|  |
| --- |
| User guide |
| Time synchronization Project |
| The aim of this project is to develop and implement a precision time synchronization protocol for industrial sensor networks. Then an IEEE 1588-like time synchronization algorithm will be developed and validated to have precision time synchronization among at least two wireless sensor nodes. |

MORAND Baptiste

7/15/2016

Contents

[I. Generalities 4](#_Toc457555049)

[A. Time synchronization 4](#_Toc457555050)

[B. FREERTOS 7](#_Toc457555051)

[1. RTOS 7](#_Toc457555052)

[2. FREERTOS 8](#_Toc457555053)

[3. Tools FREERTOS 10](#_Toc457555054)

[II. Hardware selection 11](#_Toc457555055)

[III. Application hardware 12](#_Toc457555056)

[A. Master clock 14](#_Toc457555057)

[B. Slave Clock 14](#_Toc457555058)

[IV. Application construction 15](#_Toc457555059)

[A. General 15](#_Toc457555060)

[B. Layers 16](#_Toc457555061)

[V. Peripheral use 17](#_Toc457555062)

[1. GPIO LED control 17](#_Toc457555063)

[2. External interrupt 17](#_Toc457555064)

[3. Real time counter (RTC) 19](#_Toc457555065)

[4. Serial communication 20](#_Toc457555066)

[5. RF controller AT86RF233 20](#_Toc457555067)

[VI. Application 22](#_Toc457555068)

[A. Main Function 22](#_Toc457555069)

[1. Description 22](#_Toc457555070)

[2. Function 22](#_Toc457555071)

[B. Task create 23](#_Toc457555072)

[1. Description 23](#_Toc457555073)

[2. Function 23](#_Toc457555074)

[C. TimeProtocolTask 23](#_Toc457555075)

[1. Task 23](#_Toc457555076)

[2. Sender 24](#_Toc457555077)

[3. Receiver 26](#_Toc457555078)

[4. Correction 27](#_Toc457555079)

[5. Correct Offset 28](#_Toc457555080)

[D. HMI Task 28](#_Toc457555081)

[1. Description 29](#_Toc457555082)

[2. Function 29](#_Toc457555083)

[E. Interrupt 29](#_Toc457555084)

[1. PPS 29](#_Toc457555085)

[2. Internal Timer 30](#_Toc457555086)

[VII. Communication protocol 31](#_Toc457555087)

[A. Type 31](#_Toc457555088)

[1. SYNC type=1 31](#_Toc457555089)

[2. Delay Request type=2 31](#_Toc457555090)

[3. Delay Response type=3 31](#_Toc457555091)

[4. Report type =4 31](#_Toc457555092)

[B. CRC 31](#_Toc457555093)

[IX. Test bench 32](#_Toc457555094)

[A. Accuracy 32](#_Toc457555095)

[B. Synchronization time 35](#_Toc457555096)

[C. Threshold period 35](#_Toc457555097)

[D. Test interrupt delay 36](#_Toc457555098)

[X. Use of application 37](#_Toc457555099)

[XI. Improvement 38](#_Toc457555100)

[A. Peripheral management 38](#_Toc457555101)

[B. Application 39](#_Toc457555102)

[XII. Contact 40](#_Toc457555103)

Figure

[Figure 1 application schematic 4](file:///C:\Users\Time%20Syn%20Project\Documents\baptiste_MORAND\drive\NewCastle\internship\User%20guide.docx#_Toc457555024)

[Figure 2 Drifting clock 5](#_Toc457555025)

[Figure 3 PTP algorithm 5](#_Toc457555026)

[Figure 4 principle of delay calculation 6](#_Toc457555027)

[Figure 5 logo FREERTOS 7](file:///C:\Users\Time%20Syn%20Project\Documents\baptiste_MORAND\drive\NewCastle\internship\User%20guide.docx#_Toc457555028)

[Figure 6 tree structure of FREERTOS source code 9](file:///C:\Users\Time%20Syn%20Project\Documents\baptiste_MORAND\drive\NewCastle\internship\User%20guide.docx#_Toc457555029)

[Figure 7 Hardware architecture 12](file:///C:\Users\Time%20Syn%20Project\Documents\baptiste_MORAND\drive\NewCastle\internship\User%20guide.docx#_Toc457555030)

[Figure 8 master Clock hardware digram 14](#_Toc457555031)

[Figure 9 Slave clock hardware diagram 14](#_Toc457555032)

[Figure 10 software layers diagram 16](#_Toc457555033)

[Figure 11 Device clocking diagram 20](#_Toc457555034)

[Figure 12 Microcontroller to AT86RF233 Interface 21](#_Toc457555035)

[Figure 13 flow chart main 22](file:///C:\Users\Time%20Syn%20Project\Documents\baptiste_MORAND\drive\NewCastle\internship\User%20guide.docx#_Toc457555036)

[Figure 14 Flow chart main 22](file:///C:\Users\Time%20Syn%20Project\Documents\baptiste_MORAND\drive\NewCastle\internship\User%20guide.docx#_Toc457555037)

[Figure 15 flow chart kernel config 23](file:///C:\Users\Time%20Syn%20Project\Documents\baptiste_MORAND\drive\NewCastle\internship\User%20guide.docx#_Toc457555038)

[Figure 16 : flowChart timeSync task 23](file:///C:\Users\Time%20Syn%20Project\Documents\baptiste_MORAND\drive\NewCastle\internship\User%20guide.docx#_Toc457555039)

[Figure 17 flow chart sender function time protocol 24](#_Toc457555040)

[Figure 18 flow chart receiver function timeProtocol 26](#_Toc457555041)

[Figure 19 Flow chat correction function time protocol 27](#_Toc457555042)

[Figure 20 update clock function time protocol 28](#_Toc457555043)

[Figure 21 Flow chart hmi task 29](#_Toc457555044)

[Figure 22 : Flow chart pps task 29](#_Toc457555045)

[Figure 23 Flow chart internal timer 30](#_Toc457555046)

[Figure 24 test bench schematic 32](file:///C:\Users\Time%20Syn%20Project\Documents\baptiste_MORAND\drive\NewCastle\internship\User%20guide.docx#_Toc457555047)

[Figure 25 Correction offset example 39](#_Toc457555048)

# Generalities

## Time synchronization

Time synchronization is to synchronize the drifting clock of a slave node to the reference clock at the master node. The master is chosen because it has the best clock. For example, a GPS clock can work as the master node. In general the master node is synchronize with a clock that is given by a GPS. The GPS generate a pulse per second (PPS).

Figure application schematic

GPS

Master node

Slave node

Slave node

PPS

Network

As I say tine synchronization is very important, and the main problem is the oscillator of the embedded system. Because we need to reduce the cost of our system and we cannot put a GPS on all your devices. The oscillator of our system depends of the constructor, the temperature, and the alimentation (see anexe number 1 p ??). All this things will create a disturbance. To measure the differences of frequency we use the skew. The skew is the differences of frequency between the master node and the slave node. This difference of frequency will create a temporal shift.



Figure Drifting clock

Before correcting the skew we need to measure it this is why we use a precision time protocol (PTP) who is specified in the standard IEE1588. This algorithm is use to know this 2 unknown time :

* Offset: who are the differences of time between the master and the slave.
* Delay: the delay is time between sending the information and the time to receiving the information he is the sum of 3 times: the time to code the information, the propagation time and the time to decode the information.

The algorithm work like this:

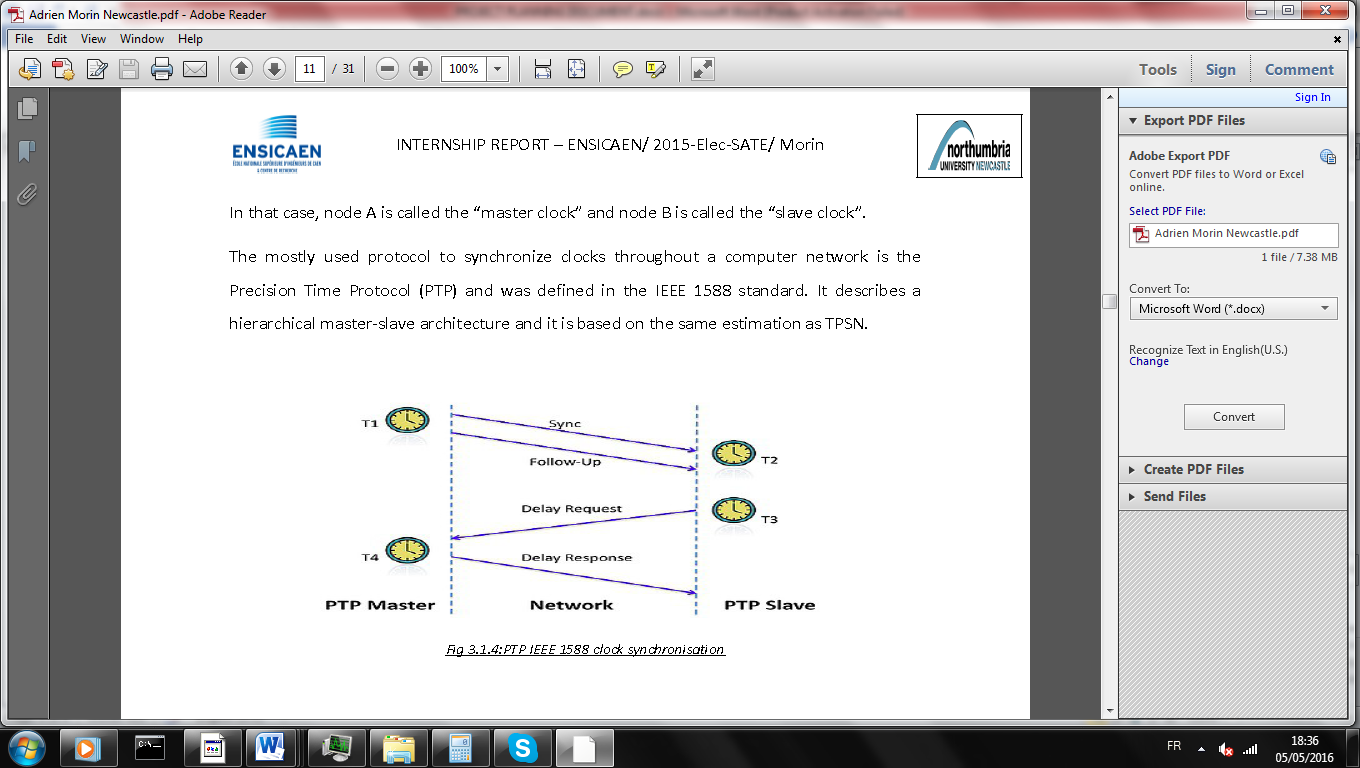


Figure PTP algorithm

Four types of packet have been defined in the PTP:

1. **Sync:** The process is started when the master send Sync packet with the timestamp who correspond to the time where master send. Sometimes generic hardware and software stacks that implement PTP don't have the ability to add a precise timestamp to a message they are in the process of generating this is why the timestamp T1 is send in a **follow up** command.
2. The Slave save the time when the packet is received (T2).
3. **Delay request:** The slave send a delay request to the master with no data. And he save the time of sending (T3)
4. **Delay Response:** The master request with the time when he have receive the request (T4).

Sync, Delay Request (DReq) and Delay Response (DRes) are three compulsory packets and the Follow Up is a optional one. When the time stamp T1 cannot be sent in packet Sync, a Follow Up packet will be used to send T1 to the slave

In practice, a slave node cannot accurately estimate the local time on the target node due to varying message or network delays between the nodes. But we can make an approximation with a little error this is why we consider that the delay is constant in the time, and the frequency of the slave and the master will not change during the period of synchronization. With the time given by the PTP we can calculate the delay and the offset.

d + Xi +

d + Xi -

T4

T3

T2

T1

Figure principle of delay calculation

d is the delay, Xi a random delay, is the offset.

T2-T1=d+Xi + T4-T3= d+Xi-

With the information we can show that:

When can conclude that time:

Tslave=Tslave+.

The aim of my project is to reduce the offset. The offset is the difference of time between the master and the slave. For calculate the offset we have to principle

Figure logo FREERTOS

## FREERTOS

### RTOS

* What is a RTOS?

Most operating systems appear to allow multiple programs to execute at the same time. This is called multi-tasking. In reality, each processor core can only be running a single thread of execution at any given point in time. A part of the operating system called the scheduler is responsible for deciding which program to run when, and provides the illusion of simultaneous execution by rapidly switching between each program.

The type of an operating system is defined by how the scheduler decides which program to run when. For example, the scheduler used in a multi user operating system (such as Unix) will ensure each user gets a fair amount of the processing time. As another example, the scheduler in a desk top operating system (such as Windows) will try and ensure the computer remains responsive to its user. [Note: FreeRTOS is not a big operating system, nor is it designed to run on a desktop computer class processor, I use these examples purely because they are systems readers will be familiar with]

The scheduler in a Real Time Operating System (RTOS) is designed to provide a predictable (normally described as deterministic) execution pattern. This is particularly of interest to embedded systems as embedded systems often have real time requirements. A real time requirements is one that specifies that the embedded system must respond to a certain event within a strictly defined time (the deadline). A guarantee to meet real time requirements can only be made if the behaviour of the operating system's scheduler can be predicted (and is therefore deterministic).

* Why use a RTOS?

We are using RTOS because it’s easy to modulate the application, for example if you want to add a functionality you just have to add a task, and because it’s offer a lot of services (Queue, task …). It offers a control of the time of the application, we can’t now exactly when a function is call but we can know a maximum time for calling a function.

* How it work?

You have to create some task. Each task has a priority, the scheduler run the task with the maximum priority. After you have a lot of option to communicate between the tasks. And obviously you can use all peripherals of the MCU without problems.

### FREERTOS

* Why use a FREERTOS?

Because it’s the most use RTOS and because it is totally free.

* What is the footprint of FREERTOS?

This depends on your application. Below is a guide based on:

* IAR STR71x ARM7 port.
* Full optimization.
* Minimum configuration
* Four priorities.

How much RAM does FreeRTOS use?

|  |  |
| --- | --- |
| Item | Bytes Used |
| Scheduler Itself | 236 bytes (can easily be reduced by using smaller data types). |
| For each queue you create, add | 76 bytes + queue storage area (see FAQ Why do queues use that much RAM?) |
| For each task you create, add | 64 bytes (includes 4 characters for the task name) + the task stack size. |

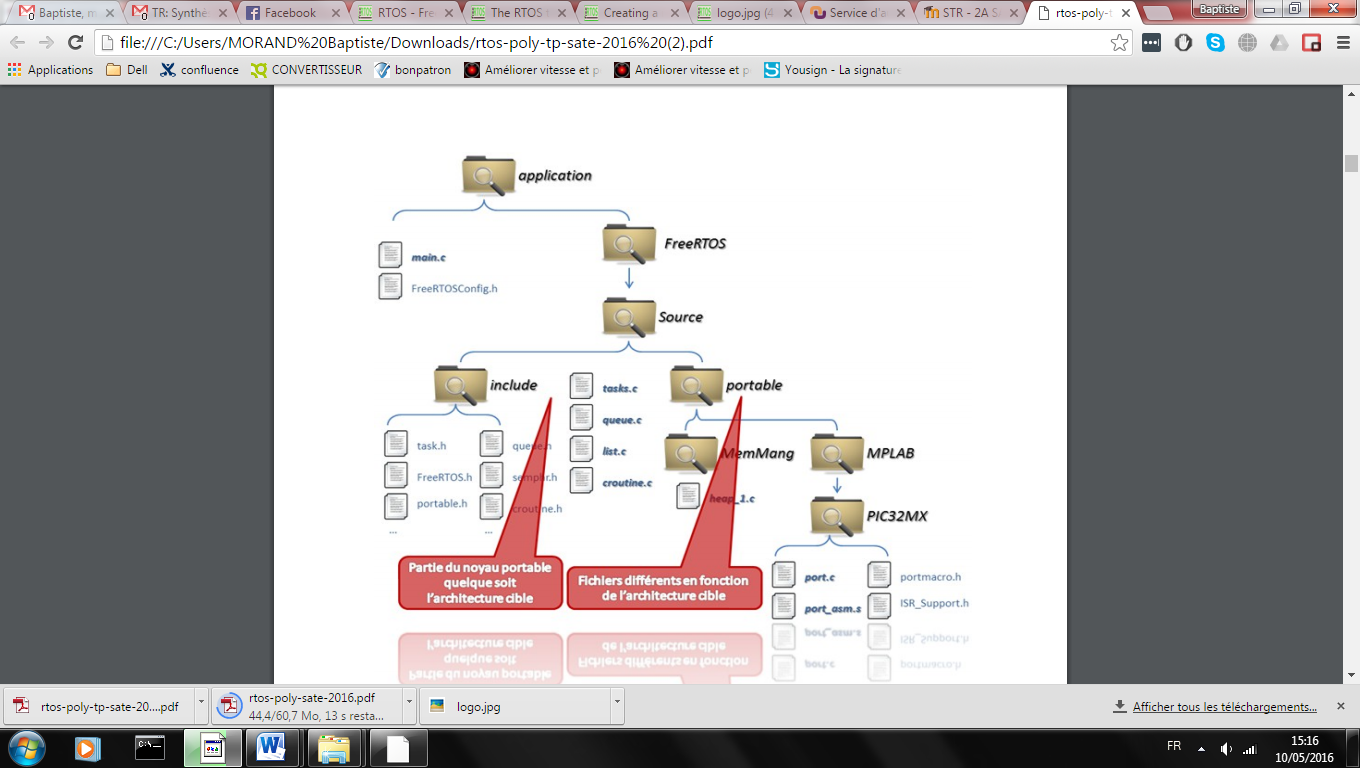
How much ROM/Flash does FreeRTOS use?

This depends on your compiler, architecture, and RTOS kernel configuration.

The RTOS kernel itself required about 5 to 10 Kbytes of ROM.

* How it will be adapt to the hardware?

FREERTOS is just a system of files (.c .h .s) it is just a software platform. The entire specified file for the microcontroller is written in assembler this is while FREERTOS is totally independent of the toolchain.



Your application

Portable not depend of the architecture

File depend of the architecture

Figure tree structure of FREERTOS source code

* With device support FREERTOS

There is a lot of device that is supported by FREERTOS. See the webpage for more information

<http://www.freertos.org/a00090.html>

### Tools FREERTOS

Read user guide to more understand how it work :

<http://www.freertos.org/FreeRTOS-quick-start-guide.html>

FreeRTOS offer a lot of services I will just make a list of tools use :

* Task management: like all the RTOS FREERTOS offer a lot of services to create task and to manage this. To understand more about the task read this :

<http://www.freertos.org/a00015.html>

* Semaphore and Mutexes : it is a system of token( for binary semaphore/mutexes) that can have 2 functionality :
  + Protection of share things
  + Synchronization of task

<http://www.freertos.org/Inter-Task-Communication.html>

* Delay: The delay of freertos is not a classic delay he will block the task for a defined time.

Make attention the wrong memory management is the first error that developer do with RTOS. If you have this problem tries to understand how the microcontroller and FREERTOS manage the memory. You have to know that all task have is own memory.

# Hardware selection

**Microcontroller:**

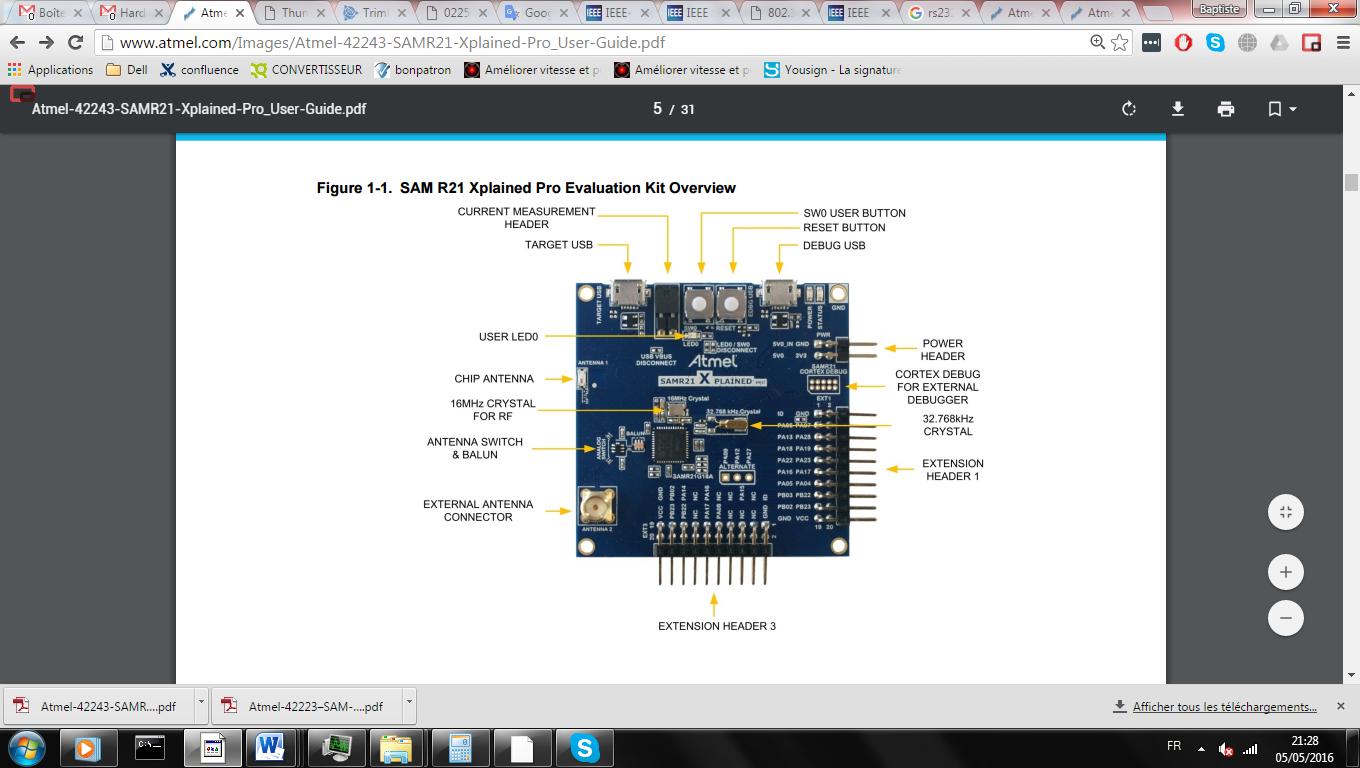
As I said before my microcontroller will be the ATSAMR21G18A this microcontroller have the particularity to have IEEE 802.15.4 controller integrated inside the chip.

ATSAMR21G18A main characteristic:

* Supply Voltage : 1.8V to 3.6V
* Memories :
  + 7256KB in-system self-programmable Flash
  + 32KB SRAM
* Peripherals
  + 28 GPIO
  + 3 timer counter
  + 1 usb interface
  + 5 communication interface (UART,I2C,SPI,…)
  + 8 ADC channel
  + Watchdog timer
  + Integrated Ultra Low Power Transceiver for 2.4GHz ISM Band

The IEEE 802.15.4 is a communication protocol defined by IEEE. It is designed for wireless networks of the family WPAN LR (Low Rate Wireless Personal Area Network) due to their low consumption, low-range and low-flow devices using this protocol.802.15.4 is used by many implementations based on proprietary or IP (Internet Protocol), such as ZigBee and 6LoWPAN.

This microcontroller will be use with industrial board SAMR21 who run at 10 MHz that will be given from the RF controller.



Component or Board

Internal peripherals

Wireless or UART

Wire

GPS

PPS

RS232

Slave Clock

LED

UART

UART

WIRELESS

Masterclock

UART

INT

UART

UART

WIRELESS

LED

Computer

Osciloscope

Figure Hardware architecture

**Global Positioning System (GPS):**

For generate a PPS we will use a ThunderBolt® E GPS Disciplined Clock this particularity is the PPS that is generate with a high accuracy.

ThunderBolt® E GPS Disciplined main characteristic:

* +24 VDC power supply.
* BNC connector :
  + Equipment to analyze the 10 MHz output frequency.
  + 1 PPS accuracy (0 to 2,4V).
* RS-232 through a DB-9/M connector.
* Antenna interface.

# Application hardware

My project is built with a master and a slave node and a GPS.

* The slave tries to be synchronizing with the slave node through the network.
* The master tries to be synchronizing with the PPS.

The oscilloscope and the computer is just use to report the value and the offset

## Master clock

WIRELESS (SPI+extINT)

See datasheet r21 p883

Masterclock

UART

INT

UART

UART

WIRELESS

LED

UART IHM **(SERCOM0)**

Rx on PA05 (via EDBG)

Tx on PA04 (via EDBG)

LED IHM

PA19( pin 8 EXT1)

UART communication **(SERCOM1)**

Tx PA16 (pin 11 EXT3)

Rx PA17 (pin 12 EXT3)

Interrupt PPS**(EXTINT[6])**

PA22 (pin 9 EXT1)

UART GPS **(SERCOM5)**

Tx PB02(pin 17 EXT1)

Rx PB03(pin 15 EXT1)

Figure master Clock hardware digram

## Slave Clock

WIRELESS (SPI+extINT)

See datasheet r21 p883

Slave Clock

LED

UART

UART

WIRELESS

UART communication **(SERCOM1)**

Tx PA16 (pin 11 EXT3)

Rx PA17 (pin 12 EXT3)

LED IHM

PA19 (pin 8 EXT1)

UART IHM **(SERCOM0)**

Rx on PA05 (via EDBG)

Tx on PA04 (via EDBG)

Figure Slave clock hardware diagram

# Application construction

## General

I have drawn some state diagram to see well how my project will be develop and specially how the time protocol will be manage.

My project has 2 tasks:

* Time protocol task:

This task will be use to communicate with the network to calculate the offset and the delay. He will be use too for correcting the frequency. This task will have the maximum of priority due to its sensitivity to delay. This task is the task the most important in my project it will consist in 3 main subtasks: Sending receiving and correction. The Sending and the receiving subtask will manage the time protocol through the network like this we can know the offset and the delay. But the main objective is to correct the frequency this is why we will use a correction subtask, the correction will use the offset to calculate the average skew, and to correct the frequency.

I will change a little the PTP protocol to use a protocol with less of packet exchange and in keeping the same accuracy. Like this we can synchronize lot of slave node without improvement of the network. And due to this it will decrease the power consumption of the device.

My protocol is constituted with 2 different processes:

* + **Delay:** This will be 2 packet one delay request, and one delay response
  + **Sync:** this command is use to correct all the device at the same time with one packet come from the master.

After define this protocol I have make some graphical of my application. (see annexes 2)

* **HMI task :** This task is just to check that the application turn well and just to inform about the accuracy with the human machine interface (HMI)

I will have 3 interrupts:

* **Timer interrupt:** This interrupt will provide the basic clock to the WSN node. By counting how many interrupts have been generated and reading the reminder in the Timer reminder, the node is aware the time. The interval between two interrupts can be configured by the timer’s threshold. For performance of comparison, this timer-interrupt clock will also be used to generate a PPS to make a comparator with the PPS that coming from the GPS clock. Like this we can measure the accuracy of the application. The period of this timer’s is the Tthreshold
* **Receive network interrupt:** This interrupt will be call to save the time when a data is received.
* **PPS interrupt:**  This interrupt will correct the master it is needed to synchronize the master with the GPS.

## Layers

My application is built with multiple layers:

Register Configuration

Atmel Studio Framework

Mbed library

Application

Assembler

FREERTOS

Figure software layers diagram

**Application:** Where my application I running and call function from other layer

**Mbed library:** This application given by Mbed is a very general layer to use peripheral without consideration of hardware this is why it not offer lot of option to control the peripheral, but it is the best for a faster development.

**Atmel Studio Framework (ASF):** This application given by Atmel can be used with the entire Atmel microcontroller. It offers a lot of option just in passing argument in some function but the management is not made by you. The difficulty of this layer is average but you need to understand partially how your peripherals work.

**Register configuration:** This layer is the lowest value just before assembler it offers the full option of the peripheral but it is most difficult to use because you have to understand very well how the peripheral use. But you have a totally control of the execution and the configuration. This layer is very important to control well some peripheral critical of your application.

**Assembler:** This layer is very difficult, you need to understand well how you peripheral work but also how your microcontroller work. But it offers a real time control of a function. You can know the delay of calculation and can be really useful.

**FREERTOS:** This is just the RTOS management.

# Peripheral use

Make attention one peripheral must be configure with the same layer because there is some conflict between the layers. For example we can’t use 2 diferents external interrupt one configure by mbed and an other with asf.

### GPIO LED control

* General

This peripheral is use to manage the output and the output pin directly it is use in my application to control a blinking led. This GPIO is not very important and I can use the mbed library to manage it.

* Configuration

DigitalOut led(LED1) : declare the pin LED1 in output and make all the configuration, the name to control it is “led”

See <https://developer.mbed.org/handbook/DigitalOut> for more detail.

* Application management

I have manage my led that change in the interrupt

### External interrupt

* General

This peripheral will create an interrupt when a change in detect in a pin. This peripheral is very important because the accuracy of the master node will depend of this I have choose to manage this layer with the layer ASF.

* Configuration

|  |
| --- |
| //pps.rise(ppsISR);  struct extint\_chan\_conf eint\_chan\_conf;  extint\_chan\_get\_config\_defaults(&eint\_chan\_conf);  // put the button as EXTINT  eint\_chan\_conf.gpio\_pin = PIN\_PA28A\_EIC\_EXTINT8;  eint\_chan\_conf.gpio\_pin\_mux = MUX\_PA28A\_EIC\_EXTINT8;  eint\_chan\_conf.detection\_criteria = EXTINT\_DETECT\_RISING;  eint\_chan\_conf.filter\_input\_signal = false;  extint\_chan\_set\_config(8, &eint\_chan\_conf);  //configuration callback  extint\_register\_callback(&ppsISR,8,EXTINT\_CALLBACK\_TYPE\_DETECT);  //activation callback  extint\_chan\_enable\_callback(8,EXTINT\_CALLBACK\_TYPE\_DETECT); |

The configuration is very easy to understand:

1. First we put the basic configuration this mean:
   * Wake the device if edge detection occurs whilst in sleep.
   * Input filtering disabled.
   * Internal pull-up enabled.
   * Detect falling edges of a signal.
2. After we need to select the multiplexer configuration, because for all the pin of the microcontroller you have to choose the multiplexer configuration. In this case we choose to link the pin PA28 to the EXTINT8.
3. Now, we apply the configuration on the channel 8 (EXTINT8).
4. We configure the callback function. In this case ppsISR().
5. Finally, we activate the callback function.

For more detail see: <http://asf.atmel.com/docs/3.30.1/samr21/html/group__asfdoc__sam0__extint__group.html>

* Application management

It is use to manage the PPS like this the master know that a PPS occurred.

### Real time counter (RTC)

* General

This peripheral will increase a counter and generate an interrupt when the counter fire. We can specify the maximum value of the counter.

It is the most important peripheral in my application it is him who will manage the time, the accuracy will depend of this timer.

* Configuration

|  |
| --- |
| //gENERIC CLOCK GENRERATOR  GCLK->GENCTRL.reg=0x00010402;  //Generic Clock Generator Division  GCLK->GENDIV.reg=0x00000102;  //Generic Clock Control  GCLK->CLKCTRL.reg=0x4204;  //peripheral RTC configuration  RTC->MODE0.CTRL.bit.PRESCALER=0x0; //prescaller at 0  RTC->MODE0.CTRL.bit.MODE=0x0; //mode 0 : 32 bit counter  RTC->MODE0.CTRL.bit.SWRST=0; //no software reset  RTC->MODE0.CTRL.bit.MATCHCLR=1; //automatic reset counter  //event configuration  RTC->MODE0.INTENSET.bit.OVF=1; //enable the event overflow  RTC->MODE0.INTENSET.bit.CMP0=1; //enable the event compare  RTC->MODE0.INTFLAG.bit.CMP0=1; //disable the flag COMPARE  RTC->MODE0.INTFLAG.bit.OVF=1; //disable the flag COMPARE  RTC->MODE0.COUNT.bit.COUNT=0; //set the RTC value  RTC->MODE0.COMP[0].reg = count\_set\_period;//set the value of the period  RTC->MODE0.CTRL.reg |= RTC\_MODE0\_CTRL\_ENABLE;//enable the timer  NVIC\_SetPriority (RTC\_IRQn, (1<<\_\_NVIC\_PRIO\_BITS) - 1);//configure interrupt  NVIC\_EnableIRQ(RTC\_IRQn);//enable interrrupt |
|  |

Like all peripherals that need a clock (clk) for this microcontroller we need to configure some mux and demux see datasheet of the microcontroller p90 for more detail.

In this case we have configure the OSC32K to going to the RTC clk this clock will go thought a GCLKGEN[4].

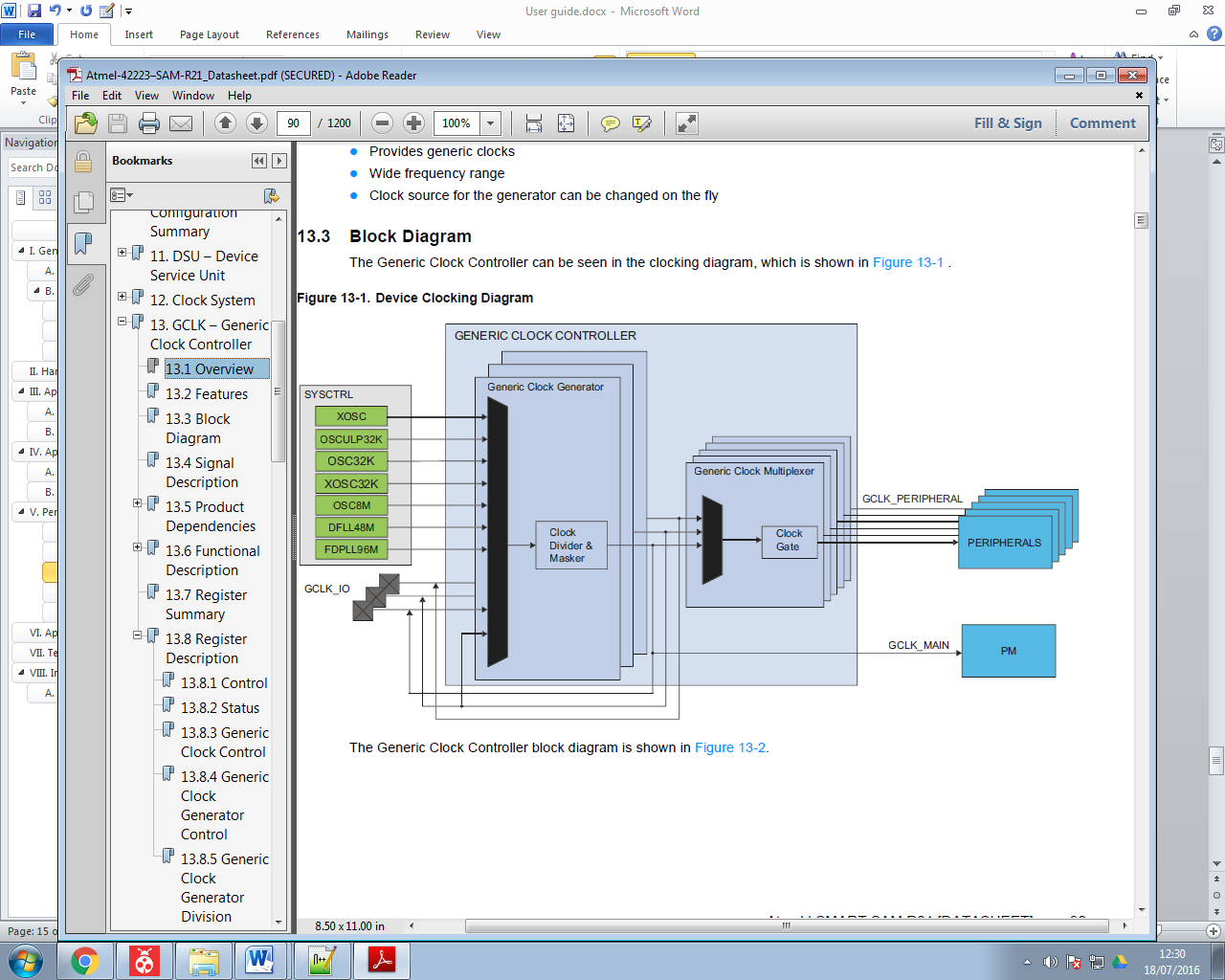


Figure Device clocking diagram

Once the clk set we need to configure the RTC. I have chosen to make a counter 32 bit in mode 0 (32 bit counter) without prescaler with an automatic reset of the timer.(see datasheet p 224 for more detail).

Now we need just to configure the interrupt, the function call after interrupt is RTC\_Handler();

### Serial communication

* General

This peripheral is use for the human machine interface (HMI). This peripheral is not really important because it is just need for human; you have just to be sure that this peripheral will not disturb the others.

* Configuration

Serial hmi(USBTX,USBRX) : begin a serial communication at 9600 bauds with no bit of parity and 1 stop bit.

See <https://developer.mbed.org/handbook/Serial> for more detail

### RF controller AT86RF233

* General

This chip is integrated in the microcontroller it is control the IEEE 802.15.4 radio. It is connected thought a SPI function. Some pins connected to interrupt are usefull to detect reception value and transmission.

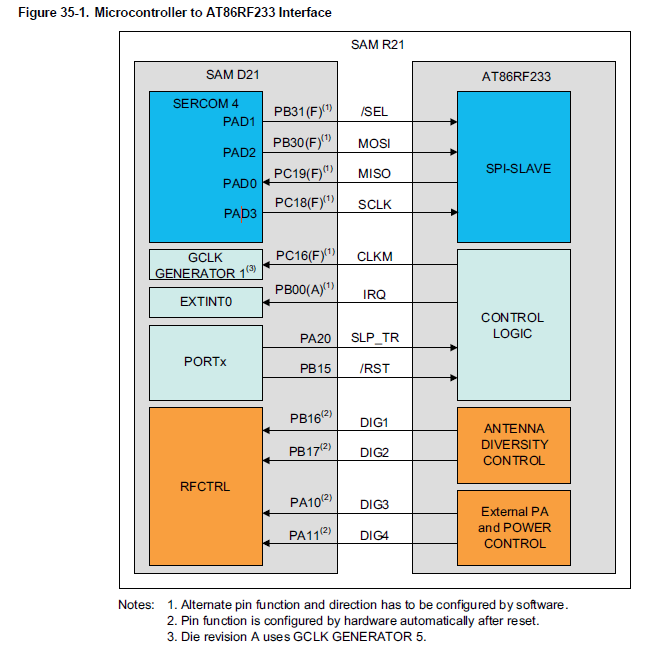


Figure 12 Microcontroller to AT86RF233 Interface

* Configuration

|  |
| --- |
| cMxRadio radio(PB30,PC19,PC18,PB31,PB15,PA20,PB00);//constructor radio pin of the R21 |

|  |
| --- |
| radio.attachIrq(ISRNetworkReception);//attach the radio interrupt  radio.begin(RADIO\_CHANNEL); // set the channel of the radio |

For more details see: <https://developer.mbed.org/teams/Smeshlink/code/MxRadioRF2xx>

* Application management

The interrupt will be manage directly by this library you have some function to read and write to the radio

# Application

In this part I will describe how my application work and with witch function. For each important function I have make a flow chart and define the most use function.

The flow chart have been generate with UMLET software you can find the flow chart at

<https://drive.google.com/file/d/0B2fXjrZjV__9aGswSUd1SWJHMG8/view?usp=sharing>

## Main Function

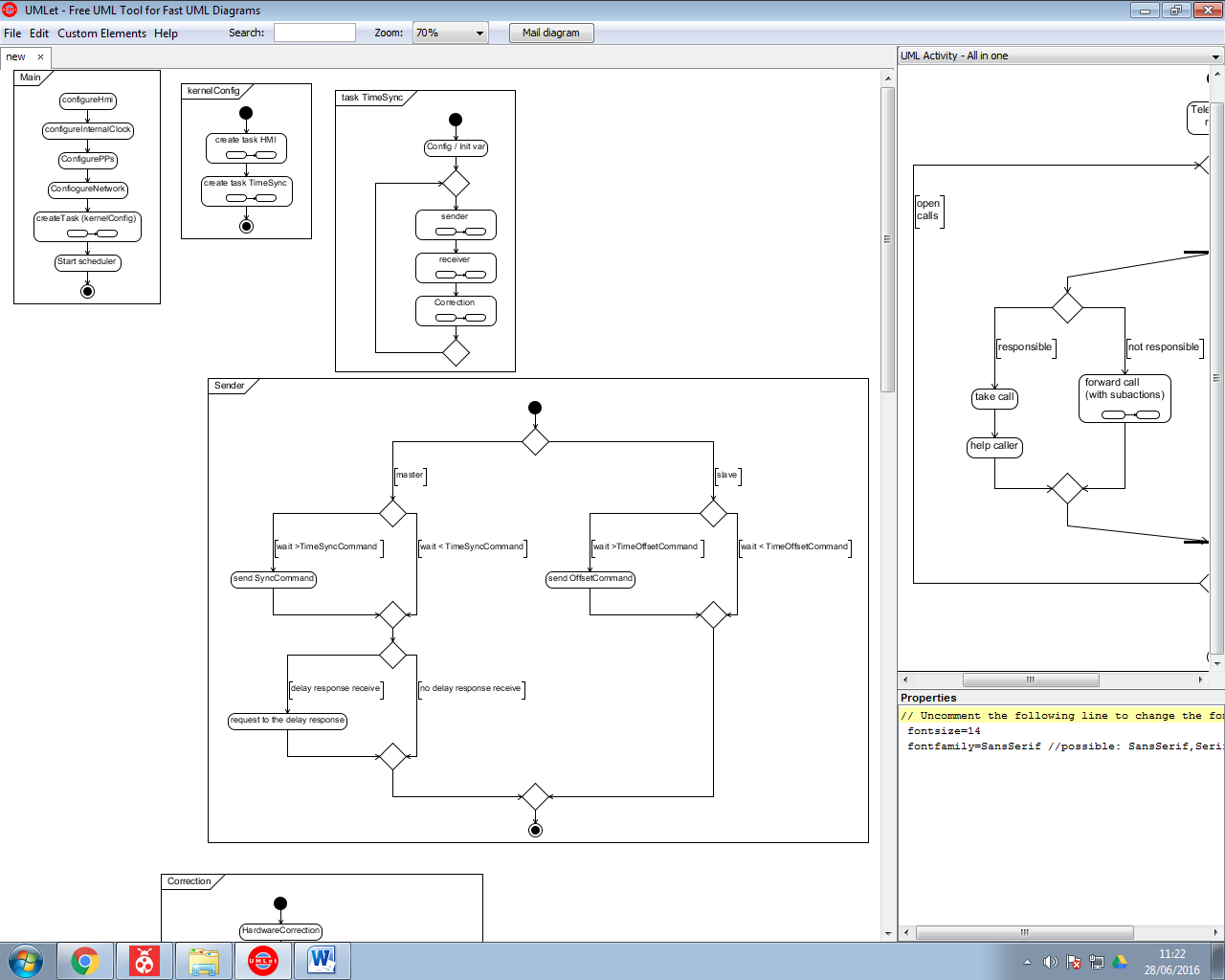
int main (void) (main.cpp)

Figure flow chart main

### Description

Call at the beginning of the application, configure peripherals create the task and run the scheduler

### Function

Hmi : void configureHMI(void) (hmi.h)

InternalClock : void configureInternalClock(void)(internalClock.h)

Pps : void configurationPPS(void) (ppsGPS.h)

Figure Flow chart main

Network : void configurationNetwork(void) (network.h)

Create Task : void kernelConfig(void) (utask.h)

Start Scheduler : void vTaskStartScheduler(void) (task.h)

## Task create

void kernelConfig(void) (utask.h)

### Description

Create the task, the task will be manage by the scheduler :

* Task HMI is priority 1 (low)
* Task TimeProtocol is priority 3 (high)

Figure flow chart kernel config

### Function

**Hmi task :** void HMITask(void) (hmi.h)

**Time Protocol task :** void timeProtocolTask(void) (timeProtocol.h)

## TimeProtocolTask

### Task

#### Description

This task will be running in a loop, and consist in 3 function to sending receiving and make the correction.

#### Function

**Function receiving:** void receiver(void) (timeProtocol.h)

**Function sending:** void sender(void) (timeProtocol.h)

**Function correction:** void correction(void) (timeProtocol.h)

Figure : flowChart timeSync task

### Sender

void sender(void) (timeProtocol.h)

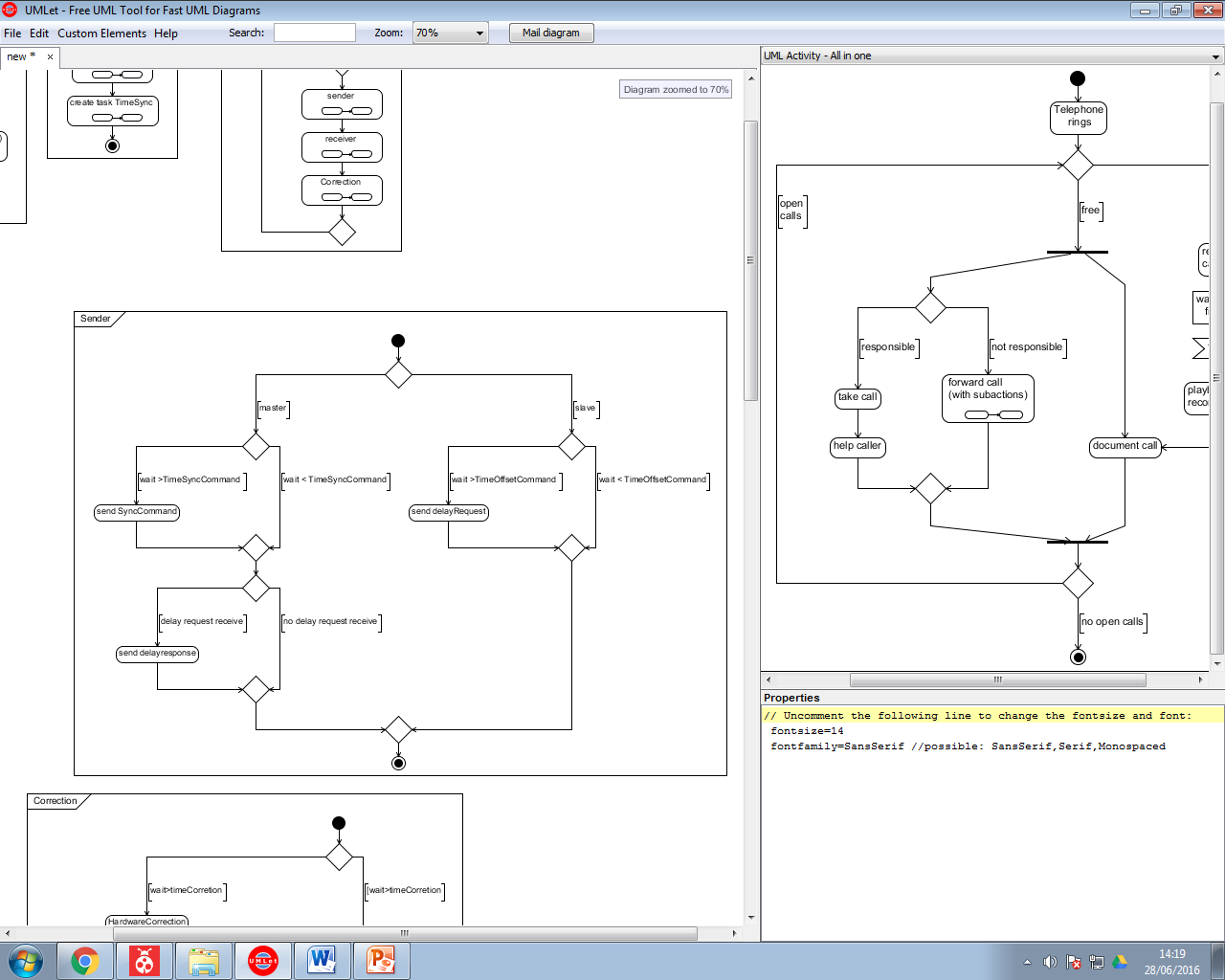


Figure flow chart sender function time protocol

#### Description

In function of the node it will send the correct command, all command are trigger by a time or an event.

Trigger event :

* **Sync Request :** time define in conf\_time\_protocol.h manage by the time of the RTOS
* **DelayRequest :** time define in conf\_time\_protocol.h manage by the time of the RTOS
* **Delay Response :** event declare if timeRequest is receive

#### Function

**Send Sync command :** void sync(void) (timeProtocol.h)

**Send Delay Request :** void delayRequest(void) (timeProtocol.h)

**Send Delay Response :** void delayResponse(uint8\_t id) (timeProtocol.h) with id the id of the slave

**Time sync :** #define TIMESYNC (conf\_timeProtocol.h)

**Time Delay Request :** #define TIMEDELAYREQUEST (conf\_timeProtocol.h)

### Receiver

void receiver(void) (timeProtocol.h)

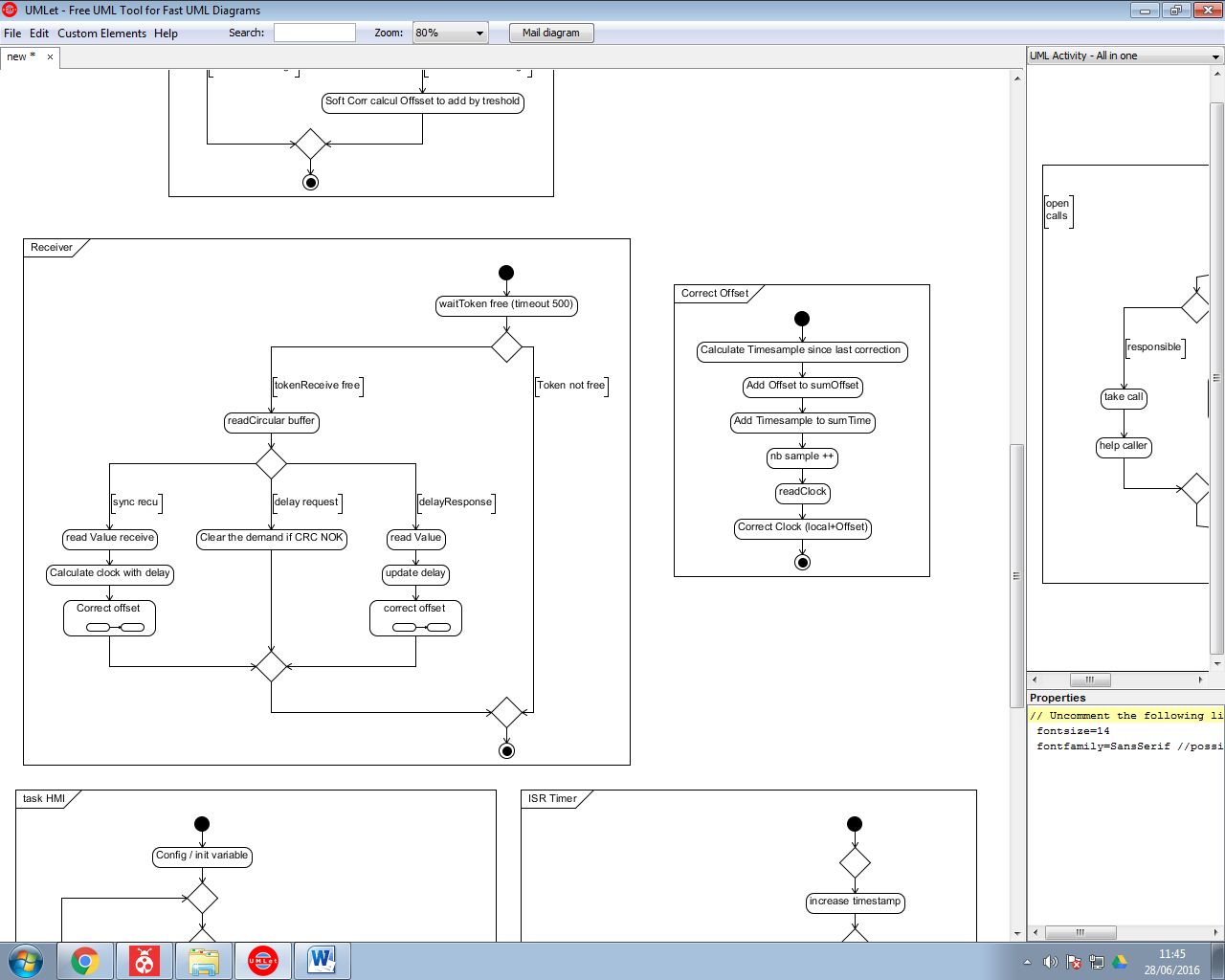


Figure flow chart receiver function timeProtocol

#### Description

The token will be free in an interrupt to synchronise the task with the reception (<http://www.freertos.org/binary-semaphore.gif>).

After he will parse the command and decide what he have to do.

#### Function

All the code are in void receiver(void) (timeProtocol.h)

**Function correctOffset :** void updateClock(void) (timeProtocol.h) the current offset is a global value. (timeprot.offset)

### Correction

void correction(void) (timeProtocol.h)

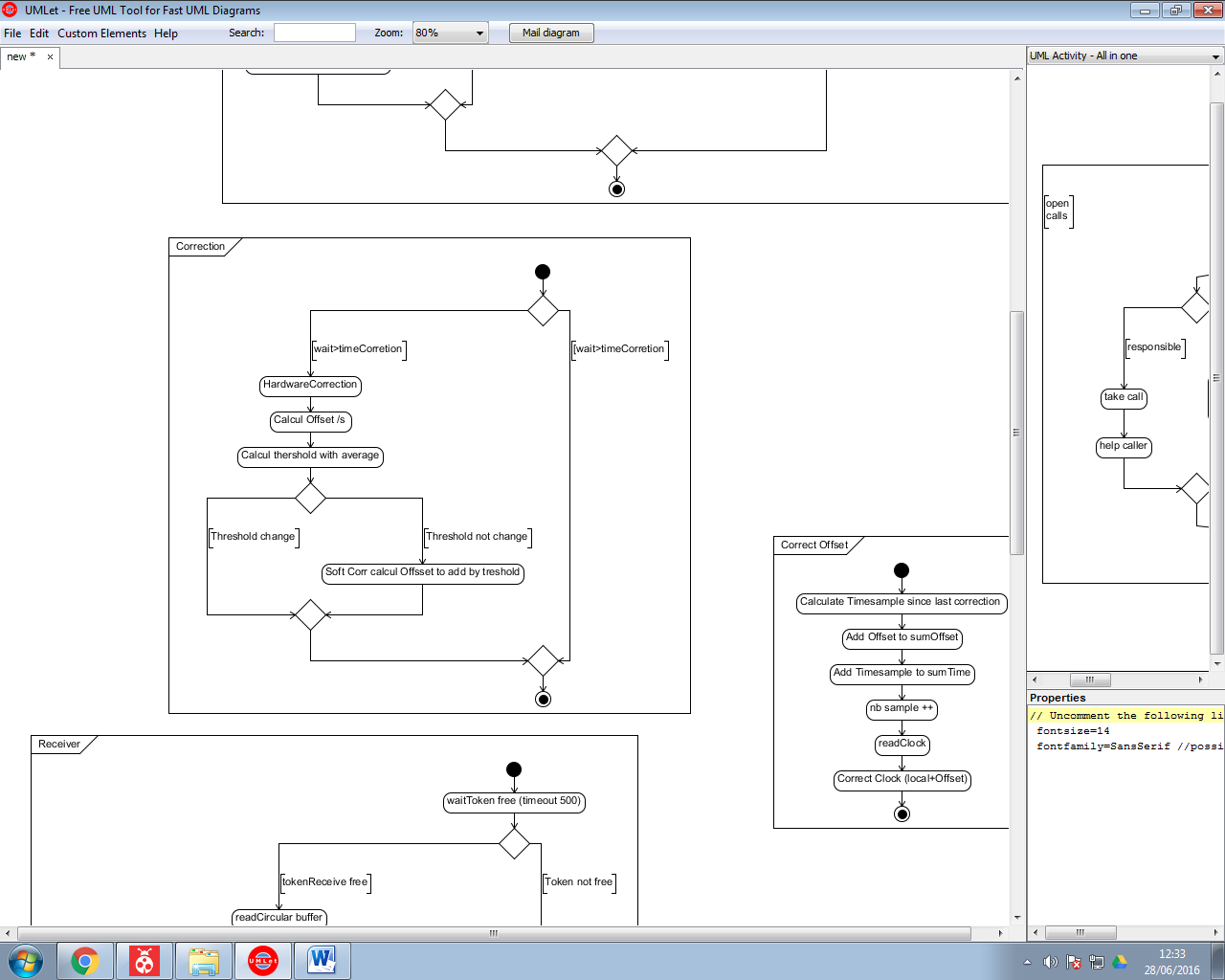


Figure Flow chat correction function time protocol

#### Description

Correction is trigger by a time or an event.

The programme have 2 things to correct :

* Hardware Correction : Hardware correction will change the threshold see part internalClock.
* Software Correction will correct the clock every n threshold .

If Hardware Correction is made you have to disable software correction up to the next correction.

#### Function

**Time Correction :** #define TIME\_CORRECTION (conf\_timeProtocol.h)

### Correct Offset

void updateClock(void)

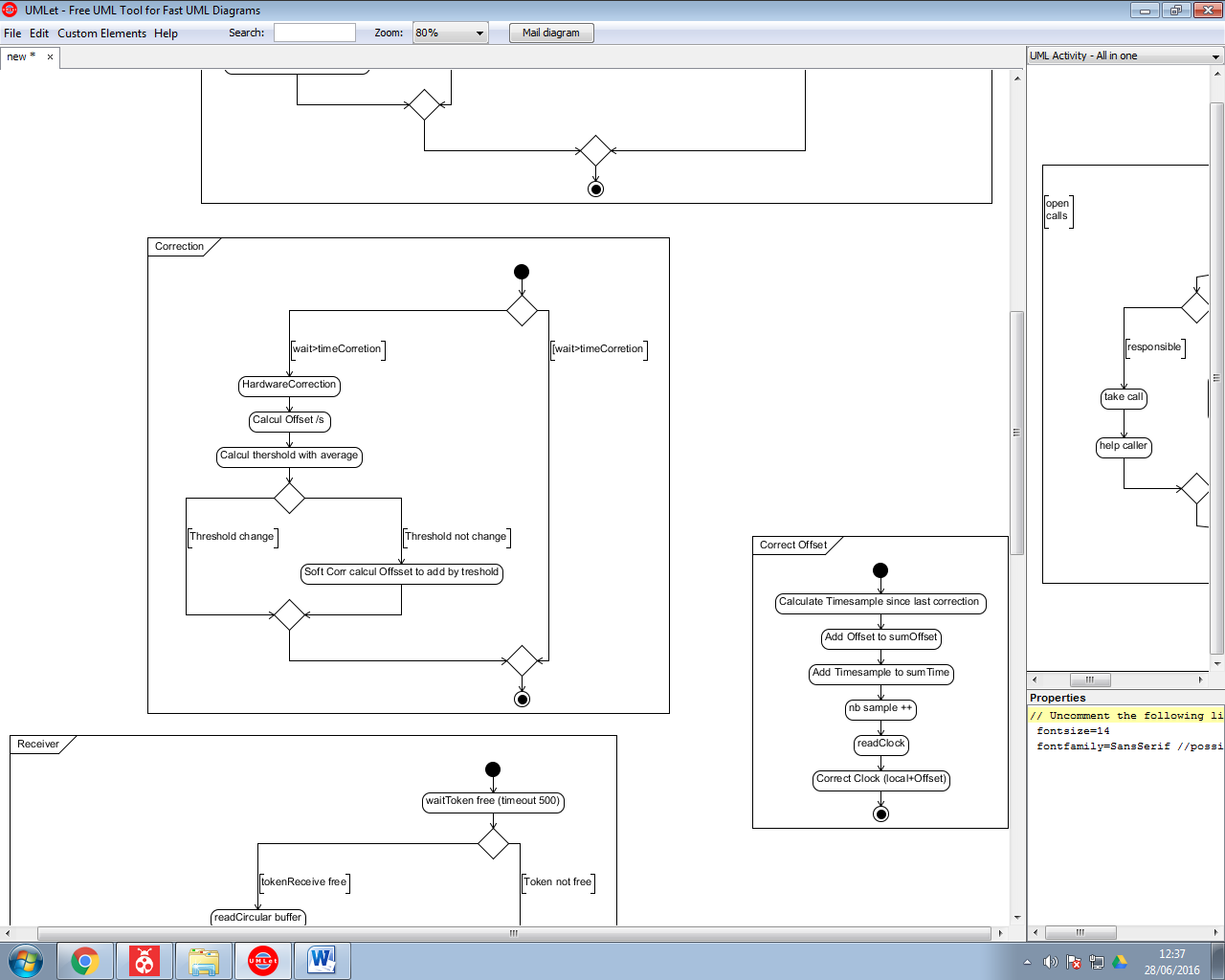


Figure update clock function time protocol

#### Description

For the correction he will save the offset and the timeSample. And after he will correct his clock.

#### Function

**Read clock :** uint32\_t readClock(Clock\* timeClock) (internalClock.h)

**Sum Offset :** sumOffset (timeProtocol.h)

**Sum Time :** timeProt.correction.sumTime(timeProtocol.h)

**Nb sample :** timeProt.correction.nbCorrection (timeProtocol.h)

## HMI Task

void HMITask(void) (hmi.h)

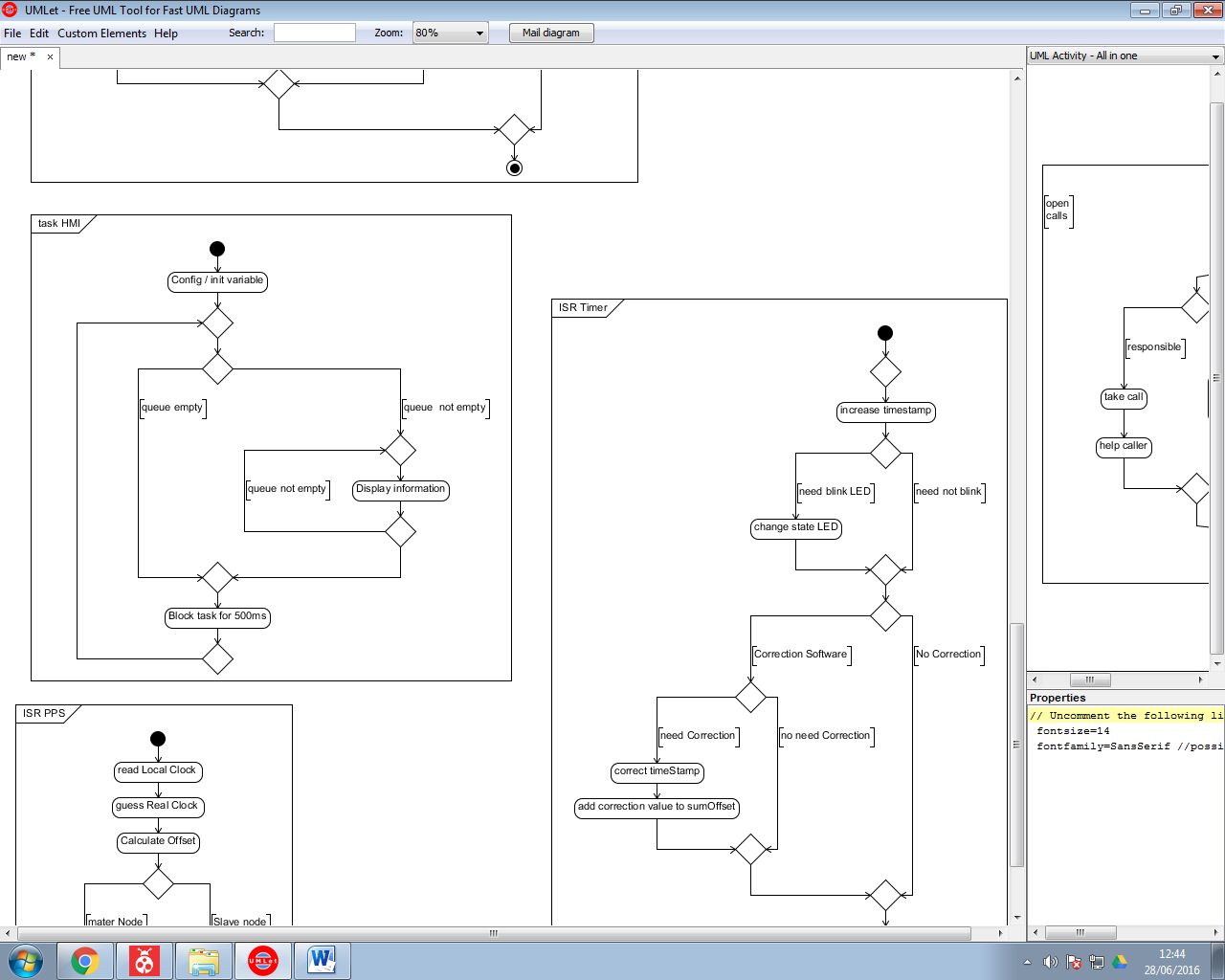


Figure 21 Flow chart hmi task

### Description

The HMI send to a UART communication the data in the queue.

### Function

**HMI queue :** xQueueHandle uartQueue (hmi.h)

## Interrupt

### PPS

void ppsISR(void)

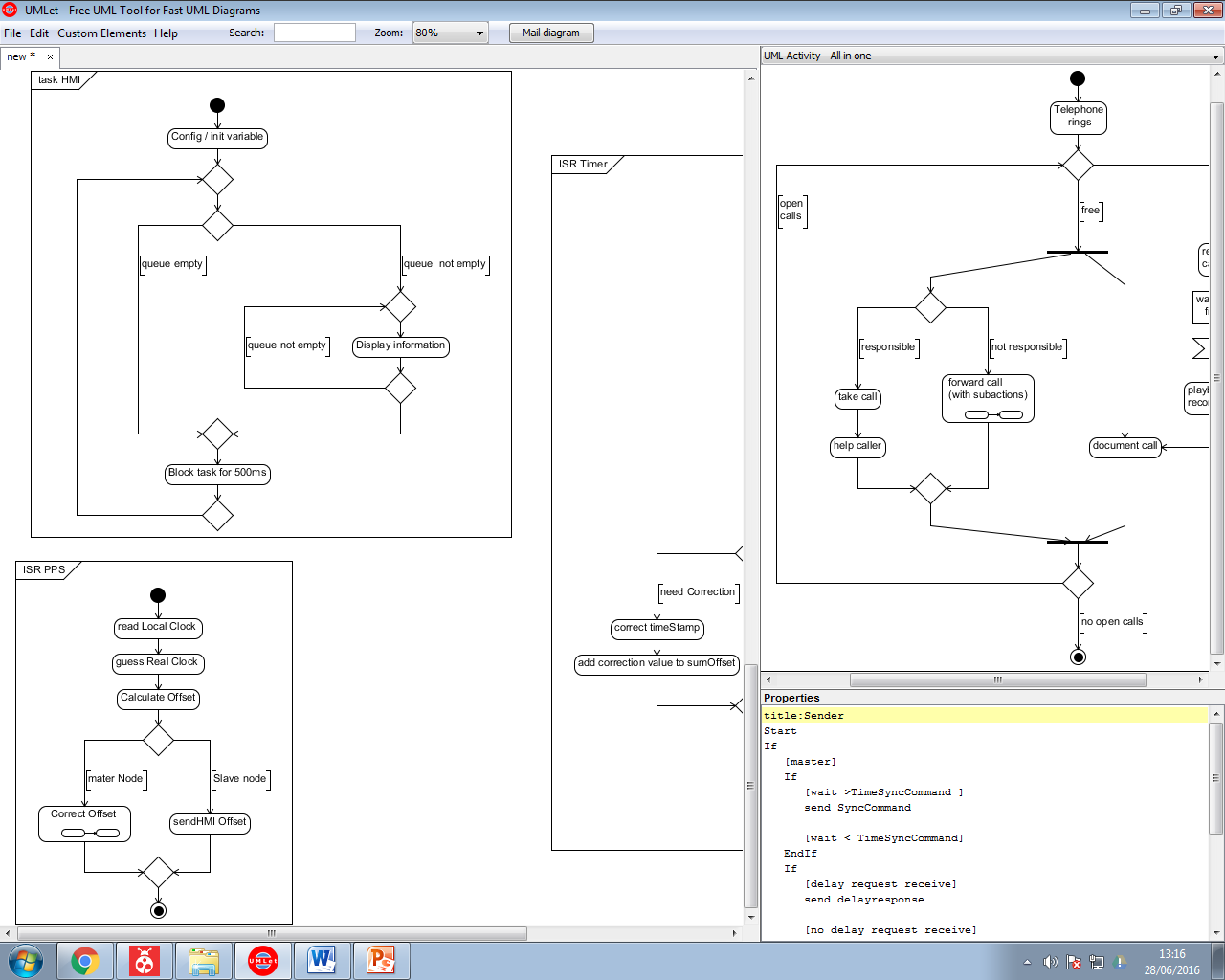


Figure : Flow chart pps task

#### Description

PPS will calculate the offset and print it for a Slave node or correct it if it is a master node.

### Internal Timer

void isrInternalClok(void )



Figure 23 Flow chart internal timer

#### Description

This interrupt will manage the led, and the timestamp. If there is a Software correction he will add the correction to the timestamp.

# Communication protocol

Header

Length

type

Data

CRC

ID

* Header a fix Header for the device now that is this protocol (8 bit)
* Lenght length in bytes(8bit)
* ID (8 bit) it’s the unique id of the device.
* Type type of command see section II
* Data data depend of type command.
* CRC :checksum (8bit)

## Type

There is a few types for the time synchronization

### SYNC type=1

Type = 1

Data= currentTime

In sync type master send a broadcast.

Data send his timestamp.

### Delay Request type=2

Type = 2

In delay request no have data.

The masters send a delay response after

### Delay Response type=3

The master request sends the Request timestamp. This timestamp is the time when the delay request is arrived. And send

Type = 3

id(8)

TimeStamp (8)

### Report type =4

Type = 4

Data= currentTime

Command sends by the slave for reporting the offset to the master. Like this the master can send EDMI command to the computer.

## CRC

CRC is calculed, it’s the sum of all the data+id+type.

# Test bench

For all the tests the hardware will be the same, the GPS will send a PPS to a slave and the master node. Each pulse the master and the slave will report the offset to the HMI. The network use is a wireless communication with IEEE 802.15.4.

All the measure of offset will be done with a sample time of 1s command by the PPS.

Figure test bench schematic

Slave Node

Master Node

GPS



HMI

Network

IEEE 802.15.4

PPS

Serial COM

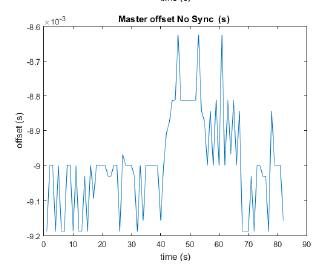
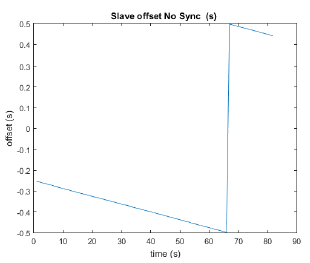
The threshold period is of 0,3125s for the entire test.

## Accuracy

This test is a final result of my project it will measure the accuracy of my project.

For the entire slave test the master will be with synchronization and with all the correction. The slave offset is the difference of time between the slave clock and the GPS clock.

**Basic system:**

In this test the network is not use just the PPS will correct the offset of the master but without correction.

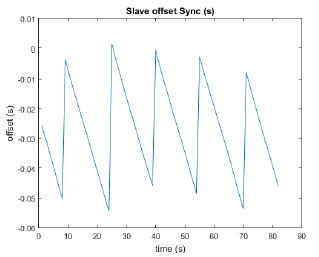
F

For the slave we can see that the offset is not change and that the difference of frequency is constant.

For the master we have accuracy quite constant.

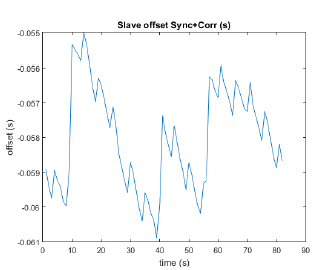
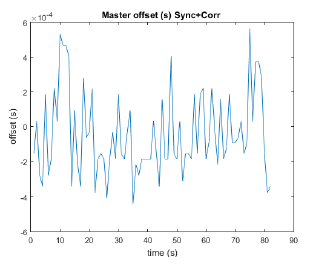
**With time synchronization:**

Now we don’t change the master and the slave will use the network to be synchronized. The synchronization will be done every 15s and we measure every second the offset.

We can see that every 15s the offset will be adjust and between 2 synchronization we have the difference of frequency that is constant.

**Correction:**

Now we can correct the threshold frequency using the time synchronization and the corrections.

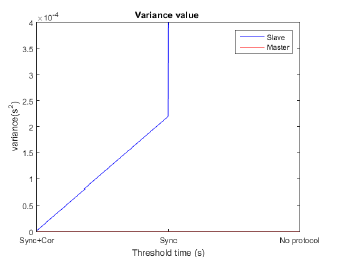
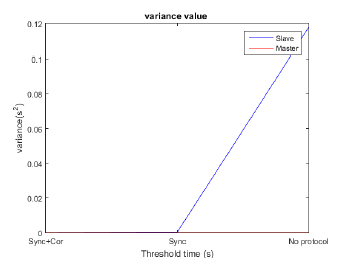


For the master the accuracy is a few hundred of us.

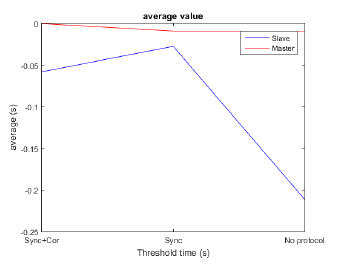
For the slave the accuracy is a few ms.

**Comparison:**

Now all done, we can make a comparison between all the update of my time protocol. What is very important to look is the variance because it is him that will influence a lot the accuracy because the average can be compensate if we know it.



We can see that the variance of the master is better than the variance of the slave it is good because it is the master needs that to synchronize the slave clock.



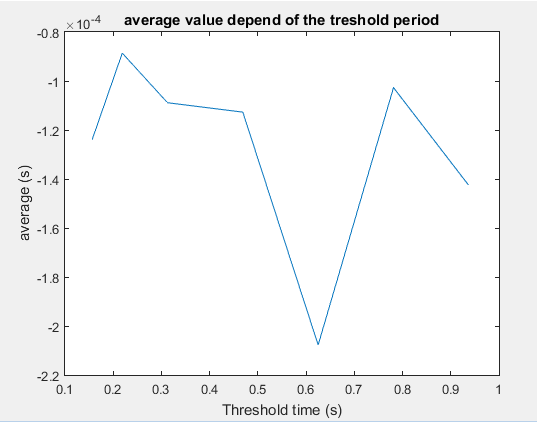
## Synchronization time

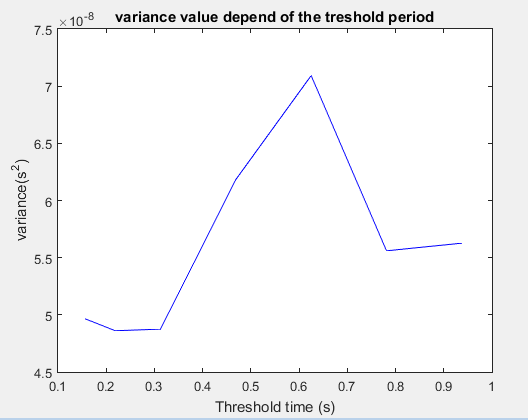
This test will be to test when you need the synchronization of the time for test this I will keep the same hardware that before and I will change the time between 2 synchronizations. I have calculated the average and the variance during 1 minute for a range of 3 to 15s.I will make the test with no delay request and a threshold period of 0.3125 s

* **Without correction**
* **With correction**

## Threshold period

With this test I want to test the influences of the threshold period I will keep the same hardware and just test for the master the average and the variance accuracy. I will calculate the value for a Tthreshold that going from 0.15 to 0.93s.

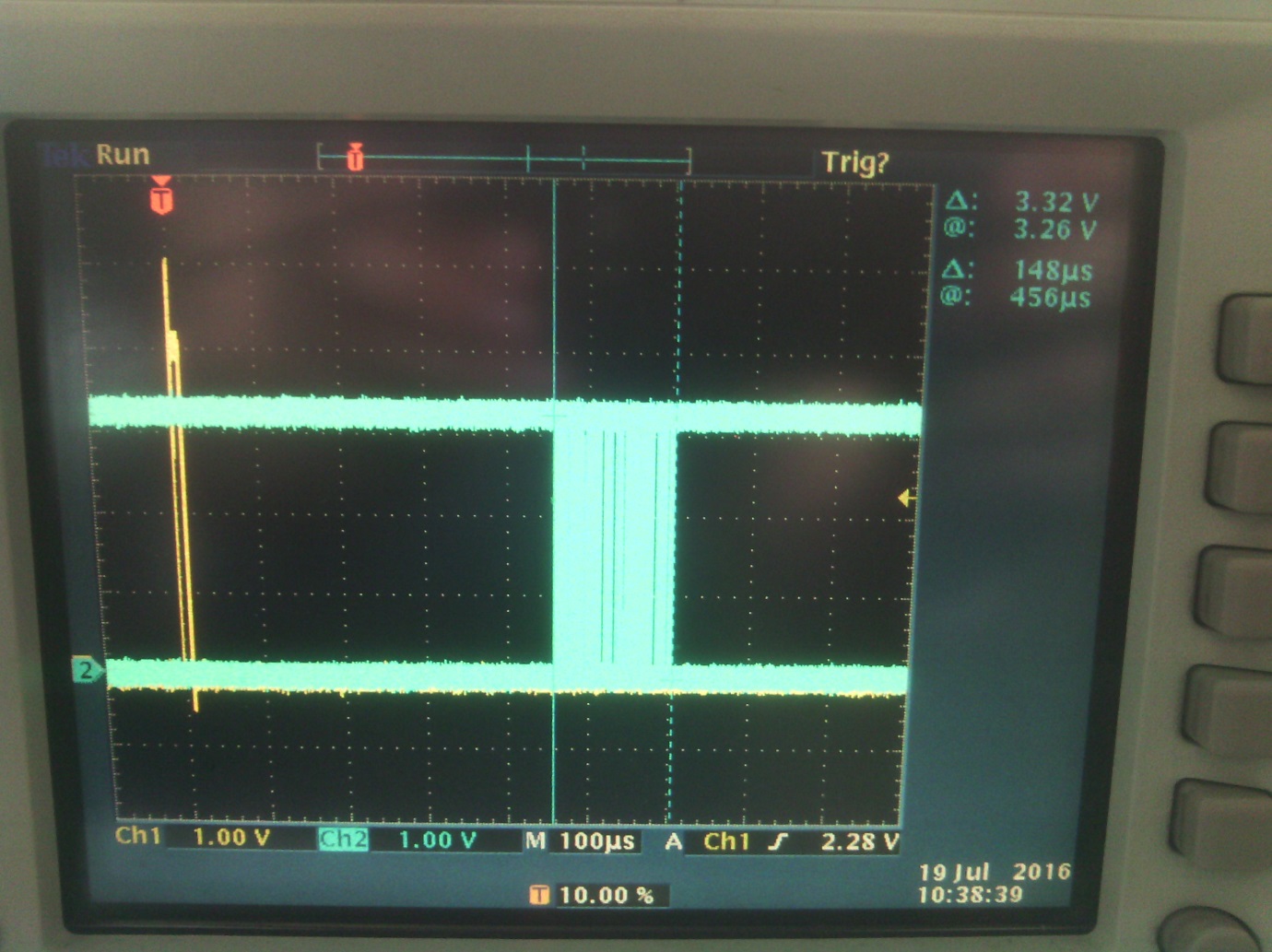




We can see that there is no relation between the threshold period and the offset.

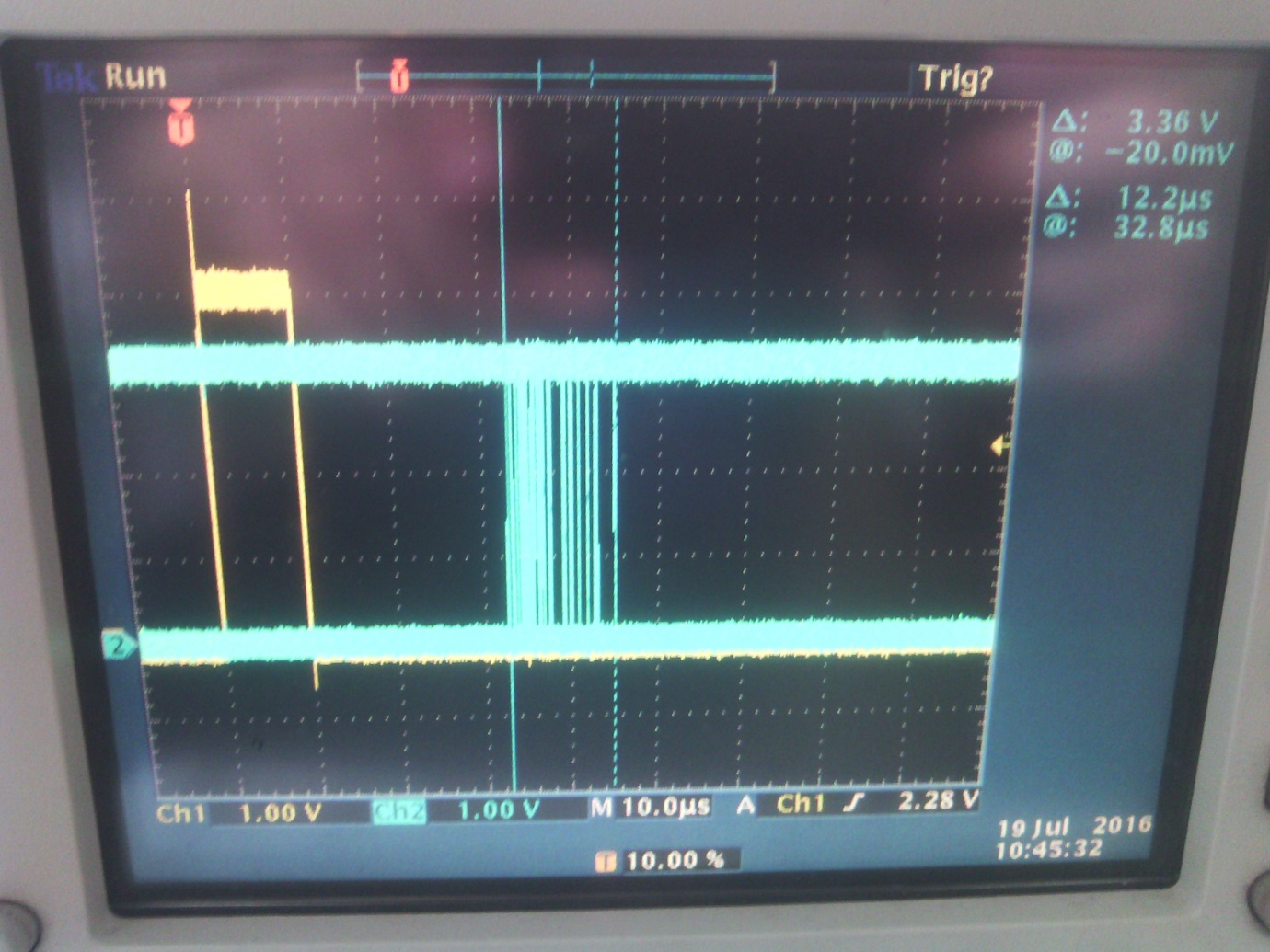
## Test interrupt delay

For testing the delay of the interrupt I will change a little my program to make a blinking led at each interrupt from the PPS the LED will be managed by mbed library. Like this I will see the average delay and the variation. The led will change just after reading the clock.



We can see that the variation of the delay is 148uS and the minimum delay is 456uS.This can be due to other interrupt that occurred at the same time (Timer, RF interrupt), and of course the calculation time. We can see too that the most of the value are in the first 60 us.

I have made another test in commenting all the line to read and to manage the time for just having the influences of the interrupt and not from some calculation.



Here we can see that the variation of the delay is 12.2us and the minimum delay is 32.8us. The variation of the delay can be explained due to the clock that will be used for detecting a rising edge. I don’t know witch clock is used for the internal Clock this depends of the other configuration.

To conclude our applications create a random delay and a constant delay.

# Use of application

This part will explain how to start the application with EDMI command:

* Download the project : <https://github.com/morandbaptiste/wirelessTimeSync-EDMIformaster/commits/master>
* Open with Atmel studio 7
* Configure file : conf\_timeProtocol.h (config folder)
  + Comment the define Master or Slave (master is the board connected to the PPS)
  + Optional : Select the different time
* Choose the board to program in Atmel studio (project/[name project] setup /tool)
* Program the board
* Make the same for the other board
* Connect the PPS to the master board
* Make the config in the file

[wirelessTimeSync-EDMIformaster\lab\node\_app\seriallogger\config.json](wirelessTimeSync-EDMIformaster/lab/node_app/seriallogger/config.json) (change the COM PORT and the baudrate if needed)

* Start the serial reader for the slave board (software tera term for example)
* Start the serial logger

[wirelessTimeSync-EDMIformaster\lab\node\_app\seriallogger\run.bat](wirelessTimeSync-EDMIformaster/lab/node_app/serialloggerrun.bat)

* The data will be save in [wirelessTimeSync-EDMIformaster\lab\node\_app\data](wirelessTimeSync-EDMIformaster/lab/node_app/data)

# Improvement

## Peripheral management

The best to have a high level of control is to manage all the application by controlling directly the register.

We know that the delay for an external interrupt for a rising edge is calculate with a clock this will generate a random delay of a few microsecond if we want a high accuracy we need to manage this delay. The timer interrupt have not the same problem because the delay of calculation is integrated in the delay when we calculate the delay with the PTP.

For the RF communication the best is to make your own library because the library that use mbed as some unpredictable bug. The best is to use a SPI using the register configuration or to be faster use ASF to configure the SPI and the interrupt.

## Application

* We have a problem for the blinking LED, the timer function will be called every threshold period this means that the LED avec an accuracy of the threshold period this means that if we use the PPS LED the accuracy can be very bad. I think the best solution is to use another timer to generate the PPS.
* The best is to use the external interrupt with assembler code for having a very high level of control of the time and like this we can create a constant delay and calculate the exactly value of this delay.
* To make an average value null we can directly compensate the offset like the figure bellow:

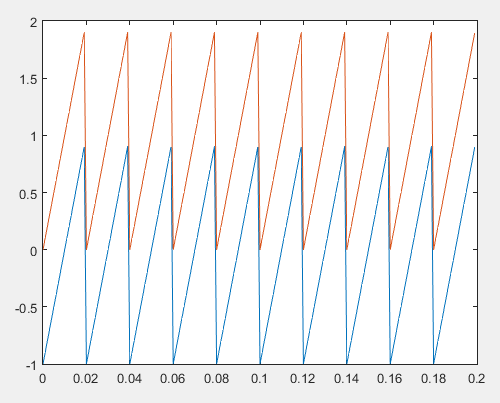


Figure Correction offset example

In the red curve it is the classical offset. In blue we compensate more the offset like this we have a better average.

* For the radio for sync command we are not able to put in the payload the data we sending, what I do is to read the clock and after sending this create a delay, what we can do is to create a second command of follow up and put the previous data in the payload. The time will be collected by an interrupt.

# Contact

If you have some problem and you have try to understand but you not arrived you can contact me at this email: [b.morand.etu@gmail.com](mailto:b.morand.etu@gmail.com)