Universidade de Aveiro
2013
Departamento de Eletrónica,
Telecomunicações e Informática

João Filipe Pereira dos Reis

Sincronização de Relógios em redes LoWPAN

Universidade de Aveiro
2013
Departamento de Eletrónica,
Telecomunicações e Informática

João Filipe Pereira dos Reis

Sincronização de Relógios em redes LoWPAN

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Eletrónica e Telecomunicações, realizada sob a orientação científica do Doutor Nuno Miguel Gonçalves Borges de Carvalho, Professor Catedrático do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro e do Doutor José Manuel Neto Vieira, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

O júri

Presidente	Prof. Doutor Paulo Bacelar Reis Pedreiras Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro
Vogais	Prof. Doutor Pedro Renato Tavares de Pinho Professor Adjunto da Área Departamental de Engenharia de Eletrónica e Telecomunicações e de Computadores do Instituto Superior de Engenharia de Lisboa
	Prof. Doutor Nuno Miguel Gonçalves Borges de Carvalho Professor Catedrático do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro
	Prof. Doutor José Manuel Neto Vieira Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

Agradecimentos

"Nada na vida se consegue sozinho e nenhuma conquista tem valor se não tivermos com quem a partilhar". Com este pensamento em mente, gostaria de agradecer:

Aos meus pais e irmãos, que são o meu porto seguro. Pessoas com quem eu sempre pude contar e que fizeram de mim uma pessoa correta e trabalhadora. Para com eles guardarei sempre um grande carinho.

Á minha namorada, por me acompanhar nos bons e maus momentos e por me aturar nas minhas alturas de *stress*. Sem ela, seria metade do que sou hoje e a luta pelo meu futuro seria um caminho muito solitário.

Aos meus amigos, antigos e novos, com quem partilhei diversos tipos de experiencias, com quem debati dos temas mais estúpidos aos tópicos mais cultos, com quem planeei dominar o mundo enquanto bebíamos umas cervejas. Eles são a razão por que levanto todos os dias. Sem eles, a vida seria uma grande mancha cinzenta, sem sentido de existir.

Ao professor Nuno Borges, com a sua incansável motivação e boa disposição. Uma pessoa que apesar de estar sempre ocupada com novos projetos, encontra sempre tempo para acompanhar o trabalho dos seus alunos. A oportunidade oferecida por ele para trabalhar no Instituto de Telecomunicações de Aveiro foi um dos grandes impulsionadores de todo o meu conhecimento adquirido na área dos microcontroladores e comunicações rádio. Por isto, fico-lhe eternamente agradecido.

Por fim a toda a máquina que é a Universidade de Aveiro e o Instituto de Telecomunicação. Desde os seguranças das portarias, ao pessoal da administração, professores e empregadas de limpeza. Todos eles são responsáveis por tornar os rapazes e raparigas que entram no primeiro ano em homens e mulheres com grandes conhecimentos intelectuais, capazes de sair para o mercado de trabalho com um grande leque de capacidades. A todos eles, agradeço a experiencia de vida que aqui vivi.

João Filipe Pereira dos Reis

Palavras-chave

IEEE 1588, Sincronização de Relógios, Sintonização de Relógios, LoWPAN, WSN, Comunicações Sem Fios.

Resumo

Neste momento vivemos numa era de crescimento tecnológico onde todos os dias são lançados novos produtos inovadores para o mercado que passado alguns meses podem já ser considerados ultrapassados.

A acompanhar este crescimento e ao mesmo tempo sendo o seu próprio impulsionador, temos as tecnologias de comunicação sem fios a aparecer com cada vez mais largura de banda, consumos reduzidos e disseminações em massa de pequenos dispositivos embebidos em equipamentos electrónicos.

Este forte crescimento possibilita que aplicações antes exclusivamente suportadas por tecnologias de comunicação com fios, possam migrar se certos requisitos forem preenchidos.

Um desses casos é o suporte da capacidade de manter várias unidades com os seus relógios sincronizados.

No desenvolvimento desta dissertação, foi solucionado este problema através da implementação de um protocolo de sincronização de relógios suportado por um algoritmo de sintonização, onde foram obtidos níveis de precisão inferiores a 200 nanossegundos usando períodos de sincronismo de 30 segundos.

Keywords

IEEE 1588, Clock Synchronization, Clock Syntonization, LoWPAN, WSN, Wireless Communications.

Abstract

Right now we live in an age of technological growth where every day are released new innovative products to the market which after a few months can already be considered outdated.

Accompanying this growth and at the same time being their own booster, we have the wireless communications technologies appearing with an increasing bandwidth, reduced power consumption and mass dissemination of small embedded devices on electronic equipment.

This strong growth makes it possible to applications previously only supported by wired communication technologies, to migrate if some requirements are fulfilled.

One of these cases is the support of the capability to maintain many units with their clocks synchronized.

In the development of this dissertation, this problem was solved with the implementation of a clock synchronization protocol supported by a clock syntonization algorithm, where it was obtained accuracy levels smaller than 200 nanoseconds while using synchronization periods of 30 seconds.

Índice

Índice ..... **i**

Índice de Figuras ..... **iii**

Índice de Acrónimos ..... **vi**

1 Introdução **1**

1.1 Motivação 3

1.2 Objetivos 3

1.3 Estrutura 4

2 Sincronismo de Relógios **7**

2.1 Problema 8

2.2 Soluções 9

2.2.1 Caracterização dos atrasos na troca de mensagens 11

2.2.2 Protocolos de sincronismo de relógios 12

3 Primeiro Protótipo **19**

3.1 Descrição dos Blocos do Sistema 21

3.1.1 Bloco RF 22

3.1.2 Bloco de Sincronismo 26

3.1.3 Bloco de Gestão de Eventos 30

3.1.4 Concorrência de acesso a recursos 35

3.2 Validação inicial do sistema 36

3.3 Sintonização de relógios 38

3.4 Validação final do sistema 41

3.4.1 Teste com *SyncPeriod* de 5 segundos 42

3.4.2 Teste com *SyncPeriod* de 30 segundos 42

3.4.3 Resultados Finais 43

3.5 Reconhecimento 43

4 Segundo Protótipo 45
4.1 Otimizações 46
4.1.1 Registo do tempo de receção e transmissão de mensagens 47
4.1.2 Execução e registo temporal de eventos críticos 48
4.1.3 Utilização de correções de sintonização finas (*SyntCorrValue*) 53
4.1.4 Cálculo do período de sintonização do relógio (*SyntPeriod*) 57
4.1.5 Cálculo do atraso do canal de comunicação (*PathDelay*) 65
4.1.6 Cálculo do desvio do relógio (*OffsetFromMaster*) 69
4.1.7 Cálculo do número de correções de sintonização (*NSyntCorr*) 72
4.2 Reconhecimento 77
5 Conclusões e Trabalho Futuro 79
Anexo A 83
Referências 87

Índice de Figuras

Figura 2.1 – Impacto do valor dos condensadores de carga do cristal 9
Figura 2.2 – Desvio do valor de relógio ao longo do tempo 9
Figura 2.3 – Sincronização por GPS....10
Figura 2.4 – Sincronização com rede dedicada....10
Figura 2.5 – Latências entre o tempo de transmissão e receção de mensagens 11
Figura 2.6 – Troca de mensagens do protocolo RBS 13
Figura 2.7 – Ilustração dos tempos t_{Delay} e t_{Offset}....14
Figura 2.8 – Troca de mensagens do protocolo TPSN....15
Figura 2.9 – Troca de mensagens do protocolo PTP 17
Figura 3.1 – Componentes da placa *NaPIS*....20
Figura 3.2 – Estruturação global dos blocos do sistema 21
Figura 3.3 – Estruturação das mensagens de rádio 22
Figura 3.4 – Validação do *ServiceID* da mensagem....23
Figura 3.5 – Estrutura das mensagens de sincronismo....27
Figura 3.6 – Estrutura do registo do valor de tempo....28
Figura 3.7 – Envio periódico das mensagens *Sync* e *FollowUp*....28
Figura 3.8 – Troca de mensagens de sincronismo para o cálculo de *PathDelay*....29
Figura 3.9 – Troca de mensagens de sincronismo para o cálculo de *OffsetFromMaster* 29
Figura 3.10 – Execução de eventos periódicos 31
Figura 3.11 – Fluxo de execução da interrupção do Bloco de Gestão de Eventos....32
Figura 3.12 – Fluxo da manutenção do relógio....33
Figura 3.13 – Fluxo da manutenção dos eventos 34
Figura 3.14 – Níveis de prioridade das interrupções 36
Figura 3.15 – Estrutura do ensaio....36
Figura 3.16 – Distribuição dos valores de *OffsetFromMaster*....37
Figura 3.17 – Valores de desvio de frequência entre os relógios....37
Figura 3.18 – Impacto da sintonização do relógio 38

Figura 3.19 – Fluxo de execução da interrupção do Bloco de Gestão de Eventos com sintonização 39
Figura 3.20 – Fluxo do cálculo do número de correções de sintonização 40
Figura 3.21 – Estrutura do ensaio 41
Figura 3.22 – Erro de sincronismo para um *SyncPeriod* de 5 segundos 42
Figura 3.23 – Distribuição do erro de sincronismo para um *SyncPeriod* de 5 segundos 42
Figura 3.24 – Erro de sincronismo para um *SyncPeriod* de 30 segundos 42
Figura 3.25 – Distribuição do erro de sincronismo para um *SyncPeriod* de 30 segundos 42
Figura 3.26 – Unidade *WAPoS* 43
Figura 4.1 – Localização dos sinalizadores associados à troca de mensagens 47
Figura 4.2 – Erro de sincronismo para um *SyncPeriod* de 1 segundo 49
Figura 4.3 – Distribuição do erro de sincronismo para um *SyncPeriod* de 1 segundo 49
Figura 4.4 – Erro de sincronismo para um *SyncPeriod* de 2 segundos 50
Figura 4.5 – Distribuição do erro de sincronismo para um *SyncPeriod* de 2 segundos 50
Figura 4.6 – Erro de sincronismo para um *SyncPeriod* de 5 segundos 50
Figura 4.7 – Distribuição do erro de sincronismo para um *SyncPeriod* de 5 segundos 50
Figura 4.8 – Erro de sincronismo para um *SyncPeriod* de 10 segundos 51
Figura 4.9 – Distribuição do erro de sincronismo para um *SyncPeriod* de 10 segundos 51
Figura 4.10 – Erro de sincronismo para um *SyncPeriod* de 30 segundos 51
Figura 4.11 – Distribuição do erro de sincronismo para um *SyncPeriod* de 30 segundos 51
Figura 4.12 – Erro de sincronismo para um *SyncPeriod* de 60 segundos 52
Figura 4.13 – Distribuição do erro de sincronismo para um *SyncPeriod* de 60 segundos 52
Figura 4.14 – Erro de sincronismo para um *SyntCorrValue* de 1000 nanossegundos 53
Figura 4.15 – Distribuição do erro de sincronismo para um *SyntCorrValue* de 1000 nanossegundos 54
Figura 4.16 – Erro de sincronismo para um *SyntCorrValue* de 500 nanossegundos 54
Figura 4.17 – Distribuição do erro de sincronismo para um *SyntCorrValue* de 500 nanossegundos 54
Figura 4.18 – Erro de sincronismo para um *SyntCorrValue* de 192 nanossegundos 54
Figura 4.19 – Distribuição do erro de sincronismo para um *SyntCorrValue* de 192 nanossegundos 55
Figura 4.20 – Erro de sincronismo para um *SyntCorrValue* de 77 nanossegundos 55
Figura 4.21 – Distribuição do erro de sincronismo para um *SyntCorrValue* de 77 nanossegundos 55
Figura 4.22 – Erro de sincronismo para um SyntCorrValue de 38 nanossegundos 55
Figura 4.23 – Distribuição do erro de sincronismo para um *SyntCorrValue* de 38 nanossegundos 56
Figura 4.24 – Distribuição no tempo das correções de sintonização 58
Figura 4.25 – Correção de sintonização teórica 58
Figura 4.26 – Correção de sintonização real 58
Figura 4.27 – Correção de sintonização fracionárias pelo primeiro método 58
Figura 4.28 – Correção de sintonização fracionárias pelo segundo método 59
Figura 4.29 – Correção de sintonização fracionárias pelo segundo método (2º caso) 59
Figura 4.30 – Correção de sintonização fracionárias pelo terceiro método 60
Figura 4.31 – Erro de sincronismo para um *SyncPeriod* de 1 segundo 61
Figura 4.32 – Distribuição do erro de sincronismo para um *SyncPeriod* de 1 segundo 61
Figura 4.33 – Erro de sincronismo para um *SyncPeriod* de 2 segundos 61
Figura 4.34 – Distribuição do erro de sincronismo para um *SyncPeriod* de 2 segundos 61

Figura 4.35 – Erro de sincronismo para um *SyncPeriod* de 5 segundos 62
Figura 4.36 – Distribuição do erro de sincronismo para um *SyncPeriod* de 5 segundos 62
Figura 4.37 – Erro de sincronismo para um *SyncPeriod* de 10 segundos 62
Figura 4.38 – Distribuição do erro de sincronismo para um *SyncPeriod* de 10 segundos 62
Figura 4.39 – Erro de sincronismo para um *SyncPeriod* de 30 segundos 63
Figura 4.40 – Distribuição do erro de sincronismo para um *SyncPeriod* de 30 segundos 63
Figura 4.41 – Erro de sincronismo para um *SyncPeriod* de 60 segundos 63
Figura 4.42 – Distribuição do erro de sincronismo para um *SyncPeriod* de 60 segundos 63
Figura 4.43 – Troca de mensagens de sincronismo para o cálculo de *PathDelay* 65
Figura 4.44 – Estrutura do ensaio 66
Figura 4.45 – Distribuição do valor de *PathDelay* para o cálculo com 1 valor 66
Figura 4.46 – Distribuição do valor de *PathDelay* para o cálculo com 2 valores 67
Figura 4.47 – Distribuição do valor de *PathDelay* para o cálculo com 4 valores 67
Figura 4.48 – Distribuição do valor de *PathDelay* para o cálculo com 8 valores 67
Figura 4.49 – Distribuição do valor de *PathDelay* para o cálculo com 16 valores 68
Figura 4.50 – Distribuição do valor de *PathDelay* para o cálculo com 32 valores 68
Figura 4.51 – Troca de mensagens de sincronismo para o cálculo de *OffsetFromMaster* 69
Figura 4.52 – Estrutura do ensaio 70
Figura 4.53 – Distribuição do valor de *OffsetFromMaster* para o cálculo com 1 valor 70
Figura 4.54 – Distribuição do valor de *OffsetFromMaster* para o cálculo com 2 valores 70
Figura 4.55 – Distribuição do valor de *OffsetFromMaster* para o cálculo com 4 valores 71
Figura 4.56 – Distribuição do valor de *OffsetFromMaster* para o cálculo com 8 valores 71
Figura 4.57 – Distribuição do valor de *OffsetFromMaster* para o cálculo com 16 valores 71
Figura 4.58 – Distribuição do valor de *OffsetFromMaster* para o cálculo com 32 valores 72
Figura 4.59 – Fluxo do cálculo do número de correções de sintonização 73
Figura 4.60 – Erro de sincronismo para um *SyncPeriod* de 1 segundo 74
Figura 4.61 – Distribuição do erro de sincronismo para um *SyncPeriod* de 1 segundo 74
Figura 4.62 – Erro de sincronismo para um *SyncPeriod* de 2 segundos 74
Figura 4.63 – Distribuição do erro de sincronismo para um *SyncPeriod* de 2 segundos 75
Figura 4.64 – Erro de sincronismo para um *SyncPeriod* de 5 segundos 75
Figura 4.65 – Distribuição do erro de sincronismo para um *SyncPeriod* de 5 segundos 75
Figura 4.66 – Erro de sincronismo para um *SyncPeriod* de 10 segundos 75
Figura 4.67 – Distribuição do erro de sincronismo para um *SyncPeriod* de 10 segundos 76
Figura 4.68 – Erro de sincronismo para um *SyncPeriod* de 30 segundos 76
Figura 4.69 – Distribuição do erro de sincronismo para um *SyncPeriod* de 30 segundos 76
Figura 4.70 – Erro de sincronismo para um *SyncPeriod* de 60 segundos 76
Figura 4.71 – Distribuição do erro de sincronismo para um *SyncPeriod* de 60 segundos 77
Figura 5.1 – Rede LoWPAN numa configuração *mesh* 81
Figura 5.2 – Capacidade de carga controlada por um DAC e um varicap 81

Índice de Acrónimos

AP	*Access Point*
CISC	*Complex Instruction Set Computer*
DAC	*Digital-to-Analog Converter*
ED	*End Device*
FCS	*Frame Check Sequence*
GPIO	*General Purpose Input/Output*
GPS	*Global Positioning System*
IEEE	*Institute of Electrical and Electronics Engineers*
IETF	*Internet Engineering Task Force*
IoT	*Internet of Things*
LoWPAN	*Low Power Wireless Personal Area Network*
MAC	*Media Access Control*
NaPIS	*Navy Positioning and Identification Systems*
PC	*Personal Computer*
PPM	*Parts Per Million*
PTP	*Precision Time Protocol*
RBS	*Reference Broadcast Synchronization*
RF	*Radio Frequency*

RTC	*Real Time Clock*
RWW	*Radio and Wireless Week*
SFD	*Start Frame Delimiter*
SoC	*System on Chip*
TPSN	*Timing-sync Protocol for Sensor Networks*
UART	*Universal Asynchronous Receiver/Transmitter*
USB	*Universal Serial Bus*
WAPoS	*Wireless Acoustic Positioning System*
WiSNet	*Wireless Sensors and Sensor Networks*
WSN	*Wireless Sensor Network*

1 Introdução

Desde o início das comunicações que um dos passos obrigatórios na evolução de uma tecnologia de comunicação por fios passa pela sua implementação em redes de comunicação sem fios. Isto verificou-se nas comunicações de voz onde o telefone evoluiu para o telemóvel e no acesso à rede global chamada *internet*, onde as ligações por cabo coaxial e por *ethernet* são cada vez mais substituídas por redes *Wi-Fi*.

Este processo normal de evolução acontece porque na implementação de uma nova tecnologia de comunicação, o meio cablado por norma oferece uma maior taxa de transmissão proveniente da sua baixa perda de sinal e imunidade ao ruído, bem como uma melhor segurança pois é necessário um acesso direto ao fio usado na ligação para se ter acesso à informação trocada.

Depois dessa mesma tecnologia começar a enraizar-se no seu mercado alvo, começa a nascer a necessidade de a tornar móvel e mais fácil de instalar. Estas duas características ao serem os pontos fortes das comunicações sem fios, forçam a inevitável expansão da tecnologia para este tipo de redes de comunicação.

O aparecimento do conceito IoT (*Internet of Things*), que se baseia no desenvolvimento de redes de comunicação orientadas a dispositivos completamente autónomos ao contrário das típicas redes destinadas a utilizadores humanos, veio quebrar este processo evolutivo. A visão deste conceito é de num futuro próximo todo o nosso quotidiano encontrar-se embebido por pequenos dispositivos que de uma forma cooperativa irão contribuir para o aumento da nossa qualidade de vida, segurança e eficiência energética.

A implementação de uma rede desta dimensão através de tecnologias de comunicação por fios é inconcebível devido aos enormes custos de implementação e manutenção inerentes, o que obriga logo à partida à utilização de redes de comunicação sem fios. Apesar de ainda nos encontrarmos relativamente longe desta utopia tecnológica, têm sido feitos grandes esforços nas últimas décadas para que tal seja possível.

Toda a evolução tecnológica dos circuitos integrados permitiu o aparecimento de pequenos microcontroladores com cada vez mais capacidade de processamento e baixo consumo, chegando mesmo a integrar rádios de baixa taxa de transferência num único chip. Estes rádios ao serem criados por diferentes fabricantes, gerou um problema de integração pois as comunicações entre eles muitas vezes não são compatíveis. Este problema levou à criação do *standard* IEEE 802.15.4 [1] e suas subsequentes revisões onde foi especificado o primeiro *standard* mundial relativamente às características de modulação, frequências e estrutura de mensagens para redes de baixo consumo e baixa taxa de transferência de dados. Com este protocolo foi estabelecido o primeiro passo de encontro à visão do conceito IoT.

A existência de um *standard* que define as características de comunicação destes rádios, proporcionou o aparecimento de um conjunto de protocolos direcionados para diferentes aplicações como o *WirelessHart* [2] para ambientes industriais e o *ZigBee* [3] para a área da domótica. Estes protocolos apesar de funcionarem sobre a mesma camada MAC (*Media Access Control*) definida pelo *standard* IEEE 802.15.4, ao estabelecerem a sua própria camada de ligação, voltaram a fragmentar as redes visto que se ambas se encontrarem no mesmo espaço de comunicação, apesar de partilharem das mesmas características de rádio, não são compatíveis no reencaminhamento de mensagens. Isto levou à criação do protocolo de ligação *6LoWPAN* [4] que é um *standard* definido pela IETF [5], a mesma entidade responsável pelos protocolos de comunicação usados na *internet*. Este protocolo define toda a camada de ligação entre as unidades da rede, de forma a tornar-se uma extensão da própria internet. O seu desenvolvimento foi de tal importância para a unificação das redes de baixo consumo que levou à criação do protocolo ISA100 [6], um *standard* de comunicação industrial e á migração do *ZigBee*, com o seu novo protocolo de ligação *ZigBee IP*.

Paralelamente a toda esta evolução apareceu o conceito WSN (*Wireless Sensor Network*). Estas redes sem fios são uma versão primordial de todo o conceito IoT, formadas por pequenos dispositivos com capacidade de registar parâmetros físicos como temperatura,

humidade e pressão atmosférica entre outros, que posteriormente são reencaminhados para uma unidade central de controlo. Generalizado o seu funcionamento, apareceu o conceito LoWPAN (*Low Power Wireless Personal Area Network*) que representa todos os tipos de redes de comunicação sem fios de baixo consumo e baixa taxa de transferência, independentemente da sua aplicação e protocolos usados.

1.1 Motivação

Com o forte crescimento das implementações baseadas em redes LoWPAN e com todo o esforço realizado com o desenvolvimento de *standards* e protocolos a elas endereçadas, estamos cada vez mais próximo da utopia IoT.

Mas para que tal seja possível, é necessário solucionar outros tipos de problemas como o caso da capacidade de sincronizar todos os relógios das unidades da rede. Muitas aplicações são completamente dependentes da existência de uma referência de relógio comum a todas as unidades envolvidas e sem esse suporte nunca poderão migrar para uma implementação composta por uma rede de comunicação sem fios.

Para agravar ainda mais o problema, temos a questão da precisão do sincronismo, que dependentemente da aplicação, podem ser exigidos erros máximos de alguns segundos como erros de alguns nanossegundos. Juntando o facto de as unidades deste tipo de redes, terem grandes fatores de limitação energética e capacidade de processamento, encontrar uma solução que consiga endereçar estas limitações e ao mesmo tempo obter graus de sincronismo na ordem dos nanossegundos, é um desafio interessante de solucionar.

1.2 Objetivos

O objetivo desta dissertação é estudar, implementar e otimizar um sistema de sincronismo de relógios endereçado às características das redes LoWPAN.

Na fase de estudo, será analisado:

- as possíveis formas de criar e manter um relógio nas unidades da rede,
- as causas que levam à sua perda de sincronismo,
- que métodos existem para sincronizar os relógios das unidades e
- que trabalho foi desenvolvido e implementado anteriormente.

Na fase de implementação, será efetuado:

- uma estruturação dos blocos fundamentais de funcionamento da unidade de forma a suportar a capacidade de sincronizar o relógio local,

- uma validação inicial do funcionamento do sistema implementado,
- a implementação de técnicas de sintonização de forma a reduzir o erro do relógio e
- uma validação final do sistema.

Na fase de otimização, será melhorado:

- o registo do tempo do relógio,
- a obtenção dos parâmetros de sincronismo e
- a execução do algoritmo da unidade com o objectivo de obter melhores precisões de sincronismo.

Para generalizar o sistema desenvolvido e possibilitar o suporte a diferentes tipos de aplicações que necessitem de um relógio sincronizado com a rede, o sistema deve ter a capacidade de registar o valor do relógio em qualquer momento de execução e de realizar de forma sincronizada com o relógio, a chamada de uma função para despoletar a execução de um evento.

1.3 Estrutura

Esta dissertação encontra-se estruturada em 5 capítulos.

No primeiro capítulo é apresentado um pequeno apanhado da evolução do conceito das redes LoWPAN ao longo das últimas décadas, de forma a permitir um melhor enquadramento da importância destas redes no presente momento e no futuro próximo. São também apresentadas as motivações que levaram ao desenvolvimento desta dissertação no tema de sincronização dos relógios das unidades de redes LoWPAN e os objectivos definidos no seu desenvolvimento.

No segundo capítulo é aprofundado o tema, começando pela análise da forma como o relógio local é gerado nas unidades e pela identificação das suas limitações de estabilidade. De seguida são estudados métodos de resolução do problema de sincronismo, passando pela apresentação de diversos protocolos e implementações realizadas anteriormente.

No terceiro capítulo inicia-se toda a descrição do desenvolvimento realizado. É apresentado inicialmente a estruturação por blocos do código, sendo aprofundado cada um deles. No final desta parte é realizado uma primeira validação e análise dos resultados. Ainda neste capítulo, é apresentado o conceito de sintonização dos relógios e por fim a validação final com este algoritmo a servir de suporte ao protocolo de sincronismo.

No quarto capítulo, são descritas e validadas um conjunto de 7 otimizações realizadas ao sistema que permitiram reduzir o erro máximo de sincronismo para valores 10 vezes inferiores aos obtidos no capítulo anterior.

Por fim, no quinto capítulo, é realizado uma breve conclusão de todo o trabalho desenvolvido e implementado nesta dissertação.

2 Sincronismo de Relógios

De forma a proceder ao sincronismo dos relógios de unidades pertencentes a uma rede LoWPAN, é necessário numa fase inicial caracterizar o problema que se pretende solucionar e analisar as técnicas e protocolos desenvolvidas para esse fim.

Como tal, este capítulo começa por apresentar os métodos que as unidades usam para gerar o seu relógio local e subsequentemente as limitações que levam à necessidade de um procedimento de sincronismo. De seguida são referidos três métodos base para solucionar o problema onde será dada uma especial atenção ao método escolhido para o desenvolvimento desta dissertação. Por fim serão exibidos protocolos baseados neste método de sincronização, onde será explicado os seus princípios de funcionamento e resultados obtidos que serão usados como referência no desenvolvimento do sistema de sincronização de relógios apresentado nesta dissertação.

2.1 Problema

Numa típica rede de comunicação LoWPAN, são usadas unidades completamente autónomas no seu funcionamento, compostas por simples microcontroladores de 8, 16 ou 32 bits como centro de controlo.

Para desencadear o seu funcionamento, é necessário fornecer uma fonte de relógio que vai desencadear toda a lógica interna do controlador, tanto em termos de processamento como no funcionamento dos periféricos suportados.

Na geração de um relógio local, existem duas formas de proceder à sua implementação. Uma é usar o periférico *RTC*, presente em alguns dos microcontroladores, que permite como o nome diz, gerar um Relógio de Tempo Real (*Real Time Clock*). Este relógio é estruturado nos campos ano, mês, dia, hora, minuto e segundo e é atualizado de forma automática. A outra forma de gerar um relógio é pela configuração do periférico *timer* que é constituído por um contador de pulsos de relógio. Consoante a sua configuração, podem ser geradas interrupções com perfil periódico que ao serem usadas para incrementar o valor de uma variável, permitem criar um relógio por *software*.

A escolha por uma das duas opções depende essencialmente do nível de resolução temporal que a aplicação usada necessita. Uma implementação com o *RTC* é preferível a uma por software com utilização do *timer* porque torna todo o processo automático, reduzindo o peso de processamento e possível concorrência aos recursos do processador. Nos casos em que é pretendida uma melhor resolução de tempo, a escolha tem sempre de passar pela implementação do relógio local por interrupções do periférico *timer*.

Em ambos os casos, a precisão do relógio criado é diretamente dependente da fonte de relógio usada. Em aplicações baseadas em redes LoWPAN, são usados tipicamente cristais com tolerâncias compreendidas entre os 10ppms e os 100ppms. Considerando que 1ppm (Parts Per Million) corresponde a um erro de 1 pulso de relógio por cada 1 milhão de pulsos, os valores de erro de 10ppms e 100ppms podem ser interpretados como um erro de precisão de 10 microssegundos e 100 microssegundos por cada segundo que passe.

Outra fonte de erro de precisão do cristal encontra-se no seu fator de estabilidade com a temperatura e do valor dos condensadores de carga usados na excitação do cristal. No primeiro caso, o seu erro tipicamente está compreendido entre os 10ppms e os 100ppms, sendo por norma idêntico ao seu valor de tolerância quando encontrado dentro da sua gama de temperaturas de funcionamento. Em relação aos condensadores de carga, cada cristal necessita de um valor de capacitância especifico para o seu correto funcionamento, por norma definido no seu *datasheet* e a utilização de valores diferentes dos recomendados vai repercutir-se num desvio da frequência gerada pelo cristal (Figura 2.1).

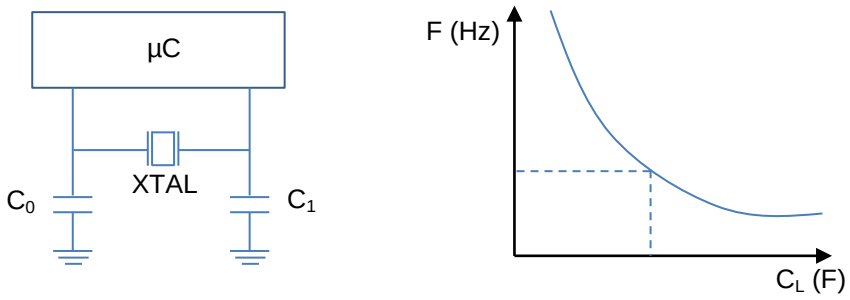

Figura 2.1 – Impacto do valor dos condensadores de carga do cristal

Em aplicações que necessitem da existência de uma referência de relógio comum a toda a rede LoWPAN, este conjunto de fontes de erro vai provocar que mesmo que os seus relógios partam de um valor idêntico inicial, vá ser acumulado ao longo do tempo um erro equivalente a toda a tolerância de erro do relógio (Figura 2.2). Este aumento constante do erro entre os relógios das unidades obriga à utilização de procedimentos de sincronismo de relógio com perfil periódico.

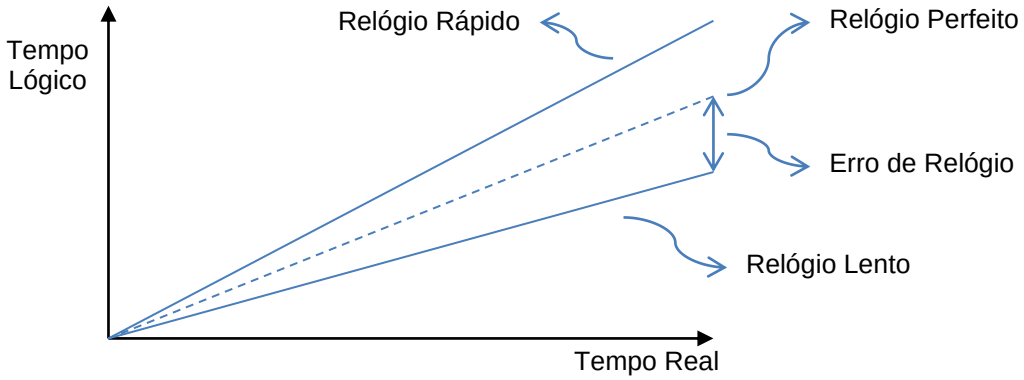

Figura 2.2 – Desvio do valor de relógio ao longo do tempo

2.2 Soluções

De forma a endereçar este problema, são normalmente usados 3 tipos de métodos de sincronismo dos relógios.

O mais simples é a utilização de uma fonte externa de sincronismo, geralmente um módulo GPS que permite que cada unidade se sincronize com precisões na ordem dos nanossegundos (Figura 2.3). A escolha deste método tem a vantagem de cada unidade se sincronizar de forma independente, dando uma maior robustez no caso de algumas unidades deixarem de funcionar corretamente. O problema levanta-se quando a aplicação desenvolvida encontra-se em espaços fechados sem cobertura do sinal de GPS ou o facto de a utilização de um módulo GPS por unidade encarecer muito a solução final, tornando a sua utilização inviável.

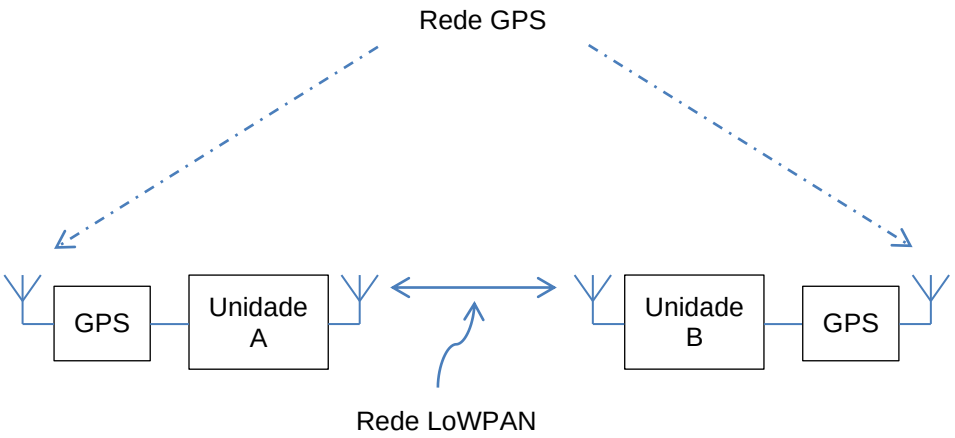

Figura 2.3 – Sincronização por GPS

O segundo método baseia-se na utilização de um canal de comunicação dedicado para o sincronismo dos relógios que é implementado em paralelo com o canal de comunicação de troca de mensagens da rede LoWPAN (Figura 2.4). Neste canal pode ser implementado tanto um conjunto de troca de mensagens de sincronismo como num caso mais simples, a transmissão de pulsos sincronizados com o relógio de uma unidade de referência, que ao serem recebidos pelas restantes unidades, permite ter uma referência para compensar o valor dos seus relógios. Apesar de permitir uma maior liberdade do local onde o sistema pode ser colocado em funcionamento, continua a ter o problema da necessidade de *hardware* extra que acaba sempre por encarecer a solução final.

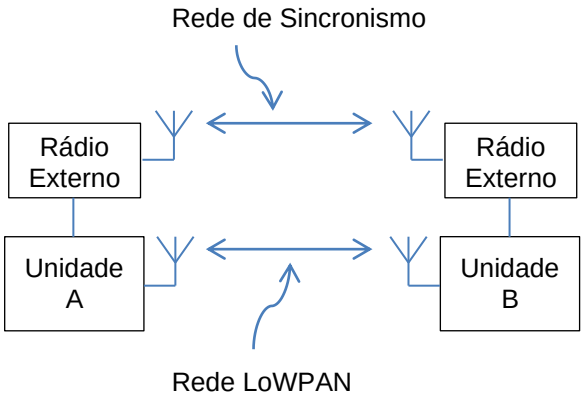

Figura 2.4 – Sincronização com rede dedicada

O terceiro método e o explorado nesta dissertação, usa o próprio canal de comunicação estabelecido para troca de mensagens da rede LoWPAN para efetuar a troca de mensagens de sincronismo. Este método acaba por ser o mais viável porque não necessita de *hardware* extra, mas por outro lado leva a um aumento da complexidade da sua implementação.

2.2.1 Caracterização dos atrasos na troca de mensagens

Ao se seguir pela utilização direta do canal de comunicação estabelecido pela rede LoWPAN, obrigatoriamente todo o processo de sincronismo dos relógios tem de ser baseado na troca de mensagens.

O caso mais simples de implementação seria o transporte direto do valor de relógio da unidade de referência para as restantes unidades da rede, o que apresentaria erros de sincronismo de alguns milissegundos ou até mesmo alguns segundos, dependendo da taxa de transferência e do tamanho das mensagens usadas.

Para se poder obter erros de sincronismo na ordem de microssegundos e nanossegundos, é necessário caracterizar todas as latências existentes desde o instante em que o tempo do relógio de referência é registado e transmitido até ao momento em que a mensagem é recebida e o seu tempo de chegada é obtido.

Pela Figura 2.5, o tempo que leva desde que os tempos de transmissão e recepção da mesma mensagem sejam registados, t_1 e t_5 correspondentemente, é composto por 4 tipos de latências. Tempo de acesso ao meio (de t_1 a t_2), tempo de transmissão (de t_2 a t_3), tempo de propagação (de t_3 a t_4) e tempo de processamento da mensagem (de t_4 a t_5).

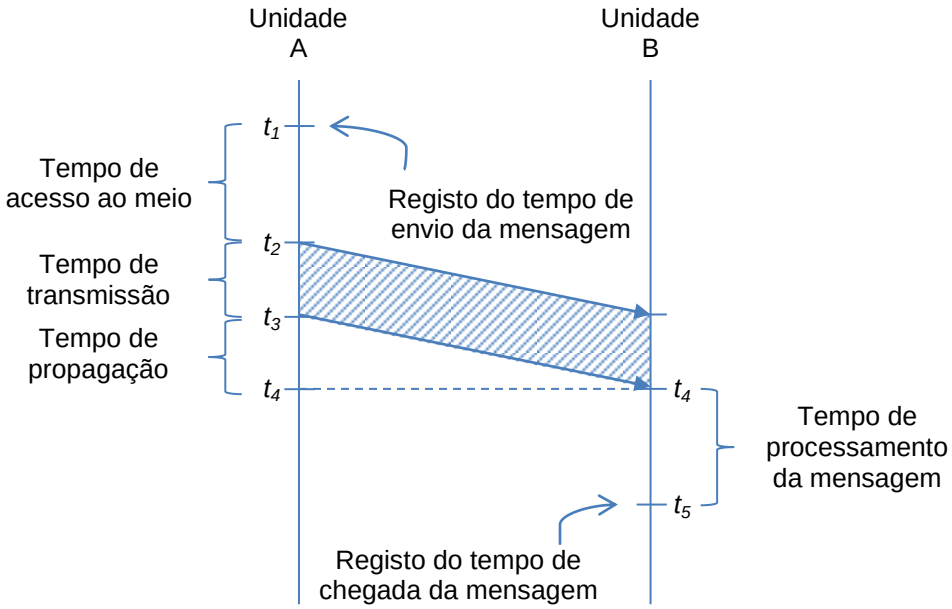

Figura 2.5 – Latências entre o tempo de transmissão e receção de mensagens

Cada um destes tempos é composto por um fator determinístico e um fator aleatório. Atrasos determinísticos, se calculados, podem ser usados na correção do erro do relógio enquanto os atrasos aleatórios por outro lado, só podem ser estimados, logo devem ser minimizados de forma a reduzir o seu impacto.

No caso do tempo de propagação, considerando que as redes LoWPAN usam comunicações RF e o ar como meio físico de comunicação, obtém-se uma propagação com uma velocidade de aproximadamente igual à velocidade da luz, o que leva a que por norma o seu tempo de atraso seja considerado desprezável. Isto deixa de ser verdade no caso de uma aplicação que use ultrassons como forma de comunicação, como no caso de aplicações que necessitem de comunicar dentro de água, onde a sua velocidade de propagação desce para 1435 m/s.

No caso do tempo de transmissão, o seu valor é maioritariamente determinístico, sendo diretamente dependente da taxa de transmissão e do tamanho da mensagem. Existe também uma parte aleatória associada a este atraso que depende essencialmente de latências do próprio rádio. Este atraso não pode ser controlado, mas por norma pode ser considerado desprezável.

No caso dos tempos de acesso ao meio e de processamento de mensagens recebidas, os seus valores de atraso dependem essencialmente da implementação do protocolo de comunicação usado. De forma a reduzir a sua latência em grande parte aleatória, proveniente da sua dependência com o nível de processamento do microcontrolador no momento em que é efetuada a transmissão e receção da mensagem, os valores de t_1 e t_5 da Figura 2.5 devem ser registados o mais perto possível do momento exato em que a mensagem é transmitida e recebida.

2.2.2 Protocolos de sincronismo de relógios

Nas últimas décadas foi realizado um grande esforço no desenvolvimento de protocolos de sincronismo pensados nas características típicas de baixo consumo, baixo poder de processamento e baixa capacidade de dados das mensagens das redes LoWPAN. A acompanhar este esforço, foram publicados vários *overviews* comparativos dos resultados obtidos ao longo dos anos [7], [8] e [9]. De seguida serão referenciados alguns desses protocolos, passando pela explicação do seu princípio de funcionamento e resultados obtidos.

2.2.2.1 Protocolo RBS

Com o protocolo RBS [10], o problema do tempo de acesso ao meio é resolvido, usando um processo de sincronismo onde o seu valor não necessita de ser considerado. Partindo do pressuposto que o tempo de propagação das mensagens é desprezável, pode ser assumido que uma mensagem enviada por uma unidade de referência é recebida praticamente no mesmo instante de tempo por todas as unidades que se encontrem na sua vizinhança. Esta particularidade permite que os tempos registados em cada uma das unidades que receberam a mensagem, possam ser diretamente correlacionados, ficando a sua precisão limitada simplesmente pelo tempo aleatório associado ao tempo de processamento da mensagem.

No protocolo RBS, é definido uma unidade responsável por enviar as mensagens de referência. As restantes unidades após receberem as mensagens, registam o seu tempo de receção e trocam entre si o valor obtido (Figura 2.6). Como não é realizado uma correção do valor dos seus relógios locais, sempre que uma unidade necessita de enviar uma mensagem com uma referência temporal, inclui na mensagem o valor do seu relógio que posteriormente é compensado visto que a unidade destino da mensagem consegue saber a diferença entre os seus relógios, subtraindo o tempo registado na receção da mensagem de referência pelo tempo que a unidade de origem registou na receção dessa mesma mensagem.

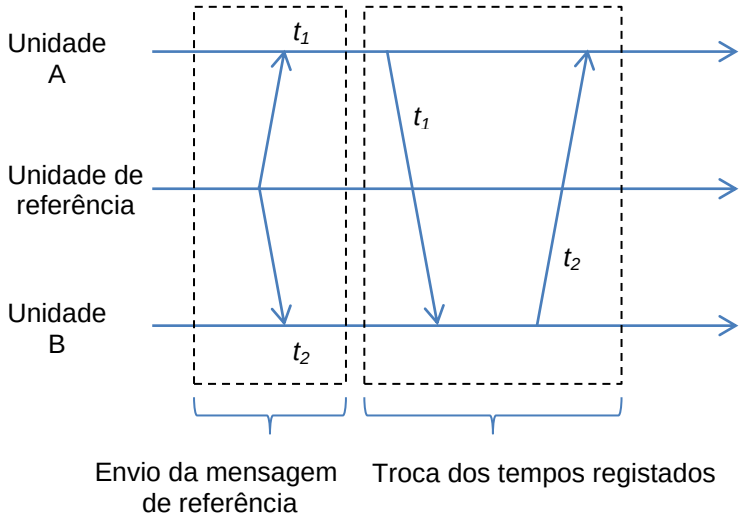

Figura 2.6 – Troca de mensagens do protocolo RBS

Este protocolo apresenta duas grandes limitações à sua implementação prática. Em primeiro, como os relógios das unidades não são corrigidos diretamente, este protocolo só pode ser usado na detecção de eventos e não na execução de eventos sincronizados entre as unidades. Em segundo, cada vez que as unidades trocam os valores de relógio registados, é necessário que cada unidade envie *n-2* mensagens, sendo *n* o número de unidades da rede, o que faz com que o número total de mensagens trocadas por cada ponto de sincronismo de relógios cresça de forma quadrática com o aumento do número de unidades da rede.

Para validação, os autores do protocolo RBS, montaram uma rede LoWPAN constituída por 5 *Berkeley Motes*. Nesta validação foram obtidos erros máximos de 11 microssegundos.

2.2.2.2 Protocolo TPSN

A diferença entre o valor dos tempos t_1 e t_5 apresentados na Figura 2.5, pode ser correlacionada pelo tempo real que leva desde o registo do tempo t_1 até ao registo do tempo t_5 (t_{Delay}) mais a diferença entre os relógios no mesmo instante de tempo (t_{Offset}). Esta relação pode ser observada na Figura 2.7 e nas equações (2.1), (2.2) e (2.3).

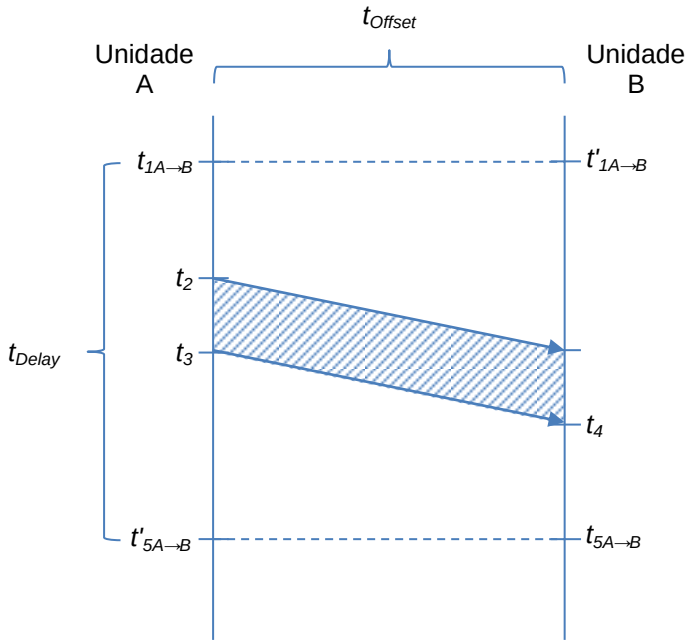

Figura 2.7 – Ilustração dos tempos t_{Delay} e t_{Offset}

$$t_{Delay} = t'_{5A\rightarrow B} - t_{1A\rightarrow B} \quad (2.1)$$

$$t_{Offset} = t'_{1A\rightarrow B} - t_{1A\rightarrow B} \quad (2.2)$$

$$t_{5A\rightarrow B} = t_{1A\rightarrow B} + t_{Delay} + t_{Offset} \quad (2.3)$$

Ao ser enviada uma mensagem no sentido oposto, passamos a ter os novos valores de t_1 e t_5 correlacionados de forma idêntica, mas com a polaridade do tempo t_{Offset} invertida (equação (2.4)).

$$t_{5B\rightarrow A} = t_{1B\rightarrow A} + t_{Delay} - t_{Offset} \quad (2.4)$$

Considerando os atrasos simétricos nos dois sentidos e que a diferença entre os relógios é constante durante a troca das duas mensagens, podem-se isolar os valores de t_{Delay} e t_{Offset} pelas equações (2.5) e (2.6).

$$t_{Delay} = \frac{(t_{4A\rightarrow B} - t_{1A\rightarrow B}) + (t_{4B\rightarrow A} - t_{1B\rightarrow A})}{2} \quad (2.5)$$

$$t_{Offset} = \frac{(t_{4A\rightarrow B} - t_{1A\rightarrow B}) - (t_{4B\rightarrow A} - t_{1B\rightarrow A})}{2} \quad (2.6)$$

O protocolo TPSN [11] usa o cálculo destes dois parâmetros para realizar o sincronismo dos relógios (Figura 2.8).

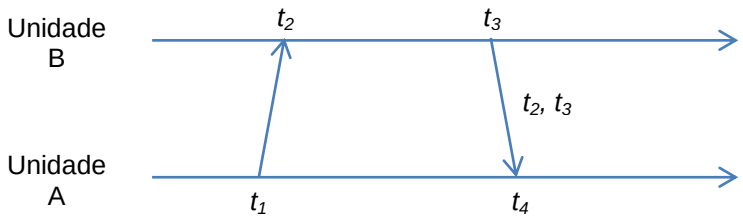

Figura 2.8 – Troca de mensagens do protocolo TPSN

O processo começa com a unidade que se pretende sincronizar a enviar a primeira mensagem onde são obtidos os tempo t_1 e t_2. De seguida, a unidade com o relógio de referência responde com outra mensagem e os tempos t_3 e t_4 são registados. Nesta mensagem de resposta são incluídos os tempos t_2 e t_3 obtidos no processo. Com estes 4 tempos a unidade calcula os dois parâmetros e efetua a correção do seu relógio local.

O envio do tempo t_3 na mensagem de resposta é possível se o rádio usado permitir modificar os campos da mensagem ao mesmo tempo que se encontra a transmiti-la, visto que o tempo t_3 corresponde ao tempo de envio dessa mesma mensagem. Isto não é o caso da maioria dos rádios presentes no mercado, o que limita assim a sua implementação na prática. Outra possibilidade seria definir o valor de t_3 como sendo uma estimativa do tempo de envio da mensagem, adicionando erros aleatórios extra ao processo.

Outra grande desvantagem deste protocolo, encontra-se no crescimento linear do número de mensagens trocadas no processo de sincronização em função do aumento do número de unidades da rede, que apesar de não ter um crescimento tão acentuado como no protocolo RBS, continua a ser um problema quando temos redes compostas por um número de unidades na ordem das centenas.

Em termos de validação do protocolo, os autores implementaram uma rede composta por 2 *Berkeley Motes* e obtiveram um erro de sincronismo médio de 16.9 microssegundos e um erro máximo de 44 microssegundos. Os autores para comparação, implementaram também o protocolo RBS onde foi obtido um erro médio de 29 microssegundos e um erro máximo de 93 microssegundos.

2.2.2.3 Protocolo IEEE 1588

Analisando os tempos t_{Delay} e t_{Offset} definidos no subcapítulo anterior, podem ser retiradas algumas ilações sobre a estabilidade dos seus valores, obtidos ao longo de múltiplos pontos de sincronismo. No caso do t_{Delay}, se for considerado uma implementação onde os atrasos aleatórios, associados ao tempo de acesso ao meio e ao tempo de processamento da mensagem, são reduzidos a um valor desprezável, o que é possível se os tempos de transmissão e receção de mensagens forem registados no exato momento em que a mensagem é transmitida e recebida, pode ser assumido que o seu valor é constante ao longo do tempo. Neste caso será necessário efetuar apenas o seu recalculo quando as características da ligação entre a unidade de referência e a unidade sincronizada sofrerem alterações. Já no caso do tempo t_{Offset}, o seu valor encontra-se diretamente ligado à estabilidade e desvio de frequência dos relógios das unidades. Mesmo compensando de alguma forma o desvio na frequência, a estabilidade do relógio é difícil de controlar porque apresenta uma forte dependência com a variação temperatura do meio onde a unidade se encontra.

O *standard* IEEE 1588 [12], também conhecido por PTP (*Precision Time Protocol*), ao tomar estas características em conta, estabelece um processo de troca de mensagens que permite calcular os tempos de t_{Delay} e t_{Offset} usando periodicidades diferentes. Desta forma é possível reduzir o número de mensagens trocadas em cada ponto de sincronismo visto que depois do cálculo do tempo de t_{Delay}, é necessário apenas a transmissão de 2 mensagens em *broadcast* pela unidade de referência para calcular o tempo de t_{Offset}, podendo ser usadas para sincronizar uma rede formada por 2 unidades como por 200 unidades.

O protocolo define diferentes tipos de mensagens que são trocadas entre as unidades, mas para a execução do sincronismo em si, são necessárias apenas 4 mensagens intituladas *Sync*, *Follow_Up*, *Delay_Req* e *Delay_Resp*. (Figura 2.9). Ao contrário do protocolo TPSN, a primeira mensagem de sincronismo é enviada pela unidade de referência com a mensagem *Sync*, onde é transportado uma estimativa do valor de t_1 enquanto nas restantes unidades é registado o tempo t_2. De seguida a mesma unidade envia a mensagem *Follow_Up* com o valor exato de t_1. As unidades sincronizadas ao receberem este último tempo, se já tiverem calculado o valor de t_{Delay} anteriormente, procedem ao cálculo do tempo t_{Offset} e á correção dos seus relógios pela equação (2.7). Caso ainda não tenham calculado este valor ou queiram efetuar o seu recalculo, iniciam o processo de obtenção do valor de t_{Delay} com o envio da mensagem *Delay_Req* e o registo do tempo t_3. Por sua vez a unidade de referência regista o tempo t_4 que de seguida é reencaminhado de volta na mensagem *Delay_Resp*. Com estes 4 tempos as unidades sincronizadas obtêm o valor de t_{Delay} pela equação (2.8)

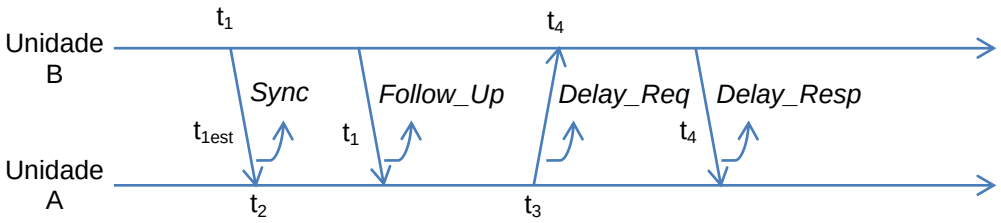

Figura 2.9 – Troca de mensagens do protocolo PTP

$$t_{Offset} = (t_2 - t_1) - t_{Delay} \tag{2.7}$$

$$t_{Delay} = \frac{(t_2 - t_1) - (t_4 - t_3)}{2} \tag{2.8}$$

Para validação do protocolo aplicado a redes LowPAN, foram analisadas duas implementações do sistema. Uma com o objectivo de implementar uma solução de baixo custo e outra com o intuito de obter o melhor nível de precisão permitido.

Na primeira solução [13] foi usado uma unidade composta por dois microcontroladores, um responsável pela execução da aplicação e outro pela manutenção do relógio local. Foi também usado um rádio de 2.4GHz e um cristal de 16MHz com um erro de 10ppms. Com esta unidade, usando um período de sincronismo de 2 segundos, obtiveram um erro máximo de sincronismo de 10 microssegundos.

Na segunda solução [14] foi implementado uma unidade composta por um microcontrolador de 32 bits, onde foi usado o seu coprocessador na manutenção do relógio de referência. Em termos de comunicação foi usado também um rádio de 2.4GHz e um cristal de 32MHz com uma tolerância de 1.5ppms. Neste caso foram obtidos melhores resultados, com um erro máximo de sincronismo de 160 nanossegundos quando usado um período de sincronismo de 100 milissegundos.

3 Primeiro Protótipo

Este primeiro protótipo serviu de suporte para uma melhor compreensão das dificuldades inerentes ao desenvolvimento e implementação de um protocolo de sincronismo de relógios em redes LoWPAN. Um dos pontos fulcrais a ser tomado em consideração é o fator de baixo consumo e de baixa capacidade de processamento, normalmente disponível nos microcontroladores das unidades usadas nestas redes, que leva à necessidade da utilização de um protocolo de sincronismo simples, tanto a nível de complexidade algorítmica como a nível do número de mensagens necessárias no seu processo.

Tendo em conta estes requisitos, decidiu-se implementar um protocolo baseado numa versão simplificada do *standard* IEEE 1588. Apesar deste protocolo ter sido desenvolvido inicialmente a pensar nas características de canais de comunicação como *Ethernet* e *WiFi*, trabalhos anteriores [14] mostraram ser viável a sua utilização em redes sem fios com características muito mais limitadas, sendo possível obter precisões na ordem dos nanossegundos.

Para servir de suporte a este desenvolvimento, usou-se a placa do projeto *NaPIS* (Figura 3.1) visto que se encaixa perfeitamente no perfil que se pretende atingir com o nosso sistema,

em termos de características de comunicações de rádio de baixa taxa de transferência e sistemas de baixo consumo alimentados a pilhas ou baterias. Nesta placa, ao contrário das unidades apresentadas na secção 2.2.2.3, que usavam centros de processamento distintos para executar a aplicação e a manutenção do relógio local, vai ser usado um único chip, *SoC CC1110*, composto por um módulo de rádio de 433MHz e um processador de 8 bits onde serão executadas todas a funcionalidades da unidade. Em relação ao cristal, este tem uma frequência de 26MHz e uma tolerância de 10ppms. Apesar de esta unidade ter um módulo de GPS, ele não vai ser usado nesta fase do desenvolvimento do sistema de sincronismo.

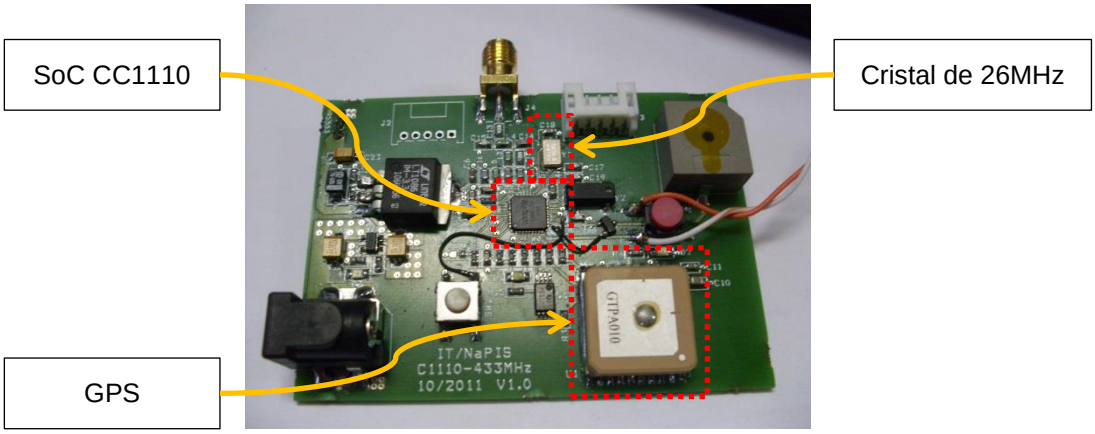

Figura 3.1 – Componentes da placa *NaPIS*

Apesar deste primeiro protótipo servir primariamente como base de estudo, foi tomado o cuidado de o tornar estruturado em blocos, de forma a que tenham uma atividade independente e transparente entre si, possibilitando que cada um deles possa ser substituído e/ou melhorado sem influenciar o funcionamento dos restantes blocos.

Este capítulo começa pela enumeração dos quatro blocos fundamentais ao funcionamento do sistema, descrevendo toda a sua estrutura de execução, funções de interação e sinalizadores globais de acesso externo que permitem o seu correto funcionamento e interação com os restantes blocos. De seguida é apresentado uma validação inicial do funcionamento do sistema e análise dos resultados obtidos. Numa terceira parte é introduzido o conceito de sintonização de relógios que vai permitir reduzir de forma crítica o número de mensagens de sincronismo por segundo ao mesmo que se melhora os resultados de erro máximo do sincronismo de relógios obtido. O capítulo é finalizado com a referência ao reconhecimento obtido numa conferência internacional e ao suporte dado com este protótipo ao projeto *WAPoS* onde este sistema serviu de base a uma aplicação de localização *indoor*.

3.1 Descrição dos Blocos do Sistema

A arquitetura do sistema foi estruturada em quatro blocos fundamentais, cada um ficando responsável por uma parte importante do funcionamento da unidade. O perfil da interação entre os blocos é apresentado na Figura 3.2.

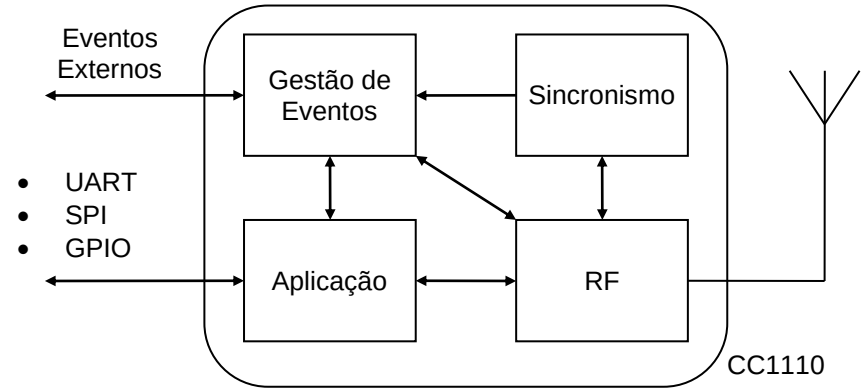

Figura 3.2 – Estruturação global dos blocos do sistema

A estrutura começa com o Bloco RF, responsável por alojar o protocolo de comunicação sem fios usado pelas unidades. No nosso caso foi usado o protocolo *SimpliciTI* da *Texas Instruments* [15], que devido à sua simplicidade permite montar facilmente um canal de comunicação sem fios entre unidades que pretendam trocar mensagens entre si.

De seguida temos o Bloco de Sincronismo responsável por todas as mensagens necessárias para a realização do processo de sincronização de relógios. Este bloco é responsável pela gestão do fluxo de troca de mensagens, que posteriormente vai permitir o cálculo dos parâmetros necessários à sincronização do relógio.

Em terceiro temos o Bloco de Gestão de Eventos responsável por três funcionalidades fundamentais tanto para o funcionamento da unidade, como para o suporte de aplicações que requeiram acesso a um relógio globalmente sincronizado com rede de comunicação. Estas funcionalidades são a manutenção do relógio da unidade com possibilidade de ser sincronizado, capacidade de executar eventos de forma síncrona com o relógio da unidade e registo do tempo de ocorrência de um evento.

Por último temos o Bloco de Aplicação que não tem um papel influente no processo de sincronismo, mas permite que diversas aplicações possam ser suportadas de forma independente aos processos internos da unidade ao mesmo tempo que usufruem da capacidade de geração de eventos e registo do tempo de ocorrências de eventos através do Bloco de Gestão de Eventos e da troca de mensagens pelo canal de comunicação mantido pelo Bloco de RF. Um exemplo de uma aplicação desenvolvida neste bloco é apresentado no capítulo 3.5 onde o trabalho desenvolvido no primeiro protótipo serviu de suporte.

3.1.1 Bloco RF

O Bloco de RF é composto essencialmente pelo protocolo *SimpliciTI* da *Texas Instruments*. Este protocolo permite criar um canal de troca de mensagens numa topologia estrela com uma unidade AP no centro da topologia e diversas unidades ED nos seus extremos. De forma a permitir uma maior liberdade na utilização do protocolo, foram implementadas adicionalmente as funcionalidades de Identificador de Rede (*PANID*) e de Identificador de Serviço (*ServiceID*).

Estes dois identificadores são adicionados no início de todas as mensagens trocadas entre as unidades como apresentado na Figura 3.3.

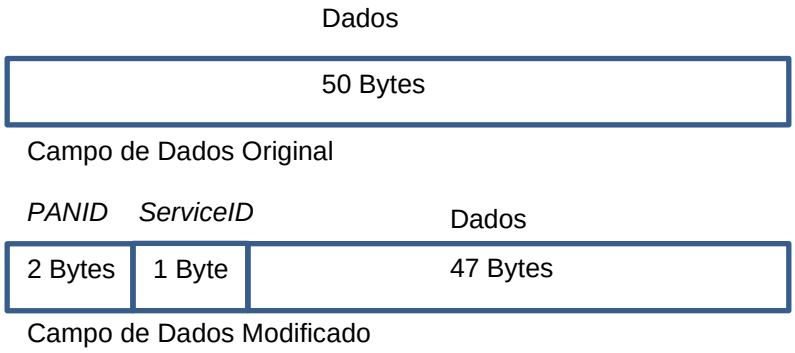

Figura 3.3 – Estruturação das mensagens de rádio

Para obtermos um correto funcionamento do protocolo de sincronismo, é necessário também a capacidade de registar o tempo em que as mensagens são transmitidas e recebidas o mais próximo da possível da sua ocorrência. Isso foi possível, adicionando uma chamada à função de registo de tempo, disponível nas funções de interação do Bloco de Gestão de Eventos.

3.1.1.1 Identificador de Rede

No protocolo *SimpliciTI*, o endereçamento das comunicações é feito especificando um *linkID* que corresponde a uma ligação estabelecida entre duas unidades que pretendam comunicar entre si. No caso de uma aplicação necessitar de transmitir uma mensagem para todas a unidade da rede, é utilizado um *linkID* de *broadcast* predefinido para esse fim. A limitação do envio de mensagens neste modo de endereçamento encontra-se no facto de as unidades que recebem a mensagem transmitida, não conseguirem saber qual foi a unidade de origem da mensagem. Esta limitação torna-se um problema crítico se uma das unidades se encontrar dentro do alcance de comunicação de uma rede vizinha, podendo originar que a unidade receba uma mensagem dessa rede e a interprete como sendo da sua própria rede de comunicação, podendo criar incongruências no funcionamento da aplicação. No protoloco de

sincronismo implementado este problema é crítico, visto que parte das mensagens vão ser transmitidas com o *linkID* de *broadcast* como destino.

Para endereçar este problema, foi definido o campo Identificador de Rede (*PANID*) de 16 bits. Inicialmente todas as unidades começam com o *PANID* igual a 0xFFFF. No caso da unidade AP, ao criar a rede, estabelece o valor do *PANID* como sendo o valor dos 2 bytes mais significativos do seu endereço. Uma unidade ED ao se ligar a uma rede define o seu *PANID* com o mesmo valor estabelecido pela unidade AP da sua rede. Na fase de transmissão de uma mensagem, cada unidade adiciona o seu *PANID* à mensagem. Na fase de receber uma mensagem, é efetuado um processo de decisão se a mensagem deve ser descartada ou processada, dependendo do valor do *PANID* presente na mensagem recebida e do valor do *PANID* da própria unidade no momento em que recebe a mensagem. O processo de decisão é apresentado na Figura 3.4.

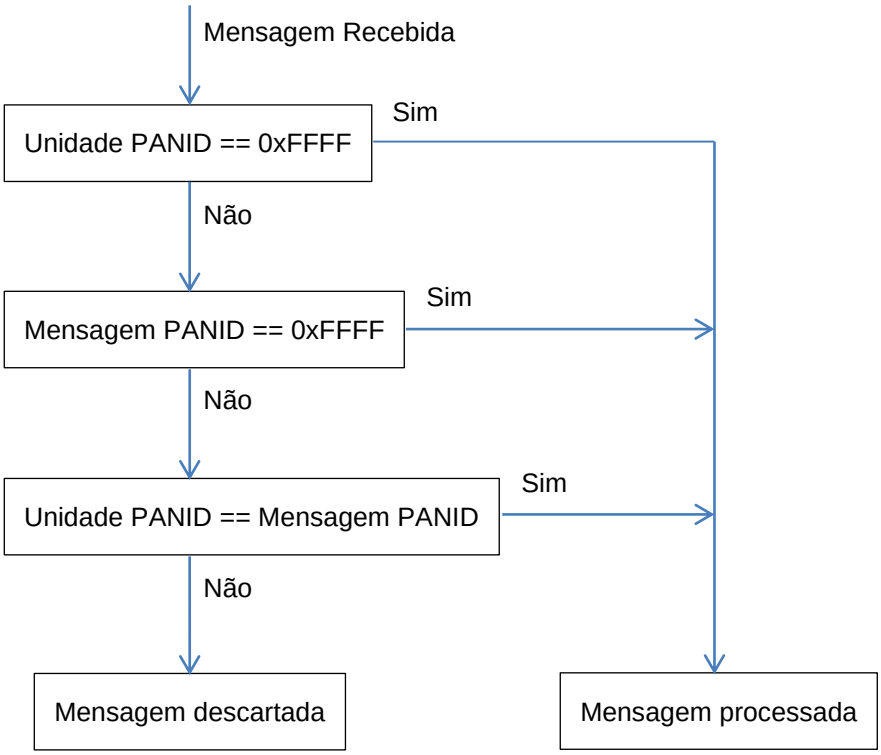

Figura 3.4 – Validação do *ServiceID* da mensagem

Após uma mensagem ser recebida, são analisadas três condições, no caso de uma delas ser verdadeira a mensagem é processada, se todas forem falsas a mensagem é descartada.

- Se o *PANID* da unidade for 0xFFFF, todas as mensagens são aceites visto que a unidade no momento não está inserida em nenhuma rede em específico.

- Se o *PANID* da mensagem for 0xFFFF, a mensagem foi enviada por uma unidade que não pertence a nenhuma rede. Este caso permite que uma unidade possa comunicar com várias redes presentes no mesmo local para tarefas de gestão e monitorização.
- Se o *PANID* da unidade for igual ao *PANID* da mensagem, estamos numa situação onde a unidade que recebeu a mensagem e a unidade que transmitiu a mensagem pertencem à mesma rede.

3.1.1.2 Identificador de Serviço

O protocolo *SimpliciTI* permite nativamente suportar comunicações orientadas a serviços, tanto internos como externos ao protocolo de comunicação. Para tal é necessário criar uma conexão com uma unidade específica, indicando o seu endereço. Esta abordagem trás algumas limitações para a implementação do protocolo de sincronismo.

Em primeiro lugar, uma unidade que se pretenda sincronizar, não sabe à partida o endereço da unidade que tem o serviço de sincronismo. A unidade necessita da capacidade de receber mensagens de sincronismo de qualquer unidade da rede onde se encontra ligada.

Em segundo lugar, mensagens recebidas de um dado serviço são tratadas de uma forma genérica, sem nenhum tipo reencaminhamento automático. Para o correto funcionamento do serviço, tem de ser suportada a capacidade de registar uma função de tratamento de mensagens recebidas, para que o processo seja transparente para os restantes serviços implementados.

Para resolver estas limitações, foi criado o campo Identificador de Serviço (*ServiceID*) que identifica o serviço de destino das mensagens enviadas. Na fase de inicialização das unidades cada serviço é registado especificando o *ServiceID* e a função que é chamada para tratar as mensagens recebidas com esse *ServiceID*. Ao ser recebida uma mensagem, este campo é analisado e podem acontecer uma de duas situações.

- O *ServiceID* não se encontra registado, logo este serviço não é suportado e a mensagem é descartada.
- O *ServiceID* encontra-se registado e a sua função de tratamento é chamada para processar a mensagem.

3.1.1.3 Tempo de Transmissão e Receção de Mensagens

De forma a registar corretamente o tempo de transmissão e receção de mensagens teve de ser realizado uma análise da forma como o protocolo *SimpliciTI* efetua este processo.

A transmissão é feita de forma bloqueante no momento em que é chamada a função de transmissão da mensagem. Por outras palavras, só após a mensagem ser enviada ou ocorrer um erro na transmissão é que a execução da função é finalizada. Após uma análise das várias

chamadas internas da função, foi descoberta a instrução responsável pelo desencadeamento da transmissão, a seguir a esta instrução foi adicionado a chamada à função de registo de tempo.

A receção por outro lado é sinalizada com uma interrupção do rádio, onde é verificada a integridade dos dados e posteriormente reencaminhada para uma tabela de mensagens recebidas. Logo no início desta interrupção foi inserido uma chamada à função de registo de tempo.

Após serem registados os tempos de transmissão e receção de mensagens, estes são guardados em duas variáveis, *TimeStampTX* para o tempo de transmissão e *TimeStampRX* para o tempo de receção. Posteriormente esses tempos são usados pelo protocolo de sincronismo no cálculo dos parâmetros necessários ao seu funcionamento.

3.1.1.4 Interface do Bloco

Funções de interação

As funções de interação implementadas foram estruturadas em três grupos. Gestão do Bloco, Configuração do Bloco e Troca de Mensagens.

Gestão do Bloco

A gestão do funcionamento do Bloco RF é constituída por três funções.

- **Init** – A função de inicialização deve ser chama no arranque da unidade de forma a configurar os estados internos do protocolo *SimpliciTI*.
- **Connect** – Esta função desencadeia o processo de uma unidade pertencer a uma rede. No caso da unidade AP, ela vai formar uma nova rede, no caso de uma unidade ED, ela vai juntar-se a uma rede existente.
- **Main** – Após o processo de inicialização e conexão a uma rede, esta função deve ser chamada periodicamente para executar o processo de reencaminhamento das mensagens para os serviços registados.

Configuração do Bloco

No processo de configuração foram definidos quatro comandos, mas podem ser definidos mais se existir a necessidade de uma aplicação ter mais controlo sobre o rádio

- **SetConnectionKey** – O protocolo *SimpliciTI* só permite que duas unidades estabeleçam uma ligação se partilharem à partida da mesma chave de conexão. Esta função permite definir essa mesma chave.

- **SetAddr** – A atribuição dos endereços nas unidades é feita de uma forma estática, visto que o protocolo não tem nenhum suporte para a criação de endereços de forma dinâmica. Esta função permite que cada unidade possa especificar o seu endereço.
- **SetJoinPermition** – Esta função possibilita ativar/desativar a permissão de novas unidades se juntarem à rede. A sua execução só é válida na unidade AP.
- **SetRadioRX** – Uma unidade quando não se encontra a transmitir uma mensagem, pode ter o seu rádio em dois estados de funcionamento. Ou encontra-se em modo de receção de forma a poder receber mensagens, ou encontra-se em modo de baixo consumo onde parte do seu *hardware* é desativado. Esta função permite estabelecer que estado de funcionamento pretende-mos em cada instante.

Troca de Mensagens

As trocas de mensagens são controladas por duas funções

- **InitSlot** – Esta função permite registar uma função a ser chamada para tratamento de mensagens recebidas, destinadas a um certo *ServiceID*. Mensagens recebidas para um *ServiceID* não registado são automaticamente descartadas.
- **Send** – Para desencadear uma transmissão, é usada esta função. A função retorna Verdadeiro ou Falso a indicar se a mensagem foi ou não enviada com sucesso.

Sinalizadores Globais

Um sinalizador foi implementado neste bloco

- **NetworkConnected** – Após uma unidade estar inserida numa rede, este sinalizador é ativado. Esta informação pode ser usada para aplicações começarem a sua execução só após estarem ligadas a uma rede.

3.1.2 Bloco de Sincronismo

No bloco de sincronismo encontra-se todo o processo necessário ao cálculo dos parâmetros de correção do relógio. Este bloco interage com o Bloco de RF na troca de mensagens com outras unidades e com o Bloco de Gestão de Eventos na correção do valor do relógio da unidade.

O protocolo de sincronismo implementado é baseado no processo criado pelo *standard* IEEE 1588. Este *standard* estabelece quatro tipos de mensagens, que ao serem trocadas entre a unidade que tem o relógio de referência e a unidade que se pretende sincronizar, permite calcular dois parâmetros essenciais ao processo, sendo eles o atraso do canal de comunicação (*PathDelay*) e o desvio do relógio da unidade sincronizada em relação ao relógio da unidade de referência (*OffsetFromMaster*).

Após estes dois parâmetros serem calculados, o relógio da unidade sincronizada é corrigido através da chamada à função de correção de desvio do Bloco de Gestão de Eventos. Como cada unidade usa o seu próprio cristal para criar o relógio local, a diferença entre o valor dos relógios após ser corrigido vai aumentar ao longo do tempo. Isto leva a que seja necessário repetir o processo de sincronismo de forma periódica (*SyncPeriod*).

3.1.2.1 Estrutura das mensagens

Apesar de o princípio do *standard* IEEE 1588 ser relativamente simples, a estruturação das mensagens é relativamente complexa e grande para sistemas baseados em comunicações rádio de baixa taxa de transferência e reduzida capacidade de dados por mensagem. Devido a isso, as mensagens foram reestruturadas como apresentado na Figura 3.5 de forma a terem somente a informação mínima necessária à realização do processo de sincronismo.

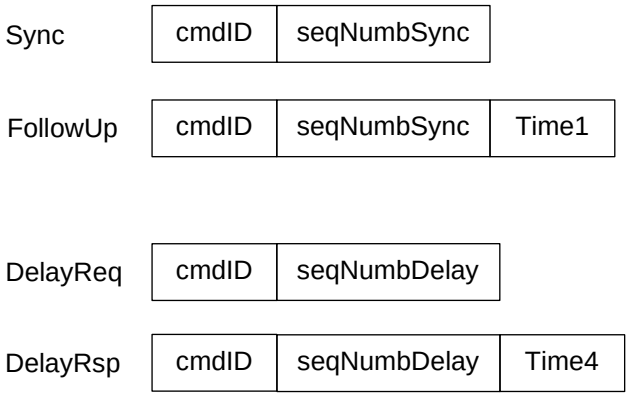

Figura 3.5 – Estrutura das mensagens de sincronismo

Cada tipo de mensagem é identificado por um identificador de comando (*cmdID*) presente no início da mensagem. De seguida temos um número de sequência para as mensagens *Sync* e *FollowUp* (*seqNumbSync*) e outro para as mensagens *DelayReq* e *DelayRsp* (*seqNumbDelay*). Este número de sequência garante a consistência dos tempos registados. No caso das mensagens *FollowUp* e *DelayRsp*, um valor de tempo é incluído para que a unidade que se pretende sincronizar, possa calcular os parâmetros necessários.

3.1.2.2 Formato do tempo registado

O formato usado no registo dos tempos é idêntico ao especificado no *standard* IEEE 1588, este formato é composto por duas variáveis de 4 bytes, uma para o registo dos segundos e outra para o registo da parte fracionária do tempo com resolução de nanossegundos. No caso do valor de *OffsetFromMaster* do relógio no processo de sincronismo, este pode apresentar um valor negativo. Para se poder registar este tipo de valores, tirou-se proveito do facto de a variável de nanossegundos poder ter o valor máximo de 10^9 – 1 nanossegundos, que pode ser representado usando apenas 30bits, isto possibilita a utilização do bit 31 para

sinalizar se o valor de tempo é negativo (bit com valor igual a um) ou positivo (bit com valor igual a zero). Na Figura 3.6, é apresentado a estrutura de tempo usada e o exemplo do formatado obtido na representação do valor de tempo negativo de 12 segundos e 34 nanossegundos.

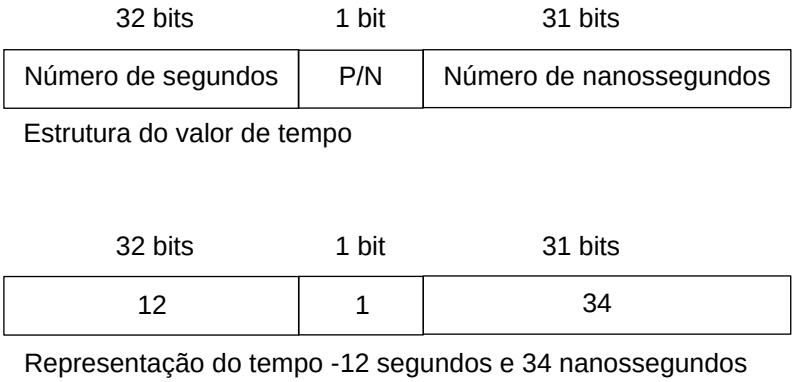

Figura 3.6 – Estrutura do registo do valor de tempo

3.1.2.3 Fluxo de transmissão das mensagens

O algoritmo de sincronismo começa com a unidade que tem o relógio de referência a enviar periodicamente as mensagens *Sync* e *FollowUp* para toda a rede. Nesta implementação, esta unidade corresponde à unidade AP do protocolo de comunicação *SimpliciTI*. Em cada mensagem *Sync* enviada, o AP guardar o tempo de envio da mensagem (t_1) e transmite-o na mensagem *FollowUp* (Figura 3.7).

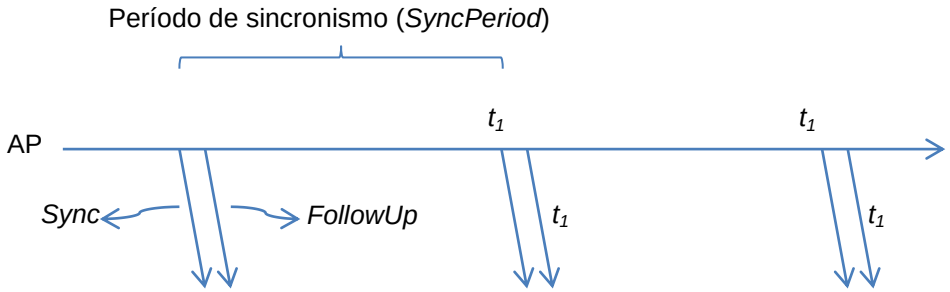

Figura 3.7 – Envio periódico das mensagens *Sync* e *FollowUp*

As unidades que se pretendem sincronizar com o relógio do AP vão corresponder às unidades ED do protocolo de comunicação *SimpliciTI*. Estas unidades depois de se juntarem à rede vão iniciar o processo de sincronismo para obterem o valor do *PathDelay* e de *OffsetFromMaster*.

A unidade ED ao receber a mensagem *Sync* regista o seu tempo de receção (t_2) e o valor de *seqNumbSync* presente na mensagem. De seguida, recebe a mensagem *FollowUp* e

analisa o valor de *seqNumbSync* presente na mensagem. Se for idêntico ao recebido na mensagem *Sync*, o valor de tempo t_1 é considerado valido e é guardado.

Para se obter o valor de *PathDelay* é necessário repetir este processo mas no sentido inverso (Figura 3.8). Para tal, é enviada a mensagem *DelayReq* para o AP e registado o seu tempo de transmissão (t_3). A unidade AP ao receber esta mensagem regista o seu valor de tempo de receção (t_4) e envia-o para a unidade ED correspondente através da mensagem *DelayRsp*, com o valor do campo *seqNumbDelay* idêntico ao da mensagem *DelayReq* recebida.

A unidade ED ao receber a mensagem *DelayRsp*, fica na posse dos quatro tempos necessários para o cálculo do valor do *PathDelay* pela equação (3.1).

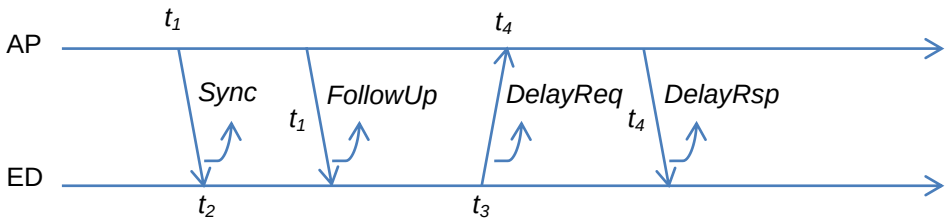

Figura 3.8 – Troca de mensagens de sincronismo para o cálculo de *PathDelay*

$$PathDelay = \frac{(t_2 - t_1) + (t_4 - t_3)}{2} \tag{3.1}$$

Depois do cálculo do valor de *PathDelay*, basta obter o valor de t_1 e t_2 para calcular o valor de *OffsetFromMaster* do relógio pela equação (3.2). Este cálculo é repetido em cada par de mensagens *Sync* e *FollowUp* recebidas (Figura 3.9).

$$OffsetFromMaster = (t_2 - t_1) - PathDelay \tag{3.2}$$

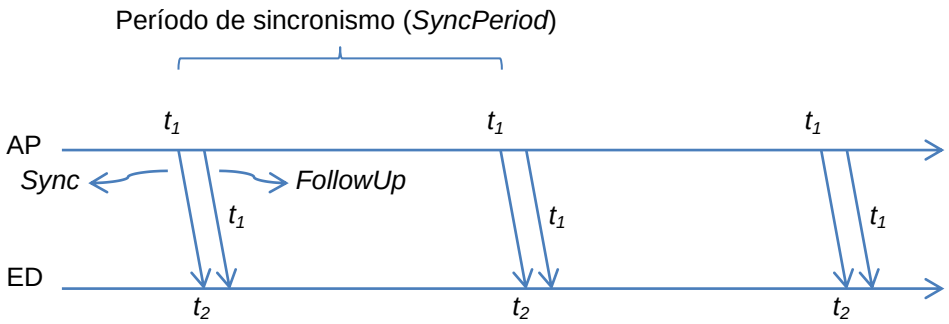

Figura 3.9 – Troca de mensagens de sincronismo para o cálculo de *OffsetFromMaster*

3.1.2.4 Interface do Bloco

Funções de interação

Como todo o processo é executado de forma automática, as funções de interação deste bloco resumem-se a funções de Gestão.

- **Init** – A função de inicialização das variáveis internas do bloco
- **Main** – Esta função deve ser chamada periodicamente para que todo o processo de sincronismo seja realizado.

Sinalizadores Globais

Este bloco tem dois sinalizadores, um para a unidade de referência e outro para a unidade sincronizada

- **SyncPointRXComplete** – Este sinalizador é ativado na unidade sincronizada sempre que o processo de sincronismo é efetuado. Pode ser usado para coordenar ações de baixo consumo da unidade.
- **SyncPointTXComplete** – Usado na unidade de referência para indicar que as mensagens de sincronismo foram enviadas. Tendo em conta o seu perfil periódico, a sua sinalização pode ser usada para coordenar outras transmissões.

3.1.3 Bloco de Gestão de Eventos

Para endereçar aplicações que necessitem de diferentes unidades sincronizadas no tempo, é necessário não só ter a capacidade de manter um relógio local sincronizado, mas também de conseguir gerar eventos síncronos com o relógio e de registar o valor de tempo do momento de ocorrência de eventos.

Esta funcionalidade é conseguida, configurando um *timer* para gerar interrupções com o período de 1 milissegundo. Esta periodicidade vai ser usada para incrementar um relógio local ao mesmo tempo que é gerido o tempo de execução de eventos síncronos.

O valor do relógio é composto por duas variáveis, uma para representar o valor de segundos e outra para o valor de milissegundo. As componentes de microssegundo e nanossegundo do relógio são inferidas diretamente do valor do contador do *timer*.

3.1.3.1 Registo de eventos para execução

O registo de eventos para serem executados pelo bloco é realizado com a chamada de uma função, onde é especificado o valor do identificador do evento, o período, o desvio e a função que é chamada para executar o evento.

O identificador do evento deve ser atribuído previamente e serve para endereçar na tabela de eventos interna, qual a entrada a configurar. O seu valor também serve como nível de prioridade no caso de existirem dois eventos a serem executados ao mesmo tempo, onde o de valor mais pequeno tem maior prioridade.

O período e o desvio permitem definir quando é que o evento vai ser executado. Um exemplo da sua utilização é apresentado na Figura 3.10 com dois casos práticos.

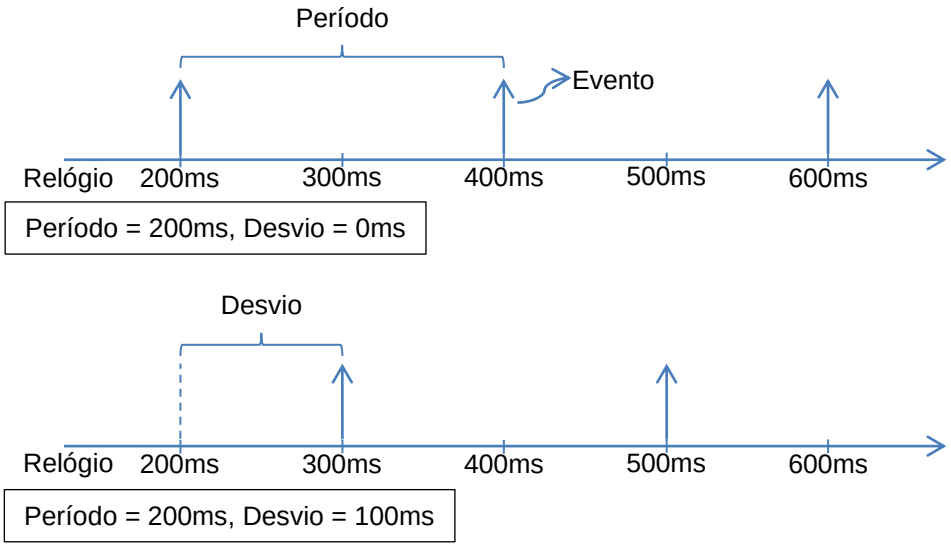

Figura 3.10 – Execução de eventos periódicos

O evento é executado através da chamada à função registada anteriormente para esse evento. Esta função deve ter uma execução rápida, como por exemplo ativar o sinal de uma porta de controlo ou ativar um sinalizador global. Uma execução muito longa desta função pode originar corrupção do relógio, visto que o evento é executado dentro da interrupção.

3.1.3.2 Obtenção do valor do relógio

Quando é pretendido obter o valor do relógio local para ser associado à ocorrência de um evento, como o caso do tempo de transmissão e receção de mensagens, é obtido os valores dos campos segundos e milissegundos da estrutura do relógio local e o valor do contador do *timer*. Estes valores são depois convertidos para o formato normalizado descrito na secção 3.1.2.2 através das seguintes equações.

$$Ts_{Normalizado} = Ts_{Relógio} \tag{3.3}$$

$$Tns_{Normalizado} = Tms_{Relógio} * 10^6 + \frac{Timer_{Counter} \times 10^6}{Timer_{CounterMax}} \tag{3.4}$$

Nesta equação o valor de $Timer_{CounterMax}$ corresponde ao valor do contador para um tempo de 1 milissegundo. O valor de $Timer_{Counter}$ é convertido para um valor em nanossegundos e somado ao valor do campo milissegundo do relógio. Os tempos *Ts*, *Tms* e *Tns* correspondem respectivamente aos valores de segundo, milissegundo e nanossegundo calculados

3.1.3.3 Fluxo de execução do bloco

O fluxo de execução da interrupção do *timer* é apresentado na Figura 3.11 e é composto por três fases. Manutenção do relógio, manutenção dos eventos e correção do relógio.

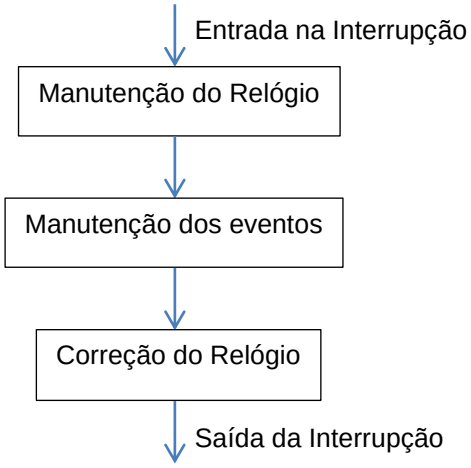

Figura 3.11 – Fluxo de execução da interrupção do Bloco de Gestão de Eventos

Manutenção do Relógio

A primeira ação realizada ao entrar na interrupção é o tratamento do valor do relógio local de forma a garantir um valor correto antes da execução dos eventos.

O valor do campo milissegundos é incrementado. Quando o seu valor chega ao valor máximo de 1000 milissegundos, o campo é posto a zero e o valor do campo segundos é incrementado (Figura 3.12).

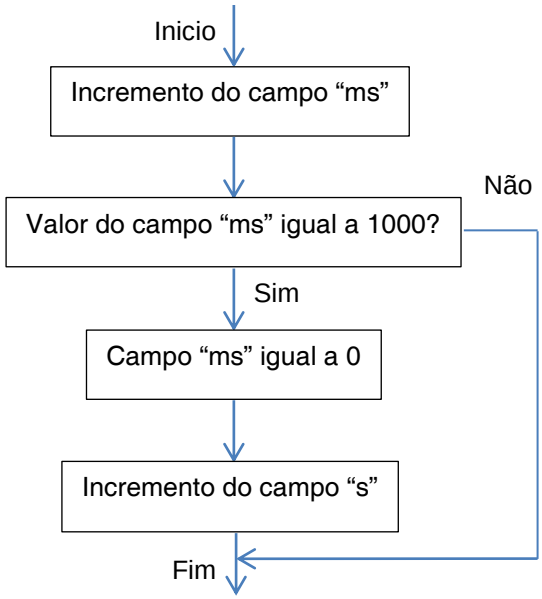

Figura 3.12 – Fluxo da manutenção do relógio

Manutenção dos eventos

Em cada entrada na interrupção do *timer* está associado uma lista de eventos a serem executados. Esta lista é construída na interrupção anterior para que a execução dos eventos possa ser uma das primeiras ações a serem realizadas na interrupção.

A cada evento registado para execução é associado um contador que é decrementado em cada interrupção. Quando chega ao valor 1 sabe-se que na próxima interrupção o evento tem de ser executado, para tal, ele é inserido na lista de eventos a executar na próxima interrupção. Se o seu valor for 0, significa que o evento acabou de ser executado e o valor do seu contador precisa de ser recalculado.

Pode acontecer o caso de um evento ter um período de execução de 1 milissegundo, o que significa que na mesma interrupção em que é executado, este deve ser inserido de novo na lista de eventos a executar na interrupção seguinte. Para este caso ser suportado, a condição do contador ser igual a zero é analisada antes da condição do contador ser igual a um.

O diagrama de funcionamento é apresentado na seguinte figura.

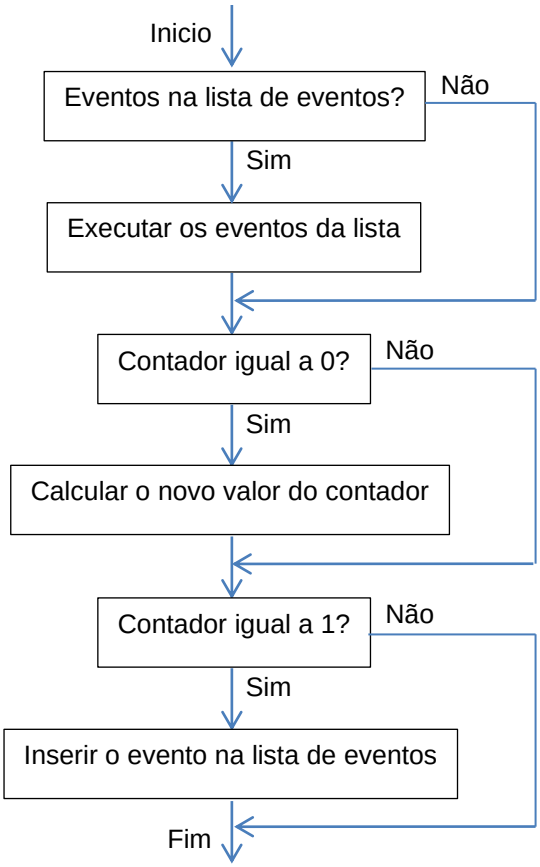

Figura 3.13 – Fluxo da manutenção dos eventos

Correção do Relógio

A correção do relógio é realizada em duas fases. Primeiro o valor de segundos e milissegundos do relógio é corrigido, em seguida é configurado o novo valor máximo do *timer* para compensar a parte de microssegundos e nanossegundos do desvio do relógio.

Este processo permite que no início da próxima interrupção, o relógio da unidade se encontre sincronizado com o relógio de referência.

3.1.3.4 Interface do Bloco

Funções de interação

Neste bloco foram implementadas as seguintes funções de interação.

- **Init** – Função de inicialização do *timer* e da configuração da interrupção periódica de 1 milissegundo.
- **EventRegistration** – Função usada para registar a execução de um evento, como especificado no capítulo 3.1.3.1.
- **GetTime** – Função que permite obter o valor de relógio no momento em que é chamada.

- **CorrectOffset** – Após o valor de desvio de relógio ser calculado pelo protocolo de sincronismo, a sua correção é iniciada por esta função.
- **AddTime** – Permite adicionar dois valores de tempo em formato normalizado.
- **SubTime** – Permite subtrair dois valores de tempo em formato normalizado.
- **DivTime** – Permite dividir um valor de tempo em formato normalizado por um valor inteiro positivo. Juntamente com as duas funções anteriores, esta função vai ser usada no cálculo dos parâmetros de sincronismo.

Sinalizadores Globais

Neste bloco foi definido um sinalizador global.

- **SyncStarted** – este sinalizador é ativo depois de o valor de desvio do relógio ser corrido pelo menos uma vez. Sinalizando que a correção do desvio do relógio foi inicializada.

3.1.4 Concorrência de acesso a recursos

Para que os eventos registados no Bloco de Gestão de Eventos sejam executados de forma síncrona entre todas as unidades, é necessário garantir que a interrupção de receção de mensagens do Bloco RF e que a interrupção do *timer* do Bloco de Gestão de Eventos ocorram sem atrasos. Numa situação em que as duas interrupções sejam criadas no mesmo momento, gera-se um problema de concorrência de acesso a recursos, neste caso, acesso ao processador do microcontrolador.

De forma a minimizar o impacto deste problema foram tomadas as seguintes considerações.

- A interrupção de receção de mensagens deve ser configurada com maior prioridade visto que um atraso na execução de um evento é preferível a ter um relógio dessincronizado, o que levava ao atraso na execução de todos os eventos.
- A parte crítica de execução na interrupção de receção de mensagens é a obtenção do valor de tempo de chegada da mensagem, o que permite estruturar a interrupção numa parte critica e numa parte não crítica de execução.

A solução passou pelo tratamento da receção de mensagens em duas interrupções distintas, uma com a parte crítica do tratamento, com prioridade de execução superior à do *timer* do Bloco de Gestão de Eventos e outra com a parte não crítica com prioridade inferior. Para se implementar esta solução, usou-se a interrupção do *timer* 4 do microcontrolador, visto ser um periférico disponível que não está a ser utilizado (Figura 3.14).

Interrupções	Prioridade
Receção de mensagens	0 (Maior)
Timer do Bloco de Gestão de Eventos	1
Timer 4	2 (Menor)

Figura 3.14 – Níveis de prioridade das interrupções

Quando é recebida uma mensagem, a interrupção do Bloco RF é gerada. Depois de se obter o valor do tempo de receção da mensagem, a interrupção do *timer* 4 é gerada por *software* pondo o seu sinalizador de interrupção a 1. Ao sair da interrupção do Bloco RF, se a interrupção do Bloco de Gestão de Eventos estiver pendente, esta é executada. Caso não seja esse o caso ou a sua execução tenha terminado, a interrupção do *timer* 4 é servida para processar a mensagem recebida.

3.2 Validação inicial do sistema

Para validar o funcionamento do sistema implementado, foi montado um ensaio composto por 3 tipos de unidades. Uma unidade AP com o relógio de referência, uma unidade ED que se vai sincronizar com o relógio da unidade AP depois de estabelecer conectividade e um PC (Figura 3.15).

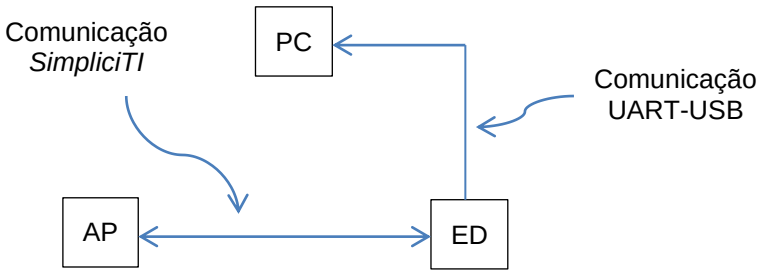

Figura 3.15 – Estrutura do ensaio

Depois da unidade ED se conectar com a unidade AP, o processo de sincronismo é desencadeado de forma automática. Por cada ponto de troca de mensagens de sincronismo, o valor de *OffsetFromMaster*, definido no capítulo 3.1.2, é calculado e transmitido para o PC através de um cabo conversor UART-USB.

O ensaio foi repetido para os períodos de sincronismo de 1, 2, 5, 10, 30 e 60 segundos onde foram registados 100 valores de *OffsetFromMaster*.

Na figura seguinte são apresentados os resultados obtidos.

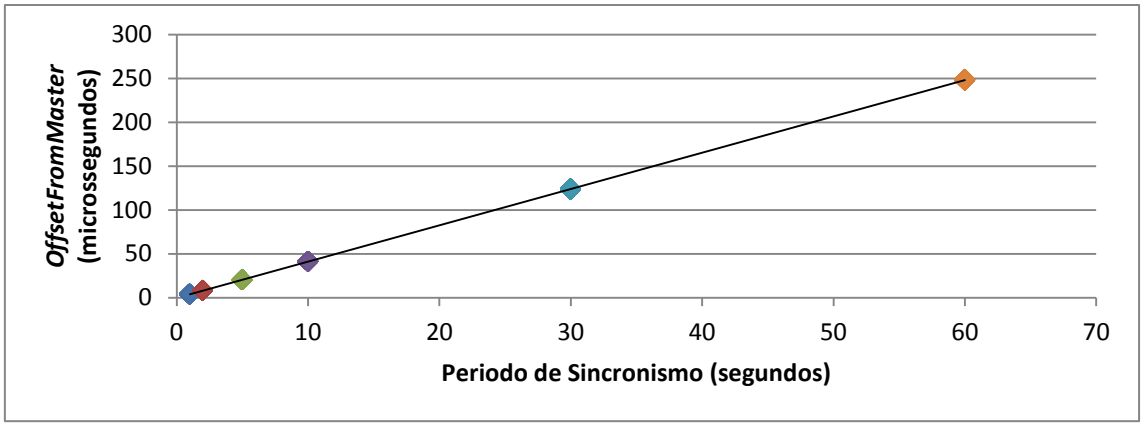

Figura 3.16 – Distribuição dos valores de *OffsetFromMaster*

Como esperado, existe uma dependência diretamente proporcional entre o período de sincronismo e o valor de *OffsetFromMaster*. Esta dependência pode ser expressa pela equação (3.5).

$$OffsetFromMaster = SyncPeriod \times Desvio\ de\ Frequencia \quad (3.5)$$

A partir desta equação, verifica-se que dividindo os valores de *OffsetFromMaster* obtidos, pelo seu período de sincronismo, podemos estimar o desvio de frequência entre os dois relógios. Na Figura 3.17 é apresentado os resultados desta divisão.

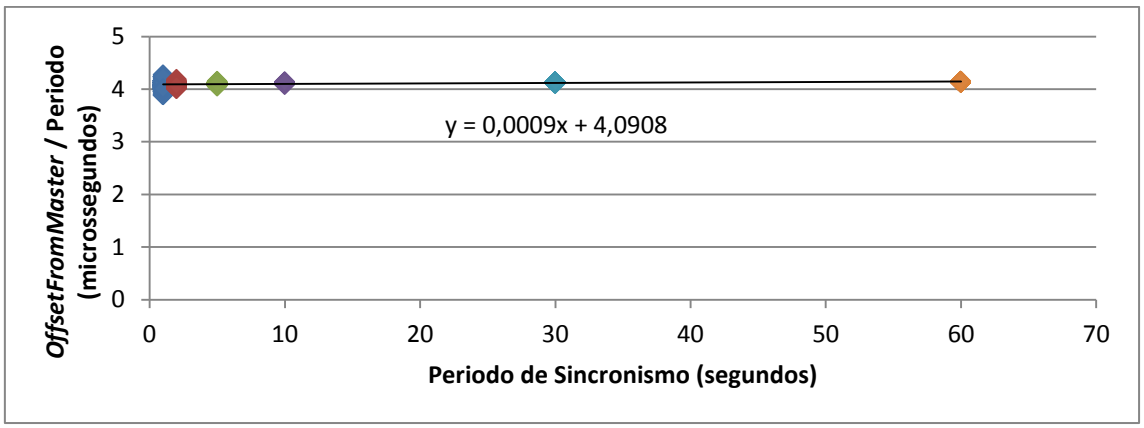

Figura 3.17 – Valores de desvio de frequência entre os relógios

Através deste gráfico podemos apurar que o relógio da unidade ED apresenta em relação ao relógio da unidade AP, um desvio na frequência corresponde-te a um atraso de 4,09 microssegundos por cada segundo.

3.3 Sintonização de relógios

No capítulo anterior foram apresentados resultados que se obtêm quando se implementa um sistema de sincronização de relógios baseado somente no protocolo de sincronismo. Por muito preciso que o valor de *OffsetFromMaster* seja calculado, continua-se sempre a ter o problema de os seus relógios serem gerados por cristais diferentes, o que faz que ao longo do tempo o valor de *OffsetFromMaster* aumente.

A solução tipicamente usada para resolver este problema passa por usar um cristal mais preciso o que normalmente encarece a solução, ou reduzir o período de sincronismo o que aumenta o consumo da unidade.

A abordagem escolhida neste trabalho foi a de sintonizar os relógios de forma a reduzir o seu desvio ao longo do tempo. Idealmente a solução passaria por de alguma forma corrigir diretamente a frequência do cristal. Como tal não é possível, foi implementado um algoritmo de sintonização baseado na divisão do valor de *OffsetFromMaster*, em correções mais pequenas que são realizadas periodicamente ao longo do tempo (Figura 3.18). Desta forma, em vez de se efetuar uma única correção de valor *OffsetFromMaster*, passam a ser realizadas múltiplas correções de valor mais baixo, conseguindo-se assim reduzir o erro máximo de sincronização.

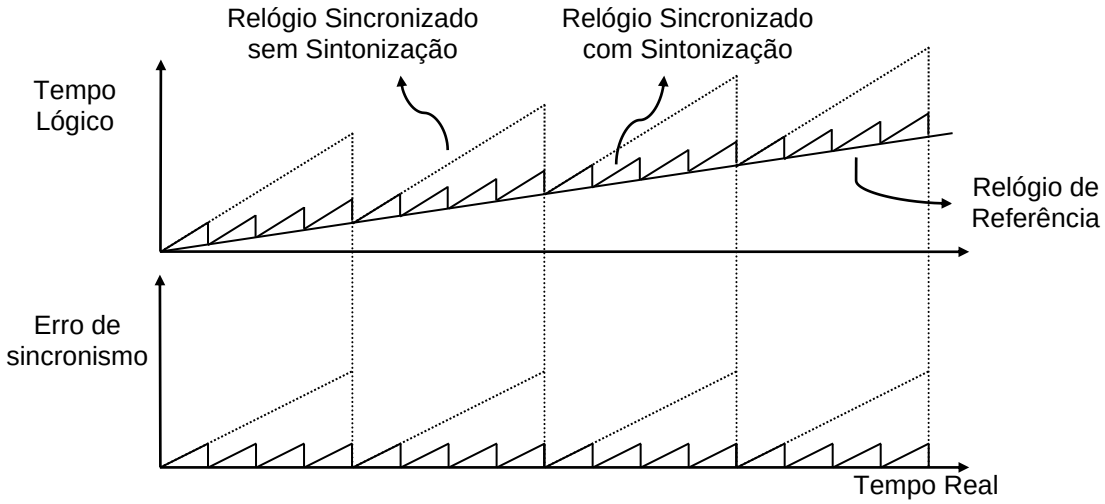

Figura 3.18 – Impacto da sintonização do relógio

Para implementar o algoritmo de sintonização do relógio das unidades, foi adicionado uma quarta fase à interrupção do *timer* do Bloco de Gestão de Eventos (Figura 3.19).

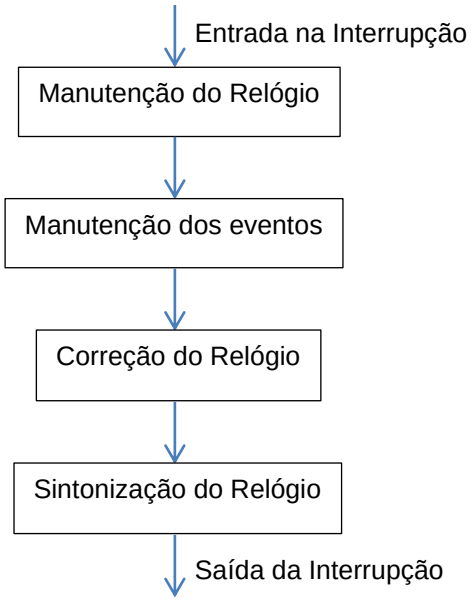

Figura 3.19 – Fluxo de execução da interrupção do Bloco de Gestão de Eventos com sintonização

A fase de sintonização do relógio começa com o cálculo do número de correções (*NSyntCorr*) pela equação (3.6), que necessitam de ser realizadas para compensar o relógio.

$$NSyntCorr = \frac{OffsetFromMaster}{SyntCorrValue} \tag{3.6}$$

Nesta equação o valor de *SyntCorrValue* corresponde ao valor de correção feito no relógio em cada ponto de sintonização e o *OffsetFromMaster* ao valor de correção de relógio realizado no ponto de sincronização anterior. O valor de *SyntCorrValue* é fixo e é definido anteriormente ao início do processo.

Sabendo o número de correções de sintonização necessárias, calcula-se o período de sintonização (*SyntPeriod*) pela equação (3.7). Com este valor calculado, inicia-se o processo de correção de sintonização do relógio.

$$SyntPeriod = \frac{SyncPeriod}{NSyntCorr} \tag{3.7}$$

Depois do algoritmo ser iniciado, é necessário continuar a ajustar o número de correções de sintonização para compensar os devidos de frequência do cristal ao longo do tempo. O recalculo do número de correções já não pode ser feito pela equação anterior porque o valor de *OffsetFromMaster* encontra-se agora compensado pelas correções de sintonização anteriores. Neste caso é usado um processo interativo onde o valor de *OffsetFromMaster* é analisado e o número de correções de sintonização é incrementado ou decrementado consoante a

polaridade das correções e do valor *OffsetFromMaster*. Na Figura 3.20 é apresentado o seu funcionamento.

Ao entrar nesta fase, começa-se por calcular o valor de incremento/decremento de correções a serem realizadas (*NSyntCorrInc*). Este valor é definido por patamares e é atribuído consoante o valor de *OffsetFromMaster*, onde pode ter o valor de 1, 10 ou 100 correções. De seguida é analisado a polaridade das correções (*SyntNegSig*) e tomado a decisão de incrementar ou decrementar o valor de *NSyntCorrInc* ao valor de *NSyntCorr* calculado anteriormente. Numa situação em que o valor *NSyntCorrInc* é superior a *NSyntCorr* e é necessário decrementar o número de correções, temos um caso de troca de polaridade das correções e o valor de *SyntNegSig* e *NSyntCorr* deve ser ajustado corretamente.

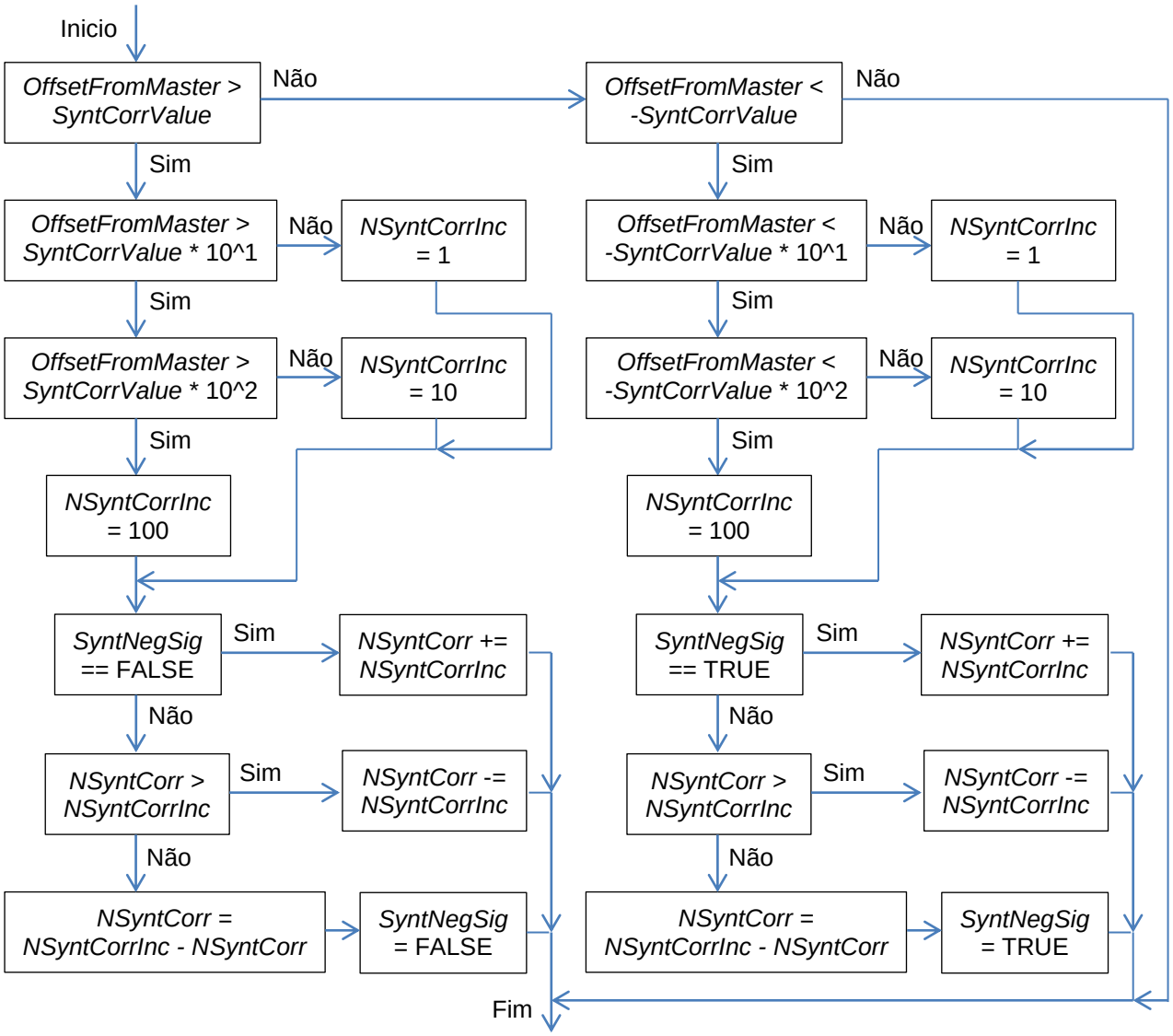

Figura 3.20 – Fluxo do cálculo do número de correções de sintonização

3.4 Validação final do sistema

Para validar o sistema agora com o algoritmo de sintonização, foi criado um novo ensaio semelhante ao usado no capítulo 3.2. Neste caso foi usado adicionalmente um gerador de pulsos (Figura 3.21).

Semelhante ao caso anterior. Depois da unidade ED estabelecer conectividade com a unidade AP, o processo de sincronização e de sintonização é inicializado automaticamente. Por outro lado vamos ter o gerador de pulsos, ligado a uma porta GPIO da unidade AP e da unidade ED, a gerar pulsos periódicos. Por cada pulso criado, ambas as unidades vão registar o seu valor de relógio no momento e transmitem-no para o PC através de um cabo conversor *UART-USB*. No lado do PC, os valores de tempo são recebidos e guardados para posterior tratamento.

O teste foi repetido para os períodos de sincronismo de 5 e 30 segundos com um *SyntCorrValue* de 1 microssegundo. O gerador de pulsos foi configurado para gerar um pulso a cada 100 milissegundos. Em cada teste foram obtidas 2000 amostras.

Nas secções seguintes serão apresentados os valores obtidos em ambos os testes.

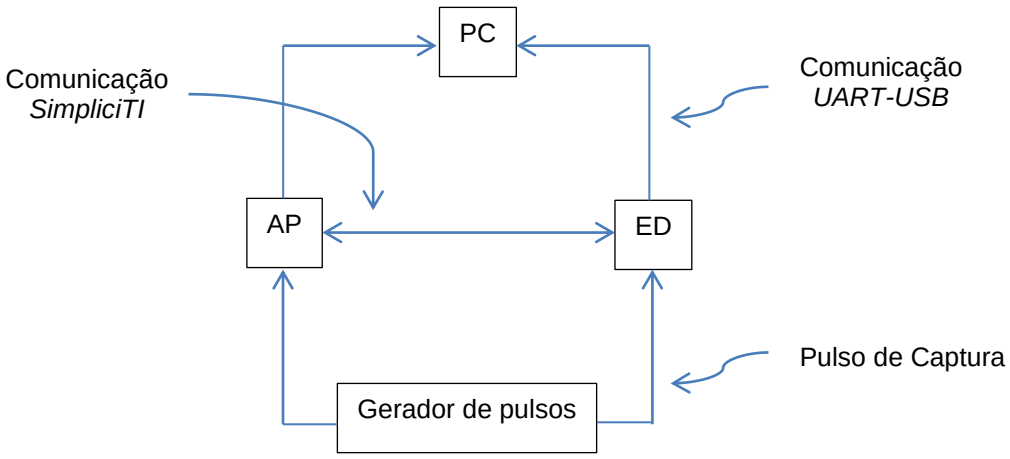

Figura 3.21 – Estrutura do ensaio

3.4.1 Teste com *SyncPeriod* de 5 segundos

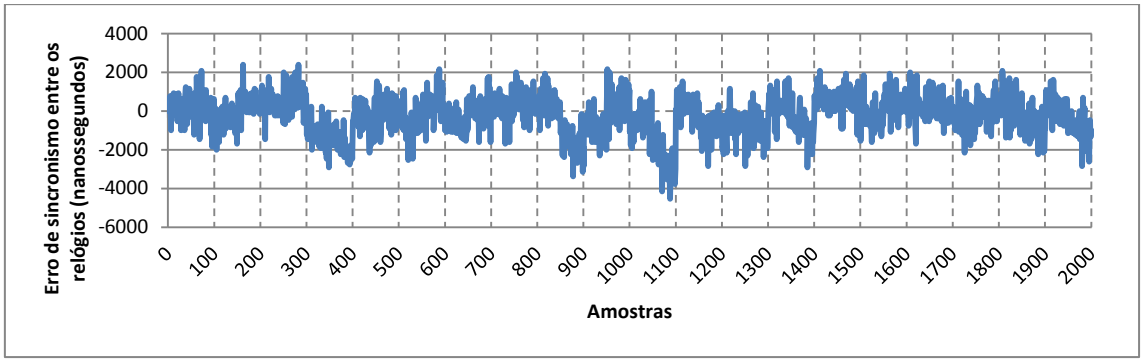

Figura 3.22 – Erro de sincronismo para um *SyncPeriod* de 5 segundos

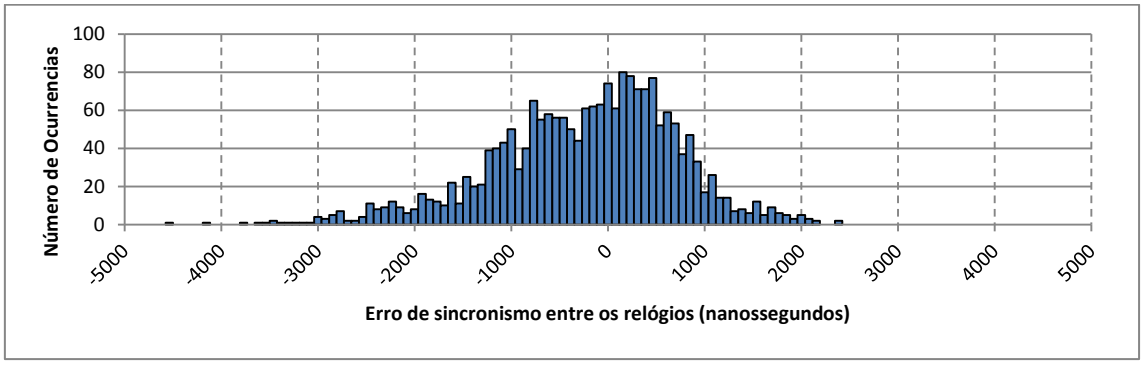

Figura 3.23 – Distribuição do erro de sincronismo para um *SyncPeriod* de 5 segundos

3.4.2 Teste com *SyncPeriod* de 30 segundos

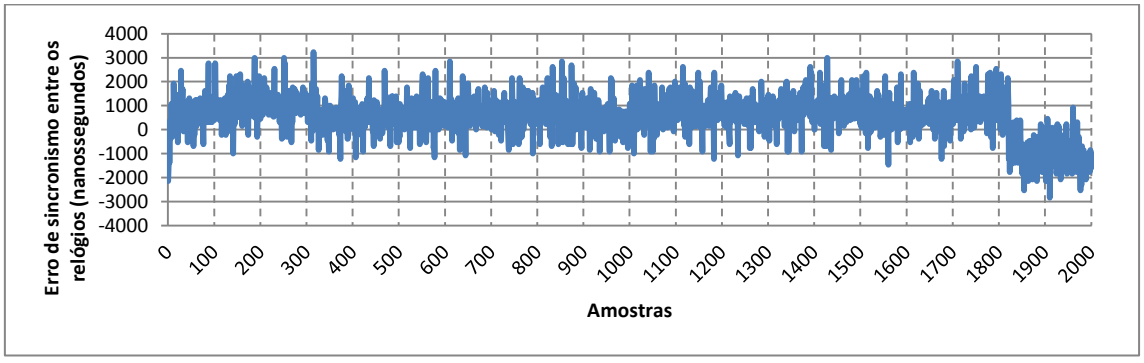

Figura 3.24 – Erro de sincronismo para um *SyncPeriod* de 30 segundos

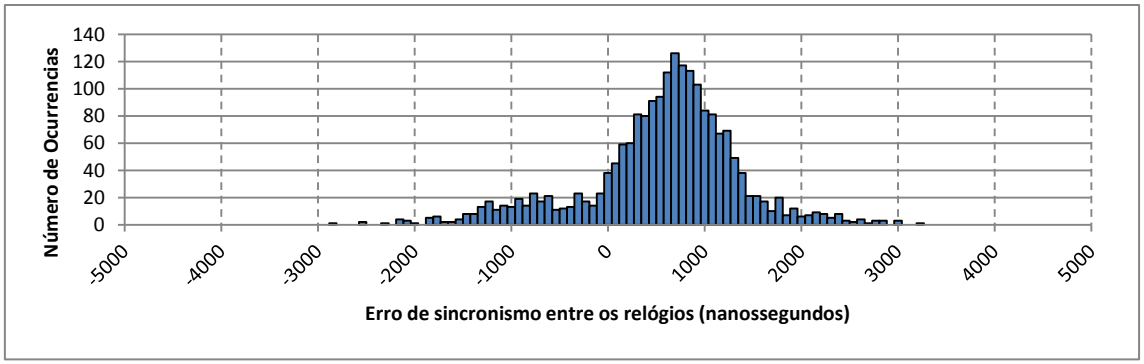

Figura 3.25 – Distribuição do erro de sincronismo para um *SyncPeriod* de 30 segundos

3.4.3 Resultados Finais

Tabela 3.1 – Resultados dos testes efetuados

Período de Sincronismo	Média	Desvio Padrão	Minimo	Maximo
5 segundos	-0,235 us	0,957 us	-4,538 us	2,385 us
30 segundos	0,563 us	0,798 us	-2,846 us	3,231 us

Com estes resultados, pode ser facilmente observado que o erro de relógio deixou de aumentar linearmente com o tempo e passou a estabilizar com um erro máximo e mínimo inferior a 5 microssegundos. Isto comprova que a utilização de um algoritmo de sintonização como suporte a um protocolo de sincronização é de extrema importância de forma a estabilizar o relógio e a permitir reduzir o número de mensagens trocadas por segundo para esse fim.

3.5 Reconhecimento

O trabalho desenvolvido neste primeiro protótipo recebeu o seu reconhecimento internacional com a publicação do *paper* intitulado "*Synchronization and Syntonization of Wireless Sensor Networks*" [16] na conferência *Wireless Sensors and Sensor Networks* (*WiSNet*) de 2013. Esta conferência fez parte de um conjunto de conferências albergadas pelo evento *Radio and Wireless Week* (*RWW*) de 2013 [17] que teve a sua organização em Austin, capital do estado do Texas dos Estados Unidos da América. Em anexo pode ser consultado o *paper* aqui referido.

Este trabalho serviu também de suporte ao desenvolvimento de uma aplicação para localização *indoor* baseada em ultrassons (Figura 3.26). Nesta aplicação, foi montado uma rede composta por quatro unidades, uma unidade AP com o relógio de referência e 3 unidades EDs. Depois de a rede ser estabelecida e das unidades EDs terem sincronizado e sintonizado os seus relógios, são registados eventos periódicos que são executados sincronizadamente entre as 4 unidades. Na execução de cada um desses eventos, é gerado um pulso numa porta GPIO que é fornecido à placa responsável pelo algoritmo de localização.

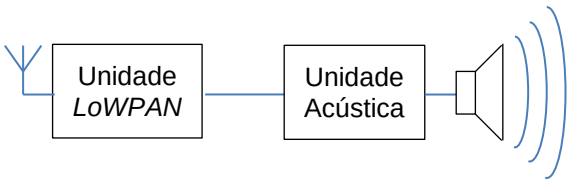

Figura 3.26 – Unidade *WAPoS*

4 Segundo Protótipo

No protótipo anterior, foi implementado um protocolo de sincronismo baseado no *standard* IEEE 1588 constituído pela troca de 4 tipos de mensagens. Mostrou-se que o suporte de um algoritmo de sintonização é de extrema importância na redução do erro entre os relógios, permitindo ter períodos de troca de mensagens de sincronismo de 30 segundos ao mesmo tempo que se obteve erros de precisão de alguns microssegundos.

Apesar de ter sido utilizado a frequência de relógio de 13MHz, que é a configuração presente por defeito no código de desenvolvimento do protoloco de comunicação *SimpliciTI*, o *SoC CC1110* usado na placa *NaPIS*, permite que seja configurado para utilizar diretamente a frequência do cristal de 26MHz. Esta frequência corresponde a uma resolução de relógio de aproximadamente de 38 nanossegundos. Considerando que o erro máximo de 4.5 microssegundos, atingido nos resultados da validação do protótipo, corresponde a mais de 100 vezes o valor de um pulso de relógio, podemos assumir que melhorias podem ser feitas de forma a obter melhores resultados com o mesmo *hardware*.

Neste segundo protótipo, foi realizado uma análise dos principais pontos de geração de erro no processo de sincronização de relógios, que resultou num conjunto de otimizações descritas e implementadas neste capítulo.

Neste capítulo são descritos os vários pontos geradores de erro, as soluções encontradas para minimizar os seus impactos e os resultados obtidos.

4.1 Otimizações

Um das principais fontes de erro encontra-se no facto do registo do tempo de ocorrência de eventos ser realizado por *software* com a chamada de uma função. Isto levanta problemas devido ao processador do *SoC CC1110* ser um 8051 de 8 bits. Este processador é baseado na arquitetura *CISC* (*Complex Instruction Set Computer*) onde o tempo de execução de cada instrução é variável. Este chip em específico apresenta uma variação que pode ir de um valor mínimo de um ciclo de relógio até um valor máximo de 11 ciclos de relógio. Considerando que na implementação anterior era usada uma frequência de relógio de 13MHz, esta variação corresponde a um tempo de 77 nanossegundos para um ciclo e 846 nanossegundos para 11 ciclos. O registo do tempo de receção de mensagens é realizado dentro de uma interrupção, que antes de poder ser servida, o processador tem de finalizar atomicamente a instrução que está a executar no momento, gerando um atraso aleatório no processo do cálculo dos parâmetros do sistema de sincronismo. Este problema também se reflete na execução de eventos síncronos, visto que são executados dentro da interrupção do Bloco de Gestão de Eventos.

Outro local onde é possível obter melhorias é no algoritmo de sintonização. Diminuindo o seu valor de correção periódico (*SyntCorrValue*), é possível obter um menor erro de relógio, que por outro lado vai obrigar ao aumento do número de correções efetuadas por segundo.

Por último, temos o próprio protocolo de sincronismo, onde o valor dos parâmetros *PathDelay* e *OffsetFromMaster* é calculado usando um único ponto de troca de mensagens. Esta característica leva a que erros na obtenção dos tempos usados nos seus cálculos se reflitam num erro constante de sincronismo no caso do parâmetro *PathDelay* e num ajuste errado do número de correções do algoritmo de sintonização no caso do parâmetro *OffsetFromMaster*. Outra limitação é em termos de robustez, visto que se uma das mensagens de sincronismo não for recebida, esse ponto de sincronismo é considerado inválido e a unidade só pode voltar a tentar-se sincronizar mais tarde, fazendo com que o erro acumulado até ao momento não seja corrigido.

De forma a facilitar a análise dos resultados obtidos com as diferentes otimizações realizadas, todos os resultados são acompanhados por uma tabela composta pelos valores de

média, desvio padrão, valor mínimo e valor máximo, expressos em nanossegundos e em pulsos de relógio (*ticks*).

4.1.1 Registo do tempo de receção e transmissão de mensagens

O registo do tempo de receção e transmissão de mensagens é uma das operações críticas para o funcionamento de qualquer protocolo de sincronismo, independentemente do tipo de mensagens enviadas e dos parâmetros calculados. Este processo deve ser feito de uma forma automática e o mais precisa possível para garantir a qualidade dos parâmetros de sincronismo obtidos.

Num estudo mais detalhado do *datasheet* do *SoC CC1110* [18], foi descoberta a capacidade do controlador gerar uma sinalização em dois pontos distintos da troca de mensagens (Figura 4.1). Uma após o campo *StartOfFrame* da mensagem (sinalizador SFD) e outra no final da mensagem (sinalizador DONE). O ideal seria usar a sinalização do *StartOfFrame* como é recomendado na documentação do *standard* IEEE 1588, mas em testes realizados, descobriu-se que esta sinalização só é gerada na receção de mensagens e não na sua transmissão. Este facto leva a optar pela sinalização de mensagem recebida ou transmitida.

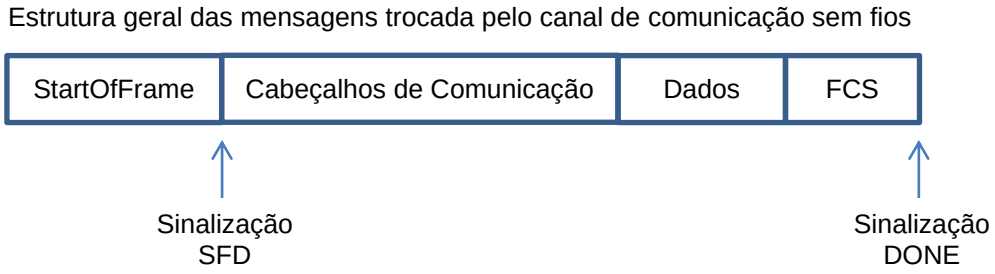

Figura 4.1 – Localização dos sinalizadores associados à troca de mensagens

Esta sinalização por sua vez pode ser usada para gerar uma captura do valor do *timer* do Bloco de Gestão de Eventos no seu canal 2. Com esta funcionalidade configurada, é possível depois da transmissão de uma mensagem e no inicio da interrupção de receção de mensagens, guardar nas correspondente variáveis *TimeStampTX* e *TimeStampRX*, os valores de tempo registados automaticamente pelo *capture* do *timer* do Bloco de Gestão de Eventos.

Neste modo de captura, deixa de ser necessário a configuração da interrupção de receção de mensagens em dois níveis de prioridade como descrito no capítulo 3.1.4, visto que a parte crítica da interrupção, correspondente ao registo do tempo de chegada das mensagens, deixa de entrar em concorrência de recursos com a interrupção do *timer* do Bloco de Gestão de

Eventos, podendo as duas ações ser executadas em paralelo. As modificações ao código realizadas com este fim foram regredidas para o seu formato original.

A validação do registo de tempo diretamente por *hardware* apresentada neste subcapítulo, é exibida na secção seguinte.

4.1.2 Execução e registo temporal de eventos críticos

Uma característica genérica dos *timers* implementados na maioria dos microcontroladores é a funcionalidade de *capture*/*compare* associada a portas de GPIO. Consoante o sentido em que é configurado, podemos usar a funcionalidade de *capture* para registar automaticamente o valor do *timer* ou a funcionalidade de *compare* para ativar, desativar ou mudar a polaridade do sinal da porta GPIO associada. Normalmente um *timer* é composto por múltiplos canais de *capture/compare* e no caso do *timer* usado no Bloco de Gestão de Eventos, temos disponíveis 3 canais denominados de canal 0, 1 e 2.

Na configuração da interrupção periódica do *timer* do Bloco de Gestão de Eventos, foi necessário utilizar o canal 0 com o valor 26000. Este valor corresponde ao valor máximo que o contador do *timer* vai registar, antes de gerar uma interrupção e de limpar o seu conteúdo, que para uma frequência de relógio de 26MHz, corresponde a um período de interrupção de 1 milissegundo.

No subcapítulo anterior, foram apresentadas as modificações feitas no processo de geração do tempo de receção e transmissão de mensagens, para este registo foi usado o canal 2 do *timer* do Bloco de gestão de Eventos.

Resta por fim o canal 1, que pode ser usado para registar automaticamente o tempo de ocorrência de um evento externo ou para desencadear atomicamente a execução de um evento externo associado à porta de *capture/compare* do canal.

Para acomodar este suporte, foram feitas algumas alterações ao Bloco de Gestão de Eventos de forma a suportar o registo de 2 tipos de eventos a serem executados. O primeiro tipo corresponde a eventos não críticos que são executados durante a interrupção do *timer* do Bloco de Gestão de Eventos, similar ao implementado no protótipo anterior. O segundo tipo corresponde à execução de eventos críticos através do *compare* do canal 1. No momento da sua execução, a porta associada ao canal 1 é colocada com o valor lógico 1 para que um evento externo seja desencadeado sem atrasos associados à execução de código no microcontrolador.

Foi também adicionada a capacidade de registo do relógio, ao configurar a porta associada ao canal 1 como *capture*. Esta funcionalidade foi usada em grande parte dos processos de validação do segundo protótipo.

Na validação das modificações realizadas, foi repetido o ensaio descrito no capítulo 3.4 com 4 unidades, uma como unidade AP com relógio de referência, uma unidade ED com o relógio a ser sincronizado, um gerador de pulsos e um PC. Neste teste a porta usada pelo gerador de pulsos para sinalizar as unidades AP e ED foi a porta correspondente ao capture 1 do *timer* do Bloco de Gestão de Eventos. O gerador de pulsos foi configurado para um período de 20 milissegundos e o teste foi repetido para os períodos de sincronismo de 1, 2, 5, 10, 30 e 60 segundos. Como valor de correção de sintonização foi usado o valor 1 microssegundo, idêntico ao usado na validação final do primeiro protótipo. Em cada teste foram obtidas 100000 valores, que correspondem a aproximadamente meia hora de amostragem.

Nos subcapítulos seguintes são apresentados os resultados obtidos, compostos pelo gráfico dos valores de erro de sincronismo entre as duas unidades com o número de amostras correspondentes a 20 pontos de sincronismo, o histograma da distribuição do erro para as 100000 amostras e a tabela com os valores de média, desvio padrão, valor mínimo e valor máximo do erro. Por fim é apresentado uma análise dos resultados obtidos.

4.1.2.1 Teste com SyncPeriod de 1 segundo

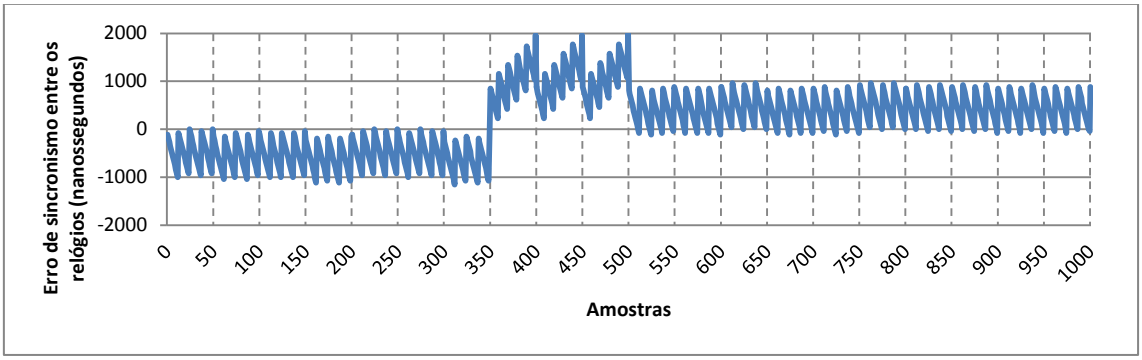

Figura 4.2 – Erro de sincronismo para um *SyncPeriod* de 1 segundo

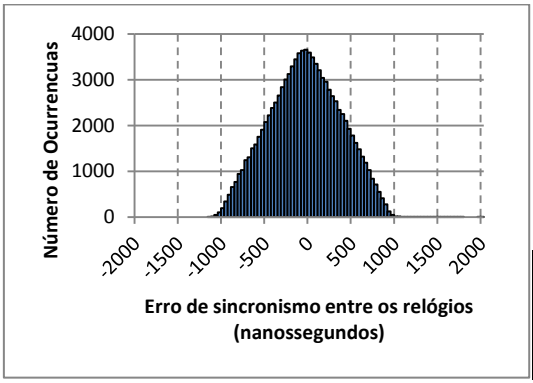

Tabela 4.1 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
-43.05 ns	413.88 ns	-1154 ns	2000 ns
-1.12 ticks	10.76 ticks	-30 ticks	52 ticks

Figura 4.3 – Distribuição do erro de sincronismo para um *SyncPeriod* de 1 segundo

4.1.2.2 Teste com SyncPeriod de 2 segundos

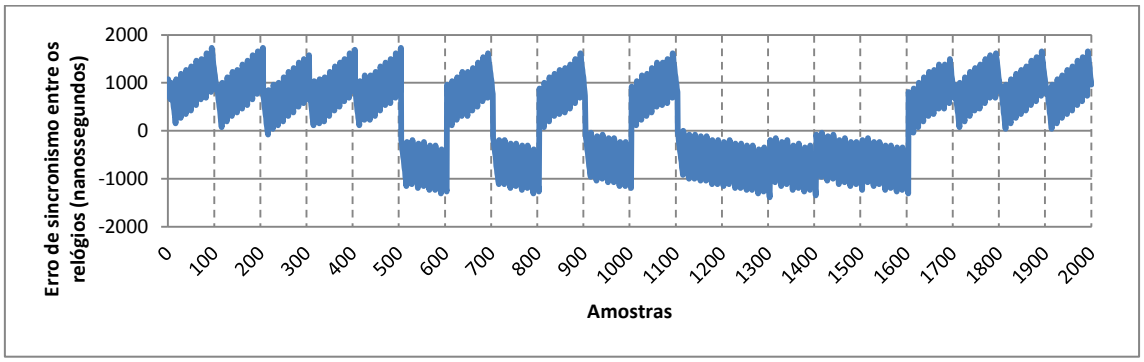

Figura 4.4 – Erro de sincronismo para um *SyncPeriod* de 2 segundos

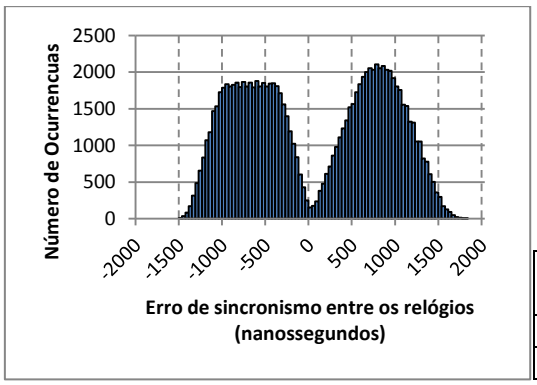

Tabela 4.2 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
52.42 ns	815.73 ns	-1500 ns	1808 ns
1.36 ticks	21.21 ticks	-39 ticks	47 ticks

Figura 4.5 – Distribuição do erro de sincronismo para um *SyncPeriod* de 2 segundos

4.1.2.3 Teste com SyncPeriod de 5 segundos

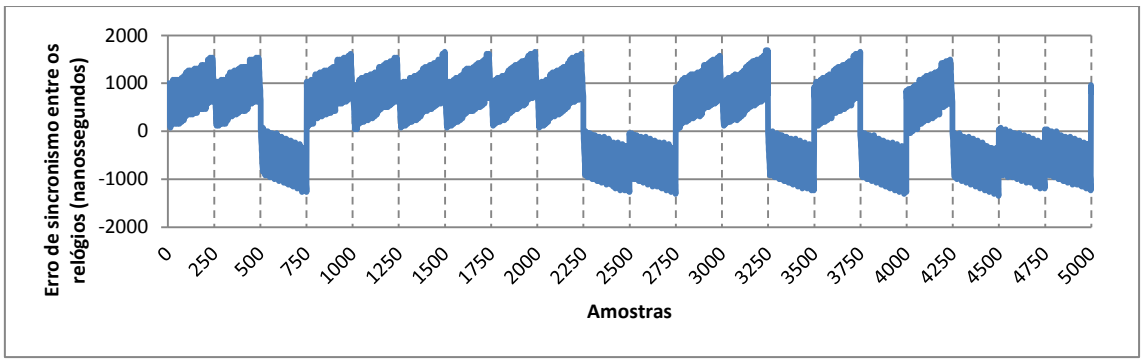

Figura 4.6 – Erro de sincronismo para um *SyncPeriod* de 5 segundos

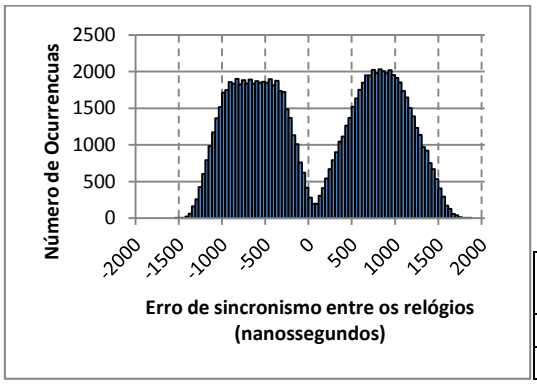

Tabela 4.3 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
-95.93 ns	811.98 ns	-1538 ns	1846 ns
-2.49 ticks	21.11 ticks	-40 ticks	48 ticks

Figura 4.7 – Distribuição do erro de sincronismo para um *SyncPeriod* de 5 segundos

4.1.2.4 Teste com SyncPeriod de 10 segundos

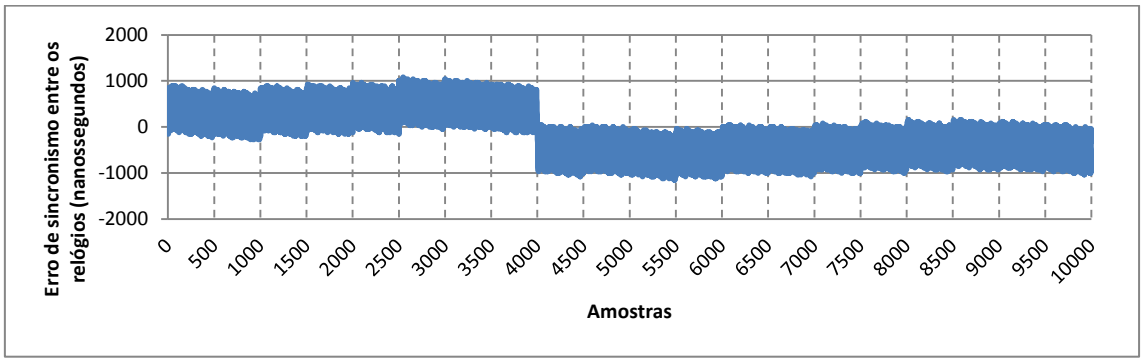

Figura 4.8 – Erro de sincronismo para um *SyncPeriod* de 10 segundos

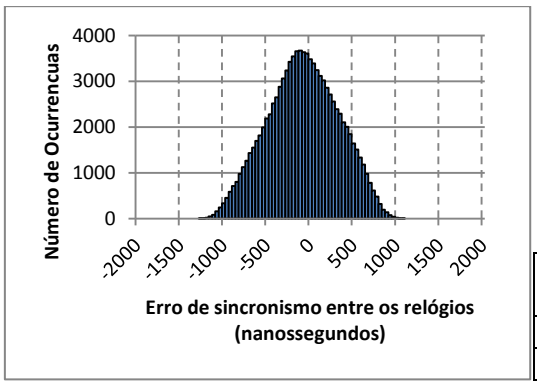

Tabela 4.4 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
-79.66 ns	408.23 ns	-1269 ns	1077 ns
-2.07 ticks	10.61 ticks	-33 ticks	28 ticks

Figura 4.9 – Distribuição do erro de sincronismo para um *SyncPeriod* de 10 segundos

4.1.2.5 Teste com SyncPeriod de 30 segundos

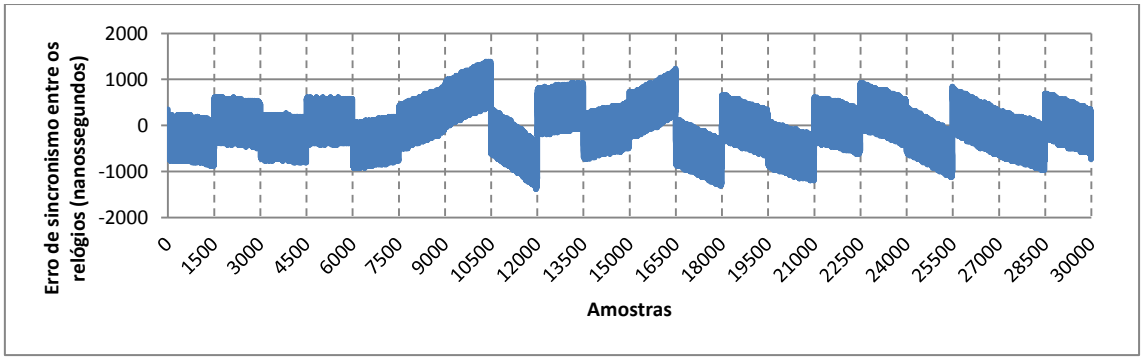

Figura 4.10 – Erro de sincronismo para um *SyncPeriod* de 30 segundos

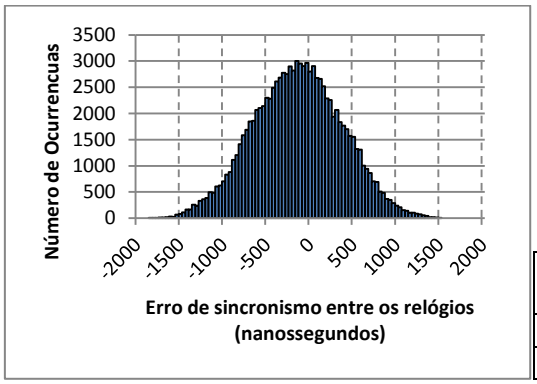

Tabela 4.5 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
-144.11 ns	509.70 ns	-1846 ns	1500 ns
-3.75 ticks	13.25 ticks	-48 ticks	39 ticks

Figura 4.11 – Distribuição do erro de sincronismo para um *SyncPeriod* de 30 segundos

4.1.2.6 Teste com SyncPeriod de 60 segundos

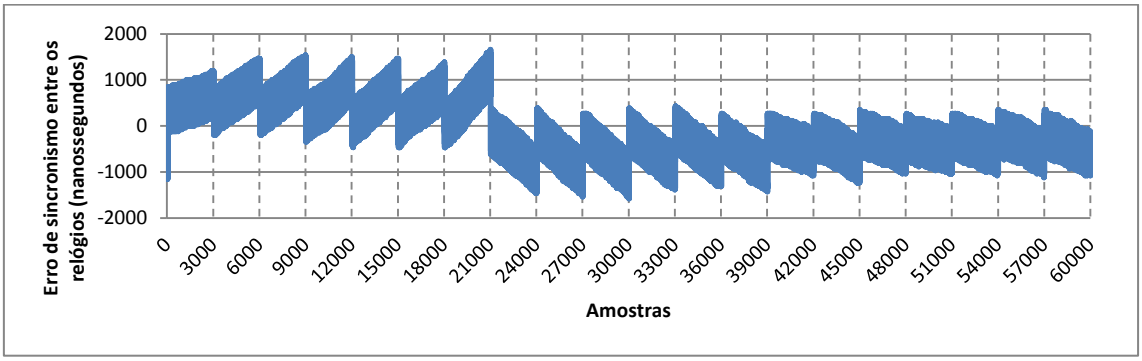

Figura 4.12 – Erro de sincronismo para um *SyncPeriod* de 60 segundos

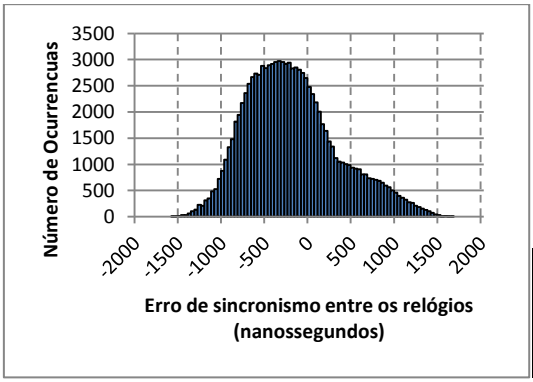

Tabela 4.6 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
-191.02 ns	535.85 ns	-1577 ns	1654 ns
-4.97 ticks	13.93 ticks	-41 ticks	43 ticks

Figura 4.13 – Distribuição do erro de sincronismo para um *SyncPeriod* de 60 segundos

4.1.2.7 Análise de Resultados

No caso dos testes com períodos de 2 e 5 segundos, o valor do desvio padrão apresenta quase o dobro do valor em relação aos restantes testes porque o algoritmo de sintonização não conseguiu estabilizar o seu número de correções por segundo. Consequentemente são gerados dois máximos ao contrário dos restantes testes, onde só é observado um único máximo.

Nos seis testes realizados, o valor máximo de erro encontra-se no teste com período de 1 segundo com um erro de 2 microssegundos. Pela Figura 4.2, verifica-se que a causa deste erro foi um incorreto ajuste do número de correções de sintonização, devido a erros na obtenção do valor *OffsetFromMaster*.

De seguida temos os erros nos testes com período de 2 e 5 segundo com um erro de aproximadamente 1.8 microssegundos. A origem deste erro encontra-se como explicado anteriormente, no facto de o algoritmo de sintonização não ter estabilizado no seu número de correções.

No caso do teste com período de 30 e 60 segundos, apresenta-se um erro máximo de 1.8 e 1.6 microssegundos. O grande responsável por este erro foi o longo período de sincronismo usado, o que torna a estabilidade do relógio suscetível ao erro de cálculo dos

parâmetros de sincronismo e à própria estabilidade do cristal em função da variação da temperatura.

Por último temos o teste com período de 10 segundos que apresenta o melhor erro máximo de 1.2 microssegundos.

Comparando os resultados com os obtidos no protótipo anterior, passamos de um erro máximo de 4.5 microssegundos para um erro máximo de 2 microssegundos, reduzindo o seu valor para metade. Em relação ao desvio dos pontos, os seus valores máximos são similares devido ao teste com período de 2 e 5 segundos.

4.1.3 Utilização de correções de sintonização finas (*SyntCorrValue*)

Com o registo dos tempos de receção e transmissão de mensagens por *hardware* conseguiu-se melhorar o erro de sincronismo entre as unidades, limitando a um erro máximo entre 1 e 2 microssegundos. Este erro encontra-se agora em grande parte limitado pelo valor de 1 microssegundo de correção usado no algoritmo de sintonização. Como tal, o próximo passo lógico na otimização do sistema passa pela utilização de valores de correção inferiores. Uma nova série de teste foram realizados usando o ensaio do capítulo 4.1.2 para valores de correção de 1000 nanossegundos (26 pulsos de relógio), 500 nanossegundos (13 pulsos de relógio), 192 nanossegundos (5 pulsos de relógio), 77 nanossegundos (2 pulsos de relógio) e 38 nanossegundos (1 pulso de relógio). Os testes foram efetuados com um período de sincronismo de 1 segundo.

Nos seguintes subcapítulos são apresentados os resultados e por fim a sua análise.

4.1.3.1 Teste com SyntCorrValue de 1000 nanossegundos (26 Pulsos)

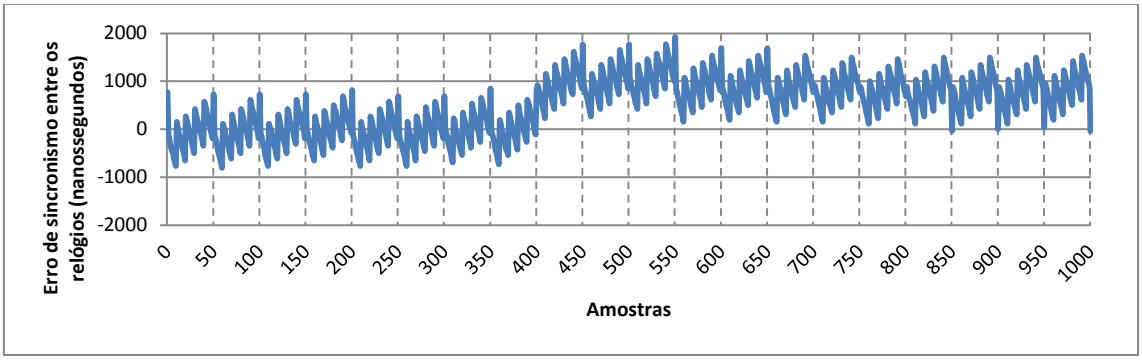

Figura 4.14 – Erro de sincronismo para um *SyntCorrValue* de 1000 nanossegundos

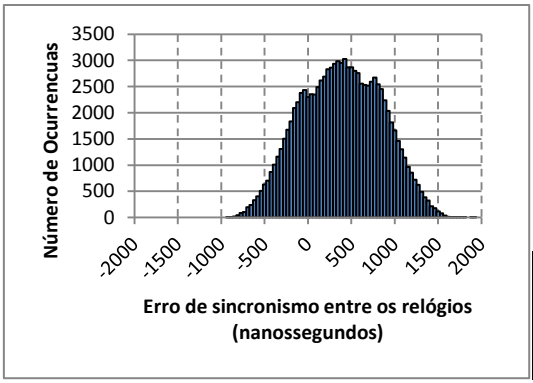

Tabela 4.7 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
386.28 ns	463.76 ns	-923 ns	1923 ns
10.04 ticks	12.06 ticks	-24 ticks	50 ticks

Figura 4.15 – Distribuição do erro de sincronismo para um *SyntCorrValue* de 1000 nanossegundos

4.1.3.2 Teste com SyntCorrValue de 500 nanossegundos (13 Pulsos)

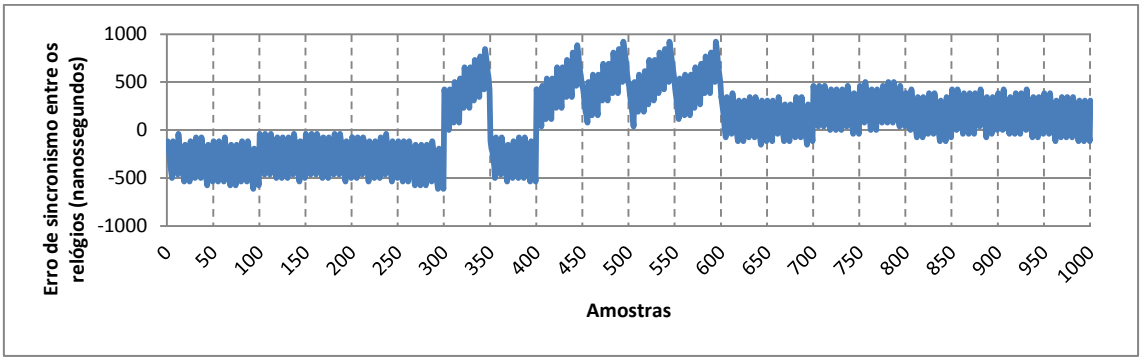

Figura 4.16 – Erro de sincronismo para um *SyntCorrValue* de 500 nanossegundos

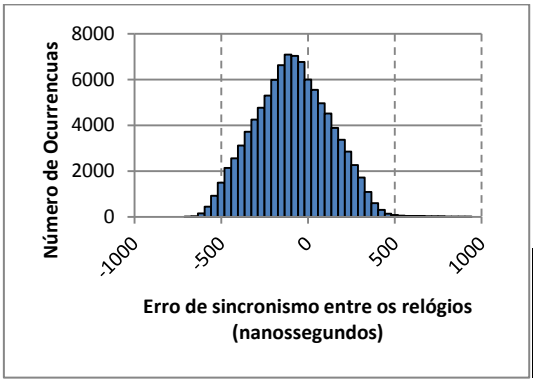

Tabela 4.8 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
-85.75 ns	217.88 ns	-692 ns	923 ns
-2.23 ticks	5.66 ticks	-18 ticks	24 ticks

Figura 4.17 – Distribuição do erro de sincronismo para um *SyntCorrValue* de 500 nanossegundos

4.1.3.3 Teste com SyntCorrValue de 192 nanossegundos (5 Pulsos)

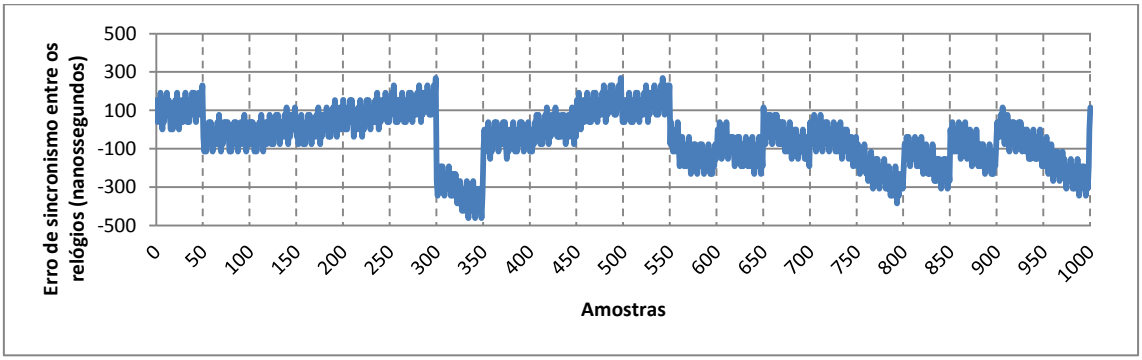

Figura 4.18 – Erro de sincronismo para um *SyntCorrValue* de 192 nanossegundos

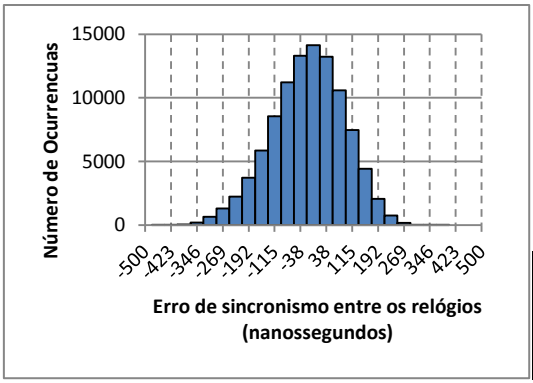

Tabela 4.9 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
-16.48 ns	108.39 ns	-462 ns	385 ns
-0.43 ticks	2.82 ticks	-12 ticks	10 ticks

Figura 4.19 – Distribuição do erro de sincronismo para um *SyntCorrValue* de 192 nanossegundos

4.1.3.4 Teste com SyntCorrValue de 77 nanossegundos (2 Pulsos)

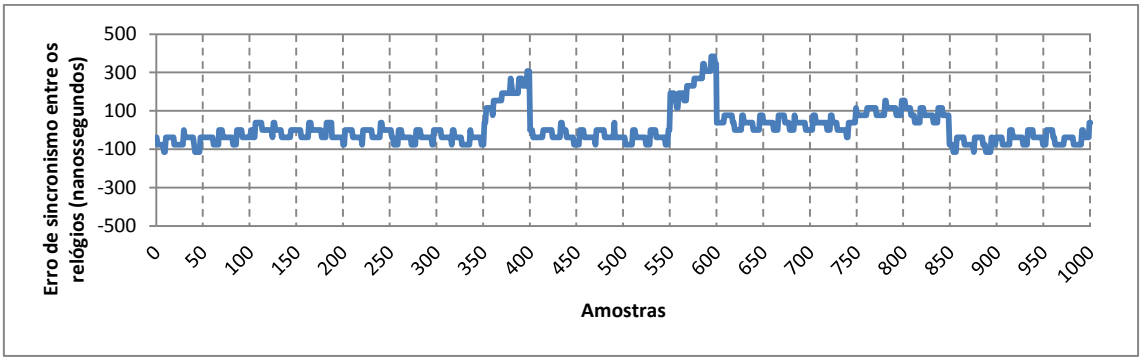

Figura 4.20 – Erro de sincronismo para um *SyntCorrValue* de 77 nanossegundos

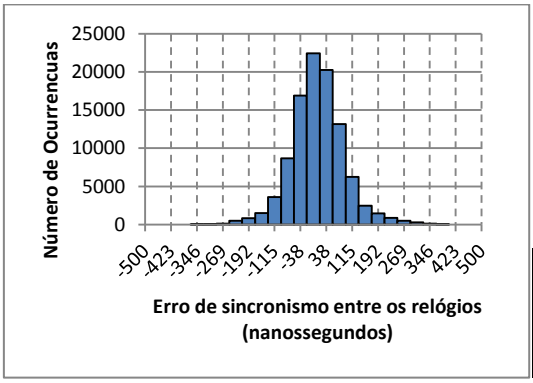

Tabela 4.10 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
13.58 ns	80.18 ns	-346 ns	385 ns
0.35 ticks	2.09 ticks	-9 ticks	10 ticks

Figura 4.21 – Distribuição do erro de sincronismo para um *SyntCorrValue* de 77 nanossegundos

4.1.3.5 Teste com SyntCorrValue de 38 nanossegundos (1 Pulsos)

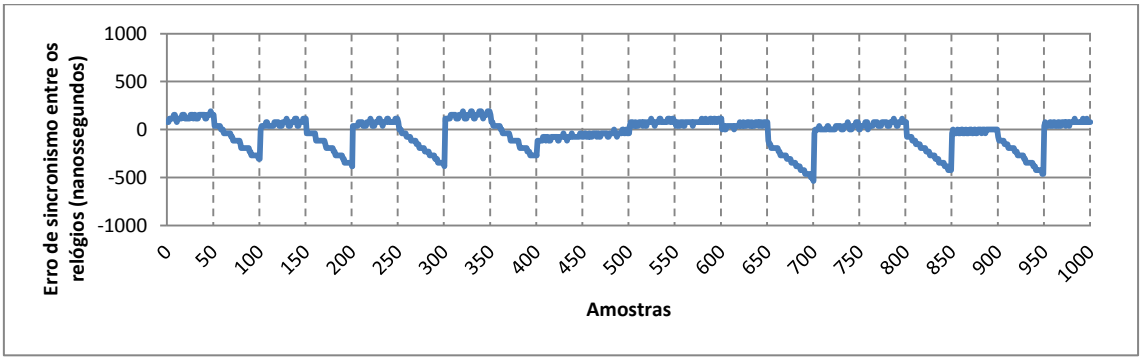

Figura 4.22 – Erro de sincronismo para um SyntCorrValue de 38 nanossegundos

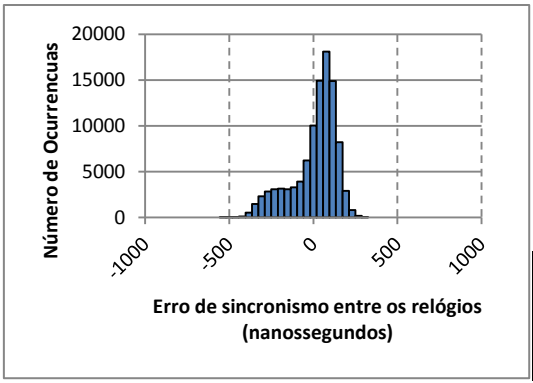

Tabela 4.11 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
7.90 ns	134.46 ns	-538 ns	308 ns
0.21 ticks	3.50 ticks	-14 ticks	8 ticks

Figura 4.23 – Distribuição do erro de sincronismo para um *SyntCorrValue* de 38 nanossegundos

4.1.3.6 Análise de Resultados

Para os testes com os valores de correção de 1000, 500 e 192 nanossegundos, verifica-se que o valor do desvio padrão decai praticamente de forma linear, descendo dos 463 nanossegundos até aos 108 nanossegundos. Com uma correção de 77 nanossegundos, o decaimento do valor do desvio deixou de ser linear e estabiliza nos 80 nanossegundos. No caso da correção de 38 nanossegundos, o valor de desvio subiu para um valor de 134 nanossegundos.

Em termos de valor médio, verifica-se que o valor desce à medida que se utiliza um valor de correção mais baixo, descendo dos 386 nanossegundos no caso de correções de 1000 nanossegundos até um valor de 7.9 nanossegundos no caso de correções de 38 nanossegundos.

Em termos do valor de erro máximo, nos 2 primeiros testes encontra-se com um valor abaixo do dobro do valor de correção, mostrando que nestes casos o valor de *SyntCorrValue* é o factor limitador do erro máximo. A partir do caso de *SyntCorrValue* de 192 nanossegundos o valor continua a descer de uma forma já não tão rápida, obtendo-se o valor de 462 nanossegundos para o caso de correção de 192 nanossegundos e o valor de 385 nanossegundos para o caso de correção de 77 nanossegundos. Por fim temos o caso de correção de 38 nanossegundos onde o erro volta a subir para 538 nanossegundos.

Neste último caso pode ser observado no seu gráfico de valores de erro (Figura 4.22) uma certa instabilidade, onde os seus valores ou se encontram a aumentar positivamente de forma ligeira ou a aumentar negativamente de forma rápida, sem que seja atingida a estabilidade. Este facto acontece devido a uma limitação do algoritmo implementado. Como explicado no capítulo 3.3, o processo de sintonização é feito calculando o número de correções que necessitam de ser realizadas ao relógio, de forma a compensar o seu desvio de frequência. Após ser obtido esse valor, é calculado o período de sintonização para controlar a cadência das correções. No seu cálculo são utilizadas variáveis do tipo inteiro, que não permitem registar a parte fracionária do valor do período de sintonização. Em casos em que o

número de correções a serem realizadas por período de sincronização é grande, esta parte fracionaria do período, ao não ser tomada em consideração, leva a que sejam feitas mais correções do que as que estavam programadas.

Para melhor perceber este problema é apresentado o seguinte exemplo com valores idênticos ao caso de teste realizado. Considerando um período de sincronização de 1 segundo, um valor de correção de sintonização de 1 pulso de relógio, aproximadamente 38 nanossegundos e um erro de *OffsetFromMaster* entre os relógios ao final de 1 segundo de 4,09 microssegundos, obtido no capítulo 3.2, determina-se pela equação (4.1) que são necessárias 106 correções de sintonização para compensar o relógio, o que corresponde pela equação (4.2) a um período de sintonização de 9.40 milissegundos. Considerando que só a parte inteira do valor é usada, o algoritmo de sintonização é configurado para um período de 9 milissegundos. Agora invertendo o processo pela equação (4.3), verifica-se que são efectuadas na realidade 111 correções, 5 correções a mais do que era previsto inicialmente.

$$NSyntCorr = \frac{OffsetFromMaster}{SyntCorrValue} = 106 \quad (4.1)$$

$$SyntPeriod = \frac{SyncPeriod}{NSyntCorr} = 9.4\,ms \quad (4.2)$$

$$RealNSyntCorr = \frac{SyncPeriod}{RealSyntPeriod} = 111 \quad (4.3)$$

Este problema agrava-se com o aumento do período de sincronização, o que invalida a utilização de valores de correções baixos se for pretendido usar períodos de sincronização maiores.

4.1.4 Cálculo do período de sintonização do relógio (*SyntPeriod*)

Para colmatar o problema da parte fracionária das correções de sintonização identificado no capítulo anterior, foram analisadas 3 possíveis soluções a serem implementadas no algoritmo de sintonização. Para melhor perceber o impacto da execução de cada uma delas, vai ser considerado um caso simples de durante um período de sincronismo, serem configuradas 5 correções de sintonização (Figura 4.24) onde na realidade deveriam ser feitas somente 3 correções, o que corresponde a serem realizadas 2 correções a mais.

Se as 5 correções realizadas fossem o número de correções certas, teríamos o caso da Figura 4.25 onde o erro máximo é igual ao próprio valor de correção de sintonização. Como

temos um erro de 2 correções, na verdade vamos ter um desvio de 2 vezes o valor de *SyntCorrValue* (Figura 4.26).

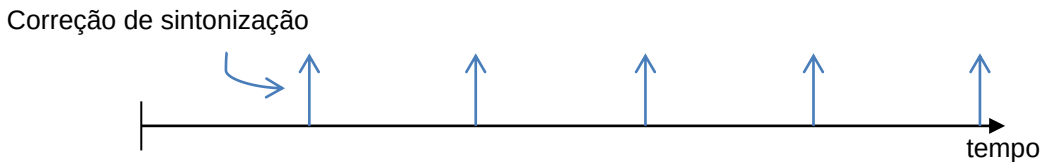

Figura 4.24 – Distribuição no tempo das correções de sintonização

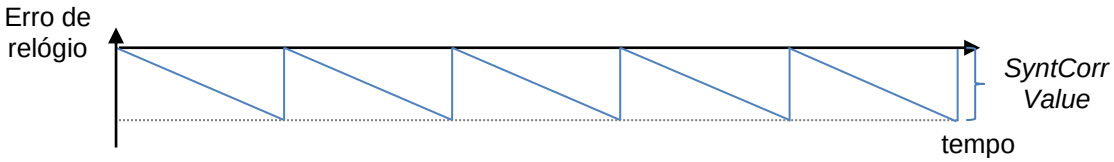

Figura 4.25 – Correção de sintonização teórica

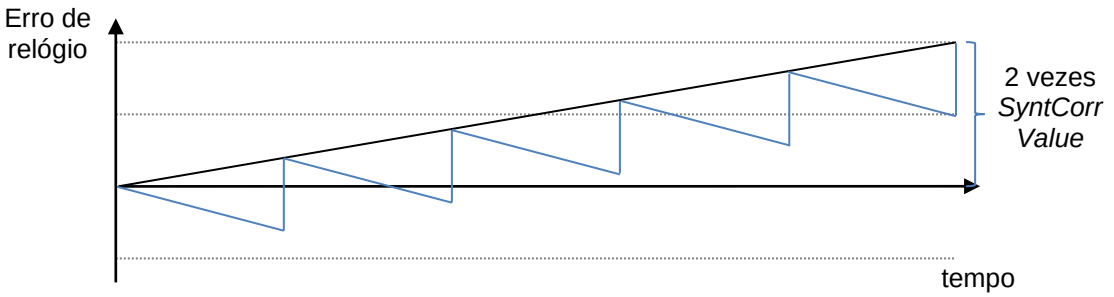

Figura 4.26 – Correção de sintonização real

Visto ser possível calcular o número de correções que estão a ser realizadas em excesso devido ao erro da parte fracionária do período de sintonização, a forma mais simples de corrigir o problema é configurar um segundo período de sintonização onde são efetuadas as correções de sintonização fracionárias mas neste caso no sentido oposto (Figura 4.27).

Neste exemplo, pode-se ver o efeito da correção fracionária a meio da figura, no final a correção normal ocorre ao mesmo tempo que a segunda correção fracionária, fazendo com que ambas anulem o seu efeito. Neste caso o erro varia entre aproximadamente menos *SyntCorrValue* e mais *SyntCorrValue*.

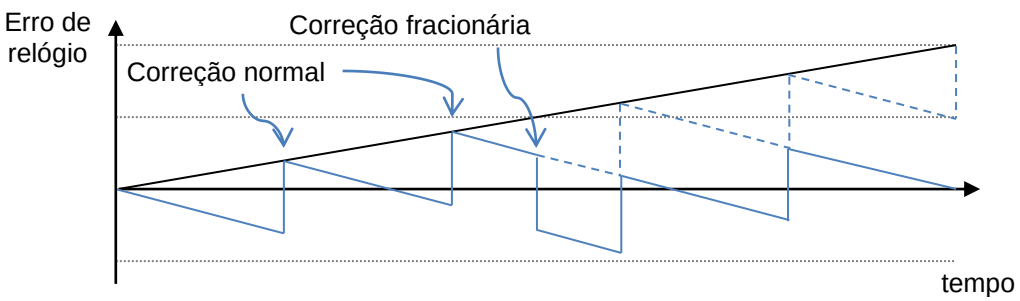

Figura 4.27 – Correção de sintonização fracionárias pelo primeiro método

Outra solução seria ajustar a amplitude do valor de correção dos pontos de sintonização de forma a compensar o erro fracionário. Nesta situação o valor do erro das correções em excesso seria distribuído ao longo das correções de uma forma uniforme (Figura 4.28).

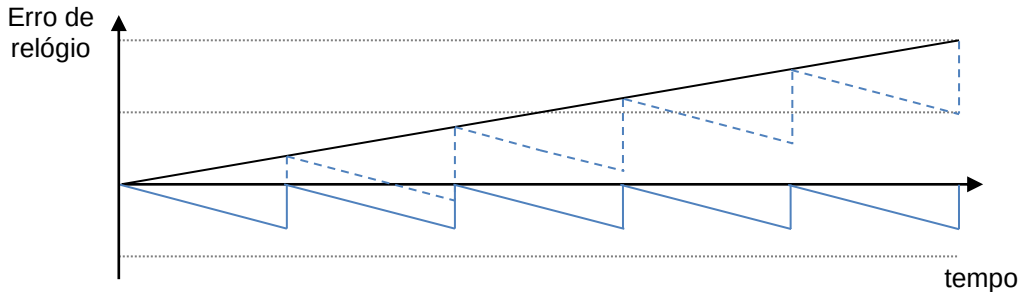

Figura 4.28 – Correção de sintonização fracionárias pelo segundo método

Este conceito é verdadeiro quando o valor de correção utilizado possibilita que seja subdividido em valores mais pequenos, o que é o caso das correções de 1 microssegundo que podem ser ajustadas em 26 valores mais baixos, visto que 1 microssegundo corresponde a 26 pulsos de relógio. No caso de serem utilizadas correções de 38 nanossegundos, o seu valor não pode ser subdividido porque já corresponde a 1 único pulso de relógio, e a sua diminuição leva a não ser feito nenhuma correção (Figura 4.29). Esta solução apresenta melhores resultados de erro máximo que a solução anterior.

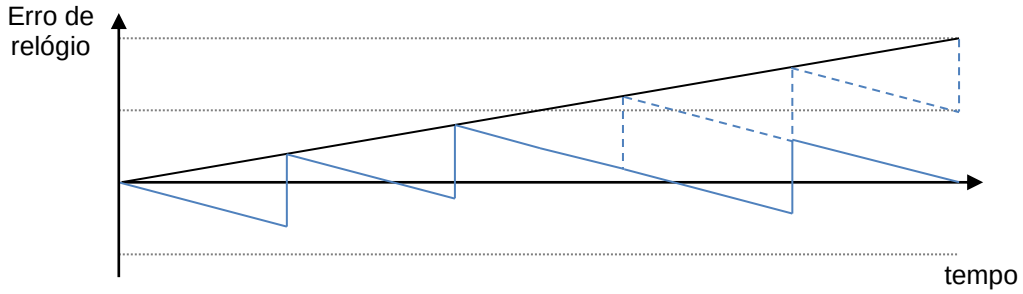

Figura 4.29 – Correção de sintonização fracionárias pelo segundo método (2º caso)

Uma terceira opção é ajustar o período de sintonização de forma a atrasar as correções o suficiente para compensar o erro fracionário (Figura 4.30). Identicamente à solução anterior, a correção do erro é distribuída de forma uniforme por todas as correções. Neste caso, como apresentado na equação (4.4), teríamos uma compensação de 4.09 nanossegundos por cada milissegundo de atraso dum ponto de sintonização. Esta solução permite uma maior resolução da compensação do erro fracionário que as duas soluções anteriores.

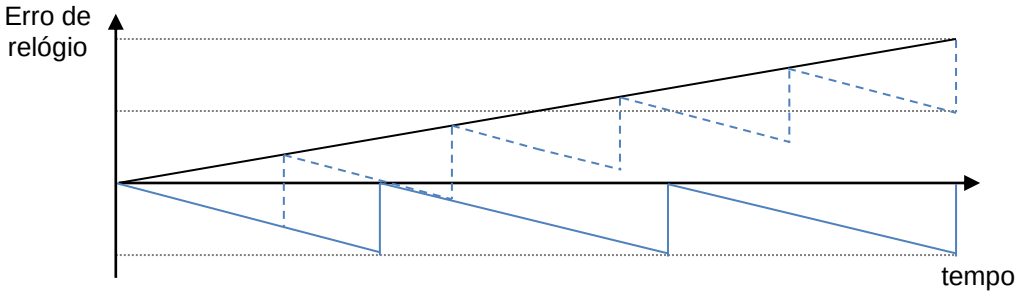

Figura 4.30 – Correção de sintonização fracionárias pelo terceiro método

$$OffsetFromMasterPerMS = \frac{OffsetFromMasterPerSecond}{1000} \quad (4.4)$$

Para implementar esta solução, em vez de o período de sintonização ser calculado uma única vez sempre que é ajustado o seu número de correções, passa a ser calculado após cada ponto de sintonização pelas equações (4.5) e (4.6).

$$SyntPeriod_n = \frac{SyncPeriod_{n-1} + Resto_{n-1}}{NSyntCorr} \quad (4.5)$$

$$Resto_n = Resto\left(\frac{SyncPeriod_{n-1} + Resto_{n-1}}{NSyntCorr}\right) \quad (4.6)$$

Em cada cálculo do período, é obtido o resto da divisão que é somado ao cálculo do período seguinte. Desta forma a parte fracionária do valor do período não é perdida e quando a soma dos restos corresponder a um atraso de 1 milissegundo, no da unidade AP e unidade ED usada vamos ter uma compensação de 4.09 nanossegundos do valor do relógio.

Para validar as modificações realizadas ao algoritmo de sintonização, o ensaio descrito no capítulo 4.1.2 foi repetido para os valores de período de sincronismo de 1, 2, 5, 10, 30 e 60 segundos com um valor de correção de sintonização de 38 nanossegundos, correspondente a 1 pulso de relógio.

De seguida são apresentados os resultados obtidos e uma análise final.

4.1.4.1 Teste com SyncPeriod de 1 segundo

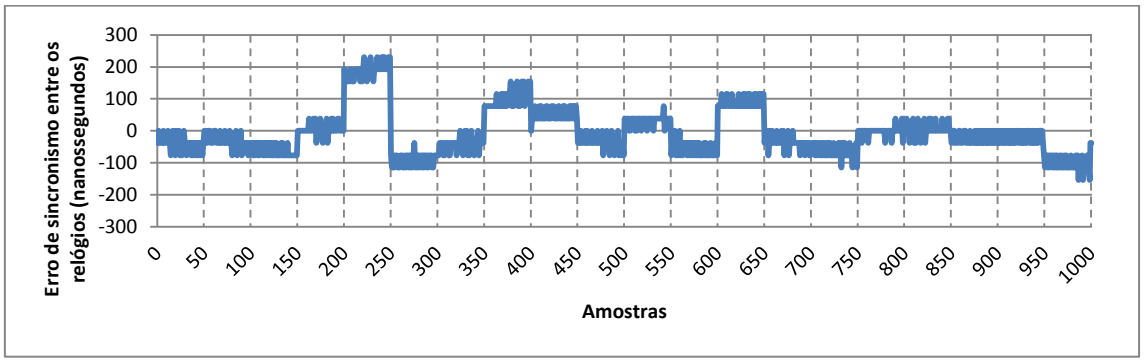

Figura 4.31 – Erro de sincronismo para um *SyncPeriod* de 1 segundo

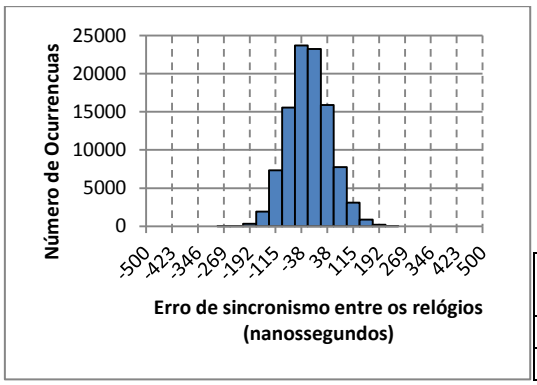

Tabela 4.12 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
-15.71 ns	63.33 ns	-269 ns	231 ns
-0.41 ticks	1.65 ticks	-7 ticks	6 ticks

Figura 4.32 – Distribuição do erro de sincronismo para um *SyncPeriod* de 1 segundo

4.1.4.2 Teste com SyncPeriod de 2 segundos

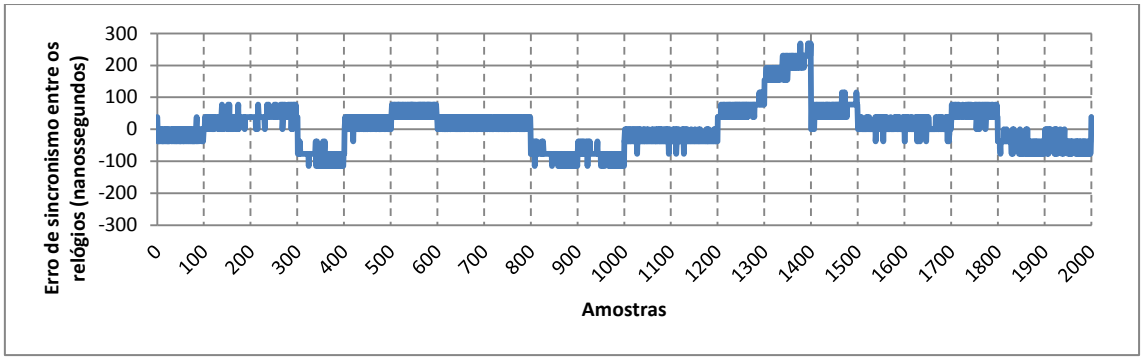

Figura 4.33 – Erro de sincronismo para um *SyncPeriod* de 2 segundos

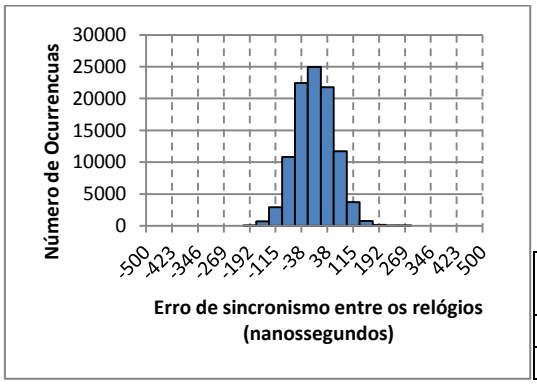

Tabela 4.13 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
1.79 ns	57.29 ns	-192 ns	269 ns
0.05 ticks	1.49 ticks	-5 ticks	7 ticks

Figura 4.34 – Distribuição do erro de sincronismo para um *SyncPeriod* de 2 segundos

4.1.4.3 Teste com SyncPeriod de 5 segundos

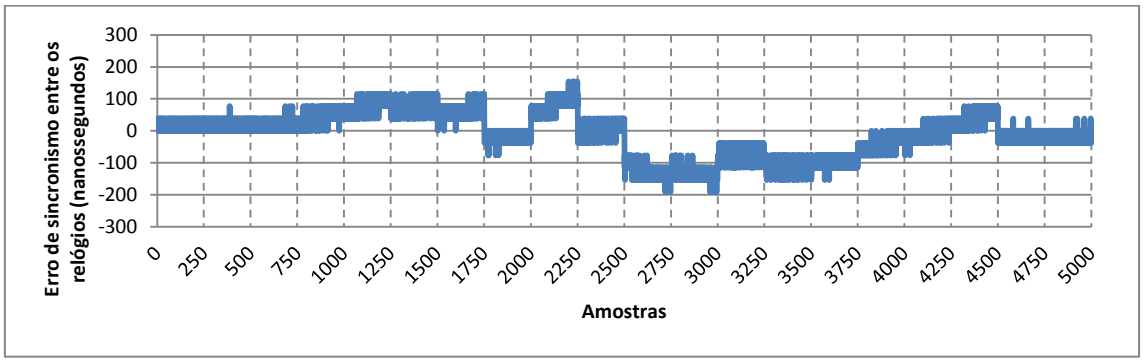

Figura 4.35 – Erro de sincronismo para um *SyncPeriod* de 5 segundos

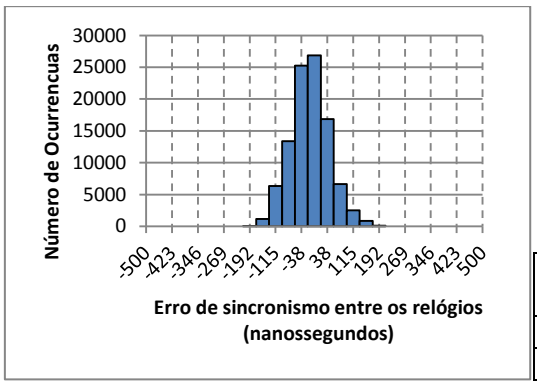

Tabela 4.14 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
-13.23 ns	57.67 ns	-192 ns	192 ns
-0.35 ticks	1.50 ticks	-5 ticks	5 ticks

Figura 4.36 – Distribuição do erro de sincronismo para um *SyncPeriod* de 5 segundos

4.1.4.4 Teste com SyncPeriod de 10 segundos

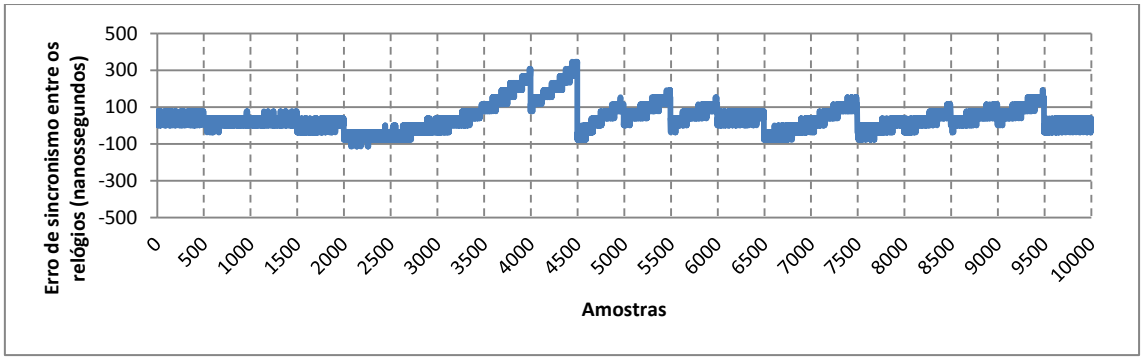

Figura 4.37 – Erro de sincronismo para um *SyncPeriod* de 10 segundos

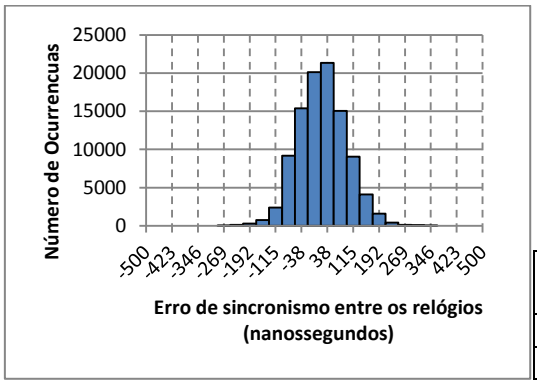

Tabela 4.15 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
23.25 ns	72.34 ns	-269 ns	346 ns
0.61 ticks	1.88 ticks	-7 ticks	9 ticks

Figura 4.38 – Distribuição do erro de sincronismo para um *SyncPeriod* de 10 segundos

4.1.4.5 Teste com SyncPeriod de 30 segundos

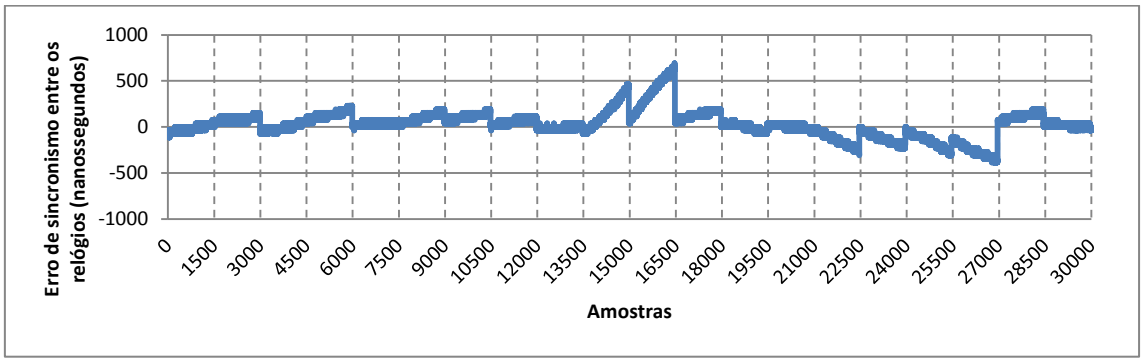

Figura 4.39 – Erro de sincronismo para um *SyncPeriod* de 30 segundos

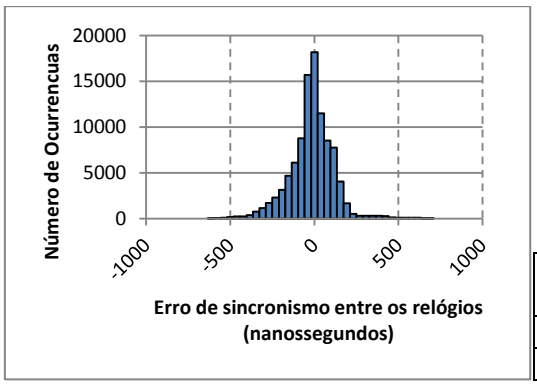

Tabela 4.16 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
-15.55 ns	134.16 ns	-615 ns	692 ns
-0.40 ticks	3.49 ticks	-16 ticks	18 ticks

Figura 4.40 – Distribuição do erro de sincronismo para um *SyncPeriod* de 30 segundos

4.1.4.6 Teste com SyncPeriod de 60 segundos

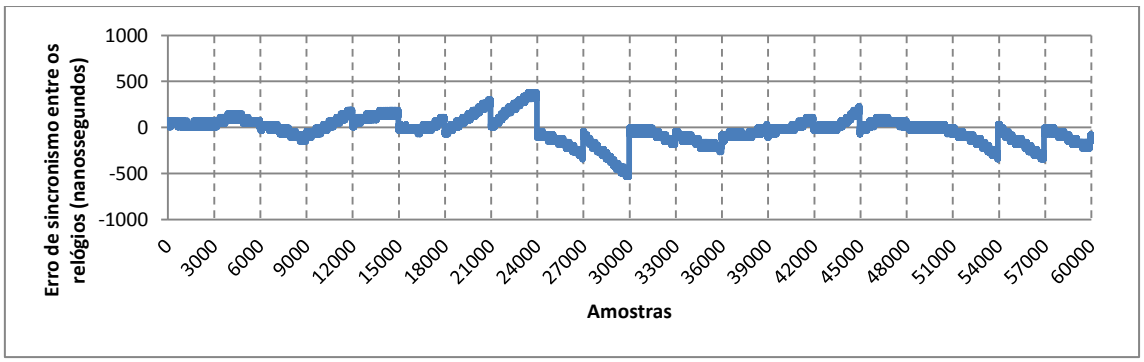

Figura 4.41 – Erro de sincronismo para um *SyncPeriod* de 60 segundos

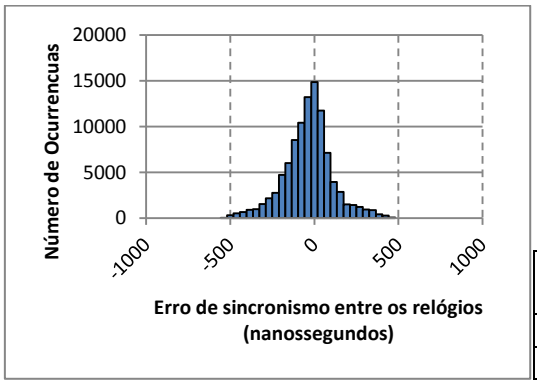

Tabela 4.17 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
-34.29 ns	146.11 ns	-538 ns	462 ns
-0.89 ticks	3.80 ticks	-14 ticks	12 ticks

Figura 4.42 – Distribuição do erro de sincronismo para um *SyncPeriod* de 60 segundos

4.1.4.7 Análise de Resultados

Comparando o resultado obtido com o período de sincronismo de um segundo no capítulo 4.1.4.1, com o resultado do capítulo 4.1.3.5 onde a única diferença entre os testes foi a implementação de correções de sintonização com a componente fraccionária do período, conseguiu-se um decréscimo do desvio padrão de 134.46 nanossegundos para 63.33 nanossegundos, uma diminuição de mais de metade do seu valor.

No teste com período de 2 segundos obtemos uma diminuição de 5 nanossegundos do desvio padrão ao descer para o valor de 57.29 nanossegundos. Com o acréscimo do período de sincronismo os valores do desvio sobem para valores de 57.67 nanossegundos, 72.34 nanossegundos, 134.16 nanossegundos e 146.11 nanossegundos, correspondentemente os períodos de sincronismo de 5, 10, 30 e 60 segundos.

Pelos resultados obtidos, temos o teste com período de sincronismo de 2 segundos como o melhor caso. Diminuído ou aumentando o valor do período, os resultados pioram. No caso de períodos mais pequenos o aumento do erro é dominado pelo erro do cálculo do valor de *OffsetFromMaster*. O seu valor ao ser calculado de forma imprecisa, vai influenciar tanto a correção direta do valor do relógio, como vai aumentar ou diminuir o número de correções de sintonização para um valor incorreto. Este impacto pode ser facilmente observável nos gráficos de erro de sincronismo apresentados para os testes com períodos de sincronismo de 1 (Figura 4.31), 2 (Figura 4.33), e 5 segundos (Figura 4.35). Nos testes com períodos de sincronismos acima de 5 segundos, o erro passa a ser dominado pela variação da frequência do relógio com a temperatura. Pelo *datasheet* do cristal usado [19] observamos que existe uma variação de ±10ppm para uma gama térmica de -20°C a 70°C. Considerando a sua variação linear com a temperatura, pode ser deduzido uma variação de 0.22ppm/°C. Este valor pode ser interpretado como o número de microssegundos de erro do cristal ao final de 1 segundo se durante esse tempo tivermos uma variação instantânea de 1°C na sua temperatura. As temperaturas não variam instantaneamente, por isso este caso não é completamente realista, mas pode ser usado como referência de pior caso. Agora considerando que a temperatura do cristal varia em 0.1°C, correspondendo a uma variação de 22 nanossegundos por segundo, obtém-se um erro final de 660 nanossegundos para um período de 30 segundos e 1320 nanossegundos para um período de 60 segundos. Este problema torna-se uma grande vulnerabilidade quando se pretende obter erros de sincronismo abaixo dos 500 nanossegundos para esta ordem de períodos de sincronismo, visto que uma variação de 0.1°C é facilmente atingível se o cristal das unidades não tiver algum tipo de isolamento térmico.

4.1.5 Cálculo do atraso do canal de comunicação (*PathDelay*)

O cálculo do atraso do canal de comunicação é realizado pela troca das mensagens *DelayReq* e *DelayRsp* entre a unidade que têm o relógio de referência (unidade AP) e a unidade que pretende se sincronizar (unidade ED), enviadas após a transmissão das mensagens *Sync* e *FollowUp*. Com estas 4 mensagens a unidade ED calcula o atraso do canal de comunicação e usa o seu valor para compensar os subsequentes cálculos *OffsetFromMaster*. Erros no cálculo deste parâmetro são reflectidos como um erro constante entre o valor dos relógios das duas unidades.

Para aumentar a fiabilidade do cálculo são obtidos múltiplos valores intermédios do atraso do canal de comunicação e posteriormente realizada a média desses valores para determinar o valor final. Isto levanta outro problema relativo à dependência que o cálculo do atraso do canal de comunicação tem com as mensagens *Sync* e *FollowUp*. Numa situação em que é usado um período de sincronismo de 30 segundos e são usados 10 cálculos intermédios do atraso do canal de comunicação, só ao final de 5 minutos é que seria obtido um valor final.

Este problema é solucionado separando o processo do cálculo do valor de atraso do canal de comunicação (*PathDelay*) da transmissão das mensagens *Sync* e *FollowUp*. Este novo processo é composto pelas mensagens *DelayReq*, *DelayRsp* e *DelayReply* apresentadas na Figura 4.43.

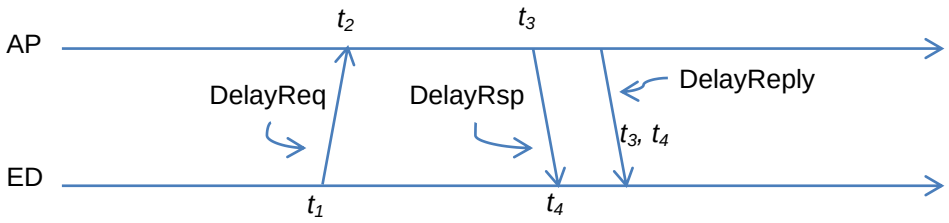

Figura 4.43 – Troca de mensagens de sincronismo para o cálculo de *PathDelay*

Neste caso quem começa o processo é a unidade ED. A unidade em qualquer momento envia a mensagem *DelayReq* e regista o tempo t_1 enquanto a unidade AP recebe a mensagem e regista o tempo t_2. De seguida processo é invertido e a unidade AP responde com a mensagem *DelayRsp* onde regista o tempo t_3 e a unidade ED regista o tempo t_4 ao receber a mensagem. Para finalizar, a unidade AP envia a mensagem *DelayReply* para a unidade ED onde transporta os tempos t_2 e t_3 registados no processo. Nas três mensagens é incluído o campo *seqNumbDelay* para manter a consistência da informação trocada.

Com estes 4 tempos a unidade ED calcula o valor do atraso do canal de comunicação pela equação (4.7). Como o canal de comunicação é considerado como tendo um atraso simétrico, o valor calculado com os tempos obtidos por este processo é idêntico ao valor obtido pelo processo usado anteriormente.

$$PathDelay = \frac{(t_2 - t_1) + (t_4 - t_3)}{2} \quad (4.7)$$

Este novo processo permite que a unidade ED tenha total controlo de quando é que o valor de atraso do canal de comunicação é calculado e de quantos valores intermédios são usados no cálculo do valor final.

Nos ensaios seguintes, pretendeu-se validar a qualidade do valor do atraso do canal de comunicação obtido em função do número de valores intermédios usados no cálculo. Neste caso a unidade ED inicia o processo de sincronismo e fica repetidamente a calcular o valor de atraso do canal de comunicação. Após cada valor obtido, este é reencaminho para o PC e posteriormente tratado (Figura 4.44).

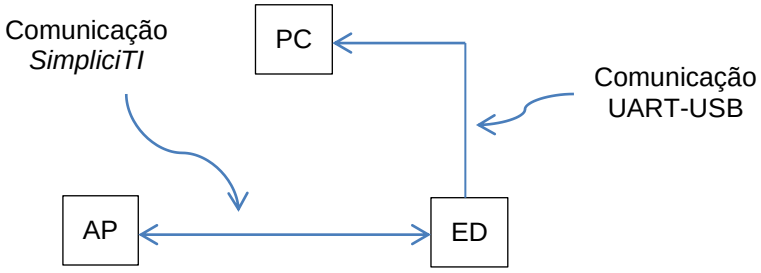

Figura 4.44 – Estrutura do ensaio

De seguida são apresentados os diagramas do valor do atraso do canal de comunicação e as tabelas com os seus valores de média, desvio padrão, valor mínimo e valor máximo. Os testes foram repetidos para os casos da utilização de 1, 2, 4, 8, 16 e 32 valores intermédios no seu cálculo e foram registados 50000 valores finais do atraso do canal de comunicação.

4.1.5.1 Cálculo com 1 valor de PathDelay

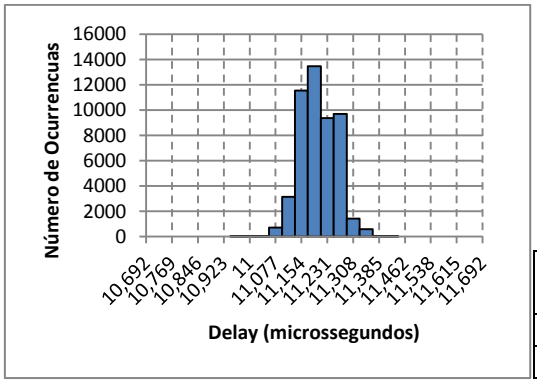

Tabela 4.18 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
11.20 us	53.76 ns	10.96 us	11.42 us
291.31 ticks	1.40 ticks	285 ticks	297 ticks

Figura 4.45 – Distribuição do valor de *PathDelay* para o cálculo com 1 valor

4.1.5.2 Cálculo com 2 valores de PathDelay

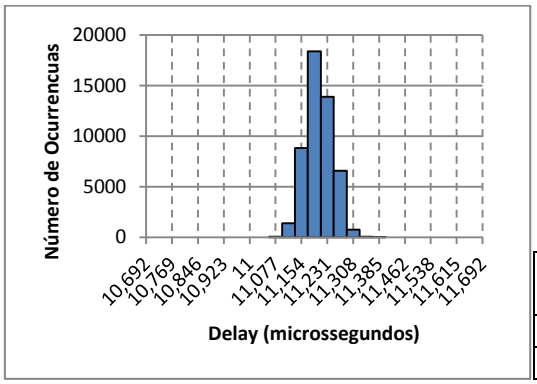

Tabela 4.19 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
11.21 us	41.11 ns	11.08 us	11.39 us
291.36 ticks	1.07 ticks	288 ticks	296 ticks

Figura 4.46 – Distribuição do valor de *PathDelay* para o cálculo com 2 valores

4.1.5.3 Cálculo com 4 valores de PathDelay

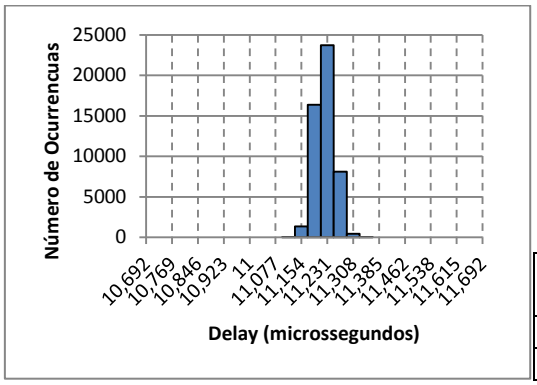

Tabela 4.20 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
11.22 us	29.73 ns	11.12 us	11.35 us
291.80 ticks	0.77 ticks	289 ticks	295 ticks

Figura 4.47 – Distribuição do valor de *PathDelay* para o cálculo com 4 valores

4.1.5.4 Cálculo com 8 valores de PathDelay

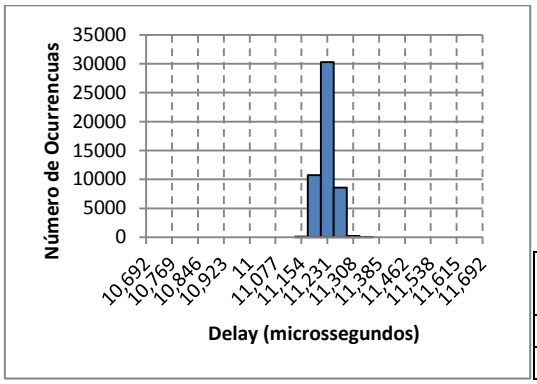

Tabela 4.21 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
11.23 us	24.82 ns	11.15 us	11.35 us
291.96 ticks	0.64 ticks	290 ticks	295 ticks

Figura 4.48 – Distribuição do valor de *PathDelay* para o cálculo com 8 valores

4.1.5.5 Cálculo com 16 valores de PathDelay

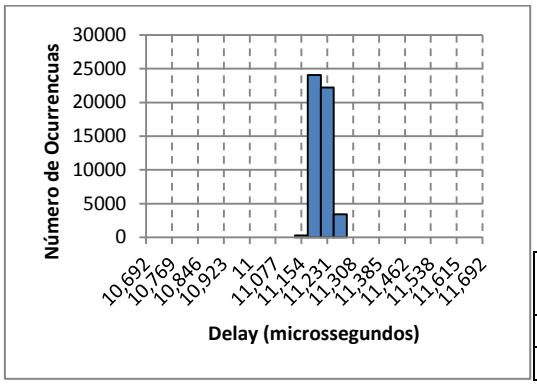

Tabela 4.22 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
11.21 us	24.31 ns	11.15 us	11.27 us
291.58 ticks	0.63 ticks	290 ticks	293 ticks

Figura 4.49 – Distribuição do valor de *PathDelay* para o cálculo com 16 valores

4.1.5.6 Cálculo com 32 valores de PathDelay

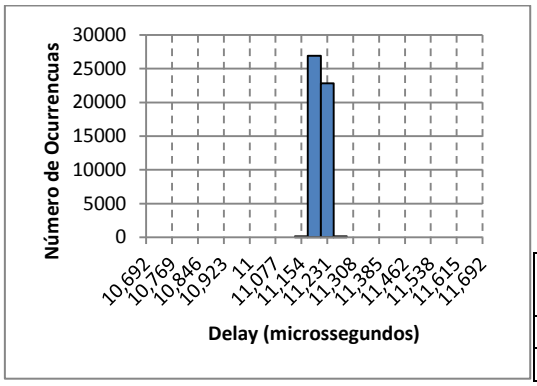

Tabela 4.23 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
11.21 us	19.85 ns	11.15 us	11.27 us
291.46 ticks	0.51 ticks	290 ticks	293 ticks

Figura 4.50 – Distribuição do valor de *PathDelay* para o cálculo com 32 valores

4.1.5.7 Análise de Resultados

Com o aumento do número de valores intermédios usados no cálculo dos valores finais, menor é o desvio padrão apresentado nos testes, obtendo-se 53.76 nanossegundo no caso da utilização de um valor intermédio e descendo até 19.85 nanossegundo no caso da utilização de 32 valores intermédios. Este aumento de precisão é obtido à custa de um maior número de mensagens trocadas, onde para o cálculo de cada valor intermédio, é necessário a troca de 3 mensagens. No caso da média de 32 valores intermédio, são necessárias 96 mensagens para completar o processo.

Nos testes realizados nas secções seguintes serão usados 32 valores intermédios no cálculo do valor do *PathDelay*, visto que o objectivo é obter um menor erro de sincronismo possível para validar a qualidade do sistema. Numa situação real é necessário pesar a relação entre a precisão do valor obtido e o número de mensagens trocadas, caso o factor energético seja um dos fatores limitantes do sistema.

4.1.6 Cálculo do desvio do relógio (*OffsetFromMaster*)

De nada serve obter um valor de atraso do canal de comunicação preciso se não for acompanhado por um valor de desvio de relógio também preciso. Para tal foi usado o mesmo princípio de utilizar cálculos de *OffsetFromMaster* intermédios e no final efetuar a média dos valores para se obter um valor final.

Para acomodar esta alteração no processo e ao mesmo tempo minimizar as mensagens usadas. A unidade AP que enviava anteriormente uma mensagem *Sync* seguida de uma mensagem *FollowUp*, passa a enviar múltiplas mensagens *Sync*, registando o tempo de envio de cada uma delas. Por sua vez a unidade ED regista o tempo de chegada das mensagens e guarda os seus valores. No final a unidade AP envia os tempos que registou para a unidade ED através de mensagens *FollowUp*. No caso anterior cada mensagem *FollowUp* era constituída por um único tempo registado. Neste caso a mensagem vai ser composta por múltiplo tempos e no caso de uma mensagem não ter espaço para transportá-los todos, múltiplas mensagens de *FollowUp* são usadas. Na Figura 4.51 é apresentado um diagrama de fluxo da troca de mensagem para um caso de utilização de 8 valores de desvio de relógio intermédios no cálculo do valor final.

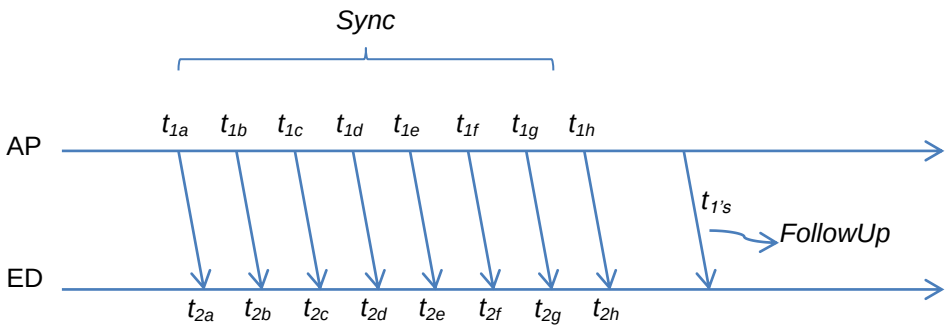

Figura 4.51 – Troca de mensagens de sincronismo para o cálculo de *OffsetFromMaster*

Para validar esta modificação do sistema, montou-se um ensaio idêntico ao usado no capítulo 4.1.2 mas sem o gerador de pulsos. A unidade ED após calcular um valor de desvio de relógio efetua a correção e gera um pulso. Esse pulso vai ser usado tanto pela unidade ED como pela unidade AP para registar um valor de relógio, encaminhando-o de seguida para o PC. Neste teste o algoritmo de sintonização não é inicializado para que se possa avaliar a estabilidade do cálculo do valor *OffsetFromMaster*. O diagrama da montagem é apresentado na Figura 4.52.

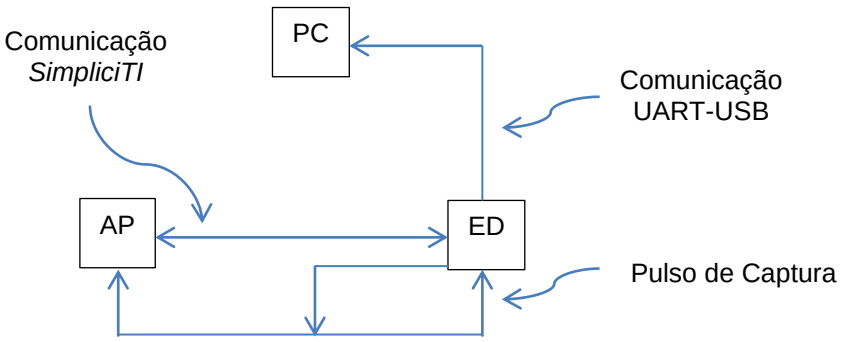

Figura 4.52 – Estrutura do ensaio

Os testes foram repetidos para 1, 2, 4, 8, 16 e 32 valores intermédios do cálculo do valor de desvio de relógio final onde foram registados 35000 valores por teste. De seguida são apresentados os resultados obtidos.

4.1.6.1 Cálculo com 1 valor de OffsetFromMaster

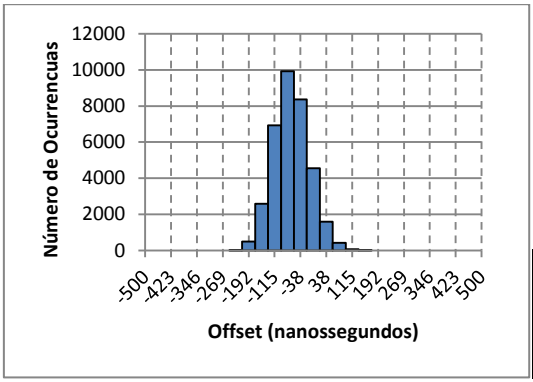

Tabela 4.24 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
-65.03 ns	53.73 ns	-231 ns	154 ns
-1.70 ticks	1.4 ticks	-6 ticks	4 ticks

Figura 4.53 – Distribuição do valor de *OffsetFromMaster* para o cálculo com 1 valor

4.1.6.2 Cálculo com 2 valores de OffsetFromMaster

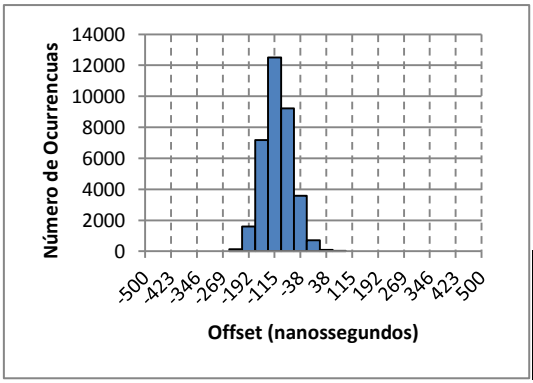

Tabela 4.25 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
-106.32 ns	43.27 ns	-231 ns	77 ns
-2.77 ticks	1.12 ticks	-6 ticks	2 ticks

Figura 4.54 – Distribuição do valor de *OffsetFromMaster* para o cálculo com 2 valores

4.1.6.3 Cálculo com 4 valores de OffsetFromMaster

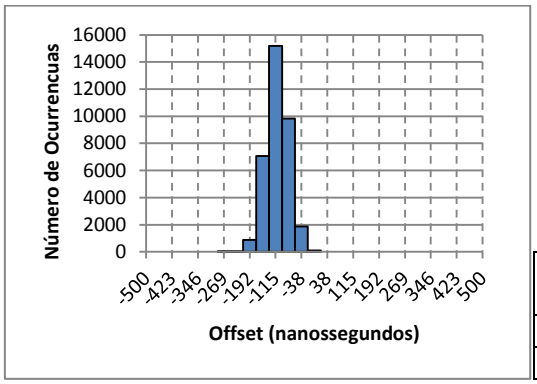

Tabela 4.26 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
-109.72 ns	34.73 ns	-269 ns	0 ns
-2.86 ticks	0.90 ticks	-7 ticks	0 ticks

Figura 4.55 – Distribuição do valor de *OffsetFromMaster* para o cálculo com 4 valores

4.1.6.4 Cálculo com 8 valores de OffsetFromMaster

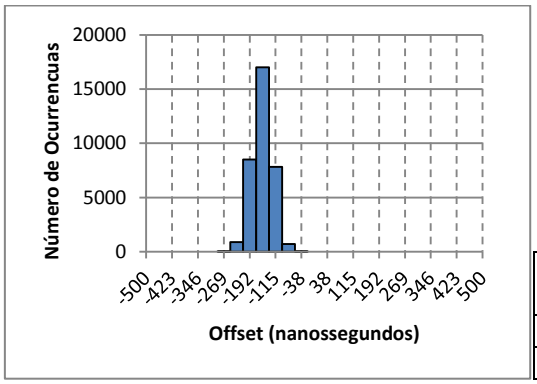

Tabela 4.27 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
-154.85 ns	31.15 ns	-269 ns	-38 ns
-4.03 ticks	0.81 ticks	-7 ticks	-1 ticks

Figura 4.56 – Distribuição do valor de *OffsetFromMaster* para o cálculo com 8 valores

4.1.6.5 Cálculo com 16 valores de OffsetFromMaster

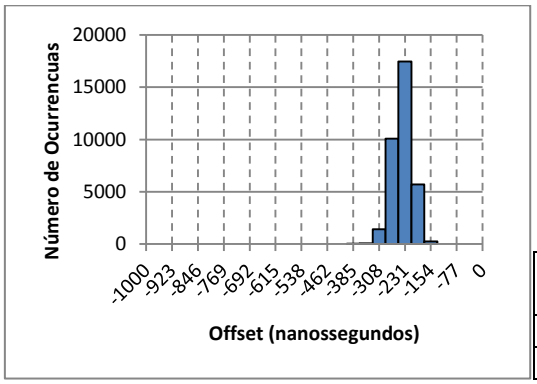

Tabela 4.28 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
-238.46 ns	30.50 ns	-385 ns	-154 ns
-6.20 ticks	0.79 ticks	-10 ticks	-4 ticks

Figura 4.57 – Distribuição do valor de *OffsetFromMaster* para o cálculo com 16 valores

4.1.6.6 Cálculo com 32 valores de OffsetFromMaster

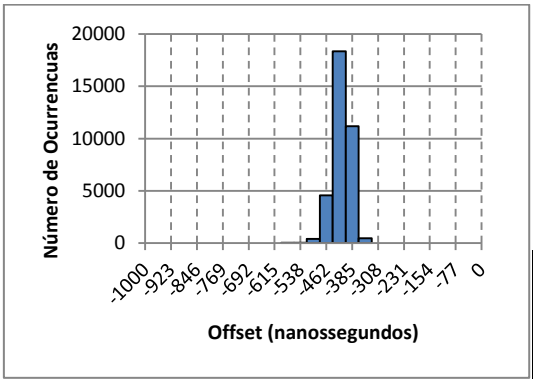

Tabela 4.29 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
-415.93 ns	27.84 ns	-577 ns	-346 ns
-10.81 ticks	0.73 ticks	-15 ticks	-9 ticks

Figura 4.58 – Distribuição do valor de *OffsetFromMaster* para o cálculo com 32 valores

4.1.6.7 Análise de Resultados

Com o aumento do número de pontos intermédios, o desvio padrão diminui, começando com o valor de 53.73 nanossegundos para o teste com 1 valor intermédio e acabando com o valor de 27.84 nanossegundos para o teste com 32 valores intermédios. Por outro lado, o valor médio afasta-se de zero começando com -65.03 nanossegundos no teste com 1 valor intermédio até ao valor de -415.93 nanossegundos no teste com 32 valores intermédios.

Este último facto deve-se à latência de todo o processo desde a obtenção dos valores intermédios dos desvio do relógio até que o valor final seja calculado e o relógio corrigido. Durante este tempo, o relógio continua constantemente a aumentar o seu desvio, fazendo que o valor final calculado esteja atrasado no momento em que é usado para corrigir o relógio. O impacto deste atraso aumenta à medida que a latência do processo aumenta.

Numa situação em que não fosse usado um algoritmo de sintonização, este atraso poderia constituir um problema grave, que invalidaria a utilização de múltiplos valores intermédios de desvio de relógio no cálculo de um valor final mais preciso. Mas com a utilização de um algoritmo de sintonização, o incremento do desvio do relógio é atenuado, reduzindo desta forma o seu impacto.

4.1.7 Cálculo do número de correções de sintonização (*NSyntCorr*)

Na implementação do algoritmo de sintonização, a adaptação do número de correções de sintonização ao longo do tempo é feito por níveis como explicado no capítulo 3.3, onde são usados 3 patamares de ajuste do número de correções em função do valor de *OffsetFromMaster* calculado. A ideia desta implementação é criar uma certa latência no incremento ou decremento do número de correções de sintonização para proteger o algoritmo dos erros gerados no cálculo do valor de desvio de relógio. Esta mesma latência acaba por ser responsável por um maior erro de sincronismo nos períodos de 30 e 60 segundos, onde a

frequência do relógio varia devido à temperatura e o sistema ao fazer a compensação por patamares não consegue resposta suficiente.

Ao ser melhorado a precisão do cálculo do desvio do relógio com o método apresentado no subcapítulo 4.1.6, deixa de ser necessário a compensação em patamares. Como tal, a compensação passa a ser feita diretamente, incrementando ou decrementando o número de correções de sintonização correspondentes ao desvio de relógio (equação (4.8)).

$$NSyntCorrInc = \frac{OffsetFromMaster}{SyntCorrValue} \tag{4.8}$$

Na Figura 4.59 é apresentado o novo diagrama de cálculo do número de correções de sintonização realizado após cada ponto de cálculo do valor de *OffsetFromMaster*.

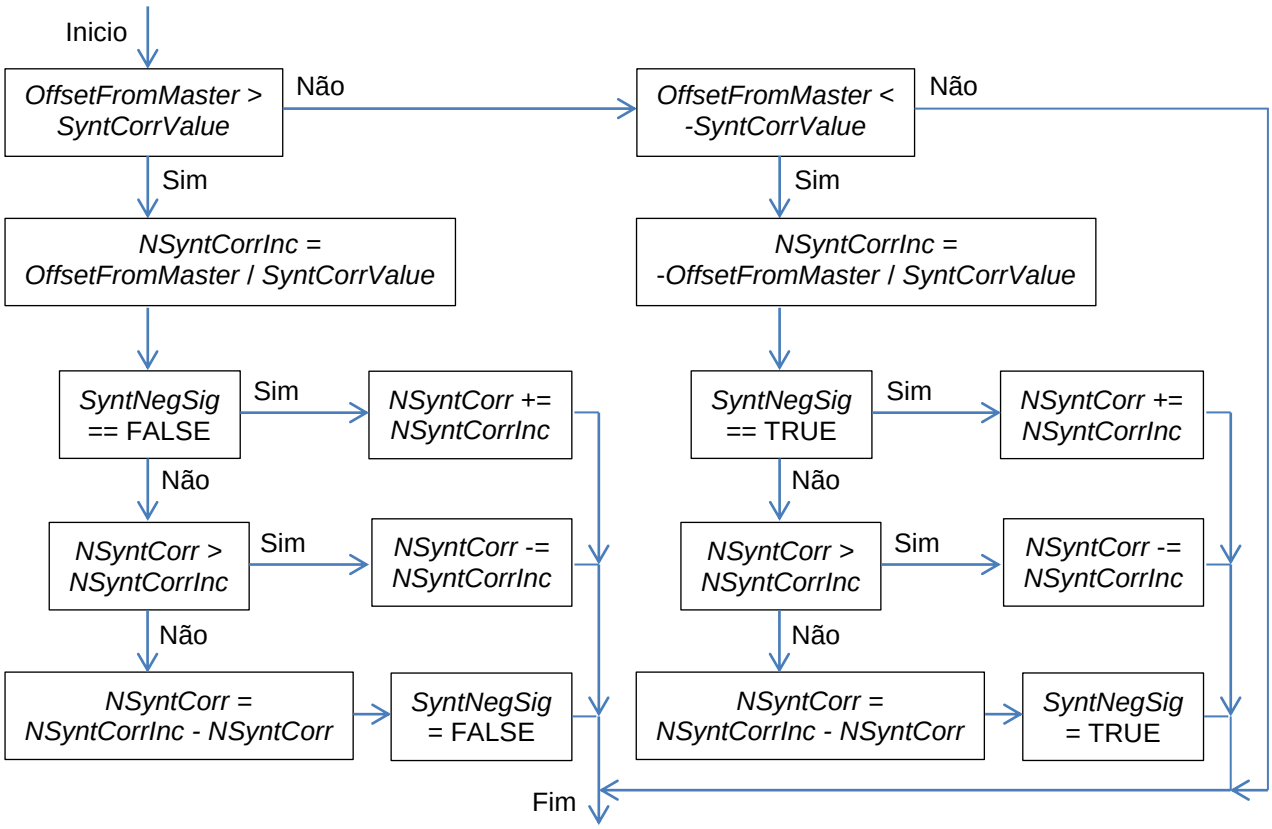

Figura 4.59 – Fluxo do cálculo do número de correções de sintonização

Para validação final do sistema foi repetido o ensaio do capítulo 4.1.2. O valor de *PathDelay* e de *OffsetFromMaster* é obtido através do cálculo da média de 32 valores intermédios como explicado no capítulo 4.1.5 e 4.1.6. O teste é repetido para os tempos de 1, 2, 5, 10, 30 e 60 segundos de período de sincronismo com um valor de correção de sintonização de um ciclo de relógio, aproximadamente 38 nanossegundos. Devido à grande vulnerabilidade do sistema a variações de temperatura como constatado no capítulo 4.1.4.7, foi

tomado o cuidado de realizar os testes com o sistema dentro de uma caixa para permitir uma maior estabilidade da temperatura.

4.1.7.1 Teste com SyncPeriod de 1 segundo

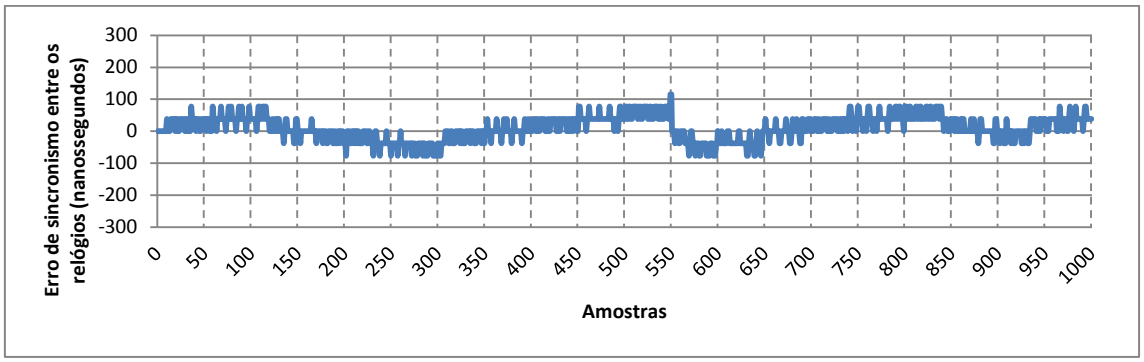

Figura 4.60 – Erro de sincronismo para um *SyncPeriod* de 1 segundo

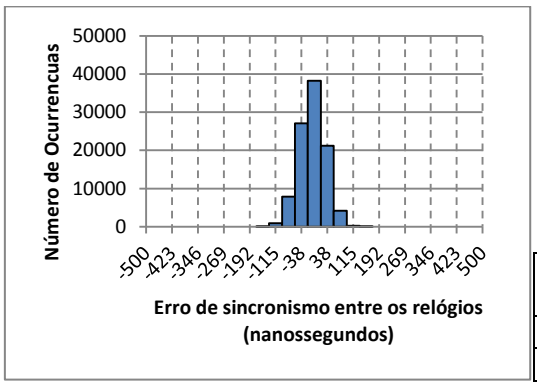

Tabela 4.30 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
-5.88 ns	39.48 ns	-154 ns	154 ns
-0.15 ticks	1.03 ticks	-4 ticks	4 ticks

Figura 4.61 – Distribuição do erro de sincronismo para um *SyncPeriod* de 1 segundo

4.1.7.2 Teste com SyncPeriod de 2 segundos

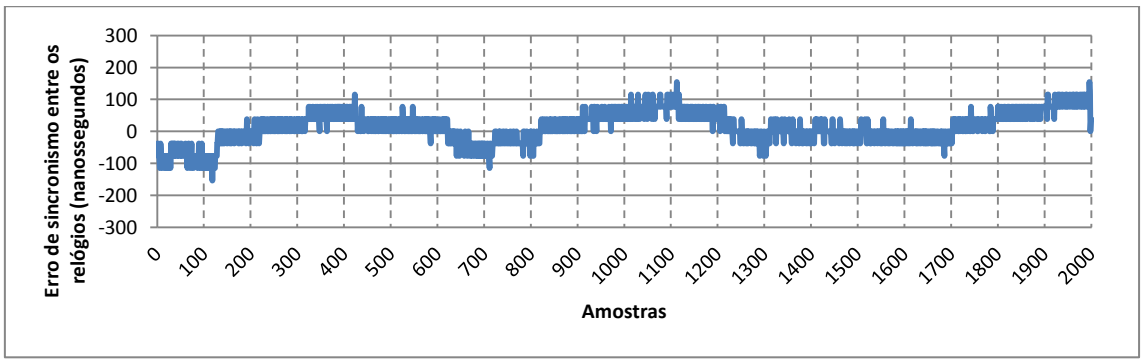

Figura 4.62 – Erro de sincronismo para um *SyncPeriod* de 2 segundos

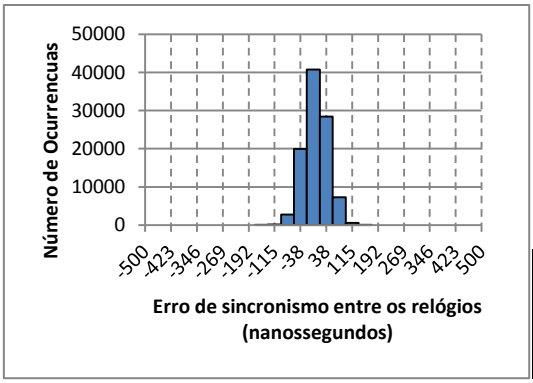

Tabela 4.31 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
7.21 ns	36.82 ns	-154 ns	154 ns
0.19 ticks	0.96 ticks	-4 ticks	4 ticks

Figura 4.63 – Distribuição do erro de sincronismo para um *SyncPeriod* de 2 segundos

4.1.7.3 Teste com SyncPeriod de 5 segundos

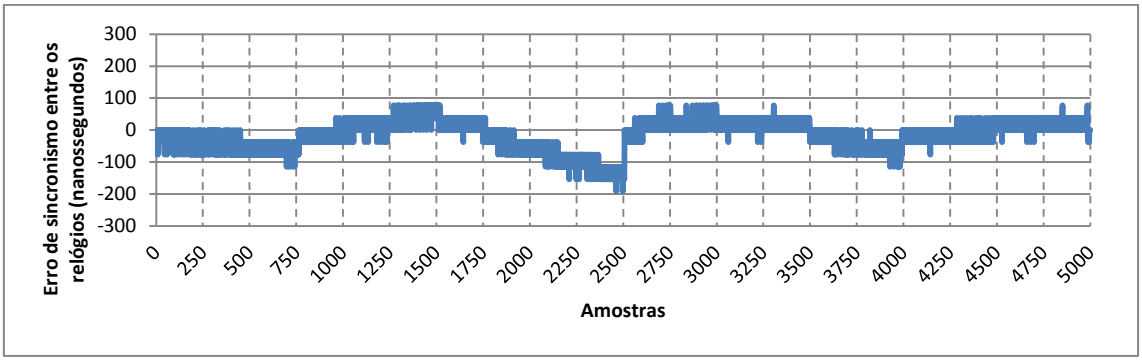

Figura 4.64 – Erro de sincronismo para um *SyncPeriod* de 5 segundos

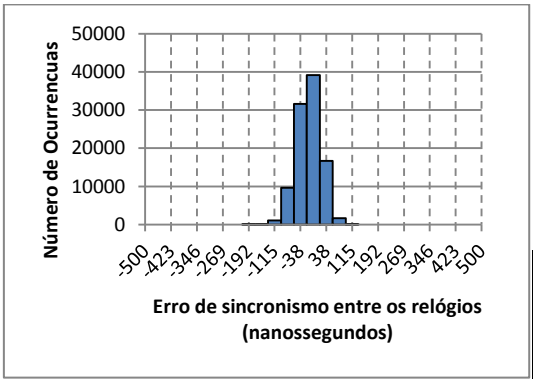

Tabela 4.32 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
-13.16 ns	36.97 ns	-192 ns	115 ns
-0.34 ticks	0.97 ticks	-5 ticks	3 ticks

Figura 4.65 – Distribuição do erro de sincronismo para um *SyncPeriod* de 5 segundos

4.1.7.4 Teste com SyncPeriod de 10 segundos

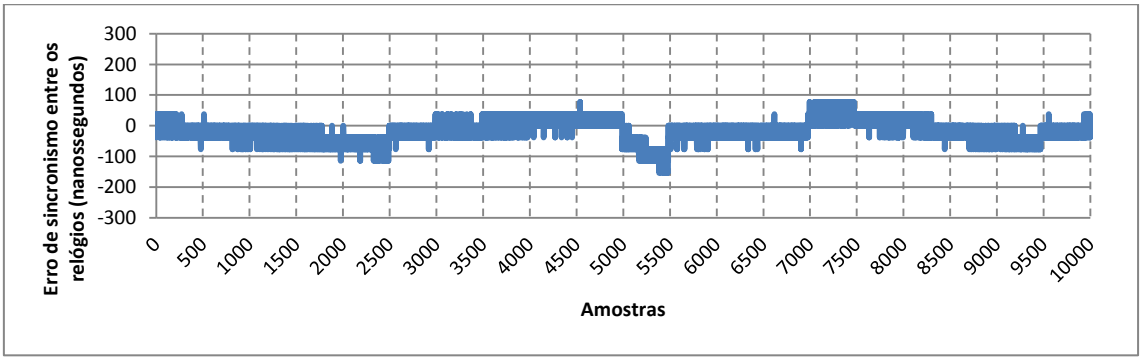

Figura 4.66 – Erro de sincronismo para um *SyncPeriod* de 10 segundos

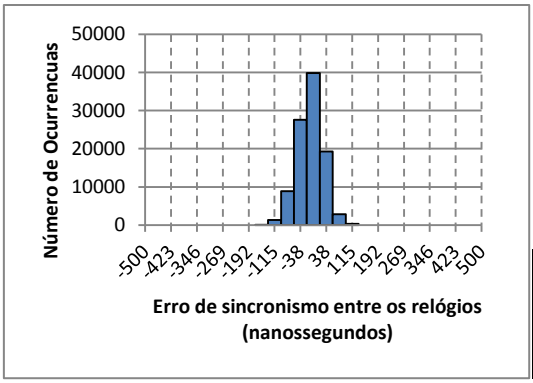

Tabela 4.33 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
-9.03 ns	38.85 ns	-154 ns	115 ns
-0.24 ticks	1.02 ticks	-4 ticks	3 ticks

Figura 4.67 – Distribuição do erro de sincronismo para um *SyncPeriod* de 10 segundos

4.1.7.5 Teste com SyncPeriod de 30 segundos

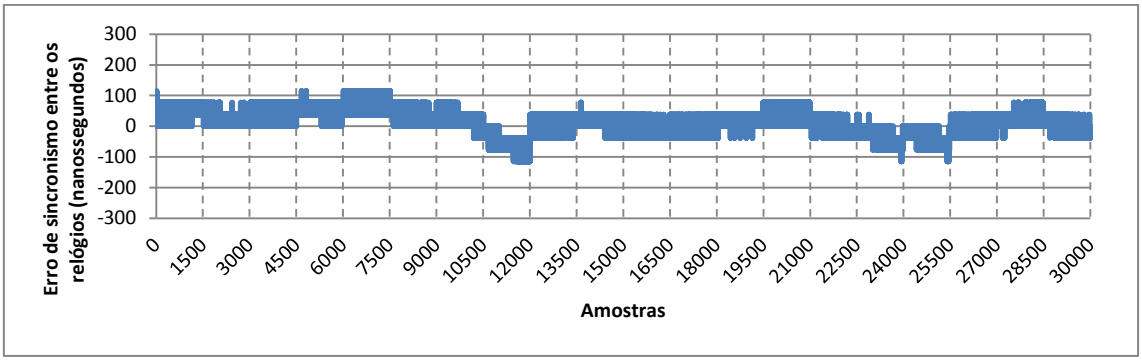

Figura 4.68 – Erro de sincronismo para um *SyncPeriod* de 30 segundos

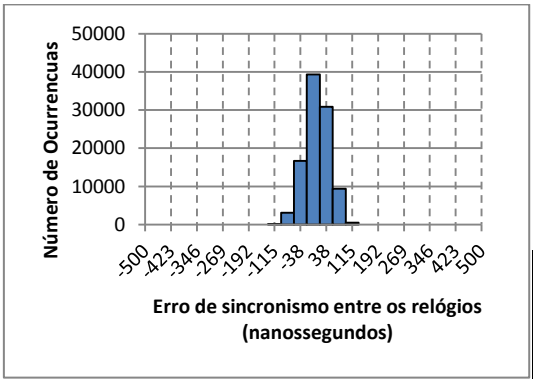

Tabela 4.34 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
10.59 ns	37.39 ns	-115 ns	115 ns
0.28 ticks	0.98 ticks	-3 ticks	3 ticks

Figura 4.69 – Distribuição do erro de sincronismo para um *SyncPeriod* de 30 segundos

4.1.7.6 Teste com SyncPeriod de 60 segundos

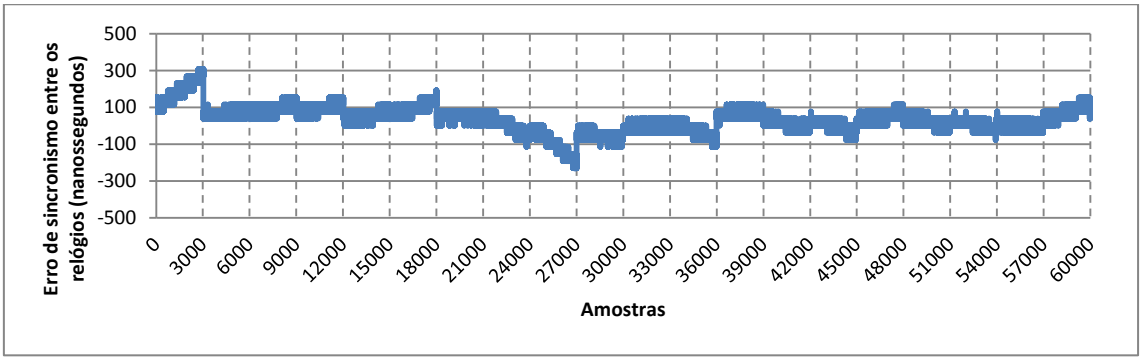

Figura 4.70 – Erro de sincronismo para um *SyncPeriod* de 60 segundos

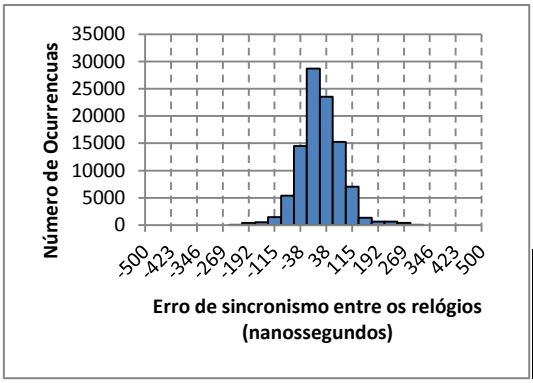

Tabela 4.35 – Resultados obtidos

Média	Desvio Padrão	Mínimo	Máximo
21.83 ns	63.10 ns	-231 ns	308 ns
0.57 ticks	1.64 ticks	-6 ticks	8 ticks

Figura 4.71 – Distribuição do erro de sincronismo para um *SyncPeriod* de 60 segundos

4.1.7.7 Análise de Resultados

Com estes resultados finais obteve-se um desvio padrão compreendido entre os 37 e os 40 nanossegundos para os períodos de 1, 2, 5, 10 e 30 segundos. Nestes casos o erro máximo encontra-se inferior a 200 nanossegundos, sendo o seu valor máximo no teste com 5 segundos onde se obteve um erro máximo de 192 nanossegundos. No caso do teste com período de 60 segundos, o desvio padrão subi-o para 62,53 nanossegundos e o erro máximo para 308 nanossegundos.

Comparando com os resultados obtido no capítulo 4.1.4, verifica-se uma redução do desvio padrão para todos os períodos de sincronização usados. No caso dos períodos de sincronismo de 1, 2 e 5 segundos, o seu valor reduzi-o praticamente para metade devido ao aumento da estabilidade dos valores de *OffsetFromMaster* calculados. Para os períodos de sincronismo de 10 e 30 segundos, que no caso anterior apresentavam valores de desvio padrão que aumentavam devido à influência da variação da temperatura, verifica-se uma melhor estabilidade, mantendo o seu desvio padrão inferior a 40 nanossegundos. No caso do teste com período de 60 segundos, o seu desvio padrão aumentou para 63 nanossegundos, sendo mesmo assim, inferior a mais de metade do obtido no teste do capítulo 4.1.4.7.

4.2 Reconhecimento

Com os resultados obtidos no primeiro protótipo foi possível publicar um *paper* e participar na conferência *WisNet* 2013 acolhida pelo evento *RWW* 2013.

Com os resultados obtidos com este segundo protótipo, pretende-se voltar a submeter um *paper* na mesma conferência *WisNet* que vai ser recebida novamente pelo RWW [20]. A conferência vai ocorrer de 19 a 24 de Janeiro de 2014 em Newport Beach, Califórnia, Estados Unidos da América.

5 Conclusões e Trabalho Futuro

Numa fase inicial deste estudo identificou-se o cristal usado nos microcontroladores das unidades da rede LoWPAN como o originador de erro de sincronismo do relógio. Este erro por outro lado pode ser provocado por 3 causas distintas, sendo elas a tolerância da frequência do cristal, a sua estabilidades com a temperatura e o valor dos condensadores de carga usados. Em relação à tolerância do cristal e aos condensador de carga, o seu impacto é constante ao longo do tempo enquanto que no caso da estabilidade com a temperatura o seu impacto está diretamente ligado à variação da temperatura, sendo necessário algum tipo de isolamento térmico do cristal para atenuar o seu efeito.

Na fase de implementação, foi estruturado um algoritmo composto por 4 blocos funcionais da unidade. O Bloco RF responsável por manter o canal de comunicação, o Bloco de Sincronismo responsável pela gestão de troca de mensagens de sincronismo e pelo cálculo dos seus parâmetros de correção, o Bloco de Gestão de Eventos responsável pela manutenção do relógio, registo do seu valor e execução de eventos sincronizados, e o Bloco

de Aplicação onde pode ser implementado uma aplicação que pode usufruir de um relógio sincronizado e de um canal de comunicação com a restante rede. Na primeira validação do sistema, constatou-se como era esperado, que o erro de sincronismo cresce linearmente com o período de sincronismo usado, conseguindo-se estimar um desvio de 4.09 microssegundos de erro por segundo, entre os relógios da unidade de referência e da unidade sincronizada. Para eliminar esta dependência, foi implementado um algoritmo de sintonização baseado na correção periódica do relógio da unidade sincronizada com valores de ajuste pequenos. Na validação final observou-se que o erro deixou de crescer linearmente com o período e passou a estabilizar com um valor máximo inferior a 5 microssegundos. Estes resultados permitiram a publicação de um *paper* numa conferência internacional.

Na fase de otimização, constatou-se que o erro obtido anteriormente era superior a 100 vezes a resolução do relógio da unidade, sendo possível obter melhores resultados com a mesma unidade. Para tal foram realizadas 7 otimizações do sistema que passaram pelo registo dos tempos de envio e receção de mensagens diretamente por *hardware*, o registo e execução de eventos também por *hardware*, a diminuição do valor de correção de sintonização para 38 nanossegundos (1 pulso de relógio), o cálculo do período de sintonização de forma adaptativa, o cálculo dos parâmetros de sincronismo através da média de 32 valores intermédios e um ajuste do número de correções de sintonização de uma forma mais rápida. Este conjunto de otimizações permitiu reduzir o erro máximo para um valor inferior a 200 nanossegundos para períodos de sincronismo até 30 segundos e um valor inferior a 400 nanossegundos para um período de 60 segundos. Comparando estes resultados com as implementações referidas no capítulo 2.2.2.3, podemos constatar que usando uma unidade equivalente à unidade de baixo custo [13], foi obtido resultado idênticos á implementação de alta precisão [14], usando períodos de sincronismo 300 vezes maiores se considerarmos o caso dos resultados obtidos com período de 30 segundos. Isto valida por completo a qualidade do trabalho desenvolvido e o valor que esta solução trás para o crescimento das redes LoWPAN na sua jornada em direção da utopia da *Internet of Things*.

Em relação a trabalho futuro, existem duas vertentes que podem ser exploradas num terceiro protótipo.

Uma é a migração da implementação atual para uma rede *mesh* como o 6LoWPAN. Neste caso o sistema teria de ser expandido para suportar que unidades sincronizadas se tornassem também unidades de referência para outras unidades e teria de ser adicionado a capacidade de se escolher a unidade com melhor referência no caso de uma situação idêntica à da Figura 5.1, onde a unidade C pode se sincronizar através da unidade A ou da unidade B

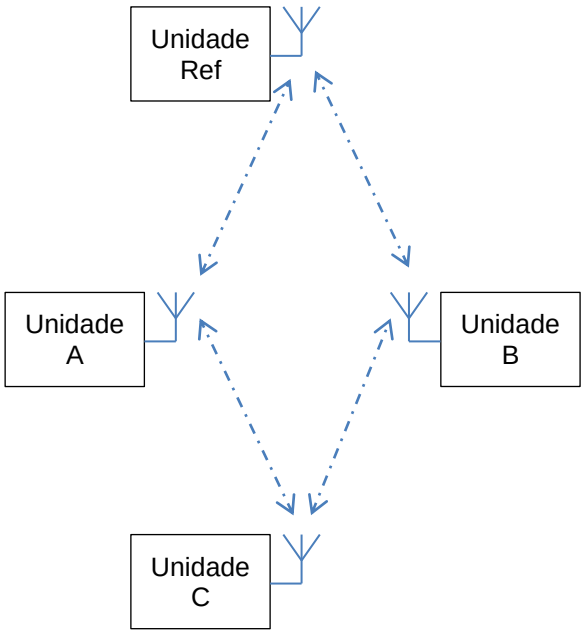

Figura 5.1 – Rede LoWPAN numa configuração *mesh*

A outra possibilidade será melhorar o próprio sistema de sintonização do relógio, mudando o processo atual para um processo por *hardware*. Isto é possível através da influência do condensador de carga na frequência do cristal, que pode ser controlado se for adicionado uma DAC juntamente com um *varicap* ao circuito do cristal como apresentado na Figura 5.2.

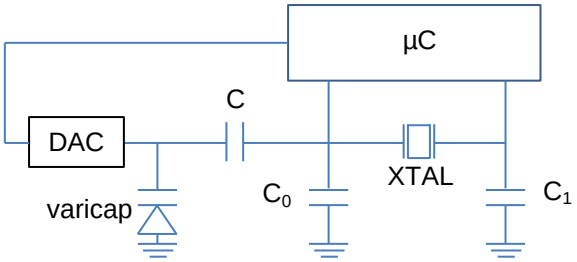

Figura 5.2 – Capacidade de carga controlada por um DAC e um varicap

Anexo A

Synchronization and Syntonization of Wireless Sensor Networks

João Reis and Nuno Borges Carvalho

Instituto de Telecomunicações, Campus Universitário de Santiago, Aveiro, Portugal

***Abstract* — With the purpose of providing support for WSN solutions, that require the execution of coordinated events between multiple device units, a time synchronization protocol based on a simplified version of the IEEE 1588 standard was developed. Additionally, a clock syntonization algorithm to reduce the offset error between synchronization points was also developed, allowing the increase of the synchronization period. Practical results show offset errors lower than 5 microseconds for a synchronization period of 30 seconds.**

***Index Terms* — IEEE 1588, clock synchronization, clock syntonization, wireless sensor network.**

I. Introduction

Wireless Sensor Network (WSN) is a wireless distributed system comprised of independent device units that cooperate in the execution of different tasks. In several of its applications, it is critical the presence of a global time reference shared by the entire network, allowing that events detected in these units can be properly interpreted and execute coordinated events between them. Depending on the application requirements, different degrees of synchronization accuracy is demanded.

WSN units are in general comprised by devices with limited energy power sources, low bandwidth communication channels and reduced processing capability, leading to the need of simple but effective synchronization protocols. In literature, several methods, protocols and implementations addressed to solve this problem in WSNs can be found [1].

Taking into account that one of the main points of energy consumption in WSN lays in the radio operations of the devices, it is important for low power applications to achieve a low message transmission rate needed for the clock synchronization process without increasing the synchronization error.

Targeting this issue, a synchronization protocol based on IEEE 1588 standard was developed, assisted with a syntonization algorithm to minimize de rate of messages exchange.

This paper is structured in the following way. In Section II a time synchronization analysis of the problem and its solution is presented, followed by the description of two previous implementations of the IEEE 1588 protocol and their achievements in Section III. Our implementation is revealed in Section IV and in Section V the results are presented. In Section VI the paper is summarized.

II. Time Synchronization Analysis

Typically the WSN units are developed with low cost crystal oscillators with tolerances ranging from 10ppm to 100ppm. This value can easily be inferred as the number of microsecond's deviation after 1 second. Without a synchronization protocol, this deviation will increase over time as shown in Fig. 1.

The solution to this, is to perform periodic clock synchronization between the units, known as synchronization points, in order to correct this offset error. In Fig. 2 it is shown that the maximum offset error is correlated to the crystal oscillator tolerance and the synchronization period

$$MaxOffsetError(us) = SyncPeriod(s) \times CrystalTolerance(ppm) \quad (1)$$

The synchronization points are comprised by a set of messages exchanged between the units of the network, defined by the synchronization protocol used. Some caution should be taken in this process to minimize nondeterministic variations in order to obtain a good degree of accuracy.

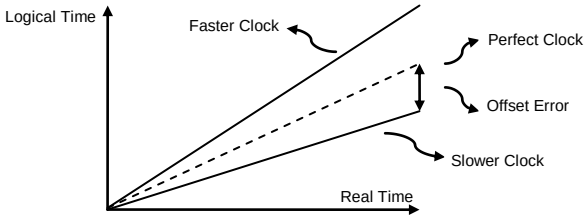

Fig. 1. Increase of the clock offset error over time

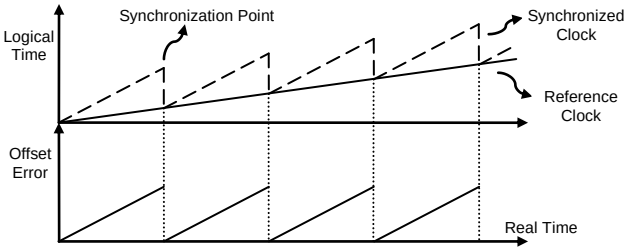

Fig. 2. Impact of the synchronization over the offset error

III. IEEE 1588 PROTOCOL

The IEEE 1588 standard [2] was developed thinking in distributed networks requiring a better precision than offered by NTP but that cannot support a GPS module per unit due to indoor application or costs constrains. With this protocol, it is possible to create a distributed network using a single GPS module to provide a time reference for the network while maintaining accuracies with orders of magnitude of nanoseconds.

Two previous implementations of this protocol were taken as reference in the development of our solution. In [3] a unit consisting of a TI CC2420 transceiver, a Freescale's MCF5235 microcontroller and a 37.5MHz crystal oscillator with a tolerance of 1.5ppm was implemented. The coprocessor of the MCF5235 was responsible for generating the time of reception and transmission of messages while the main core in handling the messages. With this unit, the authors were able to obtain a maximum offset error of 160 nanoseconds and an average offset error of 19.3 nanoseconds with a synchronization period of 100 milliseconds. In [4] the main focus was the development of a low consumption and low cost unit, using a SoC (transceiver + microcontroller) TI CC2430 for main processing and the microcontroller 18F2620 together with a 16MHz crystal oscillator to generate the time of reception and transmission of messages. With a synchronization period of 2 seconds they show an offset error of 10 microseconds.

IV. SYSTEM IMPLEMENTATION

Our system was developed on a development board with the TI CC1110 chip. This SOC consists of an 8-bit 8051 processor and a 433 MHz transceiver. The crystal oscillator used has a frequency of 26MHz and a tolerance of 10 ppm. Unlike the implementations mentioned earlier, we have a single core to perform the tasks of message handling, time stamping and event trigger.

A. Simplified Implementation of the IEEE 1588 standard

The standard defines different types of messages and its structures. Due to the limitations of the WSN units, a simple version comprised by a set of 4 messages as presented in Fig. 3 was developed. Each one composed by the minimum information necessary to proceed with the time synchronization.

1) Sync: The synchronization process starts by sending a broadcast message that the remaining units use to register the time t_2.

2) Follow_Up: Simultaneously, the reference unit registers the time t_1 after sending the Sync message and sends its value in the Follow_Up message.

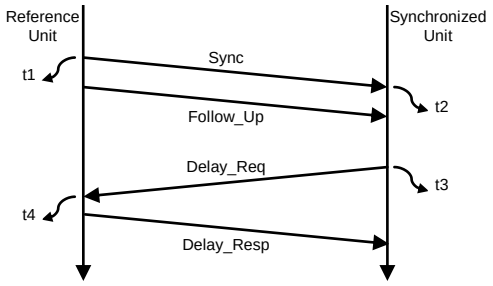

Fig. 3. Synchronization message exchange

3) Delay_Req: Later the synchronized unit sends the Delay_Req message and registers the time t_3.

4) Delay_Resp: When received the Delay_Req message, the t_4 is registered and transmitted back in the Delay_Resp message.

Assuming a symmetric communication path delay, its value can be obtained by

$$PathDelay = \frac{(t_2 - t_1) + (t_4 - t_3)}{2} \quad (2)$$

and the offset between the clock of the reference unit and the synchronized unit by

$$OffsetFromMaster = (t_2 - t_1) - PathDelay\ . \quad (3)$$

B. Clock Syntonization

The clock syntonization was performed by conducting small periodic corrections as shown in Fig. 4.

The number of necessary corrections can be obtained by

$$NSyntCorr = Offset / SyntCorrValue \quad (4)$$

and its period by

$$SyntPeriod = SyncPeriod / NSyntCorr\ . \quad (5)$$

The correction value used depends essentially on the level of accuracy required by the application. The smaller this value, the greater will be the number of adjustments needed to be performed between synchronization points.

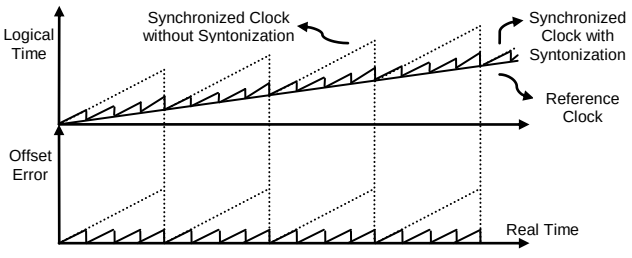

Fig. 4. Improvement of clock offset error when using a syntonization algorithm

C. Internal Architecture

Fig. 5 shows the structure of the internal architecture implemented in the controller:

1) RF Protocol: To enable connectivity between the units, we used the SimpliciTI protocol from Texas Instruments, allowing the creation of a star network. Some changes have been made to permit recording the time of transmission and reception of messages as close as possible to its occurrence.

2) TimeSync: This module is responsible for managing the exchange of synchronization messages. This way the process of clock synchronization is performed transparently to the rest of the application.

3) TimeTrigger: To enable the creation of coordinated events between units, we implemented the TimeTrigger module, capable of generating events with a resolution of 1 millisecond, but aligned in phase with the remaining units. This module is also responsible for maintaining the clock and adjusts the syntonization period.

4) Main Application: In a higher layer we can implement the code of a given application, with access to the communication channel to exchange messages, generate timestamps for events detected and schedule events to be triggered in coordination with other units.

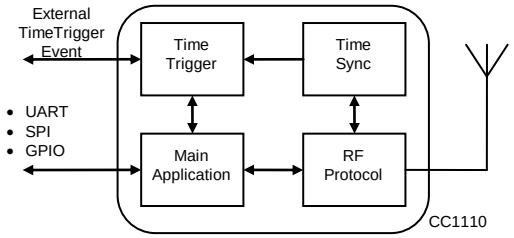

Fig. 5. Block Diagram of the Implementation

V. System Evaluation

Using a similar setup as in [3], a reference unit and a synchronized unit were connected to a pulse generator. When receiving a pulse, each unit registers a timestamp and sends it to the PC through an USB-UART cable. The evaluation was performed with a synchronization period of 5 seconds and 30 seconds. The syntonization correction value was set to 1 microsecond and the pulse generator to 100 milliseconds. 2000 timestamp were taken for each round. In Table I the average and standard deviation of the timestamps obtained is presented and in Fig. 6 it is shown a histogram representation of the timestamps with a synchronization period of 30 seconds.

TABLE I
EVALUATION RESULTS

Sync Period	Average	Standard Deviation	Minimum	Maximum
5 seconds	-0,235 us	0,957079068 us	-4,538 us	2,385 us
30 seconds	0,563 us	0,798037367 us	-2,846 us	3,231 us

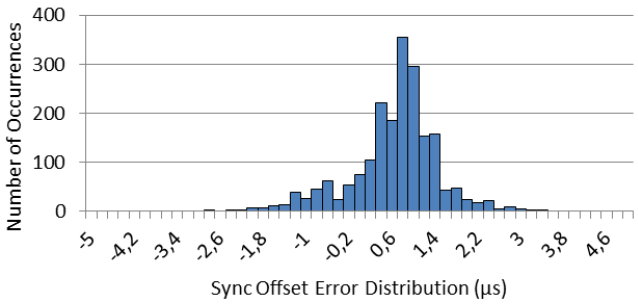

Fig. 6. Sync Error distribution for a 30 seconds Sync Period

VI. Conclusion

A time synchronization protocol based in the IEEE 1588 standard was implemented in a single chip WSN unit. With a syntonization algorithm, it was possible to increase the time synchronization period and decrease the offset error between synchronization points, lowering the units' power consumption.

References

[1] S. Rahamatkar, A. Agarwal, N. Kumar, "Analysis and Comparative Study of Clock Synchronization Schemes in Wireless Sensor Networks", *Int. Journal on Computer Science and Engineering (IJCSE)*, vol. 02, no. 03, pp. 536-541, 2010.

[2] "*IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*", IEEE Std. 1588-2008 (Revision of IEEE Std 1588-2002), 2008.

[3] H. Cho, J. Jung, B. Cho, Y. Jin, S.-W. Lee, Y. Baek, "Precision Time Synchronization using IEEE 1588 for Wireless Sensor Networks", *in Proc. Int. Conf. on Computational Science and Engineering*, pp. 579-586, 2009.

[4] D. Wobschall, Y. Ma, "Synchronization of wireless sensor networks using a modified IEEE 1588 protocol", *2010 Int. IEEE Symp. on Precision Clock Synchronization for Measurement Control and Communication (ISPCS)*, pp. 67-70, 2010.

Referências

[1] "*IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks Specific Requirements Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)*", IEEE Std. 802.15.4-2003, 2003

[2] *WirelessHart*
Web: http://www.hartcomm.org/protocol/wihart/wireless_technology.html

[3] *ZigBee*
Web: http://www.zigbee.org/

[4] *6LoWPAN*
Web: http://datatracker.ietf.org/wg/6lowpan/

[5] *IETF*
Web: http://www.ietf.org/

[6] *ISA100*
Web: http://www.isa.org/isa100

[7] S. Rahamatkar, A. Agarwal, N. Kumar, "*Analysis and Comparative Study of Clock Synchronization Schemesin Wireless Sensor Networks*", Int. Journal on Computer Science and Engineering (IJCSE), vol. 02, no. 03, pp. 536-541, 2010.

[8] I. Rhee, J. Lee, J. Kim, E. Serpedin, Y. Wu, "*Clock Synchronization in Wireless Sensor Networks: An Overview*", Sensors 2009, pp. 56-85, 2009

[9] F. Sivrikaya, B. Yener, "*Time Synchronization in Sensor Networks: A Survey: An Overview*", Network, IEEE, vol.18, no.4, pp.45-50, July-Aug. 2004

[10] J. Elson, L. Girod, and D. Estrin, *"Fine-Grained Time Synchronization usingReference Broadcasts*", Proc. 5th Symp. Op. Sys. Design and Implementation, Boston, MA, Dec. 2002.

[11] S. Ganeriwal, R. Kumar, and M. Srivastava, "*Timing Sync Protocol for Sensor Networks*," ACM SenSys, Los Angeles, CA, Nov. 2003.

[12] "*IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*", IEEE Std. 1588-2008 (Revision of IEEE Std 1588-2002), 2008

[13] D. Wobschall, Y. Ma, "*Synchronization of Wireless Sensor Networks Using a Modified IEEE 1588 Protocol*", Precision Clock Synchronization for Measurement Control and Communication (ISPCS), 2010 International IEEE Symposium on, pp.67-70, Sept. 27 2010-Oct. 1 2010.

[14] H. Cho, J. Jung, B. Cho, Y. Jin, S.-W. Lee, Y. Baek, "*Precision Time Synchronization using IEEE 1588 for Wireless Sensor Networks*", in Proc. Int. Conf. on Computational Science and Engineering, pp. 579-586, 2009

[15] *SimpliciTI*

Web: http://www.ti.com/corp/docs/landing/simpliciTI/index.htm

[16] Reis, J.; Carvalho, N.B., "*Synchronization and syntonization of wireless sensor networks*", Wireless Sensors and Sensor Networks (WiSNet), 2013 IEEE Topical Conference on, pp.151-153, 20-23 Jan. 2013

[17] *RWW 2013*

Web: http://www.radiowirelessweek.org/2013/

[18] *CC1110F32 Datasheet*

Web: http://www.ti.com/product/cc1110f32

[19] *Cristal Datasheet*

Web: http://www.txccrystal.com/images/pdf/7b-accuracy.pdf

[20] *RWW 2014*

Web: http://www.radiowirelessweek.org/