading** among **ALL** the channels managed by the alternation process.

More specifically if n channels are supported by an ALTR structure and if p ($p < n$) of them are set to read then p is the second parameter of NEWALTR() and the 0 .. (p-1) channels will only be tested as ready.

signature: NEWALTR (ALTR *, unsigned)

Examples:

```
CHAN channels [6]; // 4 channels in reading and 2 channels in writing
....
```

```
ALT myProcess (CHAN * ptcannels) { // all 6 channels as parameters
    static ALTR altstruct ;
    // other declarations
    NEWALTR (&altstruct, 4); // manages only the index channels 0,1,2,3
    .....
}
```

0.5.2 The readyALTR function

This function tests the presence of a ready channel as input.

If at least one channel is ready ,one is selected (see above), and the index of that channel is returned. Otherwise readyALTR () returns -1.

This function has two parameters.

The first is of type ALTR and the second of type (CHAN *) is a pointer to **all** the channels passed as parameters to the alternative process.

signature: int readyALTR (ALTR, CHAN *)

Example

```
CHAN channels[6]; // 4 channels in reading and 2 channels in writing
```

```

ALT myProcess (CHAN * inout) { // 6 channels as parameters
    static ALTR altstruct;
    static int idc;
    .....
    NEWALTR (&altstruct, 4); // The first four as inputs
    .....
    idc = readyALTR(altstruct, inout);
    .....
}

```

Once a channel has been chosen, it can be read by a `readCHAN()`. The essential point to remember about the management of `idc` is the following sequence:

```

// We suppose having three input channels

idc = readyALTR (altern, inout);
// test the presence of a ready channel
// if a channel is ready idc >= 0 contains the channel index

if (idc >= 0) {
    // This channel is read
    data = readCHAN (((CHAN *) inout + idc));
    switch (idc) {
        case 0: {
            // the index channel 0 is read
            // execute some function
            break;
        }
        case 1: {
            // the index channel 1 is read
            // execute some function
            break;
        }
        case 2: {
            // the index channel 2 is read
            // execute some function
            break;
        }
        default: break;
    } // switch
} // if

```

This schema is very general and will be found in all the examples where we find alternation processes.

It is also very close to that implemented by OCCAM for managing ALT processes.

0.5.3 alt1.ino

This program illustrates how an alternation process manages its multiple inputs.

It implements 5 processes that communicate by means of 4 channels. The Sender0 process writes 10 times in channel 0.

The Sender1 process writes 10 times in channel 1.

The Sender2 process writes 10 times in channel 2.

The alternation process Control gathers these 3 channels and writes the channel 3.

The receive process reads the channel 3 .

If channel 0 is ready then after reading by Control, the associated function prints a message.

If channel 1 is ready then after reading by Control, the associated function is written in channel 3.

If channel 2 is ready then after reading by Control, the associated function causes a led to flash.

```
// alt1.

#include <LOccam2.h>

const int green led = 2;

CHAN channel[4]; // 3 input, 1 output

PROC receive (CHAN * in3) { // read channel 3
    int i, data;

    i = 0;
    do{
        data = readCHAN(in2);
        i ++;
        Serial.print (data);
        Serial.print ("");
    } while (i <30);
    EXIT();
}

ALT Control (CHAN * in_out) { // alternation process
    static int data, idc;
    static unsigned counter;
    static ALTR altern;

    NEWALTR (&altern, 3); // 3 channels (0, 1,2) as input
    counter = 0;
```

```

do{
    idc = readyALTR (altern, in_out);
    if (idc> = 0) {
        data = readCHAN (((CHAN *)in_out + idc);
        switch (idc) {
            case 0: {
                Serial.println ("Well received from Sender0"); // print a message on the screen
                counter ++;
                break;
            }
            case 1: {
                writeCHAN (((CHAN *)in_out + 3), data); // written in channel 3
                counter ++;
                break;
            }
            case 2: {
                digitalWrite (ledverte, HIGH); // flash the led
                delay (1000);
                digitalWrite (ledverte, LOW);
                counter ++;
                break;
            }
            default: break;
        }
    }
} while (counter <30);

Serial.println ("\nControl BY ...");
EXIT();
}

PROC Sender0 (CHAN * out0) {
    static int msg;
    static int count;

    count = 0;
    msg = 0;
    while (count <10) {
        writeCHAN (out0, msg);
        count ++;
    }
    EXIT();
}

PROC Sender1 (CHAN * out1) {
    static int msg;
    static int count;

```

```

        count = 0;
        msg = 1;
        while (count <10) {
writeCHAN (out1, msg);
count ++;
        }
        EXIT();
}

PROC Sender2 (CHAN * out2) {
    static int msg;
    static int count;

    count = 0;
    msg = 2;
    while (count <10) {
        writeCHAN (out2, msg);
        count ++;
    }
    EXIT();
}

void setup() {
int i;

    Serial.begin (9600);
    Serial.flush();
    pinMode (ledverte, OUTPUT);
    for (i = 0; i <4; i ++) NEWCHAN(&channel[i]);

    PAR (Sender0, &channel[0]) ;
    PAR (Sender1, &channel[1]) ;
    PAR (Sender2, &channel[2]) ;
    PAR (receive, &channel[3]) ;
    PAR (Control, channel) ;
    START();
}

void loop(){}

```

0.6 Priority Alternation Processes

The PALT processes differ from the ALT alternation processes only in the choice of a channel in the set of ready channels.

A PALTR type process is managing a PALTR structure which is the analog of the ALTR structure.

The readyPALTR() function, the readyALTR() counterpart, relying on a PALTR structure chooses a channel ready.

Channel indexes of ready channels are examined in ascending order by readyPALTR(). The index channel 0 being scanned first.

The index of the first ready channel found is selected and returned.

syntax

The following are statements about priority alternation processes. They are identical in all respects to statements about alternation processes.

In what follow :

palt_id is a process ID of type PALT.

palt_struct is a structure identifier of type PALTR

Signature : PALT palt_id (void * parametres)

Declaration of a PALTR structure: PALTR palt_struct ;

Signature of NEWPALTR: NEWPALTR (&palt_struct, unsigned) \\\

Signature of the readyPALTR function: int readyPALTR (palt_struct, unsigned)

0.6.1 prialt.ino

This program implements 3 processes connected by 2 channels:

keyboard: Waiting for a character is keyed in it and then writes it in the channel of index 0.

process: Writes the value 1 every 2 seconds in the index 1 channel.

Control is a priority alternation process listening to the writings in channels 0 or 1.

```
// prialt.ino

#include <L0ccam2.h>

CHAN channel[2]; // 2 channels inputing Control process

PROC keyboard (CHAN * out0) { // write in channel 0
    static int dispo ;
    static unsigned lu ;

    while (TRUE) {
        dispo = Serial.available(); // test if available char
        if (dispo != 0) {
            lu = Serial.read();
            writeCHAN (out0, lu);
        }
    }
}
```

```

} // keyboard

PROC Process (CHAN * out1) { // write in channel 1 every 2 seconds
    while (TRUE) {
        WAIT (2000);
        writeCHAN(out1, 1);
    }
}

PALT Control (CHAN * in) {
    static int again, idc, data;
    static PALTR altern;

    again = TRUE;
    NEWPALTR (&altern, 2);
    do{
        WAIT (1000); // highlights the PRI ALT
        idc = readyPALTR (altern, in);
        switch(idc) {
            case 0: { // written by the keyboard process
                data = readCHAN (((CHAN *) in + idc));
                Serial.print ("A char is available:");
                Serial.println(data);
                break;
            }
            case 1: { // written by the process Process
                data = readCHAN (((CHAN *) in + idc));
                Serial.println ("written by PROC process .....");
                break;
            }
            default: { // default is -1
                Serial.println ("DEFAULT.Execute some action");
                break;
            }
        }
    } while (again);
} // Control

void setup () {
    int i;

    Serial.begin (9600);
    for (i = 0; i <2; i ++) NEWCHAN (&channel [i]);

    PAR (keyboard, &channel[0]) ;
    PAR (Process, &channel[1]) ;
}

```

```

    PAR (Control, channel) ;
    START() ;
}

void loop(){}
```

0.7 Guards

As we have seen a channel is set ready if , in particular, the associated guard is set to TRUE . By default the guards are set to TRUE.
The following functions perform guards management for ALT and for PALT processes.

The setGUARD() function (resp psetGUARD() for priority alternative processes) set the guard of that channel TRUE.
The resetGUARD() function (resp presetGUARD() for priority alternative processes) set the guard of that channel FALSE.

The first argument of these functions is an identifier of an ALT structure (resp PALT)
The second argument is a channel index.

signatures: void setGUARD (ALT, unsigned). void resetGUARD(PALT,unsigned)
 void resetGUARD(ALT,unsigned) void presetGUARD(PALT,unsigned)

Example

```

ALT altern      ;

resetGUARD(altern,0)    ; // set the guard of channel 0 FALSE
setGUARD(altern,0)      ; // set the guard of channel 0 TRUE
```

0.7.1 guardes.ino

This program illustrates the action of the guards.

It has 4 processes connected by 3 channels.
The Sender0 process writes to channel 0.
The Sender1 process is written in channel 1.
The Sender2 process writes to channel 2.
The alternation process Control reads channels 0,1,2.

Control prohibits channel 0 first by running resetGUARD(altern,0).
The values written in channel 0 are thus blocked.
When Sender2 process writes to channel 2 then Control unblocks channel 0 by executing setGUARD(altdern,0).

NB: For Sender2 to start AFTER Sender0 , it runs as first statement a WAIT(1000).

```

// guardes.ino

#include <LOccam2.h>

#define NBRM 5 // number of messages emitted by each process

CHAN channel[3];

ALT Control (CHAN * in) {
    static int data, idc;
    static int count, NBRMC;
    ALTR altern;
    //
    NEWALTR (& altern, 3); // 3 channels (0,1,2) as input
    NBRMC = 2 * NBRM;
    count = 0;
    resetGUARD (altern, 0); // channel 0 disabled
    while (count < NBRMC) {
        idc = readyALTR (altern, in);
        if (idc >= 0) {
            data = readCHAN (((CHAN *) in + idc));
            //
            switch (idc) {
                case 0: {
                    Serial.print ("channel 0:");
                    Serial.println (data);
                    count ++;
                    break;
                }
                case 1: {
                    Serial.print ("channel 1:");
                    Serial.println (data);
                    count ++;
                    break;
                }
                case 2: { // on
                    setGUARD (altern, 0); // channel 0 enabled
                    break;
                }
            }
            default: break;
        } // switch
    }
    WAIT(100); // for klean printing at the end
    Serial.println ("\nControl BY ....");
    EXIT();
} // Control

```

```

PROC Sender0 (CHAN * out0) { // write channel 0
    static int data0;
    static int i;

    i = 0;
    data0 = 10;
    while (i < NBRM) {
        writeCHAN (out0, data0);
        i ++;
    }
    WAIT (100);
    Serial.println ("\nS0 BY ...");
    EXIT();
}

```

```

PROC Sender1 (CHAN * out1) { // write channel 1
    static int data1;
    static int i;

    i = 0;
    data1 = 20;
    while (i < NBRM) {
        writeCHAN (out1, data1);
        i ++;
    }
    WAIT (100);
    Serial.println ("\nS1 BY ...");
    EXIT ();
}

```

```

PROC Sender2 (CHAN * out2) { // write channel 2
    static int msg;

    msg = -1; // value not important
    WAIT (2000);
    writeCHAN(out2, msg);
    Serial.println ("\nS2 BY ...");
    EXIT();
}

```

```

void setup() {
    int i;

    Serial.begin(9600);
    Serial.flush();

    //////////////////////////////////

```

```
    for (i = 0; i < 3; i++) NEWCHAN (&channel[i]);

    PAR (Sender0, &channel[0]);
    PAR (Sender1, &channel[1]);
    PAR (Sender2, &channel[2]);
    //
    PAR (Control, channel);
    START();
}

void loop(){}

```

The timer

0.8 The timer

A timer is a special process that delivers a value associated with time.

This value is incremented at constant time intervals, the TICKS, which are expressed in milli seconds.

The TICKS constant, defined in LOccam2.h is worth 10 milliseconds.

It can be redefined if necessary.

The values taken by the timer are expressed by the type `u_long` (unsigned long 32 bits). They are cyclically spread over the range of 0 to $2^{32} - 1$.

0.8.1 Declaring a timer

A timer has the type `TIMER`. It associates with it the process `runTIMER`

syntax: `TIMER identifier;`

Example:

```
TIMER tmr;
```

Before being activated a timer must be referenced by the RTOS in the `setup()`.

This is the purpose of the `NEWTIMER()` function. This function has two parameters. The first is a pointer to a timer and the second, possibly `NULL`, a pointer to a channel.

signature: `NEWTIMER (TIMER *, CHAN *)`

Examples:

```
TIMER t ;
```

```
CHAN canal ;
```

```
NEWTIMER (&t, NULL) ;
```

```
NEWTIMER (&t, &canal) ;
```

0.8.2 The runTIMER process

The runTIMER process actually starts the timer.

This process is activated in parallel with the other processes by the PAR function. In the PAR the second parameter must be a pointer to a TIMER type variable.

signature: PROC runTIMER (TIMER *)

Example:

TIMER t ;

PAR (runTIMER, &t); launches the timer t

0.8.3 The functions associated with a timer

readTIMER()

This function returns the current value of the timer.

signature: u_long readTIMER (TIMER *)

Example: x = readTIMER (&t); // x is of type u_long

setDELAY()

This function asks the timer to write to a channel past a certain time. It is only valid if a channel is declared in NEWTIMER(). An important application, watchdog.ino, uses this function.

signature: void setDELAY(TIMER *,u_long)

example:

TIMER tim ;

CHAN mcanal ;

in the setup (): NEWTIMER (&tim, &mcanal);

If setDELAY (&tim, 400L) is invoked by a process after 4 seconds the mcanal channel is written.

SUB()

This function returns the value of the time interval that separates 2 readings from the timer.

The first argument is associated with a reading **after** that of the second

signature: u_long SUB (u_long, u_long)

Example:

```

PROC P() {
    u_long t1, t2, delta;
    .....
    t1 = readTIMER (&t);
    .....
    t2 = readTIMER (&t);
    delta = SUB (t2, t1);
    .....
}

```

0.8.4 timer0.ino

This very simple program prints a star every second for 20 seconds.

```

// timer0.ino

#include <L0ccam2.h>

TIMER t;

PROC p0 (TIMER * tim) {
    static u_long tim1, tim2, st;
    int count ;

    tim1 = readTIMER (tim);
    count = 0 ;
    do{
        tim2 = readTIMER (tim);
        st = SUB (tim2, tim1);
        if (st>= 100L) { // If a second has elapsed
            Serial.print ("*"); // print a star
            tim1 = tim2 ;
            count ++ ;
        }
    } while (count <20);
    Serial.println ("\np0 BY ...");
    EXIT();
}

void setup(){

    Serial.begin (9600);

    NEWTIMER (&t, NULL);

    PAR (runTIMER, &t) ;
    PAR (p0, &t) ;
}

```

```

    START() ;
}

void loop(){}

```

0.8.5 watchdog.ino

This program tests if a character enters the keyboard within 4 seconds. If yes, it displays a message and continues by granting again a delay of 4 seconds otherwise it stops with the message “ TIME OUT ”.

The PALT process watchdog (with priority to the keyboard) manage 2 channels of indices 0 and 1 in input.

The channel of index 0, is written by the keyboard process as soon as a character is entered on the keyboard. If a delay of 4 seconds is past the timer write in the index 1 channel.

(This is the case because of the WAIT (4000) in the do loop of the keyboard process). As watchdog is of PALT type if channel 0 is not ready then channel 1 is necessarily ready and the keyboard process ends.

```

// watchdog.ino

#include <L0ccam2.h>

typedef struct st {
    TIMER * timer;
    CHAN * channel;
} STC;

TIMER t ;
CHAN channel [2];
STC tmrchan ;

PROC keyboard (CHAN * channel0) {
    static int dispo;
    static unsigned read;

    while(TRUE) {
        dispo = Serial.available ();
        if (available! = 0) {
            read = Serial.read();
            writeCHAN (channel0, read);
        }
    }
}

```

```

PALTR watchdog (STC *stc) { // PRI_ALT alternation process

    static CHAN * in ;
    static TIMER * rtim ;
    static PALTR palt ;

    in = stc-> channel ; // in = array [] of two channels
    rtim = stc-> timer ;

    static int idc, data, encore ;

    NEWPALTR (& palt, 2) ; // two input channels
    encore = TRUE ;
    setDELAY (rtim, 400L) ; // prescribes a delay of 4 seconds
    do{
        WAIT (4000);
        idc = readyPALTR (palt, in);
        if (idc >= 0) {
            data = readCHAN ((CHAN *) (in + idc));
            switch (idc) {
                case 0: { // channel0: keyboard channel
                    Serial.println ("keyboard input received");
                    setDELAY (rtim, 400L);
                    break ;
                }
                case 1: { // channel1: TIMER channel
                    Serial.println ("exceeded time .. ABORT");
                    again = FALSE;
                    break ;
                }
                default: break ;
            }
        }
    } while (encore);
    //
    Serial.println ("TIME OUT ....");
    EXIT();
}

void setup () {
    int i;

    Serial.begin (9600);

    tmrchan.timer = &t ;
    tmrchan.channel = channel ;

    for (i = 0; i < 2; i++) NEWCHAN (&channel[i]);
    NEWTIMER (&t, &channel[1]); // timer t use channel [1]
}

```

32

```
    PAR (runTIMER, &t) ; // start timer (who use channel [1])
    PAR (keyboard, &channel[0]); // use channel [0]
    PAR (watchdog, &tmrchan) ;
    START();
}

void loop(){}

```

Interrupts

0.9 Interrupts

0.9.1 Binary semaphores

In system programming semaphores are mainly used by processes for the management of critical common resources.

Binary semaphores are used in LOccam as a linking tool between interrupts and processes .

Definition

A binary semaphore is a special system variable that does take only two values 0 and 1 . Once initialized by `semCreateBinary()` the only operations allowed on this variable are `semGive()` and `semTake()`.

After its creation by `semCreateBinary()` the value of the semaphore is 0.

Creating a semaphore

A semaphore (not necessarily binary) has the type `sem_t`.

```
sem_t semBin ; // declare the semaphore semBin
```

```
signature: sem_t semCreateBinary().
```

Example:

```
semBin = semCreateBinary(); // create semBin as binary semaphore
```

0.9.2 `semGive()` and `semTake()`

```
signature: void semGive (sem_t), void semTake (sem_t)
```

example:

```
semGive(semBin); semTake (semBin);
```

Functional Definitions

Let S be a binary semaphore:

```
semGive(S)
```

If a process P executes semGive (S) then if there exist a process Q suspended on this semaphore it is reactivated.

Else the value of the semaphore is set to 1.

Note: A process that runs semGive (S) is never suspended.

semTake(S)

If a process P executes semTake(S) then:

If the value of S is 0 this process is suspended.

Otherwise the value of the semaphore is set to 0 and P continues its progression.

0.9.3 Interrupt management

This management is ensured by the Arduino library's attachInterrupt() system function which is documented in :

<https://www.arduino.cc/reference/en/language/functions/external-interrupts/>

The function attachInterrupt()

This function declared in the setup() said how to manage an interrupt given his number and the Handler() function activated when the interrupt occurs.

signature: void attachInterrupt (unsigned, void *, string)

1st parameter: number attached to the interrupt. here 0

2nd parameter: The handler of the interruption. here Handler()

This function is called when interrupt occurs.

3rd parameter: The mode attached to the interrupt. here FALLING

Example: attachInterrupt (0, Handler, FALLING);

Managing interrupts

A binary semaphore S has been declared .His value is zero.

The Handler of the interrupt has for unique instruction semGive(S).

A process P execute as his first instruction semTake(S) and is suspended when activated

An interrupt occurs.

The Handler execute semGiveS() and P previously suspended is activated .

0.9.4 intrupt.ino

This program involves 3 processes proc0, proc1, control.

proc0 : when enabled by the interrupt handler Handler, writes to channel 0.

proc1 : writes every second in channel 1.

Control : is a priority alternative process for reading channels 0 and 1.

When an interrupt occur the channel 0 is written and control after reading it prints "IT CANAL0 " .

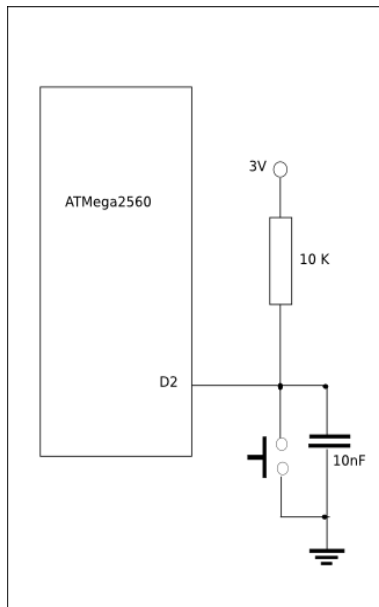


Figure 1: shema cabling

```
//intrupt.ino

// WARNING: Put the voltage under 3Volts with the ATMEGA2560

#include <L0ccam2.h>

sem_t semBin ; // binary semaphore

CHAN channel[2] ;

void Handler() { // capture the interruption
    semGive(semBin) ;
}

PROC proc0 (CHAN * out0) { // handle the it
    static int data = 0 ; // He writes in channel 0

    while(1) {
        semTake (semBin) ;
        writeCHAN (out0, data) ;
    }
}

PROC proc1 (CHAN * out1) { // write channel 1 every second
    static int data = 0;
```

```

    while (TRUE) {
        WAIT (1000) ;
        writeCHAN (out1, data) ;
        data ++ ;
    }
}

PALT control (CHAN * in) {
    static PALTR palt ;
    static int idc, data ;

    NEWPALTR (&palt, 2); // Manage channels 0 and 1
    while (TRUE) {
        idc = readyPALTR (palt, in);
        if (idc >= 0) {
            data = readCHAN((CHAN *) (in + idc));
            switch (idc >= 0) {
                case 0: {
                    Serial.print ("IT on channel 0:");
                    break;
                }
                case 1: {
                    Serial.print ("data emitted by proc1 :");
                    Serial.println(data);
                    break ;
                }
                default: break ;
            } // switch
        }
    }
} // control

void setup () {
    int i;

    Serial.begin (9600);
    attachInterrupt (0, Handler, FALLING);
    semBin = semCreateBinary() ;
    for (i = 0; i < 2; i ++) NEWCHAN (&channel[i]);

    PAR (proc0, &channel[0]) ;
    PAR (proc1, &channel[1]) ;
    PAR (control, &channel) ;
    START() ;
}

void loop () {}

```

Note: The D2 pin of the ATMega2560 corresponds to interrupt numero 0.

A little robot

This little robot is a demonstrator that illustrates the facilities of development in fun robotics with LOccam. It is equipped with a light sensor and a range finder. These send information to the control unit (alternation process) which, after analysis, sends an order to the 2 motors.

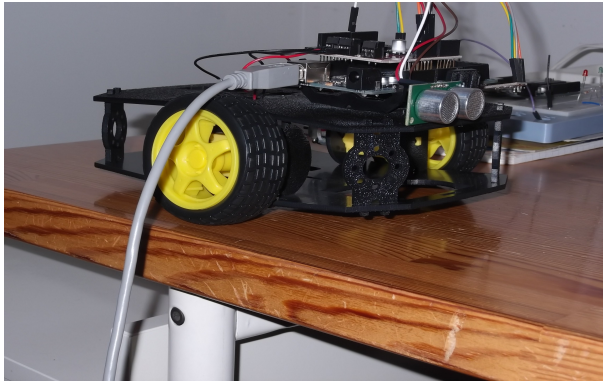


Figure 2: The robot

0.10 General description of the robot

0.10.1 The hardware

The microcontroller

The microcontroller is an Arduino ATmega2560.

We can consult the specifications of this microcontroller at :

<https://store.arduino.cc/arduino-mega-2560-rev3>

The light sensor

The light sensor is a TSL2561 from Sparkfun. We can consult the specifications of this sensor at :

<https://www.sparkfun.com/products/retired/12055>.

At the moment these lines are written Sparkfun announces its withdrawal from

the catalog. It can be replaced by the Sparkfun APDS-9301.

The documentation of the latter is a:

<https://www.sparkfun.com/products/14350>.

This small module dialogs with the micro controller using the I2C bus.

As a result, the SDA pin of the TSL2561 is connected to its SDA counterpart (20).and likewise the SCL pin has the pin SCL (21) micro side.

The INT pin is unused.

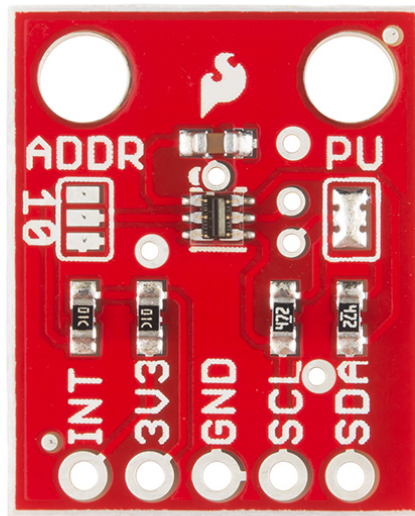


Figure 3: Le TSL2561

The range finder The rangefinder is a SRF05 whose maximum scope is 4 meters. The detailed documentation on this one is may be found at :

<https://www.robot-electronics.co.uk/htm/srf05tech.htm>

This rangefinder is wired in Mode 2 which uses a single pin for the signal and its echo. The “Trigger Input, Echo Mode ” pin is connected to the D5 input of the ATmega2560.

In the process (below) srf05 the number of this pin is given by the constant **TRIGGER**.

The INT brochr is not used.

The SDA pin of the srf05 is connected to the SDA pin(20) of the ATmega2560

The scrf05 SCL pin is connected to the SCL pin(21) of the ATmega2560

The chassis

It is a Shadow chassis from Sparkfun. It is designed to accept 2 wheel drive (not included). You can consult its documentation a:

<https://www.sparkfun.com/products/13301>

The motors These are two DC motors specially designed to fit the Shadow chassis. They are references ROB 13302 at Sparkfun and documents a:

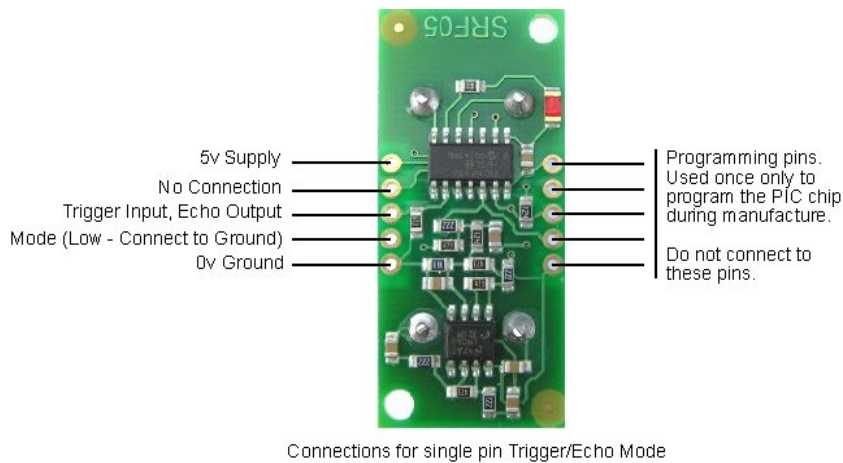


Figure 4: Le SRF05

<https://www.sparkfun.com/products/13302>

The wheels

The 65 mm wheels are designed to be adapted to the Shadow chassis. Their documentation is at:

<https://www.sparkfun.com/products/13259>

Motors control

Motors control is provided by Sparkfun's Ardumoto board. Its very detailed documentation is at:

<https://www.sparkfun.com/products/14129>

0.10.2 The software

The robot is programed in LOccam2. The software is structured around 4 processes connected by 3 channels. The processes are srf05, tsl2561, control and motors.

Channel 0 links the srf05 process to the process control.

Channel 1 links the tsl2561 process to the process control.

Channel 2 links the process control to the motors process.

The process srf05

It is recalled that the plate SRF05 is connected to the pin D5 of the AT-Mega2560. The srf05 process handles this platinum and continually emits queries of distance and writes their result on channel 0.

1 / before making a request D5 must be at low level.

2 / A pulse of 10 micro seconds on D5 announces a request.

3 / D5 is low again, waiting for the answer.

4 / The answer read by pulseIn(), converted to cm and then written in channel 0.

NB : The function pulseIn() associated with the reading of a digital input is documented at:

<https://www.arduino.cc/reference/en/language/functions/advanced-io/pulsein/>

```
PROC srf05(CHAN *out0){
    //
    //  pilote le detecteur a ultra sons SRF05
    //  communique avec control sur le canal 0

    unsigned duration, distance ;
    const int TRIGGER = 5          ; // connection du SRF05 a la broche D5

    while(true){
        pinMode(TRIGGER, OUTPUT)    ;
        digitalWrite(TRIGGER, LOW)  ; //      1
        delayMicroseconds(2)        ;

        digitalWrite(TRIGGER, HIGH) ; //      2
        delayMicroseconds(10)       ;

        digitalWrite(TRIGGER, LOW)  ; //      3
        pinMode(TRIGGER, INPUT)     ;

        duration = pulseIn(TRIGGER, HIGH); //      4

        distance = duration/58      ; // distance en cm
        writeCHAN(out0,distance)    ;
    }
} // srf05
```

The process TSL2561

The software managing the TSL2561 is defined by the `#include "tsl2561.h"` at the head of the program . The documentation of this module ,provided by Sparkfun, has been given above.

It gives very precise details. Also essential information are they given in commentary in the program below .

```
PROC tsl2561 (CHAN *out1) {

    // Emet vers ctrl sur le canal 1
    // la variable lux = intensite de la lumiere
```

```

int data                                ;

unsigned char gain                      ; // Gain setting, 0 = X1, 1 = X16 definit en partie m
unsigned char ltime                    ; // definit , en partie, la valeur de ms . cf doc
unsigned ms                            ; // ms : temps d'integration ("shutter") en milliseconds

unsigned data0, data1                  ; // determinent la valeur de lux
double lux                             ; // donne l'intensite lumineuse

boolean goodLux                        ; // True s'il n'y a pas de saturation
byte error                             ;

// Initialisations generales

begin()                                ; // initialise I2C
gain = 0                               ;
ltime = 2                              ;
setTiming(gain,ltime,&ms)               ; // definit ms
setPowerUp()                           ; // start

// fin des initialisations

while(true){
    WAIT(ms)                            ;

    // Determine data0, data1 , lux

    if (getData(&data0,&data1)) {
        goodLux = getLux(gain,ms,data0,data1,&lux);
        if(goodLux)
            data = (int)lux             ;
        //
        writeCHAN(out1,data)           ; // ecrit canal 1
    }
} // while()

} //TSL2561()

```

The process motors

This process drives both engines by executing the commands which are communicated to it on channel 2 by the process control.

Structure of a command

A command is coded on 16 bits.

bit 0: motor A (0) or B (1).
 bit 1: clockwise (1) or not (0).
 bits 8 .. 15 speed of rotation (from 0 to 255).

The determination of the speed of DC motors makes use of the PWM signals on the pins 3 and 11 of the Arduino.

For more details please refer to the documentation provided by Sparkfun (see above).

```
const char PWMA = 3; PWM ctrl (speed) for motor A
const char PWMB = 11; PWM ctrl (speed) for motor B
```

```
const char DIRA = 12; digital .Direction ctrl for motor A
const char DIRB = 13; digital .Direction ctrl for motor B
```

```
while(true){
    command = readCHAN(in2)          ;

    motor =  command & 1              ;
    sens   = (command & 2) >> 1      ;
    sped   = (command & 0xFF00) >> 8 ;

    if (motor == MOTOR_A){
        digitalWrite(DIRA, sens)  ;
        analogWrite(PWMA, sped)   ;
    }
    else if (motor == MOTOR_B){
        digitalWrite(DIRB, sens)  ;
        analogWrite(PWMB, sped)   ;
    }
}
} // motors
```

The process control

Its role is determinant. It analyzes the data emitted by the processes sfr05 and tsl2561 on channels 0 and 1 respectively.

The result of the analysis is transmitted via channel 2 to the motors process charged of the motors management.

```
ALT control(CHAN *ctrl){
    //
    // Recoit des codes du srf05 (sons)    sur le canal 0
    // recoit des codes du tsl2561 (lux)   sur le canal 1
```

```

// emet sur le canal 2 vers les moteurs

ALTR altern                                ;
int idc , data ;

NEWALTR(&altern,2)                        ; // 2 canaux ( 0, 1) en entrée

// idc = numero d'un canal pret (0 ou 1)
do{
    idc = readyALTR(altern, ctrl)          ;
    if(idc >= 0){
        data = readCHAN(((CHAN *)ctrl+ idc)) ;
        switch(idc){
            case 0:{
                // ecrit par le srf05
                // traite data
                // ecrit eventuellement le canal 2
                break ;
            }
            case 1:{
                // ecrit par le tsl2561
                // traite data
                // ecrit eventuellement le canal 2
                break ;
            }
            default : break ;
        }
    }
} while(true)                             ;
} // ctrl

```

0.10.3 A test program

This program tests the interactions of the motors, srf05 and tsl2561 processes with the process control.

Warning: This program must run with the micro controller connected to the computer

At first control sends the commands to turn the wheels for 3 seconds in one direction then in 3 seconds in the other. Then the engine stop command is issued.

The guard of channel 1 is deactivated so that only the codes coming from channel 0 (issued by srf05) are taken into account.

If the distance read on this channel is less than 50 cm then the guard of channel 1 is reactive and that of channel 0 inhibited and so the information from the srf05 will no longer be taken into account while those from channel 1 (tsl2561) will be.

Finally, if the light intensity drops below 10 lux, the test program ends.

```

/*  tstrobot.ino
 *
 *  Basic tests.
 *
 */

#include <L0ccam2.h>
#include "tsl2561.h"

#define MOTOR_A 0
#define MOTOR_B 1

// Parameters for Ardumoto's motors control

const char PWMA = 3; // PWM ctrl (speed) for motor A
const char PWMB = 11; // PWM ctrl (speed) for motor B

const char DIRA = 12; // Direction ctrl for motor A
const char DIRB = 13; // Direction ctrl for motor B

////////////////////////////////////

CHAN    CTRL[3]                ; // canaux associés au controleur

PROC tsl2561 (CHAN *out1) {

    // Managing the light received by the TSL2561
    // Emet vers ctrl sur le canal 1

    int data                    ;

    unsigned char  gain          ; // Gain setting, 0 = X1, 1 = X16 definit en partie
    unsigned char ltime          ; // permits de definition of ms
    unsigned  ms                 ; // ms : Integration ("shutter") time in millisecon

    unsigned  data0, data1       ; // make available lux value
    double lux                   ; // Resulting lux value

    boolean goodLux              ; // True if neither sensor is saturated
    byte error                   ;

    begin()                      ; // initialise I2C
    gain = 0                     ;
    ltime = 2                    ;
    setTiming(gain,ltime,&ms)     ; // set ms
    setPowerUp()                 ; // start

```

```

while(true){
    WAIT(ms)
    // Retrieve data0, data1 and determine lux

    if (getData(&data0,&data1)) {
        goodLux = getLux(gain,ms,data0,data1,&lux);
        if(goodLux)
            data = (int)lux
    }
    writeCHAN(out1,data)
} // while()

} //TSL2561()

PROC srf05(CHAN *out0){
    //
    // manage srf05 ultra sonic rangefinder
    // communicate with ctrl on channel 0

    unsigned duration, distance ;
    const int TRIGGER = 5 ;

    while(true){
        pinMode(TRIGGER, OUTPUT) ;
        digitalWrite(TRIGGER, LOW) ; // Make sure pin is low before sending a short high to TRIGGER
        delayMicroseconds(2) ;

        digitalWrite(TRIGGER, HIGH) ; // Send a short 10 microsecond high burst on pin to start range finding
        delayMicroseconds(10) ;

        digitalWrite(TRIGGER, LOW) ; // Send pin low again before waiting for pulse back in
        pinMode(TRIGGER, INPUT) ;
        duration = pulseIn(TRIGGER, HIGH); // Reads echo pulse in from SRF05 in micro seconds

        distance = duration/58 ; // distance in cm
        writeCHAN(out0,distance) ;
        //WAIT(100) ;

    } // while
} // srf05

PROC motors(CHAN *in2){
    //
    // manage motors
    // receive commands from ctrl on channel 2

    // structure of commands

```

```

// bit 0 : motor A ou B
// bit 1 : sens clockwise or not
// bits 8.. 15 speed

unsigned command                ;
byte motor , sens, sped        ;

while(true){
    command = readCHAN(in2)      ;

    motor =  command & 1         ;
    sens   = (command & 2) >> 1  ;
    sped   = (command & 0xFF00) >> 8 ;

    if (motor == MOTOR_A){
        digitalWrite(DIRA, sens) ;
        analogWrite(PWMA, sped)  ;
    }
    else if (motor == MOTOR_B){
        digitalWrite(DIRB, sens) ;
        analogWrite(PWMB, sped)  ;
    }
}
} // motors


ALT control(CHAN *ctrl){
//
// Recoit des codes du srf05 (sons) sur le canal 0
// recoit des codes du tsl2561 (lux) sur le canal 1

// emet sur le canal 2 vers les moteurs

    ALTR altern                ;

    int idc, data              ;
    unsigned lux ,dist         ;

    unsigned encore            ;
    unsigned stopptest         ;

    NEWALTR(&altern,2)          ; // 2 canaux ( 0, 1) en entrée

// Tests des moteurs
// Les moteurs tournent 3 secondes en avant a vitesse moyenne (code 128)

writeCHAN(((CHAN *)ctrl+2),0x8002) ; // moteur A
WAIT(10);
writeCHAN(((CHAN *)ctrl+2),0x8003) ; // moteur B

```

```

WAIT(2000)                                ;
//
//  Les moteurs tournent 3 secondes en arriere a vitesse moyenne (code 128)

//
writeCHAN(((CHAN *)ctrl+2),0x8000)          ; // moteur A
WAIT(10);
writeCHAN(((CHAN *)ctrl+2),0x8001)          ; // moteur B
WAIT(2000)
//
Serial.println("STOP MOTEURS .....")      ;
//

writeCHAN(((CHAN *)ctrl+2), 0)              ; // stoppe A
WAIT(10);
writeCHAN(((CHAN *)ctrl+2), 1)              ; //stoppe B

//  Tests du srf05 et du tsl2561
//  le test du srf05 est fait en premier
//  le test du tsl2561 en second
//  On notera l'utilisation des gardes

WAIT(2000)                                ;
encore = TRUE                             ;
stoptest = 0                              ;
resetGUARD(altern,1)                      ; // interdit le canal 1
// idc = numero d'un canal pret (0 ou 1)
do{
    idc = readyALTR(altern, ctrl)          ;
    if(idc >= 0){
        data = readCHAN(((CHAN *)ctrl+ idc)) ;
        switch(idc){
            case 0:{                        // ecrit par le  srf05
                Serial.print("distance : ") ;
                Serial.println(data)        ;
                if(data < 50){              // distance < 50 cm
                    stoptest++              ;
                    Serial.println("STOP test srf05");
                    //
                    setGUARD(altern,1)      ; // autorise le canal 1
                    resetGUARD(altern,0)   ; // interdit le canal 0
                    WAIT(100)              ;
                }
                break ;
            }
            case 1:{                        // ecrit par le tsl2561
                Serial.print("lumiere lux : ");
                Serial.println(data)        ;
                if(data< 10){               // < luminosite < 10 lux
                    stoptest++              ;
                }
            }
        }
    }
}

```

```

        Serial.println("STOP test tsl2561") ;
    }
    break ;
}
default : break ;
}
if(stoptest == 2){
    Serial.println("All tests OK stopped") ;
    encore = FALSE ;
}
}
} while(encore) ;
Serial.println("BY ...") ;
EXIT() ;
} // ctrl

void setup() {

    Serial.begin(9600) ;

    // initialisations pour Ardumoto

    pinMode(PWMA, OUTPUT) ; // analogiques
    pinMode(PWMB, OUTPUT) ;

    pinMode(DIRA, OUTPUT) ; // digitales
    pinMode(DIRB, OUTPUT) ;

    digitalWrite(PWMA, LOW) ;
    digitalWrite(PWMB, LOW) ;
    digitalWrite(DIRA, LOW) ;
    digitalWrite(DIRB, LOW) ;

    //////////////////////////////////////

    unsigned i ;

    for(i=0; i< 3; i++)NEWCHAN(CTRL+i);

    PAR(srf05 , &CTRL[0]) ; // communique avec ctrl via le canal 0
    PAR(tsl2561, &CTRL[1]) ; // communique avec ctrl via le canal 1
    PAR(motors , &CTRL[2]) ; // communique avec ctrl via le canal 2
    PAR(control, CTRL ) ;

    START() ;
    Serial.println("BY ...") ;

}

```

```
void loop(){}  

```


Contents

0.1	Install LOccam	1
0.1.1	Arduino tools	1
0.1.2	Install LOccam	1
0.2	Structure of a LOccam program	3
0.2.1	Example	3
0.3	The processes	5
0.3.1	The process declaration	5
0.3.2	The different types of processes	5
0.3.3	SKIP, STOP	5
0.3.4	The processes-specific system functions	6
0.3.5	RTOS specific system functions	6
0.4	Channels	9
0.4.1	The declaration of channels	9
0.4.2	The referral of the Channels	9
0.4.3	Reading and writing channels	10
0.4.4	tokenring.ino	11
0.5	Alternation processes	15
0.5.1	The ALTR structure	16
0.5.2	The readyALTR function	16
0.5.3	alt1.ino	18
0.6	Priority Alternation Processes	20
0.6.1	prialt.ino	21
0.7	Guards	23
0.7.1	guardes.ino	23
0.8	The timer	27
0.8.1	Declaring a timer	27

0.8.2	The runTIMER process	28
0.8.3	The functions associated with a timer	28
0.8.4	timer0.ino	29
0.8.5	watchdog.ino	30
0.9	Interrupts	33
0.9.1	Binary semaphores	33
0.9.2	semGive() and semTake()	33
0.9.3	Interrupt management	34
0.9.4	intrupt.ino	34
0.10	General description of the robot	39
0.10.1	The hardware	39
0.10.2	The software	41
0.10.3	A test program	45

Bibliography

- [Dhu15] Gilbert Dhuime. *Une introduction à l'algorithmique du parallélisme avec Occam-Pi*. 2015. Available at : gilbert.dhuime.pagesperso-orange.fr/.
- [Gre] Bill Greimann. *FreeRtos implemantation for Arduino's*. Available at : <https://github.com/greiman>.
- [Hoa04] C.A.R Hoare. *Communicating Sequential Processes*. Prentice Hall, 2004. Available at : <http://www.usingcsp.com/cspbook.pdf>.
- [Ros98] A.W Roscoe. *The théory and practice of concurrency*. Prentice Hall, 1998. Available at : <http://web.comlab.ox.uk/people/Bill.Roscoe/publications/68b.pdf>.
- [Ser] Amazon Web Services. *The FreeRTOS Reference Manual*. Available at : <https://www.freertos.org/Documentation>.
- [Tho95] SGS Thomson. *Occam2.1 Reference Manual*. SGS Thomson Microelectronics, 1995. Available at : <http://www.wotug.org/occam/documentation/oc21refman.pdf>.

Index

ALT, 5, 15
alternation, 15
ALTR, 16
Arduino, iii
ATMega2560, 39
attachInterrupt, 33

channel, 9

EXIT, 6

FreeRTOS, iii

Greimann, iii
guards, 15, 23

Kent, iii
Kroc, iii

LOccam, iii

NEWALTR, 16
newchan, 9
NEWPALTR, 20
NEWTIMER, 27

Occam, iii
Occam-pi, iii
Occam2, iii

PALT, 5
PALTR, 20
PAR, 6
PAR replicated, 7
presetGUARD, 23
PROC, 5
processus, 5
program, 3
psetGUARD, 23

readCHAN, 9
readTIMER, 28
readyALTR, 16
readyPALTR, 20
resetGUARD, 23
runTIMER, 27

semaphores, 33
semCreateBinary, 33
semGive, 33
semTake, 33
sequentiel, 5
setDELAY, 28
setGUARD, 23
setup, 3
SKIP, 5
Sparkfun, 39
SRF05, 40
START, 6
STOP, 5
SUB, 28

TIMER, 6, 27
token ring, 11
TSL2561, 39

WAIT, 6
watchdog, 30
writeCHAN, 9