UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

VICENTE BUENO CARVALHO

Desenvolvimento e Teste de um Monitor de Barramento I2C para Proteção Contra Falhas Transientes

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Profª. Drª. Fernanda G. L. Kastensmidt

Porto Alegre
2016

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Carvalho, Vicente Bueno
Desenvolvimento e Teste de um Monitor de Barramento I2C para Proteção Contra Falhas Transientes / Vicente Bueno Carvalho. -- 2016.
87 f.

Orientadora: Fernanda Gusmão de Lima Kastensmidt.

Dissertação (Mestrado) -- Universidade Federal do Rio Grande do Sul, Instituto de Informática, Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2016.

1.Protocolo I2C 2.Tolerância à Falhas 3.APSoC. 4.PSoC I. Kastensmidt, Fernanda Gusmão de Lima, orient. II. Título.

AGRADECIMENTOS

Dedico este trabalho a toda minha família, especialmente a minha esposa Adriana. Agradeço a todos pelo apoio, paciência e companheirismo durante toda a trajetória de desenvolvimento do curso de mestrado.

Desenvolvimento e Teste de um Monitor de Barramento I2C para Proteção Contra Falhas Transientes

RESUMO

A comunicação entre circuitos integrados tem evoluído em desempenho e confiabilidade ao longo dos anos. Inicialmente os projetos utilizavam barramentos paralelos, onde existe a necessidade de uma grande quantidade de vias, utilizando muitos pinos de entrada e saída dos circuitos integrados resultando também em uma grande suscetibilidade a interferências eletromagnéticas (EMI) e descargas eletrostáticas (ESD). Na sequência, ficou claro que o modelo de barramento serial possuía ampla vantagem em relação ao predecessor, uma vez que este utiliza um menor número de vias, facilitando o processo de leiaute de placas, facilitando também a integridade de sinais possibilitando velocidades muito maiores apesar do menor número de vias.

Este trabalho faz uma comparação entre os principais protocolos seriais de baixa e média velocidade. Nessa pesquisa, foram salientadas as características positivas e negativas de cada protocolo, e como resultado o enquadramento de cada um dos protocolos em um segmento de atuação mais apropriado.

O objetivo deste trabalho é utilizar o resultado da análise comparativa dos protocolos seriais para propor um aparato de *hardware* capaz de suprir uma deficiência encontrada no protocolo serial I2C, amplamente utilizado na indústria, mas que possui restrições quando a aplicação necessita alta confiabilidade. O aparato, aqui chamado de Monitor de Barramento I2C, é capaz de verificar a integridade de dados, sinalizar métricas sobre a qualidade das comunicações, detectar falhas transitórias e erros permanentes no barramento e agir sobre os dispositivos conectados ao barramento para a recuperação de tais erros, evitando falhas. Foi desenvolvido um mecanismo de injeção de falhas para simular as falhas em dispositivos conectados ao barramento e, portanto, verificar a resposta do monitor. Resultados no PSoC5, da empresa Cypress, mostram que a solução proposta tem um baixo custo em termos de área e nenhum impacto no desempenho das comunicações.

Palavras-chave: Protocolo I2C. Tolerância à Falhas. APSoC. PSoC.

Development and Test of an I2C Bus Monitor for protection against Transient Faults

ABSTRACT

The communication between integrated circuits has evolved in performance and reliability over the years. Initially projects used parallel buses, where there is a need for a large amount of wires, consuming many input and output pins of the integrated circuits resulting in a great susceptibility to electromagnetic interference (EMI) and electrostatic discharge (ESD). As a result, it became clear that the serial bus model had large advantage over predecessor, since it uses a smaller number of lanes, making the PCB layout process easier, which also facilitates the signal integrity allowing higher speeds despite fewer pathways.

This work makes a comparison between the main low and medium speed serial protocols. The research has emphasized the positive and negative characteristics of each protocol, and as a result the framework of each of the protocols in a more appropriate market segment.

The objective of this work is to use the results of comparative analysis of serial protocols to propose a hardware apparatus capable of filling a gap found in the I2C protocol, widely used in industry, but with limitations when the application requires high reliability. The apparatus, here called I2C Bus Monitor, is able to perform data integrity verification activities, to signalize metrics about the quality of communications, to detect transient faults and permanent errors on the bus and to act on the devices connected to the bus for the recovery of such errors avoiding failures. It was developed a fault injection mechanism to simulate faults in the devices connected to the bus and thus verify the monitor response. Results in the APSoC5 from Cypress show that the proposed solution has an extremely low cost overhead in terms of area and no performance impact in the communication.

Keywords: I2C Protocol. Fault Tolerance. Aerospace. APSoC. PsoC.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ICs	Integrated Circuit
I2C	Inter IC Communication
I/O	Input/Output
EMI	Electromagnetic Interference
ESD	Electrostatic Discharge
CIs	Circuitos Integrados
SCL	Serial Clock
SDA	Serial Data
FPGA	Field Programmable Gate Array
SoC	System on Chip
HDL	Hardware Description Language
CPU	Central Processing Unit
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver/Transmitter
CAN	Controller Area Network
RS-232	Recommended Standard 232
USB	Universal Serial Bus
RTS	Ready to Send
CTS	Clear to Send
SCLK	Serial Clock
MOSI	Master Output Slave Input
MISO	Master Input Slave Output
CANH	Can High
CANL	Can Low
DSP	Digital Signal Processor
SATA	Serial ATA
PCIe	Peripheral Component Interconnet Express
SMBus	System Management Bus
OSI	Open Systems Interconnection
PEC	Packet Error Checking
ARP	Address Resolution Protocol
CRC	Cyclic Redundancy Check

BMC	Baseboard Management Controller
FSM	Finite State Machine
APSoC	All Programmable System on Chip
PGA	Programmable Gain Amplifier
TIA	Transimpedance Amplifier
DMA	Direct Memory Access
ADC	Analog to Digital Converter
SAR ADC	Successive Approximation ADC
DAC	Digital to Analog Converter
CPLD	Complex Programmable Logic Device
UDB	Universal Digital Block
PLDs	Programable Logical Device
LUTs	Look Up Tables
ULA	Unidade Lógica e Aritmética
FIFOs	First In First Out
SRAM	Static Random Memory Access
APIs	Application Program Interface

SUMÁRIO

1 INTRODUÇÃO

A comunicação entre circuitos integrados é fundamental. Com o avanço da tecnologia, há cada vez mais circuitos integrados a se comunicarem e as velocidades e confiabilidade da comunicação são cada vez mais importantes. A complexidade e custo de cada interface de comunicação também são requisitos prioritários na escolha do protocolo de comunicação mais adequado a um projeto.

A comunicação pode ocorrer somente entre dois dispositivos, sendo chamada de comunicação ponto a ponto, ou entre vários dispositivos através de um barramento ou rede. Além disso, pode-se adotar um meio físico paralelo ou serial. Portanto, a interface de comunicação define o meio físico pela qual as informações são transferidas, nesse nível é definido se a interface é serial ou paralela. Acima da interface é utilizado um protocolo de comunicação, este é responsável por definir padrões lógicos e regras que devem ser respeitadas por todos os dispositivos que implementam o protocolo em questão.

Dentre as possibilidades de comunicação ponto a ponto está o protocolo UART (BELL et al., 1978), amplamente utilizado em conjunto com a interface serial RS232 (INSTRUMENTS, 2009) para comunicação entre um circuito eletrônico e um PC. Um exemplo de barramento de comunicação é o protocolo I2C (Inter IC Communication) (NXP SEMICONDUCTORS, 2014), que também define a sua interface física, largamente utilizada em micro-controladores. O I2C permite que diversos circuitos integrados se comuniquem entre si em uma mesma placa utilizando somente duas vias. Os protocolos SPI (MOTOROLA, 2002) e CAN (BOSCH, 1998), também discutidos neste trabalho, são outros exemplos de protocolos de comunicação que utilizam barramento.

O protocolo I2C é amplamente difundido entre os engenheiros de hardware e firmware, largamente utilizado em diversos segmentos e por diferentes públicos, desde pequenos projetos como aparelhos de som e televisores até os mais avançados projetos tecnológicos como foguetes e satélites. Atualmente existem mais de 1000 diferentes circuitos integrados que implementam o protocolo I2C, sendo fabricados por mais de 50 companhias. O protocolo, que tem como meta simplificar a comunicação entre circuitos integrados, foi desenvolvido pela empresa Philips em 1982. Nos laboratórios da empresa, localizada na Holanda, os engenheiros buscavam uma alternativa para a única forma de comunicação entre circuitos integrados disponível na época, a memória mapeada. Essa solução era extremamente custosa, pois utilizava um barramento de dados, usualmente de oito bits, além de um barramento de endereçamento somado aos sinais de controle. Além de gastar preciosos pinos

de I/O, encarecendo os circuitos integrados da época, tornava o leiaute mais complicado e caro, e ainda deixava os circuitos vulneráveis a interferências eletromagnéticas (EMI) e a descargas eletroestáticas (ESD) (HIMPE, 2011).

O objetivo da empresa Philips na época era achar uma solução eficiente e barata para fazer uma unidade de controle, como um processador central, se comunicar com diversos circuitos integrados (CIs) dedicados, coletando e centralizando todas as informações sobre um dado sistema. Os sistemas em questão eram aparelhos de televisão e rádio, o barramento I2C foi utilizado pela primeira vez para coletar informações de controle de volume e contraste desses aparelhos. (CORCORAN, 2013)

O protocolo I2C foi desenvolvido com foco em substituir a interface de dados até então paralela por uma interface serial, tornando desnecessário o uso do barramento de dados. Além disso, no protocolo I2C, tanto o endereçamento do dispositivo a ser comunicado quanto o endereço do dado a ser lido ou escrito é feito através do próprio protocolo serial, dispensando assim a lógica para a seleção de CI e também do barramento de endereçamento. O protocolo utiliza somente dois fios para realizar o tráfego dos dados, um deles transporta o *clock* (SCL) e a outro transporta o dado (SDA).

A versatilidade e facilidade de uso do protocolo I2C o tornaram o padrão da indústria para comunicações de baixa velocidade entre circuitos integrados. Infelizmente o protocolo I2C não previu mecanismos para detecção e tratamento de falhas, tornando o seu uso impeditivo em certos equipamentos ou aplicações de alta disponibilidade. As causas de uma eventual falha podem ser inúmeras, bem como o efeito que a falha pode acarretar em um circuito. Os exemplos vão desde ruídos na linha que geram corrupção de dados, dispositivos defeituosos que falham ao transmitir o bit de verificação (*acknowledge bit*), interferências externas que podem causar falhas transientes como também a propagação de falhas transientes dentro do circuito do controlador I2C que pode gerar um travamento permanente de uma das linhas do barramento, fazendo com que nenhum dispositivo conectado ao mesmo possa ser acessado. Vale lembrar que a norma I2C (NXP SEMICONDUCTORS, 2014) descreve unicamente o comportamento que um dispositivo deve respeitar para ser compatível com o protocolo I2C, mas não sugere o design de hardware ou software que deve ser utilizado para implementá-lo. Na prática, vemos uma gama imensa de diferentes produtos com suporte ao protocolo I2C no mercado com uma infinidade de implementações proprietárias nas mais variadas formas como chips dedicados, *soft cores* (componentes lógicos programados em um dispositivo lógico programável como uma FPGA) e componentes em software puro.

Neste trabalho foi desenvolvida uma solução que visa tornar o barramento I2C mais robusto a falhas e interferências, conferindo a este, controle de integridade de dados, indicadores de falhas transientes e ação de recuperação do barramento em caso de falha permanente, atuando nos dispositivos conectados a ele. O Monitor de Barramento I2C é um core desenvolvido utilizando lógica programável, ele pode ser inserido em um sistema já existente caso este tenha um dispositivo programável que esteja conectado ao barramento I2C a ser monitorado.

Para validação do core proposto, foi desenvolvida uma aplicação teste utilizando o SoC PSoC5 da Cypress, no qual um cenário com comunicações através de um barramento I2C são exercitadas. Para validar o funcionamento do Monitor de barramento I2C aqui proposto, foi desenvolvida uma aplicação teste que exercita a comunicação entre os dispositivos I2C conectados ao barramento e reporta dados sobre a saúde das comunicações. Além do software de teste que roda no core ARM Cortex-M3 contido no próprio PSoC5, este trabalho também propõe uma metodologia de injeção de falhas em componentes *soft core* desenvolvidos em HDL. O processo de injeção de falhas em um *soft core* visa uma modelar falhas transientes.

Erro transiente, conhecido no inglês como *soft error,* é o efeito de uma falha transiente que muda temporariamente o comportamento de um sistema. Com uma reinicialização do sistema esse erro deve desaparecer. Falha transiente pode surgir quando partículas energizadas, como partículas alpha do material de encapsulamento e nêutrons da atmosfera, interagem com a área ativa de um transistor desligado gerando pares de elétron-lacuna e consequentemente uma corrente transiente que pode carregar ou descarregar o nó gerando assim um pulso de tensão transiente. Esse pulso de tensão transiente pode ser capturado por um elemento de memoria ou pode ocorrer em um transistor que compõe uma célula de memoria gerando assim o que chamamos de bit-flip. (MUKHERJEE, 2011)

Através da metodologia aqui proposta, foi possível estabelecer um comparativo entre o funcionamento do barramento sem o Monitor de Barramento I2C, objeto principal deste trabalho, e o barramento com o controle de falhas.

1.1 Organização da Dissertação

O segundo capítulo apresenta o protocolo I2C em detalhes. Ele detalha o funcionamento das comunicações entre circuitos integrados na atualidade. Descreve os protocolos seriais de baixo custo mais utilizados pela indústria e apresenta uma comparação entre eles.

O terceiro capítulo apresenta o estado-da-arte relativo às técnicas de mitigação de falhas em barramentos I2C. Ainda no capítulo estado-da-arte, foi traçada uma análise comparativa entre as funcionalidades presentes nas soluções apresentadas e as funcionalidades propostas neste trabalho. Desse modo, esse capítulo serve como referência a estudantes e pesquisadores que objetivam atuar na área de protocolos de baixa e média velocidade, com ênfase no protocolo I2C.

O capítulo quatro inicia com a proposta teórica do Monitor de Barramento I2C, separando suas funcionalidades em componentes lógicos e explicando como os componentes funcionam em conjunto. Na sequência do capítulo, a subseção dois apresenta o objeto de estudo deste trabalho, o APSoC PSoC5 da Cypress em conjunto com uma explicação aprofundada sobre o seu hardware digital programável, utilizado na implementação do Monitor de Barramento I2C. Ainda no capítulo quatro, a implementação do Monitor é explicada em detalhes, restando ainda o fechamento do capítulo que contém um comparativo em termos de recursos de hardware utilizados entre o Monitor de Barramento I2C e outros componentes I2C disponibilizados pelo fabricante.

O capítulo cinco apresenta o ambiente de testes utilizado para provar o conceito do Monitor. Na subseção um do capítulo cinco é apresentada uma metodologia para a simulação de falhas do tipo bit-flip desenvolvida no decorrer deste trabalho. Na subseção subsequente a aplicação teste é descrita e tem seu fluxo de operação apresentado. A última subseção do capítulo cinco apresenta os resultados consolidados obtidos através dos testes bem como comentários do autor sobre o desempenho do Monitor.

O capítulo seis descreve a metodologia de injeção de falhas utilizada, apresenta a aplicação teste e analisa os resultados obtidos da emulação da bateria de injeções de placa usando a placa CY8CKIT-050 (CYPRESS, 2012a).

O capitulo sete apresenta por fim as principais conclusões do trabalho, contribuições e discussões sobre trabalhos futuros.

2 COMUNICAÇÃO ENTRE CIRCUITOS INTEGRADOS

Atualmente a grande maioria dos produtos eletrônicos fabricados pela indústria possui um ou mais circuitos integrados em seu projeto. A divisão de tarefas entre circuitos integrados dedicados é uma ótima forma de modularizar as funcionalidades, tornando-as portáveis entre soluções. Dessa forma, se uma solução necessita de um monitoramento de temperatura, por exemplo, um sensor LM75 (TEXAS INSTRUMENTS, 2009) pode ser adicionado ao projeto de hardware para fazer a medição de temperatura. Ele poderá ser consultado por uma unidade de processamento e controle (CPU) através de um barramento I2C. Da mesma forma, o mesmo sistema acima citado pode necessitar de uma memória não volátil para armazenar dados de configuração da solução. Para essa tarefa uma memória flash (ATMEL CORPORATION, 2009) com acesso por protocolo SPI poderá ser utilizada.

Dentro de um sistema embarcado, não há a necessidade de utilização de protocolos de comunicação complexos para tarefas de gerenciamento e sincronização de dispositivos internos, pois normalmente quanto maior a complexidade de um protocolo, maior é o consumo de recursos para implementá-lo. Existem diversas comunicações dentro de um sistema embarcado que não necessitam de desempenho, mas de simplicidade e baixo consumo. Exatamente por essa razão, a maioria dos processadores de uso geral possui diversos controladores de interfaces seriais como I2C, SPI, UART, e um pouco menos comum CAN. Elas são conhecidas como interfaces de baixa ou média velocidade. Faremos um comparativo entre esses quatro protocolos na sequência deste capítulo.

É bastante comum uma solução utilizar mais de um protocolo serial para realizar tarefas que envolvam diferentes necessidades. Como foi exemplificada acima, a utilização do protocolo I2C na comunicação com sensores de temperatura é recomendada por se tratar de uma quantidade mínima de dados trafegados por transação, onde a velocidade de transferência não é relevante. Além disso, o I2C permite que uma grande quantidade de dispositivos escravos possa ser conectada a um mesmo barramento, e necessita de somente dois pinos da unidade de processamento que contém o controlador I2C.

Em outro cenário, caso a CPU necessite executar código de uma memória flash externa, o protocolo SPI é o mais recomendado, principalmente por se tratar de uma situação onde a velocidade de transferência é diretamente proporcional ao desempenho do sistema. O protocolo SPI pode atingir taxas de transferência até quatro vezes maiores do que o protocolo I2C.

A seguir serão apresentados alguns dos protocolos mais utilizados pela indústria em diferentes segmentos e aplicações. Ao final do capítulo um comparativo (**Erro! Fonte de referência não encontrada.**) entre os protocolos discutidos é exposto.

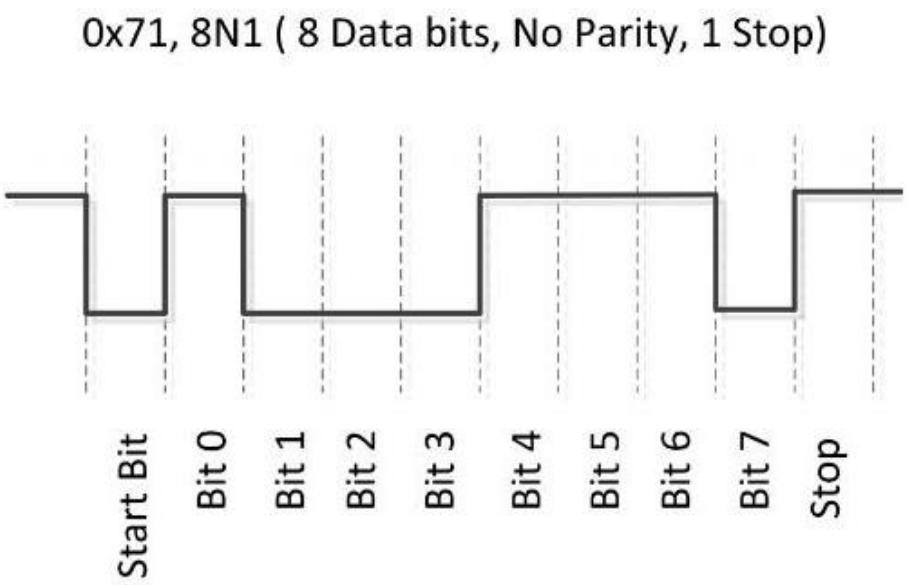
2.1 Protocolo UART

O protocolo UART (*Universal Asynchronous Receiver/Transmitter*) foi criado por Gordon Bell, durante o desenvolvimento do computador PDP-1 fabricado pela empresa DEC entre 1960 e 1965. O PDP-1 foi largamente utilizado pela empresa ITT (*International Telephone & Telegraph*) para realizar a comutação de mensagens de telégrafo (SLATER, 1989). Gordon desenvolveu um módulo de recebimento e transmissão de dados que basicamente convertia os dados a serem transmitidos de um barramento paralelo para a forma serial, e fazia o inverso com os dados seriais recebidos, convertendo-os para a forma paralela (BELL et al., 1978).

Desde a sua criação o protocolo UART é amplamente utilizado. A interface TIA/EIA-232-F (conhecida por RS-232) esteve presente na maioria dos computadores de uso pessoal até os anos 2000, onde era a interface padrão para a conexão do mouse. Ela continua sendo utilizada até hoje através de conversores USB-Serial (FTDI, 2008) disponíveis no mercado. A interface RS-232 estabelece as características elétricas, físicas e mecânicas para a implementação da interface que utiliza o protocolo UART, o que é imprescindível para a comunicação entre sistemas independentes. Dentro de um mesmo sistema, porém, normalmente não há necessidade de uma padronização de níveis elétricos e muito menos um padrão mecânico de conector, dado que os diferentes componentes são internos à solução e alimentados por uma mesma fonte. Neste caso o padrão RS-232 é desnecessário e o protocolo UART pode ser utilizado respeitando apenas os níveis elétricos do próprio sistema.

O protocolo UART utiliza um barramento desbalanceado capaz de efetuar comunicações *full-duplex* entre pares receptor/transmissor. Por se tratar de um protocolo assíncrono, sinais de sincronismo precisam ser enviados pela mesma linha onde trafegam os dados. Normalmente uma comunicação UART consiste em um start bit (sinaliza o início da comunicação), bits de dados, bits de paridade (se houver), e stop bits (sinaliza o fim). A configuração de conexão mais comum encontrada em campo é 8 bits de dados, sem paridade e um stop bit (8N1) (OSBORNE, 1980). Atualmente a maioria dos dispositivos que implementam uma controladora UART trabalha com velocidades entre 600 e 921600 bps.

Figura 2-1 Diagrama Comunicação UART



Fonte: (“ECE353: Introduction to Microprocessor Systems”, [s.d.])

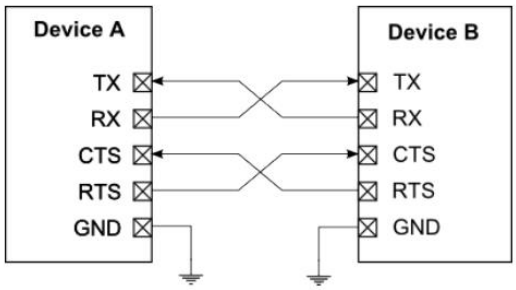
Por limitações de alguns sistemas, o armazenamento em buffer de dados trafegados por um canal UART não está disponível indefinidamente, especialmente se o mesmo buffer de dados é utilizado para o recebimento e transmissão. Para contornar este problema, sinais opcionais de controle de fluxo de dados foram criados. Para a utilização do controle de fluxo (*flow control*) duas vias extras são necessárias:

RTS – *Ready to Send* (Pronto para enviar)

CTS – *Clear to Send* (De acordo com o envio)

RTS e CTS são conectados de forma cruzada, RTS do dispositivo A é conectado ao CTS do dispositivo B e vice-versa. Cada dispositivo utiliza a saída RTS para informar ao dispositivo par que está pronto para receber dados e verifica a entrada CTS para se certificar que pode enviar dados ao dispositivo par (SILICON LABS, 2013).

Figura 2-2 Interconexão entre dispositivo UART com controle de fluxo.



Fonte: (SILICON LABS, 2013)

2.2 Protocolo SPI

O protocolo SPI (*Serial Peripheral Interface*) foi desenvolvido pela empresa Motorola em 1979. Ele foi lançado incorporado ao micro controlador derivado da mesma arquitetura do Motorola 68000, o M68HC03 (LEENS, 2009). Diferentemente do I2C, o protocolo SPI não possui uma documentação formal, para se obter uma descrição detalhada de seu funcionamento, a documentação do micro controlador deve ser consultada.

O protocolo SPI utiliza quatro sinais para realizar a conexão entre um dispositivo mestre e um dispositivo escravo:

SCLK - *Serial* de *clock*, todos os sinais do barramento são sincronizados com este sinal;

SSn – Seletor de dispositivo escravo (*Slave Selection*), utilizado para selecionar o dispositivo escravo que o mestre deseja realizar uma comunicação;

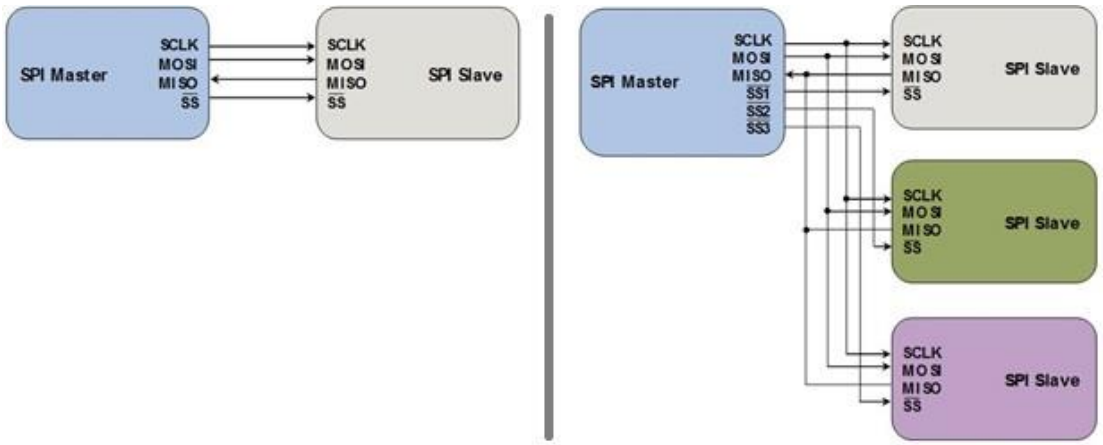
MOSI – Linha de dados do mestre para o escravo (*Master-Out Slave-In*);

MISO – Linha de dados do escravo para o mestre (*Master-In Slave-Out*).

Um dos sinais utilizados tem o papel de seletor de dispositivo escravo (*Slave Select - SSn*), o que significa que cada novo dispositivo escravo conectado ao barramento necessita de um sinal exclusivo conectado ao dispositivo mestre. O cálculo de largura do barramento SPI, portanto, é dependente do número de dispositivos escravos (NXP SEMICONDUCTORS, 2014).

Largura_Barramento_SPI = 3 + Número_Dispositivos_Escravos

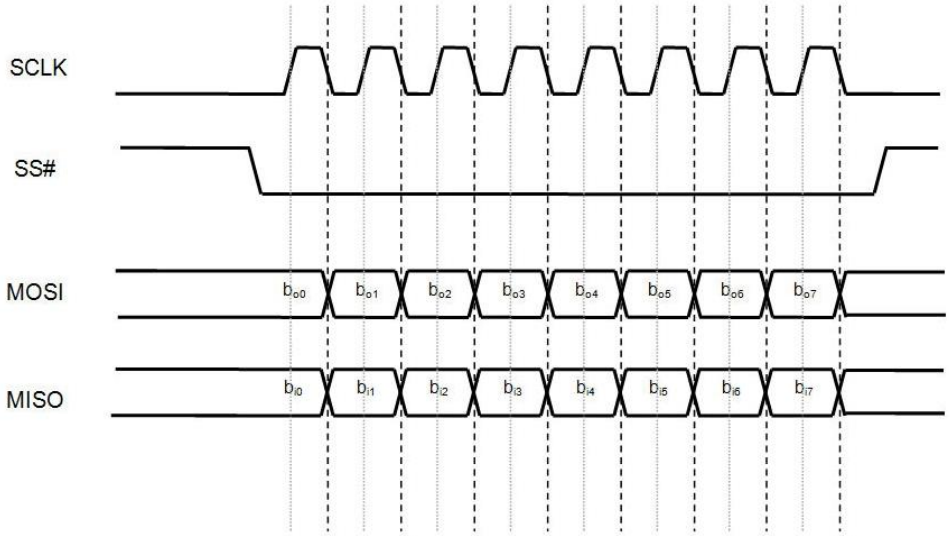
Figura 2-3 Opções de configuração de barramento SPI.



Fonte: (LEENS, 2009)

O barramento SPI permite a utilização de um único mestre, e não limita o número de escravos a ele conectados. Por haver somente um dispositivo mestre no barramento, qualquer comunicação é iniciada a partir dele. Quando o dispositivo mestre deseja enviar dados e/ou receber dados de um dispositivo escravo, ele seleciona o dispositivo alvo através da linha seletora (SSn), ativa o *clock*, envia dados através da linha MOSI enquanto amostra os dados da linha MISO. No protocolo SPI os dispositivos são capazes de enviar e receber dados simultaneamente, portanto é um protocolo *full-duplex*.

Figura 2-4 Diagrama de comunicação SPI.



Fonte: (LEENS, 2009)

O protocolo SPI não define uma taxa máxima de transferência de dados. Também não há qualquer mecanismo de *acknowledgement* para a confirmação de dados recebidos e não

oferece qualquer tipo de controle de fluxo. Além disso, SPI não descreve as características físicas como nível de tensão ou uma norma para padronizar o meio entre os dispositivos (LEENS, 2009).

2.3 Protocolo CAN

O protocolo CAN (*Controller Area Network*), desenvolvido pela empresa BOSCH, foi oficialmente lançado em 1986. O objetivo do seu desenvolvimento foi o de facilitar a fiação em veículos automotivos, substituindo uma grande quantidade de fios por um barramento de apenas dois fios que percorreria todo o veículo. Para atingir esta meta, requerimentos como imunidade a interferências elétricas e a habilidade de diagnosticar e reparar erros na integridade de dados foram incorporados ao projeto.

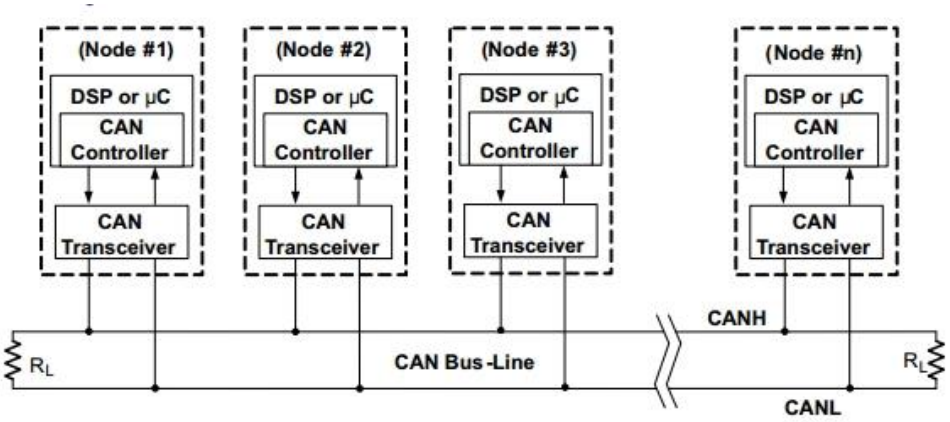
O protocolo é classificado como *broadcast multi-drop,* o que significa que todos os dispositivos conectados ao barramento recebem todas as mensagens trafegadas no mesmo, porém cada dispositivo trata somente das mensagens pertinentes a si, baseando-se no identificador da própria mensagem. O barramento do protocolo CAN é composto por dois sinais, CANH e CANL. Estes dois sinais formam um par diferencial, o que garante ao barramento CAN robustez contra ruídos e tolerância a falhas (CORRIGAN, 2002).

De acordo com a especificação do protocolo (BOSCH, 1998), taxas de até 1 Mbps podem ser alcançadas utilizando CAN, porém a taxa de dados trafegados é muito menor dada a quantidade de informações de controle e o limite de dados úteis por pacote que o protocolo pode suportar. Cabe dizer que se trata de um protocolo *half-duplex* uma vez que o barramento suporta o tráfego de dados apenas em uma direção em um dado momento.

Dentre as características de detecção de erros e reparos, é importante citar:

- Quando um nó recebe uma mensagem incompleta ou com erro, ele transmite uma mensagem de erro. Isso faz com que a mensagem inicial seja retransmitida;
- Se um nó trava um barramento por transmitir repetidas mensagens de erro, sua capacidade de transmissão é bloqueada pelo controlador após o limite de mensagens ser atingido;
- Todo o nó (dispositivo CAN) ao transmitir um bit no barramento, monitora o barramento para garantir que o dado escrito coincide com o valor lido do barramento. Caso essa condição seja falsa, um erro é gerado;

Figura 2-5 Rede CAN exemplificando os componentes que compõem um nó.



Fonte: (CORRIGAN, 2002)

Um dispositivo CAN, também chamado de nó, é composto por uma unidade de controle (micro controlador/processador/DSP), um controlador CAN que pode estar embutido no micro controlador ou DSP, e um CAN *transceiver* que converte os níveis elétricos utilizados pelo controlador CAN para o padrão CAN. Como o nó CAN possui um custo elevado dado a quantidade de componentes que o compõe, aplicações industriais com necessidades simples, como a leitura de diversas temperaturas, conectam todos os sensores à unidade de processamento, que por sua vez, consolida todas as informações e as provê através de uma única interface CAN.

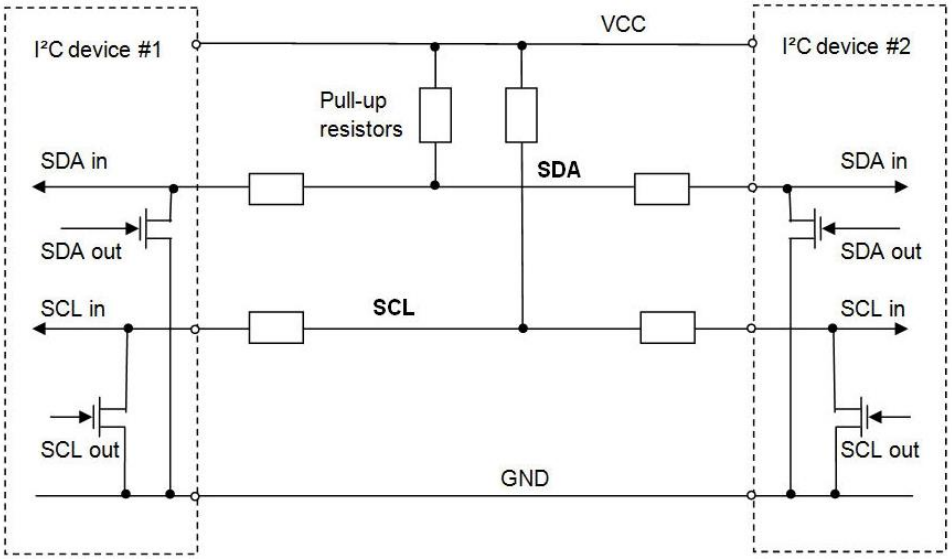
2.4 Protocolo I2C

O protocolo I2C (*Inter IC Communication*), desenvolvido pela empresa Philips em 1982, tem como principal objetivo simplificar a comunicação entre uma unidade de controle e diversos dispositivos de baixa complexidade pertencentes ao mesmo circuito (KALINSKY; KALINSKY, 2001). O I2C tem como ponto forte a baixa complexidade de hardware aliado a um elegante conjunto de funcionalidades embutidas no próprio protocolo como: arbitragem, endereçamento do dispositivo escravo, confirmação de recebimento, detecção de colisões, entre outras que também serão discutidas neste capítulo.

O barramento é composto por apenas dois sinais, SCL que transporta o *clock*, caracterizando o I2C como um protocolo síncrono, e o SDA, que é o sinal responsável por transportar os dados em ambas as direções, o que por sua vez caracteriza o protocolo como *half-duplex*. Ambas as linhas SCL e SDA são bidirecionais, logo os dispositivos I2C necessitam utilizar saídas do tipo dreno aberto, além de cada linha também necessitar de um

resistor de *pull-up* externo. Este tipo de configuração tem como característica o nível lógico baixo como dominante, isto é, se mais de um dispositivo estiver controlando uma linha bidirecional ao mesmo tempo, o dispositivo que estiver transmitindo o nível lógico baixo irá vencer. Outra característica das linhas bidirecionais utilizadas pelo protocolo I2C é a separação dos registradores de entrada e saída na porta dos dispositivos. Assim, um dispositivo mestre pode, por exemplo, escrever no registrador de saída da linha SDA o valor alto e verificar através do registrador de entrada que a linha SDA, na realidade, está em nível baixo, possivelmente sendo controlada por outro dispositivo conectado ao barramento. É como se todos os dispositivos presentes no barramento tivessem os registradores de saída de cada linha conectados a uma porta lógica *AND*. O protocolo I2C suporta múltiplos dispositivos mestre e múltiplos dispositivos escravos no mesmo barramento.

Figura 2-6 Conexão física de dispositivo I2C à um barramento I2C.



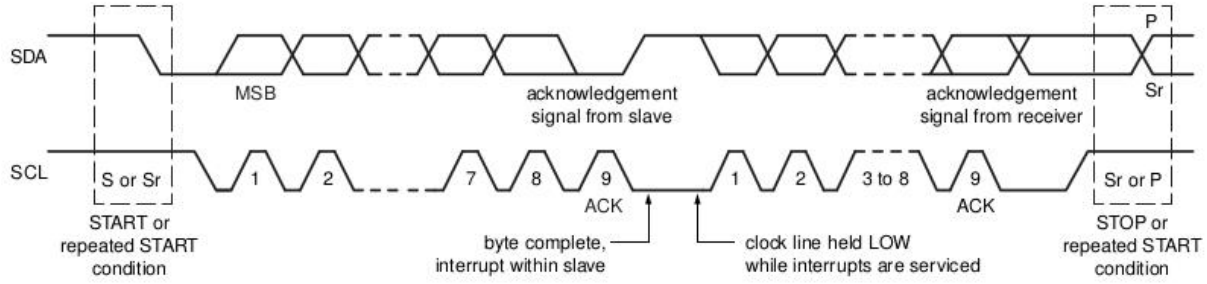
Fonte: (LEENS, 2009)

A comunicação entre dispositivos I2C é orientada a palavras de oito bits, para cada palavra de 8 bits transmitida um sinal de confirmação de recebimento, chamado *acknowledgement bit (ACK)*, deve ser retornado pelo dispositivo receptor. Somente um dispositivo mestre pode iniciar uma comunicação, é sempre um dispositivo mestre que controla a linha de *clock* e cada dispositivo escravo possui um endereço físico único no barramento. O protocolo I2C também prevê alguns sinais de controle utilizados antes, durante e depois de qualquer transação.

Antes de um dispositivo mestre iniciar uma comunicação ele deve verificar se o barramento está disponível, isso é verdade quando ambas as linhas SCL e SDA estão em nível

lógico alto. Depois de confirmar que o barramento está disponível, o mestre envia um sinal de controle chamado *start bit* que indica aos demais dispositivos que a palavra seguinte a ser transmitida contém um endereço de dispositivo escravo. Os endereços I2C possuem 7 bits de tamanho, o que permite endereçar pouco menos de 128 dispositivos escravos, pois alguns endereços são reservados para ações específicas. O bit menos significativo do byte que contém o endereço do dispositivo I2C sinaliza o tipo de transação, escrita ou leitura, da perspectiva do dispositivo mestre.

Figura 2-7 Diagrama lógico de uma transação I2C.



Fonte: (NXP SEMICONDUCTORS, 2014)

Os dados que trafegam na linha SDA são sempre amostrados na borda de subida da linha de *clock* SCL, e durante a transmissão dos dados a linha SDA só deve ser atualizada quando a linha de *clock* estiver em seu nível lógico zero. Os sinais de controle, porém, não respeitam essas regras, esses só podem ser emitidos por dispositivos mestres.

Start bit: É um sinal de controle enviado pelo mestre antes de iniciar uma transação. Tem como intuito alertar todos os dispositivos pertencentes ao barramento, mestres e escravos, que uma transação irá iniciar.

Para um dispositivo mestre enviar um *Start bit,* o barramento deve estar ocioso. O *Start bit* consiste em realizar a transição da linha SDA de nível lógico alto para nível lógico baixo enquanto a linha SCL permanece em nível lógico alto.

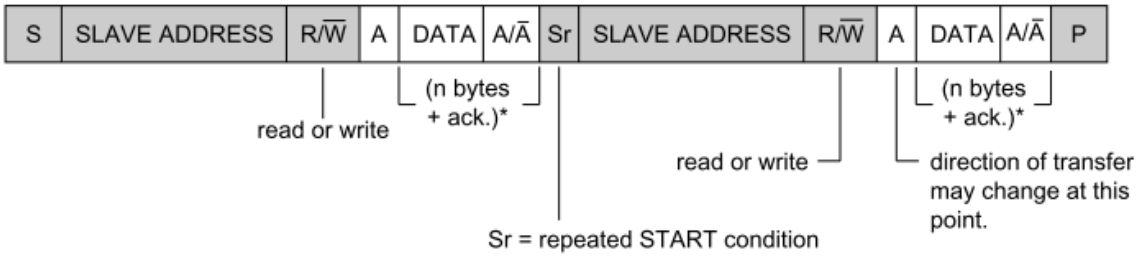
Stop bit: Tem papel oposto ao start bit, ele sinaliza à todos os demais dispositivos no barramento que o mestre que detinha o controle barramento até então o está liberando. O *Stop bit* consiste em realizar a transição da linha SDA de nível lógico baixo para o nível lógico alto enquanto a linha SCL permanece em nível lógico alto.

Repeated Start: É utilizado quando um dispositivo mestre deseja continuar controlando o barramento, porém necessita alterar o tipo de operação, leitura ou escrita, ou ainda quando o mestre deseja endereçar outro dispositivo escravo. O *Repeated Start* consiste em enviar um *Start bit* quando o mestre já possui o controle do barramento, na sequência deve

ser enviada a palavra contendo o endereço do dispositivo escravo e o tipo de operação desejado.

Acknowledgement bit: Toda palavra que trafega em um barramento I2C necessita de um sinal de confirmação, chamado *acknowledgement bit,* do dispositivo que está recebendo o dado, não importando se este é um dispositivo mestre ou escravo. Em uma transação de escrita, onde um dispositivo mestre escreve em um dispositivo escravo, a cada palavra de 8 bits transmitida o dispositivo mestre põe a linha de dados SDA em nível lógico alto, transiciona a linha de clock SCL de nível baixo para nível alto e simultaneamente verifica o estado da linha SDA. Cabe ao dispositivo escravo segurar a linha SDA em nível baixo enquanto o mestre efetua a transição de subida da linha de *clock*. Caso o dispositivo escravo não sinalize o *acknowledgement bit,* o dispositivo mestre verificará que a linha SDA permanece em nível alto e interromperá imediatamente a transação. No caso de uma leitura, onde o mestre está lendo dados de um dispositivo escravo, a cada palavra de oito bits lida pelo mestre, este transiciona a linha SCL do nível baixo para o nível alto mantendo a linha SDA em nível baixo, sinalizando o *acknowledgement bit*. Da mesma forma, o dispositivo escravo durante a transmissão do *acknowledgement bit* deve manter a linha SDA em nível lógico alto e ler seu valor como nível lógico baixo.

Figura 2-8 Sequência de escrita e leitura utilizando o *Repeated Start*.

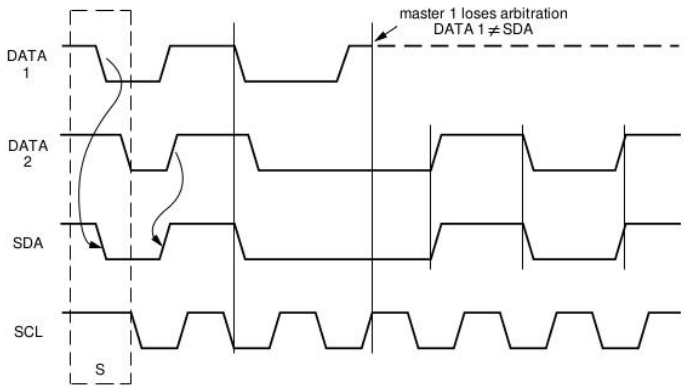


Fonte: (NXP SEMICONDUCTORS, 2014)

Por se tratar de um protocolo multi-mestre, onde em um mesmo barramento se pode ter diversos dispositivos mestres, é necessário um mecanismo para arbitrar qual dispositivo mestre tem o controle do barramento em cada momento. Para isso, o dispositivo mestre que pretende ter o controle do barramento deve, antes de enviar o *Start bit*, assegurar que o barramento está livre. Da mesma forma, quando um dispositivo mestre percebe um *Start bit* no barramento, ele deve aguardar um *Stop bit,* antes de tentar ter o controle do barramento. Caso dois dispositivos mestres iniciem uma transação ao mesmo tempo, o protocolo I2C resolve a questão da arbitragem utilizando as características das linhas bidirecionais. Sempre que um dispositivo mestre altera o estado de uma linha do barramento, SCL ou SDA, através

dos registradores de saída, ele deve monitorar os registradores de entrada para garantir que o dado escrito está, de fato, no barramento. Caso não esteja, o dispositivo mestre deve cessar sua atividade no barramento imediatamente, sinalizando à camada superior erro de arbitragem.

Figura 2-9 Arbitragem ocorrendo entre dois dispositivos mestres.



Fonte: (NXP SEMICONDUCTORS, 2014)

No protocolo I2C, apesar da linha de *clock* ser sempre controlada por um dispositivo mestre, é possível que um dispositivo escravo "segure" o barramento por um determinado tempo, pausando a transação até ser capaz de processar um comando e montar uma resposta. Essa funcionalidade, conhecida como *clock stretching*, é importante em casos onde a velocidade do protocolo é maior do que o tempo de processamento ou armazenamento do dado recebido pelo dispositivo escravo. Seu funcionamento é simples, o dispositivo escravo segura a linha de *clock* (SCL) em nível lógico baixo enquanto estiver processando internamente dados previamente recebidos. Uma ver que o dispositivo escravo está pronto para responder ao comando solicitado, este libera a linha de clock. O dispositivo mestre então volta a controlar a linha SCL continuando a transação do ponto onde foi pausada pelo dispositivo escravo.

Todas as informações fornecidas sobre o protocolo I2C até aqui se referem aos modos *Standard*, *Fast e Fast Plu*s. Além destes modos de operação existem ainda os modos *High Speed e Ultra Fast*, o último, porém, possui varias diferenças em relação ao protocolo I2C original. No modo Standard o protocolo I2C permite velocidades de *clock* de até 100 kHz, no modo *Fast* o limite passa a ser 400 kHz, no modo *Fast Plus* é de 1 MHz, no modo *High Speed* é de 3.4MHz e no modo *Ultra Fast* chega até 5 Mhz. Todos os modos são compatíveis entre si, com a exceção do modo *Ultra Fast* (I2C-BUS.ORG, [s.d.]).

2.5 Protocolos Ethernet, USB, SATA, PCIe e outros.

Os protocolos I2C, SPI, CAN e UART são considerados protocolos lentos quando comparados com outros protocolos seriais tais como *Ethernet*, *USB*, *SATA* e *PCIe*, que podem chegar a velocidades maiores do que 100Mb/s, se não Gb/s. No entanto, é importante não esquecer a finalidade de cada protocolo. Protocolos como Ethernet e os demais citados, são destinados à intercomunicações entre sistemas. Quando existe a necessidade de implementar uma comunicação entre um circuito integrado como um micro controlador e um conjunto de periféricos lentos, não há sentido em utilizar um protocolo excessivamente complexo. Neste contexto os protocolos apresentados nos itens anteriores são muito úteis, garantindo um excelente custo benefício para a solução (LEENS, 2009).

2.6 Comparativo Entre Protocolos de Baixa e Média Velocidade

Tabela 2-1 Comparativo de características dos protocolos seriais.

	I2C	SPI	CAN	UART
Área (Gates) [1]	2300/1000[2]	1500	5500	2300
Largura Barramento	2	3 + n [3]	2	2 [4]
Velocidade (kbps)	100/400/ 1000/3.400	10.000	1.000	921,6
Síncrono	Síncrono	Síncrono	Assíncrono	Assíncrono
Troca a quente	Sim	Sim	Sim	Sim
Half/Full duplex	Half-duplex	Full-duplex	Half-duplex	Full-duplex
Multi-master	Sim	Não	Sim	Não
Arbitragem	Embutida	Não	Embutida	N/A
Endereçamento	Embarcado [5]	Externo [3]	Embarcado [6]	N/A
Overhead [7]	Baixo	Nenhum	Alto	Baixo
Robustez	Média	Baixa	Alta	Baixa
Aplicação Alvo	Poucos dispositivos mestres e muitos dispositivos escravos.	Um dispositivo mestre e poucos escravos.	Grande quantidade de nós em ambientes com ruído.	Comunicação entre dois dispositivos.
Tratamento de erros	Não	Não	Sim	Não
Roteamento e montagem do	Fácil	Médio [8]	Médio [9]	Fácil

circuito				
Local de utilização	Circuitos Integrados montados em uma mesma placa.	Circuitos Integrados montados em uma mesma placa.	Entre diferentes circuitos.	Circuitos Integrados montados em uma mesma placa. [10]

Fonte: Autor

[1] Todos os *cores* utilizados na comparação são propriedade de IPextreme (http://www.ip-extreme.com/IP) e utilizam tecnologia de 130nm. Os *gates* são equivalentes a NAND2.

[2] 2300 *gates*: I2C Escravo / 1000 *gates* I2C multimestre

[3] Necessidade de uma linha extra no barramento por dispositivo escravo adicionado.

[4] Sem a utilização de controle de fluxo.

[5] Previsto no protocolo. O endereçamento pode ser de 7 bits ou 10 bits.

[6] – Endereçamento de mensagens e não de dispositivos, com definição de prioridade. Possibilidade de utilização de endereçamento de 11 bits ou 19 bits.

[7] – Overhead: Quantidade de dados de controle necessários para transportar dados úteis.

[8] - Um número alto de dispositivos dificulta o roteamento dado à quantidade de linhas necessárias para a seleção do dispositivo escravo (*Slave Selection*).

[9] - Necessita um dispositivo conversor de nível elétrico (*CAN driver*).

[10] – Pode ser utilizado entre diferentes circuitos através da interface RS-232.

3 ESTADO DA ARTE DO PROTOLOCO I2C SOB FALHAS TRANSIENTES

Desde a criação do protocolo I2C, foram despendidos esforços no sentido de torná-lo mais robusto, possibilitando seu uso em uma quantidade maior de aplicações. Na camada lógica do protocolo, existe uma série de iniciativas que utilizam como base a interface SMBus (SBS, 2000), uma camada acima do protocolo I2C que permite o gerenciamento de dispositivos compatíveis com a interface. A interface SMBus também realiza o controle de integridade de dados trafegados, além de muitas outras funcionalidades. Referente à camada física, pode-se encontrar exemplos que utilizam a técnica de *hardware hardening*, como descreve o artigo "*ESD protection of open-drain I2C using fragile devices in embedded systems*", que eleva a imunidade dos circuitos I2C a descargas eletrostáticas (ESD) através da utilização de uma técnica de projeto de hardware própria (FARBIZ; ALI; SANKARALINGAM, 2015). Existem também outras iniciativas, mais próximas à proposta neste trabalho, que utilizam um hardware adicional para atingir o objetivo de maior robustez no uso do protocolo, é o caso proposto no artigo intitulado "*Improvement of I2C Bus and RS-232 Serial Port under Complex Electromagnetic Environment*" (CHAI et al., 2008).

Serão discutidos neste capítulo, além da interface SMBus, duas patentes sobre o assunto:

- US 2004/6728908 B1 I2C BUS PROTOCOL CONTROLLER WITH FAULT TOLERANCE (FUKUHARA et al., 2004);

- US 2007/0240019 A1 SYSTEMS AND METHODS FOR CORRECTING ERRORS IN I2C BUS COMMUNICATIONS (BRADY et al., 2007);

3.1 Interface SMBus

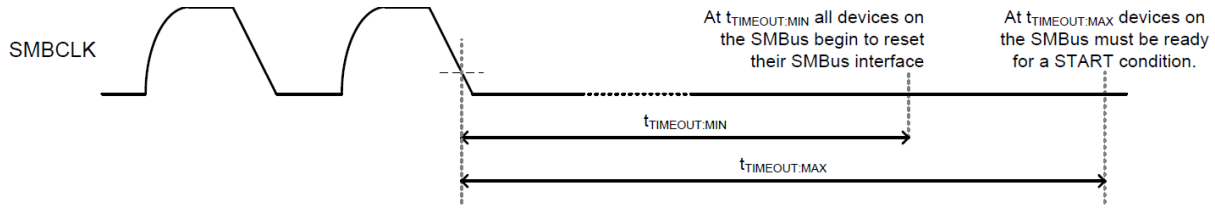
A interface SMBus foi originalmente criada para padronizar o acesso a dispositivos I2C. Sua aplicação inicial foi em carregadores de bateria inteligentes, normalmente utilizados em computadores portáteis para auxiliar o sistema operacional a realizar tarefas de gerenciamento de energia. A especificação da interface foi desenvolvida em 1994 pelas companhias Duracell e Intel. Logo ela foi adotada em diversos outros segmentos de mercado (BERGVELD; KRUIJT; NOTTEN, 2002).

Além da camada de rede do modelo OSI (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 1996), também conhecida por camada 3, a interface SMBus

especifica as camadas Física e de Enlace do modelo, camadas 1 e 2 respectivamente. O principal aspecto na camada física especificado pela interface SMBus que difere da norma I2C é a existência de um *timeout* que permite que um dispositivo mestre ou escravo defeituoso não permaneça indefinidamente segurando a linha de *clock* (SCL) em nível lógico baixo mantendo o barramento em uma condição de travamento.

A interface especifica dois valores de timeout, $t_{TIMEOUT,MAX}$ e $t_{TIMEOUT,MIN}$, a recomendação é que um dispositivo libere o barramento sempre que a linha de *clock* for mantido em nível lógico baixo por um período maior que $t_{TIMEOUT,MIN}$. Os dispositivos que detectarem esta situação devem reiniciar suas interfaces de comunicação e estarem disponíveis para iniciar uma nova comunicação (receber *Start bit*) em até $t_{TIMEOUT,MAX}$ (SBS, 2000).

Figura 3-1 Ilustração das definições de timeout mínimo e máximo.



Fonte: (SBS, 2000)

A camada 2 (Enlace) é similar à especificação do próprio protocolo I2C, variando somente alguns parâmetros relativos às tolerâncias de tempo, que na interface SMBus são mais restritas.

A camada de rede da interface SMBus garante a possibilidade de verificação da integridade dos dados em cada transação, essa funcionalidade é chamada de PEC (*Packet Error Checking* – Checagem de erro de pacote em tradução livre) na documentação. Além disso, a interface SMBus descreve em sua camada de rede o protocolo ARP (*Address Resolution Protocol* – Protocolo de Resolução de Endereço em tradução livre). O protocolo ARP permite que os dispositivos SMBus sejam endereçados dinamicamente, isto é, uma reconfiguração de hardware e software que permite atribuir um endereço único a um dispositivo recém conectado ao barramento tornando seu uso possível imediatamente sem uma reconfiguração do dispositivo mestre. Não entraremos a fundo no funcionamento do protocolo ARP, pois este não está relacionado com o escopo deste trabalho.

Na camada de rede do SMBus, o conceito de comando é proposto. Existe uma gama de diferentes comandos possíveis, e dependendo do comando, o uso ou não do PEC se faz necessário (SBS, 2000).

O byte de PEC, presente na maioria dos comandos, é obtido através do cálculo do CRC-8 sobre todo o pacote, incluindo o bit de leitura/escrita que compõe o byte de endereço. Para o cálculo do CRC-8 o seguinte polinômio é utilizado (MILIOS, 1999):

$$x^8 + x^2 + x + 1 \quad (1)$$

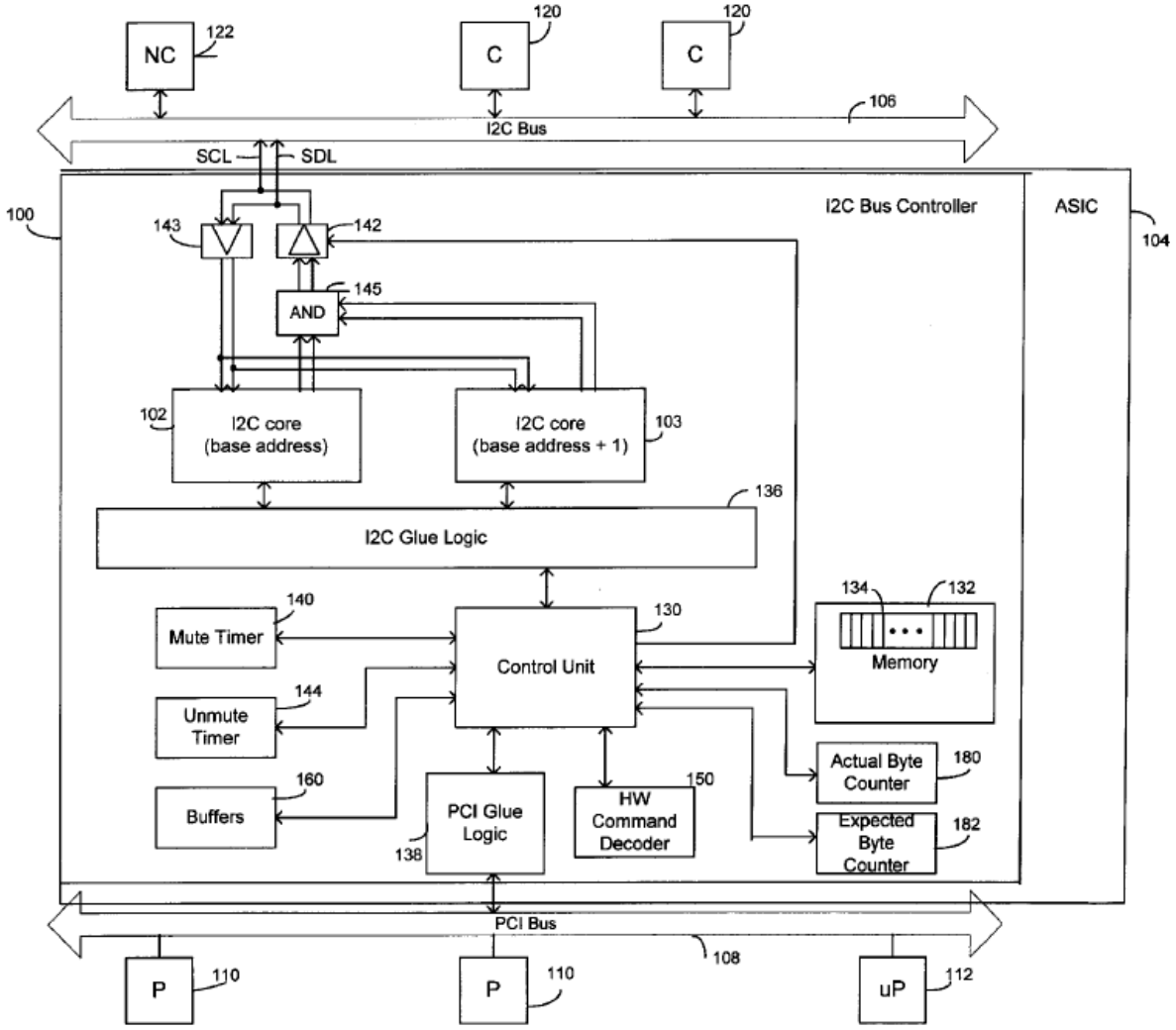
3.2 I2C Bus Protocol Controller with Fault Tolerance (US2004/6728908)

Esta patente propõe um controlador I2C próprio com características que, de acordo com seus autores, o tornam tolerante a falhas. Em sua descrição ainda, é sugerido que a solução proposta está apta a ser utilizada em aplicações aeronáuticas e astronáuticas. Os principais recursos presentes na solução são o suporte a falhas silenciosas, verificação de integridade de dados, verificação da quantidade de dados trafegados e capacidade de operar no modo mestre ou escravo.

Como pode ser observado na Figura 3-2, a solução é composta por uma unidade de controle (130) conectada, através de um circuito customizado de hardware (136 - *glue logic*), a um *core* I2C padrão (102) com um endereço base. Existe ainda um segundo *core* I2C padrão (103) conectado a unidade de controle, através do mesmo *hardware* customizado, com endereço base mais um. Outro aspecto relevante, relativo ao hardware utilizado, são os *drivers* transmissor e receptor (142 e 143) que permitem que os controladores padrão I2C (102 e 103) se comuniquem através do barramento, o *driver* de transmissão (142) pode ser desconectado eletricamente do barramento se o controlador estiver apresentando problemas em seu funcionamento.

A operação do controlador é baseada no envio periódico de mensagens de teste, que asseguram que o hardware I2C está funcionando. Para garantir o funcionamento adequado, um temporizador (140) conta o tempo transcorrido desde que a última mensagem de teste foi recebida, caso o temporizador expire, uma ação corretiva é tomada.

Figura 3-2 Topologia de controladora I2C proposta na patente US2004/6728908.



Fonte: (FUKUHARA et al., 2004)

A solução foi projetada para atuar em barramentos com um único dispositivo mestre e diversos escravos, desta forma, caso o controlador proposto (100) esteja atuando como dispositivo mestre, as mensagens de testes são enviadas do *core* I2C com endereço base (102) para o segundo *core* I2C com endereço base mais um (103). Ao receber a mensagem teste no segundo core I2C com endereço base mais um (103), que funciona unicamente no modo escravo, o temporizador que conta o tempo entre mensagens de teste (140) é recarregado. Caso o controlador proposto (100) esteja operando no modo escravo, o core I2C padrão que responde pelo endereço base mais um (103) é desabilitado, uma vez que as mensagens teste serão enviadas pelo dispositivo operando no modo mestre conectado ao barramento. A ação exercida no recebimento da mensagem teste pelo controlador (100), através do core I2C com endereço base (102), quando operando no modo escravo, é a mesma de quando está operando

em modo mestre, recarregar o temporizador (140). Caso o temporizador (140) que conta o tempo entre mensagens teste venha a expirar, a unidade de controle (130) toma a ação de desconectar eletricamente o *driver* de transmissão das linhas SCL e SDA (142), além de colocar o controlador em um estado inicial válido. Essa medida impede que um potencial dispositivo defeituoso impacte o funcionamento de todo o barramento I2C a qual ele pertence. Como o *driver* de recebimento permanece ativo, no caso do controlador (100) estar operando no modo escravo, este só poderá reabilitar seu *driver* de transmissão (142) caso o dispositivo mestre do barramento solicitar. Em contrapartida, se o controlador (100) estiver funcionando no modo mestre, junto com o desligamento do *driver* de transmissão (142) um segundo temporizador é inicializado (144), ele é responsável por contar o tempo que o dispositivo mestre ficará incapacitado de atuar no barramento. Isso é necessário por haver somente um dispositivo mestre no barramento, e cabe ao dispositivo mestre comandar o religação dos escravos em caso de falha. Ao expirar o temporizador 144, o dispositivo mestre deve religar o seu *driver* de transmissão e tentar se comunicar novamente. Em dispositivos que operam no modo escravo, o temporizador que conta o tempo em que o controlador está incapacitado de atuar no barramento (144) não é utilizado.

A patente justifica a utilização de um contador de *bytes* transferidos, e se beneficia do uso de *buffers* de transmissão para implementá-los. Sempre que um conjunto de dados está pronto para ser transmitido, este deve ser copiado para um *buffer* (160) dentro do controlador I2C (100), e de lá o controlador assume a tarefa de fazer com que o conjunto de dados chegue a seu destino. Assim, o controlador (100) sabe de antemão a quantidade de dados que irá transmitir pela quantidade utilizada de seu buffer, e está informação é transmitida para o dispositivo destino para que este possa verificar se a quantidade de dados recebidos está de acordo com o esperado.

Para garantir a integridade dos dados trafegados, a solução propõe o uso de CRC-16, uma técnica que gera um valor único, ou quase, a partir de um conjunto de dados. A ideia é que qualquer alteração no conjunto original de dados reflita em um valor de CRC diferente do esperado. No caso do CRC-16, um valor de 16 *bits* é gerado através do polinômio:

$$x^{16} + x^{15} + x^2 + 1 \qquad (2)$$

Todos os dados trafegados no barramento estão presentes no cálculo do CRC, inclusive o endereço do dispositivo escravo. O valor do CRC é calculado por hardware e transmitido nos últimos dois bytes de cada transação I2C.

É sugerido ainda que a solução possa ser utilizada em conjunto com dispositivos escravos padrão, não compatível com as funcionalidades descritas até aqui. Isso implica que o dispositivo mestre, que necessariamente precisa ser compatível, saiba de antemão quais dispositivos escravos não são compatíveis para desativar todas as funcionalidades bem como seus benefícios.

Outros aspectos da patente como comandos customizados de hardware, configuração através de barramento PCI e organização de memória foram omitidos dessa revisão por não serem relevantes em relação ao conteúdo deste trabalho.

3.3 Systems and Methods for Correcting Errors in I2C Bus Communications (US2007/0240019)

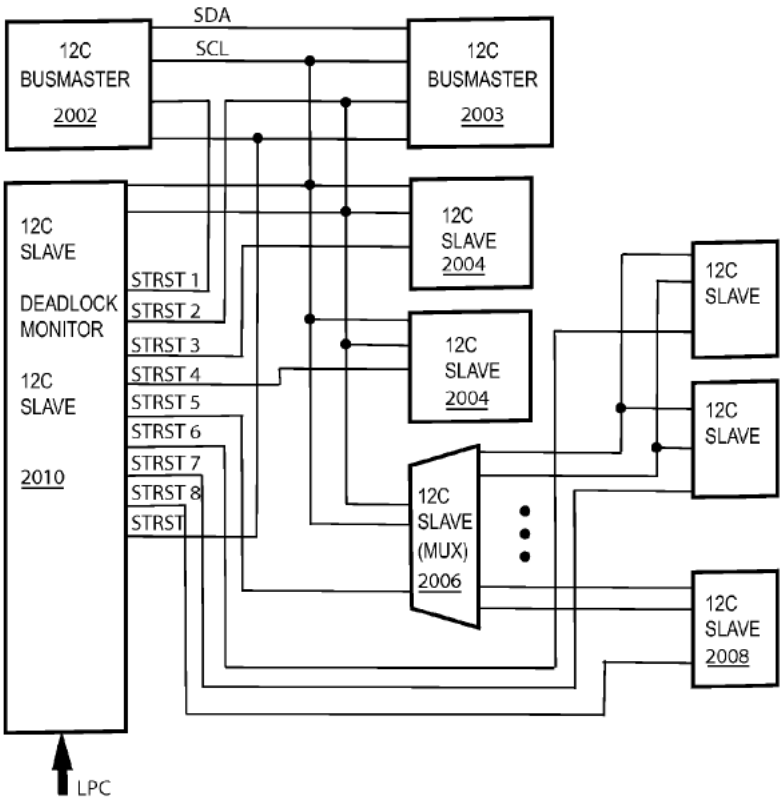
Esta patente, arquivada em 2007, sugere um sistema de controle de barramentos I2C composto por um monitor de barramento e uma unidade de controle chamada *Baseboard Management Controller* (BMC) (Controlador de Gerenciamento da Placa Base, em tradução livre) para ser aplicado em servidores. O monitor de barramento tem a função de monitorar travamentos permanentes em até oito barramentos I2C diferentes (a quantidade de barramentos pode ser aumentada), informar a unidade de controle BMC sobre a condição de cada barramento e de atuar nos dispositivos I2C conectados ao barramento quando comandado pela unidade de controle BMC.

A unidade de controle BMC, que está conectada a todos os barramentos I2C presentes na solução, tem como função realizar atividades de monitoramento e manutenção do servidor. Por exemplo, o BMC pode ler um valor de temperatura. Se o valor lido exceder um valor máximo pré-definido, a unidade de controle BMC comandará os ventiladores para acelerarem o giro de modo a mover o calor para longe dos componentes que estão esquentando (BRADY et al., 2007). Ainda na Figura 3-3, é importante salientar que o monitor de barramento (2010) está conectado ao barramento I2C principal somente, porém ele é capaz de reinicializar através de sinais de hardware (STRST x) qualquer dispositivo I2C alcançável pelos dispositivos mestres, inclusive as ramificações de barramentos após o multiplexador.

Figura 3-3 ilustra a composição da unidade de monitoramento proposta na patente. Pode-se observar uma série de dispositivos I2C mestres e escravos, bem como um multiplexador I2C (2006) conectado a outros dispositivos I2C. O multiplexador I2C é um dispositivo I2C escravo que é capaz de segmentar uma topologia I2C em diversos barramentos diferentes. O dispositivo mestre (2002 ou 2003) envia um comando ao

multiplexador I2C (2006) para acionar o canal do dispositivo I2C 2008, depois o dispositivo mestre pode se comunicar com o dispositivo final (2008) diretamente. A solução de utilizar multiplexadores I2C é adotada largamente pela indústria, pois minimiza a quantidade de dispositivos perdidos caso aconteça uma falha no barramento. Ainda na Figura 3-3, é importante salientar que o monitor de barramento (2010) está conectado ao barramento I2C principal somente, porém ele é capaz de reinicializar através de sinais de hardware (STRST x) qualquer dispositivo I2C alcançável pelos dispositivos mestres, inclusive as ramificações de barramentos após o multiplexador.

Figura 3-3 Monitor de Barramento proposto na patente US2007/0240019



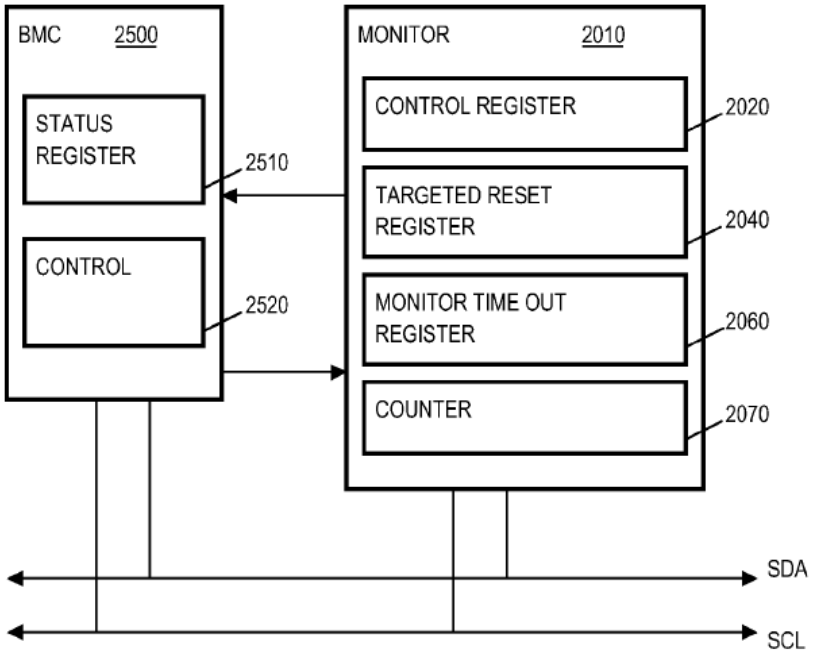
Fonte: (BRADY et al., 2007)

A operação do sistema é bastante simples. Como pode ser observado na Figura 3-4, o monitor de barramento (2010) possui internamente três registradores além de um contador (2070), um registrador de controle (2020), um registrador de reinicialização (2040) e um registrador de tempo máximo de ociosidade (2060) no barramento. A unidade de controle BMC (2500) escreve no registrador de tempo máximo de ociosidade (2060) o tempo máximo que um barramento pode permanecer travado sem ser reportado. A partir daí, o monitor

(2010) utiliza seu contador interno (2070) para monitorar se o barramento sob sua supervisão extrapola o tempo máximo de travamento. Caso um travamento exceda o tempo limite, a unidade de controle (BMC) será notificada pelo monitor. A unidade de controle BMC (2500), por sua vez, deve saber quais dispositivos I2C estavam se comunicando no momento da notificação de travamento e enviar um controle ao monitor para reinicializar os dispositivos apresentando problemas. Para comandar o monitor (2010) a reinicializar um dispositivo através de sua respectiva linha STRSTx (Figura 3-3), a unidade de controle BMC (2500) deve escrever no registrador de reinicialização (2040) do monitor (2010). Este registrador deve possuir o mesmo número de bits que a quantidade de dispositivos I2C presentes no barramento. Somente dois bits do registrador de controle (2020) do monitor de barramento são utilizados, o primeiro para habilitar/desabilitar o monitor, o segundo para forçar a reinicialização dos dispositivos mestres presentes no barramento.

A descrição da patente não deixa claro como a unidade de controle BMC (2500) e o monitor de barramento I2C (2010) se comunicam entre si, ela se limita a dizer que a unidade de controle BMC (2500) é capaz de ler e escrever em registradores internos do monitor (2010).

Figura 3-4 Comunicação Entre BMC e Monitor de Barramento (US2007/0240019)



Fonte: (BRADY et al., 2007)

3.4 Análise Comparativa Entre as Soluções Apresentadas e a Proposta Deste Trabalho

Abaixo estão listados os principais pontos positivos (+) e negativos (-) das soluções discutidas previamente neste capítulo. Por último estão descritos os pontos positivos (+) e negativos (-) da solução proposta neste trabalho.

Tabela 3-1 Análise Comparativa Entre Trabalhos Relacionados

Interface SMBus	US2004/6728908: Bus Protocol Controller with Fault Tolerance	US2007/0240019: Systems and Methods for Correcting Errors in I2C Bus Communications
- Necessita que os dispositivos presentes no barramento respeitem a interface SMBus para usufruir de suas vantagens; - Aceita atuar com dispositivos não compatíveis com SMBus, porém coloca o barramento todo em risco e sem suporte a verificação de integridade de dados nos dispositivos não compatíveis; - Aumento de *overhead* nas comunicações por implementar um nível de protocolo acima da camada de Enlace. - Existe uma grande quantidade de dispositivos I2C que não implementam a interface SMBus. + Excelente controle de integridade de dados;	- Custo de hardware maior que o dobro de um dispositivo I2C padrão, utiliza além de lógica própria, dois *cores* I2C padrão; - Permite somente um dispositivo I2C mestre; - Apesar de ser possível a utilização de dispositivos escravos padrão, nenhuma segurança é garantida a estes que podem impedir (travar) o funcionamento de todos os outros dispositivos presentes no barramento; - Aumento de *overhead* nas comunicações e na quantidade de transações; - Má escalabilidade, todos os dispositivos necessitam implementar a solução; + Habilidade de bloquear um dispositivo apresentando mau funcionamento de interferir no barramento;	- Não permite que um dispositivo seja desconectado do barramento I2C; - Não descreve como o BMC e o Monitor interagem fisicamente; - O BMC precisa saber quais os dispositivos que estavam se comunicando para tomar uma ação (lógica de decisão de qual dispositivo deve ser reinicializado não está no monitor, mas na unidade de controle (BMC)); - Não é uma solução transparente por necessitar de software/firmware dedicado na unidade de controle; + Excelente escalabilidade, um monitor por barramento;

Fonte: Autor

4 PROPOSTA E IMPLEMENTAÇAO DO MONITOR DE BARRAMENTO I2C

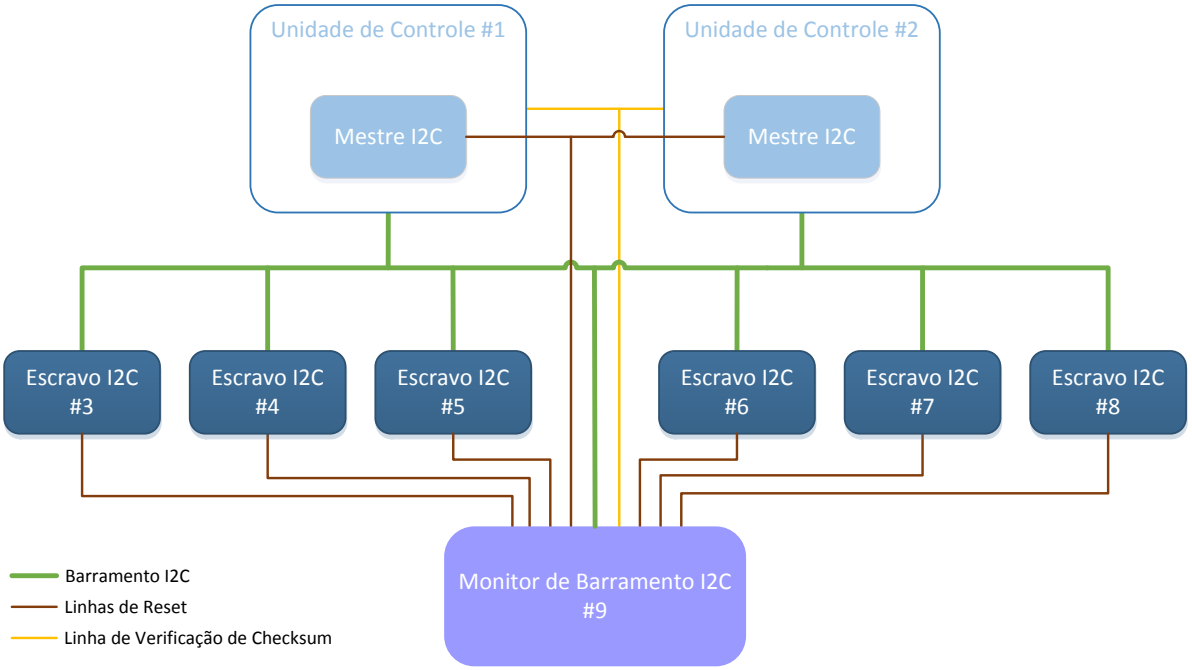
O Monitor de Barramento I2C proposto neste trabalho visa elevar a robustez do protocolo I2C sem grandes mudanças em projetos de hardware ou software. Ele ainda possui um custo muito baixo em termos de área e impacto em desempenho quando comparado a soluções equivalentes e pode ser inserido em produtos e projetos já existentes.

Dentre as principais funcionalidades do Monitor de Barramento I2C estão:

- Verificação de integridade de dados: No fim de cada transação realizada no barramento I2C sendo monitorado, o Monitor de Barramento I2C sinaliza através da Linha de Verificação de *Checksum* (Figura 4-1) se os dados trafegados na última transação estão íntegros. Cabe às aplicações que controlam os dispositivos mestres utilizar ou não esta indicação.

- Recuperação de Falha no Barramento: O Monitor de Barramento I2C é capaz de detectar falhas permanentes no barramento I2C através do monitoramento das linhas SCL e SDA. No caso de detecção de falha permanente, o último dispositivo escravo a ser endereçado e os *cores* dos dispositivos mestres serão reinicializados (Figura 4-1).

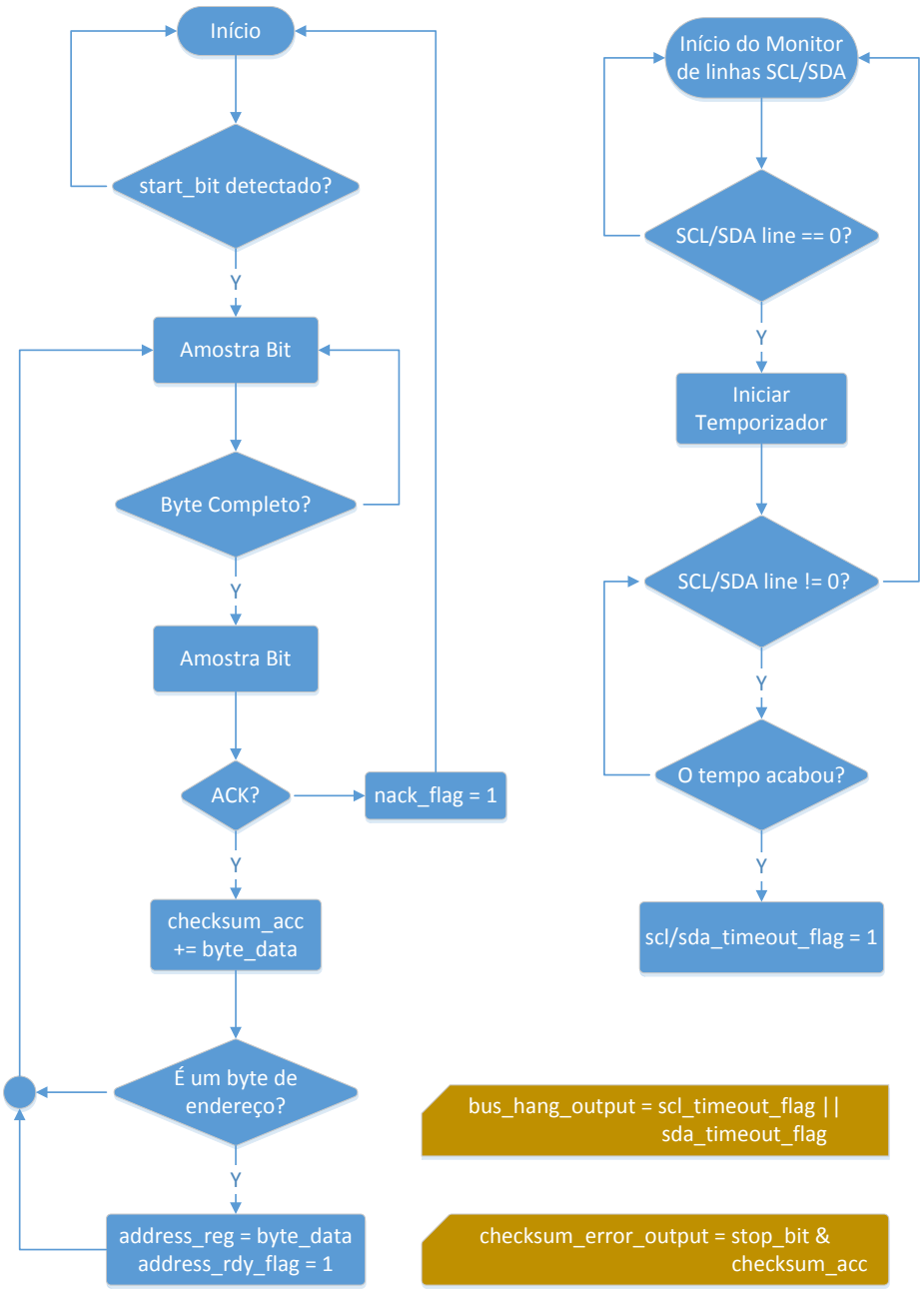
Figura 4-1 Topologia de barramento I2C utilizando o Monitor de Barramento I2C



Fonte: Autor

Como pode ser observada nos fluxogramas da Figura 4-2, toda transação iniciada no barramento I2C sendo monitorado é detectada pelo Monitor de Barramento I2C através da primitiva de *start_bit*. O primeiro *byte* que segue a primitiva de *start_bit* é obrigatoriamente o endereço do dispositivo escravo com o qual o dispositivo mestre deseja se comunicar.

Figura 4-2 Fluxograma de operação do Monitor de Barramento I2C



Fonte: Autor

Caso o monitor detecte uma primitiva de *Acknowledgement* após o *byte* de endereço recém-lido, ele o guardará em um registrador interno (*address_reg*). O *byte* de endereço é também somado ao acumulador do *checksum*, este acumulador é limpo sempre que uma nova

transação é iniciada. Na sequência, todos os demais *bytes* trafegados no barramento e confirmados através da primitiva de *Acknowledgement* são somados ao acumulador de *checksum* de 8 *bits* até que uma primitiva de *stop_bit* seja detectada.

Para a verificação de integridade de dados funcionar corretamente, o último *byte* de qualquer transação I2C deve conter o valor do cálculo do *checksum* de toda a transação. Em uma operação de escrita, o dispositivo mestre deve calcular e enviar o valor do *checksum*. No caso de uma operação de leitura, a informação *de checksum* deve ser enviada pelo dispositivo escravo. O cálculo do *checksum* é o complemento da soma de todos os *bytes* incluindo o *byte* de endereço do dispositivo escravo. Dessa forma a soma de todos os *bytes* de uma transação, incluindo o *byte* de *checksum*, deve ser sempre zero (Figura 4-3).

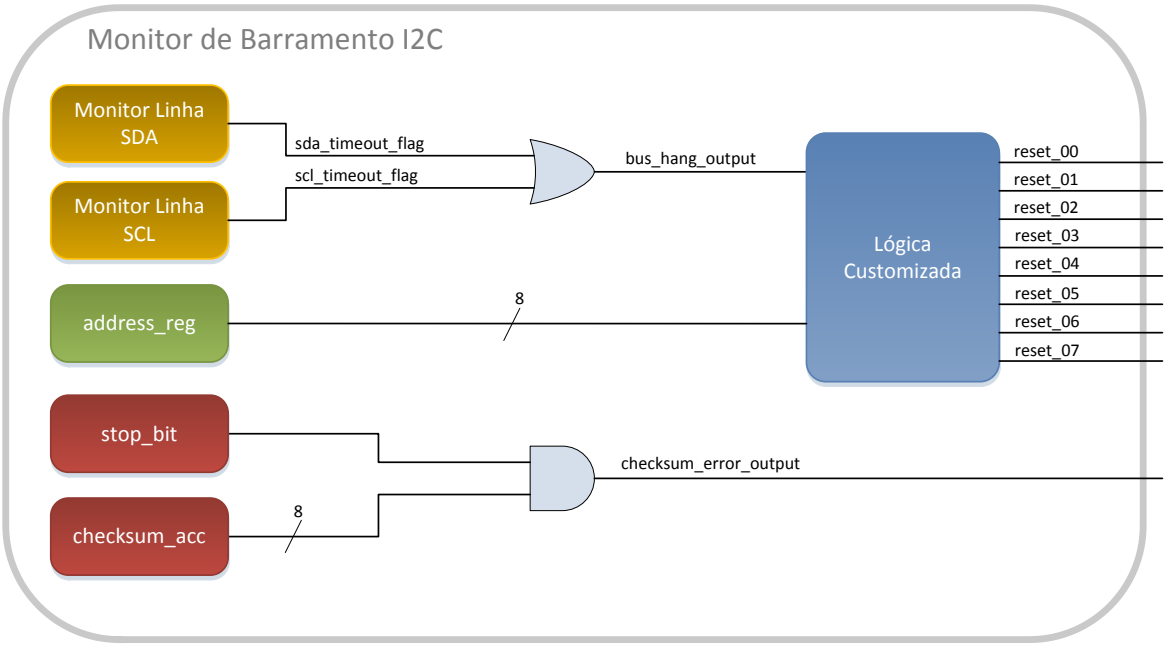
Figura 4-3 Transação I2C com cálculo de *checksum*

ΣTransação = 0x10 + 0x01 + 0x02 + 0x03 + 0xEA = 0x00

Fonte: Autor

Para garantir que o barramento possa ser recuperado de falhas permanentes, o Monitor de Barramento I2C monitora constantemente o estado das linhas SCL e SDA. O fluxo exemplificado à direita na Figura 4-2 é individualizado e independente para cada uma das linhas, SCL e SDA. No momento em que uma linha é acionada para o estado logico baixo um temporizador é inicializado. Este temporizador é limpo e desativado caso a linha volte ao estado lógico alto. Caso o temporizador venha a expirar, um sinal indicando o travamento do barramento é acionado (bus_hang_output). Internamente ao monitor, a sinalização de travamento de barramento é utilizada em conjunto com o registrador que armazena o último endereço trafegado no barramento (address_reg) para associá-lo, através de uma lógica customizada (*glue logic)*, à linha de reset na qual o dispositivo escravo está conectado (Figura 4-4).

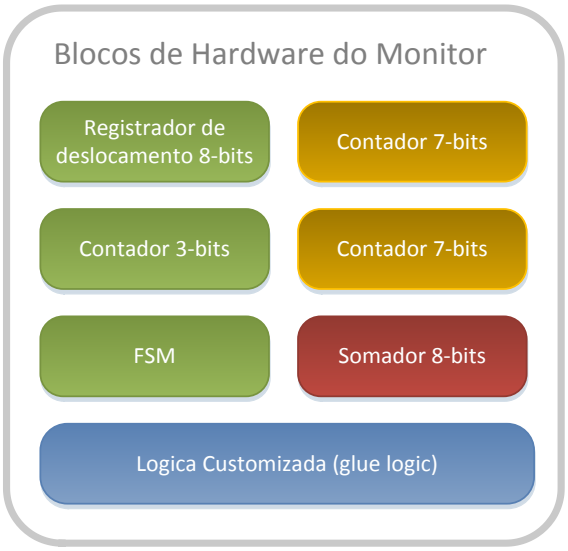
Figura 4-4 Detecção e recuperação de falhas permanentes e sinalização de erro de *checksum*



Fonte: Autor

A Figura 4-5 divide as funcionalidades do Monitor de Barramento I2C em blocos de hardware, a implementação da solução proposta por este trabalho utiliza blocos equivalentes. Os blocos na cor verde são responsáveis por decodificar o protocolo I2C, extraindo as informações necessárias do barramento. O bloco FSM (*Finite State Machine*, Máquina de Estados Finita em tradução livre), implementa uma máquina de estados equivalente à de um dispositivo I2C escravo, porém com algumas customizações. Os blocos na cor amarela são utilizados no mecanismo de detecção de falhas permanentes no barramento, os contadores fazem parte dos monitores de linha apresentados na Figura 4-4. O somador, bloco na cor vermelha é usado na realização do cálculo do *checksum* em cada transação I2C. Por último está o bloco azul, que contém lógicas customizadas. Este bloco representa a conversão do endereço I2C em uma linha de reset de um dispositivo I2C e também as conexões entre os componentes da solução.

Figura 4-5 Separação do Monitor de Barramento I2C em blocos de Hardware

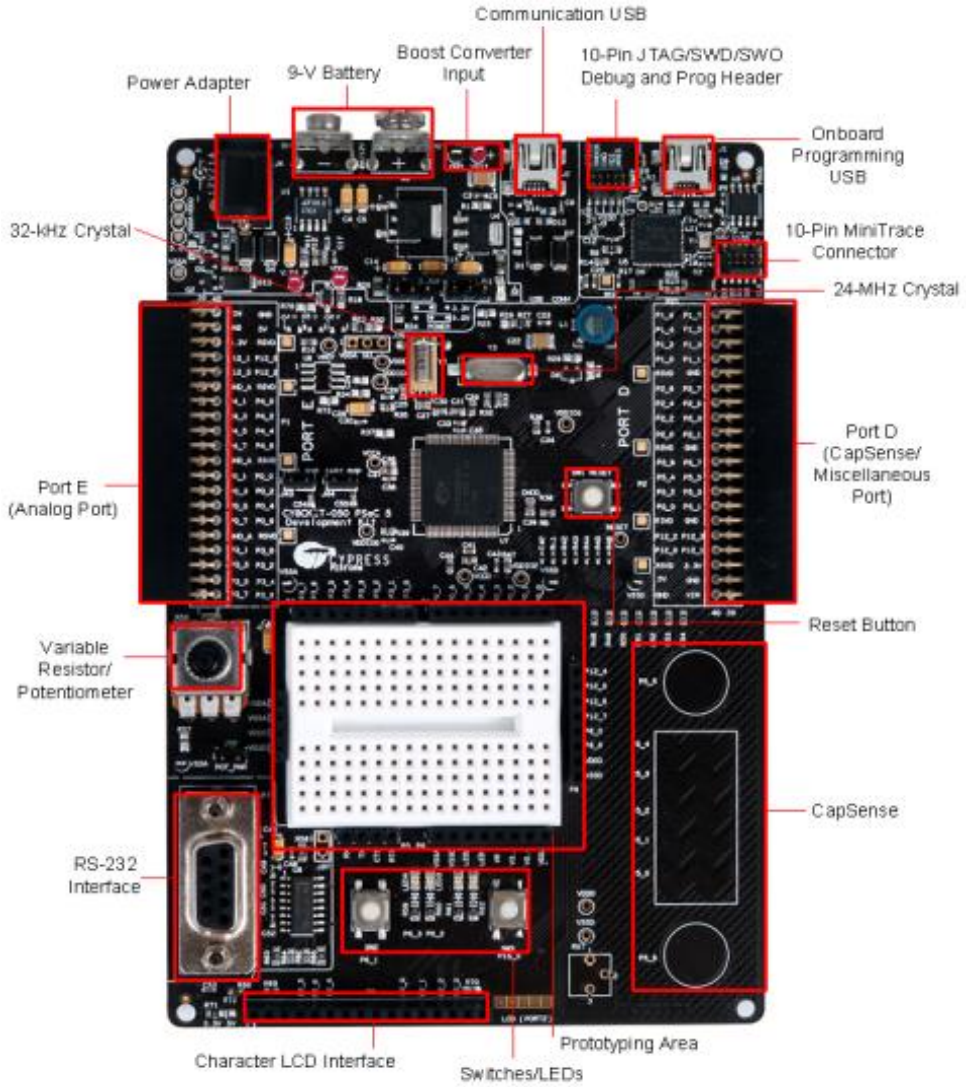


Fonte: Autor

4.1 Objeto de Estudo PSoC5

Para a realização de testes de desenvolvimento e simulações foi utilizado o kit de desenvolvimento CY8CKIT-050 da Cypress Semiconductor Corp (CYPRESS, 2012a). Este kit utiliza o SoC PSoC5 (CYPRESS, 2012b), também da Cypress, como componente principal de controle. O kit ainda contém duas conexões USB, a primeira voltada para programação e depuração no componente PSoC5 e a segunda para ser utilizada no desenvolvimento, podendo inclusive, servir como uma interface serial auxiliar uma vez que a Cypress dispõe um componente em *software* que efetua a conversão USB-UART no PSoC5. Além disso, a placa do kit também possui um *driver* conversor UART-RS232, chamado MAX3232 (INSTRUMENTS, 2009), ligado à um conector DB9 (VALCOSS ELECTRONICS, 2012), o que facilita a criação de uma interface de depuração que possa ser controlada através de um PC. Uma pequena *proto-board* localizada no centro da placa facilita a interconexão dos próprios sinais pertencentes a placa, disponíveis através de três barras de pinos, bem como a interconexão com dispositivos externos, como circuitos integrados ou dispositivos de medição e depuração como o osciloscópio.

Figura 4-6 Apresentação do kit de desenvolvimento do PSoC5



Fonte: (CYPRESS, 2011)

4.2 APSoC Cypress PSoC5

Além do *core* ARM Cortex-M3, o PSoC5 possui periféricos de hardware analógico e digital, ambos programáveis.

O hardware analógico programável é composto por quatro blocos iguais com as seguintes funcionalidades:

- Amplificador de ganho programável (*PGA*)
- Amplificador de transimpedância (*TIA*)
- *Mixer*
- Circuito de amostragem e armazenamento ("*Sample and hold circuit*")

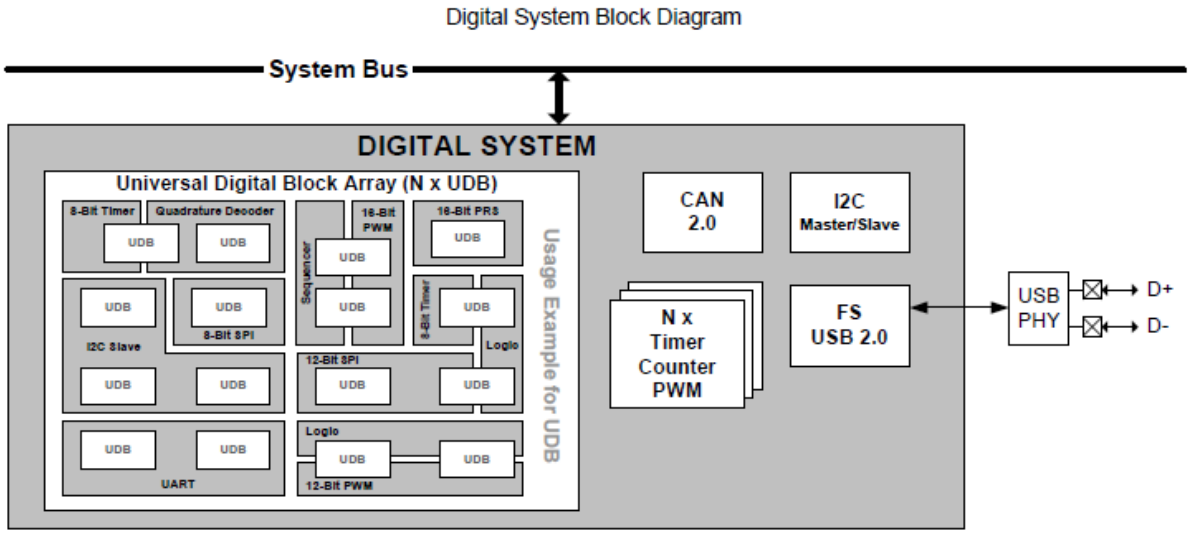
Para este trabalho o hardware analógico programável presente no chip não foi utilizado. Outra característica do PSoC5 são os *fixed functions blocks* (blocos de função fixa, em tradução livre) também presentes em seu hardware. Estes blocos de hardware estão presentes no componente e podem ser configurados apenas pela aplicação caso sejam utilizados. A lista a seguir apresenta os blocos de hardware fixos presentes no PSoC5 (CYPRESS, 2012b):

- 4 x temporizadores de 16 bits
- 1 x canal *I2C* (mestre ou escravo)
- Controlador de *DMA* de 24 canais
- Processador de filtro digital de 80MHz
- 1 x *USB* 2.0
- 1 x canal *CAN*
- 1 x 12-bit delta-sigma *ADC*
- 2 x 12-bit *SAR ADC*
- 4 x 8-bit *DAC*
- 4 x comparadores
- 4 x opamps

4.2.1 Hardware Digital Programável no PSoC5

Diferentemente de uma *FPGA* ou uma *CPLD*, o PSoC5 utiliza blocos de hardware digital programáveis (*UDB – Universal Digital Block*, bloco digital universal em tradução livre) para configurar o circuito digital. Estes blocos são conectados entre si, fazendo uma rede de 24 blocos digitais disponíveis para o usuário. A Figura 4-7 exemplifica como o vetor de *UDBs* deve ser utilizado. À esquerda da figura, a união de blocos *UDBs* compõe diferentes componentes com funcionalidades específicas como, temporizadores, controlador *I2C*, controlador *SPI*, *UART*, entre outros. À direita da figura estão representados os blocos de função fixa já discutidos no início do capítulo.

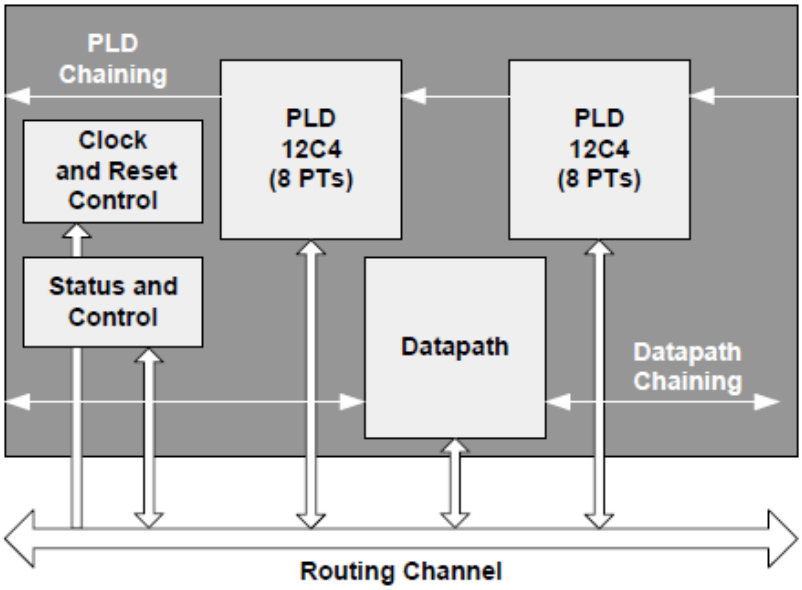
Figura 4-7 Rede de blocos *UDB* exemplificado



Fonte: (CYPRESS, 2012c)

Cada *UDB* é composto por duas *PLDs* (*Programable Logical Device*, dispositivo lógico programável em tradução livre), controle de *clock* e *reset*, um módulo de *status*/controle e um *datapath* (CYPRESS, 2012c).

Figura 4-8 Bloco UDB exemplificado



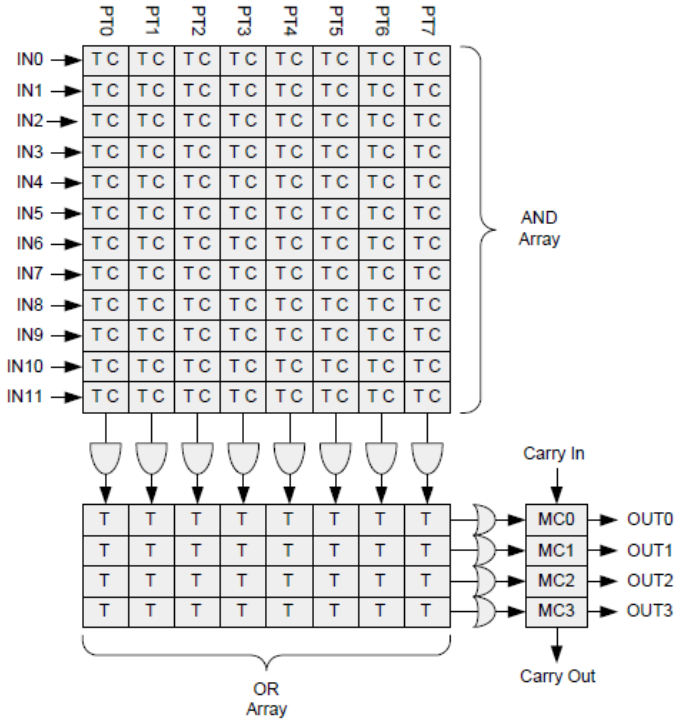
Fonte: (CYPRESS, 2012c)

As *PLDs* podem ser utilizadas para implementar máquinas de estado, para realizar condicionamento de entradas e saídas e também para criar *LUTs* (*Look Up Tables*, tabelas de

consulta em tradução livre). Elas também podem ser utilizadas para realizar funções aritméticas, controlar a sequência do *datapath* e gerar *status*.

Cada *PLD* possui 12 entradas que alimentam 8 *product terms* (Matriz de termos *AND*). Cada entrada pode ser diretamente associada a matriz de termos ou seu complemento também pode ser utilizado. A saída dos 8 termos "*AND*" é direcionada para a entrada de uma segunda matriz de termos, agora "*OR*". Esta segunda matriz composta por quatro termos alimenta as quatro macro células presentes em cada *PLD* respectivamente (CYPRESS, 2012c).

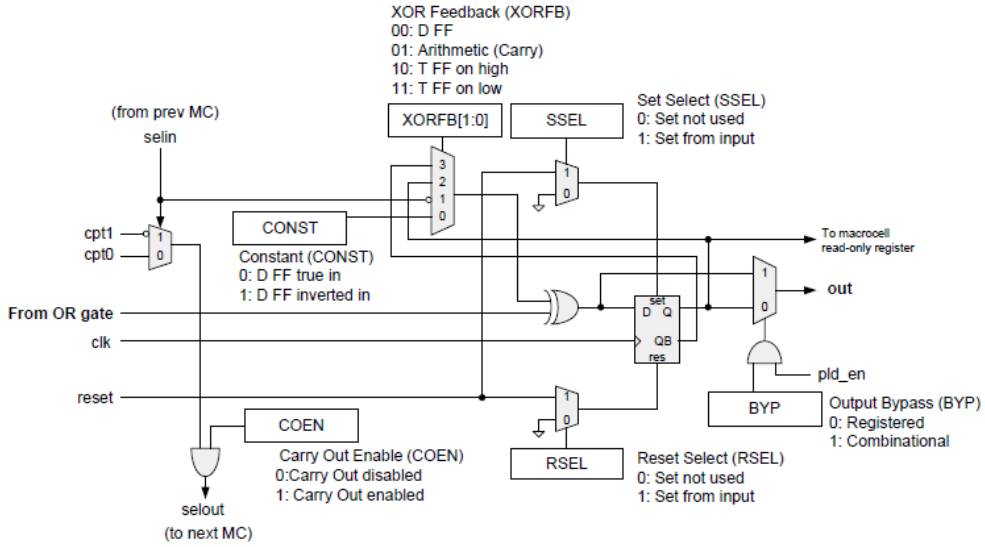
Figura 4-9 Diagrama interno de uma das duas *PLDs* contidas em um bloco *UDB*



Fonte: (CYPRESS, 2012c)

A macro célula controla a saída da matriz de termos "*OR*" para o barramento de roteamento responsável pela comunicação entre *UDBs*. Ela permite que a saída seja registrada ou combinacional. No caso de registrada, pode ser utilizado um *flip-flop* D ou *flip-flop* T (*Toggle*, alternado em tradução livre). Além disso, o registrador de saída pode ser configurado para um estado específico durante a inicialização do sistema ou assincronamente, através de um sinal roteado pelo barramento, durante operação (CYPRESS, 2012c).

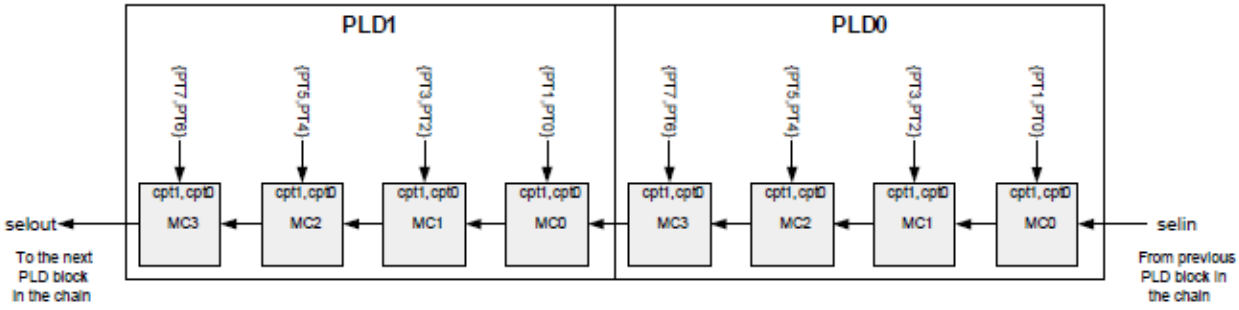
Figura 4-10 Macro célula



Fonte: (CYPRESS, 2012c)

As *PLDs* são conectadas em cadeia por ordem de endereçamento. O "*carry in*" ("*selin*") da *PLD0* é alimentado pelo sinal "*selout*" do *UDB* anterior. E o sinal "*selout*" da *PLD1* alimenta o "*selin*" do *UDB* seguinte (CYPRESS, 2012c).

Figura 4-11 Cadeia de ligação das PLDs

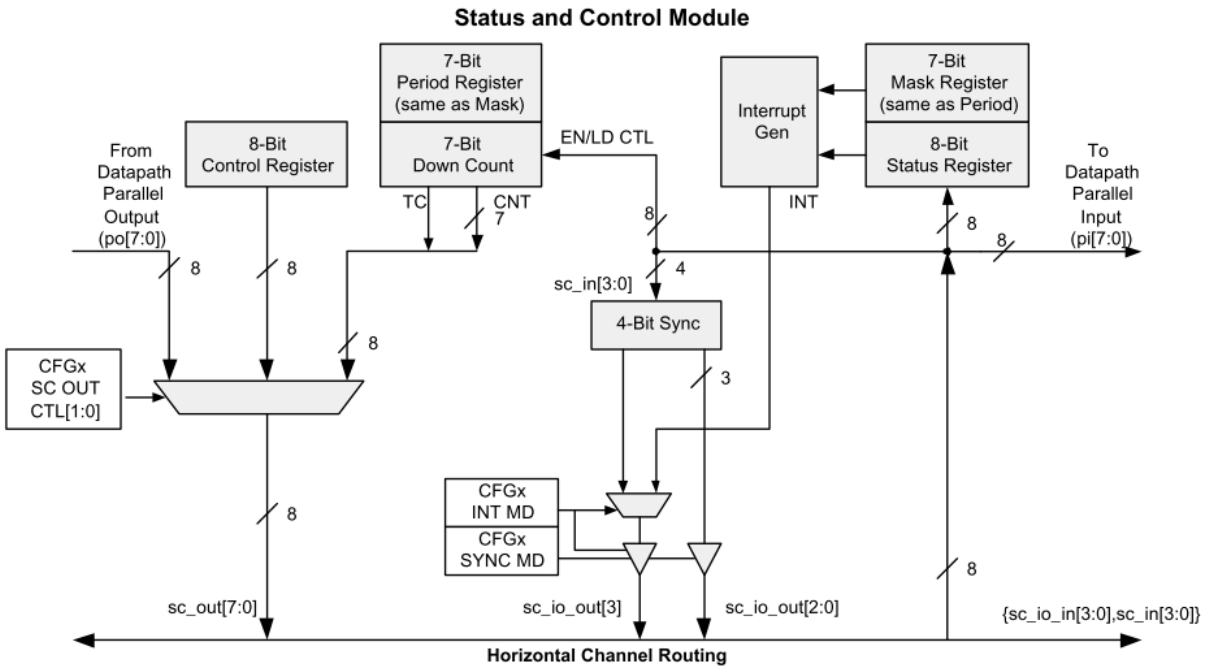


Fonte: (CYPRESS, 2012c)

O bloco de controle de *clock* e *reset* é responsável por selecionar o *clock* de entrada do bloco e habilitar o *reset* do bloco respectivamente. O módulo de *status*/controle (Figura 4-12) é o meio pelo qual o firmware da *CPU* (ARM Cortex-M3) se comunica com o *UDB* e vice-versa. Através da entrada de *status* o firmware é capaz de ler informações contidas no barramento de roteamento, enquanto a saída de controle permite que o firmware escreva informações no barramento. Apesar da função principal do módulo de *status*/controle ser a comunicação entre *CPU* e *UDBs*, outras configurações importantes podem ser utilizadas

dado ao hardware presente no módulo e sua conexão direta com matriz de roteamento de sinais.

Figura 4-12 Blocos lógicos que compõem o módulo de *status*/controle



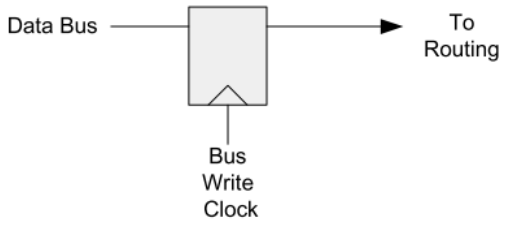
Fonte: (CYPRESS, 2012c)

O módulo de *status*/controle pode funcionar nas seguintes configurações:

- Entrada de *Status* – O estado dos sinais roteados podem ser amostrados e capturados como *status* e lidos pela *CPU* ou por *DMA*.
- Saída de Controle – A *CPU* ou *DMA* podem escrever no registrador de controle para modificar o estado do roteamento.
- Entrada Paralela – Direcionada à entrada paralela do *datapath*.
- Saída Paralela – Sinais provenientes da saída paralela do *datapath*.
- Modo Contador – Neste modo o registrador de controle opera como um contador regressivo de 7 *bits* com período programável e recarregamento automático. Entradas roteadas podem ser configuradas para controlas ambos os sinais de habilitar (*enable*) e recarregar (*reload)* do contador. Quando este modo está habilitado operações com o registrador de controle ficam indisponíveis.
- Modo Sincronismo – Neste modo, o registrador de *status* opera como um sincronizador duplo de 4 *bits*. Ao habilitar este modo, as operações com o registrador de *status* se tornam indisponíveis.

Este trabalho faz uso direto do módulo de *status*/controle em dois dos seis modos disponíveis, saída de controle e contador de 7 *bits*. Como pode ser observado na Figura 4-13, quando o registrador de controle é escrito pela *CPU* ou por *DMA* através do barramento de dados (*Data Bus*), a saída do registrador de controle aciona o canal de roteamento (*Routing*) naquele ciclo de escrita (*Bus Write Clock*).

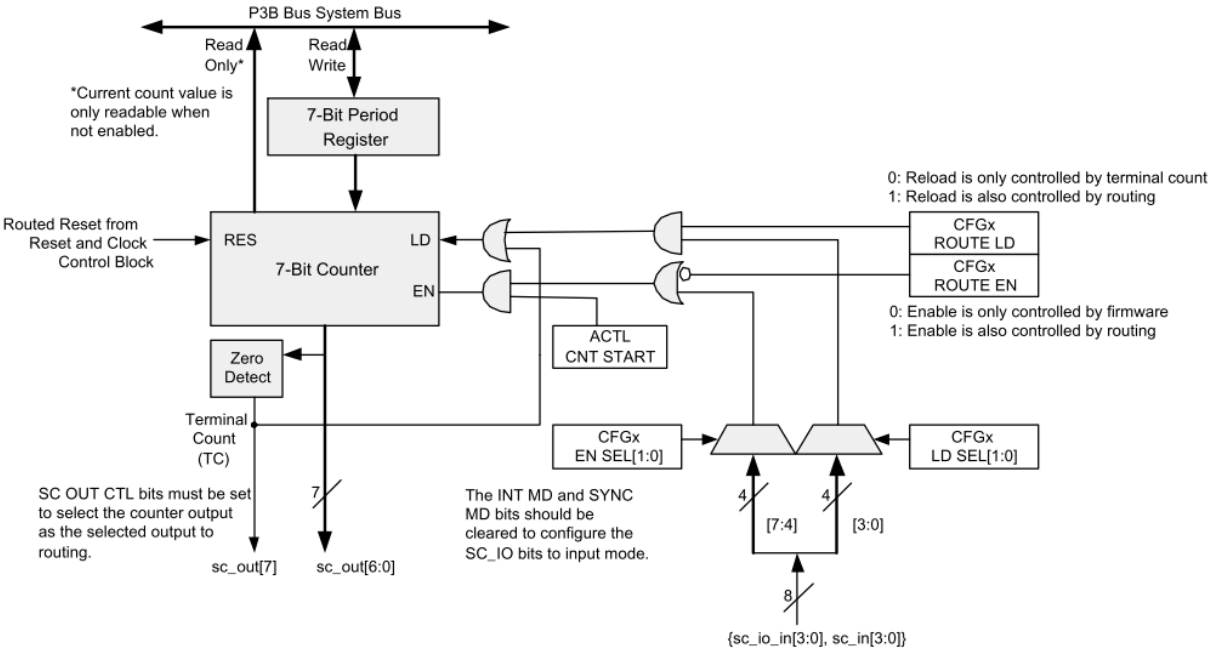
Figura 4-13 Módulo *status*/controle, funcionamento do controle direto.



Fonte: (CYPRESS, 2012c)

A **Erro! Fonte de referência não encontrada.** mostra a operação do módulo *status*/controle quando na configuração de contador regressivo de 7 *bits*. Neste modo um contador é exposto para utilização do firmware (*CPU*) ou por blocos *UDB*, como utilizado neste trabalho.

Figura 4-14 Módulo *status*/controle no modo contador.



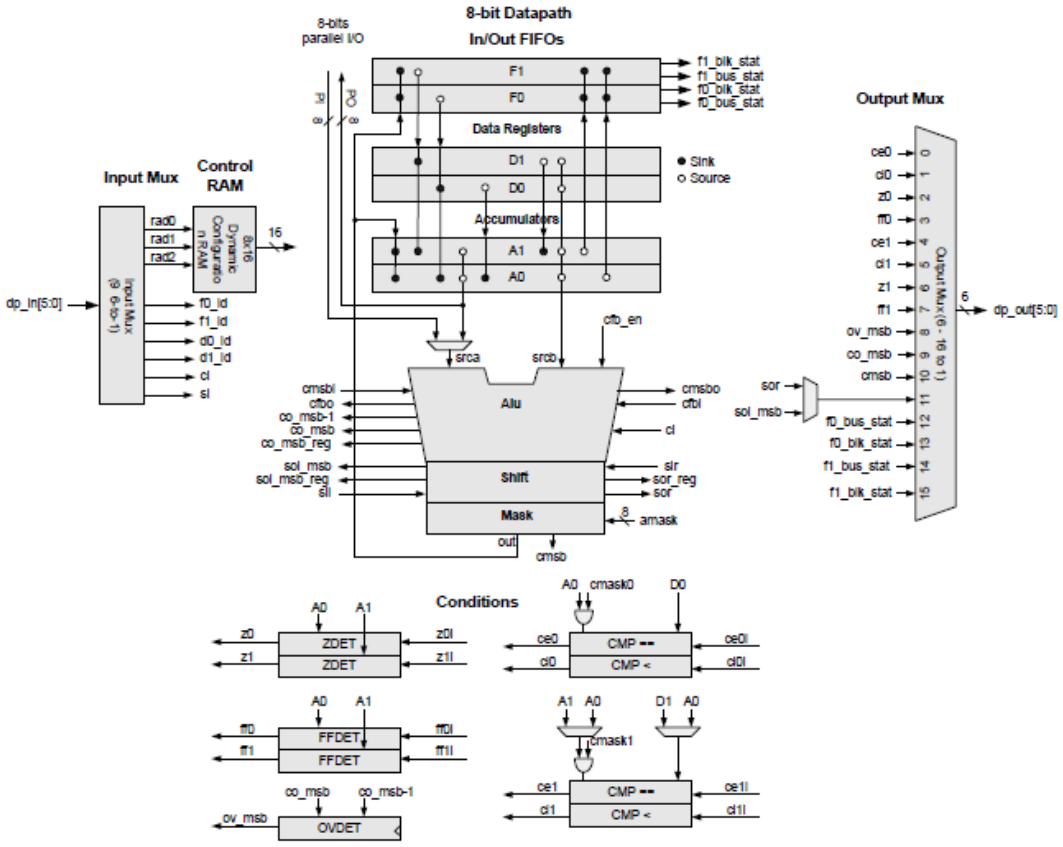
Fonte: (CYPRESS, 2012c)

Das funcionalidades exibidas na figura é importante salientar a capacidade de habilitar (*EN*) e recarregar (*LD*) o contador através do canal de roteamento, que pode ser acessado pelos

UDBs. Além disso, o sinal de detecção de zero (*zero detect*) também é direcionado ao canal de roteamento e pode ser usado na lógica contida nas *PLDs* dos *UDBs*.

O *datapath* presente em cada *UDB* é *single-cycle* (ciclo único em tradução livre), ele é composto por uma ULA (unidade lógica e aritmética), quatro registradores, duas *FIFOs*, comparadores e geradores de condições. A memória de instruções do *datapath* consiste em uma pequena memória volátil que permite a utilização de até 8 instruções diferentes por *datapath*. Cada instrução utilizada deve ser criada pelo usuário através dos sinais de controle do *datapath* (CYPRESS, 2012c). A Figura 4-15 mostra o diagrama de blocos do *datapath* contido em cada *UDB*.

Figura 4-15 Diagrama do *Datapath* de um *UDB*



Fonte: (CYPRESS, 2012c)

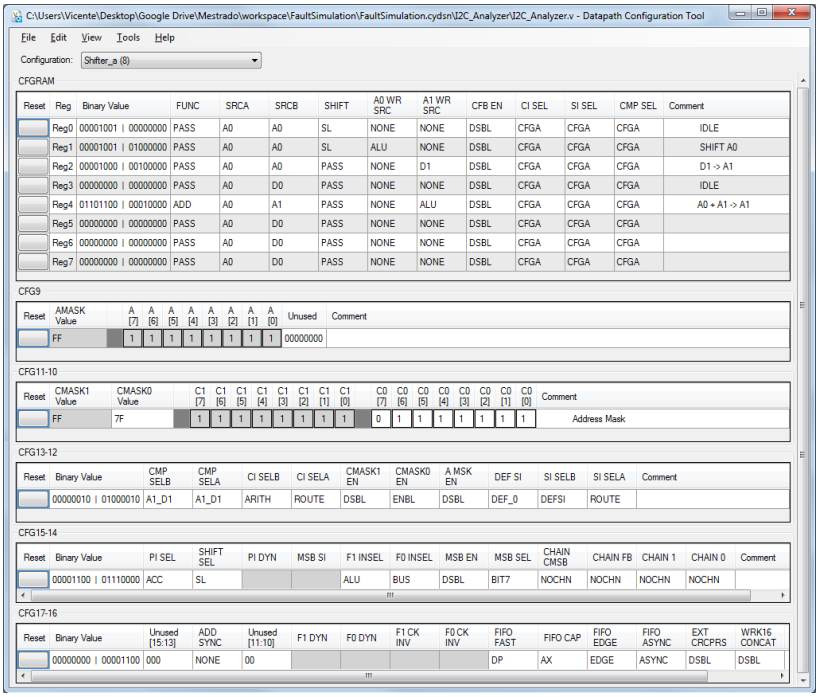
Os registradores A0 e A1 são acumuladores, eles podem funcionar como origem, destino ou ambos em uma operação da ULA. Eles também podem ser carregados com os valores dos registradores de dados D0 e D1 ou pelas *FIFOs*.

Os registradores de dados D0 e D1 normalmente armazenam constantes ou variáveis temporárias utilizadas em algoritmos.

Os registradores *FIFO0* e *FIFO1* possuem 4 bytes cada. Sua finalidade é funcionar como buffer serial de entrada ou saída. Sinais de controle específicos como status de leitura e escrita, fila cheia, fila vazia, entre outros, auxiliam a utilização dos registradores *FIFO*.

A instanciação do *datapath* ocorre dentro do código *HDL* (*Verilog)* do componente. O *datapath* é configurado através de um editor/gerador de código específico chamado "*Datapath Config Tool*" criado pela Cypress. O código gerado pelo editor é colado no arquivo *HDL* criado pelo usuário, funcionando como um componente externo.

Figura 4-16 *Datapath Configuration Tool* (Ferramenta de Configuração do *Datapath*)



Fonte: (CYPRESS, 2012c)

É importante salientar que todas as configurações do *datapath*, bem como as configurações das *PLDs* estão armazenadas em memória flash. Durante o processo de inicialização o conteúdo da memória flash é copiado para as memórias tipo *SRAM* (*Static Random Memory Access*, Memória Estática de Acesso Randômico em tradução livre).

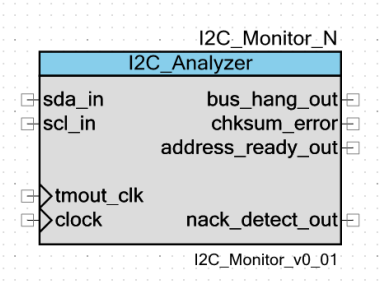
4.2.2 Implementação do Monitor de Barramento I2C em um PSoC5

Este trabalho propõe a utilização de um core especificamente voltado para o monitoramento e recuperação do barramento caso seja detectada uma falha não recuperável.

O monitor sinaliza qualquer falha detectável à CPU, que por sua vez, consolida as falhas e pode manter um histórico da qualidade do barramento.

Dentre as falhas detectáveis nessa primeira versão do core estão: Integridade do *payload* (*checksum*), responsividade dos dispositivos presentes no barramento (*nack*) e travamento permanente do barramento (*bus hang*). Para evitar qualquer tipo de interferência no barramento o monitor I2C se conecta ao mesmo através de duas entradas digitais, sda_in e scl_in, sem *drivers* de saída, e não através de linhas bidirecionais (entradas e saídas) como em um dispositivo I2C padrão (Figura 4-17). Isso acontece pelo fato do monitor I2C, por definição, não atuar no barramento em qualquer circunstância, sua atuação é feita unicamente através dos dispositivos conectados ao barramento.

Figura 4-17 Bloco Lógico do Componente I2C Monitor



Fonte: Autor

A Figura 4-17 mostra o bloco lógico do componente implementado no PSoC5 , nele é possível se observar os sinais de entrada no lado esquerdo do bloco e os sinais de saída no lado direito.

A entrada *clock* (relógio em tradução livre) define a frequência de amostragem dos sinais do barramento. Já a entrada *tmout_clk*, é necessária para utilização de uma base de tempo maior para a contagem do tempo de time out (tempo limite em tradução livre) de travamento de barramento. O sinal de *clock tmout_clk* alimenta os temporizadores de monitoramento das linhas *scl* e *sda* apresentados no início deste capítulo (Figura 4-2). Por limitações de hardware que serão mostradas na sequência deste trabalho, um divisor de frequência com a dimensão necessária não é viável.

A saída *bus_hang_out* (saída barramento travado em tradução livre) é acionada sempre que o monitor detecta um travamento permanente do barramento, isto é, sempre que uma das linhas SCL ou SDA permanecer em nível lógico baixo por um tempo maior do que o tempo limite. Para o experimento foi utilizado um tempo limite de 3 ms e volocidade de transmissão de 100kbps.

A saída *checksum*_error (erro de *checksum* em tradução livre) permanece acionada enquanto a soma de todos os bytes da última transação for diferente de zero. Este sinal deve ser verificado pela aplicação (ou por qualquer dispositivo I2C) ao final de cada transação I2C.

A saída *address_ready_out* (saída endereço pronto em tradução livre) informa que o byte da transação corrente correspondente ao endereço I2C do dispositivo escravo com o qual o mestre está tentando se comunicar está pronto no buffer do monitor. Este sinal é utilizado para disparar uma transferência por DMA que copia o dado do buffer do monitor I2C para um registrador de controle.

A saída *nack_detect_out* (saída detecção de nack em tradução livre) serve para a aplicação conseguir medir a qualidade das comunicações através do barramento. Cada borda de subida deste sinal significa uma transação finalizada com falha.

Para um melhor aproveitamento dos recursos disponíveis no PSoC 5, o monitor I2C foi projetado para utilizar ao máximo os recursos do *datapath*, poupando assim os escassos termos disponíveis nas *PLDs*. Além disso, nos preocupamos em utilizar somente um *datapath* na concepção do monitor I2C para o mesmo ser justificável em termos de recursos utilizados quando comparados com um dispositivo I2C escravo ou mestre implementado em blocos *UDBs* pela própria fabricante Cypress.

Figura 4-18 Instruções criadas no *datapath*

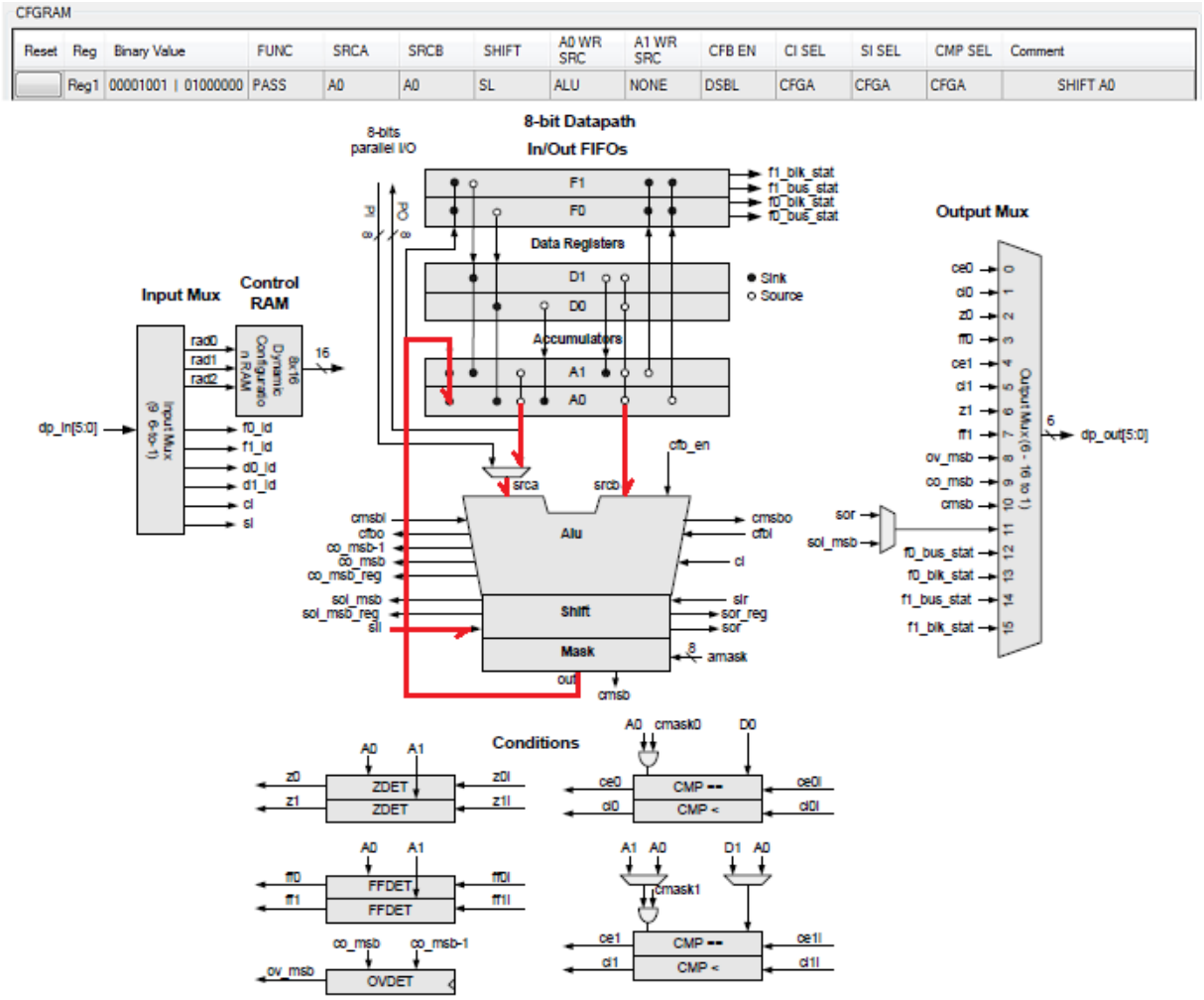
CFGRAM

Reset	Reg	Binary Value	FUNC	SRCA	SRCB	SHIFT	A0 WR SRC	A1 WR SRC	CFB EN	CI SEL	SI SEL	CMP SEL	Comment
	Reg0	00001001 | 00000000	PASS	A0	A0	SL	NONE	NONE	DSBL	CFGA	CFGA	CFGA	IDLE
	Reg1	00001001 | 01000000	PASS	A0	A0	SL	ALU	NONE	DSBL	CFGA	CFGA	CFGA	SHIFT A0
	Reg2	00001000 | 00100000	PASS	A0	A0	PASS	NONE	D1	DSBL	CFGA	CFGA	CFGA	D1 -> A1
	Reg3	00000000 | 00000000	PASS	A0	D0	PASS	NONE	NONE	DSBL	CFGA	CFGA	CFGA	IDLE
	Reg4	01101100 | 00010000	ADD	A0	A1	PASS	NONE	ALU	DSBL	CFGA	CFGA	CFGA	A0 + A1 -> A1
	Reg5	00000000 | 00000000	PASS	A0	D0	PASS	NONE	NONE	DSBL	CFGA	CFGA	CFGA	
	Reg6	00000000 | 00000000	PASS	A0	D0	PASS	NONE	NONE	DSBL	CFGA	CFGA	CFGA	
	Reg7	00000000 | 00000000	PASS	A0	D0	PASS	NONE	NONE	DSBL	CFGA	CFGA	CFGA	

Fonte: (CYPRESS, 2013) modificado pelo autor.

A figura acima foi retirada do software "*Datapath Configuration Tool*" da Cypress, esse foi o software utilizado para a configuração do *datapath* do Monitor de Barramento I2C. Como pode ser visto na Figura 4-18, cinco instruções foram criadas. A primeira instrução ocupa o registrador 0 (Reg0), o que significa que seu *opCode* (código de operação) é 0x00. A instrução 0x00 não executa nenhuma função, o mesmo vale para a instrução 0x03, que ocupa o registrador 3 (Reg3). A instrução 0x01, que ocupa o registrador 1 (Reg1), tem como função deslocar o valor de A0 (acumulador 0) uma posição para a esquerda. Foi usado como referência para a implementação do registrador de deslocamento (*shift register*) no *datapath* o componente I2C_v3.30 (CYPRESS, 2010), presente na biblioteca de componentes do software PSoC Creator 3.0 (CYPRESS, 2015).

Figura 4-19 Instrução *SHIFT LEFT* (Deslocamento para a esquerda) do *datapath*



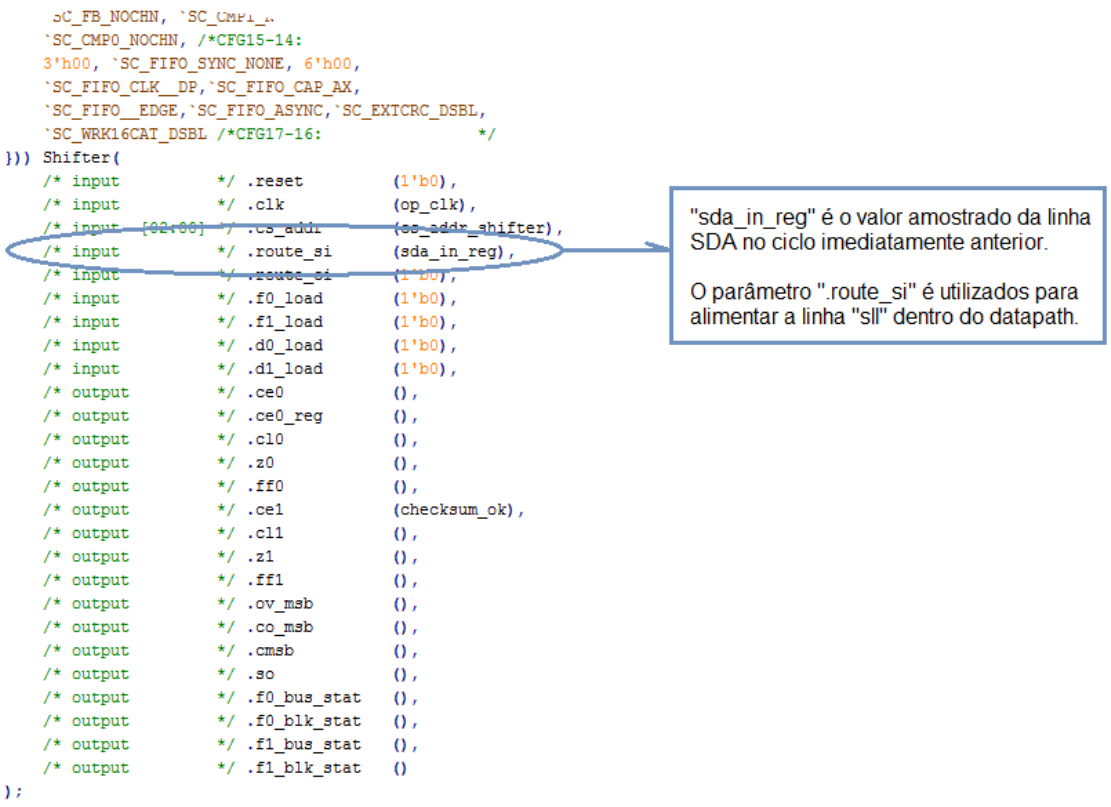
Fonte: (CYPRESS, 2013) Modificado pelo autor.

A Figura 4-19 mostra o caminho do dado e a sequência de controles necessários para deslocar em um *bit* o registrador A0 e armazená-lo no próprio registrador. Após deslocar o registrador de 8 *bits* A0 para esquerda em uma posição, o valor do sinal "sll" (*Shift Left Line*, linha de deslocamento à esquerda em tradução livre) é utilizado para preencher a posição do bit menos significativo. O valor do sinal *sll* descrito acima é fornecido como uma entrada do componente "*datapath*" que por sua vez é instanciado no código HDL. Desconsiderando os sinais de controle de início e fim da transação, o Monitor de Barramento I2C deve, basicamente, utilizar o "*shifter*" sempre que for amostrada uma borda de subida na linha do barramento SCL. Nesse caso, o "*shifter*" deve amostrar imediatamente o valor da linha SDA e armazená-lo em um registrador.

Seguindo esta lógica, após oito bordas de subida da linha SCL um byte estará pronto para ser lido no registrador A0 do *datapath*. A Figura 4-20 mostra como o componente

datapath é instanciado no código HDL. A integração da lógica contida nas *PLDs* (HDL) com o *datapath* é feita através da passagem de parâmetros para componente instanciado.

Figura 4-20 Instanciação do componente *datapath* no código *HDL* (Verilog)



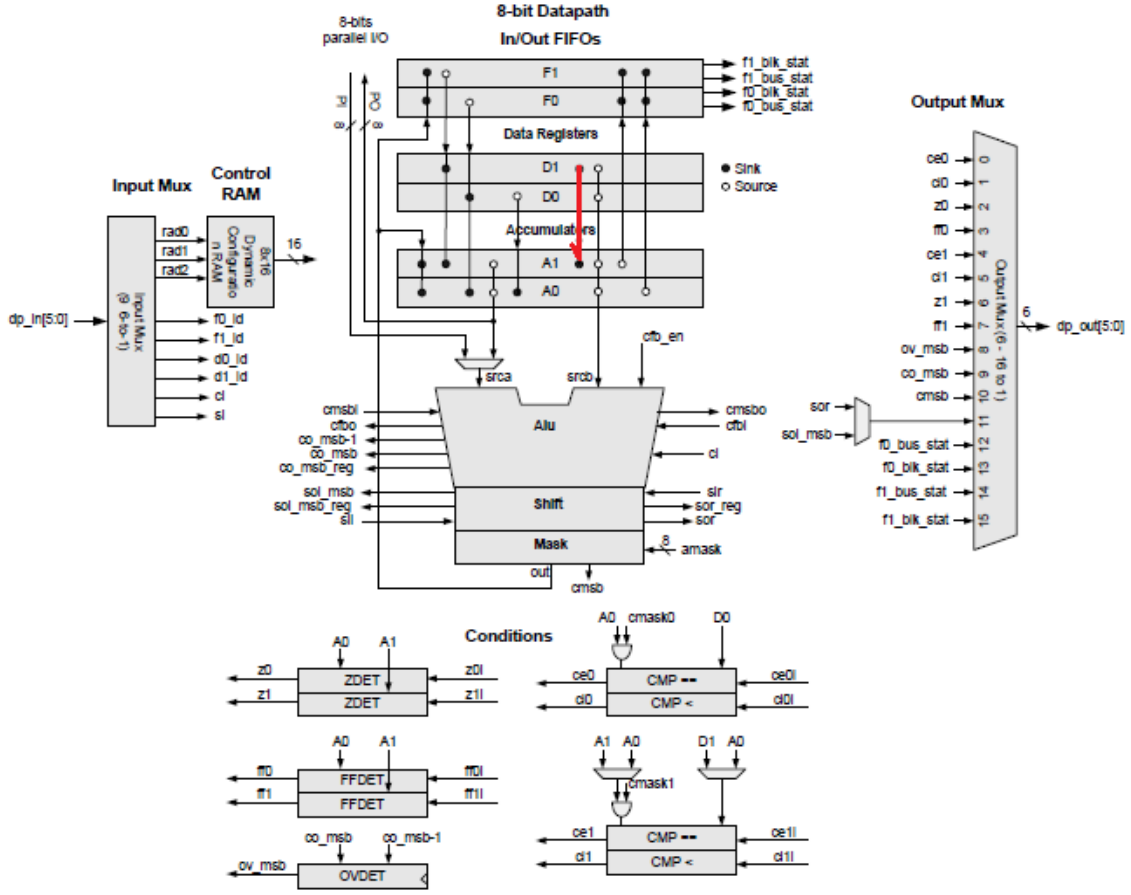
Fonte: Autor

A instrução 0x02 (Figura 4-18), que ocupa o registrador 2 (Reg2), tem a função de atribuir o valor presente no registrador de dados 1 (D1) ao acumulador (A1). Durante a configuração do *datapath*, o registrador de dados D1 é inicializado com o valor zero, e assim permanece durante todo o funcionamento do componente. Portanto, o valor do acumulador 1 (A1) será limpo ao ser executada a instrução 0x02.

Figura 4-21 Instrução 0x02 do *datapath* (A1 <- D1)

CFGRAM

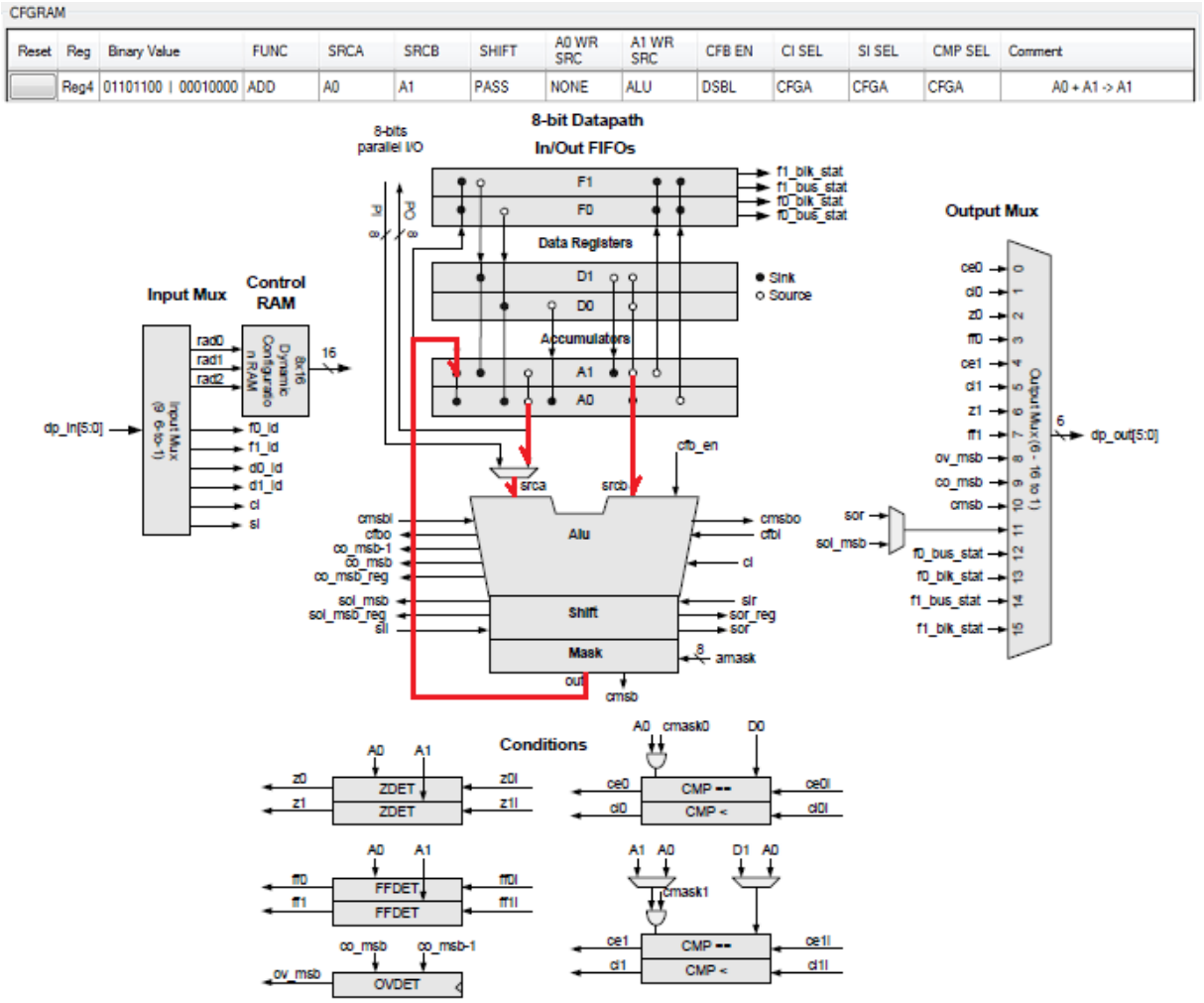
Reset	Reg	Binary Value	FUNC	SRCA	SRCB	SHIFT	A0 WR SRC	A1 WR SRC	CFB EN	CI SEL	SI SEL	CMP SEL	Comment
	Reg2	00001000 | 00100000	PASS	A0	A0	PASS	NONE	D1	DSBL	CFGA	CFGA	CFGA	D1 -> A1



Fonte: (CYPRESS, 2013) Modificado pelo autor.

A instrução 0x02 é utilizada em conjunto com a instrução 0x04 para efetuar o cálculo do *checksum* a cada transação. A instrução 0x04, que ocupa o registrador 4 (Reg4), tem a função de somar o conteúdo dos dois acumuladores A0 e A1, e armazenar o resultado no próprio acumulador 1 (A1).

Figura 4-22 Instrução 0x04 do datapath, somador (A1 <- A0 + A1)



Fonte: (CYPRESS, 2013) Modificado pelo autor.

A lógica do cálculo do checksum é simples. O resultado da soma de todos os bytes de uma transação deve ser sempre zero. Para atingir este objetivo, sempre que um byte inteiro é trafegado no barramento, e consequentemente lido pelo monitor I2C, ele é acumulado no acumulador A1 (instrução 0x04). Sempre que um início de frame é detectado, o acumulador A1 é limpo (instrução 0x02). Enquanto o valor do acumulador A1 for diferente de zero, a saída "*checksum_error*", do componente monitor I2C, estará acionada. Ao finalizar qualquer transação, os dispositivos I2C envolvidos na comunicação devem verificar o estado do sinal "*checksum_error*" para garantir a integridade dos dados trafegados no barramento.

Um efeito colateral dessa abordagem é a necessidade de um tamanho de *payload* pré-definido nas operações de leitura. Isso acontece porque o mestre pode encerrar a leitura a qualquer momento, e assim, o dispositivo escravo não consegue "adivinhar" o momento de inserir o último byte, que deve ser o byte de *checksum*. Apesar do trabalho não implementar

uma solução para esta limitação, ela pode ser resolvida fazendo com que o mestre, antes de fazer uma leitura, escreva, além do endereço de memória a ser lido, o quantidade de bytes que serão lidos. Dessa forma, o dispositivo escravo possuirá a informação de posição de byte de *checksum*. Nas operações de escrita este problema não existe, uma vez que o mestre inicia e termina a operação ele sabe o momento de inserir o último byte, de *checksum*.

Para realizar a contagem dos oito bits que compõem um byte trafegado no barramento, é necessário um contador. Conforme já foi mostrado anteriormente, a implementação de um contador utilizando a lógica de *PLDs* é muito "cara" no *SoC* utilizado neste trabalho. Seguindo a implementação dos componentes I2C mestre/escravo presentes na biblioteca de componentes do PSoC5, a opção encontrada foi a utilização de um contador regressivo de até 7 bits também disponível na biblioteca de componentes. O contador regressivo de 7 bits (Count7) pega emprestado alguns recursos de outras partes do *UDB* que não suas *PLDs*. Mais especificamente, ele utiliza o módulo de *status*/controle do *UDB* para realizar a tarefa. A implementação do contador regressivo de 7 *bits* através do módulo de *status*/controle foi discutida no capítulo 4.2.1, e exemplificado na Figura 4-14.

O *clock* principal do componente Monitor I2C (entrada *clock*), bem como o dos componentes mestre e escravo I2C fornecidos pela Cypress é de 1.6MHz. Esse valor foi calculado de modo a se obter uma taxa de *oversampling* de 16x relativo à frequência do barramento I2C que é 100kbps. O valor de 16x *oversampling* recomendado é baseado na implementação da máquina de estados dos componentes I2C da Cypress (CYPRESS, 2010), na qual o monitor I2C está baseado.

Além da entrada principal de *clock*, o componente Monitor I2C também possui uma entrada de *clock* secundário, de frequência significativamente inferior a do *clock* principal. Esta foi a abordagem escolhida para evitar a implementação de um divisor de frequência com a lógica das *PLDs*. A entrada de *clock* secundário é chamada de "*tmout_clk*", é baseado nela que o mecanismo de timeout do barramento I2C foi implementado. Este mecanismo detecta uma falha permanente em qualquer uma das linhas SCL ou SDA. Foram utilizados dois contadores regressivos de até 7 *bits* para fazer a contagem, um contador para cada uma das linhas do barramento I2C. O valor do tempo limite, que define que uma falha é permanente, não é exclusivamente dependente da velocidade que o barramento opera, mas também depende do modo de operação das aplicações. Em um barramento que opera na taxa 100 kpbs, como no experimento desse trabalho, temos uma taxa de 1bit/10us. Isso significa que durante uma transação normal, a linha SCL não deve permanecer em estado baixo (nível lógico zero) por mais de 10us. Porém, no protocolo I2C não existe uma dependência de tempo, e caso o

dispositivo escravo ou mestre estiverem ocupados com outras atividades, existem mecanismos para pausar a comunicação por tempo indeterminado. No caso do dispositivo mestre é muito simples, basta parar de acionar a linha SCL do barramento que tanto os dispositivos escravos conectados ao barramento, quanto os outros dispositivos mestres irão permanecer em espera. Caso seja um dispositivo escravo que esteja ocupado fazendo outras tarefas, este pode segurar a linha de SCL em nível baixo (estado lógico zero) até finalizá-las e soltar (alta impedância) a linha assim que estiver pronto para comunicar. Esse tipo de operação do dispositivo escravo é conhecida pelo nome de *clock stretching* (esticar o relógio em tradução livre).

Para nosso ambiente de testes foi utilizado no *clock* secundário (*tmout_clk*) uma frequência de 42,67kHz e um valor de timeout (SBS, 2000) de 127 *clocks*.

$$\text{Timeout} = (1 / 42{,}67\text{kHz}) * 127 = 3\text{ms} \qquad (3)$$

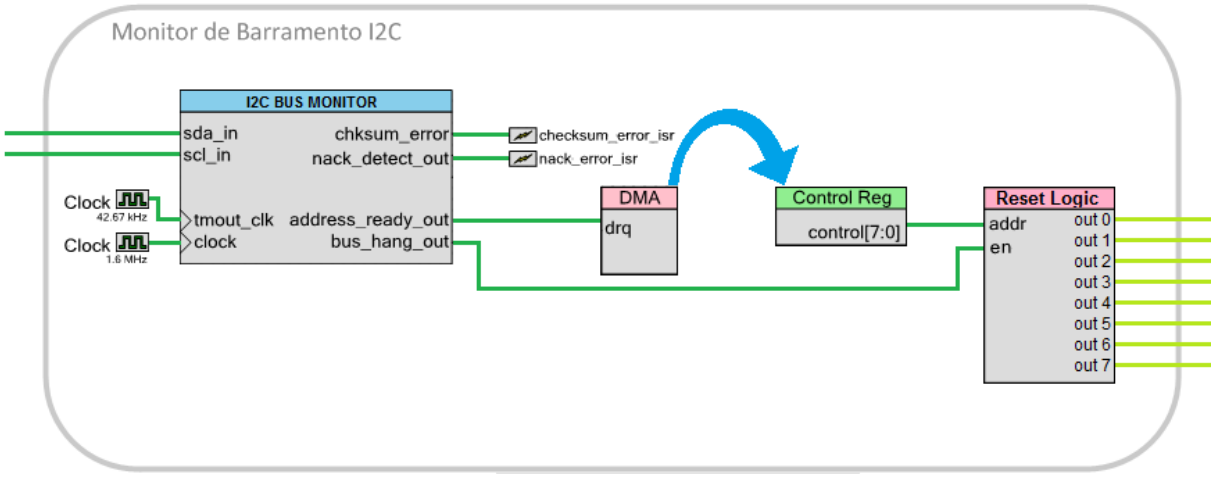
Chegamos a esse valor empiricamente através de diversos testes em nosso ambiente de desenvolvimento. É importante que essa constante não seja menor que o tempo de uma pausa no meio de uma transação. Quanto ao limite superior, afeta unicamente o tempo de detecção da falha. A interface SMBus (SBS, 2000), por exemplo, que garante uma série de funcionalidades ao I2C padrão, também utiliza um mecanismo de timeout, podendo demorar de 25 a 30 milissegundos para detecção de uma falha semelhante.

Ao detectar uma falha permanente no barramento I2C o monitor I2C, em teoria, deve saber qual o dispositivo escravo estava sendo endereçado durante a falha. Em um ambiente com um único mestre, com a exceção de alguns casos específicos, o monitor I2C sabe qual o par (dispositivo mestre e dispositivo escravo) estava se comunicando imediatamente antes da falha acontecer.

O monitor I2C guarda sempre o último endereço I2C escravo que é selecionado no barramento, baseado no último endereço armazenado o monitor pode então tomar ações de recuperação do barramento baseado em ações sobre os dispositivos que contém uma falha em potencial. Mais uma vez, por limitações de hardware e também pela topologia do mesmo, o seguinte mecanismo de armazenamento de endereço é proposto; Como já foi explicado anteriormente, o registrador de deslocamento (*shift register* - Figura 4-19) localizado no *datapath* receberá todos os dados transmitidos no barramento serialmente. Porém, a topologia do PSoC5 não permite que um registrador do *datapath* seja acessado diretamente pela lógica das *PLDs*. A lógica do monitor I2C, no entanto, necessita do último endereço acessado pelo barramento para tomar ações necessárias. A solução para essa questão é a utilização de um

sinal de saída do componente monitor I2C que notifica o restante do design quando um byte de endereço está pronto para ser lido do *datapath*. Este sinal então dispara uma operação de *DMA* que copia o conteúdo do registrador A0 do *datapath*, contendo o último endereço I2C acessado no barramento, para um registrador de controle. Com a utilização de uma lógica adicional, quando o sinal "*bus_hang*" for acionado, sinalizando um travamento no barramento, o conteúdo do registrador de controle que contém o endereço do último dispositivo escravo acessado é utilizado tanto para recuperação do barramento, através de lógica customizada para a seleção da saída de reset referente ao endereço armazenado (*Reset Logic*-Figura 4-23), quanto para a aplicação reportar o erro apropriadamente. A Figura 4-23 mostra todos os componentes apresentados e implementados no PSoC5 que compõem o Monitor de Barramento I2C.

Figura 4-23 Componentes do Monitor de Barramento I2C no PSoC5



Fonte: Autor

4.2.3 Comparativo de Recursos Utilizados pelo Monitor de Barramento I2C

Como já foi discutido no início do capítulo, a topologia de hardware programável do PSoC5 é diferente de uma FPGA ou de uma CPLD somente. Sabendo disso, a Tabela 4-1 mostra um comparativo entre o consumo de recursos de hardware do Monitor de Barramento I2C e outros dispositivos implementados em UDBs pela fabricante Cypress com funcionalidades na mesma área. Todos os dispositivos apresentados na Tabela 4-1 são referentes a implementação no PSoC5.

Tabela 4-1 Comparativo de Recursos Utilizados

	Monitor de Barramento I2C	Controlador I2C Escravo	Controlador I2C Mestre

Macro células	11.98%	13.02%	17.19%
pterms únicos	7.81%	15.36%	25.52%
Total de *pterms*	34	64	100
Células de *datapath*	4.17%	4.17%	8.33%
Módulo *status*/controle	12.50%	8.33%	4.17%

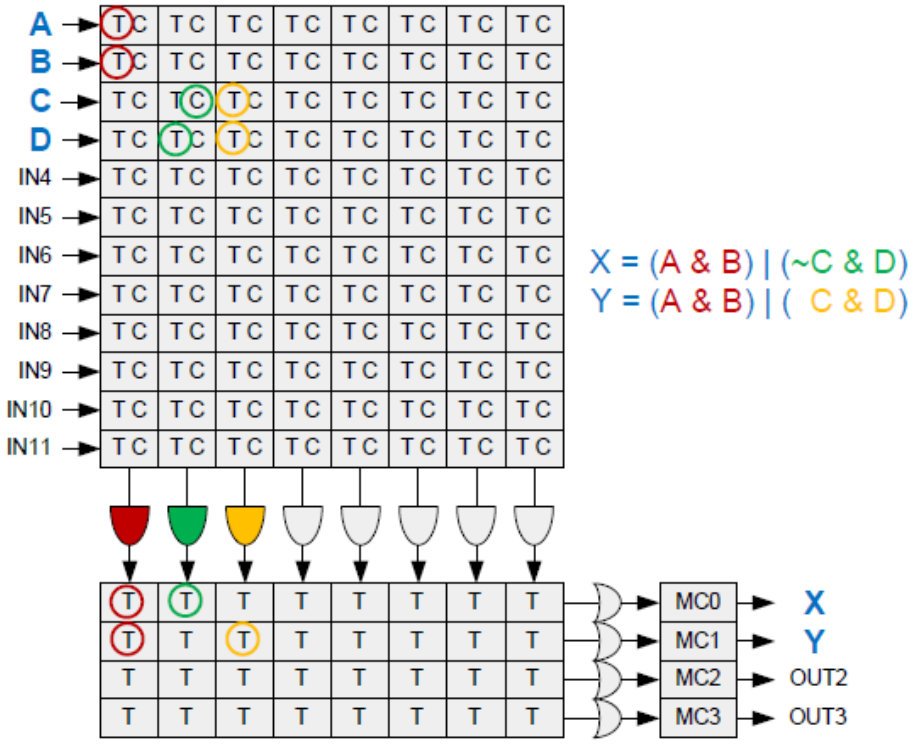
Fonte: Autor

Na primeira linha da tabela comparativa temos o número de macro células utilizadas em cada implementação. Este número tem impacto direto na quantidade de CPLDs utilizadas, lembrando que cada PLD possui apenas 4 macro células de saída, isso faz com que o PSoC5 possua um total de:

$$24 \text{ UDBs} = 48 \text{ PLDs} = 192 \text{ macro células} \quad (4)$$

A segunda linha se refere aos termos únicos (*unique pterms*) utilizados nas PLDs. Os termos únicos são termos que não são utilizados ou compartilhados com outras lógicas. A Figura 4-24 apresenta um exemplo de mapeamento de lógica em uma PLD, no exemplo os termos “~C & D” e “C & D” são considerados únicos. Na terceira linha da tabela aparece o comparativo do total de termos utilizados, isto é, a soma de termos únicos mais os termos compartilhados, como a termo “A & B” da Figura 4-24. A quarta linha da tabela compara a quantidade de células de *datapaths* utilizada em cada componente. O PSoC5 possui 24 UDBs com 1 *datapath* em cada UDB. Logo o valor percentual de 4,17 representa a utilização de um *datapath* somente. O mesmo vale para a quinta linha que compara a utilização de módulos *status*/controle, o Monitor de Barramento I2C faz uso de 3 módulos em razão da implementação dos contadores regressivos, as soluções I2C Escravo e I2C Mestre fazem uso de 2 e 1 módulos respectivamente.

Figura 4-24 Mapeamento da lógica na PLD



Fonte: (CYPRESS, 2016)

4.3 Comparativo Entre Trabalhos Relacionados e o Monitor de Barramento I2C

A Tabela 4-2 mostra a comparação do Monitor de Barramento I2C com as soluções relacionadas apresentadas no capítulo 3. Pode ser observado que a solução proposta consegue reunir a maioria dos pontos positivos contidos nas outras soluções, além de ser autônomo e transparente. O único ponto negativo do Monitor de Barramento I2C está na impossibilidade de desconectar do barramento um dispositivo apresentando mau funcionamento, funcionalidade presente na solução proposta na patente US2004/6728908.

Tabela 4-2 Análise Comparativa Entre Trabalhos Relacionados e o Monitor Proposto

Interface SMBus	US2004/6728908: Bus Protocol Controller with Fault Tolerance	US2007/0240019: Systems and Methods for Correcting Errors in I2C Bus Communications	Monitor de Barramento I2C Proposto
- Necessita que os dispositivos presentes no barramento respeitem a interface SMBus para usufruir	- Custo de hardware maior que o dobro de um dispositivo I2C padrão, utiliza além de lógica própria,	- Não permite que um dispositivo seja desconectado do barramento I2C;	- Não permite que um dispositivo seja desconectado do barramento I2C;

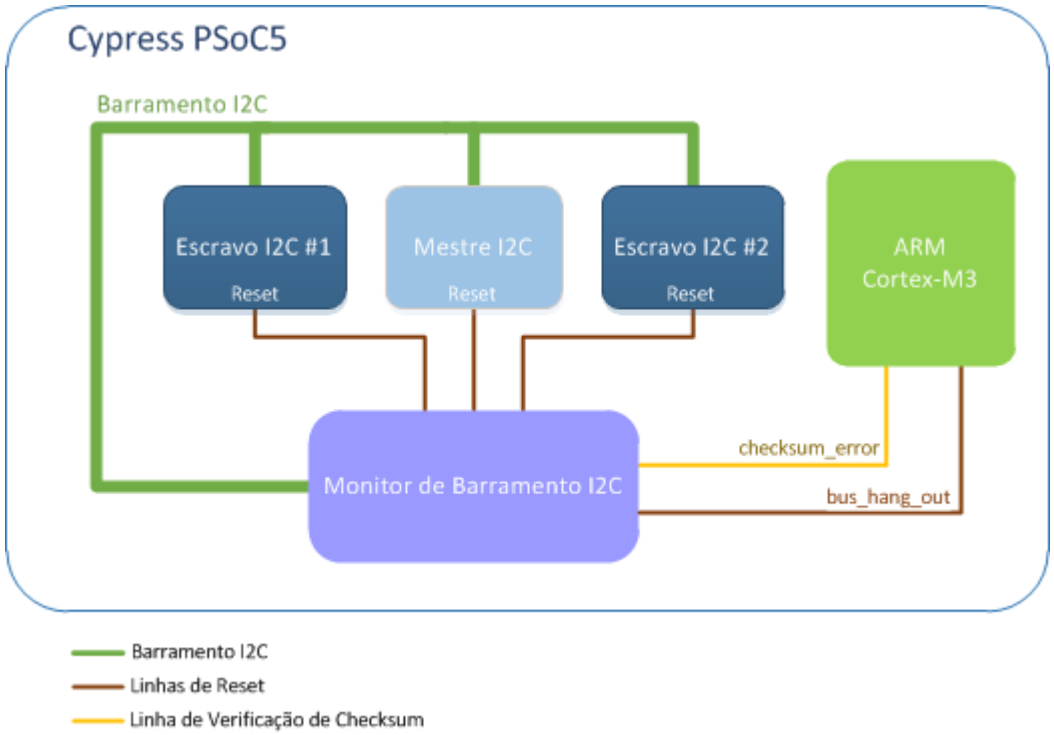
de suas vantagens; - Aceita atuar com dispositivos não compatíveis com SMBus, porém coloca o barramento todo em risco e sem suporte a verificação de integridade de dados nos dispositivos não compatíveis; - Aumento de *overhead* nas comunicações por implementar um nível de protocolo acima da camada de Enlace. - Existe uma grande quantidade de dispositivos I2C que não implementam a interface SMBus. + Excelente controle de integridade de dados;	dois *cores* I2C padrão; - Permite somente um dispositivo I2C mestre; - Apesar de ser possível a utilização de dispositivos escravos padrão, nenhuma segurança é garantida a estes que podem impedir (travar) o funcionamento de todos os outros dispositivos presentes no barramento; - Aumento de *overhead* nas comunicações e na quantidade de transações; - Má escalabilidade, todos os dispositivos necessitam implementar a solução; + Habilidade de bloquear um dispositivo apresentando mau funcionamento de interferir no barramento;	- Não descreve como o BMC e o Monitor interagem fisicamente; - O BMC precisa saber quais os dispositivos que estavam se comunicando para tomar uma ação (lógica de decisão de qual dispositivo deve ser reinicializado não está no monitor, mas na unidade de controle (BMC)); - Não é uma solução transparente por necessitar de software/firmware dedicado na unidade de controle; + Excelente escalabilidade, um monitor por barramento;	+ Pode ser inserido em um design já em operação/validado por ser transparente ao sistema; + Seu custo em relação à utilização de área é quase a metade de um *core* I2C escravo, o que o torna extremamente barato; + Verificação de integridade de dados (opcional), sem perder a maior funcionalidade do monitor, evitar travamentos; + Excelente escalabilidade, um monitor por barramento; + Autônomo, não necessita de uma unidade de controle;

Fonte: Autor

5 TESTES E RESULTADOS

Para testar o Monitor de Barramento I2C proposto neste trabalho, foi desenvolvida uma aplicação no SoC PSoC5 capaz de realizar comunicações I2C, simular falhas de hardware e coletar as informações fornecidas tanto pelos componentes nativos da solução quanto do Monitor. Consolidando os resultados obtidos é fácil observar a eficácia do Monitor de Barramento I2C em cenários de travamentos.

Figura 5-1 Aplicação teste do Monitor de Barramento I2C



Fonte: Autor

Como pode ser observado na Figura 5-1, a aplicação teste consiste em dois controladores I2C escravos, um controlador I2C mestre, o Monitor de Barramento I2C e a CPU ARM Cortex-M3. Todos os componentes do sistema são internos ao SoC PSoC5. Ainda na aplicação teste é importante salientar que o controlador I2C mestre é um componente nativo (*fixed function block*) do PSoC5, diferentemente dos controladores I2C escravos utilizados na aplicação, que são implementados pela própria fabricante Cypress utilizando blocos UDBs. Todos os componentes I2C, incluindo o Monitor de Barramento I2C, estão interligados entre si através de um barramento I2C (na cor verde). O Monitor de Barramento I2C tem acesso à entrada de *reset* de todos os dispositivos I2C ativos (na cor marrom), além

ainda de estar conectado à CPU, que roda a aplicação teste, através do sinal de indicação de erro de *checksum* (conexão na cor amarela).

5.1 Mecanismo de Injeção de Falhas

Para injetar falhas em um ambiente emulado, este trabalho propõe a modificação do controlador I2C escravo, implementado em UDBs, pelo fabricante do SoC PSoC5, Cypress (CYPRESS, 2010). As modificações tem a preocupação de não interferir no funcionamento original do componente e a aplicação teste se encarrega de garantir o bom funcionamento de todos os componentes pertencentes ao ambiente de teste antes de iniciar a injeção de falhas. Para reduzir a possibilidade de interferir no funcionamento do componente modificado para receber injeção de falhas, os testes foram executados alterando somente um registrador por bateria de testes, deixando os demais registradores do componente I2C escravo intactos.

A Figura 5-2 exemplifica como o código HDL do componente I2C escravo foi modificado para permitir injeção de falhas. No exemplo, o registrador *start_sample_reg* está sendo exercitado, isso significa que em todos os pontos onde há uma atribuição para o registrador *start_sample_reg* esta lógica foi adicionada. O registrador *error_injection* é na verdade um módulo de controle operando no modo direto, isto é, o valor contido neste registrador pode e é controlado pela CPU ARM Cortex-M3 rodando a aplicação teste. A *flag* de hardware *clear_injection* tem a função de desligar, ou mascarar, a injeção de falha no ciclo de *clock* subsequente à atribuição. Desta forma, caso a *flag* esteja em uso, a simulação de falha é acionada, atribuída ao registrador em teste e somente no ciclo seguinte a atribuição a injeção é desligada.

Foram realizados dois tipos de teste neste trabalho, cada um com um modelo de falha de diferente duração. O primeiro teste não faz uso da *flag clear_injection* já apresentada. Desta forma o teste injeta e remove a falha por um período pré-determinado de tempo através da aplicação. No segundo tipo de teste, a *flag clear_injection* é utilizada e a falha injetada é desligada no ciclo seguinte à primeira atribuição ao registrador sendo exercitado. Assim temos dois casos de bit-flips ou falhas transientes que ocasionam erros transientes. Os erros que podem ser observados são erros de dados e travamento no barramento, sendo este último crítico, pois impede a comunicação entre quaisquer dispositivos conectados ao mesmo barramento de forma permanente.

É importante salientar que as falhas são injetadas somente nos registradores internos às PLDs. Os registradores do *datapath*, que também tem papel importante na lógica dos componentes, não foram exercitados.

O controle de injeção de falhas parte sempre da CPU, através da aplicação teste que será apresentada a seguir.

Figura 5-2 Lógica de Injeção de Falhas

```
// Código Original
always @(posedge op_clk)
begin
   start_sample_reg <= start_sample0_reg;
end

// Código Modificado
always @(posedge op_clk)
begin
`ifdef START_SAMPLE_REG_TEST
   // Error Injection
   start_sample_reg <= start_sample0_reg                    ^
                       error_injection[START_SAMPLE_REG]    ^
                       clear_injection;

   if( error_injection[START_SAMPLE_REG] )
   begin
      clear_injection <= 1'b1;
   end
   else
   begin
      clear_injection <= 1'b0;
   end
`else
   start_sample_reg <= start_sample0_reg;
`endif
end
```

Fonte: Autor

5.2 Aplicação Teste

A aplicação teste, que roda na CPU ARM Cortex-M3 (ver Figura 5-1), é responsável por exercitar as comunicações I2C, controlar a injeção de falhas e coletar e consolidar as informações provenientes dos componentes que compõem o ambiente de teste. A aplicação avalia a qualidade do barramento baseado nas seguintes métricas:

a) Verificação de status do controlador I2C mestre:

Ao final de cada transação, a aplicação teste verifica o registrador de *status* do controlador I2C mestre. Caso o controlador apresente algum erro, a iteração é considerada defeituosa. Os tipos de *status* disponíveis no controlador I2C mestre, através do conjunto de APIs da fabricante Cypress são:

- Leitura Completa;
- Escrita Completa;
- Transferência em execução;
- Transferência interrompida;
- Mestre *NACKed* antes do fim da transferência;
- Escravo não enviou ACK;
- Mestre perdeu a arbitragem I2C durante a transferência;
- Erro durante a transferência.

b) Verificação do checksum por software:

Nesse passo, o *checksum* da transferência é recalculado no lado do recebimento. Caso esteja sendo exercitada uma operação de escrita, o *checksum* é calculado a partir dos dados do *buffer* de recebimento do dispositivo escravo. Caso seja uma operação de leitura, o cálculo de *checksum* é feito a partir dos dados do buffer de leitura do dispositivo mestre.

c) Endereço I2C capturado pelo Monitor:

Sempre que uma transação é finalizada, o registrador de controle do Monitor de Barramento I2C é lido para garantir que o último dispositivo endereçado no barramento foi corretamente amostrado pelo monitor.

d) Número de erros de checksum informados pelo Monitor:

A saída *checksum_error* pertencente ao Monitor de Barramento I2C é conectada a uma interrupção externa da CPU ARM Cortex-M3. Sempre que uma borda de subida ocorrer no sinal *checksum_error* a interrupção é gerada na CPU que incrementa um contador erros de *checksum* e o reporta ao final de cada iteração (ver Figura 4-23).

e) Número de travamentos informados pelo Monitor:

Segue a mesma lógica do item anterior, a saída *bus_hang_out* está conectada a outra interrupção externa da CPU. Um contador contendo o número de travamentos detectado pelo monitor é reportado ao final de cada transação I2C.

f) Tempo de detecção de falha:

Como já foi explicada anteriormente, a injeção de falha é controlada pela aplicação, e uma vez que a aplicação injeta uma falha, uma interrupção de tempo que ocorre a cada 10us é habilitada. Esta interrupção incrementa um contador, que funciona como uma base de tempo de 10us. Sempre que uma falha for detectada pelo Monitor de Barramento I2C, pelo menos uma das saídas de notificação de falhas é acionada: *bus_hang_out* ou *checksum_error*. Cada uma dessas saídas gera uma interrupção individual, que além de incrementar um contador próprio de número de falhas, capturam o valor da base de tempo com período de 10us. Desta forma a aplicação é capaz de reportar o tempo de detecção de falhas desde a injeção com uma precisão de ±10us.

g) Tempo de recuperação da falha;

   Este valor só é obtido quando acontece previamente uma detecção de travamento de barramento. Basicamente o tempo de recuperação da falha é coletado a partir do momento que o sinal *bus_hang_out* é liberado (borda de descida), o que coincide com a liberação das linhas de reset dos dispositivos envolvidos na última transação. A borda de descida do sinal *bus_hang_out* gera uma interrupção que coleta o valor do temporizador disparado no momento da injeção da falha.

h) Número de resets no dispositivo escravo:

   Segue o mesmo mecanismo dos outros contadores, cada linha de reset possui uma interrupção atribuída a si que incrementa um contador individual que será reportado ao final de cada transação.

i) Tempo da iteração;

   Mede o tempo total de cada iteração através de temporizadores.

j) Tempo total da bateria de testes:

   Mede o tempo total da bateria de testes através de temporizadores.

Uma iteração só é considerada bem sucedida caso nenhum dos itens acima reporte erro. Da mesma forma, uma bateria de testes só é considerada bem sucedida caso todas as iterações que compõem a bateria não tenham apresentado nenhuma falha. Cada registrador contido nas PLDs dos componentes testados foi exercitado em uma bateria individual. Os dados reportados pela aplicação durante os testes são enviados para um console através de uma interface USB-SERIAL.

5.2.1 Fluxo de Teste

Cada registrador testado passou pelos dois tipos de testes mencionados, cada teste é composto por duas baterias, uma de escrita e uma de leitura. O primeiro tipo de teste injeta a falha e a remove por tempo. Nesse caso a aplicação é responsável não só por escrever no registrador que causará a falha, mas também por limpá-lo subsequentemente. A limpeza da falha é feita através de uma interrupção de tempo, executada 10us após o acionamento da falha. Em um barramento I2C com velocidade de 100kbps, como o utilizado no experimento, a velocidade de transmissão de dados é de aproximadamente 1 *bit* a cada 10us, portanto o valor de 10us seria equivalente à frequência de linha de *clock* do barramento (SCL).

No segundo tipo de teste, a aplicação é responsável somente pela injeção da falha, o código HDL modificado no componente em teste é encarregado de limpar a falha no ciclo subsequente à atribuição.

Todos os testes seguem o mesmo fluxo, conforme mostra a Figura 5-3. Cada bateria de testes é composta por 8 iterações, 4 iterações entre o dispositivo I2C mestre e o dispositivo I2C escravo #1 e mais 4 iterações entre o dispositivo I2C mestre e o dispositivo I2C escravo #2. As comunicações são sempre alternadas, iniciando pelo dispositivo I2C escravo #1. As duas primeiras iterações (iteração #0 e iteração #1) são executadas apenas para garantir que todos os componentes presentes no sistema estão funcionando corretamente. Na iteração #2 o mecanismo de injeção de falhas é acionado. A injeção de falhas foi testada somente no dispositivo I2C escravo #1 e no próprio Monitor de Barramento I2C, o dispositivo I2C escravo #2 não foi exercitado por se tratar de um hardware semelhante ao dispositivo I2C escravo #1. As demais iterações, de #3 a #7, servem para garantir que o sistema foi capaz de se recuperar em caso de falha, e não apresenta comportamento intermitente (Tabela 5-1).

Tabela 5-1 Ação do Teste por Iteração

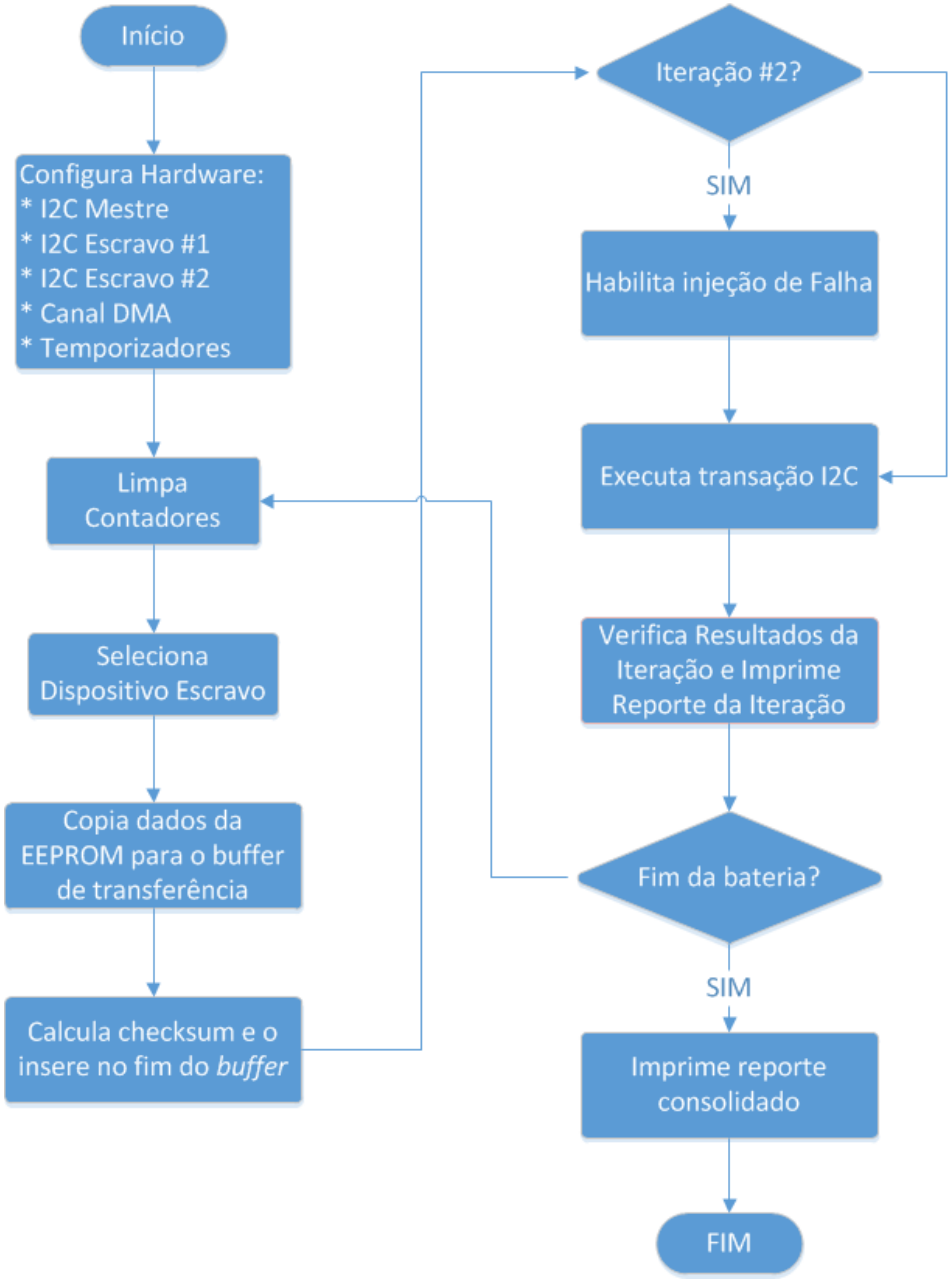
Iteração	Ação Teste
Iteração #0	Comunicação Mestre – Escravo #1
Iteração #1	Comunicação Mestre – Escravo #2
Iteração #2	Comunicação Mestre – Escravo #1 **Injeção Falha no Escravo #1 ou Monitor I2C**
Iteração #3	Comunicação Mestre – Escravo #2
Iteração #4	Comunicação Mestre – Escravo #1
Iteração #5	Comunicação Mestre – Escravo #2
Iteração #6	Comunicação Mestre – Escravo #1

Iteração #7	Comunicação Mestre – Escravo #2

Fonte: Autor

A quantidade de dados transferidos em cada transação I2C é fixa, além do *byte* de endereço, também é transferido um *byte* de dado e, por último, o *byte* de *checksum*. Os testes utilizam um tamanho fixo de dados nas transferências, pois em uma operação de leitura o dispositivo mestre pode parar a comunicação a qualquer momento, sendo impossível o dispositivo escravo saber em qual momento da transação inserir o *byte* de *checksum*. Embora este trabalho não implementa uma solução para esta limitação, ela pode ser contornada adotando um comportamento semelhante ao utilizado pela interface SMBus, onde o dispositivo mestre informa previamente ao dispositivo escravo a quantidade de dados que será lida na operação seguinte. Este problema não acontece em operações de escrita uma vez que o dispositivo mestre, que controla o barramento, é o responsável por incluir o *byte* de *checksum* na última posição da transação.

Figura 5-3 Fluxo de operação da aplicação teste



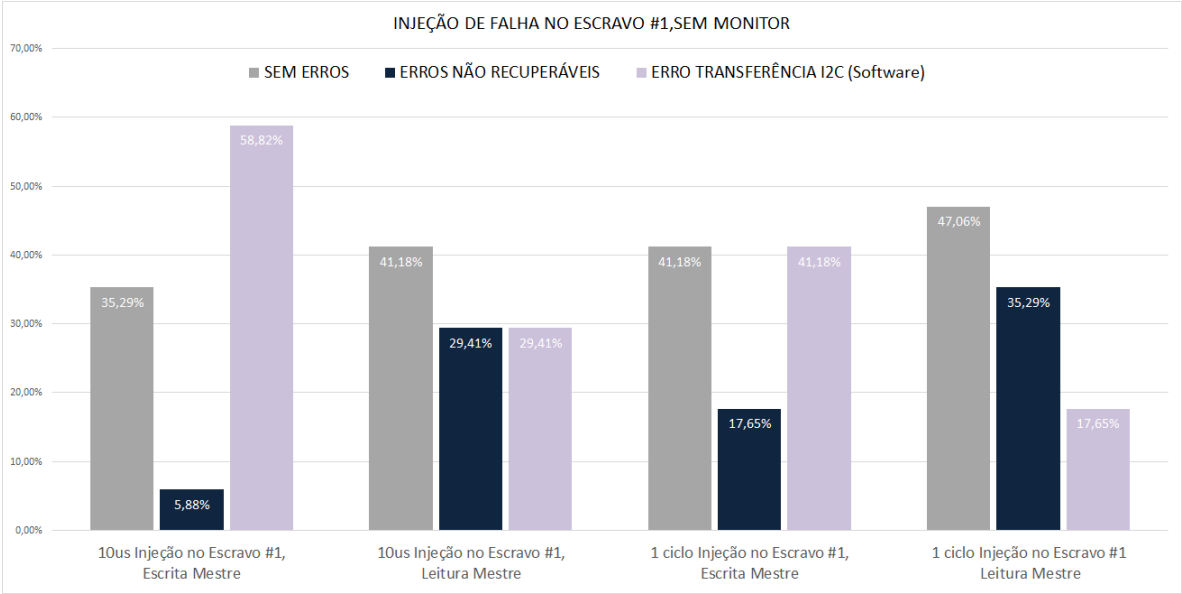
Fonte: Autor

5.3 Resultados Obtidos

Primeiramente deve-se observar o funcionamento do sistema sem a presença do Monitor de Barramento I2C. Nesse cenário, 17 registradores pertencentes ao controlador I2C escravo #1 foram exercitados um a um de acordo com a metodologia já descrita anteriormente. A Figura 5-4 mostra o gráfico contendo os resultados deste primeiro experimento nas categorias de injeção de falha por um intervalo de 10us e na categoria de

injeção de falha em uma atribuição. Em ambos os modos foram diagnosticados erros não recuperáveis, onde o barramento permaneceu travado. A coluna “erro transferência I2C (Software)” se refere a uma bateria de testes onde pelo menos uma transação cujo *status*, verificado pela aplicação através de um registrador no controlador I2C Mestre, reportou algum erro. Essa categoria de erro foi coletada somente em testes sem a presença do monitor.

Figura 5-4 Gráfico com os resultados da emulação de injeção de falhas no Controlador I2C Escravo #1 sem a presença do Monitor de barramento I2C no sistema.

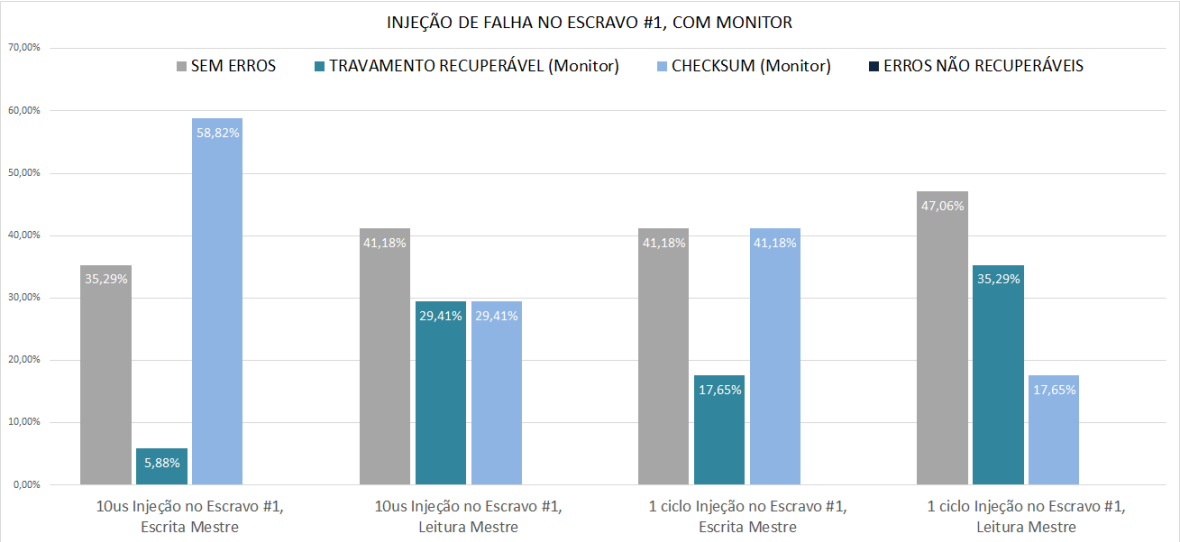


Fonte: Autor

Na sequência, o mesmo teste foi repetido, porém com a presença do Monitor de Barramento I2C no sistema. Como pode ser visto na

Figura 5-5, os erros não recuperáveis do experimento anterior foram substituídos por travamentos recuperáveis. As falhas não persistentes foram em sua totalidade diagnosticadas pelo Monitor de Barramento I2C através da sinalização de falha de *checksum*.

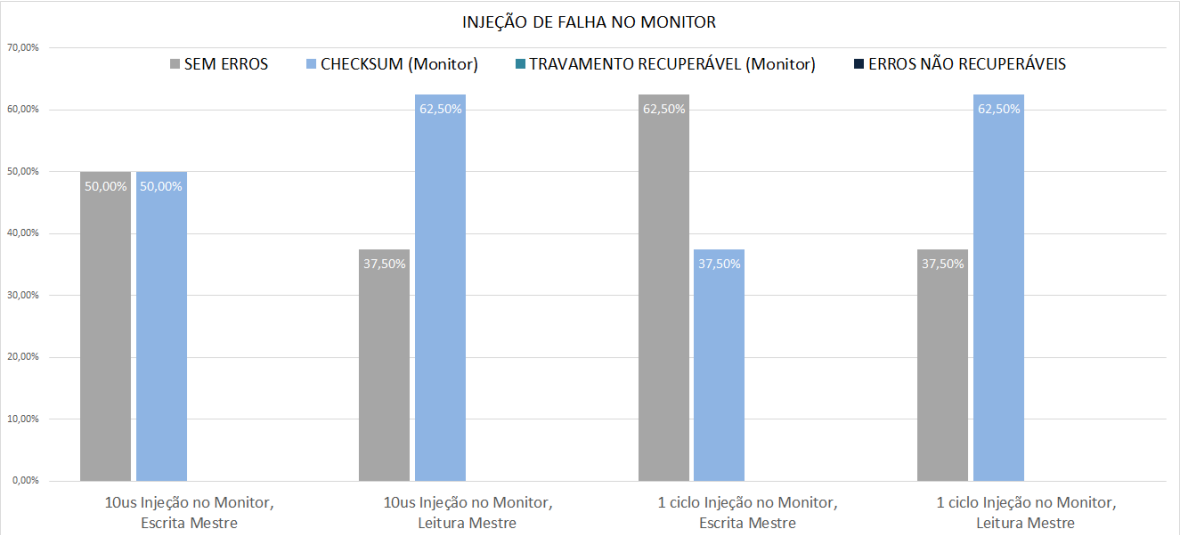
Figura 5-5 Gráfico com os resultados da simulação de injeção de falhas no Controlador I2C Escravo #1 com a presença do Monitor de barramento I2C no sistema



Fonte: Autor

Após a verificação do funcionamento do Monitor em um ambiente suscetível a travamentos, repetimos o mesmo conjunto de testes, mas agora injetando falhas em registradores do Monitor de Barramento I2C, e não mais em um controlador I2C escravo. Cada um dos 8 registradores utilizados no código HDL do Monitor de Barramento I2C foi exercitado, a Figura 5-6 mostra os resultados. Apesar do Monitor de Barramento I2C gerar falsas indicações de erro de *checksum*, nenhum travamento no barramento foi causado por se interferir no funcionamento interno do Monitor, isso acontece pelo fato do monitor não possuir *drivers* de saída em sua conexão com o barramento I2C. Este cenário em uma aplicação que faz uso do Monitor de Barramento I2C implicaria na repetição da transação faltosa, o que pode ser considerado um problema contornável. As tabelas contendo os resultados completos de todos os testes podem ser verificadas no APÊNDICE A - RESULTADO COMPLETO DOS TESTES.

Figura 5-6 Gráfico com os resultados da simulação de injeção de falhas no Monitor de Barramento I2C



Fonte: Autor

CONCLUSÕES

Este trabalho consiste em uma proposta de um Monitor de Barramento I2C cuja finalidade é aumentar a robustez do protocolo através do controle de integridade de dados e prevenção contra travamentos permanentes. Inicialmente foi traçado um comparativo entre os principais protocolos seriais, pertencentes ao mesmo segmento do protocolo I2C, utilizados na indústria. Foi identificada a aplicação alvo para cada protocolo, bem como áreas onde mais de um tipo de protocolo é aplicável.

Embora o trabalho realizado nessa dissertação seja suficiente para provar a eficácia do Monitor de Barramento I2C, ainda existe muito a ser investigado. Em relação à metodologia de injeção de falhas, estas precisam ser estendidas aos registradores internos dos *datapaths* e módulos de controle/*status.* Além disso, casos onde um travamento ocorre durante a transmissão do *byte* de endereçamento do dispositivo I2C escravo não foram estudados nesse trabalho. Finalmente, está em planejamento um teste prático de irradiação do ambiente proposto neste trabalho, assim será possível traçar um comparativo entre os resultados teóricos e práticos deste experimento. É importante ressaltar que o monitor proposto também pode ser utilizado em aplicações industriais e não somente em aplicações espaciais, pois existem diversas outras causas que podem acarretar travamentos em barramentos I2C, como implementações problemáticas, ruído nas linhas e inserção de módulos de hardware a quente.

Apesar do objeto de estudo deste trabalho permitir a emulação de injeção de falhas utilizando somente o kit de desenvolvimento, as características do hardware digital programável do PSoC5 impuseram dificuldades na elaboração do mecanismo de injeção de falhas. Além disso, a limitada quantidade de PLDs não permitiu a instrumentação do controlador I2C mestre provido pelo fabricante.

Outra questão deixada em aberto neste trabalho é o possível travamento de um controlador I2C que está funcionando em modo mestre. Por o controlador mestre normalmente estar vinculado a uma unidade de controle, este pode ser reinicializado por software na maioria das soluções, inclusive fazendo uso do sinal *bus_hang_out*, que pode ser conectado a todos os controladores I2C mestres pertencentes ao barramento. Outra possibilidade é adicionar uma lógica para reinicializar todos os controladores I2C mestres caso a tentativa inicial de destravar o barramento, através do último controlador escravo que acessou o barramento, não funcione.

No futuro, pretendemos substituir o algoritmo de checksum, utilizado na verificação de integridade de dados, pelo algoritmo CRC. Desta forma dispositivos compatíveis com a

interface SMBus terão suas transações validadas pelo monitor. Além disso, temos o interesse de incorporar ao monitor um analisador de protocolo I2C, capaz de registrar todas as transações mal sucedidas em conjunto com o *timestamp* da transação. Esta funcionalidade permitirá a apuração de problemas em dispositivos I2C remotamente, garantindo a manutenibilidade da solução.

REFERÊNCIAS

ATMEL CORPORATION. SPI Serial Flash Memory. 2009.

BELL, C. G. et al. **Computer engineering: A DEC view of hardware systems design**. [S.l.] Digital Press, 1978.

BERGVELD, H. J.; KRUIJT, W. S.; NOTTEN, P. H. L. **Battery Management Systems**. Dordrecht: Springer Netherlands, 2002.

BOSCH. Bosch Controller Area Network (CAN) Version 2.0. **Network**, v. 1939, 1998.

BRADY, P. et al. **Systems and methods for correcting errors in I2C bus communications**, 2007. Disponível em: <http://www.google.com.br/patents/US20070240019>

CHAI, Y. J. et al. Improvement of 12 C bus and RS-232 serial port under complex electromagnetic environment. In: INTERNATIONAL CONFERENCE ON COMPUTER SCIENCE AND SOFTWARE ENGINEERING, 2008. **Proceedings...** [S.l. : s.n.], 2008. p.178-181

CORCORAN, P. Two Wires and 30 Years : A Tribute and Introductory Tutorial to the I2C Two-Wire Bus. **Consumer Electronics Magazine, IEEE**, v. 2, n. July, p. 30–36, 2013.

CORRIGAN, S. Introduction to the controller area network (CAN). **Application Report, Texas Instruments**, n. August 2002, p. 1–15, 2002.

CYPRESS, S. **I2C Master/Multi-Master/Slave 3.30**. Disponível em: <http://www.cypress.com/file/135151/download>.

CYPRESS, S. **PSoC ® 5 Development Kit Guide Quick Start**. Disponível em: <http://www.cypress.com/file/45271/download>.

CYPRESS, S. **PSoC ® 5LP Development Kit Guide**. Disponível em: <http://www.cypress.com/file/45276/download>.

CYPRESS, S. **PSoC ® 5LP: CY8C56LP Family**. Disponível em: <http://www.cypress.com/file/45936/download>.

CYPRESS, S. **PSoC ® 5LP Architecture TRM**. Disponível em: <http://www.cypress.com/file/123561/download>.

CYPRESS, S. **Datapath Configuration Tool Cheat Sheet**. Disponível em: <http://www.cypress.com/file/42106/download>.

CYPRESS, S. **PSoC ® Creator ™ User Guide**. Disponível em: <http://www.cypress.com/file/137441/download>.

CYPRESS, S. PSoC® 3, PSoC 4, and PSoC 5LP – Implementing Programmable Logic Designs with Verilog. n. 1, p. 1–29, 2016.

ECE353: Introduction to Microprocessor Systems. Disponível em: <https://ece353.engr.wisc.edu/>.

FARBIZ, F,; ALI, M. Y.; SANKARALINGAM, R. ESD protection of open-drain 12C using fragile devices in embedded systems. In: SYMPOSIUM ELECTRICAL OVERSTRESS/ELECTROSTATIC DISCHARGE, 37, 2015. **Proceedings...** [S.l.] : IEEE, 2015. Disponivel em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7314755>

FTDI. FT232R USB UART IC. **Technology**, p. 1–40, 2008.

FUKUHARA, R. et al. **I2C Bus Protocol Controller With Fault Tolerance**, 2004.

HIMPE, V. **Mastering the I2C Bus**. [S.l.] Elektor International Media, 2011.

I2C-BUS.ORG. **I2C Bus**. Disponível em: <http://www.i2c-bus.org/>.

INSTRUMENTS, T. 3-V TO 5 . 5-V MULTICHANNEL RS-232 LINE DRIVER / RECEIVER WITH ± 15-kV ESD PROTECTION MAX3223E. **Production**, n. January 2006, p. 1–11, 2009.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO/IEC 7498-1:1994 Information Technology - Open Systems Interconnection - Basic Reference Model: The Basic ModelInternational Standard ISOIEC 74981**, 1996. Disponível em: <http://standards.iso.org/ittf/PubliclyAvailableStandards/s025022_ISO_IEC_7498-3_1997(E).zip>

KALINSKY, D.; KALINSKY, R. **Introduction to I2C**. Disponível em: <http://www.embedded.com/electronics-blogs/beginner-s-corner/4023816/Introduction-to-I2C>.

LEENS, F. An introduction to I2C and SPI protocols. **IEEE Instrumentation and Measurement Magazine**, v. 12, n. 1, p. 8–13, 2009.

MILIOS, J. CRC-8 firmware implementations for SMBus. **SBS IF DevCon Japan**, p. 25, 1999.

MOTOROLA. SPI Block User Guide. **Power**, p. 1–36, 2002.

MUKHERJEE, S. **Architecture Design for Soft Errors**. Burlington, MA: Elsevier Science, 2011.

NXP SEMICONDUCTORS. UM10204 I^2C-bus specification and user manual. n. April, p. 64, 2014.

OSBORNE, A. **An Introduction to Microcomputers: Basic concepts**. Berkeley, CA: Osborne/McGraw-Hill, 1980.

SBS, I. F. System Management Bus Specification. n. December, p. 1–59, 2000.

SILICON LABS. UART Flow Control. p. 1–11, 2013.

SLATER, R. **Portraits in Silicon**. [S.l.] MIT Press, 1989.

TEXAS INSTRUMENTS. LM75x Digital Temperature Sensor and Thermal Watchdog with Two-Wire Interface Datasheet. v. SNIS153D, n. July, 2009.

VALCOSS ELECTRONICS. D-SUB, 318" Right Angle PCB Mount. p. 1, 2012.

APÊNDICE A - RESULTADO COMPLETO DOS TESTES

Resultado dos testes de escrita nos controladores I2C #1 e #2, com injeção de falhas por um período de 10us no Monitor de Barramento I2C.

Registrador	Tempo de Detecção desde a injeção (µs)	Tempo de Recuperação desde a injeção (µs)	Duração Iteração Faltosa (µs)	Duração total da bateria de testes, oito iterações (µs)	Travamento do Barramento	Recuperável	Falhas Diagnosticadas
S_STATE_REG_0	240	*	4810	37650	NÃO	*	CHECKSUM
S_STATE_REG_1	*	*	4830	37730	NÃO	*	NENHUM ERRO
S_STATE_REG_2	240	*	4830	37620	NÃO	*	CHECKSUM
START_SAMPLE_REG_0	240	*	4820	38530	NÃO	*	CHECKSUM
START_SAMPLE_REG	240	*	4840	37670	NÃO	*	CHECKSUM
S_BYTE_COMPLETE_REG	*	*	4830	37630	NÃO	*	NENHUM ERRO
ADDRESS_BYTE_COMMING	*	*	4820	37700	NÃO	*	NENHUM ERRO
CHECKSUM_ERROR_REG	*	*	4830	37680	NÃO	*	NENHUM ERRO

Fonte: Autor

Resultado dos testes de leitura dos controladores I2C #1 e #2, com injeção de falhas por um período de 10us no Monitor de Barramento I2C.

Registrador	Tempo de Detecção desde a injeção (µs)	Tempo de Recuperação desde a injeção (µs)	Duração Iteração Faltosa (µs)	Duração total da bateria de testes, oito iterações (µs)	Travamento do Barramento	Recuperável	Falhas Diagnosticadas
S_STATE_REG_0	150	*	4080	31390	NÃO	*	CHECKSUM
S_STATE_REG_1	150	*	4070	31290	NÃO	*	CHECKSUM
S_STATE_REG_2	150	*	4070	31320	NÃO	*	CHECKSUM
START_SAMPLE_REG_0	160	*	4070	31250	NÃO	*	CHECKSUM
START_SAMPLE_REG	160	*	4080	31230	NÃO	*	CHECKSUM
S_BYTE_COMPLETE_REG	*	*	4070	31360	NÃO	*	NENHUM ERRO
ADDRESS_BYTE_COMMING	*	*	4080	31400	NÃO	*	NENHUM ERRO
CHECKSUM_ERROR_REG	*	*	4080	31340	NÃO	*	NENHUM ERRO

Fonte: Autor

Resultado dos testes de escrita nos controladores I2C #1 e #2, com injeção de falhas por do tipo *bit-flip* na atribuição no Monitor de Barramento I2C.

Registrador	Tempo de Detecção desde a injeção (µs)	Tempo de Recuperação desde a injeção (µs)	Duração Iteração Faltosa (µs)	Duração total da bateria de testes, oito iterações (µs)	Travamento do Barramento	Recuperável	Falhas Diagnosticadas
S_STATE_REG_0	230	*	4650	36080	NÃO	*	CHECKSUM
S_STATE_REG_1	*	*	4660	36100	NÃO	*	NENHUM ERRO
S_STATE_REG_2	*	*	4640	36100	NÃO	*	NENHUM ERRO
START_SAMPLE_REG_0	230	*	4650	35970	NÃO	*	CHECKSUM
START_SAMPLE_REG	230	*	4670	35770	NÃO	*	CHECKSUM
S_BYTE_COMPLETE_REG	*	*	4060	31230	NÃO	*	NENHUM ERRO
ADDRESS_BYTE_COMMING	*	*	4640	36030	NÃO	*	NENHUM ERRO
CHECKSUM_ERROR_REG	*	*	4630	35990	NÃO	*	NENHUM ERRO

Fonte: Autor

Resultado dos testes de leitura dos controladores I2C #1 e #2, com injeção de falhas por do tipo *bit-flip* na atribuição no Monitor de Barramento I2C.

Registrador	Tempo de Detecção desde a injeção (µs)	Tempo de Recuperação desde a injeção (µs)	Duração Iteração Faltosa (µs)	Duração total da bateria de testes, oito iterações (µs)	Travamento do Barramento	Recuperável	Falhas Diagnosticadas
S_STATE_REG_0	140	*	4050	31150	NÃO	*	CHECKSUM
S_STATE_REG_1	140	*	4050	31150	NÃO	*	CHECKSUM
S_STATE_REG_2	140	*	4050	31090	NÃO	*	CHECKSUM
START_SAMPLE_REG_0	140	*	4040	33150	NÃO	*	CHECKSUM
START_SAMPLE_REG	140	*	4050	31170	NÃO	*	CHECKSUM
S_BYTE_COMPLETE_REG	*	*	4050	31260	NÃO	*	NENHUM ERRO
ADDRESS_BYTE_COMMING	*	*	4080	31240	NÃO	*	NENHUM ERRO
CHECKSUM_ERROR_REG	*	*	4060	31290	NÃO	*	NENHUM ERRO

Fonte: Autor

Resultado dos testes de escrita no controlador I2C #1, com injeção de falhas por um período de 10 us.

Registrador	Tempo de Detecção desde a injeção (µs)	Tempo de Recuperação desde a injeção (µs)	Duração Iteração Faltosa (µs)	Duração total da bateria de testes, oito iterações (µs)	Travamento do Barramento	Recuperável	Falhas Diagnosticadas
S_BYTE_COMPLETE_REG	*	*	4850	37740	NÃO	*	NENHUM ERRO
S_ADDRESS_REG	*	*	4840	37750	NÃO	*	NENHUM ERRO
START_SAMPLE0_REG	130	*	7080	39890	NÃO	SIM	CHECKSUM
START_SAMPLE_REG	130	*	7070	39870	NÃO	SIM	CHECKSUM
SCL_IN_REG	130	*	7080	39840	NÃO	SIM	CHECKSUM
SCL_IN_LAST_REG	*	*	4790	39830	NÃO	*	NENHUM ERRO
SCL_IN_LAST2_REG	130	*	7080	39830	NÃO	SIM	CHECKSUM
S_SDA_OUT_REG	*	*	4840	37720	NÃO	*	NENHUM ERRO
S_SCL_OUT_REG	*	*	4830	37700	NÃO	*	NENHUM ERRO
SDA_IN_REG	130	*	7070	39840	NÃO	SIM	CHECKSUM
SDA_IN_LAST_REG	130	*	7060	39830	NÃO	SIM	CHECKSUM
SDA_IN_LAST2_REG	130	*	7080	39830	NÃO	SIM	CHECKSUM
SLAVE_RST_REG	130	*	7070	39860	NÃO	SIM	CHECKSUM
S_STATE_REG_0	130	*	7070	39860	NÃO	SIM	CHECKSUM
S_LRB_REG	*	*	4870	37750	NÃO	*	NENHUM ERRO
S_STATE_REG_1	2980	2980	10070	42850	SIM	SIM	TRAVAMENTO
S_STATE_REG_2	130	*	7060	39830	NÃO	SIM	CHECKSUM

Fonte: Autor

Resultado dos testes de leitura no controlador I2C #1, com injeção de falhas por um período de 10 us.

Registrador	Tempo de Detecção desde a injeção (µs)	Tempo de Recuperação desde a injeção (µs)	Duração Iteração Faltosa (µs)	Duração total da bateria de testes, oito iterações (µs)	Travamento do Barramento	Recuperável	Falhas Diagnosticadas
S_BYTE_COMPLETE_REG	2980	2980	6740	33980	SIM	SIM	TRAVAMENTO
S_ADDRESS_REG	3100	3100	6770	33950	SIM	SIM	TRAVAMENTO
START_SAMPLE0_REG	3050	3050	6710	33910	SIM	SIM	TRAVAMENTO
START_SAMPLE_REG	3050	3050	6710	33910	SIM	SIM	TRAVAMENTO
SCL_IN_REG	140	*	4060	34140	NÃO	SIM	CHECKSUM
SCL_IN_LAST_REG	*	*	4090	31370	NÃO	*	NENHUM ERRO
SCL_IN_LAST2_REG	*	*	4060	31380	NÃO	*	NENHUM ERRO
S_SDA_OUT_REG	*	*	4070	31420	NÃO	*	NENHUM ERRO
S_SCL_OUT_REG	*	*	4070	31430	NÃO	*	NENHUM ERRO
SDA_IN_REG	*	*	4070	31400	NÃO	*	NENHUM ERRO
SDA_IN_LAST_REG	140	*	4080	31320	NÃO	SIM	CHECKSUM
SDA_IN_LAST2_REG	*	*	4090	31350	NÃO	*	NENHUM ERRO
SLAVE_RST_REG	150	*	4070	31290	NÃO	SIM	CHECKSUM
S_STATE_REG_0	150	*	4070	31310	NÃO	SIM	CHECKSUM
S_LRB_REG	*	*	4080	31440	NÃO	*	NENHUM ERRO
S_STATE_REG_1	2970	2970	6720	33970	SIM	SIM	TRAVAMENTO
S_STATE_REG_2	150	*	4070	31290	NÃO	SIM	CHECKSUM

Fonte: Autor

Resultado dos testes de escrita no controlador I2C #1, com injeção de falhas do tipo *bit-flip* na atribuição.

Registrador	Tempo de Detecção desde a injeção (µs)	Tempo de Recuperação desde a injeção (µs)	Duração Iteração Faltosa (µs)	Duração total da bateria de testes, oito iterações (µs)	Travamento do Barramento	Recuperável	Falhas Diagnosticadas
S_BYTE_COMPLETE_REG	3060	3060	9060	40470	SIM	SIM	TRAVAMENTO
S_ADDRESS_REG	120	*	7050	38440	NÃO	SIM	CHECKSUM
START_SAMPLE0_REG	120	*	7040	38440	NÃO	SIM	CHECKSUM
START_SAMPLE_REG	120	*	7060	38510	NÃO	SIM	CHECKSUM
SCL_IN_REG	120	*	7040	38420	NÃO	SIM	CHECKSUM
SCL_IN_LAST_REG	*	*	*	36150	NÃO	*	NENHUM ERRO
SCL_IN_LAST2_REG	*	*	*	36110	NÃO	*	NENHUM ERRO
S_SDA_OUT_REG	230	*	4650	36000	NÃO	SIM	CHECKSUM
S_SCL_OUT_REG	*	*	4640	36130	NÃO	*	NENHUM ERRO
SDA_IN_REG	*	*	4630	36070	NÃO	*	NENHUM ERRO
SDA_IN_LAST_REG	*	*	4640	36150	NÃO	*	NENHUM ERRO
SDA_IN_LAST2_REG	*	*	4660	36110	NÃO	*	NENHUM ERRO
SLAVE_RST_REG	120	*	7050	38470	NÃO	SIM	CHECKSUM
S_STATE_REG_0	120	*	7050	38460	NÃO	SIM	CHECKSUM
S_LRB_REG	*	*	4640	36100	NÃO	SIM	NENHUM ERRO
S_STATE_REG_1	2990	2990	9050	40540	SIM	SIM	TRAVAMENTO
S_STATE_REG_2	2950	2950	9080	40490	SIM	SIM	TRAVAMENTO

Fonte: Autor

Resultado dos testes de leitura no controlador I2C #1, com injeção de falhas do tipo *bit-flip* na atribuição.

Registrador	Tempo de Detecção desde a injeção (µs)	Tempo de Recuperação desde a injeção (µs)	Duração Iteração Faltosa (µs)	Duração total da bateria de testes, oito iterações (µs)	Travamento do Barramento	Recuperável	Falhas Diagnosticadas
S_BYTE_COMPLETE_REG	2980	2980	6600	33690	SIM	SIM	TRAVAMENTO
S_ADDRESS_REG	*	*	4070	33710	NÃO	*	NENHUM ERRO
START_SAMPLE0_REG	3070	3070	6570	33680	SIM	SIM	TRAVAMENTO
START_SAMPLE_REG	3070	3070	6560	33680	SIM	SIM	TRAVAMENTO
SCL_IN_REG	150	*	4070	31190	NÃO	SIM	CHECKSUM
SCL_IN_LAST_REG	*	*	4050	31270	NÃO	*	NENHUM ERRO
SCL_IN_LAST2_REG	*	*	4060	31210	NÃO	*	NENHUM ERRO
S_SDA_OUT_REG	*	*	4060	31230	NÃO	*	NENHUM ERRO
S_SCL_OUT_REG	*	*	4060	31220	NÃO	*	NENHUM ERRO
SDA_IN_REG	*	*	4040	31240	NÃO	*	NENHUM ERRO
SDA_IN_LAST_REG	*	*	4060	31210	NÃO	*	NENHUM ERRO
SDA_IN_LAST2_REG	*	*	4050	31220	NÃO	*	NENHUM ERRO
SLAVE_RST_REG	140	*	4040	31200	NÃO	SIM	CHECKSUM
S_STATE_REG_0	140	*	4060	31200	NÃO	SIM	CHECKSUM
S_LRB_REG	3090	3090	6570	33720	SIM	SIM	TRAVAMENTO
S_STATE_REG_1	2970	2980	6570	33700	SIM	SIM	TRAVAMENTO
S_STATE_REG_2	2950	2950	6570	33760	SIM	SIM	TRAVAMENTO

Fonte: Autor