FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”
CENTRO UNIVERSITÁRIO “EURÍPIDES DE MARÍLIA” - UNIVEM
PROGRAMA DE MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

RICARDO LUÍS MARTINS VERONESI

RTRASSOC51 - MÓDULO DE COMUNICAÇÃO I2C RECONFIGURÁVEL rI2C.

MARÍLIA
2005

RICARDO LUÍS MARTINS VERONESI

RTRASSOC51 - MÓDULO DE COMUNICAÇÃO I2C RECONFIGURÁVEL rI2C.

Dissertação apresentada ao Programa de Mestrado do Centro Universitário Eurípides de Marília, mantido pela Fundação de Ensino Eurípides Soares da Rocha, para a obtenção do Título de Mestre em Ciência da Computação. (Área de Concentração: Arquitetura de Sistemas Computacionais).

Orientador:
Prof. Dr. Jorge Luiz e Silva

MARÍLIA
2005

VERONESI, Ricardo Luís Martins

RtrASSoc51 – Módulo de Comunicação I2C Reconfigurável – rI2C / Ricardo Luís Martins Veronesi; orientador: Jorge Luiz e Silva. Marília, SP: 2005.

114 f.

Dissertação (Mestrado em Ciência da Computação) – Centro Universitário Eurípides de Marília – Fundação de Ensino Eurípides Soares da Rocha.

1.Introdução 2.Microcontroladores 3.FPGA 4.Mecanismos de Comunicação 5.VHDL 6. O rI2C na plataforma RtrASSoc51 7.Conclusão

CDD: 005.74

RICARDO LUÍS MARTINS VERONESI

RTRASSOC51 - MÓDULO DE COMUNICAÇÃO I2C RECONFIGURÁVEL rI2C.

Banca examinadora da dissertação apresentada ao Programa de Mestrado da UNIVEM / F.E.E.S.R., para obtenção do Título de Mestre em Ciência da Computação. Área de Concentração: Arquitetura de Sistemas Computacionais.

Resultado: APROVADO

A Comissão Julgadora:

ORIENTADOR: Prof. Dr. Jorge Luiz e Silva _____

Prof. Dr. Ildeberto Aparecido Rodello _____

Prof. Dr. Eduardo Marques _____

Marília – SP, 24 Agosto de 2005.

Dedico este trabalho à minha amada Carla.

AGRADECIMENTOS

À minha amada Carla, pela confiança e motivação em absolutamente todos os momentos.

Aos meus pais Ivanir e Aparecida, à minha sogra Nayr, e meus familiares pela compreensão, incentivo e apoio prestados nos momentos mais importantes.

Principalmente ao meu orientador Prof. Dr. Jorge Luiz e Silva, por toda dedicação, atenção, orientação e apoio, fundamentais na concretização desta dissertação, meus mais sinceros agradecimentos.

Aos professores, Prof. Dr. Ildeberto Aparecido Rodello, Prof. Dr. Edward David Moreno Ordonez, Prof. Dr. Marcos Luiz Mucheroni, e Prof. Dr. Shusaburo Motoyama, dos quais aprendi admirar.

Aos meus novos e grandes amigos do Mestrado, André Gobbi, César “Itararé”, Claudete, Fábio Modesto, Fábio Montanha, Gisele, Leila, Márcio “Filhão”, pela convivência, pelos momentos de estudo, trabalho e diversão compartilhados.

Aos meus amigos e companheiros inseparáveis dessa jornada, Mestre Marcelo “Bariri” Storion e Márcio “Perpétua” Cardim, meus mais sinceros agradecimentos pela vibração, amizade, companheirismo e principalmente apoio nos momentos de dúvidas e dificuldades.

Aos meus coordenadores, diretores e amigos Aderson Bini, Fred Dallalana, Marco Antônio Torres e Vera Casério pela convivência, confiança e principalmente incentivo.

As queridas Roberta, Beth (e família) do PPGCC, meus mais sinceros agradecimentos pela amizade e dedicação.

A todos os meus amigos e aos meus alunos da Facol, Fênix e Unip que, sem dúvida, foram os principais incentivadores.

A Deus pelo Dom da Vida.

"A vitória não está no
fim do caminho.
Ela é o próprio
caminho".

VERONESI, Ricardo Luís Martins. RtrASSoc51 - Módulo de Comunicação I2C Reconfigurável – rI2C, 2005, 114 f. Dissertação (Mestrado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005.

RESUMO

Aplicações embarcadas são sistemas que se caracterizam como elementos essenciais em produtos que estão presentes em quase tudo em nossas vidas: aviões, automóveis, monitoração médica, TV´s digitais, celulares, videogames, impressoras, copiadoras, fax, telefones, etc, muitos desses contendo microprocessadores ou microcontroladores, que estão sendo interligados uns aos outros, processando informações. O RtrASSoc51 é um Sistema em Chip, Adaptável, Superescalar, e Reconfigurável, em desenvolvimento, sendo implementado em FPGA da Xilinx e será utilizado em aplicações embarcadas para reconhecimento de objetos. Esta dissertação descreve a implementação do protocolo de comunicação rI2C (Reconfigurable Inter Integrated Circuits) que consiste do protocolo I2C original acrescido das técnicas de reconfiguração, mais especificamente a reconfiguração das diferentes taxas de comunicação previstas no protocolo original. O rI2C será utilizado na plataforma RtrASSoc51 como um Core de comunicação e foi implementado em VHDL, e resultados de simulação bem como propostas futuras são apresentadas no final desta dissertação.

Palavras-chave: FPGA, I2C, Protocolo de Comunicação, Reconfigurável, Reconhecimento de Objetos, rI2C, RtrASSoc51, Sistema em Chip, VHDL.

VERONESI, Ricardo Luís Martins. RtrASSoc51 - Módulo de Comunicação I2C Reconfigurável – rI2C, 2005, 114 f. Dissertação (Mestrado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005.

ABSTRACT

Embedded applications are systems that are characterized as essential elements in products that are present in almost everything in our lives: airplanes, automobiles, medical monitor, TV´s digital, cellular, videogames, printers, copiers, fax, telephones, etc, many of those containing microprocessors or microcontrollers, that are being interlinked each other, processing information. RtrASSoc51 is a System on Chip, Adaptable, Superescalar, and Reconfigurable, in development, being implemented in FPGA of Xilinx and it will be used in embedded applications for pattern recognition. This dissertation describes the implementation of the protocol of communication rI2C (Reconfigurable Inter Integrated Circuits) that consists of the original protocol I2C added of a reconfigurable part, more specifically the rates of communication. The rI2C will be used in the platform RtrASSoc51 as one CORE of communication and it was implemented in VHDL. Simulation results as well as proposed future are presented in the end of this dissertation.

Keywords: FPGA, I2C, Protocol of Communication, Reconfigurable, Recognition of Objects, rI2C, RtrASSoc51, System on Chip, VHDL.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

A/D	Analógico para Digital
ALU	Unidade Lógica Aritmética
AMBA	Advanced Microcontroller Bus Architecture
API	Application Program Interface
ASIC	Application Specific Integrated Circuit
CAN	Controller Area Network
CI	Circuito Integrado
CLB	Configurable Logig Block
CLP	Controlador Lógico Programável
CPLD	Complex Programmable Logic Device
CRC	Cyclic Redundancy Check
D/A	Digital para Analógico
DISC	Dynamic Instruction Set Computer
DPGA	Dinamically Programmable Gate Array
DSP	Digital Signal Processing
E/S	Entrada e Saída
EDA	Eletronic Design Automation
EEPROM	Electrically Erasable PROM
EPLD	Erasable PLD
EPROM	Erasable PROM
ESB	Embedded System Block
FIFO	First-In First-Out
FIP	Factory Instrumentation Protocol
FIPSoC	Field Programmable SoC
FPGA	Field Programmable Gate Array
FPIC	Field Programmable Interconnect Component
FSM	Finite State Machine
GPP	General Poupouse Processor
HDL	Hardware Description Language
I2C	Inter Integrated Circuit
IOB	Input/Output Block

IOE	Input/Output Element
IRL	Internet Reconfigurable Logic
ISO	International Organization for Standardization,
LUT	Look-Up Table
MCU	Memory Control Unit
MPGA	Mask Programmable Gate Array
OSI	Open Systems Interconnection
PAL	Programmable Array of Logic
PLA	Programmable Logic Array
PLD	Programmable Logic Device
PROFIBUS	Process Fieldbus, um protocolo de barramento industrial
PROM	Programmable Read-Only Memory
PSOC	Programable System-on-Chip
rI2C	Reconfigurable Inter Integrated Circuits
RPU	Reconfigurable Processing Unit
RTL	Register Transfer Level
RTR	Run-Time Reconfiguration
SCL	Clock Signal
SDA	Data Signal
SoC	System-on-Chip
SoPC	System-on-a-Programable Chip
SRAM	Static Random Access Memory
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver/Transmitter
ULA	Unidade Lógica e Aritmética
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VLSI	Very Large Scale Integration

SUMÁRIO

1 - INTRODUÇÃO

Aplicações embarcadas são sistemas que se caracterizam como elementos essenciais em produtos que estão presentes em quase tudo em nossas vidas: aviões, automóveis, monitoração médica, TVs digitais, celulares, videogames, impressoras, copiadoras, fax, telefones, etc, todos contendo microprocessadores ou microcontroladores que estão sendo interligados uns aos outros, processando informações.

Os custos destas tecnologias continuam caindo e, tanto a computação quanto a comunicação tenderão a ser cada vez mais embarcados (embedded) em objetos que estarão se modificando conforme o ambiente (entende-se por isso reconfigurabilidade dinâmica). É o que vem sendo mostrado na computação "*ubiquitous*" (computação onipresente), destacado em um documento gerado em 2001 - "*Embedded, Everywhere - A Research Agenda for Networked Systems of Embedded Computers"* - por alguns órgãos nos Estados Unidos (CNSEC, 2001).

Uma constatação desses dados é o sistema que faz uso de processamento digital de sinais (DSP) que cresceu de 200 milhões de unidades vendidas no ano de 1997, para 1200 milhões em 2001 (DANDALIS, 2001). Uma outra previsão é a de que por volta de 2005, cada pessoa estará carregando ao redor de si cinco microcontroladores em média (BANNATYNE, 1998).

Microcontroladores são dispositivos que têm sido muito utilizados no mercado mundial de automação, controle e principalmente sistemas embarcados, devido suas características de fácil utilização, baixo custo, robustez e com um conjunto de instruções voltadas para essas aplicações. Dentre eles destaca-se os microcontroladores da família 8051, um dos dispositivos mais utilizados no mercado mundial, dadas suas características, custo e disponibilidade de software (ARÓSTEGUE, 2004; ATMEL, 2004; DOLPHIN, 2003).

Com a evolução da microeletrônica, os FPGAs (*Field Programmable Gate Array*) surgiram como um elemento intermediário entre os processadores de propósito geral GPP (*General Purpose Processor*) e os ASICs (*Applications Specific Integrated Circutis*) (HAUCK, 1998; HAUCK, 2000; DEHON, 1996; VILLASENOR, 1997). Originalmente, os FPGAs vinham sendo utilizados como dispositivos "*glue logic*", reduzindo significantemente o número de componentes em um sistema, e diminuindo os riscos de desenvolvimento de projetos, ao mesmo tempo em que proporcionava a flexibilidade para correções e atualizações de forma rápida e mais segura. Agora os FPGAs são utilizados no projeto e controle de inúmeros sistemas com aplicações específicas, significativamente complexas, e dedicadas (PIACENTINO, 1999).

Muitos recursos têm sido aplicados para definir sistemas reconfiguráveis baseados em FPGAs (ARNOLD, 1993; ARNOLD, 1992; ATHANAS, 1992; QUENOT, 1994; GOKHALE, 1991; RATHA, 1995, WIRTHLIN e HUTCHINGS, 1995), tornando a computação reconfigurável uma realidade. Da mesma forma, pesquisadores vêm despendendo muito esforço e tempo em implementar esses sistemas, em especial sistemas reconfiguráveis gerados diretamente a partir de um programa escrito em linguagem C ou assembler.

Ainda com a evolução agressiva da indústria de semicondutores, vêm sendo desenvolvidos chips com alta densidade e maior complexidade, onde engenheiros passam a criar sistemas em chip complexos com alto desempenho (SHALAN e MOONEY, 2002). Arquiteturas de Sistema em Chip Programável (*SOPC*) são frutos desse desenvolvimento, também como alternativa aos ASICs que sofrem de longos ciclos de projetos e pouca flexibilidade, enquanto que o GPP não atendem aos requisitos de desempenho para diversas aplicações (LEE et al, 2002; PHILLIPS e HAUCK, 2002).

A confirmação da atual situação dos SOPCs pode ser comprovada pelo número de plataformas comerciais disponíveis (MASSIMO et al, 2002), dentre as quais pode-se citar o

ambiente SOPC da Altera (ALTERA, 2004) e o sistema MicroBlaze da Xilinx (XILINX, 2004), ferramentas poderosas para desenvolvimento desses sistemas.

O RtrASSoc51 é um sistema em chip programável (SOPC) sendo implementado em FPGAs da Xilinx tendo como base a família de microcontroladores 8051, acrescido de uma estrutura *pipeline* em três níveis a ser utilizado em aplicações embarcadas que necessitem de maior capacidade, melhor desempenho e reconfiguração dinâmica podendo ser interconectado em rede (ARÓSTEGUE, 2004; CARRILLO, 2001; DOLPHIN, 2003; MASSIMO, 2002). A plataforma RtrASSoc51 vem sendo desenvolvida à três anos, inicialmente através da simulação e validação de alguns conceitos fundamentais como medida de desempenho do sistema para aplicações em processamento de imagens, e mais recentemente através da implementação de partes da plataforma e do mecanismo de interconexão dos dispositivos RtrASSoc51 através do protocolo de comunicação I2C (SILVA, 2003; LOPES, 2004; SILVA, 2004; COSTA, 2004; ZANGUETTIN, 2004; FORNARI, 2004).

A aplicação a ser desenvolvida consiste no reconhecimento de objetos através de redes neurais booleanas no modelo *N-tuple* (BONATO, 2004). Uma câmera modelo CMOS será conectada à plataforma RtrASSoc51 que irá capturar imagens a serem comparadas a imagens pré-definidas de alguns objetos. Inicialmente o teste será realizado em uma única plataforma RtrASSoc51 e em seguida em um conjunto de plataformas RtrASSoc51 interconectados em rede em uma aplicação industrial (LOPES, 2004).

Este projeto descreve a implementação do protocolo de comunicação reconfigurável rI2C (*Reconfigurable Inter Integrated Circuits*) (VERONESI, 2005).

O Protocolo I2C foi originalmente proposto pela Philips, e nesta nova versão foi acrescido de técnicas de reconfiguração, mais especificamente reconfiguração das diferentes taxas de comunicação prevista no protocolo original.

O rI2C será utilizado na plataforma RtrASSoc51 como um *CORE* de comunicação e foi implementado em VHDL para FPGAs da Xilinx.

O capítulo 2 apresenta um breve histórico dos Microcontroladores das famílias 8051 e PIC e também é citado o processador DSP, apresentando os aspectos mais relevantes à arquitetura, instruções e vantagens, buscando assim a assimilação das principais características dos mesmos para a caracterização deste projeto.

O capítulo 3 apresenta as definições e características de FPGAs, tratando também de arquiteturas, tecnologia de programação, roteamento, acoplamentos e reconfiguração parcial e dinâmica.

O capítulo 4 apresenta o conceito de mecanismos de comunicação internos e externos entre FPGAs, descrevendo tipos de barramentos e protocolos como AMBA, *Field Bus* , CAN e principalmente o I2C com a caracterização das taxas de transmissão deste último protocolo. Apresenta-se a padronização de interfaces de comunicação EIA/RS, citando alguns padrões estudados como o RS-232, RS-422 e RS-485.

O capítulo 5 apresenta definições, histórico e características da linguagem de descrição de hardware VHDL.

O capítulo 6 apresenta o desenvolvimento e implementação do protocolo de comunicação rI2C na plataforma RtrASSoc51, com suas características de comunicação, diagramas de fluxo e estados, e também simulações utilizando a ferramenta *Xilinx Foundation.* Também apresenta características dos trabalhos relacionados a este projeto que foram desenvolvidos e estão em desenvolvimento por alunos do Mestrado em Ciência da Computação do UNIVEM.

Finalmente no capítulo 7 são apresentadas as conclusões sobre o desenvolvimento do protocolo rI2C e sugestões de trabalhos futuros.

2 - MICROCONTROLADORES

Segundo Souza (2000), o microcontrolador é um “pequeno” componente eletrônico, dotado de uma “inteligência” programável utilizado no controle de processos lógicos.

Um microcontrolador é um microprocessador acrescido de endereçadores de I/O de memória de programa (PROM, EPROM, EEPROM), memória RAM, tudo isto em um único CI (RIBEIRO, 2002).

Através da programação dos microcontroladores pode-se controlar suas saídas, tendo como referência as entradas ou um programa interno. O que diferencia os diversos tipos de microcontroladores, é a quantidade de memória interna (instruções e dados), velocidade de processamento, quantidade de pinos de entrada/saída (I/O), alimentação, periféricos, arquitetura e conjunto de instruções.

Um microcontrolador é um componente que tem, num único *chip*, além de uma CPU, elementos tais como memórias ROM e RAM, temporizadores, contadores, canais de comunicação e conversores analógico-digitais. Esta característica diferencia os sistemas baseados em microcontroladores daqueles baseados em microprocessadores, onde normalmente se utilizam vários componentes para implementar essas funções. Com isso, os microcontroladores permitem a implementação de sistemas mais compactos e baratos do que aqueles baseados em microprocessadores.

Em contrapartida, as CPUs dos microcontroladores são, em geral, menos poderosas do que os microprocessadores. Seu conjunto de instruções costuma se limitar às instruções mais simples encontradas nestes, sua freqüência de *clock* é mais baixa e o espaço de memória endereçável costuma ser bem menor. Nota-se daí que o campo de aplicação dos microcontroladores é diferente daquele dos microprocessadores, e que um sistema que possa ser controlado por um microcontrolador tende a ter menor complexidade e menor custo do que um sistema que exija a capacidade de processamento de um microprocessador.

Exemplos de sistemas onde os microcontroladores encontram aplicação incluem controle de semáforos, balanças eletrônicas, microterminais, telefones públicos, controle de veiculação de comerciais de TV, controle de carregadores de baterias, inversores, controles de acesso, taxímetros, sistemas de aquisição de dados de manufatura e eletrodomésticos em geral.

A programação dos microcontroladores é, em geral, mais simples do que a dos microprocessadores, ao menos no que diz respeito às exigências de conhecimento dos componentes periféricos. Isto acontece porque os periféricos *on-chip* dos microcontroladores são acessados de uma forma padronizada e integrada na própria linguagem de programação, dispensando o conhecimento de detalhes externos.

Não se deve pensar, porém, que isto simplifique a tarefa do programador em todos os níveis: é necessário que ele conheça bem o *hardware* conectado ao microcontrolador para poder produzir programas que funcionem corretamente.

Cabe citar ainda uma vantagem particular dos microcontroladores que possuem memória ROM, que é a possibilidade de armazenar programas internamente, dificultando sensivelmente a cópia ilícita do código.

Alguns fabricantes de microcontroladores são: Motorola, Mitsubish, Siemens, Nec, Hitachi, Philips, Intel, Microchip, Matsushitta, Toshiba, National Semiconductor, Zilog, Texas, Sharp, AMD, Atmel, Dallas, OKI, Matra, SMC, SSI dentre outros.

2.1 – A família de Microcontroladores 8051

O microcontrolador 8051, aqui chamado apenas de 8051, da Intel, é, sem dúvida, um microcontrolador muito popular.

O dispositivo em si é um microcontrolador de 8 *bits* relativamente simples, mas com ampla aplicação. Porém, o mais importante é que não existe somente o CI 8051, mas sim uma

família de microcontroladores baseada no mesmo. Entende-se família como sendo um conjunto de dispositivos que compartilha os mesmos elementos básicos, tendo também um mesmo conjunto básico de instruções.

A Intel iniciou a produção do 8051 em 1981 e, em 1982 foram produzidos 2 milhões de unidades, em 1985 foram 18 milhões e em 1993, 126 milhões (INTEL , 1998).

As principais características encontradas na família do microcontrolador 8051 são mostradas na Tabela 1.

Tabela 1 - Características encontradas na família do microcontrolador 8051

Microcont. 8051	**Características**	**Observações**
Freqüência de clock.	12 MHz.	Com algumas versões que alcançam os 40 MHz.
Memória de dados externa.	Até 64 Kb.	
RAM interna.	128 bytes.	
Memória de programa.	Até 64 Kb.	
Portas de I/O.	4 portas bidirecionais de I/O.	Cada uma com 8 bits individualmente endereçáveis.
Temporizadores /contadores.	2 temporizadores /contadores de 16 bits.	
Comunicação serial.	1 canal.	
Interrupção.	5 fontes de interrupção, com 2 níveis de prioridade selecionáveis por software.	Dois temporizadores, dois pinos externos e o canal de comunicação serial.
Clock.	Oscilador de clock interno.	

Além do 8051 propriamente dito, existem variantes como o 8031 (sem memória ROM interna e com apenas 128 bytes de memória RAM), o 8751 (4 Kb de memória EPROM) e o 8052 (8 Kb de memória ROM, um terceiro temporizador (timer) e 256 bytes de memória

RAM). A menos dessas diferenças, os modelos microcontrolador 8051 citados são idênticos, e será utilizado o termo “8051” de forma genérica, citando as outras versões apenas onde for necessário.

As características citadas são básicas e formam o núcleo da família 8051, que pode ser acrescido de uma ou mais destas características, conforme a Figura 1.

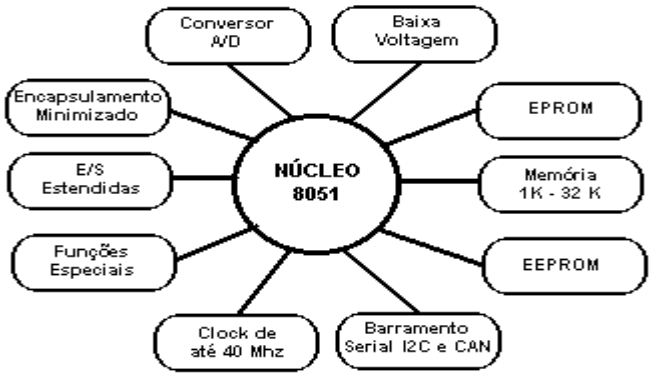

Figura 1 - Características especiais da família 8051 (RIBEIRO, 2002)

Dentre tais características especiais que podem ser acrescidas, apresentadas na Figura1, nota-se a referente à freqüência do *clock*, onde se encontram algumas versões que alcançam 40 MHz, como também conversores analógico/digital e possibilidade de utilização de barramentos (*buses)* seriais tais como o CAN e o I2C, sendo este último um dos principais objetos deste projeto.

Pode-se considerar o 8051 um microcontolador popular, prontamente disponível e com amplo suporte, além de uma gama completa de produtos de suporte disponíveis. É considerado rápido e eficaz, onde a arquitetura se correlaciona de perto com o problema a ser solucionado (sistemas de controle). Tem um baixo custo, devido ao alto nível de integração do sistema em um único componente. Poucos componentes são necessários para se criar um sistema que funcione.

Em relação à gama de produtos, uma única família de microcontroladores cobre as opções que outros fornecedores só conseguem cobrir com um número razoável de famílias

diferentes e incompatíveis. Desse modo, o 8051 proporciona economia real em termos de custo de ferramentas, treinamento e suporte para *software*. Existem aperfeiçoamentos constantes proporcionando assim, melhorias na manufatura que aumentam a velocidade e potência constantemente.

Em relação à compatibilidade, as opções e código binário são os mesmos para todas as variações do 8051, diferente de outras famílias de microcontroladores.

2.2 – O Microcontrolador PIC

O PIC é um circuito integrado produzido pela *Microchip Technology Inc*., que pertence à categoria dos microcontroladores, ou seja, um componente integrado que em um único dispositivo contém todos os circuitos necessários para realizar um completo sistema digital programável. O PIC internamente dispõe de todos os dispositivos típicos de um sistema microprocessado, conforme mostrado na Tabela 2 (SOUZA, 2000).

Tabela 2 - Dispositivos internos do PIC

Dispositivos	**Função**
Uma CPU (Unidade Central de Processamento)	Sua finalidade é interpretar as instruções de programa.
Uma memória PROM (Memória Programável Somente para Leitura)	Na qual irá memorizar de maneira permanente as instruções do programa.
Uma memória RAM (Memória de Acesso Aleatório)	Utilizada para memorizar as variáveis utilizadas pelo programa.
Uma série de LINHAS de I/O (entrada e saída)	Para controlar dispositivos externos ou receber pulsos de sensores, chaves, etc.
Gerador de clock, barramento, contador, etc.	Uma série de dispositivos auxiliares ao funcionamento

Segundo Souza (2000):

> "Os microcontroladores PIC apresentam uma estrutura de máquina interna do tipo *Harvard*, enquanto grande parte dos microcontroladores tradicionais apresenta uma arquitetura tipo *Von-Neumann*. A diferença está na forma como os dados e o programa são processados pelo microcontrolador. Na arquitetura tradicional, tipo *Von-Neumann*, existe apenas um barramento (*bus*) interno, por onde passam as instruções e os dados. Já na arquitetura tipo *Harvard*, existem dois barramentos internos, sendo um de dados e outro de instruções.... Esse tipo de arquitetura permite que enquanto uma instrução é executada outra seja 'buscada' da memória , o que torna o processamento mais rápido."

Os microcontroladores da família PIC são dispositivos RISC (*Reduced Instruction Set Computer* – Computador com conjunto reduzido de instruções) com a já citada arquitetura *HARVARD* (barramentos de memória de programa e de memória de dados diferentes) e fluxo de instruções *PIPELINE*. Seu uso é praticamente ilimitado e seu preço é baixo, levando-se em conta as suas excelentes características. Os PICs são muito versáteis, podem possuir de 6 até 66 pinos de I/O, trabalhar em freqüências de até 40MHz. Conforme Tabela 3, eles podem ter ainda algumas características especiais.

Tabela 3 - Características da capacidade do Microcontrolador PIC

Características	
Conversores - analógico para digital	ADCs – A/C
Conversores - digital para analógico	DACs – D/A
Endereçadores	de memória externa
I/Os	seriais
Memória EEPROM	de dados

A presença de todos estes dispositivos em um espaço extremamente pequeno, dá ao projetista ampla gama de trabalho e enorme vantagem em usar um sistema microprocessado, onde em pouco tempo e com poucos componentes externos pode-se fazer o que seria oneroso fazer com circuitos tradicionais.

O PIC está disponível em uma ampla gama de modelos para melhor adaptar-se as exigências de projetos específicos, diferenciando-se pelo número de linha de I/O e pelo conteúdo do dispositivo. Inicia-se com modelo pequeno identificado pela sigla PIC12Cxx dotado de 8 pinos, até chegar a modelos maiores com sigla PIC17Cxx dotados de 40 pinos.

2.2.1 – Aplicações PIC

O PIC16F84, usado como base desta pesquisa, é perfeitamente adequado para uma variedade de aplicações, tais como na indústria de automóveis, sensores remotos, fechaduras elétricas e dispositivos de segurança. É também um dispositivo ideal para cartões inteligentes, bem como para dispositivos alimentados por baterias, devido ao seu baixo consumo.

A memória EEPROM faz com que se torne mais fácil usar microcontroladores em dispositivos onde o armazenamento permanente de vários parâmetros seja necessário (códigos para transmissores, velocidade de um motor, freqüências de recepção, etc.).

O baixo custo, facilidade de manuseio e flexibilidade fazem com que o PIC16F84 possa ser utilizado em áreas em que os microcontroladores não eram anteriormente empregados (exemplo: funções de temporização, substituição de interfaces em sistemas de grande porte, aplicações de co-processamento, etc.).

A possibilidade deste *chip* ser programável no sistema (usando somente dois pinos para a transferência de dados), dá flexibilidade ao produto, mesmo depois da sua montagem e testes estarem completos.

2.2.2 – Organização da memória no PIC

O PIC16F84 tem dois blocos de memória separados, um para dados e o outro para o programa. A memória EEPROM e os registros de uso genérico (GPR) na memória RAM constituem o bloco para dados e a memória FLASH constitui o bloco de programa.

A memória de programa é implementada usando tecnologia FLASH, o que torna possível programar o microcontrolador muitas vezes antes do mesmo ser instalado num dispositivo e, mesmo depois da sua instalação, pode-se alterar o programa e parâmetros contidos.

O tamanho da memória de programa é de 1024 endereços de palavras de 14 bits. Os endereços zero e quatro estão reservados respectivamente para o *reset* e para a interrupção.

A memória de dados por sua vez, compreende memória EEPROM e memória RAM. A memória EEPROM consiste em 64 posições para palavras de oito bits e cujos conteúdos não se perdem em razão de uma falha na alimentação.

Como a memória EEPROM serve usualmente para guardar parâmetros importantes (por exemplo, de uma temperatura em reguladores de temperatura), existe um procedimento para escrever na EEPROM que tem que ser seguido de modo a evitar uma escrita acidental.

2.3 – DSP (Processador Digital de Sinal)

DSP (*Digital Signal Processor*) é um processador voltado para processamento digital de sinais. Sua arquitetura é composta por dois barramentos de endereços independentes e dois de dados que também são independentes. Enquanto um desses barramentos serve para a leitura de instruções de um programa, o outro serve para a leitura de dados. Com isso, é possível operar simultaneamente uma instrução e um byte de dados, garantindo maior velocidade de processamento (Figura 2).

Nosso mundo está se tornando cada vez mais digital e os Processadores Digitais de Sinal (DSP) estão no centro desta revolução. Estes processadores, capazes de operar em tempo real, representam o segmento que mais cresce no mercado de semicondutores e são capazes de atender a crescente demanda por processamento rápido de informações. Os DSPs são os principais componentes em 70% dos telefones celulares existentes em todo o mundo,

melhoram o desempenho dos sistemas de freios em automóveis e permitem conexões mais rápidas entre redes de computadores (COSTA, 2000).

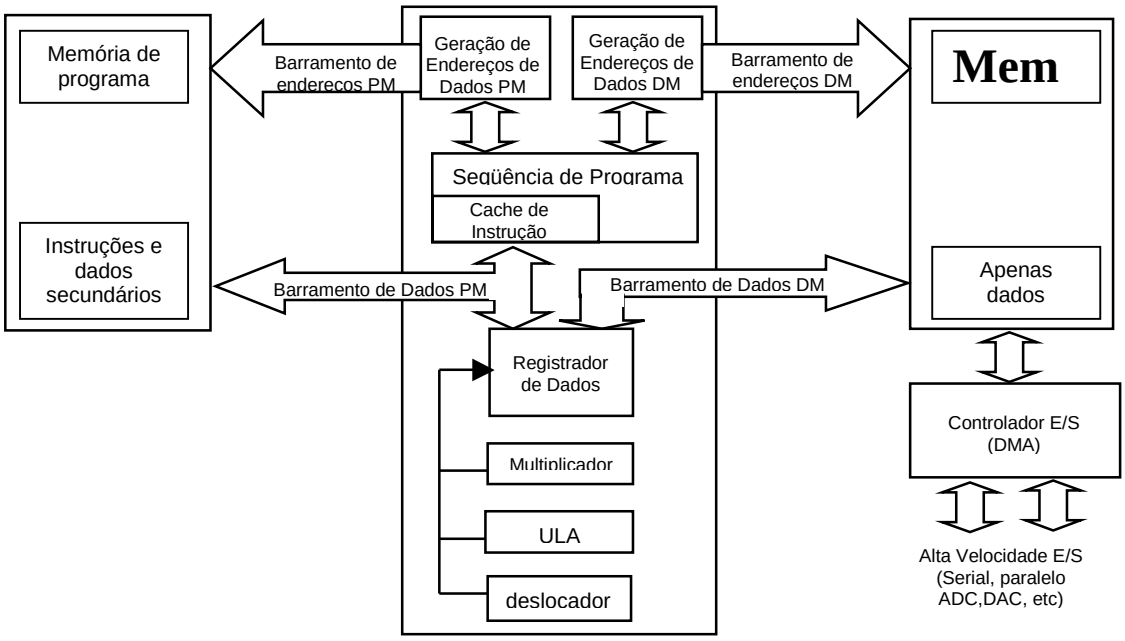

Figura 2 – Arquitetura Básica de um DSP (COSTA, 2000).

Muitas aplicações principalmente nas áreas de industria automobilística, médica, militar e telecomunicações utilizam o processamento digital de sinais para o tratamento de imagens e sinais. Em geral, tais aplicações requerem alto desempenho de processamento e muitas vezes utiliza processamento concorrente, sendo necessário o uso de um processador digital de sinais especifico (COSTA, 2000).

Os DSPs são microprocessadores com características próprias que podem ser programados e operam em tempo real, com velocidades muito superiores aos microprocessadores para aplicações genéricas. A capacidade de processar grandes quantidades de números em pouco tempo é um dos principais benefícios que os DSPs oferecem.

Este processo é, na realidade, um "devorador" de números e um solucionador de equações matemáticas. Compõe-se basicamente de:

- Conversor Analógico para Digital (A/D);

- Processador (Processor, CPU, etc);
- Conversor Digital para Analógico (D/A).

As principais características de um DSP são (COSTA, 2000):

- Uma arquitetura avançada;
- Processamento paralelo;
- Conjunto de instruções dedicado para processamento digital de sinais;
- Capacidade de processamento de milhões de instruções por segundo (MIPS)

Sistemas que utilizam DSP são utilizados, por exemplo, para aplicações que têm uma entrada de sinais analógicos em intervalos fixos de tempo. Os sinais analógicos são amostrados em intervalos de tempo e a sua amplitude é convertida em valores digitais por conversores, conforme a Figura 3. O que é processado, é a representação digital do sinal como uma seqüência de números. Este processamento feito pelo DSP pode ser utilizado para análise ou síntese de sinais (síntese de voz, etc.), modificação de freqüência ou resposta de amplitude de sinais (filtros e controladores).

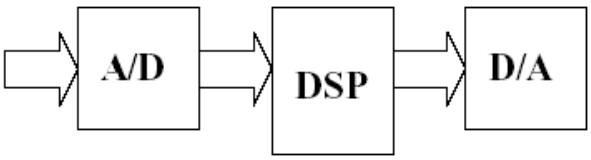

Figura 3 - Conversão de sinais utilizando sistema com DSP

Depois do processamento, os dados são convertidos novamente na forma analógica com o uso de um conversor Digital/Analógico (THOMPSOM, 2001).

3 – FPGA (*FIELD PROGRAMMABLE GATE ARRAY*)

Os circuitos digitais têm sofrido grande evolução nas últimas décadas. As constantes mudanças na tecnologia têm transformado de forma radical todo o processo de projeto de *hardware*.

Os componentes dos circuitos digitais evoluíram de transistores individuais para circuitos integrados VLSI (*Very Large Scale Integration*). A utilização de ferramentas EDA (*Eletronic Design Automation*) tem simplificado e acelerado todo o ciclo de projeto. Atualmente, não é mais necessário desenhar portas lógicas individuais e planejar todas suas interconexões. As linguagens de descrição de *hardware* (HDLs) estão hoje consolidadas no meio acadêmico e industrial como forma padrão na elaboração de projetos. Existem também, ferramentas de síntese lógica automática, disponíveis para mapear circuitos em diversas tecnologias (CHAN e MOURAD, 1994).

Todas essas mudanças na tecnologia exigem uma prototipação cada vez mais rápida, pois o ciclo de vida dos produtos modernos está se tomando cada vez mais curto em relação ao tempo necessário para o projeto e desenvolvimento dos mesmos.

Os CIs digitais podem ser construídos utilizando-se de diversas tecnologias diferentes (Figura 4). A escolha da tecnologia adequada deve ser realizada com base no tipo de projeto que se pretende executar.

Segundo Wolf (2001), as implementações de circuitos podem ser agrupadas em diversas categorias de projeto, tais como, CIs (Circuitos Integrados) customizados ou ASICs (*Application Specific Integrated Circuit* – Circuito Integrado com Aplicação Específica), MPGAs (*Mask Programmable Gate Arrays*), *Standard Cells* e PLDs (*Programmable Logic Device* – Dispositivo Lógico Programável).

Os CIs customizados ou ASICs, são aqueles que necessitam de um processo de fabricação especial, que requer metodologias específicas para cada projeto. Características

desse tipo de implementação são os custos de projeto extremamente altos e o tempo de desenvolvimento longo. Em aplicações que requerem um grande volume de produção, o alto custo do projeto e dos testes é amortizado.

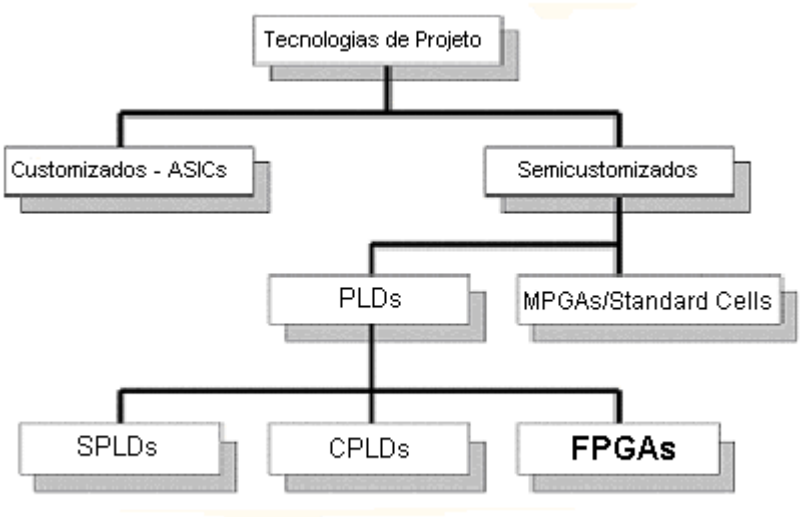

Figura 4 - Tecnologias para projetos de sistemas digitais (RIBEIRO, 2002).

Em MPGAs, o processo de fabricação é agilizado pelo uso de máscaras genéricas de módulos pré-projetados, mas ainda necessita de máscaras específicas para a interconexão dos módulos. O projeto é normalmente facilitado por uma biblioteca de células, proporcionando um tempo de desenvolvimento mais curto e custos mais baixos em relação aos CIs customizados;

A tecnologia *Standard Cells* se assemelha muito a das MPGAs. O projeto também é facilitado pelo uso de módulos pré-projetados. Os módulos (*Standard Cells*) são geralmente gravados em bancos de dados, onde os projetistas selecionam as células desejadas para realizar seus projetos. Em comparação aos CIs customizados, os circuitos implementados em *Standard Cells* são menos eficientes em tamanho e desempenho, entretanto, seu custo de desenvolvimento é mais baixo.

A tecnologia dos PLDs (Dispositivos Lógicos Programáveis) possui como principal característica a capacidade de programação (configuração) pós-fabricação pelo usuário,

facilitando assim as mudanças de projetos. Em comparação com outras tecnologias, os PLDs apresentam ciclo de projeto muito curto com baixos custos de desenvolvimento.

Segundo Ribeiro (2002):

> **"**O mercado de PLDs encontra-se em plena expansão, de forma que atualmente existem diversos fabricantes e modelos de dispositivos desse tipo. Uma das principais tarefas do projetista hoje é pesquisar e selecionar, dentre as opções dispositivos no mercado, qual a que melhor atende suas necessidades".

3.1 – Arquiteturas FPGAs.

Segundo Brown e Rose (1996):

> "Um FPGA consiste de um grande arranjo de células configuráveis (ou blocos lógicos) contidos em um único *chip*. Cada urna dessas células contém certa capacidade computacional para implementar funções lógicas, e/ou realizar roteamento para permitir a comunicação entre células. Essas operações podem acontecer simultaneamente no arranjo das células".

A arquitetura de um FPGA é bastante semelhante à arquitetura convencional de um MPGA. A principal diferença é que no MPGA as interconexões são feitas durante o processo de fabricação, como em circuitos integrados, enquanto os FPGAs são programados via comutadores programáveis, assim como os PLDs.

A arquitetura básica de um FPGA, ilustrada na Figura 5, consiste de um arranjo 2-D de blocos lógicos. A comunicação entre blocos é feita através dos recursos de interconexão. A borda externa do arranjo consiste de blocos especiais capazes de realizar operações de entrada e saída (I/O).

Existem várias arquiteturas de FPGAs comercialmente disponíveis, onde três aspectos principais definem a arquitetura de um FPGA:

- tipo de tecnologia de programação;
- arquitetura das células; e
- estrutura de roteamento.

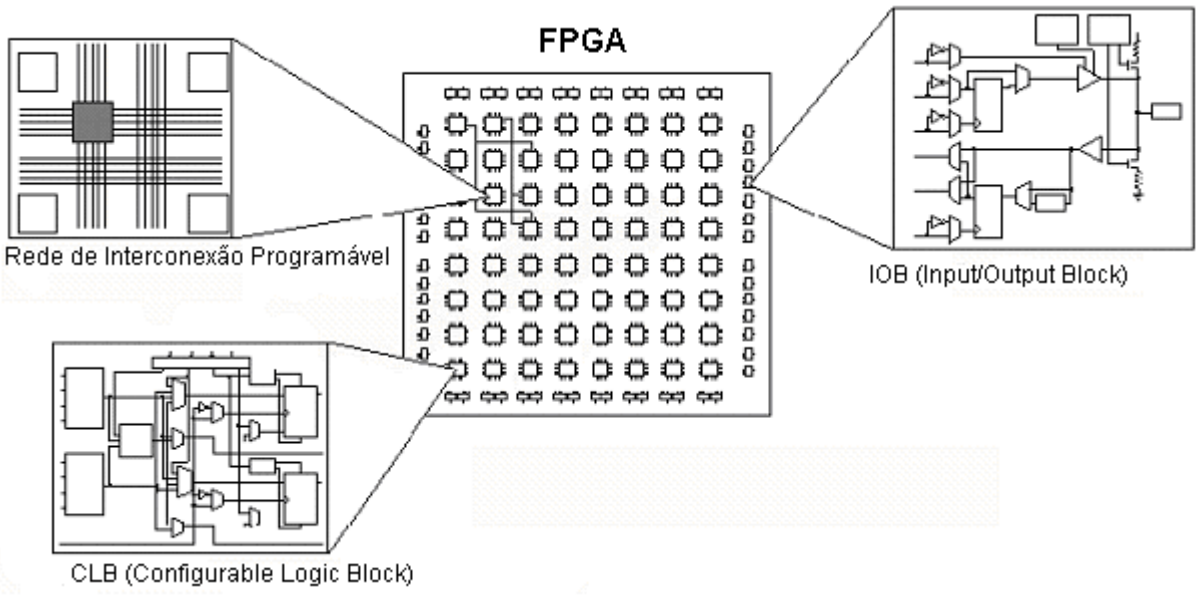

Figura 5 - Estrutura de um FPGA (RIBEIRO, 2002).

Estes aspectos influenciam diretamente o desempenho e a densidade das diferentes arquiteturas de FPGAs, entretanto não se pode afirmar que há uma melhor arquitetura, e sim a mais adequada para uma determinada aplicação (RIBEIRO, 2002).

3.1.1 – Tecnologia de programação.

Um FPGA é programado usando comutadores programáveis eletricamente. As propriedades desses comutadores, tais como tamanho, resistência, capacitância, tecnologia, afetam principalmente o desempenho e definem características como volatilidade e capacidade de reprogramação, que devem ser avaliadas na fase inicial do projeto para a escolha do dispositivo.

Em todos os tipos de FPGAs os comutadores programáveis ocupam uma grande área e apresentam valores de resistência e capacitância muito maior que as de um contato físico típico. Como conseqüência, o desempenho de um FPGA, se comparado a um MPGA de mesma tecnologia de fabricação, é menor.

Basicamente existem três tipos de tecnologias de programação:

- *Antifuse*: é originalmente um circuito aberto, que quando programado forma um

caminho de baixa resistência;

- *Gate Flutuante*: o comutador é um transistor com gate flutuante; e
- SRAM (*Static Randon Access Memory*): o comutador é um transistor de passagem controlado pelo estado de um bit de SRAM.

3.1.2 – Arquitetura de blocos lógicos

Os blocos lógicos dos FPGAs variam muito de tamanho e capacidade de implementação lógica. A construção dos blocos lógicos pode ser tão simples como um transistor ou tão complexo como um microprocessador. Geralmente são capazes de implantar uma ampla variedade de funções lógicas combinacionais e seqüenciais.

Os blocos lógicos dos FPGAs comerciais são baseados em um ou mais componentes, tais como, pares de transistores, portas básicas do tipo *NAND* ou *XOR* de duas entradas, multiplexadores, *Look Up Tables* (LUTs), estruturas *AND* e *OR* de múltiplas entradas e *flip-flops* associados com diversos tipos de portas lógicas.

A fim de classificar os FPGAs quanto à capacidade lógica dos blocos lógicos, pode-se dividi-los em três categorias de granularidade: fina, média e grossa. A primeira categoria designa os blocos simples e pequenos, a segunda os blocos mais complexos e a terceira, sistemas completos em um único bloco lógico.

3.1.3 – Arquitetura de roteamento

A arquitetura de roteamento de um FPGA é a maneira pela qual os comutadores programáveis e segmentos de trilhas são posicionados para permitir a interconexão dos blocos lógicos (SALCIC e SMAILAGIC, 2000). As arquiteturas de roteamento podem ser descritas a partir de um modelo geral, conforme a Figura 6.

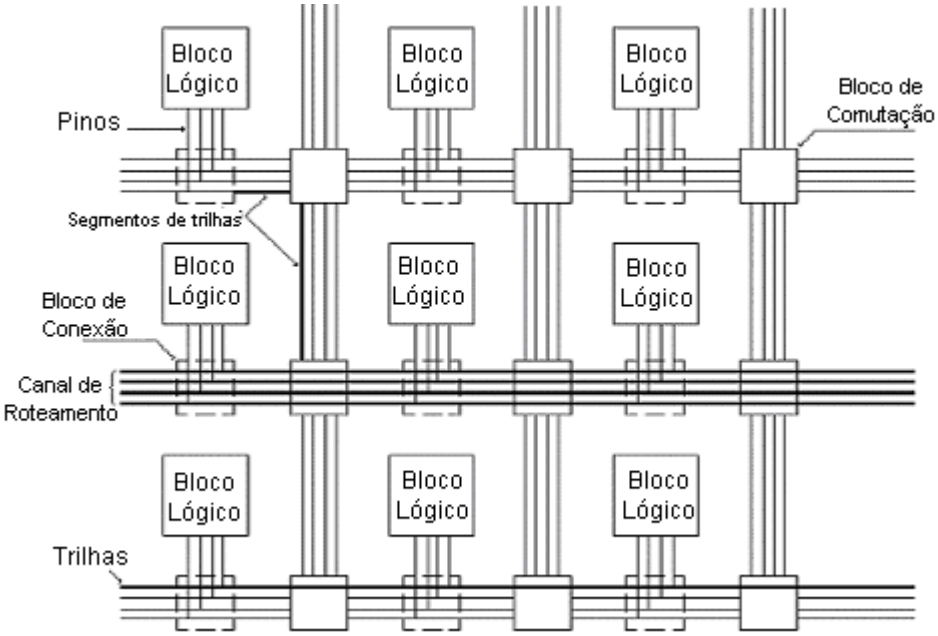

Figura 6 - Arquitetura geral de roteamento de um FPGA (RIBEIRO, 2002).

São necessários alguns conceitos para um melhor entendimento desse modelo, sendo os mesmos descritos a seguir.

Os pinos são as entradas e saídas dos blocos lógicos, é válido ressaltar que estes pinos são internos aos FPGAs, conforme indicado na Figura 6, e não devem ser confundidos com os pinos externos do encapsulamento que são ligados aos blocos de I/O.

Conexão é a ligação elétrica entre um pino e um segmento de trilha, as conexões são realizadas pelo bloco de conexão.

Bloco de conexão é o dispositivo utilizado para conectar eletricamente um pino e um segmento de trilha, estes dispositivos utilizam alguma tecnologia de programação, como as anteriormente apresentadas.

Os segmentos de trilha são os fios entre dois blocos de comutação. Já uma seqüência de um ou mais segmentos de trilha em uma direção, estendendo-se por todo comprimento de um canal de roteamento, conceitua-se como trilha e pode ser composta de segmentos de tamanhos variados;

O bloco de comutação (*switch*), é o dispositivo utilizado para conectar eletricamente

dois segmentos de trilha, estes dispositivos também utilizam alguma tecnologia de programação, como as já apresentadas.

O canal de roteamento é a área entre duas linhas ou colunas de blocos lógicos, sendo que um canal é formado por várias trilhas paralelas.

O modelo da Figura 6 contém duas estruturas básicas. A primeira é o bloco de conexões que aparece em todas as arquiteturas. O bloco de conexão permite a conectividade das entradas e saídas (pinos) de um bloco lógico com os segmentos de trilhas nos canais. A segunda estrutura é o bloco de comutação que permite a conexão entre os segmentos de trilhas horizontais e verticais. Em algumas arquiteturas, o bloco de comutação e o bloco de conexões são distintos, em outras estão combinados numa mesma estrutura. Nem todas as arquiteturas seguem esse modelo, pois há arquiteturas que apresentam uma estrutura hierárquica e outras que possuem somente canais horizontais (RIBEIRO, 2002).

Uma importante questão a ser considerada é se a arquitetura de roteamento permite que se alcance um roteamento completo e, ao mesmo tempo, uma alta densidade lógica. Sabe-se que usando um grande número de comutadores programáveis torna-se fácil alcançar um roteamento completo, mas esses comutadores consomem área, que é desejável minimizar. Algumas pesquisas estabelecem uma relação entre a flexibilidade da arquitetura de roteamento, a capacidade de roteamento e uso eficiente da área.

3.2 – Os FPGAs da Xilinx

O mercado oferece uma ampla gama de dispositivos FPGAs, de diversos fabricantes, em geral a estrutura de roteamento, os blocos lógicos e a forma de programação variam. Nota-se, no entanto, a existência de arquiteturas semelhantes que em geral podem ser classificadas em quatro classes, como ilustra a Figura 7. Dependendo da aplicação, uma arquitetura pode ter características mais desejáveis para a aplicação em questão.

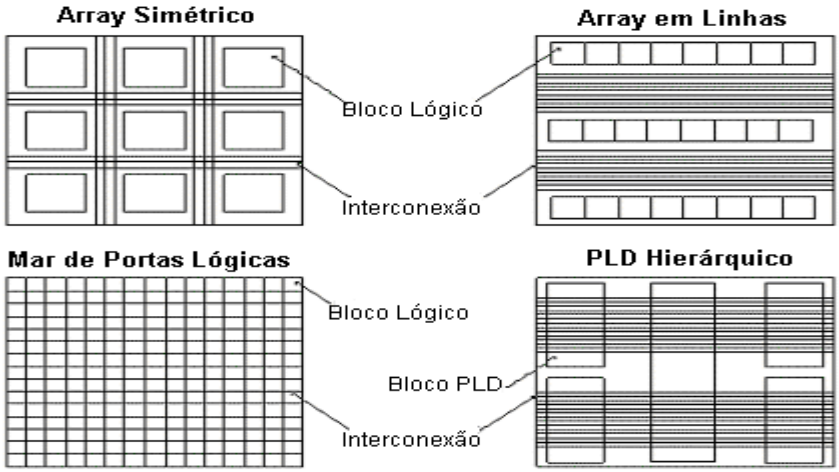

Figura 7 - Tipos de arquiteturas.

A Tabela 4 apresenta os principais fabricantes de FPGAs e suas tecnologias de programação.

Tabela 4: Fabricantes FPGAs X Tecnologia de programação

Fabricantes	**Tecnologia de Programação**
Actel Corp.	*Antifuse* / EEPROM
Altera Corp.	SRAM / EEPROM
Atmel Corp.	SRAM
Cypress Semiconductor Corp.	EEPROM
Integrated Circuit Technology (ICT) Corp.	EEPROM
Lattice Semiconductor Corp.	SRAM / EEPROM
Quicklogic Corp.	*Antifuse*
Xilinx Corp.	SRAM / EEPROM

A *Xilinx*, fundada em 1984, se tornou a primeira fabricante de FPGAs do mercado, lançando em 1985 a família XC2000. A cada ano ela oferece novas gerações, produzindo inovações tecnológicas que aumentam a capacidade lógica e a velocidade dos dispositivos, que já ultrapassam o patamar de 10 milhões de portas lógicas (XILINX, 2001). Algumas das famílias de dispositivos mais conhecidos e recentes da *Xilinx* são: a XC4000; a família

XC6200 que possui capacidade de reconfigurabilidade dinâmica e as atuais famílias Virtex, Virtex II e Virtex II Pro.

Em se tratando da família Virtex II, os dispositivos tiveram sua arquitetura otimizada para alta densidade e projetos lógicos de alto desempenho (XILINX, 2001). A Figura 8 mostra a arquitetura deste dispositivo, que entre outros elementos, é composto de: IOBs, CLBs, DCMs, blocos de SelectRAM, *clock* global e multiplicadores.

Os IOBs programáveis do dispositivo proporcionam interface entre os pinos físicos do encapsulamento e a lógica configurável interna, como em qualquer outro FPGA.

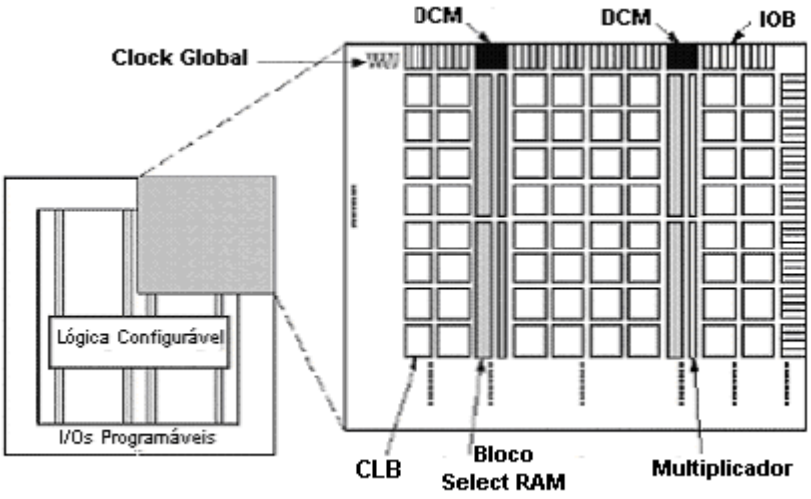

Figura 8 - Arquitetura do FPGA Virtex II (XILINX, 2001).

A lógica configurável interna possui quatro elementos principais organizados em um *array* regular, que são CLBs, os blocos SelectRAm, os blocos multiplicadores dedicados e os blocos DCM (*Digital Clock Manager*).

Os CLBs proporcionam elementos funcionais, de granularidade média, para lógica seqüencial e concorrente, assim como nas famílias XC4000 e Virtex, incluindo elementos básicos de armazenamento. Cada CLB deste é composto por 4 *slices* como os da família Virtex, mostrado na Figura 8.

Os blocos SelectRAM são módulos de memória RAM *dual* que fornecem elementos de armazenamento de 18 kbits, distribuído em colunas de espaçamentos regulares pelo dispositivo e não apenas nas extremidades como na família Virtex.

Os blocos multiplicadores dedicados: fornecem o produto de 18-bits X 18-bits.

Os blocos DCM (*Digital Clock Manager*) junto com o *Clock* Global fornecem uma completa solução para sistemas de alta freqüência e atrasos de *clocks* distribuídos.

Uma nova geração de recursos de roteamento chamado *Active Interconnect Technology* interconecta todos estes elementos. O GRM (*General Routing Matrix*) é um *array* de comutadores de roteamento. Cada elemento programável é ligado a uma *matrix* de comutadores, permitindo múltiplas conexões ao GRM. As interconexões programáveis globais são hierárquicas e projetadas para suportar altas freqüências.

Todos elementos programáveis, incluindo os recursos de roteamento, são controlados por valores armazenados em células de memória estática (SRAM). Estes valores são carregados nas células de memória durante a configuração e podem ser recarregados para mudar as funções dos elementos programáveis. Permitindo assim, a reconfiguração parcial e/ou dinâmica, necessária para suportar a implementação de hardware virtual. Esta capacidade de reconfiguração dinâmica também já era suportada pela família Virtex.

Em 9 de abril de 2001 a *Xilinx* anunciou seu kit de desenvolvimento *Microblaze Soft Processor*, para a plataforma Virtex II, como tentativa de competir com o sistema *Excalibur* da fabricante e concorrente Altera, lançado mundialmente 12 de junho de 2000. O *Microblaze* tem como processador alvo o *hardware core* do *PowerPC*, desenvolvido pela IBM. O processador alvo da Altera até então era o software *core* Nios, sendo que ela também veio a disponibilizar o *hardware core* ARM.

Em 4 de março de 2002 a *Xilinx*, junto com a IBM, anunciaram uma parceria para construção de *chips* híbridos utilizáveis em telecomunicações, armazenamento e aplicações de

consumo generalizado. Esta parceria utiliza a tecnologia da *Xilinx* de construção de FPGAs e a tecnologia da IBM de construção de *chips* em cobre a 0.13 e 0.10 mícron, sendo a primeira vez que a IBM constrói componentes para um cliente utilizando tecnologia geralmente aplicada em microprocessadores, *chips* customizados e memórias (IBM, 2002). Por norma, a IBM costuma desenvolver a tecnologia, que depois licenciava a outras empresas que depois se encarregam da construção dos produtos. O resultado desta parceria, por enquanto, foi o lançamento da família de dispositivos FPGAs Virtex II Pro da *Xilinx*, que entre outros elementos possui hardware *cores* do processador RISC *PowerPC* da IBM.

A Tabela 5 apresenta uma comparação entre os elementos de alguns dispositivos das famílias XC4000, XC6200, Virtex, VirtexII e Virtex II Pro da *Xilinx*. Esta comparação torna-se complicada, pois não existe um padrão certo para a contagem de cada elemento. Por exemplo, embora os *slices* são elementos que vêem mantendo um padrão desde a família Virtex, os CLBs da Virtex II e Virtex II Pro tem o dobro de *slices*.

Tabela 5 - Comparação de dispositivos *Xilinx* (XILINX, 2001)

Família	**Disposit.**	**CLBs**	**Slices**	**Cel. Lógicas**	**IOBs(Max)**	**Multip.**	**DCMs**	**mP**
XC4000	XC4085XL	3136	_	6.272	448	0	0	0
XC6200	XC6464	_	_	16.384	512	0	0	0
Virtex	XCV1000	6.144	12.288	24.576	512	0	0	0
Virtex II	XC2V8000	11.648	46.592	93.184	1.108	168	12	0
Virtex II Pro	XC2VP125	13.904	55.616	111.232	1.200	556	12	4

3.3 – Reconfiguração de FPGAs

Os dispositivos FPGAs podem ser classificados de acordo com suas configurabilidades como mostra a Figura 9 (LYSAGHT e DUNLOP, 1993). Todos dispositivos FPGAs são por definição programáveis, ou seja, configurável pelo menos uma única vez, como qualquer

PLD. Um pequeno subconjunto destes dispositivos pode ser reconfigurado várias vezes por alguma operação que recarrege completamente a configuração do dispositivo.

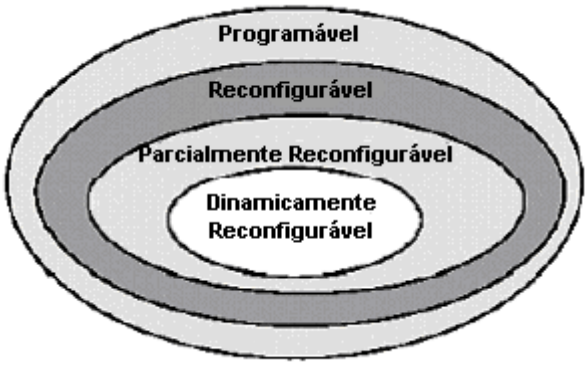

Figura 9 - Classificação de FPGAs de acordo com sua configurabilidade (RIBEIRO, 2002)

3.3.1 – Reconfiguração Parcial

Um dispositivo é definido como reconfigurável parcialmente se é possível reconfigurá-lo seletivamente, enquanto o resto do dispositivo permanece inativo, mas retém sua configuração.

Caso o dispositivo só possa ser reconfigurado por completo, mesmo que na própria placa do sistema, ou fora dela como com dispositivos EPROMs, ele é considerado apenas reconfigurável, pois obviamente ele não permanece ativo em nenhum dos casos, por não permitir configuração seletiva.

3.3.2 – Reconfiguração Dinâmica

FPGAs são classificados como dinamicamente reconfiguráveis se seus circuitos internos de armazenamento da configuração podem ser atualizados seletivamente, sem prejudicar o funcionamento da lógica restante que pode estar em operação. Estes dispositivos podem assim ser reconfigurados seletivamente enquanto estiverem ativos. Outros termos atualmente encontrados na literatura que se referem a reconfiguração dinâmica são, *run-time*

reconfiguration (RTR), *real-time reconfiguration*, *on-the-fly reconfiguration* e *in-circuit reconfiguration*. Todas essas expressões podem ser traduzidas também como reconfiguração em tempo de execução (RIBEIRO, 2002).

Estes FPGAs possuem características comuns: eles são *arrays* de blocos lógicos, de granularidade fina ou média, cuja configuração é controlada por escrita de dados em uma memória estática. Para um FPGA ser reconfigurável dinamicamente, a definição implica que ele deve ser capaz de ser reconfigurado parcialmente enquanto ativo (energizado e em operação). Em nível de sistema, um módulo que contenha múltiplos FPGAs reconfiguráveis, total ou parcialmente, pode ser classificado como reconfigurável dinamicamente se os componentes FPGAs são reconfigurados individualmente.

Como os tempos de configuração não são desprezíveis, a habilidade de intercalar execução e reconfiguração, sem prejuízo do desempenho, é uma questão que ainda merece atenção e esforços de pesquisa (RIBEIRO, 2002). Um sistema de reconfiguração dinâmica inclui pelo menos uma área de reconfiguração onde blocos lógicos podem ser carregados em tempo de execução. A Figura 10 ilustra a reconfiguração dinâmica de um sistema composto de cinco circuitos ou tarefas. A tarefa A é uma função de entrada que alimenta as tarefas B, C ou D, que alimentam a tarefa E que é a função de saída do sistema. As tarefas de entrada e saída são permanentemente residentes no FPGA enquanto as três tarefas dinâmicas alternam-se sob o controle de um sinal de *Swap*.

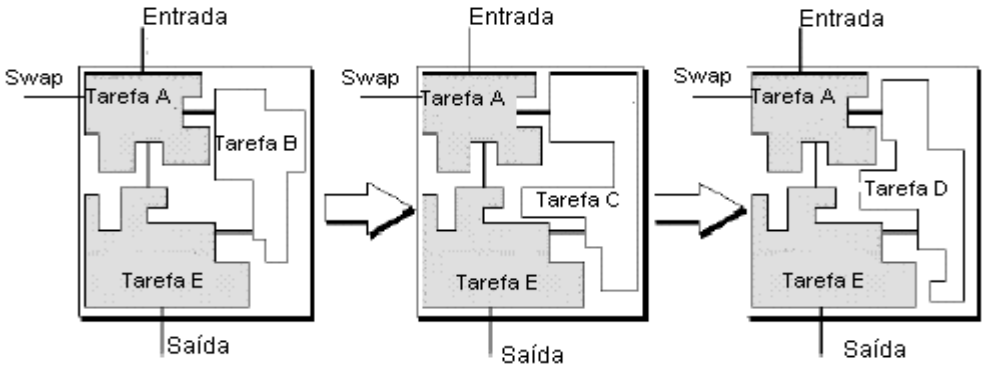

Figura 10 - Exemplo de reconfiguração dinâmica (RIBEIRO, 2002).

O gráfico da Figura 11, Ribeiro (2002), ilustra a execução, em função do tempo, das tarefas A, B, C, D e E da Figura 10, durante reconfiguração total do dispositivo. No gráfico da Figura 12 é ilustrada a execução das mesmas tarefas, durante reconfiguração dinâmica.

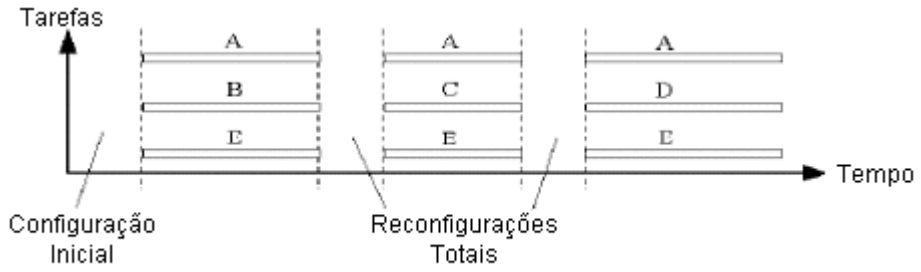

Figura 11 - Execução das tarefas com reconfiguração total (RIBEIRO, 2002).

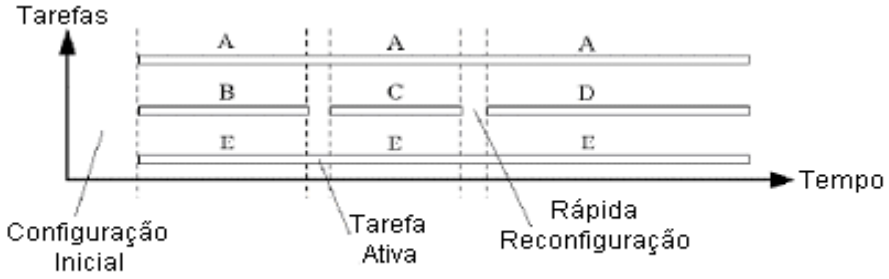

Figura 12 - Execução das tarefas com reconfiguração dinâmica (RIBEIRO, 2002).

A capacidade de reconfiguração dinâmica permite o compartilhamento ao longo do tempo de tarefas diferentes, o que pode reduzir significativamente a área de silício exigida. Esta tecnologia torna possível o conceito de *hardware* ilimitado ou "*Hardware Virtual*" (CARDOSO e VÉSTIAS, 1999). Este conceito é similar ao de memória virtual e está sendo adotado para permitir que múltiplos conjuntos de configuração sejam armazenados para serem usados pelas RPUs (*Reconfigurable Processing Units*), quando necessário. O *hardware* virtual é implementado por compartilhamento de tempo de uma dada RPU, necessitando que um escalonador seja responsável pelas configurações, execuções e comunicações entre as partições temporais.

O projeto de um sistema que explore essa possibilidade possui vários problemas (MERINO et al., 1998):

- o particionamento da descrição em alto nível de um sistema em blocos (tarefas) que possam ser trocadas (carregadas e descarregadas) na área reconfigurável;
- a busca de um escalonamento otimizado para esses blocos; e
- a determinação de um conjunto de serviços que auxiliam a executar apropriadamente a funcionalidade requerida.

A questão crucial nesta abordagem é a implementação de um controlador capaz de manipular a reconfiguração de todas essas tarefas. Este controlador deve oferecer (assim como Sistemas Operacionais) serviços como o carregamento e remoção de tarefas, escalonamento de tarefas e gerenciamento dos recursos.

A reconfiguração dinâmica tem sido proposta no projeto de alguns sistemas experimentais. A *Brighem Young University*, em Provo, tem usado computação reconfigurável para construir um computador com um conjunto de instruções dinâmicas – DISC: *Dynamic Instruction Set Computer* (WIRTHLIN e HUTCHINGS, 1995), que efetivamente casa um microprocessador com um FPGA e demonstra o potencial da reconfiguração automática usando configurações armazenadas. À medida que um programa é executado, o FPGA requisita a reconfiguração se o circuito designado não está residente. A arquitetura DISC permite que o projetista crie e armazene um grande número de configurações e as ative assim como um programador costuma chamar uma sub-rotina de *software* em um microprocessador.

3.3.3 – Dispositivos que permitem Reconfiguração Dinâmica

Nota-se, nas diversas pesquisas na área (SIDHU, 2000; SIMMLER e LEVISON, 2000; BREBNER e DIESSEL, 2001; MACBETH e LYSAGHT, 2001; MESQUITA, 2002), que a tecnologia de programação de FPGAs utilizada em arquiteturas de reconfiguração parcial e reconfiguração dinâmica, é a baseada em SRAM. É obvio o porque de não se utilizar

FPGAs *antifuse* ou baseados em EPROM, devido à incapacidade de reconfiguração parcial destas tecnologias.

Quanto á utilização de FPGAs baseados em EEPROMs, embora seja possível, a tecnologia SRAM apresenta inúmeras vantagens para arquiteturas dinamicamente reconfiguráveis, tais como: maior velocidade de reconfiguração, maior densidade lógica; mecanismos de roteamento que permitem funções mais complexas; e etc. Estes mecanismos apresentam maior nível de granularidade que FPGAs baseados em EEPROMs, que em geral fazem o roteamento por termos de produtos de *ANDs* e *ORs*.

Outro fato relevante a considerar é que a tecnologia SRAM, quando comparada com o processo de fabricação EEPROM, possui um processo de fabricação bem mais simples, tendo os dispositivos FPGAs baseados em SRAM evoluído tanto quanto as memórias de computadores que utilizam esta tecnologia. Uma desvantagem da tecnologia SRAM comparada com a EEPROM é a volatilidade.

Comparando a tecnologia SRAM com a, *antifuse*, além da desvantagem da volatilidade, a tecnologia SRAM consome mais energia, dada a maior impedância das conexões, são mais lentas e expõe a propriedade intelectual, ficando vulnerável à pirataria (RIBEIRO, 2002). Porém, como já citado, é impossível se quer obter reconfiguração parcial com a tecnologia *antifuse*.

Vale observar que embora a tecnologia SRAM apresenta inúmeras vantagens para a reconfiguração dinâmica de dispositivos FPGAs, não significa que todos dispositivos FPGAs que utilizam esta tecnologia são dinamicamente reconfiguráveis. Isso dependerá de como são as arquiteturas e mecanismos de configuração e roteamento de cada família e fabricante destes dispositivos.

A Tabela 6 compara os dispositivos FPGAs com tecnologia de programação SRAM, quanto à capacidade de reconfiguração.

Tabela 6 - Capacidade de reconfiguração de FPGAs SRAM (RIBEIRO, 2002)

Família	Fabricante	Parcial	Dinâmica
XC4000	Xilinx	Sim	Não
XC6200	Xilinx	Sim	Sim
Virtex	Xilinx	Sim	Sim
Virtex II	Xilinx	Sim	Sim
Virtex II Pro	Xilinx	Sim	Sim
Apex 20K	Altera	Não	Não
Apex II	Altera	Não	Não
Stratix	Altera	Não	Não
AT6000	Atmel	Sim	Sim
AT40K	Atmel	Sim	Sim
ORCA 4	Lattice	Sim	Sim

3.4 – Arquiteturas de computação reconfigurável

Normalmente um sistema de computação reconfigurável é implementado com múltiplos FPGAs (baseados em SRAM), os quais atuam como RPUs (*Reconfigurable Processing Units*), acoplados a um computador hospedeiro – *host* (CARDOSO e VÉSTIAS, 1999).

A junção de *hardware* dedicado com um computador *host*, para acelerar algumas tarefas computacionalmente intensivas, como mostra a Figura 13, não é um conceito novo. Um exemplo comum na indústria de *PCs* é o uso de placas aceleradoras gráficas. Porém, tais circuitos aceleradores para aplicações específicas requerem grande esforço de projeto e tipicamente usam mais área de silício que o próprio microprocessador.

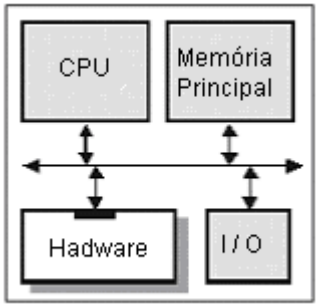

Figura 13 - *Hardware* dedicado com computador *HOST*, para acelerar tarefas computacionais.

O uso de *hardware* reconfigurável torna possível juntar o computador *host* com recursos de *hardware* mais flexíveis, melhor adaptados à aplicação sendo executada, e com a possibilidade de adaptar-se às novas versões de uma aplicação (CARDOSO e VÉSTIAS, 1999).

A partir do amadurecimento da tecnologia FPGAs, alguns centros de pesquisa criaram as primeiras arquiteturas reconfiguráveis, com o intuito principal de aumentar o desempenho de algoritmos, que até então eram executados em software. Dentro desta primeira geração estão projetos como o *PRISM* (ATHANAS e SILVERMAN, 1993) e o *Splash* (GOKHALE et al., 1990). Alguns sistemas mais modernos ainda utilizam essa abordagem, como o *Transmogrifier-2* (LEWIS et al.,1998), o *RPM-2* (DUBOIS et al., 1998) e o *Spyder* (SANCHEZ et al., 1999).

Já neste primeiro momento verificou-se a eficiência da utilização de FPGAs no domínio de aplicação em questão, tanto em termos de desempenho, com relação a abordagens em *software*, quanto no que tange ao critério econômico, quando comparada a soluções ASIC. Contudo também alguns problemas foram levantados. Em geral esses sistemas possuíam um gargalo de comunicação entre *host* e FPGAs e apresentavam um tempo de reconfiguração muito alto, além de poderem ser reconfigurados apenas totalmente. Essa última desvantagem significa que o sistema precisava necessariamente ser parado para que pudesse ser reconfigurado.

Em função destas dificuldades novas propostas de arquiteturas surgiram. A possibilidade de reconfiguração dinâmica, apresentada anteriormente, permitiu que as arquiteturas em questão pudessem ser reconfiguradas sem que precisassem parar totalmente de desempenhar suas funções, além de reconfigurar-se mais rapidamente por não precisar carregar todo o contexto novamente. Outro avanço tecnológico ocorrido foi o aumento do número de transistores por *chip*, tornando possível o desenvolvimento de sistemas SoCs que viessem a combater o problema de comunicação entre *hosts* e FPGAs.

O tipo de interconexão entre a RPU e o sistema hospedeiro e o nível de granularidade da RPU constituem dois importantes aspectos a serem considerados na implementação de sistemas reconfiguráveis. Muitos pontos relativos a esses tópicos já foram explorados e muitos ainda remanescem para pesquisa adicional. Porém, ainda não é possível identificar uma solução dominante para todos os tipos de aplicações (CARDOSO e VÉSTIAS, 1999).

3.5 – Acoplamentos de FPGAs

Embora seja possível (dependendo da capacidade do dispositivo e complexidade do sistema) implementar em um único dispositivo FPGA um completo sistema SoC, pode ser necessária a utilização de vários FPGAs para implementar pelo menos uma ou mais RPUs, ou para fornecer a capacidade de *hardware* desejado ao sistema. Assim surge a necessidade do acoplamento de FPGAs.

Neste caso não seria necessário o uso de barramentos complexos e genéricos como o PCI. Como se trata de acoplamento entre *chips*, pode-se utilizar algum barramento específico, ou até mesmo algum barramento independente da tecnologia como o AMBA (*Advanced Microcontroller Bus Architecture*) da empresa ARM (ARM, 1999).

A Figura 14 ilustra o acoplamento de diversos dispositivos do mercado, independentemente da tecnologia, por meio do barramento AMBA.

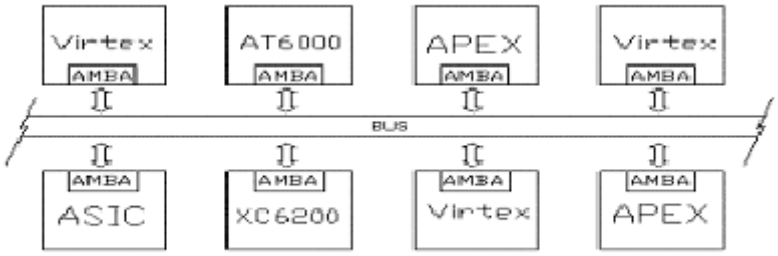

Figura 14 - Acoplamento de dispositivos através do barramento AMBA (ARM, 2004).

É interessante observar que para um sistema multi-FPGAs, que possa ser composto por um barramento de controle, como o AMBA, ou uma topologia de acoplamento, o *hardware*, que pode tratar-se de módulo FPGA, pode ser dinamicamente reconfigurável, mesmo utilizando-se dispositivos que não provêem tal capacidade individualmente. Considerando-se que a reconfiguração total de um único dispositivo do sistema não afete todo o funcionamento do mesmo, este sistema será dinamicamente reconfigurável. Embora tal reconfigurabilidade seja de granularidade muito grossa, esse artifício pode significar substancial economia de *hardware* para o sistema, visto que inúmeros contextos podem ser carregados por um *host* ou armazenados em memórias externas, caracterizando um sistema muito mais flexível, com um custo muito menor do que o de implementar todo o *hardware*.

4 – MECANISMOS DE COMUNICAÇÃO

Descrevem-se alguns mecanismos de comunicação, por os mesmos apresentarem características pertinentes ao objeto de estudo. Tais mecanismos trazem conceitos tanto de interconexão e roteamento interno, conhecidos como redes *intrachip*, como conexão entre dispositivos externos através de comunicação serial, dentre outras características particulares apresentadas por cada um deles.

A crescente complexidade de sinais, imagens e controle do processamento em aplicações em tempo real (*real-time)* embarcadas requerem alto poder computacional para satisfazer restrições desse tipo de sistema. Este poder pode ser alcançado através de arquiteturas de alto desempenho com sistemas embarcados diversificados, chamado ”*multicomponent*”, constituído por tipos diferentes de componentes programáveis (processadores RISC ou CISC, DSP etc) ou componentes do tipo ASIC e FPGA, usados para executar eficientemente tarefas de nível baixo como processamento de sinais, de imagem e controle de dispositivos. Eles são feitos de tipos diferentes de processadores interconectados por tipos diferentes de meios de comunicação (GRANDPIERRE et al., 1999).

Implementando tais arquiteturas heterogêneas distribuídas, verificaram-se as limitações severas do *real-time* geralmente apresentada como uma tarefa difícil e complexa (EDWARDS et al., 1997).

Embora as arquiteturas monoprocessadas (estações de trabalho), já previam um poder de computação crescente, eles não podem contar com a complexidade crescente de controle, sinal e imagem que processam aplicações. Necessitam de arquiteturas paralelas, para melhor satisfazer as limitações do tempo real (*real-time),* devido à natureza distribuída do recurso (sensor, memória etc) destas aplicações em tempo real (DIAS et al. 1998).

Uma função importante em sistemas em tempo real embarcados é a monitoração, que consiste na observação dos eventos do sistema, tornando possível validar seu comportamento temporal e o cumprimento de requisitos temporais em tempo de execução.

Possibilidades abertas pela monitoração em sistemas em tempo real (*real-time)* são, por exemplo a avaliação do desempenho da aplicação; a identificação da possibilidade de indisponibilidade de algum recurso; e os requisitos temporais sob risco de não atendimento pela ocorrência de alguma condição não prevista.

Barramentos, inserindo-se em um contexto de sistemas em tempo real (*real-time)*, compartilham de sua problemática relativa à validação do comportamento em tempo de execução. Em um sistema de controle baseado em barramento de campo, existem múltiplos instrumentos gerando informação, que devem comunicar-se através de um barramento de dados único e compartilhado. Em um tal ambiente surge a necessidade de escalonar a comunicação, isto é, coordenar a ocupação do barramento pelas mensagens de cada instrumento, sem que haja superposição de mensagens e de maneira que cada mensagem cumpra os requisitos de tempo impostos pela sua utilização.

As ferramentas disponíveis para configuração em redes, destes barramentos, normalmente apóiam a validação do sistema resultante apenas de forma restrita. Mecanismos apresentados por estas ferramentas incluem diagramas em nível de usuário, com indicação de falha de conexões de comunicação entre instrumentos, ou então, listagens de mensagens com registro de tempo (de instante de comunicação).

A importância da tecnologia de barramentos para o cenário da automação e controle industrial, por exemplo, e a carência de ferramentas de apoio à monitoração e validação de requisitos temporais para estes sistemas, levou a proposta de uma abordagem para estas questões.

Este projeto poderá se tornar uma contribuição para observar o desempenho em tempo real (*real-time)* de distintas configurações em barramentos. Os resultados podem esclarecer algumas condições dentro das quais o desempenho, como por exemplo, da solução I2C, que será brevemente descrita, ou de outros barramentos.

4.1– AMBA (*Advanced Microcontroller Bus Architecture*)

O barramento AMBA (*Advanced Microcontroller Bus Architecture*) tem se tornado um dos mais populares padrões de arquitetura à medida que SoCs (*Systems on Chip*) estão ficando cada vez mais complexos e as pressões de *time-to-market* têm aumentado. Além disso, pode-se observar o grande crescimento da arquitetura AMBA entre as mais variadas plataformas.

Embora iniciado e conduzido pela empresa ARM, o padrão AMBA é aberto e pode ser licenciado sem a cobrança de taxas.

A Figura 15 mostra um típico sistema AMBA. AMBA é uma especificação de barramento que detalha uma estratégia para a interconexão e gerenciamento de blocos funcionais num SoC.

AMBA especifica três tipos de barramento, O ASB (*Advanced System Bus*), o AHB (*Advanced High- performance Bus*) e o APB (*Advanced Peripheral Bus*).

O principal objetivo do barramento AMBA é facilitar o desenvolvimento de projetos de sistemas embarcados, ser independente de tecnologia, assegurar a reusabilidade de periféricos e macrocélulas, encorajar projetos modulares de sistemas e diminuir a infra-estrutura de silício necessária para dar suporte à comunicação *on-chip* e *off-chip*.

A primeira geração de barramentos AMBA foi a ASB, usado como barramento principal do sistema, suportando a conexão eficiente de módulos de alto desempenho, como processadores e controladores de memórias. ASB implementa as características necessárias

para alto desempenho, como operação em *pipeline*, transferências em rajadas (*burst transfers*), múltiplos mestres de barramento e permissão para transferência de dados de 8, 16 ou 32 bits.

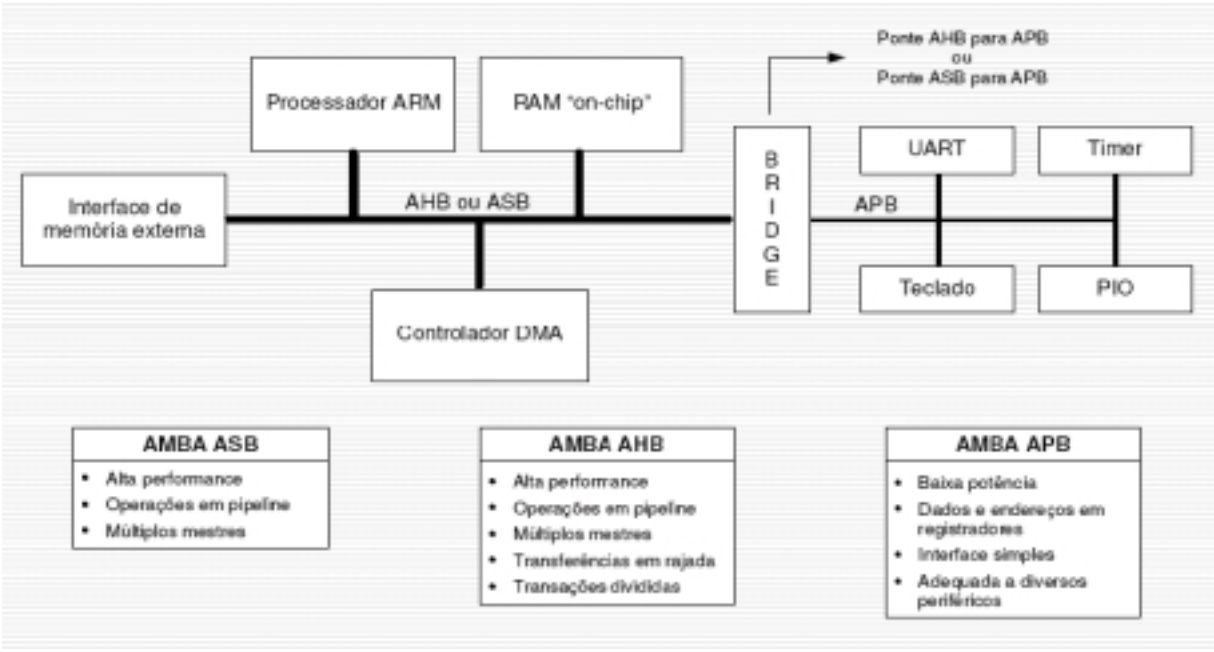

Figura 15 – Um típico sistema AMBA (ARM, 1999).

Um projeto de um sistema ASB geralmente contém os componentes: mestre, escravo, decodificador e árbitro.

O Mestre tem a função de iniciar as operações de leitura e escrita, fornecendo um endereço e informações de controle. O Escravo responde às operações de leitura e escrita executadas em um determinado espaço de endereço. Já o Decodificador tem a função de decodificar os endereços das operações de leitura e escrita, escolhendo o escravo corretamente e o Árbitro assegura que apenas um único mestre inicialize operações de leitura e escrita em cada momento.

AHB é a nova geração de barramento AMBA, consistindo em uma versão melhorada do barramento ASB. O AHB suporta sistemas com alta freqüência de *clock*, transações divididas (*split transactions*), troca de mestre de barramento em um único ciclo de *clock*,

transferência de dados de até 1024 *bits*, implementação com multiplexadores e pode ser facilmente integrado a sistemas ASB/ APB existentes.

Um microcontrolador baseado em barramentos AMBA consiste basicamente de barramento de alto desempenho, onde estão localizados os dispositivos com maior taxa de transferência, barramento de baixo desempenho onde estão localizados a maior parte dos periféricos e de *bridge* conectando o barramento de alto desempenho ao de baixo desempenho.

Observam-se algumas notas sobre a especificação AMBA na Tabela 7.

Tabela 7 - Notas sobre a especificação AMBA (ARM, 1999).

Especificações	**Notas**
Independência de tecnologia	O AMBA é um protocolo que independe da tecnologia do chip no qual será usado.
Características elétricas	A especificação AMBA não fornece informações sobre as características elétricas do sistema, pois elas serão inteiramente dependentes do processo de fabricação utilizado.
Especificações de tempo	O protocolo AMBA define o comportamento dos sinais ao nível de clock. Os requisitos de tempo exatos dependem da tecnologia usada e da freqüência de operação.

4.2 – *Fieldbus (Foundation Fieldbus)*

Fieldbuses constituem-se em um moderno paradigma para sistemas de controle e automação industrial (SILVA NETO et al., 1995). Tal paradigma é caracterizado pela utilização de dispositivos (instrumentos) de campo, com capacidade local de processamento, e que se comunicam entre si. Combinam-se assim benefícios oriundos da instrumentação digital, da distribuição do processamento e da comunicação de dados entre dispositivos.

Fieldbuses, enquanto componentes de sistemas de controle e automação industrial, são considerados sistemas tempo real, significando que devem atender a determinados requisitos temporais, além de garantir a correção lógica do processamento. Em tais sistemas de natureza distribuída, a comunicação desempenha um papel crítico no comportamento temporal e, por conseqüência, no atendimento a requisitos temporais.

Entre as diversas propostas existentes para uma normatização na área de barramentos de campo, destaca-se o *Foundation Fieldbus*, oriundo de uma organização chamada *Fieldbus Foundation (FIELDBUS FOUNDATION*, 1999).

Fieldbuses eliminam a necessidade de se utilizar várias interfaces ponto-a-ponto, uma para cada equipamento, substituem as interfaces digitais ponto-a-ponto (como RS232, RS422, etc.) por um barramento ao qual todos os equipamentos são conectados e são geralmente usados na comunicação em ambiente industrial e veicular.

Existem diversos tipos de *fieldbus,* como por exemplo, o *Profibus* (norma alemã), o ISA SP-50 (norma americana), o FIP (norma francesa), dentre outros.

Fieldbus é um sistema de comunicação digital bidirecional que interliga equipamentos inteligentes de campo com sistema de controle ou equipamentos localizados na sala de controle.

Algumas vantagens apresentadas pelo *Fieldbus* seriam a redução no custo de fiação, instalação, operação e manutenção de plantas industriais; informação imediata sobre diagnóstico de falhas nos equipamentos de campo, onde os problemas podem ser detectados antes deles se tornarem sérios, reduzindo assim o tempo de inatividade da planta; a distribuição das funções de controle nos equipamentos de campo - instrumentos de medição e elementos de controle final.

Outra vantagem do *Fieldbus é* o aumento da robustez do sistema, visto que dados digitais são mais confiáveis que analógicos; a melhoria na precisão do sistema de controle,

visto que conversões D/A e A/D não são mais necessárias, conseqüentemente a eficiência da planta será aperfeiçoada; e também redução de custo de engenharia, e de cabos; melhoria na qualidade das informações; indicação de falhas em equipamentos em tempo real; facilidade na manutenção.

Em relação ao aspecto da rede, as vantagens seriam a maior modularidade da rede, facilitando sua expansão; baixo custo, o que torna os *fieldbuses* ideais para interligação de componentes mais simples; e maior compatibilidade devido ao uso de padrões.

Por se tratar de uma comunicação puramente digital é necessário que se estabeleçam regras para que seja possível a interoperabilidade entre instrumentos de fabricantes diferentes.

Inicialmente cada fabricante procurou desenvolver sua própria tecnologia, ficando o usuário final subordinado àquela rede proprietária.

A partir da união de grandes empresas surgem duas vertentes mundiais, a *FIELDBUS FUNDATION* formada basicamente por empresas americanas e a *FIELDBUS PROFIBUS* formada por empresas européias.

A *FIELDBUS PROFIBUS* saiu na frente e estabeleceu seus padrões, tendo mais de 1400 instrumentos de diversos fabricantes aprovados nos testes de conformidade e com o certificado da fundação (SILVA NETO et al., 1995).

Já a *FIELDBUS FUNDATION* completou o seu processo de padronização no final do ano de 1997. Na Figura 16 tem-se um exemplo de uma arquitetura de rede *Fielbus*, onde pode-se observar a estação de supervisão, uma placa de interface com múltiplos canais, o barramento linear, terminador do barramento (BT-302), fonte de alimentação (PS-302), impedância (PSI-302) e diversos instrumentos, inclusive um CLP com placa de interface para o barramento.

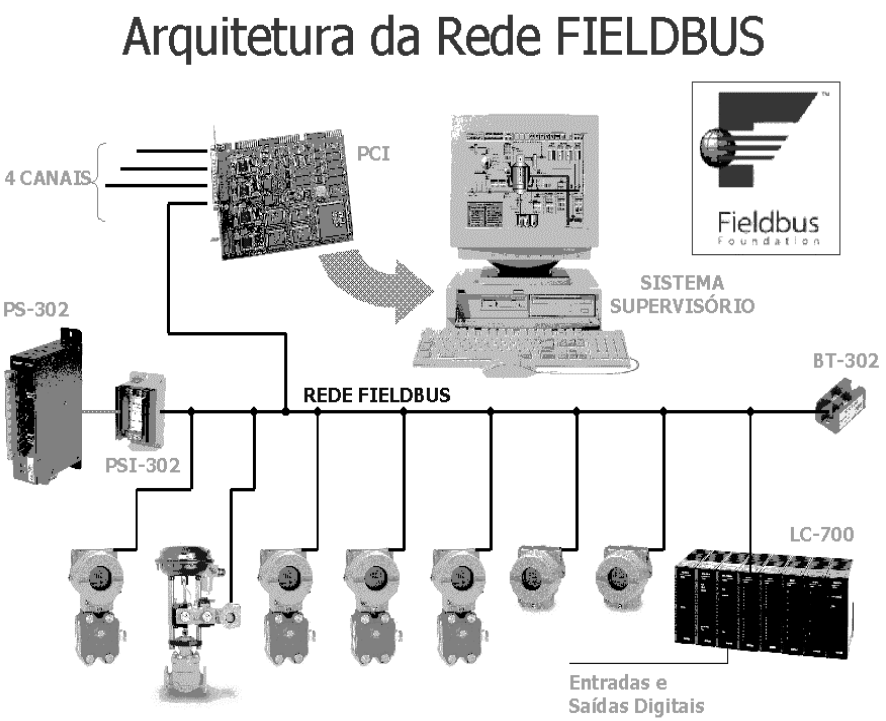

Figura 16 - Exemplo de uma arquitetura de rede *Fieldbus*. Ref.: (SILVA NETO, 1995).

4.3– CAN (*Controller Area Network*)

O barramento CAN é um protocolo de comunicação serial desenvolvido inicialmente pela *Bosch* em 1986 para aplicações automotivas e recentemente vem sendo utilizado em sistemas de automação industrial e domótica.

Sendo originalmente concebido para a indústria automóvel, isso lhe conferiu logo duas vantagens, o elevado volume de fabricação (baixo preço) e a elevada imunidade ao ruído, uma vez que se destinava a aplicação num ambiente tradicionalmente hostil.

O CAN tem características próprias que o vocacionam para aplicações em áreas onde alguns outros barramentos não são boas soluções. Podendo atingir velocidades e distâncias até Mbps e Km (uma ou outra). O CAN usa apenas dois condutores como meio físico de comunicação.

O Protocolo CAN é baseado na técnica de CSMA/CR (*Carrier Sense Multiple Access/Collision Resolution*), às vezes também chamado de CSMA/CD + AMP (*Carrier*

Sense Multiple Access/Collision Detection and Arbitration on Message Priority), de acesso ao meio de transmissão. Isto significa que sempre que ocorrer uma colisão entre duas ou mais mensagens, a de mais alta prioridade terá o acesso ao meio físico assegurado e prosseguirá a transmissão.

As características básicas do barramento CAN são as seguintes: 8 *bytes* de dados, velocidade de até 1Mbit/s, priorização de mensagens, recepção *multicast* com sincronização, detecção de erros e sinalização e retransmissão automática de mensagens corrompidas. Todas estas características propiciam simplicidade, alta confiabilidade e segurança, além de baixo custo.

A primeira versão do protocolo, CAN 2.0A (BOSCH, 1991), especificava somente mensagens do tipo padrão com identificadores de 11 bits enquanto que o CAN 2.0B (BOSCH, 1991), admite também mensagens estendidas com identificadores de 29 *bits*. O protocolo CAN foi adotado em 1993/94 como padrão mundial ISO11898 pela *International Standard Organization*.

O protocolo é usualmente implementado em um controlador na forma de um circuito integrado, mas também se pode encontrar no mercado microcontroladores de 8, 16 e 32 bits com controladores CAN integrados. Os controladores CAN e os microcontroladores com controladores CAN integrados são fabricados por um grande número de indústrias tais como Intel, Motorola, Philips, Siemens e Texas Instruments, resultando em torno de 50 circuitos integrados de 15 fabricantes diferentes.

A utilização do protocolo CAN na industria automobilística resultou numa produção em grande escala de controladores CAN, atualmente na casa dos milhões ao ano.

A afirmação e crescimento do protocolo CAN está fortemente baseado na organização dos fabricantes em torno de associações tais como a CiA (*CAN in Automation*) assim como na existência de uma série de ferramentas de *software* para o desenvolvimento, simulação,

configuração e monitoração de aplicações bem como na disponibilidade de *hardware* na forma de placas de controle, placas ISA, PCI e outros.

De acordo com o modelo OSI/ISSO, o padrão CAN 2.0B – Tabela 8, é constituído somente de duas camadas: *Data Link Layer* e *Physical Layer*. As características do barramento CAN podem ser resumidas por:

- É baseado no conceito de *broadcast*.

- Um esquema de arbitragem não destrutiva (*bitwise arbitration*) descentralizada, baseada na adoção dos níveis dominante e recessivo, é usada para controlar o acesso ao barramento.

- As mensagens de dados são pequenas (no máximo oito *bytes* de dados) e são conferidas por *checksum*.

- Não há endereço explícito nas mensagens. Em vez disso, cada mensagem carrega um identificador que controla sua prioridade no barramento e que pode servir como uma identificação do conteúdo da mesma.

- Utiliza um elaborado esquema de tratamento de erros que resulta na retransmissão das mensagens que não são apropriadamente recebidas.

- Fornece meios efetivos para isolar falhas e remover nós com problemas do barramento.

Tabela 8 - Modelo OSI/ISO do protocolo CAN

CAMADA	**FUNÇÃO**	**ESPECIFICAÇÃO**
OSI – Camada 7	Aplicação	Especificado pelo projetista
OSI – Camada 6	Apresentação	Vazio
OSI – Camada 5	Sessão	Vazio
OSI – Camada 4	Transporte	Vazio
OSI – Camada 3	Rede	Vazio
OSI – Camada 2	Enlace	Coberto pelo CAN e padrão ISO
OSI – Camada 1	Física	Coberto pelo ISO e parcialmente pelo CAN

- Oferece meios para filtragem das mensagens.

- O meio físico de transmissão pode ser escolhido de acordo com as necessidades. O mais comum é o par trançado, mas também podem ser utilizados outros meios de transmissão tais como fibra ótica e rádio freqüência.

Uma das propriedades mais importantes do barramento é o esquema de arbitração binária (*bitwise arbitration*) que fornece uma boa maneira de resolver colisão de mensagens. Sempre que dois nós começarem a transmitir mensagens ao mesmo tempo, o mecanismo de arbitragem garante que a mensagem de mais alta prioridade será enviada.

Isto é conseguido através da definição de dois níveis de barramento chamados recessivo e dominante. Um nível dominante sempre sobrescreve um nível recessivo. Assim, enquanto um nó está enviando uma mensagem, ele compara o nível do bit transmitido com o nível monitorado no barramento. Se um nó tenta enviar um nível recessivo e detecta um dominante ele perde a arbitragem e interrompe o processo de transmissão.

A taxa de transmissão, por sua vez, é limitada pelo comprimento do barramento utilizado. No caso do uso de par trançado, por exemplo, tem-se uma determinada relação entre a taxa de transmissão e o comprimento máximo do barramento, como mostra a Tabela 9.

Tabela 9 - Taxa de transmissão X Distância para barramento CAN

Taxa (Kbit/s)	Distância Max. (m)
1000	40
500	130
250	270
125	530
100	620
50	1300
20	3300
10	6700
5	10000

Essa limitação decorre da necessidade de sincronização entre as estações componentes do barramento a qual dá suporte ao esquema de arbitração binária.

O barramento CAN na sua versão 2.0B utiliza-se de quatro tipos de quadros (*frames*) para controlar a transferência de mensagens. São eles, o quadro de Dados (*Data Frame*), o

quadro Remoto (*Remote Frame*), o quadro de Erro (*Error Frame*) e o quadro de Sobrecarga (*Overload Frame*).

O Quadro de Dados leva dados do transmissor para os receptores e é composto de sete campos diferentes mostrados na Figura 17:

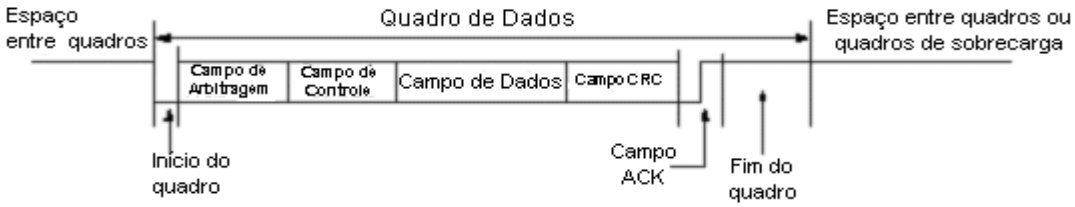

Figura 17 - Quadro de Dados

- Início do quadro: marca o início do quadro com um *bit* dominante simples.

- Campo de arbitragem: no formato padrão - Figura 18, consiste de um identificador de 11 bits e do *bit* RTR, o qual indica se é um Quadro de Dados (RTR = _dominante_) ou Quadro Remoto (RTR = _recessivo_). No formato estendido - Figura 19, é formado por um identificador de 29 *bits*, do *bit* SRR (_recessivo_, substitui o *bit* RTR do formato padrão), *bit* IDE (identifica se o quadro é padrão, IDE = _dominante_, ou estendido, IDE = _recessivo_) e pelo *bit* RTR.

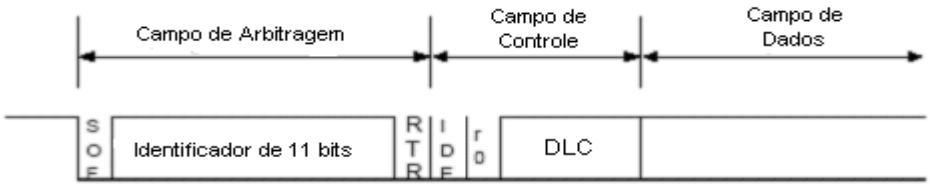

Figura 18 - Quadro de Dados Padrão

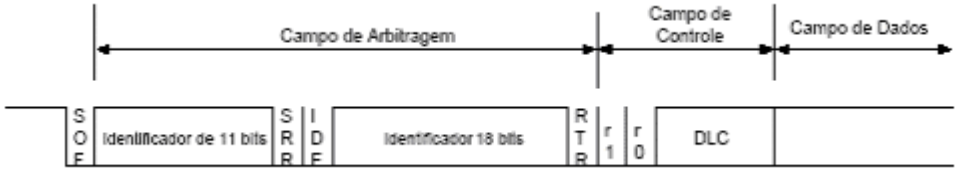

Figura 19 - Quadro de Dados Estendido

- Campo de controle: no formato padrão é constituído do DLC (*Data Length Code*), que indica o número de bytes no campo de dados, e dos *bits* IDE e r0 (reservado). No formato estendido é formado pelo DLC e pelos *bits* reservados r1 e r0.

- Campo de dados: consiste dos dados a serem transmitidos pelo Quadro de Dados.

- Campo CRC: contém a seqüência de CRC (*Cyclic Redundancy Code*) de 15 *bits* seguida por um delimitador (1 *bit*).

- Campo ACK: é composto pelo ACK *Slot* (1 *bit*) e Delimitador de ACK (1 *bit*). É utilizado pelos nós receptores para indicar o correto recebimento de uma mensagem.

- Fim do quadro: é formado por uma seqüência de sete *bits* _recessivos_ e serve para delimitar o quadro.

4.3.1 Interfaceamento com microcontrolador

Um nó é geralmente conectado a um barramento constituído de dois fios terminados em ambas as pontas por dois resistores. O controlador CAN pode estar diretamente conectado ao microcontrolador ou como dito anteriormente, o microcontrolador pode possuir um controlador CAN internamente. A ligação do nó com o meio físico normalmente se dá através de um *transceiver*, o qual se comunica serialmente com o controlador através do pino de saída Tx e do pino de entrada Rx.

A função do *transceiver*, no caso do uso de par trançado, é transformar os bits que entram e saem do controlador em tensão diferencial a ser aplicada ao barramento. O modo de transmissão diferencial é utilizado para proporcionar imunidade a interferências eletromagnéticas ao barramento. A alimentação do *transceiver* é feita com cinco volts e o nível _recessivo_ corresponde a uma diferença de tensão menor do que 0,5V entre CAN_H e CAN_L, com a tensão em CAN_H sendo normalmente maior do que CAN_L. Já o nível _dominante_ é detectado quando a diferença de tensão for de no mínimo 0,9V, sendo que a tensão nominal neste estado é de 3,5V para CAN_H e 1,5V para CAN_L. Convém lembrar

que os sinais dos pinos Tx e Rx possuem direções definidas enquanto que os sinais CAN_H e CAN_L não. O esquema deste tipo de conexão é mostrado na Figura 20.

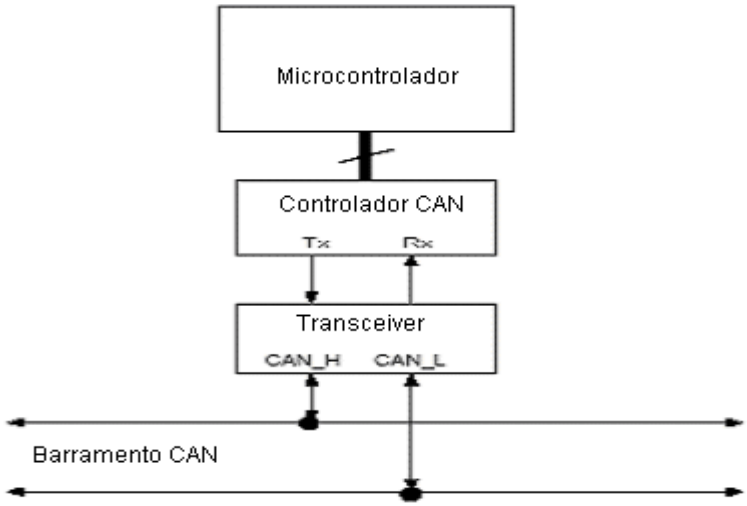

Figura 20 - Esquema para a interface do microcontrolador com o barramento CAN

4.3.2 - Conexão ao Barramento CAN

A forma recomendada de conexão ao barramento, mostrada na Figura 21, é feita através de dois fios: CAN_H e CAN_L. Além disso, recomenda-se utilizar dois terminadores compostos de resistores.

Figura 21 - Forma recomendada de conexão ao barramento

4.4– I2C (*Inter – IC*)

O I2C-bus (*Inter Integrated Circuit-bus*) é um protocolo de comunicação que foi desenvolvido pela divisão de semicondutores da Philips no início da década de 80. Este

protocolo tem por objetivo criar um caminho mais simples para se conectar dispositivos através da comunicação serial bidirecional, composta por dois fios: a linha serial de dados (SDA) e a linha serial de *clock* (SCL).

O I2C é considerado padrão que define tecnologia para interligação de circuitos integrados através de duas vias. I2C é a contração de *Inter-IC*, que significa "entre circuitos integrados".

I2C é um Bus (barramento) serial de comunicação bidirecional a dois fios. É um eficiente método de transmissão de dados em curtas distâncias entre vários dispositivos (PHILIPS, 2000). Tem aplicações em placas com processadores embarcados, equipamentos de vídeo e áudio, aplicações em sistemas de telecomunicações e automotivos. Trata-se de um barramento serial especial, que se destina a conectar, por exemplo, os microcontroladores a seus periféricos externos. O controle do tráfego de informação entre eles é assumido por seu próprio protocolo.

Os dados são transferidos em ambas as direções com taxas de 100kbits/s em modo padrão, taxa de 400kbits/s em modo rápido ou até 3,4Mbits/s em modo *High Speed*. Tendo como requisito apenas duas linhas seriais, que são: SCL e SDA.

A instalação pode ter até 8.0m de distância, suportando um máximo de 40 componentes I2C, com uma taxa de transferência de até 100 Kb/s, no modo padrão. Através desta tecnologia, que conta com um crescente número de componentes no mercado, é possível interconectar microcontroladores, ou controlar cargas externas, interfaces analógicas, sintetizadores de voz, sensores, *chips* de áudio e de controle compatíveis, tudo com grande flexibilidade de projeto.

Presta-se àquelas aplicações cujos critérios de projeto encaixam-se em sistemas completos que consistem de pelo menos um microcomputador e outros dispositivos periféricos, tais como memória e portas de I/O; custo de conexão dos vários dispositivos do

sistema deve ser o mínimo possível; sistemas que realizam funções de controle e não requerem altas taxas de transferência de dados; a eficiência global do sistema depende dos dispositivos escolhidos e da estrutura do barramento de interconexão.

Para construir tais aplicações, qualquer sistema que se encaixe nos critérios acima enumerados deverá possuir um barramento serial para interligação dos CIs. Embora este tipo de barramento não tenha a capacidade de fluxo de dados dos barramentos paralelos, requerem menos fiação e uma quantidade menor de pinos de conexão.

O barramento I2C atende a todos estes critérios e prevê procedimentos para evitar todas as possibilidades de colisão, perda de dados e bloqueio de informações (travamentos), determinação do dispositivo controlador do barramento, e a fonte do *clock* de sincronismo no caso de existirem dispositivos de diferentes velocidades interligados ao barramento.

Este protocolo tem por objetivo criar um caminho mais simples para se conectar *chips* através da comunicação serial bidirecional, composta por dois fios - (SDA – linha serial de dados, e SCL - linha serial de *clock*).

Cada componente conectado é reconhecido pelo *software* com um único endereço e uma simples relação mestre/escravo é quem faz a comunicação. O mestre pode operar como transmissor ou receptor. Durante a transmissão pode-se enviar mensagens sem restrição do número de *bytes* a serem enviados.

Os dispositivos assumem também a postura de mestre (M) ou escravo (S), quando realizando transferência de dados. O M é o dispositivo que inicia uma transferência de dados no barramento e gera o sinal de *clock* (Clk) para sincronizar a transferência; e o S é qualquer dispositivo endereçado pelo M.

O I2C é um barramento multi-master, ou seja, mais de um dispositivo pode controlar o barramento, mas não simultaneamente. Como exemplo de ocorrência desta condição, tem-se a transferência de dados entre dois microcomputadores conectados ao barramento I2C:

Seja a transferência de dados entre dois microcomputadores, cujas ações seriam resumidas da seguinte forma:

a) Micro A quer transmitir informação para Micro B:
Micro A (M) endereça Micro B (S);
M envia dados para S;
M termina a transferência.

b) Micro A quer receber informação do Micro B
Micro A (M) endereça Micro B (S);
M recebe dados enviados pelo S;
M termina a transferência.

Obs.: mesmo neste caso, o M (Micro A) é aquele que gera a temporização (Clk) e termina a transferência.

A possibilidade da existência de mais de um dispositivo do tipo M ser interligado no barramento I2C leva a uma situação indesejada em que mais de um M pode tentar iniciar uma transferência de dados ao mesmo tempo, gerando colisão. Para evitar este problema, existe um procedimento de escolha do M, baseado na lógica *wire-AND* (lógica E por fio) da conexão de todos os dispositivos ao barramento I2C: os dois *masters* colocam dados na linha SDA, e o primeiro a produzir um *bit* 1 quando o outro produzir um *bit* 0 perderá a escolha.

As linhas SDA e SCL são bidirecionais e são conectadas a uma fonte de tensão positiva através de resistores de *pull-up* (quando o barramento está livre, ambas as linhas estão em nível alto - *HIGH*). O número de dispositivos que podem ser conectados a elas é limitado apenas pela capacitância total do barramento.

Um exemplo genérico do barramento I2C é apresentado na Figura 22, onde é apresentado o barramento com diferentes dispositivos conectados ao mesmo.

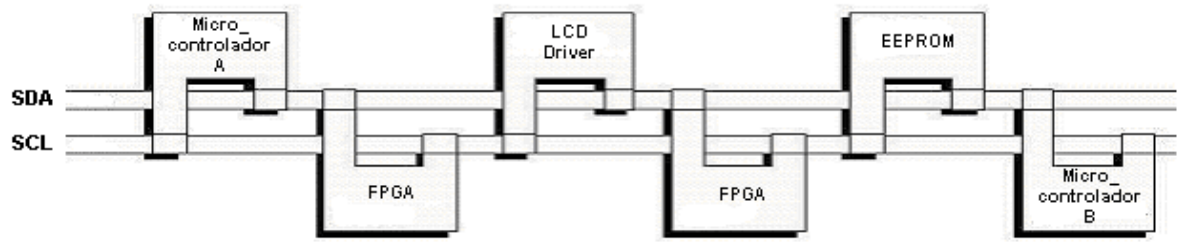

Figura 22 - Exemplo genérico de um barramento I2C

4.4.1 - Características do modo de velocidade padrão do barramento I2C

Para acompanhar a crescente demanda de dispositivos com velocidades mais altas e para tornar acessíveis mais endereços, a especificação I2C foi melhorada através dos anos e hoje está disponível com os seguintes aperfeiçoamentos:

- Modo rápido, com uma taxa de transferência de 400 kbits/s.

- *HIGH-SPEED MODE (HS-Mode),* com uma taxa de transmissão de 3.4 Mbits/s.

- 10 bits de endereçamento, que permite o uso de mais de 1024 endereços escravo adicionais.

Há duas razões principais para estender a especificação do Protocolo I2C:

- Muitas das aplicações atuais necessitam transferir grandes quantidades de dados em série e requer taxa de transferência acima de 100 Kbits/s (modo padrão), ou mesmo 400 Kbits/s (modo rápido), como resultado de contínuos melhoramentos em tecnologia de semicondutores, dispositivos I2C estão agora disponíveis com taxa de 3.4 Mbits/s(Hs-Mode) sem nenhum aumento no custo de fabricação do circuito.

- Como a maioria dos 112 endereços disponíveis com o endereçamento de 7 bits foram rapidamente utilizados, tornou-se claro que mais combinações de endereços tornaram-se necessários. Este problema foi resolvido com o novo projeto de endereçamento de 10 bits, que permitiu um aumento de aproximadamente 10 vezes nos endereços disponíveis.

4.4.2 - Características do modo rápido

Dispositivos no modo rápido podem receber e transmitir a uma taxa de 400 Kbits/s. O requisito mínimo é que possa haver sincronia com uma taxa de 400 Kbits/s, podendo então prolongar o período baixo do sinal SCL para diminuir a taxa de transferência. Dispositivos no modo rápido são compatíveis e podem comunicar-se com dispositivos no modo padrão, em sistemas de 0 a 100 Kbits/s. Porém dispositivos no modo padrão não podem ser incorporados

juntamente com dispositivos no modo rápido, pois não podem operar com uma taxa de transferência mais alta, podendo ocasionar conseqüências imprevisíveis (PHILIPS, 2003).

4.4.3 - Características do modo *High Speed (Hs-Mode)*

Dispositivos em *High Speed Mode (Hs-Mode)* oferecem um salto na velocidade de transferência no barramento I2C, podendo transmitir informações a uma taxa de até 3.4 Mbits/s.

A arbitragem e sincronização do *clock* não são realizados durante a transferência de dados em *Hs-Mode*, o mesmo protocolo do barramento e o formato de dados é mantido com o sistema padrão ou rápido. Dependendo da aplicação, novos dispositivos devem ter uma interface de modo rápido ou *Hs-Mode.* O modo *Hs-Mode* é mais requisitado atualmente, pois pode ser utilizado em um número maior de aplicações (PHILIPS, 2003).

4.4.4 - Transferência de dados no modo *High Speed*

Para uma transferência de bit de 3.4 Mbits/s, as seguintes modificações tem que ser feitas para a especificação I2C convencional:

• Dispositivos *Hs-Mode* mestres tem um “dreno aberto” *PuII Down* e um circuito corrente *PuII-Up* na saida SCLH (linha SCL em modo *High Speed).* Este circuito corrente encurta o tempo do sinal SCLH. Somente a corrente é capacitada a qualquer tempo e somente durante o modo *Hs-Mode.*

• Nenhuma arbitragem ou sincronização de *clock* é realizada durante a transferência *Hs-Mode* em sistemas multi-mestre. O procedimento de arbitragem sempre termina depois de uma transmissão do código mestre anterior em modo rápido ou padrão.

• A única diferença entre dispositivos escravos em *Hs-Mode* e dispositivos escravos no modo padrão ou rápido é que a velocidade à qual eles operam tem saída de “dreno aberto” na

saída SCLH (SCL em *high speed)* e SDAH (SDA em *high speed).* Transistores opcionais *PuII-Down* no pino SCLH podem ser usados para estender o nível baixo do sinal SCLH, embora isto somente é permitido depois do bit de reconhecimento na transferência *Hs-Mode.*

A Figura 23 mostra a configuração física do barramento I2C em um sistema composto somente com dispositivos *Hs-Mode.* Pinos SDA e SCL no dispositivo *Master* são usados somente em sistemas de barramentos com velocidade mista e não são conectados somente em dispositivos *Hs-Mode.* Em tais casos, esses pinos podem ser usados para outras funções.

Resistores opcionais Rs, protegem os estágios de I/O dos dispositivos no barramento I2C de picos de alta voltagem nas linhas de barramento e minimizam interferências.

Resistores *PuII-Up* mantém as linhas SDAH e SCLH em nível alto quando o barramento é livre, garantindo que os sinais sejam mudados de um nível *Low* para *High* dentro de um determinado tempo requisitado.

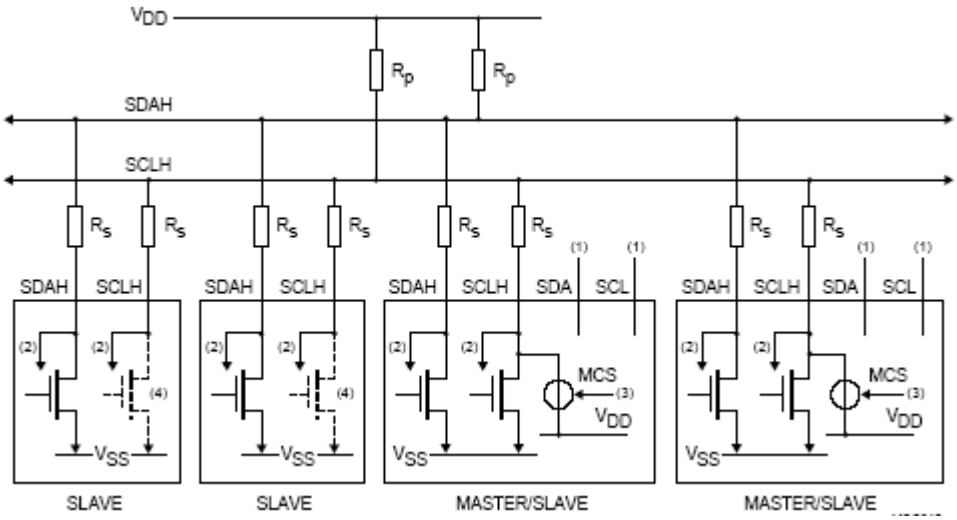

Figura 23 - Configuração do Barramento I2C no modo Hs-Mode. (PHILIPS, 2003)

Dispositivos Hs-Mode são completamente compatíveis, e podem ser conectados a um dispositivo no modo rápido ou padrão no sistema de barramento I2C. Como nenhum código mestre será transmitido em tal configuração, todos os dispositivos Hs-Mode permanecem no modo rápido ou padrão e comunicam-se à velocidade compatível. Os pinos SDAH e SCLH

são usados para conectar ao sistema do barramento no modo rápido ou padrão, permitindo aos pinos SDA e SCL no dispositivo Hs-Mode mestre, serem usados por outros fins.

4.4.5 - Vantagens do I2C

Muitas vantagens podem ser atribuídas ao protocolo I2C. Destaca-se entre elas a:

- organização funcional em blocos, providenciando um simples diagrama esquemático final;
- não há necessidade dos projetistas desenvolverem interfaces, e todos os dispositivos integram as interfaces "*on-chip*", o que aumenta a agilidade no desenvolvimento;
- o endereçamento e protocolo de transferência de dados totalmente definido via *software*;
- a possibilidade de inclusão ou exclusão de dispositivos no barramento sem afetá-lo ou outros dispositivos conectados a este;
- o diagnóstico de falhas extremamente simples. O mau funcionamento é imediatamente detectado;
- o desenvolvimento simplificado do *software* através do uso de bibliotecas e módulos de software reutilizáveis;
- e a facilidade no desenvolvimento de placas de circuito impresso, devido à quantidade de interconexões.

4.5 – Interface de comunicação padrão EIA/RS

Uma interface é o ponto de contato entre dois ambientes diferentes, ou seja, entre os circuitos de um computador e dispositivos externos. Assim, ela é uma espécie de "porta" para o mundo dos computadores, e é muitas vezes chamada porta I/O.

O objetivo principal de qualquer interface é fornecer um meio para transferência de dados. Conseqüentemente, autoproteção e usabilidade são importantes para qualquer interface.

Atualmente há dois tipos de transmissão de dados: paralela e serial. Ao considerar as transmissões paralelas de dados, dois grandes problemas aparecem. O primeiro, são os próprios fios: são necessários pelo menos nove fios, caso seja 8 bits, ou seja, oito para transmissão de dados e um para terra comum do circuito. Ainda assim, mais fios são necessários para controle de fluxo de dados. O outro problema surge da própria natureza das tensões nos bits. Quando ocorre mudança de estado (de zero para um ou vice-versa), ela é muito rápida (na ordem de nanossegundos). Essa rápida mudança é parte essencial no processo de transferência de dados. Mudanças lentas entre zeros e uns não são nem reconhecidas como dados diferentes. À medida que um cabo se estende, as propriedades elétricas (capacitância e indutância) do cabo restringem a abrupção e os dados podem ser corrompidos ou perdidos. Por isso, a velocidade de transmissão em linhas paralelas é problemática para maiores distâncias.

A alternativa óbvia é então, enviar os bits de forma serial, um após o outro, reduzindo os custos da transmissão paralela. Esta economia, no entanto, é compensada por uma deficiência: leva-se oito vezes mais tempo para transmitir os bits que na forma paralela. Este limite é insignificante para muitas aplicações (FRANCO et al., 2003) - os dispositivos seriais são lentos se comparados com a velocidade interna dos processadores.

Em cada dispositivo há um fator limitador de velocidade. Por exemplo, as impressoras estão limitadas pela velocidade das suas cabeças de impressão; os *modems*, pela freqüência das linhas telefônicas e os *drives* de disco pela velocidade de rotação.

O setor de comunicação distribuída conta atualmente com uma quantidade muito grande de produtos - hardware e software - e protocolos usados nas comunicações entre as plataformas de computadores e os dispositivos usados nas aplicações de automação.

Muitos dispositivos utilizados, por exemplo, em aplicações industriais utilizam os padrões EIA RS-232, RS-422 ou RS-485 entre os computadores. Erroneamente tem-se o conceito de que estes padrões definem protocolos de comunicação específicos. Os padrões ANSI/EIA RS-xxx especificam apenas as características elétricas (FRANCO et al., 2003).

Deve-se lembrar, por exemplo, que o padrão RS-485 não é por si só um *fieldbus*, descrito anteriormente, mas pode ser usado para tal.

O padrão RS-232, também referido como interface CCITT V.24, é uma conexão serial encontrada tipicamente em *PCs*. É utilizado para diversos propósitos como conexão de mouse, impressora, modem, bem como instrumentação industrial. Porém este padrão é limitado à conexão ponto-a-ponto entre a porta serial do PC e o dispositivo.

O padrão RS-422 é a conexão serial utilizada tipicamente em computadores *Apple Macintosh*. Este padrão apresenta grande imunidade a ruído quando comparado com RS-232. Isto se deve à transmissão diferencial que utiliza duas linhas para transmissão e duas para recepção.

Por suas características e aplicabilidade, optou-se pelo estudo do padrão EIA RS-485, pelo mesmo ser o padrão de comunicação bidirecional mais utilizado em aplicações industriais (FRANCO et al.,2003). Possui transmissão balanceada e suporta conexões multiponto (*multidrop*), o que permite a criação de redes com até 32 nós e transmissão à distância de até 1200m. Através da inserção de repetidores RS-485 pode-se estender a distância de transmissão de mais 1200m e adicionar outros 32 módulos. Este padrão suporta comunicação *half-duplex*, requer apenas 2 fios para a transmissão e recepção dos dados e possui alta imunidade a ruído.

Tabela 10 - Tabela de parâmetros do padrão EIA RS-485

Parâmetros	EIA RS-485
Taxa de transmissão	10 Mbps (max.)
Distância de transmissão	1200 m (max.)
Processo	Diferencial (balanceado)
Transmissores	32
Receptores	32
Princípio	Half-duplex, multidrop
Tensão de saída do transmissor	1,5 V (min.)
Resistência de carga	60 ohms (min.)
Máxima corrente de saída em curto-circuito	150mA (p/ o terra) - 250mA (p/ -8V ou -12V)
Resistência de saída em alta impedância	120 kohms (on/off)
Sensibilidade do receptor	200 mV
Resistência de entrada do receptor	12 kohms
Tensão máxima de modo comum	+12 V a -7 V

O RS-485 é um padrão de interface por tensão balanceada semelhante ao RS-422-A mas que permite múltiplos transmissores e receptores operando no mesmo barramento, composto por dois fios. Tipicamente, o padrão RS-485 permite um comprimento máximo de 1200m de cabo entre os equipamentos e uma taxa máxima de 10Mbps de transmissão de dados. Uma outra vantagem adicional que o RS-485 apresenta sobre o RS-422-A é que ele é capaz de aceitar tensões de modo comum de +12V a -7V, bem como gerar menor quantidade de ruídos, visto que o tempo de subida do sinal (*time rise*) é três vezes comparado com o RS-422-A.

O padrão RS-485 de interface serial fornece conectividade a vários dispositivos industriais, de manufatura e de aquisição de dados laboratoriais. Muitas aplicações necessitam de mais portas seriais do que as disponíveis nos *PCs* padrões. Em aplicações industriais, muitos dispositivos, CLPs, *data loggers* e módulos E/S *single-point* utilizam RS-485 para longas distâncias e capacidade *multidrop* (FRANCO et al.,2003).

4.5.1 – Componentes e produtos

Existe atualmente uma enorme gama de produtos e componentes disponíveis para aplicações que utilizam RS-485, o que inclui desde placas de interface a contadores e frequencímetros.

Normalmente, encontram-se no mercado placas de interface serial disponíveis para barramentos ISA, PCI e PCMCIA, assíncronas, para comunicação com dispositivos seriais.

Estas interfaces podem ser instaladas e utilizadas como portas COM padrões. Para que haja cem por cento de compatibilidade com as portas COM padrões de PCs, todas as interfaces seriais utilizam UARTs (*Receptor-Transmissor Universal Assíncrono*). A maioria dessas placas acessam os registros da UART por software para controlar a alternância entre a transmissão e recepção requerida na comunicação com dispositivos de dois fios.

Um problema comum nas aplicações industriais refere-se ao *loop* de corrente do terra, que ocorre quando o nível de tensão do terra difere entre os dispositivos conectados. Isto se torna um problema crítico quando dois dispositivos se encontram a grandes distâncias. Em RS-485, este problema resulta em uma tensão de modo-comum produzida pela diferença entre os níveis de terra, ou em ruídos induzidos em ambas as linhas. Para eliminar este problema, portas seriais isoladas são utilizadas, bem como para a proteção dos *PCs* em ambientes industriais agressivos.

Encontram-se também dispositivos conversores RS-232/485. Utilizados nos casos onde o dispositivo possui como padrão o RS-232, como *PCs* que não possuam interface serial RS-485.

Os repetidores RS-485 são utilizados quando se deseja estender a dimensão do barramento, permitindo a conexão de outros 32 dispositivos e mais 1200 m por repetidor. Pode-se conectar até 256 dispositivos em um único barramento RS-485. Tanto conversores

como repetidores controlam automaticamente a direção do barramento RS-485 sem *handshaking* externo de sinal com o servidor.

Os Módulos de entradas analógicas utilizam conversores A/D de alta resolução (geralmente de 16 bits) microprocessados para a obtenção de sinais de sensores como tensão, corrente e outros. Os sinais analógicos são traduzidos para formatos como "unidade de engenharia", complemento de dois, ohms e etc.

O estudo da interface de comunicação padrão EIA/RS mostrou as características e a possível utilização do mesmo na elaboração deste projeto.

5 – VHDL

A linguagem VHDL foi desenvolvida como um padrão de linguagem de programação que descreve a estrutura e o funcionamento de circuitos integrados digitais.

Na tabela 11 é dado o significado da sigla VHDL.

Tabela 11 - Significado da sigla VHDL

Very **H**ight **S**peed **I**ntegrated **C**ircuit (**VHSIC**)
Hardware
Description
Language

VHDL é uma linguagem de descrição de *hardware* que representa entradas e saídas, comportamento e funcionalidade de circuitos. Atualmente faz parte da maioria das ferramentas de projetos eletrônicos (Marques, 2000). VHDL é uma forma de se descrever, através de um programa, o comportamento de um circuito ou componente digital (Marchi, 2002).

A utilização do VHDL possibilita o desenvolvimento metodológico de sistemas complexos, e permite a descrição do sistema em partes, ou seja, a decomposição de um grande sistema em subsistemas indicando como estes subsistemas estão conectados. Ele permite a utilização de formas padrões de programação no desenvolvimento de um sistema digital e, como conseqüência, também permite a simulação do sistema digital antes de sua implementação (Parma, 2003). Além disto, por ser uma linguagem de programação de circuitos digitais, o código VHDL é desenvolvido independentemente do CI a ser utilizado permitindo, desta forma, uma grande flexibilidade na hora da implementação do sistema.

A estrutura de um programa VHDL é baseada em blocos, com a finalidade de expor os recursos que a linguagem oferece para a realização de um projeto qualquer, em suas diversas etapas, demonstrando como são utilizados. Os principais blocos são: a declaração de

entidades (*entity*), a arquitetura (*architecture*), os sub-programas, a declaração de pacotes (*package*) e o corpo do pacote (*package body*).

Uma linguagem de descrição de hardware descreve o que um sistema faz e como. Esta descrição é um modelo do sistema hardware, que será executado em um software chamado simulador. Um sistema descrito em linguagem de hardware pode ser implementado em um dispositivo programável (FPGA), permitindo assim o uso em campo do seu sistema, tendo a grande vantagem da alteração do código a qualquer momento.

5.1 – Breve Histórico sobre VHDL

A idéia da criação de uma linguagem de descrição de *hardware* partiu do Departamento de Defesa dos Estados Unidos da América. As forças armadas americanas compravam grande quantidade de placas de circuitos impressos, sendo que muitas delas compostas de circuitos integrados de aplicação específica (ASIC).

Como era comum, empresas da área de eletrônica, mudarem de área ou trocar de ramo, necessitava-se garantir a reposição das peças durante a vida útil das placas, com isso o Departamento de Defesa Americano iniciou o desenvolvimento de uma linguagem padrão de descrição de *hardware.*

A linha cronológica da história do surgimento do VHDL é apresentada, a seguir, na tabela 12.

Tabela 12 - Cronologia do surgimento da Linguagem VHDL

Ano	**Ocorrência**
1968	Foram desenvolvidas as primeiras linguagens de descrição de *hardware.*
1970	Tinham-se inúmeras linguagens com sintaxe e semântica incompatíveis
1973	Surge o primeiro esforço de padronização da linguagem, comandado pelo projeto CONLAN

	(*CONsensus LANguage*), cujo o objetivo principal era: definir formalmente uma linguagem de multi-nível com sintaxe única e semântica igual. Paralelamente, iniciou-se outro projeto financiado pelo Departamento de Defesa Americano cujo o objetivo era criar uma linguagem de programação.
1983	Em janeiro de 1983, foi publicado o relatório final do projeto CONLAN. Neste mesmo ano, também foi publicado o relatório final do projeto Departamento de Defesa Americano, que deu origem a linguagem ADA. Em março de 83, o Departamento de Defesa Americano começou o programa VHSIC, afim de melhorar a tecnologia de concepção, engenharia e construção nos EUA. Participaram deste projeto a IBM, Intermetrics e Texas Instruments.
1986	A Intermetrics desenvolveu um compilador e um simulador. Além disso, foi criado um grupo de padronização da IEEE para VHDL.
1988	Primeiros *softwares* são comercializados.
1991	Recomeçou um novo processo de padronização, cujo objetivo era a coleta e análise de requisitos, definição dos objetivos e a especificação das modificações à linguagem.
1993	Um novo padrão é publicado, chamado VHDL-93, padronizado IEEE Std 1164-1993.
1997	Em dezembro de 97 foi publicado o manual de referência da linguagem VHDL.

5.2 – Vantagens na utilização do VHDL

A descrição de um sistema em VHDL apresenta inúmeras vantagens, tais como:

- Intercâmbio de projetos entre grupos de pesquisa sem a necessidade de alteração;
- permite ao projetista considerar no seu projeto os *delay´s* comuns aos circuitos digitais;
- a linguagem independe da tecnologia atual, ou seja, você pode desenvolver um sistema hoje e implementá-lo depois;
- os projetos são fáceis de serem modificados; o custo de produção de um circuito dedicado é elevado, enquanto que usando VHDL e Dispositivos Programáveis, isto passa a ser muito menor;

- reduz consideravelmente o tempo de projeto e implementação.

6 – RtrASSoc51

O RtrASSoc51 é um SOPC e sua estrutura é descrita na Figura 24, onde o Processador Superescalar Embarcado (PSE) terá como base a estrutura dos microcontroladores da família 8051 acrescido de um pipeline em três níveis configurando assim uma estrutura superescalar. Além do pipeline superescalar e, em função da aplicação, poderão ser ainda configurados no RtrASSoc51 os elementos normalmente presentes nos microcontroladores 8051 que são os temporizadores, as memórias, as portas paralelas, uma porta serial, os mecanismos de interrupção, além do controle para execução das instruções presentes no programa de aplicação.

O Sistema Operacional Embarcado (SOE), terá como função principal gerenciar internamente os recursos do RtrASSoc51, recursos como memórias, comunicação com elementos externos (hospedeiro ou memórias SRAM ou memórias flash), mas principalmente gerenciar o processo de reconfiguração parcial utilizando o espaço das Rotinas Reconfiguráveis (RR).

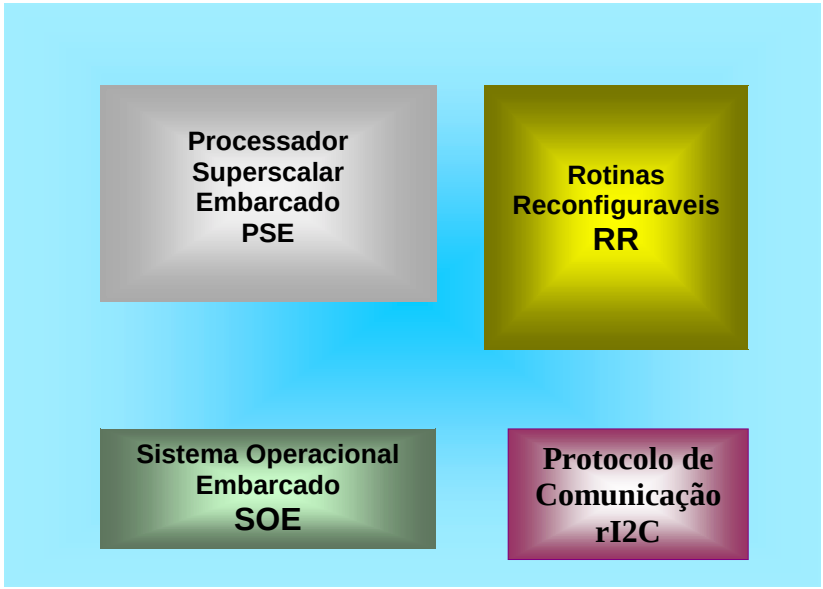

Figura 24 - A Estrutura do RtrASSoc51

Esses elementos, que fazem parte de projetos distintos de alunos do Mestrado de Ciência da Computação – Univem (CARDIM, 2004; COSTA, 2004; FORNARI, 2004; ZANGUETTIN, 2004).

6.1 – O rI2C (*RECONFIGURABLE INTER INTEGRATED CIRCUITS*) NA PLATAFORMA RTRASSOC51.

O protocolo rI2C é a implementação do protocolo de comunicação I2C, desenvolvido pela Philips, acrescido de técnicas de reconfiguração, mais especificamente reconfiguração das diferentes taxas de comunicação prevista no protocolo original.

O rI2C foi implementado, tendo como base a plataforma RtrASSoc51 como um *CORE* de comunicação e foi implementado em VHDL para FPGAs da *Xilinx*.

Em se tratando de um dispositivo SOPC, é possível reconfigurar o *CORE* rI2C no RtrASSoc51 em função das necessidades da aplicação em um determinado momento.

Considerando uma aplicação simples, contendo vários RtrASSoc51 interligados, conforme descrito na Figura 25, pode ocorrer que um evento force o sistema a se reconfigurar para se comunicar a uma taxa de comunicação mais veloz. É possível, portanto reconfigurar cada elemento RtrASSoc51 para que todos operem agora na nova taxa.

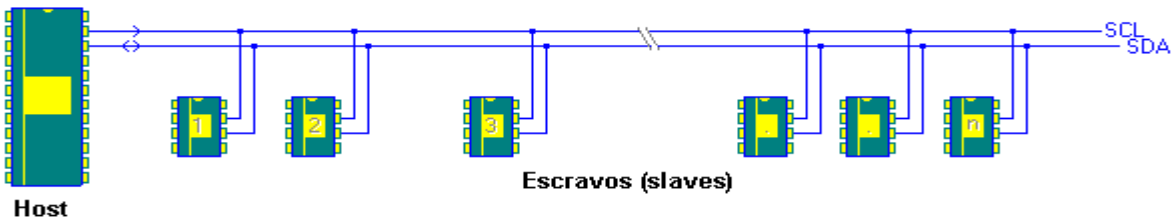

Figura 25 - Comunicação de dispositivos RtrASSoc51, usando protocolo rI2C

Inicialmente, o principal objetivo para a escolha do protocolo I2C, foi a facilidade na integração entre componentes que possam trabalhar diretamente com seus respectivos sinais, simplificando desta forma a comunicação com outros componentes da mesma família. E a

proposta do rI2C só foi possível pela possibilidade das diferentes taxas de transmissão no barramento I2C original, conforme Tabela 13.

Tabela 13 - Taxas de transmissão do I2C.

BARRAMENTO	Taxa (bits/s)
I2C Padrão	100K
I2C Modo Rápido	400K
I2C High Speed (Hs-Mode)	3,4M

6.2 – Características do rI2C

As características descritas a seguir seguem os mesmos requisitos do protocolo original I2C da Philips acrescidos do módulo de reconfiguração.

Na Figura 26 é apresentada a estrutura RTL (*Register Transfer Level*) da interface rI2C, juntamente com os módulos do RtrASSoc51.

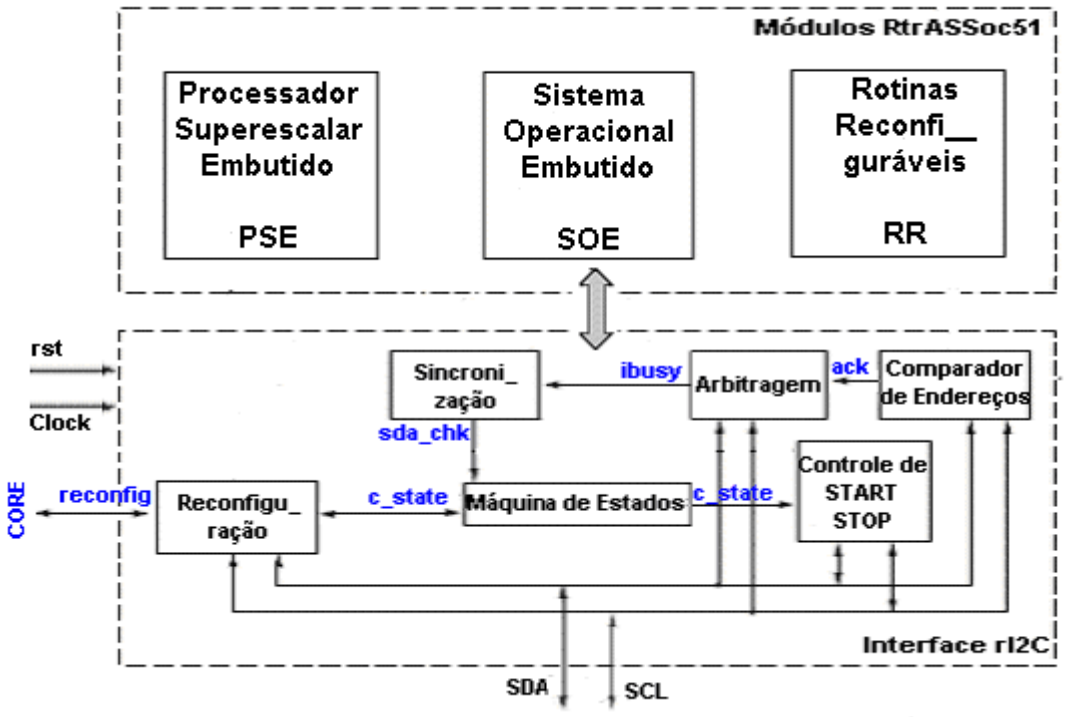

Figura 26: Diagrama RTL do rI2C

O bloco de Sincronização tem a função de sincronizar os sinais de clock de dois ou mais dispositivos.

No processo de transmissão de dados do rI2C surge uma situação única, a qual é definida como condição de partida e parada, desempenhada pelo bloco de Controle START/STOP. As condições de partida e parada são sempre geradas pelo MASTER.

O bloco Arbitragem administra a disponibilidade do barramento. O barramento é considerado ocupado após a condição de partida e livre um certo período de tempo após a condição de parada. Controla também o processo que assegura que um ou mais mestres tentem, de um modo simultâneo, adquirir o controle do barramento, sendo que apenas um conseguirá o controle naquele momento.

O bloco Comparador de endereços tem a função de verificar o endereçamento de uma comunicação iniciada por um mestre a um escravo, sendo este endereço um identificador único de cada dispositivo.

Todo processo de transferência de dados entre um mestre e um escravo no barramento rI2C deve conter ao menos um estado previsto, sendo administrado pelo bloco Máquina de Estados.

O bloco Reconfiguração tem função de iniciar a chamada ao SOE que acionará o CORE de reconfiguração das taxas de transmissão no barramento rI2C.

6.2.1 – O protocolo rI2C

Algumas características do rI2C são apresentadas a seguir, complementadas com os dados da Tabela 14:

- Duas linhas de barras seriais, as quais consistem de uma linha de *clock* (SCL) e uma linha de dados (SDA).

- Transferência de dados bidirecional, ou seja, a mesma linha transmite e recebe dados.

- Barra *MULTI-MASTER* real, ou seja, mais de um CI capaz de controlar a barra, pode ser conectado a ela. Cada *MASTER* gera seu próprio *clock*.

- Cada CI compatível com o rI2C possui um endereço único (7-*bits*) e pode operar como transmissor ou receptor (*MASTER* ou *SLAVE*).

Tabela 14: Terminologia básica do rI2C

Termo	**Descrição**
Transmissor	Dispositivo que envia dados ao barramento.
Receptor	Dispositivo que recebe dados do barramento.
Mestre	Dispositivo que inicia uma transferência, gera o sinal de clock e termina uma transferência.
Escravo	Dispositivo que é endereçado pelo mestre.
Multi-mestre	O controle do barramento pode ser solicitado por um ou mais dispositivos (mestres) ao mesmo tempo, sem que ocorra mensagem corrompida.
Arbitragem	Processo que assegura que um ou mais mestres tentem de um modo simultâneo adquirir o controle do barramento, sendo que apenas um conseguirá o controle naquele momento.
Sincronização	Processo para sincronizar os sinais de clock de dois ou mais dispositivos.

- CIs compatíveis com rI2C podem atuar como *MASTERs* ou *SLAVEs* durante a transferência de dados.

- O procedimento de decisão previne a perda ou a alteração dos dados quando os *MASTERs* estão competindo pelo uso do barramento.

- O primeiro byte de uma transferência contém 7 bits de endereço *SLAVE*.

- Cada byte transferido é reconhecido pelo *SLAVE*.

- A implementação de uma interface *SLAVE* é muito simplificada.

- O protocolo é padronizado.

- Cada dispositivo conectado no barramento rI2C deve ser identificado com um único endereço: cada dispositivo conectado tem um endereço único no barramento, não podendo haver outro com o mesmo endereço. O endereço deve ter 7 bits de tamanho.

- Só pode existir 1 mestre e 1 escravo de maneira simultânea: quando um mestre deseja informação de um escravo, este envia pelo barramento o endereço do escravo.

Todos os dispositivos conectados ao barramento ficam “escutando” os endereços que o mestre envia, porém só responde o escravo mencionado. Neste momento inicia-se o protocolo de comunicação.

- O mestre pode trabalhar como mestre-transmissor ou mestre-receptor. As operações que o mestre poderá solicitar a um escravo serão de leitura e/ou escrita.

- Os dados seriais que se enviam/transmitem são de 8 bits: todos os dados compartilhados entre o mestre e o escravo tem 8 bits de tamanho, sendo a exceção do endereço do escravo, que como citado anteriormente será de 7 bits. Estes dados seriais devem ser enviados pela linha SDA.

- Módulo de reconfiguração entre as taxas: 100 Kbits/s (modo padrão), 400 Kbits/s (modo rápido), e 3.4 Mbits/s (modo ultra-rápido).

Tanto a linha SCL quanto na SDA são bidirecionais e, o funcionamento do protocolo rI2C, independentemente dos distintos modos que existem para enviar/transmitir dados, divide-se em diferentes estados:

- Estado *NO BUSY* ou *IDLE* -> disponível
- Estado *START* -> início
- Estado *STOP* -> parada
- Estado *ACK* -> recepção correta
- Estado *READ* -> leitura
- Estado *WRITE* -> escrita
- Estado RECONFIG -> reconfiguração

6.3 – Definição dos estados

Todo o processo de transferência de dados entre um mestre e um escravo, deve conter pelo menos os seguintes estados:

- *NO BUSY / IDLE, START, STOP, ACK, READ, WRITE* e RECONFIG.

Em função do tipo de comunicação que se estabeleça, pode-se ter a ocorrência de distintas combinações dos estados acima citados.

A Figura 27 demonstra um diagrama de estados do protocolo rI2C, e a Figura 28 apresenta diagrama de fluxo de uma comunicação rI2C.

Os estados de *STOP* e *START* são sempre gerados pelo mestre, enquanto que os demais podem ser gerados pelo emissor e/ou pelo receptor em função do tipo de comunicação estabelecida.

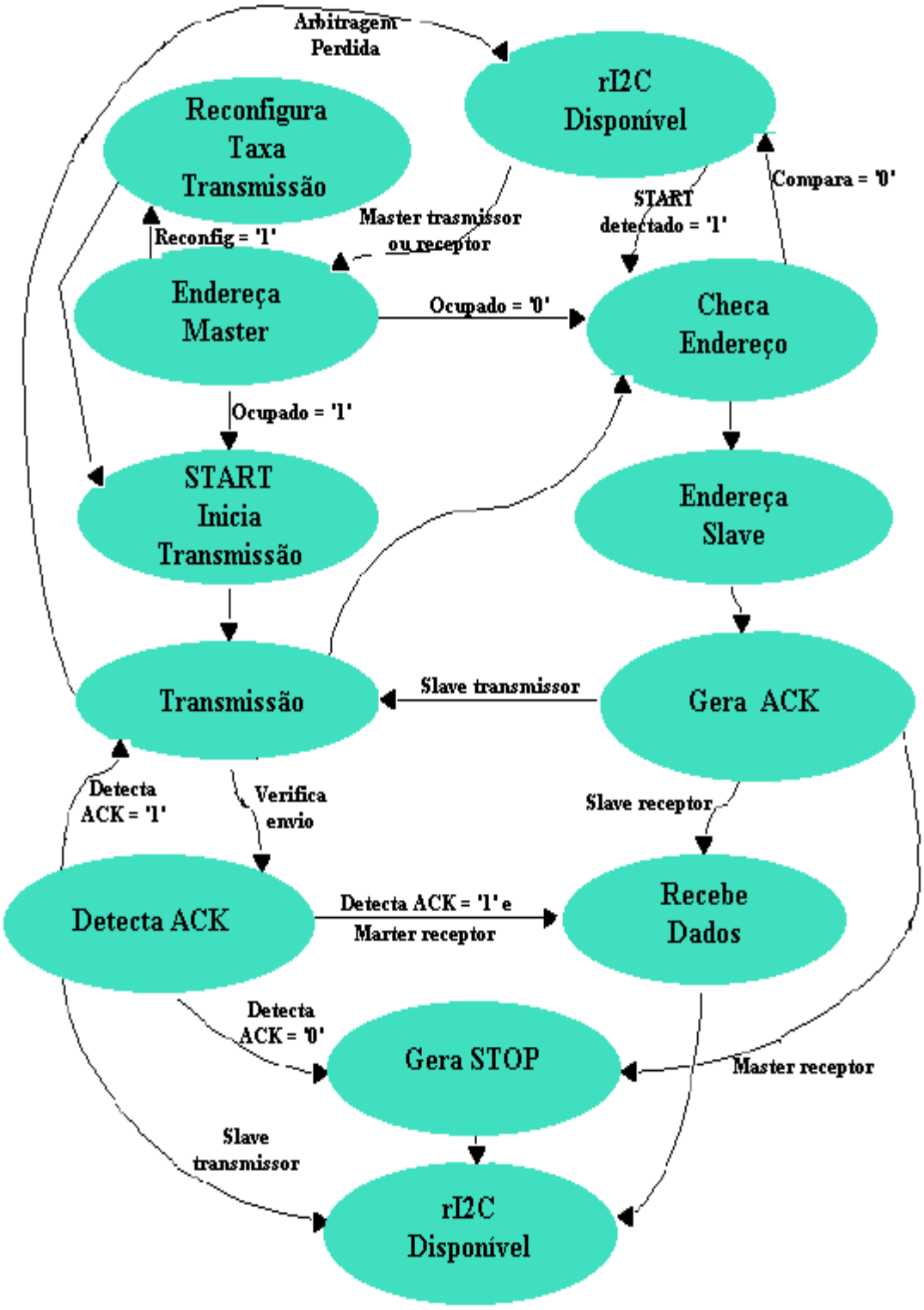

Figura 27: Diagrama de estados do barramento rI2C

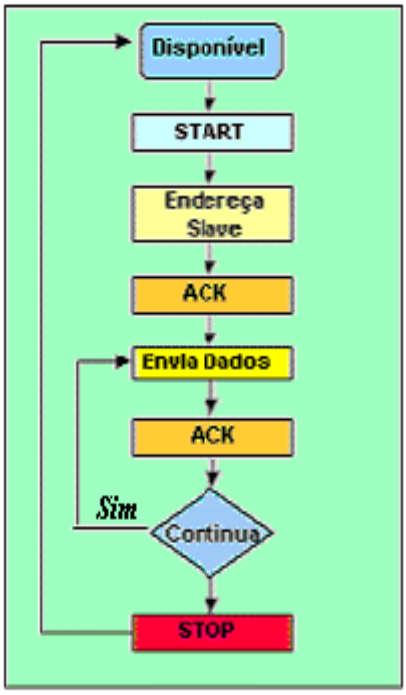

Figura 28: Diagrama de fluxo de uma comunicação rI2C

Estado Disponível - *NO BUSY* ou *IDLE*: é o estado disponível ou situação inicial do barramento (Figura 29). Neste caso as linhas estão em nível alto. Um mestre, para iniciar uma comunicação, deve verificar se o barramento está na condição de não ocupado, caso não esteja deverá aguardar até que esteja disponível.

Figura 29: Estado de NO BUSY / IDLE

Condição de Início – *START*: uma vez verificado que o barramento está em situação de *NO BUSY*, o mestre pode iniciar uma comunicação. Para ele é obrigatório que sempre se comece por uma situação de *START* (Figura 30).

O conjunto de dispositivos conectados ao barramento rI2C sabe que o mestre deseja começar uma comunicação mediante da transição do nível alto para o baixo do sinal SDA.

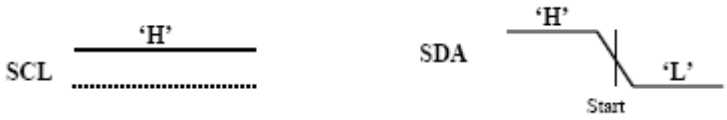

Figura 30: Estado de *START*

A Figura 31 apresenta a simulação feita na ferramenta *Xilinx Foundation*, mostrando a condição de início (*START*) na comunicação rI2C, representada pelas linhas SDA, SCL, sSDA e sSCL.

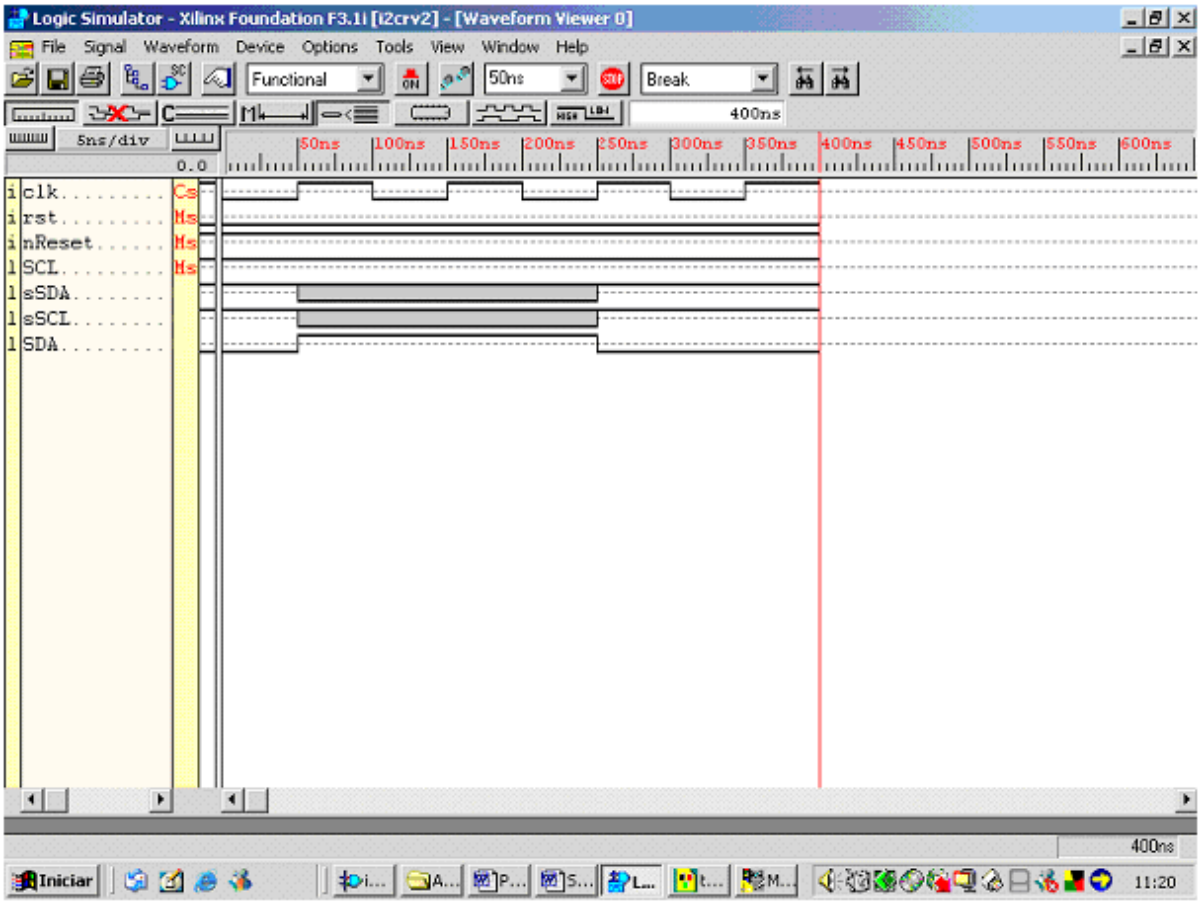

Figura 31: Simulação da condição de início (*START*)

Condição de Parada – *STOP*: quando o mestre decide concluir o processo de comunicação, o SDA deverá estar em transição de baixo para alto e o SCL estar em nível alto.

Uma vez alcançada esta premissa, o barramento fica em situação de *NO BUSY*, podendo ocorrer nova ocorrência de comunicação.

Figura 32: Estado de *STOP*

Portanto, segundo o exemplo, os estados de *START* e *STOP* definem o início e o fim de um processo de comunicação entre um mestre e um escravo. Isto se pode ver na Figura 33.

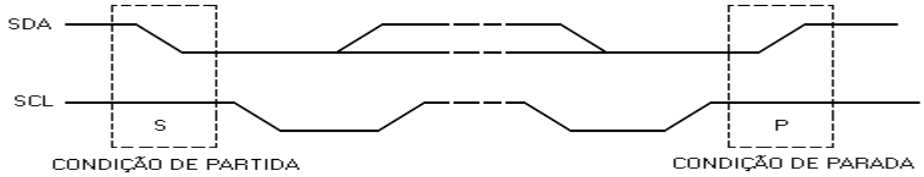

Figura 33: Condição de partida (*START*) e parada (*STOP*)

A Figura 34 apresenta a simulação feita na ferramenta *Xilinx Foundation*, mostrando a condição de parada (STOP) na comunicação rI2C, representada pelas linhas SDA, SCL, sSDA e sSCL.

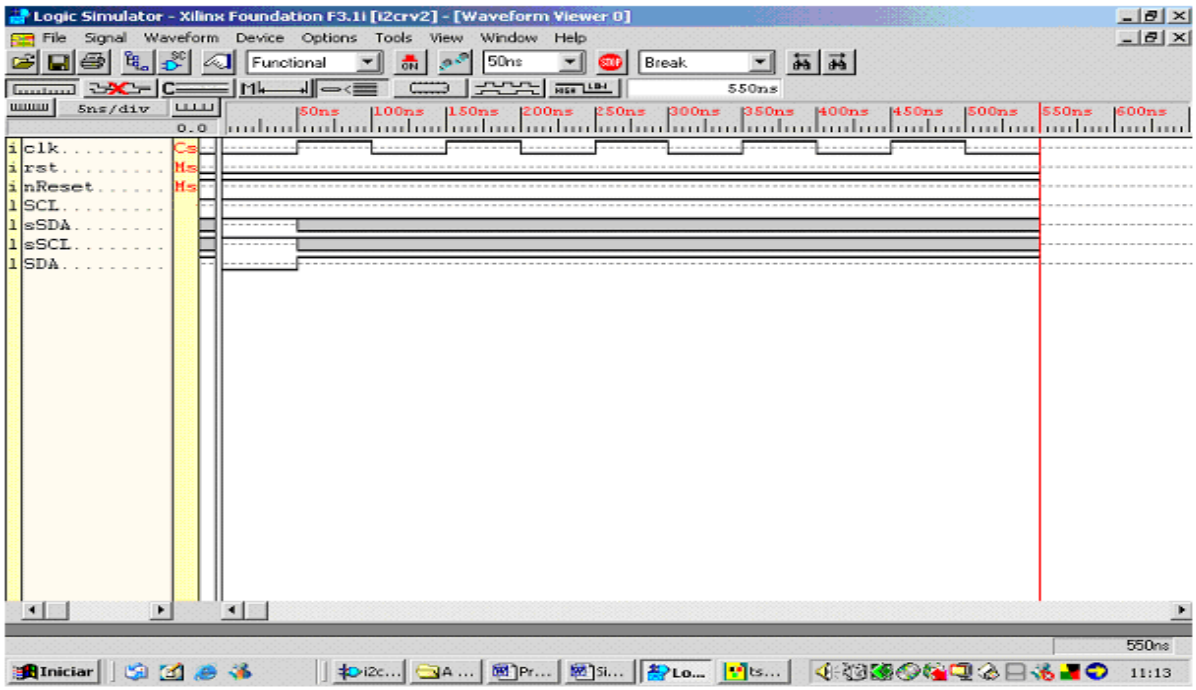

Figura 34: Simulação da condição de parada (*STOP*)

Condição de Reconfiguração – RECONFIG: esta situação ocorre quando se deseja iniciar a reconfiguração das taxas de transmissão no barramento rI2C, onde, a partir de uma chamada ao SOE (Sistema Operacional Embarcado), o mesmo encaminhará a solicitação ao CORE de reconfiguração das taxas de comunicação.

Este estado só ocorrerá depois de um estado de *START* como mostrado na Figura 35.

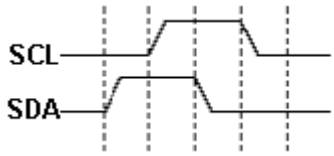

Figura 35: Condição de chamada do *CORE* para reconfiguração das taxas de transmissão

A Figura 36 apresenta a simulação feita na ferramenta *Xilinx Foundation*, mostrando a condição de chamada de reconfiguração no rI2C – Reconfig, representada pelas linhas SDA, SCL e Reconfig.

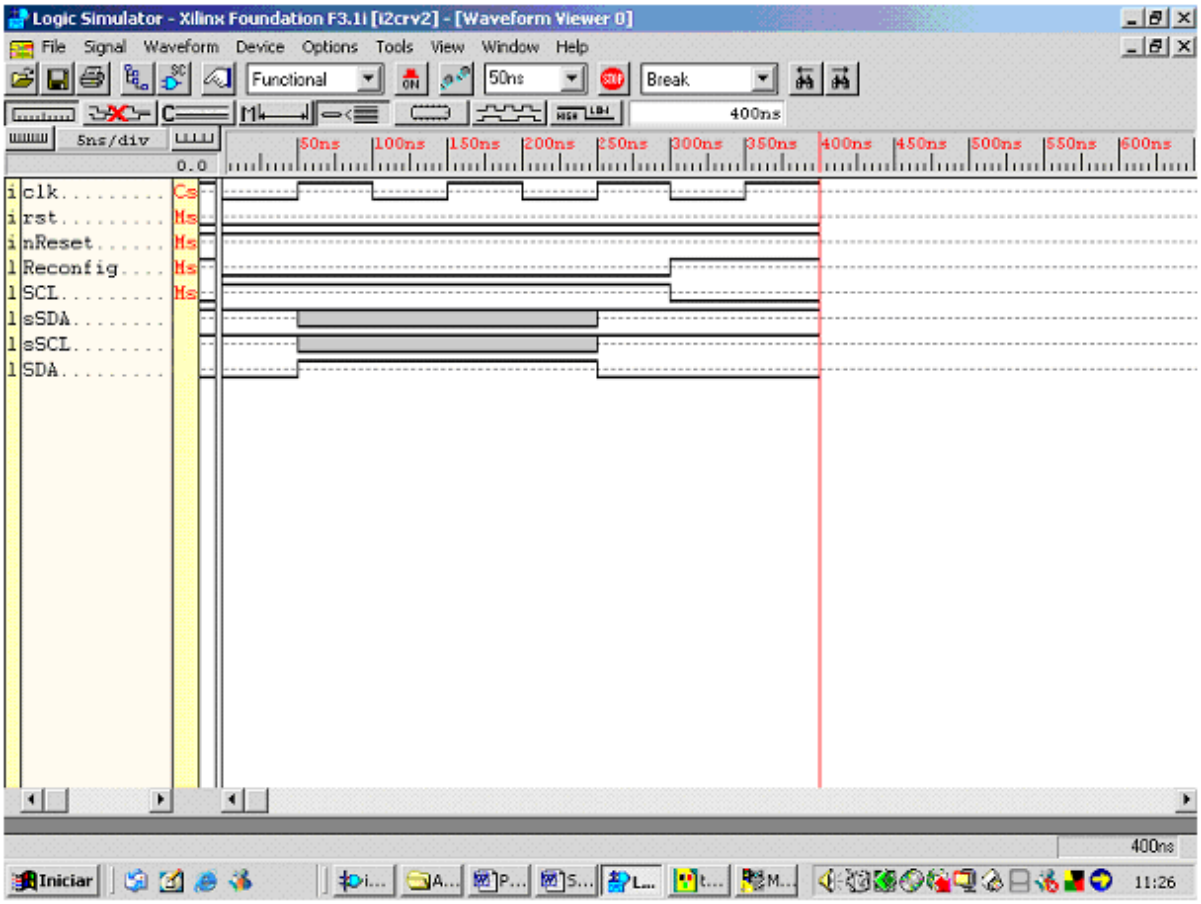

Figura 36: Simulação da chamada de reconfiguração (Reconfig)

***ACK* (*Acknowledge*):** este estado deve ser gerado sempre que se transmitir ou receber um dado, bem como um endereço. Esta situação será gerada pelo receptor do dado (é importante constatar que o receptor pode ser o mestre ou o escravo) quando o 8º. *bit* que é transmitido via serial é recebido corretamente. Neste caso o transmissor colocará a linha SDA em nível alto e o receptor colocará em nível baixo a linha SDA na subida de borda de SCL.

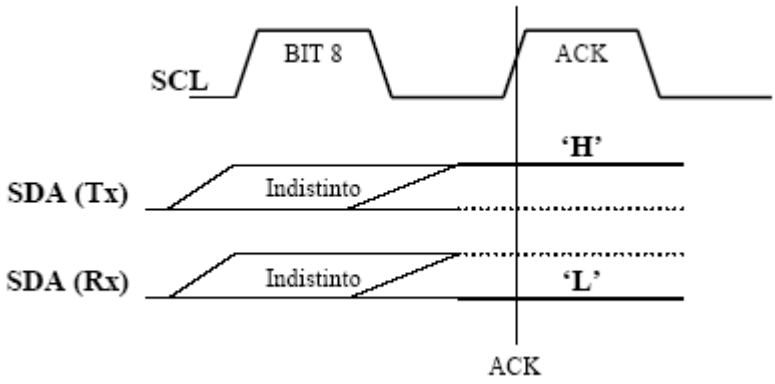

Figura 37: Geração do estado *ACK*

Endereçamento: esta situação ocorre quando se deseja iniciar uma comunicação com o escravo, portanto, é um estado executado pelo mestre. É obrigatório estabelecer o endereço do escravo através do barramento rI2C. Lembrando que o endereço de cada escravo é como seu identificador, não podendo existir dois dispositivos com o mesmo endereço. Este processo só se realizará uma vez em processo de leitura e/ou escrita (*read/write*) de vários bytes.

O endereço do escravo será um número de 7 *bits*, sendo o 8º. *bit* o que completa o byte de endereço, identificando o processo a ser realizado. Assim, caso se deseja fazer uma operação de leitura, este bit será um "1", e se caso seja uma escrita será um "0", como indicado na Figura 38.

Este estado só ocorrerá depois de um estado de START como mostrado na Figura 30.

Uma vez finalizado o processo de endereçamento do escravo, deve-se gerar um estado de ACK para comprovar que o escravo mencionado tenha recebido corretamente seu dado.

É importante observar que a transmissão de cada um destes bits através da linha de dados (SDA) deve ser ativada em cada borda de subida da linha de clock (SCL).

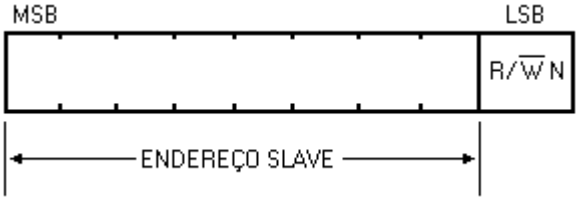

Figura 38: Configuração de byte de endereço

Dados: este estado de funcionamento é totalmente similar ao citado anteriormente, com exceção de que nesta situação o dado a ser transmitido é de 8 *bits*. Os dados serão transmitidos começando pelo bit de maior peso, como se pode observar na Figura 39.

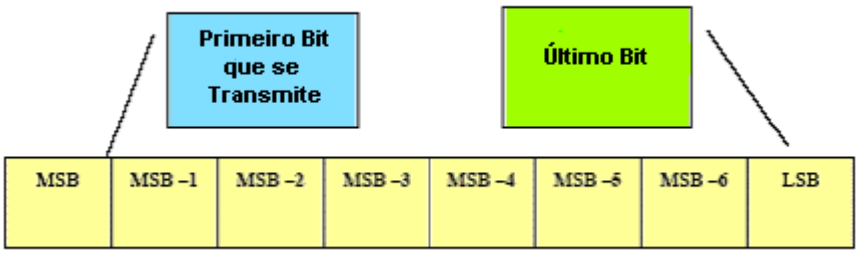

Figura 39: Esquema do byte de dados

6.4 - Transferência de dados

Pode-se definir várias formas de transmissão de dados entre um escravo em direção a um mestre. Porém a norma estabelece algumas exigências ao iniciar uma comunicação, por exemplo, sempre ao iniciar uma comunicação o mestre deve iniciá-la quando o barramento esteja em situação de *NO BUSY* (*IDLE*). Assim estando, deve-se iniciar com a ativação do estado *START* para depois endereçar o dispositivo. Para isso será emitido um dado na linha de dados (SDA) onde os 7 bits de maior importância (peso) serão o endereço do escravo e o último indicará o tipo de operação a ser realizada.

Diante dos modos de realizar uma transferência, cabe levar em conta as seguintes observações:

- Um processo de comunicação começa com uma situação de *START* e finaliza com uma situação de *STOP*. Isto não implica que só se pode transferir um dado, mas sim que se pode transferir tantos dados quanto desejados, sem precisar enviar o dado seguinte depois da geração de um *ACK*.
- Caso deseja-se trocar o sentido de transferência de dados, basta gerar um novo *START* sem a necessidade de finalizar o processo com uma situação de *STOP*.

A Figura 40 demonstra uma comunicação através do protocolo rI2C.

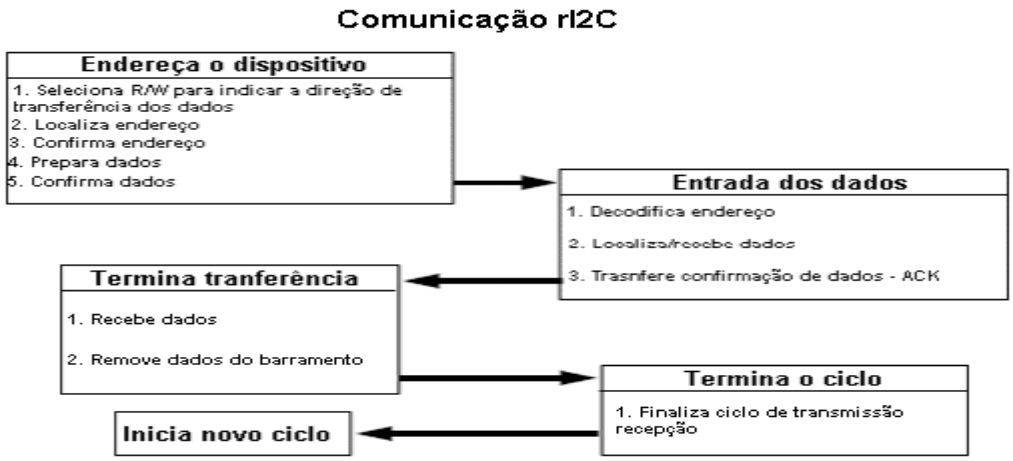

Figura 40: Esquema de Comunicação do Protocolo rI2C

6.4.1 – O componente rI2C na plataforma RtrASSoc51

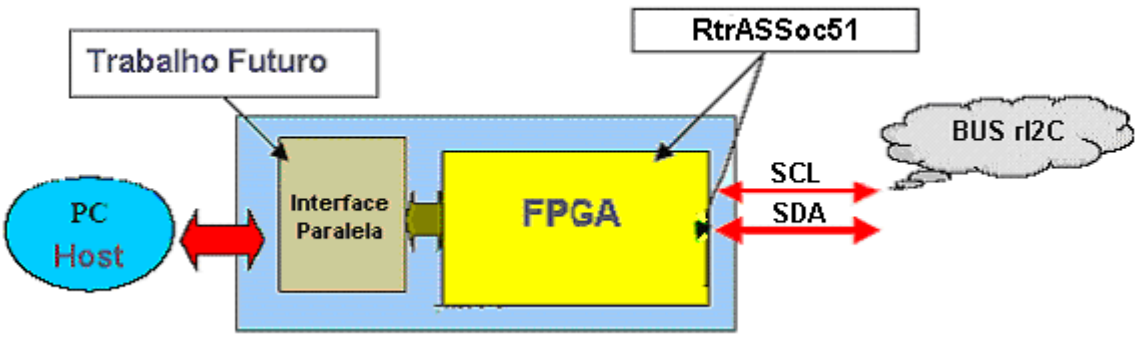

Figura 41: Plataforma RtrASSoc51 e barramento rI2C

Os sinais definidos são usados para a comunicação no dispositivo. Sua principal função é a geração dos comandos de controle que o rI2C necessita para poder gerar os sinais SCL e SDA corretamente para a comunicação dos dispositivos RtrASSoc51.

As portas da entidade são definidas a seguir, na tabela 15:

Tabela 15: Portas da entidade da implementação rI2C

Nome	E/S	Tipo	Descrição
CLK	Entrada	std_logic	Clock Principal
RST	Entrada	std_logic	Reset síncrono ativado em alto - High
NReset	Entrada	std_logic	Reset Assíncrono ativado em baixo - Low
START	Entrada	std_logic	Ativação do bit de início
STOP	Entrada	std_logic	Ativação do bit de parada
DIN	Entrada	std_logic_vector	Dado a trasnmitir
CMD_ACK	Saída	std_logic	Reconhecimento do comando
DOUT	Saída	std_logic-vector	Dado a trasnmitir
SCL	Entrada	std_logic	Linha de Clock
SDA	Entrada	std_logic	Linha de Dados
sSCL	Entrada	std_logic	Sincroniza entrada de SCL
sSDA	Entrada	std_logic	Sincroniza entrada de SDA
RECONFIG	Saída	std_logic	Chamada ao CORE de reconfiguração de taxas de transmissão

As três primeiras portas são sinais de controle genéricos. Assim, o sinal CLK é o sinal de *clock* mestre da implementação empregada para sincronizar a lógica interna do dispositivo. O sinal RST é um sinal de *reset* síncrono, e o sinal nReset é um sinal de *reset* assíncrono ativado em nível baixo.

O seguinte grupo de sinais denominam-se sinais de comandos e especificam o formato do esquema rI2C na transferência a ser realizada, já fazendo a transmissão e recepção de um novo byte. Neste caso o conceito de byte se aplica tanto ao endereço do escravo que se pretende acessar como aos dados recebidos ou transmitidos.

- *START* – quando está em 1, indica que se deve gerar um bit de *START* para começar a transferência seguinte de um novo byte.
- *STOP* – quando este sinal está em 1, indica que se deve gerar um bit de *STOP* para concluir a transferência de um byte.

- *A* saída *CMD_ACK* é um sinal de reconhecimento de comando. Uma vez realizada a transferência solicitada, esta saída ativa-se durante um ciclo de CLK.
- Através da porta de entrada *DIN*, proporciona-se um novo byte a transmitir. De forma semelhante, através da porta de saída *DOUT*, proporciona-se um novo byte recebido.

Finalmente, os últimos sinais que aparecem na Tabela 18, são utilizados para controlar os terminais do SCL e SDA de conexão ao barramento rI2C, e executar a chamada para a reconfiguração das taxas de transmissão.

A Figura 42 apresenta a simulação feita na ferramenta *Xilinx Foundation*, mostrando a condição de transferência de dados no rI2C, considerando as condições de START e STOP e o sinal de ACK.

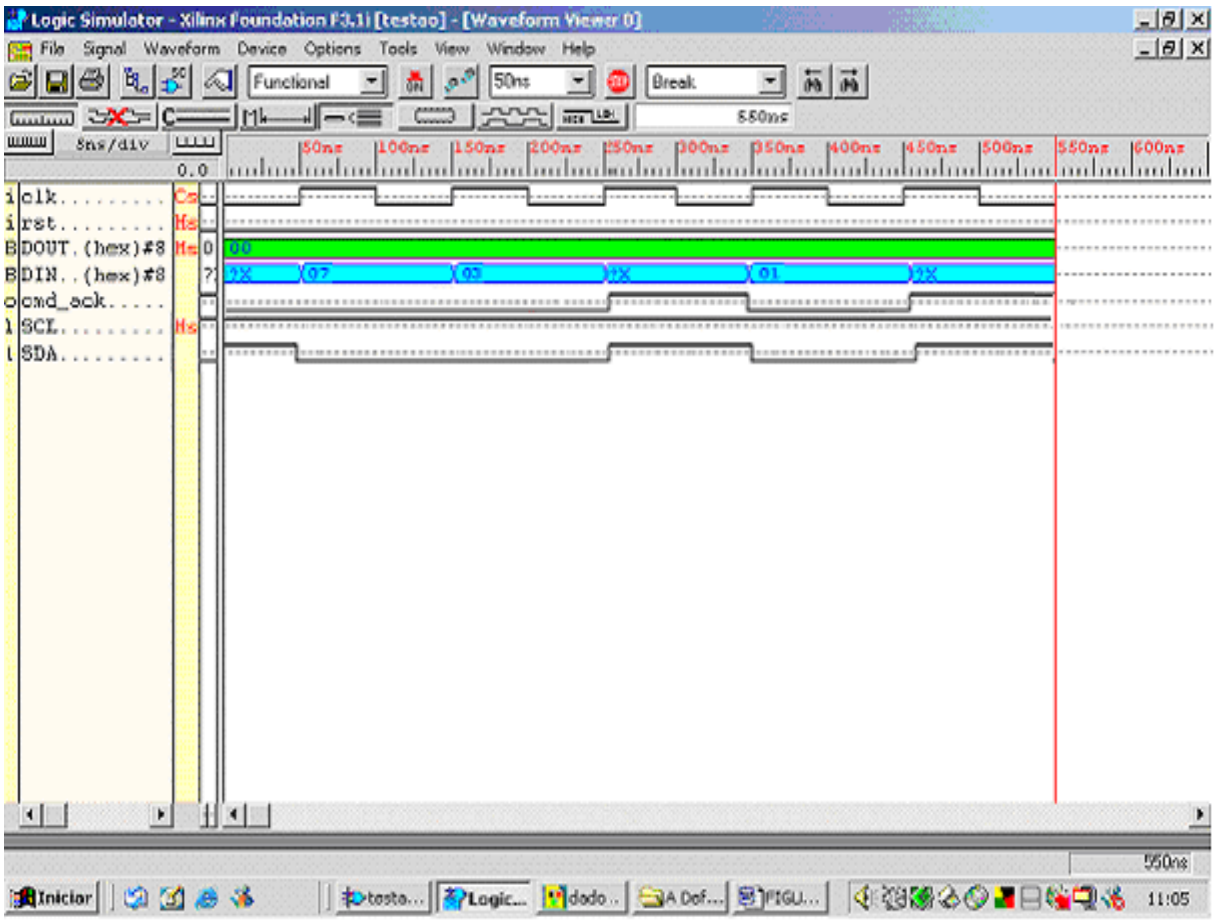

Figura 42: Simulação da transferência de dados no rI2C.

O seguinte diagrama da Figura 43, ilustra o funcionamento do dispositivo com o rI2C.

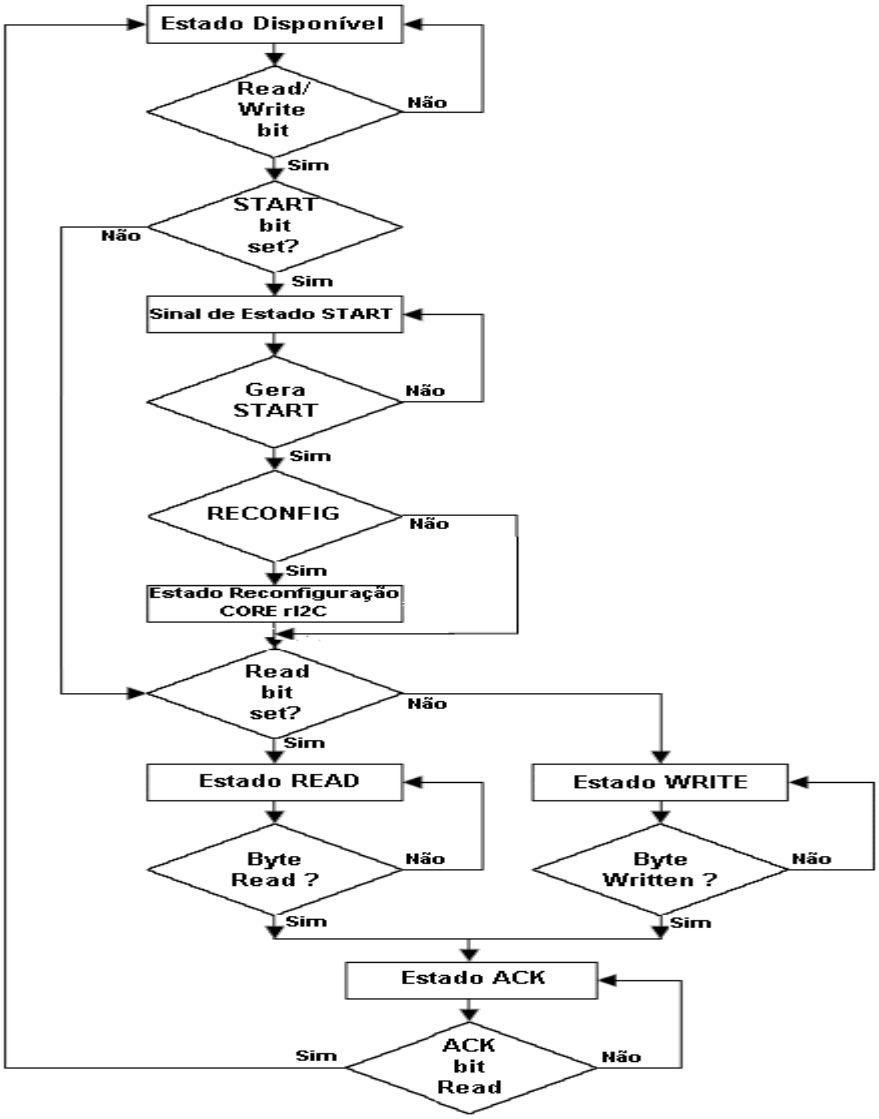

Figura 43: Diagrama de fluxo do funcionamento do módulo rI2C.

7 - CONCLUSÃO

Conforme proposta inicial foi apresentada uma série de caracterizações de dispositivos microcontroladores, dispositivos FPGAs, em particular as FPGAs da Xilinx, e finalmente os mecanismos de comunicação sendo utilizados nesses sistemas, dentre eles o AMBA (Advanced Microcontroller Bus Architecture), Field Bus (Foundation Fieldbus), CAN (Controller Area Network), e o protocolo I2C.

Em seguida foram apresentadas características do RtrASSoc51, um System on Programmable Chip - SOPC, seus diferentes componentes, que tem como base a arquitetura dos microcontroladores da família 8051.

Finalmente foram apresentadas as características do rI2C, um protocolo reconfigurável cuja base é a do protocolo original I2C acrescido dos elementos de reconfiguração, mais especificamente as diferentes taxas de comunicação (100Kbits, 400Kbits, 3.4Mbits), que serão incorporados à plataforma RtrASSoc51 na execução das aplicações que necessitem de vários RtrASSoc51 interconectados. Conforme implementação, a aplicação solicita a reconfiguração da taxa de comunicação e cada elemento RtrASSoc51 individualmente se reconfigura para gerar a taxa de comunicação solicitada. Além dos códigos em VHDL, foram apresentados os resultados da simulação, validando a processo do protocolo rI2C.

Nos relatórios de implementação do protocolo, utilizando a ferramenta Foundation Series da Xilinx em uma FPGA XC4000XL (4002XLPC84), foi observado que a taxa de ocupação para IOB foi de 57% e CLB foi de 73%, e que o tempo máximo de execução foi de 8,354ns.

Como trabalhos futuros, podemos destacar a implementação completa do módulo de reconfiguração dinâmica das taxas de transmissão de dados no protocolo I2C (100 Kbits/s,

400 Kbits/s e 3,4 Mbits/s), e da implementação de uma interface de comunicação com um Host.

APÊNDICE – Código VHDL

Implementação em VHDL do controle de conexão do RtrASSoc51 utilizando o protocolo de comunicação I2C, e a chamada para reconfiguração – módulo rI2C.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity i2cRV is
	port (
		clk	: in std_logic;		-- clock principal
		rst	: in std_logic;		-- reset sinc ativado em high
		nReset	: in std_logic;		-- reset assinc ativado em low
		habilita : in std_logic; -- habilita sinal
		clk_cnt : in unsigned(15 downto 0);  -- clock
		cmd	: in std_logic_vector(3 downto 0);
		cmd_ack : out std_logic;	-- retorna que completou comando
		ocupado : out std_logic;	-- Barramento I2C Ocupado
		al	: out std_logic;	-- Perda da arbitragem
		din	: in std_logic;
		dout : out std_logic;

		-- Linhas do I2C
		scl_e	: in std_logic;		 -- Linha de entrada do clk I2C
		scl_s	: out std_logic;	-- Linha se saida do clock I2C
		scl_oen : out std_logic;	-- Linha de saida clock, habilita baixo
		sda_e	: in std_logic;		-- Linha de entrada de dados
		sda_s	: out std_logic;	-- Linha de saida de dadost
		sda_oen : out std_logic		-- Linha de saida de dados, habilita baixo
	);
end entity i2cRV;

architecture structural of i2cRV is
	constant I2C_CMD_NOP	: std_logic_vector(3 downto 0) := "0000";
	constant I2C_CMD_START	: std_logic_vector(3 downto 0) := "0001";
	constant I2C_CMD_STOP	: std_logic_vector(3 downto 0) := "0010";
	constant I2C_CMD_READ	: std_logic_vector(3 downto 0) := "0100";
	constant I2C_CMD_WRITE	: std_logic_vector(3 downto 0) := "1000";
	type states is (idle, start_a, start_b, start_c, start_d, start_e,
		stop_a, stop_b, stop_c, stop_d, rd_a, rd_b, rd_c, rd_d, wr_a, wr_b, wr_c, wr_d);
	signal c_state : states;
	signal iscl_oen, isda_oen : std_logic;	-- Linha interna do I2C
	signal sda_chk		: std_logic;		-- checa status SDA0 (arbitragem multi mestre)
	signal dscl_oen		: std_logic;
```

```
        signal sSCL, sSDA         : std_logic;      -- sincroniza entradas de SCL and SDA
        signal clk_en, slave_wait : std_logic;      -- gera sinais de clock
        signal ial                : std_logic;      -- internal arbitration lost signal
        signal cnt : unsigned(15 downto 0);         -- clock divider counter (synthesis)

begin
        -- sempre que o escravo não está pronto pode demorar o ciclo puxando SCL baixo
        -- scl_oen de demora
        process (clk)
        begin
           if (clk'event and clk = '1') then
             dscl_oen <= iscl_oen;
           end if;
        end process;
        slave_wait <= dscl_oen and not sSCL;

        -- Gerador de sinal de Clock I2C
        gen_clken: process(clk, nReset)
        begin
           if (nReset = '0') then
             cnt    <= (others => '0');
             clk_en <= '1';
           elsif (clk'event and clk = '1') then
             if (rst = '1') then
               cnt    <= (others => '0');
               clk_en <= '1';
             else
               if ( (cnt = 0) or (habilita = '0') ) then
                 if (slave_wait = '0') then
                   cnt    <= clk_cnt;
                   clk_en <= '1';
                 else
                   cnt    <= cnt;
                   clk_en <= '0';
                 end if;
               else
                 if (slave_wait = '0') then
                   cnt <= cnt -1;
                 end if;
                 clk_en <= '0';
               end if;
             end if;
           end if;
        end process gen_clken;


        -- Gerador de controle de stop/start I2C
        bus_status_ctrl: block
          signal Reconfig, dSDA    : std_logic;       -- atrasos (delay) sSCL e sSDA
          signal SCL   : std_logic;                   -- condicao de inicio
```

```
  signal SDA   : std_logic;                    -- condicao de parada
  signal cmd_stop      : std_logic;            -- comando de parada
  signal ibusy         : std_logic;            -- sinal de ocupado interno (busy)
begin
   -- Controle de sincronizacao das entradas SCL e SDA
   synch_scl_sda: process(clk, nReset)
   begin
      if (nReset = '0') then
        sSCL <= '1';
        sSDA <= '1';
        Reconfig <= '1';
        dSDA <= '1';
      elsif (clk'event and clk = '1') then
        if (rst = '1') then
          sSCL <= '1';
          sSDA <= '1';
          Reconfig <= '1';
          dSDA <= '1';
        else
          sSCL <= scl_e;
          sSDA <= sda_e;
          Reconfig <= sSCL;
          dSDA <= sSDA;
        end if;
      end if;
   end process synch_SCL_SDA;

   -- detecta condicao de inicio => descida de borda em SDA e SCL alto
   -- detecta condicao de parada => subida de borda em SDA e SCL alto
   detect_sta_sto: process(clk, nReset)
   begin
      if (nReset = '0') then
        SCL <= '0';
        SDA <= '0';
      elsif (clk'event and clk = '1') then
        if (rst = '1') then
          SCL <= '0';
          SDA <= '0';
        else
          SCL <= '0';
          SDA <= '1';
        end if;
      end if;
    end process detect_sta_sto;

   -- gera sinal de ocupado do Bus i2c
   gen_ocupado: process(clk, nReset)
   begin
      if (nReset = '0') then
        ibusy <= '0';
```

```
      elsif (clk'event and clk = '1') then
        if (rst = '1') then
          ibusy <= '0';
        else
          ibusy <= (SCL or ibusy) and not SDA;
        end if;
      end if;
    end process gen_ocupado;
    ocupado <= ibusy;

    -- Gerador de arbitragem (generate arbitration lost signal)
    gen_al: process(clk, nReset)
    begin
      if (nReset = '0') then
        cmd_stop  <= '0';
        ial       <= '0';
      elsif (clk'event and clk = '1') then
        if (rst = '1') then
          cmd_stop  <= '0';
          ial       <= '0';
        else
          if (clk_en = '1') then
            if (cmd = I2C_CMD_STOP) then
              cmd_stop <= '1';
            else
              cmd_stop <= '0';
            end if;
          end if;

          if (c_state = idle) then
            ial <= (sda_chk and not sSDA and isda_oen);
          else
            ial <= (sda_chk and not sSDA and isda_oen) or (SDA and not cmd_stop);
          end if;

        end if;
      end if;
    end process gen_al;
    al <= ial;

    -- gera sinal de dout, armazena dout em subida de SCL
    gen_dout: process(clk)
    begin
      if (clk'event and clk = '1') then
        if (sSCL = '1' and Reconfig = '0') then
          dout <= sSDA;
        end if;
      end if;
    end process gen_dout;
  end block bus_status_ctrl;
```

```
-- Gerador de Maquina de estado
nxt_state_decoder : process (clk, nReset, c_state, cmd)
begin
   if (nReset = '0') then
     c_state  <= idle;
     cmd_ack  <= '0';
     iscl_oen <= '1';
     isda_oen <= '1';
     sda_chk  <= '0';
   elsif (clk'event and clk = '1') then
     if (rst = '1' or ial = '1') then
       c_state  <= idle;
       cmd_ack  <= '0';
       iscl_oen <= '1';
       isda_oen <= '1';
       sda_chk  <= '0';
     else
       cmd_ack <= '0'; -- default no acknowledge

       if (clk_en = '1') then
         case (c_state) is
           -- idle
           when idle =>
             case cmd is
               when I2C_CMD_START => c_state <= start_a;
               when I2C_CMD_STOP  => c_state <= stop_a;
               when I2C_CMD_WRITE => c_state <= wr_a;
               when I2C_CMD_READ  => c_state <= rd_a;
               when others        => c_state <= idle;
             end case;

             iscl_oen <= iscl_oen;        -- mantem SCL em mesmo estado
             isda_oen <= isda_oen;        -- mantem SDA em mesmo estado
             sda_chk  <= '0';             -- nao checa SDA

           -- start
           when start_a =>
             c_state  <= start_b;
             iscl_oen <= iscl_oen;        -- mantenha SCL em mesmo estado (para
                                          -- repetir inicio)
             isda_oen <= '1';             -- SDA fixo alto
             sda_chk  <= '0';             -- não confira SDA

           when start_b =>
             c_state  <= start_c;
             iscl_oen <= '1';             -- selecione SCL high
             isda_oen <= '1';             -- mantenha SDA high
             sda_chk  <= '0';             -- nao check SDA
```

```
when start_c =>
   c_state  <= start_d;
   iscl_oen <= '1';              -- mantenha SCL high
   isda_oen <= '0';              -- selecione SDA low
   sda_chk  <= '0';              -- nao check SDA

when start_d =>
   c_state  <= start_e;
   iscl_oen <= '1';              -- mantenha SCL high
   isda_oen <= '0';              -- mantenha SDA low
   sda_chk  <= '0';              -- nao check SDA

when start_e =>
   c_state  <= idle;
   cmd_ack  <= '1';              -- command completed
   iscl_oen <= '0';              -- selecione SCL low
   isda_oen <= '0';              -- mantenha SDA low
   sda_chk  <= '0';              -- nao check SDA

-- stop
when stop_a =>
   c_state  <= stop_b;
   iscl_oen <= '0';              -- mantenha SCL low
   isda_oen <= '0';              -- selecione SDA low
   sda_chk  <= '0';              -- nao check SDA

when stop_b =>
   c_state  <= stop_c;
   iscl_oen <= '1';              -- selecione SCL high
   isda_oen <= '0';              -- mantenha SDA low
   sda_chk  <= '0';              -- nao check SDA

when stop_c =>
   c_state  <= stop_d;
   iscl_oen <= '1';              -- mantenha SCL high
   isda_oen <= '0';              -- mantenha SDA low
   sda_chk  <= '0';              -- nao check SDA

when stop_d =>
   c_state  <= idle;
   cmd_ack  <= '1';              -- command completed
   iscl_oen <= '1';              -- mantenha SCL high
   isda_oen <= '1';              -- selecione SDA high
   sda_chk  <= '0';              -- nao check SDA

-- read
when rd_a =>
   c_state  <= rd_b;
   iscl_oen <= '0';              -- mantenha SCL low
```

```
   isda_oen <= '1';             -- tri-state SDA
   sda_chk  <= '0';             -- nao check SDA

when rd_b =>
   c_state  <= rd_c;
   iscl_oen <= '1';             -- selecione SCL high
   isda_oen <= '1';             -- tri-state SDA
   sda_chk  <= '0';             -- nao check SDA

when rd_c =>
   c_state  <= rd_d;
   iscl_oen <= '1';             -- mantenha SCL high
   isda_oen <= '1';             -- tri-state SDA
   sda_chk  <= '0';             -- nao check SDA

when rd_d =>
   c_state  <= idle;
   cmd_ack  <= '1';             -- comando completado
   iscl_oen <= '0';             -- selecione SCL low
   isda_oen <= '1';             -- tri-state SDA
   sda_chk  <= '0';             -- nao check SDA

-- write
when wr_a =>
   c_state  <= wr_b;
   iscl_oen <= '0';             -- mantenha SCL low
   isda_oen <= din;             -- selecione SDA
   sda_chk  <= '0';             -- nao check SDA (SCL low)

when wr_b =>
   c_state  <= wr_c;
   iscl_oen <= '1';             -- selecione SCL high
   isda_oen <= din;             -- mantenha SDA
   sda_chk  <= '1';             -- check SDA

when wr_c =>
   c_state  <= wr_d;
   iscl_oen <= '1';             -- mantenha SCL high
   isda_oen <= din;             -- mantenha SDA
   sda_chk  <= '1';             -- check SDA

when wr_d =>
   c_state  <= idle;
   cmd_ack  <= '1';             -- comando completado
   iscl_oen <= '0';             -- selecione SCL low
   isda_oen <= din;             -- mantenha SDA
   sda_chk  <= '0';             -- nao check SDA (SCL low)

when others =>
```

```
                end case;
              end if;
            end if;
          end if;
        end process nxt_state_decoder;


        -- assign outputs
        scl_s   <= '0';
        scl_oen <= iscl_oen;
        sda_s   <= '0';
        sda_oen <= isda_oen;
end architecture structural;
```

REFERÊNCIAS

ALTERA – *"System On Programmable Chip"* – San Jose, California, 2004

ARM. *AMBA Specification2.0.* Disponível em: <http://www.arm.com/armtech/AMBA_Spec?OpenDocument>.Mai, 1999. Acessado em: Fev. de 2004.

ARNOLD, J.– *"The SPLASH 2 Software Evironment"*, IEEE Workshop on FPGAs for Custom Computing Machines, pp 88-93, April, 1993

ARNOLD, J. and D. Buell and E. Davis– *"SPLASH 2"*, Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures, pp 316-324, 1992.

ARÓSTEGUE, J.M.M – *"FIPSoc – Sistema Chip Programable en Campo"* - Universitat Politècnica de Catalunya, Espanha, 2004.

ATHANAS, P. M. - *"An Adaptive Machine Architecture and Compiler for Dynamic Processor Reconfiguration*", Ph.D. thesis, Brown Univerity, USA, 1992

ATHANAS, P. M.; SILVERMAN, H. F. *Processor Reconfiguration through Instruction-Set Metamorphosis*; IEEE Computer, v. 26, no. 3, Mar. 1993

ATMEL – *"8 BIT Microcontroller with 32K bytes QuickFlash"* – San Jose, 2004.

BANNATYNE, Ross and Greg Viot – *"Introduction to Microcontrollers – Part 1 and 2"*, Embedded Systems Conference, San Francisco, 1998.

BONATO, V. – *"Projeto de um módulo de aquisição e pré-processamento de imagem colorida baseado em computação reconfigurável e aplicado a robôs móveis"* – Defesa e Dissertação Apresentada ao Programa de Pós-Graduação em Ciência da Computação do ICMC-USP para obtenção do título de Mestre, São Carlos, S.P., 2004

BOSCH, G. *Object Oriented Design*. Redwood City: The Benjamin/Cunnings, 1991.

BREBNER, G.; DIESSEL, O. *Chip-Based Reconfigurable Task Management;* in Proceedings of the 11a. International Workshop on Field Programmable Logic and Applications (FPL´2001), Belfast, Northen Ireland, Springer (LNCS 2147), Ago. 2001.

BROWN, S.; ROSE, J. *Architecture of FPGA and CPLDs: A tutorial;* IEEE Design and Test of Computers, v. 13, 1996.

BROWN, S.; ROSE, J.; FRANCIS, R.; VRANESIC, Z. *Field Programmable Gate Array*. Kluwer Academic Publishers, Jun. 1992.

CARDIM, M.H.C. *"RtraSSoc51 – Adaptável Superescalar Reconfigurável Sistema em Chip – O Processodor Harward Superescalar"* – Qualificação Apresentada ao Programa de Pós-Graduação em Ciência da Computação do UNIVEM para obtenção do título de Mestre, Marília, S.P., 2004.

CARDOSO J. M. P.; VESTIAS M. P. *Architectures and compilers to support reconfigurable computing*.; ACM Crossroads Student Magazine, 2000. Disponível em: <http:// www.acm.org/crossroads/xrds5-3/rcconcept.html>; Acessado em: 15 fev. 2004.

CARRILLO, J. E.; CHOW, P. The Effect of Reconfigurable Units in Superscalar Processors. In: Acm/Sigda International Symposium on Field Programmable Gate Array – FPGA 2001, Fev. 2001.

CHAN, P. K.; MOURAD, S. *Digital design using field programmable gate arrays*. Prentice Hall, Abr. 1994.

CNSEC, - *"Embedded, Everywhere - A Research Agenda for Networked Systems of Embedded Computers"*- Committee on Networked Systems of Embedded Computers; Computer Science and Telecommunications Board; Division on Engineering and Physical Sciences; National Research Council; NATIONAL ACADEMY PRESS Washington, DC, 2001

COSTA, R. M. *Microcontroladores em Computação Reconfigurável – Uma Aplicação*. Dissertação de Mestrado em Ciências da Computação – Universidade Federal de São Carlos, São Carlos - SP, 2000.

COSTA, K. A. P., - *" RtrASSoc - Adaptável Superescalar Reconfigurável Sistema em Chip - O Sistema Operacional Embutido"* – Defesa de Dissertação Apresentada do Programa de Pós-Graduação em Ciência da Computação do UNIVEM para obtenção do título de Mestre, Marília, S.P., 2004

DANDALIS, Andreas and Viktor K. Prasana – *"Configuration Compression for FPGA-based Embedded Systems"*, ACM/SIGDA International Symposium on Field Progammable Gate Array – FPGA´2001, pp 173-182, February, 2001.

DEHON, A. - *"Reconfigurable Architecture for General-Purpose Computing"*, Ph.D. thesis, Massachusetts Institute of Technology, 1996.

DEHON, A.; WAWRZYNEK, J. *Reconfigurable Computing: What, Why, and Design Automation;* in Proceedings of the 36a. ACM IEEE Design Automation Conference (DAC´99), New Orleans, ACM Press, Jun. 1999.

DIAS A. F.; LAVARENNE C.; AKIL M.; SOREL Y. *Optimized implementation of real-time image processing algorithms on field programmable gate arrays*. Proc. Of the 4th Intl. Conference on Signal Processing, Beijing, Oct. 1998.

DOLPHIN – *"Flip 8051 – CORE 8051"* – France, 2003. Disponível em: <http://www.xilinx.com/bvdocs/whitepapers/wp114.pdf>; Acessado em: 15 fev. 2004.

DOUCET, F.; GUPT, R. K. *Microelectronic System-on-Chip Modeling using Objects and their Relationships.* Disponível em: <http://www.cecs.uci.edu/~balboa/pubs/doucet-osee1.pdf>. Acesso em: 22 jan. 2004.

DUBOIS, M. et al. *Rapid hardware prototyping on RPM-2*; IEEE Design and Test of Computers, v.15, no. 3, Set. 1998.

EDWARDS S.; LAVAGNO L.; LEE E. A.; SANGIOVANNI-VINCENTELLI A.*Design of embedded systems: formal models, validation, and synthesis*. Proceedings of IEEE, v.85, n.3, March 1997.

FALLSIDE, H.; SMITH, M. J. *Internet Connected FPL;* in Proceedings of the 10o. International Workshop on Field Programmable Logic and Applications (FPL´2000), Villach, Austria, Springer (LNCS 1896), Set 2000.

FIELDBUS FOUNDATION. *Foundation Fieldbus Technical Overview: FD-043 versão 2.0*, 1996 (rev. 1998). Disponível em: http://www.fieldbus.org/pdf/fd-043s.pdf . Acessado em: 19 mar. 2004.

FORNARI, A. – *" RtrASSoc - Adaptável Superescalar Reconfigurável Sistemas em Chip Rotinas Reconfiguráveis no Modelo de Compressão Relocação e Defragmentação"* – Defesa de Dissertação Apresentada do Programa de Pós-Graduação em Ciência da Computação do UNIVEM para obtenção do título de Mestre, Marília, S.P., 2004

FRANCO, L.R.H.R.; SOUZA, G.G.; NAZÁRIO, R.P.; CAVALCANTI, M.J.L. EIA RS-485 Fieldbus. GAII – Grupo de automação e Informática Industrial. Escola Federal de Engenharia de Itajubá/ MG, 2003. Disponível em : <http://www.iee.efei.br/~gaii/rs485/hp_rs485.htm#RS-485>. Acessado em: 06 mai. 2004.

GOKHALE, M. et al. SPLASH: *A reconfigurable Linear Logic Array*; in Proceedings of the Internationsl Conference on Parallel Processing (ICPP´97), v. 9, Ago. 1990.

GOKHALE, M and W. Holmes and A. Kopser and S. Lucas and R. Minnich and D. Sweely and D. Lopresti – *"Building and Using a Highly Parallel Programmable Logic Array"* – IEEE Computer, pp 81-89, Jan. 1991.

GOLDBERG, I.; WAGNER, D. *Architectural Considerations for Cryptanalytic Hardware*. Disponível em: <http://www.cs.berkeley.edu/~iang/isaac/hardware/paper.ps> University of California, Berkeley, 1996. Acessado em: Fev. 2004.

GONÇALVES R. A. *ARCHITECT-R: Uma ferramenta para o desenvolvimento de robôs móveis reconfiguráveis*, ICMC-USP, Dissertação de Mestrado, Agosto 2000.

GONÇALVES R. A.; Wolf D. F.; Coelho F. A. S.; Teixeira M. A.; Ribeiro A. A. L.; Marques E. *Architect: Um sistema de computação reconfigurável*. In CORE2000 Workshop de Computação Reconfigurável, 2000.

GRANDPIERRE T.; LAVARENNE C.; SOREL Y. *Optimized rapid prototyping for real-time embedded heterogeneous multiprocessors*. CODES'99 7th Intl. Workshop on Hardware/Software Co-Design, Rome, May 1999.

HADZIC, I. UDANI, S.; SMITH, J. *FPGA viruses;* in Proceedings of the 9o. International Workshop on Field Programmable Logic and Applications (FPL´99), Glasgow, Scotland, Springer (LNCS 1973), Set 1999.

HAUCK, S.; FRY T. W.; HOSLER M. M.; KAO J. P. *The Chimaera Reconfigurable Functional Unit* , IEEE Symposium on FPGAs for Custom Computing Machines, pp. 87-96, 1997.

HAUCK, S. *The Roles of FPGAs in Reprogrammable Systems***.** Proceedings of the IEEE. V. 86, p.615-638, Abr. 1998.

_____. *Reconfigurable Computing: A Survey of Systems and Software***,** submitted *to* ACM Computing Surveys, *2000*.

IBM. *IBM, Xilinx announce multi-milion dollar manufacturing deal*. Disponível em <http://www-3.ibm.com/chips/news/2002/0304_xilinx.html. Mar. 2002.

INTEL CORPORATION. *8-bit Embedded Controller Handbook*. Santa Clara, 1998.

ISELI, Christian – *"Spyder - A Reconfigurable Processor Development System"*, Thèse de Docteur, École Polytechnique Fédéraçe de Lausanne, Lausanne, 1996.

LAHIRI , K.; RAGHUNATHAN, A.; DEY, S. *Evalution of the Traffic-Performance Characteristics of System-on-Chip Communication Architectures***.** Disponível em <http://www.computer.org/proceedings/vlsid/0831/0831toc.htm>. Acesso em: 25 jan. 2004.

LEE, S., *et al –"Reconfigurable SoC Design with Hierarchical FSM and Synchronous Dataflow Model"*- ACM/SIGDA – 10th International Workshop on Hardware/Software Codesign, PP 199-204, 2002.

LEVINSON, L. et al. *Preemptive Multitasking on FPGAs*; in Proceedings of the 8o. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM´00), Napa Valley, California, IEEE Computer Society Press, Abr. 2000.

LEWIS, D. M. et al. *The Transmogrifer-2: A 1 million gate rapid prototyping System*; IEEE Transactions on VLSI Systems, Jun 1998.

LOPES, J. J. – *"RtrASSoc –Adaptável Superescalar Sistema em Chip – hw/sw codesign"* – Relatório FAPESP, Proc: 03/06913-6, 2004.

LUDWIG, S.; SLOUS, R.; SINGH, S. *Implementing Photoshop Filters in Virtex*. In Proceedings of the 9o. International Workshop on Field Programmable Aplications (FPL´99), Glasgow, Scotland, Springer (LNCS 1673), Set. 1999.

LYSAGHT, P.; DUNLOP, J. *Dynamic Reconfiguration of Field Programmable Gate Arrays*; in Proceedings of the 3o. International Workshop on Field Programmable Logic and Applications (FPL´93), Oxford, UK, Abingdon EE&CS Books, Set. 1993.

MACBETH, J.; LYSAGHT, P. *Dynamically Reconfigurable Cores;* in Proceedings of the 11o. International Workshop on Field Programmable Logic and Applications (FPL´2001), Belfast, Northen Ireland, Springer (LNCS 2147), Ago. 2001.

MARCHI, L. *Projeto e Implementação de uma Unidade Lógica Arimética (ULA) utilizando VHDL e PLD's***.** Universidade Católica Dom Bosco**,** Campo Grande, 2002. Disponível em:

<http://www.Ec.ucdb.br/engenharia/projeto-graduação/projetos-2002/projetos/proj-g2.pdf>. Acessado em: 20 fev. 2004.

MARQUES, F. S. *Microcontrolador BDLC – Uma Implementação para Aplicação na Indústria Automobilística*. Universidade Federal de Pelotas, Pelotas – RS, 2000. Disponível em: <http://www.ufpe.tche.br/prg/sisbi/bibct/acervo/info/2000/Mono-Felipe.pdf>. Acessado em: 16 jan. 2004.

MASSIMO, B., *et al – "HW/SW Partitioning and Code Generation of Embedded Control Applications on a Reconfigurable Architecture Platform"*. ACM/SIGDA – 10th International Workshop on Hardware/SoftwaREcODESIGN, PP 151-156, 2002.

MERINO, P.; LOPES, J. C.; JACOME, M. *A Hardware Operating System for Dynamic Reconfiguration of FPGAs*, in Proceedings of the 8o. Internacional Workshop on Field Programmable Logic and Applications (FPL´98), Tallin, Estonia, Springer (LNCS 1482), Set. 1998.

MESQUITA, D. G. *Contribuições para Reconfiguração Parcial, Remota e Dinâmica de FPGAs*; Dissertação de Mestrado, PUCRS, Mar. 2002.

MICROCHIP. *Microcontrolador PIC*. Disponível em <http://www.microchip.com>. Acessado em: 25/01/2004.

MORAES, F.; MESQUITA, D. *Tendências em Reconfiguração dinâmica de FPGAs*. Faculdade de Informática – PUCRS. Disponível em: <htt://www.inf.pucrs.br/~moraes/papers/2001-scr2001-palestra.pdf>. Acesso em: 20 jan. 2004.

NAKAMURA , T. *Preface Special Issue on System-on-Chip (SOC)***,** 2000**.** Disponível em: <http://www.temple.edu/scdc/icee2000.pdf>.Acesso em: 22 jan. 2004.

ORDONEZ, E. D. M; SILVA, J. L. – *Reconfigurable Computing – Experiences and Perspectives*, Fundação de Ensino Eurípides Soares da Rocha (FEESR), Marília, 294pg., Agosto, 2000.

PARMA, G. G. *Introdução ao VHDL***.** Universidade Federal de Minas Gerais**,** M.G. Disponível em: <http://www.cpdee.ufmg.br/~parma/spp/Prat02.pdf>. Acessado em: 15 fev. 2004.

PHILIPS Semiconductors. *The I2C bus specification version 2.1*. Jan. 2000. Disponível em: <http://www.semiconductors. Philips.com/buser/i2c/>. Acessado em 15 Dez. 2003.

PHILLIPS, S. & HAUCK, S. –*"Automatic Layout of Dpmain-Specific Reconfigurable Subsystem for System-on-Chip"* - 10th ACM International Symposium on Field Programmable Gate Array – FPGA'2002, pp 135-173,2002.

PIACENTINO, Michael R., Gooitzen S. Van der Wal, Michael W. Hansen – "*Reconfigurable Elements for a Video Pipeline Processor"*, IEEE Symposium on FPGAs for Custom Computing Machines, pp 1-10, 1999.

QUENOT, G., I. Kraljic, J. Serot and B. Zavidovique – *"A Reconfigurable Computing Engine for Real-Time Vision Automata Prototyping"*, IEEE Workshop on FPGAs ofr Custom Computing Machine, April, 1994.

RATHA, N. and A. Jain and D. Rover, *"Convolution on SPLASH 2"*, IEEE Workshop on FPGA for Custom Computing Machine, pp 204-213, April, 1995

RIBEIRO, A. A. L. *Reconfigurabilidade dinâmica e remota de FPGAs*. Tese de mestrado, Universidade de são Paulo – ICMC, São Carlos –SP, Jul. 2002.

ROBINSON, D.; LYSAGHT, P. *Verification of Dynamically Reconfigurable Logic;* in Proceedings of the 10o. International Workshop on Field Programmable Logic and Applications (FPL´2000), Villach, Austria, Springer (LNCS 1896), Set 2000.

SALCIC, Z.; SMAILAGIC, A. *Digital Systems Design and Prototyping: Using Field Programmable Logic and Hardware Description Languages*. Kluwer Academic Plubishers, 2a. Edition, 2000.

SANCHEZ, E. et al. Tatic and Dinamic Configurable Systems; IEEE Transactions on Computers, 1999.

SHALAN, M. & MOONEY, V.J. –*"Hardware Support for Real Time Embedded Multiprocessor System-on-Chip Memory Management"*. - ACM/SIGDA – 10th International Workshop on Hardware/SoftwaREcODESIGN, PP 79-84, 2002.

SIDHU, R. et al. *A Self-Reconfigurable Gate Array Architecture*; in Proceedings of the 10o. International Workshop on Field Programmable Logic and Applications (FPL´2000), Villach, Austria, Springer (LNCS 1896), Set 2000.

SIDSA. *FIPSOC Mixed Signal System-on-Chip*. Disponível em: <http://www.sidsa.com/FIPSOC/fipsocl.pdf>, 2002

SILVA, J. L, et al. – *"RtraSSoc - Adaptable, Superscalar, Reconfigurable, Programmable System on Chip - The Embedded Operating System - EOS"*- The 4rd IEEE International Workshop on System-on-Chip for Real Time Applications, 196-200,Alberta, Canada, 2004.

SILVA, J. L, et al. – *"RtrASSoc - An Adaptable Superscalar Reconfigurable System On Chip - The Simulator"* - The 3rd IEEE International Workshop on System-on-Chip for Real Time Applications, 196-200,Alberta, Canada, 2003.

SILVA, J.L. & ORDONEZ, E. D. M. – *" Reconfigurable Computing –Experiences and Perspectives"* , Fundação de Ensino Eurípides Soares da Rocha (FEESR), Marília, 294pg., Agosto, 2000.

SILVA Jr., V. P. da. *Aplicações Práticas do Microcontrolador 8051*. Editora Érica Ltda., São Paulo – SP, 1994.

SILVA NETO, E.; HASLER, H.; SOUZA, L.; FRANCO, L.; ARAÚJO, O.; ROCHA, W. *Comunicação em Chão de Fábrica: Solução Fieldbus*.In: Congresso Brasileiro da ISA. **Anais...** 1995. pp. 337-345.

SIMMLER, H.; LEVINSON, L.; MANNER, R. *Multitasking on FPGA Coprocessors;* in Proceedings of the 10o. International Workshop on Field Programmable Logic and Applications (FPL´2000), Villach, Austria, Springer (LNCS 1896), Set 2000.

SOUZA, D. J. *Desbravando o PIC*. Editora Érica, 2ª. Edição. São Paulo – SP, 2000.

THOMPSOM, T. *ComputerWorld An IDG Company***.** *Digital Signal Processor*, [s.l.], 2001. Disponível em: < http://www.computerworld.com/cwi/story/0,1199,nav47-68-85-1950_sto58487,00.html > . Acesso em: 5 jan. 2004.

VERONESI, R. L. M. *RtrASSoc51 – Adaptável Superescalar Reconfigurável Programável Sistema em Chip rI2C (reconfigurable Inter Integrated Circuits*). In: Workshop em Sistemas Computacionais de Alto Desempenho – WSCAD 2005. Artigo submetido. Rio de Janeiro – RJ, 2005.

VILLASENOR, J.; MANGIONE-SMITH, W. H. *Configurable Computing*. In Scientific American Magazine, New York, Jun. 1997.

WIRTHLIN, M.; HUTCHINGS, B. L. *A Dynamic Instruction Set Computer.* In proceddings of the 3rd IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM´95), Napa Valley, Califirnia, IEEE Computer Society Press, Abr 1995.

WOLF, D. F. *Projeto de uma rede neural usada no reconhecimento de gestos por robôs móveis*. Dissertação de Mestrado, ICMC – USP, Set. 2001.

XILINX Inc. Virtex II 1.5v Field Programmable Gate Arrays. Disponível em: <http://www.xilinx.com/partinfo/ds031.pdf>, Out. 2001.

XILINX - *"Embedded Operating System"*- San Jose, California, 2004.

ZANGUETTIN, O. F. – *"RtrASSoc - Sistema em Chip Adaptável Superescalar Reconfigurável - Processador Superescalar Embutido"* – Defesa de Dissertação Apresentada ao Programa de Pós-Graduação em Ciência da Computação do UNIVEM para obtenção do título de Mestre, Marília, S.P., 2004.