Time Synchronization in Limited Resources Wireless Devices

F.B. Nascimento and P.B. Lopes
Universidade Presbiteriana Mackenzie
São Paulo, Brazil

Corresponding Author: Paulo Batista Lopes
Address: Universidade Presbiteriana Mackenzie
Rua da Consolação 896
01302-907 São Paulo Brazil
E-mail: paulo.lopes@mackenzie.br
Telephone: +55 11 2114-8512

Abstract

This article deals with implementation of a time distribution protocol among limited resources wireless sensor devices. This type of protocol aims to distribute the same time reference among two or more devices within a network. Timestamps on measures help to relate them on causality studies. The PTP protocol defined at IEEE 1588-2008 is used, with little adaptations, to synchronize and tune software clocks inside the target boards of the Texas Instruments ez430-RF2480ZigBee kit. Initially only synchronization was done, but it was verified the need to implement clock tuning. The devices clocks can be synchronized and tuned, even in heavily limited resources devices.

Keywords: Time synchronization; Limited resource wireless nodes; IEEE 1588; PTP; IEC 6158;

1. Introduction

With the deployment of smart grid and IoT technologies, the use of wireless sensor networks is becoming more widespread, almost ubiquous. However, to enable the collection of essential information in a distributed system, it is necessary to establish cause and effect relationships between different events that occur at different points and instants from the data sent by the sensors. Therefore, there is a need to synchronize all sensors to the same time reference, regardless of the network topology. To enable time synchronization of devices on networks, some protocols have been created, such as Daytime protocol [1], Time protocol [2], Network Time Protocol (NTP) [3], and Precision Time Protocol (PTP) [4].

In NTP protocol, the client can work with multiple servers and estimates the average time between them and consider the average to be the most likely correct time. In the PTP protocol, each client has only one time reference server at a time, which may change according to algorithms that periodically analyze which is the best server to obtain the time, and the synchronization is done as accurately as possible for the reference server clock. The PTP further specifies the clock frequency adjustment of the devices that implement it when various measurements can be compared and a drift rate is detected. Thus, in addition to

synchronization, it is also possible to tune the clocks of the devices.

The purpose of these protocols is to distribute a single time reference, despite uncertainties in the propagation time of messages on the network. Therefore, the time reference distribution process consists of several steps, starting with the formation of the network itself, including the creation of logical gates that will be used by the deployed protocol, decisions about the master / slave topology, estimation of the average delay time between the two devices, etc. In this context, it is important to highlight the popularity of the PTP protocol [4] specified in the IEEE 1588 standard.

In wireless sensor networking, there is a serious constraint on the amount and quality of resources available for time synchronization, as many wireless sensor nodes are based on low-cost microcontrollers. In particular, such nodes operate from batteries and their energy consumption is severely restricted. In addition to these restrictions, the propagation time between two nodes of a wireless sensor network can be variable, causing extra difficulty in synchronizing their clocks. For these resons, the task of clock synchronization in wireless network has been a focus of research lately.

PTP was used in [5] in order to synchronize clocks on energy meters using the ETE mechanism, but using an RTC on each node. A latency detection algorithm has also been added to its protocol implementation, specific to ZigBee networks. This implementation involves additional hardware and consequently increases the cost of components and power consumption.

An improvement to account for asymmetries in network path delays is proposed in [6]. After an initial synchronization, the authors propose a second round of messages, similar to the first, to detect and correct asymmetries in message propagation time. The technique is intended for application on 3G mobile networks where synchronized devices have many features. Simulations are presented, but there is no test information on real devices.

A bridge to extend the time distribution between ZigBee and Ethernet networks is implemented in [7]. The time reference can be on the ZigBee network or wired network. However, for the execution of the algorithm, the authors use a 32-bit RISC processor in the network nodes, with consequent cost and complexity increase.

In [8], a technique is presented to compensate for the effect of different clock counting rhythms to be synchronized. However, the algorithm is not compatible with the specification of IEC 61588: 2009 / IEEE 1588-2008.

A hybrid system, implemented by in [9], performs GPS time acquisition and PTP distribution to produce synchrophasor measurements in a power distribution system. However, the implementation was done in the laboratory with experimental equipment. There is no report of the difficulties of an application on wireless networks. In addition, a GPS clock receiver may add a substantial increase in costs.

In [10], it was proposed an improved synchronization algorithm to compensate for the error resulting from the dynamic data rate of wireless communication systems and thus improve the accuracy of distributed time. Simulations are done to verify the improvement in time synchronization, but the work does not present tests with real hardware.

This paper presents an experimental implementation of the time distribution algorithm, compatible with IEEE 1588 standard, tested on available and accessible hardware, with a functional application (measuring sensor values and sending information). In addition, for very cost-sensitive applications, the focus was on devices with severe memory and processing capacity limitations. Results obtained from a experimental working Zigbee network are reported. The source code for the new software may be obtained in [13].

This article is organized as follows: section 2 describes the main features of IEEE standard 1588 as well as the necessary data structures used in its implementation, section 3 contains the description of the implementation of this standard in low cost limited resources devices and section 4 informs about the experimental resuts obtained in a Zigbee network.

2. The time distribution protocol

IEC 61588 is the publication of IEEE 1588 by IEC [4]. This standard defines PTP, which enables precise synchronization between device clocks that have a communication channel with each other. The default behavior allows the synchronization of devices joining networks in an ad hoc manner without the need for reconfiguration with a primary reference clock, called the grandmaster clock, even if the devices present different accuracy capabilities.

The time allocation protocol reported in this article has been implemented according to IEC61588: 2009 / IEEE 1588-2008, following its rules as much as possible, to create a PTP watch with synchronization and tuning support, but yet resilient enough to support implementations on devices with severe memory and processing power limitations. Although the purpose of this paper is not to detail this standard, understanding some related concepts is important for comprehending the algorithms and the decisions made during its implementation.

2.1 Operation modes of the protocol

The standard allows the choice of two modes of operation:

- Delay request-response mechanism - messages are exchanged between two devices and may pass through other devices in the path that the packets follow. Also called end-to-end and abbreviated below as ETE.

- Peer delay mechanism - messages are exchanged between two devices directly without any intermediate routing except for pear-to-pear transparent clocks. Also called peer-to-peer, abbreviated below as PTP.

Section 6.5.1 of the standard states that there are five types of PTP devices:

- Ordinary Clock – It has only one PTP port;
- Border Clock – It may have multiple PTP ports;
- Transparent ETE Clock – It forwards ETE engine messages transparently;
- Transparent PTP Clock – It forwards PTP engine messages transparently;
- Manager Node – A device used to manage the application of the protocol.

Each ordinary and border clock port can be configured to operate on both ETE and PTP. The ports of an ETE transparent clock operate independently of the mechanism and those of the PTP transparent clock use PTP only. The two mechanisms (ETE and PTP) do not work on the same communication path, and PTP is restricted to topologies where each PTP port communicates only with another equivalent port as per section 6.5.1 of the standard.

PTP does not depend on the master-slave hierarchy to read the delay, unlike ETE, in which the master requests synchronization. Time distribution, however, must always occur from master to slave in both cases. PTP enables faster reconfiguration when the network changes its topology, as each port has already stored the communication delays with each other port on other devices. In addition, it uses two to three messages for delay determination, while the ETE uses three to four messages. On the other hand, ETE allows the propagation of time with less message exchange, since the master requests, in each active port, that the slave of the respective port checks the delay to it, while in PTP each port checks the delay in the communication with its partner. Therefore, both sides check the delay. Time propagation occurs in the same way for both protocols, according to section 12 of the standard.

Another feature to choose from is whether the watch will be single step or two step. Single step clocks can perform the protocol with fewer messages exchanged, but with additional processing while sending some messages, as described below.

When the clock is single-step, the message whose timeout or residence time stamp should be sent to other device is placed in the message itself, but the time stamp should be read very close to the instant the message leaves the sender. The standard determines reading at the moment when the message was sent and modifying the proper stamp field and recalculating any field dependent on the packet content. The message exchange for delay measurement ETE is shown in Fig. 1 (a).

When the clock is two-step, the time stamp or residence time of a given message is sent in another message, so as to avoid the processing and recalculation of content dependent fields during the transmission of the message. Message exchange for PTP delay measurement is shown in Fig. 1 (b).

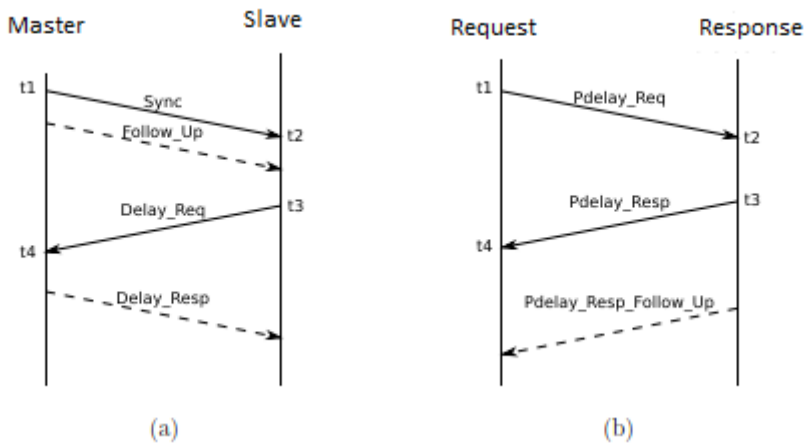

Figure 1. Message exchange for delay measurement in ETE (a), and PTP (b) mechanisms [4]

Considering the ZigBee network to be an ad hoc mesh network, the faster reconfiguration feature of the PTP engine may be useful. This was the mechanism chosen, in contrast to the more frequent choice of ETE that was studied, for example, in the studies in [5], [7] and [8]. Since it is possible to add and remove devices to configured network, which allows for the addition of more devices and to extend the physical range of the network, it is interesting that all devices can play the role of border clock to distribute time. Section 6.5.3 of the standard states that border clocks are typically used as network elements with no associated application, but this increases the required number of devices. This restriction was not considered in this paper.

In the implementation reported here, the time between the start of sending the message and the moment it should be modified is negligible in comparison to the step of the clocks to be adjusted, which are controlled by the software itself. Thus, the model chosen was the one-step clock, but the time stamp was made at the beginning of the assembly of the packet to be sent, thus reducing the complexity of having an additional message, the use of memory to record the information until they are sent, the number of messages transmitted and consequently the power consumption. The resulting error is negligible.

Some standard specifications have not been met as they require more RAM than available or in some cases the implemented algorithm was already prepared for different processing purpose.

2.2 Data structures for handling the information

The main data structures used by PTP clocks are described in chapter 8 of the standard:

- defaultDS - Stores features of the device's local clock;
- currentDS - Stores connection information with the master device;
- parentDS - Stores master device information;
- timePropertiesDS - Stores information about the local clock time of the device;
- portDS - Stores information about the device's PTP ports.

The standard further defines in section 9.3.2.4, for use with the Best Master Clock (BMC) algorithm, a foreignMasterDS data set for each port, which stores announcement message information from devices that can operate as a master on that port. According to section 9.3.2.4.5 of the standard, the minimum capacity of this data set should be five registers, however, due to the limited available memory and the implemented model, only one register will be stored for each port in the present algorithm.

In addition to the data sets defined in the standard, the implementation reported in this paper has the following data sets:

- announceDS - Stores information from announcement messages received by the device;
- PTPDelayReqDS - Stores response requests to determine the delay to the peer node while the information is not used yet.

The format of packets sent over the network is defined in section 13 of the standard. Relevant packets for the developed algorithm are:

- Announce (announcement) - Must be broadcast to all ports in the master state;
- Pdelay_Req (peer delay request) - Sent to request response to determine delay to peer device.
- Pdelay_Resp (peer delay reply) - Sent as reply to a message Pdelay Req.

The other messages are not relevant to the choices made for implementation or not essential to protocol operation. So they were not included in the program but can be included in future releases if they do not interfere with operation and fall within the limits of memory and processing of the processor to be used.

The standard states in section 5.4 that packages should be assembled so that data structures items are sent in the same order as they are defined (first item closest to the beginning of the package) and arrays or vectors with the smallest elements index first. When an octet has more than one field, the order of the fields must be preserved, and numeric fields of multiple octets must have the most significant octets sent first followed by the least significant ones. All of these requirements are met except for the last one, which determines a big endian format for data transmission. In general, in microcontrollers, numeric values are handled and stored in memory in little-endian format. Conversion between formats can be done, but requires processing time and stack or memory usage. Because resource utilization is being severely minimized, data will be kept in little-endian format. The conversion can be implemented by modifying the "PTP Utilities" unit described in section 3.3.

2.3 Programming structures

According to the choices made to implement PTP, the PTP ports will all be in PTP mode, which limits each port to communicating with only one port from a single other device. Since the network layer of the communication protocol already maintains an indexed list of peer devices to which it can communicate, this list will be used to limit the input and output of data from each PTP port to a single peer. Messages are sent only to addresses in the list and, when received, are passed to PTP code for processing only if the source (address and port) is in the list.

According to section 6.2 item c) of the standard, PTP was designed assuming multicast messaging model, but supports unicast messaging with the condition to preserve protocol behavior. Messages that are not multicast should be replicated as unicast messages for each port. Also, if there is no possibility of multicast messages, the discovery of the topology, usually made by means of announcement messages, should be performed in another way, an action already performed by the network layer.

Anyway, announcement messages are sent in broadcast mode: a single message is sent through all ports, consuming less power than sending multiple messages and making the program simpler. This is a violation of the standard, which states that only ports in the master state send announcement messages. In addition, it generates a problem because the outgoing port number is recorded in the header of the sent message. Since recipients do not need the port number to determine the topology, the output port number is set to 0xFFFF and, when this port number is received, the message is forwarded to PTP processing only if coming from the master. This is not described in the standard and is a specific case of this implementation.

Information from announcement messages that are not from the master clock is stored in external master records. Each port must have an external master data set according to section 9.3.2.4.1 of the standard, and section 9.3.2.4.5 requires a minimum capacity of five registers. However, storing five records per port would consume a lot of memory, and under the control of the network unit, each PTP port only receives PTP messages from a single device, even the announcement ones. Therefore, each port will have only one external master record, because even if considering a larger number as per the standard, the implementation would not use the other records. External master records are used in the BMC (Best Master Clock) algorithm.

In the BMC algorithm specified in section 9.3 of the standard, each port can determine the best master based on its external master records. In this implementation, this procedure is not necessary, since each port has only one record. Later, the best of all registered masters on all ports will be selected. Finally, the state decision algorithm represented in figure 2 recommends the state for each port of the device considering its internal clock.

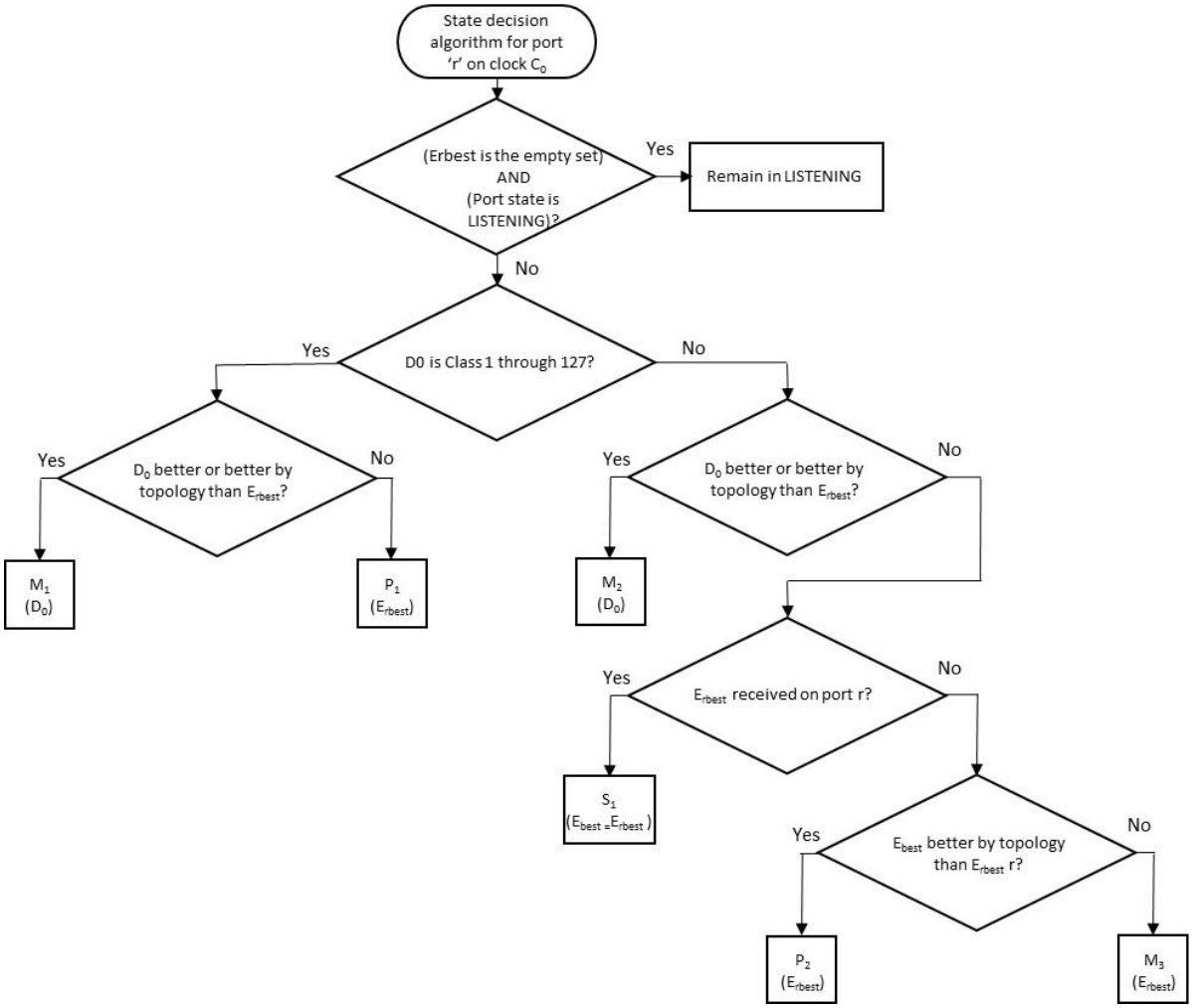

Figure 2: State decision algorithm [4]

The possible recommended states for each port are:

M1 - Internal clock from class 1 to 127 as network master;

M2 - Class 128 or greater internal clock as network master;

M3 - Master mode port, but not grandmaster;

S1 - Port in slave mode;

P1 - Internal clock from class 1 to 127 in passive mode, because it is not the grandmaster of the network or to interrupt a loop on the network;

P2 - Class 128 or greater internal clock in passive mode, to interrupt a network loop;

The recommended state for each port is defined according to the algorithm result: results M1, M2 and M3 are indications to master mode recommendation; state S1 for slave mode recommendation and states P1 and P2 for passive mode.

This algorithm is used in BMC, which in turn is used in the status decision event, which in fact defines states of the PTP ports of the device. The logic is represented in figure 3. It is important to note that the BMC algorithm is also represented. In the reported implementation, there is no step for each port to select

its best master (Erbest), because each one has only one external master record.

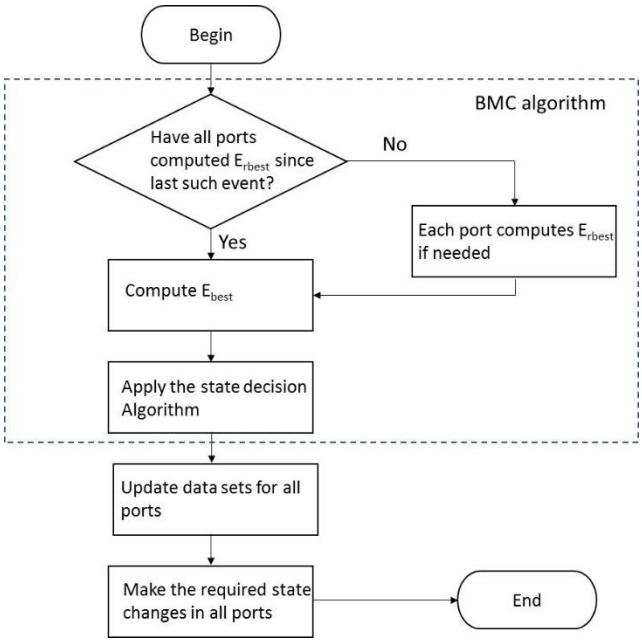

Figure 3: State decision event logic [4]

According to section 6.6.2.1 of the standard, each port performs an implementation of the protocol state machine independent of the others, represented in figure 4. Part of the state changes is executed within the state decision event and another part in time expired events for receiving messages, communication packet failures or network topology change.

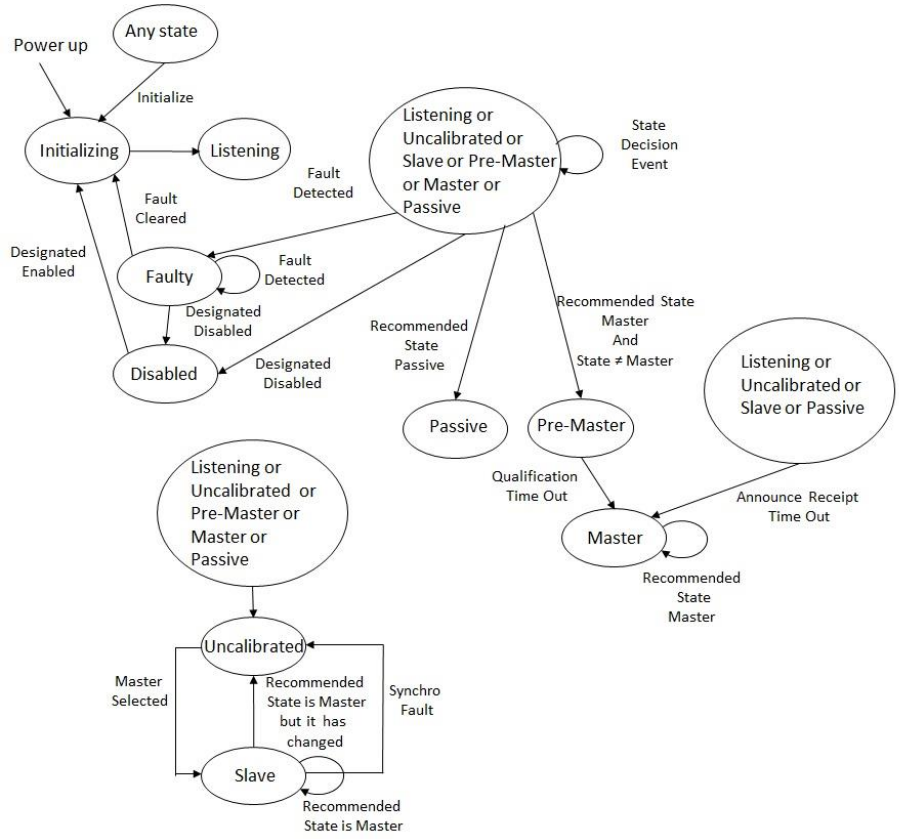

Figure 4: State machine for a full implementation [4]

In this implementation, only the states necessary for the operation of the protocol are used. “Fault" and "Disabled" states are not used, but are defined in the programming structures in order to allow a future development.

3. The time distribution protocol implementation

The division of the implementation of the time distribution protocol into units was performed by means of functions in the program. The defined units are:
PTP - Interface unit that the host program uses to access the time distribution protocol;
PTP types - Unit that declares the types of variables used by the protocol;
PTP Settings - Values that can be modified before the final build to change characteristics or protocol behavior;
PTP constants - Predefined values used;
PTP data sets - Definition of the data sets used by the protocol;
PTP Clock - Contains the definition of the (programming) object that stores the data sets that control the protocol and startup procedure (despite the title, it has no time count);
PTP Messages - It sends messages to the network and handle incoming messages relevant to the protocol;
PTP Utilities - Tools used by the protocol that are not part of other drives.

3.1 PTP Unit implementation

The initial protocol implementation planning considered only the PTP unit as the interface between the host program and the protocol, that is, only this unit would have access of host program units and vice versa. Protocol units using external calls would use PTP unit calls. However, to reduce stack usage, these intermediate calls have been eliminated and external functions are called directly from the units that use them. Thus the protocol units have calls to other program units.

The PTP Unit is used by the host program to access the time distribution protocol. It also contains variables used by other protocol units with external access. There are two functions that should be called periodically, the regular processing loop, with calls every 1 second, and the peer delay request check for which, the higher the call frequency, more accurate the protocol is. However to reduce power consumption, the frequency of calls for the latter may be reduced with the consequence of reducing the accuracy of the protocol if the oscillator frequencies of the synchronizing devices are not equal and drift from each other.

Periodic processing (function ptp_process) performs the tasks shown in figure 5. When in the diagram a value is assigned to the result, it indicates that the function returns this value without further processing in the current call. The return value is the time, in milliseconds, after which the function must be called again. When the protocol is started, it is not running, and all ports are put into the “booting” state. In the processing function the first task is to verify that the protocol is already running. If not, the state is checked. If any of them are still in the “initializing” state, the function only returns the value 1000; otherwise, the protocol is executed for subsequent calls and processing continues. The definition of running or not has been created so that there is no need to check the state of all ports each time the function is called. With the protocol

running, the function continues its regular processing

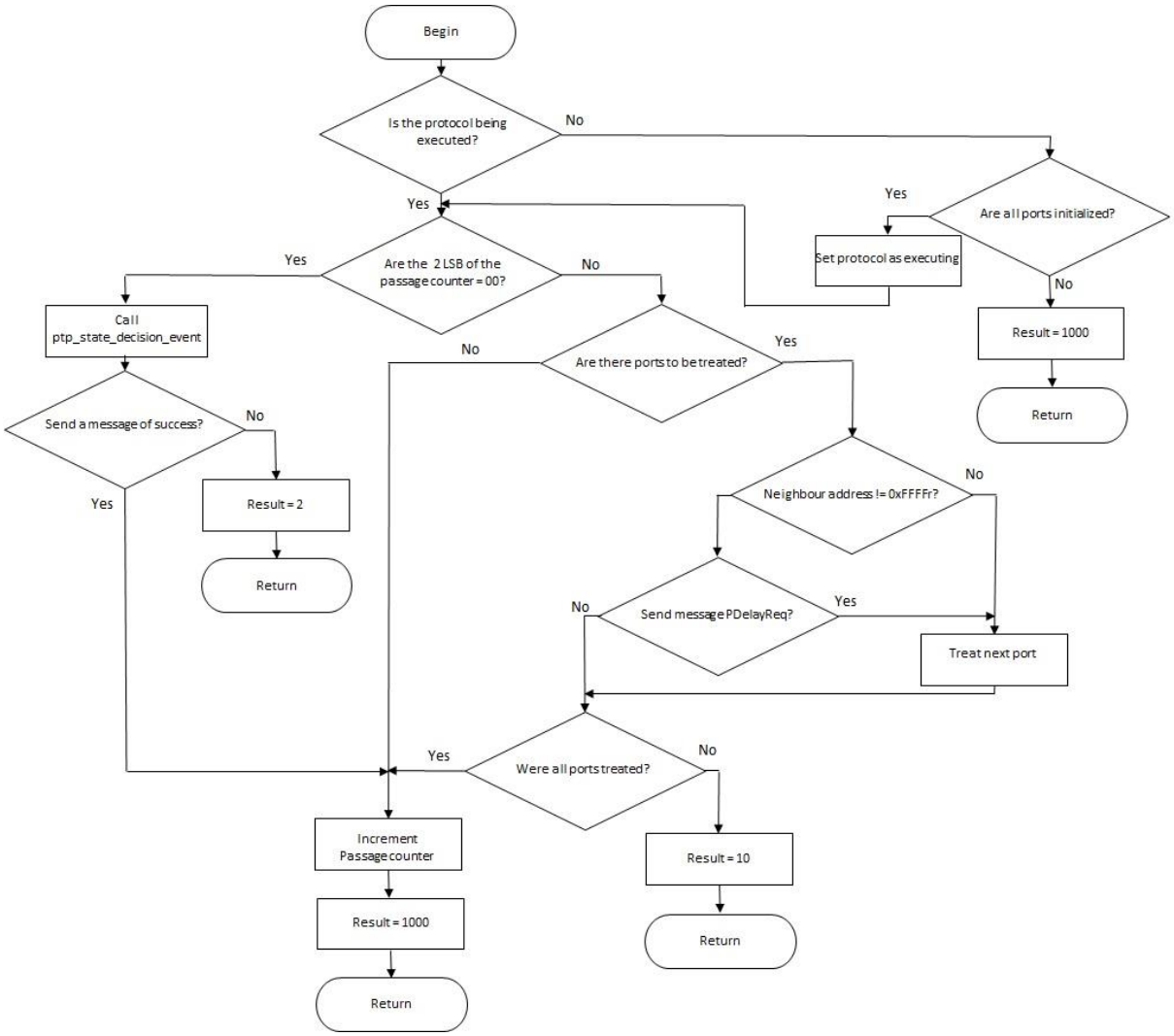

Figure 5: Flowchart of ptp_process unit

A ticket counter is maintained to keep track of function execution and allows periodic execution of code runs based on the number of calls, with additional functionality: it is incremented only when there is regular return of the function. This way, if the desired action cannot be performed, the next execution may repeat the attempt.

3.2 PTP Messages Unit implementation

This unit prepares and sends messages over the network and handles incoming protocol messages. The header format is common among all PTP messages, and is assembled by the ptp_MsgHeaderSnd function, which receives in its parameters the header content to be sent.

There are two classes of messages: event and general messages. When the device processes an event message, it must create a time stamp upon receipt or sending of the message. General messages do not have this need. The standard specifies the existence of two separate interfaces for receiving messages, one for each type of treatment, but gives the freedom to have only one interface that always makes the time stamp,

and uses it only when necessary.

Each message has a function for its assembly and sending. There is also a procedure for treatment when receiving messages. Methods for all message types exist in the unit but only those of the messages used in this implementation have been implemented, the others do not take any action.

The messages used are defined in section 7.3.3 of the standard as:

- PdelayReq - Event message, delay request to peer, PTP engine only;
- PdelayResp - Event message, delay request message response to peer, PTP engine only. Section 7.3.3.1 d) of the standard;
- Announce - General message, issues presence and feature announcements to other devices. Section 7.3.3.2 a) of the standard;

Message processing is defined in section 9.5 of the standard. It determines when messages are sent and the proper treatment when they are received.

Announcement messages should be sent only through ports in the master state periodically. In this implementation, to reduce processing and power consumption, the message is sent as broadcast and without a defined port. It is a violation of the standard because the message is not suppressed for ports in other states but it saves considerable resources in the context of this implementation.

Upon receiving an announcement message the device performs the flowchart procedure shown in figure 6.

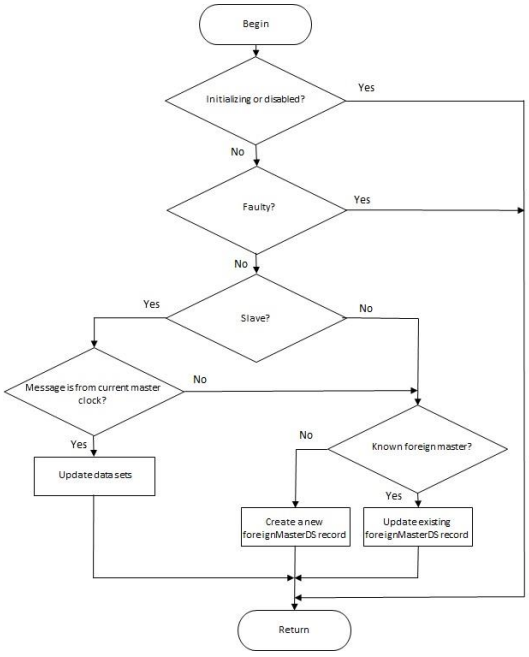

Figure 6: Treatment of announcement messages logic.

In this implementation there is no fault condition handling, so there is no such test. In addition to this

difference, each port stores information from a single external master. Thus, the record will be created and thereafter only updated, unless the network topology changes, in which case the record will be marked empty because it has expired. Later it will be populated if another external device is assigned to the port.

The initial PdelayReq message can be transmitted as needed, and subsequent messages should be transmitted at intervals no less than the minimum obtained from the logMinPdelayReqInterval value of the PTP_portDS member in the port-specific register.

When the device receives a PdelayReq message, it should respond with PdelayResp as soon as possible so that interference from the clock frequency difference is reduced in channel delay measurement. In this implementation, it is not possible to send a message through the SPI interface of the microcontroller if there is a message being handled. Since the end of handling when receiving a message is the return of receive interrupt, it is not possible to respond to the request message immediately. Thus, the request is placed in a queue, in the PTP PTPDelayReq dataset that is checked once every millisecond, and if the queue has elements, a response is sent.

Upon receiving the PdelayResp message, the device calculates the channel delay and stores it for processing.

3.3 PTP Utilities implementation

This unit contains procedures for executing the state decision event (ptp_state_decision_event), BMC algorithm (ptp_bmc), port data set update (ptp_portUpdateDS), and port state update (ptp_portUpdateStatus), and functions for comparison between external masters (ptp_dataSetComparisonEE), between external master and local clock (ptp_dataSetComparisonDE) and enter PTP clock identification (ptp_clockIdentityCompare). All these methods are used in determining the master / slave structure topology in the time distribution.

This unit also includes the method for storing announcement messages (ptp_storeForeignMaster) and functions for sending and receiving data over the network.

Sending a message over the network begins with calling the ptp_startFrame function, followed by zero or more calls to data sending functions, and ending with calling the ptp_endFrame procedure. There is a function for sending data regardless of the specific type, ptp_snd and six others, one for each integer numeric type, prefixed with ptp_snd and ending with the data type name.

Receiving a message has no start and end methods, but there are functions for reading message data, as well as sending. There is one for receiving data regardless of the specific type, ptp_rcvOctets and six others, one for each. integer numeric type, which has prefix ptp rcv and ends with the data type name.

The functions of sending / receiving data allow modifying the program to the conversion of data formatting into little-endian or big-endian messages, such as it was mentioned in section 2.2. Nevertheless, as

explained, the implementation for low resources devices uses the native mode of microcontrollers, little-endian format.

3.3 Challenges of PTP implementation on limited resource microcontrollers

The challenges faced were all related to the amount of available memory, causing stack overflows and memory address misalignment .

The first, most obvious action to decrease memory usage was to reduce the buffer size from the serial interface because its use is restricted to short messages of clock management and time stamp management.

Many of the members of structures and datasets are pointers. This was done to decrease the amount of redundant data in memory. If the same data is used in two different points of the algorithm , one may contain a reference to the data itself and the other one pointer to it, so the value is stored only once. However, the pointer itself takes up memory space. For the microcontroller used in this implementation, each pointer occupies 16 bits (two octets), so it is useful to decrease memory usage when the referenced information occupies more than 16 bits. This was particularly useful for the representation of of external and internal clocks that need 64-bit data containers and are used multiple times in several data instances.

When a method is called, the parameters are passed through the microcontroller registers, but the values that previously existed need to be saved to be recovered after the method is executed, and the location used to store the previous contents is the stack. To reduce stack usage in this case, all method calls with parameters that take up more than 16 bits use pointers that indicate the required values.

The microcontroller used does not have multiplication and division hardware and these operations are done by software pre-programmed in the C compiler. In program debug mode, it was noted that multiplications were making too much use of the stack. Further analysis revealed the native C compiler performed the conversion 64-bit integers to floating point numbers, multiplied these numbers and finnally did a conversion of the product back to 64-bit integer format. The solution in this case was to program a custom integer multiplication routine.

Some data format conversions using pointers presented a bug. This was caused by the fact that all types handled by the compiler, except those occupying 8 bits, must have an address of even value in memory. As the smallest division of the buffer is an octet, the messages could be misaligned, or contain misaligned fields. To solve this alignment problem, data format conversion operations are performed one octet at a time.

4. Experimental results

4.1 Test environment

In order to implement and test the time distribution algorithm developed in this article, a physical network

was needed. Hence, the concepts and algorithms presented so far were applied to a Zigbee wireless sensor network.

The IEEE 802.15.4 standard defines a wireless protocol for use by low power, low power consumption and limited resources devices. According to ZigBee Alliance [14], the ZigBee specification improves the IEEE 802.15.4 standard by establishing criteria so that devices provided by different manufacturers can communicate among them, based on profiles for specific applications.

For this work, the ez430-RF2480 kit from Texas Instruments [11] was used. This kit includes three target boards that connect to a computer via the USB port. The boards, with a surface of approximately 2cm x 3cm, are designed around a MSP430F2274TM microcontroller [12] that will be the programmed device, a light sensor, a push button and a CC2480TM wireless transceiver chip. The MSP430F2274TM microcontroller has 32KB + 256B flash memory and 1KB RAM memory, as well as peripherals such as communication ports and analog-to-digital converter (ADC). Each board is already preprogrammed with the ZASA (ZigBee Accelerator Sample) application, which displays network status via Light Emmiting Diode (LED's) and, along with the eZ430-RF2480 Sensor Monitor software that runs on a personal computer, displays the readings of the temperature sensors and supply voltage of each board in the network.

However, both the ZASA and the eZ430-RF2480 Sensor Monitor feature source code difficult to understand, either by having a deep and complex chain of calls or by simple lack of documentation, making it very difficult to change the data that is sent and displayed on your computer. This is why it was decided to create other software for the microcontroller and for the computer. The purpose of developing new software with similar functionality to the existing ones is to make easy to perform modifications in their fatures as well as to facilitate the understanding of the algorithms involved in this article. The source code for the new software may be obtained in [13].

The new microcontroller software, called SZS, has the function of establishing and controlling the network in order to send the data to the computer. And the computer software performs the task of displaying the data and sending commands to the connected devices.

Table 1 depicts the feature differences between the original ZASA software by Texas Instruments and the developed SZS software.

Table 1. Feature comparison between the ZASA and SZS programs

Software	ZASA	SZS
Ad Hoc node incorporation	Yes	Yes
Serial port monitoring	Only the network coordinator	Any device
Power Level reading	Yes, except by the coordinator	Yes
Temperature reading	Yes, except by the coordinator	Yes
Ligth sensor reading	No	Yes
Time stamp at each reading	No	Yes
Time stamp internal clock set up	No	By serial port command

The utilization of resources by both software is compared in Table 2. It can be seen that the amount of program memory used by the SZS software is more than double than the amount used by the ZASA. This is due to the fact that programming tweaks had to be used to properly manage the stack, integer multiplication, etc.

Table 2. Memory usage of the ZASA and SZS programs

		ZASA	SZS	Difference
Memory (bytes)	Total RAM	1024		
	Used RAM	657	607	-50 (-7.61%)
	Free RAM	367	417	50 (+13.62%)
	Total Flash	32734		
	Used Flash	4389	10895	+6506 (+148.23%)
	Free Flash	28345	21839	-6506 (-22.95%)

For the synchronization and tuning tests, a laboratory arrangement was set up for measuring and recording the delays and adjustments that occurred during program operation both in the devices and the computer. The arrangement assembly scheme is shown in figure **7**. In this setup, there are three devices, identified by the last four digits of their physical addresses connected to a ZigBee network. The clock of the 3D9B device has been synchronized with the computer clock with periodic corrections as needed, and the other devices use the time distribution protocol to sync with the 3D9B device over the ZigBee network.

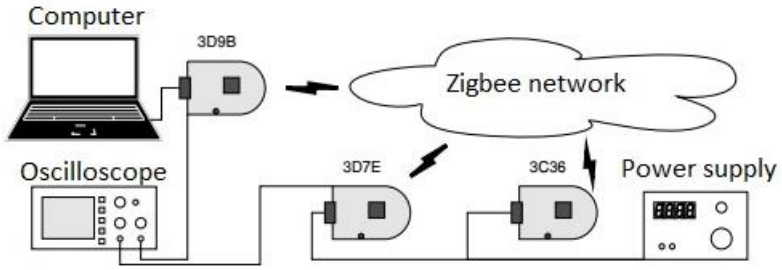

Figure **7:** Test and measurement environment

The files with the values of the sensors and adjustments of time from reports sent by network devices were recorded at the computer. The oscilloscope was used to check and record the time delay between the watches of the 3D7E and 3D9E devices by means of reference 0.5 Hz square waves generated by them based on their respective internal clocks after synchronization.

For the experimental tests, the three devices were connected. Device 3C36 was initially synchronized with the time reference provided by the computer and then used as clock master for the overall wireless network.

Then, device 3D9B was disconnected from the Zigbee network and then reconnected. This experiment was executed during approximately two hours. Figure 8 shows the clock adjustments perfomed during this time. It can be seen in this figure that the algorithm worked well even after long time disconnections caused by the nature of the Zigbee network, during which the devices clock drift from each other but were quickly synchronized after reconnection.

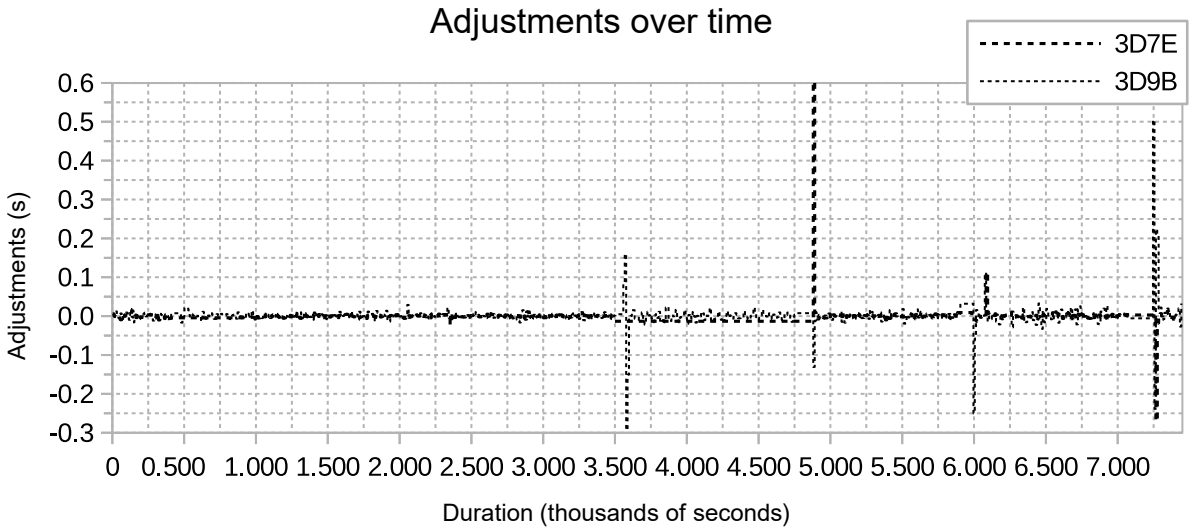

Figure 8. Clock adjustments performed in a two hours interval

Table 3 shows the statistical characterization of the measured channel delay and clock adjustments performed by devices 3D7E and 3D9B.

Table 3: Statistical characterization of measured channel delay and clock adjustments

Device	Parameter	Channel Delay (ms)	Clock Adjustments (ms)
3D7E	Average	17.26	3.14
	Median	17.53	0.49
	Standard Deviation	0.75	80.38
3D9B	Average	18.46	0.91
	Median	17.37	0.34
	Standard Deviation	7.18	29.28

This statistical data is further shown as histograms as shown in Figure 9.

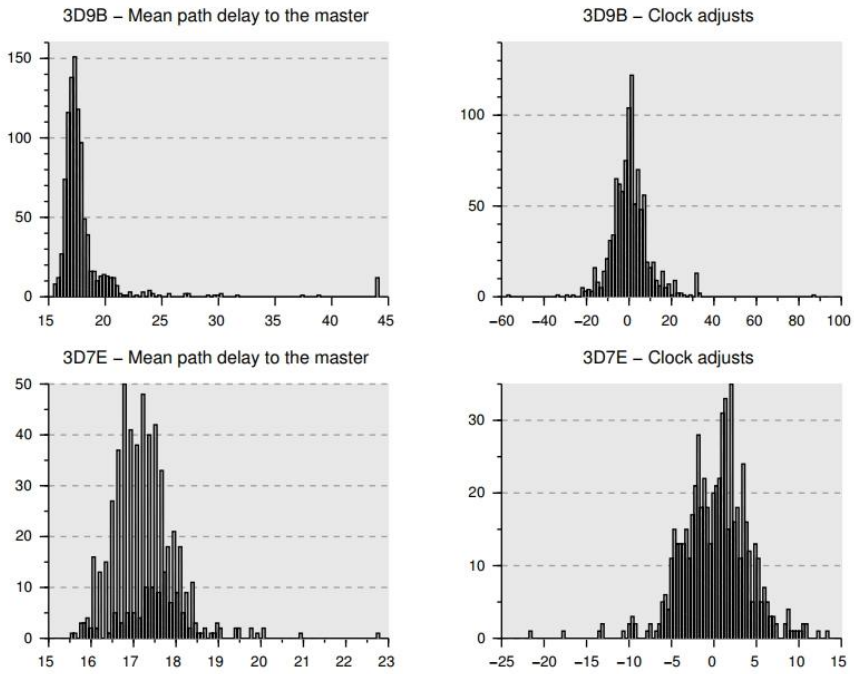

Figure 9. Histograms of the mean path delay and clock adjustments in the experimental tests

The experimental data indicates the good performance of the implemented algorithms, considering the uncertainty of channel delays in the Zigbee network. The biggest adjustments were performed after long disconnections in both devices 3D7E and 3D9B, corresponding to time differences in reference to the master clock of 0.142% and 0.294%, respectively. The adjustments depended only upon the quality of the connection of each device to the master clock.

5. Conclusions

It was possible to develop a time synchronization protocol resilient enough to be embeded in severe resource limitated wireless devices. The implemented protocol meets the IEC 61588 standard almost completely, with the exception of sending rules, byte order and message storage, and the ability to store external log records for each port. These limitations are due to restrictions on the available features of low-cost wireless devices. Clearly, these features were traded by in order to make possible to implement the refered standard in low cost limited devices.

Experimental data was obtained by implementing the developed algorithms in a Zigbee network employing readily available commercial devices.

The time synchronization protocol worked properly, providing the time distribution, and can be used on another devices. It was even possible to detect discontinuities in the clock of the master device when this device was synchronized to an external device. The implementation was resilient and accurate even under the Zigbee non deterministic behavior. The same source code can be adapted to use on other devices or refined, being available in [13].

6. Acknowledgement

The authors want to thank the Fundo Mackenzie de Pesquisa (MackPesquisa) for their support during the realization of this study.

7. References

[1] J. Postel, Request for Comments: 867 - Daytime Protocol Network Working Group, 1983. (http://tools.ietf.org/html/rfc867)

[2] J. Postel and K. Harrenstien, Request for Comments: 868 -Time Protocol, Network Working Group, 1983, (http://tools.ietf.org/html/rfc868)

[3] D. Mills,et al., Request for Comments: 5905 – Network Time Protocol Version4: Protocol and Algorithms Specification, Internet (https://tools.ietf.org/html/rfc5905)

[4] IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, IEC 61588:2009(E), pages C1–274, Feb 2009.

[5] F.C. Luiz, Sistema de Calibração de Energia sem Fio usando Zigbee, Ph.D. Thesis, UFRJ, Rio de Janeiro, 2012, (www.gta.ufrj.br/~cunha/Docs/Dissertacao_Fernando_da_Cunha_Luiz_Revisao_1.pdf)

[6] S. Lv, Y. Lu, and Y. Ji, An Enhanced IEEE 1588 Time Synchronization for Asymmetric Communication Link in Packet Transport Network, IEEE Communications Letters, IEEE, v. 14, n. 8, 2010, pp. 764-766.

[7] H. Cho, Y. Jin, J. Heo and Y. Baek, Implementation of a PTP Bridge to Extend IEEE 1588 to Zigbee Networks, IEEE 10th International Conference on. Computer and Information Technology (CIT), 2010, pp. 2604-2611.

[8] Z. Du, Y. Lu, and Y. Ji, An Enhanced End-to-end Transparent Clock Mechanism with a Fixed Delay Ratio, IEEE Communications Letters, IEEE, v. 15, n. 8, 2011, pp. 872-874.

[9] M. Lixia, N. Locci, C. Muscas and S. Sulis, Synchrophasors Measurement in a GPS- IEEE 1588 Hybrid System, European Transactions on Electrical Power, Wiley Online Library, v. 21, n. 4, 2011, pp.1509-1520.

[10] S. Lee, S. Lee and C. Hong, An Accuracy Enhanced IEEE 1588 Synchronization Protocol for Dynamically Changing An Asymmetric Wireless Links. IEEE Communications Letters, IEEE, v. 16, n. 2, 2012, pp. 190-192.

[11] Texas Instruments, eZ430-RF2480 Demonstration Kit User´s Guide, available in

http://www.ti.com/lit/ug/swru151a/swru151a.pdf

[12] Texas Instruments, MSP430F22x2 and MSP430F22x4 data sheet, available in https://www.ti.com/lit/ds/symlink/msp430f2274.pdf

[13] http://www.biazi.eng.br/en/

[14] Zigbee Alliance, available in https://zigbeealliance.org/