

Informatique embarquée

TP4 : Systèmes d'exploitation temps réel (RTOS)

Namolaru Leonard Rima Rami Benhammou Mohammed Rachid

12 décembre 2022

Informations générales

Les informations d'identification du document

Date du document : 12/12/22

Version du document : 1.00

Les auteurs du document

Namolaru Leonard

Numéro d'étudiant : 51704115

Rima Rami

Numéro d'étudiant : 22121495

Benhammou Mohammed Rachid

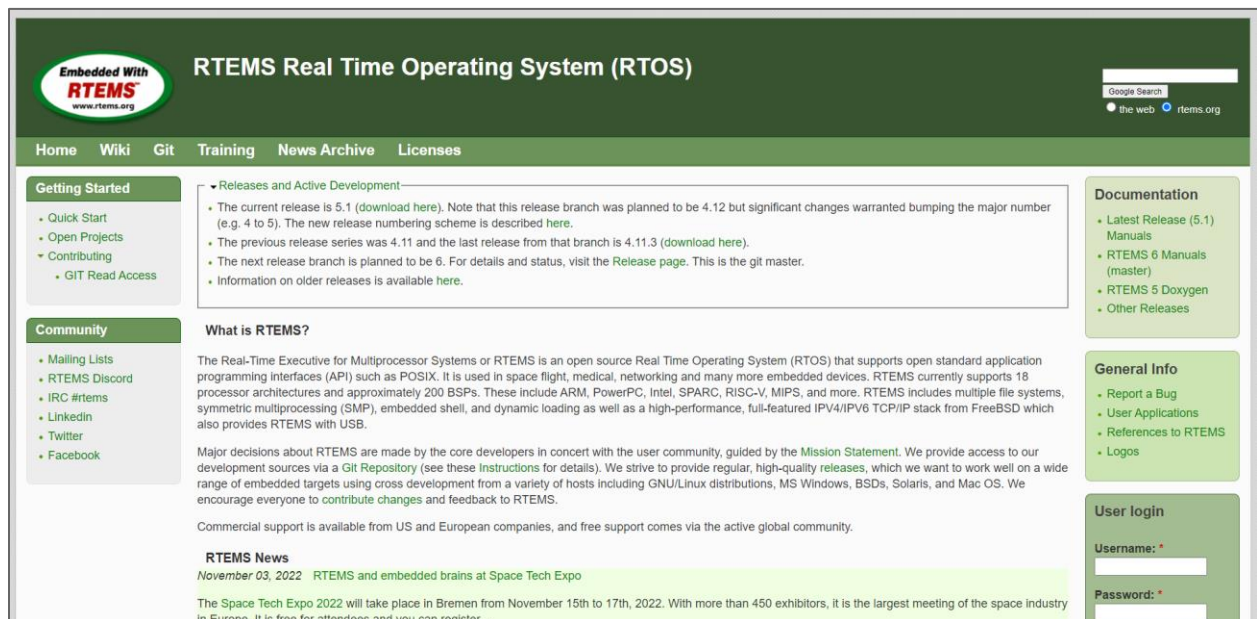
Numéro d'étudiant : 22218399

Sommaire

Sommaire	2
1. Introduction	3
2. Création d'un projet RTEMS	4
3. Rappel : fin du TP 3	7
3. Développer une tâche d'acquisition	11
4. Utiliser un timer et une queue de messages	16
5. Ajouter une tâche de traitement	19
6. Transmission des données entre tâches et création d'une tâche de gestion de télémétrie	25

1. Introduction

Le quatrième et dernier TP de l'unité d'enseignement Informatique embarquée est consacré au travail avec les systèmes d'exploitation temps réel (RTOS), et en particulier avec « RTEMS » (Real-Time Executive for Multiprocessor Systems), un RTOS conçu pour les systèmes embarqués. C'est un logiciel gratuit et open-source.¹



Capture d'écran de la page d'accueil du site rtems.org

Un système d'exploitation temps réel, ou encore noyau temps réel, apporte un certain nombre de services facilitant la conception et la mise au point des applications embarquées temps réel. En particulier, un RTOS apporte la notion de programmation multi-tâches qui permet de concevoir une application sous la forme d'un ensemble de tâches indépendantes, c'est-à-dire ayant leur propre fil d'exécution.²

L'objectif de ce TP est donc de se familiariser avec les services et composants fonctionnels d'un RTOS tels que les tâches, les queues de message, les timers et les sémaphores.

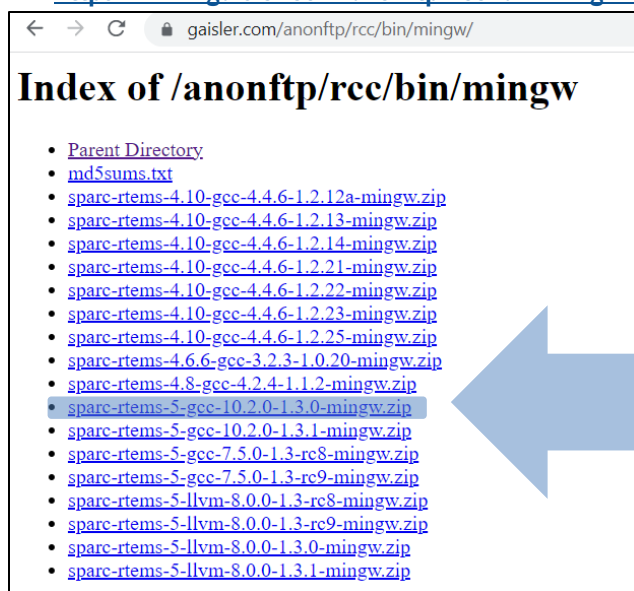
¹ Wikipedia - RTEMS

² Cours 6 - Systèmes d'exploitation temps réel (RTOS)

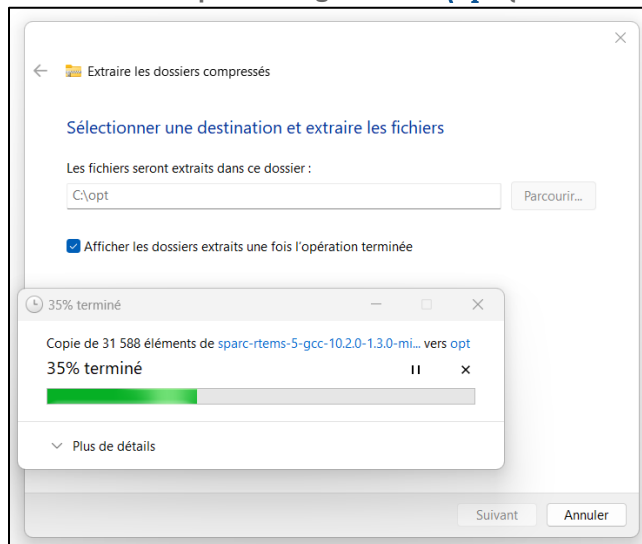
2. Création d'un projet RTEMS

Afin de créer un projet RTEMS via l'IDE Eclipse, sous le système d'exploitation Windows 11, nous devons suivre les étapes suivantes :

1. Il faut d'abord télécharger le fichier `sparc-rtems-5-gcc-10.2.0-1.3.0` depuis le lien <https://www.gaisler.com/anonftp/rcc/bin/mingw/>



2. Extraire le zip téléchargé dans `C:\opt` (afin de suivre au plus près les instructions au début du sujet).



3. Ajouter C:\opt\rcc-1.3.0-gcc\bin à la variable d'environnement Path.

The sequence of screenshots illustrates the process of adding a new path to the Windows environment variable 'Path':

- 1**: A Windows search bar with 'var' entered.
- 2**: The search results showing 'Modifier les variables d'environnement système' as the best result.
- 3**: The 'Propriétés système' window with the 'Variables d'environnement' button highlighted.
- 4**: The 'Variables système' dialog box with the 'Path' variable selected.
- 5**: The 'Modifier...' button for the selected 'Path' variable.
- 6**: The 'Modifier la variable d'environnement' dialog box showing the current list of paths.
- 7**: The same dialog box with the new path 'C:\opt\rcc-1.3.0-gcc\bin' added to the list.

Variables système

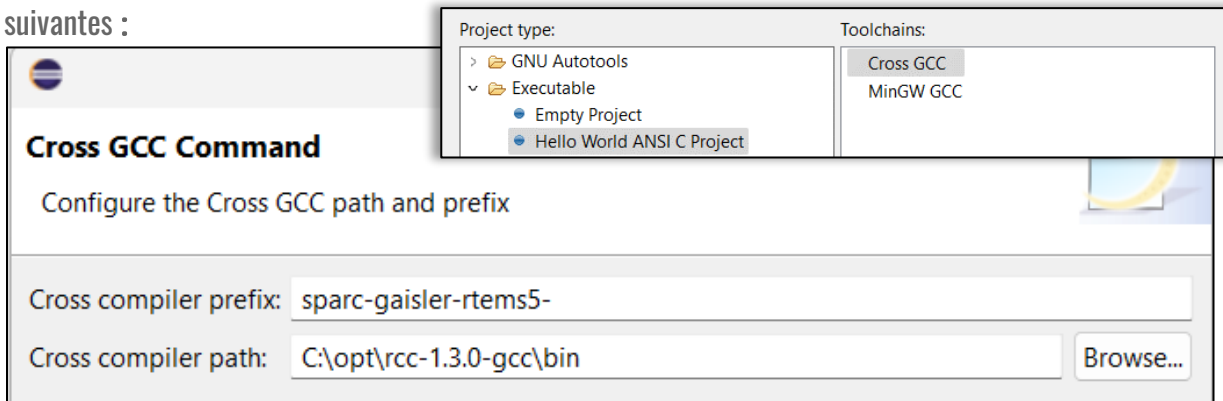
Variable	Valeur
Path	C:\Python310\Scripts;C:\Python310;C:\Program Files\Common Files\Oracle\Java\javapath;.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.PY;.PYW
PATHTEXT	
PROCESSOR_ARCHITECTURE	AMD64
PROCESSOR_IDENTIFIER	Intel64 Family 6 Model 140 Stepping 1, GenuineIntel
PROCESSOR_LEVEL	6
PROCESSOR_REVISION	8c01
PSModulePath	%ProgramFiles%\WindowsPowerShell\Modules;C:\WINDOWS\system32\WindowsPowerShell\v1.0\Modules
PSHOME	C:\Program Files\WindowsPowerShell\Microsoft.PowerShell

Variables d'environnement

Modifier la variable d'environnement

Variable	Valeur
Path	C:\Python310\Scripts\;C:\Python310\;C:\Program Files\Common Files\Oracle\Java\javapath;C:\Program Files (x86)\Common Files\Oracle\Java\javapath;%SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem;%SYSTEMROOT%\System32\WindowsPowerShell\v1.0;%SYSTEMROOT%\System32\OpenSSH\C:\Program Files\Git\cmd;C:\ProgramData\chocolatey\bin;C:\Program Files (x86)\Objective Caml\bin;C:\Program Files\dotnet\;C:\Program Files\nodejs\;C:\Program Files\Go\bin;%ANT_HOME%\bin;C:\PragmaDev\Studio\bin;C:\PragmaDev\Studio\share\3rdparty\MinGW\bin;%JENAROOT%\bat;C:\opt\rcc-1.3.0-gcc\bin

- Créer un nouveau projet sur Eclipse en sélectionnant **Cross GCC**, et en indiquant les informations suivantes :



Cross GCC Command

Configure the Cross GCC path and prefix

Cross compiler prefix:

Cross compiler path:

- Ajouter ensuite, **-qbsp=gr712rc** aux options de compilation du projet (clic droit sur le projet ⇒ Properties ⇒ C/C++ Build ⇒ Settings ⇒ Cross GCC Compiler ⇒ Miscellaneous). De même pour le linkage (clic droit sur le projet ⇒ C/C++ Build ⇒ Settings ⇒ Cross GCC Linker ⇒ Miscellaneous).
- Copier le contenu du fichier **rcc-1.3.0-gcc/src/samples/rtems-tasks.c** à la place du contenu du fichier source par défaut.
- Afin de vérifier si tout marche correctement, on compile le projet et ensuite on exécute le programme en utilisant **tsim** : **load C:/Users/lenny/eclipse-workspace/EmbarqueeTP4/Debug/EmbarqueeTP4**

```
tsim> load C:/Users/lenny/eclipse-workspace/EmbarqueeTP4/Debug/EmbarqueeTP4
section: .text, addr: 0x40000000, size: 155680 bytes
section: .rtemsroset, addr: 0x40026020, size: 160 bytes
section: .data, addr: 0x400280c0, size: 5072 bytes
read 1546 symbols

tsim> run
Initializing and starting from 0x40000000
```

Voici le résultat attendu :

```
*** CLOCK TICK TEST ***
TA1 - rtems_clock_get - 09:00:00 12/31/1988
TA2 - rtems_clock_get - 09:00:00 12/31/1988
TA3 - rtems_clock_get - 09:00:00 12/31/1988
TA1 - rtems_clock_get - 09:00:05 12/31/1988
TA2 - rtems_clock_get - 09:00:10 12/31/1988
TA1 - rtems_clock_get - 09:00:10 12/31/1988
TA3 - rtems_clock_get - 09:00:15 12/31/1988
TA1 - rtems_clock_get - 09:00:15 12/31/1988
TA2 - rtems_clock_get - 09:00:20 12/31/1988
TA1 - rtems_clock_get - 09:00:20 12/31/1988
TA1 - rtems_clock_get - 09:00:25 12/31/1988
TA3 - rtems_clock_get - 09:00:30 12/31/1988
TA2 - rtems_clock_get - 09:00:30 12/31/1988
TA1 - rtems_clock_get - 09:00:30 12/31/1988
TA1 - rtems_clock_get - 09:00:35 12/31/1988
TA2 - rtems_clock_get - 09:00:40 12/31/1988
TA1 - rtems_clock_get - 09:00:40 12/31/1988
TA3 - rtems_clock_get - 09:00:45 12/31/1988
TA1 - rtems_clock_get - 09:00:45 12/31/1988
TA2 - rtems_clock_get - 09:00:50 12/31/1988
TA1 - rtems_clock_get - 09:00:50 12/31/1988
TA1 - rtems_clock_get - 09:00:55 12/31/1988
TA3 - rtems_clock_get - 09:01:00 12/31/1988
TA2 - rtems_clock_get - 09:01:00 12/31/1988
TA1 - rtems_clock_get - 09:01:00 12/31/1988
TA1 - rtems_clock_get - 09:01:05 12/31/1988
TA2 - rtems_clock_get - 09:01:10 12/31/1988
TA1 - rtems_clock_get - 09:01:10 12/31/1988
TA3 - rtems_clock_get - 09:01:15 12/31/1988
TA1 - rtems_clock_get - 09:01:15 12/31/1988
TA2 - rtems_clock_get - 09:01:20 12/31/1988
TA1 - rtems_clock_get - 09:01:20 12/31/1988
TA1 - rtems_clock_get - 09:01:25 12/31/1988
Simulation time overflow (2^32) - halting

Stopped at time 4294967040 (00:01:25)
tsim>
```

3. Rappel : fin du TP 3

3.1. Le projet PLATO³

L'observatoire spatial PLATO, en cours de développement, scrutera dès 2026 de large zone du ciel. Ses mesures photométriques continues sur plusieurs mois permettront d'identifier et caractériser les systèmes planétaires, notamment en détectant les transits d'exoplanètes et en mesurant les oscillations (mouvements) des étoiles.

Pour être étudiée, chaque étoile est extraite de l'image dans une fenêtre de 6x6 pixels. Lui est associé un masque de même dimension correspondant à la déformation attendue sur le capteur.

Voici un exemple de fenêtre 6x6 et de masque :

```
float window[] = {
    2.4, 52.38, 2.36, 2.34, 82.32, 92.3,
    12.4, 1710.38, 3716.36, 3558.34, 7.32, 1.3,
    7.4, 1852.38, 4516.36, 6558.34, 1689.32, 1.3,
    2.4, 52.38, 1289.36, 1289.34, 1646.32, 92.3,
    2.4, 52.38, 9.36, 1610.34, 1486.32, 92.3,
    2.4, 52.38, 2.36, 2.34, 1486.32, 92.3,
};

float mask[] = {
    0, 0, 0.12, 0.14, 0, 0,
    0, 0, 1, 0.9, 0, 0,
    0.09, 0.19, 0.96, 0.75, 0.47, 0.19,
    0, 0.13, 0.15, 0.39, 0.88, 0,
    0, 0, 0.15, 0.39, 0.88, 0,
    0, 0, 0.07, 0.19, 0, 0
};
```

Pour étudier l'évolution de la luminosité d'une étoile, doit être étudié un algorithme de photométrie, une fonction prenant en paramètres 2 tableaux de 36 float (fenêtre et masque) :

```
double computeFluxPondere(float window[], float mask[])
```

³ TP 2 Mesure du temps et de la charge CPU

3.2. La bibliothèque libwindows-producer.a

L'objectif du dernier exercice du TP précédent, le TP numéro 3, était de s'interfacer avec une bibliothèque externe : `libwindows-producer.a`. La méthode principale de cette bibliothèque est `produce_images()` qui produit de nouvelles fenêtres pour la même étoile mais avec une luminosité évoluée.

void produce_images(Windows_producer* wp)

Le dernier exercice du TP3 consistait, en premier temps, à appeler la fonction `produce_images()` toutes les 100 ms :

```
#define INTERRUPT_MASK_REGISTER 0x80000240
volatile uint32_t* interrupt_mask_register = (uint32_t*) INTERRUPT_MASK_REGISTER;
#define TIMER_2_COUNTER_VALUE_REGISTER 0x80000320
volatile uint32_t* timer_2_counter_value_register = (uint32_t*) TIMER_2_COUNTER_VALUE_REGISTER;

uint8_t interruption = 0;

void activate_interrupt(uint32_t irq, void* handler){
    modifier_registre_1(interrupt_mask_register, irq);
    // void *bcc_isr_register(int source, void (*handler)(void *arg, int source), void *arg);
    bcc_isr_register(irq, handler, 0);
}

void handler(void* arg, int irq){ interruption = 1; }

void start_timer(volatile uint32_t* timer_counter_register, uint32_t period){
    // 31: 0 Timer Counter value. Decrement by 1 for each prescaler tick.
    // 31: 0 Timer Reload value.
    // (Source : https://www.gaisler.com/doc/gr712rc-usermanual.pdf , p. 92)

    volatile uint32_t* timer_reload_value_register = timer_counter_register + 1;
    volatile uint32_t* timer_control_register = timer_counter_register + 2;

    *timer_counter_register = period;
    *timer_reload_value_register = period;

    modifier_registre_1(timer_control_register, EN); // #define EN 0 // Enable : Enable the timer.
    modifier_registre_1(timer_control_register, LD); // #define LD 2 // Load
    modifier_registre_1(timer_control_register, RS); // #define RS 1 // Restart
    modifier_registre_1(timer_control_register, IE); // #define IE 3 //Interrupt Enable
}

activate_interrupt(9, handler); // timer 2 => irq 9

// void start_timer(volatile uint32_t* timer_counter_register, uint32_t period)
// period (µs), 1 µs = 0.001 ms , 1 ms = 1,000 µs
// 100 ms = 100,000 µs => period = 100000
start_timer(timer_2_counter_value_register, 100000);

while (...) {
    if(interruption == 1) {
        produce_images(&wp);
        interruption = 0;
    }
}
```

Deuxième étape : définir un handler qui indiquera dans une variable globale que les nouvelles images sont prêtes (c-à-d : la fonction `produce_images()` à terminer une exécution), cela peut être fait facilement grâce à la fonction suivante de la librairie :

```
/**
 * Enable the trigger of an IRQ when produce_images() finish.
 * @param wp pointer on the structure which store the internals variables of the component
 * @param irq IRQ to trigger
 */
extern void enable_irq(Windows_producer* wp, int irq);
```

La documentation nous indique que la valeur du paramètre `irq` doit être 10 :

```
...
 * Trigger the IRQ 10 when the \a img_buffer contains the new windows.
...
extern void produce_images(Windows_producer* wp );
```

C'est pourquoi nous avons ajouté les parties suivantes à notre code :

```
uint32_t new_images_are_ready = 0;

void produce_images_finished(void* arg, int irq){
    new_images_are_ready = 1;
}
```

```
enable_irq(&wp, 10);
activate_interrupt(10, produce_images_finished);
```

La dernière étape : surveiller le paramètre globale précédent et calculer le flux pondéré pour chaque images 6x6 lorsqu'elles sont prêtes.

```
uint32_t count_calls_to_produce_images = 0;

while (count_calls_to_produce_images < NB_CALLS_TO_PRODUCE_IMAGES) {
    if(new_images_are_ready == 1) {
        uint32_t i;
        for (i = 0; i < NB_WINDOWS_TO_PRODUCE_FOR_EACH_CALL_TO_PRODUCE_IMAGES ; i++) {
            float* mask = get_mask(&wp, i);
            float* window = img_buffer[i];
            flux_pondere[i][count_calls_to_produce_images] = computeFluxPondere(window, mask);
        }

        count_calls_to_produce_images++;
        new_images_are_ready = 0;
    }

    if(interruption == 1) {
        ...
    }
}
```

Il est important de comprendre le fonctionnement de la bibliothèque **libwindows-producer.a** pour réussir les exercices du TP4, et c'est pourquoi nous nous sommes attardés sur le dernier exercice du TP 3.

3.3. Les résultats obtenus

```
./sparc-gaisler-elf-gdb C:\Users\lenny\eclipse-  
workspace\EmbarqueeTP3_3\Debug\EmbarqueeTP3_3 -d C:\Users\lenny\eclipse-  
workspace\EmbarqueeTP3_3\src -batch -x C:\Users\lenny\eclipse-  
workspace\EmbarqueeTP3_3\tests\gdb_batch.txt
```

Vous trouverez ci-dessous les résultats obtenues en exécutant le script gdb :

	100ms	200ms	300ms	400ms	500ms	600ms	700ms	800ms	900ms	1000ms
Étoile 1	9752.15137	5705.05566	7828.98633	4164.97266	3882.55664	3643.65918	3995.21021	2263.17993	3087.81885	2117.39819
Étoile 2	12249.8369	17243.2246	14368.084	19519.2031	21593.4688	24878.377	31920.8555	43319.3711	41490.1055	24324.1992
Étoile 3	47512.3828	28626.6602	34134.7031	20446.1582	29193.7949	41713.9492	34957.5586	51639.9492	67143.2578	76419.5
Étoile 4	20128.2441	28621.9141	30251.8223	41658.8047	48502.75	70287.4375	91742.875	127490.141	175133.922	128518.172
Étoile 5	2166.51294	2182.8728	1881.45129	1785.98438	1599.10864	1291.01758	1861.901	1946.19995	2572.52295	2476.62378

Ces valeurs représentent la variation de flux de 5 étoiles au cours d'une seconde. Nous allons utiliser ces données par la suite comme valeurs de références, afin de s'assurer que le code du tp4 est correct.



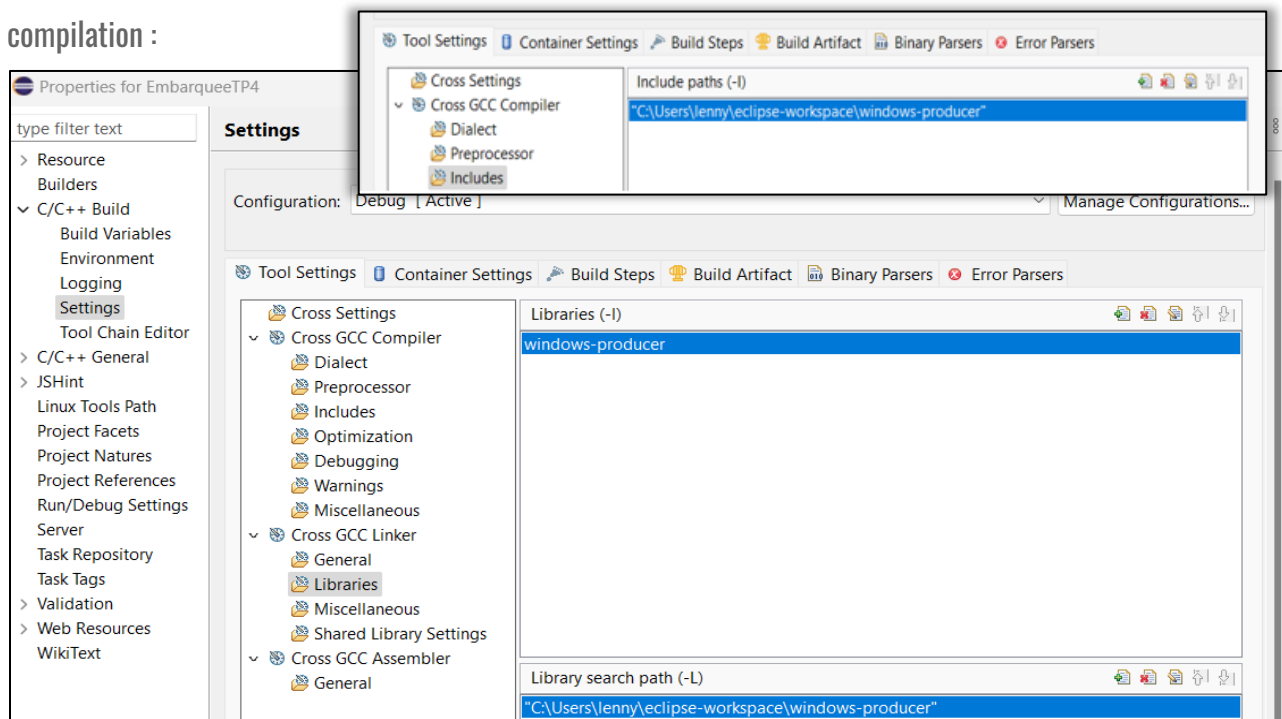
```
$1 = { {9752.15137,  
5705.05566,  
7828.98633,  
4164.97266,  
3882.55664,  
3643.65918,  
3995.21021,  
2263.17993,  
3087.81885,  
2117.39819},  
{12249.8369,  
17243.2246,  
14368.084,  
19519.2031,  
21593.4688,  
24878.377,  
31920.8555,  
43319.3711,  
41490.1055,  
24324.1992},  
{47512.3828,  
28626.6602,  
34134.7031,  
20446.1582,  
29193.7949,  
41713.9492,  
34957.5586,  
51639.9492,  
67143.2578,  
76419.5},  
{20128.2441,  
28621.9141,  
30251.8223,  
41658.8047,  
48502.75,  
70287.4375,  
91742.875,  
127490.141,  
175133.922,  
128518.172},  
{2166.51294,  
2182.8728,  
1881.45129,  
1785.98438,  
1599.10864,  
1291.01758,  
1861.901,  
1946.19995,  
2572.52295,  
2476.62378}}
```

4. Développer une tâche d'acquisition

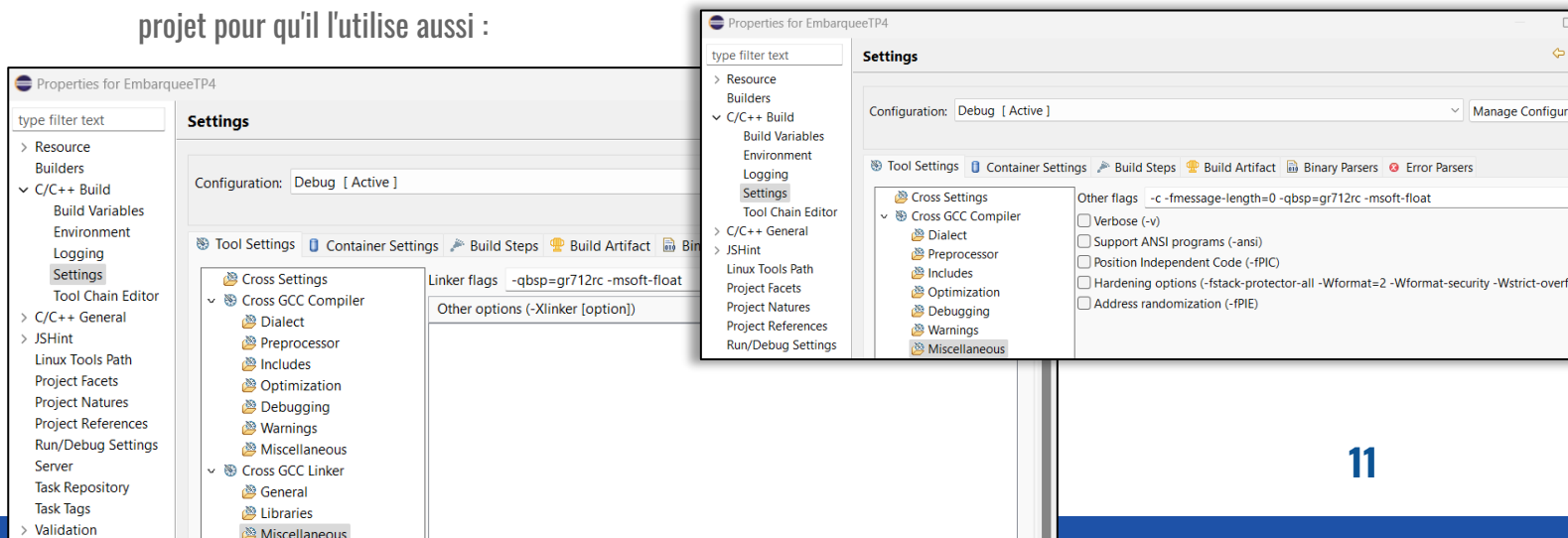
Un RTOS offre des services pour initialiser, configurer et démarrer les tâches. Avec RTEMS, ce sont les fonctions `rtems_task_create()` et `rtems_task_start()` qui permettent de faire cela.

Actuellement, notre programme (issus du code du fichier `rtems-tasks.c`) démarre avec la fonction `Init()`. Celle-ci initialise le temps, crée puis démarre trois tâches ayant pour point d'entrée la fonction `Test_task()`. Enfin, la tâche ayant lancée la fonction `Init()` est effacée afin de forcer RTEMS à exécuter une autre tâche.

À la manière du TP précédents, on intègre la bibliothèque `windows-producer` à notre chaîne de compilation :



De plus, la bibliothèque `windows-producer` étant compilée avec l'option `soft-float`, on configure notre projet pour qu'il l'utilise aussi :



Maintenant, on va créer une tâche qui jouera le rôle d'un driver qui interrogeait le CCD (CCD = un capteur, dans notre cas : **windows-producer**) pour acquérir une image après un temps de pause de 500ms.

La tâche initialise le producteur d'images avec un tableau global pouvant enregistrer 100 imageries de 36 pixels. Dans une boucle infini on appelle **produce_image()** puis on attend 500 ms et on mesure le temps entre 2 appels. Notre tâche doit être créé avec l'option **RTEMS_FLOATING_POINT** pour qu'elle puisse utiliser des nombre flottant.

Afin d'alléger le code, nous supprimons les taches 2 et 3 et on transfère les différentes configurations et définitions de macros dans un fichier **.h** séparer : **configuration.h**.

configuration.h

```
/*
 * COPYRIGHT (c) 1989-1999.
 * On-Line Applications Research Corporation (OAR).
 *
 * The license and distribution terms for this file may be
 * found in the file LICENSE in this distribution or at
 * http://www.OARcorp.com/rtems/license.html.
 */

/* ***** CONFIGURATION INFORMATION ***** */
#include <bsp.h> // For device driver prototypes

#define CONFIGURE_INIT
#define CONFIGURE_APPLICATION_NEEDS_CONSOLE_DRIVER
#define CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER

#define CONFIGURE_MAXIMUM_TASKS          4

#define CONFIGURE_RTEMS_INIT_TASKS_TABLE

#define CONFIGURE_EXTRA_TASK_STACKS      (3 * RTEMS_MINIMUM_STACK_SIZE)

#include <rtems/confdefs.h>

/* If --drvmgr was enabled during the configuration of the RTEMS kernel */

....

/* ***** HANDY MACROS AND STATIC INLINE FUNCTIONS ***** */

....
```

EmbarqueeTP4.c

```
#include <rtems.h>
#include <assert.h> // assert
#include <stdio.h>
#include <stdlib.h>
#include <inttypes.h>
#include "windows-producer.h" // init(), produce_images(), get_mask()

/* ***** MACROS ***** */
#define TASK_STACK_SIZE 10240
#define NB_WINDOWS_TO_PRODUCE_FOR_EACH_CALL_TO_PRODUCE_IMAGES 100

/* ***** FUNCTIONS ***** */
rtems_task Init(rtems_task_argument argument);
rtems_task acquisition_task(rtems_task_argument argument);
float get_elapsed_time();
void break_point(float elapsed_time);

/* ***** GLOBAL VARIABLES ***** */
// Keep the names and IDs in global variables so another task can use them.
rtems_id task_id;
rtems_name task_name;
float img_buffer[NB_WINDOWS_TO_PRODUCE_FOR_EACH_CALL_TO_PRODUCE_IMAGES][36];

/* ***** ***** */
#include "configuration.h" // Including the line : #include <rtems/confdefs.h>

/* ***** AUXILIARY FUNCTIONS ***** */
void break_point(float elapsed_time) {}

float get_elapsed_time() {
    static uint32_t last_time = 0;
    uint32_t elapsed_time = 0, current_time;

    // rtems_interval rtems_clock_get_ticks_since_boot( void );
    current_time = rtems_clock_get_ticks_since_boot();

    elapsed_time = current_time - last_time;
    last_time = current_time;

    return (float)(elapsed_time) / rtems_clock_get_ticks_per_second();
}

/* ***** TASKS ***** */
rtems_task acquisition_task(rtems_task_argument argument) {
    Windows_producer wp;
    rtems_status_code status;
    float elapsed_time;

    // extern char init(Windows_producer* wp, float* img_buffer, unsigned int windows_max_number);
    char init_result = init(&wp, (float *) img_buffer, NB_WINDOWS_TO_PRODUCE_FOR_EACH_CALL_TO_PRODUCE_IMAGES);
    if (init_result == -1)
        exit(EXIT_FAILURE);

    while (1) {
        // extern void produce_images(Windows_producer* wp );
        // Produce new windows for the same star but with an evolved brightness
        produce_images(&wp);

        // rtems_status_code rtems_task_wake_after( rtems_interval ticks );
        // Wakes up after an interval in clock ticks or yields the processor.
        // A system tick worth here 10 ms: i.e. the hardware timer associated with the time manager of the
        // RTOS is set to issue an interrupt every 10 ms. (Sources: Lesson 6, RTEMS Classic API Guide)
        status = rtems_task_wake_after( 50 ); // 50 ticks = 500 ms
        assert(status == RTEMS_SUCCESSFUL);

        elapsed_time = get_elapsed_time();
        break_point(elapsed_time);
    }
}

/* ***** INIT ***** */

rtems_task Init(rtems_task_argument argument) {
    rtems_status_code status;
```

```

task_name = rtems_build_name('T','A','S','K');

// rtems_status_code rtems_task_create(rtems_name name, rtems_task_priority initial_priority, size_t stack_size,
rtems_mode initial_modes, rtems_attribute attribute_set, rtems_id *id);
status = rtems_task_create(task_name, 1, TASK_STACK_SIZE, RTEMS_PREEMPT | RTEMS_NO_TIMESLICE |
RTEMS_INTERRUPT_LEVEL(0), RTEMS_LOCAL | RTEMS_FLOATING_POINT, &task_id);
assert(status == RTEMS_SUCCESSFUL);

// rtems_status_code rtems_task_start(rtems_id id, rtems_task_entry entry_point, rtems_task_argument argument)
status = rtems_task_start(task_id, acquisition_task, 1);
assert(status == RTEMS_SUCCESSFUL);

status = rtems_task_delete( RTEMS_SELF );
assert(status == RTEMS_SUCCESSFUL);
}

```

La fonction `get_elapsed_time()` permet d'obtenir le temps écoulé entre deux acquisitions. Elle utilise la fonction `rtems_clock_get_ticks_since_boot()` qui retourne le nombre de ticks système écoulés depuis le démarrage de l'application et ensuite on calcule le temps écoulé entre deux acquisitions par une simple soustraction entre l'appel à `rtems_clock_get_ticks_since_boot()` à l'instant `t` et l'instant `t - 1`.

Afin d'obtenir les résultats on va utiliser le script gbd suivant :

gdb_batch.txt

```

# Sources des commentaires :
# le cours et le manuel de l'utilisateur du simulateur TSIM3
# https://www.gaisler.com/doc/tsim3.pdf

# tar = target
# Se connecter à TSIM à l'aide de extended-remote protocol
tar extended-remote localhost:1234

# Charger l'image du programme dans le simulateur
load

mon perf reset

# Declaration d'un point d'arrêt via la commande hbreak
hbreak break_point
commands
    silent
    printf "Elapsed time : %.2f sec \n", elapsed_time
    cont
end

start

```

Ensuite, on exécute ce script à l'aide de la commande :

```
./sparc-gaisler-rtems5-gdb C:\Users\lenny\eclipse-  
workspace\EmbarqueeTP4\Debug\EmbarqueeTP4 -d C:\Users\lenny\eclipse-  
workspace\EmbarqueeTP4\src -batch -x C:\Users\lenny\eclipse-  
workspace\EmbarqueeTP4\tests\gdb_batch.txt
```

```
Hardware assisted breakpoint 1 at 0x400012e8: file ../src/EmbarqueeTP4.c, line 30.  
Function "main" not defined.  
Make breakpoint pending on future shared library load? (y or [n]) [answered N; input not from terminal]  
[New Thread 2]  
Elapsed time : 0.59 sec  
Elapsed time : 0.58 sec  
Elapsed time : 0.58 sec  
Elapsed time : 0.58 sec  
Elapsed time : 0.58 sec  
Elapsed time : 0.58 sec  
Elapsed time : 0.58 sec
```

Le temps entre le démarrage et la première acquisitions est 0.59 sec, et le temps entre les acquisitions n et $n + 1$ est égale à 0.58 sec. Par conséquent, la contrainte d'acquérir une image chaque 500 ms n'a pas été respectée vu le temps de réponse de la fonction `produce_images()`.

5. Utiliser un timer et une queue de messages

Dans le domaine des applications temps réel et embarquées, il est souvent nécessaire de déclencher des traitements périodiques. Ceci peut être réalisé, par exemple, en utilisant des tâches qui vont s'endormir pendant x ticks systèmes. De plus, la plupart des RTOS intègre des composants « timer logiciel » permettant de faciliter la mise en œuvre de ces traitements périodiques.⁴

Dans notre cas, pour que la période d'acquisition ne soit pas dépendante du temps de réponse de la fonction `produce_images()`, on remplace l'utilisation des pauses par l'utilisation d'un timer et d'une queue de messages.

Pour cela, dans la fonction `Init()` on a créé un timer et on l'a armé pour se déclencher au bout d'une demi-seconde, puis, on l'a lié à une nouvelle fonction dans laquelle on réarme le timer après chaque déclenchement :

```
// rtems_timer_create()
// rtems_status_code rtems_timer_create( rtems_name name, rtems_id *id );
// This directive creates a timer which resides on the local node. The timer has the user-defined
// object name specified in name. The assigned object identifier is returned in id. This identifier is
// used to access the timer with other timer related directives.
// Source : RTEMS Classic API Guide
timer_name = rtems_build_name('T','I','M','E');
status = rtems_timer_create(timer_name, &timer_id);
assert(status == RTEMS_SUCCESSFUL);

// rtems_status_code rtems_timer_fire_after(rtems_id id, rtems_interval ticks, rtems_timer_service_routine_entry
routine, void *user_data);
// Fires (starts) the timer after the interval.
// This directive initiates the timer specified by id. If the timer is running, it is automatically
// canceled before being initiated. The timer is scheduled to fire after an interval of clock ticks
// has passed specified by ticks. When the timer fires, the timer service routine routine will be
// invoked with the argument user_data in the context of the clock tick ISR.
// Source : RTEMS Classic API Guide
status = rtems_timer_fire_after(timer_id, rtems_clock_get_ticks_per_second() / 2, reload_timer ,NULL);
assert(status == RTEMS_SUCCESSFUL);

// rtems_status_code rtems_message_queue_create(rtems_name name, uint32_t count, size_t max_message_size,
rtems_attribute attribute_set, rtems_id *id);
status = rtems_message_queue_create(rtems_build_name('M','S','Q','1'), MESSAGE_QUEUE_COUNT, MESSAGE_QUEUE_SIZE,
RTEMS_LOCAL | RTEMS_PRIORITY, &message_queue_id);
assert(status == RTEMS_SUCCESSFUL);
```

⁴ Cours 6 - Systèmes d'exploitation temps réel (RTOS)

On définit ensuite les macros de configuration (`CONFIGURE_MAXIMUM_TIMERS`, `CONFIGURE_MAXIMUM_MESSAGE_QUEUES`) avant d'inclure `configuration.h` qui contient l'inclusion de `<rtems/confdefs.h>`.

```
/* ***** RTEMS CONFIGURATION MACROS ***** */
// The RTEMS configuration macros must be located before #include <rtems/confdefs.h>
#define CONFIGURE_MAXIMUM_TIMERS 5
#define CONFIGURE_MAXIMUM_MESSAGE_QUEUES 10
```

Pour assurer la synchronisation des acquisitions sur le déclenchement du timer, on a créé une queue de messages dans `Init()`, et dans la fonction `reload_timer()` on poste un message dans la queue à chaque déclenchement du timer. Cela permet la réception des messages par la tâche `acquisition_task`. Voici donc le code de la fonction `reload_timer()` :

```
void reload_timer(unsigned int arg1, void* arg2) {
    rtems_status_code status;
    float elapsed_time;
    uint32_t buffer;

    status = rtems_timer_fire_after(timer_id, rtems_clock_get_ticks_per_second() / 2, reload_timer, NULL);
    assert(status == RTEMS_SUCCESSFUL);

    do{
        buffer = 1;
        // rtems_status_code rtems_message_queue_send(rtems_id id, const void *buffer, size_t size);
        status = rtems_message_queue_send(message_queue_id, buffer, sizeof(uint32_t));
    }while(status == RTEMS_TOO_MANY);

    elapsed_time = get_elapsed_time();
    break_point(elapsed_time);
}
```

Le code est optimal, en particulier la boucle `do while`, nous allons optimiser cette fonction par la suite. Enfin, voici la boucle de notre tâche d'acquisition qui fait un appel à `produce_images()` après chaque lecture réussie de message.

```
while (1) {
    // rtems_status_code rtems_message_queue_receive(rtems_id id, void *buffer, size_t *size, rtems_option
    option_set, rtems_interval timeout);
    status = rtems_message_queue_receive(message_queue_id, &buffer, &size, RTEMS_WAIT, RTEMS_NO_TIMEOUT);

    if(status == RTEMS_SUCCESSFUL) {
        // extern void produce_images(Windows_producer* wp );
        // Produce new windows for the same star but with an evolved brightness
        produce_images(&wp);
    }
}
```

Après avoir ajouté un timer et une queue de messages au lieu d'utiliser des pauses, on a configuré un point d'arrêt à chaque fois qu'on recharge (`reload_timer`) le timer, cela nous a permis d'avoir le résultat ci-dessous.

```
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
```

Le temps écoulé entre chaque deux acquisitions est égal à 0,50 secondes, en le comparant avec le résultat obtenu en utilisant les pauses on peut en déduire les critères suivants:

	En utilisant des pauses	En utilisant un timer et une queue de messages
temps écoulé	0.58 sec	0.50 sec
respecte la contrainte de temps 500 ms	non (prends plus de temps)	oui

La contrainte de temps est donc respectée maintenant (exécution chaque 500 ms), la périodicité est garantie.

6. Ajouter une tâche de traitement

Si l'acquisition doit être réalisée dans une tâche de haute priorité pour garantir sa périodicité, le traitement des données n'a pas la même contrainte. Nous réaliserons donc cette opération dans une autre tâche, de moindre priorité.

On crée donc une nouvelle tâche dans `Init()` et une queue de messages supplémentaire qui permettra à la tâche d'acquisition d'indiquer à la tâche de traitement que le buffer d'images a été mis à jour. On ajoute l'émission du message par la tâche d'acquisition et, au début de la boucle de la tâche de traitement, on attend la réception de ce message. Le message contiendra la valeur du compteur d'acquisitions. Une fois le message reçu, la photométrie par masques pondérés est calculé pour chacune des images :

```
rtems_task processing_task(rtems_task argument) {
    rtems_status_code status;
    uint32_t acquisition_counter;
    size_t size;

    while (1) {
        // rtems_status_code rtems_message_queue_receive(rtems_id id, void *buffer, size_t *size, rtems_option
        option_set, rtems_interval timeout);
        status = rtems_message_queue_receive(message_queue_synchronize_acquisition_processing_id,
        &acquisition_counter, &size, RTEMS_WAIT, RTEMS_NO_TIMEOUT);

        if(status == RTEMS_SUCCESSFUL) {
            uint32_t i;
            for (i = 0; i < NB_WINDOWS_TO_PRODUCE_FOR_EACH_CALL_TO_PRODUCE_IMAGES ; i++) {
                float* mask = get_mask(&wp, i);
                float* window = img_buffer[ flux_pondere[i].id_window ];

                flux_pondere[i].measures[flux_pondere[i].id_first_acquisition +
                acquisition_counter] = computeFluxPondere(window, mask);
            }
            acquisition_counter++;
        }
    }
}
```

```
rtems_task acquisition_task(rtems_task argument) {
    rtems_status_code status;
    uint32_t acquisition_counter = -1, timer_count;
    size_t size;

    // extern char init(Windows_producer* wp, float* img_buffer, unsigned int windows_max_number);
    // Initialize the windows producer. Must be call before produce_images()
    char init_result = init(&wp, (float*) &img_buffer, NB_WINDOWS_TO_PRODUCE_FOR_EACH_CALL_TO_PRODUCE_IMAGES);
    if (init_result == -1)
        exit(EXIT_FAILURE);

    while (1) {
        // rtems_status_code rtems_message_queue_receive(rtems_id id, void *buffer, size_t *size, rtems_option
        option_set, rtems_interval timeout);
        status = rtems_message_queue_receive(message_queue_synchronize_timer_acquisition_id, &timer_count,
        &size, RTEMS_WAIT, RTEMS_NO_TIMEOUT);

        if(status == RTEMS_SUCCESSFUL) {
            for(uint32_t i = 0 ; i < timer_count ; i++){
                // extern void produce_images(Windows_producer* wp );
                // Produce new windows for the same star but with an evolved brightness
            }
        }
    }
}
```

```

        produce_images(&wp);
        acquisition_counter++;

        // rtems_status_code rtems_message_queue_send(rtems_id id, const void *buffer,
size_t size);

        status =
rtems_message_queue_send(message_queue_synchronize_acquisition_processing_id, &acquisition_counter, sizeof(uint32_t));
    }
}
}
}

```

Et voici donc l'intégralité du code du fichier source principale :

```

#include <rtems.h>
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <inttypes.h>
#include "windows-producer.h" // init(), produce_images(), get_mask()
#include "algo.h" // computeFluxPondere()

/* ***** MACROS ***** */
#define TASK_STACK_SIZE 10240

#define MESSAGE_QUEUE_COUNT 1
#define MESSAGE_QUEUE_SIZE 4

#define ACQUISITION_TASK_PRIORITY 1
#define PROCESSING_TASK_PRIORITY 2

#define NB_WINDOWS_TO_PRODUCE_FOR_EACH_CALL_TO_PRODUCE_IMAGES 100

#define FLUX_LENGTH 10

/* ***** RTEMS CONFIGURATION MACROS ***** */
// The RTEMS configuration macros must be located before #include <rtems/confdefs.h>
#define CONFIGURE_MAXIMUM_TIMERS 5
#define CONFIGURE_MAXIMUM_MESSAGE_QUEUES 10

/* ***** FUNCTIONS ***** */
rtems_task Init(rtems_task_argument argument);
rtems_task acquisition_task(rtems_task_argument argument);
rtems_task processing_task(rtems_task_argument argument);

float get_elapsed_time();
void reload_timer(unsigned int arg1, void* arg2);
void break_point(float elapsed_time);

/* ***** DATA STRUCTURES ***** */
typedef struct flux{
    uint32_t id_window ;
    uint32_t id_first_acquisition ; ///< measurement acquisition counter value[0]
    float measures[FLUX_LENGTH];
} flux ;

/* ***** GLOBAL VARIABLES ***** */
// Keep the names and IDs in global variables so another task can use them.
rtems_id acquisition_task_id, processing_task_id, timer_id, message_queue_synchronize_timer_acquisition_id,
message_queue_synchronize_acquisition_processing_id;
rtems_name acquisition_task_name, processing_task_name, timer_name, message_queue_synchronize_timer_acquisition_name,
message_queue_synchronize_acquisition_processing_name;

float img_buffer[NB_WINDOWS_TO_PRODUCE_FOR_EACH_CALL_TO_PRODUCE_IMAGES][36];
flux flux_pondere[NB_WINDOWS_TO_PRODUCE_FOR_EACH_CALL_TO_PRODUCE_IMAGES];

Windows_producer wp;

/* ***** AUXILIARY FUNCTIONS ***** */

#include "configuration.h" // Including the line : #include <rtems/confdefs.h>

/* ***** AUXILIARY FUNCTIONS ***** */

```

```

void break_point(float elapsed_time) {}

float get_elapsed_time() {
    static uint32_t last_time = 0;
    uint32_t elapsed_time = 0, current_time;

    // rtems_interval rtems_clock_get_ticks_since_boot( void );
    // Gets the number of clock ticks since some time point during the system initialization or the last overflow of
the clock tick counter
    // Source : RTEMS Classic API Guide
    current_time = rtems_clock_get_ticks_since_boot();

    elapsed_time = current_time - last_time;
    last_time = current_time;

    return (float)(elapsed_time) / rtems_clock_get_ticks_per_second();
}

void reload_timer(unsigned int arg1, void* arg2) {
    rtems_status_code status;
    float elapsed_time;
    static uint32_t timer_count = 0;

    status = rtems_timer_fire_after(timer_id, rtems_clock_get_ticks_per_second() / 2, reload_timer ,NULL);
    assert(status == RTEMS_SUCCESSFUL);

    timer_count++;

    // rtems_status_code rtems_message_queue_send(rtems_id id, const void *buffer, size_t size);
    status = rtems_message_queue_send(message_queue_synchronize_timer_acquisition_id, &timer_count,
sizeof(uint32_t));
    if (status == RTEMS_SUCCESSFUL)
        timer_count = 0;

    elapsed_time = get_elapsed_time();
    break_point(elapsed_time);
}

/* ***** TASKS ***** */

rtems_task acquisition_task(rtems_task_argument argument) {
    rtems_status_code status;
    uint32_t acquisition_counter = -1, timer_count;
    size_t size;

    // extern char init(Windows_producer* wp, float* img_buffer, unsigned int windows_max_number);
    // Initialize the windows producer. Must be call before produce_images()
    char init_result = init(&wp, (float*) &img_buffer, NB_WINDOWS_TO_PRODUCE_FOR_EACH_CALL_TO_PRODUCE_IMAGES);
    if (init_result == -1)
        exit(EXIT_FAILURE);

    while (1) {
        // rtems_status_code rtems_message_queue_receive(rtems_id id, void *buffer, size_t *size, rtems_option
option_set, rtems_interval timeout);
        status = rtems_message_queue_receive(message_queue_synchronize_timer_acquisition_id, &timer_count,
&size, RTEMS_WAIT, RTEMS_NO_TIMEOUT);

        if(status == RTEMS_SUCCESSFUL) {
            for(uint32_t i = 0 ; i < timer_count ; i++){
                // extern void produce_images(Windows_producer* wp );
                // Produce new windows for the same star but with an evolved brightness
                produce_images(&wp);
                acquisition_counter++;

                // rtems_status_code rtems_message_queue_send(rtems_id id, const void *buffer,
size_t size);
                status =
rtems_message_queue_send(message_queue_synchronize_acquisition_processing_id, &acquisition_counter, sizeof(uint32_t));
            }
        }
    }
}

rtems_task processing_task(rtems_task_argument argument) {
    rtems_status_code status;
    uint32_t acquisition_counter;

```

```

        size_t size;

        while (1) {
            // rtems_status_code rtems_message_queue_receive(rtems_id id, void *buffer, size_t *size, rtems_option
            option_set, rtems_interval timeout);
            status = rtems_message_queue_receive(message_queue_synchronize_acquisition_processing_id,
            &acquisition_counter, &size, RTEMS_WAIT, RTEMS_NO_TIMEOUT);

            if(status == RTEMS_SUCCESSFUL) {
                uint32_t i;
                for (i = 0; i < NB_WINDOWS_TO_PRODUCE_FOR_EACH_CALL_TO_PRODUCE_IMAGES ; i++) {
                    float* mask = get_mask(&wp, i);
                    float* window = img_buffer[ flux_pondere[i].id_window ];

                    flux_pondere[i].measures[flux_pondere[i].id_first_acquisition +
                    acquisition_counter] = computeFluxPondere(window, mask);
                }
                acquisition_counter++;
            }
        }
    }

/* ***** INIT ***** */

rtems_task Init(rtems_task_argument argument) {
    rtems_status_code status;

    for (uint32_t i = 0; i < NB_WINDOWS_TO_PRODUCE_FOR_EACH_CALL_TO_PRODUCE_IMAGES ; i++) {
        flux_pondere[i].id_window = i;
        flux_pondere[i].id_first_acquisition = 0;
    }

    acquisition_task_name = rtems_build_name('A','C','U','I');
    processing_task_name = rtems_build_name('P','R','O','C');
    timer_name = rtems_build_name('T','I','M','E');
    message_queue_synchronize_timer_acquisition_name = rtems_build_name('T','M','A','C');
    message_queue_synchronize_acquisition_processing_name = rtems_build_name('A','C','T','M');

    // rtems_status_code rtems_timer_create( rtems_name name, rtems_id *id );
    status = rtems_timer_create(timer_name, &timer_id);
    assert(status == RTEMS_SUCCESSFUL);

    // rtems_status_code rtems_timer_fire_after(rtems_id id, rtems_interval ticks, rtems_timer_service_routine_entry
    routine, void *user_data);
    status = rtems_timer_fire_after(timer_id, rtems_clock_get_ticks_per_second() / 2, reload_timer ,NULL);
    assert(status == RTEMS_SUCCESSFUL);

    // rtems_status_code rtems_message_queue_create(rtems_name name, uint32_t count, size_t max_message_size,
    rtems_attribute attribute_set, rtems_id *id);
    status = rtems_message_queue_create(message_queue_synchronize_timer_acquisition_name, MESSAGE_QUEUE_COUNT,
    MESSAGE_QUEUE_SIZE, RTEMS_LOCAL | RTEMS_PRIORITY, &message_queue_synchronize_timer_acquisition_id);
    assert(status == RTEMS_SUCCESSFUL);
    status = rtems_message_queue_create(message_queue_synchronize_acquisition_processing_name, MESSAGE_QUEUE_COUNT,
    MESSAGE_QUEUE_SIZE, RTEMS_LOCAL | RTEMS_PRIORITY, &message_queue_synchronize_acquisition_processing_id);
    assert(status == RTEMS_SUCCESSFUL);

    // rtems_status_code rtems_task_create(rtems_name name, rtems_task_priority initial_priority, size_t stack_size,
    rtems_mode initial_modes, rtems_attribute attribute_set, rtems_id *id);
    status = rtems_task_create(acquisition_task_name, ACQUISITION_TASK_PRIORITY, TASK_STACK_SIZE, RTEMS_PREEMPT |
    RTEMS_NO_TIMESLICE | RTEMS_ASR | RTEMS_INTERRUPT_LEVEL(0), RTEMS_LOCAL | RTEMS_FLOATING_POINT, &acquisition_task_id);
    assert(status == RTEMS_SUCCESSFUL);
    status = rtems_task_create(processing_task_name, ACQUISITION_TASK_PRIORITY, TASK_STACK_SIZE, RTEMS_PREEMPT |
    RTEMS_NO_TIMESLICE | RTEMS_ASR | RTEMS_INTERRUPT_LEVEL(0), RTEMS_LOCAL | RTEMS_FLOATING_POINT, &processing_task_id);
    assert(status == RTEMS_SUCCESSFUL);

    // rtems_status_code rtems_task_start(rtems_id id, rtems_task_entry entry_point, rtems_task_argument argument)
    status = rtems_task_start(acquisition_task_id, acquisition_task, 0);
    assert(status == RTEMS_SUCCESSFUL);
    status = rtems_task_start(processing_task_id, processing_task, 0);
    assert(status == RTEMS_SUCCESSFUL);

    status = rtems_task_delete( RTEMS_SELF );
    assert(status == RTEMS_SUCCESSFUL);
}

```

On remarquera que dans la fonction `reload_timer()` on a changé légèrement le code :

```
void reload_timer(unsigned int arg1, void* arg2) {
    rtems_status_code status;
    float elapsed_time;
    static uint32_t timer_count = 0;

    status = rtems_timer_fire_after(timer_id, rtems_clock_get_ticks_per_second() / 2, reload_timer, NULL);
    assert(status == RTEMS_SUCCESSFUL);

    timer_count++;

    // rtems_status_code rtems_message_queue_send(rtems_id id, const void *buffer, size_t size);
    status = rtems_message_queue_send(message_queue_synchronize_timer_acquisition_id, &timer_count,
sizeof(uint32_t));
    if (status == RTEMS_SUCCESSFUL)
        timer_count = 0;

    elapsed_time = get_elapsed_time();
    break_point(elapsed_time);
}
```

Explication : si on n'arrive pas à poster un message, on cumule le nombre de fois que le timer a été déclenché depuis le dernier message qu'on a envoyé, cette valeur représente en effet le nombre d'acquisition que la tâche d'acquisition doit effectuer :

```
while (1) {
    // rtems_status_code rtems_message_queue_receive(rtems_id id, void *buffer, size_t *size, rtems_option
option_set, rtems_interval timeout);
    status = rtems_message_queue_receive(message_queue_synchronize_timer_acquisition_id, &timer_count,
&size, RTEMS_WAIT, RTEMS_NO_TIMEOUT);

    if(status == RTEMS_SUCCESSFUL) {
        for(uint32_t i = 0 ; i < timer_count ; i++){
            // extern void produce_images(Windows_producer* wp );
            // Produce new windows for the same star but with an evolved brightness
            produce_images(&wp);
            acquisition_counter++;

            // rtems_status_code rtems_message_queue_send(rtems_id id, const void *buffer,
size_t size);
            status =
rtems_message_queue_send(message_queue_synchronize_acquisition_processing_id, &acquisition_counter, sizeof(uint32_t));
        }
    }
}
```

Cependant, nous sommes pas certains que cette manœuvre été nécessaire car selon nos vérifications, la valeur de `timer_count` est égal à 1 à chaque fois que cette valeur est transféré entre les tâches. Cependant, cette solution est sans doute bien meilleur que celle utilisé à la section précédente (une boucle infini jusqu'à qu'on arrive à envoyer le message).

Afin de tester notre code, on change temporairement la macro suivante :

```
#define NB_WINDOWS_TO_PRODUCE_FOR_EACH_CALL_TO_PRODUCE_IMAGES 5
```

Et dans notre script `gbd` on change légèrement le point d'arrêt :


```
# Declaration d'un point d'arrêt via la commande hbreak
hbreak break_point
commands
    silent
    printf "Elapsed time : %.2f sec \n", elapsed_time
    print flux_pondere
cont
end
```

On exécute le script gdb :

```
./sparc-gaisler-rtems5-gdb C:\Users\lenny\eclipse-
workspace\EmbarqueeTP4_3\Debug\EmbarqueeTP4_3 -d C:\Users\lenny\eclipse-
workspace\EmbarqueeTP4_3\src -batch -x C:\Users\lenny\eclipse-
workspace\EmbarqueeTP4_3\tests\gdb_batch.txt
```

Et on obtient le résultat suivant :

```
$11 = {{id_window = 0, id_first_acquisition = 0, measures = {9752.151
37, 5705.05566, 7828.98633, 4164.97266, 3882.55664, 3643.65918, 3995.
21021, 2263.17993, 3087.81885, 2117.39819}}, {id_window = 1, id_first
_acquisition = 0, measures = {12249.8369, 17243.2246, 14368.084, 1951
9.2031, 21593.4688, 24878.377, 31920.8555, 43319.3711, 41490.1055, 24
324.1992}}, {id_window = 2, id_first_acquisition = 0, measures = {475
12.3828, 28626.6602, 34134.7031, 20446.1582, 29193.7949, 41713.9492,
34957.5586, 51639.9492, 67143.2578, 76419.5}}, {id_window = 3, id_fir
st_acquisition = 0, measures = {20128.2441, 28621.9141, 30251.8223, 4
1658.8047, 48502.75, 70287.4375, 91742.875, 127490.141, 175133.922, 1
28518.172}}, {id_window = 4, id_first_acquisition = 0, measures = {21
66.51294, 2182.8728, 1881.45129, 1785.98438, 1599.10864, 1291.01758,
1861.901, 1946.19995, 2572.52295, 2476.62378}}}
```

On remarquera qu'on obtient les mêmes résultats qu'on a déjà obtenu pour le dernier exercice de tp3, cela confirme que notre démarche est correcte. Cependant, nous avons pas encore fait le nécessaire afin que le procédure ci-dessus recommence à l'infini, à chaque fois avec une nouvelle valeur pour le champ `id_first_acquisition`. Cela se fera par la suite.

7. Transmission des données entre tâches et création d'une tâche de gestion de télémétrie

Pour transmettre ces données vers une tâche gérant l'émission des télémétries (TM), on va poster ces structures dans un buffer circulaire. Un buffer circulaire est un composant capable de stocker un certain nombre de données puis de les récupérer dans l'ordre où elles ont été ajoutées, selon le principe First In First Out (FIFO). Aussi, que le buffer soit plein ou vide, il n'empêche pas la lecture ou l'écriture des données.

Afin d'utiliser cette structure de données de manière atomique en situation de programmation multi-tâches nous avons créé les fichiers `buffer_circulaire_partagee.h` et `buffer_circulaire_partagee.c` qui contient l'encapsulation de la structure `buffer_circulaire`.

`buffer_circulaire_partagee.h`

```
/*
 * buffer_circulaire_partagee.h
 */

#ifndef BUFFER_CIRCULAIRE_PARTAGEE_H_
#define BUFFER_CIRCULAIRE_PARTAGEE_H_

#include <rtems.h>
#include "buffer_circulaire.h"

extern rtems_id semaphore_id;
extern rtems_name semaphore_name;

typedef struct buffer_circulaire_partagee {
    buffer_circulaire buffer;
} buffer_circulaire_partagee;

void init_buffer_circulaire_partagee(buffer_circulaire_partagee* b, uint8_t* buffer, uint16_t taille_donnee, uint32_t
nombre_donnees);
int push_partagee(buffer_circulaire_partagee* fifo, uint8_t* source, uint16_t taille);
int pop_partagee(buffer_circulaire_partagee* fifo, uint8_t* destination, uint16_t taille_max);
unsigned int test_unitaire_buffer_circulaire_partagee();

#endif /* BUFFER_CIRCULAIRE_PARTAGEE_H_ */
```

buffer_circulaire_partagee.c

```
/*
 * buffer_circulaire_partagee.c
 */
#include <rtems.h>
#include <assert.h> // assert
#include "buffer_circulaire_partagee.h"
#include "flux.h"

rtems_id semaphore_id;
rtems_name semaphore_name;

void init_buffer_circulaire_partagee(buffer_circulaire_partagee* b, uint8_t* buffer, uint16_t taille_donnee, uint32_t
nombre_donnees) {
    rtems_status_code status;
    semaphore_name = rtems_build_name('S', 'E', 'M', 'A');

    init_buffer_circulaire(&(b->buffer), buffer, taille_donnee, nombre_donnees);

    // rtems_status_code rtems_semaphore_create(rtems_name name, uint32_t count, rtems_attribute attribute_set,
rtems_task_priority priority_ceiling, rtems_id* id);
    status = rtems_semaphore_create(semaphore_name, 1, RTEMS_PRIORITY | RTEMS_INHERIT_PRIORITY |
RTEMS_BINARY_SEMAPHORE, 0, &semaphore_id);
    assert(status == RTEMS_SUCCESSFUL);
}

int push_partagee(buffer_circulaire_partagee* fifo, uint8_t* source, uint16_t taille) {
    rtems_status_code status;
    int result;
    // rtems_status_code rtems_semaphore_obtain(rtems_id id, rtems_option option_set, rtems_interval timeout);
    status = rtems_semaphore_obtain(semaphore_id, RTEMS_WAIT, RTEMS_NO_TIMEOUT);
    assert(status == RTEMS_SUCCESSFUL);

    result = push(&(fifo->buffer), source, taille);

    // rtems_status_code rtems_semaphore_release( rtems_id id );
    status = rtems_semaphore_release(semaphore_id);
    assert(status == RTEMS_SUCCESSFUL);

    return result;
}

int pop_partagee(buffer_circulaire_partagee* fifo, uint8_t* destination, uint16_t taille_max) {
    rtems_status_code status;
    int result;

    // rtems_status_code rtems_semaphore_obtain(rtems_id id, rtems_option option_set, rtems_interval timeout);
    status = rtems_semaphore_obtain(semaphore_id, RTEMS_WAIT, RTEMS_NO_TIMEOUT);
    assert(status == RTEMS_SUCCESSFUL);

    result = pop(&(fifo->buffer), destination, taille_max);

    // rtems_status_code rtems_semaphore_release( rtems_id id );
    status = rtems_semaphore_release(semaphore_id);
    assert(status == RTEMS_SUCCESSFUL);

    return result;
}
```

Un changement supplémentaire qu'on a effectué est de transférer la déclaration de la structure flux dans un fichier `.h` séparé :

flux.h

```
/*
 * flux.h
 */

#ifndef FLUX_H_
#define FLUX_H_

#define FLUX_LENGTH 10

typedef struct flux{
    uint32_t id_window ;
    uint32_t id_first_acquisition ; ///< measurement acquisition counter value[0]
    float measures[FLUX_LENGTH];
} flux ;

#endif /* FLUX_H_ */
```

Et voici donc la mise a jour de notre fichier source principale:

```
#include <rtems.h>
#include <assert.h> // assert
#include <stdio.h>
#include <stdlib.h>
#include <inttypes.h>
#include "windows-producer.h" // init(), produce_images(), get_mask()
#include "algo.h" // computeFluxPondere()
#include "buffer_circulaire_partagee.h" // init_buffer_circulaire_partagee()
#include "flux.h" // flux, FLUX_LENGTH

/* ***** MACROS ***** */
#define TASK_STACK_SIZE 1000

#define MESSAGE_QUEUE_COUNT 1
#define MESSAGE_QUEUE_SIZE 4

#define ACQUISITION_TASK_PRIORITY 1
#define PROCESSING_TASK_PRIORITY 2
#define TELEMETRY_MANAGER_TASK_PRIORITY 3

#define NB_WINDOWS_TO_PRODUCE_FOR_EACH_CALL_TO_PRODUCE_IMAGES 100

/* ***** RTEMS CONFIGURATION MACROS ***** */
// The RTEMS configuration macros must be located before #include <rtems/confdefs.h>
#define CONFIGURE_MAXIMUM_TIMERS 5
#define CONFIGURE_MAXIMUM_MESSAGE_QUEUES 10
#define CONFIGURE_MAXIMUM_SEMAPHORES 3

/* ***** FUNCTIONS ***** */
rtems_task Init(rtems_task_argument argument);
rtems_task acquisition_task(rtems_task_argument argument);
rtems_task processing_task(rtems_task_argument argument);
rtems_task telemetry_manager_task(rtems_task_argument argument);

float get_elapsed_time();
void reload_timer(unsigned int arg1, void* arg2);
void break_point(float elapsed_time);
void send_flux(uint32_t time, flux f);

/* ***** GLOBAL VARIABLES ***** */
// Keep the names and IDs in global variables so another task can use them.
rtems_id acquisition_task_id, processing_task_id, telemetry_manager_task_id;
rtems_id timer_id, message_queue_synchronize_timer_acquisition_id, message_queue_synchronize_acquisition_processing_id;

rtems_name acquisition_task_name, processing_task_name, telemetry_manager_task_name;
rtems_name timer_name, message_queue_synchronize_timer_acquisition_name,
message_queue_synchronize_acquisition_processing_name;

float img_buffer[NB_WINDOWS_TO_PRODUCE_FOR_EACH_CALL_TO_PRODUCE_IMAGES][36];
```

```

flux flux_pondere[NB_WINDOWS_TO_PRODUCE_FOR_EACH_CALL_TO_PRODUCE_IMAGES];
flux flux_pondere_fifo[NB_WINDOWS_TO_PRODUCE_FOR_EACH_CALL_TO_PRODUCE_IMAGES];

buffer_circulaire_partagee circular_buffer;

Windows_producer wp;

/* ***** */

#include "configuration.h" // Including the line : #include <rtems/confdefs.h>

/* ***** AUXILIARY FUNCTIONS ***** */

void break_point(float elapsed_time) {}

float get_elapsed_time() {
    static uint32_t last_time = 0;
    uint32_t elapsed_time = 0, current_time;

    // rtems_interval rtems_clock_get_ticks_since_boot( void );
    // Gets the number of clock ticks since some time point during the system initialization or the last overflow of
the clock tick counter
    // Source : RTEMS Classic API Guide
    current_time = rtems_clock_get_ticks_since_boot();

    elapsed_time = current_time - last_time;
    last_time = current_time;

    return (float)(elapsed_time) / rtems_clock_get_ticks_per_second();
}

void reload_timer(unsigned int arg1, void* arg2) {
    rtems_status_code status;
    float elapsed_time;
    static uint32_t timer_count = 0;

    status = rtems_timer_fire_after(timer_id, rtems_clock_get_ticks_per_second() / 2, reload_timer ,NULL);
    assert(status == RTEMS_SUCCESSFUL);

    timer_count++;

    // rtems_status_code rtems_message_queue_send(rtems_id id, const void *buffer, size_t size);
    status = rtems_message_queue_send(message_queue_synchronize_timer_acquisition_id, &timer_count,
sizeof(uint32_t));
    if (status == RTEMS_SUCCESSFUL)
        timer_count = 0;

    elapsed_time = get_elapsed_time();
    break_point(elapsed_time);
}

/**
 * Simule l'émission d'une séquence de mesures de photométrie.
 * @param f structure enregistrant la séquence
 * @param time moment de l'émission, exprimé en ticks depuis le démarrage du programme .
 */
void send_flux(uint32_t time, flux f){
    static int cpt = 0 ;
    cpt++ ;
}

/* ***** TASKS ***** */

rtems_task acquisition_task(rtems_task_argument argument) {
    rtems_status_code status;
    uint32_t acquisition_counter = -1, timer_count;
    size_t size;
    // extern char init(Windows_producer* wp, float* img_buffer, unsigned int windows_max_number);
    // Initialize the windows producer. Must be call before produce_images()
    char init_result = init(&wp, (float*) &img_buffer, NB_WINDOWS_TO_PRODUCE_FOR_EACH_CALL_TO_PRODUCE_IMAGES);
    if (init_result == -1)
        exit(EXIT_FAILURE);

    while (1) {
        // rtems_status_code rtems_message_queue_receive(rtems_id id, void *buffer, size_t *size, rtems_option
option_set, rtems_interval timeout);
        status = rtems_message_queue_receive(message_queue_synchronize_timer_acquisition_id, &timer_count,
&size, RTEMS_WAIT, RTEMS_NO_TIMEOUT);

```

```

        if(status == RTEMS_SUCCESSFUL) {
            for(uint32_t i = 0 ; i < timer_count ; i++){
                // extern void produce_images(Windows_producer* wp );
                // Produce new windows for the same star but with an evolved brightness
                produce_images(&wp);
                acquisition_counter++;

                // rtems_status_code rtems_message_queue_send(rtems_id id, const void *buffer,
size_t size);

                status =
rtems_message_queue_send(message_queue_synchronize_acquisition_processing_id, &acquisition_counter, sizeof(uint32_t));
            }
        }
    }

rtems_task processing_task(rtems_task_argument argument) {
    rtems_status_code status;
    uint32_t acquisition_counter;
    size_t size;
    while (1) {
        // rtems_status_code rtems_message_queue_receive(rtems_id id, void *buffer, size_t *size, rtems_option
option_set, rtems_interval timeout);
        status = rtems_message_queue_receive(message_queue_synchronize_acquisition_processing_id,
&acquisition_counter, &size, RTEMS_WAIT, RTEMS_NO_TIMEOUT);

        if(status == RTEMS_SUCCESSFUL) {
            uint32_t i;
            for (i = 0; i < NB_WINDOWS_TO_PRODUCE_FOR_EACH_CALL_TO_PRODUCE_IMAGES ; i++) {
                if(flux_pondere[i].measures[FLUX_LENGTH -1] != 0) {
                    flux_pondere[i].id_first_acquisition = acquisition_counter;

                    for(uint32_t j = 0 ; j < FLUX_LENGTH ; j++) {
                        flux_pondere[i].measures[j] = 0;
                    }

                    float* mask = get_mask(&wp, i);
                    float* window = img_buffer[ flux_pondere[i].id_window ];

                    flux_pondere[i].measures[(flux_pondere[i].id_first_acquisition +
acquisition_counter) % FLUX_LENGTH] = computeFluxPondere(window, mask);

                    if(flux_pondere[i].measures[FLUX_LENGTH -1] != 0) {
                        push_partagee(&circular_buffer, (uint8_t*) &flux_pondere[i],
sizeof(flux));
                    }
                }
            }
        }
    }
}

rtems_task telemetry_manager_task(rtems_task_argument argument) {
    const uint32_t max_num_streams_on_time_slot = 30;
    const uint32_t regulation_interval = 100; // ms
    flux buffer;
    int32_t result;

    uint32_t slot_start_time = rtems_clock_get_ticks_since_boot();
    uint32_t count_streams_since_start_of_slot = 0;

    while (1) {
        result = pop_partagee(&circular_buffer, (uint8_t*) &buffer, sizeof(flux));
        if(result == sizeof(flux)) {
            uint32_t current_time = rtems_clock_get_ticks_since_boot();

            if((current_time - slot_start_time) >= regulation_interval){
                slot_start_time = rtems_clock_get_ticks_since_boot();
                count_streams_since_start_of_slot = 0;
            }

            send_flux(current_time, buffer);
            count_streams_since_start_of_slot++;

            //if(count_streams_since_start_of_slot == max_num_streams_on_time_slot) {
            //    rtems_task_wake_after( regulation_interval - (current_time -slot_start_time) );

```

```

    //}

    }

}

/* ***** INIT ***** */

rtems_task Init(rtems_task_argument argument) {
    rtems_status_code status;

    for (uint32_t i = 0; i < NB_WINDOWS_TO_PRODUCE_FOR_EACH_CALL_TO_PRODUCE_IMAGES ; i++) {
        flux_pondere_fifo[i].id_window = i;
        flux_pondere[i].id_window = i;

        for (uint32_t j = 0; j < FLUX_LENGTH ; j++) {
            flux_pondere_fifo[i].measures[j] = 0;
            flux_pondere[i].measures[j] = 0;
        }
    }

    init_buffer_circulaire_partagee(&circular_buffer, (uint8_t*) flux_pondere_fifo, sizeof(flux),
    NB_WINDOWS_TO_PRODUCE_FOR_EACH_CALL_TO_PRODUCE_IMAGES);

    acquisition_task_name = rtems_build_name('A','C','U','I');
    processing_task_name = rtems_build_name('P','R','O','C');
    telemetry_manager_task_name = rtems_build_name('T','E','L','E');
    timer_name = rtems_build_name('T','I','M','E');
    message_queue_synchronize_timer_acquisition_name = rtems_build_name('T','M','A','C');
    message_queue_synchronize_acquisition_processing_name = rtems_build_name('A','C','T','M');

    // rtems_status_code rtems_timer_create( rtems_name name, rtems_id *id );
    status = rtems_timer_create(timer_name, &timer_id);
    assert(status == RTEMS_SUCCESSFUL);

    // rtems_status_code rtems_timer_fire_after(rtems_id id, rtems_interval ticks, rtems_timer_service_routine_entry
    routine, void *user_data);
    status = rtems_timer_fire_after(timer_id, rtems_clock_get_ticks_per_second() / 2, reload_timer ,NULL);
    assert(status == RTEMS_SUCCESSFUL);

    // rtems_status_code rtems_message_queue_create(rtems_name name, uint32_t count, size_t max_message_size,
    rtems_attribute attribute_set, rtems_id *id);
    status = rtems_message_queue_create(message_queue_synchronize_timer_acquisition_name, MESSAGE_QUEUE_COUNT,
    MESSAGE_QUEUE_SIZE, RTEMS_LOCAL | RTEMS_PRIORITY, &message_queue_synchronize_timer_acquisition_id);
    assert(status == RTEMS_SUCCESSFUL);
    status = rtems_message_queue_create(message_queue_synchronize_acquisition_processing_name, MESSAGE_QUEUE_COUNT,
    MESSAGE_QUEUE_SIZE, RTEMS_LOCAL | RTEMS_PRIORITY, &message_queue_synchronize_acquisition_processing_id);
    assert(status == RTEMS_SUCCESSFUL);

    // rtems_status_code rtems_task_create(rtems_name name, rtems_task_priority initial_priority, size_t stack_size,
    rtems_mode initial_modes, rtems_attribute attribute_set, rtems_id *id);
    status = rtems_task_create(acquisition_task_name, ACQUISITION_TASK_PRIORITY, TASK_STACK_SIZE, RTEMS_PREEMPT |
    RTEMS_NO_TIMESLICE | RTEMS_ASR | RTEMS_INTERRUPT_LEVEL(0), RTEMS_LOCAL | RTEMS_FLOATING_POINT, &acquisition_task_id);
    assert(status == RTEMS_SUCCESSFUL);
    status = rtems_task_create(processing_task_name, PROCESSING_TASK_PRIORITY, TASK_STACK_SIZE * 2, RTEMS_PREEMPT |
    RTEMS_NO_TIMESLICE | RTEMS_ASR | RTEMS_INTERRUPT_LEVEL(0), RTEMS_LOCAL | RTEMS_FLOATING_POINT, &processing_task_id);
    assert(status == RTEMS_SUCCESSFUL);
    status = rtems_task_create(telemetry_manager_task_name, TELEMETRY_MANAGER_TASK_PRIORITY, TASK_STACK_SIZE, RTEMS_PREEMPT
    | RTEMS_NO_TIMESLICE | RTEMS_ASR | RTEMS_INTERRUPT_LEVEL(0), RTEMS_LOCAL | RTEMS_FLOATING_POINT,
    &telemetry_manager_task_id);
    assert(status == RTEMS_SUCCESSFUL);

    // rtems_status_code rtems_task_start(rtems_id id, rtems_task_entry entry_point, rtems_task_argument argument)
    status = rtems_task_start(acquisition_task_id, acquisition_task, 0);
    assert(status == RTEMS_SUCCESSFUL);
    status = rtems_task_start(processing_task_id, processing_task, 0);
    assert(status == RTEMS_SUCCESSFUL);
    status = rtems_task_start(telemetry_manager_task_id, telemetry_manager_task, 0);
    assert(status == RTEMS_SUCCESSFUL);

    status = rtems_task_delete( RTEMS_SELF );
    assert(status == RTEMS_SUCCESSFUL);
}

```

La mis à jour principale concerne bien évidemment l'ajout d'une nouvelle tâche :

```
rtems_task telemetry_manager_task(rtems_task_argument argument) {
    const uint32_t max_num_streams_on_time_slot = 30;
    const uint32_t regulation_interval = 100; // ms
    flux buffer;
    int32_t result;

    uint32_t slot_start_time = rtems_clock_get_ticks_since_boot();
    uint32_t count_streams_since_start_of_slot = 0;

    while (1) {
        result = pop_partagee(&circular_buffer, (uint8_t*) &buffer, sizeof(flux));
        if(result == sizeof(flux)) {
            uint32_t current_time = rtems_clock_get_ticks_since_boot();

            if((current_time - slot_start_time) >= regulation_interval){
                slot_start_time = rtems_clock_get_ticks_since_boot();
                count_streams_since_start_of_slot = 0;
            }

            send_flux(current_time, buffer);
            count_streams_since_start_of_slot++;

            //if(count_streams_since_start_of_slot == max_num_streams_on_time_slot) {
            //rtems_task_wake_after( regulation_interval - (current_time -slot_start_time) );
            //}

        }
    }
}
```

On remarquera qu'en premier temps la régularisation du débit est désactivé :

```
//if(count_streams_since_start_of_slot == max_num_streams_on_time_slot) {
//rtems_task_wake_after( regulation_interval - (current_time -slot_start_time) );
//}
```

Afin d'activer la régulation du débit il suffit de décommenter ces lignes.

On a fait également le nécessaire afin de gérer la transmission de données depuis [processing_task](#) vers cette nouvelle tâche :

```
rtems_task processing_task(rtems_task_argument argument) {
    rtems_status_code status;
    uint32_t acquisition_counter;
    size_t size;
    while (1) {
        // rtems_status_code rtems_message_queue_receive(rtems_id id, void *buffer, size_t *size, rtems_option
        option_set, rtems_interval timeout);
        status = rtems_message_queue_receive(message_queue_synchronize_acquisition_processing_id,
        &acquisition_counter, &size, RTEMS_WAIT, RTEMS_NO_TIMEOUT);

        if(status == RTEMS_SUCCESSFUL) {
            uint32_t i;
            for (i = 0; i < NB_WINDOWS_TO_PRODUCE_FOR_EACH_CALL_TO_PRODUCE_IMAGES ; i++) {
                if(flux_pondere[i].measures[FLUX_LENGTH -1] != 0) {
                    flux_pondere[i].id_first_acquisition = acquisition_counter;

                    for(uint32_t j = 0 ; j < FLUX_LENGTH ; j++) {
                        flux_pondere[i].measures[j] = 0;
                    }
                }

                float* mask = get_mask(&wp, i);
                float* window = img_buffer[ flux_pondere[i].id_window ];

                flux_pondere[i].measures[(flux_pondere[i].id_first_acquisition +
                acquisition_counter) % FLUX_LENGTH] = computeFluxPondere(window, mask);
            }
        }
    }
}
```



```

        if(flux_pondere[i].measures[FLUX_LENGTH -1] != 0) {
            push_partagee(&circular_buffer, (uint8_t*) &flux_pondere[i],
sizeof(flux));
        }
    }
}
}

```

Enfin, voici la nouvelle version de notre script gdb :

`gdb_batch.txt`

```

# Sources des commentaires :
# le cours et le manuel de l'utilisateur du simulateur TSIM3
# https://www.gaisler.com/doc/tsim3.pdf

# tar = target
# Se connecter à TSIM à l'aide de extended-remote protocol
tar extended-remote localhost:1234

# Charger l'image du programme dans le simulateur
load

mon perf reset

# Declaration d'un point d'arrêt via la commande hbreak
hbreak break_point
commands
    silent
    printf "Elapsed time : %.2f sec \n", elapsed_time
    cont
end

# Declaration d'un point d'arrêt via la commande hbreak
hbreak send_flux
commands
    silent
    printf "*****\n"
    printf "%d \t %d \t %d \t %d", time, f.id_first_acquisition, f.id_window, cpt
    printf "\t %f ", f.measures[0]
    printf "\t %f ", f.measures[1]
    printf "\t %f ", f.measures[2]
    printf "\t %f ", f.measures[3]
    printf "\t %f ", f.measures[4]
    printf "\t %f ", f.measures[5]
    printf "\t %f ", f.measures[6]
    printf "\t %f ", f.measures[7]
    printf "\t %f ", f.measures[8]
    printf "\t %f\n", f.measures[9]
    printf "*****\n"
    cont
end

start

```

On remarque qu'au cours des 12 premiers secondes les dernière valeurs des 60 premiers fenêtres dans le tableau **flux_pondere** sont les même, avec ou sans régularisation du débit.

1016	10	0	100	14010.194336	13140.085938	19217.835938	18199.400391	20171.099609	10422.158203	8579.107422	11070.264648	8645.449219	9780.735352
1016	10	1	101	6327.452148	5699.705566	3951.451904	2845.152100	3151.291992	4694.883789	2955.593750	2164.656738	2209.821533	3242.388672
1016	10	2	102	5425.893066	4117.424316	5698.046387	5508.911621	3333.216064	2954.187500	2595.167969	1514.593384	2019.022583	1791.194580
1016	10	3	103	27223.384766	18754.884766	21342.337891	20892.056641	14161.737305	10500.391602	15387.536133	22597.458984	22039.031250	18656.492188
1016	10	4	104	930.128479	647.439575	686.932983	1025.155518	1069.522095	723.881104	707.941162	425.103119	596.248535	663.897766
1016	10	5	105	5415.618652	6995.371094	4595.128418	3585.683350	4765.355469	3599.741699	5300.234375	4446.469727	5727.204590	3483.814941
1016	10	6	106	12971.703125	11606.693359	16635.433594	8760.964844	7871.355469	4673.913574	4413.978516	5973.242188	6030.900879	7796.528320
1016	10	7	107	8624.421875	12051.271484	13341.833984	19574.562500	10854.169922	11872.243164	15732.502930	10662.940430	15183.001953	19077.919922
1016	10	8	108	25136.496094	33935.367188	25359.818359	20438.542969	29271.509766	17338.435547	19925.785156	28686.798828	17775.724609	23121.351562
1016	10	9	109	3858.457275	3084.321777	3336.410156	4798.958008	4717.592285	3997.915771	4488.369629	3136.935303	1575.293213	1591.490112
1016	10	10	110	12468.756836	8237.813477	10501.685547	8587.916016	5763.785156	4265.650391	4160.652832	6035.156738	6645.558105	3940.784180
1016	10	11	111	19772.332031	23903.345703	34981.917969	50102.531250	66624.562500	60511.828125	56256.718750	59457.160156	74807.789062	65347.031250
1016	10	12	112	93585.914062	96733.390625	79127.250000	108333.140625	139691.156250	174881.703125	244535.734375	314516.343750	357808.875000	492939.531250
1016	10	13	113	14792.305664	9102.631836	8794.429688	4752.769531	4885.770508	6701.721191	8437.761719	12161.690430	17046.744141	12313.872070
1016	10	14	114	6457.453125	4987.894531	5054.975098	5916.521973	7464.798828	10929.391602	8463.161133	9747.836914	7623.022949	4529.689941
1016	10	15	115	40674.582031	23857.671875	28192.359375	41656.894531	23972.349609	22799.496094	21690.033203	28150.730469	29514.505859	31042.560547
1016	10	16	116	1817.580444	1753.794800	2457.799072	3225.640869	3901.336670	5227.620605	5847.330566	5627.899902	3727.331543	1941.668335
1016	10	17	117	4257.634277	4415.354980	2462.833740	2246.840576	2005.039062	2570.741943	3610.916504	3367.303711	2104.572998	2301.760742
1016	10	18	118	4424.699707	3375.829590	4708.923340	3616.492188	2307.222168	1553.297485	852.851562	544.939941	390.244049	461.178345
1016	10	19	119	26032.287109	20422.880859	22070.111328	19071.767578	21675.083984	22001.462891	25985.642578	26651.685547	28835.261719	16287.575195
1016	10	20	120	6200.861816	9017.312500	8934.526367	4823.440430	2794.449951	2033.559326	1520.147217	2156.336426	1439.026611	1548.966064
1016	10	21	121	2352.976318	1930.095459	1577.184692	1628.790161	2272.634033	2548.256836	2425.644043	3285.805176	3336.484131	3734.722900
1016	10	22	122	12111.988281	15230.704102	19397.949219	17406.591797	24514.228516	32252.150391	27561.193359	31115.708984	21573.667969	28403.089844
1016	10	23	123	14087.414062	15421.652344	21028.724609	25448.005859	16860.837891	17973.472656	17273.132812	24507.431641	21907.039062	19747.976562
1016	10	24	124	15293.415039	14204.696289	12439.875977	13659.190430	13710.102539	10295.645508	13584.036133	8865.659180	10451.493164	12631.913086
1016	10	25	125	10965.122070	11605.643555	5920.169922	4982.554199	6856.093750	5685.731934	7876.372070	10894.500000	11932.351562	12340.866211
1016	10	26	126	4669.723145	3901.314453	4524.636719	5663.619629	8124.117188	6687.385742	3457.120117	2739.524414	2909.166016	1888.883301
1016	10	27	127	2792.522217	2527.641602	1460.243042	1723.507568	1259.937866	1345.961060	715.642029	826.124329	465.671387	468.570648
1016	10	28	128	3819.453369	3391.556641	4559.377930	5636.727051	3515.602783	3891.518311	3431.796387	2183.199463	1814.619873	920.617065
1016	10	29	129	7490.955566	7482.547363	4092.711670	3364.479492	2129.719238	1972.747681	1112.737793	859.748596	1226.514038	1189.602905
1116	10	30	130	10141.414062	11797.550781	13491.190430	19210.373047	22721.724609	23615.177734	12250.359375	6781.063965	6041.742676	6179.770996
1116	10	31	131	18193.212891	19913.119141	16911.437500	24503.220703	16226.463867	14624.058594	11503.748047	12242.949219	14822.608398	16190.506836

1116	10	32	132	21836.044922	12521.349609	7849.408203	4847.721680	2921.579346	2585.896240	3077.236328	2814.758789	1932.278931	1845.474976
1116	10	33	133	5542.813477	5543.548828	8310.587891	11350.579102	6585.328125	5940.785156	7778.985352	8750.534180	10570.170898	8439.466797
1116	10	34	134	3211.929443	1918.589600	2720.071289	3288.929199	2080.686279	1952.961792	2569.986816	2521.762207	2843.770996	2016.279053
1116	10	35	135	562.183105	757.969421	931.455994	751.387146	410.220734	457.864288	292.685974	286.880920	416.387299	330.866425
1116	10	36	136	35791.273438	47053.218750	27520.150391	15874.226562	11234.230469	11598.425781	12614.988281	6561.940918	9098.668945	11729.193359
1116	10	37	137	5724.338379	4300.403320	2499.938965	2125.714844	3136.216309	2968.171631	2977.966797	2094.898193	2239.442627	1258.328491
1116	10	38	138	19824.322266	12503.887695	12557.047852	18129.244141	10954.751953	11091.541016	9548.431641	13215.931641	12013.204102	13205.887695
1116	10	39	139	1983.087646	1456.507446	817.941284	1105.952759	1255.985962	981.512756	1234.446899	810.644592	582.853821	300.391388
1116	10	40	140	8897.452148	13177.684570	12786.736328	17543.101562	21961.855469	31722.066406	33947.402344	17633.917969	17221.763672	9332.480469
1116	10	41	141	155035.687500	103883.203125	60554.445312	45216.660156	53915.527344	61444.527344	59714.843750	87041.570312	52022.281250	73030.976562
1116	10	42	142	2382.449219	2207.635986	2112.314453	1071.023193	1195.325195	1317.812012	815.743896	449.333252	319.420593	338.996246
1116	10	43	143	4420.139648	4811.040527	3807.908936	3746.110840	5425.951660	3041.674316	1570.496704	794.109985	588.157654	359.476257
1116	10	44	144	4859.784668	6464.683105	4563.750977	5805.343262	6003.087891	7994.779785	4373.860840	2196.898438	1311.395996	1401.911011
1116	10	45	145	3477.567383	4743.611816	6076.479492	3901.627441	3927.417480	4603.795410	3615.837158	1946.643433	2562.173096	2376.467285
1116	10	46	146	51051.578125	49909.785156	39965.976562	39571.632812	21174.335938	14238.660156	7669.128418	10672.627930	8041.089355	9824.309570
1116	10	47	147	2330.363037	2546.554932	3400.903320	3950.896484	4814.695312	2886.202881	2624.522949	2019.439575	1662.643799	1499.999512
1116	10	48	148	25710.029297	26828.523438	35215.695312	52646.968750	62796.820312	74079.531250	37096.554688	45915.289062	66798.523438	82900.382812
1116	10	49	149	2692.102783	1625.452515	1133.871460	1104.967651	1579.892090	2183.952148	1647.309814	2380.091309	3370.708496	3200.457275
1116	10	50	150	5870.921387	4489.902832	4024.615234	3727.054443	5286.745605	4303.834961	5345.715332	3123.968994	2462.373779	1745.125977
1116	10	51	151	122918.984375	137860.343750	79368.296875	47822.003906	51850.652344	50457.761719	69881.414062	94502.804688	140016.203125	82686.351562
1116	10	52	152	23748.960938	34541.937500	50792.515625	55743.949219	53228.292969	67893.257812	85586.335938	125347.671875	76034.421875	61884.031250
1116	10	53	153	6283.539062	5250.717773	3640.373047	1868.904053	2698.800293	3444.244141	3886.609375	5712.124023	4756.925781	2992.118652
1116	10	54	154	5121.597168	3749.601562	5402.963867	8016.025879	9182.329102	8773.294922	11627.774414	12745.082031	14778.630859	9491.689453
1116	10	55	155	19822.261719	10163.730469	7110.718262	7579.782227	4779.879395	5410.250977	4657.883301	4711.747070	3739.661377	2617.900879
1116	10	56	156	29089.871094	24324.464844	35631.105469	50453.898438	65683.515625	54461.136719	71564.421875	94198.687500	79698.773438	97768.265625
1116	10	57	157	9410.987305	11889.776367	13313.144531	8413.688477	12229.994141	7937.528320	5188.042969	7478.637207	10713.973633	7926.563477
1116	10	58	158	7794.057129	11562.162109	11456.376953	11133.251953	12322.366211	8136.107422	10652.720703	9460.883789	10769.346680	5807.732910
1116	10	59	159	2590.295898	1381.944092	784.974487	643.518433	848.431824	672.686951	647.250122	799.006714	693.027771	701.292236

Cependant, dans le cas sans régularisation du débit, durant les 12 premiers secondes, on arrive a transférer jusqu’au flux numéro 199, ce qui signifie qu’on arrive a compléter l’envoi des flux de tous les 100 fenêtres avec un identifiant de première acquisition égale à 10 (ce n’est pas le cas pour l’exécution avec régularisation du débit).

1017	10	60	160	5966.723145	3874.443115	5292.367676	5066.957031	6707.538086	5235.456543	6843.159180	9980.289062	12675.858398	16133.893555
1017	10	61	161	16873.830078	13965.627930	19937.425781	19917.818359	18608.087891	12198.267578	6179.979980	6291.595215	3196.425293	3185.898438

1017	10	62	162	9480.875000	5990.099609	6452.273438	7309.641602	10498.492188	8570.117188	5173.593262	5819.247070	5766.483398	4022.799561
1017	10	63	163	11885.180664	10177.935547	10488.515625	10398.299805	13015.923828	11271.243164	7199.251953	6049.802246	4269.960449	3547.134766
1017	10	64	164	2013.204590	1991.990234	1058.887939	1424.020264	1682.037231	935.831604	659.671570	380.833710	553.231384	672.718872
1017	10	65	165	3288.093750	1744.321777	1657.524170	2085.427734	1689.941162	1165.155640	1151.609619	720.603577	845.345154	504.258331
1017	10	66	166	13751.938477	19798.083984	9945.013672	12820.258789	18217.068359	9679.677734	13216.279297	19648.472656	18543.539062	10589.670898
1017	10	67	167	23701.701172	30401.976562	29344.128906	30644.851562	27685.316406	34585.042969	49182.816406	72983.585938	82560.195312	79426.164062
1017	10	68	168	31067.179688	46410.566406	64305.890625	33096.433594	24289.267578	34346.300781	20466.222656	26534.330078	21555.402344	26578.353516
1017	10	69	169	20357.509766	27036.933594	38953.523438	23336.378906	33266.074219	35089.871094	27134.375000	15324.943359	22160.570312	16064.181641
1017	10	70	170	9490.343750	11749.508789	6845.573242	6833.832520	10146.448242	8506.372070	12581.433594	12291.184570	14514.621094	14115.058594
1017	10	71	171	8175.227539	10464.473633	13555.937500	17090.056641	9541.980469	5984.674805	5830.698242	7808.231934	9423.082031	4780.541504
1017	10	72	172	944.659790	1382.574463	1805.562866	2078.395264	2878.928711	4038.994141	5150.505859	6040.253906	3854.523926	4163.725586
1017	10	73	173	115512.718750	145927.531250	106178.218750	56086.445312	35504.308594	31785.507812	46595.230469	35428.964844	19244.943359	23403.855469
1017	10	74	174	23156.748047	19828.939453	22713.947266	27635.359375	18750.677734	10879.809570	16074.969727	13284.088867	8586.431641	11644.949219
1017	10	75	175	3374.927246	3562.353760	3022.887695	2315.032715	2093.671143	1364.185303	805.000488	684.519775	1012.463318	662.883057
1017	10	76	176	16322.761719	13208.528320	14092.273438	15202.657227	15088.917969	8279.254883	9186.214844	9004.958008	10173.128906	14429.807617
1017	10	77	177	7753.887695	8354.537109	4930.170898	3333.244629	2859.655762	3001.074951	3305.689697	4854.418945	7221.914551	5358.709961
1017	10	78	178	1468.283203	1039.661377	790.021729	463.917419	400.825623	586.037292	668.408142	899.474731	584.124573	608.206238
1017	10	79	179	4616.098145	5563.384766	7102.258789	10151.008789	11861.327148	7483.692871	8097.753418	7255.977051	5987.889160	6207.848145
1017	10	80	180	584.048767	798.911194	650.030151	344.282166	280.998535	198.789200	234.031693	246.951172	130.331131	139.195724
1017	10	81	181	9154.820312	7982.115723	5777.963379	4009.349121	3375.186035	2550.921875	2973.020020	2527.398438	3007.572021	1657.958496
1017	10	82	182	4844.881348	6844.497559	4327.406738	2462.541016	3096.340332	3728.586182	2632.123779	2402.237305	3130.402832	2252.453369
1017	10	83	183	14168.999023	7151.998047	5279.925781	2812.816650	3952.730469	4267.130371	4910.342773	3672.033203	3803.831055	4290.891602
1017	10	84	184	101548.335938	79677.617188	72315.351562	53684.792969	61159.066406	74820.429688	74318.359375	75886.281250	83395.234375	67986.546875
1017	10	85	185	7488.387695	9513.275391	10032.007812	13757.390625	15249.780273	11440.217773	14336.875977	8503.722656	12280.413086	17617.539062
1017	10	86	186	39303.195312	54792.324219	43712.324219	43754.347656	32070.718750	27115.064453	16977.808594	17159.126953	20843.552734	27624.580078
1017	10	87	187	3662.203369	4190.589844	5343.006836	3189.316162	3816.250732	3598.385986	1902.949219	2418.450439	1325.454346	1637.669189
1017	10	88	188	16430.472656	17916.080078	16628.515625	11424.242188	7705.685547	6084.008789	5982.898438	8413.449219	12442.511719	12462.222656
1017	10	89	189	39784.332031	23150.621094	26351.792969	37726.597656	40852.359375	28792.031250	42855.496094	24836.427734	13036.441406	9898.901367
1017	10	90	190	93407.632812	134082.000000	112446.500000	114814.304688	117451.164062	153284.218750	139132.046875	169499.640625	168846.703125	223183.718750
1017	10	91	191	13868.397461	8887.251953	6716.630371	8100.620605	8259.460938	6894.002441	10229.753906	9026.556641	7223.172363	8502.367188
1017	10	92	192	4414.210449	3903.564453	3185.569336	4052.406250	5092.882324	3731.403809	2042.056396	1043.836548	869.830139	1109.939331
1017	10	93	193	10520.839844	6479.578125	5416.493652	3867.065674	3655.577881	3065.196533	1669.293945	2251.087891	2116.099609	1123.438721
1017	10	94	194	8205.338867	5852.855957	4254.330078	4818.308105	4325.693848	4160.055664	5555.958496	2840.337891	2569.627197	1472.369995
1017	10	95	195	5790.767090	4267.687988	4241.723633	5582.424316	5022.759766	3127.439209	2513.751709	1317.744751	1462.116089	902.157532
1017	10	96	196	102395.421875	151764.062500	93248.562500	80535.273438	103518.398438	130221.031250	194197.968750	256187.921875	225337.171875	193601.921875
1017	10	97	197	3682.362549	2453.288818	2695.629395	3151.411621	4197.458008	4223.576172	5554.449707	2930.722656	2136.995117	1871.431274

```

1017 10 98 198 33034.136719 29906.941406 24410.818359 22295.669922 30779.511719 25667.863281 14643.420898 19276.962891 16317.039062 17275.572266
1017 10 99 199 16084.166016 14899.714844 9631.902344 12407.606445 16517.138672 23250.042969 24251.630859 21752.945312 26236.337891 25076.076172

```

Pour terminer, on vérifie le comportement du système si la tâche d'émission de TM est de plus haute priorité que la tâche de traitements :

```

#define ACQUISITION_TASK_PRIORITY 1
#define PROCESSING_TASK_PRIORITY 3
#define TELEMETRY_MANAGER_TASK_PRIORITY 2

```

Dans une tel situation on obtient le résultat suivant en exécutant le script gbd :

```

Hardware assisted breakpoint 1 at 0x400012e8: file ../src/EmbarqueeTP4_4d_final.c, line 63.
Hardware assisted breakpoint 2 at 0x40001480: file ../src/EmbarqueeTP4_4d_final.c, line 106.
Function "main" not defined.
Make breakpoint pending on future shared library load? (y or [n]) [answered N; input not from terminal]
[New Thread 2]
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec
Elapsed time : 0.50 sec

```

Explication : comme la tâche d'émission de TM est de plus haute priorité que la tâche de traitements, la tâche d'émission de TM n'arrive pas à obtenir des flux depuis la tâche de traitements (vu que celle-ci n'a pas le temps d'effectuer le traitement), donc elle a rien à transmettre.

On peut le vérifier avec notre script gbd, il suffit de changer légèrement le point d'arrêt suivant :

```

# Declaration d'un point d'arrêt via la commande hbreak
hbreak break_point
commands
    silent
    printf "Elapsed time : %.2f sec \n", elapsed_time
    print flux_pondere
cont
end

```

On exécute le script jusqu'à qu'il s'arrête au bout d'un moment (vu qu'on utilise la version d'évaluation du logiciel [tsim](#)) :

```
ow = 89, id_first_acquisition = 0, measures = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, {id_window = 90, id_first_acquisition = 0, measures = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}}, {id_window = 91, id_first_acquisition = 0, measures = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}}, {id_window = 92, id_first_acquisition = 0, measures = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}}, {id_window = 93, id_first_acquisition = 0, measures = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}}, {id_window = 94, id_first_acquisition = 0, measures = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}}, {id_window = 95, id_first_acquisition = 0, measures = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}}, {id_window = 96, id_first_acquisition = 0, measures = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}}, {id_window = 97, id_first_acquisition = 0, measures = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}}, {id_window = 98, id_first_acquisition = 0, measures = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}}, {id_window = 99, id_first_acquisition = 0, measures = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}}}}
Simulation time overflow (2^32) - halting
```

Ainsi, on remarque aisément qu'il n'y a aucun résultat de traitement enregistré dans notre tableau globale.