UNIVERSIDADE DA BEIRA INTERIOR

Engenharia

Desenvolvimento do Módulo de Interface de Transdutor Inteligente: Norma ISO/IEC/IEEE-21451

Versão final após defesa

Ntunitangua René Pindi

Dissertação para obtenção do Grau de Mestre em

Engenharia Eletrotécnica e de Computadores

(2º ciclo de estudos)

Orientador: Prof. Doutor António Eduardo Vitória do Espírito Santos

Co-orientador: Prof. Doutor Bruno Ribeiro

Covilhã, julho de 2018

Dedicatória

Dedico este trabalho a minha família, a Escola superior politécnica do Cuanza Norte, aos colegas e amigos.

Agradecimentos

A Deus todo poderoso, pela vida, pelo amor, pela misericórdia, por cuidar de mim nos momentos bons e difíceis.

A minha esposa, Sandra Catarina Gueve Pindi, pela paciência e amparos nos momentos difíceis ajudando a superá-los.

A minha família pelo apoio integral em todos as circunstâncias, pelo estímulo manifestado na minha formação.

Ao meu orientador, Professor, Doutor António Eduardo Vitória do Espírito Santos, pela confiança, pela orientação e por contribuir para o meu crescimento profissional e também como pessoa.

Ao meu Co-orientador, Professor, Doutor Bruno Ribeiro, pela disponibilidade, pela sinceridade, pelas contribuições e sugestões.

A todos os Professores do curso do mestrado de Engenharia eletrotécnica e de computador da Universidade da Beira Interior.

Aos colegas, Bruno Silva, Dário Justo, Gaspar António, Analciso Roodino, Rooderson Andrade, Gildo Luemba, Pedro Patrício, Arlindo Isaac, Sofia Ambrósio e César Lima pelo partilho e disposição.

Este trabalho foi desenvolvido no âmbito do projeto Energy and Water Systems Integration and Management (EdgeWise), apoiado pela Fundação para a Ciência e Tecnologia (FCT), com a referência ERANETMED/0004/2014, enquadrado na iniciativa ERANETMET dos Estados Membros, Países Associados e Países Parceiros Mediterrâneos (Projeto com o ID eranetmed_nexus-14-044).

Resumo

Atualmente monitorar, controlar e gerir processos em tempo real em ambiente industrial é um fator indispensável. Porém cada vez mais, as redes de transdutores têm conquistado o seu espaço neste mundo. Apesar do transdutor ser um elemento chave para estas redes pode também levar, ao mesmo tempo, à não interoperabilidade. Este fator constitui uma das características fundamentais das redes dos transdutores.

Este trabalho consiste no desenvolvimento de um nodo transdutor (TIM) em conformidade com o padrão ISO/IEC/IEEE 21451-0, capaz de comunicar-se com qualquer nodo da rede (NCAP) implementado em conformidade com o mesmo padrão e, o módulo de comunicação baseado no padrão IEEE P1452-2 UART. A comunicação entre o nodo transdutor e o nodo da rede é baseado no modelo cliente-servidor. O nodo do transdutor foi desenvolvimento num microcontrolador Ultra-Lower-Power MSP-EXP430F5529 da Texas Instruments utilizando o Code Composer Studio 6.0.

Palavras-chave

TIM, Transdutores, IEEE 1451-0, IEEE 1451-2, MSP430F5529.

Abstract

Nowadays, monitoring, controlling and managing in real time industrial processes is an indispensable task. But more and more, transducer's networks have conquered their space in this world. The transducer is a key element for these networks and, at the same time, a factor that can result in non-interoperability. This factor is one of the fundamental characteristics of transducer networks.

This work consists in the development of a transducer node (TIM) in compliance with the ISO/IEC/IEEE 21451-0 standard, capable of communicating with any node of the network through the NCAP implemented in full compliance with the same standard and communication module based on the IEEE P1452-2 UART standard. The communication between the transducer node and the network node is based on the client-server model. The transducer node was developed in a Texas Instruments Ultra-Lower-Power MSP-EXP430F5529 microcontroller using Code Composer Studio 6.0.

Keywords

TIM, Transducer, IEEE 1451-0, IEEE 1451-2, MSP430F5529.

Índice

Lista de Figuras

Lista de tabelas

Lista de Acrónimos

ADC	*Analog to Digital Converter*
ASIC	Application Specific Integrated Circuit
API	Application program interface
CAN	Controller Area Network
CSS	Code Composer Studio
DAC	*Digital to Analog Converter*
EVM	Error Vector Magnitude
FPGA	Field-Programmable Gate Array
HAL	Hardware Abstract Layer
I2C	Inter-Integrated Circuit
IBSG	Cisco Internet Business Solutions Group
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronic Engineers
IoT	Internet of Things
ISO	International Organization for Standardization
JTAG	Joint Test Action Group
LCD	Liquid Crystal Display
LED	Light Emitting Diode
lsb	Least Significant bit
MCU	Unit microcontroller
msb	Most Significant bit
NCAP	Network Capable Application Processor
RF	Radio Frequency
RFID	Radio Frequency Identification
SPI	Serial Peripheral Interface

STIM	Smart Transducer Interface Module
TEDS	Transducer Electronic Data Sheet
TI	Texas Instruments
TII	Transducer Independent Interface
UART	Universal Asynchronous Receiver/ Transmitter
USB	Universal Serial Bus
WCCC	Write Channel Control Command
WGCC	Write global Control Command
Wi-Fi	Wireless Fidelity

Capítulo 1 - Introdução

1.1. Enquadramento

As culturas têm vivenciado uma evolução tecnológica, particularmente na última década, estimulados pelo crescimento da Internet e da microeletrónica, onde a grande disponibilidades e variedades de transdutores é um fator a ter em conta na automação industrial. Os transdutores são os principais atores no cenário da automação industrial e, de acordo com a visão descrita em [1], o fato de vivemos atualmente num mundo de informações e comunicações exige, não somente que as informações relacionadas com os transdutores num determinado ambiente sejam transmitidas, de um ponto a outro, como também monitorizar e compartilhadas. Nasce, assim, o conceito de rede de transdutores ou simplesmente de redes de instrumentação. No entanto, a ampla variedade de redes de campo e protocolos existentes ou a criação de soluções independentes pelos fabricantes, trouxeram problemas relacionados com a flexibilidade e soluções mais caras. Para resolver estes problemas a família de padrões ISO/IEC/IEEE 21451 propõem um conjunto padronizado de interfaces de hardware e software que funcionam como *plug and play*, permitindo que diferentes transdutores sejam conectados à mesma rede [2]. É importante destacar que a flexibilidade em instrumentação está associada aos conceitos de interoperabilidade, conectividade e o *plug and play*. A interoperabilidade é a possibilidade de um elemento hardware/software, de um determinado fabricante, ser substituído por um outro de fabricante diferente sem que haja qualquer perturbação. A conectividade e a possibilidade de integração de diferentes equipamentos ou dispositivos através da tecnologia de rede. O *plug and play* é possibilidade de evitar a reconfiguração do sistema sempre que é adicionado um novo dispositivo.

1.2. Motivação

O desenvolvimento de uma rede de transdutores que permita detetar, visualizar, registar, processar, converter dados em informações e monitorar os elementos que compõem uma unidade industrial automatizado, é um processo complexo, pois envolve diferentes áreas do saber, como por exemplo, instrumentação, telecomunicação, engenharia de software, eletrónica e outras. Sendo assim, ligar o transdutor numa rede não é uma tarefa fácil, pois é necessário ter um domínio elevado de conhecimento em conceitos e ambientes de atuação variados. Nos dias de hoje, o campo de atuação dos transdutores é cada vez mais inclusivo. Encontra aplicabilidade nas mais diversas áreas, como, por exemplo, em biomedicina, aeronáutica, telemedicina, petroquímica, automação residencial e outras. Atualmente, a conexão de transdutores numa rede de comunicação é realizada de forma proprietária, tornando as aplicações pouco flexíveis em termo de expansão e com dependência de um único fabricante na manutenção [1]. No entanto, ligar o transdutor numa rede de comunicação de forma flexível e padronizada torna o sistema de monitoramento e controlo

distribuído mais praticável. O desenvolvimento do módulo de interface do transdutor observando as diretrizes da norma ISO/IEC/IEE 21451 e a filosofia de código aberto, as indústrias poderão obter inúmeros benefícios, tais como: redução de custos de sistemas de instrumentação devido à redução de complexidade, maior facilidade e flexibilidade para transmitir informações através de uma rede de comunicação e maior facilidade para ampliar os sistemas sem efetuar grandes configurações [1].

1.3. Objetivos

Construir uma plataforma que suporte o desenvolvimento de sensores inteligentes que observe a norma ISO/IEC/IEEE 21451. Esta norma possui dois grandes elementos: a *Network Capable Application Processor* (NCAP) e a *Transducer Interface Module* (TIM). Esta última é responsável por implementar as funcionalidades de medida. Este trabalho pretende desenvolver a TIM numa plataforma a *Experimenter's Board* para microcontroladores de baixo consumo energético MSP430F5529. O resultado deve ser capaz de se interligar com a NCAP para demonstrar o funcionamento do sensor em rede. A solução vai ser desenvolvida em torno de um processador de baixo consumo energético da família MSP430. A linguagem de programação a utilizar na implementação será o C/C++. O resultado final deve ser demonstrado em funcionamento.

1.4. Organização da tese

O presente trabalho está estruturado em cinco capítulos. O primeiro capítulo tem o seu foco na importância dos transdutores nas redes de campo ou instrumentação. O segundo, resume-se na necessidade de normas no contexto da internet das coisas e dos transdutores inteligentes, bem como das tecnologias utilizadas e o módulo da TIM. O terceiro aborda as especificações funcionais do transdutor inteligente. O quarto concentra-se nos serviços implementados e no módulo de comunicação utilizada. Finaliza-se com apresentação dos resultados.

Capítulo 2 - Padrão ISO/IEC/IEEE 21451

2.1. A necessidade de normas no contexto da Internet das coisas e de sensores inteligentes

Hoje em dia, a vida cotidiana vem sofrendo intervenções profundas nas suas infraestruturas. No entanto, cidades, casas, indústrias, carros, e eletrodomésticos estão a ser equipados com sensores que permitem, em tempo real, administrar o ambiente das pessoas e os objetos através da Internet. Este novo paradigma promove a relação entre objetos, ou entre objetos e seres humanos, a Interligarem-se através de diferentes mecanismos de comunicação, recebe a designação de *Internet das Coisas ou simplesmente Internet of Things (IoT)*. De acordo com o Cisco Internet Business Solutions Group (IBSG), a IoT acontece no momento exato em que foram ligados à Internet mais coisas, ou objetos, do que pessoas. Com base nessa definição, a IoT não existia antes de 2008, pois o número de dispositivos ligados à Internet era inferior, com relação ao número da população mundial. No entanto, em 2010 graças ao crescimento explosivo do número de Smartphones e Tablet, a quantidades de dispositivos ligados à internet, pela primeira vez havia superado o número de pessoas ligados à internet em conformidade com a Tabela 1.

Tabela 1 - Comparação entre dispositivos conectados e população mundial.

Ano	2003	2010	2015	2020
População mundial (mil milhões):	6,3	6,8	7,2	7,6
Dispositivos conectados (milhões):	500	12,5	25	50
Dispositivos conectados por pessoa:	0.08	1.84	3.47	6.58

A IoT pode ser entendida como a interação de todos os objetos que fazem parte da nossa vida cotidiana usando como meio de comunicação a Internet, ou seja, quando os objetos podem sentir o ambiente e se comunicar automaticamente, sem intervenção humana atualizando-se com as tarefas do dia-a-dia [2]. Este conceito introduziu na Internet um protagonista fundamental: Transdutores Inteligentes, o que possibilitou uma evolução efetiva da Internet, um salto revolucionário com potencial para melhorar significativamente a forma como os humanos vivem. Porém, deu origem a um outro problema, a conectividade entre interface dos transdutores Inteligentes e de diferentes tecnologias de rede existentes. No entanto, para resolver esse problema foi introduzido o padrão IEEE 1451 que teve origem no final da década de 1990. Esta norma propõe um conjunto de protocolos e de interfaces de comunicação para interligar transdutores através de uma rede, ao mesmo tempo que oferece a independência necessária entre a natureza de um nodo de rede, envolvendo transdutores, e os protocolos de comunicação de rede.

O transdutor é um dispositivo capaz de transformar um tipo de energia em outro. Ele pode ser formado por sensores ou atuadores e, em certos casos, por ambos: nesse sentido um sensor é

um transdutor que produz um sinal elétrico proporcional à variação de um parâmetro físico, químico ou biológico, enquanto um atuador é um transdutor que recebe sinal proveniente do controlador e agem sobre o sistema controlado, ou seja, quando se pretende a variação de parâmetros de um processo a ser controlado [3]. No entanto, neste trabalho usar-se-á o termo transdutor para referenciar sensores e atuadores. Segundo Al-Ali *et al* em 2005, os transdutores podem ser vistos em duas grandes gerações, ordinário ou inteligente, respetivamente. Na visão do mesmo, os transdutores ordinários são aqueles que necessitam de um circuito externo dedicado para realizar a análise de sinal, compensação de erro e de filtragem dos dados monitorados, enquanto que os transdutores inteligentes são aqueles que podem ser programados para atender novos requisitos do utilizador e que possuem um circuito interno que realiza o condicionamento do sinal e proporciona um processamento específico dos dados no sistema. Um transdutor inteligente deve possuir capacidade de processamento e de diagnóstico. A capacidade de processamento do transdutor inteligente faz com que o transdutor possa tomar decisões em tempo real, enquanto que a capacidade de diagnóstico permite identificar e gerenciar o consumo de energia, testar a comunicação com outos transdutor ou dispositivos ligados a ele.

Segundo [4],um transdutor inteligente é aquele que possui funcionalidades além daquelas que são necessárias para gerar a correta representação dos dados adquiridos ou das medidas controladas.

Um transdutor inteligente tipicamente é composto por microcontroladores, memória, fonte de energia, dispositivo de condicionamento do sinal, e elemento sensor. O microcontrolador é um dispositivo que pode ser programado com finalidades específicas. É responsável por realizar todo o processamento necessário e atribuir inteligência ao sensor. A memória é o dispositivo responsável por armazenar dados e o programa a ser executado no microcontrolador. O elemento sensor é o dispositivo responsável por detetar, captar e realizar a medição e o controlo dos fenómenos físicos (variáveis) a serem monitorizados. O dispositivo de condicionamento do sinal é responsável por realizar a conversão de domínio e posterior o armazenamento na forma digital. A fonte de energia é o bloco funcional responsável por fornecer a energia necessária para que o transdutor possa realizar as suas funções.

A norma ISO/IEC/IEEE 21451 resultou da aceitação por parte da ISO e da IEC do padrão IEEE 1451 que permite a interligação independente de interface de transdutor inteligente através de uma rede de utilizador. No entanto, o padrão foi desenvolvido como um conjunto de especificações de domínio aberto de acordo com o consenso da indústria. A família ISO/IEC/IEEE 21451 tem como objetivo:

- ✓ Definir um conjunto de interfaces de comunicação comuns para ligar transdutores inteligentes (sensores ou atuadores) a sistemas, instrumentação (redes de

controlo) e redes baseados em microcontroladores em um ambiente independente de rede [5].

A norma ISO/IEC/IEEE 21451 é constituída por oito membros, sendo que, alguns, estão aprovados, embora outros estejam em fases de revisão, ou mesmo em desenvolvimento conforme listado na Tabela 2. O asterisco que acompanha o estado atual significa que a norma se encontra em revisão.

Tabela 2 - Família de padrões ISO/IEC/IEEE 21451.

Norma	Estado atual	Ano
21451.0	Aprovado	2007
21451.1	Aprovado*	1999
21451.2	Aprovado*	1997
21451.3	Aprovado*	2003
21451.4	Aprovado	2004
21451.5	Aprovado	2007
21451.6	Em desenvolvimento	
21451.7	Em desenvolvimento	

A ISO/IEC/ IEEE 21451-0: consiste em fornecer uma base comum para todos membros da família 1451 que utilizam interfaces digitais promovendo a compatibilidade entre eles. Também divide o transdutor inteligente em dois módulos, Transdutor Interface Module (TIM) e *Network Capable Application Processor* (NCAP) e especifica os formatos para *Transdutor Electronic Data Sheet* (TEDS). [6]

A ISO/IEC/ IEEE 21451-1: a norma define o modelo de objeto para NCAP e permite a troca de informações entre dois ou mais NCAP através da rede de alto nível. A mesma baseia-se numa linguagem de orientação a objeto, que permite especificar toda hierarquia de classes que um software NCAP deve possuir. Atualmente em revisão e, suplantada pelo padrão ISO/IEC/IEEE 21451.0. Para mais informações consultar [7].

A ISO/IEC/ IEEE 21451-2: define uma interface digital e os protocolos necessários para ligar os transdutores a microprocessadores, define ainda uma ligação padrão entre a STIM e a NCAP, designado de Interface independente do transdutor ou *Transducer Independente Interface* (TII). O padrão inclui também a definição das especificações das TEDS. O objetivo principal do padrão é ativar o *plug end play* no nível do transdutor, fornecendo uma interface de comunicação comum para transdutores e simplificar a ligação numa rede [8].

A ISO/IEC/ IEEE 21451-3: projetada para desenvolver uma interface digital padrão que permita a ligação de vários transdutores fisicamente separada numa rede multiponto, cujas informações precisam ser lidas de forma sincronizada [9]. Esta norma é muito semelhante à ISO/IEC/ IEEE 21451-2, mas a principal diferença existente está no número de pontos de conexão na rede.

A ISO/IEC/ IEEE 21451-4: projetada para desenvolver uma interface de modo misto para transdutor analógico com modos de operação analógico e digital. O modo misto possibilita que um modo analógico, por exemplo, o sinal de um transdutor, e um modo digital, por exemplo, um dado contido em memória, possam ser disponibilizados através da mesma interface com a NCAP. Este fato é importante para aplicações que requerem elevadas taxas de transferência de dados [10],[11].

A ISO/IEC/ IEEE 21451-5: projetada para fornecer um modelo de comunicação sem fio (802.11 (Wi-Fi), 802.15.4 (ZigBee),802.15.1 (Bluetooth) e 6LoWPAN) para facilitar o acesso da TIM à NCAP[6],[12].

A ISO/IEC/ IEEE 21451-6: define uma interface entre transdutor inteligente e NCAP usando a rede *Controller Area Network* (CAN) de alta velocidade. Neste caso propõe-se o uso de uma rede de alta velocidade baseada no sistema CAN-open com diversos módulos contendo transdutores e a definição de uma camada de segurança no modelo de comunicação [10].

A ISO/IEC/ IEEE 21451-7: define uma interface e protocolo de comunicação entre os transdutores e sistemas de identificação por radiofrequência ou *Radio-Frequency Identification* (RFID) [13]. Para além disso define os novos formatos da Folha de Dados Eletrónicos Transdutores (TEDS) com base na série de padrões ISO/IEC/IEEE 21451.

2.2. Módulo constituinte da STIM e interligação com a NCAP de acordo com o padrão ISO/IEC/ IEEE 21451-2

O módulo da STIM é a entidade ou plataforma necessária para interagir, gerir e controlar os transdutores inteligente. Os diferentes Módulos STIM apresentados na Figura 1 estão em conformidade com o padrão ISO/IEC/IEEE 21451-2 da família ISO/IEC/IEEE 21451, sendo a sua implementação flexível e podendo apresentar formatos diferentes, como, por exemplo, utilizando microcontroladores de baixo custo*: Application Specific Integreted Circuit* (ASIC) e *Field-Programmable Gate Array* (FPGA), ou mesmo computador pessoal [14]. O item a), apresenta o módulo STIM formado apenas por um sensor, um conversor analógico/digital e uma folha de dados eletrónica. O item b), apresenta um módulo STIM com oito canais, sendo quatro canais para entrada e quatro para saída. O item c), apresenta uma STIM constituído por três sensores com finalidades diferentes, e finalmente, O item d), apresentando um módulo STIM constituído por dois sensores e dois atuadores.

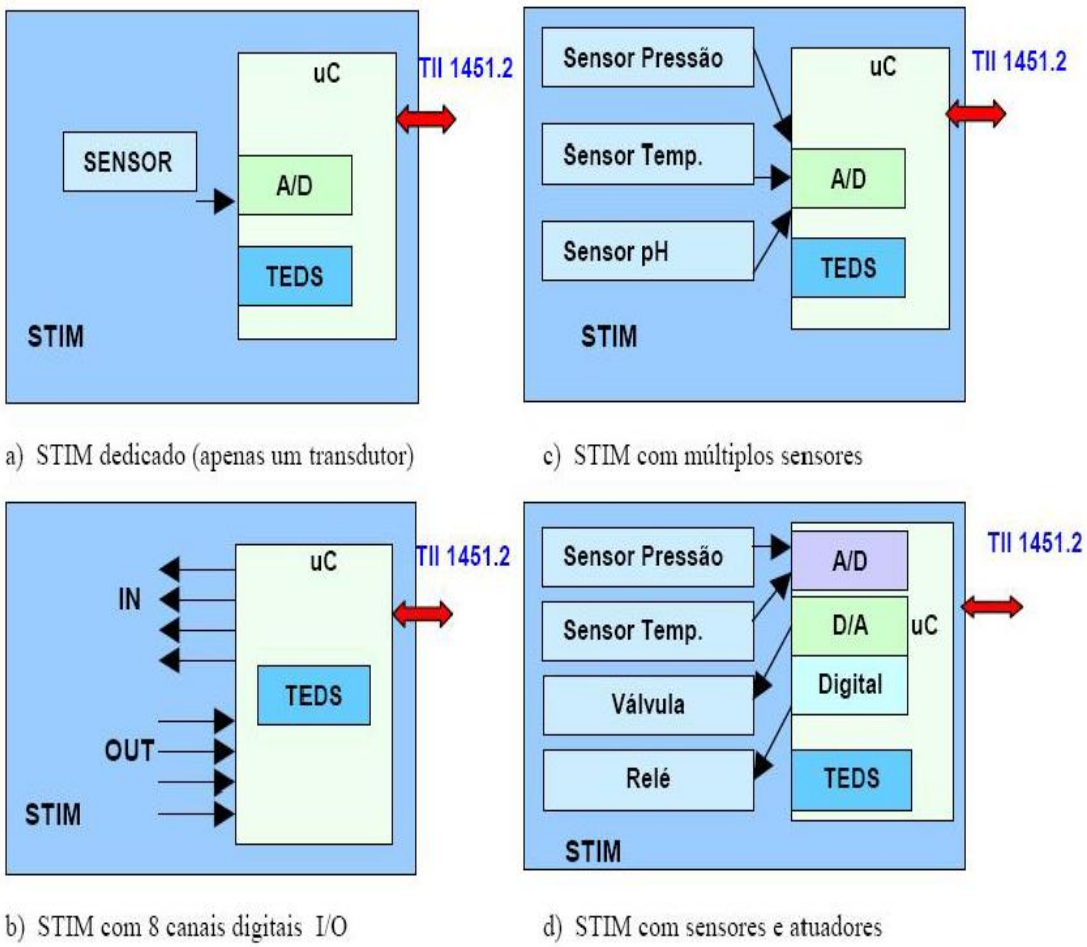


Figura 1 - Diferentes formas de implementar módulo da TIM [1].

2.2.1. Módulo constituinte da STIM

Este módulo encontra-se constituído por transdutores, conversores de sinal (AD /DA), circuitos de condicionamento de sinal e as TEDS. O conversor tem a função de converte um sinal analógico num sinal digital, ou em vice-versa, para que fiquem disponíveis para processamento num domínio digital ou para controlo de variáveis. As TEDS são arquivos armazenados no TIM que permitem a identificação automática de transdutores inteligentes e contêm informações relacionadas ao fabricante, como nome, número de série, tipo de sensor, dados de calibração, entre outras[15]. A norma ISO/IEC/IEEE 21451-2 define oito formatos para as TEDS, estabelecendo o formato lógico e o conteúdo das mesmas. No entanto, apenas duas das oitos são de implementação obrigatória, as restantes têm um caracter opcional em conformidade com a Tabela 3.

Tabela 3 - Diferentes formatos de TEDS.

TEDS	Descrição
Meta-TEDS	Descreve de forma global o módulo STIM. A sua implementação é de caracter obrigatório e contêm campos de dados que são comuns a todos os transdutores, tais como: números de canais implementados, especificações temporais, e número identificador de cada transdutor, entre outras.
TEDS de canal	Também possui caracter obrigatório. Disponibiliza informação relativa ao canal do transdutor, tais como: limites máximos e mínimos de escala, unidades físicas, incertezas, restrições de tempo, entre outras.
TEDS de Calibração	É opcional. Contêm informações de parâmetros de calibração, tais como: os coeficientes de correção, a data e hora da última calibração e, os intervalos requeridos para a calibração de cada canal.
TEDS de Aplicação especifica-	Utilizado em aplicações específicas do utilizador final. Disponibiliza mecanismos de armazenamento de dados do tipo *String*.
TEDS para identificação de canal	Possuem informação semelhante à TEDS de meta identificação, exceto que este é para canal individual.
TEDS para identificação de calibração	Fornecem uma descrição de qualquer informação relevante à calibração de canal.
TEDS de Extensões especificas	Reservados para implementações futuras em conformidade com o padrão e os setores industriais.
TEDS de meta-identificação	Fornece identificação de dados para a TIM tais como: nome do fabricante, número do modelo, número de série, versão e data dos códigos dos transdutores.

2.2.2. Interligação com a NCAP

O processo de interligação com o módulo NCAP permite que os transdutores possam ser interligados e geridos em rede. Para isso é necessário dividi-lo em três interfaces ou módulos, de acordo com a Figura 2.

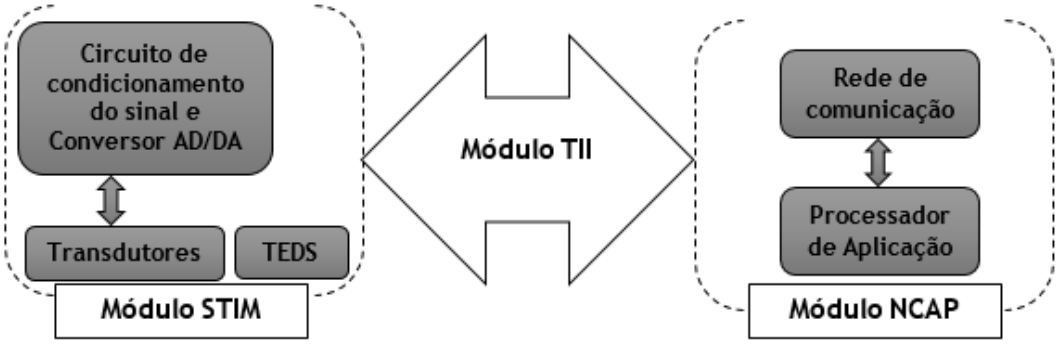


Figura 2 - Interligação entre STIM e NCAP através da TII.

Interface para módulo STIM

A interface para módulo STIM, consiste em encapsular as funções de comunicação e de troca de dados, como por exemplo: endereçamento; disparo; transporte de dados; controlo; estado e interrupção. Esta interface permite ao módulo da STIM comunicar com o módulo da NCAP através da Interface do transdutor Independente.

A função de endereçamento consiste juntamente com a interface de transporte de dados especificar se os dados estão a ser lidos ou escritos. O endereço completo, Figura 3, é constituído por dois Bytes, onde o primeiro Byte, ou Byte mais significativo, representa o endereço funcional, enquanto que o segundo Byte, ou Byte menos significativo, representa o endereço do canal. O Byte mais significativo, especifica qual a função (escrita/leitura) a ser realizado pelo módulo da STIM, enquanto que o Byte menos significativo especifica o endereço do canal que ira realizar a função. O Byte mais significativo, especifica qual a função (escrita/leitura) a ser realizado pelo módulo da STIM, enquanto que, o Byte menos significativo, especifica o endereço do canal que ira realizar a função. No entanto, a cada transdutor associado a STIM representa um canal e possui um endereço único. A função do endereço funcional depende do endereço de canal, mas é transmitido primeiro para tornar a implementação mais fácil. A função do endereço funcional depende do endereço de canal, mas é transmitido primeiro para tornar a implementação mais fácil. A NCAP utiliza o

Figura 3 - Estrutura de endereço completo. Adaptado: [6].

endereço funcional para indicar qual ação que a STIM deve realizar, isto é, indicar a direção da comunicação, se é de escrita ou de leitura de acordo com a Tabela 4. Importa aqui salientar que, os primeiros 128 endereços funcionais são reservados para escrita e os demais são utilizados para leitura.

Tabela 4 - Direção da comunicação.

Valor	Direção da comunicação
0	Representa operação de escrita na STIM
1	Representa operação de leitura da STIM

O endereço do canal é utilizado pela NCAP para especificar o canal do transdutor para o qual será enviada a solicitação. O padrão ISO/IEC/IEEE 21451-2, especifica o comportamento de 6 tipos de canais. Um sétimo tipo de canal, é identificado para permitir extensões ao comportamento da STIM, além das aqui especificadas. Os sete tipos de canais são:

1. **Canal sensor**: mede parâmetros físicos sob demanda e retorna dados digitais que representam esse parâmetro. Um novo conjunto de dados deve ser amostrado apenas na sequência de um evento de disparo. O conjunto de dados disponível para leitura deve ser o conjunto de dados resultante do evento de disparo mais recente;
2. **Canal atuador**: um atuador faz com que ocorra uma ação física ou virtual que estará relacionada ao conjunto de dados com o atuador. O estado do atuador muda para combinar o conjunto de dados mais recentemente enviado quando um evento acionador atende;
3. **Canal sensor de *buffer***: um sensor de *buffer* difere de um sensor simples na medida em que possui um único nível de *buffer* de dados no canal de saída. Um novo conjunto de dados deve ser amostrado uma vez para cada evento disparo. o conjunto de dados disponível para leitura deve ser o conjunto de dados conquistado no segundo evento de disparo mais recente;
4. **Canal sensor de sequência de dados**: um sensor de sequência de dados adquire dados continuamente, com tempos de amostragem de registro sob o controlo da STIM. A função de disparo seleciona um conjunto de dados a partir deste fluxo de medidas contínuas e disponibiliza para ser lido pela NCAP. O conjunto de dados selecionado deve ser o adquirido imediatamente após o disparo.
5. **Canal Sensor de sequência de dados com *buffer***: um sensor de sequência de dados em *buffer* adquire dados continuamente, com tempos de amostragem sob controlo da STIM. O disparo seleciona um conjunto de dados desse fluxo de medições contínuas e torna disponível para a NCAP fazer uma leitura. o conjunto de dados selecionado deve ser habilitado ou desativado por meio de um comando de controlo. O tempo de aquisição do conjunto de dados não precisa ser periódico. O número de pontos de dados no conjunto de dados deve ser determinado pelo campo de repetição de dados do canal.

6. **Canal sensor de eventos**: um sensor de sequência de evento produz um sinal sempre que ocorre um evento específico. O sinal será o mesmo usado por sensores e atuadores para reconhecer eventos disparados. O evento pode ser uma transição de sinal digital ou um cruzamento de sinal analógico em torno de um ponto de ajuste. Um sensor de sequência de eventos pode ser configurado para sinalizar um evento na transição de flanco descendente ou na transição de flanco ascendente, ou ambos. A leitura ou escrita para o canal do sensor de sequência de evento deve ocorrer apenas para informação de configuração de estado. A hora do evento é transmitida pelo sinal de confirmação do disparo. Outros canais de atuadores e sensores podem estar associados a um sensor de sequência de eventos para permitir a alteração de pontos de ajuste ou histerese[1], ou a leitura do valor analógico detetado. Esta associação é comunicada à NCAP através da informação de agrupamento na Meta-TEDS.
7. **Canal transdutor geral**: esta categoria de canal permite a presença de um canal que se comporta de forma diferente dos tipos acima. Os tipos devem implementar as mesmas funções exigidas dos canais, mas este padrão não especifica o comportamento com respeito ao disparo e escrita ou leitura de dados de tais canais.

A função de disparo fornece meios para que uma NCAP envie para uma STIM um comando para que uma ação ocorra (o sinal de disparo) e para a STIM sinalizar o momento em que a ação ocorreu (confirmação do disparo). Esta função interage com a função de transporte de dados de modo que o sinal de disparo esteja inativo enquanto ocorrer o transporte de dados. Assim, a operação de disparo é controlada pela NCAP e deve ser aplicada apenas a um único canal, ou a todos os canais ao mesmo tempo (disparo global). O canal ao qual o disparo se aplica é selecionado pelo endereço do canal acionado. A porta *NTRIG* da TII, especifica se o sinal de disparo esta ativo ou inativo. O comportamento do sistema acionador do ponto de vista da STIM é ilustrado pelo diagrama de estados na figura 4. Nesta figura é ilustrado o modo operante da função de disparo, onde, logo após o processo de inicialização esteja terminado, a STIM fica em estado de hibernação até que ocorra o envio de um sinal de disparo para ele proveniente da NCAP. Assim, quando este sinal ocorre, a STIM segue para estado de ação de disparo. No entanto, caso o transdutor seja um sensor, ele realizará a aquisição e conversão de dados. Caso seja um atuador realizará a validação e o envio para o *buffer* do atuador do último dado escrito naquele canal. Depois de ocorrer o disparo ativo, a STIM pode seguir ou para abortar (disparo negado) ou para reconhecimento de disparo (estado de resposta do sinal de disparo), onde permanece até que a NCAP retire o sinal de disparo, isto é, remoção do sinal de reconhecimento e voltar para estado de espera, e neste momento ocorrer o transporte de dados. É importante referir que, a STIM só vai para o estado de abortar quando a NCAP remover o sinal de disparo, antes que o mesmo seja reconhecido pela STIM. A NCAP só vai para estado de abortar após ter esperado um intervalo de tempo ou *Channel Update Time* que se encontra definido no Canal- TEDS. Assim, após este tempo a NCAP assume que o canal transdutor está com problemas. O transporte de dados só pode ocorrer quando o ciclo de disparo for completado.

[1] é a tendência de um sistema ou material de conservar suas propriedades na ausência de um estímulo que as gerou.

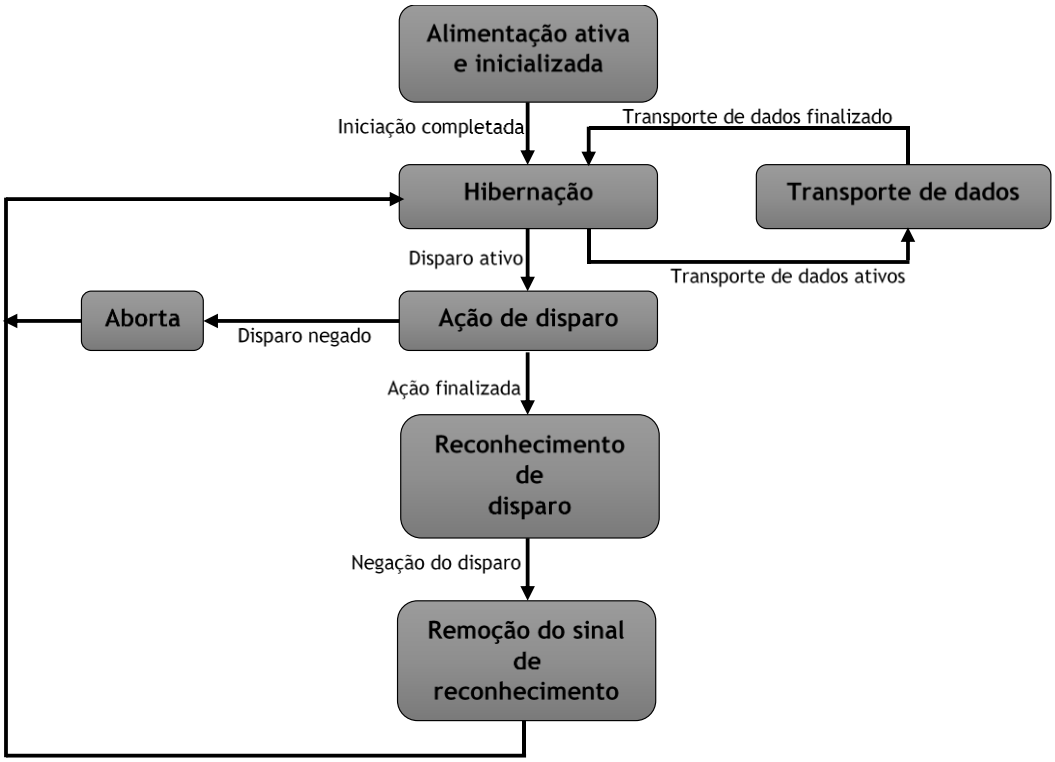


Figura 4 - Ciclo de disparo.

A função de transporte de dados é utilizada durante as operações de leitura e escrita de dados da STIM. Estas operações podem ser realizadas tanto sobre os transdutores como também sobre as TEDS. O transporte de dados pode ainda ser utilizado para escrita de um comando de controlo ou leitura do status da STIM. O transporte de dados pode ocorrer através das portas NIOE, DIN e DOUT da TII. A NIOE é a porta utilizada para identificação do transporte de dados, e deve sempre estar activa quando estiver a ocorrer o transporte de dados. A DIN é exclusivamente uma porta para transporte de dados de operações de escrita, em quanto que, a DOUT para operações exclusivas de leitura.

A função de controlo permite que os comandos sejam enviados da NCAP para a STIM a um canal específico, ou a todos os canais, que afeta o seu estado de operação. A NCAP utiliza o endereço de função *Write global Control Command* para enviar comandos a todos os transdutores da STIM, enquanto que, para um canal específico, utiliza o endereço de função *Write Channel Control Command*. O CANAL_ZERO ou CHANNEL_ZERO é exclusivo para escrever comandos de controlo global. Os comandos de controlo devem ser apenas de 2 Bytes. Os comandos de controlo CHANNEL_ZERO devem ser definidos de forma que afetem todos os canais. Os comandos de controlo definidos por este padrão estão listados na Tabela 5.

Tabela 5 - Padrões de comandos definidos para a STIM.

Valor	Definição para CANAL_ZERO	Definição para CANAIS_INDIVIDUAIS
0	Nenhuma operação	Nenhuma operação

1	Reiniciar a STIM	Reiniciar o canal
2	Autotestes na STIM	Autoteste no canal
3	Calibrar todos canais	Calibrar o canal
4	Zera todos canais	Zera o canal
5	Habilitar todos os sensores com sequencia de eventos	Habilitar sensor com sequencia de eventos
6	Desabilitar todos os sensores com sequencia de eventos	Desabilitar o sensor com sequencia de eventos
7	Modo de configuração para todos os sensores com consequência de eventos	Modo de configuração para um sensor com consequência de eventos
8	Reservado	Reservado
9	Habilitar todos os sensores com sequência de dados ou sequência de dados armazenados no *buffer*	Habilitar o sensor com sequência de dados ou sequência de dados armazenados no *buffer*
10	Desabilitar todos os sensores com sequência de dados ou sequencia de dados armazenados no *buffer*	desabilitar o sensor com sequencia de dados ou sequência de dados armazenados no *buffer*
11-255	Reservado	Reservado
256-65.535	Aberto para indústria	Aberto para indústria

Os comandos de controlo 0 e 1 devem ser implementados por todos as STIM, para todos os canais. Os comandos de controlo 2 a 4 referem-se a funções opcionais que podem ser implementadas na STIM. Os comandos de controlo 5 a 7 devem ser implementados por todos as STIM que contenham sensores de sequência de eventos. Os comandos de controlo 9 e 10 devem ser implementados por todos as STIM que contenham sensores de sequência de dados ou sensores de sequência de dados em *buffer*. Todos os comandos reservados não devem ser implementados na STIM. Todos os comandos abertos para indústria podem ser implementados na STIM. Pretende-se que os comandos de controlo designados como abertos para a indústria devem ser definidos de forma cooperativa por setores da indústria [8].

A função de estado permite que a NCAP determine o estado da STIM, como um todo, ou tratando-se de canais individuais. É implementado por meio de registros de estado padrão e auxiliar. Cada bit num registro de estado específico representa a presença ou ausência de uma condição específica. A presença de uma condição é representada através do bit "1" na posição apropriada, enquanto que, a ausência é através do bit "0". Esta função também é utilizada em conjunto com a máscara de interrupções, sendo que, as interrupções para indicar que a STIM esta requisitar serviços e para que propósito.

A função de interrupção é utilizada por dispositivos de entrada e saída para que durante a execução do programa em curso a unidade microprocessadora passe a executar um programa especial, chamada de rotina de serviço de interrupção. A porta NINT da TII é utilizada pela STIM para realizar esta função sobre a NCAP. Quando esta porta está ativa, a NCAP executa um programa especial, que, normalmente envolve atender a STIM. Ao terminar a execução da rotina de serviço de interrupção, a NCAP volta a executar o programa em que estava a trabalhar antes da ocorrência da interrupção.

Interface para o módulo da NCAP: Este módulo é responsável por realizar o processamento de dados recolhidos pela STIM em tempo real, ou seja, enviar comandos a serem realizadas pela STIM usando a TII.

Interface para o módulo TII: Este módulo interliga o módulo STIM ao módulo NCAP através da comunicação série síncrona do tipo *half-Duplex* (ISO/IEC/IEEE 21451.2) ou através de uma ligação sem fios (ISO/IEC/IEEE 21451.5). De acordo com os dois membros do padrão, uma NACP de um fabricante diferente pode comunicar com qualquer STIM, de um fabricante diferente, desde que ambos os módulos sigam os padrões. As diretrizes de protocolos, sincronização, elétricas e físicas da TII digital são definidas pelo padrão ISO/IEC/IEEE 21451.2, de acordo com a Figura 5. Nesta figura é apresentada a interface de transdutor independente constituída de dez linhas, sendo que, oitos das linhas permitem suportar as funcionalidades do padrão e as duas restantes fornecem a alimentação. As linhas que compõem a TII podem

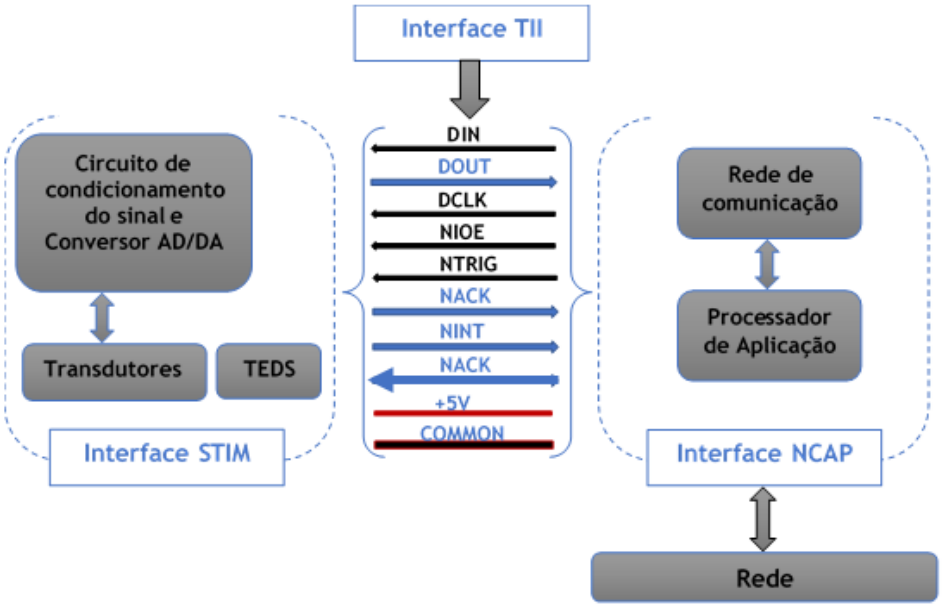


Figura 5 - Linhas de TII entre módulo STIM e NCAP.

ser agrupadas em quatro grupos de acordo com a Tabela 6.

Grupo	Linha	Abreviação	Operador	Função
Dados	DATA_IN	DIN	NCAP	Transporte de dados e endereços de NCAP para STIM.
	DATA_OUT	DOUT	TIM	Transporte de dados da STIM para NCAP.
	DATA_CLOCK	DCLK	NCAP	A cada evento de Clock permite guardar dados associados a DIN e DOUT.
	N_IO_ENABLE	NIEO	NCAP	Permite verificar se transporte de dados está ativo e delimita a estrutura de dados.
Disparo	N_TRIGGER	NTRIG	NCAP	Responsável de executar as funções de disparo.
Suporte	POWER	POWER	NCAP	Responsável pela fonte de potência nominal de 5 V.
	COMMON	COMMON	NCAP	Sinal comum ou terra.
	N_ACKNOWLEDGE	NACK	STIM	Usada para reconhecimento de disparo e reconhecimento de dados.
	N_STIM_DETECT	NSDET	STIM	Usada pela NCAP para detetar a presença de um STIM.
Interrupção	N_IO_INTERRUPT	NINT	STIM	Usada pela STIM para solicitar um serviço à NCAP.

Tabela 6 - Definição de grupos de pinos da TII e suas funcionalidades.

2.3. Plataformas de desenvolvimento

2.3.1. Code Composer Studio (CCS)

O *Code Composer Studio* (CCS) é um ambiente de desenvolvimento integrado (IDE) que permite desenvolver e executar aplicativos para microcontroladores da *Texas Instruments* (TI) utilizando a linguagem C/C++ e *assembler*. Esta plataforma é constituída por um compilador C/C++ otimizado, editor de código-fonte, ambiente de compilação de projeto, depurador e muitos outros recursos. O CSS é intuitivo e combina as vantagens da estrutura do software eclipse com recursos de depuração integrados avançados da TI, resultando num ambiente de desenvolvimento convincente e rico em recursos para programadores.

2.3.2. LaunchPad MSP430F5529

A Placa de Experiência do MSP-EXP430F5529 é uma plataforma de desenvolvimento baseada no MSP430F5529 com USB integrado. A placa de desenvolvimento experimental apresenta os recursos deste processador da família MSP430 e ajuda ao desenvolvimento rápido de aplicações que envolvam USB. Esta placa é constituída por um LCD de matriz de pontos 102 × 64, uma interface de cartão de memória microSD, um acelerómetro de 3 eixos, cinco almofadas de toque capacitivo, cabeçalhos de expansão RF EVM, nove LED, roda de polegar analógica, duas portas USB e outras de acordo com a ilustração da figura 6.

A placa MSP-EXP430F5529 possui um interruptor (*Power switch*) que define a fonte de energia a ser utilizada. No entanto, quando o interruptor é posicionado em *LDO* permite fornecer energia à placa através da porta 5529USB se está ligada ao computador. Quando é posicionado em *eZ*, permite fornecer energia à placa utilizando a porta *eZ-FET* USB e, quando é posicionado em JTAG permite fornecer energia através de uma bateria. A interface *eZ-FET* USB para além de alimentar a placa também permite a comunicação UART que é utilizada para transferir dados para o computador. O acelerómetro desta placa suporta a comunicação SPI e saída de dados para cada um dos três eixos.

2.3.3 Ligação a um elemento sensor

A placa MSP-EXP430F5029 conforme já foi referido, possui um sensor de aceleração (acelerómetro CMA3000) que será aproveitado para demostrar como fazer esta ligação. Este acelerómetro possui 3 eixos, apresenta dimensões reduzidas e baixo consumo de energia. Ele

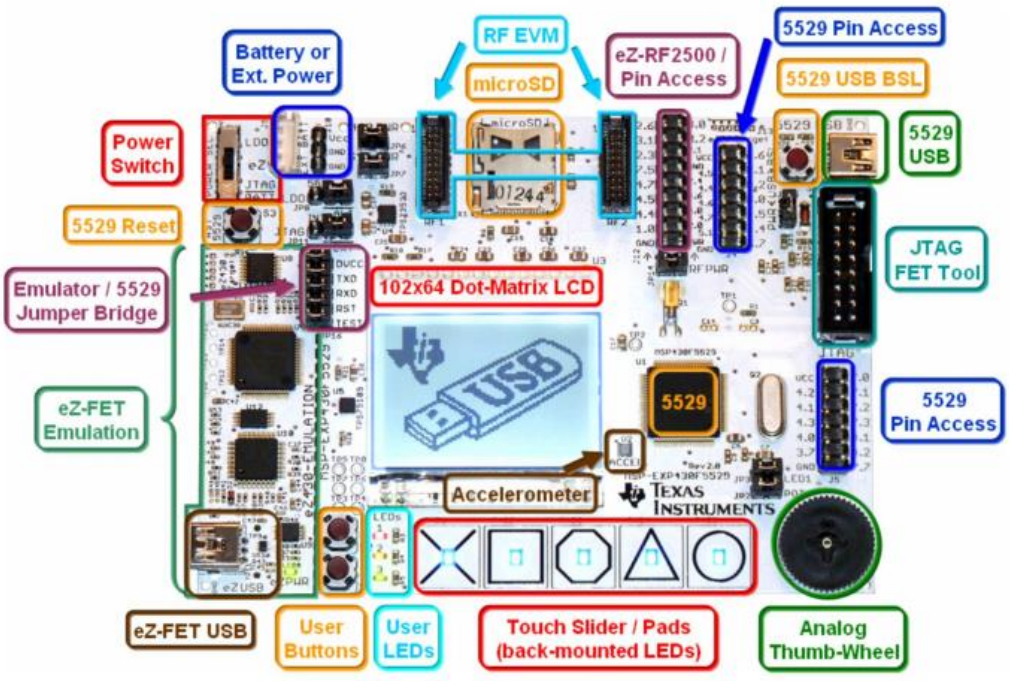


Figura 6 - Estrutura da placa MSP430F5529 [20].

usa um elemento de deteção 3D-MEMS que consiste em três massas de aceleração, um oscilador interno, referências de corrente, tensão e memória. A interface entre o MSP430F5529 e o acelerómetro CMA3000, é suportada pelo *Hardware Abstract Layer* (HAL), permitindo, assim, que as aplicações sejam desenvolvidas com esta bibliotecas de funções.

O seu princípio de funcionamento baseia-se no facto de a aceleração provocar uma variação da capacidade. Um conversor (capacidade/tensão) é usado para fornecer um sinal analógico que, após a calibração, é convertido no domínio digital e depois filtrado digitalmente. Parâmetros como ganho, *offset*, corrente de referência e frequência do oscilador vêm ajustados de fábrica. Os valores de calibração são lidos automaticamente, a partir da memória não volátil, no decorrer do processo de inicialização. A saída pode levar a configuração SPI ou I2C selecionada pelo utilizador. [16]

O acelerómetro suporta quatro modos de operações: *Power Down, measure-ment mode*; *Fall-free mode* e *Motion detection mode*. Este último modo, permite ao utilizador especificar o nível de aceleração que gerará uma interrupção do MCU.

2.3.4. Exemplo demonstrativo de operacionalidade

O exemplo demostrativo de operacionalidade entre o MCU e o acelerómetro tem como principal objetivo exibir os valores de aceleração adquiridos segundo as direções de cada um dos três eixos do acelerómetro. O código seguinte permite desenhar simultaneamente no LCD as variações de aceleração às quais a placa é submetida em cada uma das direções. Conforme já foi referido utilizaram-se as funcionalidades da HAL para facilitar a interação entre acelerómetro e o MSP430F5529. O código fonte do exemplo demostrativo entre o MCU e o acelerómetro encontram-se dividido nas 4 funções seguintes:

Função Draw_LCD_CMA3000(): é responsável por gerenciar dados no LCD, isto é, permite configurar o LCD para mostrar os valores de aceleração adquiridos nos três eixos. De acordo com a Figura 7, a função *Dogs102x6_clearScreen()* permite limpar a tela do LCD. A função *Dogs102x6_charDrawXY(0,2,'X',0)* permite escrever o caractere X nas coordenadas (x, y) do LCD, enquanto que a função *Dogs102x6_horizontalLineDraw()* permite desenhar uma linha horizontal ou ati[illegible]ões consultar a referência [16];

```
void Draw_LCD_CMA3000(void)
{
Dogs102x6_clearScreen();
Dogs102x6_charDrawXY(0,2,'X',0);
Dogs102x6_horizontalLineDraw(0,5,10,0);
Dogs102x6_horizontalLineDraw(0,102,20,0);
Dogs102x6_charDrawXY(0,22,'Y',0);
Dogs102x6_horizontalLineDraw(0,5,30,0);
Dogs102x6_horizontalLineDraw(0,102,40,0);
Dogs102x6_charDrawXY(0,42,'Z',0);
Dogs102x6_horizontalLineDraw(0,5,50,0);
Dogs102x6_horizontalLineDraw(0,102,60,0);
}
```

Figura 7 - Função gerenciador do LCD.

Função Draw_LCD_CMA3000_vector(): permite a gestão dos dados adquiridos e a sua exibição no LCD. Em cada nova sequência de aquisição, os valores dos dados são armazenados em *buffers* circulares **vector_x []**, **vector_y [] e vector_z []**, com um tamanho de 90 amostras cada e controladas pelos ponteiros **ptr_vector_x**, **ptr_vector_y** e **ptr_vector_z**. Os valores de aceleração são armazenados nos *buffers* circulares depois de serem convertidos em coordenadas do LCD. A função começa por limpar os três gráficos anteriores e desenha a curva em estilo de pixel invertido. Em seguida, segue-se a gestão dos dados nos *buffers* circulares onde a amostra mais recente entra no *buffer*, o que faz com que a mais antiga seja descartada. A função termina com o desenho dos três gráficos. Este modo de operação

```
void Draw_LCD_CMA3000_vector(void)
{
      unsigned char i;
      // Clean x-axis data on LCD
    for(i = 0; i < 90; i++)
    {
      Dogs102x6_pixelDraw(i+12,*ptr_vector_x,1);
      ptr_vector_x++;
        if (ptr_vector_x > &vector_x[89])
              ptr_vector_x = &vector_x[0];
    }
    // Clean y-axis data on LCD
    for(i = 0; i < 90; i++)
    {
      Dogs102x6_pixelDraw(i+12,*ptr_vector_y,1);
      ptr_vector_y++;
        if (ptr_vector_y > &vector_y[89])
              ptr_vector_y = &vector_y[0];
    }
    // Clean z-axis data on LCD
    for(i = 0; i < 90; i++)
    {
      Dogs102x6_pixelDraw(i+12,*ptr_vector_z,1);
      ptr_vector_z++;
        if (ptr_vector_z > &vector_z[89])
              ptr_vector_z = &vector_z[0];
    }

// Refresh x-axis circular buffer
   *ptr_vector_x = (Cma3000_xAccel >> 4) + 10;
   ptr_vector_x++;
   if (ptr_vector_x > &vector_x[89])
              ptr_vector_x = &vector_x[0];

   // Refresh y-axis circular buffer
    *ptr_vector_y = (Cma3000_yAccel >> 4) + 30;
    ptr_vector_y++;
    if (ptr_vector_y > &vector_y[89])
              ptr_vector_y = &vector_y[0];
```

Figura 8 - Exibir dados coletados no LCD.

```
    // Refresh  z-axis circular buffer
    *ptr_vector_z = (Cma3000_zAccel >> 4) + 50;
    ptr_vector_z++;
    if (ptr_vector_z > &vector_z[89])
             ptr_vector_z = &vector_z[0];

    // Draw x-axis on LCD
    for(i = 0; i < 90; i++)
    {
      Dogs102x6_pixelDraw(i+12,*ptr_vector_x,0);
      ptr_vector_x++;
        if (ptr_vector_x > &vector_x[89])
               ptr_vector_x = &vector_x[0];
    }
    // Draw y-axis on LCD
    for(i = 0; i < 90; i++)
    {
      Dogs102x6_pixelDraw(i+12,*ptr_vector_y,0);
      ptr_vector_y++;
        if (ptr_vector_y > &vector_y[89])
               ptr_vector_y = &vector_y[0];
    }
    // Draw z-axis on LCD
    for(i = 0; i < 90; i++)
    {
      Dogs102x6_pixelDraw(i+12,*ptr_vector_z,0);
      ptr_vector_z++;
        if (ptr_vector_z > &vector_z[89])
               ptr_vector_z = &vector_z[0];
    }
}
```

Figura 8 - Continuação.

Função MotionApp(): é a função responsável por inicializar o acelerómetro, o LCD e os *buffers* circulares. Em seguida, entra num ciclo de aquisição de valores de aceleração até que o botão S2 seja pressionado. O período de aquisição (~ 50 msec) é controlado pelas interrupções do TimerA0. A rotina de serviço de temporização ativa a CPU, que segue as operações de amostragem dos valores de aceleração e a atualização do LCD (ver a figura 9);

Função main(void): é a função de onde será chamada a aplicação do acelerómetro *(MontionApp())*, conforme ilustrado na Figura 10. No entanto, é necessário adicionar as bibliotecas de acordo com a Figura 11 no cabeçalho do código fonte.

O projeto do acelerómetro foi repartido em duas partes, sendo que ***main*.c** é responsável pelo código fonte da figura 10 e 11. Enquanto que ***aceler*.c** possui o código das Figuras 7,8 e 9.

```
void MotionApp(void)
{
      unsigned char i;
 //inicializar os valores de deslocamento do acelerometro na memória
Cma3000_setAccel_offset(*((unsigned char*)accelXcalibrationAddress),
                        *((unsigned char *)accelYcalibrationAddress),
                       *((unsigned char *)accelZcalibrationAddress));
 /*Initialize vector data ou inicializar os vetores de dados que tem
um tamanho de 90 amostra e cada eixo é controlado pelo seu respetivo
ponteiro.*/
* ptr_vector_x para eixo-x, ptr_vector_x para eixo-y e ptr_vector_x
para eixo-z
            */
            ptr_vector_x = &vector_x[0];            // x-axis
            ptr_vector_y = &vector_y[0];            // y-axis
            ptr_vector_z = &vector_z[0];            // z-axis

           for(i = 0;i < 90; i++)
           {
             vector_x[i] = 10;
             vector_y[i] = 30;
             vector_z[i] = 50;
           }

           Cma3000_init();//inicializa o acelerómetro

           /* A função Draw_LCD_CMA3000() permite configurar o
            * LCD para mostrar os valores de aceleração coletados nos
           três eixos.*/

           Draw_LCD_CMA3000();
           /* Timer A configured to wakeup CPU ou configurando o
            temporizador A para acordar CPU*/
            TA0CCTL0 = CCIE; // Ativada Interrupção CCR0
            TA0CCR0 = 1638;//Período de interrupção ~ 50 msec
            TA0CTL = TASSEL_1 + MC_1 + TACLR;/* ACLK, upmode, clear
            TAR*/

           // main loop
            buttonsPressed = 0;
            while (!(buttonsPressed & BUTTON_S2))
            {
              __bis_SR_register(LPM0_bits + GIE);/* LPM0,
             TimerA0_ISR will force exit*/
              // read acceleration
              Cma3000_readAccel_offset();
              // Draw data
              Draw_LCD_CMA3000_vector();
            }
            // Disable the CMA3000
            Cma3000_disable();
            // Stop Timer
             TA0CTL = 0x00;
}
```

Figura 9 - Acelerómetro método.

```
void main(void)
{
    // Stop WDT
    WDTCTL = WDTPW + WDTHOLD;
    //Inicializa o contraste e o brilho do LCD a partir da memória
   uint8_t contrast = *((unsigned char*)contrastSetpointAddress);
   uint8_t brightness = *((unsignedchar*)brightnessSetpointAddress);
   //Inicialização básica de GPIO
   Board_init();
   // Set Vcore to accomodate for max. allowed system speed
   SetVCore(3);
   // Use 32.768kHz XTAL as reference
   LFXT_Start(XT1DRIVE_0);
    // Set system clock to max (25MHz)
   Init_FLL_Settle(25000, 762);
   SFRIFG1 = 0;
   SFRIE1 |= OFIE;
   // Globally enable interrupts
    __enable_interrupt();
    // Set up LCD
    Dogs102x6_init();
    Dogs102x6_backlightInit();
    // Contrast not programmed in Flash Yet
    if (contrast == 0xFF)
        // Set Default Contrast
        contrast = 11;
    // Brightness not programmed in Flash Yet
    if (brightness == 0xFF)
        // Set Default Brightness
        brightness = 11;
    Dogs102x6_setBacklight(brightness);
    Dogs102x6_setContrast(contrast);
    Dogs102x6_clearScreen();
    MotionApp();
}
//*********************************************************************
************
// TimerA0 interrupt service
//*********************************************************************
************
#pragma vector=TIMER0_A0_VECTOR
__interrupt void TimerA0_ISR(void)
{
__bic_SR_register_on_exit(LPM0_bits);      // Exit active CPU

}
```

Figura 10 - Função main.

```
#include "msp430.h"
#include "HAL_PMM.h"
#include "HAL_UCS.h"
#include "HAL_Board.h"
#include "HAL_Buttons.h"
#include "HAL_Cma3000.h"
#include "HAL_Dogs102x6.h"
#include "HAL_Menu.h"
#include "HAL_Wheel.h"
#include "LaunchpadDef.h"
#include "Settings.h"
#include "lab4.2.h"
```

Figura 11 - Bibliotecas em C.

Capítulo 3 - Especificação Funcional - ISO/IEC/IEEE 21451-0.

3.1. Especificação funcional do transdutor Inteligente

As especificações funcionais são descrições a nível físico entre a TIM e a NCAP do ponto de vista de ligação e elucidam dos tipos de canais de transdutores que podem ser implementados na TIM e suas funcionalidades. Estas especificações são elucidadas em função do membro 21451-0 da família, que fornece a explicação das funcionalidades esperadas a serem implementadas. A Figura 12 ilustra o modelo referencial definido pelo padrão, tanto para a NCAP quanto para a TIM. Este padrão estabelece os três seguintes tipos de canais de transdutor: Sensor, Atuador e Sensor de eventos.

3.2. Estados operacionais

O Padrão define diferentes estados operacionais para a TIM e para cada um dos transdutores nela integrados. Na Figura 13 é mostrado o diagrama de estado para um transdutor. O transdutor transita para o estado Transdutor ocioso quando sai da inicialização, e pode entrar no estado Operacional depois de receber um determinado comando, no qual, dependendo da sua configuração, ele começará a capturar dados diretamente ou aguardar uma ordem de disparo (*trigger*). Uma TIM possui três estados, e permanece no estado de inicialização através do comando *Reset* ou através de um evento de ativação. Quando o processo de inicialização estiver concluído, a mudança para o estado ativo da TIM é realizada. A TIM pode transitar para o estado de adormecimento através do comando *Sleep*, e do estado de adormecimento para o ativo através do comando *Wake-up* conforme é apresentado na Figura 14.

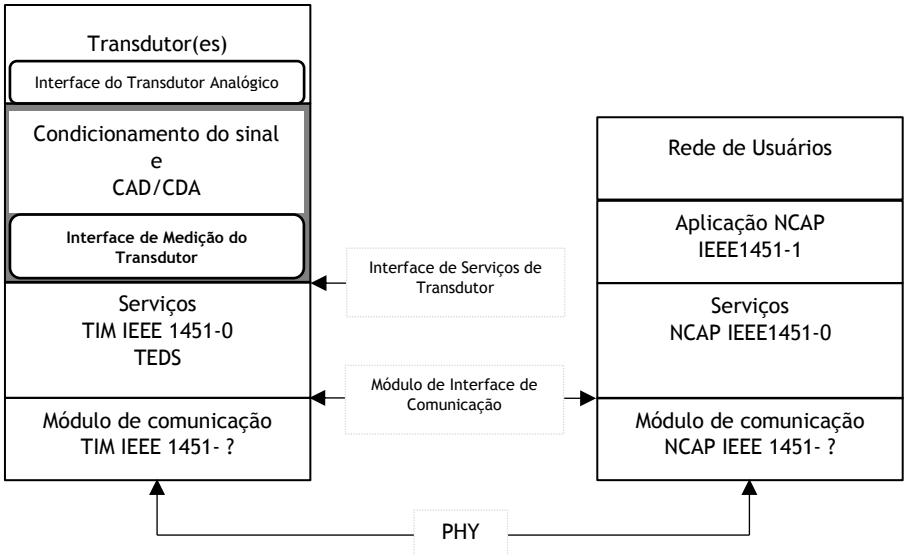


Figura 12 - Modelo referencial.

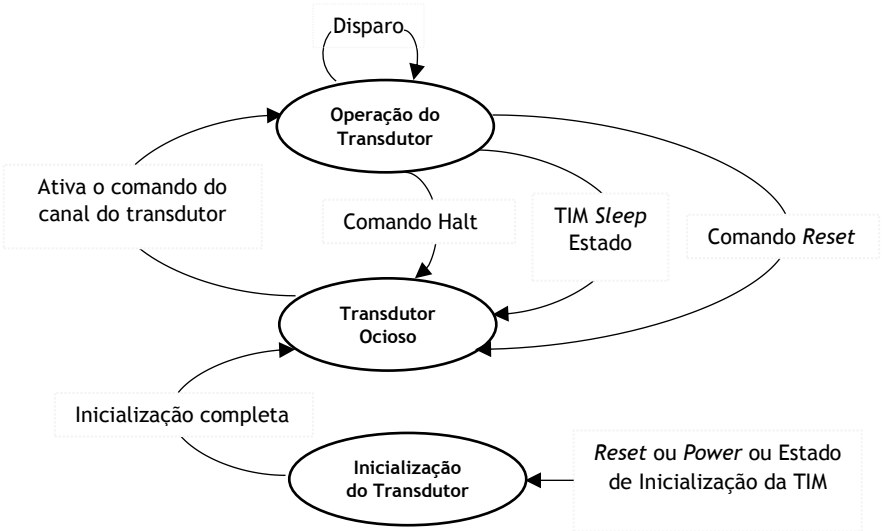


Figura 13 - Estados operacionais do canal do transdutor.

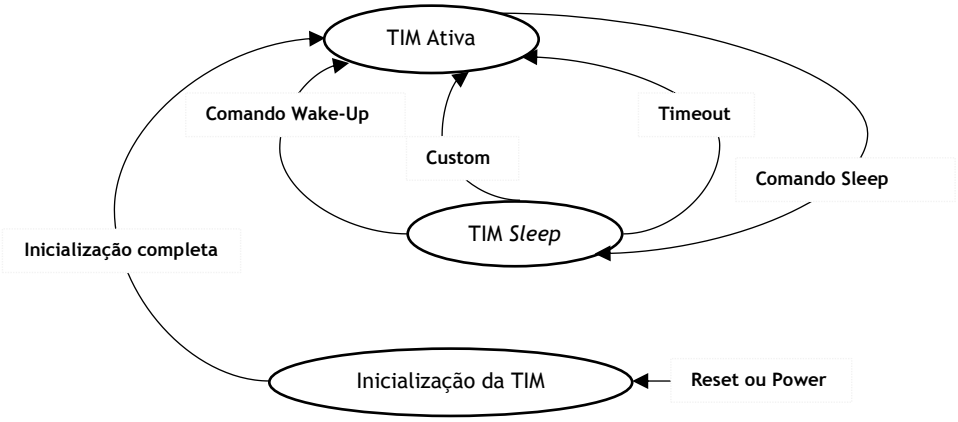


Figura 14 - Estados operacionais da TIM.

3.3. Endereços

O padrão define dois níveis de endereçamento. O primeiro nível de endereçamento está associado à implementação da camada física e permite que mensagens sejam trocadas entre uma TIM e uma NCAP. No entanto, os detalhes deste nível são abordados noutros membros da família ISO/IEC/IEEE 21451 que define os meios de comunicação. O segundo nível de endereçamento é associado ao número do canal do transdutor para um determinado canal do transdutor dentro da TIM. Trata-se de um número de 2 octetos utilizados em mensagens de comandos ou de respostas para identificar o canal do transdutor de destino ou de origem. Existem 4 classes de endereços definida pelo padrão:

- ✓ **Global**: é um grupo especial de endereço que pertence a todos os canais do transdutor na TIM. O valor em decimal reservado para este é 65535. Os comandos endereçados ao endereço global devem ser recebidos e respeitados por todos os canais dos transdutores na TIM endereçada. Se o comando recebido não for implementado por um TIM nenhum erro será gerado;

- ✓ ***AddressGroup*:** Existem dois tipos de grupos de endereços, o *Bit Mapped* e o *Binary*. O grupo *Bit Mapped* é utilizado quando alguns grupos de endereços são necessários e é desejável enviar um comando para vários grupos de endereços ao mesmo tempo. Os valores em decimal que podem ser atribuídos a este grupo estão no intervalo]32768;49151]. O valor 32768 é um endereço não funcional. O grupo *Binary* é utilizado quando um grande número de grupos de endereços diferentes é necessário. Os valores em decimal que podem ser atribuídos estão no seguinte intervalo [49152;65534]. Os comandos emitidos para um Grupo de Endereços serão executados por todos os canais dos transdutores que foram inicializados como membros desse Grupo de Endereços;
- ✓ ***TransducerChannel*:** Um endereço com o bit mais significativo definido como zero. Os 15 bits restantes identificam o canal do transdutor para o qual a mensagem é destinada. Os valores em decimal que podem ser atribuídos para este, estão no seguinte intervalo [1;32767]. Estes valores devem ser recebidos e respeitados pelo canal do transdutor para o qual são endereçados. Se o comando recebido não for implementado pelo canal do transdutor, o comando será ignorado e um erro será gerado conforme especificado para o comando individual;
- ✓ **TIM**: Um zero no campo que representa o canal na estrutura de comando de mensagem indica que a mensagem é destinada à TIM e não a um canal do transdutor individual. Este valor deve ser recebido e respeitado pela TIM ao qual é endereçado.

3.4. Estruturas usadas para armazenar e transmitir dados

Três estruturas são utilizadas para armazenar e transmitir dados neste padrão. Estas são o conjunto de dados, a mensagem e o pacote. As aplicações acima da pilha de protocolos trabalham com conjuntos de dados. As camadas superiores da pilha de protocolos lidam com mensagens. Se um conjunto de dados contiver mais octetos do que o que pode ser enviado com uma única mensagem, ele deverá ser dividido pela aplicação em várias mensagens [6].

3.4.1. Conjunto de dados

Todos os Canais do Transdutor atuam com um conjunto de dados. Porém, este conjunto de dados são definidos por três campos existentes dentro do Canal Transdutor TEDS a saber:

- ✓ Campo máximo: O campo máximo de repetições de dados define o número máximo de amostras de dados individuais num conjunto de dados;
- ✓ Campo de incremento da série: é utilizado para determinar o intervalo entre as amostras e pode ser substituído por um comando definido pelo fabricante ou por um atuador incorporado;
- ✓ Campo unidades de série: define as unidades para o campo de incremento da série. A implicação do campo de unidades da série é que as unidades do campo de incremento da série não precisa ser a hora e, se este for o caso, o intervalo de tempo entre as amostras pode não ser uniforme.

3.4.2. Mensagens e pacotes

As mensagens podem conter até 65 535 octetos de dados mais os octetos nos cabeçalhos. No entanto, o *link* de dados e as camadas físicas da pilha de protocolo transmitem pacotes. O tamanho máximo do pacote é definido no padrão que define as camadas físicas e de enlace de dados. Se uma mensagem for muito longa para caber em um único pacote, é responsabilidade da camada de enlace de dados na pilha de protocolos dividir as mensagens em vários pacotes para transmissão [6].

Capítulo 4 - Desenvolvimento da TIM baseado nos padrões 21451.0 e p1451.2-UART

4.1. Estrutura da TIM desenvolvida

A Figura 15 mostra o módulo da TIM desenvolvido e ligado ao nó da rede. A TIM foi desenvolvida e implementada no microcontrolador da *Texas Instruments* MSP430F5529 utilizando a linguagem de programação C. O módulo consiste num transdutor de temperatura, nas TEDS baseadas no padrão ISO/IEC/IEEE 21451, e num módulo de interface de comunicação baseada na IEEE p1451-2 UART para definir a *PhyTEDS*.

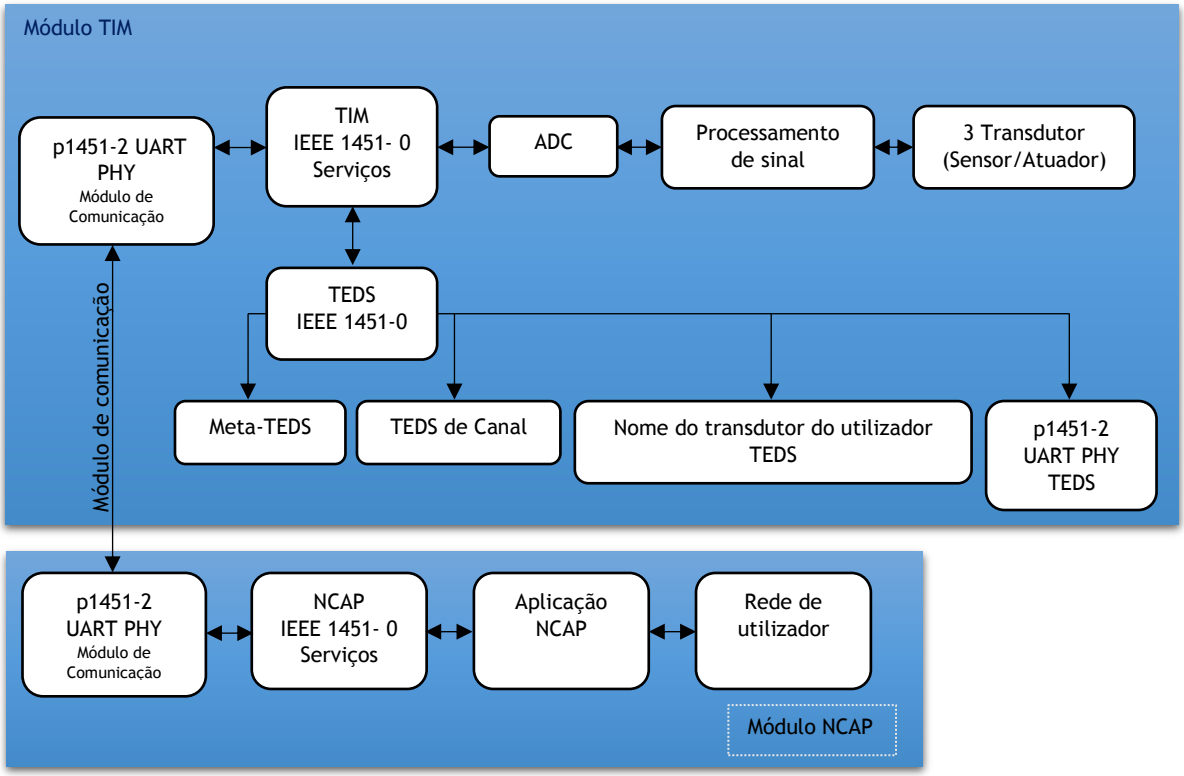


Figura 15 - Módulo da TIM desenvolvida ligado à NCAP.

4.2. Serviços desenvolvidos na TIM

Para garantir que os transdutores possuam a funcionalidade *plug-and-play*, foram implementados serviços em conformidade com o padrão tais como: estrutura de comandos, estrutura de mensagens, estrutura das TEDS, bem como o meio físico e seu protocolo de comunicação.

4.2.1. Estruturas de mensagens de comando

A estrutura das mensagens enviadas através da interface de comunicação do módulo pode ser classificada em três tipos: estruturas de mensagem do comando, estrutura de mensagem gerada pela TIM e estrutura de mensagem de resposta. A Tabela 7 define a estrutura de mensagem do comando utilizada quando um nodo da rede solicita alguma informação ou

envia alguma ordem na rede [17]. Esta estrutura consiste no número do canal do transdutor de destino, na classe do comando, na função do comando, no comprimento da mensagem e nos octetos relacionados ao comando dependentes.

Tabela 7 - Estrutura da mensagem de comando.

1- Octeto							
7	6	5	4	3	2	1	0
Número do canal do transdutor de destino (octeto mais significativo)							
Número do canal do transdutor de destino (octeto menos significativo)							
Classe de comando							
Função do comando							
Comprimento (octeto mais significativo)							
Comprimento (octeto menos significativo)							
Octetos dependentes do comando							

O número do canal do transdutor é constituído por dois octetos que contêm o destino da mensagem, ou seja, indica o sensor ou atuador alvo dentro da TIM, num grupo ou a TIM. A classe do comando permite identificar o tipo de comando a ser utilizado através do campo *cmdClassID* e é definida na Tabela 11. A função do comando permite identificar através do seu campo *FuncID* se o comando é de escrita, ou de leitura, ou outros definida na Tabela 13. No entanto, esta função deve ser compreendida no contexto da classe de comando. O comprimento é o número de octetos dependentes do comando nesta mensagem.

4.2.2. Estrutura de mensagem de resposta

A TIM pode gerar uma resposta a um comando sob duas circunstâncias. A primeira é quando o comando requer uma resposta e a segunda é quando o protocolo do *Status-event* estiver ativado[18]. No entanto, a estrutura de uma mensagem de resposta é definida na Tabela 8. Esta mensagem consiste em um sinalizador de sucesso/falha, comprimento da mensagem e os octetos dependentes do comando.

Tabela 8 - Estrutura da mensagem de resposta.

1- Octeto							
7	6	5	4	3	2	1	0
Flag de sucesso ou falha							
Comprimento (octeto mais significativo)							
Comprimento (octeto menos significativo)							
Octetos dependentes do comando							

A *flag de* sucesso/falha permite identificar se comando enviado foi bem-sucedido ou não. Se o valor da *flag* for diferente de zero, o comando foi bem-sucedido. Se possuir valor zero, o comando falhou, e o sistema deve verificar o estado para determinar o motivo. O

comprimento é definido em função de números de octetos dependentes da resposta da mensagem. Os octetos dependentes do comando contêm as informações que devem ser definidas para o comando.

4.2.3. Estrutura de mensagem iniciada pela TIM

O formato da estrutura de mensagem inicializada pela TIM é fornecido na Tabela 9. Exemplos destas mensagens são dados de fluxos e mensagens de estados [6]. O número do canal do transdutor é constituído por dois octetos que contém a origem da mensagem. O conteúdo de outros campos é o mesmo definido anteriormente.

Tabela 9 - Estrutura de mensagem iniciada pela TIM.

1- Octeto							
7	6	5	4	3	2	1	0
Número do canal do transdutor de origem (octeto mais significativo)							
Número do canal do transdutor de origem (octeto menos significativo)							
Classe de comando							
Função do comando							
Comprimento (octeto mais significativo)							
Comprimento (octeto menos significativo)							
Octetos dependentes do comando ….							

4.2.4. Ordem de transmissão de dados e significados de bit

A ordem de transmissão do cabeçalho e dos dados é resolvida a nível de octetos, sendo a ordem de transmissão de dados destes octetos da esquerda para direita. Por exemplo, na Tabela 10, os octetos são transmitidos na ordem em que são numerados. É importante salientar que, a ordem de transmissão acima descrita aplica-se à interface de comunicação do módulo e é apenas conceitual. A ordem de transmissão de dados e o significado do bit podem ser diferentes na camada física [6].

Tabela 10 - Ordem de transmissão dos octetos.

1º Octeto								2º Octeto								3º Octeto								4º Octeto							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Primeiro octeto transmitido																								último octeto transmitido							
10								9								8								5							

4.2.5. Comandos comuns à TIM e ao canal do transdutor

Os comandos estão divididos em duas categorias, os definidos por fabricantes e os padrões. A estrutura do comando é formada por dois octetos, sendo que o octeto mais significante define a classe do comando e o menos significante define uma função especifica do comando dentro

da classe. A estrutura de uma mensagem de comando esta definida na Tabela 7. No entanto, a Tabela 11 define classes de comandos padrões.

Tabela 11 - Classes de comandos padrões.

cmdClassId	Atributo	Categoria
0	Reservad	Reservado
1	CommonCmd	Comandos comuns à TIM e aos canais dos transdutores
2	XdcrIdle	Transdutor no estado ocioso
3	XdcrOperate	Transdutor no modo operante
4	XdcrEither	Transdutor no estado ocioso ou operante
5	TIMsleep	Comandos de TIM no estado de repouso
6	TIMActive	Comandos de TIM no estado ativo
7	AnyState	Qualquer estado
8-127	ReservedClass	Reservado
128-255	ClassN	Abertos para fabricantes – números da classe

A classe de comandos é utilizada para especificar se o comando deve ser executado pela TIM ou pelo canal do transdutor. Estes comandos não devem ser endereçados a um grupo de endereço ou a um proxy de canal do transdutor. Caso aconteça, o comado deve ser ignorado e o bit rejeitado do comando no registrador de condição de estado da TIM deverá ser definido. A Figura 16 apresenta o código da função desenvolvida que permite selecionar as classes de comandos que podem ser utilizadas. Através do conteúdo da variável ***cmdClasID*** é selecionado a classe a ser executada. O ***strcmp*** faz a comparação entre o conteúdo da variável ***cmdClasID*** passado no comando com os valores predefinidos pelo padrão para determinar a classe a executar. Se nenhum valor for encontrado a função retornará um valor não definido. No entanto, para este projeto apenas foram desenvolvidos alguns comandos da classe comum. A Tabela 12 lista os comandos comuns implementados. Além destes comandos implementados existem outros comandos que são comuns e que não serão abordados, tais como: *Run self-test*; *Write service request mask*; *Read service request mask*; *Read status-event register*; entre outros.

Os comandos *QueryTEDS*, *ReadTEDS segment*, *WriteTEDS segmet e* Update podem ser utilizados em qualquer estado do canal, desde que a TIM esteja inicializada para ter acesso as TEDS. No entanto, destes comandos mencionados apenas do comando *WriteTEDS segmet* não é esperada uma resposta. A Figura 17 apresenta o código da função desenvolvida que permite selecionar as funções de comandos que podem ser utilizadas definidas pelo padrão. O código da função do comando dependente é apresentado na Figura 18.

```
void cmdClassSelect(char * cmdClasID){

// function that allows you to select the command classes to be used.
      if(strcmp(cmdClasID, "001") == 0){

            UART_Printf(UART_CHANNEL, "Commands common to the
            TIM and TransducerChannel\n\r");

            FunctionSelect(FuncID,TedsCod,ChanID);
      }
```

Figura 16 - Função que permite selecionar as classes de comando a serem utilizados.

```
        else if(strcmp(cmdClasID, "002") == 0){

                UART_Printf(UART_CHANNEL, " Transducer idle state
                Is not implemented!");

        }
        else if(strcmp(cmdClasID, "003") == 0){

                UART_Printf(UART_CHANNEL, "Transducer operating
                State is not implemented!");

        }
        else if(strcmp(cmdClasID, "004") == 0){

                UART_Printf(UART_CHANNEL, "Transducer either idle or
                operating state is not implemented!");

        }
        else if(strcmp(cmdClasID, "005") == 0){

                UART_Printf(UART_CHANNEL, "Sleep state is not
                implemented!");

        }
        else if(strcmp(cmdClasID, "006") == 0){

                UART_Printf(UART_CHANNEL, "TIM active state commands
                is not implemented!");

        }
        else if(strcmp(cmdClasID, "007") == 0){

                UART_Printf(UART_CHANNEL, "Any state is not
implemented!");

        }
        else{
                UART_Printf(UART_CHANNEL, "\n\r");
                UART_Printf(UART_CHANNEL, " class value not set by
                default.. \n\r");
        }

}
```

Figura 16 - Continuação.

Tabela 12 - Comandos comuns à TIM e ao canal do transdutor.

CmdFunclD	Comando	Estado		Octeto dependente de comando		Octeto dependente de resposta	
		transdChann	TIM	Nº	Descrição	Nº	Descrição
0	Reservado	----------	----	--	----------	--	----------
1	*QueryTEDS*	qualquer	ativo	1	Código de acesso	1	Atributo da TEDS
						1	Estado da TEDS
						4	Comprimento TEDS

						2	*Checksum da* TEDS
						4	Comprimento máximo da TEDS
2	*ReadTEDS segment*	qualquer	ativo	1	Código de acesso	4	*Offset* para a *ReadTEDS*
				4	*Offset* para *ReadTEDS*	N	Octetos de composição TEDS
3	*WriteTEDS segment*	qualquer	ativo	1	Código de acesso	--	----------
				4	Offset para a *WriteTEDS*	--	----------
4	*UpdateTEDS*	qualquer	ativo	1	Código de acesso	--	Os mesmos utilizados com o comando *QueryTEDS*

***QueryTEDS*:** este comando é utilizado pela NCAP para solicitar informações necessárias para ler ou escrever a TEDS. Trata-se de um comando dependente do código de acesso da TEDS que permite identificar a TEDS a ser acedida. A Tabela 14 lista os códigos de acesso às TEDS. Por exemplo, se a NCAP quiser ler ou escrever uma das TEDS, o formato do comando a enviar à TIM pode ser definido de acordo com a Tabela 13.

Tabela 13 - Estrutura do camando QueryTEDS.

Octetos nº	2	1	1	2	1
	ChannID	cmdClassID	FuncID	Comp	TEDSaccesscode
	chch	01	02	cc	03

O *chch* refere-se a um transdutor específico ou à TIM, o *01* indica a classe dos comandos comuns, o *02* indica que o comando comum utilizado é o *ReadTEDS segment*, o *cc* corresponde ao comprimento da mensagem e o 03 corresponde as TEDS do canal do transdutor.

```
static void FunctionSelect(char * FunctionID,char *
TedscodaccessID,char * ChannelID)
{
/* function that allows to select the type of common command to be
used and the command access code of the command.*/
      int IsGlobal;

      IsGlobal = strcmp(ChannelID, "000");

      if(strcmp(FunctionID, "001") == 0){
            if(IsGlobal == 0){
                  UART_Printf(UART_CHANNEL, "Going to query Meta-
                  TEDS\r\n");
                  //selecionar as TEDS para a TIM
                  TEDSAccessCodeSelect(TedscodaccessID);
            }
```

Figura 17 - Função que seleciona o comando comum, o comando dependente e o canal.

```
        else{
              UART_Printf(UART_CHANNEL, "Quering Channel TEDS
               for Channel %s\r\n", ChannelID);
           //selecionar as TEDS para o canal
              TEDSAccessCodeSelect2(TedscodaccessID);
        }
  }
  if(strcmp(FunctionID, "002") == 0){
        if(IsGlobal == 0){
              //ReadMETA-Teds
              UART_Printf(UART_CHANNEL, "Going to read Meta-
               TEDS\r\n");

              TEDSAccessCodeSelect(TedscodaccessID);
        }
        else{
              UART_Printf(UART_CHANNEL, "Reading Channel TEDS
             for Channel %s\r\n", ChannelID);

              TEDSAccessCodeSelect2(TedscodaccessID);
        }
  }
  if(strcmp(FunctionID, "003") == 0){
        if(IsGlobal == 0){
              UART_Printf(UART_CHANNEL, "Going to Write Meta-
               TEDS\r\n");
           //selecionar as TEDS para a TIM
              TEDSAccessCodeSelect(TedscodaccessID);
        }
        else{
              UART_Printf(UART_CHANNEL, "Writing Channel TEDS
               for Channel %s\r\n", ChannelID);
           //selecionar as TEDS para o canal
              TEDSAccessCodeSelect2(TedscodaccessID);
        }
  }
  else if(strcmp(FunctionID, "004") == 0){
        if(IsGlobal == 0){
              UART_Printf(UART_CHANNEL, "Going to update Meta-
              TEDS\r\n");
              TEDSAccessCodeSelect(TedscodaccessID);
        }
        else{
              UART_Printf(UART_CHANNEL, "updating Channel TEDS
             for
              Channel %s\r\n", ChannelID);
              TEDSAccessCodeSelect2(TedscodaccessID);
        }
  }
```

Figura 17 - Continuação 1.

```
	else if(strcmp(FunctionID, "005") == 0){
		// Is not implemented
		UART_Printf(UART_CHANNEL, "Going to Run self-test\r\n");
	}
	else if(strcmp(FunctionID, "006") == 0){
		// Is not implemented
		UART_Printf(UART_CHANNEL, "Going to Write service
		request mask\r\n");
	}
	else if(strcmp(FunctionID, "007") == 0){
		// Is not implemented
		UART_Printf(UART_CHANNEL, "Going to Read service request
		mask\r\n");
	}
	else if(strcmp(FunctionID, "008") == 0){
		// Is not implemented
		UART_Printf(UART_CHANNEL, "Going to Read status-event
		register\r\n");
	}
	else if(strcmp(FunctionID, "009") == 0){
		// Is not implemented
		UART_Printf(UART_CHANNEL, "Going to Read status-
		condition register\r\n");
	}
	else if(strcmp(FunctionID, "010") == 0){
		// Is not implemented
		UART_Printf(UART_CHANNEL, "Going to Clear status-event
		 register\r\n");
	}
	else if(strcmp(FunctionID, "011") == 0){
		// Is not implemented
		UART_Printf(UART_CHANNEL, "Going to Write status-event
		 protocol state\r\n");
	}
	else if(strcmp(FunctionID, "012") == 0){
		// Is not implemented
		UART_Printf(UART_CHANNEL, "Going to Read status-event
		 protocol state\r\n");
	}
	else{
		UART_Printf(UART_CHANNEL, "value not set by default..");
	}
}
```

Figura 17 - Continuação 2.

Tabela 14 - Códigos de acesso à TEDS.

Código de acesso à TEDS	Atributo de nome TEDS	TEDS	Obrigatório	Opcional
0	Reservado			
1	Meta-TEDS	Meta-TEDS[1]	Sim	
2	MetaIdTEDS	Meta de identificação TEDS[2]		Não
3	ChanTEDS	TEDS[1] de Canal de transdutor	Sim	
4	ChanIdTEDS	TEDS[2] de Canal de transdutor		Não
5	CalTEDS	TEDS[1] de calibração		Não
6	CalIdTEDS	TEDS[2] Calibração de identificação		Não
7	EUASTEDS	TEDS[3] específico do aplicativo dos utilizadores finais	Sim	
8	FreqRespTEDS	Resposta de frequência de TEDS[1]		Não
9	TransferTEDS	Função de transferência		Não
10	CommandTEDS	Comandos TEDS[2]		Não
11	TitleTEDS	Localização e Titulo TEDS		Não
12	XderName	Nome do transdutor do utilizador TEDS[3]	Sim	
13	PHYTEDS	PHY TEDS[1]	Sim	
14	GeoLoc TEDS2	Localização geográfica TEDS[2]		Não
15	UnitsExtention	Extensão das unidades TEDS		Não
16-127	-----	Reservado		Não
128-255	-----	TEDS definido pelo fabricante		Não

Notas:

1 – Uma TEDS binária. 2 – Uma TEDS baseada em texto. 3- Conteúdo da informação definida pelo utilizador.

```
void TEDSAccessCodeSelect(char * TEDSCode){
      //estrutura para comando QueryTEDS
      if(strcmp(FuncID, "001") == 0){
            if(strcmp(TEDSCode, "001") == 0){
               UART_Printf(UART_CHANNEL, "presents the information
               necessary to read or write Meta-TEDS\r\n");
               ReadReplayQueryMetaTEDS();
            }
            else if(strcmp(TEDSCode, "003") == 0){
               UART_Printf(UART_CHANNEL, "presents the information
               necessary to read or write TransducerChannel
               TEDS1\r\n");
               ReadReplayQueryChannTEDS();

            }
            else if(strcmp(TEDSCode, "005") == 0){
                  UART_Printf(UART_CHANNEL, "presents the
                  information necessary to read or write
                  Calibration TEDS1\r\n");
                  ReadReplayQueryCalibTEDS();
            }
            else if(strcmp(TEDSCode, "007") == 0){
                 UART_Printf(UART_CHANNEL, "presents the information
                 necessary to read or write  End users'
                 application-specific TEDS3\r\n");
                 ReadReplayQueryUserApplicSpecTEDS();
            }
            else if(strcmp(TEDSCode, "012") == 0){
                  UART_Printf(UART_CHANNEL, "presents the
                 information necessary to read or write  transducer
                 name TEDS3\r\n");
                 ReadReplayQueryUserNameTransTEDS();
            }
            else if(strcmp(TEDSCode, "013") == 0){
                UART_Printf(UART_CHANNEL, "presents the information
                necessary to read or write PHY TEDS1\r\n");
                ReadReplayQueryPhyTEDS();
            }
            else{
               UART_Printf(UART_CHANNEL,"Falha no comando....\r\n");

            }
      }

       //estrutura para comando read segment TEDS
       else if(strcmp(FuncID, "002") == 0){
            UART_Printf(UART_CHANNEL,"Comando de leitura de segmento
            TEDS....\r\n");
            if(strcmp(TEDSCode, "001") == 0){
               UART_Printf(UART_CHANNEL, "GO to read Meta-
               TEDS\r\n");
               ReadMetaTEDS(OffSetTEDS);
            }
```

Figura 18 - Função que seleciona o código de acesso.

```
            else if(strcmp(TEDSCode, "003") == 0){
               UART_Printf(UART_CHANNEL, "GO to read
               TransducerChannel
               TEDS1\r\n");
               ReadChanTEDS1();
            }
            else if(strcmp(TEDSCode, "005") == 0){
               UART_Printf(UART_CHANNEL, "GO to read  Calibration
               TEDS1\r\n");
            }
            else if(strcmp(TEDSCode, "007") == 0){
               UART_Printf(UART_CHANNEL, "GO to read  End users'
               application-specific TEDS3\r\n");
            }
            else if(strcmp(TEDSCode, "012") == 0){
               UART_Printf(UART_CHANNEL, "GO to read  transducer
               name TEDS3\r\n");
            }
            else if(strcmp(TEDSCode, "013") == 0){
               UART_Printf(UART_CHANNEL, "GO to read  PHY
               TEDS1\r\n");
               ReadReplayQueryPhyTEDS();
            }
            else{
                  UART_Printf(UART_CHANNEL,"Falha no
                  comando....\r\n");
            }

      }

      //estrutura para comando write segment TEDS

     else if(strcmp(FuncID, "003") == 0){

         UART_Printf(UART_CHANNEL," Comando de escrita de

         segmento TEDS....\r\n");

          WriteTEDSSegment(OffSetTEDS,RawTedsBlock);

      }
      //estrutura para comando Update segment TEDS
      else if(strcmp(FuncID, "004") == 0){
           UART_Printf(UART_CHANNEL,"Comando de update de
           segmento TEDS....\r\n");
           UpdateTEDS();

      }

}
```

Figura 18 - Continuação.

É obrigatório a TIM fornecer uma resposta a todos os comandos *QueryTEDS*, independentemente do código de acesso TEDS selecionado. A resposta deste comando deve conter as informações apresentadas na Tabela 15.

Tabela 15- Consultar a resposta do comando QueryTEDS no campo de dados.

Campo	Tipos de dados	Nome de atributo de campo	Função
1	UInt8	TEDSAttrib	Atributo TEDS (consulte a Tabela 16)
2	UInt8	TEDSStatus	Estado da TEDS
3	UInt32	TEDSSize	Comprimento atual da TEDS
4	UInt16	TEDSCkSum	Soma de verificação da TEDS
5	UInt32	MaxTEDSSize	Comprimento máximo da TEDS

Tabela 16 - Atributo das TEDS.

Bit	Tipos de dados	Nome de atributo de campo	Definição
0(lsb)	Boolean	TEDSAttrib.ReadOnly	Somente leitura - verdadeiro se a TEDS puder ser lida, mas não for possível escrever.
1	Boolean	TEDSAttrib.NotAvail	Não suportado - verdadeiro se a TEDS não for suportada por este *TransducerChannel*.
2	Boolean	TEDSAttrib.Invalid	Inválido - verdadeiro se a imagem atual da TEDS for inválida.
3	Boolean	TEDSAttrib.Virtual	TEDS virtuais - verdadeiro se este for uma TEDS virtual. (Um TEDS virtual é qualquer TEDS que não é armazenado na TIM. A responsabilidade de a cessar uma TEDS virtual é atribuída à NCAP ou ao processador do *host*)
4	Boolean	TEDSAttrib.TextTEDS	TEDS de texto - verdadeiro se a TEDS for baseada em texto.
5	Boolean	TEDSAttrib.Adaptive	Adaptável - verdadeiro se o conteúdo da TEDS puder ser alterado pela TIM ou *TransducerChannel* sem que a NCAP emita um comando do segmento WriteTEDS.
6	Boolean	TEDSAttrib.MfgrDefine	MfgrDefine - verdadeiro se o conteúdo desta TEDS for definido pelo fabricante.
7(msb)	Boolean	TEDSAttrib.Reserved	Reservado.

***Read TEDS segment command*:** este comando é utilizado pela NCAP para ler um segmento do bloco de dados de uma TEDS específica da TIM. Este comando é dependente de dois argumentos, o código de acesso das TEDS e o seu *offset*. O tamanho máximo de uma TEDS sempre será maior que o tamanho máximo de um *array* de octetos. No entanto, nem sempre

o conteúdo de uma TEDS vai caber num *array* de octetos. O *offset* da TEDS é empregado para identificar onde a TEDS deve iniciar o acesso de leitura. Se uma TEDS caber dentro de um único segmento do comando, nestes casos, o conteúdo do campo de deslocamento *offset* deve ser zero. Se as TEDS forem maiores que o tamanho máximo de um *array* de octetos, a transmissão destas TEDS requerem uma segmentação da TEDS para transmissão. Este comando tem um comprimento de 5 octetos, onde 1 octeto é atribuído ao código de acesso à TEDS e 4 associados ao *offsetTEDS*. No entanto, a estrutura de mensagem de comando definida anteriormente ganha mais um campo de acordo com a Tabela 18.

Este comando utiliza a mesma estrutura da mensagem de resposta definida anteriormente. Os octetos dependentes da resposta são definidos na Tabela 17. O primeiro campo contém o deslocamento na TEDS no qual o bloco de dados foi obtido e, na maioria dos casos, corresponderá ao deslocamento do segmento da TEDS do comando *Read TEDS segment*. Os octetos restantes contêm os dados lidos das TEDS. A resposta deve abarcar todos os valores contidos no deslocamento do segmento TEDS. Se a *TEDSOffset* for maior que o comprimento da TEDS, a *TEDSOffset* na resposta será igual ao comprimento da TEDS e a resposta conterá 0 octetos.

Tabela 17 - Campo de dados para uma resposta do comando do segmento da TEDS.

Campo	Tipo de dados	Nome do atributo de campo	Função
1	UInt8	TEDSOffset	Deslocamento de segmento da TEDS (0 to [current size – 1]).
2	OctetArray	RawTEDSBlock	Octetos de dados da TEDS.

Tabela 18 - Estrutura de mensagem do comando ReadTEDS segment

ChannID	cmdClassID	FuncID	Comp	TEDSaccesscode	OffsetTEDS
chch	01	02	cc	03	10

O valor 10 no campo offsetTEDS especifica a posição onde a leitura da TEDS deve iniciar. Os outros campos são os mesmos definidos na estrutura de mensagem do comando *queryTEDS*.

***Write TEDS segment command*:** este comando é utilizado para escrever uma parte da TEDS. Os argumentos para este comando são descritos na Tabela 19. Se a *TEDSOffset* for maior que o comprimento máximo da TEDS, os dados devem ser descartados e o bit rejeitado do comando na palavra de estado deve ser activado.

Tabela 19 - Campo de dados para um comando Write TEDS segment.

Campo	Tipo de dados	Nome do atributo de campo	Função
1	UInt8	TEDSAccessCode	Permite selecionar uma TEDS específico. Ver a Tabela 14.
2	UInt32	TEDSOffset	Deslocamento de segmento TEDS (0 to [current size – 1]).
3	OctetArray	RawTEDSBlock	TEDS Conteúdo.

A estrutura deste comando é definida na Tabela 20. O tamanho máximo da TEDS não pode ser excedido. Se acontecer, os dados adicionais não devem ser gravados na memória e, o tamanho máximo atual deve ser definido como zero. Este comando também pode criar uma nova TEDS se ainda não existir, mas só se a TIM for projetada para permitir que as TEDS sejam criadas. Se não for projetada para criar novas TEDS, então o comando não gravará dados na memória das TEDS porque não é suportado.

Tabela 20 - Estrutura de mensagem de comando writeTEDS segment.

ChannID	cmdClassID	FuncID	Comp	TEDSaccesscode	OffsetTEDS	RawTEDSBlock
xx xx	01	03	cc	01	04	asdfghijk

A TIM pode reescrever uma TEDS existente. Nestes casos, a TEDS a ser escrito será marcada como inválida. Esta TEDS deverá permanecer marcada como inválida até que o comando *Update* da TEDS seja recebido. O comando *Write TEDS Segment* não gera uma resposta.

***Update TEDS command*:** este comando é utilizado para validar e copiar para a memória permanente uma TEDS gravada anteriormente através do comando *writeTEDS*. Se a validação falhar, as TEDS permanecerão marcadas como inválidas. Este comando é dependente do código de acesso as TEDS e, também utiliza a mesma estrutura de mensagem do comando e de resposta definida para o comando *QueryTEDS*.

4.2.6. Tipos de TEDS

O padrão define 4 tipos das TEDS necessárias e de caracter obrigatório para qualquer TIM, os outros são opcionais. Todas as TEDS apresentam a estrutura exposta na Tabela 21, onde o primeiro campo é *TEDS length,* utilizada para indicar o comprimento das TEDS, o segundo é o *Data Block* (Bloco de dados) utilizado para mostrar o conteúdo da informação das TEDS, e, o último campo é o *CheckSum* (soma de verificação) utilizada para verificar a integridade dos dados. O comprimento das TEDS é o número total de octetos no bloco de dados mais os dois octetos do *Checksum*. A Tabela 22 apresenta os diferentes tipos das TEDS definidas por este padrão. É importante destacar que serão desenvolvidas apenas TEDS de caracter obrigatório.

Tabela 21 - Formato Geral para qualquer TEDS

Campo	Descrição	octetos
----	TEDS length	4
1 a N	Data Block	Variável
----	Checksum	2

Tabela 22 - Tipos das TEDS

Tipo de TEDS	Obrigatório	Opcional
Cabeçalho de identificação TEDS		Sim
Meta-TEDS	Sim	
TransducerChannel TEDS	Sim	
Calibration TEDS		Sim
Frequency Response TEDS		Sim
Transfer Function TEDS		Sim
Text-based TEDS		Sim
End User Application Specific TEDS		Sim
User's Transducer Name TEDS	Sim	
Manufacturer-defined TEDS		Sim
PHY TEDS	Sim	

4.2.7. Meta-TEDS

O Meta-TEDS disponibiliza na interface todas as informações necessárias para obter o acesso a qualquer canal do transdutor, ou a todos os canais. A Meta-TEDS é utilizada pela NCAP para determinar se a TIM parou de responder. Além disso, descreve as relações entre os diferentes transdutores no dispositivo. O acesso à Meta-TEDS é feito utilizando os comandos *QueryTEDS, ReadTEDS, WriteTEDS* e *UpdateTEDS*. O argumento do comando deve especificar o código de acesso TEDS, conforme definido na Tabela 14.

A Tabela 24 define o bloco de dados de uma Meta-TEDS. Este bloco de dados é formado por um conjunto de dados variáveis com uma estrutura *Type/Length/Value* (TLV). O dado *Type* refere-se ao que está contido no campo de valor, com exceção de 2 e 3. O comprimento indica o tamanho correspondente a cada conjunto TLV. O valor contém qualquer informação específica. Os campos de comprimento e *checksum* não são parte do bloco de dados das TEDS, mas são expostos nesta tabela para manter o formato geral das TEDS.

O Cabeçalho de identificação TEDS ou *TEDS identification header* (TEDSID), é um campo padrão e obrigatório para todas as TEDS. Este campo encontra-se sempre no início das TEDS.

O comprimento da tupla para este campo é assumido como um. O mesmo é constituído por quatro campos mostrados na Tabela 23.

Tabela 23 - Estrutura de identificação das TEDS (TEDSID).

Campo	Conteúdo	Função
Tipo	03	Tipo de campo para identificador da TEDS.
Comprimento	04	Este campo é sempre definido como 04, indicando que o campo de valor contém 4 octetos.
Família	00	Este campo identifica o membro da família de padrões IEEE1451 que define a TEDS.
Classe	Tabela 11	Identifica a classe de comando a que se deseja ter acesso. O conteúdo para este campo é definido na Tabela 11.
Versão	0 ou 1	Este campo identifica a versão da TEDS. O valor é o número da versão identificado no padrão. Se o valor neste campo é zero significa que as TEDS não estão em conformidade com nenhum padrão publicado. Se o valor neste campo é um (1), então corresponde ao lançamento original deste padrão. Os valores no intervalo de 2 a 255 são reservados para futuras versões do padrão.
Tuple Length	Número de octetos	Este campo fornece o número de octetos no campo de comprimento de todas as tuplas nas TEDS, exceto este tuplo.

Tabela 24 - Estrutura do bloco de dados Meta-TEDS.

Campo	Nome	Descrição	Tipos de dados	octetos
----		Comprimento	UInt32	4
0 a 2	----	Reservado	----	----
3	TEDSID	Cabeçalho de Identificação TEDS	UInt8	4
4	UUID	Identificador Global Exclusivo	UUID	10
5 a 9	----	Reservado	----	----
10	OholdOff	Tempo de espera operacional.	Float32	4
11	SHoldOff	Tempo limite de acesso lento.	Float32	4
12	TestTime	Tempo de Autoteste.	Float32	4
13	MaxChan	Número de canais de transdutores implementados.	UInt16	2
14	CGroup	Sub-bloco de informações do Controlo Grupo. Os	----	----

		campos 20 e 21 definem este sub-bloco.		
15	VGroup	Sub-bloco de informações do Vetor Grupo. Os campos 20 e 21 definem este sub-bloco.	----	----
16	GeoLoc	Grupo de vetores especializados para localização geográfica. Os campos 20,21 e 24 definem este sub-bloco.	----	----
17	Proxies	Sub-bloco de definição de proxy do canal do transdutor. Os campos 21,22 e 23 definem este sub-bloco.	----	----
18 a 19	-----	Reservado.	----	----
20	GrpType	Tipo de Grupo de Controlo. É necessário para a TIM que implementa controlo de grupo e deve ser omitida quando não houver grupos.	UInt8	1
21	MemList	Lista de membros do grupo de controlo.	array of UInt16	NTV
22	ChanNum	Número do canal do transdutor do *proxy* do canal transdutor.	UInt16	1
23	Organiz	Organização do conjunto de dados do *proxy* do canal do transdutor.	UInt8	1
24	LocEnum	Uma enumeração que define como as informações de localização é fornecida.	UInt8	1
25-127	----	Reservado.	-----	-----
128-255	----	Aberto aos fabricantes.	------	-----
----		*Checksum*.	UInt16	2

O Identificador Global Exclusivo ou *Globally Unique Identifier* (UUID) é um campo obrigatório, único e universal ligado à TIM que contém 10 octetos e é definido pelo fabricante.

A Tabela 24 agrupa os transdutores em quatro grupos: O grupo de controlo identifica os transdutores que são utilizadas para controlar as operações de outros transdutores. Por exemplo: se ocorrer um evento de um sensor, pode-se associar a este, mais três transdutores, onde o primeiro será um sensor utilizado para medir a entrada analógica do sensor de eventos, o segundo seria um atuador utilizado para selecionar a histerese a definir no sensor de evento, e o terceiro seria também um atuador, mas para selecionar o limite a aplicar ao sensor de eventos. O grupo de vetores pode ser utilizado para definir uma relação matemática entre os diferentes transdutores. O grupo *GeoLoc* é um campo necessário para a TIM que armazena informações de localização geográfica. O grupo *proxies* é um campo necessário para a TIM que define os *proxies* nos canais dos transdutores.

O campo comprimento é o número total de octetos no bloco de dados da TEDS, mais os dois octetos resultantes da soma de verificação.

O *Checksum* deve ser o complemento da soma de todos os octetos anteriores, incluindo o campo de comprimento inicial da TEDS e o bloco de dados inteiro da TEDS. O cálculo da soma de verificação exclui o campo de soma de verificação.

A Figura 19 apresenta a estrutura do bloco de dados do Meta-TEDS implementado. Dentro desta estrutura encontram-se outras estruturas para representar os campos que formam a TEDS *ID* e os tipos de grupos.

```
typedef struct{

	uint32_t MetaTEDSLength;
	struct {
		uint8_t Type;
		uint8_t Length;
		uint8_t family;
		uint8_t Class;
		uint8_t Version;
		uint8_t TupleLength;
	}TEDSID;

	char GloballyUID[10];
	float OholdOff;
	float SHoldOff;
	float TestTime;
	uint16_t MaxChan;
	uint8_t GroupType;

	struct{
		uint8_t CGroupType;
		uint16_t CMemList[10];
	}CGroup;

	struct{

		uint8_t VGroupType;
		uint16_t VMemList[10];

	}VGroup;

	struct{
		uint8_t GLocEnum;
		uint8_t GGroupType;
		uint16_t GMemList[10];
	}GeoLoc;


	struct{

		uint16_t  ChanNum ;
		uint8_t Organiz;
		uint16_t PxMemList[10];
	}ProxiesTC;

	uint16_t MTChecksum;

}META_TEDS;
```

Figura 19 - Estrutura da Meta-TEDS implementada.

4.2.8. TEDS do canal do transdutor

A TEDS do canal do transdutor contém informações detalhada de um transdutor específico e disponibiliza na interface todas as informações referentes ao canal do transdutor a ser endereçado para permitir a operação e o acesso adequado ao transdutor. Os comandos para o acesso a esta TEDS são os mesmos definidos na Tabela 12. A estrutura do bloco de dados desta TEDS é definida na Tabela 25.

Tabela 25 - Estrutura do bloco de dados da TEDS do canal do transdutor.

Campo	Nome	Descrição	Tipos de dados	octetos
----		Comprimento	UInt32	4
0 a 2	----	Reservado	----	----
3	TEDSID	Cabeçalho de Identificação da TEDS	UInt8	4
4-9	----	Reservado		
Informações relacionadas ao canal do transdutor				
10	CalKey	Chave de calibração: estrutura obrigatória, contém a informação de onde será feita a calibração do sensor, se será fornecida pela TIM ou algum outro tipo de calibração.	UInt8	1
11	ChanType	Chave do tipo de canal do transdutor: é um campo obrigatória, indica um dos tipos de transdutores definidos pelo padrão.	UInt8	1
12	PhyUnits	Unidades Físicas: Campo obrigatório, indica as unidades físicas da informação do sensor. É usado para definir as unidades do SI para a quantidade física que está sendo medida ou controlada.	UNITS	11
13	LowLimit	Limite de alcance inferior operacional do projeto: Campo obrigatório, informa o limite inferior da operação do sensor.	Float32	4
14	HiLimit	Limite de alcance superior operacional do projeto: Campo obrigatório, informa o limite superior da operação do sensor.	Float32	4
15	OError	Incerteza do pior caso: Campo obrigatório, informa a maior incerteza possível.	Float32	4
16	SelfTest	Chave de autoteste: Campo obrigatório, informa	UInt8	1

		se o sensor possui autoteste.		
17	Mrange	Capacidade de multi-faixa: Este campo é opcional para a TEDS do canal do transdutor. Se não for implementado, a NCAP assumirá que não existe capacidade multi-faixa dentro deste Canal do Transdutor.	UInt8	1
Informações relacionadas ao conversor de dados				
18	Sample	Amostra: Campo obrigatório, informa dados referentes à amostragem do sinal do transdutor.		
19	DataSet	Definição de conjunto de dados: Campo obrigatório.		
Informações relacionadas ao tempo				
20	UpdateT	Tempo de atualização do canal do transdutor (t_u): Campo obrigatório, informa a taxa de atualização do sinal do transdutor. Operando livremente o valor deve ser zero.	Float32	4
21	WSetupT	Tempo de configuração de gravação do canal do transdutor (t_{ws}):	Float32	4
22	RSetupT	Tempo de configuração de leitura do canal do transdutor (t_{rs}): Campo obrigatório, informa quanto tempo após um sinal de Disparo o canal é lido. Operando livremente o valor deve ser zero	Float32	4
23	SPeriod	Período de amostragem do canal do transdutor (t_{sp}): Campo obrigatória, informa o tempo que o sensor demora para realizar a amostragem, limitado basicamente pelo tempo de conversão do AD. Quando o valor não é relevante o valor deve ser zero.	Float32	4
24	WarmUpT	Tempo de energização do canal do transdutor: Campo obrigatório, informa o tempo que o sensor demora para estabilizar sua operação ao ser energizado.	Float32	4
25	RDelayT	Tempo de atraso de leitura do canal do transdutor (t_{ch}): campo obrigatório, informa o tempo que o sensor demora entre o comando de leitura e o envio do *frame* de dados.	Float32	4
26	TestTime	Requisito de tempo de autoteste do canal do transdutor: Este campo é obrigatório para todos os canais de transdutor que implementam uma	Float32	4

		capacidade de autoteste. Se a chave de autoteste indicar que nenhum autoteste está implementado, esse campo pode ser omitido. Se a chave de autoteste indicar que existe uma capacidade de autoteste e este campo for omitido, a NCAP deverá reportar um erro da TEDS não fatal. A NCAP pode assumir um requisito de tempo de autoteste ou não a cessar o recurso de autoteste. Este campo contém o tempo máximo, expresso em segundos, necessário para executar o autoteste.		
Tempo da informação da amostra				
27	TimeSrc	Fonte para o tempo da amostra: este campo é opcional	UInt8	1
28	InPropDl	Atraso de propagação de entrada através da lógica de transporte de dados: Este campo é obrigatório só se o campo 27 contiver o valor “Entrada”; caso contrário, pode ser omitido. Se o campo Fonte para a hora da amostra contiver o valor Entrada e este campo for omitido, um erro TEDS não fatal será relatado.	Float32	4
29	OutPropD	Atraso de propagação de saída através da lógica de transporte de dados: Este campo é obrigatório se o campo Fonte para o tempo da amostra contiver o valor de “saída”.	Float32	4
30	TSError	Incerteza de atraso do disparo à amostra: Este campo é opcional para todos os canais de transdutor. Se for omitido, a NCAP assumirá um valor zero para indicar que a informação não é fornecida.	Float32	4
Atributos				
31	Sampling	Atributo de amostragem:	UInt8	1
32	DataXmit	Atributo de transmissão de dados:	UInt8	1
33	*Buffer*ed	Atributo *Buffer*ed	UInt8	1
34	EndOfSet	Atributo de fim do funcionamento do conjunto de dados:	UInt8	1
35	EdgeRpt	Atributo de margem para relatório:	UInt8	1
36	ActHalt	Atributo de parada do atuador:	UInt8	1
Sensibilidade				

37	Directon	Direção de sensibilidade:	Float32	4
38	DAngles	Ângulos de direção:	Float64	8
Opções				
39	ESOption	Opções do sensor de eventos:	UInt8	
61-127		Reservado		
128-255		Aberto aos Fabricante	----	----
		Checksum	UInt16	2

O campo Unidades físicas ou *PhyUnit* é constituído pelos 10 campos seguintes: *UnitType; Radians; SterRad; Meters; Kilograms; Seconds; Amperes; Kelvins; Moles; Candelas.* A Enumeração de interpretação das unidades físicas (*UnitType*), é um campo obrigatório para todas as TEDS do canal do transdutor. Se este campo for omitido, a NCAP deverá reportar um erro fatal da TEDS.

O campo amostragem (*Simple*) está constituído por mais três campos seguintes: O Data Model (*DatMode*) é um campo que descreve o modelo de dados utilizado ao emitir o segmento de conjunto de dados *Read TransducerChannel* ou o segmento de conjunto de dados *Write TransducerChannel* para este Canal de Transdutor; O *Data Model Length (Modlenth)* é um campo obrigatório. Esse campo pode ser omitido no caso dos tipos de dados real, de precisão dupla e de precisão real, de precisão única, pois esses tipos têm comprimentos fixos. Para todos os outros casos, se este campo for omitido, a NCAP deverá reportar um erro fatal da TEDS; O *Model significant bits* (SigBit) é um campo obrigatório. A NCAP deve reportar um erro fatal se for omitido.

A Figura 20 apresenta a estrutura do bloco de dados da TEDS do canal implementado. Dentro desta estrutura encontram-se duas estruturas para representar os campos que formam a TEDS *ID* e de unidades de medida.

```
typedef struct {


      uint32_t ChanTEDSLen;

      struct {
                  uint8_t family;
                  uint8_t Class;
                  uint8_t Version;
                  uint8_t TupleLength;
            }TEDSID;

            // TransducerChannel related information

            uint8_t CalKey;

            uint8_t ChanType;

            char PhysUnits[11];
```

Figura 20 - Estrutura da TEDS do canal implementado.

```
struct{

      uint8_t UnitType;
      uint8_t Radians;
      uint8_t SterRad;
      uint8_t Meters;
      uint8_t Kilogram;
      uint8_t Seconds;
      uint8_t Amperes;
      uint8_t Kelvins;
      uint8_t Moles;
      uint8_t Candelas;
      uint8_t UnitsExt;
}Units;

float LowLimit;
float HiLimit;
float OError;
uint8_t SelfTest;
uint8_t MRange;

//Data converter-related information Sample

uint8_t DatModel;/**<Data model.*/
uint8_t ModLenth;/**<Data model length.*/
uint16_t SigBits;/**<Model significant bits.*/

//Data converter-related information DataSet

uint16_t Repeats;/**<Maximum data repetitions.*/
float SOrigin;/**<Series origin.*/
float StepSize;/**<Series increment.*/
char SUnits[11];
uint16_t PreTrigg;/**<Maximum pre-trigger samples.*/

// Timing-related information
float UpdateT;
float WSetupT;
float RSetupT;
float SPeriod;
float WarmUpT;
float RDelayT;
float TestTime;
```

Figura 20 - Continuação 1.

```
            //Time of the sample information

            uint8_t TimeSrc;
            float InPropDl;
            float OutPropD;
            float TSError;

            // Attributes

            uint8_t Sampling;
            uint8_t SampMode;
            uint8_t SDefault;
            uint8_t DataXmit;
            uint8_t Buffered;
            uint8_t EndOfSet;
            uint8_t EdgeRpt;
            uint8_t ActHalt;

            //Sensitivity
            float Directon;
            double DAngles;

            //Option
            uint8_t ESOption;
            uint16_t MTChecksum;
} CHANNEL_TEDS;
```

Figura 20 - Continuação 2.

4.2.9. Nome do transdutor do utilizador TEDS

Este campo fornece um local para armazenar o nome pelo qual o sistema ou utilizador final conhecerá este transdutor. É recomendado para todos os Transdutores. O acesso a esta TEDS pode ser feito utilizando os comandos definidos na Tabela 12, mas com o argumento do comando do código de acesso diferente. O valor do argumento está definido na Tabela 14. A estrutura do bloco de dados para esta TEDS é apresentada na Tabela 26, mas o conteúdo e a estrutura do campo Bloco de dados são definidos pelo utilizador. É importante destacar que o cabeçalho de identificação é o mesmo que foi definido anteriormente. Este campo é obrigatório e não pode ser omitido. Se acontecer, a NCAP deverá reportar um erro fatal da TEDS. A Tabela 26 apresenta a estrutura do bloco de dados da TEDS do nome do transdutor do utilizador.

Tabela 26 - Estrutura do nome do transdutor do utilizador do bloco de dados da TEDS.

Campo	Nome	Descrição	Tipos de dados	octetos
---		Comprimento	UInt32	4
0 a 2	----	Reservado	----	----
3	TEDSID	Cabeçalho de Identificação TEDS	UInt8	4
4	Format	Descrição do formato deste TEDS	UInt8	1

5	TCName	TIM ou nome do *TransducerChannel*	----	----
---		*Ckecksum*	UInt16	2

O *Formato* é um campo obrigatório, cujo conteúdo e a estrutura do campo bloco de dados pode indicar se o bloco é definido pelo utilizador ou se for baseado em textos. Se o conteúdo for definido pelo utilizador o seu conteúdo deverá ser 0, e se for baseado em texto deverá ser 1. Para este campo são reservados para futuras expansões valores que estão de 2 a 255.

O fabricante deve determinar o tamanho desta TEDS, mas no mínimo, a TEDS deve ser grande o suficiente para acomodar um campo de bloco de dados de pelo menos 160 octetos com um nome de parâmetro de 32 caracteres mais um cabeçalho de identificação TEDS e um possível texto-baseado Cabeçalho dos TEDS. O utilizador não é obrigado a usar o cabeçalho TEDS baseado em texto [6].

```
typedef struct {

      uint32_t ChanTEDSLen;
      struct {
                  uint8_t family;
                  uint8_t Class;
                  uint8_t Version;
                  uint8_t TupleLength;
            }TEDSID;


            uint8_t Format;
            char TCName;

}UTName_TEDS;
```

Figura 21 - Estrutura da TEDS do nome do transdutor do utilizador.

4.2.10. PHY TEDS

É uma TEDS que descreve a camada física de comunicação entre a NCAP e a TIM. Estes módulos têm a função de disponibilizar todas as informações necessárias para obter o acesso a qualquer canal, além das informações comuns a todos os canais. Esta TEDS é somente de leitura e, a sua implementação é baseada no padrão IEEE P1451-2 UART. O acesso a esta TEDS é feito utilizando os mesmos comandos definidos na Tabela 12.

A Tabela 27 define a estrutura do bloco de dados da TEDS da camada física. A estrutura do bloco de dados desta TEDS implementada é definida na Figura 22.

Tabela 27 - Estrutura de bloco de dados PhyTEDS.

Campo	Nome	Descrição	Tipos de dados	octetos
3	TEDSID	Cabeçalho de Identificação TEDS	UInt8	4
10	RS232	IEEE 1452-2 camada física	UInt8	1

11	MaxBPS	Taxa de transferência máxima de dados	UInt32	4
12	MaxCDev	Dispositivos conectados máximos	UInt16	2
13	MaxRDev	Máximo de dispositivos registrados	UInt16	2
14	Encrypt	Encriptação	UInt16	2
15	Authent	Autenticação	Boolean	1
16	MinKeyL	Comprimento mínimo da chave	UInt16	2
17	MaxKeyL	Comprimento máximo da chave	UInt16	2
18	MaxSDU	Tamanho máximo de SDU	UInt16	2
19	MinALat	Latência mínima de acesso	UInt32	4
20	MinTLat	Min latência de transmissão	UInt32	4
21	MaxXact	Máximo de transações simultâneas	Uint8	1
22	Baterry	Dispositivo é alimentado por bateria	Uint8	1
23	Version	Versão	UInt16	2
24	MaxRetry	Tentativas máximas antes de desconectar	Uint8	1

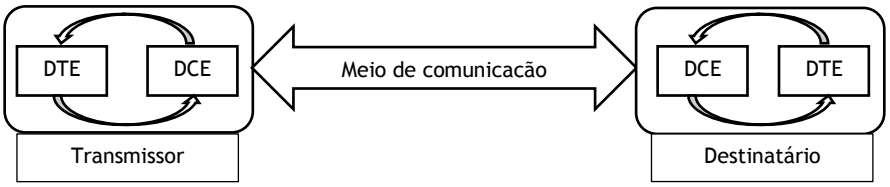
```
typedef struct{

      uint32_t ChanTEDSLen;
      struct {
                  uint8_t family;
                  uint8_t Class;
                  uint8_t Version;
                  uint8_t TupleLength;
            }TEDSID;

      uint8_t RS232;
      uint32_t MaxBPS;
      uint16_t MaxCDev;
      uint16_t MaxRDev;
      uint16_t Encrypt ;
      uint16_t MinKeyL ;
      uint16_t MaxKeyL ;
      uint16_t MaxSDU ;
      uint32_t MinALat ;
      uint32_t MinTLat ;
      uint8_t MaxXact ;
      uint8_t Baterry ;
      uint16_t Version ;
      uint8_t MaxRetry ;

}Phy_TEDS;
```

Figura 22 - Estrutura da PHYTEDS.

4.3. Módulo de comunicação

Uma característica importante dos sistemas modernos baseados em microprocessadores é a sua capacidade de comunicação. Isto é, a sua capacidade para trocar informações com outros

sistemas no ambiente envolvente. As interfaces de comunicação podem ser usadas para atualização de *firmware* ou carregamento de parâmetros locais. Num nível superior, essas interfaces podem ser usadas para trocar informações em aplicativos com processos distribuídos [19]. O modelo do sistema de comunicação de qualquer sistema de comunicação digital possui três dispositivos: Transmissor; Destinatário; Meio de comunicação. Figura 23.



DTE - Data Terminal Equipment.
DCE - Data Communications Equipment.

Figura 23 - Sistema de comunicação.

O modo de comunicação entre dispositivos digitais pode ser paralela ou serie. No entanto, o modo de comunicação utilizado para comunicação entre a NCAP e a TIM, e vice-versa, é o modo serie UART por atingir altas taxas de transmissão. Nesta comunicação, o meio físico de transmissão precisa apenas de uma linha de sinal, ao contrário da comunicação paralela que precisa de mais linhas de sinal independentes para cada um dos bits do valor digital transmitido. Na comunicação serie a informação é enviada pelo transmissor como uma sequência de bits, a uma taxa comum estabelecida entre o transmissor e o destinatário. Informações adicionais são necessárias para permitir a sincronização entre os participantes da comunicação: *Start* bit; *Stop* bit [18]. A Figura 24 mostra a comunicação cablada do tipo cliente-servidor entre a NCAP e a TIM utilizando o protocolo p1451-2 UART, a uma taxa de 115,200 bps. A NCAP envia uma solicitação à TIM, a que esta retorna uma resposta.

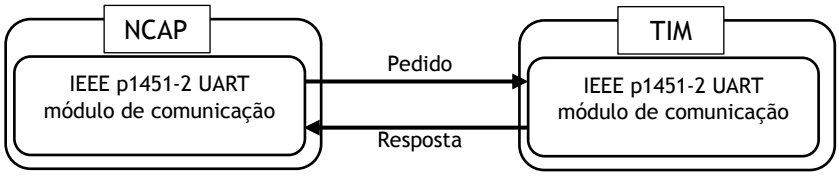


Figura 24 - módulo de comunicação entre a NCAP e o TIM.

4.4. Resultados das simulações

Para realização das simulações entre a NCAP e a TIM foi necessário recorrer a um emulador de terminal de console serial (*PuTTY*) para fazer o papel da NCAP. Na Figura 25 é apresentada um comando *queryTEDS* para obter informações necessárias para ler um Meta-TEDS enviado a partir do *PuTTY*. O $ no início do comando é para inicializar a comunicação. A resposta a este comando é apresentada na Figura 26. O primeiro valor corresponde à *flag* de sucesso, o segundo valor corresponde ao tamanho da mensagem e, do terceiro ao último valor, representam valores do comando dependentes definidas na Tabela 15. Outros comandos

também testados são apresentados na Tabela 28. A resposta para cada um destes comandos são apresentados na Figura 27.

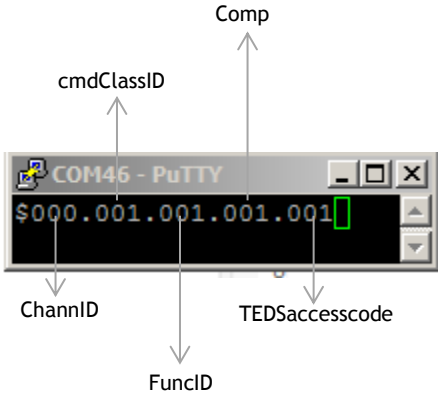


Figura 25 - Comando queryTEDS.

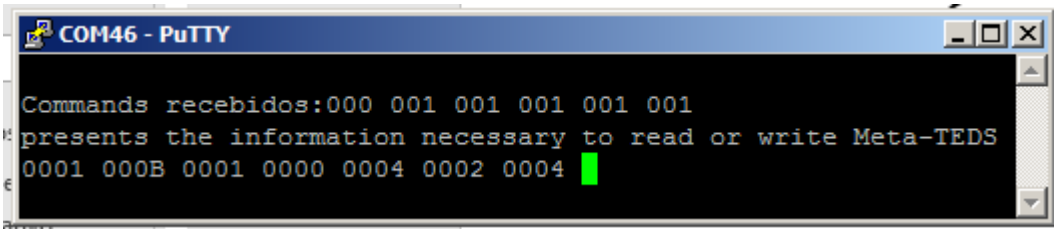


Figura 26 - Resposta ao comando queryTEDS.

Nesta Figura 27, a primeira linha apresenta o comando recebido pela TIM para ler o Meta-TEDS. As três linhas seguintes presentam a resposta da solicitação. Na quinta linha a TIM recebe uma ordem da NCAP para escrever "Angola" no campo *4* do Meta-TEDS. O comando *write segment* apenas permite guardar a informação temporariamente na memória volátil. Para testar a veracidade desta funcionalidade é recebida um comando para ler o Meta-TEDS no campo 4 *onde foi escrito Angola*. Após a execução deste comando verifica-se que o valor para este campo continua a ser o mesmo predefinido, o que significa, que o comando *write* não gravou a informação na memória permanente. Para gravar a informação contida na memória volátil para permanente, a TIM recebe um comando *Update*. De seguida é solicitado novamente o comando *read segment* para verificar se a informação foi gravada.

Tabela 28 - Comandos Testados.

Comandos	ChannID	ClassID	FuncID	Comp	AccessCode	Offset	RawTEDSBlock
Read S.TEDS	000	001	002	005	001	000	---
Write S. TEDS	000	001	003	006	001	004	Angola
Update TEDS	000	001	004	001	001	---	---
Read S.TEDS	001	001	002	005	001	004	---

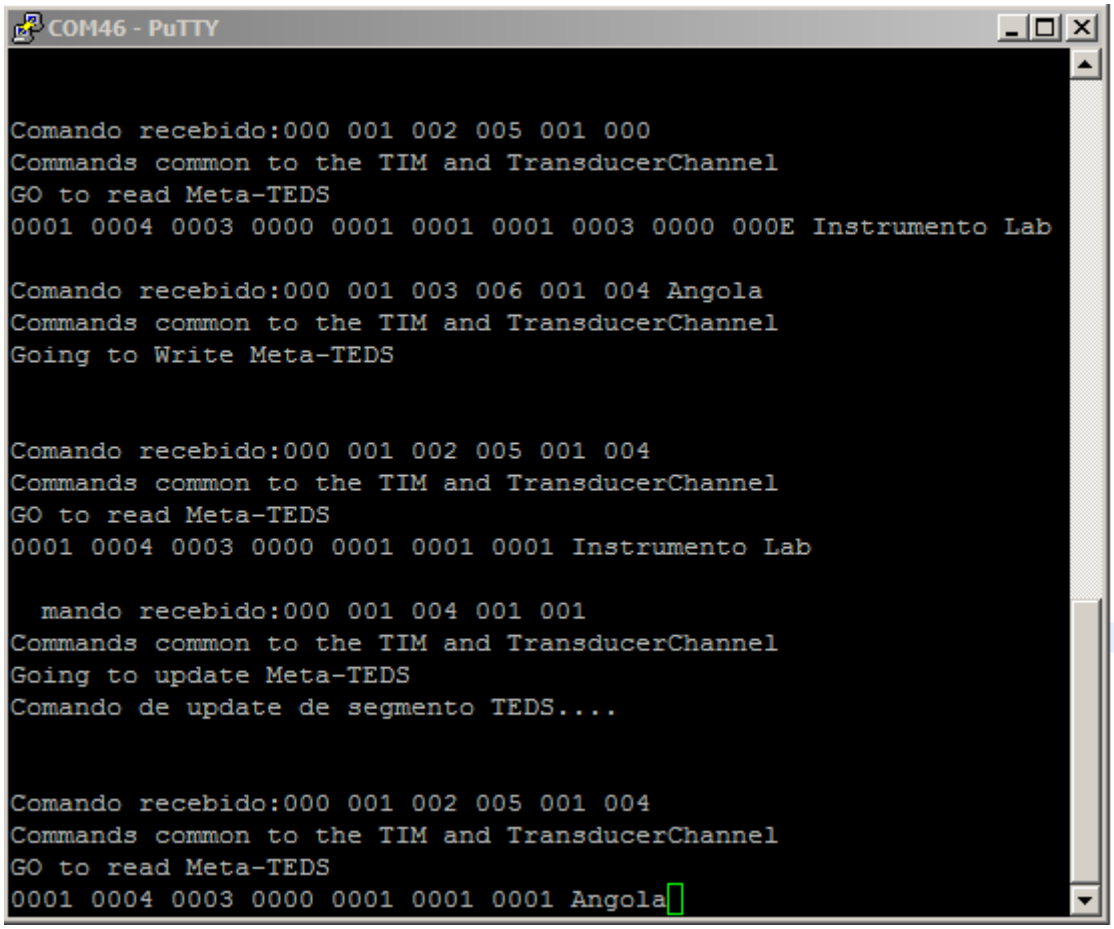


Figura 27 - Resposta dos comandos da Tabela 28.

Capítulo 5 - Conclusões e recomendações

5.1. Conclusões e recomendações

Graças à evolução tecnológica, particularmente observada na última década, estimulada pelo crescimento da Internet das coisas e da microeletrónica, é possível encontrar uma grande variedade de transdutores. Este aspeto deve ser tido em conta na automação de infraestruturas. Este trabalho apresenta o desenvolvimento de um nodo do transdutor inteligente (TIM) capaz de se comunicar através de uma ligação serial UART em qualquer nodo da rede (NCAP) baseados nos padrões ISO/IEC/IEEE 21451-0 e p21451-2 para ajudar a obter a flexibilidade desejável. Apesar da existência da família de padrões ISO/IEC/IEEE 21451 que propõem um conjunto padronizado de interfaces de hardware e software que permite transdutores conectados à rede, a flexibilidade ainda não foi alcançada devido à ampla variedade de redes e diversidade de protocolos existentes ou à criação de soluções independentes pelos fabricantes. No entanto, para garantir a flexibilidade neste projeto foram desenvolvidos serviços de folhas de dados eletrónicos de transdutores (TEDS), estruturas de mensagem de comando e de respostas baseados na norma ISO/IEC/IEEE 21451-0.

Para comunicar um módulo da TIM de um fabricante diferente com qualquer módulo da NCAP de outro fabricante é necessário que ambos os módulos utilizem o mesmo protocolo de comunicação. A padronização de interface de hardware/software que funcionam com a flexibilidade tornará os sistemas de monitoramento e controlo distribuídos mais praticáveis.

A implementação de um sistema em conformidade com as diretrizes da norma ISO/IEC/IEEE 21451 e a filosofia do código aberto permitirá: reduzir os custos dos sistemas de instrumentação; simplificar a transmissão de informações através de uma rede; facilitar na ampliação dos sistemas sem efetuar grandes configurações.

Para trabalhos futuros, recomenda-se: implementar outros modelos de comunicação previstos na norma P21451-2 e 21451; desenvolver funções que permitam o uso de todos os comandos definidos pelo padrão; construir funções que possibilita controlar e reduzir o consumo energético do nodo do transdutor.

Referências bibliográficas

[1] E. A. Batista, *"Implementação de uma plataforma HW / SW para Automação Industrial , utilizando Hardware Reconfigurável com Processador NIOS II em conformidade com o padrão IEEE 1451,"* FEIS. Ilha solteira - SP, 2009.

[2] M. Presser do Alexandra, *"Inspirando a Internet das Coisas,"* Agns Gráfi. 2015.

[3] K. B. L. Eugene Y Song, "'Wireless sensor network based on ieee 1451.0 and ieee 1451.5-802.11. Electronic Measurement and Instruments,'" in *CEMI'07. 8th International Conference on*, 2007.

[4] K. B. L. Eugene Y Song, "'An implementation of the proposed IEEE 1451.0 and 1451.5 standards,'" in *In: Sensors Applications Symposium, 2006. Proceedings of the 2006 IEEE.*, 2006.

[5] E. Y. Song and K. B. Lee, "'Sensor network based on IEEE 1451.0 and IEEE p1451.2-RS232,'" *Conf. Rec. - IEEE Instrum. Meas. Technol. Conf.*, pp. 1728–1733, 2008.

[6] IEEE Instrumentation and Measurement Society, *"IEEE Standard for a Smart Transducer Interface for Sensors and Actuators Wireless Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats,"* no. September. 2007.

[7] I. Standard, *"Information technology - Smart transducer interface for sensor and actuadors - Part 1: Network Capable Application Processor (NCAP) information model,"* vol. 2010. 2010.

[8] I. Standard, *"INTERNATIONAL STANDARD ISO / IEC / IEEE transducer interface for sensors and communication protocols and formats,"* vol. 2010. 2010.

[9] IEEE Standards, *"Overhead Transmission Line,"* vol. 2003, no. March. 2004.

[10] R. Silvano Renato, "'Implementação de um nó IEEE 1451, baseado em ferramentas abertas e padronizadas, para aplicações em ambientes de instrumentação distribuída,'" Universidade Estadual Paulista "Júlio de Mesquita Filho" campus de Ilha Solteira, Ilha Solteira – SP, 2005.

[11] I. Standard and I. S. Activities, *"IEEE Standard for a Smart Transducer Interface for Sensors and Actuators Wireless Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats,"* vol. 2010. 2010.

[12] E. Y. Song, K. B. Lee, S. E. Fick, and A. M. Donmez, "An IEEE 1451.5-802.11 standard-based wireless sensor network with embedded WTIM," *Conf. Rec. - IEEE Instrum. Meas. Technol. Conf.*, pp. 1201–1206, 2011.

[13] I. Standard, "'INTERNATIONAL STANDARD ISO / IEC / IEEE transducer interface for

sensors and communication protocols and formats,'" vol. 2010, p. 82, 2010.

[14] R. L. Fischer and J. Burch, "The PICmicro ® MCU as an IEEE 1451.2 Compatible Smart Transducer Interface Module (STIM)," *Meas. Control*, 2000.

[15] L. PRYTULA, I. MULLER, V. BRUSSAMARELLO, C. E. PEREIRA, and S. Y. C. CATUNDA, "Ieee 1451 smart sensor: low cost, low power wireless case study," in *Anais do XIX Congresso Brasileiro de Automática*, 2012, pp. 3731–3736.

[16] P. D. Gaspar, A. Espírito Santo, B. Ribeiro, and H. Santos, "MSP430 Teaching ROM, 2st Edition. 2012.," 2012.

[17] L. PRYTULA, "'PROJETO DE DIPLOMAÇÃO - SENSOR INTELIGENTE,'" Universidade Federal do Rio Grande do Sul, Escola de Engenharia, Porto Alegre, 2011.

[18] M. de O. Namba, "Modelagem e Especificação de um Middleware para Redes de Sensores Sem Fio Aplicado à Saúde," Universidade Federal de Goiás, 2011.

[19] P. D. Gaspar, A. Espírito Santo, B. Ribeiro, and H. Santos, *MSP430 Teaching ROM, 1st Edition*. 2009.

[20] I. Texas, "'MSP-EXP430F5529 Experimenter ' s Board User ' s Guide,'" no. April, 2017.