UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

THIAGO POUZA MUSSOLINI

DESENVOLVIMENTO DE UM MICROCONTROLADOR DE 8 BITS EM VHDL BASEADO NO CONJUNTO DE INSTRUÇÕES DO 8051 COM COMUNICAÇÃO SERIAL I^2C E CRIPTOGRAFIA AES128

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Elétrica da UNIFEI como requisito parcial para a obtenção do título de Mestre em Engenharia Elétrica

Área de Concentração: Microeletrônica

Orientador: Professor Tales Cleber Pimenta
Co-orientador: Professor Robson Luiz Moreno

Julho de 2011
Itajubá – MG

“Nunca se afaste de seus sonhos, pois se eles se forem, você continuará vivendo, mas terá deixado de existir.”

Charles Chaplin.

Agradecimentos

Aos meus pais, Reinaldo e Maria do Carmo.

À minha namorada Juliana.

Ao meu orientador Tales Cleber Pimenta.

Ao meu co-orientador Robson Luiz Moreno.

À Universidade Federal de Itajubá.

Ao Grupo e aos amigos da Microeletrônica.

Ao pessoal da MINASIC.

Resumo

Apresenta-se neste trabalho a estrutura de um microcontrolador de 8 bits em linguagem VHDL. Ele é baseado no conjunto de instruções do popular 8051 e modificações em sua arquitetura original foram realizadas para adequação as necessidades do Grupo de Microeletrônica da UNIFEI.

Com a utilização da linguagem VHDL é possível realizar modificações na arquitetura do microcontrolador para acrescentar novas funcionalidades ou simplesmente retirar as que não sejam necessárias e imediatamente realizar simulações por *softwares* e/ou prototipação para testes de funcionamento em FPGA.

O microcontrolador M8051, desenvolvido foi validado através de simulações de cada hierarquia construída e também seu conjunto. Através dos programas desenvolvidos em linguagem *assembly* foram realizados testes do funcionamento da estrutura em FPGA para garantir a funcionalidade de todos os recursos presentes.

Abstract

This paper presents the structure of an 8-bit microcontroller in VHDL. It is based on the instruction set of the popular 8051 and modifications on the original architecture were made to meet the demands and needs of the Microelectronics Group.

VHDL allows modifications to be made in the microcontroller architecture such as adding new functionality or just removing those that are not necessary and perform simulation or FPGA implementation.

The microcontroller M8051, was developed and validated through simulations of each hierarchy and the whole set tests were performed in the through assembly programming and FPGA implementation to ensure functionality of all features presented.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de abreviaturas

PLD	*Programmable Logic Devices*
FPGA	*Field Programmable Gate Array*
IC	*Integrated Circuit*
CLB	*Configurable Logic Block*
LUT	*Lookup Table*
SRAM	*Static Random Access Memory*
ASIC	*Application Specific Integrated Circuit*
VHDL	*Very High Speed Integrated Circuit Hardware Description Language*
DoD	*Department of Defense*
VHSIC	*Very High Speed Integrated Circuit*
IEEE	*Institute of Electrical and Electronic Engineers*
CPU	*Center Processing Unit*
I/O	In put/ Out put
UART	*Universal Asynchronous Receiver Transmitter*
CISC	*Complex Instruction Set Computer*
SFR	*Special Function Register*
PSW	*Program Status Word*
POR	*Power On Reset*
ULA	Unidade Lógica Aritmética
RAM	*Random Access Memory*
FIFO	*First-In, First*-Out
ROM	*Read Only Memory*
SDA	*Serial Data*
SCL	*Serial Clock*

Capítulo 1

1 Introdução

Com a evolução da tecnologia de semicondutores e consequentemente do mundo digital, os microcontroladores tornaram-se presentes em praticamente todos os produtos digitais, desde um simples controle de portão de garagem até aos mais modernos celulares.

Outra vertente que evoluiu junto com os semicondutores foram os Dispositivos Lógicos Programáveis – PLD (*Programmable Logic Device*), principalmente o Arranjo de Portas Programável em Campo - FPGA (*Field Programmable Gate Arrays*). Eles permitem que um circuito digital seja prototipado e testado pelo usuário em campo, sendo que, caso seja necessário alterações no circuito desenvolvido estas possam ser realizadas rapidamente sem nenhuma perda.

Com isso surge um novo segmento de pesquisa e desenvolvimento que é a implementação de microcontroladores ou microprocessadores em FPGAs. Empresas como Altera e Xilinx possuem suas próprias arquiteturas, o NIOS II e o Microblaze, respectivamente. Outras empresas, como a ARM e a Intel, em parcerias com a Xilinx, também disponibilizam versões de núcleos comerciais para projetos em FPGA.

1.1 Motivação e Objetivos

O Grupo de Microeletrônica da Universidade Federal de Itajubá – UNIFEI através de diversos trabalhos realizados com FPGA e com o auxilio das estruturas proprietárias dos microprocessadores adquiriu ao longo dos anos uma vasta experiência. Contudo, para diminuir a dependência dessas arquiteturas auxiliares proprietárias iniciaram-se os trabalhos para que o grupo pudesse adquirir também o conhecimento em desenvolvimento de microcontroladores que, em paralelo, atendesse as demandas dos trabalhos em desenvolvimento e futuros.

O objetivo deste trabalho é desenvolver um circuito flexível, que seja possível realizar alterações em sua estrutura como, por exemplo, a introdução de novos periféricos ou a

exclusão de uma porta para a diminuição da quantidade de pinos, porém, com um conjunto de instruções fixo e altamente difundido.

1.2 Organização da dissertação

Esta dissertação está dividida em sete capítulos e dois apêndices. No Capítulo 2 é feita uma apresentação da tecnologia FPGA e da linguagem VHDL destacando suas principais características. Também é apresentado o sistema de desenvolvimento utilizado para realizar este trabalho.

O Capítulo 3 aborda as características do microcontrolador 8051 da Intel como capacidade de memória, conjunto de instruções, pinos e entre outros para a familiarização com as características do circuito.

A estrutura concebida para o desenvolvimento do microcontrolador M8051 é apresentada no Capítulo 4. Nele encontram-se as representações dos blocos implementados com os sinais de entrada e saída, comprovações de funcionamento correto através de *waveforms* ou *testbenchs* e uma explicação das ações realizadas.

No Capítulo 5 são apresentadas as etapas envolvidas no desenvolvimento da interface de comunicação serial I^2C, como ela foi integrada ao M8051, um exemplo de como utilizá-la e as comprovações de seu correto funcionamento.

O algoritmo de criptografia AES128 é apresentado no Capítulo 6. Nele encontram-se, também, as etapas realizadas para o desenvolvimento do bloco, os testes realizados para comprovação do funcionamento correto, como foi realizada a integração e um exemplo de como utilizar a estrutura.

No Capítulo 7 é feita uma análise dos resultados apresentados e a conclusão do estudo desenvolvido. Ainda nesse capítulo, são sugeridas idéias para trabalhos futuros.

O apêndice A contém o programa desenvolvido em linguagem *assembly* para utilização e gerenciamento da interface I^2C e o apêndice B contém o programa desenvolvido em linguagem *assembly* para utilização e gerenciamento da criptografia AES128.

Capítulo 2

2 FPGA e VHDL

2.1 Introdução

Esse capítulo apresenta a estrutura básica de um FPGA e o conceito da linguagem VHDL. O FPGA que terá suas características abordadas é a Cyclone II EP2C35F672C6 da Altera, presente no sistema de desenvolvimento DE2 da Terasic Technologies. A linguagem adotada para o desenvolvimento deste *hardware* foi a VHDL devido ao maior domínio do Grupo de Microeletrônica quando comparado com as demais linguagens existentes para tal finalidade.

2.2 PLD

Os PLDs são Circuitos Integrados – IC (*Integrated Circuit*) compostos basicamente por portas lógicas, *flip-flop* e registradores. Estas estruturas estão interconectadas através de fusíveis ou chaves especiais que podem ser configuradas e reconfiguradas para cada lógica desenvolvida [1]. Existem vários tipos de PLD e um de grande importância para a indústria é o FPGA [2].

2.2.1 FPGA

Dos diversos tipos de PLD, o FPGA é o mais complexo. Consiste em um arranjo bidimensional de blocos lógico-programáveis conectados entre si por meio de uma estrutura de interconexões (*Switch Matrix*), como mostrado na Figura 2.1. A estrutura interna do Bloco Lógico Configurável - CLB (*Configurable Logic Block*) é baseada em LUT (*Lookup Table*).

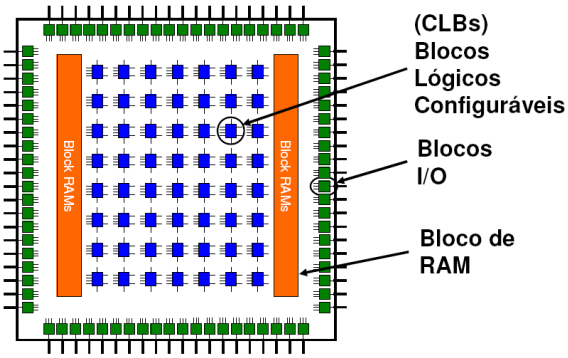

Figura 2.1 - Arquitetura do FPGA.

Essa tecnologia foi desenvolvida pela empresa Xilinx Inc., em 1985, para ser um dispositivo programável de acordo com as necessidades do projetista. Também permiti realizar prototipações de etapas intermediárias do circuito desenvolvido e/ou alterações do projeto final em tempo extremamente reduzido e com custos mais baixos em comparação com todo o processo de um Circuito Integrado para Aplicação Específica – ASIC (*Application Specific Integrated Circuit*).

Por ser um circuito lógico programável ou computação reconfigurável se torna uma opção intermediária na solução de lógicas complexas, pois combina o alto desempenho do *hardware* com a flexibilidade do *software*.

Para configurar o FPGA é utilizado um arquivo binário que contém as informações necessárias para especificar a função de cada unidade lógica e fechar as chaves da matriz de interconexão necessárias. Para gerar o arquivo binário utilizam-se ferramentas de *software* seguindo um determinado fluxo de projeto [2]. Neste projeto foi utilizado o *software* Quartus II 9.1 SP2 Web Edition que é uma versão disponibilizada pela Altera gratuitamente.

Dentre os vários segmentos em relação às arquiteturas reconfiguráveis, destacam-se os processadores reconfiguráveis. Estes processadores combinam as funções de um microprocessador com uma lógica reconfigurável e podem ser adaptados depois do processo de desenvolvimento [3].

2.2.2 Cyclone II EP2C35F672C6

Adotou-se para o desenvolvimento desse projeto, o FPGA Cyclone II EP2C35F672C6 da Família Cyclone II da Altera que é implementado em tecnologia 90nm [4]. As características principais da EP2C35F672C6 são apresentadas na Tabela 2.1.

Tabela 2.1- Características do FPGA EP2C35F672C6 [4].

Características	EP2C35F672C6
Quantidade de Pinos	672
Elementos Lógicos	33216
Blocos RAM M4K (4 kbits mais 512 bits de paridade)	105
Total RAM bits	483840
Multiplicadores Embarcados 9 X 9 bits	70
PLLs	4
Quantidade máxima de pinos utilizáveis para I/O	475

Os FPGAs da família Cyclone II utilizam Memórias Estáticas de Acesso Aleatório – SRAMs (*Static Random Access Memory*), as quais são voláteis, para armazenar suas configurações. Com isso, o arquivo de configuração deve ser transferido para o FPGA toda vez que o sistema for energizado.

Para a transferência do arquivo de configuração do FPGA com o circuito desenvolvido é utilizada a tecnologia Grupo de Ação Conjunta de Teste - JTAG (*Joint Test Action Group*) [5]. O *software* Quartus II da Altera gera automaticamente o arquivo binário de configuração do FPGA e a tecnologia JTAG, através do cabo *USB-Blaster*, torna o processo de gravação do circuito no sistema de desenvolvimento extremamente simples e rápido.

2.2.3 DE2

O sistema de desenvolvimento DE2 apresenta diversos periféricos, facilitando a realização de testes de circuitos desenvolvidos. Estes periféricos são mostrados na Figura 2.2 e as informações técnicas do sistema podem ser encontradas na referência [5].

2.3 VHDL

A Linguagem de Descrição de *Hardware* de Circuitos Integrados de Altíssima Velocidade – VHDL (*Very High Speed Integrated Circuit Hardware Description Language*) foi desenvolvida com o intuito de ser utilizada em todas as fases da criação de um sistema eletrônico digital. Ela promove o desenvolvimento, verificação, síntese e teste no desenvolvimento do *hardware*, bem como sua manutenção, modificação e expansão [6].

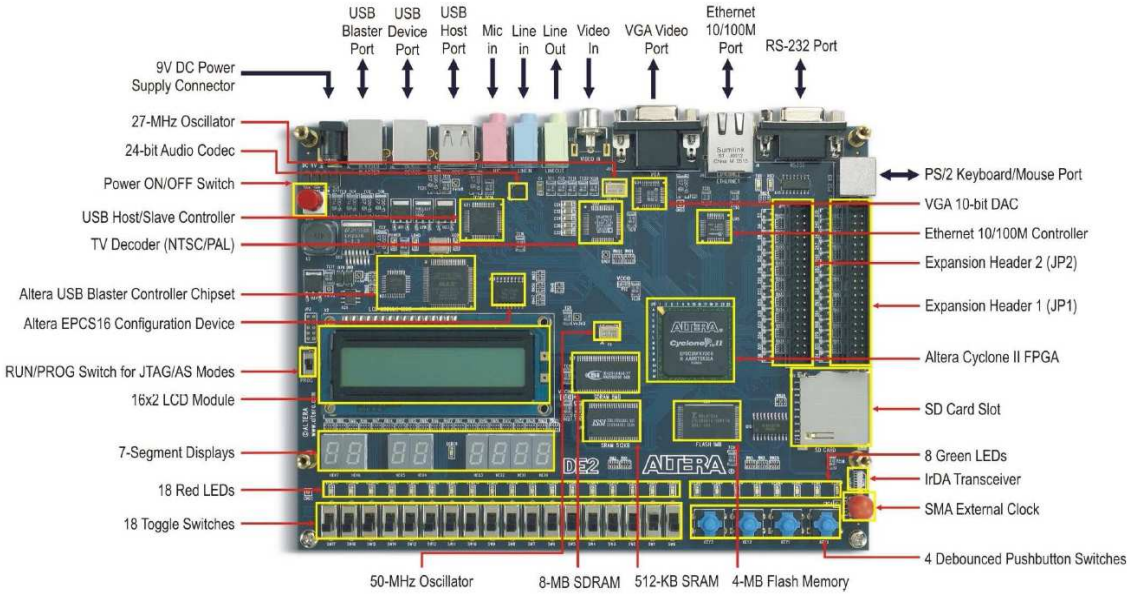

Figura 2.2 - Sistema de Desenvolvimento DE2 [5].

2.3.1 História

Foi originalmente desenvolvida sob o comando do Departamento de Defesa – DoD (*Department of Defense*) dos Estados Unidos, em meados da década de 1980, para documentar o comportamento de ASICs que compunham os equipamentos vendidos às Forças Armadas Americanas. A linguagem foi desenvolvida para substituir os complexos manuais que descreviam o funcionamento dos ASICs. A única metodologia largamente utilizada no projeto de circuitos era a criação através de diagramas de esquemáticos que possuíam como problema o fato de terem menor portabilidade, serem mais complexos para compreensão e extremamente dependentes da ferramenta utilizada para produzi-los [7].

Como os projetos de Circuito Integrado de Altíssima Velocidade - VHSIC (*Very High Speed Integrated Circuit*) eram de alta prioridade militar e haviam dezenas de fornecedores envolvidos, o DoD estava preocupado principalmente com as questões de portabilidade, documentação e compreensibilidade dos projetos. Cada um destes fornecedores atuava desenvolvendo partes dos projetos ou mesmo fornecendo componentes que viriam a se encaixar em outros sistemas maiores. Desta forma, o DoD optou por buscar desenvolver uma linguagem que servisse como base para a troca de informações sobre estes componentes e projetos. A linguagem deveria ser independente do formato original do circuito, e deveria servir como uma descrição e documentação eficiente do circuito, possibilitando os mais diferentes fornecedores e participantes a entender o funcionamento das outras partes e padronizando a comunicação [7].

O desenvolvimento da linguagem VHDL serviu inicialmente aos propósitos de documentação do projeto VHSIC. Entretanto, nesta época buscava-se uma linguagem que facilitasse o projeto de um circuito, ou seja, a partir de uma descrição textual, um algoritmo, pudesse desenvolver o circuito sem necessidade de explicitar as ligações entre componentes. A linguagem encaixa-se adequadamente a tais propósitos, podendo ser utilizada para as tarefas de documentação, descrição, síntese, simulação, teste, verificação formal e em alguns casos a compilação de *software* [7].

Após o sucesso inicial do uso da VHDL, a sua definição foi posta em domínio público o que levou a ser padronizada pelo Instituto de Engenheiros Elétricos e Eletrônicos - IEEE (*Institute of Electrical and Electronic Engineers*) em 1987. O fato de ser padronizada e de ser de domínio público ampliou ainda mais a sua utilização. Novas alterações foram propostas, como é natural num processo de aprimoramento, e a linguagem sofreu uma revisão e um novo padrão mais atualizado, que foi lançado em 1993. Pequenas alterações foram feitas em 2000 e 2002. Em setembro de 2008 foi aprovada pelo Comitê de Revisão do IEEE a mais recente versão, IEEE 1076-2008 [7].

2.3.2 Definição

O VHDL é uma linguagem estruturada que oferece a possibilidade de descrever e simular o *hardware*, com isso facilitando a validação ou verificação, tanto em termos de funcionamento quanto em tempos de atraso dos componentes e desempenho, antes da prototipação do sistema [7].

A descrição pode ser feita basicamente usando dois tipos (modelos) de descrição: estrutural e comportamental. Na descrição estrutural a organização física e topológica do sistema é descrita, ou seja, são especificadas as entradas e/ou saídas, os componentes lógicos, a interligação deles e os sinais que compõem o sistema.

Na descrição comportamental não é preciso descrever a organização física e topológica do sistema, mas, somente, as funções do sistema. Um programa que utiliza esse tipo de descrição possui o mesmo formato de um programa fonte escrito em uma linguagem de programação de alto nível como C++. Essa abordagem diminui a necessidade de conhecimento aprofundado em projeto de *hardware* aumentado a facilidade de desenvolvimento do sistema [7].

2.3.3 Vantagens e desvantagens

Algumas vantagens no uso da linguagem VHDL são projetos independentes da tecnologia, maior facilidade de atualização, exploração de alternativas arquiteturais em um nível mais alto de abstração, eliminação de erros de baixo nível, redução do tempo e custos de projeto e simplificação da documentação. Como desvantagem, o *hardware* gerado é menos otimizado, a controlabilidade/observabilidade de projeto reduzido e as simulações são mais lentas em relação à prototipagem no FPGA [7].

Capítulo 3

3 Microcontrolador 8051

3.1 Introdução

O microcontrolador 8051 é o mais popular [8] entre os microcontroladores e pode operar como um microcontrolador ou como um microprocessador [8] sendo membro original da família MCS-51 e sendo o núcleo (*core*) de todos os dispositivos MCS-51 [9]. A Figura 3.1 apresenta a arquitetura do 8051 com todos os blocos presentes em sua estrutura original.

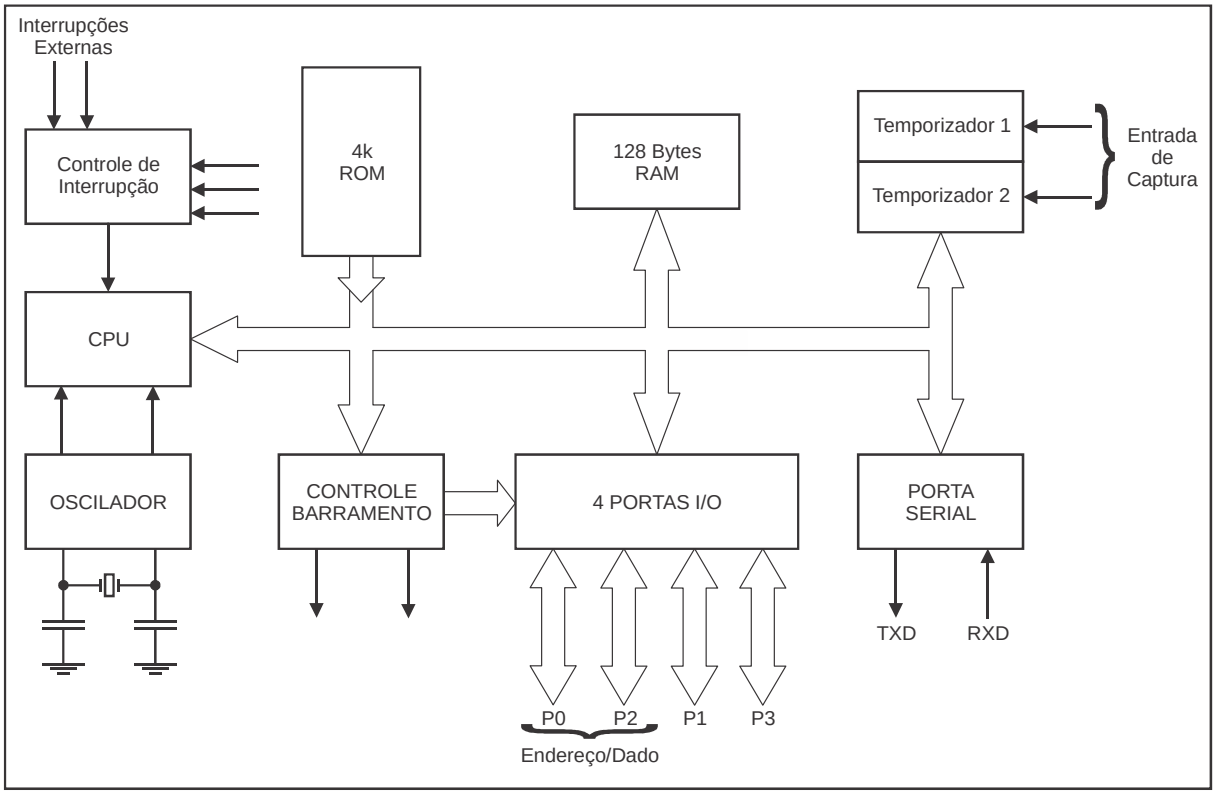

Figura 3.1 – Representação da arquitetura da estrutura original do 8051 [9].

A Unidade Central de Processamento – CPU (*Central Processing Unit*) do microcontrolador 8051 é otimizada para aplicações de controle e opera com 8 bits. Ela possui a capacidade de processamento booleano que é a ação de realizar processamentos bit a bit.

O microcontrolador 8051 conta com a capacidade de endereçamento externo de até 64 kbytes de espaço de memória, tanto para programa quanto para dados, além de possuir internamente 4k de memória de programa e 128 bytes de memória de dados.

A estrutura do 8051 tem 32 linhas de entrada/saída – I/O (*Input/*Output) bidirecionais e endereçadas individualmente para realizar a interação com outros dispositivos e periféricos como outros microcontroladores, memórias externas, chaves, sensores e *displays*, entre outros. Para utilização de memórias externas, todos os pinos das portas P0 e P2 são utilizados além de alguns pinos específicos.

Para realizar os procedimentos de temporização e/ou contagem, a arquitetura conta com dois registros dedicados de 16 bits, e para a comunicação serial conta com uma interface Transmissor/Receptor Universal Assíncrono – UART (*Universal Asynchronous Receiver Transmitter*) *full duplex* (transmissão bidirecional). Portanto, para o melhor aproveitamento e gerenciamento dessas estruturas, o microcontrolador tem 5 fontes de interrupções com 5 vetores de tratamento sendo 2 níveis de prioridade (alta ou baixa) selecionáveis por *software*. As fontes de interrupções são o RESET, dois eventos externos, dois blocos de temporização/contagem e um serial UART, e os níveis de prioridade obedecem à sequência de endereços da Tabela 3.1.

Tabela 3.1– Endereço e Ordem de prioridade
das Interrupções na Memória ROM.

Endereço das Interrupções - ROM		
Prioridade	**Endereço**	**Tipo**
	0x0000	*RESET*
1°	0x0003	Interrupção 0
2°	0x000B	Temporizador 0
3°	0x0013	Interrupção 1
4°	0x001B	Temporizador 1
5°	0x0023	Serial 1

O 8051 contêm a possibilidade de utilizar um oscilador interno ou um oscilador externo com freqüência máxima de 12 MHz, e conta com um conjunto de instruções do tipo Computador com um Conjunto Complexo de Instruções – CISC (*Complex Instruction Set Computer*) de 111 tipos. Considerando as variações de cada tipo totaliza-se 255 instruções em que os códigos de operação estão entre 00_h e FF_h, excluindo $A5_h$, pois é um código reservado.

3.2 Arquitetura do microcontrolador 8051

3.2.1 Organização das Memórias

O 8051 possui locações de endereços separados para a memória de programa e para a memória de dados.

A memória de programa pode ser expandida até 64 kbytes. Internamente possui uma memória de 4 kbytes e as possibilidades de utilização das memórias são apresentadas na Figura 3.2.

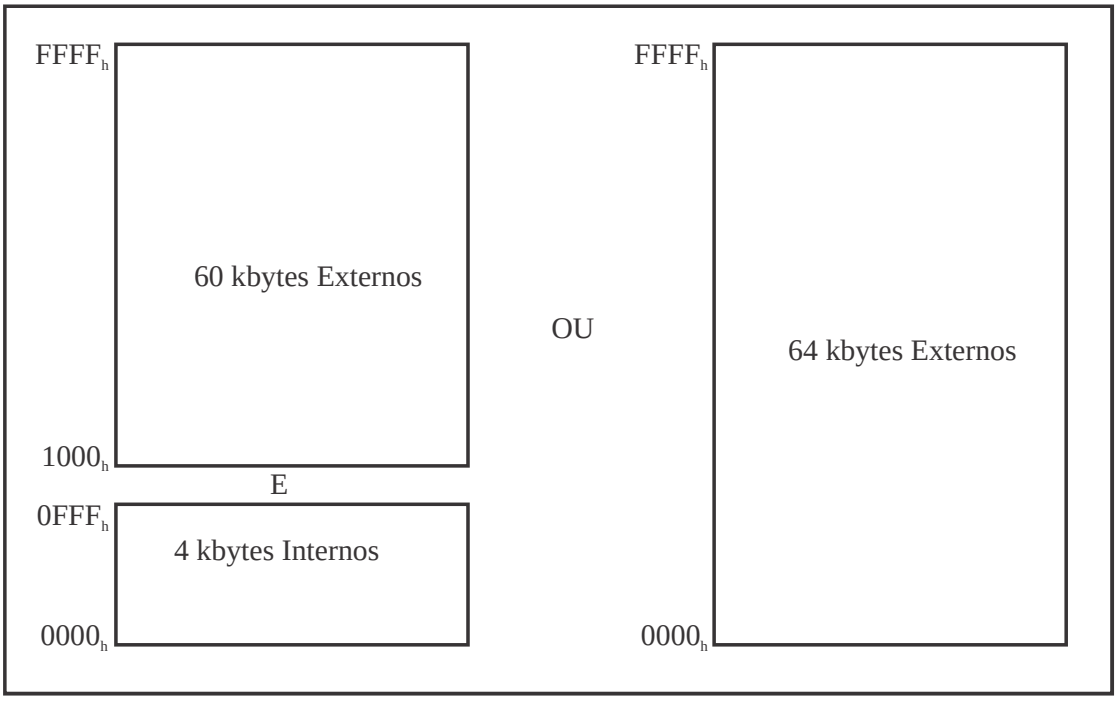

Figura 3.2 - Organização da memória de programa [9].

O 8051 pode endereçar até 64 kbytes de memória de dados externa. A instrução MOVX é utilizada para o acesso da memória externa e o microcontrolador possui internamente 128 bytes de RAM e mais 128 bytes de Registros de Funções Especiais – SFR (*Special Function Registers*). Os 128 bytes iniciais podem ser acessados tanto por um endereçamento direto (MOV dado, endereço) ou por um endereçamento indireto (MOV @Ri). A Figura 3.3 apresenta a organização da memória de dados.

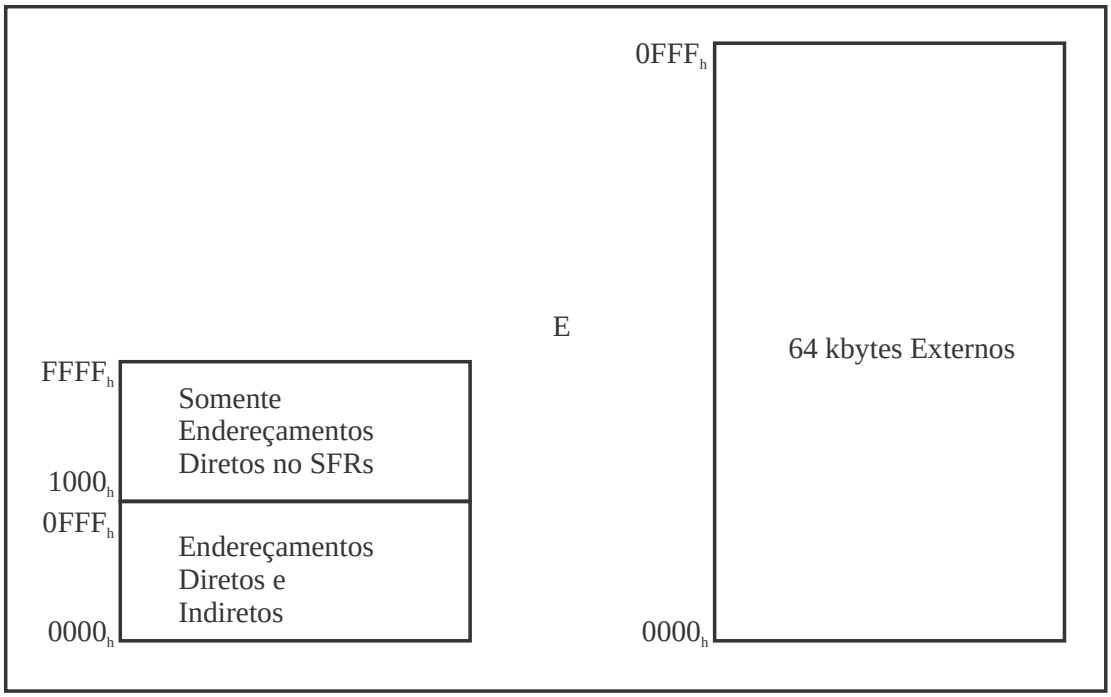

Figura 3.3 – Organização da memória de dados [9].

Os 128 bytes da RAM que podem ser acessados por ambos os endereçamentos, direto e indireto, podem ser divididos em 3 segmentos:

1. Bancos de Registros 0 até 3: posição de 00_h até $1F_h$ (32 bytes). Após o *RESET*, o banco de registro padrão será o banco 0. Para utilizar os demais bancos do 8051 deve-se alterar, via *software*, os valores do registro Palavra do Estado do Programa - PSW (*Program Status Word*). Cada banco de registro contém oito registros de um byte.

2. Área de endereçamento por bit: composta por 16 bytes, sendo da posição 20_h até $2F_h$. Cada um dos 128 bits deste segmento pode ser endereçado diretamente, bit a bit, ou por byte.

3. Área para armazenamento temporário: os bytes 30_h até $7F_h$ estão disponíveis para o usuário como dados da RAM. Entretanto, se o apontador de pilha for inicializado nesta área, um número suficiente de bytes deve ser reservado para evitar a perda do dado do apontador de pilha.

3.2.2 Registro de Funções Especiais

A Tabela 3.2 apresenta uma lista de todos os registros de funções especiais e seus respectivos endereços. Os SFRs que são bits endereçáveis são apresentados com um * na frente do nome.

Tabela 3.2 – Registros de Funções Especiais [9].

Símbolo	Nome	Endereço
*ACC	Acumulador	$E0_h$
*B	Registrador B	$F0_h$
*PSW	Palavra do Estado do Programa	$D0_h$
SP	Apontador de Pilha	81_h
DPTR	Apontador de dado 2 bytes	
DPL	Byte Inferior	82_h
DPH	Byte Superior	83_h
*P0	Porta 0	80_h
*P1	Porta 1	90_h
*P2	Porta 2	$A0_h$
*P3	Porta 3	$B0_h$
*IP	Controle de Prioridade de Interrupção	$B8_h$
*IE	Controle de Habilitação de Interrupção	$A8_h$
TMOD	Modo de Controle do Temporizador/Contador	89_h
*TCON	Controle do Temporizador/Contador	88_h
TH0	Byte Superior do Temporizador/Contador 0	$8C_h$
TL0	Byte Inferior do Temporizador/Contador 0	$8A_h$
TH1	Byte Superior do Temporizador/Contador 1	$8D_h$
TL1	Byte Inferior do Temporizador/Contador 1	$8B_h$
*SCON	Controle da Serial	98_h
SBUF	Armazenamento de Dado Serial	99_h
PCON	Controle de Energia	87_h

Os valores dos SFR após o *Power-On Reset* - POR ou de um *RESET* são apresentados na Tabela 3.3.

Tabela 3.3 – Valores do SFRs após RESET [9].

Registro	Valor em Binário
*ACC	00000000
*B	00000000
*PSW	00000000
SP	00000111
DPTR	
DPL	00000000
DPH	00000000
*P0	11111111
*P1	11111111
*P2	11111111
*P3	11111111
*IP	00000000
*IE	00000000
TMOD	00000000
*TCON	00000000
TH0	00000000
TL0	00000000
TH1	00000000
TL1	00000000
*SCON	00000000
SBUF	Indeterminado
PCON	HMOS 0XXXXXXX
	CHMOS 0XXX0000

3.2.3 Instruções

O microcontrolador 8051 possui diversas instruções, sendo cada uma representada por seu mnemônico e por um código em hexadecimal. A nomenclatura apresentada na Tabela 3.4 é adotada para exemplificar o conjunto das instruções:

Tabela 3.4 – Modo de endereçamento [9].

Rn	pode ser registradores de R0 a R7
Ri	indica registrador R0 ou R1
@Ri	endereçado pelo valor de R0 ou R1
#Dado	valor constante, numeral de 8 bits: #20H(hex), #30(dec), #01010101B(bin)
#Dado16	valor constante, numeral de 16 bits: #1FF2H(hex)
Direto	um endereço de memória RAM interna (8 bits), registradores de status e controles, e portas
End16	endereço de 16 bits para ROM (usado por LCALL e LJMP)
End11	endereço de 11 bits para ROM (usado por ACALL e AJMP)
rel	endereço relativo ou utilização de label
bit	variável da RAM interna, bits de I/O, bits de status e controle

A Tabela 3.5 contém os mnemônicos e as descrições de todos os 111 tipos de instruções.

Tabela 3.5 – Conjunto de instruções do 8051 [9].

Operações Aritméticas	
Mnemônico	**Descrição**
ADD A,Rn	Soma o conteúdo de Rn com ACC, o resultado da soma é gravado no ACC.
ADD A, direto	Soma o conteúdo da posição de memória com ACC, o resultado é gravado no ACC.
ADD A, @Ri	Soma o conteúdo da posição de memória indicado por Ri (R1 ou R0) com ACC, gravando o resultado no ACC.
ADD A, # dado	Soma o dado ao ACC. O resultado é gravado no ACC.
ADDC A, Rn	Soma o conteúdo de Rn (R0 à R7) ao ACC, e com a flag carry, o resultado é gravado no ACC.
ADDC A, direto	Soma o conteúdo da posição de memória com o ACC, e com flag carry, o resultado é gravado no ACC.
ADDC A, @Ri	Soma o conteúdo da posição de memória indicado por Ri (R1 ou R0) com o ACC, e com a flag carry, o resultado é gravado no ACC.
ADDC A, # dado	Soma o dado ao ACC, e a flag carry, o resultado é gravado no ACC.
SUBB A, Rn	Subtrai do ACC, o conteúdo de Rn (R0 à R7) e o ‘vem um’ (borrow), o resultado é gravado no ACC.
SUBB A, direto	Subtrai do ACC. o conteúdo da posição de memória e o ‘vem um’ (borrow), o resultado é gravado no ACC

SUBB A, @Ri	Subtrai do ACC, o conteúdo da posição de memória indicado por Ri (R1 ou R0) e o vem um (se existir) o resultado é gravado no ACC.
SUBB A, # dado	Subtrai do ACC, o dado e o 'vem um' (se existir), o resultado é gravado no ACC.
INC A	Soma 1 ao ACC.
INC Rn	Soma 1 ao conteúdo de Rn (R0 a R7).
INC direto	Soma 1 ao conteúdo da posição de memória.
INC @Ri	Soma 1 ao conteúdo de memória indicado por Ri (R0 ou R1).
DEC A	Subtrai 1 do ACC.
DEC Rn	Subtrai 1 do conteúdo de Rn (R0 a R7).
DEC direto	Subtrai 1 do conteúdo da posição de memória.
DEC @Ri	Subtrai 1 do conteúdo da posição de memória indicado por Ri (R0 ou R1).
INC DPTR	Soma 1 ao registrador DPTR.
MUL AB	Multiplica o conteúdo do ACC pelo conteúdo do registrador B. O resultado fica em B (MSB) e ACC (LSB)
DIV AB	Divide o conteúdo do ACC pelo conteúdo do registrador B. O resultado fica em A e o resto em B.
DA A	Converte em BCD, o conteúdo do ACC.
Operações Lógicas	
ANL A, Rn	Grava no acumulador o resultado da operação lógica AND entre o acumulador e registrador Rn
ANL A, direto	Grava no acumulador o resultado da operação lógica AND entre o acumulador e conteúdo do endereço "direto"
ANL A, @Ri	Grava no acumulador o resultado da operação lógica AND entre o acumulador e o conteúdo endereçado pelo registrador Ri
ANL A, #dado	Grava no acumulador o resultado da operação lógica AND entre o acumulador e o dado
ANL direto, A	Grava no endereço "direto" o resultado da operação lógica AND entre o endereço "direto" e o acumulador
ANL direto,#dado	Grava no endereço "direto" o resultado da operação lógica AND entre o endereço "direto" e o dado
ORL A, Rn	Grava no acumulador o resultado da operação lógica OR entre o acumulador e registrador Rn
ORL A, direto	Grava no acumulador o resultado da operação lógica OR entre o acumulador e conteúdo do endereço "direto
ORL A, @Ri	Grava no acumulador o resultado da operação lógica OR entre o acumulador e o conteúdo endereçado pelo registrador Ri
ORL A, #dado	Grava no acumulador o resultado da operação lógica OR entre o acumulador e dado
ORL direto, A	Grava no endereço "direto" o resultado da operação lógica OR entre o endereço "direto" e o acumulador
ORL direto,#dado	Grava no endereço "direto" o resultado da

	operação lógica OR entre o endereço “direto” e o dado
XRL A, Rn	Grava no acumulador o resultado da operação lógica XOR entre o acumulador e registrador Rn
XRL A, direto	Grava no acumulador o resultado da operação lógica XOR entre o acumulador e conteúdo do endereço “direto”
XRL A, @Ri	Grava no acumulador o resultado da operação lógica XOR entre o acumulador e o conteúdo endereçado pelo registrador Ri
XRL A, #dado	Grava no acumulador o resultado da operação lógica XOR entre o acumulador e o dado
XRL direto, A	Grava no endereço “direto” o resultado da operação lógica XOR entre o endereço “direto” e o acumulador
XRL direto,#dado	Grava no endereço “direto” o resultado da operação lógica XOR entre o endereço “direto” e o dado
CLR A	Zera o acumulador
CPL A	Inverte todos os bits do acumulador
RL A	Rotaciona todos os bits do acumulador para esquerda
RLC A	Rotaciona todos os bits do acumulador para esquerda junto com a flag carry
RR A	Rotaciona todos os bits do acumulador para direita
RRC	Rotaciona todos os bits do acumulador para direita junto com a flag carry
SWAP A	Troca os nibbles do acumulador (equivale a 4 instruções RL A)
Transferência de Dados	
MOV A, Rn	Carrega o acumulador com o conteúdo do registrador Rn
MOV A, direto	Carrega o acumulador com o conteúdo do endereço “direto” (endereços dos registradores internos)
MOV A, @Ri	Carrega o acumulador com o conteúdo endereçado pelo registrador Ri
MOV A, #dado	Carrega o acumulador com o dado
MOV Rn, A	Carrega o registrador Rn com o conteúdo do acumulador
MOV Rn, direto	Carrega o registrador Rn com o conteúdo do endereço “direto”
MOV Rn, #dado	Carrega o registrador Rn com o dado
MOV direto, A	Carrega o endereço “direto” com o conteúdo do acumulador
MOV direto, Rn	Carrega o endereço “direto” com o conteúdo do registrador Rn
MOV direto1,direto2	Carrega o endereço “direto1” com o conteúdo endereço2 “direto2”
MOV direto, @Ri	Carrega o endereço “direto” com o conteúdo endereçado pelo registrador Ri
MOV direto, #dado	Carrega o endereço “direto” com o dado
MOV @Ri, A	Carrega o registrador endereçado por Ri com o conteúdo do acumulador
MOV @Ri, direto	Carrega o registrador endereçado por Ri

	com o conteúdo do endereço “direto”
MOV @Ri, dado	Carrega o registrador endereçado por Ri com o dado
MOV DPTR,dado16	Carrega o registrador DPTR com o dado de 16 bits
MOVC A,@A+DPTR	Carrega o acumulador com o conteúdo endereçado pelo acumulador mais o registrador DPTR
MOVC A, @A+PC	Carrega o acumulador com o conteúdo endereçado pelo acumulador mais o registrador PC
MOVX A, @Ri	Carrega o acumulador com o conteúdo da RAM externa endereçado pelo registrador Ri
MOVX A, @DPTR	Carrega o acumulador com o conteúdo da RAM externa endereçado pelo registrador DPTR
MOVX @Ri, A	Carrega o registrador da RAM externa endereçado pelo registrador de 8 bits Ri , com o conteúdo do acumulador
MOVX @DPTR, A	Carrega o registrador da RAM externa endereçado pelo registrador de 16 bits DPTR, com o conteúdo do acumulador
PUSH direto	Incrementa o registrador SP e salva na pilha o conteúdo do endereço “direto”
POP direto	Grava na memória o conteúdo da pilha e decrementa o registrador SP
XCH A, Rn	Troca os dados do acumulador com o registrador Rn
XCH A, direto	Troca os dados do acumulador com o endereço “direto”
XCH A, @Ri	troca os dados do acumulador com o registrador endereçado por Rn
XCHD A, @Ri	Troca os nibbles menos significativos do acumulador com o registrador endereçado por Ri
Variáveis Booleanas	
CLR C	Grava 0 no flag carry
CLR Bit	Grava 0 no bit endereçável (bit de pino ou bit de registradores)
SETB C	grava 1 no flag carry
SETB Bit	Grava 1 no bit endereçável (bit de pino ou bit de registradores)
CPL C	Complementa o flag carry
CPL Bit	Complementa o bit endereçável (bit de pino ou bit de registradores)
ANL C, Bit	Grava no flag carry o resultado da operação lógica AND entre o flag carry e o bit endereçável (bit de pino ou bit de registradores)
ANL C, \Bit	Grava no flag carry o resultado da operação lógica AND entre o flag carry e o complemento do bit endereçável (bit de pino ou bit de registradores).
ORL C, Bit	Grava no flag carry o resultado da operação lógica OR entre o flag carry e o bit endereçável (bit de pino ou bit de registradores)
ORL C, \Bit	Grava no flag carry o resultado da operação lógica OR entre o flag carry e o complemento do bit endereçável (bit de pino ou

	bit de registradores)
MOV C, Bit	Grava no flag carry o conteúdo do bit endereçável (bit de pino ou bit de registradores)
MOV Bit, C	Grava no bit endereçável (bit de pino ou bit de registradores) o conteúdo do flag carry
Fluxo de Execução	
ACALL End 11	Chamada curta de sub-rotina (11 bits, 2 kbytes da posição atual)
LCALL End 16	Chamada longa de sub-rotina (16 bits , qualquer posição da EPROM)
RET	Retorno de sub-rotina
RETI	Retorno de sub-rotina de interrupção
AJMP End 11	Desvio curto para endereço de 11 bits, 2 kbytes da posição atual
LJMP End 16	Desvio longo para endereço de 16 bits, qualquer posição da EPROM
SJMP rel	Desvio relativo curto
JMP @A+DPTR	Desvio indireto para a posição de memória endereçada pelo acumulador mais o registrador DPTR
JZ rel	Desvia para o endereço “rel” se o acumulador for igual a zero
JNZ rel	Desvia para o endereço “rel” se o acumulador for diferente de zero
JC rel	Desvia para o endereço “rel” se o flag carry estiver setado
JNC rel	Desvia para o endereço “rel” se o flag carry estiver zerado
JB bit, rel	Desvia para o endereço “rel” se o bit endereçável estiver setado
JNB bit, rel	Desvia para o endereço “rel” se o bit endereçável estiver zerado
JBC bit, rel	Desvia para o endereço “rel” se o bit endereçável estiver setado e depois zera o bit
CJNE A,direto,rel	Compara o conteúdo do acumulador com o conteúdo do endereço “direto”, e desvia para o endereço “rel” se for diferente
CJNE A,#dado,rel	Compara o conteúdo do acumulador com o dado, e desvia para o endereço “rel” se for diferente
CJNE Rn,#dado,rel	Compare o conteúdo do registrador Rn com o dado, e desvie para o endereço “rel” se for diferente
CJNE @Ri,#dado,rel	Compara o conteúdo do endereçado por Ri com o dado, e desvie para o endereço “rel” se for diferente
DJNZ Rn, rel	Decrementa o registrador Rn e desvia para o endereço “rel” se o resultado for diferente de zero
DJNZ direto, rel	Decrementa o endereço “direto” e desvia para o endereço “rel” se o resultado for diferente de zero
NOP	Não faz nada.

3.2.4 Pinos

O 8051 possui 40 pinos alocados conforme a Figura 3.4, a Tabela 3.6 apresenta as funcionalidades de cada um.

Tabela 3.6 – Funcionalidades dos pinos do 8051.

PINO	FUNÇÃO
1	Entrada ou saída da porta 1.
2	Entrada ou saída da porta 1.
3	Entrada ou saída da porta 1.
4	Entrada ou saída da porta 1.
5	Entrada ou saída da porta 1.
6	Entrada ou saída da porta 1.
7	Entrada ou saída da porta 1.
8	Entrada ou saída da porta 1.
9	Entrada do sistema de reinicialização do 8051.
10	Entrada ou saída da porta 3 ou entrada da UART.
11	Entrada ou saída da porta 3 ou saída da UART.
12	Entrada ou saída da porta 3 ou entrada da interrupção externa.
13	Entrada ou saída da porta 3 ou entrada da interrupção externa.
14	Entrada ou saída da porta 3 ou entrada do contador de eventos.
15	Entrada ou saída da porta 3 ou entrada do contador de eventos.
16	Entrada ou saída da porta 3 ou sinalizador de escrita de dados.
17	Entrada ou saída da porta 3 ou sinalizador de leitura de dados.
18	Entrada para o cristal do oscilador interno.
19	Entrada para o cristal do oscilador interno.
20	GND
21	Entrada ou saída da porta 2 ou entrada do endereço no processo de gravação.
PINO	**FUNÇÃO**
22	Entrada ou saída da porta 2 ou entrada do endereço no processo de gravação.
23	Entrada ou saída da porta 2 ou entrada do endereço no processo de gravação.
24	Entrada ou saída da porta 2 ou entrada do endereço no processo de gravação.
25	Entrada ou saída da porta 2 ou entrada do endereço no processo de gravação.
26	Entrada ou saída da porta 2 ou entrada do endereço no processo de gravação.
27	Entrada ou saída da porta 2 ou entrada do endereço no processo de gravação.
28	Entrada ou saída da porta 2 ou entrada do endereço no processo de gravação.
29	Saída para habilitação do programa externo.
30	Saída habilitadora do *latch* de endereços.
31	Entrada de seleção de memórias.
32	Entrada ou saída da porta 0 ou entrada do endereço ou do dado no processo de gravação.
33	Entrada ou saída da porta 0 ou entrada do endereço ou do dado no processo de gravação.
34	Entrada ou saída da porta 0 ou entrada do endereço ou do dado no processo de gravação.

PINO	FUNÇÃO
35	Entrada ou saída da porta 0 ou entrada do endereço ou do dado no processo de gravação.
36	Entrada ou saída da porta 0 ou entrada do endereço ou do dado no processo de gravação.
37	Entrada ou saída da porta 0 ou entrada do endereço ou do dado no processo de gravação.
38	Entrada ou saída da porta 0 ou entrada do endereço ou do dado no processo de gravação.
39	Entrada ou saída da porta 0 ou entrada do endereço ou do dado no processo de gravação.
40	VCC

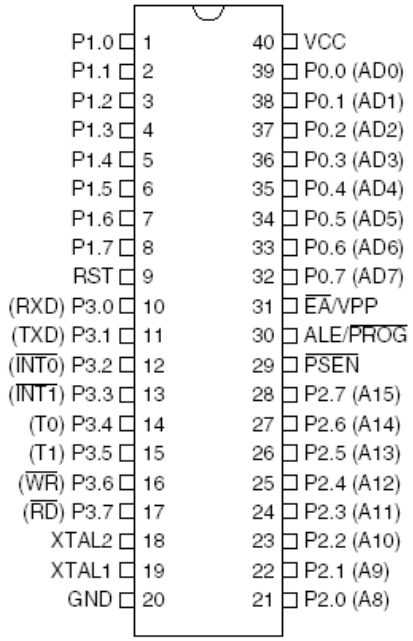

Figura 3.4 – Pinos do microcontrolador 8051.

Para informações mais detalhadas, sobre o microcontrolador, consultar a referência [9].

Capítulo 4

4 Estrutura do projeto em VHDL

4.1 Introdução

Esse capítulo apresenta todos os blocos que fazem parte da estrutura do microcontrolador M8051, seus sinais de entrada e de saída, as operações realizadas, as representações das estruturas geradas pela síntese e os resultados das simulações para comprovação do funcionamento correto através de *waveforms* ou *testbenchs*.

Essa etapa é de extrema importância para que a lógica desenvolvida para a criação do microcontrolador seja prototipada em FPGA e posteriormente possa ser obtido o *layout* do circuito integrado.

4.2 Estrutura em blocos do microcontrolador

Para a criação do M8051 foi elaborado a estrutura de blocos apresentado na Figura 4.1. Ela é composta por vários níveis hierárquicos com todos os blocos do projeto e as interligações.

Os blocos pertencentes ao Microcontrolador M8051 são a memória de programa ROM, a memória de dados interna RAM, a memória de dados externa RAM e o núcleo do microcontrolador. Esse núcleo contém a interface de comunicação serial UART, a interface de comunicação serial I^2C, os temporizadores/contadores, a unidade de criptografia AES128, a unidade de controle e a Unidade Lógica Aritmética – ULA.

A ULA contém todos os blocos responsáveis pelas operações lógicas e aritméticas do microcontrolador. Essas operações foram baseadas no conjunto de instruções MCS-51 da Intel [9].

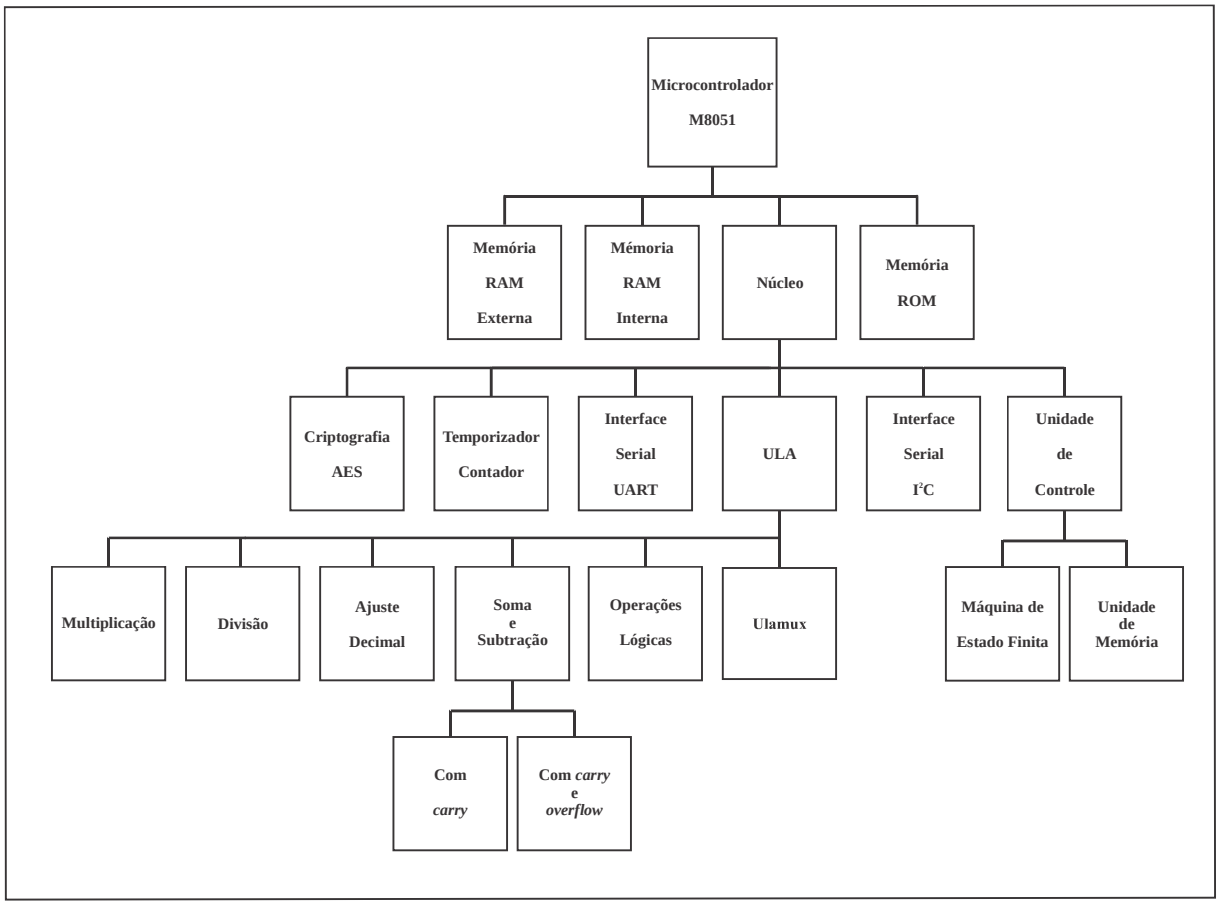

Figura 4.1 - Estrutura em blocos dos níveis hierárquico do Microcontrolador M8051.

Os detalhes de cada bloco serão apresentados, iniciando pelo nível mais baixo da hierarquia até o mais alto, e com isso, serão abrangidas todas as estruturas, sinais e lógicas envolvidas na concretização dessa etapa de desenvolvimento e validação do projeto.

A quantidade de elementos lógicos utilizados, a frequência máxima de operação e entre outras características de cada bloco não será apresentada individualmente e, sim, os valores de todo o conjunto, pois o objetivo principal do projeto é desenvolver uma estrutura funcional para que posteriormente, em trabalhos futuros, sejam realizadas otimizações nos circuitos.

4.2.1 Bloco de Adição e Subtração com *carry/borrow*

Este bloco realiza as operações aritméticas de soma e subtração com dados de entrada de 4 bits (*nibble)* e sinal de entrada de *carry/borrow*. Como resultado, apresenta um *nibble* e a linha de *carry/borrow*. A Figura 4.2 apresenta uma representação do bloco resultante da síntese do código desenvolvido.

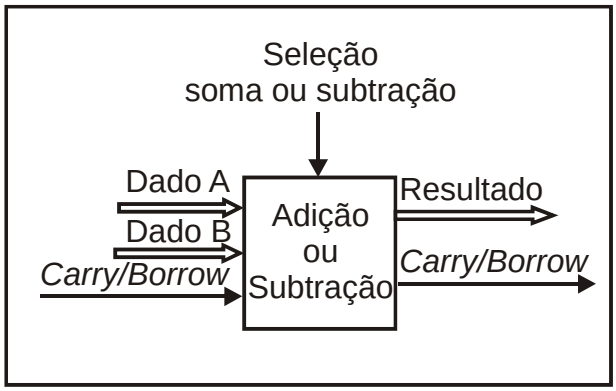

Figura 4.2 – Representação do bloco de adição e subtração com *carry/borrow*.

A Figura 4.3 apresenta os resultados da simulação das operações soma e subtração, com e sem a consideração de *carry/borrow* e com ela pode-se comprovar o funcionamento correto da lógica da estrutura. Para exemplificar, consideram-se as operações em binário destacadas na Figura 4.3 pelos retângulos pontilhados em que se tem dado de entrada A igual a 1111_b, dado de entrada B igual 1111_b e *carry/borrow* de entrada igual a 1. O bloco irá realizar a soma caso o selecionador de soma ou de subtração esteja em 1 e a subtração caso o selecionador esteja em 0. Para a soma tem-se como resultado 1111_b e *carry/borrow* igual a 1 e para a subtração tem-se como resultado 1111_b e *carry/borrow* igual a 1. Este bloco realiza a operação do *nibble* menos significativo do bloco de adição e subtração de 8 bits.

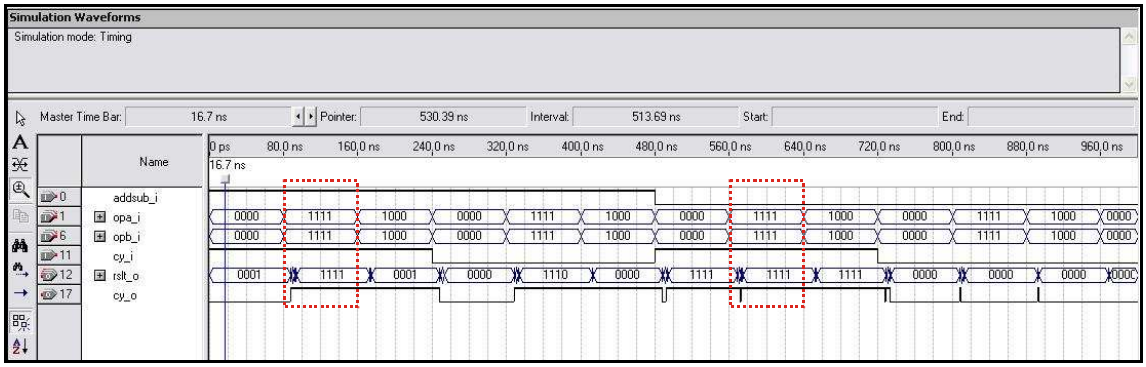

Figura 4.3 - Resultado das operações de soma e subtração com e sem *carry* de entrada.

4.2.2 Bloco de Adição e Subtração com *carry/borrow* e *overflow*

Este bloco realiza as operações aritméticas de soma ou de subtração com dados de entrada de 4 bits e sinal de *carry/borrow*. Como resultado apresenta uma palavra de 4 bits e linha de *carry/borrow* e *overflow*. A Figura 4.4 ilustra a representação do bloco resultado da síntese do código desenvolvido.

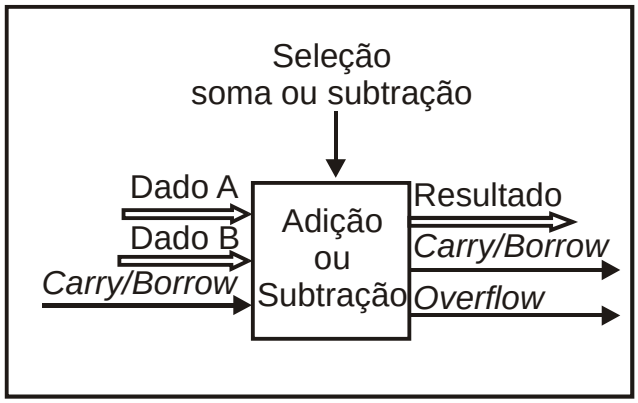

Figura 4.4 – Representação do bloco de adição e subtração com *carry/borrow* e *overflow*.

A Figura 4.5 apresenta os resultados da simulação das operações soma e subtração com e sem a consideração de *carry/borrow* de entrada e *overflow* e com ela pode-se comprovar o funcionamento correto da lógica da estrutura. Para exemplificar as operações consideram-se as operações em binário destacadas na Figura 4.5 pelos retângulos pontilhados em que se tem dado de entrada A igual a 1111_b, dado de entrada B igual 1111_b e *carry/borrow* de entrada igual a 0 para a operação soma. Para a operação de subtração tem-se dado de entrada A igual a 1000_b, dado de entrada B igual 1000_b e *carry/borrow* de entrada igual a 1. O bloco irá realizar a soma caso o selecionador de soma ou subtração esteja em 1 e a subtração caso o selecionador esteja em 0. Para a soma tem-se como resultado 1110_b, *carry/borrow* igual a 1 e *overflow* igual a 0 e para a subtração tem-se como resultado 1111_b, *carry/borrow* igual a 1 e *overflow* igual a 0. Este bloco realiza a operação do *nibble* mais significativo do bloco de adição e subtração de 8 bits.

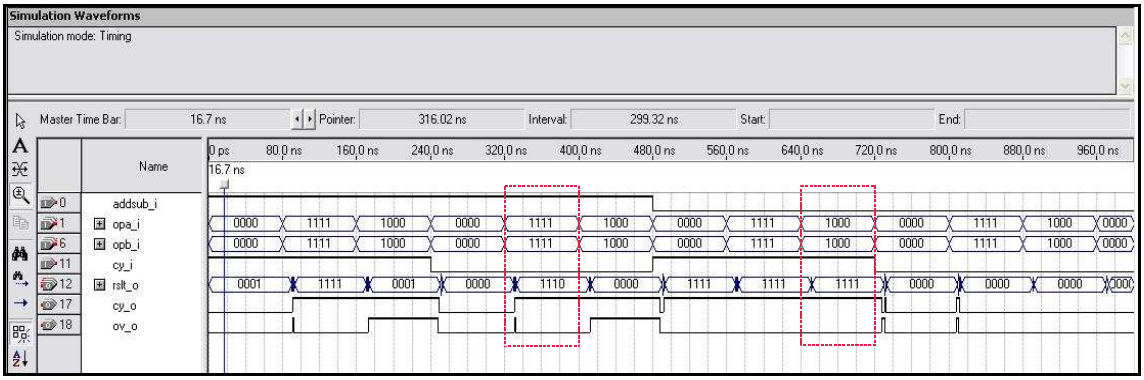

Figura 4.5 - Resultado das operações de soma e subtração
com e sem *carry* de entrada e considerando *overflow*.

4.2.3 Bloco de Adição ou Subtração

Este bloco realiza as operações aritméticas de soma e de subtração de 8 bits considerando *carry/borrow*. Como resultado apresenta uma palavra de 8 bits e sinais de *carry/borrow* e *overflow*. A Figura 4.6 mostra a representação do bloco resultado da síntese do código desenvolvido.

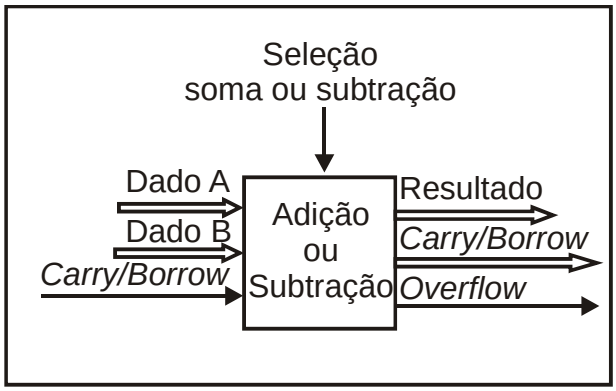

Figura 4.6 – Representação do bloco de adição e subtração com *carry/borrow* e *overflow*.

Ele realiza esse processamento utilizando os dois blocos apresentados anteriormente, o bloco de adição e de subtração com *carry/borrow* e o bloco de adição e de subtração com *carry/borrow* e *overflow*, sendo realizadas as operações em *nibbles*, portanto, o primeiro bloco para a parte menos significativa do dado de 8 bits e o segundo bloco para a mais significativa. As operações são realizadas em dois *nibbles* para que nas operações em BCD seja possível a identificação do *carry/borrow* da parte menos significativa a qual será sinalizada no bit 6 do registro PSW.

A Figura 4.7 apresenta os resultados da simulação das operações soma e subtração com e sem *carry/borrow* de entrada e *overflow*. Pode-se assim comprovar o funcionamento correto da lógica da estrutura. Para exemplificar consideram-se as operações em hexadecimal destacadas na Figura 4.7 pelos retângulos pontilhados em que se tem dado de entrada A igual às 80_h, dado de entrada B igual $B5_h$ e *carry/borrow* de entrada igual a 1 para a soma. Para a subtração tem-se dado de entrada A igual a BC_h, dado de entrada B igual a 01_h e *carry/borrow* de entrada igual a 0. O bloco irá realizar a soma caso o selecionador de soma ou subtração esteja em 1 e a subtração caso o selecionador esteja em 0. Para a operação de soma tem-se como resultado 36_h, *carry/borrow* igual a 10_b (o bit mais significativo representa o *carry/borrow* do *nibble mais* significativo e o bit menos significativo representa o *carry/borrow* do *nibble* menos significativo do dado de 8 bits) e *overflow* igual a 1. Para a

operação de subtração tem-se como resultado BB_h, *carry/borrow* igual a 00_b e *overflow* igual a 0_b.

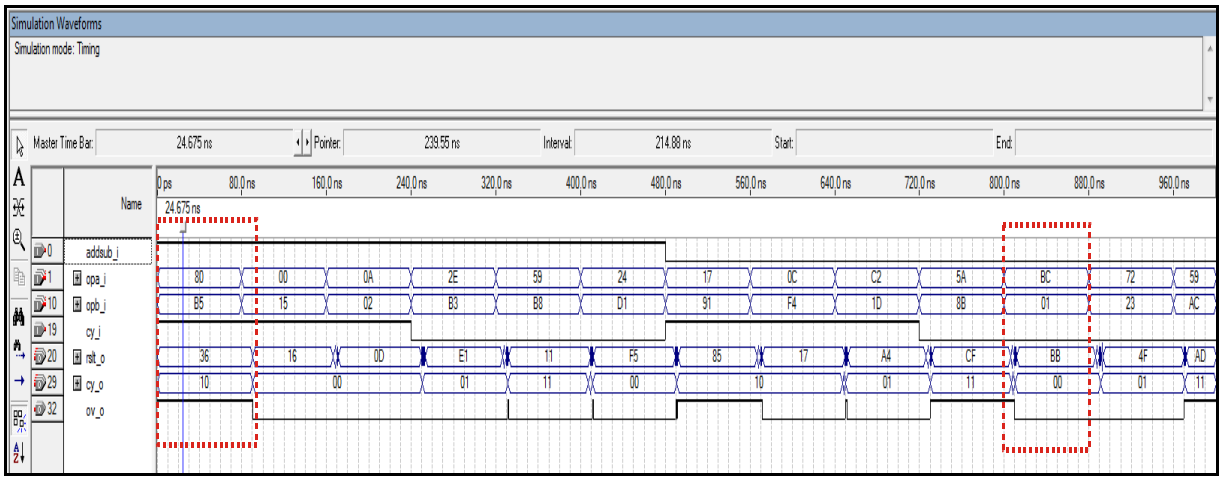

Figura 4.7- Resultado das operações de soma e subtração com e sem *carry* de entrada e considerando *overflow*.

4.2.4 Bloco Multiplicação

Este bloco realiza a operação de multiplicação entre dois operando de 8 bits através de lógica combinacional e apresenta como resultado uma palavra de 16 bits. Na Figura 4.8 encontra-se a representação do resultado da síntese do código desenvolvido.

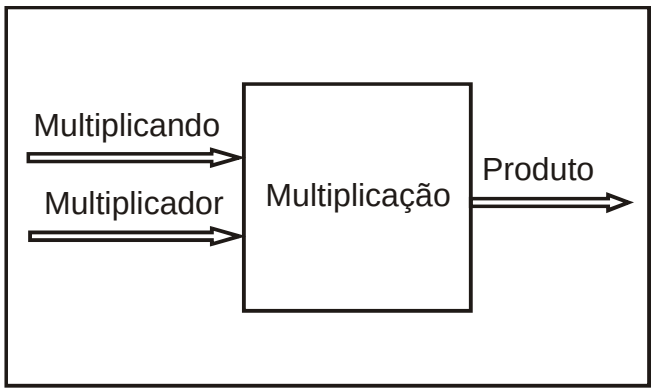

Figura 4.8 – Representação do bloco de multiplicação.

A Figura 4.9 apresenta o resultado da multiplicação que valida o funcionamento correto da lógica da estrutura. Para exemplificar considera-se a multiplicação em decimal destacada na Figura 4.9 pelo retângulo pontilhado em que o multiplicando é 25_d e o multiplicador é 15_d obtendo-se o resultado da operação igual a 375_d.

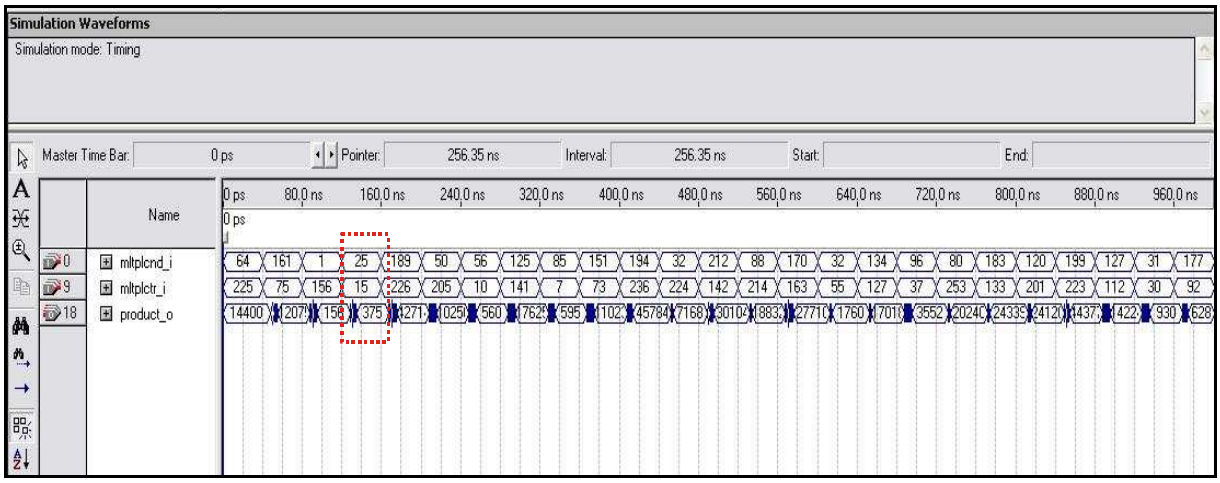

Figura 4.9 - Resultado da operação multiplicação.

4.2.5 Bloco Divisão

Este bloco realiza a operação de divisão entre dois operando de 8 bits através de lógica combinacional e apresenta como resultado duas palavras de 8 bits, o quociente e o resto. A Figura 4.10 apresenta a representação do resultado da síntese do código desenvolvido.

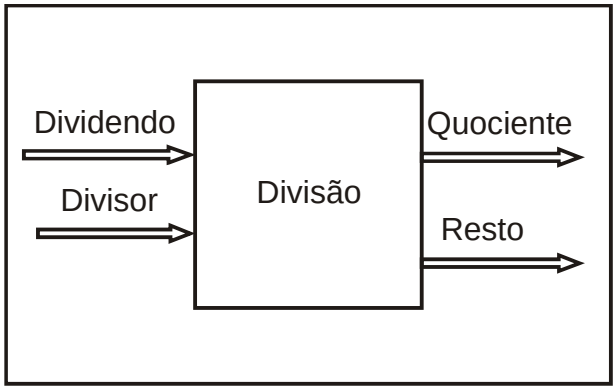

Figura 4.10 – Representação do bloco de divisão.

A Figura 4.11 apresenta os resultados da divisão utilizada para comprovar o funcionamento correto da lógica da estrutura. Para exemplificar considera-se a divisão em decimal destacada na Figura 4.11 pelo retângulo pontilhado em que o dividendo é 104_d e o divisor é 50_d e têm-se o quociente da operação igual a 2_d e o resto igual 4_d.

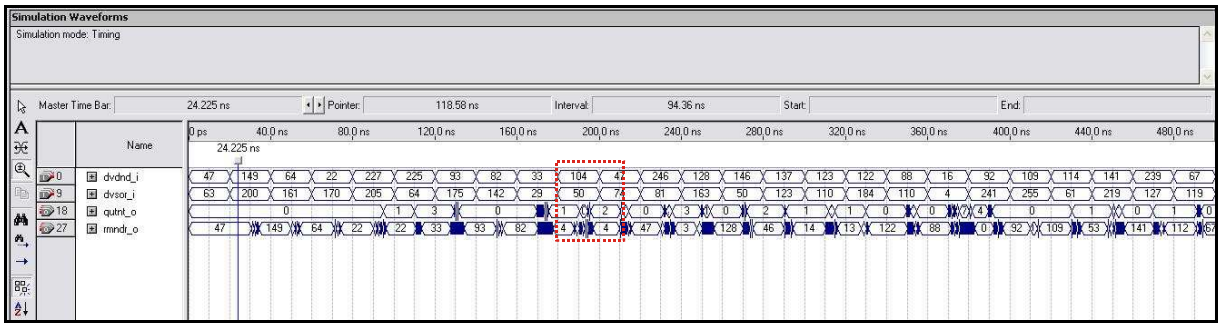

Figura 4.11 - Resultado da operação divisão.

4.2.6 Bloco Operações Lógicas

Este bloco realiza as operações lógicas: *E*, *OU*, *OU Exclusivo*, *Deslocamento para esquerda* (utilizando o operando A), *Deslocamento para direita* (utilizando o operando A), *Deslocamento para esquerda com carry* (utilizando o operando A), *Deslocamento para direita com carry* (utilizando o operando A), *Comparação*, *Complemento de 1* (utilizando o operando A). Como dados de entrada têm-se duas palavras de 8 bits e o *carry*, como resultado uma palavra de 8 bits e um novo valor de *carry*. A Figura 4.12 mostra a representação do resultado da síntese do código desenvolvido.

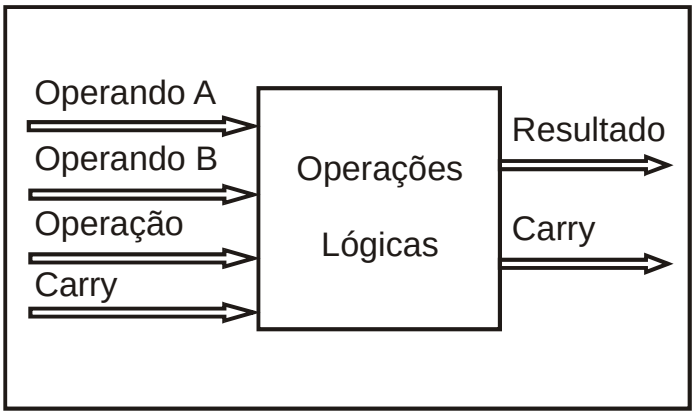

Figura 4.12 – Representação do bloco de Operações Lógicas.

A Figura 4.13 apresenta os resultados das operações lógicas *E*, *OU*, *OU Exclusivo*, *Deslocamento para esquerda*, *Deslocamento para direita*, *Deslocamento para esquerda com carry*, *Deslocamento para direita com carry*, *Comparação*, *Complemento de 1* não considerando o *carry*, respectivamente. Com ela pode-se comprovar o funcionamento correto da lógica da estrutura. Cada operação possui um valor em binário que a representa, sendo *E* = "0011", *OU* = "0101", *OU Exclusivo* = "0110", *Deslocamento para esquerda* = "0111", *Deslocamento para esquerda com carry* = "1000"; *Deslocamento para direita* = "1001",

Deslocamento para direita com carry = "1010", *Comparação* = "1011" e *Complemento de 1* = "1100".

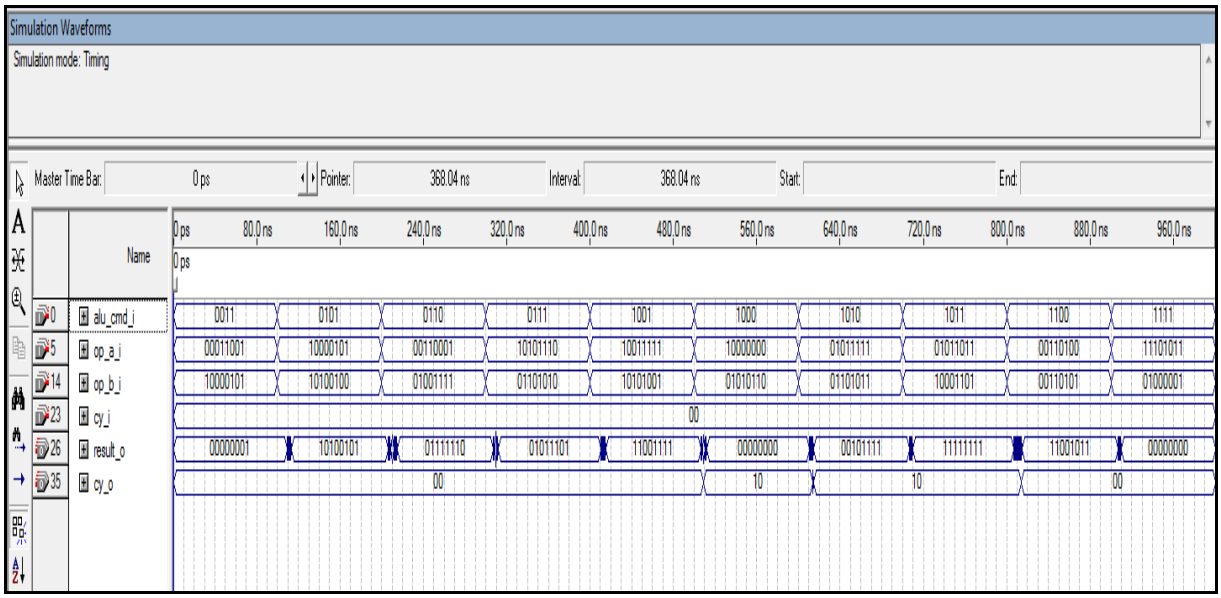

Figura 4.13 - Resultado das operações lógicas.

4.2.7 Bloco Ajuste Decimal

Este bloco realiza o ajuste decimal da representação decimal em BCD em cada *nibble* da palavra de 8 bits, os quais representam a unidade e dezena. A Figura 4.14 é a representação do resultado da síntese do código desenvolvido.

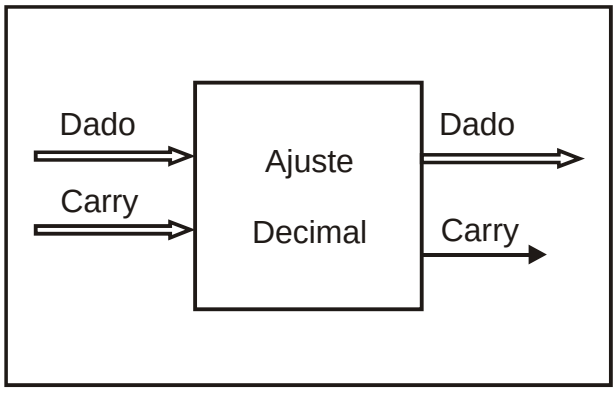

Figura 4.14 – Representação do bloco de Ajuste Decimal.

A Figura 4.15 apresenta os resultados deste ajuste e com ela pode-se comprovar o funcionamento correto da lógica da estrutura. Para exemplificar considera-se o ajuste realizado em hexadecimal destacado na Figura 4.15 pelo retângulo pontilhado em que se tem

como dado de entrada (data_i) igual a $0C_h$, e as posições a serem realizadas o ajuste, com o valor 1_b, são informadas pelo sinal de entrada (cy_i). Neste exemplo, o valor é igual a 01_b, ou seja, o ajuste será realizado apenas no algarismo C_h. Com a operação de ajuste realizada, que consiste em somar o valor seis nas posições indicadas por 1, o valor de resposta obtido é 12_h ($0C_h + 06_h = 12_h$) e o valor do *carry* de saída é 0_b (zero).

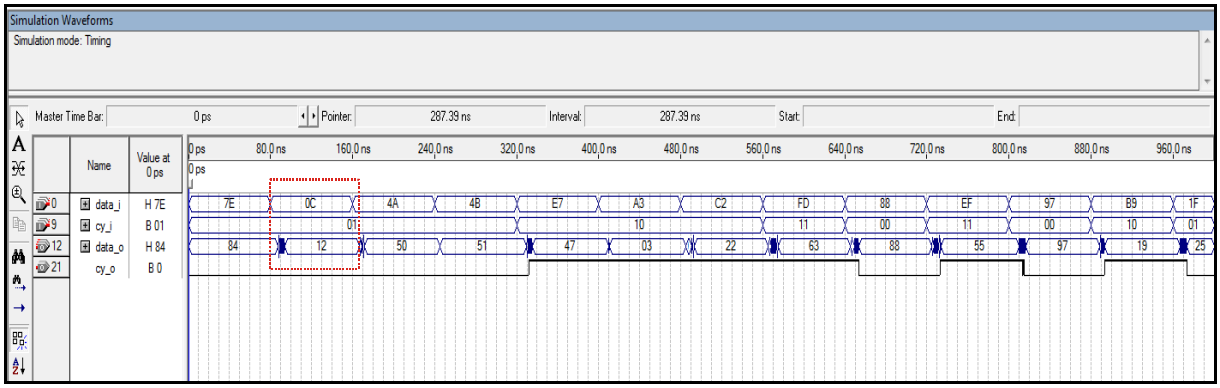

Figura 4.15 - Resultado do ajuste decimal.

4.2.8 Bloco Ulamux

Este bloco realiza a função de transferir os dados de outros blocos do microcontrolador para os respectivos blocos de acordo com a instrução (comando) utilizada, ou seja, é o gerenciador do tráfego das informações dentro da ULA. Todo dado é gerenciado por esse bloco e tanto as informações de entrada quanto as de saída da ULA serão controladas por ele.

4.2.9 Bloco ULA

Esse bloco conecta os blocos Ulamux, Operações Lógicas, Soma e Subtração, Multiplicação, Divisão e Ajuste Decimal, gerando a estrutura completa da ULA e sendo identificados os sinais de entrada, os sinais de saída e os blocos que a compõem. A Figura 4.16 representa o resultado da síntese do código desenvolvido.

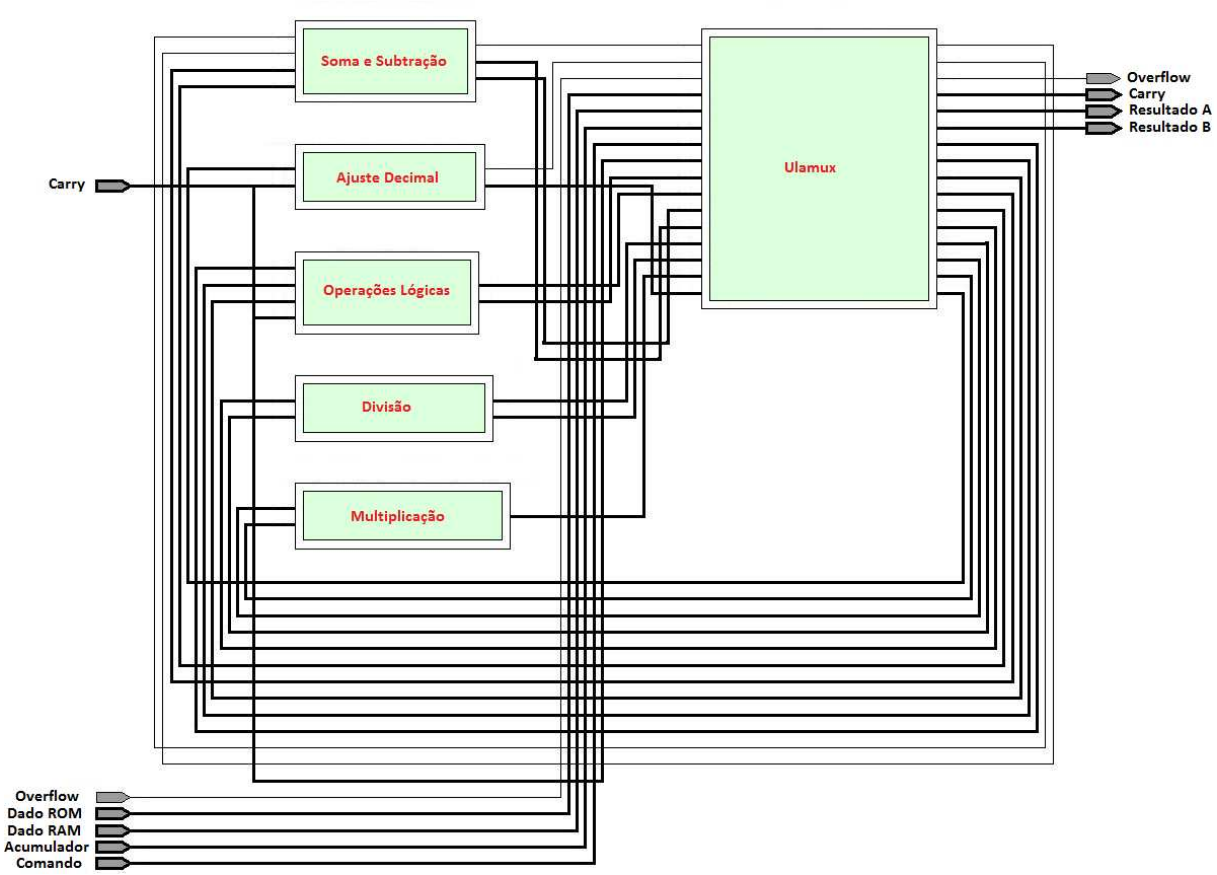

Figura 4.16 – Representação do Bloco ULA baseado no circuito gerado pelo *RTL Viewer* do *software* Quartus II.

Devido ao bloco da ULA apresentar uma estrutura mais complexa que os demais blocos até então apresentados e a complexidade para a geração dos vetores de teste, torna-se inviável a implementação de *waveforms* para a comprovação das funcionalidades do bloco e, por isso, foi gerado um arquivo de *testbench* para este propósito. Este arquivo realiza a análise de todas as operações implementadas com todas as possibilidades de valores de entrada e gera respostas positivas ou negativas após a comparação de cada conjunto de dados obtido após a operação com os valores que deveria ser encontrado. As informações de saída do arquivo de *testbench* e o próprio *testbench* do bloco ULA encontram-se na mídia anexa.

4.2.10 Bloco Temporizador/Contador

Para esse trabalho foi implementado inicialmente o bloco temporizador/contador com as características originais apresentadas na referência [9], porém, pela necessidade de aperfeiçoamento da estrutura o bloco teve sua estrutura ampliada para 4 temporizadores/contadores de 16 bits. Simplesmente, baseou-se nas características dos

temporizadores/contadores e replicaram-se para os dois novos. A Figura 4.17 representa sua estrutura.

Para testar o funcionamento correto da estrutura, o bloco foi verificado individualmente através de *testbench* e as informações de saída do arquivo de *testbench* e o próprio *testbench* do bloco Temporizador/Contador encontram-se na mídia anexa.

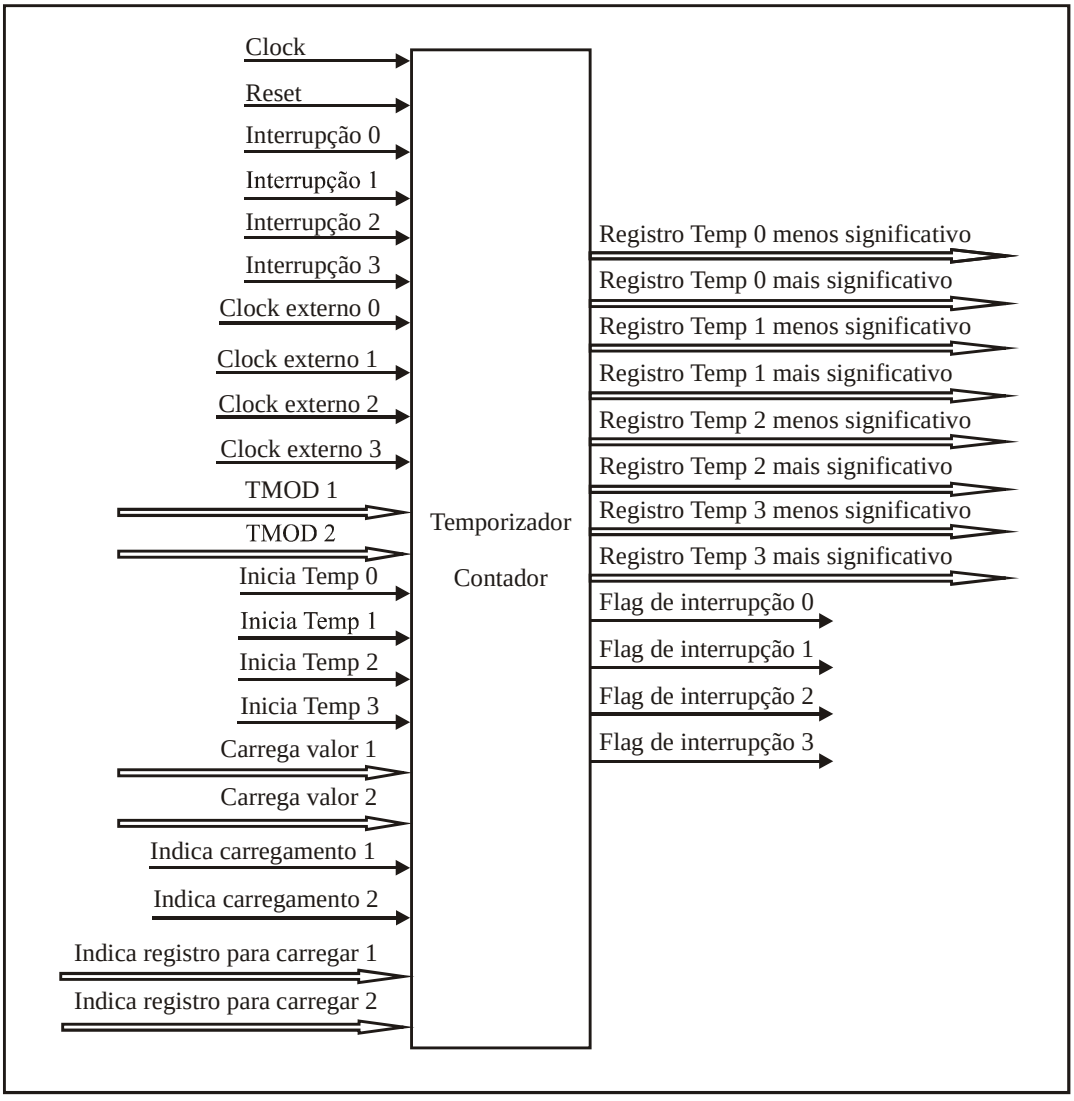

Figura 4.17 – Representação do Bloco Temporizador/Contador.

4.2.11 Bloco Interface Serial UART

A interface serial UART implementada no M8051 possui as mesmas características que o modelo original. Para esse trabalho foi desenvolvido, inicialmente, o bloco interface serial UART com as características originais apresentadas na referência [9], porém, pela necessidade de aperfeiçoamento da estrutura o bloco teve sua estrutura ampliada para duas

interfaces seriais UART com as mesmas características da primeira. A Figura 4.18 apresenta a sua representação.

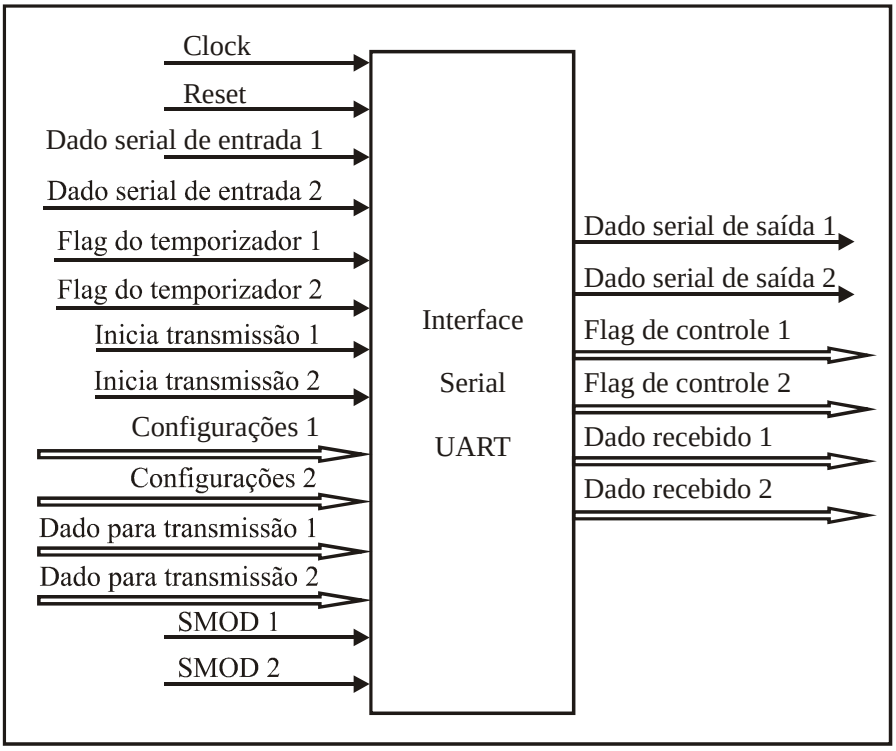

Figura 4.18 – Representação do Bloco Interface Serial UART.

Para verificar o funcionamento correto da estrutura, o bloco foi testado individualmente através de *testbench* e as informações de saída do arquivo de *testbench* e o próprio *testbench* do bloco encontram-se na mídia anexa.

4.2.12 Bloco Unidade de Controle

Esse bloco conecta os blocos Máquina de Estado Finita e Unidade de Memória gerando a estrutura completa do bloco Unidade de Controle. Esse bloco é necessário para melhorar a organização do projeto para a realização das conexões dos diversos sinais envolvidos no sistema através do método de instanciar cada bloco como um componente e, assim, organizar as conexões de todos os sinais.

4.2.13 Bloco Máquina de Estado Finita

Esse bloco é o responsável pela execução de todas as instruções presentes no microcontrolador M8051. As instruções foram baseadas no conjunto de instruções do 8051 da Intel [9], realizando as mesmas ações, porém, com melhorias na arquitetura, com isso elas gastam menos ciclos de *clock* para serem executadas e foram comprovadas através de simulações e análises no *software* ModelSim da quantidade de ciclos de *clock* utilizados por cada instrução. Nesse bloco, também, ocorre o gerenciamento das etapas dos processos de interrupções.

4.2.14 Bloco Unidade de Memória

Esse bloco é o responsável pelo processo de leitura e escrita de dados nos Registradores de Funções Especiais – SFR (*Special Function Registers*), detecção e inicialização dos eventos de interrupção.

4.2.15 Bloco Interface Serial I^2C

Esse bloco será apresentado no Capítulo 5 em que será explicada a lógica de funcionamento da comunicação serial I^2C, a sua integração com a estrutura do M8051, o modo de programação e apresentação dos resultados do arquivo de *testbench* e simulações.

4.2.16 Bloco Criptografia AES 128

Esse bloco será apresentado no Capítulo 6 em que será explicada a lógica de funcionamento da Criptografia AES 128, a sua integração com a estrutura do M8051, o modo de programação e apresentação dos resultados do arquivo de *testbench*.

4.2.17 Bloco Núcleo

Nesse bloco são realizadas apenas as conexões dos sinais dos pinos externos e das memórias com as estruturas apresentadas nos itens anteriores. Esse bloco é necessário para melhorar a organização do projeto para a realização das conexões dos diversos sinais envolvidos no sistema através do método de alocar cada bloco como um componente e, assim,

organizar as conexões de todos os sinais. A Figura 4.19 apresenta os sinais de entrada e de saída que compõem este bloco, os quais são relacionados aos pinos de entrada, saída e as memórias contidas na estrutura do microcontrolador.

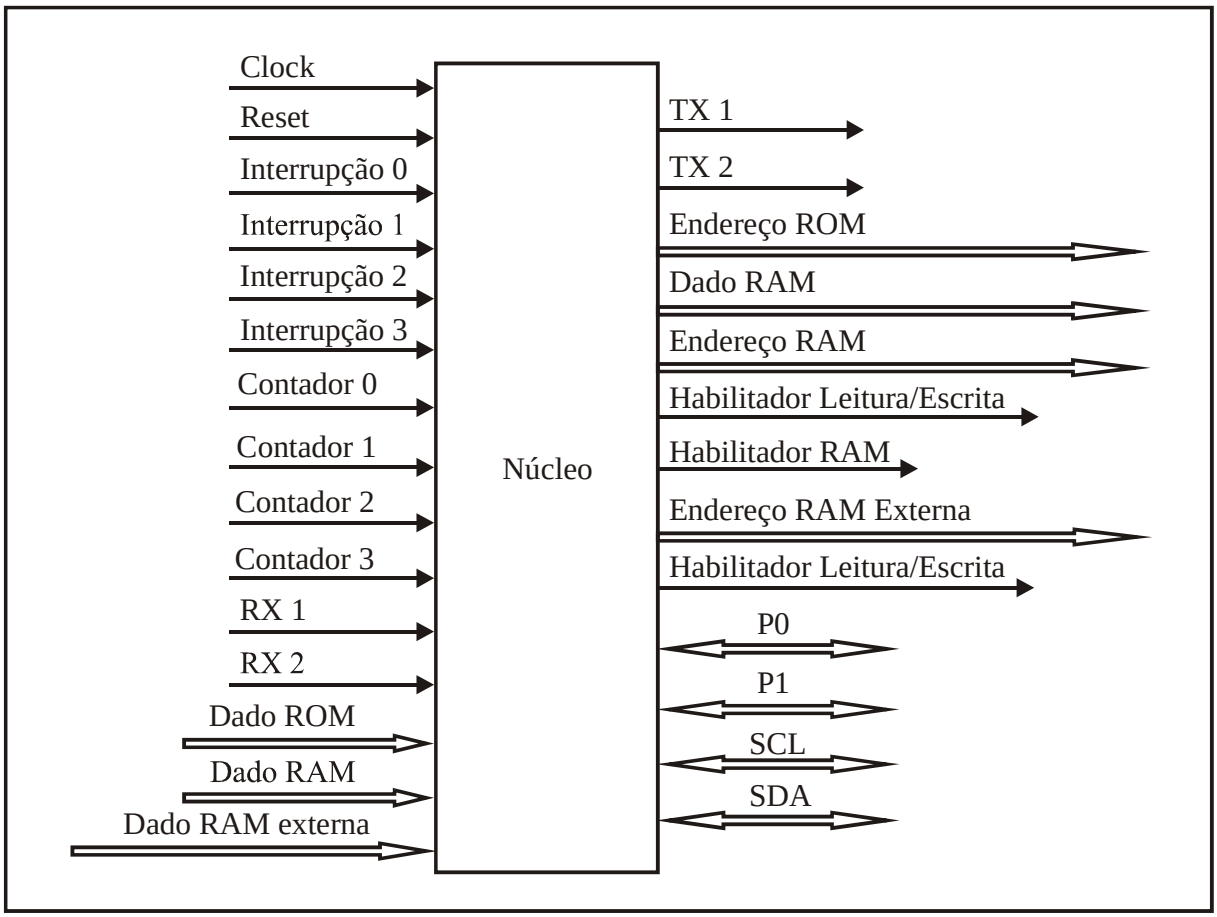

Figura 4.19 – Representação do bloco núcleo.

4.2.18 Blocos das Memórias ROM, RAM Interna e RAM Externa

Os blocos de memórias do microcontrolador foram implementados utilizando-se a função *MegaWizard Plug-In Manager* do *software* Quartus II que possibilita utilizar blocos de memórias configuráveis da FPGA, os M4K. Esses blocos são regiões específicas do FPGA, colunas dedicadas para a configuração das funções de memórias como a Memória de Acesso Aleatório – RAM (*Random Access Memory)*, a fila - FIFO (*First-In, First*-Out) e a Memória Somente de Leitura – ROM (*Read Only Memory*). Cada bloco possui 4 kbits de memória mais 512 bits de paridade, ou seja, 4608 bits de memória e o FPGA utilizado neste trabalho contam com um total de 105 blocos M4K ou 472,5 kbits de memória [10].

Como vantagem em utilizar esse recurso da ferramenta, tem-se a facilidade para embarcar o código desenvolvido (programa) na memória ROM, o qual será executado no M8051. O *software* Keil µVision4 da empresa ARM é utilizado para o desenvolvimento dos programas e a geração de um arquivo no formato *Intel-Format* (.hex) que pode ser associado na memória ROM para que o *software* Quartus II realize o processo de alocação das instruções (os códigos em hexadecimal) e dados nas posições de memória correspondentes. Outra vantagem é na etapa de geração do *layout* para fabricação do circuito integrado em que esses blocos podem ser substituídos pelos blocos pertencentes ao *Design Kit* de Memória da *Foundry* responsável pelo processo de fabricação.

Para o M8051 foi alocado um espaço de 32 kbytes (256 kbits) de memória ROM, pois devido ao limite da quantidade de blocos M4K disponíveis no FPGA utilizado, não foi possível reservar os 64 k originais do 8051, senão, não seria possível alocar as duas memórias RAM. A Figura 4.20 ilustra o bloco da memória ROM que possui como sinais de entrada o C*lock* do sistema e o endereço das posições dos dados composto de 15 bits (2^{15} = 32768 bits) e como sinal de saída, o dado de 8 bits.

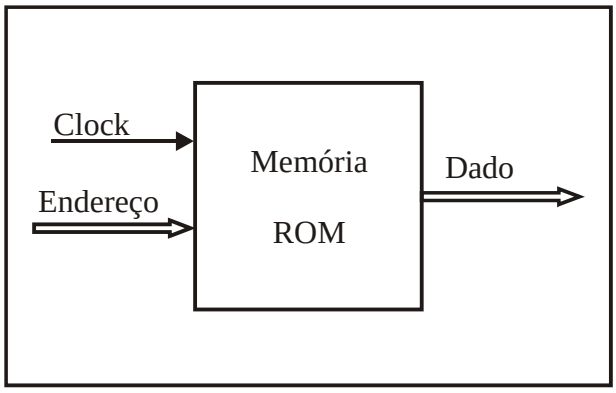

Figura 4.20 - Representação do bloco de memória ROM.

Para a memória RAM interna foi reservado 128 bits. Ela possui como sinais de entrada, o *Clock* do sistema, o sinal habilitador da ação de leitura ou escrita, o endereço das posições dos dados composto de 7 bits (2^7 = 128 bits) e o dado de 8 bits. Como sinal de saída, possui um dado de 8 bits. A Figura 4.21 mostra o bloco da memória RAM interna.

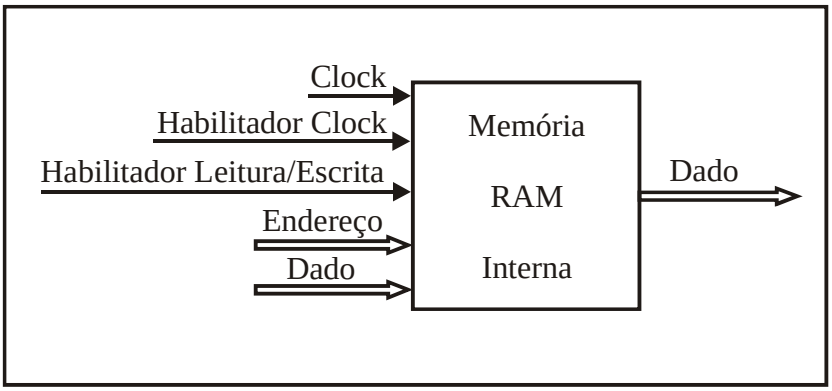

Figura 4.21 - Representação do bloco de memória RAM Interna.

Já para a memória RAM externa foram reservados 16 kbits. Ela contém como sinais de entrada o *Clock* do sistema, dado de 8 bits, endereço das posições dos dados composto de 14 bits (2^{14} = 16384 bits) e o sinal habilitador da ação de leitura ou escrita. Como sinal de saída possui um dado de 8 bits. A Figura 4.22 apresenta o diagrama da memória RAM externa.

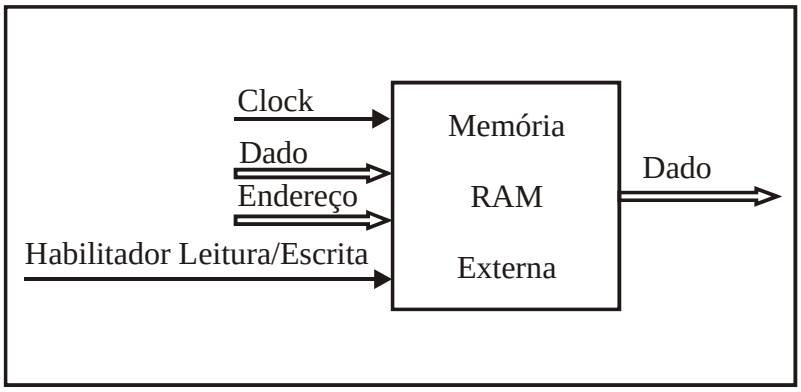

Figura 4.22 - Representação do bloco de memória RAM Externa.

4.2.19 Bloco Microcontrolador M8051

O bloco microcontrolador M8051, apresentado na Figura 4.23, é o último nível hierárquico do projeto e contém todos os blocos apresentados anteriormente. Neste nível encontram-se todos os pinos disponíveis para realizar as conexões com os diversos componentes que se queira utilizar ou testar em conjunto com M8051. O total de pinos nessa fase de prototipação é de 32 sendo 18 bidirecionais, 12 de entrada e 2 de saída.

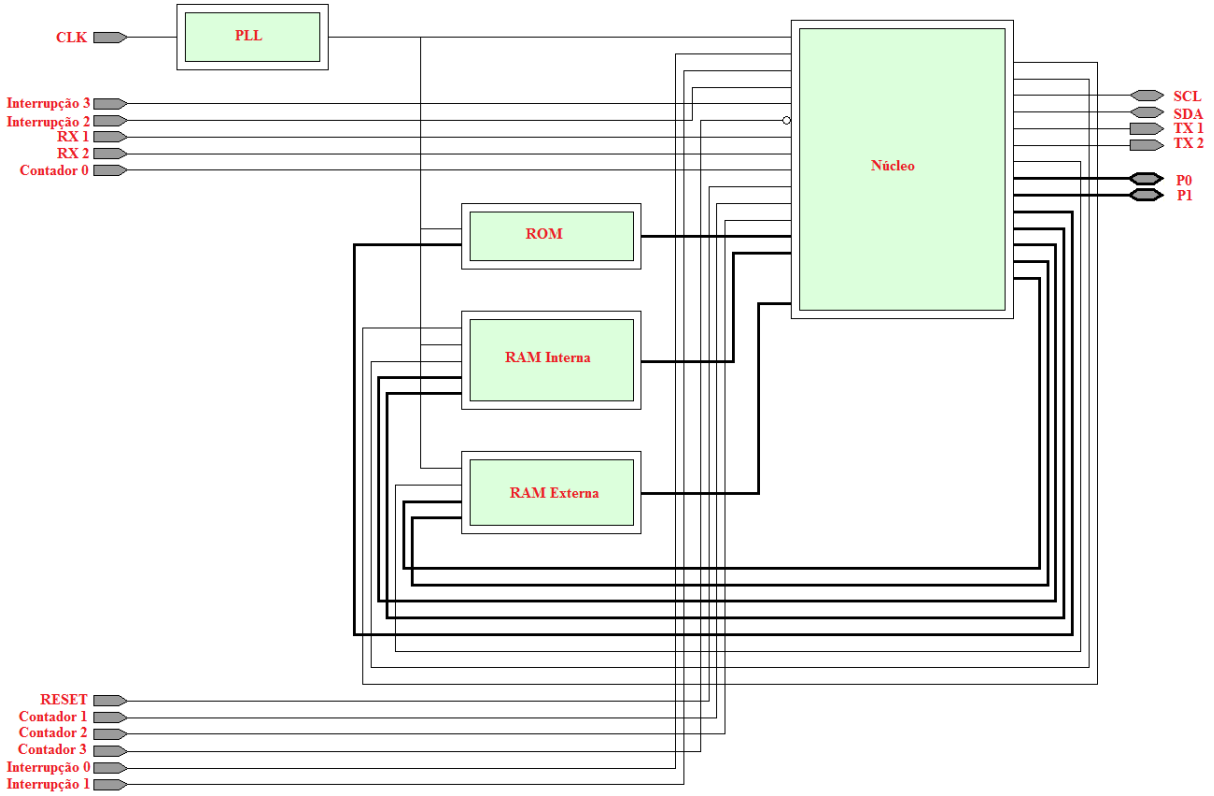

Figura 4.23 – Representação do Bloco M8051 baseado no circuito gerado pelo *RTL Viewer* do *software* Quartus II.

O pino SCL, responsável por transmitir o *clock*, e o SDA, responsável pelos dados transmitidos entre o microcontrolador e os periféricos do barramento, são utilizados na comunicação serial I²C. Ambos são bidirecionais.

TX1 e TX2 são os pinos de transmissão e RX1 e RX2 são os pinos de recepção da comunicação serial UART, em que TX1 e RX1 pertencem à estrutura da primeira serial e o TX2 e o RX2 a estrutura da segunda serial.

As portas P0 e P1 são compostas por oito pinos bidirecionais cada uma. Para este projeto optou-se em não implementar as portas P2 e P3 para realizar uma diminuição da quantidade de pinos do CI, devido ao acréscimo das novas funcionalidades.

Os pinos de entrada Interrupção 0, Interrupção 1, Interrupção 2, Interrupção 3, Contador 0, Contador 1, Contador 2 e Contador 3 são os pinos responsáveis pelo monitoramento de ações externas, sendo que os quatro primeiros, quando acionados geram a interrupção do programa principal quando habilitados, e respeitam a ordem de prioridade da Tabela 4.1. Os quatro últimos realizam a contagem da alternância de estados do sinal caso estejam habilitados, ou seja, a mudança do estado lógico de nível baixo para alto.

Tabela 4.1 – Endereço e Ordem de prioridade
das Interrupções na Memória ROM.

Endereço das Interrupções - ROM		
Prioridade	**Endereço**	
	0x0000	RESET
1°	0x0003	Interrupção 0
2°	0x000B	Temporizador 0
3°	0x0013	Interrupção 1
4°	0x001B	Temporizador 1
5°	0x0023	Serial 1
6°	0x002B	Interrupção 2
7°	0x0033	Temporizador 2
8°	0x003B	Interrupção 3
9°	0x0043	Temporizador 3
10°	0x004B	Serial 2

O pino de RESET é acionado quando seu estado é levado do nível lógico alto para o nível lógico baixo e reinicia as ações de todos os blocos do microcontrolador levando todos os sinais para os valores iniciais de operação.

O pino CLK recebe o sinal de um cristal, oscilador ou gerador e através do bloco PLL, implementado através da função *MegaWizard Plug-In Manager*. Pode-se configurar o valor da freqüência que se deseja trabalhar com o circuito respeitando o valor máximo de operação da estrutura que é de 48 MHz obtido através das análises realizadas pela função *Timing Analyzer* do *software* Quartus II.

O projeto conta com 15386 elementos lógicos de 33216 (46%), 2049 registros, 32 pinos de 475 (7%) e 394240 bits de memória de 483840 (81%).

Para a verificação funcional foi desenvolvido um arquivo de *testbench* que realiza o teste do funcionamento de todos os blocos implementados operando em conjunto. As informações de saída do arquivo de *testbench* e o próprio *testbench* do bloco encontram-se em anexo na mídia.

4.3 Conclusão

Com esse capítulo todos os blocos da estrutura do microcontrolador M8051 foram apresentados, discutidos e validados. Os blocos I^2C e AES128 serão apresentados nos Capítulos 5 e 6 respectivamente.

Capítulo 5

5 Interface de Comunicação Serial I^2C

5.1 Introdução

Esse capítulo apresenta a interface de comunicação serial I^2C, sua estrutura, seu modo de programação através do conjunto de instruções do microcontrolador M8051 e as formas de ondas obtidas do osciloscópio MSO2014 *Mixed Signal Oscilloscope*. Apresenta, também, alguns exemplos de comunicação para comprovação do funcionamento e exemplificação.

5.2 O Pradrão de Comunicação Serial I^2C

O I^2C é um padrão mundial de comunicação implementado em mais de 1000 diferentes CIs, como memórias, acelerômetros, microcontroladores e entre outros; é fabricado por mais de 50 companhias e utilizado em diversas arquiteturas de controle [11].

O protocolo de comunicação serial I^2C foi desenvolvido pela Philips (atualmente NXP) na década de 80 e significa integração entre circuitos (*Inter-Integrated Circuit)*. O objetivo era desenvolver um padrão de comunicação simples entre diversos periféricos de uma mesma placa de circuito impresso [11].

Apenas duas linhas de barramento são necessárias, sendo uma linha o dado serial – SDA (*Serial Data*) e a outra o *clock* serial – SCL (*Serial Clock*). Cada dispositivo conectado ao barramento é endereçável por um endereço exclusivo e comunicações simples entre mestre/escravo ocorrem a todo o momento. Os mestres podem trabalhar como mestres-transmissores ou como mestres-receptores.

As transferências são bidirecionais e podem ser realizadas em 4 velocidades: até 100 kbit/s no modo *Standard*, até 400 kbit/s no modo *Fast*, até 1 Mbit/s no modo *Fast Plus* ou até 3,4 Mbit/s no modo *High-speed*.

O número máximo de periféricos no barramento é limitado pelo espaço de endereçamento, e pela capacitância total do barramento, de até 400 [pF], pois a relação entre o

resistor de *pull-up* e a capacitância do fio ou trilha afetam o comportamento temporal dos sinais SDA e SCL. Contudo, valores maiores de capacitância podem ser permitidos em algumas condições [11].

A Figura 5.1 mostra um exemplo de como pode ser constituído uma aplicação I²C com diversos periféricos e barramentos.

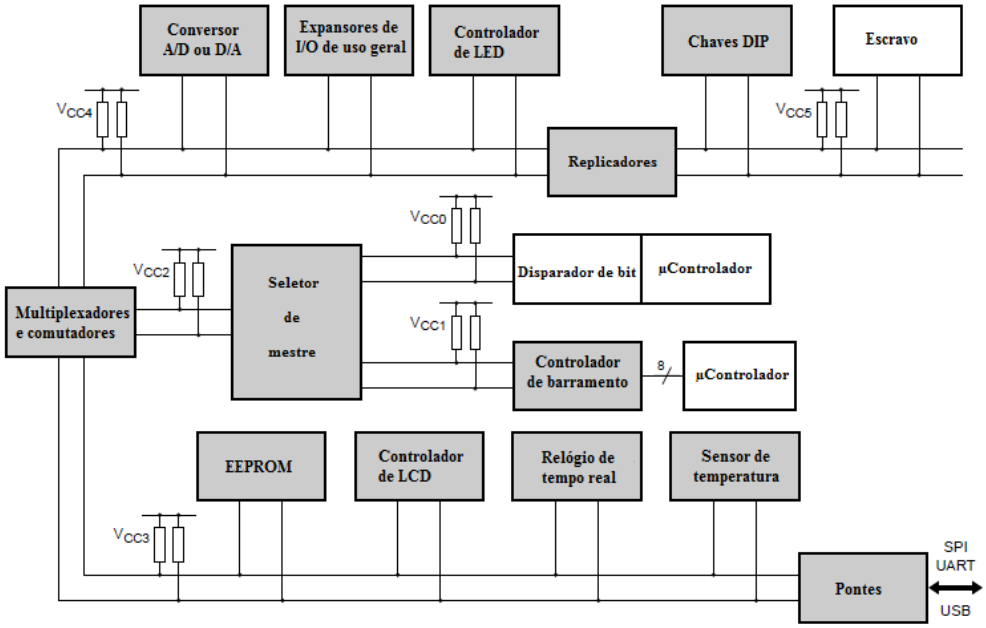

Figura 5.1 - Exemplo de aplicação do barramento I²C [11].

Nenhuma taxa de licenciamento é cobrada para implementar o protocolo desde 1º de outubro de 2006, porém, para se obter um endereço de escravo I²C é cobrada uma taxa.

O I²C utiliza apenas dois pinos drenos-abertos, bidirecionais e com resistores de *pull-up* para realizar a comunicação entre os dispositivos. Os componentes apenas levam o sinal para o nível baixo ou o deixam flutuando em V_{dd}. Os resistores de *pull-up* garantem o nível lógico alto quando nenhum componente estiver utilizando o barramento, sendo as tensões típicas de V_{dd} iguais a 3.3V ou 5V.

Existem 4 modos de operações possíveis entre o mestre e o escravo, sendo:

- Mestre Transmite: o mestre está controlando o *clock* e enviando dados para um escravo;
- Mestre Recebe: o mestre está controlando o *clock* e recebendo dados de um escravo;

- Escravo Transmite: o escravo não está controlando o *clock* e está enviando dados para um mestre;
- Escravo Recebe: o escravo não está controlando o *clock* e está recebendo dados de um mestre.

O mestre é o responsável pelo sinal de *clock* e pelo início da comunicação a qual é iniciada no momento em que ele leva o nível do sinal SDA para zero e mantém o SCL em nível alto, ou seja, realiza a ação de START bit (S). A Figura 5.2 exemplifica os sinais referentes às operações citadas para a comunicação entre o mestre e o escravo. Em seguida, o mestre prepara o primeiro bit de dado a ser enviado (faixa azul) enquanto o nível lógico do SCL está em baixo, e quando ele sobe o nível do SCL o bit é enviado (faixa verde). Esse processo acontece até que todos os bits necessários serem enviados. No final da transferência, o mestre gera o STOP bit (P) que é a ação de levar para nível alto o sinal SDA enquanto o sinal SCL está em nível alto.

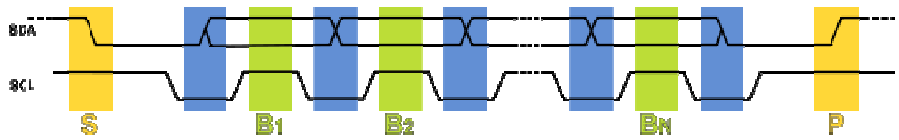

Figura 5.2 - Diagrama de tempo de uma transferência de dados.

Após o START bit, o mestre envia o endereço do escravo com quem ele quer realizar a comunicação. Este endereço é composto por 7 bits. Para completar a comunicação é enviado um bit que indica se o mestre deseja escrever ou ler no escravo, sendo nível baixo para escrita e nível alto para leitura. Caso o endereço de escravo exista no barramento, ele irá devolver um bit de *acknowledge* (A), em nível lógico baixo. A Figura 5.3 representa o protocolo de comunicação I²C.

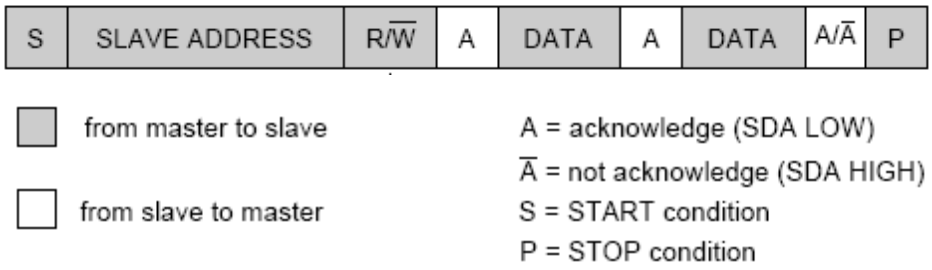

Figura 5.3 – Protocolo de comunicação I²C [11].

O endereço do escravo é enviado e constituído do bit mais significativo ao menos significativo. Para completar um byte, o bit menos significativo é o bit referente à ação que o mestre deseja executar, leitura ou escrita. A Figura 5.4 demonstra à constituição do byte do endereço do escravo mais a ação a ser executada.

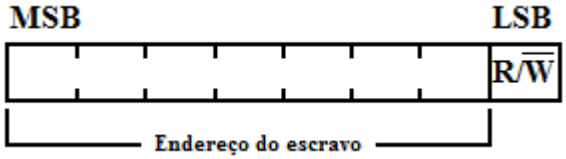

Figura 5.4 – Orientação do endereço do escravo.

5.2.1 Estrutura do bloco I²C

O bloco I²C é composto por 4 sub-blocos e 5 registros que em conjunto são os responsáveis pelas operações de transmissão e recepção de dados.

O sub-bloco *Clock Generator* é o responsável pela geração do *clock* da estrutura I²C através do valor *prescale* calculado conforme a Equação 1.

$$prescale = \frac{f_{\mu c}}{5 * f_{opKHz}} - 1 \quad (1)$$

Como exemplo, considerando $f_{\mu c}$ com a mesma freqüência que trabalha o microcontrolador, ou seja, aproximadamente 50 MHz e a freqüência de operação de transmissão de f_{opKHz} = 100 kHz, tem-se que se inserir um *prescale* de 99_d ou 63_h no registro específico.

O sub-bloco *Byte Command Controller* gerencia o tráfico de dados do protocolo I²C em nível de byte. Ele recebe o comando do Registro *Command Register* e traduz o dado em uma seqüência de ações a serem realizadas. Ao definir, por exemplo, o START, STOP e READ bit, o *Byte Command Controller* gera uma seqüência que resulta na geração do sinal de START, a leitura de um byte do dispositivo escravo e a geração do sinal de STOP. Ele realiza esta ação dividindo cada operação em comandos de bits que são então enviados para o *Bit Command Controller*. A Figura 5.5 apresenta o fluxograma do processo que ocorre no sub-bloco *Byte Command Controller.*

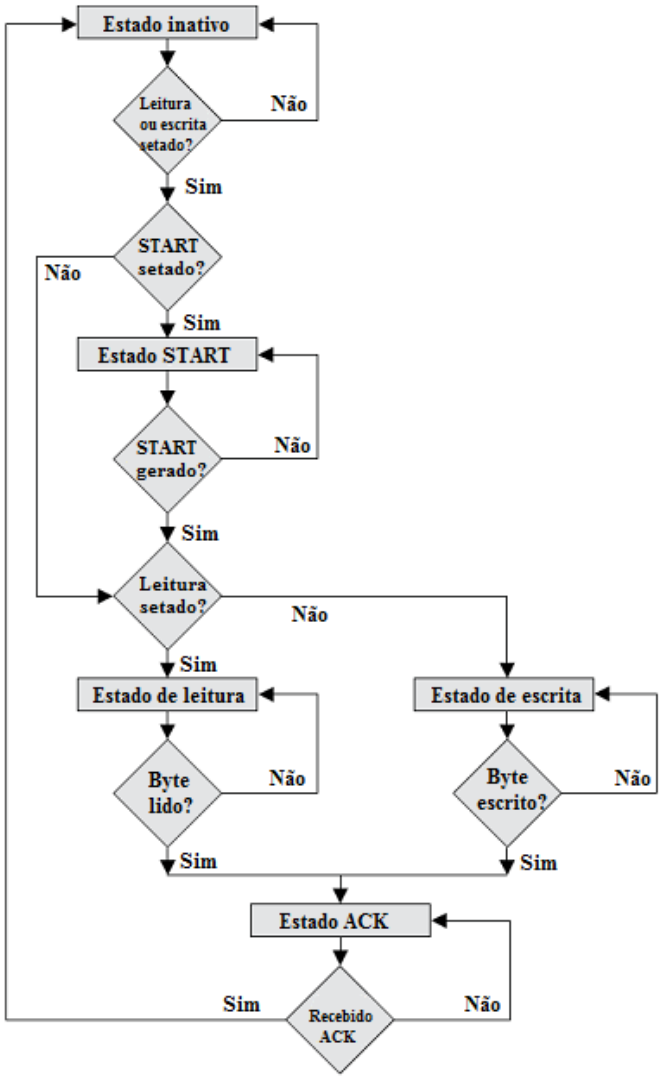

Figura 5.5 – Fluxograma do processo do sub-bloco *Byte Command Controller*.

O sub-bloco *Bit Command Controller* gerencia o momento da transferência do dado e gera os níveis de sinais para as ações de START, *Repetição do* START, e STOP através do controle das linhas SCL e SDA. O sub-bloco *Byte Command Controller* comunica ao sub-bloco *Bit Command Controller* qual operação deve ser realizada. Para apenas um byte de leitura, o *Bit Command Controller* recebe separadamente 8 comandos de leitura. Cada bit-operação é dividido em 5 partes (idle, A, B, C e D), exceto a ação de STOP que é divida em 4 partes (idle, A, B e C). A Figura 5.6 exemplifica os níveis de tensão em cada etapa de cada ação ao longo do tempo.

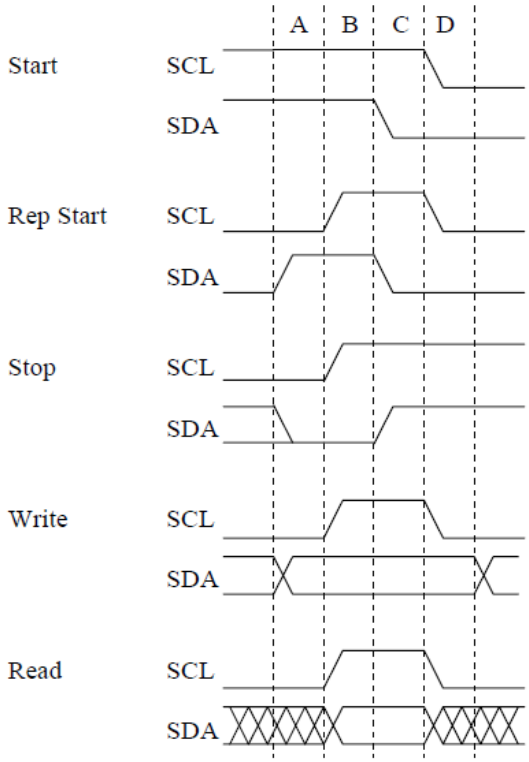

Figura 5.6 – Níveis lógico dos sinais de cada ação em relação ao tempo.

O sub-bloco *DataIO Shift Register* armazena o dado associado a transferência corrente. Como exemplo, durante a ação de leitura, o dado é deslocado da linha do SDA para o sub-bloco, após todos os bits serem recebidos o conteúdo é copiado para o registro *Receive Register*. Durante a ação de escrita, o conteúdo do registro *Transmit Register* é copiado para o *DataIO Shif Register* e transmitido bit a bit pela linha SDA.

A Figura 5.7 representa o fluxo dos dados e das ações por todos os sub-blocos e registros do bloco I²C.

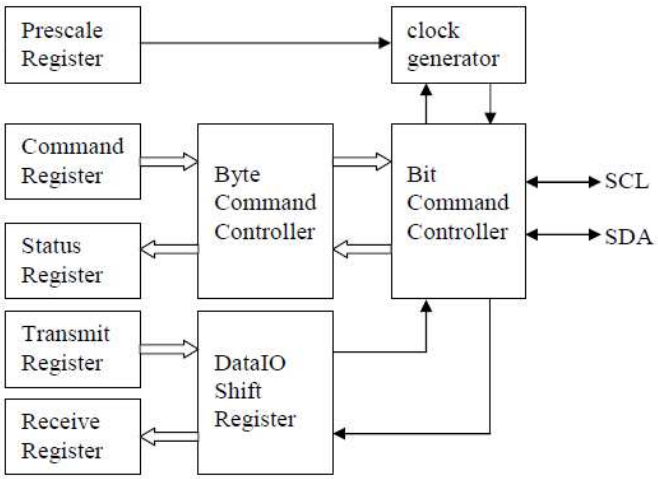

Figura 5.7 – Representação do fluxo de dados do bloco I²C.

O bloco de comunicação I²C foi desenvolvido, inicialmente, separado do M8051 e, posteriormente, integrado ao microcontrolador. Para a validação do bloco foi desenvolvido um *testbench* que se encontra na mídia anexa junto com os resultados.

5.2.2 Integração e programação do bloco I²C

Para a utilização da interface de comunicação serial I²C em conjunto com o microcontrolador desenvolvido, foram reservadas as posições de memória do SFR que não estavam sendo utilizadas pelos demais periféricos para alocar os registros do bloco I²C. Como há registros que o programador necessita alterar o seu valor bit a bit, foram priorizados para ocuparem as posições de endereço com as terminações em 0 ou 8 pois nem todos os SFR são bits endereçáveis. Apenas os SFRs cujos endereços são divisíveis por 8 são bits endereçáveis. O *nibble* inferior do endereço do SFR deve ser 0 ou 8. Por exemplo, o SFR 0xA0 e 0xD8 são bits endereçáveis, enquanto o SFR 0xC7 e 0xEB não são. Para calcular um endereço SFR bit, é necessário adicionar a posição do bit no endereço SFR byte. Então, para acessar o bit 6 do SFR no endereço 0xC8 o endereço SFR bit seria 0xCE (0xC8 + 6) [12].

A Figura 5.8 é a representação do bloco I²C que será integrado a arquitetura do microcontrolador M8051 e a Tabela 5.1 apresenta todas as posições de memórias utilizadas e seus respectivos registros.

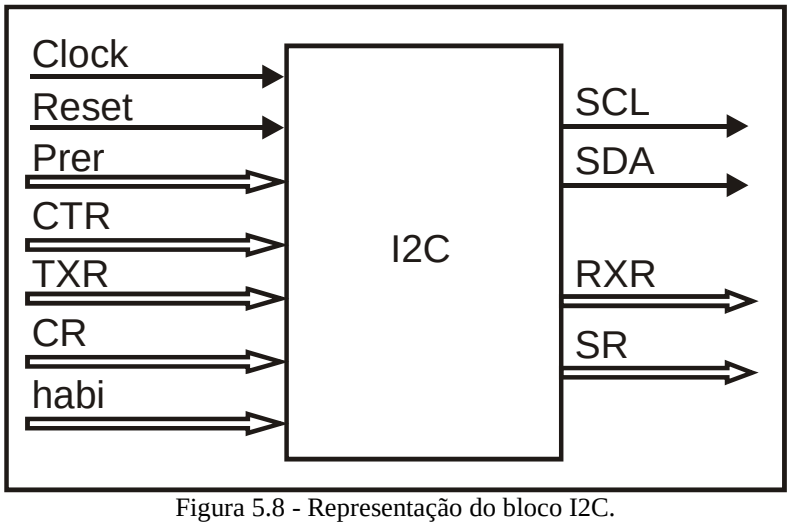

Figura 5.8 - Representação do bloco I2C.

Tabela 5.1 - Posições de memórias e registros referentes ao bloco I2C.

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	
A0	EN	U	U	U	U	U	U	U	CTR
A1	PRER_LO								PRER
A2	PRER_HI								
A3	X	X	X	X	X	X	X	RW	TXR
A4	RXR								
A5	U	U	U	U	U	U	cr_habi	habi	HABI
B0	STA	STO	RD	WR	ACK	U	U	U	CR
C0	RxACK	BUSY	AL	U	U	U	TIP	U	SR

U: reservado X: indefinido

O Registro de Controle (CTR) é o responsável por habilitar ou desabilitar todo o bloco, permitindo ou não que uma comunicação entre mestre e escravo ocorra.

O Registro Transmite (TXR) armazena o dado a ser enviado para o escravo. Quando se envia um endereço de escravo o bit menos significativo corresponde à ação de leitura ou escrita.

O Registro Receber (RXR) armazena o dado recebido pelo mestre quando ele está operando no modo de receber.

O Registro Comando (CR) contém todas as operações que o bloco realiza, sendo que, cada bit representa uma ação. O bit 7 corresponde ao Start ou Restart. O bit 6 é o responsável pelo Stop. O bit 5 habilita a função de recebimento de dado e o bit 4 a função de envio de dado. O bit 3 corresponde ao sinal de *Acknowlegde*, sendo que, quando o mestre estiver operando no modo receber, para ele enviar um sinal de ACK deve colocar zero no bit 3 e colocar um no bit 3 para enviar um NACK. Os bits 2 e 1 são posições reservadas. As posições 7, 6, 5 e 4 são sempre levadas ao nível lógico 0 automaticamente após cada operação.

O Registro Status (SR) apresenta as informações do processo através de 8 posições, sendo o bit 7 responsável pela informação de recebimento ou não do *Acknowledge*. O bit 6 indica se o barramento está disponível ou não. O bit 5 indica a perca de sincronismo. Os bits 4, 3, 2 e 0 são reservados. O bit 1 indica o fim ou não de um processo, como por exemplo, de uma transferência de um dado.

Para a utilização do bloco integrado ao M8051, foi desenvolvido um programa, listado no Apêndice A, com a finalidade de comprovação da integração e funcionamento correto da nova estrutura implementada na arquitetura do microcontrolador. A Figura 5.9 apresenta um exemplo de comunicação realizada entre o M8051 com o bloco I²C e a memória PCF8570 [13] atuando como escravo. A imagem foi extraída do osciloscópio MSO2014 *Mixed Signal*

Oscilloscope e permite comprovar o funcionamento correto do bloco integrado ao microcontrolador desenvolvido. No barramento B1 foi habilitada a opção de análise do protocolo I²C e na linha 1 tem-se o sinal do SDA e na linha 0 o sinal do SCL. Analisando o barramento B1 identifica-se com o colchete verde o sinal de inicialização da comunicação e em seguida o envio do endereço do escravo (53_h), mais o bit de escrita (0_b). Após o recebimento do sinal de *acknowlege* pelo M8051 a segunda etapa do processo de comunicação é o envio do endereço da posição de memória que será lida ($C7_h$). Após o recebimento do segundo *acknowlege* por parte do M8051 é necessário gerar um novo início, indicado pelo colchete verde, junto com o endereço do escravo (53_h) e o bit de leitura (1_b). Após o recebimento do terceiro *acknowlege* o escravo irá enviar o dado (84_h) referente a posição de memória solicitada e o mestre, o M8051, enviará um *acknowlege* para o escravo confirmando o recebimento do dado e assim pode-se gerar um sinal de STOP, conforme indicado pelo colchete vermelho ou ler uma nova posição de memória.

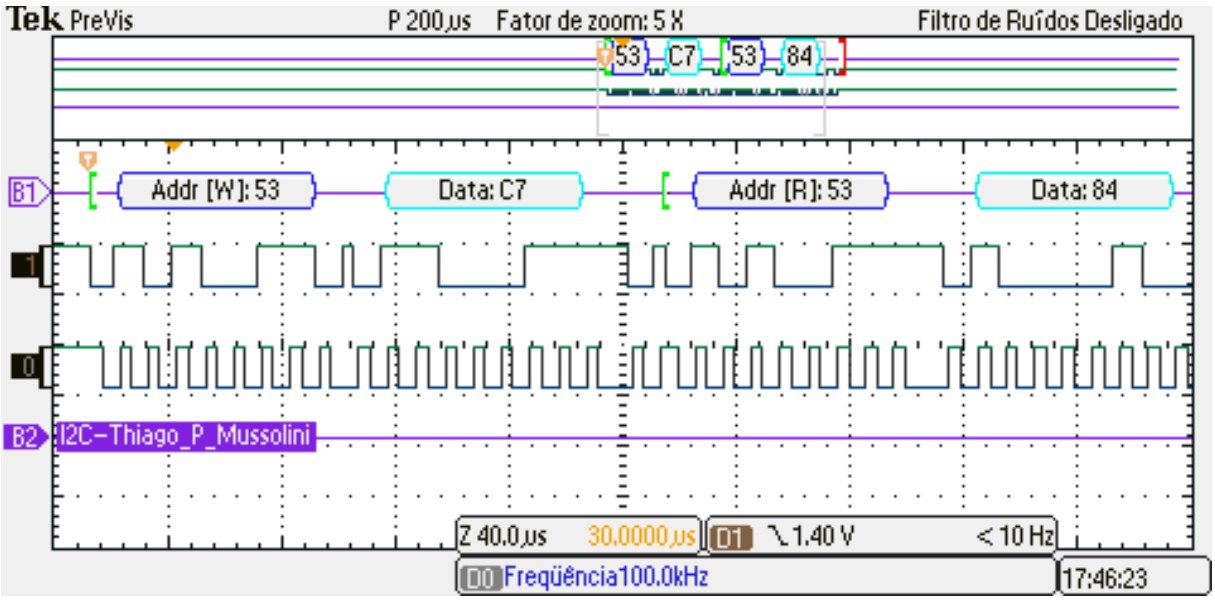

Figura 5.9 – Processo de leitura de posição de memória através da comunicação I²C.

A Figura 5.10 apresenta a estrutura utilizada para realizar os testes da comunicação serial I²C e a Figura 5.11 apresenta no retângulo vermelho o valor do dado lido, 84_h, numa representação em binário através do conjunto de oito LEDs, portanto, 10000100_b.

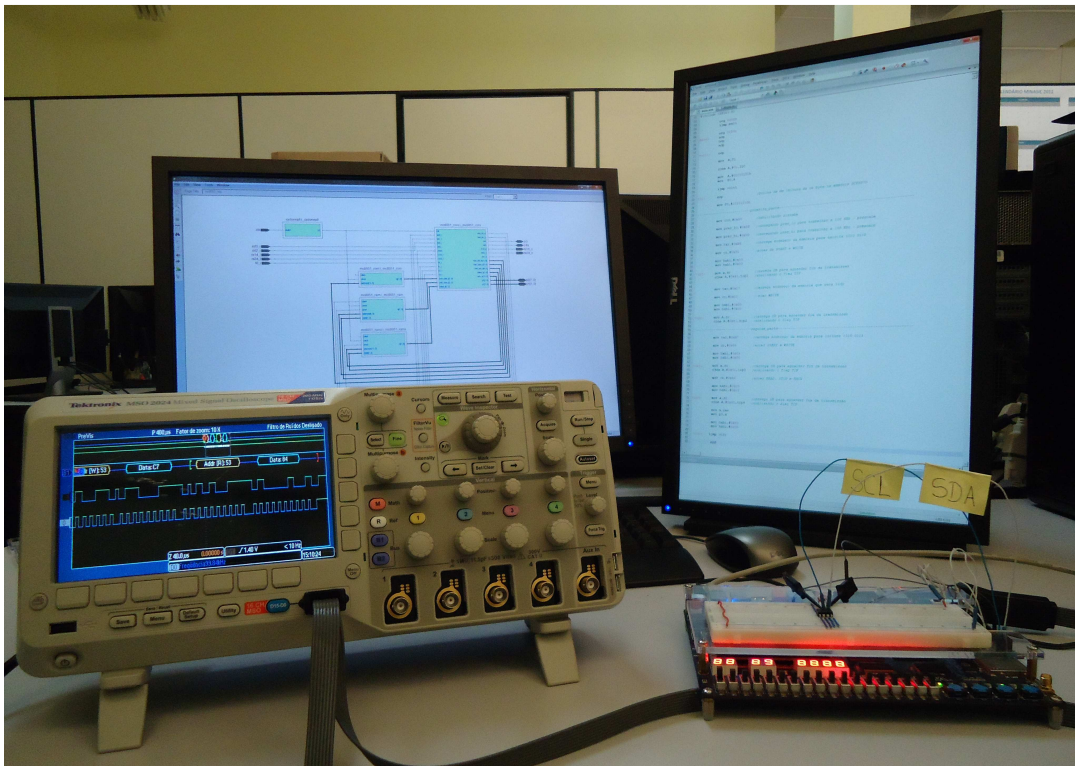

Figura 5.10 – Estrutura para teste da comunicação serial I²C com vista frontal.

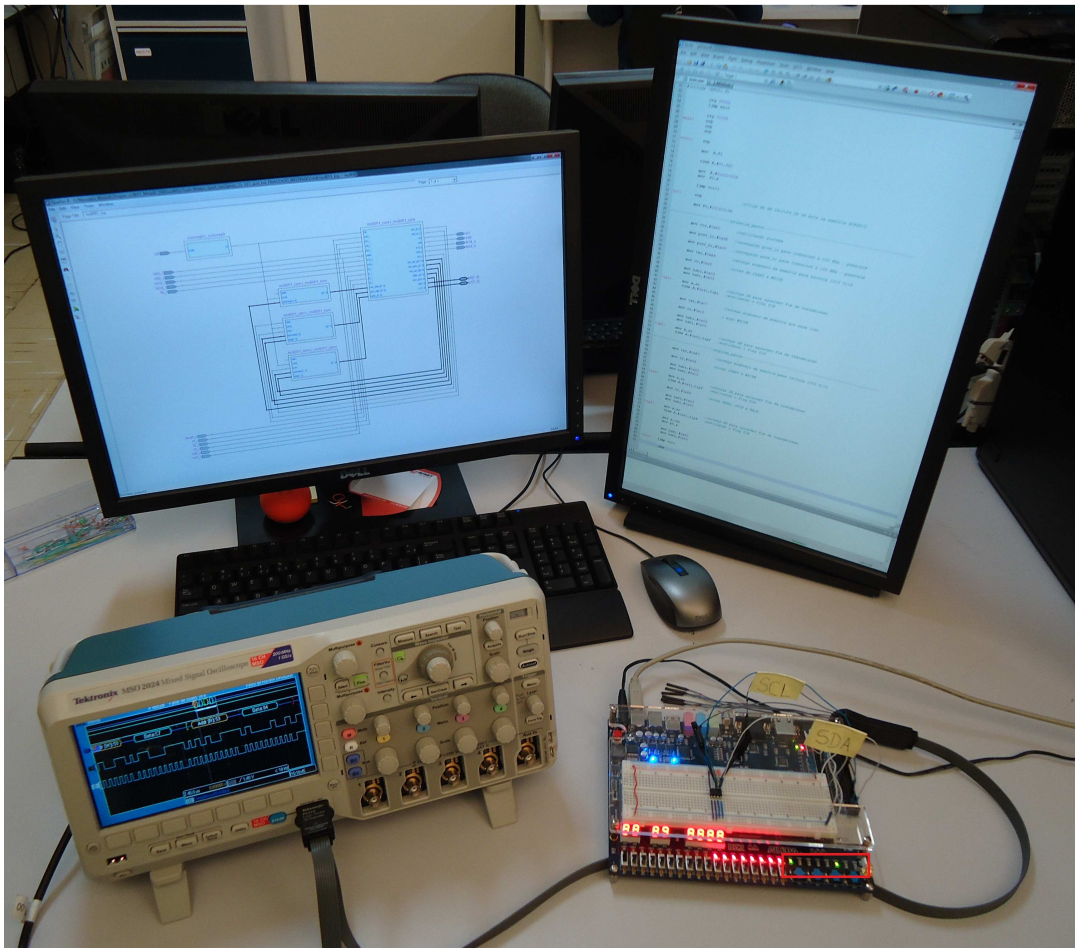

Figura 5.11 – Estrutura para teste da comunicação serial I²C com vista superior.

5.3 Conclusão

Com esse capítulo comprova-se o funcionamento correto do bloco I^2C e sua integração ao microcontrolador M8051. Também foi apresentado um exemplo de código, Apêndice A, desenvolvido na ferramenta µVision4 para a utilização do processo e imagem extraída do osciloscópio MSO2014 *Mixed Signal Oscilloscope* que comprova o funcionamento correto da estrutura e a sua integração ao M8051.

Capítulo 6

6 Criptografia AES 128

6.1 Introdução

Esse capítulo apresenta a Criptografia AES 128, sua estrutura, seu modo de programação através do conjunto de instruções do microcontrolador M8051 e o resultado das simulações realizadas para comprovação de funcionamento.

6.2 Histórico

No ano de 1997 o Instituto Nacional de Normas em Tecnologia – NIST (*National Institute of Standards and Technology*) dos EUA iniciou um processo para substituir o algoritmo DES (*Data Encryption Standard*). Para isso foi realizado um concurso em que determinados requisitos deveriam ser atendidos: direitos autorais livres, maior rapidez em relação ao 3DES (Tiple DES – uma otimização do DES), cifrar em blocos de 128 bits com chaves de 128, 192 e 256, possibilidade de implementação em *software* e *hardware* [14].

Em 1998 , o NIST selecionou 15 algoritmos e, em 1999, selecionou 5 finalistas. Em 2 de outubro de 2000 foi definido o algoritmo vencedor, o Rijndael, desenvolvido por dois pesquisadores belgas, Vicent Rijmen e Joan Daemen.

6.3 O algoritmo Rijndael

O Rijndael é um algoritmo de criptografia de bloco simétrico, com tamanho de bloco e de chaves variáveis, podendo ser especificado independentemente para 128, 192 ou 256 bits. Possui facilidade de implementação, propiciando o uso em *Smart Cards* (cartões magnéticos utilizados em operações bancárias ou de compra eletrônica) e em outros equipamentos que utilizam pouca memória RAM, além disso, utiliza poucos ciclos de processamento. O código desse algoritmo é reduzido e não depende de nenhum outro tipo de componente criptográfico,

como gerador de números randômicos. Esse aspecto faz com que sua utilização apresente um nível de segurança superior [15].

6.4 Operação do AES

O algoritmo AES pode operar com tamanho de chaves diferentes sendo 128, 192 ou 256, porém, o tamanho do dado ou palavra sempre será de 128 bits. Para a operação de criptografia com uma chave de 128 bits serão necessárias 10 rodadas para o encerramento do processo. Para 192 bits, 12 rodadas, e para 256, 14 rodadas [15]. A Tabela 6.1 apresenta um resumo das possibilidades de configurações.

Tabela 6.1 – Possibilidades de configurações do AES.

AES	Palavra	Chave	Rodadas
128	4 x 4	4 x 4	10
192	4 x 4	4 x 6	12
256	4 x 4	4 x 8	14

Para o processo de criptografia segue-se uma seqüência de etapas até que se obtenha o dado criptografado. A Figura 6.1 representa todas as etapas envolvidas no processo.

A etapa *AddRoundKey* é a operação Ou-Exclusivo entre um bloco palavra e um bloco chave (uma das chaves da expansão), resultando em um novo bloco de mesma dimensão, conforme Figura 6.2 [16].

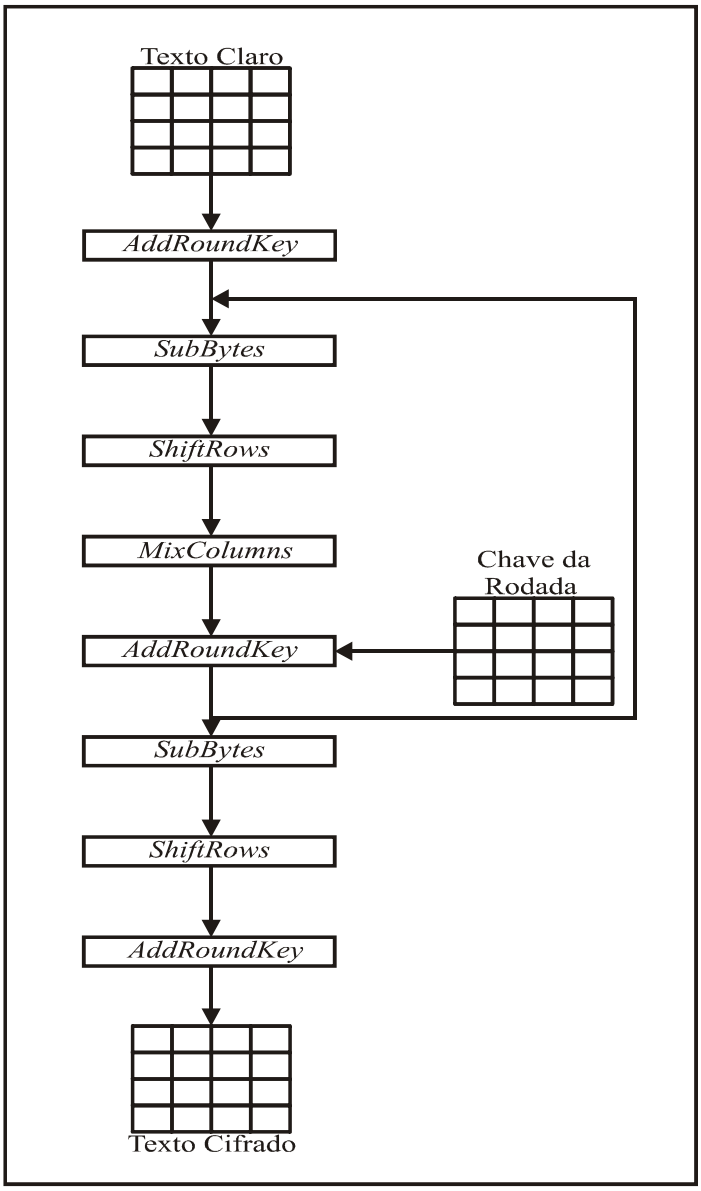

Figura 6.1 – Processo de Criptografia.

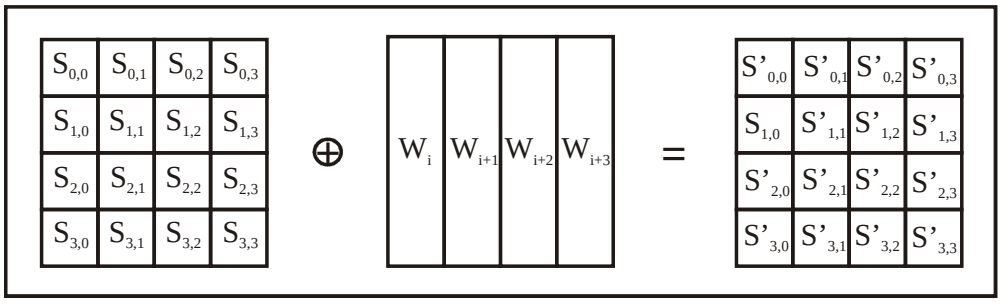

Figura 6.2 – Etapa *AddRoundKey*.

A etapa *SubBytes* utiliza a matriz S-Box calculada pela matemática polinomial de Galois para realizar as substituições. A matriz está representada na Tabela 6.2.

Tabela 6.2 – S-Box.

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

A Figura 6.3 demonstra de maneira sucinta como é feita a substituição dos valores. Tem-se o bloco de palavra representado pela Figura 6.3a e faz-se a operação *SubBytes* no primeiro termo: 5F. É selecionada a linha 5 e a coluna F da matriz S-Box, encontrando-se na intercessão desta linha e coluna o valor CF. É feita a substituição de 5F por CF para realizar a operação Sub Bytes, conforme incida a Figura 6.3b. Essa operação de busca e substituição é realizada para cada uma das células do bloco (palavra) que está sendo calculado [16].

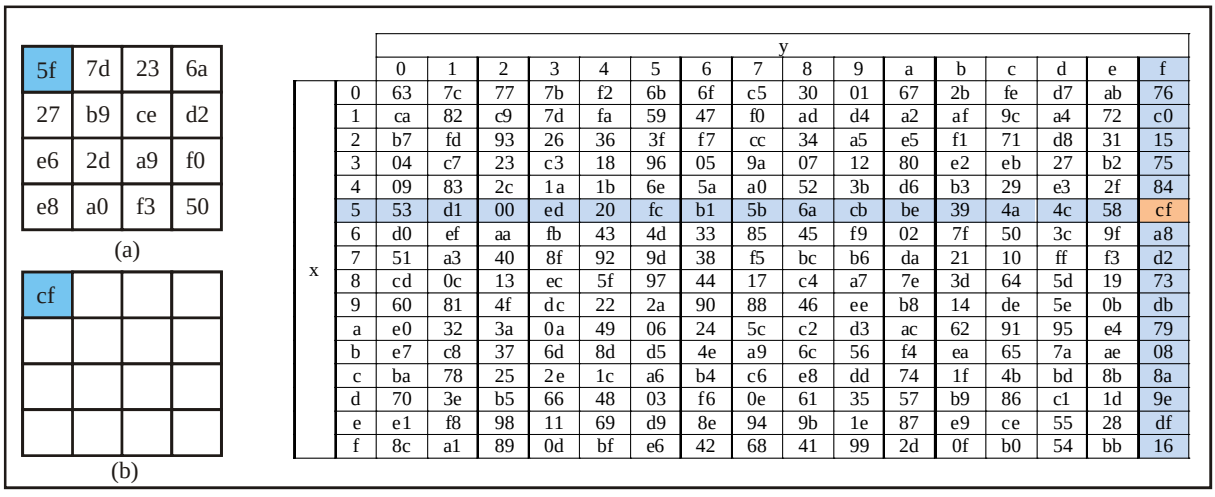

Figura 6.3 – Etapa *SubBytes*.

A etapa *ShiftRows* realiza um deslocamento das células que estão à esquerda. O número de células deslocadas obedece ao número da linha que sofrerá a alteração. A Figura 6.4a ilustra o bloco antes e a Figura 6.4b depois da operação *ShifRows*: a primeira linha (linha número 0) não sofre alteração, a segunda linha (linha número 1) sofre apenas um

deslocamento, a terceira linha (linha número 2) sofre dois deslocamentos e quarta linha (linha número 3) sofre três deslocamentos [16].

5F	7D	23	6A
27	B9	CE	D2
E6	2D	A9	F0
E8	A0	F3	50

(a)

5F	7D	23	6A
B9	CE	D2	27
A9	F0	E6	2D
50	E8	A0	F3

(b)

Figura 6.4 – Etapa *ShiftRows.*

Para o processo de criptografia segue-se uma seqüência de etapas até que se obtenha o dado criptografado. A Figura 6.5 representa todas as etapas envolvidas no processo.

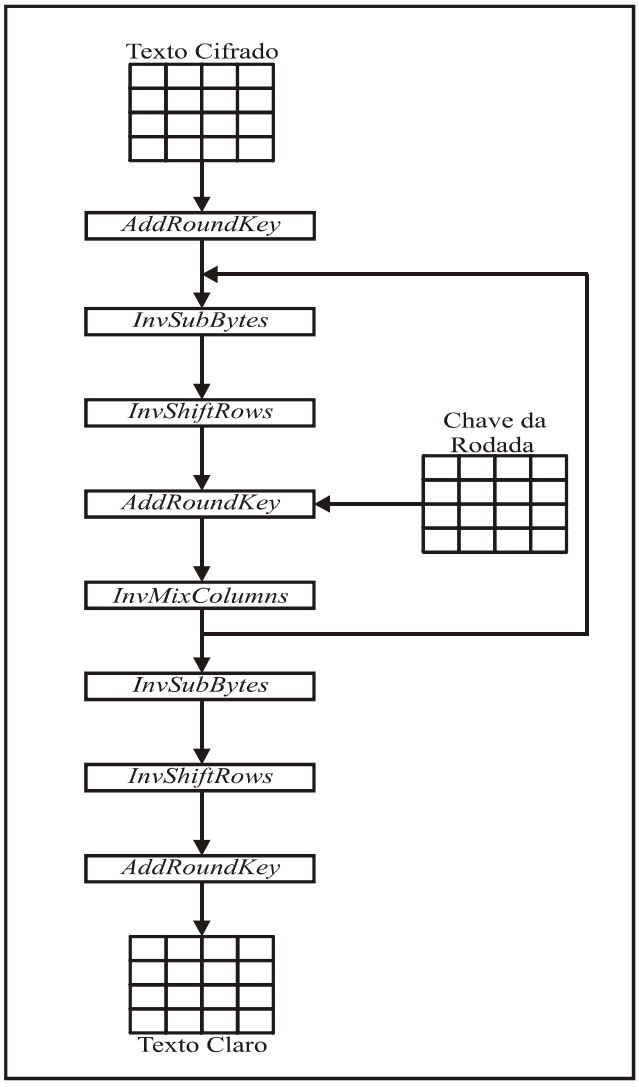

Figura 6.5 – Processo de Decriptografia.

A etapa *InvSubBytes* realiza uma operação similar à função *SubBytes*. É importante ressaltar que a matriz utilizada para as substituições é diferente da função *SubBytes*, é chamada *InvS-Box.*

Tabela 6.3 – InvS-Box.

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

A etapa *InvShiftRows* é similar à função *ShiftRows* mas realiza um deslocamento das células que estão à direita. A Figura 6.6 mostra um esquemático de como é realizada esta operação. O deslocamento das células tem como objetivo fazer com que o bloco fique embaralhado. As funções de decifragem visam reverter às mudanças realizadas pela cifragem para que a mensagem volte ser legível ao usuário [16].

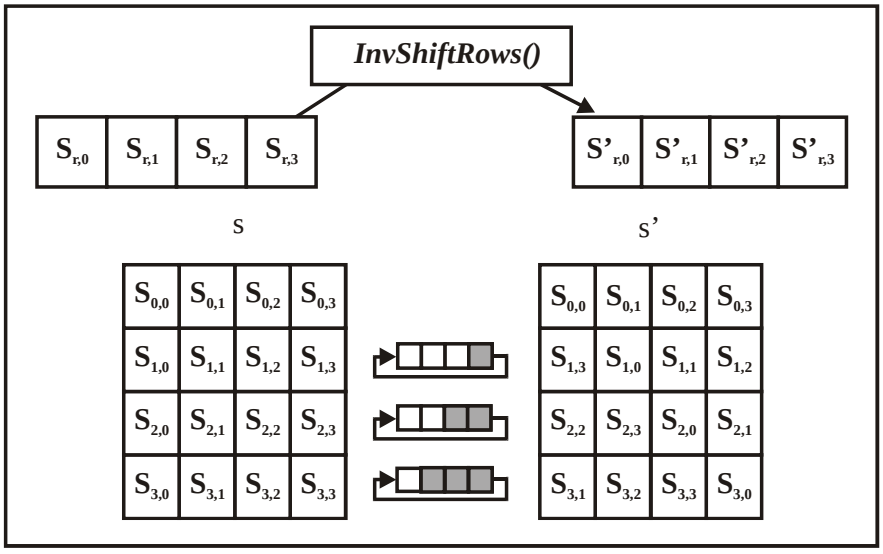

Figura 6.6 – Etapa *InvShiftRows*.

A etapa *InvAddRoundKey* e *AddRoundKey* são idênticas, isto é, uma operação Ou-Exclusivo entre um bloco palavra e um bloco chave (uma das chaves da expansão), resultando em um novo bloco de mesma dimensão.

A etapa *InvMixColumms* realiza uma operação similar à função *MixColumns*. A matriz utilizada para realizar a operação é diferente daquela utilizada na operação *MixColumns* e está representada na Figura 6.7.

0E	0B	0D	09
09	0E	0B	0D
0D	09	0E	0B
0B	0D	09	0E

Figura 6.7 – Constante para a etapa *InvMixColumns.*

A etapa Expansão da Chave utiliza-se uma matriz de constantes que é obtida pela matemática polinomial de Galois [15]. A matriz de 4 linhas e 10 colunas está representada na Figura 6.8 para 10 rodadas de expansão [16].

01	02	04	08	10	20	40	80	1B	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	001	00	00	00
00	00	00	00	00	00	00	00	00	00

Figura 6.8 – Constantes das rodadas.

A expansão da chave ocorre da seguinte maneira:

a) A chave é disponibilizada em um bloco de 4 linhas e 4 colunas;

b) As primeiras operações serão feitas sobre a quarta coluna, da seguinte forma:

b.1) uma rotação na quarta coluna de modo que a célula que está acima de todas as outras fique abaixo delas.

b.2) a partir do resultado da rotação é feita a substituição utilizando a tabela *S-Box,* como mostrado na Tabela 6.2.

b.3) após a substituição é feita uma operação de Ou-Exclusivo com a primeira coluna da matriz de constantes (Figura 6.8) e, em seguida, é feita uma operação de Ou-Exclusivo com a primeira coluna do bloco inicial.

b.4) tem-se assim a primeira coluna do próximo bloco de chave.

c) A segunda coluna do bloco de chave é o resultado de uma operação Ou-Exclusivo entre a primeira coluna do novo bloco (calculada em b.4) e a segunda coluna do bloco inicial.

d) A terceira coluna do bloco de chave é o resultado de uma operação Ou-Exclusivo entre a segunda coluna do novo bloco (calculado em c) e a terceira coluna do bloco inicial.

e) A quarta coluna do bloco de chave é o resultado de uma operação Ou-Exclusivo entre a terceira coluna do novo bloco (calculado em d) e a quarta coluna do bloco inicial.

f) Tem-se, assim, um segundo bloco formado por 4 linhas e 4 colunas com 8 bits em cada uma das células e que será a chave da primeira rodada.

g) A partir do bloco da primeira rodada (calculado em f) inicia-se todo este procedimento utilizando a quarta coluna. As operações se repetem até que todas as chaves de cada rodada estejam calculadas.

6.5 Implementação e Integração no M8051

O bloco de criptografia AES 128 foi implementado, verificado e validado separadamente para garantir o seu funcionamento de forma correta. Após essa etapa inicial o bloco foi integrado ao microcontrolador M8051. A Figura 6.9 apresenta uma representação do bloco resultado da síntese do código desenvolvido.

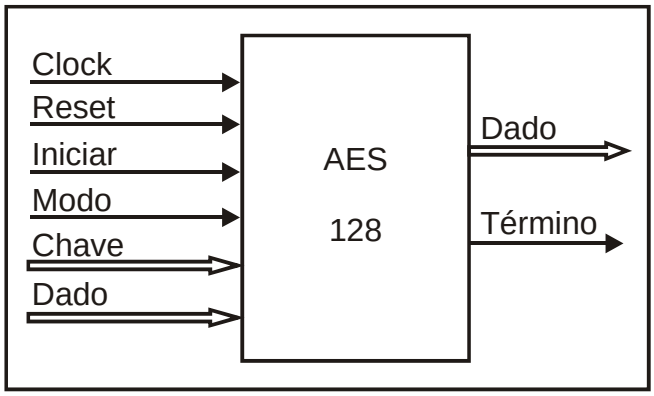

Figura 6.9 – Representação do bloco de criptografia AES128.

Para o funcionamento do bloco é necessário o fornecimento do dado de entrada de 128 bits, a chave de 128 bits, o modo de operação (criptografia ou decriptografia) e um pulso para iniciar o sistema. Quando o processo de criptografia ou decriptografia terminar o sistema fornece um dado de saída de 128 bits e um sinal de término (*flag*).

A Figura 6.10 apresenta o resultado da simulação do bloco para o processo de criptografia. Como dado de entrada foi utilizado a palavra $00112233445566778899AABBCCDDEEFF_h$ e como chave a palavra $000102030405060708090A0B0C0D0E0F_h$. Escolheram-se esses valores, pois são os dados utilizados como exemplo no documento do NIST [17]. O resultado obtido após a operação de criptografia foi $69C4E0D86A7B0430D8CDB78070B4C55A_h$ que é o mesmo apresentado como resultado pelo documento, confirmando o funcionamento correto do bloco.

Para a simulação do processo de decriptografia foi utilizado o resultado obtido na operação de criptografia como dado de entrada e a mesma chave do processo de criptografia. O resultado obtido foi $00112233445566778899AABBCCDDEEFF_h$ como dado de saída. O processo é apresentado na Figura 6.11.

Para a simulação do bloco foi utilizado um sinal de *clock* de 50 MHz o qual é suficiente para a integração com o M8051. O sinal de *reset* permaneceu em nível alto por 20ns e depois levado a nível lógico baixo. O sinal de início deve permanecer em nível lógico alto durante uma borda de subida do sinal de *clock*. Se o sinal do modo estiver em nível lógico alto será realizado o processo de criptografia e após 13 ciclos de *clock* o dado de saída estará disponível e o sinal de término será levado para o nível alto. Porém se o sinal do modo estiver em nível lógico baixo será realizado o processo de decriptografia e após 23 ciclos de *clock* o dado de saída estará disponível e o sinal de término será levado para o nível alto.

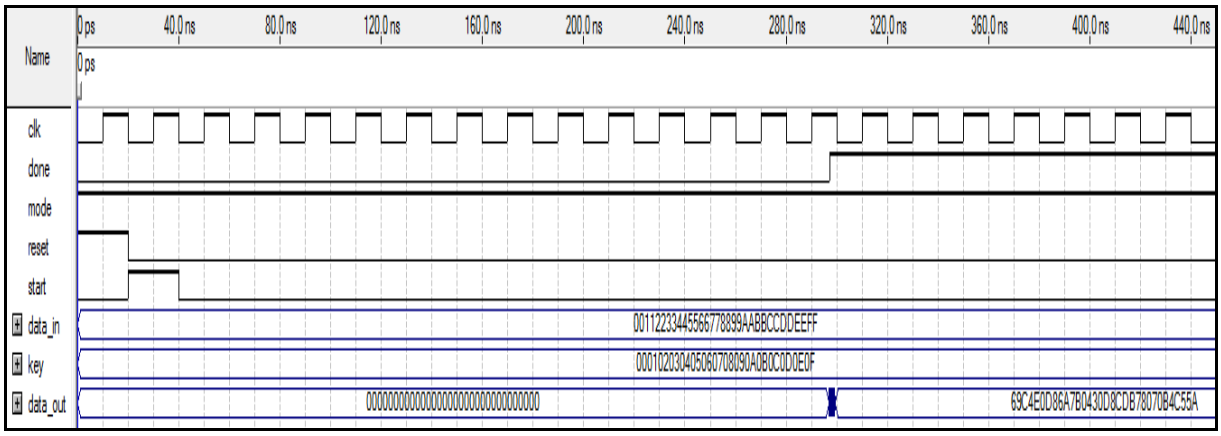

Figura 6.10 – Resultado da simulação do bloco AES128 para criptografia.

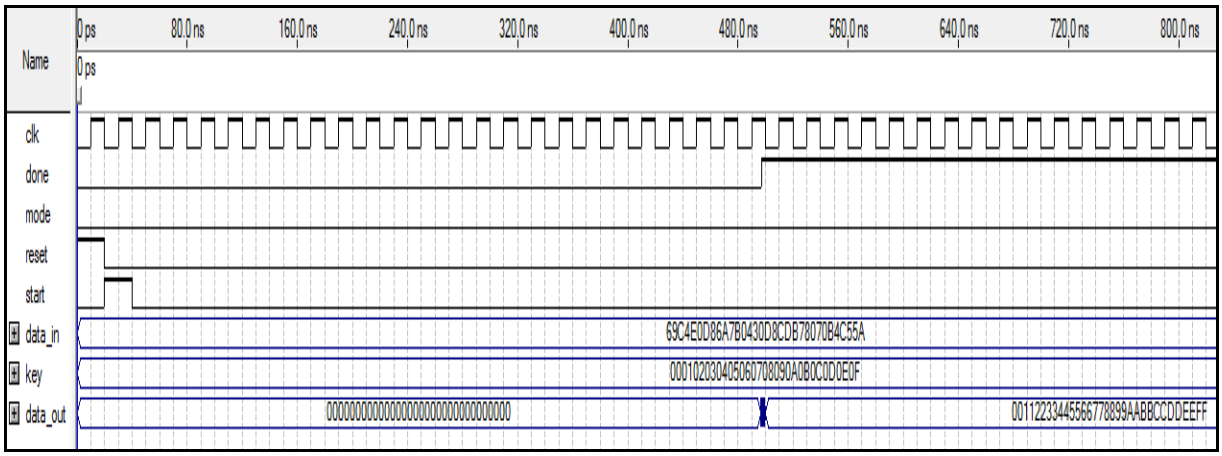

Figura 6.11 – Resultado da simulação do bloco AES128 para decriptografia.

Caso seja necessário realizar outro processo em seqüência basta carregar os novos dados e gerar um novo sinal de inicio. O sinal de término será levado para nível lógico baixo automaticamente. A Figura 6.12 apresenta à simulação de dois processos sequenciais em que no primeiro é realizada a criptografia do dado $00112233445566778899AABBCCDDEEFF_h$ com a chave $000102030405060708090A0B0C0D0E0F_h$.e obtém-se, como esperado, a palavra $69C4E0D86A7B0430D8CDB78070B4C55A_h$. Para o segundo processo utilizou-se a palavra resultante do primeiro processo como dado de entrada e a mesma chave. A palavra chave obtida foi $00112233445566778899AABBCCDDEEFF_h$ e um novo sinal de término foi gerado.

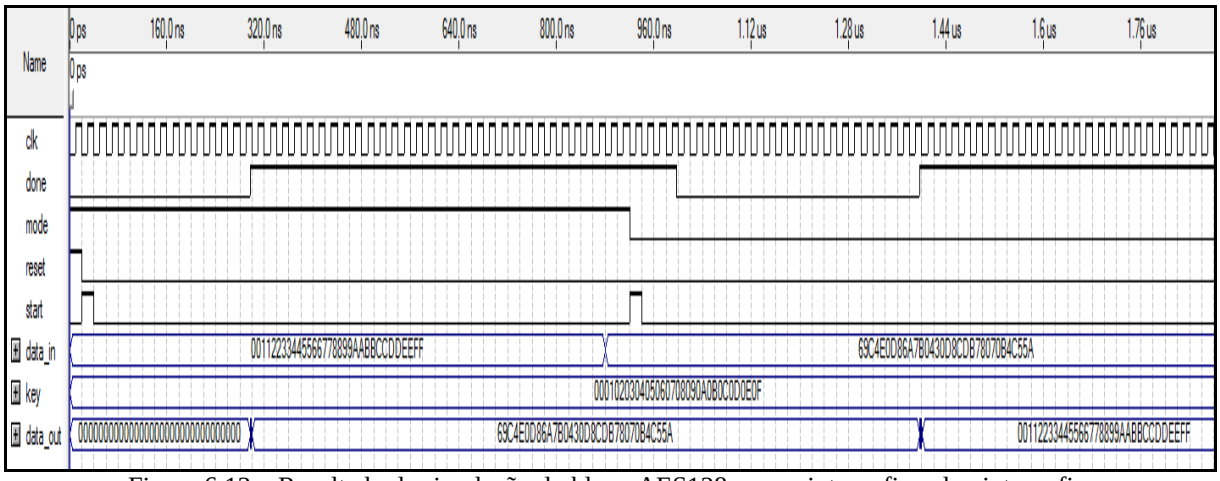

Figura 6.12 – Resultado da simulação do bloco AES128 para criptografia e decriptografia.

Para a integração do bloco AES128 no M8051, cada dado de entrada, saída, chave e comandos foram alocados em posições não utilizadas do SFR do microcontrolador. O dado de entrada, saída e chave ocuparam 16 posições cada, pois cada registro do SFR é composto de 8 bits. Foram utilizadas 50 posições do SFR para que a integração do bloco AES128 ao microcontrolador M8051 fosse possível. A Tabela 6.5 apresenta cada posição (endereço) com seu respectivo dado.

O *software* µVision4 da empresa Keil Software foi utilizado para realizar a programação do M8051, pois através das suas instruções é possível acessar os registros referentes aos dados de entrada e saída do bloco AES128 e desenvolver rotinas para que informações sejam criptografadas antes de serem trafegadas por redes. O programa desenvolvido para utilizar e validar o bloco encontra-se no Apêndice B.

Tabela 6.4 – Registros do bloco AES 128.

A9	Dado de entrada 0 de 8 bits							
AA	Dado de entrada 1 de 8 bits							
AB	Dado de entrada 2 de 8 bits							
AC	Dado de entrada 3 de 8 bits							
AD	Dado de entrada 4 de 8 bits							
AE	Dado de entrada 5 de 8 bits							
AF	Dado de entrada 6 de 8 bits							
B1	Dado de entrada 7 de 8 bits							
B2	Dado de entrada 8 de 8 bits							
B3	Dado de entrada 9 de 8 bits							
B4	Dado de entrada 10 de 8 bits							
B5	Dado de entrada 11 de 8 bits							
B6	Dado de entrada 12 de 8 bits							
B7	Dado de entrada 13 de 8 bits							
B9	Dado de entrada 14 de 8 bits							
BA	Dado de entrada 15 de 8 bits							
BB	X	X	X	X	X	X	INICIO	MODO
BC	Dado de saída 0 de 8 bits							
BD	Dado de saída 1 de 8 bits							
BE	Dado de saída 2 de 8 bits							
BF	Dado de saída 3 de 8 bits							
C1	Dado de saída 4 de 8 bits							
C2	Dado de saída 5 de 8 bits							
C3	Dado de saída 6 de 8 bits							
C4	Dado de saída 7 de 8 bits							

C5	Dado de saída 8 de 8 bits							
C6	Dado de saída 9 de 8 bits							
C7	Dado de saída 10 de 8 bits							
C9	Dado de saída 11 de 8 bits							
CA	Dado de saída 12 de 8 bits							
CB	Dado de saída 13 de 8 bits							
CC	Dado de saída 14 de 8 bits							
CD	Dado de saída 15 de 8 bits							
CE	X	X	X	X	X	X	X	TERMINO
D1	Chave 0 de 8 bits							
D2	Chave 1 de 8 bits							
D3	Chave 2 de 8 bits							
D4	Chave 3 de 8 bits							
D5	Chave 4 de 8 bits							
D6	Chave 5 de 8 bits							
D7	Chave 6 de 8 bits							
D9	Chave 7 de 8 bits							
DA	Chave 8 de 8 bits							
DB	Chave 9 de 8 bits							
DC	Chave 10 de 8 bits							
DD	Chave 11 de 8 bits							
DE	Chave 12 de 8 bits							
DF	Chave 13 de 8 bits							
E1	Chave 14 de 8 bits							
E2	Chave 15 de 8 bits							

6.6 Conclusão

Com esse capítulo comprova-se o funcionamento correto do bloco AES128 e sua integração ao microcontrolador M8051. Também foi apresentado um exemplo de código, Apêndice B, desenvolvido na ferramenta µVision4 para a utilização do processo e a teoria para o desenvolvimento do bloco.

Capítulo 7

7 Conclusões

Neste trabalho foi desenvolvido um microcontrolador de 8 bits em linguagem VHDL baseado no conjunto de instruções do conhecido 8051 e prototipado em FPGA. Na primeira etapa foram estudadas as características do microcontrolador, o que permitiu que uma estrutura em hierarquia fosse elaborada para facilitar a concepção do circuito. Depois, cada bloco (nível da hierarquia) foi implementado e testado individualmente. Para finalizar, todos os blocos desenvolvidos foram integrados em uma única estrutura, o M8051, e testados em *software* através de simulações e também prototipado em FPGA. Com isso, pode-se afirmar que o objetivo do trabalho foi atingido.

Alterações das características originais do 8051 foram realizadas para adequar o circuito às necessidades de futuros projetos do Grupo de Microeletrônica. Entre elas encontram-se o acréscimo da segunda comunicação serial UART, mais dois temporizadores/contadores, uma interface de comunicação serial I^2C e um bloco de criptografia AES128.

Devido à nova estrutura desenvolvida se gasta menos ciclos de *clock* para executar as ações das instruções, porém, nenhuma alteração foi realizada no conjunto de instruções, ou seja, todas as 255 instruções realizam as mesmas ações apresentadas no manual do microcontrolador, possuem o mesmo mnemônico e código de operação.

Com as simulações apresentadas ao decorrer deste trabalho, prova-se que todos os blocos desenvolvidos operam corretamente tanto individualmente como integrados na estrutura do M8051. Através da prototipação em FPGA do circuito, reforça-se a validação do funcionamento da estrutura.

Este trabalho teve como principal objetivo obter o domínio da técnica de como desenvolver em linguagem VHDL um microcontrolador de 8 bits e acrescentar novas funcionalidades, por isso, a quantidade de elementos lógicos utilizados e a frequência máxima de operação não foram fatores determinantes neste projeto.

Como sugestão para trabalhos futuros pode-se considerar o acréscimo de novos periféricos na estrutura, a continuidade do fluxo digital do processo ASIC gerando o layout do circuito e sua prototipação em *Standard Cell*. Sugere-se também o desenvolvimento de uma plataforma proprietária para programação e simulação do microcontrolador para atender as novas características. Pode-se ainda desenvolver os blocos analógicos para integrar ao layout do circuito ASIC tais como oscilador, POR, conversores analógico - digital e/ou digital – analógico, entre outros.

8 Apêndices

8.1 Apêndice A – Rotina desenvolvida no *software* µVision4 para utilização do bloco I^2C integrado ao M8051.

```
I2C:    nop                     ;rotina de leitura de um byte na memória PCF8570
;-------------------------primeira_parte-----------------------------------------------------------
        mov ctr,#0x80           ;habilitando sistema

        mov prer_lo,#0x5E       ;carregando prer_lo para trabalhar a 100 kHz - prescale

        mov prer_hi,#0x00       ;carregando prer_hi para trabalhar a 100 kHz - prescale

        mov txr,#0xA6           ;carrega endereço da memória para escrita 1010011 + 0b

        mov cr,#0x90            ;ações de START e WRITE

        mov habi,#0x03          ;habilita as ações selecionadas do CR
        mov habi,#0x00

tip1:   mov a,sr                ;carrega SR para aguardar fim da transmissão
        cjne A,#0x41,tip1       ;analisando o flag TIP

        mov txr,#0xC7           ;carrega endereço da memória que será lido

        mov cr,#0x10            ; ação WRITE

        mov habi,#0x03          ;habilita as ações selecionadas do CR
        mov habi,#0x00

tip2:   mov A,sr                ;carrega SR para aguardar fim da transmissão
        cjne A,#0x41,tip2       ;analisando o flag TIP
;-------------------------segunda_parte------------------------------------------------------------
        mov txr,#0xA7           ;carrega endereço da memória para leitura 1010 011 + 1b

        mov cr,#0x98            ;ações START e WRITE

        mov habi,#0x03          ;habilita as ações selecionadas do CR
        mov habi,#0x00

tip3:   mov a,sr                ;carrega SR para aguardar fim da transmissão
        cjne A,#0x41,tip3       ;analisando o flag TIP

        mov cr,#0x68            ;ações READ, STOP e NACK

        mov habi,#0x03          ;habilita as ações selecionadas do CR
        mov habi,#0x00

tip4:   mov a,sr                ;carrega SR para aguardar fim da transmissão
        cjne A,#0x81,tip4       ;analisando o flag TIP
```

8.2 Apêndice B – Rotina desenvolvida no *software* µVision4 para utilização do bloco AES128 integrado ao M8051.

```
AES:    nop                     ;rotina de criptografia

        mov di0,#0x00           ;carregando dado de 128 bits
        mov di1,#0x11
        mov di2,#0x22
        mov di3,#0x33
        mov di4,#0x44
        mov di5,#0x55
        mov di6,#0x66
        mov di7,#0x77
        mov di8,#0x88
        mov di9,#0x99
        mov di10,#0xaa
        mov di11,#0xbb
        mov di12,#0xcc
        mov di13,#0xdd
        mov di14,#0xee
        mov di15,#0xff

        mov k0,#0x00            ;carregando chave de 128 bits
        mov k1,#0x01
        mov k2,#0x02
        mov k3,#0x03
        mov k4,#0x04
        mov k5,#0x05
        mov k6,#0x06
        mov k7,#0x07
        mov k8,#0x08
        mov k9,#0x09
        mov k10,#0x0a
        mov k11,#0x0b
        mov k12,#0x0c
        mov k13,#0x0d
        mov k14,#0x0e
        mov k15,#0x0f

        mov s_s_m,#00000011b    ;gerando o pulso para iniciar o processo
        nop
        mov s_s_m,#00000001b

do:     mov A,done              ;aguarda o término do processo
        cjne A,#0x01,do
```

9 Referências Bibliográficas

[1] LIPSETT, Roger; SCHAEFER, Carl; USSERY, Cary. **VHDL: Hardware Description and Design.** Kluwer Academic Publishers, 1989.

[2] ARANTES, Dalton; CARDOSO, Fabbryccio. **FPGA e Fluxo de Projeto.** DECOM-FEEC-UNICAMP, Campinas: 2008.

[3] CASILLO, Leonardo Augusto. **Projeto e implementação em FPGA de um processador com conjunto de instrução reconfigurável utilizando VHDL**. 126 p. Dissertação (Mestrado) – Programa de Pós-Graduação em Sistemas e Computação, Universidade Federal do Rio Grande do Norte, Natal, maio 2005.

[4] CORPORATION, Altera. **Cyclone II Device Handbook.** EUA, v.1, fev. 2007.

[5] CORPORATION, Altera. **Development and Education Board:** DE2 user manual. EUA, v. 1.41, 2007.

[6] RUSHTON, Andrew. **VHDL for Logic Synthesis**. McGraw-Hill book Company, 1995.

[7] SOUZA, Andre R. Ciraulo de. **Desenvolvimento e implementação em FPGA de um sistema portátil para aquisição e compressão sem perdas de eletrocardiogramas.** 180 p. Dissertação (Mestrado) – Programa de Pós-Graduação em Informática, Centro de Ciências Exatas e da Natureza, Departamento de Informática, Universidade Federal da Paraíba, João Pessoa, abril 2008.

[8] MARINHO, José E. Santos; MARINHO, Ednaldo Santos. Mini-curso de microcontrolador. **Saber Eletrônica**, São Paulo, v. especial, n. 2, jan. 2001.

[9] CORPORATION, Intel. **MCS@51 Microcontroller:** family user´s manual. EUA, n. 272383-002, fev. 1994.

[10] CORPORATION, Altera. **Cyclone II Device Handbook**. EUA, fev. 2008. Cap. 8, p. 8-32.

[11] NXP**,** founded by Philips. **UM10204:** I^2C-bus specification and user manual. EUA, rev. 3, junho 2007.

[12] ARM Ltd and ARM Germany GmbH. **Cx51 User´s Guide.** EUA, 2011. Disponível em: <http://www.keil.com/support/man/docs/c51/c51_le_sbit.htm> Acesso em: 19 jan. 2011.

[13] PHILIPS, Semiconductors. **PCF8570:** 256 x 8-bit static low-voltage RAM with I^2C-bus interface. EUA, 06 jan. 1999.

[14] MITSURI, Matsui. **Linear Criptanalysis Method for DES Cipher**. Japan: Mitsubishi Eletric Corporation, 1993. P.386-397.

[15] DAEMEN, Joan; RIJMEN, Vincent. **A Specification for Rijndael, the AES Algorithm**. NIST (National Institute of Standards and Technology).

[16] GOMES, Otávio de Souza Martins. **Desenvolvimento de Hardware Configurável de Criptografia Simétrica utilizando FPGA e linguagem VHDL**. 69 p. Dissertação (Mestrado) – Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Itajubá, Itajubá, janeiro 2011.

[17] **Advanced Encryption Standard (AES)**. EUA: nov. 2001. 47 p. <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf> Acesso em: 28 out. 2010.