PROJECT PLANNING DOCUMENT

Time synchronization for wireless Sensor Network

5/9/2016

MORAND Baptiste

Contents

I. Introduction, Background and research

With the development of technology and application of new products, more accurate time control is needed, for example, the times of occurrence of physical events are often crucial for the observer to associate event reports with the originating physical events. Physical time is also crucial for determining properties such as speed or acceleration.

In sensor networks, many sensor nodes may observe a single physical phenomenon. One of the key functions of a sensor network is hence the assembly of those distributed observations into a coherent estimate of the original phenomenon. This process is known as data fusion.

II. Aim and objectives

A. AIM

The aim of this project is to develop and implement a precision time synchronization protocol for industrial wireless 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.

The C/C++ language will be used to develop the driver for IEEE 802.15.4 radio chip on an ATSAMR21G18A microcontroller.

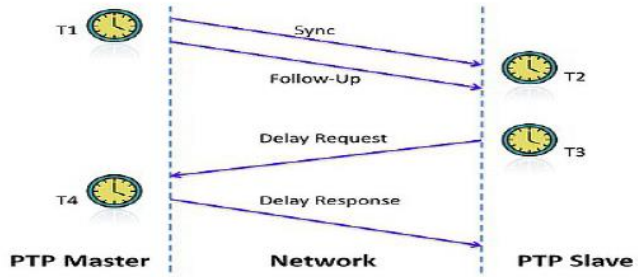
B. Objectives

- Understanding IEEE 1588 Precise time protocol.
- Understanding IEEE 802.15.4 and use it with the microcontroller.
- Write code in C/C++ language to realize the algorithm.
- Load the C/C++ into the microcontroller.
- Test the algorithm and test his accuracy.
- Use the GPS to be at the time of UTC, and have the localization of the device
- Synchronize the master with the PPS of the GPS.

III. Literature Review

A. IEEE 1588

IEEE1588 is standardized and contains the precision time protocol (PTP). It is a network protocol which secures the synchronism of the time settings of all devices within a network with the focus on high accuracy in a local defined network. The precise time synchronization is necessary to permit the accurate reconstruction of an event from operational logs of different devices or for high-precision applications like the process bus. A PTP network consists of communicating clocks. It will be determined with the best master clock algorithm (BMCA) which device has the time with the highest accuracy. This device is then the according reference clock and will be called as the Grandmaster clock.



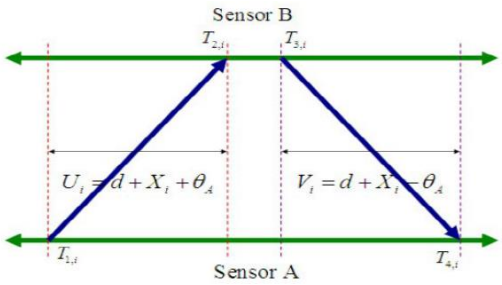
There is 2 times to know:

- Offset : who is the retard between the master time and the slave time
- The delay: who is the time of transmission.

To know this time IEEE1588 use this protocol

1. Sync: The process is started when the master send Sync packet with the timestamp who correspond to the time where master send.
2. The Slave save the time when the packet is received.
3. Delay request: The slave send a delay request to the master with no data
4. Delay Response: The master request with the time when he have receive the request

With this information he can correct the delay and the offset.



d is the delay, X_i a random delay, θ_A is the offset.

With the information we can show that:

$$\theta = \frac{(T2-T1)-(T4-T3)}{2} \quad d = \frac{(T2-T1)+(T4-T3)}{2}$$

When can conclude that time:

$T_{slave}=T_{slave}$-delay+offset

B. IEEE 802.15.4

The 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.

LR WPAN characteristics are:

- the formation of a star or mesh-type network,
- the allocation of a 16-bit or 64-bit address,
- using CSMA / CA to communicate,
- low power consumption,
- the energy detection (ED)
- an indication of the link quality (LQI),
- the use of :
    - 16 channels in the frequency band from 2.4 to 2.4835 GHz,
    - 10 channels in the frequency band of 902-928 MHz,
    - 1 channel in the 868 to 868.6 MHz band.

C. FREERTOS

If you want more information go to freertos website.

- **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,…).

- **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.

- **What is the footprint of FREERTOS ?**

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

- IAR STR71x ARM7 port.
- Full optimisation.
- Minimum configuration
- Four priorities.

How much RAM does FreeRTOS use?

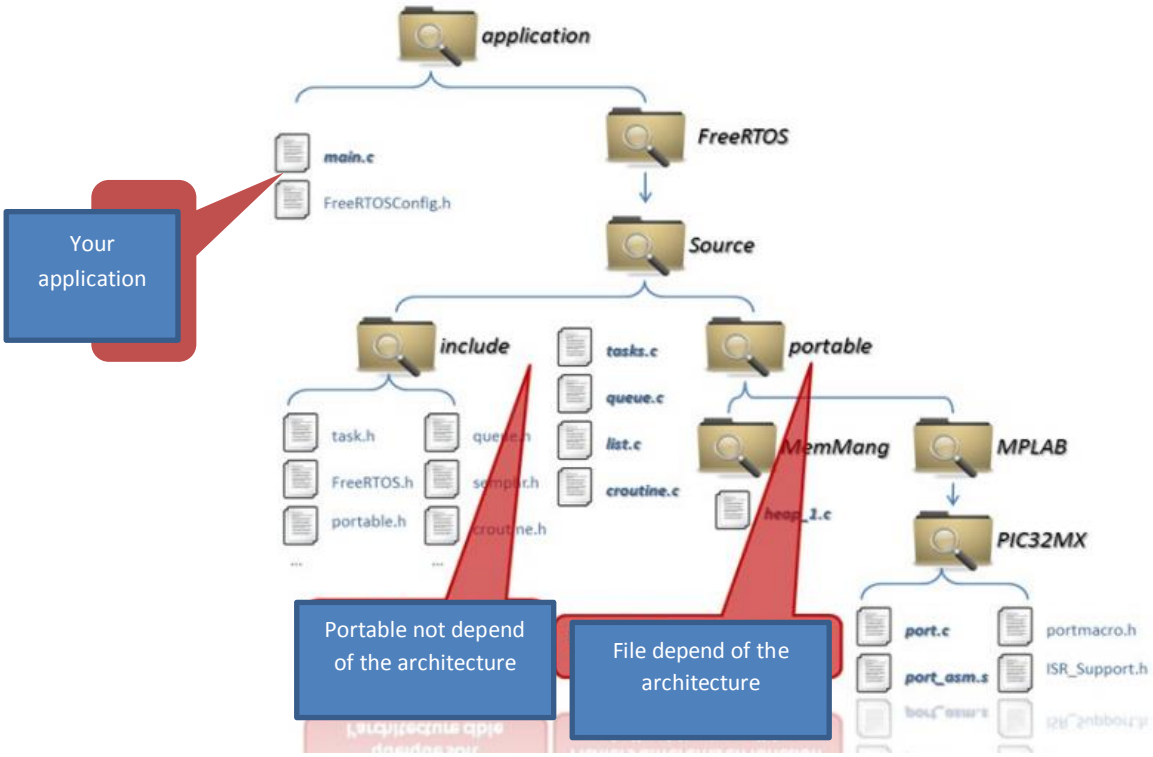
Item	Bytes Used
Scheduler Itself	236 bytes (can easily be reduced by using smaller data types).
For each queue you create, add	76 bytes + queue storage area (see FAQ Why do queues use that much RAM?)
For each task you create, add	64 bytes (includes 4 characters for the task name) + the task stack size.

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.



- **What is the MCU supported by FREERTOS?**

These demos can be adapted to any microcontroller within a supported microcontroller family. See the Creating a new FreeRTOS application and Adapting a FreeRTOS Demo documentation pages. FreeRTOS ports are categorised as either being officially supported, or contributed. The Official and Contributed Definitions page describes the categories, and the rationale for making the distinction.

This page only lists the official RTOS ports:

- Actel (now Microsemi)
  - Supported processor families: SmartFusion, SmartFusion2 - see Microsemi listing
  - Supported tools: IAR, Keil, SoftConsole (GCC with Eclipse)
- Altera
  - Supported processor families: Cyclone V SoC (ARM Cortex-A9), Nios II
  - Supported tools: Altera SoC EDS (ARM DS-5 with GCC), Nios II IDE with GCC
- Atmel
  - Supported processor families: SAMV7 (ARM Cortex-M7), SAM3 (ARM Cortex-M3), SAM4 (ARM Cortex-M4), SAMD20 (ARM Cortex-M0+), SAMA5 (ARM Cortex-A5), SAM7 (ARM7), SAM9 (ARM9), AT91, AVR and AVR32 UC3
  - Supported tools: IAR, GCC, Keil, Rowley CrossWorks
- Cortus
  - Supported processor families: APS3
  - Supported tools: Cortus IDE with GCC
- Cypress
  - Supported processor families: PSoC 5 ARM Cortex-M3
  - Supported tools: GCC, ARM Keil and RVDS - all in the PSoC Creator IDE
- Freescale
  - Supported processor families: Kinetis ARM Cortex-M4, Coldfire V2, Coldfire V1, other Coldfire families, HCS12, PPC405 & PPC440 (Xilinx implementations) (small and banked memory models), plus contributed ports
  - Supported tools: Codewarrior, GCC, Eclipse, IAR
- Infineon
  - Supported processor families: TriCore, XMC4000 (ARM Cortex-M4F), XMC1000 (ARM Cortex-M0)
  - Supported tools: GCC, Keil, Tasking, IAR
- Fujitsu (Now Spansion)
  - Supported processor families: FM3 ARM Cortex-M3, 32bit (for example MB91460) and 16bit (for example MB96340 16FX)
  - Supported tools: Softune, IAR, Keil
- Luminary Micro / Texas Instruments
  - Supported processor families: All Luminary Micro ARM Cortex-M3 and ARM Cortex-M4 based Stellaris microcontrollers
  - Supported tools: Keil, IAR, Code Red, CodeSourcery GCC, Rowley CrossWorks

- Microchip
  - Supported processor families: PIC32MX, PIC32MZ, CEC13xx, PIC32MZ EF, PIC24, PIC24EP, dsPIC,
  - Supported tools: MPLAB C32, MPLAB C30
- Microsemi
  - Supported processor families: SmartFusion, SmartFusion2
  - Supported tools: IAR, Keil, SoftConsole (GCC with Eclipse)
- NEC (now Renesas)
  - Supported processor families: V850 (32bit), 78K0R (16bit)
  - Supported tools: IAR
- NXP
  - Supported processor families: LPC1500 (ARM Cortex-M3), LPC1700 (ARM Cortex-M3), LPC1800 (ARM Cortex-M3), LPC1100 (ARM Cortex-M0), LPC2000 (ARM7), LPC4000 (ARM Cortex-M4F/ ARM Cortex-M0)
  - Supported tools: GCC, Rowley CrossWorks, IAR, Keil, LPCXpresso IDE, Eclipse
- Renesas
  - Supported processor families: RZ/A1 (ARM Cortex-A9), RX700 / RX71M, RX600 / RX64M / RX62N / RX63N, RX200, RX100, SuperH, RL78, H8/S plus contributed ports
  - Supported tools: GCC, HEW (High Performance Embedded Workbench), IAR Embedded Workbench
- Silicon Labs [ex Energy Micro]
  - Supported processor families: EFM32 Gecko (Cortex-M3 and Cortex-M4F), 8051 compatible microcontrollers.
  - Supported tools: Simplicity Studio (GCC), IAR, SDCC
- Spansion
  - Supported processor families: FM3 ARM Cortex-M3, 32bit (for example MB91460) and 16bit (for example MB96340 16FX)
  - Supported tools: Softune, IAR, Keil
- ST
  - Supported processor families: STM32 (ARM Cortex-M0, ARM Cortex-M7, ARM Cortex-M3 and ARM Cortex-M4F), STR7 (ARM7), STR9 (ARM9)
  - Supported tools: IAR, Atollic TrueStudio, GCC, Keil, Rowley CrossWorks
- TI
  - Supported processor families: RM48, TMS570, ARM Cortex-M4F MSP432, MSP430, MSP430X, Stellaris (ARM Cortex-M3, ARM Cortex-M4F)
  - Supported tools: Rowley CrossWorks, IAR, GCC, Code Composer Studio
- Xilinx
  - Supported processor families: Zynq, Zynq UltraScale+ MPSoC (64-bit ARM Cortex-A53 and 32-bit ARM Cortex-R5), Microblaze, PPC405 running on a Virtex4 FPGA, PPC440 running on a Virtex5 FPGA.
  - Supported tools: GCC
- Intel/x86

- Supported processor families: IA32 (32-bit flat memory model), Quark SoC X1000 (32-bit flat memory model), any x86 compatible running in Real mode only, plus a Win32 port
- Supported tools: GCC, Visual Studio 2010 Express, MingW, Open Watcom, Borland, Paradigm

- Contributed ports...
  - Contributed ports exist for Tricore, MICO32, Blackfin, Jennic, eZ80, SuperH and others. Contributed ports are provided "as is" and are not supported directly.

D. Hardware platform

1. ThunderBolt® E GPS Disciplined Clock

- +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 to connect to a pc
- antenna interface

2. Board Atmel SAMR21 Xplained Pro

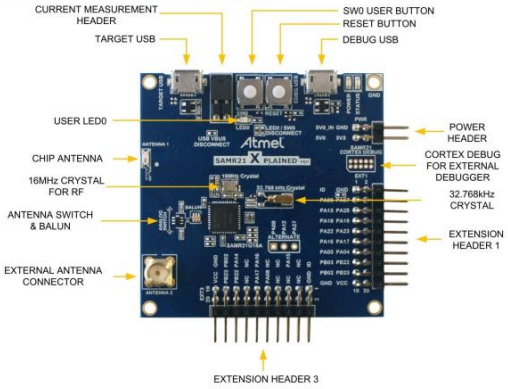
a) Microcontrôleur ATSAMR21G18A

- 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,…)
  - 8 ADC channel
  - Watchdog timer
  - Integrated Ultra Low Power Transceiver for 2.4GHz ISM Band

b) Board

- Supply voltage : 5V (4,4 to 5,25 via debugger or target usb or 4,3 to 5,5V via external power)
- peripherals

Figure 1-1. SAM R21 Xplained Pro Evaluation Kit Overview



IV. Proposed Project

A. Task of the project

This bloc diagram illustrates the framework of the time synchronization. We can divide this project in part:

- **Use of board Atmel SAMR21 Xplained Pro**
  - UART for debugging and communication.
  - Interrupt for PPS.
  - Timer for keeping the time.
  - SPI interface for Wireless IEEE802.15.4
- **RTOS**
  - Add FREERTOS in the project.
- **IEEE 1588**
  - Make the protocol
  - Test the protocol in UART
  - Test the protocol in wireless Sensor
  - Improve the protocol
- **GPS synchronization**
  - Use the GPS PPS to synchronize the master
  - Test GPS synchronization
  - Take the UTC time via RS232.

Obviously, the Use of Atmel SAMR21 Xplained Pro have to be make first, and after IEE 1588 and GPS synchronization can be made in parallel.

B. Gantt planning

timeSynchro 6 mai 2016

Tâches 2

Nom	Date de début	Date de fin
Use of the board	10/05/16	07/06/16
Use of UART	10/05/16	10/05/16
Use of interrupt	10/05/16	10/05/16
Use of timer	11/05/16	11/05/16
Use of wirelless	30/05/16	07/06/16
RTOS	12/05/16	13/05/16
Add Freertos	12/05/16	13/05/16
IEEE 1588	16/05/16	15/06/16
make the protocol	16/05/16	27/05/16
Test on UART	27/05/16	30/05/16
Test on Wireless	07/06/16	15/06/16
GPS syncronization	18/05/16	13/07/16
Use the GPS PPS to synchronize the master	18/05/16	26/05/16
Test the sychronization	26/05/16	26/05/16
Use the GPS RS232 to synchronize to UTC time	16/06/16	13/07/16
Test and improve	13/07/16	29/07/16

See file planning.jpeg.

C. Project proposal

1. Toolchain

I will use the toolchain Atmel AVR Toolchain, and I will use the computer software atmel studio.

2. Use of the board

I have seen in some report that people have already use the board and have already use the peripheral of interrupt and timer. I will use their work to use the board. The wireless has not been use already and will be make after.

3. RTOS

4. IEEE 1588

a) How we will proceed

Before use the application on a wireless we are using on a UART because I know more the communication protocol and because we have not a lot of problem like packet lost…

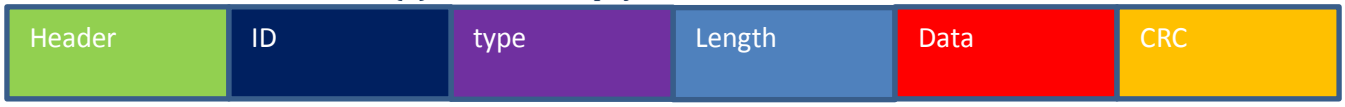
After the time synchronization protocol validate we can go to the wireless.

b) How validate

For validate first we will make a blink gpio all the 500 ms in the slave and in the master. And we can watch it in the oscilloscope.

c) Protocol

(1) Protocol payload



- Header a fix Header for the device now that is this protocol (8 bit)
- Lenght length in bytes(8bit), not send if length=0

- ID (8 bit) it's the unique id of the device.
- Type type of command see section (2)
- Data data depend of type command.
- CRC :checksum (8bit)

(2) Type

There is a few types for the time synchronization

(a) SYNC type=1

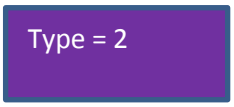
In sync type master send a broadcast.

Data send is his timestamp.

(b) Delay Request type=2

In delay request no have data.

The master sending a delay response after



(c) Delay Response type=3

The master sends the request timestamp. This timestamp is the time when the delay request is arrived and the time when he send the delay response. And he sends the id of the request.

(d) ACK type=4

Data=0 if acknowledge is ok. Else send an error code.

There is different error type that exists:

- Error CRC (data=1);

(e) Ping type=5

Send in the data the device id you want to ping.

(3) CRC

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

5. GPS synchronization

PPS to synchronize the master

First we wil use interrupt for detect the pulse, and a counter to know the clock between 2 pulse.

For testing we will use an oscilloscope, and blinking a gpio all 500ms and we can compare to the GPS PPS.

Use the RS232 to take the UTC time

We can synchronize to the UTC time via RS232 of the GPS. For this we need a component RS232 to UAR transceiver. And after we need reading the UART command of the GPS.

D. Application

1. Software

a) Master

We have 5 tasks for FREERTOS:

- Blinking led (priority 5)
- UART or WIRELESS protocol (Priority 4)
- UART GPS reading (priority 3)
- IHM (Priority 1)
- Read data(priority 1)

We have several interrupt:

- External interrupt.
- TIME counter Interrupt.

Now we will see how the Task of RTOS will be use:

- Blinking led test

This is a task who validate the time synchronization, all 500ms the led is changing of state. If you take this information to a oscilloscope you can see if there Is a synchronization.

- UART or Wireless protocol

Each 1s the master send a broadcast timeSynchronisation to synchronize all the slave clock.

If a command of delay request is coming the master request.

- UART GPS reading

Use to update the timeStamps to the correct value. With this function we can update the timestamps to the correct UTC time.

- IHM

When the microcontroller has the time he sends his current timeStamps to an UART.

- Read data

If you have to read a data you can make this in this task.

Now we will see how Interrupt work

- External Interrupt

When the GPS send a pulse, the interrupt ubdate the timestamps, and reset the time counter.

- TimeCounter Interrupt

All the 10ms there is an interrupt and the timeStamps will be update.

b) Slave

We have 4 tasks for FREERTOS:

- Blinking led test (Priority 5)
- UART or WIRELESS protocol (Priority 4)
- IHM (Priority 1)
- Read data(priority 1)

We have several interrupts:

- TIME counter Interrupt

Now we will see how the Task of RTOS will be use :

- Blinking led test

This is a task who validate the time synchronization, all 500ms the led is changing of state. If you take this information to a oscilloscope you can see if there Is a synchronization.

- UART or WIRELESS protocol

At the beginning when the slave clock receive a broadcastSynchronization the slave send a delay request.

After all the 1 minutes, a delay request is request.

- IHM

When the microcontroller has the time he sends his current timeStamps, his delay to an UART.

- Read data

If you have to read a data you can make this in this task.

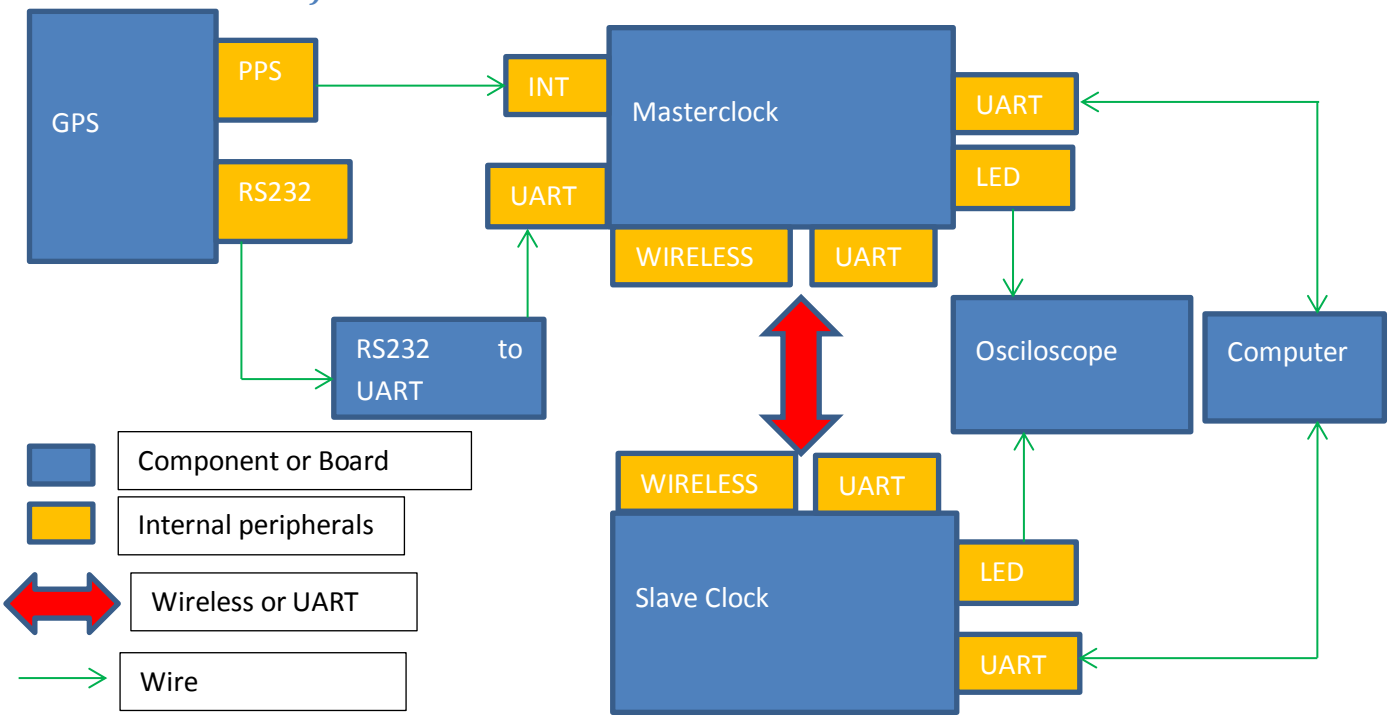
Now we will see how Interrupt work

- Time Counter interrupt

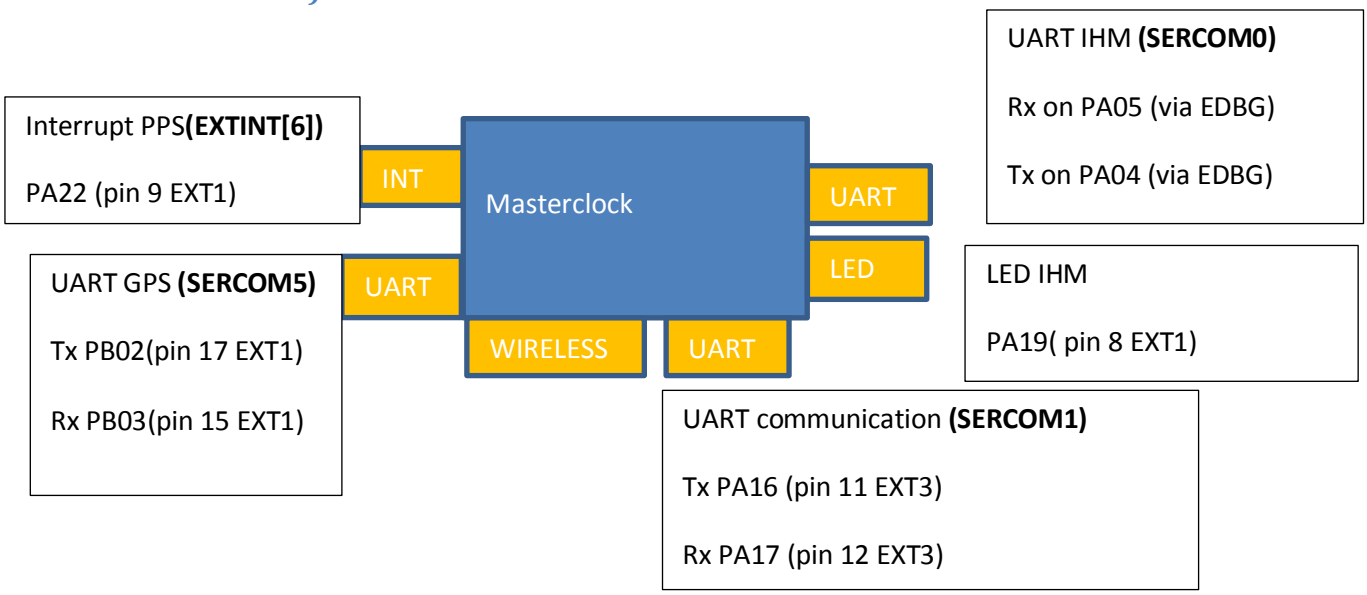
All the 10ms there is an interrupt and the timeStamp will be update.

2. Hardware

a) General



b) Master clock

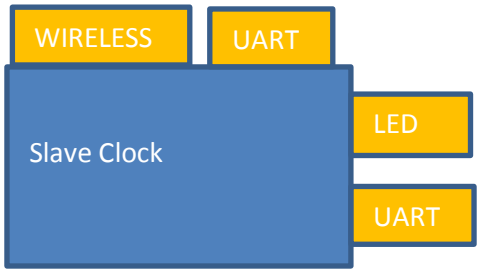


c) Slave Clock

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)