UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE FÍSICA DE SÃO CARLOS

PEDRO VICTOR BRONDINO DUARTE DE SOUZA

Desenvolvimento de um subsistema *non-real-time* para o gerenciamento de
dispositivos periféricos e desenvolvimento de interfaces gráficas

São Carlos
2016

PEDRO VICTOR BRONDINO DUARTE DE SOUZA

Desenvolvimento de um subsistema *non-real-time* para o gerenciamento de dispositivos periféricos e desenvolvimento de interfaces gráficas

Dissertação submetida ao Programa de Pós-Graduação em Física do Instituto de Física de São Carlos da Universidade de São Paulo, para a obtenção do título de Mestre Ciências.

Área de Concentração: Física Aplicada
Opção: Física Computacional
Orientador: Prof. Dr. Alberto Tannús

Versão Corrigida

(versão original disponível na Unidade que aloja o Programa)

São Carlos

2016

AUTORIZO A REPRODUÇÃO E DIVULGAÇÃO TOTAL OU PARCIAL DESTE TRABALHO, POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO PARA FINS DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

Ficha catalográfica revisada pelo Serviço de Biblioteca e Informação do IFSC,
com os dados fornecidos pelo(a) autor(a)

Souza, Pedro Victor Brondino Duarte de
Desenvolvimento de um subsistema *non-real-time*
para o gerenciamento de dispositivos periféricos e
desenvolvimento de interfaces gráficas / Pedro
Victor Brondino Duarte de Souza; orientador Alberto
Tannús - versão corrigida -- São Carlos, 2016.
120 p.

Dissertação (Mestrado - Programa de Pós-Graduação em
Física Computacional) -- Instituto de Física de São
Carlos, Universidade de São Paulo, 2016.

1. Instrumentação de ressonância magnética. 2.
Transducer electronic data sheet. 3. Subsistema de
periféricos *non-real-time*. I. Tannús, Alberto,
orient. II. Título.

Dedico este trabalho à Amanda C. Murgo pela paciência, compreensão, zelo e carinho
que me auxiliaram no desenvolvimento deste trabalho.

AGRADECIMENTOS

Agradeço a Deus pela vida e por me guiar nos caminhos pedregosos mas de felicidade certa.

Agradeço ao Professor Doutor Alberto Tannús pela oportunidade e orientação neste trabalho.

Agradeço ao Edson L. G. Vidoto e ao Mateus J. Martins pelas valorosas contribuições que puderam engrandecer este trabalho.

Agradeço aos colegas de laboratório Daniel C. Pizetta, José Teixeira, Felipe B. Coelho, Danilo M. D. D. da Silva, Douglas Shimada, Gustavo Lourenço e todos os outros que contribuíram de alguma forma para a elaboração deste trabalho.

Agradeço aos meus pais, irmã e família por todo o esforço, empenho e dedicação de modo que eu pudesse chegar até este ponto.

Agradeço à Amanda C. Murgos que tem sido muito importante em minha vida, sendo fonte de forças e de encorajamento para enfrentar todos os obstáculos e por me ajudar a não desviar do caminho e dos objetivos traçados.

RESUMO

SOUZA, P. V. B. D. de. Desenvolvimento de um subsistema non-real-time para o gerenciamento de dispositivos periféricos e desenvolvimento de interfaces gráficas. 2016. 120 p. Dissertação (Mestrado em Ciências) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2016.

Ressonância Magnética (RM) é uma técnica bastante versátil, pois é utilizada em muitas áreas de pesquisa, como biologia, física, química, engenharia e medicina. Apesar disso, constitui-se uma desvantagem o alto custo do equipamento e suas restrições físicas para alguns experimentos. Para reduzir essas desvantagens, o grupo de pesquisadores do CIERMag está desenvolvendo um equipamento de RMN multipropósito e, para complementá-lo, este trabalho vem desenvolver um subsistema *non-real-time* que gerencia os dispositivos periféricos de um experimento. Foi proposta uma rede de dispositivos que é controlada por um *Raspberry Pi* como elemento central, o qual está conectado ao terminal computadorizado do sistema através de uma rede local (*Local Area Network* - LAN) via Ethernet e conectado aos dispositivos periféricos via *Serial Peripheral Interface* (SPI). Com o objetivo de ser possível gerenciar qualquer tipo de dispositivo, foi desenvolvido um conjunto de parâmetros baseado no *Transducer Electronic Data Sheet* (TEDS), definido no padrão IEEE 1451, de modo que cada dispositivo, normalmente um transdutor, possui seu próprio. Foram elaboradas duas interfaces de software: uma desenvolvida em Python e a outra uma interface web HTML. Ambas as interfaces possuem as mesmas funcionalidades: editor e gerenciador de TEDS, visualização gráfica de medidas dos sensores e interface para os atuadores.

Palavras-chaves: Instrumentação de ressonância magnética. *Transducer electronic data sheet*. Subsistema de periféricos *non-real-time*.

ABSTRACT

SOUZA, P. V. B. D. de. Development of a non-real-time subsystem to manage peripheral devices and development of graphical interfaces. 2016. 120 p. Dissertação (Mestrado em Ciências) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2016

Magnetic Resonance (MR) is a very versatile technique, since it is used in many research areas such as biology, physics, chemistry, engineering and medicine. Despite this, the cost of the equipment and its physical restrictions in some experiments constitute a serious drawback. To minimize these problems, the CIERMag research team is developing multipurpose MR equipment and, to complement this equipment, this work develops a non-real-time subsystem that manages the peripheral devices of the experiment. It was proposed a network of devices controlled by a Raspberry Pi as its central element, which is connected to the terminal computer of the system. The protocol adopted for this was Ethernet via Local Area Network (LAN); communication with peripheral transducers was performed with the Serial Peripheral Interface (SPI). With the objective to be able to manage any type of device, we created parameter sets based on Transducer Electronic Data Sheet (TEDS), defined in the IEEE 1451 standard, so each device, normally a transducer, has its own parameter set. We created two software interfaces: one developed with Python and the other is a HTML web interface. Both have the same functionalities: a TEDS editor and manager, a graphical visualization of sensor measurements and also an actuator interface.

Keywords: Magnetic resonance instrumentation. Transducer electronic data sheet. Non-real-time peripheral subsystem.

LISTA DE FIGURAS

Figura 1 - Raspberry Pi modelo B.	19
Figura 2 - Modelo de camadas do OSI	26
Figura 3 - Camadas do Modelo Internet.	28
Figura 4 - Arquitetura do protocolo SPI.	29
Figura 5 - Diagrama de pulsos para o protocolo SPI.....	30
Figura 6 - Arquitetura do protocolo I2C	31
Figura 7 - Diagrama de pulsos para o protocolo I ² C	31
Figura 8 - Interconexão entre os padrões IEEE 1451	33
Figura 9 - Arquitetura geral do sistema ao integrar o subsistema de periféricos.	42
Figura 10 - Arquitetura do subsistema de periféricos dividida em módulos.	43
Figura 11 - Arquitetura final para o subsistema de periféricos.	44
Figura 12 - Pinos de entrada e saída disponível no Raspberry Pi.....	45
Figura 13 - Diagrama do circuito para multiplexar os sinais do Raspberry Pi.	46
Figura 14- Arquitetura lógica de conexão entre os diferentes módulos do subsistema.....	47
Figura 15 - Cabeçalho do TEDS do CIERMag.	49
Figura 16 - TEDS com o cabeçalho e um registrador de função.....	53
Figura 17 - Definição dos Bytes e Bits no processo de comunicação entre o Raspberry Pi e o transdutor. A sequência é da esquerda para direita, ou seja, o byte0 é enviado primeiro, seguido do byte1 e sucessivamente. Existem casos em que, durante uma comunicação, para o escravo enviar dados ao mestre, é necessário que o mestre envie bytes de comando seguido de bytes sem importância de maneira que o clock gerado permita ao escravo responder ao comando solicitado, retornando os bytes necessários.....	53
Figura 18 - Exemplo de preenchimento dos campos: os bytes são enviados do byte0 ao byte4 nessa ordem. Os bits marcados são referentes ao dado utilizado pelo transdutor. Portanto o campo <i>First Data Byte</i> será preenchido com o valor 1, o campo <i>Last Data Byte</i> será preenchido com o valor 2, o campo <i>First Data Bit</i> será preenchido com o valor 3 e o campo <i>Last data Bit</i> será preenchido com o valor 2. No caso de um sensor, o dado conterá o valor de uma	

medição, já para o caso de um atuador, o dado representa o valor que deve ser enviado.....	54
Figura 19 - Formato genérico do arquivo de dados dos atuadores.	58
Figura 20 - Exemplo do formato dos dados contidos no arquivo dos sensores.	59
Figura 21 - Trecho de código usado no Servidor desenvolvido, em que é definida a função para retornar a página de login.....	60
Figura 22 - Editor e gerenciador de TEDS da interface gráfica em Python.....	63
Figura 23 - Trecho do código HTML para página "main_page".....	67
Figura 24 - Preenchimento do TEDS de exemplo	72
Figura 25 - Interface Python após a criação do TEDS de exemplo.....	73
Figura 26 - Interface Python com as 4 áreas: editor de TEDS, lista de TEDS disponíveis, interface dos atuadores e gráfico dos sensores.	74
Figura 27 - Página de login da interface web do subsistema	75
Figura 28 - Página principal da interface web do subsistema	76
Figura 29 - Página do editor de TEDS da interface web.....	77
Figura 30 - Continuação da página do editor de TEDS, mostrando os campos do registrador de função para serem preenchidos.	78
Figura 31 - Fim da página do editor de TEDS da interface web, mostrando os demais links da página.	79
Figura 32 - LEDs indicando o valor inicial no registrador de deslocamento antes do envio do comando para a mudança do valor efetuado através da interface web.....	80
Figura 33 – LEDs indicando o valor final no registrador de deslocamento após o envio do comando para a mudança do valor efetuado através da interface web	81
Figura 34 - Página dos sensores com o gráfico interativo para cada dispositivo.	82

SUMÁRIO

1	INTRODUÇÃO.....	15
2	MATERIAIS E MÉTODOS	17
2.1	Equipamentos de Ressonância Magnética	17
2.2	Espectrômetro do CIERMag	18
2.3	Raspberry Pi	19
2.4	Python	20
2.4.1	Módulos Padrão Utilizados	20
2.4.2	Módulos de Terceiros Utilizados.....	22
2.5	Hypertext Transfer Protocol - HTTP	22
2.6	Hiper Text Markup Language - HTML	24
2.7	Cascading Style Sheets - CSS	25
2.8	Jinja.....	25
2.9	Modelo ISO/OSI e Modelo Internet	26
2.10	Escolha do protocolo de comunicação entre dispositivos.....	28
2.11	IEEE 1451 e Transducer Electronic Data Sheet (TEDS).....	32
3	DESENVOLVIMENTO	39
3.1	Análise do sistema e seus requisitos.....	39
3.2	Arquitetura do subsistema de periféricos.....	41
3.3	Transducer Electronic Data Sheet do CIERMag.....	47
3.3.1	Implementação do TEDS em Python	54
3.4	Gerenciador de transdutores.....	56
3.5	Servidor HTTP	59
3.6	Integração do Gerenciador de Transdutores com o Servidor.....	61

3.7	Cliente Python	62
3.8	Cliente Web.....	65
4	RESULTADOS	69
4.1	Criação de um TEDS através da interface gráfica Python.....	69
4.2	Criação de um TEDS através de um navegador web.....	74
5	CONCLUSÕES	83
5.1	Trabalhos futuros.....	83
	REFERÊNCIAS.....	85
	ANEXO A – DATASHEET MAX31856	89

1 INTRODUÇÃO

A Ressonância Magnética Nuclear (RM) é uma técnica bastante importante e versátil, pois permite que, de forma não invasiva, sejam observadas e analisadas as substâncias presentes no corpo de teste, quer seja ele uma amostra inanimada, ou um ser vivo. (1-2) Contudo, a técnica é bastante complexa, necessitando de equipamento de alta tecnologia. Nesse sentido, existem muitos fabricantes que disponibilizam equipamentos que atendem às necessidades tanto da área da saúde e biologia, para estudo e análise de tecidos e metabólitos, quanto da área da química e física para análises de substâncias. Porém, existe uma área em que os equipamentos comerciais não atendem idealmente qual seja, a de desenvolvimento de novas metodologias e sequências de pulsos, que desenvolve a base utilizada pelas áreas anteriormente citadas.

O sucesso de um experimento de RM depende da metodologia empregada, pois sequências de pulsos específicas atendem a específicos metabólitos, tecidos ou substâncias químicas. O desenvolvimento dessas metodologias e sequências de pulsos dependem da versatilidade do equipamento em poder atender, tanto a nível de hardware quanto a nível de software a realização de modificações necessárias, o que não ocorre com os equipamentos comerciais. Por serem sistemas proprietários, dependem que o fabricante faça as modificações necessárias, o que nem sempre ocorre ou ocorre dentro do tempo necessário para a pesquisa acadêmica.

Nesse sentido, o grupo Centro de Imagens e Espectroscopia in vivo por Ressonância Magnética (CIERMag), encontrando esses obstáculos, tomou a decisão de desenvolver o próprio espectrômetro, elemento central do equipamento de RM. O projeto foi idealizado para sanar as principais dificuldades: custo alto, obsolescência e acesso completo do pesquisador ao hardware e ao software. Foi desenvolvido, então, o espectrômetro baseado em conceitos de Logica Programáveis (FPGAs) que permite com que ele seja genérico o suficiente para ser utilizado, teoricamente, em qualquer experimento, seja de imagens ou de espectroscopia.

O espectrômetro sintetizado em FPGA possui todos os requisitos necessários para operar com os componentes principais, como as bobinas de gradiente, bobinas transmissoras e receptoras. Porém, ainda era necessário associar ao equipamento um subsistema capaz de controlar e monitorar os componentes periféricos, tais como radiofrequência, cama para paciente, gerenciador de amostras, magneto, sistema eletrônico, entre outros. Desses requisitos é que surge o objetivo desse trabalho, conceber e desenvolver esse subsistema.

Como o espectrômetro controla todas as linhas de *real-time* que são diretamente dependentes da metodologia do experimento, o subsistema de periféricos vem controlar e monitorar os elementos que são estáticos ao longo do experimento, tal como temperatura de amostra, potência de RF transmitida ou refletida, temperatura do magneto e do conjunto de bobinas de gradientes, ou seja, os elementos *non-real-time*.

O subsistema de periféricos foi concebido para ser um sistema modular, de modo que qualquer uma de suas partes possa ser configurada independente do experimento e independente do equipamento. Para que isso fosse possível, foi utilizado o conceito de *Transducer Electronic Data Sheet* (TEDS), que é definido pela norma IEEE 1541, e adaptado para o conjunto de dispositivos periféricos disponíveis.

O elemento centralizado de cada módulo é um *Raspberry Pi*, um micro controlador ARM montado sobre uma placa de desenvolvimento, contendo pinos de GPIO, USB, Ethernet, HDMI e capacidade de processamento para gerenciar os periféricos, com a vantagem de ter um baixo custo e possuir uma versão de Linux, chamada Raspbian, que permite inclusive executar scripts em Python.

A interface com o usuário foi toda elaborada utilizando a linguagem Python, por ser uma linguagem de alto nível, modular e, por ser interpretada, independe do sistema operacional. Entre os seus módulos, foram utilizados, por exemplo, o módulo PyQtGraph para a criação dos gráficos para os sensores e a interface de entrada para o TEDS, o módulo Flask, que permitiu criar um Servidor HTTP para a troca de informações entre o Raspberry Pi e o terminal do usuário e também o módulo *httplib* que permitiu a criação do Cliente Web.

No próximo capítulo serão discutidas, detalhadamente, cada parte do sistema, sua elaboração e seu desenvolvimento. No capítulo final, as conclusões, limitações e perspectivas de futuros trabalhos serão abordadas.

2 MATERIAIS E MÉTODOS

2.1 Equipamentos de Ressonância Magnética

A Ressonância Magnética (3) é um fenômeno que se origina no spin nuclear e seu alinhamento com um campo magnético externo estacionário. Ao se aplicar pulsos eletromagnéticos, seguindo a frequência de Larmor, estes perturbam o estado de equilíbrio desse alinhamento. Quando o spin é removido de seu estado de repouso ele tende a retornar ao equilíbrio com o efeito de precessão, produzindo uma variação de fluxo de indução magnética no espaço que o circunda, e consequente uma força eletromotriz, na mesma frequência de Larmor que pode ser detectada por uma bobina que intercepte aquele fluxo.

Dessa forma, um equipamento simples pode ser constituído de um magneto e as bobinas de transmissão e de recepção dos sinais de radiofrequência (RF). Os magnetos têm evoluído de modo que podem constituir em um módulo com a sua complexidade variando se é um magneto de imã permanente (4) ou um magneto de imã supercondutor. As bobinas de transmissão necessitam de eletrônica adequada para a potência dos pulsos de RF e também podem ser agrupados em um módulo. É possível alocar as bobinas de gradiente também em um módulo que contém os amplificadores para o sinal recebido e os digitalizadores.

Em sistemas mais complexos utilizados atualmente, para produzir a seleção dos planos em que os núcleos serão excitados, existe a necessidade de se criar um campo magnético com gradiente utilizando bobinas específicas para este propósito, que também necessitam de eletrônica dedicada para mudar constantemente a orientação do plano de seleção, podendo ser também incluído em um módulo.

Os equipamentos para espectroscopia, que têm por finalidade realizar análises em amostras, tanto de sólidos quanto de líquidos, podem necessitar de dispositivos que possam aquecer ou resfriar a amostra, ou girá-la em diferentes velocidades, e podem ser agrupados em outro módulo dedicado.

Nos equipamentos para imagens, de humanos ou animais, pode-se necessitar de dispositivos para acomodar adequadamente o objeto de estudo, também podendo ser agrupados em um módulo.

Nota-se que o equipamento de ressonância magnética pode adquirir elevada complexidade do sistema, evidenciando as várias possibilidades de dispositivos que podem

ser integrados e a importância de um subsistema capaz de gerenciar adequadamente todos os dispositivos envolvidos.

2.2 Espectrômetro do CIERMag

Existem, atualmente, alguns fabricantes de equipamentos de RM como Bruker, Philips, GE, Siemens entre outros, que possuem equipamentos altamente sofisticados voltados para segmentos específicos de mercado, como espectroscopia de sólidos, espectroscopia de líquidos, imagens em seres humanos, entre outras. Dessa maneira, os equipamentos possuem um número específico pré-programado de sequências de pulsos que contemplam as necessidades dos usuários do equipamento para as aplicações definidas. Isso é bastante vantajoso quando o equipamento é uma ferramenta para se obter uma medição ou uma imagem, porém quando o equipamento é utilizado para se desenvolver novas sequências de pulsos, não possuem a liberdade necessária, por razões diversas, tais como por segurança (do operador ou, como nos casos de imagens em humanos, o próprio paciente), limitação do equipamento ou por ser necessário que o fabricante libere o acesso a componentes (software e hardware) sigilosos de sua arquitetura e funcionalidade.

Deste modo, a pesquisa de novas sequências de pulso fica limitada, sendo esse o principal motivo que levou o grupo de pesquisa CIERMag, que estuda e desenvolve esse segmento da área de RM, a decidir desenvolver desde 2012 o seu próprio espectrômetro.

O Espectrômetro (5) foi desenvolvido utilizando as *Field Programmable Gate Arrays* (FPGA) as quais constituem-se de elementos de hardware que podem ser rearranjados de modo a se produzir sistemas mais complexos, processo denominado de *síntese de hardware*. O hardware sintetizado é descrito através de uma linguagem chamada VHDL, VHSIC (*Very High Speed Integrated Circuits*) *Hardware Description Language*.

As FPGAs possuem uma importante característica que é a possibilidade de serem rearranjadas várias vezes, de modo que a mesma FPGA possa sintetizar diversas configurações de hardware. Essa característica foi fundamental para a escolha do espectrômetro do CIERMag, pois permite que um mesmo equipamento possa ser reprogramado, por exemplo, de um sistema de Imagens para um sistema de Espectroscopia Analítica. Outra possibilidade é que, à medida que novas necessidades de hardware apareçam, o mesmo equipamento possa ser melhorado sem a necessidade de sua substituição física.

Esses pontos se tornam muito atraentes para pesquisa de novas sequências de pulsos, pois garantem flexibilidade e aprimoramentos do sistema conforme a demanda.

O sistema composto pelo Espectrômetro (6) engloba também outras ferramentas fundamentais como o Console (7) (sistema onde o usuário opera o equipamento e conduz os experimentos) o IDE (8) (sistema para desenvolvimento e configuração do sistema), Linguagem F (linguagem desenvolvida especificamente para a elaboração de sequências de pulsos) e Editor de Sequências Gráfico (editor que permite a elaboração de sequências de pulsos por meio gráfico).

2.3 Raspberry Pi

Raspberry Pi (9) é um microcomputador de baixíssimo custo (35 dólares) com dimensões aproximadas de um cartão de crédito, possuindo uma série de características que o tornam bastante atrativo e versátil.

Figura 1 - Raspberry Pi modelo B.
Fonte: RASPBERRY... (10)

O modelo utilizado neste trabalho foi o Raspberry Pi modelo B (11), mostrado na Figura 1, cujo processador é um *System on a Chip* (SoC) Broadcom BCM2835 de arquitetura

ARM (*Advanced RISC Machine*), que possui CPU e GPU integrados. O processador possui arquitetura ARM11 operando a 700MHz e memória RAM montada sobre o processador de 512 MB. A GPU possui conector de vídeo HDMI e conector padrão de vídeo componente RCA. Possui ainda conector de áudio padrão de 3,5mm, duas portas USB 2.0, conector RJ45 para conexão Ethernet, leitor de cartão de memória SD, 26 pinos de entrada e saída de uso geral (*General Purpose Input and Output – GPIO*), conector para câmera de vídeo CSI (*Camera Serial Interface*) e conector para display DSI (*Display Serial Interface*).

Para utilizar o Raspberry Pi basta carregar em um cartão SD o sistema operacional Raspbian, uma versão do Debian (Linux) otimizada. O Raspbian fornece acesso simples e rápido a todas as funcionalidades do sistema e ainda possibilita a utilização da linguagem de programação Python.

2.4 Python

Python (12) é uma linguagem de programação de alto nível, interpretada, orientada a objetos, de propósito geral que vem sendo muito utilizada. Seu código favorece a legibilidade por ter uma sintaxe clara e concisa. Apesar de ser interpretada, Python permite que trechos de código em C sejam executados aumentando o desempenho em regiões críticas do código.

Outra grande vantagem da linguagem é a modularidade do código que permite que módulos sejam utilizados, podendo ser módulos padrões, módulos desenvolvidos pelo próprio usuário ou módulos desenvolvidos por terceiros.

Os módulos padrões possuem a maior parte dos tipos e funções básicas para programar em Python. A linguagem oferece os tipos básicos como números inteiros, de ponto flutuante, números complexos elementos booleanos, e estruturas mais complexas como enumeradores, listas, *strings*, e dicionários, e operações utilizando esses tipos.

2.4.1 Módulos Padrão Utilizados

Os módulos padrão de Python estão presentes desde o processo de instalação, bastando apenas serem referenciados, através do comando *import*, no programa para serem

utilizados. O comando *import* permite que um módulo seja utilizado em sua totalidade, ou apenas algumas funções ou classes, permitindo certo grau de otimização.

Para a realização deste trabalho, foram utilizados os seguintes módulos padrão:

- Módulo *json* (13): Este módulo converte os tipos básicos de Python para o objeto *JavaScript Object Notation* (JSON). Este objeto é utilizado para troca de informações entre diferentes plataformas e diferentes linguagens de programação, com a característica de ser legível para humanos, definido pelo padrão Standard ECMA-262 3rd Edition - December 1999. O módulo também permite a conversão de/para um objeto do tipo *string* ou diretamente de/para um arquivo.

- Módulo *threading* (14): este é um módulo de alto nível, possuindo métodos simples para a construção de diferentes threads de modo que é possível executar várias tarefas em paralelo principalmente as que envolvem tarefas de entrada/saída de dados. Ele conta com travas e semáforos para organizar o acesso a arquivos e variáveis em comum.

- Módulo *time* (15): este módulo permite a utilização de funções relacionadas ao tempo e data, contados a partir da *Unix Epoch*, incluindo função para interromper, momentaneamente, a execução de uma *thread*.

- Módulo *datetime* (16): este módulo possui funções para a operação de tempo e data, como adição e subtração, bem como de visualização e customização. É a base para formatação dos eixos dos gráficos do módulo *bokeh*.

- Módulo *httplib* (17): este módulo possui funções básicas para operar, do lado do cliente, as comunicações no protocolo HTTP.

- Módulo *copy* (18): como em Python é utilizado a passagem por referência, este módulo é utilizado quando se deseja criar uma cópia de um objeto, como listas, dicionários e classes.

- Módulo *os* (19): este módulo possui funções para utilizar funções do Sistema Operacional.

- Módulo *os.path* (20): este módulo possui funções para manipular os caminhos dos arquivos.

2.4.2 Módulos de Terceiros Utilizados

A natureza modular da linguagem Python, permite facilmente que módulos sejam utilizados, criados e compartilhados. Muitos programadores criam seus próprios módulos e compartilham com a comunidade, e alguns deles são financiados pela iniciativa privada.

Os módulos de terceiros utilizados nesse trabalho foram:

- Módulo *spidev* (21): este módulo é bastante específico para o Raspberry Pi, permitindo que, em Python, se utilize o protocolo SPI para a comunicação entre dispositivos como sensores, atuadores e memórias.

- Módulo *gpio* (22): este módulo é específico para o Raspberry Pi, permitindo que, em Python, as portas de entrada e saída do processador sejam utilizadas, seja lendo ou escrevendo valores nelas.

- Módulo *pyqtgraph* (23): este módulo possui um conjunto de objetos e funções para criar gráficos e ferramentas de interface gráfica similares, voltado especificamente para a área científica. É construído sobre o PyQt4/PySide e o NumPy para ser bastante eficiente nas construções e visualizações de gráficos e imagens.

- Módulo *flask* (24): este módulo é um framework para desenvolvimento de servidores HTTP.

- Módulo *bokeh* (25): este módulo permite a criação de visualizações interativas voltadas para navegadores WEB.

2.5 Hypertext Transfer Protocol - HTTP

Hypertext Transfer Protocol (HTTP) (26-27) é um protocolo que define uma camada de aplicação de um sistema. Ele define a organização das comunicações em cliente e servidor, de modo que o servidor possui as informações e aguarda as solicitações de conteúdo do cliente.

As informações trocadas entre o cliente e o servidor são do tipo mensagens de texto que possuem três partes fundamentais: a requisição/resposta, o cabeçalho e o corpo.

A requisição, que contém o método, é a parte da mensagem que define como será a comunicação, sendo um comando verbal de modo a facilitar a leitura humana da mensagem. Os métodos definidos são

- OPTIONS: solicita ao servidor a relação das opções disponíveis na cadeia de requisição/resposta.

- GET: empregado quando o cliente faz uma requisição de conteúdo para o servidor.

- HEAD: possui a mesma função que o GET, porém, o servidor não retorna o conteúdo, ou seja, a mensagem contém apenas a resposta e o cabeçalho.

- POST: método utilizado para que o cliente envie conteúdos ao servidor, como formulários por exemplo.

- PUT: utilizado quando o cliente substituirá ou atualizará algum conteúdo já existente no servidor.

- DELETE: o cliente sinaliza qual o conteúdo a ser removido do servidor.

- TRACE: requisição para criar um retorno de modo que se possa monitorar todo o endereço ao conteúdo.

- CONNECT: utilizado para estabelecer um túnel em um servidor proxy.

Para as respostas, são definidos códigos de status com três dígitos, juntamente com uma frase que descreve brevemente o significado. Os códigos são agrupados de acordo com a natureza da resposta designada pelo primeiro algarismo, como se segue:

- 1xx: designa situações de informação, em que a requisição foi recebida e que será dado prosseguimento.

- 2xx: designa situações de sucesso, em que a requisição foi realizada corretamente.

- 3xx: designa situações de redirecionamento, em que mais alguma ação é necessária para prosseguir com a requisição.

- 4xx: designa erros do cliente, em que a requisição teve erro de sintaxe e não pôde ser concluída.

- 5xx: designa erros do servidor, em que o servidor encontrou problemas com uma requisição aparentemente válida.

Os cabeçalhos são campos com um nome separado de um valor com dois-pontos (:), e possuem informações de naturezas variadas de acordo com o seu tipo. Alguns campos são obrigatórios e outros são opcionais, porém isso pode variar de acordo com a requisição/resposta. Os tipos de cabeçalho são:

- Cabeçalho geral: são campos que podem ser usados tanto nas requisições quanto nas respostas.

- Cabeçalho de requisição: são campos que contém informações sobre o cliente e detalhes da requisição.

- Cabeçalho de resposta: são campos que contém informações sobre o servidor e informações adicionais sobre a resposta.

- Cabeçalho de entidade (corpo): são campos que contém informações sobre o conteúdo (corpo) da mensagem.

O corpo da mensagem é o conteúdo requisitado pelo cliente entregue ao servidor. Ele pode ser de vários tipos, devendo ser preenchido o campo no cabeçalho, como, por exemplo, uma página HTML ou um arquivo JSON.

2.6 Hiper Text Markup Language - HTML

HTML é uma linguagem utilizada para descrever páginas da web, definida na ISO/IEC 15445:2000 e mantida pelo World Wide Web Consortium (W3C). A programação é realizada utilizando “tags” que definem os comandos de inserção e formatação para textos, tabelas, fotos, etc.

As *tags* são compostas de um nome entre *chevrons* (< >) formando um elemento. Por exemplo, para se definir um parágrafo, deve se usar a *tag* de abertura <p>, seguido pelo conteúdo do parágrafo, finalizado pela *tag* de fechamento </p>. Porém alguns elementos não necessitam de *tags* de início e fim, como por exemplo, a *tag* <input> que define um elemento de entrada de dados, geralmente um formulário. Elas possuem atributos que modificam suas funcionalidades ou a aparência do conteúdo. A *tag* <input>, por exemplo, possui atributos que modificam o método de entrada, se será uma caixa de texto, uma lista drop-down ou um campo de texto. Já a *tag* <p> possui atributos que modificam a cor da fonte, tamanho, alinhamento e outros.

Um documento HTML possui, na primeira linha, as informações sobre a versão, seguido pela *tag* <html>, todo o conteúdo do documento, finalizado pela *tag* </html>. Esta *tag* deve possuir outros dois elementos, o head e o body, com suas respectivas *tags*.

A *tag* head não é visível e contém informações sobre o documento, como o título para ser exibido no topo da janela.

A *tag* body é a única parte visível do documento e possui os elementos, com suas respectivas *tags*, da página web.

Para se criar uma tabela, é necessário utilizar as *tags* <table> e </table>, em seguida é necessário acrescentar linhas à tabela utilizando as *tags* <tr> e </tr> e por fim é necessário criar divisões nas linhas utilizando as *tags* <td> e </td>.

Um elemento muito útil é o formulário já que através dele informações podem ser passadas para o servidor, como login. Para isso, utiliza-se a *tag* `<form></form>` e para se criar os campos a serem preenchidos, utiliza-se a *tag* `<input>` juntamente com a modificação de seus atributos para ser tipos diferentes de métodos de entrada. Por último utiliza-se uma *tag* `<input>` do tipo submit para que ele se torne um botão que envie o formulário ao servidor.

Outros elementos podem ser inseridos na página como, por exemplo, divisões (`<div></div>`), parágrafos (`<p></p>`), cabeçalhos de título (`<h1></h1>` até `<h6></h6>`), imagens (`<img>`) e até outras páginas de web através do inline frame (`<iframe></iframe>`).

2.7 Cascading Style Sheets - CSS

Através dos atributos nas *tags* é possível modificar a aparência dos conteúdos de uma página web. Contudo, deseja-se que vários trechos da página possuam a mesma formatação, ou então a mesma formatação seja utilizada em páginas diferentes. Para que não haja a necessidade de se copiar em cada trecho a modificação dos atributos, pode-se utilizar um documento que contém todas as formatações que é o documento CSS.

Para que na página web sejam aplicadas as formatações contidas no documento CSS, faz-se necessário criar um vínculo entre eles, que é realizado utilizando a *tag* `<link>` apontando para o documento CSS na seção `<head>` do documento HTML.

A partir da criação desse vínculo, cada *tag* utilizada deve ser listada no documento CSS juntamente com todos os atributos e os seus valores que serão modificados. Em algumas situações, deseja-se alterar a formatação de apenas um ou um grupo de elementos sem modificar a formatação dos demais. Para isso, é possível lançar mão dos atributos de classe e de identificação das *tags*. Diferenciando-se um elemento através do atributo `id` na *tag*, esse mesmo elemento é referenciado no documento CSS com o símbolo cerquilha (`#`) precedendo o valor atribuído no atributo, utilizando-se o atributo `class`, o elemento é referenciado com o nome da *tag* seguido pelo ponto final, seguido pelo nome da classe.

2.8 Jinja

Jinja é um gerador de templates de HTML para Python utilizado pelo módulo Flask para renderizar as páginas de web. Através de uma sintaxe própria bastante versátil que se soma ao conteúdo HTML, é possível gerar páginas dinâmicas e interativas.

O módulo de Python Flask possui a função `render_template`, que recebe como entrada um documento template HTML para o servidor gerar a página. Esse template pode possuir apenas código HTML ou ter também sintaxe Jinja, que em associação a um dicionário de variáveis passado para esta mesma função, permite que os valores sejam inseridos na página e também manipulados. O resultado desta função é uma página com HTML puro que pode ser exibido por qualquer navegador.

A sintaxe do template de Jinja contém variáveis e expressões que são substituídas após a renderização. É possível, por exemplo, passar uma lista de objetos e criar na página HTML uma tabela, iterando-se sobre a lista, de modo que cada linha contenha um objeto. Através do uso de expressões condicionais, é possível escolher quais elementos serão exibidos, tornando a criação da página dinâmica em função do seu conteúdo.

2.9 Modelo ISO/OSI e Modelo Internet

O Sistema de Padronização Internacional (International Standard Organization – ISO) estabeleceu o modelo conceitual de interconexões de sistemas abertos (Open System Interconnection – OSI) [ISO/IEC 7498-1] como uma referência para que sistemas diversos possam ter maior interoperabilidade e haja padronização de protocolos.

Esse modelo define sete camadas (Figura 2) de um sistema de telecomunicação ou sistema computacional, levando em conta a premissa que uma camada fornece informações para acima dela e recebe informações da camada abaixo dela. Isso garante a transparência entre as camadas, de modo que a alteração de protocolos em alguma camada não implique em alterações significativas das demais.

Número	Camada
7	Aplicação
6	Apresentação
5	Sessão
4	Transporte
3	Rede
2	Enlace
1	Física

Figura 2 - Modelo de camadas do OSI
Fonte: Elaborada pelo autor

A camada mais abaixo, que é a camada física, é a responsável por padronizar as características físicas da transmissão de dados, que variam de acordo com o meio, como níveis de tensão, frequência, tamanho do duto, modos de transmissão e outras características do meio. Exemplos de protocolos nessa camada: RS-232, Digital Subscriber Line (DSL) e Ethernet *physical layer*.

Acima da camada física, encontra-se a camada de enlace, que define o protocolo de acesso à camada física, início e fim de transmissão entre as conexões físicas e correção de erros e sincronização. Exemplos de protocolos: IEEE 802.11 (Wi-Fi), IEEE 802.3 (Ethernet) e Asynchronous Transfer Mode (ATM).

Na terceira camada, encontra-se a camada de rede, em que há a conversão do endereço lógico em endereço físico. Um nó pode enviar o endereço do destinatário e o conteúdo da mensagem para que ela chegue ao destinatário, e todo procedimento de como isso procede (divisão da mensagem, encaminhamento para outros nós, detecção ou não de erros, etc) é definido no protocolo da camada. Exemplos de protocolos: IPv4, IPv6 e Datagram Delivery Protocol (DDP).

A camada número quatro é a camada de transporte, que tem por finalidade garantir o transporte dos dados através das redes, definindo quais serão as formas de envio, recuperação e correção dos dados. Exemplos de protocolos: TCP e UDP.

Acima da camada de transporte, está a camada de sessão que é responsável por gerenciar a troca de mensagens, requisições e respostas, em uma conexão. Exemplos de protocolos: HTTP, FTP e SSH.

A sexta camada é a camada de apresentação, que estabelece o contexto da aplicação, normalmente também realizando a criptografia, compressão e conversão dos dados da aplicação. Exemplos de protocolos: HTML, XML e JSON.

Na última camada, a de aplicação, é onde o usuário interage, recebendo e enviando dados, existindo um número gigantesco de softwares que se encaixam nesta camada.

Muitos dos protocolos e aplicações existentes, não seguem estritamente as camadas definidas, já que este é um modelo conceitual, e muitas vezes englobam duas ou mais camadas, tornando assim, difícil a sua classificação, porém não inutilizando seus usos e funções.

O Modelo Internet (28) apresenta as características descritas no parágrafo anterior, pois seu modelo define apenas 5 camadas, que estão apresentadas na Figura 3, e é um modelo amplamente utilizado. Enquanto o Modelo OSI foca na confiabilidade das trocas de informação entre as camadas e transparência entre elas, o Modelo Internet foca na inteligência

do dispositivo conectado o qual atua em quase todos os protocolos, enquanto o sistema apenas envia os pacotes entre os diversos dispositivos conectados.

Número	Camada
5	Aplicação
4	Transporte
3	Internet
2	Interface de Rede
1	Hardware

Figura 3 - Camadas do Modelo Internet.
Fonte: Elaborada pelo autor

Na camada de aplicação estão localizados os programas que interagem com o usuário e acessam os serviços disponíveis na camada de transporte escolhendo, inclusive o tipo de transporte de dados.

Abaixo da camada de aplicação encontra-se a camada de transporte que tem por objetivo principal fornecer a comunicação entre os programas e controlar o fluxo de dados entre eles.

A terceira camada é a de internet que gerencia a comunicação entre os dispositivos conectados, enviando e recebendo os pacotes de dados encapsulados no datagrama IP.

A camada de interface de rede é responsável pela distribuição dos pacotes de dados entre as diferentes redes do sistema, que pode ser uma rede local ou redes complexas que possuem protocolos próprios.

Abaixo de todas as camadas está a camada de hardware que compõe os dispositivos que se conectam fisicamente para formar a rede, podendo possuir protocolos próprios para controlar e gerenciar os dados transportados.

Deste modo, o modelo conceitual OSI é importante para entender a importância tanto da divisão de camadas quanto da transparência entre elas, sendo utilizado como fonte de inspiração para a troca de informação entre diferentes partes internas do sistema, enquanto o modelo Internet é o modelo usado para a comunicação de fato entre o servidor e os clientes.

2.10 Escolha do protocolo de comunicação entre dispositivos

Existem muitos protocolos existentes para a comunicação entre dispositivos, mas a maioria deles possuem aplicações específicas, como o Universal Serial Bus (USB) que é

voltado para a comunicação de periféricos em computadores, o Serial AT Attachment (SATA) que é voltado para a conexão de dispositivos de armazenamento e o Peripheral Component Interconnect Express (PCIe) que é voltado para a expansão de dispositivos à placa-mãe como placas de vídeo e placas de rede.

Ao analisar os dispositivos que seriam necessários no sistema de periféricos identificou-se uma série de sensores e atuadores que, quando analisados com relação às suas interfaces de acesso presentes no mercado, foram identificados 3 protocolos mais utilizados: SPI, I²C (Inter Integrated Circuit) e One-Wire.

O protocolo SPI é do tipo síncrono, ou seja, existe um *clock* que regula a entrada e saída de dados, possui arquitetura do tipo mestre-escravo, na qual o mestre é quem inicia e realiza as requisições enquanto o escravo responde aos comandos do mestre e especifica 4 dutos (fios) para a comunicação dos dispositivos envolvidos: um de *clock* (Serial Clock - SCK), um de saída de dados do mestre e entrada de dados para o escravo (Master Output Slave Input – MOSI), um de entrada de dados para o mestre e saída de dados do escravo (Master Input Slave Output – MISO) e um para a escolha do escravo (Slave Select – SS ou Select Chip – SC), como pode ser visto na Figura 4.

Figura 4 - Arquitetura do protocolo SPI.

Fonte: Elaborada pelo autor

A comunicação SPI (mostrada na Figura 5) inicia-se com o mestre levando o duto SS de nível alto para baixo e, antes disso, toda modificação nos demais dutos é ignorada pelo escravo. Em seguida o mestre modifica o estado do duto MOSI e alta impedância para o nível de acordo com o dado que será enviado ao escravo e o duto de SCK para o estado de espera. Então a transmissão de dados se inicia com o mestre modificando o estado do clock e também

o nível no duto MOSI de acordo com a informação a ser transmitida. O mais comum é que o mestre envie um byte (8 bits) para o escravo fazendo um pequeno intervalo para que o escravo possa estar preparado para o byte seguinte, se for um procedimento de escrita de dados, ou o escravo prepare os dados solicitados pelo mestre para serem enviados no duto MISO se for um procedimento de leitura de dados. Para finalizar a transmissão, o mestre eleva o nível do duto SS.

Figura 5 - Diagrama de pulsos para o protocolo SPI
Fonte: SERIAL... (29)

A Figura 5 mostra também a diferença nos pulsos de acordo com a polarização do *clock* e a fase. Estas configurações são pré-definidas pelo fabricante nos dispositivos escravos, portanto o mestre deve alternar entre esses modos, utilizando registradores para alterações no hardware do mestre. A polarização do *clock* define o nível de espera: modo de polarização 0 indica nível de espera baixo e modo de polarização 1 indica nível de espera alto. Já a fase define se o dado deve estar estável na primeira transição do *clock* (modo 0) ou na segunda transição do *clock* (modo 1).

O protocolo SPI não define uma frequência de *clock*, sendo o único limitante as frequências máximas dos dispositivos envolvidos e também não há um número de escravos máximos permitidos ficando limitado à disponibilidade de pinos para o controle do mestre, já que é possível multiplexar outros pinos e gerar um número maior de seleção.

O protocolo I²C foi desenvolvido pela Phillips que é a proprietária, cobrando pela compra de endereços de escravos. É síncrono e possui arquitetura mestre-escravo com suporte a múltiplos mestres em um mesmo sistema, e utiliza apenas dois dutos para a comunicação, o de dados SDA e o *clock* SCL, como mostra a Figura 6.

A comunicação inicia-se com o mestre realizando a condição de início que é, estando SDA e SCL em nível alto, abaixar o nível do SDA permanecendo SCL em nível alto. Deste modo todos os escravos passam a receber os dados, aguardando o primeiro byte que indica o endereço do escravo, se a operação é de leitura e escrita, seguido por um nono pulso de *clock* em que o escravo deve responder o acknowledged (ACK) com o nível baixo no SDA. Em seguida o mestre continua enviando bytes seguidos pelo ACK do escravo ou o mestre recebe bytes do escravo também seguidos pelo ACK. Por fim, para terminar a comunicação, o mestre realiza a condição de fim, elevando o nível do duto SDA enquanto o duto SCL permanece em nível alto. A Figura 7 ilustra a comunicação do protocolo I²C.

Figura 6 - Arquitetura do protocolo I2C
Fonte: Elaborada pelo autor

Figura 7 - Diagrama de pulsos para o protocolo I²C
Fonte: MAYANK (30)

Para o protocolo I²C existem frequências de *clock*, definidas conforme a versão, que são 100 kHz, 400 kHz, 3,4 MHz e 5 MHz, sendo necessário observar a versão do protocolo suportado pelo escravo. Há também um modo de endereçamento de 10 bits para suprir a necessidade do grande número de dispositivos no mercado, e é comum os escravos possuírem

3 pinos externos para configurar o endereço do escravo, aumentando o número de um mesmo dispositivo conectados.

O protocolo *One-Wire* utiliza apenas um duto para comunicação, muitas vezes inclusive para a alimentação. Seu funcionamento é baseado na relação mestre-escravo em pulsos bem definidos no tempo. Foi observado, contudo, que ele não se trata de um protocolo bem definido, pois existem várias implementações que variam muito conforme o fabricante do componente, tanto nos tempos dos pulsos, quanto na forma de acesso.

Analisando os três protocolos, percebe-se uma vantagem do SPI com relação aos outros dois: tanto o hardware quanto o software envolvidos são mais simples, os dutos são unidirecionais permitindo o uso de isoladores de modo a ser possível utilizar outro tipo de camada física, é possível aplicar multiplexadores para aumentar a capacidade de dispositivos e não haver limites de frequência para o *clock*. Desse modo este protocolo se torna mais atrativo para ser empregado no sistema desenvolvido neste trabalho.

2.11 IEEE 1451 e Transducer Electronic Data Sheet (TEDS)

O Instituto de Engenheiros Eletricistas e Eletrônicos estabelece um conjunto de padrões denominado IEEE 1451 (31) para os chamados Smart Transducers, os transdutores inteligentes. Transdutores são dispositivos que convertem energia de um domínio a outro, podendo ser sensores ou atuadores. Segundo o padrão (31) transdutores inteligentes são aqueles transdutores que fornecem funções além daquelas necessárias para obter corretas representações de uma quantidade medida ou controlada, sendo o objetivo dessas funções facilitar a integração do transdutor à aplicação em uma rede.

O padrão visa definir vários aspectos necessários aos transdutores inteligentes, e a Figura 8 auxilia a visualização da interconexão entre os padrões:

- IEEE 1451.0: Define funções e comandos dos Transducer Electronic Data Sheet (TEDS), que são os *datasheets* virtuais disponíveis de cada transdutor que podem ser acessados pela rede em que o transdutor está conectado e fornece todas as informações necessárias para a sua operação.

- IEEE 1451.1: Define o Network Capable Application Processor (NCAP), o elemento que realiza a conexão do transdutor à rede e intermedia as funções e comandos direcionados ao transdutor, podendo ser modularizado e separado do transdutor.

- IEEE 1451.2: Define o Smart Transducer Interface Module (STIM) que é a interface de conexão entre o NCAP e o transdutor de modo a acessar as funcionalidades do transdutor.

- IEEE 1451.3: Define o Transducer Bus Interface Module (TBIM) que é o módulo que contém o duto de interface, condicionamento de sinal, conversão digital-analógica (DA) ou analógica-digital (AD) e pode ou não incluir o transdutor. Caso não inclua o transdutor ele pode se conectar com um ou mais transdutores. Também é definido o Transducer Bus Controller (TCB) que é o elemento que fornece comunicação entre o STIM com um ou mais TBIM.

- IEEE 1451.4: Define a Mixed Mode Interface (MMI) que é uma interface mestre-escravo serial que pode detectar a conexão dos elementos transdutores e separar as comunicações digitais das analógicas.

Figura 8 - Interconexão entre os padrões IEEE 1451

Fonte: Elaborada pelo autor

Os transdutores encontrados no mercado, atualmente, não são smart transducers, pois eles não são padronizados segundo o IEEE 1451, não possuem acesso a uma rede e cada fabricante desenvolve os métodos de acesso ao transdutor, ainda que muitos possuam como funções integradas o acondicionamento de sinal, conversão AD/DA e detecção de falhas. Tendo em vista esses transdutores e que o sistema a ser desenvolvido deverá operar com eles, a dificuldade surge em como ser possível utilizá-los diante de uma variedade de transdutores, métodos de acesso e fabricantes.

Nesse sentido, o padrão IEEE 1451.0 oferece uma saída: utilizando um sistema de datasheets padrão para cada transdutor, o sistema pode identificá-los, conhecer suas capacidades e funcionalidades. Por isso será analisado mais atentamente apenas o padrão IEEE1451.0, mais especificamente os Transducer Electronic Data Sheet (TEDS), já que também são definidos outros elementos como comandos de acesso, estruturas de mensagem, funcionalidades e interfaces de programação de aplicações.

Foram definidos dois tipos de TEDS, os obrigatórios e os opcionais, e subdivididos de acordo com o contexto das informações que possuem. Os TEDS obrigatórios devem estar presentes em todos os transdutores e possuem informações básicas dos transdutores, já os opcionais possuem informações mais específicas voltados a alguns tipos de transdutores.

TEDS obrigatórios:

- Meta-TEDS: fornece informações sobre os TransducerChannel existentes na TIM, juntamente com alguns parâmetros de tempo que são usados pelo NCAP para determinar quando o TIM não está respondendo.

- TransducerChannel TEDS: fornece informações detalhadas sobre o transdutor, como grandezas físicas que podem ser controladas ou medidas, limites de operação, entre outros.

- User's Transducer Name TEDS: fornece informações definidas pelo usuário de modo que o transdutor possa ser reconhecido na rede em que estiver conectado.

- PHY TEDS: suas informações dependem do meio de comunicação existente entre o TIM e o NCAP, e são definidos em outros padrões IEEE 1451.

TEDS opcionais:

- Calibration TEDS: possui constantes necessárias para converter a saída de sensores em uma grandeza física ou converter uma grandeza física em valores requisitados para atuadores.

- Frequency Response TEDS: possui uma tabela para armazenar a resposta em frequência do transdutor.

- Transfer Function TEDS: possui parâmetros para se obter a função de transferência do transdutor.

- Text-based TEDS: é uma família de TEDS que fornecem informações no formato de texto sobre o TIM ou o TransducerChannel.

- Commands TEDS: possui informações no formato de texto para que fabricantes acrescentem comandos para configuração além dos previstos pelo padrão.

- Identification TEDS: possui informações no formato de texto de outras partes do transdutor, que está definido em outros padrões IEEE 1451.

- Geographic location TEDS: possui informações definidas pelo usuário, no formato de texto, que descrevem o lugar geográfico em que o transdutor foi instalado.

- Units extension TEDS: possui informações no formato de texto que servem para os casos em que a unidade do transdutor não pode ser escrito em grandezas físicas do Sistema Internacional de Unidades (SI).

- End User Application Specific TEDS: é reservado para ser uma área de memória para que o usuário possa colocar qualquer tipo de informação em qualquer tipo de formato conforme necessário.

- Manufacturer defined TEDS: é reservado para o uso dos fabricantes definirem informações que não estão definidas no padrão, deixando todo o conteúdo, formato e possibilidade de acesso a critério deles.

Os TEDS são descritos como uma tabela de informações que deve estar contida na memória de cada transdutor para ser acessada quando solicitada pelo sistema. O formato dessa tabela é definido como na Tabela 1. Na primeira coluna, o campo *field*, é o local que consta qual é o código de identificação da informação. A segunda coluna é apenas uma descrição para facilitar a identificação do código presente na primeira coluna e não faz parte do TEDS. A terceira coluna identifica o formato do dado, podendo ser tipos inteiros sem sinal de 8 bits, ponto flutuante de 32 bits entre outros. A quarta coluna define o tamanho da informação, baseada no número de octetos que é necessário. A estrutura dos dados do TEDS segue o formato: tipo, comprimento e valor (*type, length, value* – TLV). O tipo e o comprimento são definidos em 1 octeto, já o valor possui um número variável de octetos.

A primeira linha dos TEDS sempre consta o tamanho total do TEDS e deve ser lido como um unsigned integer de 32 bits, possuindo 4 octetos contendo a informação. Nas linhas seguintes constam outros campos de identificação. Na última linha sempre consta o *checksum* para verificar se houve corrompimento nas informações, e é do tipo inteiro sem sinal de 16 bits com a informação ocupando dois octetos.

Tabela 1 - Formato geral do TEDS

Field	Description	Type	# octets
—	TEDS length	UInt32	4
1 to N	Data block	Variable	Variable
—	Checksum	UInt16	2

Fonte: INSTRUMENTATION ... (30)

A Tabela 2 mostra os campos necessários para o Meta-TEDS. É possível observar os campos 10, 11 e 12 que se referem aos tempos de acesso utilizado pelo NCAP e os demais campos que fornecem informações sobre o TransducerChannel.

Tabela 2 - Meta-TEDS

Field type	Field name	Description	Data type	# octets
—	—	Length	UInt32	4
0-2	—	Reserved	—	—
3	TEDSID	TEDS Identification Header	UInt8	4
4	UUID	Globally Unique Identifier	UUID	10
5-9	—	Reserved	—	—
Timing-related information				
10	OholdOff	Operational time-out	Float32	4
11	SHoldOff	Slow-access time-out	Float32	4
12	TestTime	Self-Test Time	Float32	4
Number of implemented TransducerChannels				
13	MaxChan	Number of implemented TransducerChannels	UInt16	2
14	CGroup	ControlGroup information sub-block	—	—
Types 20, and 21 define one ControlGroup.				
20	GrpType	ControlGroup type	UInt8	1
21	MemList	ControlGroup member list	array of UInt16	NTc
15	VGroup	VectorGroup information sub-block	—	—
Types 20 and 21 define one VectorGroup.				
20	GrpType	VectorGroup type	UInt8	1
21	MemList	VectorGroup member list	array of UInt16	NTv
16	GeoLoc	Specialized VectorGroup for geographic location	—	—
Types 24, 20, and 21 define one set of geographic location information.				
24	LocEnum	An enumeration defining how location information is provided	UInt8	1
20	GrpType	VectorGroup type	UInt8	1
21	MemList	VectorGroup member list	array of UInt16	NTv
17	Proxies	TransducerChannel proxy definition sub-block	—	—
Types 22, 23, and 21 define one TransducerChannel proxy.				
22	ChanNum	TransducerChannel number of the TransducerChannel proxy	UInt16	1
23	Organiz	TransducerChannel proxy data-set organization	UInt8	1
21	MemList	TransducerChannel proxy member list	array of UInt16	NTp
18-19	—	Reserved	—	—
25-127	—	Reserved	—	—
128-255	—	Open to manufacturers	—	—
—	—	Checksum	UInt16	2

Fonte: INSTRUMENTATION ... (30)

Já a Tabela 3 mostra o TransducerChannel TEDS, em que é possível observar os grupos de informação referentes ao próprio TransducerChannel: unidades físicas, conversão de dados, tempos de resposta e tempos de amostragem. Um exemplo simples do preenchimento do TEDS pode ser feito utilizando o campo com o código 11, chamado ChanType, que contém a informação do tipo do TransducerChannel conectado, estando no formato de inteiro sem sinal de 8 bits e possui 1 octeto de tamanho. O código 11 do campo é convertido em hexadecimal se torna 0x0B. Como ele possui 1 octeto de tamanho, convertido em hexadecimal, se torna 0x01. Se considerarmos um sensor, o código definido pelo padrão para isso é zero. Deste modo, este campo no TEDS ficaria da seguinte forma: 0B 01 00.

Tabela 3 - TransducerChannel TEDS

Field	Field name	Description	Type	# octets
—	—	TEDS length	UInt32	4
0–2	—	Reserved	—	—
3	TEDSID	TEDS Identification	UInt8	4
4–9	—	Reserved	—	—
TransducerChannel related information				
10	CalKey	Calibration key	UInt8	1
11	ChanType	TransducerChannel type key	UInt8	1
12	PhyUnits	Physical Units	UNITS	11
50	UnitType	Physical Units interpretation enumeration	UInt8	1
51	Radians	The exponent for Radians	UInt8	1
52	SterRad	The exponent for Steradians	UInt8	1
53	Meters	The exponent for Meters	UInt8	1
54	Kilogram	The exponent for Kilograms	UInt8	1
55	Seconds	The exponent for Seconds	UInt8	1
56	Amperes	The exponent for Amperes	UInt8	1
57	Kelvins	The exponent for Kelvins	UInt8	1
58	Moles	The exponent for Moles	UInt8	1
59	Candelas	The exponent for Candelas	UInt8	1
60	UnitsExt	TEDS access code for units extension	UInt8	1
13	LowLimit	Design operational lower range limit	Float32	4
14	HiLimit	Design operational upper range limit	Float32	4
15	OError	Worst-case uncertainty	Float32	4
16	SelfTest	Self-test key	UInt8	1
17	MRange	Multi-range capability	UInt8	1
—	—	Data converter-related information	—	—
18	Sample	—	—	—
40	DatModel	Data model	UInt8	1
41	ModLenth	Data model length	UInt8	1
42	SigBits	Model significant bits	UInt16	2
19	DataSet	—	—	—
43	Repeats	Maximum data repetitions	UInt16	2
44	SOrigin	Series origin	Float32	4
45	StepSize	Series increment	Float32	4
46	SUnits	Series units	UNITS	11
47	PreTrigg	Maximum pre-trigger samples	UInt16	2
Timing-related information				
20	UpdateT	TransducerChannel update time (t_u)	Float32	4
21	WSetupT	TransducerChannel write setup time (t_{ws})	Float32	4
22	RSetupT	TransducerChannel read setup time (t_{rs})	Float32	4
23	SPeriod	TransducerChannel sampling period (t_{sp})	Float32	4
24	WarmUpT	TransducerChannel warm-up time	Float32	4
25	RDelayT	TransducerChannel read delay time (t_{ch})	Float32	4
26	TestTime	TransducerChannel self-test time requirement	Float32	4
Time of the sample information				
27	TimeSrc	Source for the time of sample	UInt8	1
28	InPropDI	Incoming propagation delay through the data transport logic	Float32	4
29	OutPropD	Outgoing propagation delay through the data transport logic	Float32	4
30	TSError	Trigger-to-sample delay uncertainty	Float32	4
Attributes				
31	Sampling	Sampling attribute	—	—
48	SampMode	Sampling mode capability	UInt8	1
49	SDefault	Default sampling mode	UInt8	1
32	DataXmit	Data transmission attribute	UInt8	1
33	Buffered	Buffered attribute	UInt8	1
34	EndOfSet	End-of-data-set operation attribute	UInt8	1
35	EdgeRpt	Edge-to-report attribute	UInt8	1
36	ActHalt	Actuator-halt attribute	UInt8	1
Sensitivity				
37	Directon	Sensitivity direction	Float32	4
38	DAngles	Direction angles	Two Float32	8
Options				
39	ESOption	Event sensor options	UInt8	1
61–127	—	Reserved	—	—
128–255	—	Open to manufacturers	—	—
—	—	Checksum	UInt16	2

Fonte: INSTRUMENTATION ... (30)

É possível observar que o padrão IEEE 1451.0 que define os TEDS é bastante amplo e completo. Contudo, muitos dos campos presentes não são utilizados por todos os transdutores disponíveis no mercado, de modo que o preenchimento completo se torna difícil e trabalhoso. Por conta disso, foi decidido que seria criado um formato de TEDS para o sistema de periféricos do CIERMag inspirado no padrão IEEE, de modo que este novo TEDS seja simples e rápido de ser preenchido, mas que ao mesmo tempo pudesse ser genérico o suficiente para que o sistema pudesse operar com o transdutor corretamente. O desenvolvimento desse novo TEDS será descrito na seção 3.3 deste trabalho.

3 DESENVOLVIMENTO

3.1 Análise do sistema e seus requisitos

O Espectrômetro desenvolvido no CIERMag possui características que permite que ele possa ser modificado para se tornar, teoricamente, qualquer tipo de equipamento de Ressonância Magnética. Tendo isso em vista, o sistema que deverá controlar e monitorar os dispositivos periféricos deve possuir características para também se tornar modificável e integrar qualquer tipo de experimento de RM. Como o Espectrômetro é responsável pelo controle dos sinais necessários para a geração das sequências de pulsos, não faria sentido desenvolver um sistema que também possuísse esse tipo de tarefa. Por isso, ficou definido que o Espectrômetro seria o responsável por controlar todo o sistema de *real-time* que é intrinsicamente conectado à sequência de pulsos desejada e haveria um outro sistema que controlaria os dispositivos que fossem *non-real-time*. Em razão do exposto, o objetivo desse trabalho torna-se elaborar um subsistema de dispositivos periféricos *non-real-time*.

Observando-se os equipamentos de RM existentes e atentando-se para as partes específicas do equipamento, é possível relacionar módulos de acordo com a função que exercem no sistema e analisar seus respectivos requisitos de monitoramento ou controle. É importante ressaltar que os parâmetros podem não ser necessários a todos os experimentos e alguns módulos são mutuamente exclusivos, porém o objetivo é buscar o maior número de requisitos possível para que o desenvolvimento do subsistema de dispositivos periféricos possa abarcar a todos.

- Módulo de gerenciamento da amostra: este módulo está presente para os casos de espectroscopia e pode necessitar que a temperatura da amostra seja monitorada e controlada, que a velocidade da amostra seja monitorada e controlada e que se detecte a presença ou não da amostra.

- Módulo monitor de sinais vitais: este módulo está presente nos casos de experimentos com imagens em humanos e animais, e pode ser necessário monitorar o batimento cardíaco, a respiração e a temperatura do durante o experimento.

- Módulo de controle da cama: presente nos sistemas que possuem cama para posicionar o paciente corretamente no centro do magneto, pode ser necessário obter a posição que a cama ocupa/ocupará para os três eixos de movimento (X, Y e Z)

- Módulo de monitoramento do ambiente do magneto: pode ser necessário monitorar a abertura/fechamento da sala do magneto, monitorar a temperatura da sala, monitorar a umidade da sala e monitorar a ventilação da sala.

- Módulo de monitoramento do magneto: pode ser necessário monitorar o volume de líquido refrigerante (He ou N) no tanque, monitorar a temperatura da blindagem e monitorar a temperatura do *cryocooler*.

- Módulo de refrigeração: pode ser necessário monitorar a temperatura dos sistemas de refrigeração, monitorar a pressão do líquido refrigerante dos sistemas de refrigeração e monitorar a vazão do líquido refrigerante nos sistemas de refrigeração.

- Módulo de bobinas de gradiente: pode ser necessário monitorar a temperatura, monitorar a potência, monitorar a corrente e monitorar a tensão das bobinas de gradiente.

- Módulo de monitoramento do ambiente dos dispositivos eletrônicos: dependendo da complexidade do equipamento, amplificadores e demais dispositivos podem ser acondicionados em uma sala reservada e ser necessário monitorar a abertura/fechamento da porta, monitorar a temperatura, umidade e a ventilação da sala.

- Módulo de amplificação de radiofrequência: pode ser necessário monitorar e controlar o ganho, monitorar a potência transmitida, monitorar a potência refletida, monitorar a temperatura, monitorar a corrente e monitorar a tensão dos amplificadores.

- Módulo de transmissores e receptores: pode ser necessário monitorar e controlar o ganho, monitorar os canais existentes e controlar o roteamento de sinais tanto dos transmissores quanto dos receptores.

- Módulo de amplificação das bobinas de gradiente: pode ser necessário monitorar os canais existentes, monitorar a temperatura, monitorar a potência, monitorar a potência refletida, monitorar a corrente e monitorar a tensão dos amplificadores bobinas de gradiente.

A Tabela 4 mostra o resumo dos requisitos do sistema.

Tabela 4 - Tabela com os requisitos do sistema

Parameter	Transducer	Module										
		Sample	Vital Signs	Patient Table	Magnet Environment	Magnet	Cooling Systems	Gradient Coils	Electronic Environment	RF Amplifier	Transmitter and Reciever	Gradient Co Amplifier
Temperature	Sensor	X	X				X	X		X		
	Actuator	X			X				X			
Rotation Speed	Sensor	X										
	Actuator	X										
Presence	Sensor	X										
EKG	Sensor		X									
Respiration	Sensor		X									
Position	Sensor			X								
	Actuator			X								
Door Contact	Sensor				X				X			
Humidity	Sensor				X				X			
Fan Speed	Sensor				X				X			
	Actuator				X				X			
He/Ni Volume	Sensor					X						
Water Pressure	Sensor						X					
Water Flow	Sensor						X					
Electric Power	Sensor							X				X
Voltage	Sensor							X		X		X
Electric Current	Sensor							X		X		X
Gain	Sensor									X	X	
	Actuator									X		
Foward Power	Sensor									X		
	Actuator											
Reflected Power	Sensor									X		
Channels	Sensor										X	X
Signal Rooting	Actuator										X	

Fonte: Elaborada pelo autor

Através dos requisitos do sistema, é possível notar que basicamente o subsistema de dispositivos periféricos deve ser capaz de operar, essencialmente, com transdutores, obtendo os valores medidos pelos sensores e aplicar valores nos atuadores. Assim, se faz necessário determinar como esses transdutores estarão conectados, ou seja, a arquitetura do sistema e também é necessário desenvolver uma interface de acesso para o usuário acessar os transdutores, assuntos discutidos nas próximas seções.

3.2 Arquitetura do subsistema de periféricos

O sistema no qual o Espectrômetro do CIERMag está inserido, possui um terminal computadorizado no qual o usuário configura o sistema, programa a sequência de pulsos, recebe os resultados do experimento e manipula os resultados e está conectado, através do protocolo Ethernet, ao Espectrômetro que gera a temporização necessária, comanda os dispositivos e recebe os sinais gerados.

Deste modo, o subsistema deverá ser incluído na rede Ethernet, o que pode ser facilmente realizado utilizando-se um hub Ethernet que se encontra amplamente disponível comercialmente, para possibilitar a utilização do usuário através do terminal computadorizado, como mostrado na Figura 9. Nesta figura o subsistema foi dividido em duas partes, o gerenciador de transdutores e os transdutores.

Figura 9 - Arquitetura geral do sistema ao integrar o subsistema de periféricos.
Fonte: Elaborada pelo autor

A divisão de um sistema de RM em módulos, além de ter por objetivo facilitar o levantamento dos requisitos, também organiza a elaboração da arquitetura do subsistema, de modo que é possível incluir ou excluir cada módulo conforme a necessidade, além de não sobrecarregar o processamento de cada unidade centralizadora do módulo caso exista uma quantidade muito grande de transdutores conectados. Esta arquitetura é mostrada na Figura 10.

Figura 10 - Arquitetura do subsistema de periféricos dividida em módulos.

Fonte: Elaborada pelo autor

Para ser o elemento centralizado de cada módulo, pensou-se inicialmente em utilizar os microcontroladores existentes no mercado, por serem naturalmente os dispositivos que são conectados diretamente aos transdutores periféricos presentes no sistema. Porém, ao considerar o Raspberry Pi como uma possibilidade de ser o elemento centralizador, essa arquitetura foi revista. Por ser, praticamente, um microcomputador, o Raspberry Pi possuía mais poder de processamento, mais memória RAM, maior capacidade de armazenamento do que outros micro controladores e, além de ter um custo igual ou inferior, tem a possibilidade de ser integrado diretamente à rede Ethernet e de contar com os pinos de entrada e saída de dados possibilitando o uso de interface SPI empregado em diversos transdutores. Há ainda o fato de que, com o Raspberry Pi, é possível programá-lo em Python nativamente, aumentando as possibilidades que o sistema pode oferecer.

Por isso, foi determinado que o Raspberry Pi seria o elemento centralizador de todos os módulos, havendo apenas um que seria capaz de se comunicar com todos os transdutores,

através do protocolo SPI, e ainda com o terminal computadorizado, através do protocolo HTTP. E a arquitetura do sistema foi definida como a mostrada na Figura 11.

Figura 11 - Arquitetura final para o subsistema de periféricos.

Fonte: Elaborada pelo autor

Como já mostrado na seção 2.10, o protocolo SPI tem algumas vantagens, principalmente no que se refere à simplicidade de software e hardware necessários, e foi escolhido para ser o padrão de interface de comunicação com os transdutores.

Quando o protocolo SPI é analisado sob a perspectiva do modelo OSI apresentado na seção 2.9, percebe-se que ele é um protocolo de camada de enlace, pois determina a conexão lógica com os diferentes dispositivos, estabelece o início e fim da comunicação e define o método de sincronização. Isso é importante pois permite que sejam feitas alterações no nível da camada física necessárias ao correto funcionamento do subsistema.

Uma alteração necessária é a mudança física da forma que é feito o endereçamento dos transdutores. Originalmente, no protocolo SPI, cada escravo deve ser acionado a partir de um sinal de seleção diferente, como é possível observar na Figura 4, em que existem dois canais de seleção cada um acionando um escravo separadamente, o SC1 seleciona o *Slave1* e o SC2 seleciona o *Slave2*.

O Raspberry Pi possui apenas 2 sinais de seleção que podem ser gerados, inviabilizando a conexão de mais transdutores. Uma forma de contornar isso é utilizando

multiplexadores e outros pinos de entrada e saída (General Purpose Input Output - GPIO) de dados disponíveis para ampliar a capacidade de endereçamento. Existem 17 pinos GPIO, mostrados na Figura 12, sendo que destes, 5 são reservados para o uso do protocolo SPI definidos por hardware: o pino 19 é o MOSI, o pino 21 é o MISO, o pino 23 é o SCK, o pino 24 é o SS0 e o pino 26 é o SS1.

Figura 12 - Pinos de entrada e saída disponível no Raspberry Pi.
Fonte: GPIO... (32)

Utilizando-se os multiplexadores, que funcionam como roteadores de sinais, e os demais pinos de GPIO para o endereçamento nos multiplexadores, a capacidade de transdutores que podem ser conectados é ampliado de dois para até o máximo de 4096, quantidade mais do que satisfatória. Para se realizar essa a multiplexação dos sinais do Raspberry Pi, um circuito auxiliar é necessário, e está exibido na forma de diagrama na Figura 13. Os sinais MOSI, CLK e SS do Raspberry Pi são colocados na entrada de multiplexadores e cada saída é colocada, ordenadamente, nas entradas dos escravos. O sinal MISO é colocado em um de-multiplexador, que faz nada mais que realizar o procedimento inverso do multiplexador, de maneira que os sinais MISO dos escravos sejam colocados nas entradas do de-multiplexador e a saída deste é conectada ao Raspberry Pi. Os demais pinos de GPIO são colocados nos endereçadores dos multiplexadores e do de-multiplexador, de modo que quando uma determinada combinação lógica for utilizada, eles realizam a conexão lógica entre o Raspberry Pi e o escravo desejado.

Figura 13 - Diagrama do circuito para multiplexar os sinais do Raspberry Pi.
Fonte: Elaborada pelo autor

Outra alteração pode ser necessária de acordo com as distâncias envolvidas entre o mestre e o escravo. O protocolo SPI é utilizado para conectar dispositivos que se encontram na mesma placa. Contudo, dependendo de como for utilizado os escravos, por exemplo em uma placa separada, pode ser necessário utilizar não mais apenas um fio para enviar e receber os sinais por conta de interferência do meio e o próprio efeito de onda refletida. Nesses casos é necessário realizar a isolamento (galvânica ou óptica) dos transdutores e utilizar outra forma de transmissão como, por exemplo, pares trançados e, tanto a arquitetura quanto o protocolo SPI, não são afetados se essas medidas forem necessárias.

Foi proposta também uma arquitetura lógica para a interconexão dos módulos programados em Python, conforme mostra a Figura 14. O Cliente Python, integrado junto à Console, é executado no terminal computadorizado e se comunica com o Servidor localizado no Raspberry Pi. Alternativamente o Cliente Web pode ser executado no navegador de um outro computador para também se comunicar com o Servidor.

No Raspberry Pi, o Servidor gerencia os TEDS nos arquivos “.ted” e se comunica com o módulo Gerenciador de Transdutores, enviando os valores que devem ser programados nos atuadores e recebendo os valores que leitura dos sensores, através de arquivos específicos.

O Gerenciador de Transdutores, através da interface SPI, envia os bytes necessários para controlar apropriadamente os transdutores, criando e monitorando o ambiente do experimento de Ressonância Magnética.

Figura 14- Arquitetura lógica de conexão entre os diferentes módulos do subsistema.

Fonte: Elaborada pelo autor

Desse modo, as arquiteturas utilizadas, os protocolos e os dispositivos escolhidos, se integram de forma a formarem um conjunto adequado para fazerem parte do subsistema de periféricos.

3.3 Transducer Electronic Data Sheet do CIERMag

Como apresentado na seção 2.11, o padrão IEEE 1451 define o Transducer Electronic Data Sheet (TEDS) para descrever como gerenciar um transdutor inteligente de modo genérico. Esse conceito se torna bastante favorável para ser aplicado ao subsistema de periféricos, pois ele pode possuir um número grande de transdutores diferentes e de diferentes fabricantes, porém o modelo como foi apresentado pelo padrão é voltado para os transdutores inteligentes que não são os disponíveis no mercado, por isso o desenvolvimento de um TEDS próprio do sistema se faz necessário. O padrão IEEE servirá, portanto, de inspiração e os

campos serão próprios para a necessidade do subsistema de periféricos e sua implementação também será diferente.

O TEDS se contrapõe a outra filosofia de gerenciamento de transdutores que é a de elaboração de módulos ou bibliotecas de software. Este último se mostra eficiente para deixar disponível ao usuário todas as funcionalidades do transdutor, mas é inviável no ponto de vista prático, pois a cada inclusão de novo transdutor, é necessário que o desenvolvedor do sistema crie um novo módulo ou uma nova biblioteca e a integre no sistema, o que demanda muito tempo. Em contrapartida, o TEDS possui uma interface de entrada para a criação de novos transdutores e permite que o usuário possa também acessar todas as funcionalidades do transdutor que não são utilizadas com frequência, mas deixa em evidência apenas as informações necessárias, que para o caso dos atuadores é um método de envio de dados e para o caso dos sensores um método de recebimento de dados, considerando ainda que qualquer usuário, com o conhecimento básico de transdutores e leitura de *datasheets* pode criar e manipular os transdutores a qualquer momento no local em que o sistema estiver instalado. Dessa forma, ficam evidentes as vantagens da utilização do TEDS escolhido para este trabalho.

Ao analisar os transdutores existentes no mercado, observa-se que eles possuem diferentes parâmetros para o protocolo SPI, utilizando inclusive das variações existentes nele. Existem variações na velocidade de *clock* que cada transdutor exige, bem como as variações na polarização do *clock* e a fase.

Deste modo, foi criado um conjunto de campos básicos comuns a todos os transdutores, denominado cabeçalho, que possui informações de identificação do transdutor, e as configurações iniciais do SPI. A Figura 15 apresenta o cabeçalho do TEDS do CIERMag com os campos, ou parâmetros. A seguir, será explicado cada um dos campos na ordem em que aparecem na figura:

- Campo *Name*: este campo foi empregado para que o usuário possa colocar um nome que seja significativo para a identificação do transdutor.

- Campo *ID*: este campo está presente para ser a identificação única do transdutor no subsistema e é gerado automaticamente utilizando uma combinação da data e da hora do sistema em um dado tipo *string*.

- Campo *Active*: este campo é utilizado para ativar ou não o transdutor no sistema, ou seja, o sistema não fará nenhum envio ou recebimento de dados e nem estará visível para o usuário. Isto foi feito para que não haja a necessidade de se remover e criar o TEDS todas as vezes que o transdutor precise ou não ser utilizado.

- Campo *SPI Address*: este campo define o endereço do transdutor no sistema, cujo valor é passado para os multiplexadores de modo que eles realizem a conexão entre o Raspberry Pi e o transdutor.

- Campo *Clock Speed*: este campo deve conter a frequência do clock utilizado para a comunicação SPI que é encontrado no datasheet fornecido pelo fabricante do transdutor.

- Campo *Chip Select Idle High*: este campo apresenta como deve ser feita a seleção do transdutor, se é selecionado quando o CS (SS) estiver em nível baixo, ou quando estiver em nível alto, portanto o duto CS (SS) permanecerá inativo no nível oposto.

- Campo *Polarization/Phase Mode*: como mencionado na seção 2.10, o protocolo SPI pode variar com relação à fase e a polaridade do *clock*. A Figura 5 mostra os valores utilizados, normalmente nos registradores de configuração dos microcontroladores, para a polarização e para a fase, de modo que está convencionado utilizar dois bits para definir as 4 possibilidades de combinação entre elas, ou então o correspondente valor convertido para a base decimal.

- Campo *Data for Setup*: este campo foi empregado para que seja preparado o transdutor para utilização. É muito comum que os transdutores, por estarem bastante desenvolvidos tecnologicamente, possuam várias posições em seus registradores de memória internos para serem configurados e este campo permite que todas as funcionalidades do transdutor possam ser configuradas e preparadas para uso.

Parameter	Value
Name	
ID	
Active	
SPI Address	
Clock Speed	
Chip Select Idle High	
Clock Polarization/Phase Mode	
Data for Setup	

Figura 15 - Cabeçalho do TEDS do CIERMag.

Fonte: Elaborada pelo autor

Com este cabeçalho no TEDS, o subsistema pode identificar, operar e configurar corretamente qualquer transdutor. Porém resta uma dificuldade a ser superada que é permitir que qualquer transdutor possa ser utilizado, sendo que variam tipos e quantidade de funcionalidades. A elaboração de um novo conjunto de campos, mostrado pela Figura 16 denominado registrador de função, supera esta dificuldade.

Os transdutores possuem as informações de valores de medições, para o caso de sensores, e valores de atuação, para o caso dos atuadores, em registradores específicos em suas memórias internas. O registrador de função é responsável por descrever ao subsistema como acessar esses registradores, ler ou escrever valores neles e realizar a conversão dos dados lidos ou escritos. Abaixo, segue a descrição e função de cada:

- Campo *Active Register*: similar ao campo *Active*, ele é utilizado para que o registrador seja ignorado pelo subsistema, e não seja possível acessá-lo nem enviar ou receber dados.

- Campo *SPI Access Bytes*: este campo é o que fornece as informações de quais dados devem ser enviados para que o registrador seja acessado.

- Campo *SI Unit*: este campo possui qual a unidade do SI que o transdutor converte, porém é possível inserir qualquer tipo de unidade.

- Campo *Minimum Value*: este campo informa ao subsistema qual o menor valor que o transdutor irá operar, não precisando ser, necessariamente, o menor valor que ele é capaz de operar.

- Campo *Maximum Value*: este campo informa ao subsistema qual o maior valor que o transdutor irá operar, não precisando ser, necessariamente, o maior valor que ele é capaz de operar.

- Campo *Value Constant Conversion*: normalmente, os bits recebidos necessitam ser multiplicados por uma constante para serem recebidos ou enviados ao transdutor para refletirem a grandeza real medida ou atuada.

- Campo *First Data Byte*: neste campo deve ser informado qual o primeiro byte da sequência enviada, presente no campo *SPI Access Bytes*, onde se localiza o dado utilizado pelo transdutor. A sequência no tempo dos bytes enviados está mostrada na Figura 17, para a correta consideração do valor correspondente para este campo. A Figura 18 apresenta um exemplo de preenchimento.

- Campo *Last Data Byte*: neste campo deve ser informado qual o último byte da sequência enviada, presente no campo *SPI Access Bytes*, onde se localiza o dado utilizado pelo transdutor. A sequência no tempo dos bytes enviados está mostrada na Figura 17, para a correta consideração do valor correspondente para este campo. A Figura 18 apresenta um exemplo de preenchimento.

- Campo *First Data Bit*: similar ao campo *First Data Byte* em que informa a partir de qual bit se inicia o dado para o transdutor dentro do primeiro byte. A sequência das posições

dos bits está mostrada na Figura 17, para a correta consideração do valor correspondente para este campo. A Figura 18 apresenta um exemplo de preenchimento.

- Campo *Last Data Bit*: similar ao campo *Last Data Byte* em que informa qual o último bit do dado dentro do último byte. A sequência das posições dos bits está mostrada na Figura 17, para a correta consideração do valor correspondente para este campo. A Figura 18 apresenta um exemplo de preenchimento.

- Campo *Is Actuator*: este campo informa ao subsistema se o registrador de função representa um atuador.

- Campo *Actuator Value*: neste campo deve ser preenchido o valor que deverá ser enviado ao atuador. Este campo deverá apenas se preenchido caso o registrador de função seja um atuador.

- Campo *Is Sensor*: este campo informa ao subsistema se o registrador de função representa um sensor.

- Campo *Activate Interruption*: este campo informa ao subsistema se será utilizada a funcionalidade de interrupção que possui dois níveis que devem ser configurados pelo usuário: o *Warning* e o *Danger*. Ambas interrupções funcionam da mesma maneira e operam de forma independente, e permitem que quando as condições estabelecidas pelo usuário são atingidas, o subsistema efetue uma operação em outro transdutor ativo. O objetivo dessa funcionalidade são medidas de segurança, em que, por exemplo, no nível *Warning* haja uma medida a ser tomada como acionar uma lâmpada ou um atenuador, e no nível *Danger* seja efetuado o desligamento do sistema.

- Campo *Activate Warning Interruption*: este campo ativa ou desativa apenas a interrupção do tipo *Warning*.

- Campo *Warning Threshold*: neste campo deve conter o valor limiar para que o sistema ative a interrupção *Warning*.

- Campo *Warning Address*: este campo contém o endereço SPI do transdutor que será acionado quando ocorrer a interrupção *Warning*.

- Campo *Warning Action*: este campo contém o conjunto de dados que será enviado ao transdutor quando ocorrer a interrupção *Warning*.

- Campo *Activate Danger Interruption*: este campo ativa ou desativa apenas a interrupção do tipo *Danger*.

- Campo *Danger Threshold*: neste campo deve conter o valor limiar para que o sistema ative a interrupção *Danger*.

- Campo *Danger Address*: este campo contém o endereço SPI do transdutor que será acionado quando ocorrer a interrupção *Danger*.

- Campo *Danger Action*: este campo contém o conjunto de dados que será enviado ao transdutor quando ocorrer a interrupção *Danger*.

É importante ressaltar que um TEDS conterá apenas um cabeçalho, mas poderá conter mais de um registrador de função, permitindo a correta operação de transdutores que sejam sensores e atuadores ou que possuam mais de um tipo de função.

Desse modo, o TEDS desenvolvido neste trabalho, engloba as funções dos transdutores que estão comercialmente disponíveis, possibilitando a correta utilização deles pelo subsistema.

Parameter	Value
Name	
ID	
Active	
SPI Address	
Clock Speed	
Chip Select Idle High	
Clock Polarization/Phase Mode	
Data for Setup	
Register	
Active Register	
SPI Access Bytes	
SI Unit	
Minimum Value	
Maximum Value	
Value Constant Conversion	
First Data Byte	
Last Data Byte	
First Data Bit	
Last Data Bit	
Is Actuator	
Actuator Value	
Is Sensor	
Activate Interruption	
Activate Warning Interruption	
Warning Threshold	
Warning Address	
Warning Action	
Activate Danger Interruption	
Danger Threshold	
Danger Address	
Danger Action	

Figura 16 - TEDS com o cabeçalho e um registrador de função.

Fonte: Elaborada pelo autor

Figura 17 - Definição dos Bytes e Bits no processo de comunicação entre o Raspberry Pi e o transdutor. A sequência é da esquerda para direita, ou seja, o byte0 é enviado primeiro, seguido do byte1 e sucessivamente. Existem casos em que, durante uma comunicação, para o escravo enviar dados ao mestre, é necessário que o mestre envie bytes de comando seguido de bytes sem importância de maneira que o clock gerado permita ao escravo responder ao comando solicitado, retornando os bytes necessários.

Fonte: Elaborada pelo autor

Figura 18 - Exemplo de preenchimento dos campos: os bytes são enviados do byte0 ao byte4 nessa ordem. Os bits marcados são referentes ao dado utilizado pelo transdutor. Portanto o campo *First Data Byte* será preenchido com o valor 1, o campo *Last Data Byte* será preenchido com o valor 2, o campo *First Data Bit* será preenchido com o valor 3 e o campo *Last data Bit* será preenchido com o valor 2. No caso de um sensor, o dado conteria o valor de uma medição, já para o caso de um atuador, o dado representa o valor que deve ser enviado.

Fonte: Elaborada pelo autor

3.3.1 Implementação do TEDS em Python

Os campos do TEDS são nomes aos quais devem ser atribuídos valores e é exatamente a construção lógica de um dicionário em Python, no qual a uma chave são atribuídos valores, tornando-se, assim, natural a escolha desse tipo para esta implementação.

O dicionário em Python é definido entre chaves `{}` e os valores são inseridos atribuindo a uma chave o seu valor correspondente. Portanto a chave será o nome do campo, definido em uma *string* com o valor que pode ser qualquer outro tipo presente na linguagem. Abaixo estão relacionados os tipos empregados para cada campo:

- Campo *Name*: *string*.
- Campo *ID*: *string*.
- Campo *Active*: número booleano.
- Campo *SPI Address*: número inteiro.
- Campo *Clock Speed*: número inteiro.
- Campo *Chip Select Idle High*: número booleano.
- Campo *Polarization/Phase Mode*: número inteiro
- Campo *Data for Setup*: *string*. Neste ponto cabe uma nota importante com relação ao formato que é utilizado para este campo. Apesar de ser um tipo *string*, os dados inseridos são

definidos de uma forma particular: os dados enviados aos escravos por SPI precisam ser separados por bytes, por isso cada byte enviado é separado por vírgula. Quando se deseja enviar ou receber dados do escravo, alguns não suportam que seja realizada mais de uma operação por comunicação, e então é necessário reiniciar a comunicação e enviar/receber novos dados. Dessa forma, nesses casos, cada conjunto de bytes é separado por ponto e vírgula, e o sistema automaticamente reinicia a comunicação e segue enviando os próximos conjuntos de bytes. Por exemplo, se o preenchimento do campo for “255,4;10,0” o sistema realizará dois processos de comunicação: no primeiro processo enviará o byte 255 (11111111 em binário) seguido pelo byte 4 (100 em binário), e no processo seguinte enviará o byte 10 (1010 em binário) seguido pelo byte 0.

- Campo *Active Register*: número booleano.
- Campo *SPI Access Bytes*: *string* e segue o mesmo padrão do campo *Data for Setup*.
- Campo *SI Unit*: *string*.
- Campo *Minimum Value*: número de ponto flutuante.
- Campo *Maximum Value*: número de ponto flutuante.
- Campo *Value Constant Conversion*: número de ponto flutuante.
- Campo *First Data Byte*: número inteiro.
- Campo *Last Data Byte*: número inteiro.
- Campo *First Data Bit*: número inteiro.
- Campo *Last Data Bit*: número inteiro.
- Campo *Is Actuator*: número booleano.
- Campo *Actuator Value*: número de ponto flutuante.
- Campo *Is Sensor*: número booleano.
- Campo *Activate Interruption*: número booleano.
- Campo *Activate Warning Interruption*: número booleano.
- Campo *Warning Threshold*: número de ponto flutuante.
- Campo *Warning Address*: número inteiro.
- Campo *Warning Action*: *string* com o mesmo padrão do campo *Data for Setup*.
- Campo *Activate Danger Interruption*: número booleano
- Campo *Danger Threshold*: número de ponto flutuante.
- Campo *Danger Address*: número inteiro.
- Campo *Danger Action*: *string* com o mesmo padrão do campo *Data for Setup*.

Além dos campos citados, existe outro campo que é o próprio nome do registrador de funções e a sua utilização. Este campo é do tipo *string* que permite que o usuário o modifique

para que seja fácil a identificação do registrador. Ele é também o valor da chave do dicionário Python de modo que ele mesmo seja um dicionário e contenha os campos de registrador de função referentes apenas dele mesmo.

Portanto, cada campo do cabeçalho é uma chave de dicionário e quando é incluído um registrador de função seu nome é acrescentado como chave e os campos do registrador de função são chaves pertencentes ao corpo do seu dicionário, sendo um dicionário aninhado em outro dicionário.

Após a criação e preenchimento dos campos do TEDS, ele é salvo no servidor, que será discutido na seção 3.5, em uma pasta que contam apenas este tipo de arquivo, no formato de arquivo JSON que mantém a estrutura do TEDS e permite a fácil leitura do usuário se for necessário, com o nome do arquivo sendo o ID do TEDS e a extensão “.ted”. Ao mesmo tempo, é acrescentado em uma lista os TEDS que já foram criados no sistema para que eles possam ser mostrados ao usuário e modificados caso necessário e a lista é salva em um arquivo separado.

3.4 Gerenciador de transdutores

Foi desenvolvido o módulo Python chamado “*ciermagtransducers*” para ser o gerenciador de transdutores através de duas classes que foram criadas dentro desse módulo, Register e Transducer.

Quando um TEDS é incluído no subsistema, automaticamente é gerada a instância da classe Transducer. O TEDS é passado como argumento para o construtor da classe, que converte cada campo em um atributo da classe, exceto o campo do registrador de função, que instancia a classe Register e passa os campos presentes no registrador de função como seus atributos. A classe Register fica armazenada em um dicionário dentro da classe Transdutor, sendo o seu nome o nome da chave e o valor da chave a própria classe.

Após os atributos das classes serem gerados, o construtor da classe prossegue realizando a configuração inicial do transdutor que é:

- Gerar uma instância da classe que gerencia o protocolo SPI: o módulo *spidev* fornece uma classe para controlar em alto nível as comunicações SPI através de métodos próprios.

- Inicializa e configura os pinos de GPIO para receber o endereço SPI do transdutor: o módulo *gpio* permite a configuração dos pinos em entrada, permitindo que seja lido o valor do pino, ou saída, permitindo que seja determinado o valor do pino. Neste caso eles são

configurados como saídas para que seja escrito neles o valor do endereço de cada transdutor e enviados aos multiplexadores.

- Realiza a chamada do método que configura o transdutor: neste ponto o transdutor está pronto para ser utilizado, mas ainda necessita ser configurado para que funcione de acordo com a necessidade do usuário. Neste momento as informações inseridas no campo do TEDS *Data for Setup* são utilizadas e são enviadas ao transdutor, finalizando o processo de inicialização da classe Transdutor e permitindo o funcionamento correto do transdutor.

A classe Register não possui métodos, pois é no contexto da classe Transducer que as comunicações SPI ocorrem, possuindo apenas os atributos que são utilizados nas comunicações. A classe Transducer possui os métodos de interação com os transdutores.

O registrador de função pode ser sensor ou atuador e não pode ser os dois ao mesmo tempo, de forma então que dois métodos de interação são necessários: o de leitura para o caso do sensor e o de escrita para o caso de atuador.

O método de leitura utiliza as informações contidas no campo SPI *Access Bytes* para buscar o dado bruto no transdutor, depois utiliza as informações contidas nos campos *First Data Byte*, *Last Data Byte*, *First Data Bit* e *Last Data Bit* para separar o valor de leitura do dado bruto e finalmente realiza a conversão multiplicando o valor de leitura pela constante do campo *Value Constant Conversion* e retorna o valor correto.

O método de escrita utiliza o valor do campo *Actuator Value* e o acrescenta a um conjunto de bytes conforme as informações nos campos *First Data Byte*, *Last Data Byte*, *First Data Bit* e *Last Data Bit*. Esses bytes são substituídos nos bytes do campo SPI *Access Bytes*, e são enviados ao transdutor.

A classe Transducer ainda conta com métodos que preparam os dados criando máscaras para os bits e bytes, métodos que associam ou dissociam os bits e bytes, métodos que atuam para efetuar as interrupções e métodos que retornam mensagens de *status* de cada transdutor para o usuário.

Ainda no módulo “ciernmagtransducer” foram criadas 4 funções que auxiliam o gerenciamento dos transdutores:

- Função “create_actuator_data”: quando um novo transdutor é criado no subsistema, ele busca em todos os transdutores existentes e ativos quais são atuadores e cria um arquivo com informações básicas para ser utilizado pelas interfaces gráficas e permitir que os valores atuadores sejam alterados, conforme mostrado na Figura 19. A estrutura é de forma que cada ID do transdutor seja uma chave de dicionário e o valor é um outro dicionário contendo todos os registradores de função e o nome definido pelo usuário. Cada nome do registrador de

função é uma chave de dicionário cujo valor é uma lista com o valor mínimo, o valor máximo, a unidade definida pelo usuário e o valor atual do atuador. Isso é realizado para simplificar a troca de informações entre o gerenciador de transdutores e o servidor que fornecerá esses dados para as interfaces gráficas.

```
{'transducerid1': {'register1': ('MinValue', 'MaxValue', 'Unit', 'ActuatorValue'),
                  'register2': ('MinValue', 'MaxValue', 'Unit', 'ActuatorValue'),
                  'registerN': ('MinValue', 'MaxValue', 'Unit', 'ActuatorValue'),
                  'name': 'transducername1'},
 'transducerid2': {'register1': ('MinValue', 'MaxValue', 'Unit', 'ActuatorValue'),
                  'register2': ('MinValue', 'MaxValue', 'Unit', 'ActuatorValue'),
                  'registerN': ('MinValue', 'MaxValue', 'Unit', 'ActuatorValue'),
                  'name': 'transducername2'},
 'transduceridN': {'register1': ('MinValue', 'MaxValue', 'Unit', 'ActuatorValue'),
                  'register2': ('MinValue', 'MaxValue', 'Unit', 'ActuatorValue'),
                  'registerN': ('MinValue', 'MaxValue', 'Unit', 'ActuatorValue'),
                  'name': 'transducernameN'}
}
```

Figura 19 - Formato genérico do arquivo de dados dos atuadores.

Fonte: Elaborada pelo autor

- Função “process_actuator_data”: quando o usuário modifica algum valor dos atuadores, o servidor envia um sinal para o gerenciador dos transdutores que busca no arquivo os dados dos atuadores e chama esta função para atualizar os valores dos atuadores.

- Função “update_sensor_data”: esta função lê todos registradores de função ativos e salva em um arquivo específico. O formato deste arquivo está mostrado na Figura 20, em que existem duas chaves primárias no dicionário: a chave “sensordata” contém os dados vindos dos transdutores e a chave “timedata” contém um vetor onde é armazenada a base de tempo que é atualizada no momento em que é realizada a varredura dos sensores. Cada nome dos registradores de função é a chave de um dicionário cujo valor é uma lista com os dados lidos e com a unidade do registrador de função. Todos os registradores, de um mesmo transdutor, fazem parte de um dicionário cuja chave é o ID do transdutor. Os transdutores fazem parte de um outro dicionário cuja chave é “sensordata”.


```
{'sensordata': {'transduceridN': {'register1': ([3, 3, 3, 3, 3, 3, 3, 3, 3, 3], 'V'),
    'register2': ([4, 4, 4, 4, 4, 4, 4, 4, 4, 4], 'V'),
    'registerN': ([5, 5, 5, 5, 5, 5, 5, 5, 5, 5], 'V'),
    'name': 'transducernameN'},
    'transducerid2': {'register1': ([0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'W'),
    'register2': ([1, 1, 1, 1, 1, 1, 1, 1, 1, 1], 'A'),
    'registerN': ([2, 2, 2, 2, 2, 2, 2, 2, 2, 2], 'deg_C'),
    'name': 'transducername2'},
    'transduceridl': {'register1': ([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], 'Ohm'),
    'register2': ([1, 3, 5, 7, 9, 11, 13, 15, 17, 19], 'A'),
    'registerN': ([1, 4, 7, 10, 13, 16, 19, 22, 25, 28], '%'),
    'name': 'transducernamel'}},
    'timedata': ([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], 'seconds')}
```

Figura 20 - Exemplo do formato dos dados contidos no arquivo dos sensores.

Fonte: Elaborada pelo autor

- Função “message_reader”: esta função gerencia as mensagens de *status* geradas por cada transdutor. Existem 5 tipos de mensagem:

- “*Status ok*”, utilizada para designar o funcionamento normal do transdutor.
- “*Actuator value too high. Maximum value set.*”, utilizada para designar quando o usuário tenta inserir um valor superior ao suportado pelo transdutor no campo *Maximum Value* do TEDS.
- “*Actuator value too low. Minimum value set.*”, utilizada para designar quando o usuário tenta inserir um valor inferior ao suportado pelo transdutor no campo *Maximum Value* do TEDS.
- “*Warning: sensor value passed threshold.*”, utilizada para avisar o usuário de que o limiar da interrupção *Warning* foi atingido.
- “*Danger: sensor value passed threshold.*”, utilizada para avisar o usuário de que o limiar da interrupção *Danger* foi atingido.

Entre as mensagens acima citadas existe uma prioridade de modo que a mensagem mais importante não seja sobreposta por uma mensagem menos importante, que segue a ordem apresentada. Uma mensagem de prioridade maior somente será substituída por uma de mensagem menor depois que o usuário tiver recebido as mensagens.

3.5 Servidor HTTP

Para realizar a comunicação entre o terminal computadorizado e o Raspberry Pi, foi definido que fosse utilizado o protocolo HTTP que possui a arquitetura cliente-servidor de maneira que o cliente inicia a comunicação efetuando as requisições para o servidor que responde entregando as informações requisitadas.

Foi utilizado o módulo Flask para desenvolver o servidor, por ser simples e eficiente nesta tarefa. A Figura 21 mostra o trecho de código para o servidor retornar a página de login.

A utilização de decoradores permite que as funções que os sucedem sejam facilmente modificadas e, como exemplo mostrado pela figura, o roteamento das páginas e os métodos que ela suporta são claramente configurados não necessitando que seja alterado em nada a função em si.

```
# Serving the login page
@server_app.route('/', methods=['POST', 'GET'])
@server_app.route('/login', methods=['POST', 'GET'])
def login():
    """Login page view function. """
    error = None

    # Check if login form was sent
    if request.method == 'POST':

        # Check if it is a valid username and password
        if request.form['username'] != sc.USERNAME:
            error = 'Invalid username'
        elif request.form['password'] != sc.PASSWORD:
            error = 'Invalid password'
        else:
            session['logged_in'] = True
            return redirect(url_for('main_page'))

    # Check if it is a GET method
    elif request.method == 'GET':
        page_dict = {'current_time': sf.get_time()}
        return render_template('login_page.html', **page_dict)

    # Recieved invalid method
    else:
        return METHOD_ERROR
```

Figura 21 - Trecho de código usado no Servidor desenvolvido, em que é definida a função para retornar a página de login.

Fonte: Elaborada pelo autor

Na função de login, nota-se que existe um objeto chamado “request” que é o responsável por gerenciar as requisições. Ele contém um método chamado “method” que identifica como foi a requisição feita pelo cliente. No caso, é feita uma separação no código entre os dois tipos de requisições definidas pelo protocolo HTTP: a requisição do tipo POST que recebe o formulário preenchido pelo usuário para efetuar login e a requisição do tipo GET que retorna a página web com o formulário para ser preenchido pelo usuário.

Caso o servidor receba a requisição do tipo POST, ele acessa o objeto “request” e acessa o formulário de login preenchido pelo usuário através do atributo “form”. Com o formulário, o servidor verifica se o usuário e a senha estão corretos, e em caso afirmativo, ele

inicia a sessão para o usuário redirecionando-o para a página “main_page”. Caso o servidor receba a requisição do tipo GET, ele renderiza a página de login, através do Jinja, com base em um dicionário que contém variáveis para serem inseridas na página, e no caso a variável é a data e hora do acesso à página.

O subsistema conta com duas interfaces para o usuário, uma sendo a interface gráfica desenvolvida em Python, discutida na seção 3.7, e a outra sendo páginas web para serem acessados por navegadores, discutidas na seção 3.8. As duas interfaces possuem maneiras diferentes de operar e utilizam dados distintos de modo que o servidor deve diferenciar quais das duas interfaces estão solicitando os dados. Isto é feito alterando o cabeçalho da requisição HTTP, que é gerado automaticamente pelos navegadores, mas pode ser definido na interface Python, modificando-se o campo “user_agent”. Deste modo, o servidor pode verificar quem é o cliente acessando no objeto “request” o atributo “user_agent” e retornar para o cliente os dados compatíveis com cada um.

O servidor organiza as informações em pastas distintas: a pasta “teds” contém apenas os TEDS criados no subsistema; a pasta “files” contém os arquivos com os dados dos atuadores, sensores lista dos TEDS, lista dos gráficos dos sensores gerados e as mensagens de status dos transdutores; a pasta “templates” contém os arquivos HTML para gerar as páginas web; e a pasta “static” contém o arquivo CSS para a formatação das páginas web.

Seguindo o mesmo princípio de construção utilizado para a página de login e adequando o conteúdo a ser fornecido para cada tipo de cliente, foram desenvolvidas outras funções para o gerenciamento dos TEDS, visualização dos sensores e interface para os atuadores.

3.6 Integração do Gerenciador de Transdutores com o Servidor

O Raspberry Pi deverá executar, ao mesmo tempo, o módulo do servidor e o módulo do gerenciador de transdutores. Isto é realizado através do módulo “*threading*” que possui o ferramental necessário para que cada módulo seja executado em uma *thread*.

Foi desenvolvido então o módulo “rpi_main” que irá construir as *threads* para cada um dos dois módulos. A criação das *threads* ocorre criando uma classe que herda os atributos da classe “Thread” definida no módulo “*threading*”. Em seguida, basta definir o método “run” para a classe criada, contendo todo o código necessário para o correto funcionamento do módulo. No caso do servidor, basta importar o objeto servidor criado e chamar seu método

“run”. Já no caso do gerenciador de módulos, foram chamadas as funções de gerenciamento descritas na seção 3.4.

É importante notar que o servidor e o gerenciador precisarão acessar os mesmos arquivos, de maneira que pode haver o corrompimento de algum arquivo ou perda de dados se nenhuma medida for tomada. Por isso o módulo “*threading*” possui os objetos “*Lock*” que são empregados para ordenar o acesso de diferentes *threads* a um mesmo objeto: uma *thread* adquire a “*Lock*”, acessa o objeto, e a libera, e apenas então outra *thread* pode adquirir a mesma “*Lock*” para acessar o objeto. Isso é utilizado para acessar os arquivos TEDS, arquivos de dados dos sensores, arquivos de dados dos atuadores e outras variáveis que são utilizadas para que o servidor avise o gerenciador e vice-versa. As aquisições e liberações das “*Locks*” são gerenciadas pelo subsistema e ocorrem pelo tempo mínimo de utilização das variáveis e dos objetos. Por exemplo, quando algum TEDS é criado ou modificado, o servidor os acrescenta ao final de uma lista e, conforme o gerenciador for criando os transdutores, ele vai removendo os primeiros TEDS da lista; com o uso da “*Lock*”, apenas um dos dois acessa a lista de cada vez.

3.7 Cliente Python

O protocolo HTTP possui a arquitetura cliente-servidor, portanto, há a necessidade de se desenvolver um cliente com uma interface adequada para que o usuário possa criar, editar e gerenciar os TEDS, visualizar as medições realizadas pelos sensores e defina valores para os atuadores. Esse cliente deverá ser integrado ao Console já existente no sistema de modo a agregar as funcionalidades do subsistema. Por ser desenvolvido em Python, ele será denominado Cliente Python.

Foi escolhido o PyQtGraph para criar a interface, pois se trata de um módulo que possui diversas ferramentas para a elaboração de interfaces voltadas para área científica. Possui gráficos 2D, 3D, controles de região de interesse, diagramas de fluxo, árvore de parâmetros e muitos tipos de *widgets*.

Dentre as ferramentas que o PyQtGraph possui, uma que se mostrou bastante atrativa para o trabalho foi a árvore de parâmetros. Ela constitui-se de parâmetros que possuem um nome, um tipo (número inteiro, número de ponto flutuante, *string*, número booleano, grupo, entre outros) para o valor e uma grande quantidade de propriedades como ser renomeável, possuir limites superior e inferior, ser removível, não ser editável, um parâmetro pode ser um

pai e conter outros parâmetros filhos. Esse tipo de estrutura se encaixa com o formato dos TEDS, sendo utilizada para a elaboração do editor e gerenciador de TEDS.

Cada parâmetro da árvore se torna um campo existente no TEDS, com seus nomes e valores associados e as propriedades dos parâmetros são utilizados na interface gráfica existente. O registrador de função é construído como um parâmetro do tipo *group* sendo o pai dos seus campos do TEDS. Seguindo as definições dos tipos para cada campo do TEDS conforme a seção 3.3.1, a Figura 22 mostra a implementação.

Parameter	Value
Name	Termopar K
ID	2016May25140938
Active	☒
SPI Address	4
Clock Speed	50 kHz
Chip Select Idle High	☒
Clock Polarization/Phase Mode	1
Data for Setup	128,144,35,3
Temperature	
Active Register	☒
SPI Access Bytes	12,0,0,0
SI Unit	deg C
Minimum Value	0
Maximum Value	200
Value Constant Conversion	0.0625
First Data Byte	1
Last Data Byte	2
First Data Bit	7
Last Data Bit	0
Is Actuator	☐
Actuator Value	0
Is Sensor	☒
Activate Interruption	☒
Activate Warning Interruption	☒
Warning Threshold	35
Warning Address	2
Warning Action	127
Activate Danger Interruption	☒
Danger Threshold	40
Danger Address	2
Danger Action	255

Parameter	Value
2016May06141756	
Shift Register 8bits with leds	☐
2016May06194553	
ADC 8ch 8bits	☐
2016May25140938	
Termopar K	☒

Save TEDS
New TEDS
Remove TEDS
Copy TEDS
Add Register Function

Figura 22 - Editor e gerenciador de TEDS da interface gráfica em Python.

Fonte: Elaborada pelo autor

A figura possui duas regiões separadas: à esquerda se encontra o editor de TEDS e à direita a lista dos TEDS disponíveis no subsistema. Essas e as demais regiões dessa interface foram construídas utilizando o objeto “Dock” existente no PyQtGraph, que permite que cada “Dock” seja arrastado e movido para qualquer região da janela, dando ao usuário liberdade para customizar o ambiente de trabalho.

No editor de TEDS o usuário edita o valor do campo que já possui um método de entrada definido baseado no seu tipo associado: os campos com valores booleanos possuem caixas de seleção, os campos com valores inteiros, ponto flutuante ou *string* possuem caixas de texto, o campo ID, que não deve ser modificado pelo usuário, não possui método de entrada, e o registrador de função apresenta o formato de um parâmetro que pode esconder os parâmetros filhos.

Na parte inferior do editor, se localizam botões de ação com as seguintes funções:

- “Save TEDS”: salva o TEDS, enviando-o ao servidor.
- “New TEDS”: limpa os campos para que um novo TEDS seja criado.
- “Remove TEDS”: remove o TEDS selecionado.
- “Copy TEDS”: cria uma cópia do TEDS atual, gerando um novo ID.
- “Add Register Function”: adiciona um novo registrador de função ao TEDS selecionado.

À direita se encontra a área que mostra os TEDS disponíveis no subsistema de modo que, ao selecionar o TEDS utilizando a caixa de seleção correspondente, o TEDS é carregado automaticamente no editor para ser alterado. A lista mostra o ID do TEDS e o nome definido pelo usuário para facilitar a identificação. Na Figura 22 estão presentes, na lista, três TEDS criados.

Outra área criada é a interface para os atuadores que também utilizou da árvore de parâmetros para organizar as informações. O ID do TEDS é utilizado como o parâmetro pai, com o nome definido pelo usuário e os registradores de atuadores como filhos. Cada registrador possui como parâmetros filhos os campos da Figura 19. Os campos não são editáveis, com exceção do campo para alterar o valor do atuador. Nesta área estão presentes 2 botões, o primeiro para atualizar os atuadores existentes no subsistema, e o segundo para enviar os dados para o servidor que repassará ao gerenciador de transdutores para atualizar seus valores.

A quarta área da interface é a dos sensores, na qual estão os gráficos de todas as medições de todos os sensores ativos. Da mesma forma que ocorre na página web, cada sensor tem um gráfico com os valores das medições dos registradores de função como as curvas dos gráficos. Devido à arquitetura cliente-servidor do protocolo HTTP, o usuário deverá requisitar a atualização do gráfico, por isso existe o botão “Update Plot”.

Para que a interface possa se comunicar adequadamente com o servidor, foi necessária a utilização do módulo “httplib” que possui uma classe chamada “HTTPConnection” para gerenciar a comunicação de um cliente no protocolo HTTP. Esta classe possui o objeto

“request” no qual é possível colocar o tipo da requisição, o cabeçalho e a mensagem para serem enviados. Como mencionado na seção 3.5 em que foi discutido o servidor, o cabeçalho foi elaborado de modo que o campo “user-agent” pudesse diferenciar a conexão do cliente Python para que o servidor envie as informações necessárias. Ao contrário dos navegadores, o servidor não envia ao cliente Python páginas de web, mas sim os dados básicos no formato JSON para que o cliente os processe e os apresente na interface. Apenas dois tipos de requisição foram utilizados, a requisição do tipo GET, para receber os dados do servidor e a requisição do tipo POST para enviar os TEDS e os dados dos atuadores.

3.8 Cliente Web

Como o Cliente Python foi desenvolvido para ser integrado ao Console presente no terminal computadorizado, somente um usuário pode utilizar o sistema. Em alguns casos, como no ambiente de pesquisa em que vários pesquisadores utilizarão o mesmo equipamento, com frequência ele precisa ser reconfigurado, monitorado e ajustado. Por isso, enquanto um usuário utiliza o sistema para configurar as sequências de pulsos ou processar os dados obtidos, foi desenvolvido outro cliente possibilitando que outro usuário possa utilizar, verificar ou configurar o subsistema de dispositivos periféricos.

O Cliente Web é um conjunto de páginas web desenvolvidas de forma que o usuário possa utilizar o subsistema e todas as suas funcionalidades através de um navegador.

Como já mostrado na seção 3.5, Servidor desenvolvido com o Flask permite que sejam criados os acessos e manipulações de dados da página web. A renderização da página é realizada pelo Jinja que possibilita a criação de páginas com conteúdo gerado automaticamente pelo servidor.

Para criar as páginas web é necessário criar no servidor um dicionário com as variáveis que deverão ser passadas para o Jinja através do método “render_template” do Flask. Desse modo, no processo de renderização, essas variáveis serão substituídas e utilizadas para produzir um arquivo HTML que será enviado pelo servidor ao cliente e exibido pelo navegador.

As variáveis passadas podem ser usadas e manipuladas pelo Jinja, de acordo com os comandos, chamados de delimitadores, empregados. Para substituir a variável no arquivo HTML, deve-se utilizar o delimitador de expressão sinalizado por chaves duplas {{ }} cuja função é resolver a expressão dentro do delimitador e retornar uma *string* do resultado. A

Figura 23 mostra um trecho do código HTML da página “main_page” em que é possível ver a utilização desse delimitador para exibir variáveis como “message” e “methods”.

Outro delimitador, que foi empregado nesse exemplo, é o de declaração, sinalizado por chave e porcentagem {% %}, que é utilizado para fazer controle de fluxo. No caso citado foi usado o FOR para que fosse feita a iteração sobre um dicionário contendo os ID dos TEDS como chave e os nomes definidos pelo usuário como valor. O loop foi colocado dentro de uma tabela e a cada iteração, uma nova linha da tabela era criada com um link para o direcionamento da página de edição de TEDS, cujo texto é o ID do TEDS e o nome do TEDS separados por dois pontos (:). Pode-se ressaltar também o uso dos delimitadores de expressão para substituir os valores utilizados como texto do link.


```

<body>
  <center>
    
    .
    <h1>Ciermag Peripherals</h1>
    <br>
    <p> {{ message }} </p>
    <p> {{ method }} </p>

    <table>
      <th>Available TEDS</th>
      {% for tedsid in teds_dictlist %}
        <tr>
          <td id="tdnormal">
            <a href="{{ url_for('editor_page', selected_teds=tedsid) }}">
              <p id="ptable">{{ tedsid }}: {{teds_dictlist[tedsid]}}</p>
            </a>
          </td>
        </tr>
      {% endfor %}
      <tr>
        <td id="tdnormal">
          <a href="{{ url_for('editor_page', selected_teds='new_teds') }}">
            <p id="ptable">New TEDS</p>
          </a>
        </td>
      </tr>
    </table>
    <br>
    <br>
    <table>
      <th colspan="2">Transducer Area</th>
      <tr>
        <td id="tdnormal">
          <a href="{{ url_for('controller_page') }}">
            <p id="ptable">Controllers Page</p>
          </a>
        </td>
        <td id="tdnormal">
          <a href="{{ url_for('sensor_main_page') }}">
            <p id="ptable">Sensors Page</p>
          </a>
        </td>
      </tr>
    </table>

    <br>
    <p> {{ current_time }} </p>
    <br>
    <p> {{ print }}</p>
  </center>
</body>

```

Figura 23 - Trecho do código HTML para página "main_page"

Fonte: Elaborada pelo autor

Atualmente, o subsistema conta com cinco páginas web para realizar a o gerenciamento dos TEDS e dos transdutores:

- Página de login: é utilizada para a segurança de modo que somente usuários permitidos possam operar o subsistema.

- Página principal: é utilizada para exibir a lista dos TEDS existentes com links para as respectivas páginas para serem editados. Fornece acesso para a criação de novos TEDS, para a página de visualização dos sensores e para a página de interação com os atuadores.

- Página do editor de TEDS: de acordo com o TEDS selecionado, podendo ser um já existente ou um novo, o servidor constrói um formulário para editar ou criar um novo TEDS.

- Página dos sensores: para cada transdutor, é gerado um gráfico interativo e se o transdutor possui mais de um sensor, todos eles são mostrados no mesmo gráfico. Essa funcionalidade foi implementada utilizando-se o módulo “bokeh” que, recebe os dados dos sensores localizados no arquivo através da programação em Python, e gera um arquivo HTML contendo um trecho em JavaScript com o gráfico interativo.

- Página dos atuadores: o servidor utiliza dos dados contidos no arquivo dos atuadores para listar todos os ativos em um formulário para permitir que seus valores de saída possam ser ajustados.

4 RESULTADOS

4.1 Criação de um TEDS através da interface gráfica Python

Nesta seção será demonstrado um exemplo de criação de TEDS com a interface gráfica em Python, sendo uma das duas formas de interação do usuário com o subsistema de dispositivos periféricos.

Quando o cliente conecta ao servidor, ele requisita a lista dos TEDS existentes no subsistema e, como não há TEDS selecionados na lista, o editor apresenta um novo TEDS em branco que pode ser utilizado para preenchimento.

O TEDS a ser criado será do linearizador e digitalizador universal para termopares MAX31856 montado em uma placa de circuito impresso fabricado pela empresa Cicuitar (33), cujo *datasheet* se encontra no Anexo A no qual contém as informações necessárias para o preenchimento dos campos do TEDS.

Inicialmente deve-se preencher o cabeçalho:

- Campo *Name*: nome para ser identificado pelo usuário, que nesse caso será “MAX31856 – Termopar linearizador e digitalizador universal”.

- Campo *ID*: gerado automaticamente pelo subsistema que neste caso é “2016Jun17120338”.

- Campo *Active*: deve ser marcado para ele estar ativo e ser utilizado.

- Campo *SPI Address*: será preenchido com o valor “4”.

- Campo *Clock Speed*: na página 5 do *datasheet* mostra que a frequência máxima de *clock* permitida é 5MHz, portanto será utilizado o valor “4000000”.

- Campo *Chip Select Idle High*: na página 6 do *datasheet* a Figura 1 mostra um processo de comunicação SPI em que o *Chip Select* é mantido alto enquanto não há comunicação, portanto deve ser marcada a caixa.

- Campo *Polarization/Phase Mode*: na página 15 do *datasheet* a Tabela 5 mostra que o dado pode ser recebido com a polarização do clock sendo 0 na descida do clock, portanto esse campo será preenchido com o valor 1.

- Campo *Data for Setup*: este campo será necessário reunir várias informações do *datasheet*. A primeira delas é na página 15 na seção *Address and Data Bytes* que informa que o dispositivo permite o envio de mais de um comando de escrita por comunicação, que o

ponteiro interno da memória é acrescentado a cada byte enviado, permitindo que seja passado por todos os registradores internos para serem configurados. A próxima informação está na página 18 na Tabela 6 que mostra os endereços dos registradores de memória, neste caso, será necessário configurar os 3 primeiros registradores. Em seguida, na página 19, mostra o significado de cada bit do primeiro registrador, de endereço 128, no qual será configurado para o dispositivo realizar conversões automáticas no bit7 e será habilitado o teste de circuito aberto (mostrado na página 14, tabela 4), de modo que os demais bits serão deixados com suas configurações originais em 0 portanto, o valor que deverá ser escrito nesse registrador será 10010000 em binário ou 144 em decimal. O próximo registrador, de endereço 129 possui suas funções descritas na página 20 e será utilizado 4 medições tomadas como média e o termopar utilizado será o tipo K, portanto nesse registrador será colocado o valor 00100011 em binário ou 35 em decimal. Por fim, o registrador de endereço 130, cujas funcionalidades estão descritas na página 21, será mascarado o aviso de sob/sobre tensão e o aviso de circuito aberto do termopar, portanto será colocado o valor 00000011 em binário, ou 3 em decimal. Finalmente, o valor inserido nesse campo será “128,144,35,3”.

Em seguida será necessário criar um registrador de função, clicando no botão “Add Register Function”, para que sejam efetuadas as medições no termopar. Esse registrador será configurado da seguinte maneira:

- Campo *Register Name*: ao clicar com o botão direito sobre o nome do registrador de função, aparecem as opções de remover e renomear. Clicando na opção renomear é possível definir um nome para o registrador, que no caso será “Termopar tipo K”.

- Campo *Active Register*: este campo deve ser marcado para estar ativo no subsistema.

- Campo *SPI Access Bytes*: nas páginas 24-25 são mostrados os registradores que possuem os valores da medida do termopar com endereço 12. São 3 bytes com os dados, portanto o valor nesse campo será “12,0,0,0”.

- Campo *SI Unit*: o transdutor realizar as medidas em graus Celsius, portanto o valor nesse campo será “deg C”.

- Campo *Minimum Value*: na página 1 do *datasheet*, é mostrado que o transdutor pode medir temperaturas de até -210°C, portanto será colocado o valor de “0”.

- Campo *Maximum Value*: na página 1 do *datasheet*, é mostrado que o transdutor pode medir temperaturas de até 1800°C, portanto será colocado o valor de “200”.

- Campo *Value Constant Conversion*: na página 13 do *datasheet*, a tabela 3 mostra que a variação de um bit do segundo byte de dados apresenta uma variação de 0.0625°C, portanto será usado o valor de conversão “0.0625”. É importante observar nesta mesma tabela que o

dispositivo já realiza a linearização da temperatura e ajusta o *offset* de modo que o valor 0 nos bytes de dados corresponda à temperatura 0°C.

Para o preenchimento dos 4 campos a seguir, foi utilizado os dados presentes na página 13 do datasheet, a tabela 3, que serão utilizados os 2 primeiros bytes para os dados.

- Campo *First Data Byte*: o primeiro byte recebido representa o segundo byte contido no campo *SPI Access Bytes*, portanto será utilizado o valor “1”.

- Campo *Last Data Byte*: o segundo byte recebido representa o terceiro byte contido no campo *SPI Access Bytes*, portanto será utilizado o valor “2”.

- Campo *First Data Bit*: como o dado está contido a partir do sexto bit do byte 1, o valor será “6”.

- Campo *Last Data Bit*: como o dado termina no bit0 do byte 2, o valor será “0”.

- Campo *Is Actuator*: como é um sensor, este campo deve estar desmarcado.

- Campo *Actuator Value*: como é um sensor, este campo deve estar em branco.

- Campo *Is Sensor*: como é um sensor, este campo deve estar marcado.

Para o preenchimento da interrupção, será considerado o registrador de deslocamentos com LEDs para que possa ser visualizável a ação tomada pelo subsistema.

- Campo *Activate Interruption*: esta caixa será marcada para ativar as interrupções.

- Campo *Activate Warning Interruption*: esta caixa será marcada para ativar essa interrupção.

- Campo *Warning Threshold*: como exemplo será utilizado o valor “35”.

- Campo *Warning Address*: o endereço do registrador de deslocamento é “2”.

- Campo *Warning Action*: deverá ser enviado ao atuador o valor “64” no caso desta interrupção.

- Campo *Activate Danger Interruption*: esta caixa será marcada para ativar essa interrupção.

- Campo *Danger Threshold*: como exemplo será utilizado o valor “40”.

- Campo *Danger Address*: o endereço do registrador de deslocamento é “2”.

- Campo *Danger Action*: deverá ser enviado ao atuador o valor “128” no caso desta interrupção.

A Figura 24 mostra o exemplo de preenchimento antes de ser salvo e enviado ao subsistema, sendo possível também notar a existência de um TEDS criado anteriormente.

Parameter	Value
Name	MAX31856 - Termopar I
ID	2016Jun17120338
Active	☒
SPI Address	4
Clock Speed	4 MHz
Chip Select Idle High	☒
Clock Polarization/Phase Mode	1
Data for Setup	128,144,35,3
Termopar tipo K	
Active Register	☒
SPI Access Bytes	12,0,0,0
SI Unit	deg C
Minimum Value	0
Maximum Value	200
Value Constant Conversion	0.0625
First Data Byte	1
Last Data Byte	2
First Data Bit	6
Last Data Bit	0
Is Actuator	☐
Actuator Value	0
Is Sensor	☒
Activate Interruption	☒
Activate Warning Interruption	☒
Warning Threshold	35
Warning Address	2
Warning Action	64
Activate Danger Interruption	☒
Danger Threshold	40
Danger Address	2
Danger Action	128

Save TEDS

New TEDS

Remove TEDS

Copy TEDS

Add Register Function

Parameter	Value
2016May06194553	
ADC 8ch 8bits	☐

Figura 24 - Preenchimento do TEDS de exemplo
 Fonte: Elaborada pelo autor

Após o TEDS ser salvo, o subsistema constrói o transdutor para ser utilizado. É possível atualizar os gráficos dos sensores para que sejam exibidas as medidas realizadas. A Figura 25 mostra o caso em que o TEDS criado foi incluído na lista de TEDS disponíveis no sistema e, após pressionar o botão de atualização do gráfico, o gráfico das medidas do dispositivo é exibido.

Figura 25 - Interface Python após a criação do TEDS de exemplo
Fonte: Elaborada pelo autor

Com a criação de atuadores, a área de interface com os atuadores pode ser atualizada e todos os que estiverem disponíveis serão listados e cada atuador apresenta o seu próprio campo para ser alterado e enviado. A Figura 26 mostra não apenas a área dos atuadores, mas todas as áreas que compõem a interface. Vale ressaltar que cada área pode ser movimentada pela janela de modo a adequar melhor a interface de acordo com a necessidade do usuário.

Figura 26 - Interface Python com as 4 áreas: editor de TEDS, lista de TEDS disponíveis, interface dos atuadores e gráfico dos sensores.

Fonte: Elaborada pelo autor

4.2 Criação de um TEDS através de um navegador web

A outra opção para a interação do usuário com o subsistema é a utilização de um navegador, que será discutido nessa seção.

O TEDS a ser criado será o registrador de deslocamento, que possui um funcionamento simples: os bits colocados na entrada são deslocados para as saídas a cada pulso de *clock*.

Para acessar o subsistema, é necessário ir até o endereço “<http://192.168.42.128:5000>” e realizar o login na página mostrada na Figura 27.

Figura 27 - Página de login da interface web do subsistema
Fonte: Elaborada pelo autor

Após o login, o usuário é direcionado até a página principal, mostrado na Figura 28, que mostra a lista com os TEDS disponíveis no sistema, o link para a página dos atuadores e o link para a página dos sensores. Cada item na lista de TEDS é um link que levará até a sua respectiva página para edição. Como neste caso será adicionado um novo dispositivo, deverá ser clicado no link “New TEDS” para ser direcionado para a página do editor de TEDS em branco.

Figura 28 - Página principal da interface web do subsistema
Fonte: Elaborada pelo autor

Na página do editor, os campos dos TEDS são apresentados de forma análoga à da interface Python, com os respectivos campos para preenchimento, que deve ocorrer da seguinte maneira:

- Campo *Name*: o nome do transdutor será “Shift Register 8bits with leds”.
- Campo *ID*: gerado automaticamente pelo subsistema que neste caso é “2016Jun17164015”.
- Campo *Active*: deve ser marcado para ele estar ativo e ser utilizado.
- Campo *SPI Address*: será preenchido com o valor “2”.
- Campo *Clock Speed*: por ser um dispositivo simples, o valor será “20000”.
- Campo *Chip Select Idle High*: o dispositivo não possui *Chip Select* mas, devido à lógica utilizada, esse campo será marcado.
- Campo *Polarization/Phase Mode*: este tipo de dispositivo trabalha no modo “0”, portanto será este o valor

- Campo *Data for Setup*: por ser um dispositivo simples e não necessitar de configuração, o valor nesse campo deverá ser “0”.

Devido à filosofia de operação do protocolo HTTP, a adição de um registrador de função só pode ocorrer se o TEDS já estiver salvo. Assim, o cabeçalho do TEDS foi preenchido e em seguida utilizado o link na parte inferior da página para acrescentar o registrador de função. A Figura 29 mostra o preenchimento realizado já com a adição do registrador de função. A Figura 30 mostra a continuação da página do editor e a Figura 31 mostra o fim da página do editor de TEDS, mostrando os links para as outras páginas.

TEDS Editor Area		
Name	Shift Register 8bits with leds	
ID	2016Jun17164015	
Active	True	☒ True ☐ False
SPI Address	2	<input type="text" value="2"/>
Clock Speed	20000	<input type="text" value="20000"/>
Chip Select Idle High	True	☒ True ☐ False
Clock Polarization/Phase Mode	0	<input type="text" value="0"/>
Data for Setup	0	<input type="text" value="0"/>
Register		Remove

Figura 29 - Página do editor de TEDS da interface web.

Fonte: Elaborada pelo autor

Register		Remove
Active Register	False	☒ True ☐ False
SPI Access Bytes		0
SI Unit		
Minimum Value	0	0
Maximum Value	0	255
Value Constant Conversion	0	1
First Data Byte	0	0
Last Data Byte	0	0
First Data Bit	0	7
Last Data Bit	0	0
Is Actuator	False	☒ True ☐ False
Actuator Value	0	0
Is Sensor	False	☐ True ☒ False
Activate Interruption	False	☐ True ☒ False
Activate Warning Interruption	False	☐ True ☒ False
Warning Threshold	0	0

Figura 30 - Continuação da página do editor de TEDS, mostrando os campos do registrador de função para serem preenchidos.

Fonte: Elaborada pelo autor

Is Actuator	False	☒ True ☐ False
Actuator Value	0	<input type="text" value="0"/>
Is Sensor	False	☐ True ☒ False
Activate Interruption	False	☐ True ☒ False
Activate Warning Interruption	False	☐ True ☒ False
Warning Threshold	0	<input type="text" value="0"/>
Warning Address	1	<input type="text" value="1"/>
Warning Action	<input type="text"/>	
Activate Danger Interruption	False	☐ True ☒ False
Danger Threshold	0	<input type="text" value="0"/>
Danger Address	1	<input type="text" value="1"/>
Danger Action	<input type="text"/>	

[Add Register](#)
[New TEDS](#)
[Copy TEDS](#)
[Back to Main](#)

2016 Jun 17, Friday, 13:41:45

Figura 31 - Fim da página do editor de TEDS da interface web, mostrando os demais links da página.

Fonte: Elaborada pelo autor.

Como o TEDS criado se refere a um atuador, é possível ir até a página dos atuadores e verificar o funcionamento dele. A **Erro! Fonte de referência não encontrada.** mostra o uso da interface para alterar os valores dos atuadores: a página apresenta, em uma tabela, os atuadores com o campo aberto para o usuário alterar o valor e clicar no botão “save”. Na parte superior da figura, foi enviado o valor 202 (11001010 em binário) para o atuador, e na parte inferior da figura foi enviado o valor 1 para o atuador, sendo possível, portanto, verificar o funcionamento da interface e do subsistema.

Figura 32 - LEDs indicando o valor inicial no registrador de deslocamento antes do envio do comando para a mudança do valor efetuado através da interface web.

Fonte: Elaborada pelo autor.

Figura 33 – LEDs indicando o valor final no registrador de deslocamento após o envio do comando para a mudança do valor efetuado através da interface web

Fonte: Elaborada pelo autor

Por fim, a Figura 34 mostra a página dos sensores em que é mostrado um gráfico para cada transdutor. Os gráficos são interativos, de maneira que é possível, utilizando o mouse, arrastar, dar zoom e selecionar uma área específica, tal como é possível na interface Python.

Figura 34 - Página dos sensores com o gráfico interativo para cada dispositivo.
Fonte: Elaborada pelo autor

5 CONCLUSÕES

O subsistema desenvolvido supriu uma necessidade importante presente no Espectrômetro Digital existente do CIERMag, já que tem por objetivo gerenciar os dispositivos periféricos *non-real-time*, que não são controlados pelo mesmo, mas são fundamentais para criar as condições adequadas aos experimentos de RM.

O desenvolvimento de um modelo de *Transducer Electronic Data Sheet* (TEDS) inspirado no padrão IEEE 1451 permite que o subsistema possa gerenciar corretamente os dispositivos periféricos, normalmente transdutores, necessários aos experimentos de RM. O TEDS é genérico o suficiente para que, idealmente, qualquer transdutor possa ser configurado e operado: o cabeçalho define as propriedades básicas da configuração do protocolo SPI e de operação do transdutor; o registrador de função define as propriedades de acesso aos registradores da memória dos transdutores e determina se será um atuador ou se será um sensor com interrupções que atuam em outros transdutores.

A arquitetura do subsistema proposta é adequada, pois utiliza elementos presentes no sistema do CIERMag, e a utilização do *Raspberry Pi* permite o gerenciamento de que todos os dispositivos de todos os módulos definidos. Permite ainda estar conectado ao terminal computadorizado sem a necessidade de outros dispositivos para isso.

As interfaces desenvolvidas possuem um conjunto de funcionalidades que satisfazem as necessidades do usuário para criar, editar e gerenciar os TEDS, além de permitir a visualização gráfica das medições realizadas pelos sensores e possibilitar a modificação dos valores dos atuadores. A interface Python foi criada com o objetivo de ser integrada ao sistema para que o usuário possa utilizá-la, ao lado das demais ferramentas existentes, na configuração do sistema, bem como na preparação do experimento, enquanto a interface HTML foi criada como alternativa para acelerar o gerenciamento e configuração do subsistema.

5.1 Trabalhos futuros

Mesmo com as funcionalidades existentes, o subsistema ainda possui abertura para introdução de novas funcionalidades e aperfeiçoamento das já existentes:

- Possibilitar o subsistema a trabalhar com perfis de dispositivos de modo a acelerar o processo de reconfiguração do sistema para diferentes experimentos.

- Aperfeiçoar o sistema de aquisição de dados permitindo conversões lineares do tipo $y=a*x+b$ (introduzir um *off-set* na conversão).

- Aprimorar os procedimentos de interrupção para permitir ações em mais de um atuador de modo que o sistema possa ser desligado em casos críticos ou se autoconfigure, por exemplo.

- Permitir a importação e exportação de TEDS para dinamizar ainda mais a criação dos dispositivos.

- Acrescentar a visualização numérica das últimas medidas realizadas.

Essas e outras funcionalidades que podem surgir no decorrer do uso do subsistema agregam facilidades ao usuário e tornam o subsistema mais versátil e dinâmico.

REFERÊNCIAS

- 1 HAGE, M. C. F. N. S.; IWASAKI, M. Imagem por ressonância magnética: princípios básicos. **Ciência Rural**, v. 39, n. 4, p. 1275-1283, 2009. doi: 10.1590/S010384782009005000041.
- 2 BUXTON, R. B. **Introduction to functional magnetic resonance imaging: principles and techniques**. Cambridge: Cambridge University Press, 2002.
- 3 BLOCH, F.; HANSEN, W. W.; PACKARD, M. E. Nuclear induction. **Physical Review**, v.69, n. 3-4, p.127, 1946. doi: 10.1103/PhysRev.69.127.
- 4 STEKLY, Z. J. J. Large scale applications of permanent magnets. **IEEE Transactions on Applied Superconductivity**, v. 10, n. 1, p. 873-877, 2000.
- 5 TANNÚS, A.; PIZETTA, D. C.; SILVA, D. M. D. D.; VIDOTO, E. L. G.; MARTINS, M. J. **Subsistema multiplataforma para controle de aquisição, visualização e organização de dados de espectrômetro digital de RM: ToRM console**. Processo: BR512015001485-4. Data de registro: 07 dez. 2015. Revista da Propriedade Industrial - RPI n.2365, p.350, item 120. Registro de software.
- 6 SILVA, D. M. D. D.; PIZETTA, D. C.; FREIRE, G. M.; COELHO, F. B.; LOURENÇO, G. V.; CORRÊA, R. R. M.; MARTINS, M. J.; VIDOTO, E. L. G.; TANNÚS, A. Subsistema multiplataforma para controle de aquisição, visualização e organização de dados do Espectrômetro Digital de RM: ToRM console. **Revista Brasileira de Física Médica**, v. 7, n. 2, p. 35-40, 2013.
- 7 SILVA, D. M. D. D. **Desenvolvimento de console multiplataforma para aquisição, organização e visualização de dados do espectrômetro digital de RM do CIERMag: ToRM console**. 2014. 143 p. Dissertação (Mestrado em Ciências) – Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2014.
- 8 PIZETTA, D. C. **Biblioteca, API e IDE para o desenvolvimento de projetos de metodologias de ressonância magnética**. 2014. 93 p. Dissertação (Mestrado em Ciências) – Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2014.
- 9 THE RASPBERRY PI. **Raspberry Pi products**. Disponível em: <<https://www.raspberrypi.org/products/>>. Acesso em: 29 mar. 2014.
- 10 RASPBERRY Pi measurements and dimensions. Disponível em <<http://www.pyrosoft.co.uk/blog/tag/raspberry-pi/>>. Acesso em: 15 jun. 2016.
- 11 RPI HARDWARE. **Specifications**. Disponível em <http://elinux.org/RPi_Hardware>. Acesso em: 16 abr. 2016.
- 12 PYTHON programming language – official website. Disponível em: <<http://www.python.org/>>. Acesso em: 24 fev. 2014.

13 JSON encoder and decoder. Disponível em: <<https://docs.python.org/2/library/json.html>>. Acesso em: 3 jun. 2015.

14 THREADING. Higher-level threading interface. Disponível em: <<https://docs.python.org/2/library/threading.html>>. Acesso em: 22 abr. 2016.

15 TIME. Access and conversions. Disponível em: <<https://docs.python.org/2/library/time.html>>. Acesso em: 10 mar. 2014.

16 DATETIME. Basic date and time types. Disponível em: <<https://docs.python.org/2/library/datetime.html>>. Acesso em: 19 jan. 2016.

17 HTTPLIB. HTTP protocol client. Disponível em: <<https://docs.python.org/2/library/httplib.html>>. Acesso em: 8 out. 2015.

18 COPY: shallow and deep copy operations. Disponível em: <<https://docs.python.org/2/library/copy.html>>. Acesso em: 31 mar. 2016.

19 OS. Miscellaneous operating system interfaces. Disponível em: <<https://docs.python.org/2/library/os.html>>. Acesso em: 28 jan. 2016.

20 OS.PATH. Common pathname manipulations. Disponível em: <<https://docs.python.org/2/library/os.path.html>>. Acesso em: 19 jun. 2015.

21 GITHUB Python Spidev. Disponível em: <<https://github.com/doceme/py-spidev>>. Acesso em: 8 abr. 2014.

22 RASPBERRY-gpio-python. a Python module to control the GPIO on a Raspberry Pi. Disponível em: <<https://sourceforge.net/projects/raspberry-gpio-python>>. Acesso em: 14 abr. 2014.

23 PYQTGRAPH. Scientific graphics and GUI library for Python. Disponível em: <<http://www.pyqtgraph.org>>. Acesso em: 18 ago. 2014.

24 FLASK. Web development, one drop at a time. Disponível em: <<http://flask.pocoo.org>>. Acesso em: 16 out. 2015.

25 BOKEH. **Welcome to Bokeh**. Disponível em: <<http://bokeh.pydata.org/en/latest>>. Acesso em: 10 mai. 2016.

26 THE INTERNET ENGINEERING TASK FORCE (IETF). RFC7230: HTTP/1.1 message syntax and routing. Disponível em: <<http://httpwg.org/specs/rfc7230.html>>. Acesso em: 18 jun. 2016.

27 THE INTERNET ENGINEERING TASK FORCE (IETF). RFC7231: HTTP/1.1 Semantics and Content. Disponível em: <<http://httpwg.org/specs/rfc7231.html>>. Acesso em: 18 jun. 2016.

28 COMER, D. **Internetworking with TCP/IP**. 4ª ed. Englewood Cliffs: N.J. Prentice Hall. 1995. cap. 11, p. 183-189.

29 SERIAL PERIPHERAL INTERFACE BUS. Disponível em: <https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus>. Acesso em: 4 jun. 2016.

30 MAYANK. **Inter integrated circuit** – I2C basics. Disponível em: <<http://maxembedded.com/2014/02/inter-integrated-circuits-i2c-basics>>. Acesso em: 4 jun. 2016.

31 INSTRUMENTATION AND MEASUREMENT SOCIETY. **IEEE Std 1451.0™-2007**: IEEE standard for a smart transducer interface for sensors and actuators-common functions, communication protocols, and transducer electronic data sheet (TEDS) formats. Disponível em: <<http://standards.ieee.org/findstds/standard/1451.0-2007.html>>. Acesso em: 16 jun. 2016.

32 GPIO. Raspberry Pi models A and B. Disponível em: <<https://www.raspberrypi.org/documentation/usage/gpio>>. Acesso em: 11 jun. 2016.

33 CIRCUITAR. Nanoshield termopar. Disponível em: <<https://www.circuitar.com.br/loja/produto/158/>>. Acesso em 17 jun. 2016.

ANEXO A – DATASHEET MAX31856

EVALUATION KIT AVAILABLE

MAX31856**Precision Thermocouple to Digital Converter
with Linearization****General Description**

The MAX31856 performs cold-junction compensation and digitizes the signal from any type of thermocouple. The output data is formatted in degrees Celsius. This converter resolves temperatures to 0.0078125°C, allows readings as high as +1800°C and as low as -210°C (depending on thermocouple type), and exhibits thermocouple voltage measurement accuracy of $\pm 0.15\%$. The thermocouple inputs are protected against overvoltage conditions up to $\pm 45\text{V}$.

A lookup table (LUT) stores linearity correction data for several types of thermocouples (K, J, N, R, S, T, E, and B). Line frequency filtering of 50Hz and 60Hz is included, as is thermocouple fault detection. A SPI-compatible interface allows selection of thermocouple type and setup of the conversion and fault detection processes.

Applications

- Temperature Controllers
- Industrial Ovens, Furnaces, and Environmental Chambers
- Industrial Equipment

Ordering Information appears at end of data sheet.

Benefits and Features

- Provides High-Accuracy Thermocouple Temperature Readings
 - Includes Automatic Linearization Correction for 8 Thermocouple Types
 - $\pm 0.15\%$ (max, -20°C to +85°C) Thermocouple Full-Scale and Linearity Error
 - 19-Bit, 0.0078125°C Thermocouple Temperature Resolution
- Internal Cold-Junction Compensation Minimizes System Components
 - $\pm 0.7^\circ\text{C}$ (max, -20°C to +85°C) Cold-Junction Accuracy
- $\pm 45\text{V}$ Input Protection Provides Robust System Performance
- Simplifies System Fault Management and Troubleshooting
 - Detects Open Thermocouples
 - Over- and Undertemperature Fault Detection
- 50Hz/60Hz Noise Rejection Filtering Improves System Performance
- 14-Pin TSSOP Package

Typical Application Circuit

MAX31856

Precision Thermocouple to Digital Converter
with Linearization

Absolute Maximum Ratings

AVDD, DVDD.....-0.3V to +4.0V
 T+, T-, Bias.....±45V
 T+, T-, Bias.....±20mA
 All Other Pins.....-0.3V to (V_{DD} + 0.3V)
 Continuous Power Dissipation (T_A = +70°C)
 TSSOP (derate 9.1mW/°C above +70°C).....727.3mW
 ESD Protection (All pins, Human Body Model).....2000V

Operating Temperature Range.....-55°C to +125°C
 Junction Temperature.....+150°C
 Storage Temperature Range.....-65°C to +150°C
 Lead Temperature (soldering, 10s).....+300°C
 Soldering Temperature
 (reflow).....See IPC/JEDEC J-STD-020A Specification

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Package Thermal Characteristics (Note 1)

TSSOP

Junction-to-Ambient Thermal Resistance (θ_{JA}) 110°C/WJunction-to-Case Thermal Resistance (θ_{JC}) 30°C/W

Note 1: Package thermal resistances were obtained using the method described in JEDEC specification JESD51-7, using a four-layer board. For detailed information on package thermal considerations, refer to www.maximintegrated.com/thermal-tutorial.

Recommended DC Operating Conditions

(T_A = -55°C to +125°C, unless otherwise noted.) (Notes 2 and 4)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Power-Supply Voltage	V _{AVDD} , V _{DVDD}		3.0	3.3	3.6	V
AVDD-DVDD			-100		+100	mV
Cable Resistance	R _{CABLE}	Per lead			40	kΩ
Input Logic 0	V _{IL}				0.8	V
Input Logic 1	V _{IH}		2.1			V

Electrical Characteristics

(3.0V ≤ V_{DD} ≤ 3.6V, T_A = -55°C to +125°C, unless otherwise noted.) (Notes 2, 3, and 4)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Supply Current	I _{DD}	Standby		5.25	10	μA
		Active conversion		1.2	2	mA
Thermocouple Temperature Resolution				19		Bits
				0.0078125		°C
Cold-Junction Temperature Data Resolution				0.015625		°C
Thermocouple Input Bias Current	I _{TCBIAS}	T _A = +25°C	-10		+10	nA
		T _A = -40°C to +85°C	-10		+65	
		T _A = -55°C to +105°C	-20		+110	
		T _A = -55°C to +125°C	-20		+400	

MAX31856

Precision Thermocouple to Digital Converter
with Linearization

Electrical Characteristics (continued)

(3.0V ≤ V_{DD} ≤ 3.6V, T_A = -55°C to +125°C, unless otherwise noted.) (Notes 2, 3, and 4)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Thermocouple Input Differential Bias Current (Note 4)	I _{TCIDBIAS}	T _A = +25°C		±0.2		nA
		T _A = -40°C to +85°C	-4		+4	
		T _A = -55°C to +105°C	-5.5		+5.5	
		T _A = -55°C to +125°C	-10		+10	
Input-Referred Noise	V _N	AV = 8		1.3		μV _{RMS}
		AV = 32		0.4		
Power-Supply Rejection	PSR	Cold-junction sensor		0.15		°C/V
Power-On-Reset Voltage Threshold	V _{POR}			2.7	2.85	V
Power-On-Reset Voltage Hysteresis	V _{HYST}			0.1		V
Bias Voltage	V _{BIAS}			0.735		V
BIAS Output Resistance	R _{BIAS}			2		kΩ
Input Common-Mode Range			0.5		1.4	V
Full-Scale and INL Error (Note 6)		T _A = +25°C	-0.05		+0.05	%FS
		T _A = -20°C to +85°C	-0.15		+0.15	
		T _A = -40°C to +105°C	-0.2		+0.2	
		T _A = -40°C to +125°C	-0.3		+0.3	
		T _A = -55°C to +125°C	-0.35		+0.35	
Input Offset Voltage (Note 7)		T _A = +25°C	-0.01		+0.01	%FS
		T _A = -20°C to +85°C	-0.015		+0.015	
		T _A = -40°C to +105°C	-0.017		+0.017	
		T _A = -55°C to +125°C	-0.02		+0.02	
Input Offset Voltage	AV = 8	T _A = +25°C	-7.8		+7.8	μV
		T _A = -20°C to +85°C	-11.7		+11.7	
		T _A = -40°C to +105°C	-13.3		+13.3	
		T _A = -55°C to +125°C	-15.6		+15.6	
	AV = 32	T _A = +25°C	-2.0		+2.0	
		T _A = -20°C to +85°C	-2.9		+2.9	
		T _A = -40°C to +105°C	-3.3		+3.3	
		T _A = -55°C to +125°C	-3.9		+3.9	
Cold-Junction Temperature Error		T _A = -20°C to +85°C	-0.7		+0.7	°C
		T _A = -40°C to +105°C	-1		+1	
		T _A = -55°C to +125°C	-2		+2	
Overvoltage Rising Threshold (Note 8)			V _{AVDD} - 0.1	V _{AVDD} + 0.17	V _{AVDD} + 0.35	V
Overvoltage Hysteresis				0.09		V

MAX31856

Precision Thermocouple to Digital Converter
with Linearization

Electrical Characteristics (continued)

(3.0V ≤ V_{DD} ≤ 3.6V, T_A = -55°C to +125°C, unless otherwise noted.)(Notes 2, 3, and 4)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Undervoltage Falling Edge Threshold (Note 8)			-0.3	-0.17	0	V
Undervoltage Hysteresis				0.09		V
Thermocouple Linearity Correction Error		Type B, T _A = 0 to 125°C, T _{TC} = 95°C to +1798°C	-0.24		+0.25	°C
		Type E, T _A = -55°C to +125°C T _{TC} = -200°C to +1000°C	-0.14		+0.06	
		Type J, T _A = -55°C to +125°C T _{TC} = -210°C to +1200°C	-0.11		+0.10	
		Type K, T _A = -55°C to +125°C T _{TC} = -200°C to +1372°C	-0.13		+0.12	
		Type N, T _A = -55°C to +125°C T _{TC} = -200°C to +1300°C	-0.09		+0.08	
		Type R, T _A = -50°C to +125°C T _{TC} = -50°C to +1768°C	-0.19		+0.17	
		Type S, T _A = -50°C to +125°C T _{TC} = -50°C to +1768°C	-0.16		+0.20	
		Type T, T _A = -55°C to +125°C T _{TC} = -200°C to +400°C	-0.07		+0.07	
Temperature Conversion Time (Thermocouple + Cold Junction)	t _{CONV}	1-Shot conversion or first conversion in auto-conversion mode (60Hz)		143	155	ms
		1-Shot conversion or first conversion in auto-conversion mode (50Hz)		169	185	
		Auto conversion mode, conversions 2 through n (60Hz)		82	90	
		Auto conversion mode, conversions 2 through n (50Hz)		98	110	

MAX31856

Precision Thermocouple to Digital Converter
with Linearization

Electrical Characteristics (continued)

(3.0V ≤ V_{DD} ≤ 3.6V, T_A = -55°C to +125°C, unless otherwise noted.)(Notes 2, 3, and 4)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Common-Mode Rejection	CMR	0.5V ≤ V _{CM} ≤ 1.4V		70		dB
50/60Hz Noise Rejection		Fundamental and harmonics		91		dB
SERIAL INTERFACE						
Input Leakage Current	I _{LEAK}	(Note 5)	-1		+1	μA
Output High Voltage	V _{OH}	I _{OUT} = -1.6mA	V _{CC} - 0.4			V
Output Low Voltage	V _{OL}	I _{OUT} = 1.6mA			0.4	V
Input Capacitance	C _{IN}			8		pF
Serial Clock Frequency	f _{SCL}				5	MHz
SCK Pulse High Width	t _{CH}		100			ns
SCK Pulse Low Width	t _{CL}		100			ns
SCK Rise and Fall Time	t _R , t _F	C _L = 10pF			200	ns
CS Fall to SCK Rise	t _{CC}	C _L = 10pF	100			ns
SCK to CS Hold	t _{CCH}	C _L = 10pF	100			ns
CS Rise to Output Disable	t _{CDZ}	C _L = 10pF			40	ns
Data to SCLK Setup	t _{DC}		35			ns
SCLK to Data Hold	t _{CDH}		35			ns
SCK Fall to Output Data Valid	t _{CDD}	C _L = 10pF			80	ns
CS Inactive Time	t _{CWH}	(Note 3)	400			ns

Note 2: All voltages are referenced to GND. Currents entering the IC are specified positive, and currents exiting the IC are negative.**Note 3:** All Serial Interface timing specifications are guaranteed by design.**Note 4:** Specification is 100% tested at T_A = +25°C. Specification limits over temperature (T_A = T_{MIN} to T_{MAX}) are guaranteed by design and characterization; not production tested.**Note 5:** For all pins except T⁺ and T⁻ (see the Thermocouple Input Bias Current parameter in the [Electrical Characteristics](#) table.**Note 6:** Using a common-mode voltage other than V_{BIAS} will change this specification. See the [Typical Operating Characteristics](#) for details.**Note 7:** Input-referred full-scale voltage is 78.125mV when AV = 8 and is 19.531mV when AV = 32.**Note 8:** Overvoltage and undervoltage limits apply to T⁺, T⁻, and BIAS pins.

MAX31856

Precision Thermocouple to Digital Converter
with Linearization

Figure 1. Timing Diagram: SPI Read Data Transfer

Figure 2. Timing Diagram: SPI Write Data Transfer

MAX31856

Precision Thermocouple to Digital Converter
with Linearization

Typical Operating Characteristics

(V_{CC} = 3.3V and T_A = +25°C, unless otherwise noted.)

Typical Operating Characteristics (continued)

(V_{CC} = 3.3V and T_A = +25°C, unless otherwise noted.)

MAX31856

Precision Thermocouple to Digital Converter
with Linearization

Typical Operating Characteristics (continued)

(V_{CC} = 3.3V and T_A = +25°C, unless otherwise noted.)

MAX31856

Precision Thermocouple to Digital Converter
with Linearization

Pin Configuration

Pin Description

PIN	NAME	FUNCTION
1	AGND	Analog Ground
2	BIAS	Bias Voltage Source. Nominally 0.735V. This pin is floating when no conversions are taking place.
3	T-	Thermocouple Negative Input. See Table 1 .
4	T+	Thermocouple Positive Input. See Table 1 .
5	AVDD	Analog Positive Supply. Bypass with a 0.1μF capacitor to AGND.
6	DNC	Do Not Connect
7	DRDY	Data Ready Output
8	DVDD	Digital Positive Supply. Bypass with a 0.1μF capacitor to DGND.
9	CS	Chip Select. Set CS low to enable the serial interface.
10	SCK	Serial Clock Input
11	SDO	Serial Data Output
12	SDI	Serial Data Input
13	FAULT	Cable, thermocouple, or temperature fault output
14	DGND	Digital Ground

MAX31856

Precision Thermocouple to Digital Converter
with Linearization

Block Diagram

Detailed Description

The MAX31856 is a sophisticated thermocouple-to-digital converter with a built-in 19-bit analog-to-digital converter (ADC). Internal functions include correction for thermocouple nonlinearity, input protection, cold-junction compensation sensing and correction, a digital controller, a SPI-compatible interface, and associated control logic.

In the simplest configuration, the thermocouple wires connect directly to inputs T- and T+, with a common-mode bias voltage provided by the BIAS output. Additional filtering and/or protection components may be added if needed, as discussed in the [Applications Information](#) section. Operation is controlled by two configuration bytes and four bytes that contain over- and undertemperature detection thresholds.

Temperature Conversion

The temperature conversion process consists of five steps as described in the sections below. The input amplifier and ADC amplify and digitize the thermocouple's voltage output. The internal temperature sensor measures the cold-junction temperature. Using the internal lookup table (LUT), the ADC code corresponding to the cold-junction temperature for the selected thermocouple type is determined. The thermocouple code and the cold-junc-

tion code are summed to produce the code corresponding to the cold-junction compensated thermocouple temperature. Finally, the LUT is used to produce a cold-junction compensated output code in units of °C.

Thermocouple Voltage Conversion

T+ and T- are the thermocouple inputs. T- is biased to approximately 0.735V by the BIAS output. The amplifier provides gain to the μV - and mV -level thermocouple signals to make the amplitude appropriate for the ADC's full-scale input range. Two amplifier gains provide full-scale input ranges of $\pm 78.125\text{mV}$ and $\pm 19.531\text{mV}$ to accommodate higher- and lower-sensitivity thermocouples.

Because long thermocouple wires may pick up noise from a variety of sources, including AC power cables, the amplified signal is lowpass filtered before being applied to the ADC. The ADC provides further digital lowpass and notch filtering to attenuate input noise. The notch frequencies are either 50Hz and its harmonics or 60Hz and its harmonics, selectable using bit 0 of the Configuration 0 register (00h). In addition, bits D6:4 of the Configuration 1 register (01h) enable an averaging mode that provides additional filtering with an associated increase in conversion time. 2, 4, 8, or 16 samples may be averaged using this mode.

MAX31856

Precision Thermocouple to Digital Converter
with Linearization

The conversion mode can be either continuous or “normally off”, as selected by bit 7 of the Configuration 0 register (00h). When in the normally off mode, a single “1-shot” conversion may be selected using bit 6 of the Configuration 0 register (00h).

Thermocouple type is user-selectable using bits D3:0 of the Configuration 1 register (01h). Thermocouple types K, J, N, R, S, T, B, and E are supported by automatic cold-junction compensation and linearization. (To use a different thermocouple type, use bits D3:0 to select a gain of either 8 or 32. The linearization and cold-junction compensation calculations may then be done externally using the cold-junction temperature and thermocouple voltage data.)

Cold-Junction Temperature Sensing

The function of the thermocouple is to sense a difference in temperature between two ends of the thermocouple wires. The thermocouple's sensing junction (often called the “hot” junction regardless of its temperature) can be measured across its rated operating temperature range (see [Table 1](#) for supported thermocouple temperature ranges).

Additional thermocouples are created where the thermocouple wires make contact with different metals, usually at a connector or at the point where they are soldered to a PCB (the “cold junction”). To compensate for the errors due to these additional thermocouples, the temperature at

the cold junction must be measured. This is done with the internal precision temperature sensor, which has accuracy better than $\pm 0.7^{\circ}\text{C}$ from -20°C to $+85^{\circ}\text{C}$. By placing the MAX31856 near the cold junction, the cold-junction temperature can be measured and used to compensate for cold-junction effects.

The MAX31856 stores the cold-junction temperature data in registers 0Ah and 0Bh. When the cold-junction temperature sensor is enabled, these registers are read-only and contain the measured cold-junction temperature plus the value in the Cold-Junction Offset register. Reading the register with the cold-junction temperature sensor enabled will reset the DRDY pin high. Both bytes of this register should be read as a multibyte transfer to ensure both bytes are from the same temperature update. When the cold-junction temperature sensor is disabled, these registers become read-write registers that contain the most recent measured temperature value. If desired, data from an external temperature sensor may be written to these registers when the internal cold-junction sensor is disabled. The maximum cold-junction temperature is clamped at 128°C and the minimum is clamped at -64°C . See [Table 2](#) for the Reference Junction (Cold Junction) Temperature Data Format.

If desired, a temperature offset may be written to the Cold-Junction Offset register (09h). The value stored in registers 0Ah and 0Bh will then be equal to the measured

Table 1. Supported Thermocouples and Temperature Ranges

TYPE	T-WIRE	T+ WIRE	TEMP RANGE	NOMINAL SENSITIVITY ($\mu\text{V}/^{\circ}\text{C}$)	COLD-JUNCTION TEMP RANGE
B	Platinum/Rhodium	Platinum/Rhodium	250°C to 1820°C	10.086 ($+500^{\circ}\text{C}$ to $+1500^{\circ}\text{C}$)	0 to 125°C
E	Constantan	Chromel	-200°C to $+1000^{\circ}\text{C}$	76.373 (0°C to $+1000^{\circ}\text{C}$)	-55°C to $+125^{\circ}\text{C}$
J	Constantan	Iron	-210°C to $+1200^{\circ}\text{C}$	57.953 (0°C to $+750^{\circ}\text{C}$)	-55°C to $+125^{\circ}\text{C}$
K	Alumel	Chromel	-200°C to $+1372^{\circ}\text{C}$	41.276 (0°C to $+1000^{\circ}\text{C}$)	-55°C to $+125^{\circ}\text{C}$
N	Nisil	Nicrosil	-200°C to $+1300^{\circ}\text{C}$	36.256 (0°C to $+1000^{\circ}\text{C}$)	-55°C to $+125^{\circ}\text{C}$
R	Platinum	Platinum/Rhodium	-50°C to $+1768^{\circ}\text{C}$	10.506 (0°C to $+1000^{\circ}\text{C}$)	-50°C to $+125^{\circ}\text{C}$
S	Platinum	Platinum/Rhodium	-50°C to $+1768^{\circ}\text{C}$	9.587 (0°C to $+1000^{\circ}\text{C}$)	-50°C to $+125^{\circ}\text{C}$
T	Constantan	Copper	-200°C to $+400^{\circ}\text{C}$	52.18 (0°C to $+400^{\circ}\text{C}$)	-55°C to $+125^{\circ}\text{C}$

MAX31856

Precision Thermocouple to Digital Converter
with Linearization

value plus the offset value. The MSB of the offset register is 4°C and the LSB is 0.0625°C. The resulting range of the offset value applied to the measured CJ temperature is -8°C to +7.9375°C. The default offset value is 0°C (00h).

Optimal performance is achieved when the thermocouple cold junction and the cold-junction sensor are at the same temperature. Avoid placing heat-generating devices or components near the cold junction because this may produce cold-junction-related errors. When a significant temperature differential between the internal sensor and the cold junction is unavoidable, an external temperature sensor may be used instead. The temperature measured by the external sensor may be written to the cold-junction temperature register and used for cold-junction compensation. Bit 3 of Configuration 0 register (00h) disables the internal cold-junction temperature sensor and allows temperature values from an external sensor to be written directly into the Cold-Junction Temperature registers (0Ah and 0Bh).

Cold-Junction Temperature Translation and Compensation

Thermocouple temperature values and corresponding ADC codes are stored in an internal lookup table. After measuring the cold-junction temperature, the LUT is used to convert the temperature value to the equivalent ADC code for the type of thermocouple being used. Values between LUT entries are interpolated. The cold-junction ADC code is added to the conversion result in the thermocouple voltage register to yield a cold-junction-compensated value.

**Table 2. Reference Junction
(Cold-Junction) Temperature Data Format**

TEMPERATURE (°C)	DIGITAL OUTPUT
+127.984375	0111 1111 1111 1100
+127	0111 1111 0000 0000
+125	0111 1101 0000 0000
+64	0100 0000 0000 0000
+25	0001 1001 0000 0000
+0.5	0000 0000 1000 0000
+0.015625	0000 0000 0000 0100
0	0000 0000 0000 0000
-0.5	1111 1111 1000 0000
-25	1110 0111 0000 0000
-55	1100 1001 0000 0000

Thermocouple Linearization and Conversion of Code to Temperature

Because all thermocouples are nonlinear, the raw cold-junction-compensated value must be corrected for non-linearity and converted to a temperature value. This is done using the LUT to produce the linearized and cold-junction-compensated temperature value, which is stored after every conversion as 19 bits in the Linearized Thermocouple Temperature registers (0Ch, 0Dh, and 0Eh). All three bytes should be read as a multibyte transfer to ensure all are from the same data update. See [Table 3](#) for the Linearized Thermocouple Temperature Data Format.

Linearization accuracy varies by thermocouple type, "hot-junction" temperature, and cold-junction temperature, with the largest errors typically occurring near the hot-junction and cold-junction extremes. Worst-case values for linearization errors are shown in the [Electrical Characteristics](#) table.

Over-/Undertemperature Fault Detection

Over- and undertemperature fault detection are available for both the cold-junction temperature and the linearized and cold-junction-compensated temperature reading. Two registers (03h and 04h) contain the high and low thresholds for the cold-junction temperature. The cold-junction temperature value in registers 0Ah and 0Bh is compared to the threshold values. If a threshold is exceeded, the corresponding bit is set in the Fault Status register (0Fh) and, if not masked, the FAULT output will assert.

**Table 3. Linearized Thermocouple
Temperature Data Format**

TEMPERATURE (°C)	DIGITAL OUTPUT
+1600.00	0110 0100 0000 0000 0000 0000
+1000.00	0011 1110 1000 0000 0000 0000
+100.9375	0000 0110 0100 1111 0000 0000
+25.00	0000 0001 1001 0000 0000 0000
+0.0625	0000 0000 0000 0001 0000 0000
0.00	0000 0000 0000 0000 0000 0000
-0.0625	1111 1111 1111 1111 0000 0000
-0.25	1111 1111 1111 1100 0000 0000
-1.00	1111 1111 1111 0000 0000 0000
-250.00	1111 0000 0110 0000 0000 0000

This format also applies to the High Fault and Low Fault thresholds.

(Note that the practical temperature range varies with the thermocouple type.)

MAX31856

Precision Thermocouple to Digital Converter
with Linearization

Four registers (05h through 08h) contain over- and under-temperature thresholds for the linearized and cold-junction-compensated temperature. These threshold register values are compared to the linearized temperature reading found in registers 0Ch, 0Dh, and 0Eh. If a threshold is exceeded, the corresponding bit is set in the Fault Status register (0Fh) and, if not masked, the $\overline{\text{FAULT}}$ output will assert.

Integrated Input Protection

The internal circuitry is protected from excessive voltages applied to the thermocouple cables by integrated MOSFETs at the T+ and T- inputs, and the BIAS output. These MOSFETs turn off when the input voltage is negative or greater than V_{DD} . The MOSFETs are capable of withstanding input voltages up to $\pm 45V$. If fault voltages beyond the $\pm 45V$ limits are expected, see the [Applications Information](#) section.

When the absolute input voltage at T+ or T- is negative or greater than V_{DD} , the Under-/Overvoltage Fault bit, Bit 1, is set in the Fault Status register (0Fh) and the $\overline{\text{FAULT}}$ pin asserts if not masked. Conversions are suspended while the OVUV fault is present and will resume when the fault is removed.

Open-Circuit Fault Detection

Detection of open-circuit faults, such as those caused by broken thermocouple wires, can be enabled or disabled using bits 4 and 5 in the Configuration 0 register (00h). Fault detection is accomplished by forcing a small

current through the thermocouple wires. The time required to detect an open circuit depends on the values of the lead resistances and any filter capacitance at the thermocouple input and therefore, bits 4 and 5 also select the time allowed for open-circuit fault detection. A nominal detection time of either 10ms, 32ms, or 100ms can be selected. The Open-Circuit Detection Mode table ([Table 4](#)) shows the effect of these two bits on the conversion time. When the device is in one-shot mode, open-circuit detection can be disabled or set to occur every one-shot conversion. When the device is in automatic conversion mode, open-circuit detection may be disabled, or it may be set to automatically test for open circuits every 16 conversion cycles. If on-demand detection is desired, select "detection disabled" (00), then select the setting for the desired time constant. An open-circuit detection test will be performed immediately after the current conversion is completed. Disabling the open fault detection when in comparator mode while there is an open fault present will not clear the fault bit or $\overline{\text{FAULT}}$ pin. If this happens, to subsequently clear the fault, the MAX31856 must be placed in interrupt mode and then the fault cleared. Note that, when cold-junction sensing is enabled, open-circuit fault detection and cold-junction sensing occur concurrently. Therefore, cold-junction temperature sensing has no effect on the overall cycle time when open-circuit fault detection is enabled. An open-circuit fault is indicated by the Open Fault bit, Bit 0, in the Fault Status register (0Fh) and the $\overline{\text{FAULT}}$ pin asserts if not masked.

Table 4. Open-Circuit Detection Mode

BITS 5:4 OCFAULT1: OCFAULT0 (Config Byte 0)	FAULT TEST	INPUT NETWORK	FAULT TEST TIME (ms)			
			CJ SENSE ENABLED		CJ SENSE DISABLED	
			TYP	MAX	TYP	MAX
00	Disabled	N/A	0	0	0	0
01	Enabled (Once every 16 conversions)	$R_S < 5k\Omega$	13.3	15	40	44
10	Enabled (Once every 16 conversions)	$40k\Omega > R_S > 5k\Omega$; Time constant $< 2ms$	33.4	37	60	66
11	Enabled (Once every 16 conversions)	$40k\Omega > R_S > 5k\Omega$; Time constant $> 2ms$	113.4	125	140	154

MAX31856

Precision Thermocouple to Digital Converter
with Linearization**Cold-Junction and Thermocouple
Out-of-Range Detection**

Thermocouple characteristics, the measurement circuitry, and the linearization calculations limit the optimum temperature ranges for both the cold junction and the measurement junction ("hot junction"). Bit D7 of the Fault Status register indicates when the cold-junction temperature falls outside of the optimum range, and bit D6 indicates when the hot-junction temperature is out of range. Table 1 shows the temperature limits that apply for the supported thermocouple types. These values are rounded to the nearest °C. When the temperature falls outside of the limit for a given measurement, the reported thermocouple temperature is clamped at the limit value. Note that the FAULT pin never asserts for an out-of-range fault.

Serial Interface

Four pins are used for SPI-compatible communications: SDO (serial-data out), SDI (serial-data in), $\overline{CS}$ (chip select), and SCLK (serial clock). SDI and SDO are the serial-data input and output pins, respectively. The $\overline{CS}$ input initiates and terminates a data transfer. SCLK synchronizes data movement between the master (microcontroller) and the slave (MAX31856).

The serial clock (SCLK), which is generated by the microcontroller, is active only when $\overline{CS}$ is low and during address and data transfer to any device on the SPI bus. The inactive clock polarity is programmable in some microcontrollers. The MAX31856 automatically accommodates either clock polarity by sampling SCLK when $\overline{CS}$ becomes active to determine the polarity of the inactive clock. Input data (SDI) is latched on the internal strobe edge and output data (SDO) is shifted out on the shift

edge (see Table 5 and Figure 3). There is one clock for each bit transferred. Address and data bits are transferred in groups of eight, MSB first.

Address and Data Bytes

Address and data bytes are shifted MSB-first into the serial-data input (SDI) and out of the serial-data output (SDO). Any transfer requires the address of the byte to specify a write or a read, followed by one or more bytes of data. Data is transferred out of the SDO for a read operation and into the SDI for a write operation. The address byte is always the first byte transferred after $\overline{CS}$ is driven low. The MSB (A7) of this byte determines whether the following byte will be written or read. If A7 is 0, one or more byte reads will follow the address byte. If A7 is 1, one or more byte writes will follow the address byte.

For a single-byte transfer, 1 byte is read or written and then $\overline{CS}$ is driven high (see Figure 4 and Figure 5). For a multiple-byte transfer, multiple bytes can be read or written after the address has been written (see Figure 6). The address continues to increment through all memory locations as long as $\overline{CS}$ remains low. If data continues to be clocked in or out, the address will loop from 7Fh/FFh to 00h/80h. Invalid memory addresses report an FFh value. Attempting to write to a read-only register will result in no change to that register's contents.

DRDY

The $\overline{DRDY}$ output goes low when a new conversion result is available in the Linearized Thermocouple Temperature register. When a read-operation of the Linearized Thermocouple Temperature register or the Cold-Junction Temperature Register (if enabled) completes, $\overline{DRDY}$ returns high.

Table 5. Serial Interface Function

MODE	$\overline{CS}$	SCLK	SDI	SDO
Disable Reset	High	Input Disabled	Input disabled	High impedance
Write	Low	CPOL = 1*, SCLK rising	Data bit latch	High impedance
		CPOL = 0, SCLK falling		
Read	Low	CPOL = 1, SCLK falling	X	Next data bit shift**
		CPOL = 0, SCLK rising		

Note: CPHA bit polarity must be set to 1.

*CPOL is the clock polarity bit that is set in the control register of the microcontroller.

**SDO remains at high impedance until 8 bits of data are ready to be shifted out during a read.

MAX31856

Precision Thermocouple to Digital Converter
with Linearization

Figure 3. Serial Clock as a Function of Microcontroller Clock Polarity (CPOL)

Figure 4. SPI Single-Byte Read

Figure 5. SPI Single-Byte Write

MAX31856

Precision Thermocouple to Digital Converter
with Linearization

Figure 6. SPI Multibyte Transfer

Figure 7. DRDY Operation

MAX31856

Precision Thermocouple to Digital Converter
with Linearization

Internal Registers

Communication with the MAX31856 is accomplished through 16 8-bit registers that contain conversion, status, and configuration data. All programming is done by selecting the appropriate address of the desired register location. The Register Memory Map (Table 6) illustrates the addresses for the temperature, status, and configuration registers.

The registers are accessed using the 0Xh addresses for reads and the 8Xh addresses for writes. Data is read from or written to the registers MSB first. Attempts to write to a read-only register results in no change in the data.

Table 6. Register Memory Map

ADDRESS	READ/WRITE	NAME	FACTORY DEFAULT	FUNCTION
00h/80h	Read/Write	CR0	00h	Configuration 0 Register
01h/81h	Read/Write	CR1	03h	Configuration 1 Register
02h/82h	Read/Write	MASK	FFh	Fault Mask Register
03h/83h	Read/Write	CJHF	7Fh	Cold-Junction High Fault Threshold
04h/84h	Read/Write	CJLF	C0h	Cold-Junction Low Fault Threshold
05h/85h	Read/Write	LTHFTH	7Fh	Linearized Temperature High Fault Threshold MSB
06h/86h	Read/Write	LTHFTL	FFh	Linearized Temperature High Fault Threshold LSB
07h/87h	Read/Write	LTLFTH	80h	Linearized Temperature Low Fault Threshold MSB
08h/88h	Read/Write	LTLFTL	00h	Linearized Temperature Low Fault Threshold LSB
09h/89h	Read/Write	CJTO	00h	Cold-Junction Temperature Offset Register
0Ah/8Ah	Read/Write	CJTH	00h	Cold-Junction Temperature Register, MSB
0Bh/8Bh	Read/Write	CJTL	00h	Cold-Junction Temperature Register, LSB
0Ch	Read Only	LTCBH	00h	Linearized TC Temperature, Byte 2
0Dh	Read Only	LTCBM	00h	Linearized TC Temperature, Byte 1
0Eh	Read Only	LTCBL	00h	Linearized TC Temperature, Byte 0
0Fh	Read Only	SR	00h	Fault Status Register

Register 00h/80h: Configuration 0 Register (CR0)

The Configuration 0 register selects the conversion mode (automatic or triggered by the 1-shot command), selects open-circuit fault detection timing, enables the cold-junction sensor, clears the fault status register, and selects the filter notch frequencies. The effects of the configuration bits are described below.

Default Value: 00h

MEMORY AC- CESS	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
00h/80h	CMODE	1SHOT	OCFAULT1	OCFAULT0	CJ	FAULT	FAULTCLR	50/60Hz
	Bit 7						Bit 0	

MAX31856

Precision Thermocouple to Digital Converter
with Linearization

Register 00h/80h: Configuration 0 Register (CR0) (continued)

BIT	NAME	DESCRIPTION
7	CMODE	Conversion Mode 0 = Normally Off mode (default) 1 = Automatic Conversion mode. Conversions occur continuously every 100ms (nominal).
6	1SHOT	One-Shot Mode 0 = No conversions requested (default) 1 = This causes a single cold-junction and thermocouple conversion to take place when Conversion Mode bit = 0 (normally off mode). The conversion is triggered when $\overline{CS}$ goes high after writing a 1 to this bit. Note that if a multi-byte write is performed, the conversion is triggered when $\overline{CS}$ goes high at the end of the transaction. A single conversion requires approximately 143ms in 60Hz filter mode or 169ms in 50Hz filter mode to complete. This bit self clears to 0.
5:4	OCFAULT[1:0]	These bits enable/disable open-circuit fault detection and select fault detection timing. See Open-Circuit Fault Detection section and Table 4 for operation of these bits.
3	CJ	Cold-Junction Sensor Disable 0 = Cold-junction temperature sensor enabled (default) 1 = Cold-junction temperature sensor disabled. Data from an external temperature sensor may be written to the cold-junction temperature register. When this bit changes from 0 to 1, the most recent cold-junction temperature value will remain in the cold-junction temperature register until the internal sensor is enabled or until a new value is written to the register. The overall temperature conversion time is reduced by 25ms (typ) when this bit is set to 1.
2	FAULT	Fault Mode 0 = Comparator Mode. The $\overline{FAULT}$ output and respective fault bit reflects the state of any non-masked faults by asserting when the fault condition is true, and deasserting when the fault condition is no longer true. There is a 2°C hysteresis when in comparator mode for threshold fault conditions. (default) 1 = Interrupt Mode. The $\overline{FAULT}$ output and respective fault bit asserts when a non-masked fault condition is true and remain asserted until a 1 is written to the Fault Status Clear bit. This deasserts $\overline{FAULT}$ and respective fault bit until a new fault is detected (note that this may occur immediately if the fault condition is still in place).
1	FAULTCLR	Fault Status Clear 0 = Default 1 = When in interrupt mode, returns all Fault Status bits [7:0] in the Fault Status Register (0Fh) to 0 and deasserts the $\overline{FAULT}$ output. This bit has no effect in comparator mode. Note that the $\overline{FAULT}$ output and the fault bit may reassert immediately if the fault persists. To prevent the $\overline{FAULT}$ output from reasserting, first set the Fault Mask bits. The fault status clear bit self-clears to 0.
0	50/60Hz	50Hz/60Hz Noise Rejection Filter Selection 0 = Selects rejection of 60Hz and its harmonics (default) 1 = Selects rejection of 50Hz and its harmonics Note: Change the notch frequency only while in the "Normally Off" mode – not in the Automatic Conversion mode.

MAX31856

Precision Thermocouple to Digital Converter
with Linearization**Register 01h/81h: Configuration 1 Register (CR1)**

The Configuration 1 register selects the averaging time for the thermocouple voltage conversion averaging mode and also selects the thermocouple type being monitored.

Default Value: 03h

MEMORY ACCESS	N/A	R/W	R/W	R/W	R/W	R/W	R/W	R/W
01h/81h	Reserved	AVGSEL ₂	AVGSEL ₁	AVGSEL ₀	TC TYPE ₃	TC TYPE ₂	TC TYPE ₁	TC TYPE ₀
	Bit 7			Bit 0				

BIT	NAME	DESCRIPTION
7	Reserved	Reserved.
6:4	AVGSEL[2:0]	<p>Thermocouple Voltage Conversion Averaging Mode</p> <p>000 = 1 sample (default)</p> <p>001 = 2 samples averaged</p> <p>010 = 4 samples averaged</p> <p>011 = 8 samples averaged</p> <p>1xx = 16 samples averaged</p> <p>Adding samples increases the conversion time and reduces noise.</p> <p>Typical conversion times:</p> <p>1-shot or first conversion in Auto mode:</p> <p>= $t_{CONV} + (\text{samples} - 1) \times 33.33\text{mS}$ (60Hz rejection)</p> <p>= $t_{CONV} + (\text{samples} - 1) \times 40\text{mS}$ (50Hz rejection)</p> <p>2 thru n conversions in Auto mode</p> <p>= $t_{CONV} + (\text{samples} - 1) \times 16.67\text{mS}$ (60Hz rejection)</p> <p>= $t_{CONV} + (\text{samples} - 1) \times 20\text{mS}$ (50Hz rejection)</p> <p>The Thermocouple Voltage Conversion Averaging Mode settings should not be changed while conversions are taking place.</p>
3:0	TC TYPE[3:0]	<p>Thermocouple Type</p> <p>0000 = B Type</p> <p>0001 = E Type</p> <p>0010 = J Type</p> <p>0011 = K Type (default)</p> <p>0100 = N Type</p> <p>0101 = R Type</p> <p>0110 = S Type</p> <p>0111 = T Type</p> <p>10xx = Voltage Mode, Gain = 8. Code = $8 \times 1.6 \times 2^{17} \times V_{IN}$</p> <p>11xx = Voltage Mode, Gain = 32. Code = $32 \times 1.6 \times 2^{17} \times V_{IN}$</p> <p>Where Code is 19 bit signed number from TC registers and V_{IN} is thermocouple input voltage</p>

MAX31856

Precision Thermocouple to Digital Converter
with Linearization**Register 02h/82h: Fault Mask Register (MASK)**

The Fault Mask Register allows the user to mask faults from causing the $\overline{\text{FAULT}}$ output from asserting. Masked faults will still result in fault bits being set in the Fault Status register (0Fh). Note that the $\overline{\text{FAULT}}$ output is never asserted by thermocouple and cold-junction out-of-range status.

Default Value: FFh

MEMORY ACCESS	N/A	N/A	R/W	R/W	R/W	R/W	R/W	R/W
02h/82h	Reserved	Reserved	CJ High FAULT Mask	CJ Low FAULT Mask	TC High FAULT Mask	TC Low FAULT Mask	OV/UV FAULT Mask	Open FAULT Mask
	Bit 7							Bit 0

BIT	NAME	DESCRIPTION
7:6	Reserved	Reserved.
5	CJ High FAULT Mask	Cold-Junction High Fault Threshold Mask 0 = $\overline{\text{FAULT}}$ output asserted when the Cold-Junction Temperature rises above the Cold-Junction Temperature high threshold limit value 1 = $\overline{\text{FAULT}}$ output masked (default)
4	CJ Low FAULT Mask	Cold-Junction Low Fault Threshold Mask 0 = $\overline{\text{FAULT}}$ output asserted when the Cold-Junction Temperature falls below the Cold-Junction Temperature low threshold limit value 1 = $\overline{\text{FAULT}}$ output masked (default)
3	TC High FAULT Mask	Thermocouple Temperature High Fault Threshold Mask 0 = $\overline{\text{FAULT}}$ output asserted when the Thermocouple Temperature rises above the Thermocouple Temperature high threshold limit value 1 = $\overline{\text{FAULT}}$ output masked (default)
2	TC Low FAULT Mask	Thermocouple Temperature Low Fault Threshold Mask 0 = $\overline{\text{FAULT}}$ output asserted when the Thermocouple Temperature falls below the Thermocouple Temperature low threshold limit value 1 = $\overline{\text{FAULT}}$ output masked (default)
1	OV/UV FAULT Mask	Over-voltage or Undervoltage Input Fault Mask 0 = $\overline{\text{FAULT}}$ output asserted when an over- or undervoltage condition is detected 1 = $\overline{\text{FAULT}}$ output masked (default)
0	Open FAULT Mask	Thermocouple Open-Circuit Fault Mask 0 = $\overline{\text{FAULT}}$ output asserted when a thermocouple open condition is detected 1 = $\overline{\text{FAULT}}$ output masked (default)

MAX31856

Precision Thermocouple to Digital Converter
with Linearization**Register 03h/83h: Cold-Junction High Fault Threshold Register (CJHF)**

Write a temperature limit value to this register. When the measured cold-junction temperature is greater than this value, the CJ High fault status bit will be set and (if not masked) the **FAULT** output will assert.

Default Value: 7Fh

MEMORY ACCESS	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
03h/83h	CJHF7	CJHF6	CJHF5	CJHF4	CJHF3	CJHF2	CJHF1	CJHF0
	Sign	2 ⁶	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰
	Bit 7				Bit 0			

Register 04h/84h: Cold-Junction Low Fault Threshold Register (CJLF)

Write a temperature limit value to this register. When the measured cold-junction temperature is less than this value, the CJ Low fault status bit will be set and (if not masked) the **FAULT** output will assert.

Default Value: C0h

MEMORY ACCESS	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
04h/84h	CJLF7	CJLF6	CJLF5	CJLF4	CJLF3	CJLF2	CJLF1	CJLF0
	Sign	2 ⁶	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰
	Bit 7				Bit 0			

Register 05h/85h: Linearized Temperature High Fault Threshold Register, MSB (LTHFTH)

Write the MSB of the two-byte temperature limit value to this register. When the linearized thermocouple temperature is greater than the two-byte (05h and 06h) limit value, the TC High fault status bit will be set and (if not masked) the **FAULT** output will assert.

Default Value: 7Fh

MEMORY ACCESS	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
05h/85h	LTHFTH7	LTHFTH6	LTHFTH5	LTHFTH4	LTHFTH3	LTHFTH2	LTHFTH1	LTHFTH0
	Sign	2 ¹⁰	2 ⁹	2 ⁸	2 ⁷	2 ⁶	2 ⁵	2 ⁴
	Bit 7				Bit 0			

Register 06h/86h: Linearized Temperature High Fault Threshold Register, LSB (LTHFTL)

Write the LSB of the two-byte temperature limit value to this register. When the linearized thermocouple temperature is greater than the two-byte (05h and 06h) limit value, the TC High fault status bit will be set and (if not masked) the **FAULT** output will assert.

Default Value: FFh

MEMORY ACCESS	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
06h/86h	LTHFTL7	LTHFTL6	LTHFTL5	LTHFTL4	LTHFTL3	LTHFTL2	LTHFTL1	LTHFTL0
	2 ³	2 ²	2 ¹	2 ⁰	2 ⁻¹	2 ⁻²	2 ⁻³	2 ⁻⁴
	Bit 7				Bit 0			

MAX31856

Precision Thermocouple to Digital Converter
with Linearization**Register 07h/87h: Linearized Temperature Low Fault Threshold Register, MSB (LTLFTH)**

Write the MSB of the two-byte temperature limit value to this register. When the linearized thermocouple temperature is less than the two-byte (07h and 08h) limit value, the TC Low fault status bit will be set and (if not masked) the **FAULT** output will assert.

Default Value: 80h

MEMORY ACCESS	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
07h/87h	LTLFTH7	LTLFTH6	LTLFTH5	LTLFTH4	LTLFTH3	LTLFTH2	LTLFTH1	LTLFTH0
	Sign	2 ¹⁰	2 ⁹	2 ⁸	2 ⁷	2 ⁶	2 ⁵	2 ⁴
	Bit 7			Bit 0				

Register 08h/88h: Linearized Temperature Low Fault Threshold Register, LSB (LTLFTL)

Write the LSB of the two-byte temperature limit value to this register. When the linearized thermocouple temperature is less than the two-byte (07h and 08h) limit value, the TC Low fault status bit will be set and (if not masked) the **FAULT** output will assert.

Default Value: 00h

MEMORY ACCESS	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
08h/88h	LTLFTL7	LTLFTL6	LTLFTL5	LTLFTL4	LTLFTL3	LTLFTL2	LTLFTL1	LTLFTL0
	2 ³	2 ²	2 ¹	2 ⁰	2 ⁻¹	2 ⁻²	2 ⁻³	2 ⁻⁴
	Bit 7			Bit 0				

Register 09h/89h: Cold-Junction Temperature Offset Register (CJTO)

When the cold-junction temperature sensor is enabled, this register allows an offset temperature to be applied to the measured value. See the [Cold-Junction Temperature Sensing](#) section of this data sheet for additional information.

Default Value: 00h

MEMORY ACCESS	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
09h/89h	CJTO7	CJTO6	CJTO5	CJTO4	CJTO3	CJTO2	CJTO1	CJTO0
	Sign	2 ²	2 ¹	2 ⁰	2 ⁻¹	2 ⁻²	2 ⁻³	2 ⁻⁴
	Bit 7			Bit 0				

MAX31856

Precision Thermocouple to Digital Converter
with Linearization**Register 0Ah/8Ah: Cold-Junction Temperature Register, MSB (CJTH)**

This register contains the MSB of the two-byte (0Ah and 0Bh) value used for cold-junction compensation of the thermocouple measurement. When the cold-junction temperature sensor is enabled, this register is read-only and contains the MSB of the measured cold-junction temperature plus the value in the Cold-Junction Offset register. Also when the cold-junction temperature sensor is enabled, a read of this register will reset the $\overline{\text{DRDY}}$ pin high. When the cold-junction temperature sensor is disabled, this register becomes a read-write register that contains the MSB of the most recent cold-junction conversion result until a new value is written into it. This allows writing the results from an external temperature sensor, if desired. The maximum contained in the two cold-junction temperature bytes is clamped at 128°C and the minimum is clamped at -64°C.

Default Value: 00h

MEMORY ACCESS	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0Ah/8Ah	CJTH7	CJTH6	CJTH5	CJTH4	CJTH3	CJTH2	CJTH1	CJTH0
	Sign	2 ⁶	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰
	Bit 7			Bit 0				

Register 0Bh/8Bh: Cold-Junction Temperature Register, LSB (CJTL)

This register contains LSB of the two-byte (0Ah and 0Bh) value used for cold-junction compensation of the thermocouple measurement. When the cold-junction temperature sensor is enabled, this register is read-only and contains the LSB of the measured cold-junction temperature plus the value in the Cold-Junction Offset register. Also when the cold-junction temperature sensor is enabled, a read of this register will reset the $\overline{\text{DRDY}}$ pin high. When the cold-junction temperature sensor is disabled, this register becomes a read-write register that contains the LSB of the most recent cold-junction conversion result until a new value is written into it.

Default Value: 00h

MEMORY ACCESS	R/W	R/W	R/W	R/W	R/W	R/W	R	R
0Bh/8Bh	CJTL7	CJTL6	CJTL5	CJTL4	CJTL3	CJTL2	CJTL1	CJTL0
	2 ⁻¹	2 ⁻²	2 ⁻³	2 ⁻⁴	2 ⁻⁵	2 ⁻⁶	0	0
	Bit 7			Bit 0				

Register 0Ch: Linearized TC Temperature, Byte 2 (LTCBH)

This is the high byte of the 19-bit register that contains the linearized and cold-junction-compensated thermocouple temperature value.

Default Value: 00h

MEMORY ACCESS	R	R	R	R	R	R	R	R
0Ch	LTCBH7	LTCBH6	LTCBH5	LTCBH4	LTCBH3	LTCBH2	LTCBH1	LTCBH0
	Sign	2 ¹⁰	2 ⁹	2 ⁸	2 ⁷	2 ⁶	2 ⁵	2 ⁴
	Bit 7			Bit 0				

MAX31856

Precision Thermocouple to Digital Converter
with Linearization**Register 0Dh: Linearized TC Temperature, Byte 1 (LTCBM)**

This is the middle byte of the 19-bit register that contains the linearized and cold-junction-compensated thermocouple temperature value.

Default Value: 00h

MEMORY ACCESS	R	R	R	R	R	R	R	R
0Dh	LTCBM7	LTCBM6	LTCBM5	LTCBM4	LTCBM3	LTCBM2	LTCBM1	LTCBM0
	2 ³	2 ²	2 ¹	2 ⁰	2 ⁻¹	2 ⁻²	2 ⁻³	2 ⁻⁴
	Bit 7			Bit 0				

Register 0Eh: Linearized TC Temperature, Byte 0 (LTCBL)

This is the low byte of the 19-bit register that contains the linearized and cold-junction-compensated thermocouple temperature value.

Default Value: 00h

MEMORY ACCESS	R	R	R	R	R	R	R	R
0Eh	LTCBL7	LTCBL6	LTCBL5	LTCBL4	LTCBL3	LTCBL2	LTCBL1	LTCBL0
	2 ⁻⁵	2 ⁻⁶	2 ⁻⁷	X	X	X	X	X
	Bit 7			Bit 0				

Register 0Fh: Fault Status Register (SR)

The Fault Status Register contains eight bits that indicate the fault conditions (Thermocouple Out-of-Range, Cold Junction Out-of-Range, Cold Junction High, Cold Junction Low, Thermocouple High Temperature, Thermocouple Low Temperature, Over-Under Voltage, or Open Thermocouple) that have been detected.

Default Value: 00h

MEMORY ACCESS	R	R	R	R	R	R	R	R
0Fh	CJ Range	TC Range	CJHIGH	CJLOW	TCHIGH	TCLOW	OVUV	OPEN
	Bit 7			Bit 0				

Note: When the MAX31856 is set to operate in "comparator" fault mode (set with bit 2 of Configuration 0 register (00h)), the fault status bits simply reflect the state of any faults by asserting when the fault condition is true, and deasserting when the fault condition is no longer true.

When in "interrupt" fault mode, the fault status bits assert when a fault condition is true. The bits remain asserted until a 1 is written to the Fault Status Clear bit. This deasserts the fault bits until a new fault is detected (note that this may occur immediately if the fault condition is still in place).

MAX31856

Precision Thermocouple to Digital Converter
with Linearization

Register 0Fh: Fault Status Register (SR) (continued)

BIT	NAME	DESCRIPTION
7	CJ Range	Cold Junction Out-of-Range 0 = The Cold-Junction temperature is within the normal operating range (-55°C to +125°C for types E, J, K, N, and T; -50°C to +125°C for types R and S; 0 to 125°C for type B). 1 = The Cold-Junction temperature is outside of the normal operating range.
6	TC Range	Thermocouple Out-of-Range 0 = The Thermocouple Hot Junction temperature is within the normal operating range (see Table 1). 1 = The Thermocouple Hot Junction temperature is outside of the normal operating range. Note: The TC Range bit should be ignored in voltage mode.
5	CJHIGH	Cold-Junction High Fault 0 = The Cold-Junction temperature is at or lower than the cold-junction temperature high threshold (default). 1 = The Cold-Junction temperature is higher than the cold-junction temperature high threshold. The $\overline{\text{FAULT}}$ output is asserted unless masked.
4	CJLOW	Cold-Junction Low Fault 0 = The Cold-Junction temperature is at or higher than the cold-junction temperature low threshold (default). 1 = The Cold-Junction temperature is lower than the cold-junction temperature low threshold. The $\overline{\text{FAULT}}$ output is asserted unless masked.
3	TCHIGH	Thermocouple Temperature High Fault 0 = The Thermocouple Temperature is at or lower than the thermocouple temperature high threshold (default). 1 = The Thermocouple Temperature is higher than the thermocouple temperature high threshold. The $\overline{\text{FAULT}}$ output is asserted unless masked.
2	TCLOW	Thermocouple Temperature Low Fault 0 = Thermocouple temperature is at or higher than the thermocouple temperature low threshold (default). 1 = Thermocouple temperature is lower than the thermocouple temperature low threshold. The $\overline{\text{FAULT}}$ output is asserted unless masked.
1	OVUV	Overvoltage or Undervoltage Input Fault 0 = The input voltage is positive and less than V_{DD} (default). 1 = The input voltage is negative or greater than V_{DD} . The $\overline{\text{FAULT}}$ output is asserted unless masked. Note: The presence of the OVUV fault will suspend conversions and the ability of the MAX31856 to detect other faults (or clear faults when in comparator mode) until the fault is no longer present.
0	OPEN	Thermocouple Open-Circuit Fault 0 = No open circuit or broken thermocouple wires are detected (default). 1 = An open circuit such as broken thermocouple wires has been detected. The $\overline{\text{FAULT}}$ output is asserted unless masked.

MAX31856

Precision Thermocouple to Digital Converter
with Linearization

Applications Information

Thermocouple Temperature Sensing Guidelines

Follow these guidelines to get the best results when sensing temperature. The [Typical Application Circuit](#) shows a basic MAX31856 schematic. Connect the thermocouple wires to inputs T+ and T-; be sure that the wires are connected to the correct input as shown in [Figure 8](#). Connect the BIAS output to T-. This biases the thermocouple within the common-mode range of the inputs.

Noise Considerations

Because of the small signal levels involved, thermocouple temperature measurement is susceptible to power-supply-coupled noise. The effects of power-supply noise can be minimized by placing 0.1μF ceramic bypass capacitors close to the VDD pins and to GND.

The input amplifier is a low-noise amplifier designed to enable high-precision input sensing. Keep the thermocouple and connecting wires away from electrical noise sources. It is strongly recommended to add a 100nF ceramic surface-mount differential capacitor, placed across the T+ and T- pins, to filter noise on the thermocouple lines. In environments with high noise levels, especially significant RF fields, a 100nF capacitor between T+ and T- should be supplemented with a 10nF capacitor between T+ and GND, and another 10nF capacitor between T- and GND.

These values may need to be modified depending on the nature of the noise pickup. Other techniques, such as adding series resistance and shielding the thermocouple wires and circuit board, may also be necessary in the presence of larger noise sources. [Figure 8](#) shows the typical application circuit with input capacitors and input resistors added.

Input Protection

The ±45V input protection circuitry prevents damage to the IC caused by overvoltage conditions at T+, T-, or BIAS. If larger input faults are possible, external protection should be added. Resistors in series with T+, T-, and BIAS can increase the acceptable fault voltages. For example, adding 2kΩ in series with these inputs allows an additional ±40V of overdrive before the 20mA input current limit is reached. Note, however, that if the input has 45V across it and 20mA flowing into it, the power dissipation will be 900mW due to the overdrive at that input. Overdriving other inputs at the same time will further increase the power dissipation. Always ensure that if a continuous overdrive voltage greater than ±45V is expected, any current-limiting resistors are large enough to keep total power dissipation well under the IC's absolute maximum power dissipation. Note also that added resistance in series with T+ and T- can increase offset voltage, as mentioned in the [Effect of Series Resistance](#) section.

Figure 8. Typical Connection to Reduce the Effect of Noise Pickup in the Thermocouple Cable

MAX31856

Precision Thermocouple to Digital Converter
with Linearization

Figure 9. When Thermocouple Inputs May Be Exposed to Fault Voltages Greater than $\pm 45\text{V}$, Resistors Can Be Added to Limit Current into the MAX31856.

Effect of Series Resistance

Bias and leakage current at the thermocouple inputs will flow through input resistors and cable resistance, generating input offset voltage. For the circuits in Figure 8 and Figure 9, assuming that the thermocouple's source resistance is negligible, the offset voltage due to series resistance will be:

$$I_B \times \Delta R_S + \Delta I_B \times R_S$$

where:

- R_S is the series resistance between each input and the bias point
- ΔR_S is the difference between the two R_S values. This will generally be equal to the tolerance of any discrete series resistors plus any cable resistance.
- I_B is the input bias and leakage current
- ΔI_B is the differential input bias and leakage current

As an example, assume that the circuit in Figure 8 will be used up to a temperature of 85°C , the mismatch between the 100Ω input resistors is 1Ω , and the external cable resistance is 50Ω . This yields a worst-case offset voltage due to the external resistances of:

$$65\text{nA} \times (50\Omega + 1\Omega) + 4\text{nA} \times 100\Omega = 3.7\mu\text{V}$$

To minimize the effect of input resistance on accuracy:

- Minimize the values of any external resistors
- When the cable resistance is very low, match the values of the external resistors as closely as possible.
- If the cable resistance is known, increase the value of the resistor connected to T- by the value of the cable resistance. This will minimize the total mismatch between the two inputs.

If the cable resistance is excessive, consider using larger-gauge thermocouple wire.

MAX31856 Location

Because the MAX31856 includes an internal cold-junction temperature sensor, place it in a location whose temperature is as close as possible to that of the cold junction. If the thermocouple leads are directly soldered to the PCB, the MAX31856 should be as close as possible to the thermocouple lead connections and thermal gradients between the IC and the thermocouple connections should be minimized. If the thermocouple leads terminate in a connector, mount the IC as close as possible to the connector, and again minimize thermal gradients between the connector and the IC.

MAX31856

Precision Thermocouple to Digital Converter
with Linearization**Using “Unsupported” Thermocouple Types**

To use a thermocouple type other than B, E, J, K, N, R, S, or T, select one of the voltage mode options in Configuration 1. Selecting “Gain = 8” results in a full-scale input voltage range of $\pm 78.125\text{mV}$. “Gain = 32” results in a full-scale input voltage range of $\pm 19.531\text{mV}$. See

the transfer functions from the Configuration 1 Register table. When voltage mode is selected, no linearization is performed on the conversion data. Use the voltage data and the cold-junction temperature to calculate the thermocouple’s hot-junction temperature.

Ordering Information

PART	TEMP RANGE	PIN-PACKAGE
MAX31856MUD+	-55°C to +125°C	14 TSSOP
MAX31856MUD+T	-55°C to +125°C	14 TSSOP

+Denotes a lead(Pb)-free/RoHS-compliant package.
T = Tape and reel.

Package Information

For the latest package outline information and land patterns (footprints), go to www.maximintegrated.com/packages. Note that a “+”, “#”, or “-” in the package code indicates RoHS status only. Package drawings may show a different suffix character, but the drawing pertains to the package regardless of RoHS status.

PACKAGE TYPE	PACKAGE CODE	OUTLINE NO.	LAND PATTERN NO.
14 TSSOP	U14+2	21-0066	90-0113

MAX31856

Precision Thermocouple to Digital Converter
with Linearization

Revision History

REVISION NUMBER	REVISION DATE	DESCRIPTION	PAGES CHANGED
0	2/15	Initial release	—

For pricing, delivery, and ordering information, please contact Maxim Direct at 1-888-629-4642, or visit Maxim Integrated's website at www.maximintegrated.com.

Maxim Integrated cannot assume responsibility for use of any circuitry other than circuitry entirely embodied in a Maxim Integrated product. No circuit patent licenses are implied. Maxim Integrated reserves the right to change the circuitry and specifications without notice at any time. The parametric values (min and max limits) shown in the Electrical Characteristics table are guaranteed. Other parametric values quoted in this data sheet are provided for guidance.

Maxim Integrated and the Maxim Integrated logo are trademarks of Maxim Integrated Products, Inc. © 2015 Maxim Integrated Products, Inc. | 30