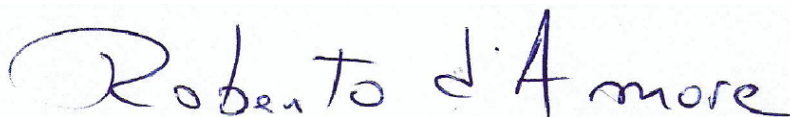


Dissertação apresentada à Pró-Reitoria de Pós-Graduação e Pesquisa do Instituto Tecnológico de Aeronáutica, como parte dos requisitos para obtenção do título de Mestre em Ciências no Programa de Pós-Graduação em Engenharia Eletrônica e de Computação, Área de Dispositivos e Sistemas Eletrônicos.

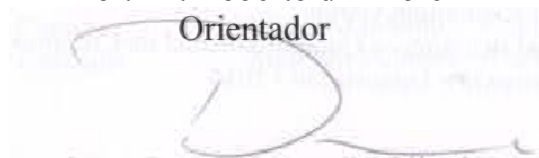
Kledermon Garcia

**SÍNTESE COMPORTAMENTAL DE CIRCUITOS
ASSÍNCRONOS OTIMIZADOS**

Dissertação aprovada em sua versão final pelos abaixo assinados:



Prof. Dr. Roberto d'Amore
Orientador



Prof. Dr. Duarte Lopes de Oliveira
Coorientador

Prof. Dr. Luiz Carlos Sandoval Góes
Pró-Reitor de Pós-Graduação e Pesquisa

Campo Montenegro
São José dos Campos, SP – Brasil
2015

Dados Internacionais de Catalogação-na-Publicação (CIP)

Divisão de Informação e Documentação

Garcia, Kledermon

Síntese Comportamental de Circuitos Assíncronos Otimizados / Kledermon Garcia.

São José dos Campos, 2015.

145f.

Dissertação de mestrado – PG/EEC Engenharia Eletrônica e Computação, EEC-D Dispositivos e Sistemas Eletrônicos – Instituto Tecnológico de Aeronáutica, 2015. Orientador: Prof. Dr. Roberto D’Amore. Coorientador: Prof. Dr. Duarte Lopes de Oliveira.

1. Circuitos assíncronos. 2. Sistemas digitais. 3. Circuitos lógicos. I. Instituto Tecnológico de Aeronáutica. II. Síntese Comportamental de Circuitos Assíncronos Otimizados.

REFERÊNCIA BIBLIOGRÁFICA

Garcia, Kledermon. **Síntese Comportamental de Circuitos Assíncronos Otimizados**. 2015. 145f. Dissertação de mestrado em dispositivos e sistemas eletrônicos – Instituto Tecnológico de Aeronáutica, São José dos Campos.

CESSÃO DE DIREITOS

NOME DO AUTOR: Kledermon Garcia

TÍTULO DO TRABALHO: Síntese Comportamental de Circuitos Assíncronos Otimizados

TIPO DO TRABALHO/ANO: Dissertação / 2015

É concedida ao Instituto Tecnológico de Aeronáutica permissão para reproduzir cópias desta dissertação e para emprestar ou vender cópias somente para propósitos acadêmicos e científicos. O autor reserva outros direitos de publicação e nenhuma parte desta dissertação ou tese pode ser reproduzida sem a sua autorização (do autor).

Kledermon Garcia

Av. Plínio Marcos, 500 – Villa Branca

CEP: 12301-350, Jacareí - SP

SÍNTESE COMPORTAMENTAL DE CIRCUITOS ASSÍNCRONOS OTIMIZADOS

Kledermon Garcia

Composição da Banca Examinadora:

Prof. Dr. Alexis Fabrício Tinoco Salazar	Presidente	- ITA
Prof. Dr. Roberto d'Amore	Orientador	- ITA
Prof. Dr. Duarte Lopes de Oliveira	Coorientador	- ITA
Prof. Dr. Wagner Chiepa Cunha	Membro Interno	- ITA
Prof. Dr. Ney Laert Vilar Calazans	Membro Externo	- PUCRS

ITA

Dedico este trabalho a minha esposa Sandra e
aos meus filhos João Vitor e Pedro Lucas
pelo apoio e paciência.

Agradecimentos

Ao orientador Prof. Roberto d'Amore pelo apoio nas revisões do trabalho, sempre feitas com presteza e precisão.

Ao coorientador Prof. Duarte Lopes de Oliveira pelas diversas sugestões de melhorias.

*"Como dizem os construtores, as pedras maiores
não ficariam bem assentadas sem as menores."*

Platão (428-348 a.C.)

Resumo

O projeto de circuitos digitais assíncronos é uma alternativa atraente ao estilo clássico, síncrono, pois, a eliminação do sinal de *clock* soluciona os problemas relacionados ao *clock skew*, oferece um potencial ganho em desempenho, redução do consumo de potência além de facilitar o roteamento, visto que a implementação é feita por módulos.

Este trabalho propõe um método para geração de circuitos assíncronos no estilo *bundled-data*, que é constituído de um controlador assíncrono e um *datapath* que utiliza componentes convencionais síncronos. A otimização proposta para o controlador busca eliminar transições de estado inseridas para satisfazer as propriedades de polaridade dos sinais de controle, mas que não executam nenhuma operação de dados. Essas transições ocorrem frequentemente em laços de repetição, o que gera uma queda expressiva no desempenho do circuito. Para avaliar o método proposto, foi desenvolvida uma ferramenta para a síntese do circuito assíncrono que gera o circuito correspondente descrito em linguagem VHDL a partir de uma descrição textual contendo as entradas, saídas e expressões. Os testes experimentais da ferramenta foram executados em dispositivos FPGAs. Os resultados demonstram a viabilidade do método proposto. Foi obtida uma sensível melhora no desempenho do circuito quando comparado ao análogo assíncrono sem as otimizações propostas.

Abstract

The design of asynchronous digital circuit is an attractive alternative to the classic, synchronous style. The clock signal elimination, solves the clock skew problem, offers a potential gain in performance, reduces power consumption and facilitates the routing, as the design employs modules with local communication.

This dissertation proposes a method to generate asynchronous circuits for the bundled-data design style, which consists of an asynchronous controller and a datapath which uses synchronous conventional components. The proposed controller optimization aims to eliminate state transitions which do not perform any data operation, but are inserted to satisfy the polarity properties of control signals. These transitions often occur in loops, which generates a significant drop in circuit performance. To evaluate the proposed method, a tool for the synthesis of asynchronous circuit has been developed. It generates the corresponding circuit, described in VHDL language, from a textual description containing the inputs, outputs and expressions. Experimental tests were performed on FPGA devices. The results demonstrate the feasibility of the proposed method. A significant improvement in circuit performance was achieved in comparison with analogous asynchronous version without optimizations.

Lista de Figuras

FIGURA 1.1 – Arquitetura GALS usando <i>wrapper</i> assíncrono (MUTTERSBACH, 2001)...	24
FIGURA 1.2 – a) Módulo assíncrono utilizando circuito gerado por BUDASYN. b) Composição de um sistema mais complexo através de módulos.	25
FIGURA 2.1 – Formas de associar os atrasos ao circuito: a) Atraso nas portas. b) Atraso nas linhas. c) Atraso nas portas e linhas.	29
FIGURA 2.2 – Classificação dos <i>hazards</i> combinacionais.	31
FIGURA 2.3 – a) <i>Hazard</i> lógico estático-1. b) Circuito livre de <i>hazard</i> lógico estático-1.	31
FIGURA 2.4 a) <i>Hazard</i> lógico dinâmico. b) Circuito livre <i>hazard</i> lógico dinâmico.	32
FIGURA 2.5 – a) <i>Hazard</i> funcional estático. b) <i>Hazard</i> funcional dinâmico.	33
FIGURA 2.6 – a) <i>Hazard</i> essencial de estado estável. b) <i>Hazard</i> essencial transiente.	34
FIGURA 2.7 – a) Corrida crítica. b) Corrida não crítica.	34
FIGURA 2.8 – a) Canal de comunicação. b) Protocolo de 4-fases. c) Protocolo de 2-fases...	35
FIGURA 2.9 – Dessincronização.	37
FIGURA 2.10 – <i>Micropipeline</i> (SUTHERLAND, 1989).	38
FIGURA 2.11 – Decomposição em controlador e <i>datapath</i>	39
FIGURA 2.12 – a) Codificação <i>dual-rail</i> . b) Porta lógica E <i>dual-rail</i>	40
FIGURA 2.13 – a) Elemento-C e comportamento simulado. b) Diagrama de tempos. c) <i>Petri</i> <i>net</i> . d) STG.	41
FIGURA 2.14 – Especificação modo rajada (BM).	42
FIGURA 2.15 – Especificação modo rajada estendido (XBM).	43
FIGURA 3.1 – Etapas da síntese comportamental.	45
FIGURA 3.2 – a) Descrição comportamental. b) GFD. c) GFC.	46
FIGURA 3.3 – Tempos de início possíveis para as operações do GFD.	49

FIGURA 3.4 – Grafo de fluxo de dados escalonado por PLI.	52
FIGURA 3.5 – Grafo de fluxo de dados e possíveis tempos de início das operações.....	53
FIGURA 3.6 – Intervalo de tempo das operações: a) Representação textual. b) Representação gráfica.....	54
FIGURA 3.7 – Grafo de fluxo de dados escalonado por <i>force-directed</i>	59
FIGURA 3.8 – GFD com parâmetros STTF e STTS.	61
FIGURA 3.9 – Expansão parcial de um grafo com três operações.	63
FIGURA 3.10 – Grafo de fluxo de dados a ser escalonado pelo algoritmo <i>branch-and-bound</i>	63
FIGURA 3.11 – Grafo de fluxo de dados escalonado por <i>branch-and-bound</i>	66
FIGURA 3.12 – Assinalamento de unidades funcionais.	67
FIGURA 4.1 – Fluxo do projeto BUDASYN.	69
FIGURA 4.2 – Exemplo de descrição usando a linguagem proposta.	70
FIGURA 4.3 – a) Grafo de fluxo de dados. b) Grafo de fluxo de controle.	72
FIGURA 4.4 – Exemplos de arquivos: a) Alocação de recursos. b) Latências.	74
FIGURA 4.5 – Exemplo de escalonamento: a) Escalonamento MTRR. b) Escalonamento MRRT.	75
FIGURA 4.6 – Pseudocódigo do algoritmo de assinalamento de UFs.....	76
FIGURA 4.7 – a) Escalonamento. b) Tempo de vida das operações. c) Descrição comportamental.	77
FIGURA 4.8 – Pseudocódigo do algoritmo de assinalamento de registradores.	78
FIGURA 4.9 – a) Escalonamento. b) Tempo de vida das variáveis. c) Descrição comportamental.	79
FIGURA 4.10 – Pseudocódigo para geração do <i>datapath</i>	80
FIGURA 4.11 – Descrição RTL do <i>datapath</i>	81

FIGURA 4.12 – Arquitetura do controlador proposto.....	82
FIGURA 4.13 – Geração do XBM inicial.....	84
FIGURA 4.14 – Exemplo de transformação da especificação XBM: a) Especificação XBM inicial. b) Especificação XBM final.....	85
FIGURA 4.15 – Pseudocódigo para geração da especificação XBM otimizada.	86
FIGURA 4.16 – Especificação XBM inicial.....	87
FIGURA 4.17 – Especificação XBM final.	88
FIGURA 4.18 – Fluxo de geração da descrição VHDL.	89
FIGURA 4.19 – Elementos de atraso.	90
FIGURA 5.1 – 1 ciclo do algoritmo TEA (MAHDI, 2011).....	91
FIGURA 5.2 – Algoritmo TEA descrito na linguagem proposta.....	92
FIGURA 5.3 – Detalhes da implementação obtidos no processo de síntese comportamental para o algoritmo TEA: a) Identificação das operações. b) Representação textual do escalonamento. c) Assinalamento de UFs. d) Assinalamento de registradores.	94
FIGURA 5.4 – Representação textual do <i>datapath</i> para o algoritmo TEA.	95
FIGURA 5.5 – XBM inicial, TEA.....	96
FIGURA 5.6 – Simulação do circuito – XBM inicial.	97
FIGURA 5.7 – XBM final, TEA.	98
FIGURA 5.8 – Simulação do Circuito – XBM final.	99
FIGURA A.1 – Opções de Escalonamento.....	112
FIGURA A.2 – Representação Gráfica do GFD não Escalonado.....	113
FIGURA A.3 – Resultado do Escalonamento; Assinalamentos; <i>Datapath</i>	114
FIGURA A.4 – Escolha da Especificação do Controlador.	115
FIGURA A.5 – Adição dos Elementos de Atraso.	116
FIGURA B.1 – Arquivo de Entrada. Diffeq V.I. MRRT.....	117

FIGURA B.2 – Simulação. Diffeq V.I. MRRT.....	117
FIGURA B.3 – Arquivo de Entrada. Diffeq V.F. MRRT.....	118
FIGURA B.4 – Simulação. Diffeq V.F. MRRT.....	118
FIGURA B.5 – Arquivo de Entrada. EWF. MRRT.....	119
FIGURA B.6 – Simulação. EWF. MRRT.....	119
FIGURA B.7 – Arquivo de Entrada. FAT V.I. MRRT.....	120
FIGURA B.8 – Simulação. FAT V.I. MRRT.....	120
FIGURA B.9 – Arquivo de Entrada. FAT V.F. MRRT.....	120
FIGURA B.10 – Simulação. FAT V.F. MRRT.....	121
FIGURA B.11 – Arquivo de Entrada. SQR. MRRT.....	121
FIGURA B.12 – Simulação. SQR. MRRT.....	121
FIGURA B.13 – Arquivo de Entrada. GCD. MRRT.....	122
FIGURA B.14 – Simulação. GCD. MRRT.....	122
FIGURA B.15 – Arquivo de Entrada. FIR. MRRT.....	122
FIGURA B.16 – Simulação. FIR. MRRT.....	123
FIGURA B.17 – Arquivo de Entrada. IIR. MRRT.....	123
FIGURA B.18 – Simulação. IIR. MRRT.....	124
FIGURA B.19 – Arquivo de Entrada. TEA-e V.I. MRRT.....	124
FIGURA B.20 – Simulação. TEA-e V.I. MRRT.....	125
FIGURA B.21 – Arquivo de Entrada. TEA-e V.F. MRRT.....	125
FIGURA B.22 – Simulação. TEA-e V.F. MRRT.....	125
FIGURA B.23 – Arquivo de Entrada. TEA-d V.I. MRRT.....	126
FIGURA B.24 – Simulação. TEA-d V.I. MRRT.....	126
FIGURA B.25 – Arquivo de Entrada. TEA-d V.F. MRRT.....	127
FIGURA B.26 – Simulação. TEA-d V.F. MRRT.....	127

FIGURA B.27 – Arquivo de Entrada. Divisão. MRRT.....	128
FIGURA B.28 – Simulação. Divisão. MRRT.....	128
FIGURA B.29 – Arquivo de Entrada. DotProd V.I. MRRT.....	128
FIGURA B.30 – Simulação. DotProd V.I. MRRT.....	129
FIGURA B.31 – Arquivo de Entrada. DotProd V.F. MRRT.....	129
FIGURA B.32 – Simulação. DotProd V.F. MRRT.....	129
FIGURA B.33 – Arquivo de Entrada. Fibonacci. V.I. MRRT.....	130
FIGURA B.34 – Simulação. Fibonacci. V.I. MRRT.....	130
FIGURA B.35 – Arquivo de Entrada. Fibonacci. V.F. MRRT.....	131
FIGURA B.36 – Simulação. Fibonacci. V.F. MRRT.....	131
FIGURA B.37 – Arquivo de Entrada. Wavelet. MRRT.....	132
FIGURA B.38 – Simulação. Wavelet. MRRT.....	132
FIGURA B.39 – Arquivo de Entrada. Diffeq. MTRR.....	133
FIGURA B.40 – Simulação. Diffeq. MTRR.....	133
FIGURA B.41 – Arquivo de Entrada. EWF. MTRR.....	134
FIGURA B.42 – Simulação. EWF. MTRR.....	134
FIGURA B.43 – Arquivo de Entrada. FAT. V.I. MTRR.....	135
FIGURA B.44 – Simulação. FAT. V.I. MTRR.....	135
FIGURA B.45 – Arquivo de Entrada. FAT. V.F. MTRR.....	135
FIGURA B.46 – Simulação. FAT. V.F. MTRR.....	136
FIGURA B.47 – Arquivo de Entrada. SQR. V.I. MTRR.....	136
FIGURA B.48 – Simulação. SQR. V.I. MTRR.....	136
FIGURA B.49 – Arquivo de Entrada. SQR. V.F. MTRR.....	137
FIGURA B.50 – Simulação. SQR. V.F. MTRR.....	137
FIGURA B.51 – Arquivo de Entrada. GCD. MTRR.....	138

FIGURA B.52 – Simulação. GCD. MTRR.....	138
FIGURA B.53 – Arquivo de Entrada. FIR. MTRR.....	138
FIGURA B.54 – Simulação. FIR. MTRR.....	139
FIGURA B.55 – Arquivo de Entrada. IIR. MTRR.....	139
FIGURA B.56 – Simulação. IIR. MTRR.....	140
FIGURA B.57 – Arquivo de Entrada. TEA-e. MTRR	140
FIGURA B.58 – Simulação. TEA-e. MTRR	141
FIGURA B.59 – Arquivo de Entrada. TEA-d. MTRR.....	141
FIGURA B.60 – Simulação. TEA-d. MTRR	141
FIGURA B.61 – Arquivo de Entrada. Divisão. MTRR.....	142
FIGURA B.62 – Simulação. Divisão. MTRR.....	142
FIGURA B.63 – Arquivo de Entrada. DotProd. V.I. MTRR.....	142
FIGURA B.64 – Simulação. DotProd. V.I. MTRR.....	143
FIGURA B.65 – Arquivo de Entrada. DotProd. V.F. MTRR.....	143
FIGURA B.66 – Simulação. DotProd. V.F. MTRR.....	143
FIGURA B.67 – Arquivo de Entrada. Fibonacci. MTRR.....	144
FIGURA B.68 – Simulação. Fibonacci. MTRR.....	144
FIGURA B.69 – Arquivo de Entrada. Wavelet. MTRR.....	145
FIGURA B.70 – Simulação. Wavelet. MTRR.....	145

Lista de Tabelas

TABELA 3.1 – Representação textual do GFD escalonado.	65
TABELA 4.1 – Conjunto de operadores permitidos por BUDASYN.	71
TABELA 6.1 – Resultados obtidos para os controladores na versão inicial e versão final.	102
TABELA 6.2 – Resultados obtidos para síntese e simulação, comparando entre versão inicial e final.	103
TABELA 6.3 – Resultados obtidos para síntese e simulação, comparando entre escalonamento MRRT e MTRR.	105

Lista de Abreviaturas e Siglas

ASAP	<i>As Soon as Possible</i>
ALAP	<i>As Late as Possible</i>
BM	<i>Burst Mode</i>
CPLD	<i>Complex Programmable Logic Device</i>
FPGA	<i>Field Programmable Gate Array.</i>
GALS	Globalmente Assíncrono e Localmente Síncrono
GDA	Grafo Dirigido Acíclico
GFC	Grafo de Fluxo de Controle
GFD	Grafo de Fluxo de Dados
GFDC	Grafo de Fluxo de Dados e Controle
lp_solve	<i>Software</i> solucionador de programação linear inteira
Matlab	<i>Software</i> para calculo numérico
MIC	<i>Multiple Input Change</i>
MOC	<i>Multiple Output Change</i>
MRRT	Minimização de Recursos com Restrição de Tempo
MTRR	Minimização de Tempo com Restrição de Recursos
PLI	Programação Linear Inteira
PN	<i>Petri Net</i>
RTL	<i>Register Transfer Level.</i>
SICARELO	Ferramenta para síntese de controladores assíncronos
SIC	<i>Single Input Change</i>
SOC	<i>Single Output Change</i>
STG	<i>Signal transition graph</i>

STTS	<i>Shortest Time To Start</i>
STTF	<i>Shortest Time To Finish</i>
UF	Unidade Funcional
ULA	Unidade Lógica Aritmética
XBM	<i>Extended Burst Mode</i>

Sumário

1	INTRODUÇÃO	21
1.1	Motivação	21
1.1.1	Limitações do Síncrono	21
1.1.2	Vantagens do Assíncrono	22
1.1.3	Limitações do Assíncrono	22
1.2	Sistemas Heterogêneos	23
1.3	Objetivo	24
1.4	Organização da Dissertação	26
1.5	Contribuições da Dissertação	26
2	CIRCUITOS ASSÍNCRONOS - CONCEITOS	28
2.1	Modos de Operação de um Circuito Assíncrono	28
2.2	Modelos de Atraso	28
2.2.1	<i>Speed-Independent</i> (SI)	29
2.2.2	<i>Delay-Insentitive</i> (DI).....	29
2.2.3	<i>Quasi-Delay-Insensitive</i> (QDI).....	30
2.2.4	<i>Bounded-Delay</i> (BD).....	30
2.3	Hazard	30
2.3.1	<i>Hazard</i> Combinacional.....	30
2.3.2	<i>Hazard</i> Funcional.....	32
2.3.3	<i>Hazard</i> Sequencial	33
2.4	Protocolo de Comunicação <i>Bundled-Data</i>	35
2.5	Síntese Comportamental	36
2.5.1	Tradução Direta.....	36
2.5.2	Dessincronização.....	37
2.5.3	<i>Pipeline</i> Assíncrono.....	38
2.5.4	Decomposição	39
2.6	Controladores Assíncronos	40
3	MÉTODOS PARA SÍNTESE COMPORTAMENTAL POR DECOMPOSIÇÃO .44	
3.1	GFDC	45
3.2	Alocação de Recursos (<i>Resource Allocation</i>)	46

3.3	Escalonamento (<i>Scheduling</i>)	46
3.3.1	Escalonamento Assíncrono: Programação Linear Inteira (PLI)	48
3.3.2	Escalonamento Assíncrono: <i>Force-Directed</i>	53
3.3.3	Escalonamento Assíncrono: <i>Branch-and-Bound</i>	59
3.4	Assinalamento de Unidades Funcionais (<i>Component Binding</i>)	66
3.5	Assinalamento de Registradores	68
3.6	Geração do <i>Datapath</i>	68
3.7	Geração da Especificação do Controlador	68
4	SÍNTESE AUTOMÁTICA DE SISTEMAS ASSÍNCRONOS POR DECOMPOSIÇÃO	69
4.1	Arquivo de Entrada	69
4.2	Extração do GFD e GFC	71
4.3	Escalonamento	73
4.4	Assinalamento de Unidades Funcionais	75
4.5	Assinalamento de Registradores	78
4.6	Geração do <i>Datapath</i>	80
4.7	Controlador	81
4.8	Geração da Especificação XBM Inicial	82
4.9	Geração da Especificação XBM Final	84
4.10	Conversão para Código VHDL	88
4.11	Implementação em FPGA	89
5	ESTUDO DE UMA APLICAÇÃO – CRIPTOGRAFIA TEA	91
5.1	Controlador – Versão Inicial	95
5.2	Controlador – Versão Final	97
6	ESTUDO DE <i>BENCHMARKS</i>: RESULTADOS	100
6.1	Descrição dos <i>Benchmarks</i>	100
6.2	Discussão	100
7	CONCLUSÃO	106
7.1	Trabalhos Futuros	106
	REFERÊNCIAS	107
	APÊNDICE A – USO DA FERRAMENTA, PASSO A PASSO	111
A.1	Arquivos de Entrada	111
A.2	Informações do GFD	111
A.3	Informações do Escalonamento e <i>Datapath</i>	113

A.4	Especificação do Controlador	113
A.5	Simulação no Quartus II	115
	APÊNDICE B – SIMULAÇÕES	117
B.1	Simulações Utilizando Escalonamento MRRT.....	117
B.2	Simulações Utilizando Escalonamento MTRR.....	133

1 Introdução

1.1 Motivação

O projeto de circuitos digitais emprega tradicionalmente o paradigma síncrono. O projeto síncrono oferece simplicidade no projeto, pois pressupõe um modelo discreto de tempo ao definir um sinal de *clock* distribuído através do circuito. Entretanto, com o crescente aumento da frequência do *clock* e diminuição do tamanho dos transistores, o sinal de *clock* passou a apresentar novos desafios aos projetistas. Em busca de soluções uma das alternativas promissoras é o projeto no paradigma assíncrono, que elimina o sinal de *clock* e realiza a comunicação e sincronização dos componentes através de protocolos de comunicação *handshake*.

1.1.1 Limitações do Síncrono

Apesar do sinal de *clock* proporcionar a simplificação do projeto, este torna-se um limitador quando operando em alta frequência e em circuitos de maior complexidade. Alguns dos problemas mais comuns estão apresentados a seguir.

Defasagem do sinal de *clock*: Em circuitos de alta complexidade e alta velocidade, o tempo de propagação do sinal de *clock* é afetado por diversos fatores, tais como o comprimento dos fios, variações de temperatura e variação das capacitâncias associadas ao *clock*. Para minimizar este problema, é necessário o balanceamento do sinal do *clock*, o que envolve a adição de *buffers* e o planejamento da rota de distribuição deste sinal ao longo do circuito integrado, o que causa uma redução no desempenho e um aumento no tamanho do circuito (JACOBSON, 1996, p.13).

Consumo de energia: A distribuição do sinal do *clock* ao longo do circuito de forma balanceada exige etapas de amplificação que estão constantemente consumindo energia. Há propostas de redução do consumo de energia pelo desligamento do *clock*, evitando que este acione partes desnecessárias do circuito. Esta técnica, entretanto, introduz uma maior defasagem do sinal de *clock* (FUHRER, 1999, p.3).

Atraso definido pelo pior caso: A máxima frequência de operação de um sinal de *clock* é definida pelas condições de pior caso. Como resultado, quando o sistema está operando em

condições normais, o desempenho é limitado pelas condições de pior caso do projeto (NOWICK, 1995, p.2).

Modularidade: O modelo síncrono não acomoda com facilidade o método de projeto modular, onde pode-se adicionar novos componentes ao projeto sem necessidade de modificar o circuito anterior. Ao inserir ou modificar módulos do circuito, este deve satisfazer as condições do sinal do *clock* ou o sinal do *clock* deve se adequar ao novo módulo.

Interferência eletromagnética (EMI): Circuitos síncronos geram picos de emissão eletromagnética na frequência do sinal do *clock* e em suas harmônicas, o que pode causar interferência em outros circuitos próximos ou facilitar a fuga de informações estratégicas. A distribuição do sinal do *clock* entre os circuitos está entre as principais fontes de emissão de EMI, principalmente quando há caminhos ortogonais (ZAFAR, 2005, p.5).

1.1.2 Vantagens do Assíncrono

O projeto assíncrono oferece o potencial para a solução de diversos problemas encontrados no projeto tradicional síncrono, entre eles pode-se citar a eliminação dos problemas de defasagem e roteamento do sinal do *clock*, redução da interferência eletromagnética associada ao sinal de *clock* e redução de potência, pois só há atividade nos circuitos quando há necessidade de processamento. Os circuitos assíncronos possuem, também, a vantagem de serem inerentemente modulares, ou seja, são construídos em módulos que se comunicam através de protocolos de comunicação, portanto são capazes de adaptarem sem necessidade de uma nova análise temporal. Pode-se destacar como vantagem, para uma classe de circuitos assíncronos, a operação altamente robusta, adaptando-se às variações das condições de funcionamento.

1.1.3 Limitações do Assíncrono

Atualmente, a maior limitação ao uso dos circuitos assíncronos está na falta de ferramentas comerciais para a síntese de alto nível. A maioria das ferramentas disponíveis é desenvolvida no meio acadêmico (NIELSEN, 2004; DUARTE, 2010). Isso se deve a dificuldades encontradas no projeto de circuitos assíncronos, conforme discute-se a seguir.

Método de projeto: Existem diversas propostas e estilos de projetos diferentes, o que acaba dificultando para os projetistas e sobretudo as empresas de EDA (*Electronic Design Automation*) encontrarem a melhor solução.

Hazard: Circuitos assíncronos são sensíveis a pulsos espúrios (*glitches*), que podem levar a um mau funcionamento. Quando o circuito apresenta a possibilidade de gerar *glitches*, então se diz que este tem *hazard*. Portanto, há a necessidade de um projeto mais robusto, capaz de identificar e limitar possíveis *glitches*. Devido a essa necessidade, os circuitos assíncronos são geralmente maiores e consomem maior potência estática que seus equivalentes síncronos.

Latência dos canais de comunicação: A de latência entre os módulos pode afetar significativamente o desempenho do circuito.

1.2 Sistemas Heterogêneos

O avanço tecnológico dos sistemas digitais permite que sejam integrados sistemas completos com processadores, periféricos e memórias em um único circuito integrado. Esse tipo de sistema, é conhecido como *system-on-chip* (SoC), onde o reuso de projetos é facilitado pois o circuito é decomposto em módulos independentes, (FERRINGER; FUCHS; STEININGER; KEMPF, 2011). Esses sistemas consistem de múltiplos módulos que são sincronizados independentemente: alguns módulos podem ser síncronos enquanto outros podem ser assíncronos. SoCs desempenham um papel importante em sistemas: automotivo, comunicação (telefonia móvel, *tablets*, etc) e multimídia tais como câmeras, televisões, etc (ENGEL; KUZ; PETERS; RUOCCO, 2004). Como exemplo, a implementação de uma câmera de vídeo em um único circuito integrado (*camera-on-a-chip*) inclui um sensor CMOS, processador de áudio e vídeo, controladores para ajuste das lentes, entre outros.

Conforme comentado, um grande desafio para sistemas digitais complexos é a distribuição do sinal de *clock* que, além do problema da defasagem, pode ser o principal consumidor de energia do projeto (FERRINGER; FUCHS; STEININGER; KEMPF, 2011). Em busca de novas alternativas, pode-se destacar a proposta de projeto Globalmente Assíncrona e Localmente Síncrona (GALS) (CHAPIRO, 1984), pois tem potencial para combinar os benefícios do paradigma síncrono e assíncrono. Para permitir o correto funcionamento do módulo localmente síncrono em um ambiente assíncrono, cada módulo recebe um circuito adicional conhecido como *wrapper*, mostrado na FIGURA 1.1. Os módulos síncronos são projetados utilizando as técnicas tradicionais do paradigma síncrono. Cada um dos módulos opera com um sinal de *clock* independente, o que minimiza os problemas relacionados à defasagem e a ligação entre os módulos é feita de forma assíncrona.

Apesar do grande interesse por parte da comunidade científica na solução GALS, há um custo relacionado à interface de comunicação, que pode gerar uma queda no desempenho. Muitas das soluções propostas para GALS, como é o caso da arquitetura proposta por Muttersbach (2001), exigem osciladores em anel para gerar o *clock* local, que apresentam desvantagens do ponto de vista industrial, pois como são sensíveis a variações do processo, exigem uma calibração bastante cuidadosa. Além disso, osciladores em anel contribuem para elevação do consumo de energia, através da comutação contínua dos inversores (KRSTIC *et al.*, 2007).

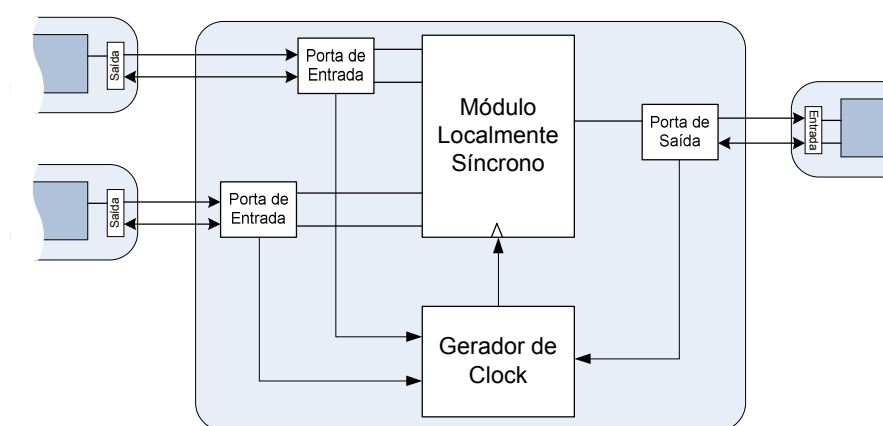


FIGURA 1.1 – Arquitetura GALS usando *wrapper* assíncrono (MUTTERSACH, 2001).

1.3 Objetivo

Este trabalho propõe um novo método e o desenvolvimento de uma ferramenta chamada BUDASYN para geração de circuitos assíncronos no estilo decomposição, onde o circuito é separado em um controlador centralizado e *datapath*. O *datapath* é gerado de forma otimizada, passando por diversas etapas de otimização que são executadas no processo de síntese comportamental. O controlador é assíncrono e opera empregando o protocolo de comunicação *bundled data*, onde cada bit é representado por um fio, permitindo que componentes do estilo síncrono sejam utilizados, tais como, unidades funcionais, registradores e multiplexadores. Para cada grupo de recursos do *datapath* responsável por executar um conjunto de operações, um par de fios adicionais (*request* e *acknowledge*) é agrupado, onde *request* indica o início de uma operação e *acknowledge* indica o término. O tempo de execução da operação é garantido pela adição de um elemento de atraso entre cada par *request* e *acknowledge*. A implementação de circuitos no estilo “*bundled data*”, assume que as operações são executadas no máximo atraso de execução para o recurso utilizado, o

que limita a possibilidade de extrair o máximo desempenho dos circuitos assíncronos. Entretanto, este estilo de projeto pode ser obtido mais facilmente se comparado com o estilo “*dual-rail*”, onde cada bit de dados é codificado por um par de fios, além de permitir a implementação em dispositivos comercialmente disponíveis como FPGAs e CPLDs. Além disso, o consumo de potência e uso de área são melhores que outras implementações.

Os circuitos gerados por esta proposta podem ser utilizados para compor sistemas mais complexos como, por exemplo, em sistemas heterogêneos. Os circuitos podem ser inseridos como módulos assíncronos, como o apresentado na Figura 1.2(a), onde a comunicação com o ambiente é feita através de protocolo de comunicação de duas fases utilizando um conversor de protocolos (GARCIA et al, 2014). A Figura 1.2(b) apresenta um exemplo de sistema composto por módulos assíncronos, onde a comunicação entre os módulos é feita por protocolo de *handshake*.

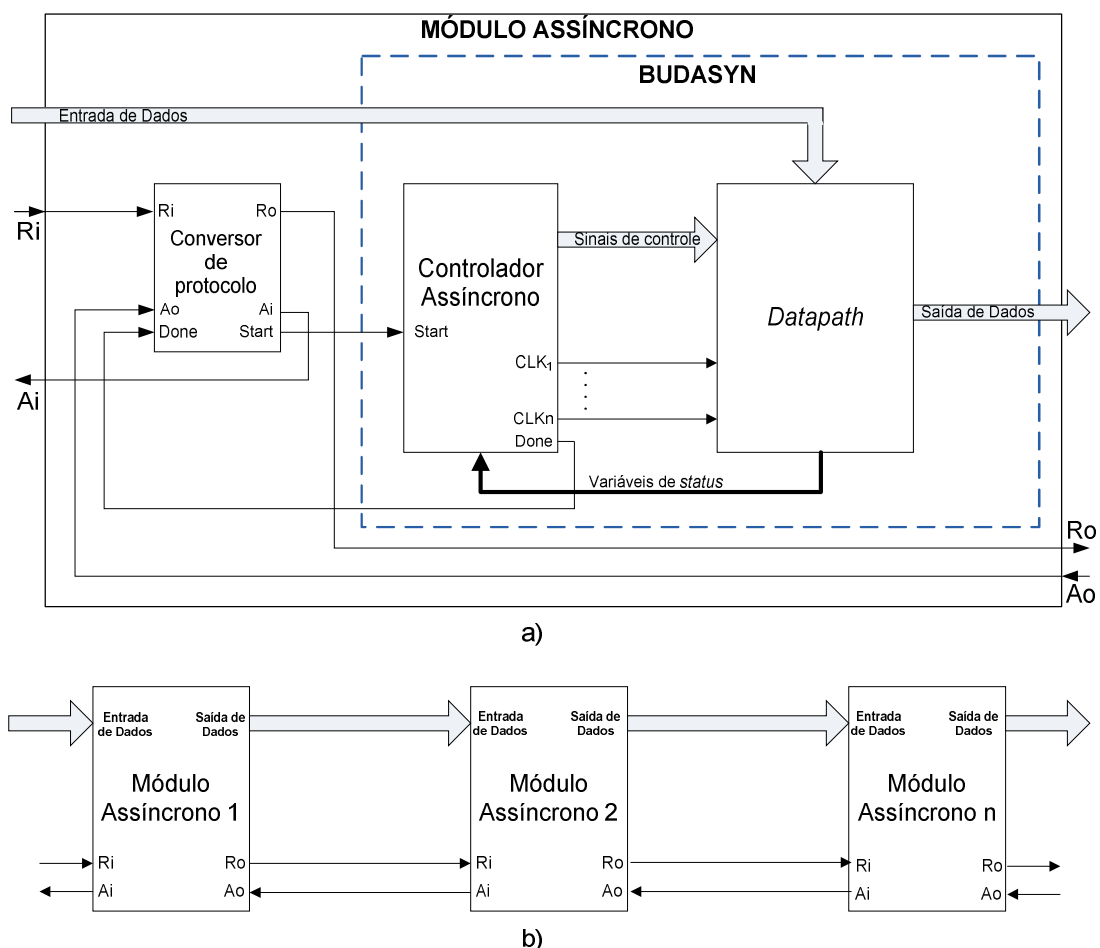


FIGURA 1.2 – a) Módulo assíncrono utilizando circuito gerado por BUDASYN.

b) Composição de um sistema mais complexo através de módulos.

1.4 Organização da Dissertação

Uma descrição mais detalhada dos conceitos envolvidos na síntese de circuitos assíncronos é apresentada no Capítulo 2.

No Capítulo 3, exibe-se as principais etapas para síntese comportamental baseado em decomposição, apresentando com maior ênfase um estudo dos métodos de escalonamento assíncrono. O Capítulo 4 apresenta o procedimento para síntese automática comportamental proposta.

No Capítulo 5 um *benchmark* de criptografia é utilizado para estudo de caso, onde este é sintetizado em duas versões, uma utilizando a otimização proposta para o controlador e outra sem a referida otimização. Uma breve comparação é feita entre as duas versões.

O Capítulo 6 apresenta os resultados para um conjunto de 13 *benchmarks* sintetizados através da ferramenta proposta nesta dissertação, e é feita uma análise dos resultados obtidos. No Capítulo 7 estão às conclusões e algumas sugestões de trabalhos futuros.

1.5 Contribuições da Dissertação

Esta dissertação apresenta as seguintes contribuições:

- Otimização proposta para o controlador, projetado segundo o modelo XBM (*extended burst mode*) com protocolo de comunicação de duas fases. A utilização deste protocolo com *datapath* síncrono pode inserir transições de estado que não executam nenhuma operação nos dados, mas são necessárias para garantir que as propriedades da especificação XBM sejam satisfeitas. No entanto, quando estas transições ocorrem em um laço de repetição, tornam-se indesejadas, pois a cada iteração do laço uma transição ocorrerá sem executar nenhuma operação nos dados. Nesta dissertação, as transições que não executam operações nos dados são nomeadas como transições mortas. A otimização proposta nesta dissertação busca eliminar as transições mortas inserindo em seu lugar uma cópia do laço em questão, de modo que as fases dos sinais do controlador sejam compatíveis.
- Para a correta operação do controlador no protocolo de duas fases com o *datapath*, propõe-se uma nova arquitetura baseada em portas XNOR e elementos de atraso.
- Uma nova ferramenta para a síntese de alto nível, contribuindo, assim, para uma melhor aceitação dos circuitos assíncronos, por acrescentar mais uma nova opção ao pequeno número de ferramentas disponíveis. A ferramenta desenvolvida permite, também,

validar e demonstrar os benefícios do método proposto, além de auxiliar na comparação de outros métodos. Os circuitos gerados operam como módulos assíncronos, podendo ser utilizados para compor circuitos mais complexos.

- Análise das principais propostas de escalonamento assíncrono. Elas consistem em um processo da síntese comportamental no qual se determina a ordem de execução das operações aritméticas e lógicas extraídas da descrição comportamental de uma determinada especificação.

2 Circuitos Assíncronos - Conceitos

Este Capítulo apresenta os conceitos básicos envolvidos no projeto de circuitos assíncronos, iniciando pelos modos de operação e classificação quanto aos modelos de atraso, em seguida apresentam-se os tipos de riscos (*hazards*) e o protocolo de comunicação *bundled data*. Por fim, é apresentada uma introdução a cerca dos principais modelos de síntese comportamental de circuitos assíncronos.

2.1 Modos de Operação de um Circuito Assíncrono

O modo de operação especifica quais as restrições a que o circuito está sujeito ao comunicar-se com o ambiente para operar corretamente. As restrições para as entradas são:

- *Single Input Change* (SIC) – apenas uma entrada deve mudar por vez. Mudanças consecutivas nas entradas devem ser separadas por um intervalo mínimo de tempo.
- *Multiple Input Change* (MIC) – pode-se mudar mais de uma das entradas por vez, mas um intervalo de tempo mínimo deve ser respeitado antes que um novo conjunto de entradas mude novamente.

A comunicação com o ambiente externo pode ocorrer de dois modos:

- Modo Entrada/Saída – Após uma mudança no valor da saída do circuito, é permitida uma nova excitação na entrada. O modo entrada/saída reduz as restrições de interação com o ambiente ao permitir a concorrência entre entrada e saída.
- Modo Fundamental – Somente poderá ocorrer uma nova excitação do circuito quando o mesmo estiver estabilizado, ou seja, todas as entradas, saídas e sinais internos estiverem com valores fixos. Adicionalmente, as entradas devem obedecer às restrições SIC. O modo fundamental pode ser expandido para Modo Fundamental Generalizado, caso alguma das entradas obedeça à restrição MIC.

2.2 Modelos de Atraso

Os circuitos digitais reais estão sujeitos a diversos atrasos inerentes aos dispositivos eletrônicos. Para viabilizar o projeto de um circuito digital é necessária a modelagem destes atrasos, como forma de simplificação do projeto. Os atrasos podem estar associados às portas,

às linhas ou a ambos, conforme apresentado na Figura 2.1. A modelagem dos atrasos dos circuitos pode ser dividida basicamente em três categorias. A primeira, denominada *bounded gate and wire delay*, considera que as portas e linhas possuem atrasos dentro de limites máximos e mínimos. Circuitos síncronos, por exemplo, utilizam este modelo de atraso porque o atraso máximo não deve exceder a largura do período do sinal do *clock* (DUARTE, 2010, p.43). A segunda categoria, conhecida como: *unbounded gate delay*, considera que as portas podem ter atrasos arbitrários, porém finitos, e as linhas devem ter atraso nulo ou desprezível. Por fim, a terceira categoria, identificada pelo termo: *unbounded gate and wire delay*, considera que o atraso das linhas e das portas é arbitrário e finito.

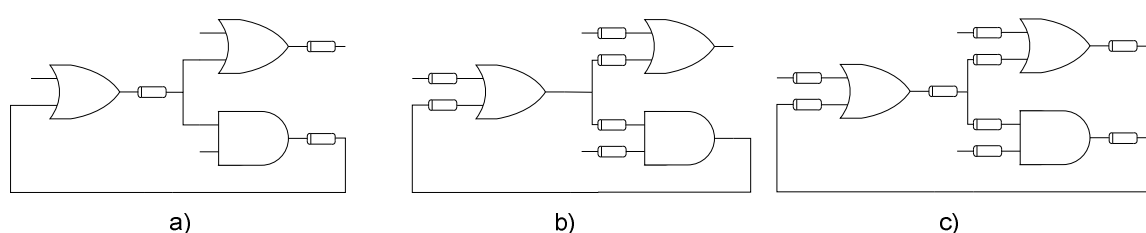


FIGURA 2.1 – Formas de associar os atrasos ao circuito: a) Atraso nas portas. b) Atraso nas linhas. c) Atraso nas portas e linhas.

2.2.1 *Speed-Independent (SI)*

São circuitos que operam corretamente no modelo de atraso *unbounded gate delay*. O modelo foi proposto em 1959 por David Muller e posteriormente estendido por (DAVIS; NOWICK, 1997).

2.2.2 *Delay-Insentitive (DI)*

Estes circuitos são projetados para operar corretamente no modelo de atraso *unbounded gate and wire delay*. O modelo DI abrange um grupo restrito de circuitos devido à dificuldade de serem satisfeitas as condições do modelo. Segundo Martin (1990) apenas circuitos implementados exclusivamente a partir de *C-elements (latch C)* são inteiramente DI.

2.2.3 *Quasi-Delay-Insensitive (QDI)*

Os circuitos classificados como QDI são, em essência, circuitos DI. Porém, uma restrição foi adicionada aos fios, onde o atraso nas diferentes terminações de uma ramificação deve ser menor que o atraso mínimo das portas. Esta propriedade é chamada de *isochronic fork* (BEEREL; OZDAG; FERRETTI, 2010).

2.2.4 *Bounded-Delay (BD)*

Os circuitos classificados como BD operam corretamente no modelo de atraso *bounded gate and wire delay* e consideram que os atrasos dos componentes e dos fios do circuito do *datapath* são delimitados e menores que o atraso de um caminho lógico de referência, comumente chamado de *matched delay*. A lógica de referência é implementada usando a mesma biblioteca dos demais componentes do *datapath* e estão sujeitas as mesmas condições de operação. O modelo BD é utilizado nos circuitos onde a interação entre o *datapath* e o controlador ocorre utilizando protocolo *bundled-data*. Em geral, são obtidos circuitos mais rápidos, menores e com menor consumo de potência que seus equivalentes em modelos de atraso mais robustos, tais como DI e QDI.

2.3 *Hazard*

Hazard é o nome dado à possibilidade do circuito apresentar transições espúrias que podem ocorrer em um determinado sinal, seja durante a transição de um nível para outro ou em um nível estável. Nos circuitos síncronos não é necessário nenhum tratamento especial para o *hazard*, porque os sinais são amostrados somente após a estabilização do circuito, através do sinal de *clock*. Nos circuitos assíncronos, como não há amostragem, uma transição espúria pode ser interpretada como uma transição válida, levando o circuito a apresentar um mau funcionamento. O *hazard* pode ser classificado como: *hazard* combinacional, *hazard* funcional e *hazard* sequencial.

2.3.1 *Hazard Combinacional*

O *hazard* combinacional é também conhecido por *hazard* lógico e está relacionado à cobertura lógica. Pode ser classificado como estático ou dinâmico, veja Figura 2.2. O *hazard*

combinacional estático-1 ocorre quando espera-se que a saída permaneça em nível lógico alto, antes ou depois de alguma mudança na entrada, porém, um pulso negativo ocorre. Analogamente, o *hazard* combinacional estático-0 ocorre quando se espera que a saída permaneça em nível lógico baixo, porém, ocorre um pulso positivo. *Hazard* combinacional dinâmico é aquele que ocorre antecipadamente a uma transição de nível lógico, tanto de alto para baixo quanto de baixo para alto.

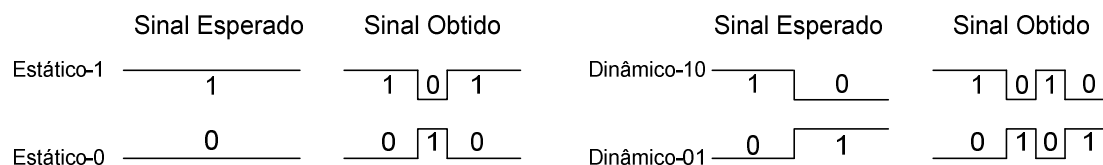


FIGURA 2.2 – Classificação dos *hazards* combinacionais.

O exemplo da Figura 2.3(a) demonstra uma situação onde a cobertura lógica pode levar a ocorrência do *hazard* estático. Observa-se através do mapa de Karnaugh, que após a cobertura mínima obtém-se a expressão lógica $F = A'B + AC$. Durante a transição das entradas de $ABC = 110$ para $ABC = 111$, espera-se que a saída F permaneça em nível lógico 1. Porém caso a porta lógica E ligada às entradas AC apresente um atraso de propagação maior que a porta lógica E ligada às entradas $A'B$, ocorrerá na saída um *hazard* estático-1. Uma implementação livre de *hazard* estático-1 está apresentada na Figura 2.3(b), onde uma função redundante é adicionada para garantir que durante a transição não ocorra a possibilidade de *hazard.*

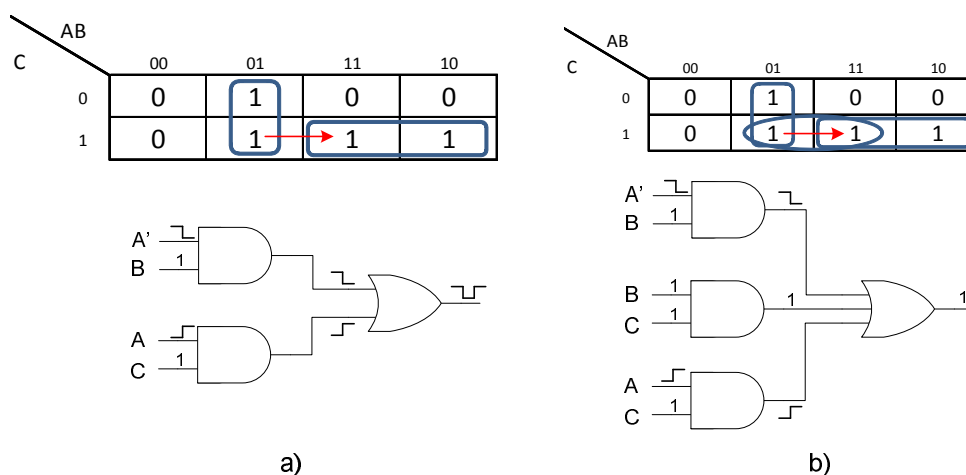


FIGURA 2.3 – a) *Hazard* lógico estático-1. b) Circuito livre de *hazard* lógico estático-1.

A Figura 2.4(a) mostra a possibilidade de ocorrência de *hazard* dinâmico. Quando ocorre a transição das entradas $ABC = 010$ para $ABC = 111$, o implicante k está interceptando irregularmente o implicante j , pois k não cobre o estado final $ABC = 111$. Este *hazard* pode ser solucionado modificando-se a cobertura lógica, conforme Figura 2.4(b), onde o implicante j não está mais sendo interceptado por k . Em algumas situações, a solução de um *hazard* dinâmico pode introduzir um *hazard* estático e vice-versa e as soluções para o *hazard* combinacional são válidas para o modelo de atraso *bounded gate and wire delay* (OLIVEIRA, 2004).

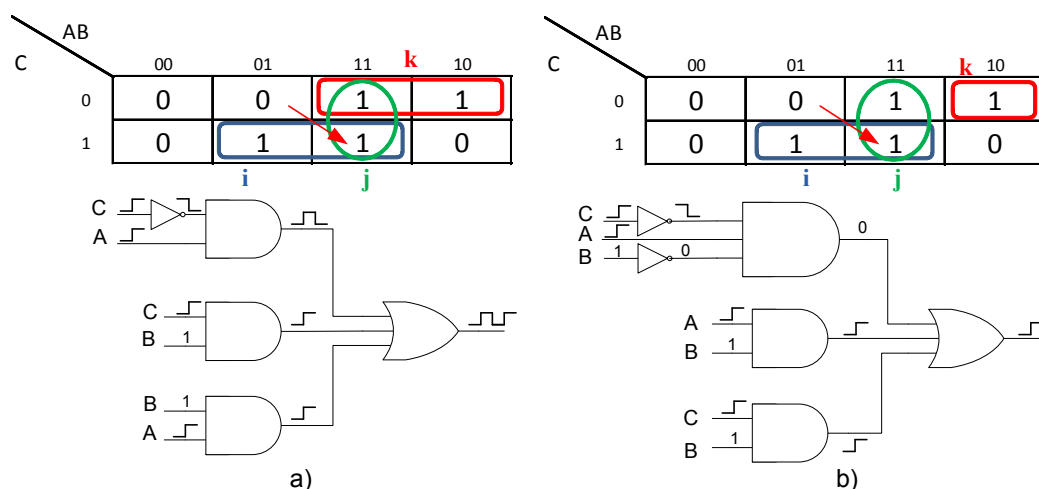


FIGURA 2.4 a) *Hazard* lógico dinâmico. b) Circuito livre *hazard* lógico dinâmico.

2.3.2 *Hazard* Funcional

Quando uma função “ f ” não muda monotonicamente durante uma transição de entrada em modo MIC, então “ f ” tem *hazard* funcional nesta transição. O *hazard* funcional está associado à especificação do problema e não pode ser evitado apenas modificando a cobertura lógica. É possível corrigi-lo garantindo que determinado caminho seja percorrido, adicionando-se elementos de atraso ao mesmo. Esta solução não é desejada, pois contribui negativamente no desempenho do circuito aumentando área e potência. O exemplo da Figura 2.5(a) mostra a possibilidade de ocorrência do *hazard* funcional estático na transição $ABCD = 0010$ para $ABCD = 0111$. Nesta situação, é possível que a lógica associada à entrada B tenha um tempo de propagação maior que o da lógica de D. Dessa forma, a saída partiria de nível lógico 1 para 0, quando $ABCD = 0011$ e num instante seguinte, quando $ABCD = 0111$, a

saída retornaria para nível lógico 1. A Figura 2.5(b) mostra um exemplo de *hazard* funcional dinâmico em que a saída esperada é uma transição de nível lógico 0 para 1, porém dependendo da trajetória das transições de entrada é possível que ocorra uma transição de saída do tipo: $0 \rightarrow 1 \rightarrow 0 \rightarrow 1$.

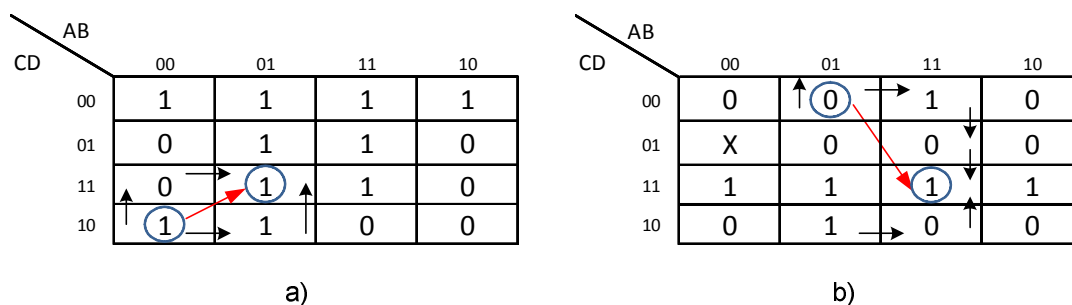


FIGURA 2.5 – a) *Hazard* funcional estático. b) *Hazard* funcional dinâmico.

2.3.3 *Hazard* Sequencial

Hazard sequencial ocorre devido a laços de realimentação em uma máquina de estados assíncrona. Há dois tipos de *hazard* sequencial: *hazard* essencial e não essencial.

Hazard essencial ocorre caso haja uma corrida entre uma variável de estado e uma entrada nas máquinas de estados finita assíncrona que obedecem ao modelo de atraso *bounded gate and wire delay*. Existem dois tipos de *hazard* essencial, conforme ilustrado na Figura 2.6. O primeiro é caracterizado pela mudança no estado final, ou seja, o circuito para em um estado diferente do previsto, este é conhecido como *hazard* essencial de estado estável. O segundo apresenta um *glitch* na saída, mas não modifica o estado final e é chamado de *hazard* essencial transitente. A Figura 2.6(a) mostra um mapa de Karnaugh com a entrada “A”, as variáveis de estado “y1” e “y2” e a saída “Z”. Os valores no interior do mapa são apresentados no formato y1y2/Z e os círculos representam os estados estáveis. Neste caso há *hazard* essencial de estado estável na transição do estado “a” para “b” caso “y2” mude antes do completo processamento da entrada “A”, levando erroneamente ao estado “c”. A Figura 2.6(b) mostra um mapa de Karnaugh com as entradas “A” e “B”, uma variável de estado “y” e uma saída “Z”. Os valores no interior do mapa são apresentados no formato y/Z e os círculos representam os estados estáveis. Há ocorrência de *hazard* essencial transitente na transição de “a” para “b” caso a variável de estado “y” mude antes do completo processamento da entrada “B”.

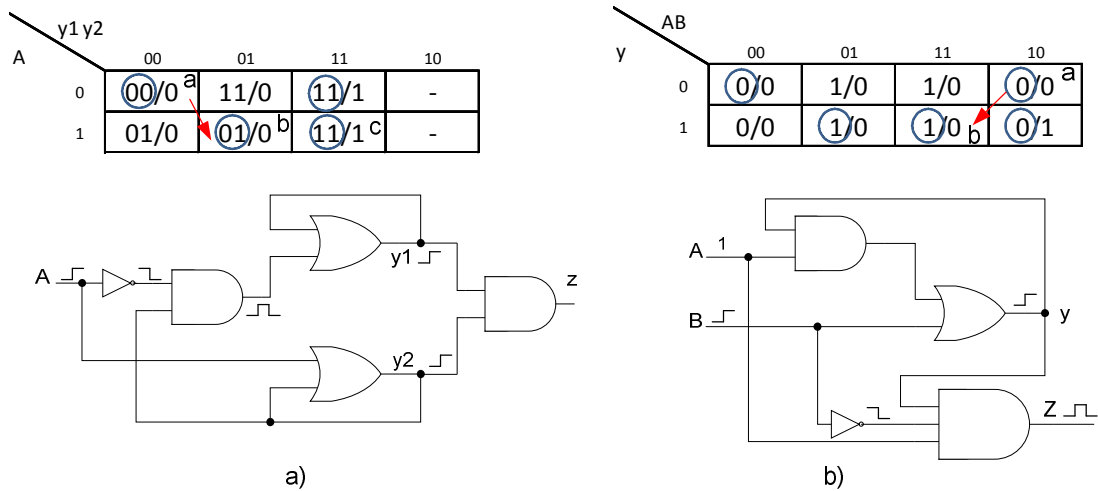


FIGURA 2.6 – a) Hazard essencial de estado estável. b) Hazard essencial transiente.

Hazard não essencial ou corrida crítica ocorre em uma máquina de estados assíncrona quando uma transição de entrada causa uma corrida entre duas ou mais variáveis de estado e o estado final depende de qual variável ganha a corrida. Corrida crítica pode ser evitada codificando os estados de maneira criteriosa. Veja a Figura 2.7(a) onde há intenção de atingir o estado “b” a partir do estado “a”. Quando há uma transição de entrada $xw = 00$ para $xw = 10$, as variáveis de estado $y1y2 = 00$ devem mudar para $y1y2 = 11$, entretanto se $y1$ mudar primeiro será atingido o estado $y1y2 = 10$ que é o estado estável “c”. Uma corrida não crítica ocorre caso o estado final não seja afetado, conforme ilustrado na Figura 2.7(b) onde qualquer ordem de ativação das variáveis de estado “y1” e “y2” levam ao estado estável “b”.

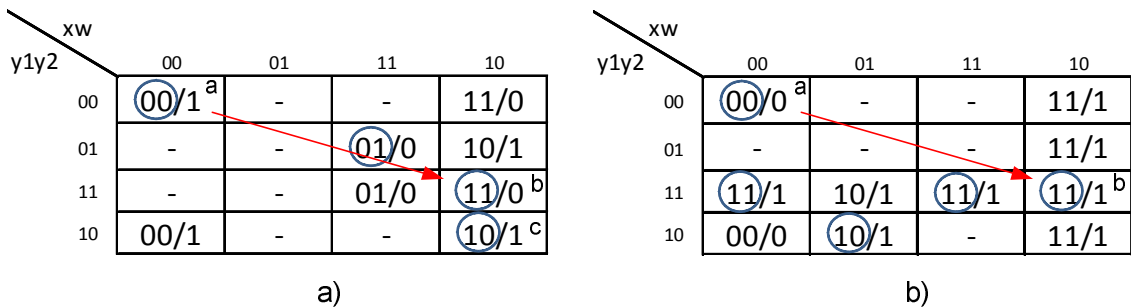


FIGURA 2.7 – a) Corrida crítica. b) Corrida não crítica.

2.4 Protocolo de Comunicação *Bundled-Data*

Bundled-data consiste em uma linha de pedido (*request*) e uma linha de aceite (*acknowledge*) agrupada com um barramento de dados unidirecional, conforme Figura 2.8(a). O sinal de pedido indica quando há dados válidos a serem transmitidos e o sinal de aceite indica quando os dados já foram recebidos.

A comunicação entre os estágios pode ser realizada com um protocolo de quatro fases (4-fases) ou com um protocolo de duas fases (2-fases). As Figuras 2.8(b) e 2.8(c) mostram, respectivamente, os comportamentos dos protocolos de 4 e 2-fases. O protocolo de quatro fases é o mais familiar aos projetistas, mas possui uma desvantagem por executar uma transição de retorno ao zero que consome tempo e energia desnecessários (SPARSO; FURBER, 2001). O protocolo de duas fases tem o potencial de ser mais eficiente, pois não necessita do retorno ao zero, ou seja, qualquer transição de nível lógico 0 para 1 e vice-versa é interpretada como um evento válido, entretanto sua implementação pode ser mais complexa.

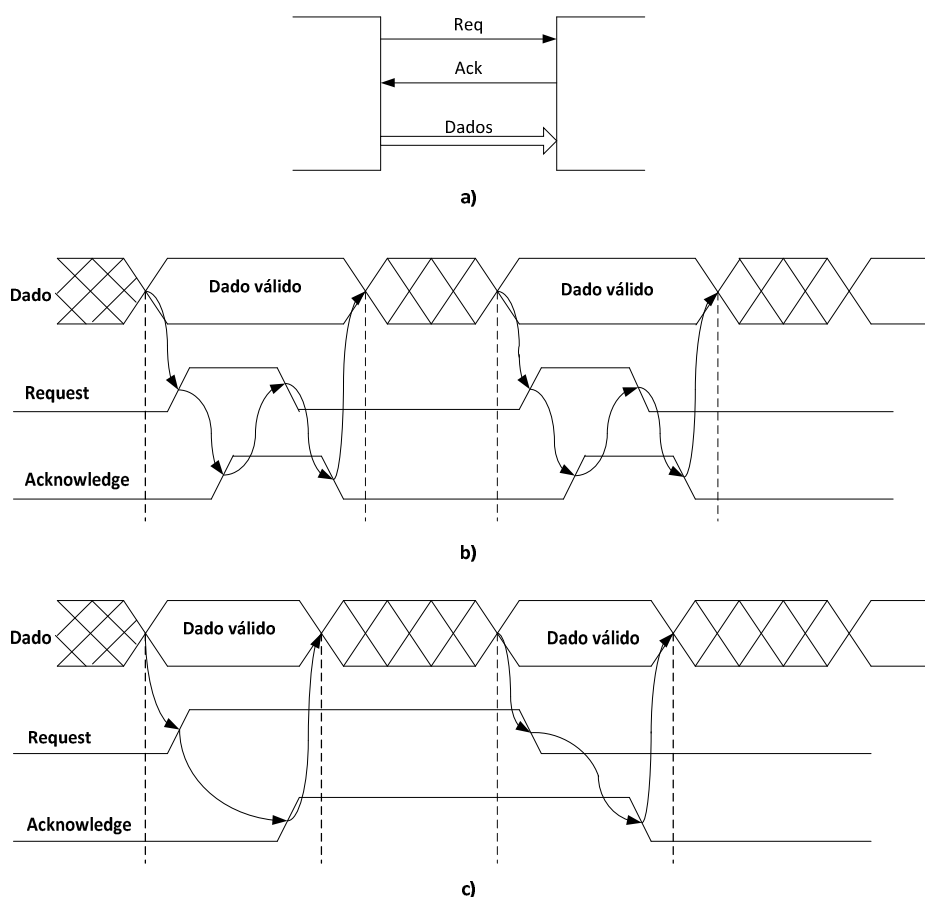


FIGURA 2.8 – a) Canal de comunicação. b) Protocolo de 4-fases. c) Protocolo de 2-fases.

2.5 Síntese Comportamental

Síntese comportamental é o processo de geração de um *hardware* digital a partir de uma descrição algorítmica de um problema. O processo de síntese traduz os objetos de alto nível, como por exemplo, operadores e estrutura de dados em objetos de *hardware*, tais como somadores, registradores e controladores de acordo com a arquitetura alvo do projeto.

A síntese comportamental de circuitos assíncronos pode ser classificada de acordo com sua categoria. As principais categorias são: Tradução Direta, Dessincronização, *Pipeline* Assíncrono e Decomposição.

2.5.1 Tradução Direta

A síntese comportamental por tradução direta é o processo de síntese que gera um circuito assíncrono a partir de uma linguagem de alto nível pela tradução de cada elemento da linguagem em um componente de *hardware*. Esta característica pode ser vista como um benefício, pois permite que o projetista saiba exatamente como seu código será transformado em um circuito. Entretanto, para a obtenção de circuitos com desempenho satisfatório, é necessária experiência por parte do projetista em identificar na especificação construções ineficientes, o que restringe e dificulta a utilização do método (NIELSEN, 2004). Outro fator importante é que as propostas existentes para tradução direta utilizam linguagens específicas, tais como Balsa (BARDSLEY, 1998), Tangram (BERKEL, 1993) e OCCAM (BRUNVAND, 1991).

A principal vantagem da síntese por tradução direta é facilitar o projeto de circuitos assíncronos complexos.

A principal ferramenta livre disponível para síntese por tradução direta é o Balsa, que traduz a especificação em componentes que utilizam canais de comunicação *handshake*. Possui um conjunto de aproximadamente 40 componentes, e os canais de comunicação podem estar associados a um *datapath*, quando há processamento de dados ou podem atuar puramente como controle, para sincronização de dados. Utiliza uma linguagem própria baseada em Tangram para a descrição do projeto, que tem o mesmo nome da ferramenta, Balsa. Diversas propostas de otimização estão surgindo (CHELCEA et al, 2002, 2003; KOMATSU; HARIYAMA; KAMEYAMA, 2012) em busca de melhor otimização dos circuitos gerados pelo Balsa. As otimizações são basicamente feitas a partir da ressíntese dos

componentes da biblioteca, utilizando métodos capazes de melhorar o desempenho, como por exemplo, a especificação *Burst-Mode* (NOWICK, 1995).

2.5.2 Dessincronização

A síntese comportamental que utiliza o modelo de dessincronização parte de um projeto convencional síncrono e substitui o sinal de *clock* por um conjunto de circuitos de controle operando por *handshake*. O primeiro requisito para aplicar a dessincronização é substituir os registradores por *latches* mestre-escravo, *latch* M e S conforme mostrado na Figura 2.9. Em seguida, adiciona-se um elemento de controle assíncrono para cada *latch*, interligando-os através dos sinais de pedido e aceite (CORTADELLA; KONDRATYEV; LAVAGNO; SOTIRIOU, 2006). Os controladores estão apresentados na Figura 2.9 pela letra C e são baseados no conceito de *micropipeline*, necessitando de elementos de atraso entre eles, que são calculados a partir da latência do caminho crítico da lógica de processamento.

A principal desvantagem deste método é o aumento da área e potência devido à adição dos circuitos de controle e elementos de atraso.

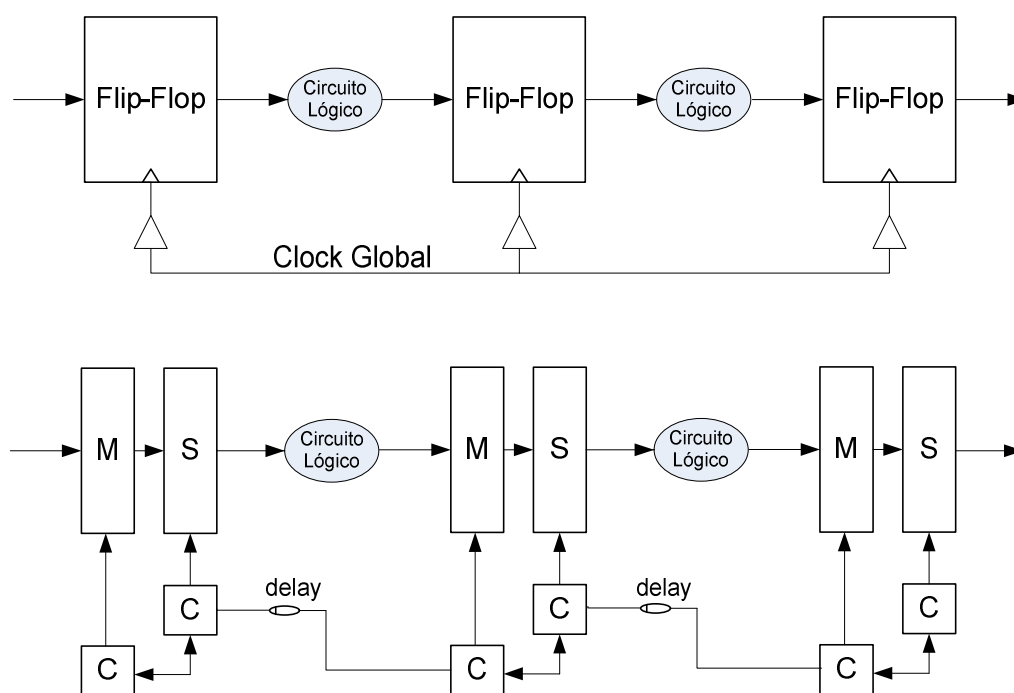


FIGURA 2.9 – Dessincronização.

2.5.3 Pipeline Assíncrono

Os conceitos básicos que envolvem o projeto de *pipelines* assíncronos foram criados por Sutherland (1989) e nomeado de *Micropipeline*.

No projeto de um *micropipeline* o controle é distribuído e responsável por realizar a comunicação entre os estágios do *pipeline*. Esta comunicação emprega o protocolo do tipo *Handshake* entre o sinal pedido (*Request*) e o sinal aceite (*Acknowledge*) (ZAFAR, 2005). Na proposta inicial (SUTHERLAND, 1989), o controle é baseado em elementos-C de Muller, que atuam como porta lógica E para eventos. Quando ambas as entradas de um elemento-C estão no mesmo nível lógico, sua saída é atualizada com o valor das entradas. Quando as entradas estão em níveis lógicos diferentes, a saída permanece com o seu valor anterior. Um exemplo de *micropipeline* é ilustrado na Figura 2.10.

Atualmente, existem diversas variantes do projeto inicial, onde as principais modificações são: a comunicação entre os estágios pode ser realizada no protocolo de quatro fases (4-fases) ou no protocolo de duas fases (2-fases). Os registradores podem ser ativados por nível (*latch*) ou por borda (flip-flop) e a lógica do *datapath* pode ser estática ou dinâmica.

Assim como em *pipelines* síncronos, é possível classificar *micropipelines* em dois tipos: lineares e não lineares. O tipo linear é aquele no qual cada estágio possui apenas um canal de entrada e um canal de saída. O tipo não linear, utilizado em sistemas digitais mais complexos, necessita de diversos componentes extras para permitir sua implementação, tais como: união e bifurcação, árbitros e estruturas de controle para laços de repetição.

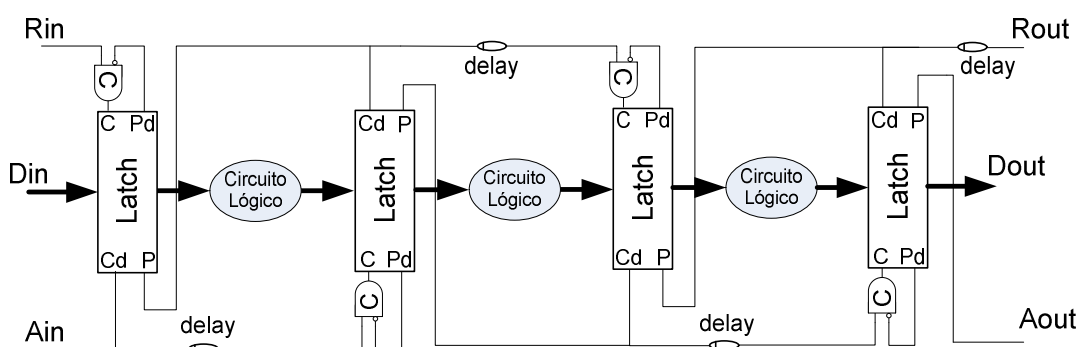


FIGURA 2.10 – *Micropipeline* (SUTHERLAND, 1989).

2.5.4 Decomposição

A arquitetura de projeto conhecida por decomposição, ou também citada com o nome *shared-resource* (HANSEN, 2012), separa o projeto em um controlador centralizado e *datapath*, semelhante ao procedimento clássico para síntese de circuitos síncronos (ver Figura 2.11). Tanto controlador, quanto *datapath* podem ser sintetizados utilizando diferentes abordagens e em diferentes classes de circuitos assíncronos, onde a classe define em que modelo de atraso o circuito irá operar corretamente e em que modo de operação o circuito se comunicará com o ambiente.

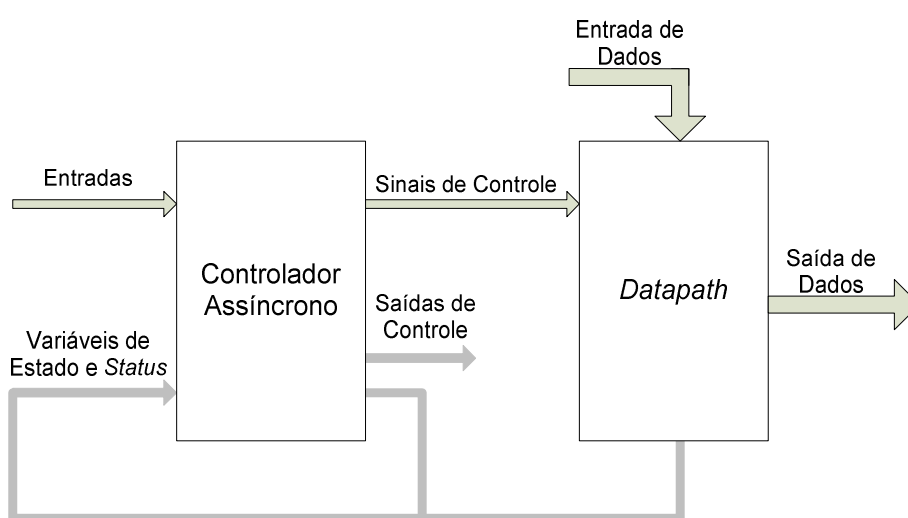


FIGURA 2.11 – Decomposição em controlador e *datapath*.

Esta arquitetura tem duas diferentes variantes. Na primeira, o projeto envolve *datapath* assíncrono (SINGH; NOWICK, 2007), por isso não apresenta elementos de atraso para detectar o término das operações, entretanto apresenta um custo elevado, pois utiliza componentes *dual-rail*. A codificação *dual-rail* consiste na utilização de um par de fios com duas combinações válidas (01,10), que codifica os valores binários 0 e 1 respectivamente. Este protocolo de codificação permite transições monotônicas partindo de (00), que não é uma palavra codificada, para uma das duas combinações válidas (01 ou 10) e em seguida retornando para (00). O estado (00) é chamado de “espaçador” do termo em inglês *spacer* e indica a ausência de dados. A Figura 2.12(a) mostra a codificação *dual-rail* e a Figura 2.12(b) uma porta lógica E com sua tabela verdade construída na codificação *dual-rail*. Na segunda variante da arquitetura decomposição, o *datapath* é implementado no protocolo *bundled-data*,

que envolve somente componentes tradicionais do paradigma síncrono, tais como unidades funcionais, registradores e multiplexadores. A detecção do término das operações é obtida utilizando elementos de atraso, que permite a comunicação correta entre o controlador e o *datapath* (requisito do modo fundamental generalizado). A segunda variante tem potencial para obtenção de circuitos com menor consumo de energia, menor área e simplificação do projeto quando comparado com a primeira variante.

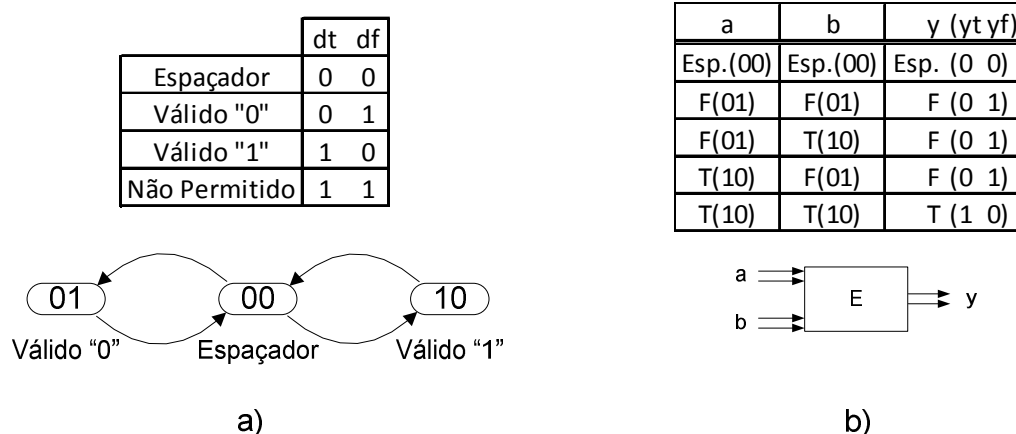


FIGURA 2.12 – a) Codificação *dual-rail*. b) Porta lógica E *dual-rail*.

2.6 Controladores Assíncronos

Três tipos de especificações são usadas de forma bastante popular para descrever controladores assíncronos, a saber:

1. A especificação grafo de transição de sinais (*signal transition graph* – STG) proposta por Chu (1987) é a mais indicada para controladores que interagem com ambientes concorrentes e é baseada em redes de Petri, ou *Petri nets* (PNs). Uma *Petri net* é um grafo bipartido de arcos dirigidos com dois tipos de vértices: transições (*transition*) e lugares (*place*). É um método para descrever sistemas concorrentes mostrando de forma explícita as concorrências e os conflitos entre os eventos. A especificação STG utiliza uma subclasse do método *Petri net* (SPARSO; FURBER, 2001). A principal desvantagem do STG é que o mesmo tende a tornar-se confuso, principalmente para descrições altamente concorrentes. A Figura 2.13(a) e (b) mostram respectivamente o elemento-C e seu diagrama de tempo. Na sequência, a Figura 2.13(c) mostra a especificação *Petri net* correspondente, onde os lugares de entrada estão marcados com *tokens* (círculo preenchido) para as transições “a+” e “b+”,

significando o estado inicial $(a,b,c) = (0,0,0)$. As transições “a+” e “b+” podem disparar em qualquer ordem e a qualquer tempo e assim que ambas estiverem ativas a transição “c+” é ativada. Na sequência ocorrem as mesmas transições, porém com os sinais invertidos. A Figura 2.13(d) apresenta a especificação STG equivalente à *Petri net*, porém de uma forma simplificada, onde os lugares estão omitidos e os arcos que interligam duas transições são considerados como lugares.

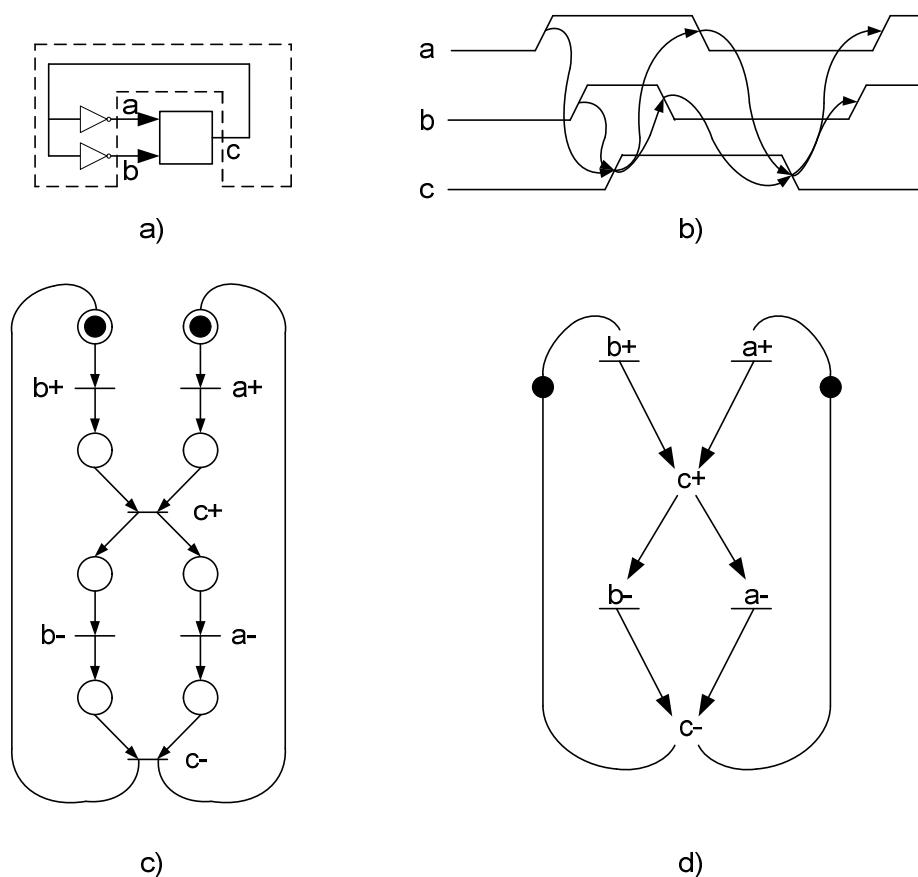


FIGURA 2.13 – a) Elemento-C e comportamento simulado. b) Diagrama de tempos. c) *Petri net*. d) STG.

2. A especificação modo rajada (*burst-mode* – BM) foi proposta por Davis e Stevens (1993) e formalizada por Nowick (1995). Nesta especificação, as transições ocorrem quando uma ou mais entradas mudam de nível lógico, $0 \rightarrow 1$ ou $1 \rightarrow 0$ e são chamadas de sinais sensíveis à transição (TSS). Quando não há transições nas entradas, o circuito permanece estável em um mesmo estado. O modo rajada é uma especificação baseada em diagramas de transição de estados, onde transições são representadas por arcos que são

nomeados com as correspondentes entradas/saídas. Rajadas devem ter comportamento monotônico, ou seja, é permitida apenas uma mudança de nível em cada sinal a cada transição. A transição entre os estados é ativada por uma rajada de entrada, onde os sinais que compõe a rajada de entrada podem mudar em qualquer ordem e a qualquer tempo, causando uma rajada de saída. Uma nova rajada de entrada só pode ocorrer após a estabilização do circuito e esta não pode ser vazia. A especificação BM deve satisfazer três propriedades: a) Polaridade do sinal, que indica a transição de um sinal, (+ e -) para as bordas de subida e descida respectivamente; b) Condição de entrada única, ou seja, cada estado é descrito por um único vetor entrada/saída. Por exemplo, na Figura 2.14 os sinais do estado 1 são atingidos com os valores $abcxy = 10010$. Este valor é alcançado através de qualquer caminho, seja do estado 0, seja do estado 4; c) Propriedade do conjunto máximo, que proíbe uma transição ser um subconjunto de outra saindo do mesmo estado. Por exemplo, se a rajada de entrada da transição $1 \rightarrow 3$ fosse $b+$ ao invés de $c+b+$, então haveria a violação desta propriedade, pois caso a entrada $b+$ ocorra primeiro, a máquina poderia ir para o estado 2 ou para o 3.

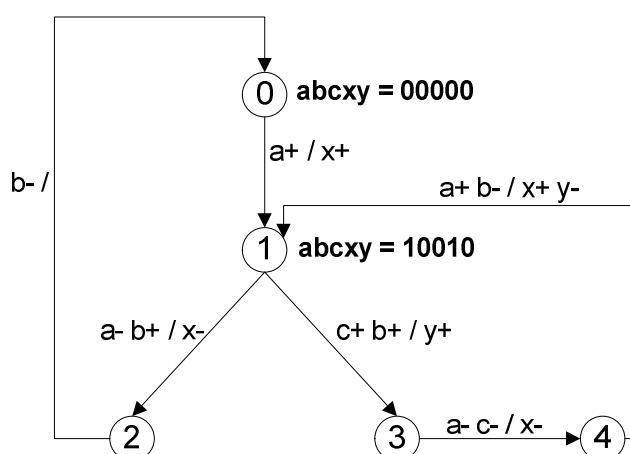


FIGURA 2.14 – Especificação modo rajada (BM).

3. O modo rajada estendido (*extended-burst-mode* – XBM) foi proposto por Yun e Dill (1999), este acrescentou ao BM os sinais de condição e irrelevantes (*directed don't-care*), permitindo a concorrência entre sinais de entrada e saída. Os sinais que aparecem entre \langle e \rangle são condicionais e os sinais terminados em + ou - e que não estão entre \langle e \rangle , são chamados sinais de terminação. Os sinais de condição são sensíveis a nível e podem mudar livremente caso não seja usado na transição em questão, mas devem satisfazer os tempos de

setup e *hold* quando usados pela transição. O sinal de condição é amostrado após todos os sinais de terminação associados à transição terem ocorrido. Os sinais sensíveis à transição podem ser descritos como *directed don't care*, o que significa que nas transições onde foi especificado é irrelevante, devendo atingir uma transição rotulada como + ou -, tornando-se um sinal de terminação. O sinal de terminação, que em uma transição anterior não foi especificado como irrelevante, é chamado de sinal compulsório. Para cada rajada de entrada deve haver pelo menos um sinal compulsório. A Figura 2.15 ilustra a especificação XBM de um controlador de barramento SCSI_INIT-SEND definido pela norma ANSI (ANSI X3.131,1986). Este exemplo possui 4 entradas (Cntgtl, Fain, Ok, Rin) e duas saídas (Aout, Frou) sendo que o estado inicial é o 0. A descrição Rin+ Fain- / Aout+ da transição 5 → 3 significa que a saída Aout: 0 para 1, será ativada quando a entrada rajada (Rin: 0 para 1 e Fain: 1 para 0) for ativada. O sinal Cntgtl é um sinal de nível e descreve a condição para a transição 3 → 6 e 3 → 4. O sinal Rin* da transição 4 → 5 é um sinal *directed don't care* e significa que esta entrada pode modificar seu valor nesta transição ou permanecer no valor anterior.

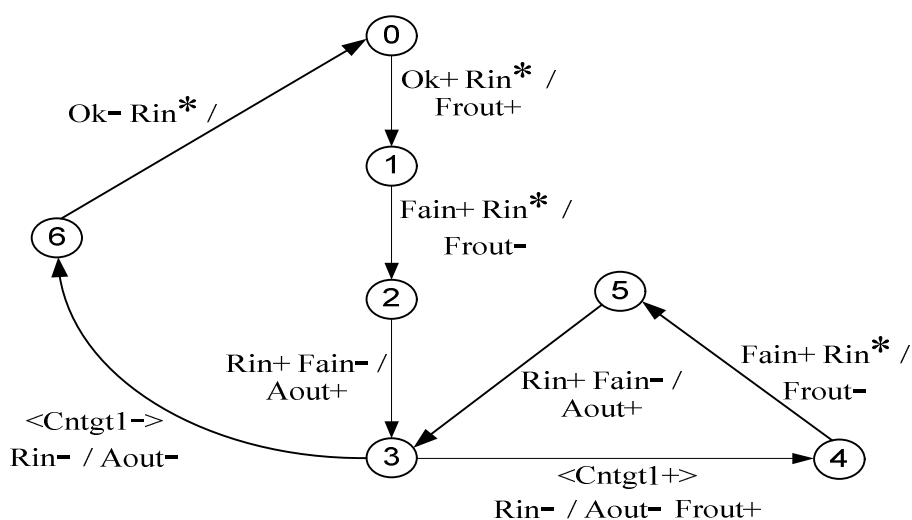


FIGURA 2.15 – Especificação modo rajada estendido (XBM).

3 Métodos para Síntese Comportamental por Decomposição

Os métodos de síntese comportamental na arquitetura de projeto decomposição seguem a mesma abordagem aplicada ao modelo síncrono.

Tipicamente, a síntese comportamental inicia com a especificação da aplicação a ser implementada, através de uma linguagem de alto nível. As ferramentas de síntese comportamental transformam automaticamente a especificação em uma arquitetura de hardware descrita em nível RTL (*Register Transfer Level*), implementando de forma eficiente a aplicação desejada.

Conforme apresentado na Figura 3.1, a partir da especificação em alto nível e com informações de uma biblioteca de componentes RTL, os principais passos da síntese comportamental são:

a) Compilação da especificação. Este primeiro passo pode incluir otimizações ao código, tais como eliminação de código desnecessário, balanceamento das árvores de expressões, etc. O resultado da compilação gera um GFDC (grafo de fluxo de dados e controle) que é capaz de representar as dependências dos dados e o fluxo de controle;

b) Alocação de recursos, UFs (Unidades Funcionais). Define os tipos e o número de recursos de *hardware* necessários para implementar a especificação;

c) Escalonamento. Define o intervalo de tempo onde cada UF será executada;

d) Assinalamento de UFs e Registradores. Define em qual componente cada operação será executada e onde serão armazenados os resultados das operações;

e) geração da arquitetura RTL. A geração da arquitetura inclui um controlador e um *datapath*. O *datapath* é constituído tipicamente de um conjunto de registradores, unidades funcionais e multiplexadores. O controlador é constituído de uma máquina de estados assíncrona que aciona os sinais de controle do *datapath*.

Os processos de alocação de recursos, escalonamento e assinalamentos de unidades funcionais e registradores são processos interdependentes e podem ser executados em conjunto. Entretanto, devido à complexidade computacional, frequentemente estas operações são executadas em sequência.

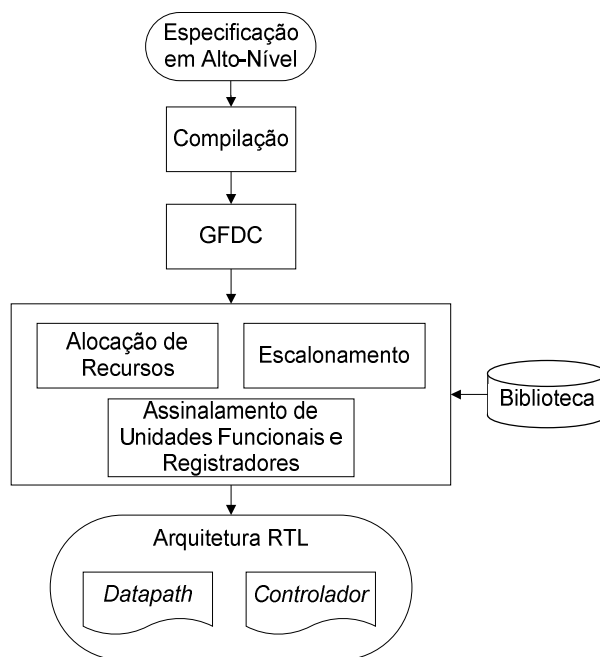


FIGURA 3.1 – Etapas da síntese comportamental.

Neste Capítulo descrevem-se as principais etapas do processo de síntese comportamental baseado na arquitetura de projeto decomposição. Estas etapas são importantes para o desenvolvimento da dissertação, pois é utilizada a mesma arquitetura. No processo de escalonamento, descrito na Seção 3.3, apresenta-se uma análise mais detalhada das principais propostas de escalonamento existentes, sendo que os algoritmos de escalonamento baseados em programação linear inteira, *force-directed* e *branch-and-bound* foram implementados manualmente para um exemplo típico da literatura. Esta análise contribuiu para a escolha do algoritmo que melhor atendeu às necessidades desta dissertação, levando em consideração a complexidade de implementação do algoritmo e a qualidade da solução obtida.

3.1 GFDC

Um dos primeiros passos da síntese comportamental é converter a descrição do problema em um grafo que descreve o fluxo de dados e de controle (GFDC). O GFDC pode ser subdividido em grafo de fluxo de dados (GFD) e grafo de fluxo de controle (GFC), como mostrado na Figura 3.2(b) e (c) respectivamente. No GFD as operações da descrição comportamental são representadas por vértices, enquanto os arcos que interligam os vértices

representam as entradas, saídas e variáveis. Na representação do GFC, os arcos representam as dependências de controle e os vértices representam blocos de operações.

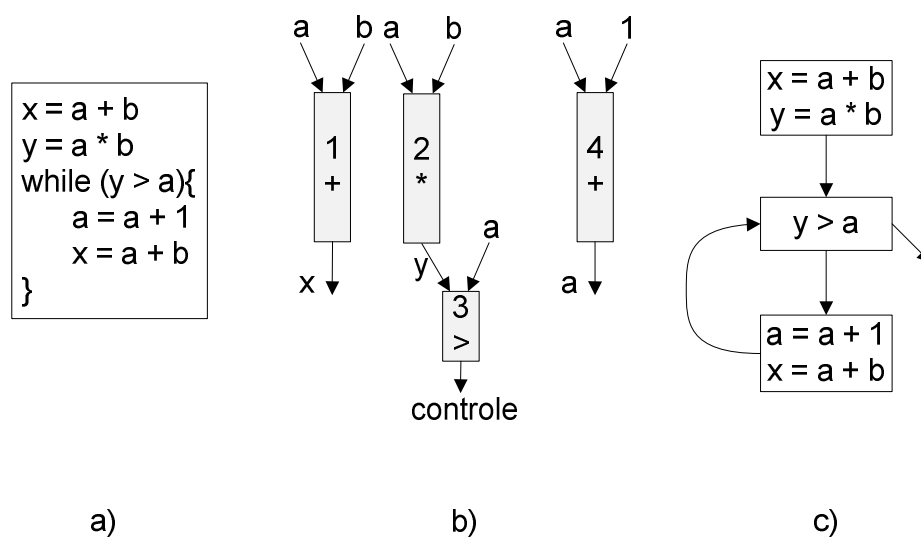


FIGURA 3.2 – a) Descrição comportamental. b) GFD. c) GFC.

3.2 Alocação de Recursos (*Resource Allocation*)

Alocação de recursos é o processo no qual se decide a quantidade e os tipos de recursos a serem utilizados em um determinado projeto. A escolha dos recursos tem impacto importante no resultado final do circuito, e a alocação dos mesmos está fortemente relacionada ao escalonamento, pois define o conjunto de componentes que pode ser utilizado para implementar determinada funcionalidade. Um maior paralelismo na execução das operações pode ser obtido alocando-se um grande número de recursos. Caso sejam alocados poucos recursos, será necessário reutilizá-los, aguardando pela disponibilidade dos mesmos (ELLIOTT, 1999).

3.3 Escalonamento (*Scheduling*)

Escalonamento é um processo da síntese comportamental que determina o intervalo de tempo em que cada uma das unidades funcionais no qual as operações foram mapeadas, estará em execução. O escalonamento é um problema de otimização combinatória e os algoritmos usados em sua solução podem ser agrupados em duas categorias: os que produzem um

escalonamento sem parâmetros de restrição, e os que produzem um escalonamento limitado por parâmetros de restrição.

Os algoritmos de escalonamento sem parâmetros de restrição são utilizados para encontrar rapidamente um escalonamento válido, devendo obedecer apenas às restrições imposta pelas dependências de dados. Os algoritmos mais populares nesta categoria são o ASAP (*As Soon As Possible*) e ALAP (*As Late As Possible*), (MICHELI, 1994, p.188).

Os algoritmos de escalonamento com parâmetros de restrição podem estar submetidos à restrições de tempo, de recursos ou ambas. Quando se utiliza restrição de recursos, o algoritmo busca encontrar a menor latência utilizando os recursos disponíveis. Para projeto sob restrição de tempo, o algoritmo busca encontrar o menor número de recursos necessário à implementação, sem exceder o tempo desejado.

Um escalonamento assíncrono ótimo não pode ser obtido a partir da formulação utilizada em escalonamento síncrono (HANSEN; SINGH, 2010), pois a formulação tradicional considera a utilização dos passos do sinal de *clock*. No entanto, as primeiras abordagens para o escalonamento assíncrono surgiram de adaptações feitas diretamente do modelo síncrono (NOWICK; SINGH, 2015).

As abordagens voltadas para o modelo assíncrono devem, idealmente, abandonar a formulação baseada na discretização do tempo e o foco do problema passa a ser a determinação da ordem de execução das operações. A primeira proposta direcionada ao escalonamento assíncrono foi apresentada por Badia e Cortadella (1993), que busca decidir qual operação será escalonada de acordo com sua prioridade, utilizando um algoritmo heurístico baseado em *list scheduling* (DAVIDSON et al, 1981). A prioridade de cada operação é calculada avaliando as operações sucessoras, e a operação que tiver o maior número de dependências terá maior prioridade. Apesar de gerar boas soluções em pouco tempo, este método não garante uma solução ótima (HANSEN; SINGH, 2010).

Outra abordagem para o problema foi apresentada por (BACHMAN, 1998). Esta consiste em inserir operações em janelas de tempo, onde especifica-se o tempo máximo e mínimo para cada operação. Antes do escalonamento, é necessário executar o mapeamento dos recursos para determinar os parâmetros de latência das unidades funcionais. Para determinar as janelas de tempo, dois algoritmos são utilizados: ASAP usado para encontrar o menor tempo de execução e ALAP para determinar a mobilidade das operações. Cada um dos algoritmos é executado três vezes, considerando um tempo de latência mínimo, médio e máximo para as operações. A exploração das possibilidades de escalonamento é feita usando-se o algoritmo *branch-and-bound* (DOIG; LAND, 1960), e as janelas de tempo são utilizadas

para reduzir o espaço de busca. Para cada operação escalonada adiciona-se um novo arco entre as operações do grafo, que deve ser analisado novamente para atualizar as janelas de tempo determinadas no passo anterior. A principal desvantagem desta abordagem, segundo o próprio autor, é a complexidade computacional, que restringe seu uso a pequenos exemplos.

Em seguida, Bachman, Zheng e Myers (1999) propuseram uma metodologia de escalonamento baseada na adição de uma nova dependência entre as operações, acrescentado novos arcos. A proposta parte de um GFD que corresponde a um escalonamento sem restrições de recursos. Em seguida, analisam-se quais operações podem compartilhar o mesmo recurso sem que haja conflitos. Consideram-se como conflito as operações que estão em execução simultaneamente. Neste caso, um arco é adicionado entre as operações conflitantes, garantindo-se, desta forma, que as operações não sejam executadas ao mesmo tempo. A cada novo arco inserido uma nova análise é iniciada, até que todos os nós do grafo tenham sido escalonados.

Posteriormente, Saito *et al.* (2006, 2007) propuseram dois algoritmos de escalonamento voltados para o paradigma assíncrono que são baseados em programação linear inteira e *force-directed* (ver detalhes nas Seções 3.3.1 e 3.3.2 respectivamente). Os algoritmos destas propostas apresentam características diferentes, sendo que o algoritmo baseado em PLI é capaz de encontrar a solução exata, porém tem como limitação o crescimento exponencial do tempo para solucionar o escalonamento, enquanto que o algoritmo *force-directed* encontra uma solução eficiente do ponto de vista computacional, mas não garante que a solução encontrada seja ótima. Para solucionar esta limitação, Hansen e Singh (2010), propuseram uma nova abordagem para o escalonamento assíncrono que é baseado em um algoritmo *branch-and-bound* (ver detalhes na seção 3.3.3). Recentemente, Andrikos e Lavagno (2011), propuseram um algoritmo de escalonamento assíncrono que é baseado em *branch-and-bound* que utiliza em sua formulação uma *Petri net* para representar o fluxo de dados e de controle, permitindo aplicar propriedades comutativas e associativas antes de efetuar o escalonamento. No entanto, este método apresenta maior dificuldade na implementação do algoritmo, além de apresentar, na maioria dos casos analisados, desempenho bastante semelhante ao método proposto por Hansen e Singh (2010).

3.3.1 Escalonamento Assíncrono: Programação Linear Inteira (PLI)

O algoritmo PLI foi bastante utilizado na tarefa de escalonamento de operações em circuitos síncronos, entretanto, quando utilizado diretamente em circuitos assíncronos, pode

não obter a solução ótima, porque o início de uma nova operação deve aguardar um novo ciclo de *clock*. Este problema pode ser contornado reduzindo o intervalo de tempo entre os passos, de modo que o início das operações possa coincidir com algum passo. Isso, porém, aumenta muito a complexidade da solução, levando a um aumento no tempo de processamento (SAITO *et al.*, 2007).

A solução proposta por Saito *et al.* (2006) reduz o espaço de busca através da atribuição de possíveis tempos para o início das operações e em seguida selecionando um subconjunto da atribuição inicial. A atribuição dos tempos de início de cada operação é feita analisando o tempo de latência das operações antecessoras, que pode ser um antecessor direto (aquele que tem dependência de dados) ou uma operação que não tem dependência de dados, mas que pode compartilhar o mesmo recurso. A Figura 3.3 mostra o exemplo dos tempos selecionados a partir da heurística descrita em Saito *et al.* (2006) para o *benchmark* solucionador de equação diferencial de segunda ordem pelo método de Euler: $y'' + 3xy' + 3y = 0$. O resultado apresenta um conjunto de possibilidades para o início das operações, semelhante ao escalonamento síncrono, porém sem um intervalo de tempo fixo.

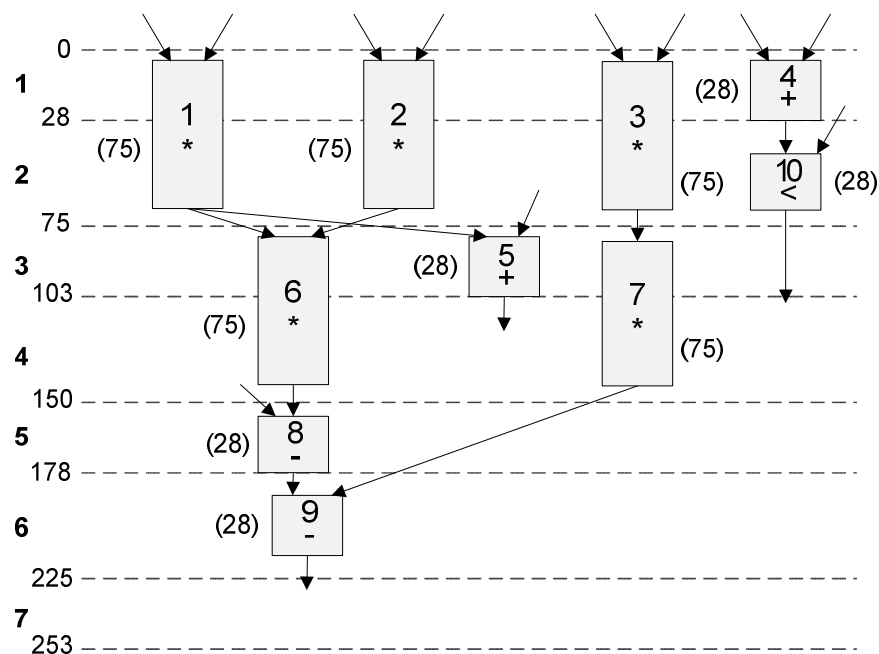


FIGURA 3.3 – Tempos de início possíveis para as operações do GFD.

A partir do conjunto de possíveis tempos de início das operações, aplica-se o algoritmo PLI para a resolução do sistema, de acordo com as seguintes equações:

- a) Função Objetivo para o escalonamento com restrição de tempo:

$$\text{Minimize: } \sum_{r \in R} \text{Custo}_r * \text{Num}_r \quad (3.1)$$

onde r representa um dos recursos disponíveis, R representa o conjunto de recursos, Custo_r representa o custo do recurso r e num_r o número do recurso r .

- b) Restrição de assinalamento do nó: Cada nó i deve ser escalonado em apenas um passo $l \in S_i$.

$$\sum_{l=F_{S_i}}^{L_{S_i}} x_{i,l} = 1, \forall_i \in N \quad (3.2)$$

- c) Restrição de dependência: Para cada dependência (i, j) do GFD a seguinte restrição deve ser satisfeita.

$$\sum_{l=F_{S_j}}^{L_{S_j}} t_l * x_{j,l} - \sum_{l=F_{S_i}}^{L_{S_i}} t_l * x_{i,l} - d_i \geq 0, \forall_{(i,j)} \in E \quad (3.3)$$

- d) Restrição de assinalamento de recurso: O número de operações executadas pelo recurso r durante o passo l deve ser menor ou igual ao número de recursos disponíveis num_r .

$$\sum_{i \in N_r} x_{i,l} - \text{num}_r \leq 0, \forall l \in L, \forall r \in R \quad (3.4)$$

Utilizando as equações descritas acima para o exemplo da Figura 3.3, obtém-se o seguinte sistema de equações:

Função Objetivo:

$$\text{Minimize: } 5\text{num}_{\text{mult}} + \text{num}_{\text{alu}}$$

Restrições de atribuição do nó:

$$X_{1,1} + X_{1,3} = 1$$

$$X_{2,1} + X_{2,3} = 1$$

$$X_{3,1} + X_{3,3} = 1$$

$$X_{4,1} + X_{4,4} + X_{4,6} = 1$$

$$X_{5,3} + X_{5,5} + X_{5,6} = 1$$

$$X_{6,3} + X_{6,5} = 1$$

$$X_{7,3} + X_{7,5} = 1$$

$$X_{8,5} + X_{8,6} = 1$$

$$X_{9,6} + X_{9,7} = 1$$

$$X_{10,2} + X_{10,4} + X_{10,6} + X_{10,7} = 1$$

Restrições de Dependência:

$$75X_{5,3} + 150X_{5,5} + 178X_{5,6} - 0X_{1,1} - 75X_{1,3} \geq 75$$

$$75X_{6,3} + 150X_{6,5} - 0X_{1,1} - 75X_{1,3} \geq 75$$

$$75X_{6,3} + 150X_{6,5} - 0X_{2,1} - 75X_{2,3} \geq 75$$

$$75X_{7,3} + 150X_{7,5} - 0X_{3,1} - 75X_{3,3} \geq 75$$

$$28X_{10,2} + 103X_{10,4} + 178X_{10,6} + 225X_{10,7} - 0X_{4,1} - 103X_{4,4} - 178X_{4,6} \geq 28$$

$$150X_{8,5} + 178X_{8,8} - 75X_{6,3} - 150X_{6,5} \geq 75$$

$$178X_{9,6} + 225X_{9,7} - 150X_{8,5} - 178X_{8,6} \geq 28$$

$$178X_{9,6} + 225X_{9,7} - 75X_{7,3} - 150X_{7,5} \geq 75$$

Restrições de Recurso

$$X_{1,1} + X_{2,1} + X_{3,1} - num_{mult} \leq 0$$

$$X_{1,3} + X_{2,3} + X_{3,3} + X_{6,3} + X_{7,3} - num_{mult} \leq 0$$

$$X_{6,5} + X_{7,5} - num_{mult} \leq 0$$

$$X_{4,4} + X_{10,4} - num_{alu} \leq 0$$

$$X_{4,6} + X_{5,6} + X_{9,6} + X_{10,6} - num_{alu} \leq 0$$

$$X_{9,7} + X_{10,7} - num_{alu} \leq 0$$

$$X_{5,5} + X_{8,5} - num_{alu} \leq 0$$

$$X_{4,1} - num_{alu} \leq 0$$

$$X_{5,3} - num_{alu} \leq 0$$

$$X_{5,5} + X_{8,5} - num_{alu} \leq 0$$

$$X_{10,2} - num_{alu} \leq 0$$

Solução:

$$num_{alu} = 1$$

$$num_{mult} = 2$$

$$X_{1,1}, X_{2,1}, X_{3,3}, X_{4,1}, X_{5,3}, X_{6,3}, X_{7,5}, X_{8,5}, X_{9,7}, X_{10,4}$$

A solução deste sistema de equações pode ser obtida utilizando *softwares* para cálculo numérico como, por exemplo, o Matlab ou de ferramentas como lp_solve (BERKELAAR; EIKLAND; NOTEBAERT, 2004). A solução indica que é necessário o uso de uma unidade lógica aritmética e dois multiplicadores. A sequência $X_{i,j}$ indica que a unidade funcional i , será escalonada no passo j , conforme ilustrado na Figura 3.4.

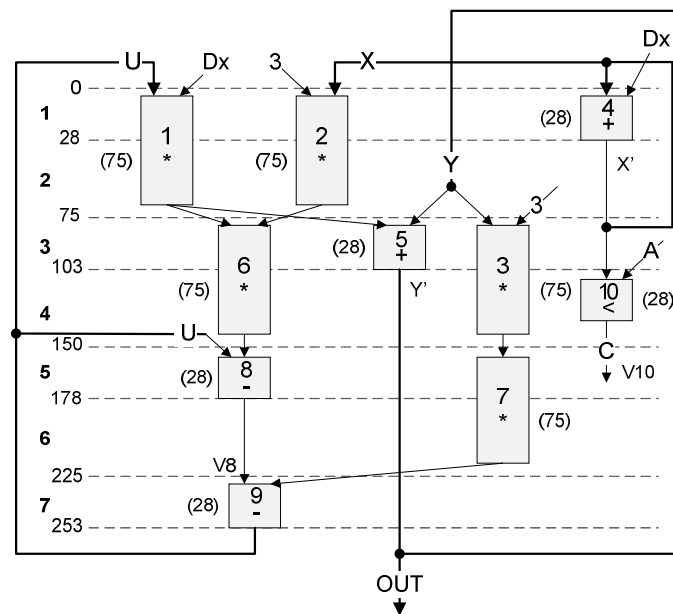


FIGURA 3.4 – Grafo de fluxo de dados escalonado por PLI.

A principal vantagem deste método é a possibilidade de obter-se um escalonamento ótimo. Entretanto, como a determinação dos tempos de início é obtida através de um algoritmo heurístico, que permite ser mais restritivo (limitando o número de tempos de início) ou menos restritivo (aumentando o número de tempos de início), a qualidade da solução está condicionada ao quão restritivo é o algoritmo. Portanto, quando o método é usado para circuitos grandes, há uma tendência de utilizar um longo tempo de processamento para obtenção do resultado.

3.3.2 Escalonamento Assíncrono: *Force-Directed*

O algoritmo *force-directed* foi idealizado por Paulin e Knight (1989) e consiste em um algoritmo heurístico de escalonamento que busca distribuir uniformemente as operações entre os recursos disponíveis. A proposta de Saito *et al.* (2007) faz a adaptação deste algoritmo para a utilização em circuitos assíncronos. Ela está apresentada a partir de um exemplo, onde a entrada é o grafo de fluxo de dados, veja Figura 3.5.

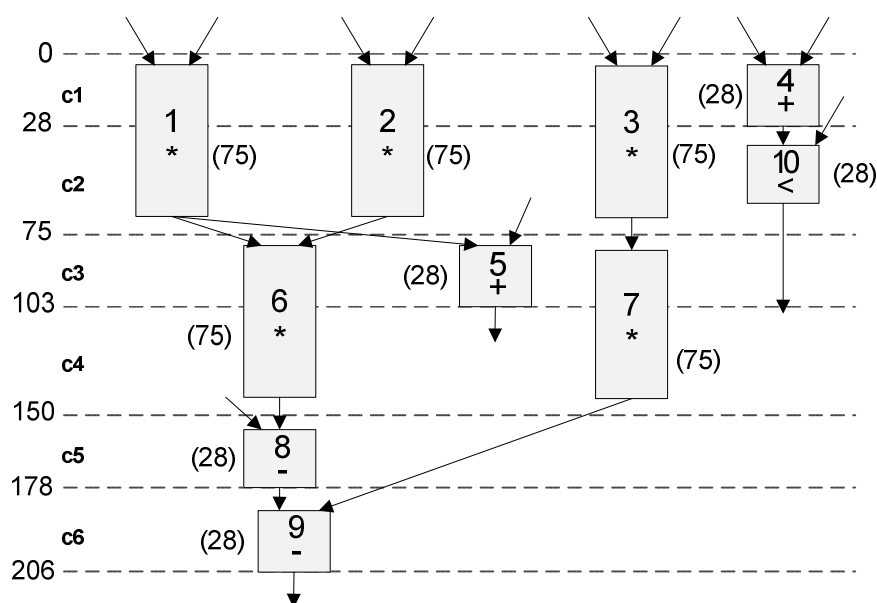


FIGURA 3.5 – Grafo de fluxo de dados e possíveis tempos de início das operações.

Os passos necessários para obtenção do escalonamento são:

a) Cálculo dos passos de controle: Assim como no método baseado em PLI, faz-se necessário o cálculo dos possíveis tempos de início das operações, conforme descrito em Saito *et al.* (2007) e apresentado na Figura 3.5.

b) Cálculo do intervalo de tempo: O intervalo de tempo F_{ni} representa sua mobilidade, ou seja o conjunto c_i de passos de controle da operação no qual o vértice i é válido, ver Figura 3.6.

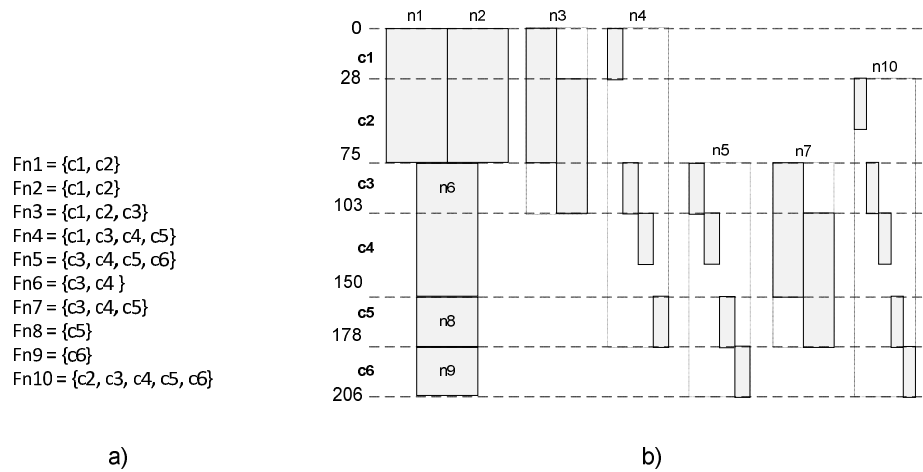


FIGURA 3.6 – Intervalo de tempo das operações: a) Representação textual. b) Representação gráfica.

c) Geração do grafo de distribuição: O grafo de distribuição representa a taxa de uso dos recursos em cada um dos passos de controle. É dado pela Equação (3.5), onde $|ST_{ni}|$ representa o número de tempos de início ativos no passo cs e $|St_{ni}|$ representa o número total de tempos de início para o recurso r :

$$DG_{(r,cs)} = \sum \left(\frac{|ST_{ni}|}{|St_{ni}|} \right) \quad (3.5)$$

$$DG_{(mul,c1)} = \frac{1}{1} + \frac{1}{1} + \frac{1}{2} = 2,5$$

$$DG_{(mul,c2)} = \frac{1}{1} + \frac{1}{1} + \frac{2}{2} = 3$$

$$DG_{(mul,c3)} = \frac{1}{2} + \frac{1}{1} + \frac{1}{2} = 2$$

$$DG_{(mul,c4)} = \frac{1}{1} + \frac{2}{2} = 2$$

$$DG_{(mul,c5)} = \frac{1}{2} = 0,5$$

$$DG_{(mul,c6)} = 0$$

$$DG_{(alu,c1)} = \frac{1}{4} = 0,25$$

$$DG_{(alu,c2)} = \frac{1}{5} = 0,2$$

$$\begin{aligned}
 DG_{(alu,c3)} &= \frac{1}{4} + \frac{1}{4} + \frac{1}{5} = 0,7 \\
 DG_{(alu,c4)} &= \frac{1}{4} + \frac{1}{4} + \frac{1}{5} = 0,7 \\
 DG_{(alu,c5)} &= \frac{1}{4} + \frac{1}{4} + \frac{1}{1} + \frac{1}{5} = 1,7 \\
 DG_{(alu,c6)} &= \frac{1}{4} + \frac{1}{1} + \frac{1}{5} = 1,45
 \end{aligned}$$

d) Cálculo das auto-forças: Representa quanto o uso do recurso está balanceado quando uma operação está escalonada em cada um dos passos de controle. O recurso é considerado balanceado quando a auto-força é negativa. É dado pela Equação 3.6, onde $w(r)$ representa o custo para o recurso calculado e $t(cs)$ é o intervalo de tempo para o passo de controle cs . O termo $x(ni,ck)$ representa o deslocamento do vértice ni , quando estiver escalonado no passo de controle cs usa-se a Equação 3.6(i), caso contrário usa-se a Equação 3.6(ii).

$$\begin{aligned}
 SF_{(ni,cs)} &= \sum_{ck \in Fni} \left(\left(DG_{(r,ck)} + \frac{1}{3} * x(ni,ck) \right) * x(ni,ck) * w(r) * t(cs) \right) \\
 & \hspace{20em} (3.6) \\
 x(ni,ck) &= \begin{cases} 1 - \left(\frac{|ST_{ni}|}{|St_{ni}|} \right) & (i) \\ - \left(\frac{|ST_{ni}|}{|St_{ni}|} \right) & (ii) \end{cases}
 \end{aligned}$$

$$\begin{aligned}
 SF_{(n3,c1)} &= \left(\left(2,5 + \frac{1}{3} * \left(1 - \frac{1}{2} \right) \right) * \left(1 - \frac{1}{2} \right) * 75 * 28 \right) + \left(\left(3 + \frac{1}{3} * (1-1) \right) * (1-1) * 75 * 47 \right) + \\
 & \left(\left(2 + \frac{1}{3} * \left(-\frac{1}{2} \right) \right) * \left(-\frac{1}{2} \right) * 75 * 28 \right) = 875 \\
 SF_{(n3,c2)} &= \left(\left(2,5 + \frac{1}{3} * \left(-\frac{1}{2} \right) \right) * \left(-\frac{1}{2} \right) * 75 * 28 \right) + \left(\left(3 + \frac{1}{3} * (1-1) \right) * (1-1) * 75 * 47 \right) + \\
 & \left(\left(2 + \frac{1}{3} * \left(1 - \frac{1}{2} \right) \right) * \left(1 - \frac{1}{2} \right) * 75 * 28 \right) = -175
 \end{aligned}$$

$$SF_{(n5,c6)} = \left(\left(0,7 + \frac{1}{3} * \left(-\frac{1}{4} \right) \right) * \left(-\frac{1}{4} \right) * 28 * 28 \right) + \left(\left(0,7 + \frac{1}{3} * \left(-\frac{1}{4} \right) \right) * \left(-\frac{1}{4} \right) * 28 * 47 \right) +$$

$$\left(\left(1,7 + \frac{1}{3} * \left(-\frac{1}{4} \right) \right) * \left(-\frac{1}{4} \right) * 28 * 28 \right) + \left(\left(1,45 + \frac{1}{3} * \left(1 - \frac{1}{4} \right) \right) * \left(1 - \frac{1}{4} \right) * 28 * 28 \right) = 359$$

$$SF_{(n7,c3)} = \left(\left(2 + \frac{1}{3} * \left(1 - \frac{1}{2} \right) \right) * \left(1 - \frac{1}{2} \right) * 75 * 28 \right) + \left(\left(2 + \frac{1}{3} * (1-1) \right) * (1-1) * 75 * 47 \right) +$$

$$\left(\left(0,5 + \frac{1}{3} * \left(-\frac{1}{2} \right) \right) * \left(-\frac{1}{2} \right) * 75 * 28 \right) = 1925$$

$$SF_{(n7,c4)} = \left(\left(2 + \frac{1}{3} * \left(-\frac{1}{2} \right) \right) * \left(-\frac{1}{2} \right) * 75 * 28 \right) + \left(\left(2 + \frac{1}{3} * (1-1) \right) * (1-1) * 75 * 47 \right) +$$

$$\left(\left(0,5 + \frac{1}{3} * \left(1 - \frac{1}{2} \right) \right) * \left(1 - \frac{1}{2} \right) * 75 * 28 \right) = -1225$$

$$SF_{(n10,c2)} = \left(\left(0,2 + \frac{1}{3} * \left(1 - \frac{1}{5} \right) \right) * \left(1 - \frac{1}{5} \right) * 28 * 47 \right) + \left(\left(0,7 + \frac{1}{3} * \left(-\frac{1}{5} \right) \right) * \left(-\frac{1}{5} \right) * 28 * 28 \right) +$$

$$\left(\left(0,7 + \frac{1}{3} * \left(-\frac{1}{5} \right) \right) * \left(-\frac{1}{5} \right) * 28 * 47 \right) + \left(\left(1,7 + \frac{1}{3} * \left(-\frac{1}{5} \right) \right) * \left(-\frac{1}{5} \right) * 28 * 28 \right) +$$

$$\left(\left(1,45 + \frac{1}{3} * \left(-\frac{1}{5} \right) \right) * \left(-\frac{1}{5} \right) * 28 * 28 \right) = -248$$

$$SF_{(n10,c3)} = \left(\left(0,2 + \frac{1}{3} * \left(-\frac{1}{5} \right) \right) * \left(-\frac{1}{5} \right) * 28 * 47 \right) + \left(\left(0,7 + \frac{1}{3} * \left(1 - \frac{1}{5} \right) \right) * \left(1 - \frac{1}{5} \right) * 28 * 28 \right) +$$

$$\left(\left(0,7 + \frac{1}{3} * \left(-\frac{1}{5} \right) \right) * \left(-\frac{1}{5} \right) * 28 * 47 \right) + \left(\left(1,7 + \frac{1}{3} * \left(-\frac{1}{5} \right) \right) * \left(-\frac{1}{5} \right) * 28 * 28 \right) +$$

$$\left(\left(1,45 + \frac{1}{3} * \left(-\frac{1}{5} \right) \right) * \left(-\frac{1}{5} \right) * 28 * 28 \right) = -69$$

$$SF_{(n10,c4)} = \left(\left(0,2 + \frac{1}{3} * \left(-\frac{1}{5} \right) \right) * \left(-\frac{1}{5} \right) * 28 * 47 \right) + \left(\left(0,7 + \frac{1}{3} * \left(-\frac{1}{5} \right) \right) * \left(-\frac{1}{5} \right) * 28 * 28 \right) +$$

$$\left(\left(0,7 + \frac{1}{3} * \left(1 - \frac{1}{5} \right) \right) * \left(1 - \frac{1}{5} \right) * 28 * 47 \right) + \left(\left(1,7 + \frac{1}{3} * \left(-\frac{1}{5} \right) \right) * \left(-\frac{1}{5} \right) * 28 * 28 \right) +$$

$$\left(\left(1,45 + \frac{1}{3} * \left(-\frac{1}{5} \right) \right) * \left(-\frac{1}{5} \right) * 28 * 28 \right) = 410$$

$$SF_{(n10,c5)} = \left(\left(0,2 + \frac{1}{3} * \left(-\frac{1}{5} \right) \right) * \left(-\frac{1}{5} \right) * 28 * 47 \right) + \left(\left(0,7 + \frac{1}{3} * \left(-\frac{1}{5} \right) \right) * \left(-\frac{1}{5} \right) * 28 * 28 \right) +$$

$$\left(\left(0,7 + \frac{1}{3} * \left(-\frac{1}{5} \right) \right) * \left(-\frac{1}{5} \right) * 28 * 47 \right) + \left(\left(1,7 + \frac{1}{3} * \left(1 - \frac{1}{5} \right) \right) * \left(1 - \frac{1}{5} \right) * 28 * 28 \right) +$$

$$\left(\left(1,45 + \frac{1}{3} * \left(-\frac{1}{5} \right) \right) * \left(-\frac{1}{5} \right) * 28 * 28 \right) = 715$$

$$SF_{(n10,c6)} = \left(\left(0,2 + \frac{1}{3} * \left(-\frac{1}{5} \right) \right) * \left(-\frac{1}{5} \right) * 28 * 47 \right) + \left(\left(0,7 + \frac{1}{3} * \left(-\frac{1}{5} \right) \right) * \left(-\frac{1}{5} \right) * 28 * 28 \right) +$$

$$\left(\left(0,7 + \frac{1}{3} * \left(-\frac{1}{5} \right) \right) * \left(-\frac{1}{5} \right) * 28 * 47 \right) + \left(\left(1,7 + \frac{1}{3} * \left(-\frac{1}{5} \right) \right) * \left(-\frac{1}{5} \right) * 28 * 28 \right) +$$

$$\left(\left(1,45 + \frac{1}{3} * \left(1 - \frac{1}{5} \right) \right) * \left(1 - \frac{1}{5} \right) * 28 * 28 \right) = 519$$

e) Adição das forças antecessoras e sucessoras: O escalonamento de uma operação pode afetar o escalonamento das operações antecessoras e sucessoras, nestes casos a auto-força do vértice antecessor ou sucessor é somado à auto força em análise.

$$\text{Força total (n3,c1): } SF(n3,c1) + SF(n7,c3) = 875 + 1925 = 2800$$

$$\text{Força total (n3,c2): } SF(n3,c2) + SF(n7,c4) = -175 -1225 = -1400$$

$$\text{Força total (n4,c1): } SF(n4,c1) + SF(n10,c2) = -347 -248 = -595$$

$$\text{Força total (n4,c3): } SF(n4,c3) + SF(n10,c4) = 6 + 410 = 416$$

$$\text{Força total (n4,c4): } SF(n4,c4) + SF(n10,c5) = 467 + 715 = 1182$$

$$\text{Força total (n4,c5): } SF(n4,c5) + SF(n10,c6) = 790 + 519 = 1309$$

f) Adição das forças de *trigger*: Quando há somente um antecessor capaz de ativar a operação em análise, então a auto-força do antecessor é somado a auto força em análise.

$$\text{Auto força(n3,c2) = } SF(n3,c2)+SF(n4,c1) = -1400 -595 = -1995$$

$$\text{Auto força (n10,c4) = } SF(n10,c4)+SF(n3,c2) = 410 -1995 = -1585$$

g) Escalonar a operação com menor auto-força: após o cálculo das auto-forças, o vértice ni com a menor auto-força é selecionada para o escalonamento. Neste caso, o vértice

escalonado foi $\langle n3, c2 \rangle$, ou seja, vértice $n3$ escalonado no passo $c2$. Os passos anteriores são repetidos até o escalonamento de todos os vértices, como mostrado na sequência abaixo.

$\langle n1, c1 \rangle; \langle n2, c1 \rangle; \langle n3, c2 \rangle; \langle n4, c1 \rangle; \langle n5, c3 \rangle; \langle n6, c3 \rangle; \langle n7, c4 \rangle; \langle n8, c5 \rangle; \langle n9, c6 \rangle;$
 $\langle n10, c4 \rangle$

h) Inserir arcos de escalonamento: Os arcos de escalonamento são inseridos quando um vértice é acionado por outro que não tem dependência de dados, para o caso do exemplo, dois arcos foram adicionados, entre o vértice $4 \rightarrow 3$ e entre $3 \rightarrow 10$. Estes novos arcos são utilizados durante o processo de síntese do controlador.

A Figura 3.7 apresenta o grafo de fluxo de dados após o método de escalonamento *force-directed*.

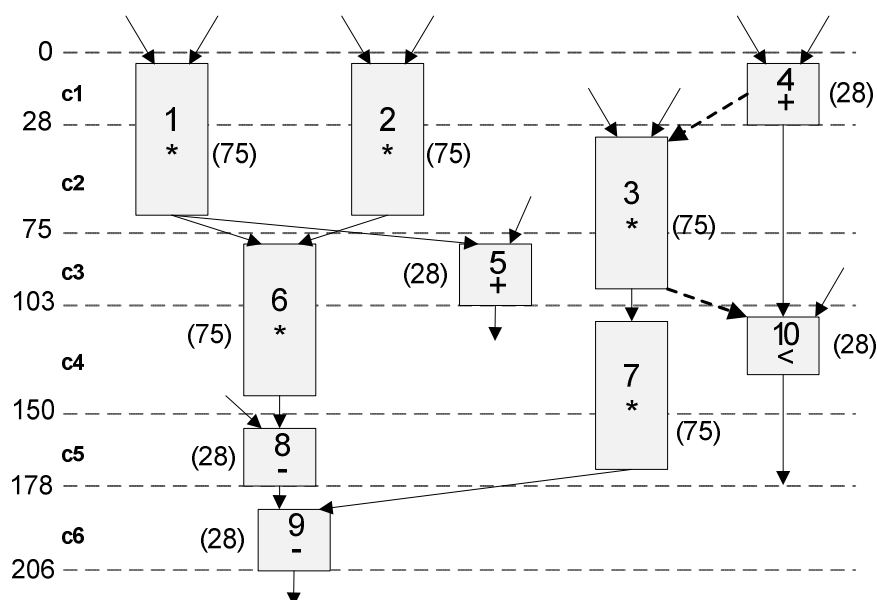


FIGURA 3.7 – Grafo de fluxo de dados escalonado por *force-directed*.

3.3.3 Escalonamento Assíncrono: *Branch-and-Bound*

Branch-and-bound é um método que enumera as soluções candidatas à solução ótima do problema, analisando apenas uma fração das soluções. Para tal, o cálculo dos limites de busca são definidos ao longo da enumeração.

A utilização do algoritmo *branch-and-bound* na solução de problemas de escalonamento assíncrono apresentado por Hansen e Singh (2010) consiste basicamente em

tratar o escalonamento como uma permutação de eventos. Os eventos correspondem às operações a serem escalonadas.

O método parte de um grafo de fluxo de dados (GFD), onde cada vértice do grafo corresponde a uma operação a ser escalonada. Entre os vértices são inseridos arcos para representar o fluxo e a dependência entre as operações.

Dois vértices adicionais são acrescentados ao GFD para permitir um único ponto de início e de término para a análise, o vértice *source* e o vértice *sink*. Todos os vértices da especificação que não tem antecessores são conectados ao vértice *source* e todos os vértices sem sucessores são conectados ao vértice *sink*.

Antes de executar o algoritmo de escalonamento é necessário nomear cada um dos vértices do GFD com uma identificação única e associar os parâmetros definidos nos recursos, tais como área e latência, aos vértices. A partir destas informações é possível calcular, conforme Equação 3.7, dois parâmetros: a) STTS: *shortest time to start*, que representa o menor tempo em que os vértices podem iniciar, considerando que não há limitações de recursos. b) STTF: *shortest time to finish*, que representa o menor tempo que o vértice em questão e todos seus sucessores diretos levarão para finalizar a execução. A dependência entre os vértices é denotada por $i \leftarrow j$, indicando que i é dependente de j .

$$\begin{aligned} STTS(V_i) &= \max_{j|i \leftarrow j} (STTS(V_j) + EXE(V_j)) \\ STTF(V_j) &= \max_{j|j \leftarrow i} (STTF(V_j) + EXE(V_i)) \end{aligned} \quad (3.7)$$

A Figura 3.8 ilustra um GFD com os vértices *sink*, *source* e também os parâmetros STTS e STTF, considerando um tempo de latência de 8 unidades de tempo.

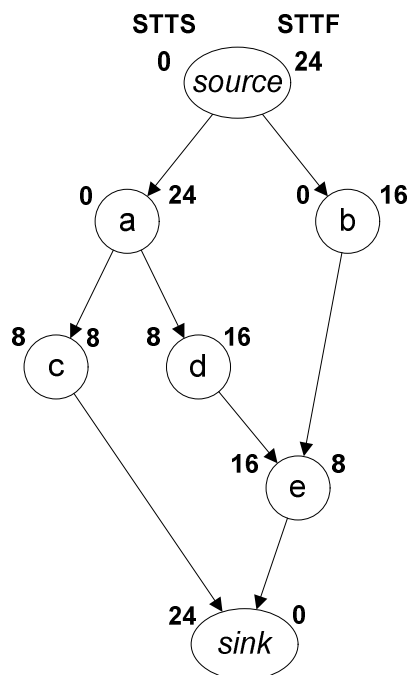


FIGURA 3.8 – GFD com parâmetros STTF e STTS.

Cada vértice do GFD pode receber dois tipos de eventos: nó+ ou nó-. Para representar o início de uma operação, o evento nó+ é associado ao vértice, da mesma maneira para representar o término da operação é usado o evento nó-. Somente é permitido um evento por vez em cada vértice e o nó+ sempre deve preceder o nó-. Sempre que houver dependência de dados entre vértices os antecessores devem finalizar antes do início do seu sucessor, então considerando a dependência $i \leftarrow j$ (i dependente de j), j - sempre deve ocorrer antes de i -. Adicionalmente, pode-se reduzir o espaço de busca aplicando as seguintes regras à enumeração dos nós:

1) Ordem lexicográfica: A sequência das operações deve ser ordenada e as sequências redundantes eliminadas. Ex. $a^+ b^+$ tem o mesmo significado que $b^+ a^+$, portanto $b^+ a^+$ não será pesquisado.

2) Ordem de término e início de operação: Quando ocorrer um evento nó+ e um nó- em um mesmo instante de tempo, nó- deve preceder nó+.

O escalonamento é representado por uma sequência monotônica de nomes que identificam as operações, seguidas pelos tipos de eventos associados. Por exemplo, considerando o exemplo da Figura 3.8, um possível escalonamento é dado por: $a^+ b^+ a^- b^- c^+ d^+ c^- d^- e^+ e^-$. Neste caso, a interpretação do escalonamento é a seguinte: os vértices “a” e “b”

iniciam simultaneamente, as demais operações não podem ser iniciadas, pois há dependência de dados. Após o término de “a” e “b”, estão disponíveis para iniciar os vértices “c” e “d”. Por fim, após o término de “c” e “d” é iniciado e finalizado o último vértice “e”. Um escalonamento alternativo seria: a+ a- b+ b- d+ d- e+ e- c+ c-. Neste último, as operações são escalonadas de forma serial, ou seja, sem que haja execução concorrente entre as operações. Para os dois exemplos o tamanho total da sequência de escalonamento é o mesmo, isto é, o dobro do número de vértices do GFD.

Para obter a sequência de escalonamento a partir do GFD, este método faz o mapeamento das operações descritas no GFD em um grafo dirigido acíclico (GDA), conforme os seguintes passos:

a) O primeiro evento inserido no GDA é o vértice *source+*, pois não há vértices antecessores.

b) Em seguida, é enumerada uma lista das operações que podem ser iniciadas após o *source+*, seguindo as restrições apresentadas anteriormente para que se obtenha um escalonamento válido. O único evento que satisfaz as restrições neste caso é o vértice *source-*.

c) Cada elemento válido da lista gera um novo vértice no GDA que poderá ser expandido da mesma maneira que o vértice anterior, até atingir o vértice *sink* do GFD, conforme mostrado parcialmente na Figura 3.9. Expandir todos os elementos no GDA significa testar todas as soluções possíveis, o que na prática não é viável devido ao tempo computacional necessário, portanto torna-se necessário aplicar restrições para a enumeração da lista de operações, capaz de reduzir o espaço de busca, sem perder a solução ótima.

d) Ao atingir o vértice *sink*, uma sequência foi obtida, representando um escalonamento válido, mas não necessariamente ótimo. A expansão continua em outros ramos do GDA, até que se encontre uma solução com latência total igual ou inferior ao tempo de latência total obtido anteriormente. O tempo inicial é definido serializando todas as operações, ou seja, somando os tempos de latência.

e) As sequências redundantes são eliminadas da enumeração. Por exemplo: a+ b+ c+ tem o mesmo significado que a sequência c+ b+ a+. Da mesma forma que a- b- c- é igual a c- b- a-.

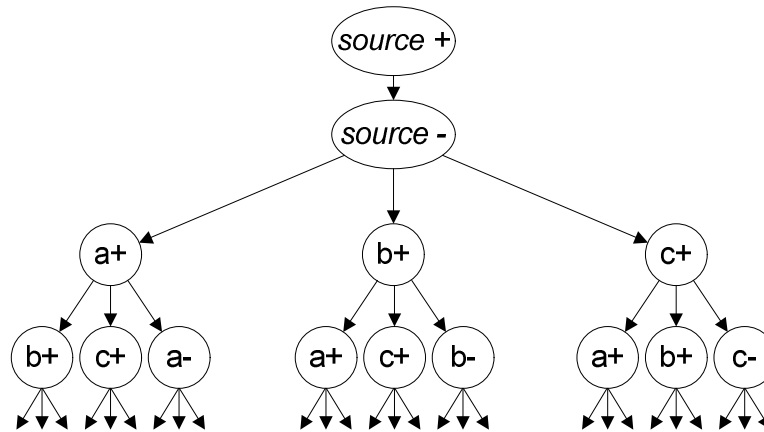


FIGURA 3.9 – Expansão parcial de um grafo com três operações.

Para exemplificar o processo, considere o GFD da Figura 3.10 como entrada. Este grafo está indicando os parâmetros STTF e STTS para cada um dos vértices, conforme descrito anteriormente e pretende-se encontrar o escalonamento com o menor tempo de latência, considerando como recursos disponíveis duas UFs (Unidade Funcional) de multiplicação com tempo de latência de 75 unidades e uma UF ULA (Unidade Lógica Aritmética) com latência de 28 unidades, que executa soma, subtração e comparação.

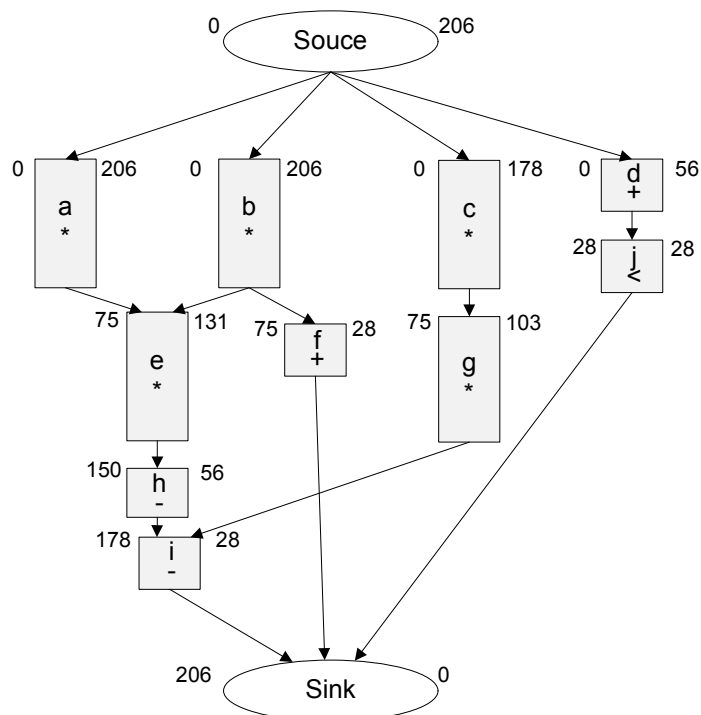


FIGURA 3.10 – Grafo de fluxo de dados a ser escalonado pelo algoritmo *branch-and-bound*.

A primeira estimativa do tempo de latência total do circuito é obtida somando-se o tempo de latência de todas as operações, dessa forma, obtém-se um limite superior.

$$mTime = 75 + 75 + 75 + 75 + 75 + 28 + 28 + 28 + 28 + 28 = 515$$

Análise do GFD e geração do GDA:

(enumeração) \rightarrow source+

Escalonamento: source+

(enumeração) \rightarrow source-

Escalonamento: source+ source-

(enumeração) \rightarrow a+, b+, c+, d+ \rightarrow (aplicando as restrições, seleciona vértice): a

Escalonamento: source+ source- a+

(enumeração) \rightarrow a-, b+, c+, d+ \rightarrow (aplicando as restrições, seleciona vértice): b

Escalonamento: source+ source- a+ b+

(enumeração) \rightarrow a-, b-, c+, d+ \rightarrow (aplicando as restrições, seleciona vértice): d

Escalonamento: source+ source- a+ b+ d+

(enumeração) \rightarrow a-, b-, c+, d- \rightarrow (aplicando as restrições, seleciona vértice): d

Escalonamento: source+ source- a+ b+ d+ d-

(enumeração) \rightarrow a-, b-, c+, j+ \rightarrow (aplicando as restrições, seleciona vértice): j

Escalonamento: source+ source- a+ b+ d+ d- j+

(enumeração) \rightarrow a-, b-, c+, j- \rightarrow (aplicando as restrições, seleciona vértice): j

Escalonamento: source+ source- a+ b+ d+ d- j+ j-

(enumeração) \rightarrow a-, b-, c+ \rightarrow (aplicando as restrições, seleciona vértice): a

Escalonamento: source+ source- a+ b+ d+ d- j+ j- a-

(enumeração) \rightarrow b-, c+ \rightarrow (aplicando as restrições, seleciona vértice): b

Escalonamento: source+ source- a+ b+ d+ d- j+ j- a- b-

(enumeração) \rightarrow c+, e+, f+ \rightarrow (aplicando as restrições, seleciona vértice): c

Escalonamento: source+ source- a+ b+ d+ d- j+ j- a- b- c+

(enumeração) \rightarrow c-, e+, f+ \rightarrow (aplicando as restrições, seleciona vértice): e

Escalonamento: source+ source- a+ b+ d+ d- j+ j- a- b- c+ e+

(enumeração) \rightarrow c-, e-, f+ \rightarrow (aplicando as restrições, seleciona vértice): f

Escalonamento: source+ source- a+ b+ d+ d- j+ j- a- b- c+ e+ f+

(enumeração) \rightarrow c-, e-, f- \rightarrow (aplicando as restrições, seleciona vértice): f

Escalonamento: source+ source- a+ b+ d+ d- j+ j- a- b- c+ e+ f+ f-
 (enumeração) → c-, e- → (aplicando as restrições, seleciona vértice): c

Escalonamento: source+ source- a+ b+ d+ d- j+ j- a- b- c+ e+ f+ f- c-
 (enumeração) → g+, e- → (aplicando as restrições, seleciona vértice): e

Escalonamento: source+ source- a+ b+ d+ d- j+ j- a- b- c+ e+ f+ f- c- e-
 (enumeração) → g+, h+ → (aplicando as restrições, seleciona vértice): g

Escalonamento: source+ source- a+ b+ d+ d- j+ j- a- b- c+ e+ f+ f- c- e- g+
 (enumeração) → g-, h+ → (aplicando as restrições, seleciona vértice): h

Escalonamento: source+ source- a+ b+ d+ d- j+ j- a- b- c+ e+ f+ f- c- e- g+ h+
 (enumeração) → g-, h- → (aplicando as restrições, seleciona vértice): h

Escalonamento: source+ source- a+ b+ d+ d- j+ j- a- b- c+ e+ f+ f- c- e- g+ h+ h-
 (enumeração) → g- → (aplicando as restrições, seleciona vértice): g

Escalonamento: source+ source- a+ b+ d+ d- j+ j- a- b- c+ e+ f+ f- c- e- g+ h+ h- g-
 (enumeração) → i+ → (aplicando as restrições, seleciona vértice): i

Escalonamento: source+ source- a+ b+ d+ d- j+ j- a- b- c+ e+ f+ f- c- e- g+ h+ h- g- i+
 (enumeração) → i- → (aplicando as restrições, seleciona vértice): i

Escalonamento: source+ source- a+ b+ d+ d- j+ j- a- b- c+ e+ f+ f- c- e- g+ h+ h- g- i+ i-
 (enumeração) → sink+ → (aplicando as restrições, seleciona vértice): sink

Escalonamento: source+ source- a+ b+ d+ d- j+ j- a- b- c+ e+ f+ f- c- e- g+ h+ h- g- i+ i-
 sink+ sink-

Ao atingir o nó *sink*, um escalonamento válido foi obtido. A Tabela 3.1 apresenta a sequência obtida juntamente com os tempos associados ao início e término de cada operação, representando de forma textual o grafo de fluxo de dados escalonado.

TABELA 3.1 – Representação textual do GFD escalonado.

a+	b+	d+	d-	j+	j-	a-	b-	c+	e+	f+	f-	c-	e-	g+	h+	h-	g-	i+	i-
0	0	0	28	28	56	75	75	75	75	75	103	150	150	150	150	178	225	225	253

Como o tempo obtido é menor que o inicial ($253 < 515$), então este escalonamento é armazenado e *mTime* passa a ser 253. A busca continua até que não haja mais vértices a serem enumerados. Caso um novo escalonamento com menor tempo for encontrado, o anterior será substituído. Neste exemplo não houve nenhum escalonamento com tempo menor, portanto o

escalonamento final é o mesmo obtido na Tabela 3.1. A Figura 3.11 mostra o resultado do escalonamento obtido numa representação gráfica.

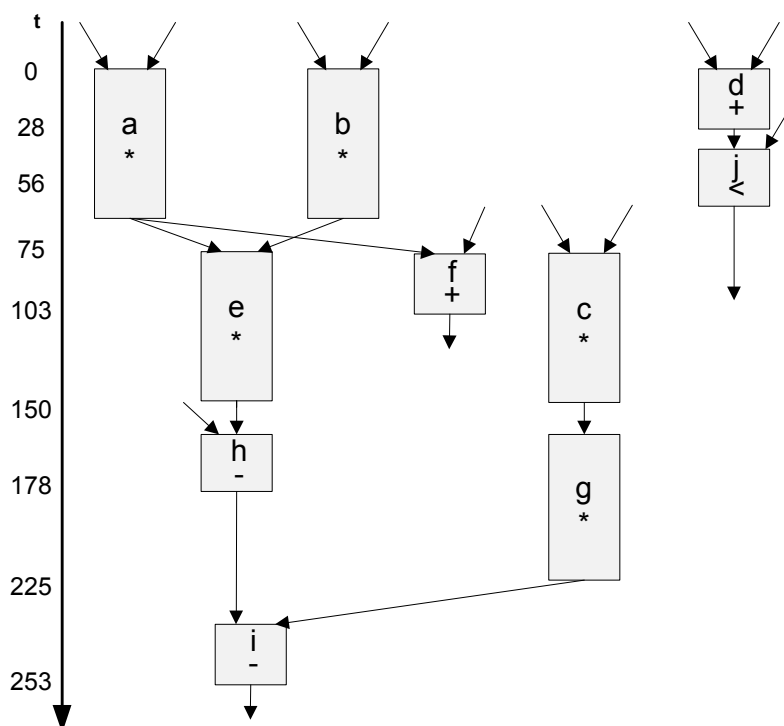


FIGURA 3.11 – Grafo de fluxo de dados escalonado por *branch-and-bound*.

3.4 Assinalamento de Unidades Funcionais (*Component Binding*)

A relação entre um componente (Unidade Funcional) e as operações do grafo de fluxo de dados é definida pelo assinalamento de unidades funcionais. Enquanto a alocação de recursos define o conjunto de recursos disponíveis e o escalonamento define a sequência de execução das operações, o assinalamento de unidades funcionais escolhe qual componente será usado para cada operação. Considerando o exemplo da Figura 3.12, que mostra um escalonamento composto de três somadores e um multiplicador, verifica-se que é possível utilizar dois assinalamentos diferentes para a operação de soma. Uma possibilidade é executar a soma “op1” e “op3” no mesmo componente e a soma “op2” em outro componente. Outra possibilidade seria executar a soma “op2” e “op3” no mesmo componente e a soma “op1” em outro. Do ponto de vista funcional, qualquer dos dois assinalamentos é válido, entretanto a

escolha entre um ou outro pode alterar consideravelmente o número de multiplexadores necessários para interligar os componentes.

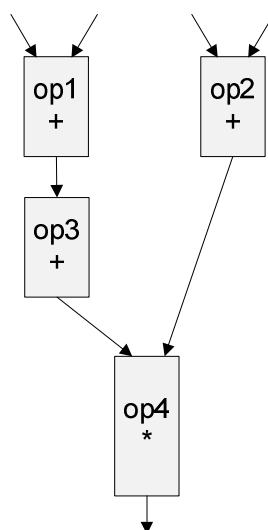


FIGURA 3.12 – Assinalamento de unidades funcionais.

Os algoritmos tradicionalmente utilizados para o assinalamento de unidades funcionais em circuitos síncronos, podem ser utilizados diretamente para os assíncronos, contanto que não sejam baseados em passos de controle (BADIA; CORTADELLA, 1993), caso contrário, o algoritmo precisa ser modificado.

A proposta apresentada em (BACHMAN, 1998), utiliza o algoritmo *left-edge* para o assinalamento das unidades funcionais. Este algoritmo é originalmente baseado em passos de controle e, portanto, esta abordagem propõe duas modificações para o algoritmo: a) substituir os passos de controle pelos tempos de início das operações; b) quando há dependência de dados entre duas operações de mesmo tipo, elas são compatíveis entre si e podem compartilhar o mesmo recurso.

Recentemente, Hansen (2012) utilizou em sua proposta um assinalamento de unidades funcionais baseado no algoritmo *round-robin*, onde a busca é feita analisando-se o escalonamento em ordem crescente de tempo, cada início de operação é assinalada a primeira unidade funcional disponível. Quando há concorrência entre operações de mesmo tipo, impõe-se a seleção em ordem lexicográfica, o que apenas garante que para um dado escalonamento, sempre será obtido o mesmo assinalamento. Devido à simplicidade do algoritmo, este não visa à obtenção da solução ótima.

3.5 Assinalamento de Registradores

O assinalamento de registradores é a operação que indica em qual registrador cada uma das variáveis será armazenada. Avaliando o tempo de vida das variáveis é possível reduzir o número de registradores, fazendo com que um registrador possa receber o valor de mais de uma variável ao longo da execução das operações. Caso duas variáveis estejam ativas em um dado intervalo de tempo, estas são consideradas incompatíveis e não podem compartilhar o mesmo registrador.

O algoritmo heurístico *left-edge* (HASHIMOTO, 1971) é frequentemente utilizado para realizar essa função. Apesar de não garantir uma solução ótima, obtém resultados satisfatórios utilizando poucos recursos computacionais (KIM, 1999). A utilização deste algoritmo para o paradigma assíncrono é feita substituindo os passos de controle associados ao *clock* pelos tempos de início e término de cada operação indicada pelo escalonamento.

3.6 Geração do *Datapath*

Após a execução do escalonamento e assinalamento de UFs e registradores, as portas de origem e destino para as conexões estão definidas no *datapath*. Dependendo da arquitetura desejada para as conexões é possível que existam conflitos no acesso a componentes. As arquiteturas mais comuns são as baseadas em multiplexadores e baseadas em barramentos.

A arquitetura baseada em multiplexador é mais simples, entretanto, quanto maior complexidade do circuito, maior a dificuldade de se obter uma solução ótima.

A arquitetura baseada em barramentos pode ser utilizada para reduzir o número de conexões. Entretanto, como diversos recursos compartilham o mesmo barramento, torna-se necessário realizar uma análise de conflitos para garantir seu funcionamento.

3.7 Geração da Especificação do Controlador

Após todo o processo de síntese comportamental aplicado ao grafo de fluxo de dados, é possível extrair do *datapath* todos os sinais de controle necessários. Essa informação associada ao grafo de fluxo de controle permite gerar a especificação de um controlador.

4 Síntese Automática de Sistemas Assíncronos por Decomposição

O procedimento de síntese por decomposição visa gerar um circuito otimizado em nível RTL a partir de uma descrição comportamental do problema. Para aplicações reais faz-se necessário o uso de ferramentas capazes de executar o procedimento de síntese de forma automática, pois permite reduzir o tempo e o número de erros de projeto.

Neste Capítulo é apresentada a ferramenta BUDASYN, que foi desenvolvida em linguagem C, com aproximadamente 8000 linhas de código que permitem gerar um circuito assíncrono no estilo decomposição *bundled-data* a partir de uma descrição comportamental do problema. O diagrama de blocos mostrado na Figura 4.1 mostra o fluxo de projeto executado pela ferramenta.

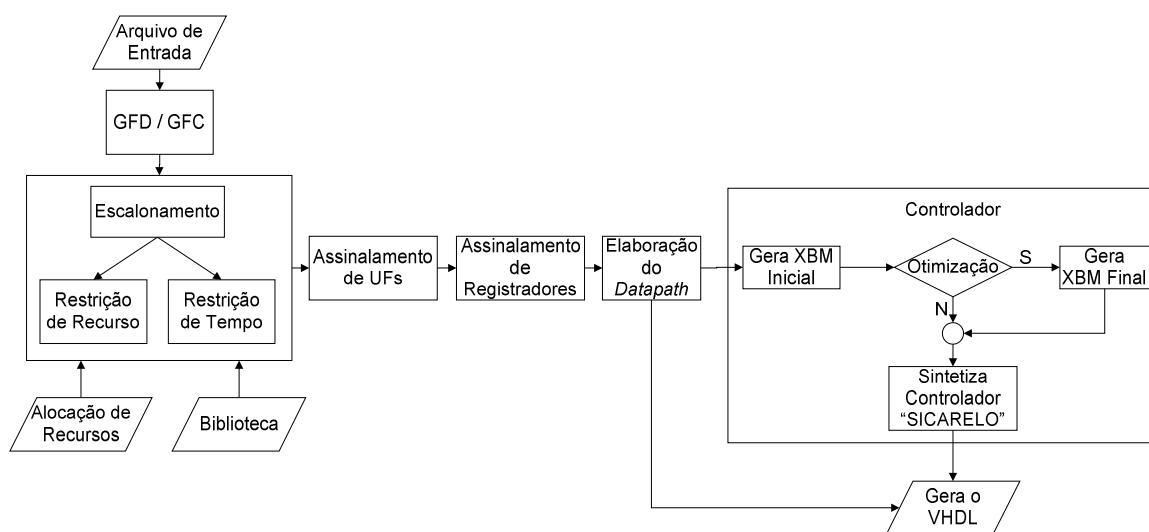


FIGURA 4.1 – Fluxo do projeto BUDASYN.

4.1 Arquivo de Entrada

A linguagem utilizada para descrever o comportamento do circuito desejado é bastante simples, apresentando alguns elementos semelhantes às linguagens de descrição de hardware. Ela permite descrever algoritmos com múltiplas entradas e saídas, além de permitir o uso de

laços e decisões. A Figura 4.2 mostra um exemplo de um algoritmo hipotético descrito na linguagem proposta.

```

in b (8)
in d (8)
in f (8)
var a (8)
var e (8)
var c (8)
out x (8)
out y (8)
out w (8)
out z (8)
out cmp(1)
out cmp2(1)
out cmp3(1)

main:
a <: 1
e <: 0
c <: 1

if[cmp <: c < f]
  c <: c * d
  z <: c
  while[cmp2 <: e < 20]
    e <: e + 2 * b
    w <: e
    while[cmp3 <: a <= 5]
      a <: (a + b * d) - 2
      y <: a
    endwhile
  endwhile
else
  x <: a - b
endif
end

```

FIGURA 4.2 – Exemplo de descrição usando a linguagem proposta.

Todas as entradas, saídas e variáveis com seus respectivos números de bits, devem ser definidas antes do início da descrição do algoritmo. Os sinais de entrada são identificados pela palavra reservada “*in*” e são utilizadas para descrever as entradas do circuito. Os sinais de saída são identificados pela palavra reservada “*out*” e são utilizados para descrever as saídas do circuito, podendo receber valores diretamente de uma expressão ou de uma variável. As variáveis, identificadas por “*var*”, representam os sinais que fazem a função de entrada e saída simultaneamente, como por exemplo a variável “*c*” do exemplo da Figura 4.2. Variáveis precisam ser inicializadas após o início do programa (“*main:*”) com uma entrada ou constante. A declaração do número de bits deve estar contida no intervalo entre 1 e 64 bits.

Os laços são definidos pelo comando ***while***[<out> <: <a> <rel>] → <block> → ***endwhile***, onde *out* especifica uma saída de 1 bit para armazenar o resultado do teste

condicional, $\langle a \rangle$ e $\langle b \rangle$ podem ser entradas, variáveis ou constantes. A palavra $\langle rel \rangle$ deve receber um operador relacional. Cada laço *while* deve ser encerrado com um identificador *endwhile*. As expressões a serem executadas dentro do laço são inseridas em $\langle block \rangle$.

Os testes condicionais são feitos usando o comando $if[\langle out \rangle \langle : \langle a \rangle \langle rel \rangle \langle b \rangle] \rightarrow \langle block \rangle \rightarrow else \rightarrow \langle block \rangle \rightarrow endif$. Assim como no laço de repetição, deve-se especificar uma saída de 1 bit para armazenar o resultado do teste condicional, $\langle a \rangle$ e $\langle b \rangle$ pode ser entradas, variáveis ou constantes. A palavra $\langle rel \rangle$ deve receber um operador relacional. Cada decisão *if* deve conter um identificador *else* e deve ser encerrada com um identificador *endif*. As expressões da condição devem ser inseridas em $\langle block \rangle$.

A leitura do código encerra-se ao encontrar a palavra (end).

Esta linguagem permite o uso de operadores relacionais para os testes condicionais e de operadores aritméticos ou lógicos de bit a bit para as expressões. Os operadores reconhecidos são descritos na Tabela 4.1.

TABELA 4.1 – Conjunto de operadores permitidos por BUDASYN.

Símbolo	Função	Tipo	Símbolo	Função	Tipo
+	soma	aritmético	<	menor	relacional
-	subtração	aritmético	>	maior	relacional
*	multiplicação	aritmético	=	igual	relacional
/	divisão	aritmético	!	diferente	relacional
<<	desl. à esquerda	aritmético	<=	menor ou igual	relacional
>>	desl. à direita	aritmético	>=	maior ou igual	relacional
&	E	bit a bit			
	OU	bit a bit			
^	OU Exclusivo	bit a bit			

4.2 Extração do GFD e GFC

A primeira etapa executada pela ferramenta é a extração do grafo de fluxo de dados e do grafo de fluxo de controle, que ocorre durante a análise sintática (*parsing*) do arquivo de entrada. Esta versão da ferramenta não executa o balanceamento automático das expressões, ficando a cargo do usuário fazê-lo. Como exemplo, considere a expressão: $a+b+c+d$ que pode ser reescrita de forma balanceada, ficando da seguinte forma: $(a+b)+(c+d)$.

Os grafos são extraídos individualmente, pois na síntese por decomposição o circuito é dividido em controlador e *datapath*, facilitando assim os processos de otimização aplicados tanto ao controlador quanto ao *datapath*.

A Figura 4.3 mostra de forma abstrata os grafos GFC e GFD, extraídos do código apresentado na Figura 4.2.

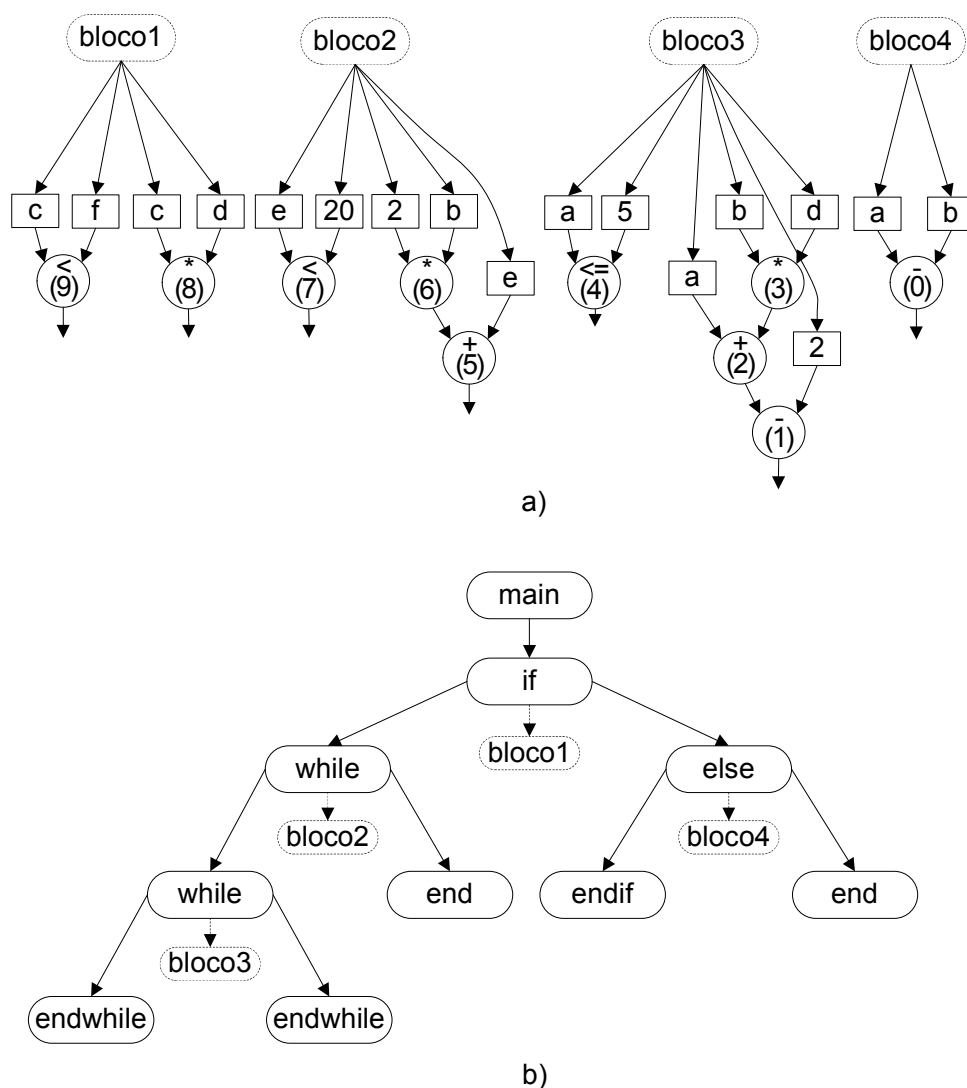


FIGURA 4.3 – a) Grafo de fluxo de dados. b) Grafo de fluxo de controle.

O GFD é extraído da especificação separado em blocos. Cada bloco representa um GFD com um conjunto de operações. Cada um dos blocos inicia com um identificador (MAIN, IF, ELSE ou WHILE) e termina em outro identificador (IF, ELSE, WHILE, END, ENDIF ou ENDWHILE). No processo de tradução do código para GFD, os operadores são enumerados em sequência decimal crescente.

O GFC é extraído da especificação buscando os identificadores (MAIN, IF, ELSE, WHILE, ENDIF, ENDWHILE, END) e gerando o grafo com a seguinte interpretação: Cada

identificador gera um vértice; Os arcos interligados à esquerda indicam o sequenciamento dos identificadores (WHILE ou IF) que terminam com um identificador (ENDWHILE, ENDIF ou END), continuando à direita do último vértice. Então, quando há um aninhamento de identificadores (IF e ou WHILE), estes estarão conectados à esquerda; Cada identificador (MAIN, IF, ELSE, WHILE) contém uma referência a um bloco correspondente.

4.3 Escalonamento

O escalonamento das operações é realizado a partir do GFD utilizando o algoritmo *branch-and-bound*, pois conforme análise descrita no Capítulo 3, este algoritmo permite representar de forma mais eficiente o escalonamento assíncrono, por não se basear em passos de controle. Além disso, consegue obter uma solução de maneira rápida sem perder a solução ótima.

BUDASYN permite escolher dois tipos de escalonamento: MTRR – Minimização do Tempo com Restrição de Recursos; MRRT – Minimização de Recursos com Restrição de Tempo.

Quando o método de escalonamento for MTRR, deve-se criar um arquivo texto com o nome “Recursos.dat” especificando a alocação de recursos desejada, ou seja, deve-se definir o número de unidades funcionais disponíveis, conforme modelo mostrado na Figura 4.4(a). Este método encontra o escalonamento de menor latência utilizando os recursos disponíveis.

Quando é utilizado o escalonamento MRRT, o arquivo “Recursos.dat” não é necessário. Neste método é solicitada a latência total desejada para o circuito e o algoritmo define o menor número de UFs (unidades funcionais) necessário. Para que o usuário da ferramenta tenha parâmetros para definir a latência desejada são apresentadas as seguintes informações: a) Identificação dos vértices e escalonamento ASAP (*as soon as possible*); b) Tempo de escalonamento mínimo permitido, calculado a partir do caminho crítico. O caminho crítico é definido pela somatória das latências individuais das UFs que apresenta a maior dependência de dados; c) Tempo de escalonamento máximo, definido pela somatória das latências de todas as UFs, ou seja, considerando a máxima serialização do circuito.

Para ambos os métodos é necessário informar previamente, através de um arquivo com o nome “Biblioteca.dat” a latência de cada um dos tipos de UF utilizada, conforme exemplo mostrado na Figura 4.4(b). As UFs permitidas são do tipo multiplicação, divisão e unidade lógica aritmética. Exceto pelas operações de multiplicação e divisão, todas as demais funções

descritas na Tabela 4.1 são executados pela ALU e devem ser definidas por uma única latência.

Como a etapa de escalonamento considera somente o GFD, para circuitos que contém vários blocos, ou seja, onde a especificação tenha mais de um laço de repetição ou decisões, duas abordagens podem ser consideradas: a) escalonar cada bloco individualmente; b) escalonar todos os blocos simultaneamente. Nesta proposta, optou-se por escalonar todos os blocos simultaneamente.

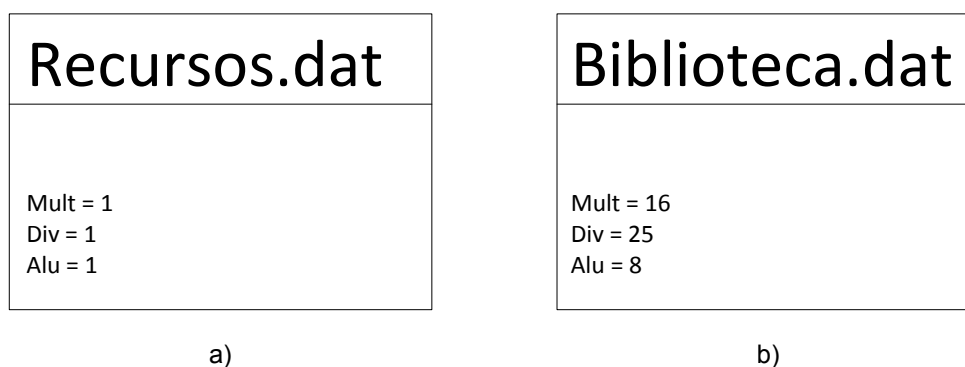


FIGURA 4.4 – Exemplos de arquivos: a) Alocação de recursos. b) Latências.

Para ilustrar os dois tipos de escalonamento disponíveis, o algoritmo da Figura 4.2 foi escalonado aplicando os tipos MTRR e MRRT, conforme apresentado na Figura 4.5(a) e (b), respectivamente. No escalonamento usando MTRR, os recursos alocados foram de uma unidade funcional de cada tipo, conforme mostrado na Figura 4.4(a). Para o escalonamento MRRT, o tempo de latência solicitado foi de 32 unidades de tempo. Para ambos os casos a latência de cada unidade funcional foi ajustada conforme apresentado na Figura 4.4(b).

O escalonamento obtido por BUDASYN pode ser verificado durante a execução, onde as informações são apresentadas ao usuário de forma textual. A representação textual do escalonamento foi apresentada no Capítulo 3.

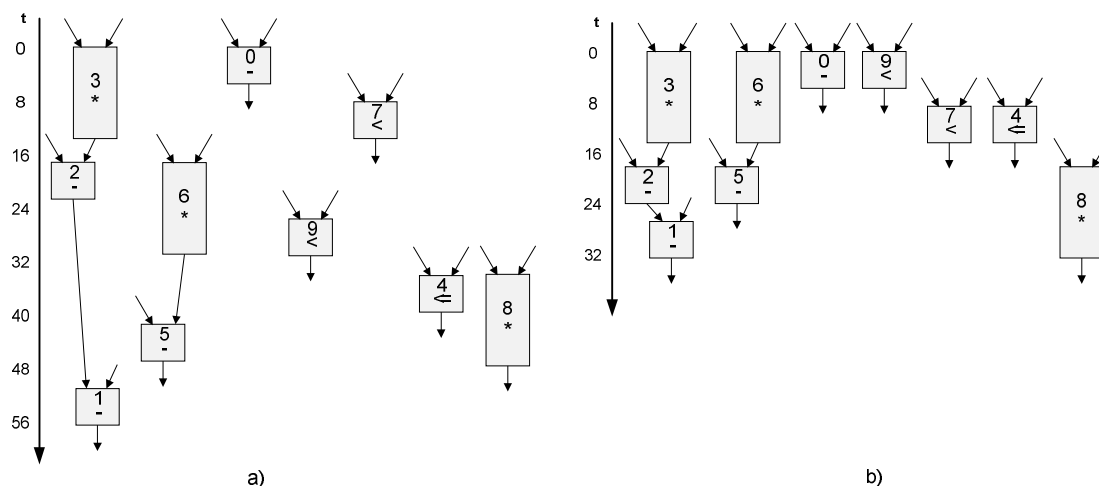


FIGURA 4.5 – Exemplo de escalonamento: a) Escalonamento MTRR. b) Escalonamento MRRT.

4.4 Assinalamento de Unidades Funcionais

BUDASYN utiliza para o assinalamento de unidades funcionais o algoritmo *left-edge* modificado proposto para o paradigma assíncrono.

Como o escalonamento é feito considerando todos os blocos, nos casos onde existem vários laços de repetição ou decisões, é possível reduzir o número de unidades funcionais determinadas inicialmente pelo escalonamento, percorrendo o grafo de fluxo de controle e identificando o conjunto de operações contidas nos blocos que serão executadas em cada parte da especificação. A Figura 4.6 apresenta o pseudocódigo utilizado para o assinalamento de unidades funcionais.

```

Left-edge Assíncrono Modificado{
  Percorrer GFC e identificar blocos contendo GFDs
  Enquanto houver blocos{
    Para cada tipo de UF ∈ blocon{
      Percorre escalonamento{
        Se início de operação{
          Gera assinalamento
        }
        Se término de operação{
          Libera recurso
        }
      }
    }
  }
}

```

FIGURA 4.6 – Pseudocódigo do algoritmo de assinalamento de UFs.

Para exemplificar o algoritmo, considere o escalonamento mostrado na Figura 4.7(a), onde na primeira linha estão as operações, na segunda linha o tempo e na terceira as UFs. Este escalonamento foi gerado a partir da descrição comportamental hipotética da Figura 4.7(c). Este escalonamento é composto por nove operações, identificadas com números inteiros em ordem crescente. As operações que tem dependência de dados estão entre parênteses, por exemplo, a operação 00 é a soma da operação (01) com “x”. A descrição das operações está apresentada a seguir:

- Operação: 00 → (01) + x;
- Operação: 01 → 5 * y;
- Operação: 02 → (03) + x;
- Operação: 03 → 3 * y;
- Operação: 04 → a = y;
- Operação: 05 → x + 1;
- Operação: 06 → b + 2;
- Operação: 07 → a + b;
- Operação: 08 → a < y.

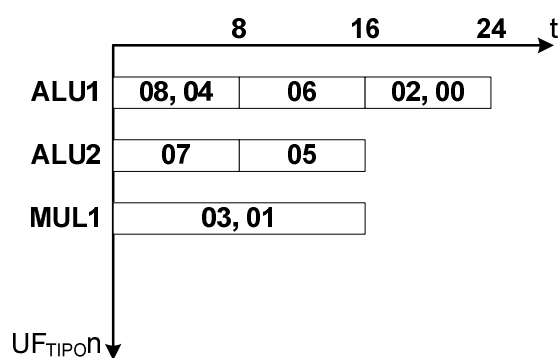
Como as operações de cada bloco não serão executadas simultaneamente, as unidades funcionais assinaladas entre os blocos podem ser reutilizadas. Por exemplo, as operações 03 e 01 podem utilizar o mesmo recurso, pois não serão executadas ao mesmo tempo. Já as

operações 08 e 07, utilizam recursos de mesmo tipo (ALU) ao mesmo tempo, portanto devem ter uma UF para cada. Na Figura 4.7(b) é apresentado um gráfico de utilização das UFs em função do tempo de vida. Neste exemplo há mais de uma operação no mesmo intervalo de tempo, conforme indicado pelo escalonamento, mas que não causam conflitos, pois estão em blocos distintos.

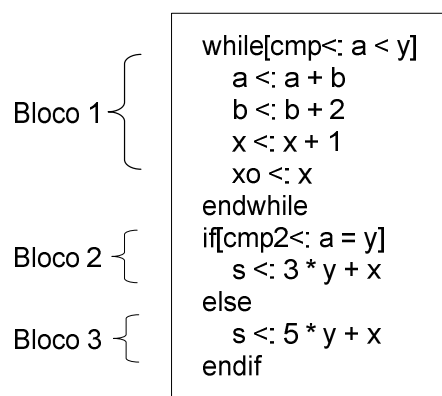
03+	01+	08+	04+	07+	04-	08-	07-	06+	05+	01-	03-	06-	05-	00+	02+	00-	02-
0	0	0	0	0	8	8	8	8	8	16	16	16	16	16	16	24	24
MUL1	MUL1	ALU1	ALU1	ALU2				ALU1	ALU2					ALU1	ALU1		

= Bloco 1
 = Bloco 2
 = Bloco 3

a)



b)



c)

FIGURA 4.7 – a) Escalonamento. b) Tempo de vida das operações. c) Descrição comportamental.

Para este exemplo, o assinalamento de unidades funcionais obtido foi (entre colchetes estão as operações):

- ALU_1{08, 04, 06, 02, 00}

- ALU_2{07, 05}

- MUL_1{03, 01}

Voltando ao exemplo apresentado no início, com a descrição comportamental da Figura 4.2, quando utilizado o escalonamento MRRT, obteve-se o seguinte assinalamento de unidades funcionais:

- MUL_1{03, 06, 08}

- ALU_1{09, 00, 07, 04, 02, 05, 01}

4.5 Assinalamento de Registradores

O assinalamento de registradores é feito utilizando o algoritmo *left-edge* modificado, conforme pseudocódigo da Figura 4.8, onde os recursos são compartilhados quando não há conflitos nos tempos de vida das operações, ou seja, o registrador só pode ser reutilizado quando os dados armazenados não precisarem mais ser utilizados. Adicionalmente, foi inserida uma restrição ao compartilhamento dos registradores, onde o assinalamento é feito individualmente, ou seja, os registradores podem ser reutilizados para as operações executadas por uma mesma unidade funcional. Desta forma, elimina-se a necessidade de inserir multiplexadores entre as unidades funcionais e os registradores, reduzindo assim a complexidade das interligações do *datapath*.

```

Assinalamento de Registradores{
  Percorrer a sequência de escalonamento{
    Para cada tipo de UF{
      Se início de operação{
        Gera assinalamento
      }
      Avalia tempo de vida das variáveis{
        Libera recurso
      }
    }
  }
}

```

FIGURA 4.8 – Pseudocódigo do algoritmo de assinalamento de registradores.

Para exemplificar o algoritmo, considere o escalonamento mostrado na Figura 4.9, onde na primeira linha estão as operações, na segunda linha o tempo e na terceira os registradores. O escalonamento e identificação das operações são os mesmos apresentados para o assinalamento de unidades funcionais.

Cada término de operação recebe um registrador, entretanto quando não há conflitos no tempo de vida das variáveis ou quando estão em blocos distintos e as operações foram assinaladas em uma mesma UF, então pode-se reutilizar o mesmo registrador. O assinalamento obtido neste exemplo foi:

Neste caso o assinalamento de registradores obtido foi:

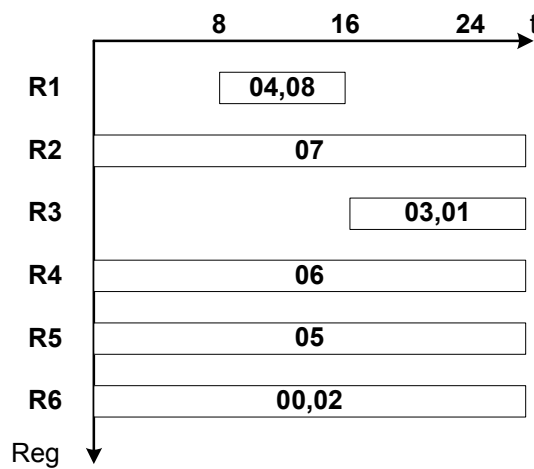
- Reg_1 {04, 08}
- Reg_2 {07}

- Reg_3{03, 01}
- Reg_4{06}
- Reg_5{05}
- Reg_6{00, 02}

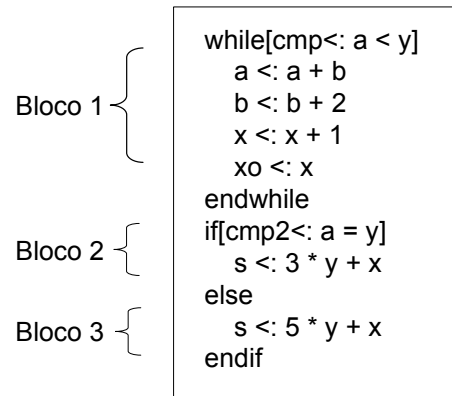
03+	01+	08+	04+	07+	04-	08-	07-	06+	05+	01-	03-	06-	05-	00+	02+	00-	02-
0	0	0	0	0	8	8	8	8	8	16	16	16	16	16	16	24	24
					R1	R1	R2			R3	R3	R4	R5			R6	R6

= Bloco 1
 = Bloco 2
 = Bloco 3

a)



b)



c)

FIGURA 4.9 – a) Escalonamento. b) Tempo de vida das variáveis. c) Descrição comportamental.

Voltando ao exemplo apresentado na Figura 4.2, o assinalamento de registradores encontrado é o seguinte:

- Reg_1{00};
- Reg_2{03, 06};
- Reg_3{07};
- Reg_4{02, 01};
- Reg_5{09};
- Reg_6{04};
- Reg_7{08};
- Reg_8{05}.

4.6 Geração do *Datapath*

O *datapath* é gerado com base nas informações obtidas dos processos de escalonamento, assinalamento de unidades funcionais e assinalamento de registradores. Um arquivo texto é gerado com as informações das conexões entre os componentes e segue o seguinte padrão:

- Cada unidade funcional é conectada aos registradores assinalados;
- Com base no GFD, cada entrada é conectada a sua respectiva unidade funcional, assim como cada registrador é conectado a sua respectiva unidade funcional. Onde houver mais de uma conexão insere-se um multiplexador;
- Em seguida, produz-se as conexões entre os multiplexadores e as UFs;
- Adicionam-se as conexões entre os registradores e as saídas;
- Por fim, se houver inicialização de variáveis, adiciona-se multiplexadores entre a unidade funcional e o registrador.

A Figura 4.10 mostra o pseudocódigo que gera o *datapath* e a Figura 4.11 apresenta a descrição RTL do *datapath* para o exemplo apresentado na Figura 4.2.

```
Datapath{
  Percorrer escalonamento{
    Para cada operação{
      Executar ligação entre UFs → Reg. analisando os assinalamentos
      Analisando o GFD, determinar ligações entre Entradas → UFs (inserir MUXes onde necessário)
      Analisando o GFD, determinar ligações entre Reg. → UFs (inserir MUXes onde necessário)
      Analisando o GFD, determinar ligações entre Reg. → saídas
    }
  }
  Se houver inicialização de variáveis{
    adiciona MUX entre UFs e Reg.
  }
}
```

FIGURA 4.10 – Pseudocódigo para geração do *datapath*.

Após a geração do *datapath*, o arquivo é convertido em um VHDL em nível RTL estrutural, onde estão descritas todas as conexões entre os componentes (UFs, registradores e multiplexadores). Para cada componente do *datapath*, um componente em VHDL é gerado. Os componentes gerados são descritos em nível RTL comportamental. Quando houver fusão de operações, o que é permitido apenas para as operações executadas pela ULA, gera-se em

VHDL uma ULA contendo todas as operações necessárias para execução das funções fundidas nesta UF.

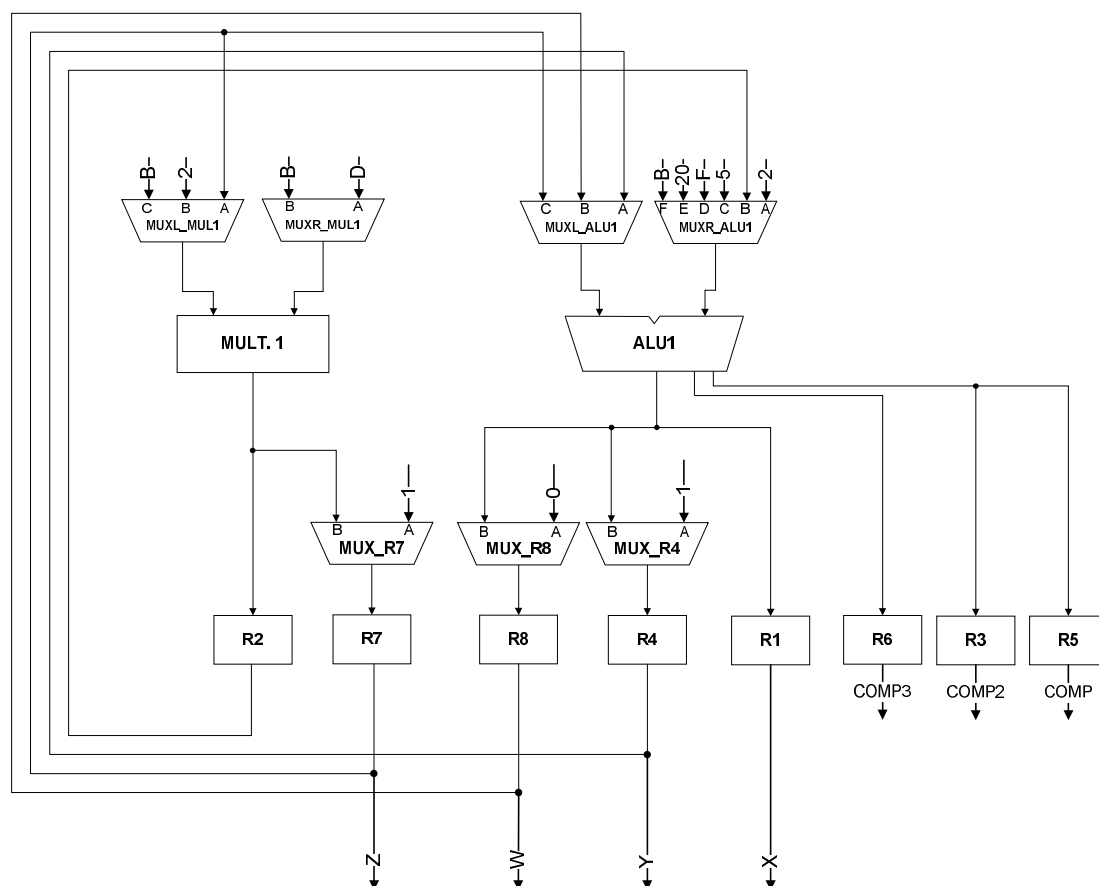


FIGURA 4.11 – Descrição RTL do *datapath*.

4.7 Controlador

O controlador é descrito usando a especificação *extended burst mode* (XBM). Esta especificação foi escolhida porque opera corretamente em ambientes que satisfazem o modo fundamental generalizado, como é o caso do protocolo de comunicação *bundled-data* usado nesta proposta.

O controlador é baseado na proposta de Oliveira *et al.* (2013), que gera uma máquina de estados assíncrona a partir de uma especificação XBM, utilizando os métodos de síntese de circuito síncrono, porém o sinal de *clock* é substituído por uma função combinacional utilizada para geração de um *clock* local. A implementação desta proposta é feita de forma

automática através de uma ferramenta chamada SICARELO, desenvolvida por Curtinhas *et al.* (2014).

A comunicação entre o controlador gerado pelo SICARELO e o *datapath* utiliza o protocolo de duas fases. No protocolo de duas fases, a ativação dos circuitos ocorre nas transições, mas os registradores utilizados no *datapath* são *single-edge* (ativados na borda de subida). Por este motivo, foi adicionado um circuito gerador de *clock*, composto de uma porta Não OU Exclusivo associada ao elemento de atraso, para permitir a comunicação entre o controlador e o *datapath*. Vide arquitetura apresentada na Figura 4.12.

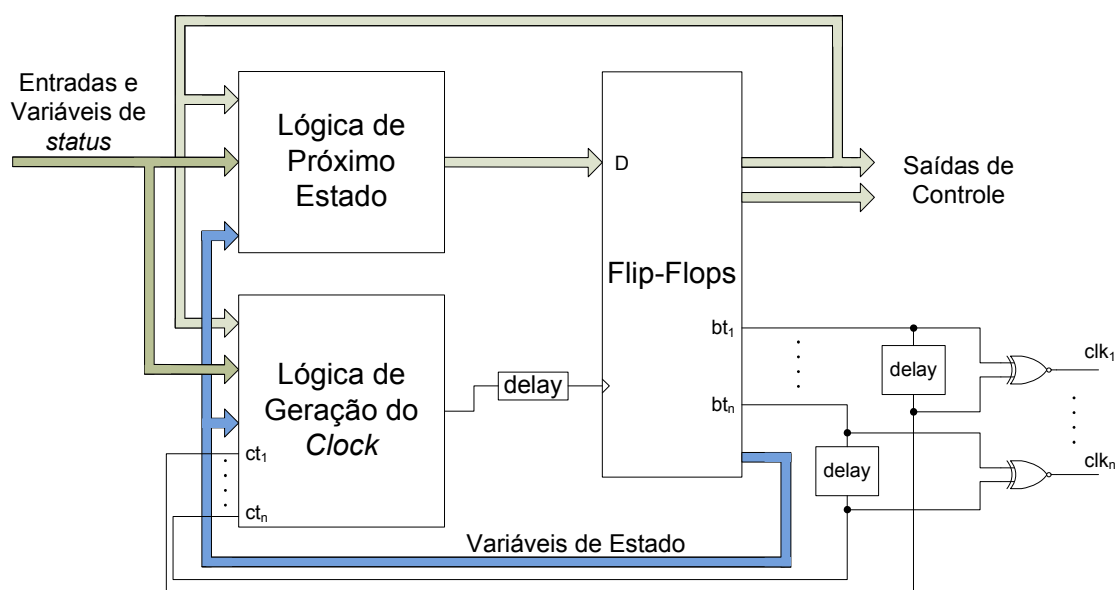


FIGURA 4.12 – Arquitetura do controlador proposto.

A geração da especificação XBM do controlador ocorre em duas etapas: a) gera-se uma especificação XBM inicial; b) Aplica-se um processo de otimização ao XBM inicial, dando origem ao XBM final.

4.8 Geração da Especificação XBM Inicial

O processo de geração do XBM inicial ocorre percorrendo o grafo de fluxo de controle e identificando as operações e seus sinais de controle. A Figura 4.13 mostra de forma simplificada um exemplo de aplicação do processo de geração do XBM inicial, onde estão indicados apenas os sinais referentes ao controle, sendo que “cmp” é o sinal condicional do

laço, “bt” é o sinal de disparo de início de uma operação (os sinais de acionamento do *datapath* estão suprimidos para facilitar o entendimento) e “Ct” é o sinal que indica o término da operação.

Quando o vértice “*main*” é encontrado, adiciona-se a primeira transição de estado à especificação XBM, onde a ativação é feita adicionando uma entrada com nome “*Start*”. Nesta primeira transição é feita a inicialização das variáveis, carregando os registradores com os valores iniciais.

Os demais vértices do GFC são analisados e geram as transições de estados na especificação XBM identificando a referência ao bloco que contém um GFD. Então executa-se uma busca através do *datapath* identificando quais sinais devem ser ativados para executar cada uma das operações do GFD em questão, seguindo a ordem das operações determinadas pelo escalonamento. Para o acionamento de todas as operações contidas no GFD em questão (contidas no bloco associado ao vértice do GFC) pode ser necessário várias transições de estado.

No exemplo da Figura 4.13, a transição $1 \rightarrow 2$ executa o teste condicional para executar o laço. Quando há laços de repetição ou decisões, identifica-se no GFD qual a operação responsável por executar o teste condicional. A identificação é executada percorrendo-se o *datapath* e encontrando-se qual sinal de controle está associado à operação. Estas operações são sempre executadas antes do restante do GFD, garantindo assim que o teste condicional seja executado antes de uma possível atualização do valor da variável envolvida na comparação.

As transições $2 \rightarrow 3$ e $3 \rightarrow 4$ fazem o acionamento do *datapath* para execução das operações (neste exemplo os sinais de acionamento do *datapath* foram suprimidos). Após gerar todas as transições necessárias para executar o GFD, busca-se no GFC o próximo vértice, que neste caso é um “*endwhile*”, que indica a finalização do laço de repetição. Para finalizar o laço, uma transição entre o último estado gerado e o primeiro estado do laço deve ser gerada. Para a correta geração desta transição é necessário realizar uma análise de polaridade dos sinais de controle e, caso necessário, uma transição de estado adicional (transição morta) é acrescentada. Para este exemplo, a transição $4 \rightarrow 5$ é uma transição morta e a transição $5 \rightarrow 2$ executa a finalização do laço, fazendo novamente o teste condicional.

Ao percorrer o GFC e encontrar o vértice “*end*” uma transição para o estado 0 deve ser gerada. Para satisfazer as condições do protocolo de comunicação, um estado é adicionado para permitir a ativação do sinal “*Done*”, que indica o fim do processamento. Neste exemplo, o estado número 6 faz esta função.

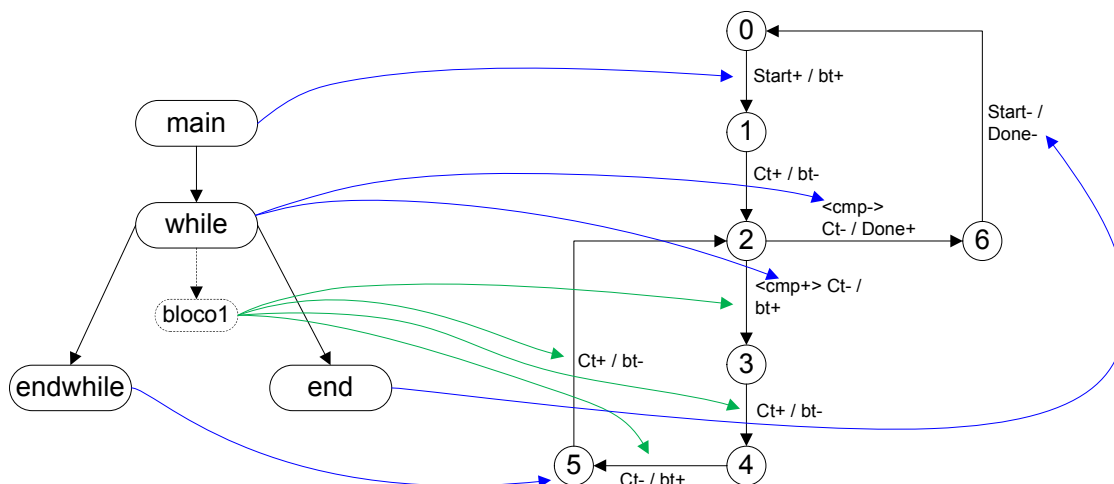


FIGURA 4.13 – Geração do XBM inicial.

4.9 Geração da Especificação XBM Final

A especificação XBM deve satisfazer algumas propriedades, conforme descrito em detalhes no Capítulo 2. Uma das propriedades de interesse para a obtenção de um controlador otimizado é a propriedade de polaridade do sinal, pois ao implementar um laço de repetição, é possível que os sinais sejam incompatíveis, necessitando que uma nova transição de estado (transição morta), seja acrescentada para ajustar as fases dos sinais e satisfazer a propriedade de polaridade do sinal. Entretanto, esta solução penaliza o desempenho do circuito, pois a transição morta não executa nenhuma operação de dados e quando ocorre em um laço de repetição, esta será executada múltiplas vezes, enquanto a condição de repetição estiver satisfeita.

Uma contribuição importante deste trabalho é a proposta de otimização realizada nas especificações XBM que contém transições mortas. Esta proposta parte de uma especificação XBM nomeada de "XBM inicial" e gera uma nova especificação, chamada de "XBM final". A especificação "XBM final" é obtida após identificar e eliminar as transições mortas dos laços de repetição da especificação inicial. Para eliminar a transição morta, uma cópia do laço de repetição é inserida em seu lugar, com ajustes na polaridade dos sinais. A Figura 4.14(a) apresenta um exemplo de uma especificação XBM, os sinais desta especificação são os mesmos apresentados no exemplo da Figura 4.13. Esta especificação contém um laço de

repetição nas transições $2 \rightarrow 3 \rightarrow 4 \rightarrow 5$, sendo que a transição do estado número 2 para o estado número 5 é identificada como transição morta, pois foi inserida somente para satisfazer a condição de polaridade dos sinais. Na Figura 4.14(b) apresenta-se a especificação após a otimização. Neste caso, a transição $4 \rightarrow 5$ é similar a transição $1 \rightarrow 2$ exceto pelos sinais “bt” e “ct” que estão com fases opostas. Da mesma forma, a transição $5 \rightarrow 6$ é similar a transição $2 \rightarrow 3$ e por fim, $6 \rightarrow 7$ é similar a $3 \rightarrow 4$. Após este arranjo, os sinais “bt” e “ct”, entre os estados 7 e 2, apresentam-se em fase, podendo ser conectados diretamente.

O algoritmo que executa esta transformação possui complexidade $O(2^n)$, onde n é o número de laços de repetição que contêm transições mortas. Devido ao comportamento exponencial, uma especificação de entrada com muitas transições mortas gera uma especificação de saída muito grande, dificultando a síntese do controlador. Num trabalho futuro, pretende-se permitir a escolha de quais laços serão otimizados, ou desenvolver um método heurístico para seleção dos laços a serem otimizados.

Apesar desta abordagem apresentar um aumento no tamanho e complexidade da especificação do controlador, após o processo de síntese, este não apresenta um acréscimo na mesma proporção, conforme demonstrado no Capítulo 6.

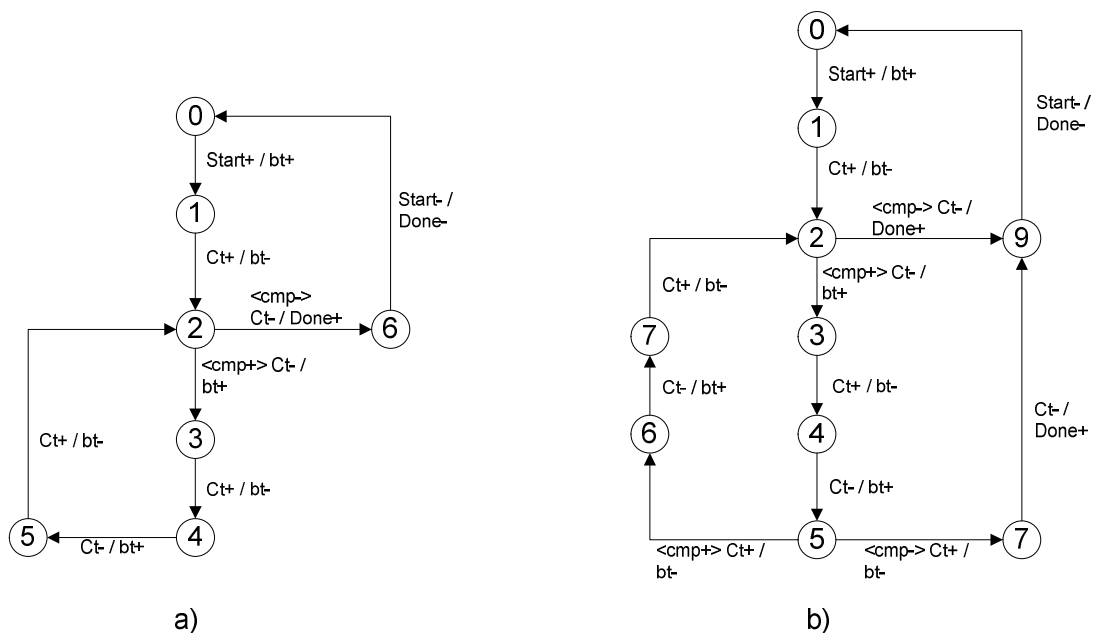


FIGURA 4.14 – Exemplo de transformação da especificação XBM: a) Especificação XBM inicial. b) Especificação XBM final.

Após a conclusão da especificação XBM inicial, avalia-se a necessidade de otimização, aplicando o algoritmo descrito pelo pseudocódigo apresentado na Figura 4.15 e explicado através do exemplo ilustrado nas Figuras 4.13 e 4.14.

```

Transformação do XBM{
    If((Busca Tansição Morta, TM)≠ ∅){
        do{
            While (Transição XBMinicial < TM){
                copia transição
            }
            Armazena estado atual, EA
            Busca transição inicial do laço, Tn
            Armazena estado inicial, EI
            While (Tn < EA){
                gera nova transição NTn
                copia sinais de Tn (exceto bt e ct)
                insere sinais de controle (bt e ct)
                Tn++
            }
            insere sinais entre NTn e EI
            gera transição de fechamento
            Busca Tansição Morta, TM
        }While(TM ≠ ∅{
    }
    else XBMfinal = XBMinicial
}

```

FIGURA 4.15 – Pseudocódigo para geração da especificação XBM otimizada.

Para a especificação apresentada no início deste Capítulo através da Figura 4.2, após aplicar este método, obteve-se a especificação XBM inicial e final que são apresentadas nas Figuras 4.16 e 4.17 respectivamente.

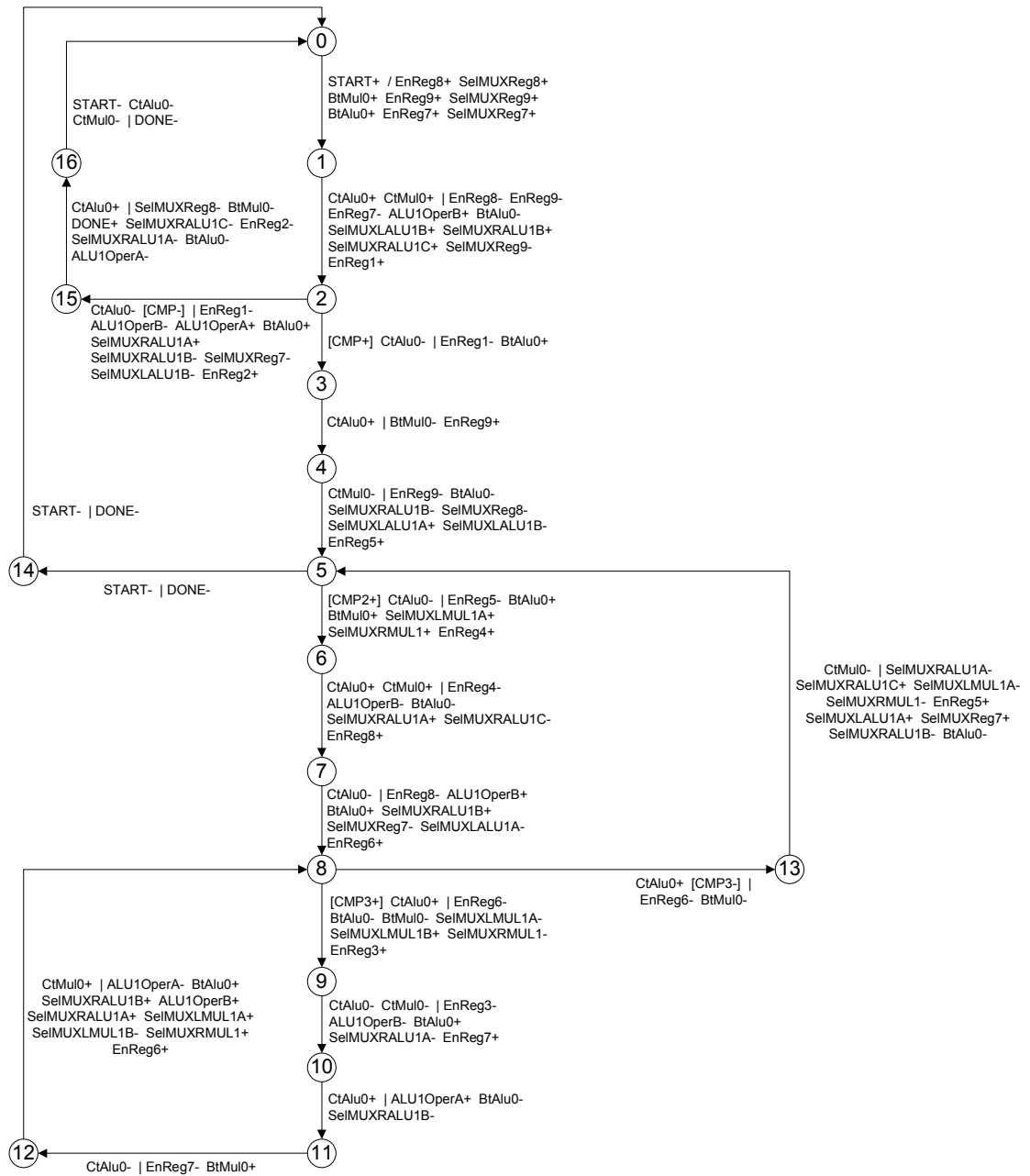


FIGURA 4.16 – Especificação XBM inicial.

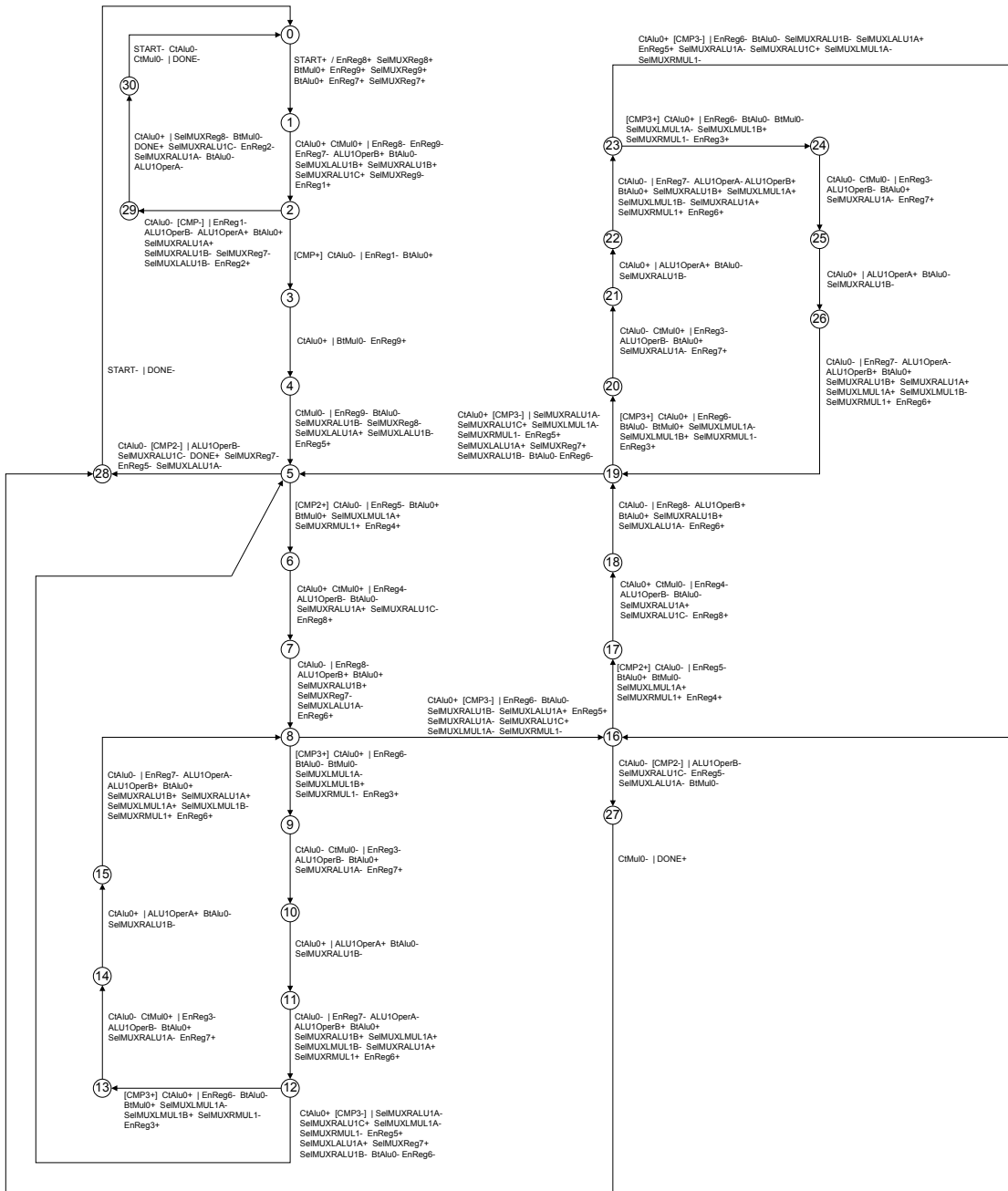


FIGURA 4.17 – Especificação XBM final.

4.10 Conversão para Código VHDL

Após a geração do *datapath* e do controlador, todo o circuito é convertido em um código VHDL, sendo que as unidades funcionais, multiplexadores e registradores são

descritos em VHDL comportamental (*process*). As conexões entre os módulos são realizadas com descrição estrutural, conforme Figura 4.18.

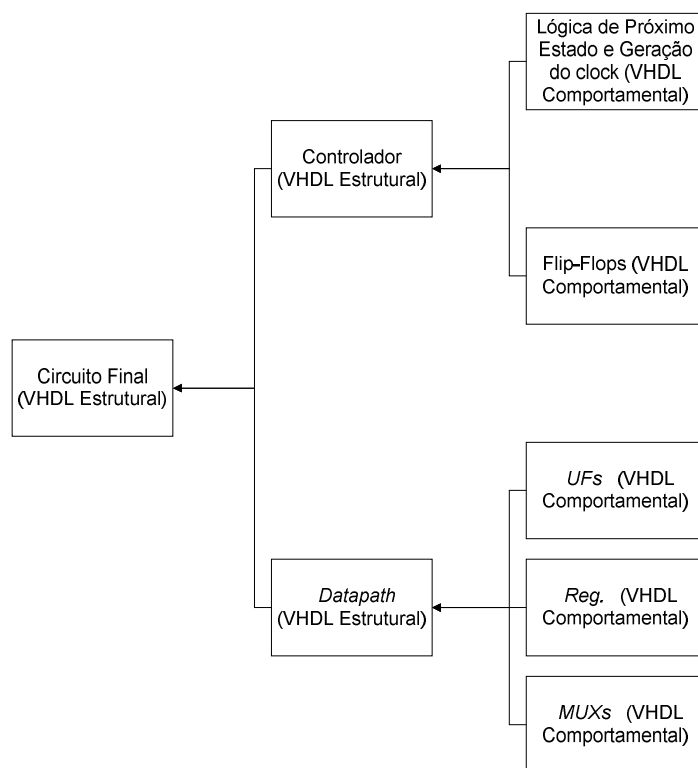


FIGURA 4.18 – Fluxo de geração da descrição VHDL.

4.11 Implementação em FPGA

Para realizar a síntese do circuito final em um dispositivo FPGA basta sintetizar o arquivo gerado pela ferramenta BUDASYN, que contém a descrição RTL do circuito. Nesta versão do BUDASYN, deve-se adicionar os elementos de atraso manualmente, sendo que a identificação para inserir os atrasos seguem o seguinte padrão:

- As entradas dos elementos de atraso devem ser ligadas às portas identificadas como BT_UFx, onde UFx é o nome da unidade funcional e seu número (ex. BT_ALU1);
- As saídas dos elementos de atraso devem ser interligados às portas identificadas como CT_UFx, onde UFx é o nome da unidade funcional e seu número.

Para os testes realizados nesta dissertação foi utilizado o software Quartus II versão 9.1. Dois modelos de elementos de atraso foram testados e a variação do tempo de atraso ocorre acrescentando ou retirando elementos. O atraso apresentado na Figura 4.19(a) é

baseado no elemento *buffer* LCELL, que neste exemplo possui três elementos básicos que podem ser acrescentados ou retirados um a um. O atraso apresentado na Figura 4.19 (b) é constituído de um arranjo com flip-flops e multiplexadores, onde para aumentar o atraso, deve-se adicionar novos conjuntos de multiplexadores e flip-flops.

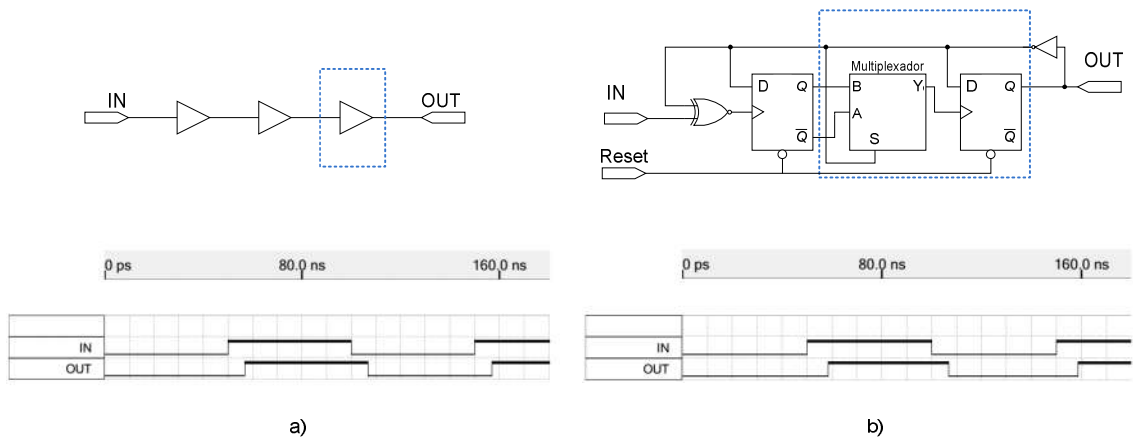


FIGURA 4.19 – Elementos de atraso.

5 Estudo de uma Aplicação – Criptografia TEA

TEA é considerado um algoritmo “*lightweight cryptography*”. Esta nomenclatura é aplicada aos algoritmos de criptografia que exigem poucos recursos de hardware sem comprometer o nível de segurança.

O algoritmo TEA (*Tiny Encryption Algorithm*) de criptografia simétrica foi desenvolvido por Wheeler e Needham (1994) e a operação básica do algoritmo é constituída de adições e lógica OU exclusivo para gerar a não-linearidade e deslocamentos lógicos à esquerda e à direita para proporcionar a mistura dos dados e da chave. Essas operações são repetidas diversas vezes, sendo que os autores sugerem ao menos seis iterações. Para garantir que a cifragem seja diferente a cada iteração, utiliza-se uma constante delta baseada na proporção áurea, conforme apresentada na Equação 5.1. O valor de delta garante que as sub chaves serão distintas e seu valor numérico não tem relevância para a qualidade da criptografia. A Figura 5.1 apresenta um diagrama que representa um ciclo do algoritmo TEA.

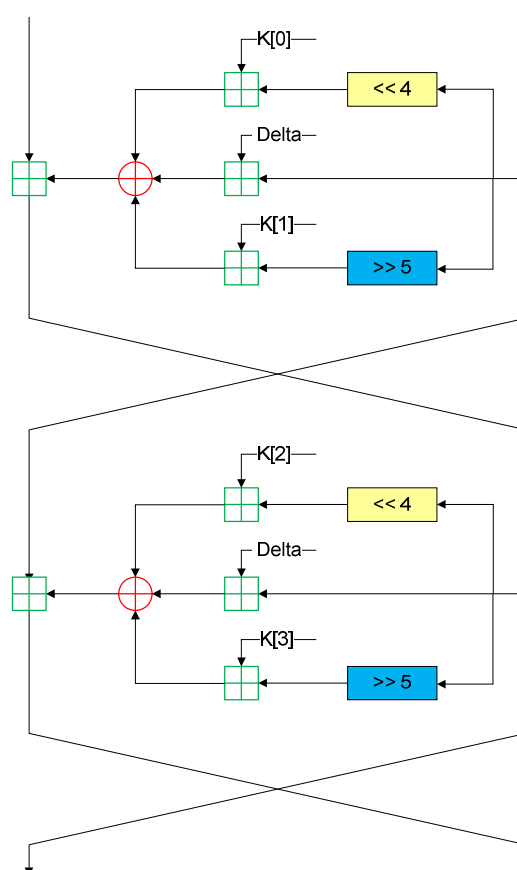


FIGURA 5.1 – 1 ciclo do algoritmo TEA (MAHDI, 2011).

$$\Delta = (\sqrt{5} - 1) \times 2^{31} \quad (5.1)$$

Para demonstrar o funcionamento da ferramenta, este Capítulo apresenta os passos para a síntese do algoritmo TEA. A alocação de recursos será definida pela ferramenta. O método de escalonamento utilizado foi o MRRT (Minimização de Recursos com Restrição de Tempo). O atraso definido para as unidades funcionais foi de 8 unidades de tempo para as unidades lógicas aritméticas, e 16 unidades de tempo para multiplicadores e divisores. Este algoritmo, entretanto, utiliza apenas unidades lógicas aritméticas.

Inicialmente, o algoritmo foi transcrito para a linguagem de entrada da ferramenta BUDASYN, seguindo as regras descritas no Capítulo 4. A Figura 5.2 apresenta o arquivo de entrada. Neste exemplo, os parâmetros adotados foram: $K[0] = 10$; $K[1] = 15$; $K[2] = 20$; $K[3] = 25$; Delta está representado pela variável “sum” e recebe o valor descrito pela Equação 5.1, truncado para obter um número inteiro; “n” representa o número de repetição do laço; “v0” e “v1” são as entradas de dados; “y1” e “y2” são as saídas de dados.

```

in v0 (32)
in v1 (32)
var y (32)
var z (32)
var sum (32)
var n (8)
out cmp (1)
out y1 (32)
out z1 (32)

main:
n <: 8
y <: v0
z <: v1
sum <: 2654435769

while[cmp<: n > 0]
  n <: n - 1
  sum <: sum + 2654435769
  y <: (y + (((z << 4) + 10) ^ (z + sum) ^ ((z >> 5) + 15)))
  z <: z + (((y + (((z << 4) + 10) ^ (z + sum) ^ ((z >> 5) + 15))) << 4) + 20) ^ ((y + (((z << 4) + 10) ^ (z + sum) ^ ((z >> 5) + 15))) + sum) ^ (((y + (((z << 4) + 10) ^ (z + sum) ^ ((z >> 5) + 15))) >> 5) + 25))
  y1 <: y
  z1 <: z
endwhile
end

```

FIGURA 5.2 – Algoritmo TEA descrito na linguagem proposta.

Após a escolha do tipo do escalonamento (MRRT), BUDASYN solicita que seja definido o parâmetro de restrição de tempo, ou seja, o tempo de latência total do *datapath* no

qual as operações serão escalonadas. Neste caso, o valor deve estar entre 80 e 344 (estes valores são apresentados ao usuário durante o processo), e foi escolhido o valor de 80 unidades de tempo, buscando a execução na menor latência total possível. A Figura 5.3(a), (b), (c) e (d) apresenta respectivamente a identificação das operações, os resultados obtidos para o escalonamento, assinalamento de UFs e assinalamento de registradores.

Nó: (0) => Z + (1);
 Nó: (1) => (2) ^ (22);
 Nó: (2) => (3) ^ (13);
 Nó: (3) => (4) + 20;
 Nó: (4) => (5) << 4;
 Nó: (5) => Y + (6);
 Nó: (6) => (7) ^ (11);
 Nó: (7) => (8) ^ (10);
 Nó: (8) => (9) + 10;
 Nó: (9) => Z << 4;
 Nó: (10) => Z + SUM;
 Nó: (11) => (12) + 15;
 Nó: (12) => Z >> 5;
 Nó: (13) => (14) + SUM;
 Nó: (14) => Y + (15);
 Nó: (15) => (16) ^ (20);
 Nó: (16) => (17) ^ (19);
 Nó: (17) => (18) + 10;
 Nó: (18) => Z << 4;
 Nó: (19) => Z + SUM;
 Nó: (20) => (21) + 15;
 Nó: (21) => Z >> 5;
 Nó: (22) => (23) + 25;
 Nó: (23) => (24) >> 5;
 Nó: (24) => Y + (25);
 Nó: (25) => (26) ^ (30);
 Nó: (26) => (27) ^ (29);
 Nó: (27) => (28) + 10;
 Nó: (28) => Z << 4;
 Nó: (29) => Z + SUM;
 Nó: (30) => (31) + 15;
 Nó: (31) => Z >> 5;
 Nó: (32) => Y + (33);
 Nó: (33) => (34) ^ (38);
 Nó: (34) => (35) ^ (37);
 Nó: (35) => (36) + 10;
 Nó: (36) => Z << 4;
 Nó: (37) => Z + SUM;
 Nó: (38) => (39) + 15;
 Nó: (39) => Z >> 5;
 Nó: (40) => SUM + 2654435769;
 Nó: (41) => N - 1;
 Nó: (42) => N > 0;

a)

09+	10+	28+	12+	18+	09-	10-	28-	12-	18-	08+	19+	31+	21+	29+	08-	19-	31-
0	0	0	0	0	8	8	8	8	8	8	8	8	8	8	16	16	16

21-	29-	17+	27+	11+	07+	20+	17-	27-	11-	07-	20-	30+	16+	26+	06+	36+	30-
16	16	16	16	16	16	16	24	24	24	24	24	24	24	24	24	24	32

16-	26-	06-	36-	25+	15+	05+	37+	39+	25-	15-	05-	37-	39-	24+	14+	04+	35+
32	32	32	32	32	32	32	32	32	40	40	40	40	40	40	40	40	40

38+	24-	14-	04-	35-	38-	23+	13+	03+	34+	42+	23-	13-	03-	34-	42-	22+	02+
40	48	48	48	48	48	48	48	48	48	48	56	56	56	56	56	56	56

33+	40+	41+	22-	02-	33-	40-	41-	01+	32+	01-	32-	00+	00-
56	56	56	64	64	64	64	64	64	64	72	72	72	80

b)

ALU_1{9, 8, 17, 30, 25, 24, 23, 22, 1, 0}
 ALU_2{10, 19, 27, 16, 15, 14, 13, 2, 32}
 ALU_3{28, 31, 11, 26, 5, 4, 3, 33}
 ALU_4{12, 21, 7, 6, 37, 35, 34, 40}
 ALU_5{18, 29, 20, 36, 39, 38, 42, 41}

c)

Reg_1{9, 8, 30, 25, 24, 23, 22, 1}
 Reg_2{18, 20, 39, 38}
 Reg_3{12, 7, 6, 37, 34}
 Reg_4{28, 11, 5, 4, 3, 33}
 Reg_5{10, 16, 15, 14, 13, 2}
 Reg_6{29, 36}
 Reg_7{21, 35}
 Reg_8{31, 26}
 Reg_9{19}
 Reg_10{27}
 Reg_11{17}
 Reg_12{42}
 Reg_13{40}
 Reg_14{41}
 Reg_15{32}
 Reg_16{0}

d)

FIGURA 5.3 – Detalhes da implementação obtidos no processo de síntese comportamental para o algoritmo TEA: a) Identificação das operações. b) Representação textual do escalonamento. c) Assinalamento de UFs. d) Assinalamento de registradores.

Após o processo de escalonamento e assinalamentos, BUDASYN gera o *datapath*, conforme apresentado na Figura 5.4.

0) ALU_1 -> Reg_1	32) Reg_4 -> MUXRALU_2 - A	64) Reg_15 -> MUXLALU_3 - C
1) MUXReg_16 -> Reg_16	33) Reg_15 -> MUXLALU_2 - A	65) Reg_2 -> MUXRALU_2 - D
2) ALU_1 -> MUXReg_16 - A	34) Reg_5 -> MUXLALU_1 - B	66) Reg_8 -> MUXLALU_1 - E
3) MUXReg_15 -> Reg_15	35) 1 -> MUXRALU_5 - A	67) 4 -> MUXRALU_5 - E
4) ALU_2 -> MUXReg_15 - A	36) Reg_14 -> MUXLALU_5 - A	68) Reg_4 -> MUXRALU_4 - E
5) MUXReg_14 -> Reg_14	37) 2654435769 -> MUXRALU_4 - A	69) Reg_3 -> MUXLALU_4 - E
6) ALU_5 -> MUXReg_14 - A	38) Reg_13 -> MUXLALU_4 - A	70) Reg_6 -> MUXRALU_3 - E
7) MUXReg_13 -> Reg_13	39) Reg_2 -> MUXRALU_3 - A	71) Reg_10 -> MUXLALU_3 - D
8) ALU_4 -> MUXReg_13 - A	40) Reg_3 -> MUXLALU_3 - A	72) Reg_9 -> MUXRALU_2 - E
9) ALU_5 -> Reg_12	41) Reg_5 -> MUXRALU_2 - B	73) Reg_11 -> MUXLALU_2 - D
10) ALU_2 -> Reg_10	42) Reg_4 -> MUXLALU_2 - B	74) 15 -> MUXRALU_1 - D
11) ALU_1 -> Reg_11	43) 25 -> MUXRALU_1 - B	75) Reg_7 -> MUXLALU_5 - D
12) ALU_5 -> Reg_6	44) Reg_1 -> MUXLALU_1 - C	76) Reg_5 -> MUXRALU_4 - F
13) ALU_4 -> Reg_7	45) 0 -> MUXRALU_5 - B	77) Reg_1 -> MUXLALU_4 - F
14) ALU_3 -> Reg_8	46) Reg_3 -> MUXRALU_4 - B	78) 15 -> MUXRALU_3 - F
15) ALU_2 -> Reg_9	47) Reg_7 -> MUXLALU_4 - B	79) 10 -> MUXRALU_2 - F
16) ALU_5 -> Reg_2	48) 20 -> MUXRALU_3 - B	80) 10 -> MUXRALU_1 - E
17) ALU_4 -> Reg_3	49) Reg_4 -> MUXLALU_3 - B	81) Reg_2 -> MUXLALU_1 - F
18) ALU_3 -> Reg_4	50) Reg_13 -> MUXRALU_2 - C	82) Reg_13 -> MUXRALU_5 - F
19) ALU_2 -> Reg_5	51) Reg_5 -> MUXLALU_2 - C	83) 5 -> MUXRALU_4 - G
20) MUXRALU_5 -> ALU_5	52) 5 -> MUXRALU_1 - C	84) 5 -> MUXRALU_3 - G
21) MUXLALU_5 -> ALU_5	53) 15 -> MUXRALU_5 - C	85) Reg_16 -> MUXLALU_3 - E
22) MUXRALU_4 -> ALU_4	54) Reg_2 -> MUXLALU_5 - B	86) Reg_16 -> MUXLALU_2 - E
23) MUXLALU_4 -> ALU_4	55) 10 -> MUXRALU_4 - C	87) 4 -> MUXRALU_1 - F
24) MUXRALU_3 -> ALU_3	56) Reg_6 -> MUXLALU_4 - C	88) Reg_12 -> CMP
25) MUXLALU_3 -> ALU_3	57) 4 -> MUXRALU_3 - C	89) Reg_15 -> Y1
26) MUXRALU_2 -> ALU_2	58) Reg_15 -> MUXLALU_1 - D	90) Reg_16 -> Z1
27) MUXLALU_2 -> ALU_2	59) 5 -> MUXRALU_5 - D	91) V0 -> MUXReg_15 - B
28) MUXRALU_1 -> ALU_1	60) Reg_16 -> MUXLALU_5 - C	92) V1 -> MUXReg_16 - B
29) MUXLALU_1 -> ALU_1	61) Reg_13 -> MUXRALU_4 - D	93) 2654435769 -> MUXReg_13 - B
30) Reg_1 -> MUXRALU_1 - A	62) Reg_16 -> MUXLALU_4 - D	94) 8 -> MUXReg_14 - B
31) Reg_16 -> MUXLALU_1 - A	63) Reg_3 -> MUXRALU_3 - D	

FIGURA 5.4 – Representação textual do *datapath* para o algoritmo TEA.

Para este exemplo foi gerado o controlador XBM tanto na versão inicial, quanto na versão final. As configurações iniciais são iguais para ambos os casos, portanto o *datapath* gerado é o mesmo para as duas versões.

5.1 Controlador – Versão Inicial

Para a geração da especificação XBM do controlador foi inserida uma transição morta no laço de repetição, como se pode observar na transição de estado número 13 da Figura 5.5. Os sinais de seleção dos multiplexadores foram suprimidos para uma melhor visualização.

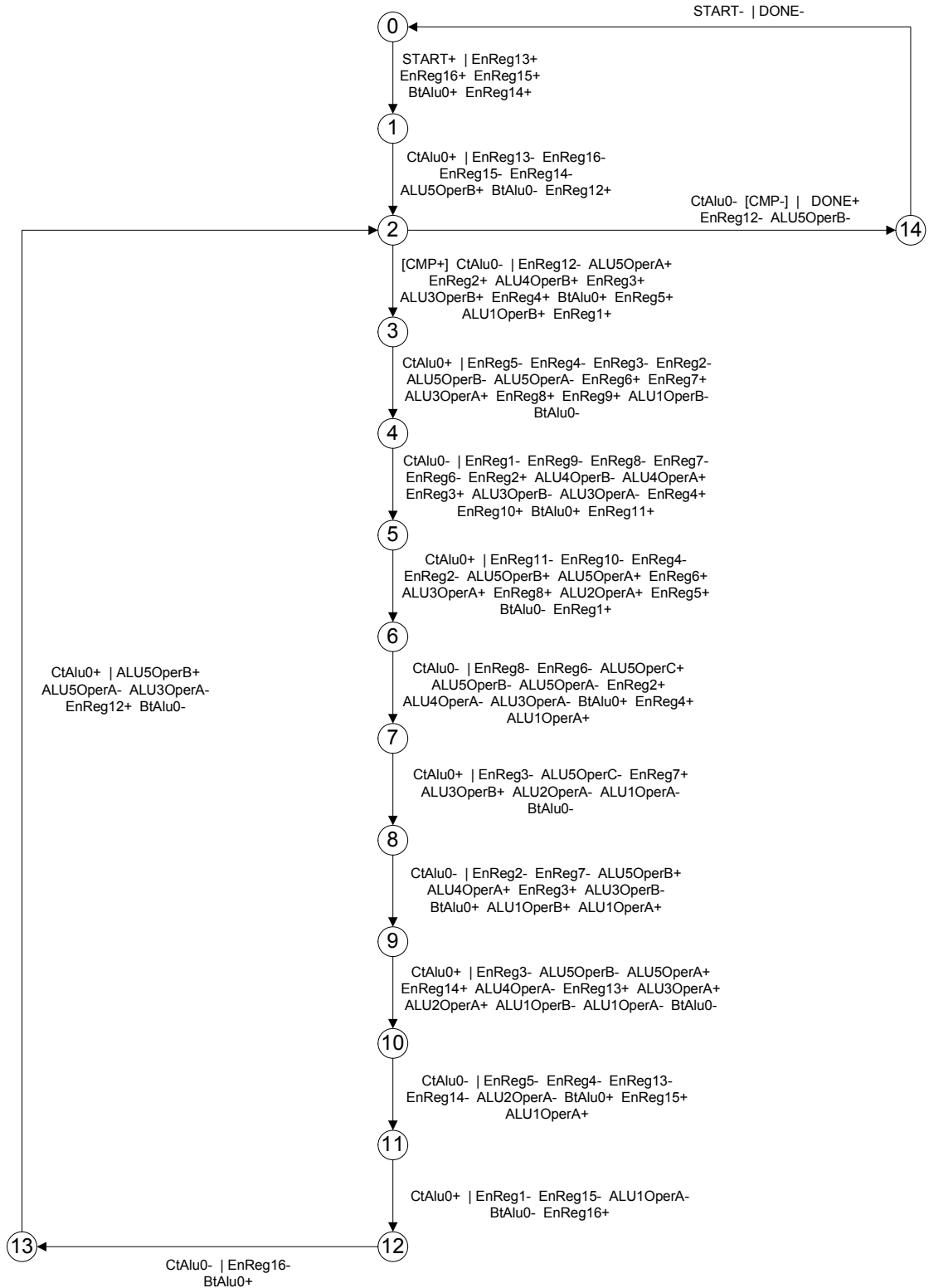


FIGURA 5.5 – XBM inicial, TEA.

A Figura 5.6 mostra a simulação do circuito usando o software Quartus II versão 9.1 da Altera®, utilizando o dispositivo EP2S15F484C3. Nesta simulação os valores de entrada

V0 e V1 foram escolhidos aleatoriamente e após 8 ciclos (definidos pela variável “n”) os valores cifrados aparecem nas saídas Y1 e Z1. Para a síntese deste circuito, foram utilizados 2067 LUTs (Look-Up Table) e 519 Flip-Flops. A latência total para obter o resultado foi de 1,95us.

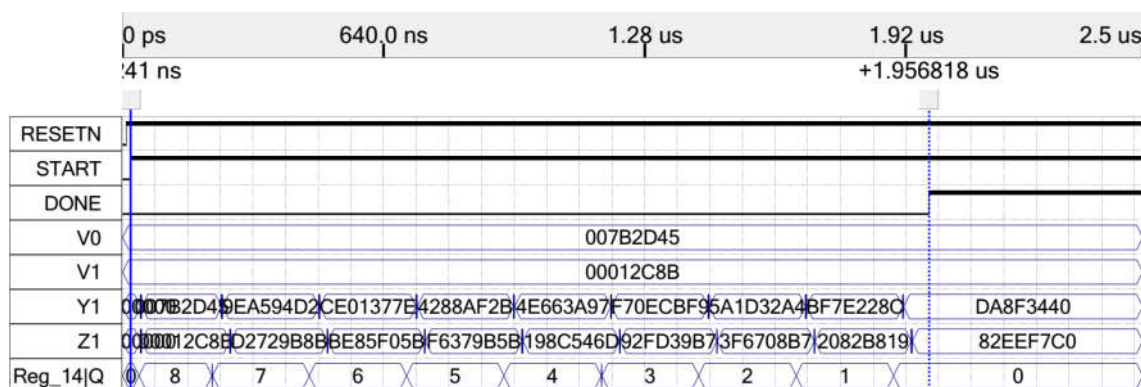


FIGURA 5.6 – Simulação do circuito – XBM inicial.

5.2 Controlador – Versão Final

Quando há possibilidade de eliminação de transições mortas, BUDASYN permite aplicar a metodologia de otimização descrita no Capítulo 4. Para este exemplo, foi gerada a especificação apresentada na Figura 5.7. Os sinais de seleção dos multiplexadores foram suprimidos para melhor visualização.

A Figura 5.8 mostra o resultado da simulação do circuito usando o controlador na versão final. As condições no ambiente de simulação foram as mesmas apresentadas para a versão inicial. Neste caso, a síntese do circuito utilizou 2074 LUTs (*Look-Up Table*) e 520 Flip-Flops. O tempo total de processamento foi de 1,5us. Comparando com a versão inicial, pode-se perceber que houve um acréscimo pouco significativo no número de LUTs e flip-flops, enquanto que a latência total registrou uma redução de aproximadamente 23%.

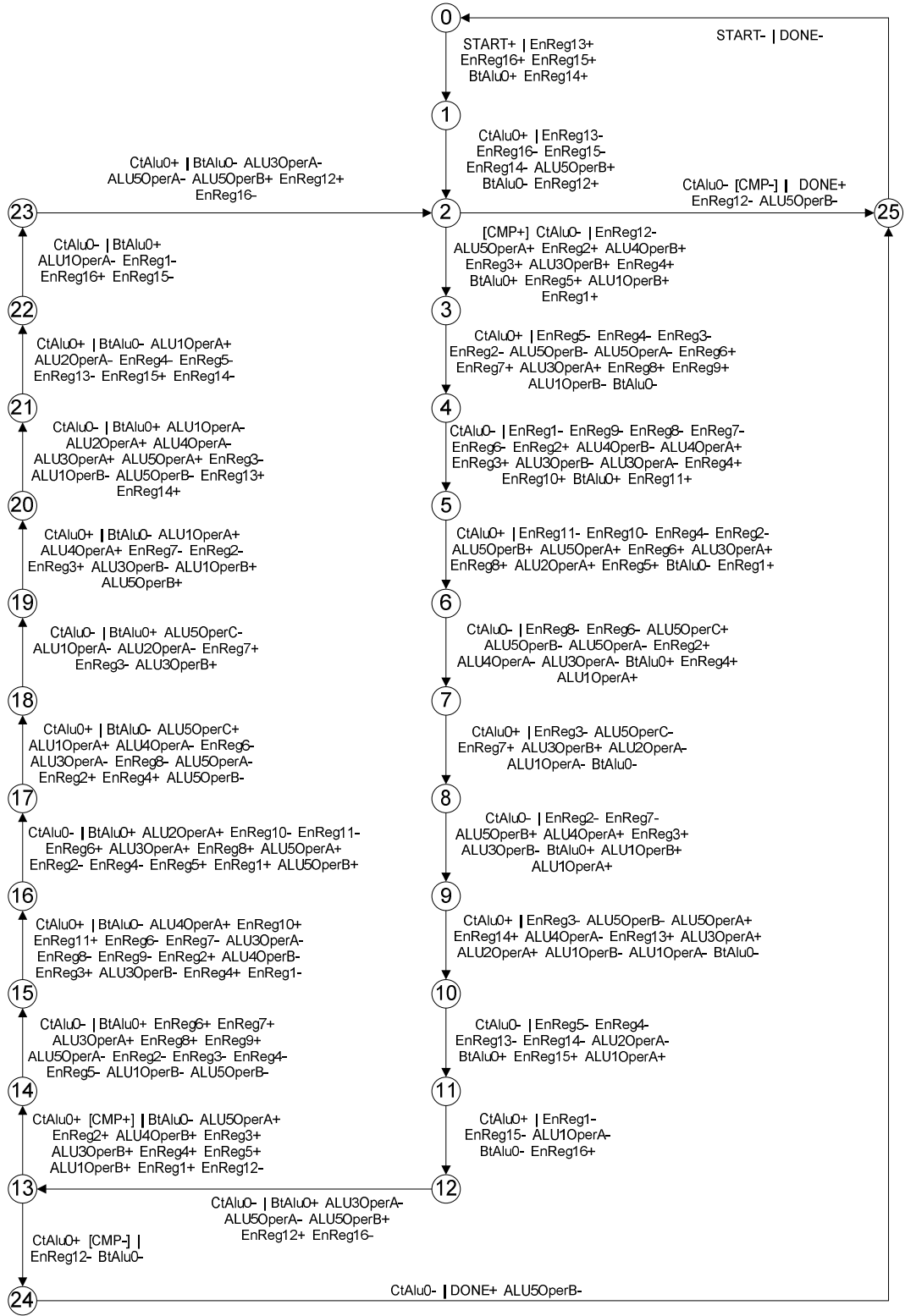


FIGURA 5.7 – XBM final, TEA.

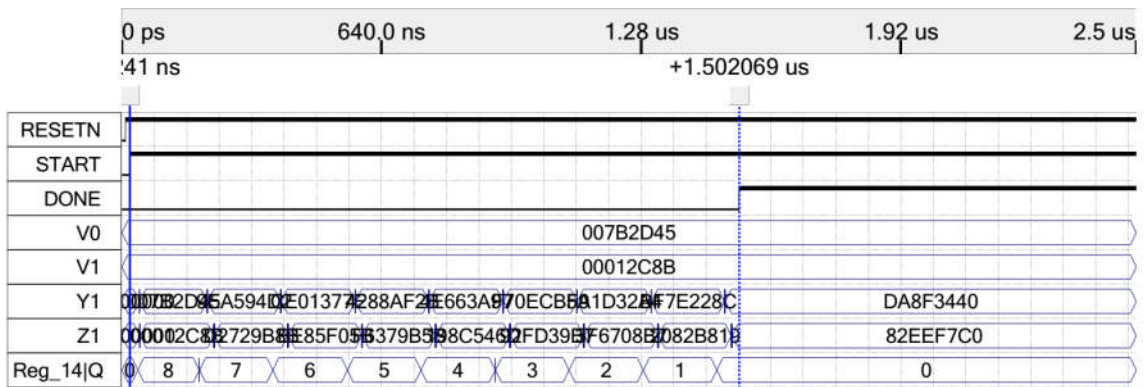


FIGURA 5.8 – Simulação do Circuito – XBM final.

6 Estudo de *Benchmarks*: Resultados

Neste Capítulo, a versão inicial e final das especificações XBM de diversos *benchmarks* são comparadas para demonstrar a eficácia da metodologia. Alguns *benchmarks* não possuem laços, ou não possuem transições mortas, o que não permite a aplicação da otimização do controlador. Para demonstrar o funcionamento e comparar o desempenho dos circuitos gerados, estes foram sintetizados e simulados no software Quartus II versão 9.1 da Altera®, utilizando o dispositivo EP2S15F484C3.

6.1 Descrição dos *Benchmarks*

- DIFFEQ – Solucionador de equação diferencial utilizando método de Euler.
- EWF – *Elliptical Wave Filter*. Filtro digital de 6ª ordem.
- FAT – Fatorial, produto dos inteiros positivos menores ou iguais a n .
- SQR – Raiz quadrada de números perfeitos usando algoritmo de progressão aritmética dos números ímpares.
- GCD – Máximo divisor comum utilizando algoritmo de Euclides
- FIR – Filtro *Finite impulse response*, 4ª ordem.
- IIR – Filtro *Infinite impulse response*, biquadrado.
- TEA-E – Algoritmo de criptografia *Tiny-encryption algorithm*. Cifrador.
- TEA-D – Algoritmo de criptografia *Tiny-encryption algorithm*. Decifrador.
- Divisão – Divisão de números inteiros usando soma e subtração.
- DotProd – Somatória da multiplicação de duas sequências numéricas.
- Fibonacci – Encontra o n -ésimo termo da série de Fibonacci.
- Wavelet – Filtro wavelet de 6 etapas.

6.2 Discussão

Os *benchmarks* citados foram gerados a partir da ferramenta BUDASYN considerando duas variações:

- a) Utilizando o escalonamento MRRT (Minimização de Recursos com Restrição de Tempo), onde o número de unidades funcionais não é restrito e pode aumentar livremente

até obter o tempo desejado. O tempo mínimo é informado pela ferramenta BUDASYN com base no caminho com maior dependência de dados. Neste caso, os tempos de escalonamento escolhidos foram sempre os menores possíveis.

b) Utilizando o escalonamento MTRR (Minimização de Tempo com Restrição de Recursos), no qual é determinado o número de unidades funcionais disponíveis e busca-se o menor tempo de execução. Neste caso, os recursos foram limitados a uma unidade funcional de cada tipo.

A primeira análise realizada diz respeito à Tabela 6.1 e trata da comparação entre os *benchmarks* nas versões iniciais e finais tanto para o escalonamento MRRT quanto para MTRR. Para os casos em que não houve otimização da especificação os resultados foram suprimidos. O objetivo desta comparação é verificar qual o aumento de área do controlador devido à otimização do mesmo, avaliando para tal, o número de produtos e literais de ambas as versões dos controladores.

Quando são observados os casos onde há um maior acréscimo no número de estados (diffeq, TEA-d e TEA-e), em torno de 70% maiores, o número de produtos e literais não aumenta na mesma proporção ficando apenas 53% e 28% maiores respectivamente. Entretanto nos demais casos, onde houve um acréscimo menor no número de estados, em média 42%, o número de produtos e literais foram respectivamente 55% e 29% maiores. A partir desses dados, pode-se perceber que há uma tendência do controlador crescer em área numa proporção menor que o acréscimo no número de estados à medida que a complexidade do controlador aumenta devido à aplicação do algoritmo de otimização do XBM. Este fato ocorre, pois o algoritmo de otimização não insere novos sinais (como pode ser visto nas colunas NSE e NSS), apenas duplica o laço de repetição.

TABELA 6.1 – Resultados obtidos para os controladores na versão inicial e versão final.

Benchmark	MRRT						Dif	Dif	Dif	Dif
	Controlador						NE(%)	NTE(%)	NL(%)	NP(%)
	NE	NTE	NSE	NSS	NL	NP				
Diffeq – V.I.	11	12	4	26	195	52	63,64	66,67	48,21	28,85
Diffeq – V.F.	18	20	4	26	289	67				
Fat – V.I.	6	7	4	8	65	17	33,33	42,86	61,54	41,18
Fat – V.F.	8	10	4	8	105	24				
Tea-e – V.I.	15	16	3	61	512	98	73,33	75,00	57,03	28,57
Tea-e – V.F.	26	28	3	61	804	126				
Tea-d – V.I.	15	16	3	64	531	103	73,33	75,00	55,18	27,18
Tea-d – V.F.	26	28	3	64	824	131				
DotProd – V.I.	8	9	4	13	100	30	50,00	55,56	72,00	36,67
DotProd – V.F.	12	14	4	13	172	41				
Fibonacci – V.I.	9	11	4	17	139	35	33,33	36,36	31,65	8,57
Fibonacci – V.F.	12	15	4	17	183	38				

Benchmark	MTRR						Dif	Dif	Dif	Dif
	Controlador						NE(%)	NTE(%)	NL(%)	NP(%)
	NE	NTE	NSE	NSS	NL	NP				
Fat – V.I.	7	8	4	10	79	21	42,86	50,00	60,76	52,38
Fat – V.F.	10	12	4	10	127	32				
Sqr – V.I.	11	13	4	16	145	34	45,45	46,15	44,83	38,24
Sqr- V.F.	16	19	4	16	210	47				
DotProd – V.I.	8	9	4	13	100	30	50,00	55,56	81,00	40,00
DotProd – V.F.	12	14	4	13	181	42				

Legenda										
NE (número de estados); NTE (número de transições de estados); NSE (número de sinais de entrada);										
NSS (número de sinais de saída); NL (número de literais); NP (número de produtos).										
Dif NE (Diferença percentual entre o número de estados da versão inicial e final);										
Dif NTE (Diferença percentual entre o número de transições de estados da versão inicial e final);										
Dif NL(Diferença percentual entre o número de literais da versão inicial e final);										
Dif NP(Diferença percentual entre o número de produtos da versão inicial e final).										

A segunda análise realizada diz respeito à Tabela 6.2 e apresenta a comparação dos resultados obtidos após a síntese e simulação no Quartus II. Assim como na análise anterior, esta foi obtida tanto para o escalonamento MRRT quanto para MTRR. O objetivo desta comparação é verificar, entre a versão inicial e final, quais as diferenças em: área, potência e latência.

TABELA 6.2 – Resultados obtidos para síntese e simulação, comparando entre versão inicial e final.

Benchmark	MRRT						Dif NLUT(%)	Dif NFF(%)	Dif P(%)	Dif L(%)
	Datapath		Síntese		Simulação					
	NUF	NR	NLUT	NFF	P(mW)	L(ns)				
Diffeq – V.I.	2,1,0	7	264	124	345	221	2,27	0,81	0,00	-14,93
Diffeq – V.F.	2,1,0	7	270	125	345	188				
Fat – V.I.	1,2,0	3	43	49	340	210	-2,33	2,04	0,88	-36,67
Fat – V.F.	1,2,0	3	42	50	343	133				
Tea-e – V.I.	0,5,0	16	2067	519	464	1957	0,34	0,19	-0,43	-23,25
Tea-e – V.F.	0,5,0	16	2074	520	462	1502				
Tea-d – V.I.	0,5,0	16	2048	522	466	1914	0,29	0,19	2,36	-19,23
Tea-d – V.F.	0,5,0	16	2054	523	477	1546				
DotProd – V.I.	1,1,0	4	246	99	344	345	4,47	1,01	0,29	-19,13
DotProd – V.F.	1,1,0	4	257	100	345	279				
Fibonacci – V.I.	0,3,0	6	213	122	341	423	0,47	0,00	-0,29	-25,06
Fibonacci – V.F.	0,3,0	6	214	122	340	317				

Benchmark	MTRR						Dif NLUT(%)	Dif NFF(%)	Dif P(%)	Dif L(%)
	Datapath		Síntese		Simulação					
	NUF	NR	NLUT	NFF	P(mW)	L(ns)				
Fat – V.I.	1,1,0	3	75	51	378	413	2,67	1,96	-9,79	-30,02
Fat – V.F.	1,1,0	3	77	52	341	289				
Sqr – V.I.	0,1,0	5	97	44	334	350	1,03	2,27	1,50	-16,29
Sqr – V.F.	0,1,0	5	98	45	339	293				
DotProd – V.I.	1,1,0	4	246	99	344	345	5,69	1,01	0,29	-17,68
DotProd – V.F.	1,1,0	4	260	100	345	284				

Legenda										
NUF (número de unidades funcionais) para multiplicadores, ULAs e divisores; NR (número de registradores).										
NLUT (número de <i>look up table</i>); NFF (número de flip-flops); P (potência total) e L (latência total).										
Dif NLUT (Diferença percentual entre NLUT da versão inicial e final);										
Dif NFF (Diferença percentual entre NFF da versão inicial e final);										
Dif P(Diferença percentual entre a potência da versão inicial e final);										
Dif L(Diferença percentual entre a latência da versão inicial e final).										

Observa-se que para este conjunto de exemplos, o número de flip-flops e LUTs, que representam a área ocupada pelo circuito, obteve-se um acréscimo pouco expressivo quando comparados entre as versões iniciais e finais, ficando sempre abaixo de 6%. Para a potência total dissipada, o mesmo pode ser observado, sendo que a diferença ficou abaixo de 1%. Analisando as latências totais, observa-se que houve uma melhora significativa no desempenho quando o circuito está utilizando a versão otimizada do controlador, sendo que em média o ganho foi de 23% e no melhor caso (Fat) o ganho alcançou 36,7%. As simulações destes *benchmarks* estão apresentadas no Anexo B.

Obteve-se em todos os casos analisados uma redução na latência dos circuitos para ambos os tipos de escalonamento aplicados, sem comprometer significativamente a área ocupada e a potência consumida. Assim, é possível verificar a eficácia do método de otimização proposto para a especificação do controlador para um grupo significativo de *benchmarks*.

Uma outra análise realizada compara os dois métodos de escalonamento executados, descritos nesta Seção nos itens (a) e (b). São comparados os circuitos gerados sob restrição de tempo com os circuitos gerados sob restrição de recursos. O comportamento esperado para o circuito obtido sob restrição de recursos é que este ocupe menor área, e como penalização ocorra uma redução do desempenho.

Nesta comparação obteve-se, na maioria dos casos, uma redução média de 27% no número de LUTs e uma redução média de 21,5% no número de flip-flops, sendo que a latência final aumentou em média 142%. No entanto, observa-se que em alguns casos (Fatorial, Sqr, Divisão e Fibonacci) o número de LUTs aumentou consideravelmente, sendo em média 70% maior e o número de registradores 7% maior, além da penalização da latência em média 132% maior. Este efeito ocorreu, pois nestes casos além do aumento no tamanho do controlador, é necessário aumentar o número de multiplexadores, tendo em vista que há somente uma unidade funcional de cada tipo. Assim, a redução de área devido à restrição no número de unidades funcionais foi suplantada pelo acréscimo dos multiplexadores. Com isso, pode-se concluir que restringir somente o número de unidades funcionais não garante uma redução de área.

TABELA 6.3 – Resultados obtidos para síntese e simulação, comparando entre escalonamento MRRT e MTRR.

Benchmark	MRRT						MTRR						Dif NLUT(%)	Dif NFF (%)	Dif P (%)	Dif L (%)
	Datapath		Síntese		Simulação		Datapath		Síntese		Simulação					
	NUF	NR	NLUT	NFF	P(mW)	L(ns)	NUF	NR	NLUT	NFF	P(mW)	L(ns)				
Diffeq – V.I.	2,1,0	7	264	124	345	221	1,1,0	6	231	104	344	253	-12,5	-16,1	-0,3	14,5
Diffeq – V.F.	2,1,0	7	270	125	345	188	-	-	-	-	-	-	-	-	-	-
Ewf	12,7,0	37	897	697	495	76	1,1,0	26	519	465	395	520	-42,1	-33,3	-20,2	584,2
Fat – V.I.	1,2,0	3	43	49	340	210	1,1,0	3	75	51	378	413	74,4	4,1	11,2	96,7
Fat – V.F.	1,2,0	3	42	50	343	133	1,1,0	3	77	52	341	289	83,3	4,0	-0,6	117,3
Sqr – V.I.	0,4,0	5	58	38	334	117	0,1,0	5	97	44	334	350	67,2	15,8	0,0	199,1
Sqr- V.F.	-	-	-	-	-	-	0,1,0	5	98	45	339	293	-	-	-	-
Gcd	0,2,0	4	76	31	333	143	0,1,0	4	71	31	333	156	-6,6	0,0	0,0	9,1
FIR	3,1,0	4	130	79	362	38	1,1,0	3	122	62	364	50	-6,2	-21,5	0,6	31,6
IIR	3,1,0	4	144	79	354	46	1,1,0	3	128	62	353	65	-11,1	-21,5	-0,3	41,3
Tea-e – V.I.	0,5,0	16	2067	519	464	1957	0,1,0	14	1003	426	442	5660	-51,5	-17,9	-4,7	189,2
Tea-e – V.F.	0,5,0	16	2074	520	462	1502	-	-	-	-	-	-	-	-	-	-
Tea-d – V.I.	0,5,0	16	2048	522	466	1914	0,1,0	14	1012	426	433	5978	-50,6	-18,4	-7,1	212,3
Tea-d – V.F.	0,5,0	16	2054	523	477	1546	-	-	-	-	-	-	-	-	-	-
Divisão	0,3,0	3	73	41	334	161	0,1,0	3	131	45	339	397	79,5	9,8	1,5	146,6
DotProd–V.I.	1,1,0	4	246	99	344	345	1,1,0	4	246	99	344	345	0,0	0,0	0,0	0,0
DotProd–V.F.	1,1,0	4	257	100	345	279	1,1,0	4	260	100	345	284	1,2	0,0	0,0	1,8
Fibonacci–V.I.	0,3,0	6	213	122	341	423	0,1,0	6	311	123	347	855	46,0	0,8	1,8	102,1
Fibonacci–V.F.	0,3,0	6	214	122	340	317	-	-	-	-	-	-	-	-	-	-
Wavelet	6,4,0	12	1216	430	369	93	1,1,0	4	456	149	375	277	-62,5	-65,3	1,6	197,8

Legenda

NUF (número de unidades funcionais) para multiplicadores, ULAs e divisores; NR (número de registradores).
NLUT (número de *look up table*); NFF (número de flip-flops); P (potência total) e L (latência total).
Dif NLUT (Diferença percentual entre número de LUTs do escalonamento MRRT e MTRR);
Dif NFF (Diferença percentual entre o número de flip-flops do escalonamento MRRT e MTRR);
Dif P (Diferença percentual entre a potência do escalonamento MRRT e MTRR);
Dif L (Diferença percentual entre a latência do escalonamento MRRT e MTRR);

7 Conclusão

As ferramentas para síntese de alto nível voltada para circuitos assíncronos otimizados ainda são, na grande maioria, resultados de trabalhos acadêmicos e não atingiram maturidade suficiente para o uso comercial. As ferramentas mais avançadas neste campo são as baseadas em tradução direta, que em geral são deficientes em otimização.

Nesta dissertação, foi apresentada uma ferramenta para a síntese de alto nível de circuitos assíncronos, capaz de gerar circuitos de menor complexidade, porém com técnicas de otimização em diversas etapas de síntese, tais como: escalonamento, assinalamentos e especificação do controlador, buscando contribuir no desenvolvimento dos métodos aplicados aos circuitos assíncronos.

O método de otimização aplicado ao controlador mostrou-se bastante eficaz, apresentando um ganho em desempenho de aproximadamente 30% (quando comparado com o mesmo circuito sem a otimização), sem prejudicar significativamente a área e a potência dos circuitos, quando implementados em FPGAs.

7.1 Trabalhos Futuros

Como trabalhos futuros, os seguintes tópicos podem ser citados:

- Substituir a linguagem de entrada por uma linguagem amplamente utilizada e que permita a descrição comportamental, tal como VHDL.
- Permitir o uso de novas bibliotecas de componentes, permitindo que o usuário possa testar diferentes arquiteturas em nível RTL.
- Expandir os métodos de escalonamento e assinalamentos, visando redução de área e potência.
- Comparar os resultados com o equivalente síncrono.
- Definir metodologia para gerar os elementos de atraso automaticamente.

Referências

AMERICAN NATIONAL STANDARDS INSTITUTE. **ANSI X3.131**. Small Computer Systems Interface-1 (SCSI-1), 1986.

ANDRIKOS, Nikos; LAVAGNO, Luciano. Optimal and heuristic scheduling algorithms for asynchronous high-level synthesis. In: INTERNATIONAL SYMPOSIUM ON ASYNCHRONOUS CIRCUITS AND SYSTEMS, Ithaca, 2011. **Proceedings...** New York: IEEE, 2011. p. 13-21.

BACHMAN, Brandon M. **Architectural-level synthesis of asynchronous systems**. 1998. 89 p. Thesis (M. Sc. in Electrical Engineer), The University of Utah, Utah.

BACHMAN, Brandon; ZHENG, Hao; MEYERS, Chris J. Architectural synthesis of timed asynchronous systems. In: INTERNATIONAL CONFERENCE ON COMPUTER DESIGN, 1999, Austin. **Proceedings...**, Los Alamitos: IEEE – Computer Society, 1999. p. 354 – 363.

BADIA, Rosa M.; CORTADELLA, Jordi. High-level synthesis of asynchronous systems: scheduling and process synchronization. In: EUROPEAN CONFERENCE ON DESIGN AUTOMATION WITH THE EUROPEAN EVENT IN ASIV DESIGN, 4, 1993, Paris. **Proceedings...**, Paris: IEEE – Computer Society, 1993. p. 70 – 74.

BARDSLEY, Andrew. **Balsa**: an asynchronous circuit synthesis system. 1998. 162 p. Thesis (M. Phil in the Faculty of Science & Engineering) - University of Manchester, Manchester.

BEEREL, Peter A.; OZDAG Recep O.; FERRETTI Marcos. **A designer's guide to asynchronous VLSI**. 2010. Cambridge University Press, 337p.

BERKEL, Kees Van. **Handshake circuits: an asynchronous architecture for VLSI programming**. 1993. Cambridge University Press, 227p.

BERKELAAR, Michel; EIKLAND, Kjell; NOTEBAERT, Peter. **lp_solve**: version 5.1.0.0. [S.l.], 2004. Disponível em: <<http://sourceforge.net/projects/lpsolve/files/latest/download>>.

BRUNVAND, Erik. **Translating concurrent communicating programs into asynchronous circuits**. 1991. Thesis (PhD in Computer Science) - Carnegie Mellon University. Pittsburgh, PA.

CHELCEA, Tiberiu et al. A burst-mode oriented back-end for the balsa synthesis system. In: DESIGN AUTOMATION AND TEST IN EUROPE CONFERENCE AND EXHIBITION, 2002. **Proceedings...**, Paris: IEEE – Computer Society, 2002. p. 330 – 337.

CHELCEA, Tiberiu et al. **Balsa-CUBE**: an optimizing back-end for the Balsa synthesis system. 14TH UK ASYNC. FORUM. 2003. Disponível em <<http://async.org.uk/ukasyncforum14/forum14-papers/forum14-chelcea.pdf>>. Acesso em: 10 Jan 2015.

CHAPIRO, D. M. **Globally-asynchronous locally-synchronous systems**, 1984. PhD thesis, Stanford University.

CHU, Tam-Anh. **Synthesis of self-timed VLSI circuits from graph-theoretic specifications**. 1987. PhD thesis, Massachusetts Institute of Technology.

CORTADELLA, Jordi; KONDRATYEV, Alex; LAVAGNO, Luciano; SOTIRIOU, Christos P. Desynchronization: synthesis of asynchronous circuits from synchronous specifications. **Computer-Aided Design of Integrated Circuits and Systems**, v. 25, n. 10, p. 1904-1921, Oct. 2006.

CURTINHAS *et al.* SICARELO: A tool for synthesis of locally-clocked extended burst-mode asynchronous controllers. In: 5th LATIN AMERICAN SYMPOSIUM ON CIRCUITS AND SYSTEMS, Santiago, 2014. **Proceedings...**, IEEE. p. 1-4.

DAVIDSON, S. *et al.* Some experiments in local microcode compaction for horizontal machines. **IEEE Transaction on Computers**, v. C30, p. 460-477, 1981.

DAVIS, Al; NOWICK, Steven M. **An Introduction to asynchronous circuit design**. Utah: The University of Utah, 1997. 58 p. (UUCS-97-013).

DAVIS, Al; STEVENS, Ken. The post office experience: designing a large asynchronous chip. In: TWENTY-SIXTH HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES, 1993. **Proceedings...**, IEEE – Computer Society, 1993. v.1 p. 409 – 418.

DOIG, A. G.; LAND, Bya H. An automatic method for solving discrete programming problems. **Econometrica**. 1960. p.497 – 520.

DUARTE, Luis Tarazona. **Performance-oriented syntax-directed synthesis of asynchronous circuits**. 2010. 269 p. Thesis (PhD in Engineering and Physical Sciences) - Manchester University, Manchester.

ELLIOTT, John P. **Understanding behavioral synthesis: a practical guide to high-level design**. New York: Kluwer Academic Publishers, 1999. 319p.

ENGEL, Frank; KUZ, Ihor; PETTERS, Stefan M.; RUOCCO, Sergio. Operating systems on SoCs: a good idea? In: EMBEDDED REAL-TIME SYSTEMS IMPLEMENTATION, 2004. **Proceedings...**, National ICT Australia, 2004.

FERRINGER, M.; FUCHS, G.; STEININGER, A.; KEMPF, G. VLSI Implementation of a fault-tolerant distributed clock generation. In: INTERNATIONAL SYMPOSIUM ON DEFECT AND FAULT TOLERANCE IN VLSI SYSTEMS, 2006. **Proceedings...**, Arlington: IEEE – Computer Society, 2006. p. 563-571.

FUHRER, Robert M. **Sequential optimization of asynchronous and synchronous finite-state machine: algorithms and tools**. 1999. 298 p. Thesis (PhD in the Graduate School of Arts and Sciences) – Columbia University, New York.

GARCIA, K. *et al.* Synthesis of locally-clocked asynchronous systems with bundled-data implementation on FPGAs. In: IEEE IX SOUTHERN CONFERENCE ON PROGRAMMABLE LOGIC (SPL), 2014, Buenos Aires. **Proceedings...**, Piscataway: IEEE, 2014. p. 101-107.

HANSEN, John B. **A behavioral design flow for synthesis and optimization of asynchronous systems**. 2012. 219 p. Thesis (PhD in the Department of Computer Science) - University of North Carolina, Chapel Hill.

HANSEN, John; SINGH, Montek. A fast branch-and-bound approach to high-level synthesis of asynchronous systems. In: IEEE SYMPOSIUM ON ASYNCHRONOUS CIRCUITS AND SYSTEMS, 2010, Grenoble. **Proceedings...**, Los Alamitos: IEEE – Computer Society, p.107 – 116.

HASHIMOTO, A.; STEVENS, J. Wire routing by optimizing channel assignment within large apertures. In: DESIGN AUTOMATION WORKSHOP, 8, 1971. **Proceedings...** New York: IEEE Computer Society Press, 1971. p. 155-169.

JACOBSON, Hans. **A case study of a framework called ACK**. 1996. 137 p. Thesis (M. Sc. in Computer Engineering) – Luleå University of Technology, Sweden.

KIM, Han Bin. **High-level synthesis and implementation of built-in self-testable data path intensive circuit**. 1999. 173 p. Thesis (PhD in Electrical Engineering) - Faculty of the Virginia Polytechnic Institute and State University, Blacksburg.

KOMATSU, Yoshiya; HARIYAMA, Masonori; KAMEYAMA, Michitaka. Area-efficient design of asynchronous circuits based on balsa framework for synchronous FPGAs. In: THE INTERNATIONAL CONFERENCE ON ENGINEERING OF RECONFIGURABLE SYSTEMS & ALGORITHMS, Bratislava, 2012. **Proceedings...** Las Vegas: CSREA Press, 2012. p.113 – 118.

KRSTIC, M. *et al.* Globally asynchronous, locally synchronous circuits: overview and outlook. **Design & Test of Computers**, v. 24, n. 5, p. 430-441, Oct. 2007.

MAHDI, Gaidaa Saeed. A modification of TEA block cipher algorithm for data security (MTEA). 2011. **Engineering & Technology Journal**, v. 29, n. 5. University of Technology, Iraq Academic Scientific Journals.

MARTIN, Alain J. The limitations to delay-insensitivity in asynchronous circuits. In: SIXTH MIT CONFERENCE ON ADVANCED RESEARCH IN VLSI, 1990. **Proceedings...** MIT Press Cambridge, 1990. p263-278.

MICHELI, Giovanni De. **Synthesis and optimization of digital circuits**. 1st ed., New York: McGraw-Hill, 1994. 579p.

MUTTERSBAACH, Jens; **Globally-asynchronous locally-synchronous architectures for VLSI systems**. 2001. 121 p. PhD Thesis - Swiss Federal Institute of Technology Zurich, Zurich.

NIELSEN, Sune Fallgaard. **Behavioral synthesis of asynchronous circuits**. 2004. 122 p. Thesis (PhD in Informatics and Mathematical Modeling) - University of Denmark, Denmark.
NOWICK, Steven Mark. **Automatic synthesis of burst-mode asynchronous controllers**. Stanford, CA: Stanford University, 1995. (CSL-TR-95-686).

NOWICK, Steven Mark; SINGH, Montek. Asynchronous design – part 2: systems and methodologies. **IEEE Design & Test**, v. 32, p. 19-28, Jun. 2015.

OLIVEIRA, Duarte Lopes. **Miriã**: uma ferramenta para síntese de controladores assíncronos multi-rajada. 2004. 218 f. Tese (Doutorado em Microeletrônica) – Departamento de Engenharia de Sistemas Eletrônicos, Escola Politécnica da Universidade de São Paulo, São Paulo.

OLIVEIRA, Duarte Lopes et al. Design of locally-clocked asynchronous finite state machines using synchronous CAD tools. In: IEEE FOURTH LATIN AMERICAN SYMPOSIUM ON CIRCUITS AND SYSTEMS (LASCAS), Cuzco, 2013. **Proceedings...** IEEE, 2013. p. 1 – 4.

PAULIN, Pierre G.; KNIGHT, John P. Force-directed scheduling for the behavioral synthesis of ASICs. In: **IEEE – Transactions on Computer-Aided Design of Integrated circuits and systems**, v. 8, n. 6, p. 661-679, 1989.

SAITO, Hiroshi et al. Ilp-based scheduling for asynchronous circuits in bundled-data implementation. In: THE SIXTH INTERNATIONAL CONFERENCE ON COMPUTER AND INFORMATION TECHNOLOGY, Seoul, 2006. **Proceedings...** Los Alamos: IEEE – Computer Society, 2006. p. 172.

SAITO, Hiroshi et al. Scheduling methods for asynchronous circuits with bundled-data implementations based on the approximation of start times. **IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences**, v. E90-A, n. 12, p. 2790-2799, Dec. 2007

SINGH, M.; NOWICK, S.M. The design of high-performance dynamic asynchronous pipelines: lookahead style. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 15, n. 11, p. 1270-1283, Nov 2007.

SPARSO, Jens; FURBER Steve. **Principles of asynchronous circuit design**: A systems perspective. Dordrecht: Kluwer Academic Publishers, 2001. 337p.

SUTHERLAND, Ivan E. Micropipelines. **Communications of the ACM**, v. 32, n. 6, p. 720–738, Jun 1989.

WHEELER, D. J.; NEEDHAM, R. M. TEA, a tiny encryption algorithm. In: FAST SOFTWARE ENCRYPTION: SECOND INTERNATIONAL WORKSHOP, Leuven, 1994. **Proceedings...** Berlin: Springer, 1994. p. 363-366.

YUN, Kenneth Y.; DILL, David L. Automatic synthesis of extended burst-mode circuits: part I (specification and hazard-free implementations). **Computer-Aided Design of Integrated Circuits and Systems**, v. 18, p. 101-117, Feb. 1999.

ZAFAR, Youzaf. **FPGA-Compliant micropipeline based asynchronous systems**. 2005. 122 p. Thesis (PhD in electronic Engineering) - Mohammad Ali Jinnah University, Karachi.

Apêndice A – Uso da Ferramenta, passo a passo

A.1 Arquivos de Entrada

Conforme descrito no Capítulo 4, são necessários três arquivos para iniciar o processo de síntese a partir desta ferramenta. Um contendo a descrição do algoritmo, podendo ser escolhido um nome qualquer para este arquivo. Outro arquivo deve ser nomeado como Biblioteca.dat e deve conter os dados referentes aos tempos de latência das unidades funcionais, que deve ser um número decimal inteiro e maior que zero. Por fim, deve-se criar um arquivo com o nome Recurso.dat, onde deve ser adicionado o número de unidades funcionais disponíveis, caso o tipo de escalonamento desejado seja MTRR (Minimização de Tempo com Restrição de Recursos).

É recomendado que uma nova pasta (sub diretório) seja criada para cada circuito sintetizado, pois os dados de saída podem sobrescrever arquivos que tenham sido gerados anteriormente.

A.2 Informações do GFD

A Figura A.1 mostra um exemplo da tela inicial, onde são apresentadas as informações do circuito descrito no arquivo de entrada, que são:

- Informações do GFD: Este item apresenta a enumeração dos nós e a operação a ser executada por cada um.
- Grafo de Fluxo de Dados não Escalonado: Esta é uma representação textual para o GFD, onde o grafo inicia em “*source*” e termina em “*sink*”, a representação gráfica para este GFD pode ser visualizado na Figura A.2.
- Tempo de Execução: De acordo com os tempos de latência definidos individualmente para cada UF, através do arquivo Biblioteca.dat, é calculado os tempos mínimo e máximo possíveis para o GFD a ser escalonado. Esta informação é útil quando utilizado o escalonamento MRRT, onde deve-se definir o tempo desejado.
- Mobilidade dos Nós: Apresenta quais nós tem mobilidade, considerando o DFG não escalonado.
- Opções de Escalonamento: Este item requer uma ação, onde deve-se escolher a opção de escalonamento desejada.

```

=====
||                               Instituto Tecnológico de Aeronautica                               ||
|| Klederman Garcia                                                       ||
|| Prof. Dr. Duarte Lopes de Oliveira                                     ||
|| Prof. Dr. Roberto d'Amore                                             ||
=====

===== Informações Do GFD =====
Nó: (0) => Y1 + (1);
Nó: (1) => U1 * DX;
Nó: (2) => (3) - (7);
Nó: (3) => U1 - (4);
Nó: (4) => (5) * DX;
Nó: (5) => (6) * U1;
Nó: (6) => 3 * X1;
Nó: (7) => (8) * DX;
Nó: (8) => 3 * Y1;
Nó: (9) => X1 + DX;
Nó: (10) => X1 < A;

===== Grafo de Fluxo de Dados - Não Escalonado =====
(Source) => (10) -> (Sink) ;
(Source) => (9) -> (Sink) ;
(Source) => (8) => (7) => (2) -> (Sink) ;
(Source) => (6) => (5) => (4) => (3) => (2) ;
(Source) => (1) => (0) -> (Sink) ;

===== Tempo de Execução =====
Shortest Time to Finish: 64
Worst Case: 136

===== Mobilidade dos Nós =====
Nó (10): 0 ~ 56
Nó (9): 0 ~ 56
Nó (8): 0 ~ 24
Nó (7): 16 ~ 40
Nó (1): 0 ~ 40
Nó (0): 16 ~ 56

=====

Opções de Escalonamento:
0) Sair
1) Minimização do Tempo com Restrição de Recursos.
2) Minimização dos Recursos com Restrição de Tempo.
  2
Escolha o tempo desejado: 64█

```

FIGURA A.1 – Opções de Escalonamento

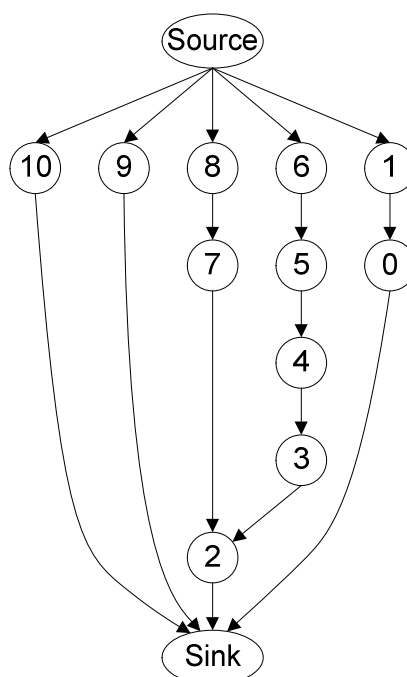


FIGURA A.2 – Representação Gráfica do GFD não Escalonado.

A.3 Informações do Escalonamento e *Datapath*

Após escolher o tipo de escalonamento desejado, é apresentado o resultado do escalonamento, conforme demonstrado na Figura A.3, usando a notação textual apresentada no Capítulo 4. São apresentados também os assinalamentos de unidades funcionais e registradores, além do *datapath* gerado. Essas informações são armazenadas em arquivos, que são gerados conforme os dados são disponibilizados pela ferramenta.

A.4 Especificação do Controlador

O último passo necessário é escolher entre a especificação do XMB inicial ou final, apresentado na Figura A.4. Em alguns casos, a versão inicial pode ser igual à versão final, portanto pode-se escolher qualquer das opções.

Após a finalização do processo de síntese, os arquivos com a descrição VHDL do circuito podem ser utilizados. A entidade com a descrição final do circuito é gerada em um arquivo com o mesmo nome do arquivo de entrada acrescido dos caracteres “_F” e com extensão VHD.

```

===== DFG ESCALONADO =====
Nó: (-1) source No+, StarTime: 0;
Nó: (-1) source No-, StarTime: 0;
Nó: (6) * No+, StarTime: 0;
Nó: (8) * No+, StarTime: 0;
Nó: (10) < No+, StarTime: 0;
Nó: (10) < No-, StarTime: 8;
Nó: (9) + No+, StarTime: 8;
Nó: (6) * No-, StarTime: 16;
Nó: (8) * No-, StarTime: 16;
Nó: (9) + No-, StarTime: 16;
Nó: (5) * No+, StarTime: 16;
Nó: (1) * No+, StarTime: 16;
Nó: (5) * No-, StarTime: 32;
Nó: (1) * No-, StarTime: 32;
Nó: (4) * No+, StarTime: 32;
Nó: (7) * No+, StarTime: 32;
Nó: (0) + No+, StarTime: 32;
Nó: (0) + No-, StarTime: 40;
Nó: (4) * No-, StarTime: 48;
Nó: (7) * No-, StarTime: 48;
Nó: (3) - No+, StarTime: 48;
Nó: (3) - No-, StarTime: 56;
Nó: (2) - No+, StarTime: 56;
Nó: (2) - No-, StarTime: 64;
Nó: (-2) sink No+, StarTime: 64;

===== ASSINALAMENTO DE UNIDADES FUNCIONAIS =====
MUL_1{6, 5, 4}
MUL_2{8, 1, 7}
ALU_1{10, 9, 0, 3, 2}

===== ASSINALAMENTO DE REGISTRADORES =====
Reg_1{10}
Reg_2{6, 5, 4}
Reg_3{8, 7}
Reg_4{9}
Reg_5{1}
Reg_6{0}
Reg_7{3, 2}

===== DATAPATH - REPRESENTAÇÃO INTERMEDIÁRIA =====
0) MUL_1 -> Reg_2 ;
1) MUXReg_7 -> Reg_7 ;
2) ALU_1 -> MUXReg_7 - A;
3) MUXReg_6 -> Reg_6 ;
4) ALU_1 -> MUXReg_6 - A;
5) MUL_2 -> Reg_5 ;
6) MUXReg_4 -> Reg_4 ;
7) ALU_1 -> MUXReg_4 - A;
8) ALU_1 -> Reg_1 ;
9) MUL_2 -> Reg_3 ;
10) MUXRALU_1 -> ALU_1 ;
11) MUXLALU_1 -> ALU_1 ;
12) MUXRMUL_2 -> MUL_2 ;
13) MUXLMUL_2 -> MUL_2 ;
14) MUXRMUL_1 -> MUL_1 ;
15) MUXLMUL_1 -> MUL_1 ;
16) Reg_3 -> MUXRALU_1 - A;
17) Reg_7 -> MUXLALU_1 - A;
18) Reg_2 -> MUXRALU_1 - B;
19) Reg_5 -> MUXRALU_1 - C;
20) Reg_6 -> MUXLALU_1 - B;
21) DX -> MUXRMUL_2 - A;
22) Reg_3 -> MUXLMUL_2 - A;
23) DX -> MUXRMUL_1 - A;
24) Reg_2 -> MUXLMUL_1 - A;
25) Reg_7 -> MUXLMUL_2 - B;
26) Reg_7 -> MUXRMUL_1 - B;
27) DX -> MUXRALU_1 - D;
28) Reg_4 -> MUXLALU_1 - C;
29) A -> MUXRALU_1 - E;
30) Reg_6 -> MUXRMUL_2 - B;
31) 3 -> MUXLMUL_2 - C;
32) Reg_4 -> MUXRMUL_1 - C;
33) 3 -> MUXLMUL_1 - B;
34) Reg_1 -> COMP ;
35) Reg_6 -> SAIDA ;
36) X -> MUXReg_4 - B;
37) Y -> MUXReg_6 - B;
38) U -> MUXReg_7 - B;

```

FIGURA A.3 – Resultado do Escalonamento; Assinalamentos; *Datapath*.

```

=====GERA XEM VERSAO INICIAL=====
Insera Transição Morta 8 9
-----
Runtime
-----
0
0 1 START+ | En_Reg_7+ Sel_MUXReg_7+ En_Reg_6+ Sel_MUXReg_6+ BtAlu0+ En_Reg_4+ Sel_MUXReg_4+
1 2 CTA1u0+ | En_Reg_7- En_Reg_6- En_Reg_4- ALU_10perB+ BtAlu0- Sel_MUXRALU_1C+ Sel_MUXReg_4- Sel_MUXLALU_1B+ En_Reg_1+
2 3 [COMP+] CTA1u0- | En_Reg_1- BtAlu0+ Sel_MUXLMUL_2B+ Sel_MUXRMUL_2+ Sel_MUXReg_6- En_Reg_3+ BtMu10+ Sel_MUXLMUL_1+ Sel_MUXRMUL_1B+ En_Reg_2+
3 4 CTA1u0+ | ALU_10perB- BtAlu0- Sel_MUXRALU_1A+ Sel_MUXRALU_1B+ Sel_MUXRALU_1C- En_Reg_4+
4 5 CTA1u0- CTA1u0+ | En_Reg_3- En_Reg_4- Sel_MUXLMUL_2A+ Sel_MUXLMUL_2B- Sel_MUXRMUL_2- En_Reg_5+ BtMu10- Sel_MUXRMUL_1A+ Sel_MUXRMUL_1B- Sel_MUXLMUL_1- Sel_MUXReg_7-
5 6 CTA1u0- | En_Reg_5- BtAlu0+ Sel_MUXLALU_1A+ Sel_MUXLALU_1B- Sel_MUXRALU_1A- En_Reg_6+ Sel_MUXLMUL_2A- En_Reg_3+ BtMu10+ Sel_MUXRMUL_1A-
6 7 CTA1u0+ CTA1u0+ | En_Reg_2- En_Reg_3- En_Reg_6- ALU_10perA+ BtAlu0- Sel_MUXRALU_1A+ Sel_MUXRALU_1B- Sel_MUXLALU_1A- En_Reg_7+
7 8 CTA1u0- | BtAlu0+ Sel_MUXRALU_1A-
8 9 CTA1u0+ | En_Reg_7- BtMu10-
9 2 CTA1u0- | BtAlu0- ALU_10perA- Sel_MUXLALU_1B+ Sel_MUXReg_7+ ALU_10perB+ Sel_MUXRALU_1C+ Sel_MUXReg_6+ En_Reg_1+
2 10 CTA1u0- [COMP-] | Sel_MUXReg_7- Sel_MUXReg_6- DONE+ En_Reg_1- Sel_MUXLALU_1B- Sel_MUXRALU_1C- ALU_10perB-
10 0 START- | DONE-

=====GERA XEM VERSAO FINAL=====
Insera Transição Morta 9 16
0 1 START+ | En_Reg_7+ Sel_MUXReg_7+ En_Reg_6+ Sel_MUXReg_6+ BtAlu0+ En_Reg_4+ Sel_MUXReg_4+
1 2 CTA1u0+ | En_Reg_7- En_Reg_6- En_Reg_4- ALU_10perB+ BtAlu0- Sel_MUXRALU_1C+ Sel_MUXReg_4- Sel_MUXLALU_1B+ En_Reg_1+
2 3 [COMP+] CTA1u0- | En_Reg_1- BtAlu0+ Sel_MUXLMUL_2B+ Sel_MUXRMUL_2+ Sel_MUXReg_6- En_Reg_3+ BtMu10+ Sel_MUXLMUL_1+ Sel_MUXRMUL_1B+ En_Reg_2+
3 4 CTA1u0+ | ALU_10perB- BtAlu0- Sel_MUXRALU_1A+ Sel_MUXRALU_1B+ Sel_MUXRALU_1C- En_Reg_4+
4 5 CTA1u0- CTA1u0+ | En_Reg_3- En_Reg_4- Sel_MUXLMUL_2A+ Sel_MUXLMUL_2B- Sel_MUXRMUL_2- En_Reg_5+ BtMu10- Sel_MUXRMUL_1A+ Sel_MUXRMUL_1B- Sel_MUXLMUL_1- Sel_MUXReg_7-
5 6 CTA1u0- | En_Reg_5- BtAlu0+ Sel_MUXLALU_1A+ Sel_MUXLALU_1B- Sel_MUXRALU_1A- En_Reg_6+ Sel_MUXLMUL_2A- En_Reg_3+ BtMu10+ Sel_MUXRMUL_1A-
6 7 CTA1u0+ CTA1u0+ | En_Reg_2- En_Reg_3- En_Reg_6- ALU_10perA+ BtAlu0- Sel_MUXRALU_1A+ Sel_MUXRALU_1B- Sel_MUXLALU_1A- En_Reg_7+
7 8 CTA1u0- | BtAlu0+ Sel_MUXRALU_1A-
8 9 CTA1u0+ | BtAlu0- ALU_10perA- ALU_10perB+ Sel_MUXRALU_1C+ Sel_MUXLALU_1B+ En_Reg_1+ En_Reg_7- Sel_MUXReg_7+ Sel_MUXReg_6+
9 10 CTA1u0- [COMP+] | BtMu10- BtAlu0+ Sel_MUXLMUL_2B+ Sel_MUXRMUL_2+ En_Reg_3+ Sel_MUXLMUL_1+ Sel_MUXRMUL_1B+ En_Reg_2+ En_Reg_1- Sel_MUXReg_6-
10 11 CTA1u0+ | BtAlu0- Sel_MUXRALU_1A+ Sel_MUXRALU_1B+ ALU_10perB- Sel_MUXRALU_1C- En_Reg_4+
11 12 CTA1u0- CTA1u0- | BtMu10+ Sel_MUXLMUL_2A+ En_Reg_5+ Sel_MUXRMUL_1A+ Sel_MUXLMUL_2B- Sel_MUXRMUL_2- En_Reg_3- Sel_MUXLMUL_1- Sel_MUXRMUL_1B- Sel_MUXReg_7- En_Reg_4-
12 13 CTA1u0+ | BtMu10- BtAlu0+ Sel_MUXLALU_1A+ Sel_MUXLMUL_2A- En_Reg_5- Sel_MUXRMUL_1A- Sel_MUXRALU_1A- En_Reg_3+ Sel_MUXLALU_1B- En_Reg_6+
13 14 CTA1u0+ CTA1u0- | BtAlu0- ALU_10perA+ Sel_MUXLALU_1A- Sel_MUXRALU_1A+ Sel_MUXRALU_1B- En_Reg_3- En_Reg_2- En_Reg_7+ En_Reg_6-
14 15 CTA1u0- | BtAlu0+ Sel_MUXRALU_1A-
15 2 CTA1u0+ | BtAlu0- ALU_10perA- ALU_10perB+ Sel_MUXRALU_1C+ Sel_MUXLALU_1B+ En_Reg_1+ En_Reg_7- Sel_MUXReg_7+ Sel_MUXReg_6+
9 16 CTA1u0- [COMP-] | En_Reg_1- BtMu10-
16 17 CTA1u0- | DONE+ ALU_10perB- Sel_MUXRALU_1C- Sel_MUXLALU_1B- Sel_MUXReg_7- Sel_MUXReg_6-
2 17 CTA1u0- [COMP-] | Sel_MUXReg_7- Sel_MUXReg_6- DONE+ En_Reg_1- Sel_MUXLALU_1B- Sel_MUXRALU_1C- ALU_10perB-
17 0 START- | DONE-

=====
Especificação XEM criada. Deseja usar a versão Inicial ou Final?
1) Versão Inicial (Com Transições Mortas).
2) Versão Final (Sem Transições Mortas).

```

FIGURA A.4 – Escolha da Especificação do Controlador.

A.5 Simulação no Quartus II

Após a geração de um novo projeto, incluir no mesmo diretório os arquivos de extensão VHD obtidos pela ferramenta e adicionar os elementos de atraso, conforme exemplificado na Figura A.5.

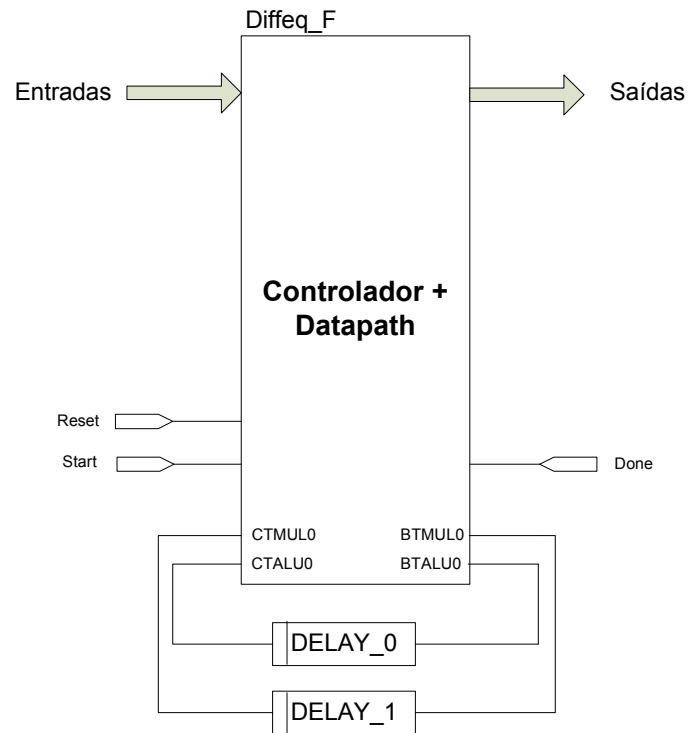


FIGURA A.5 – Adição dos Elementos de Atraso.

Apêndice B – Simulações

B.1 Simulações Utilizando Escalonamento MRRT

Os benchmarks simulados nesta seção utilizam o escalonamento MRRT, e correspondem aos dados apresentados no Capítulo 6.

1) Diffeq – Versão Inicial:

```

in x      (16)
in dx    (16)
in y     (16)
in u     (16)
in a     (16)
var x1   (16)
var y1   (16)
var u1   (16)
out comp (1)
out saida (16)

main:
x1<:x
y1<:y
u1<:u

while [comp <: x1<a]
  x1 <: x1+dx
  u1 <: u1-(3*x1*u1*dx)-(3*y1*dx)
  y1 <: y1+u1*dx
  saida <: y1
endwhile
end

```

FIGURA B.1 – Arquivo de Entrada. Diffeq V.I. MRRT

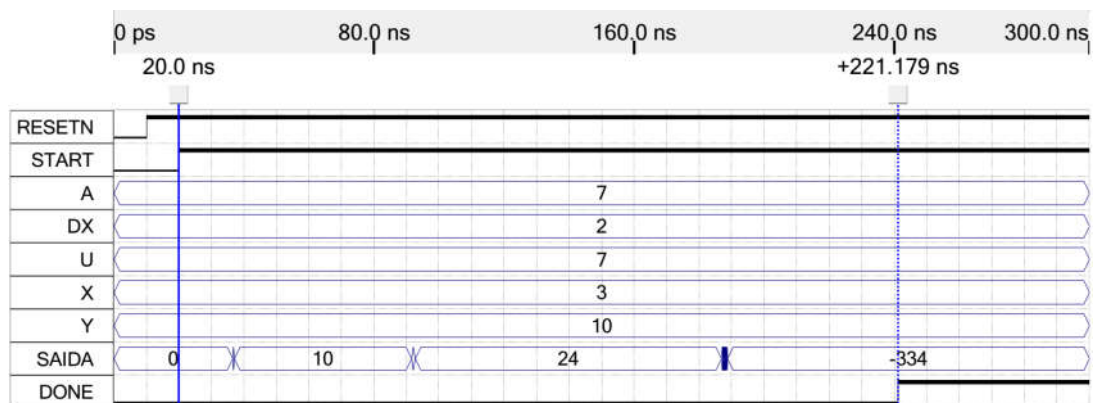


FIGURA B.2 – Simulação. Diffeq V.I. MRRT

2) Diffeq – Versão Final:

```

in x      (16)
in dx     (16)
in y      (16)
in u      (16)
in a      (16)
var x1    (16)
var y1    (16)
var u1    (16)
out comp  (1)
out saida (16)

main:
x1<:x
y1<:y
u1<:u

while [comp <: x1<a]
  x1 <: x1+dx
  u1 <: u1-(3*x1*u1*dx)-(3*y1*dx)
  y1 <: y1+u1*dx
  saida <: y1
endwhile
end

```

FIGURA B.3 – Arquivo de Entrada. Diffeq V.F. MRRT

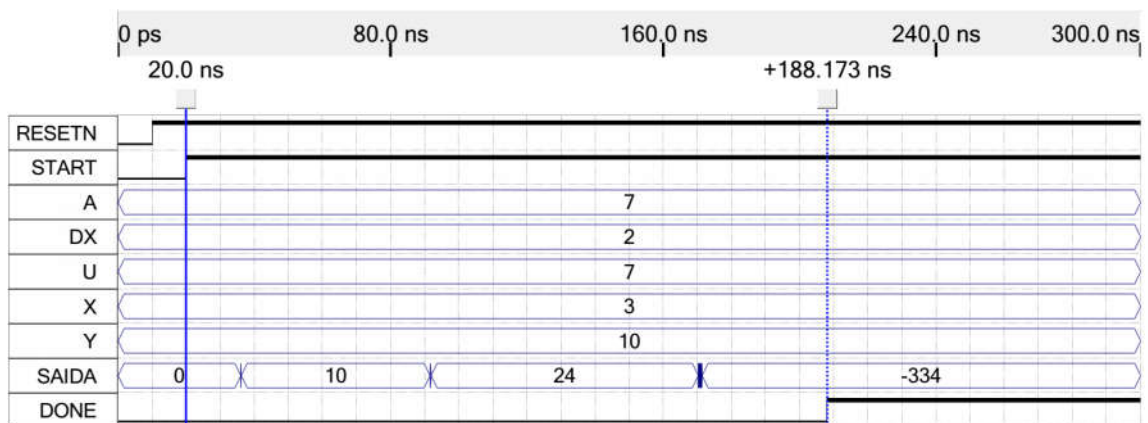


FIGURA B.4 – Simulação. Diffeq V.F. MRRT

3) EWF:

```

in i1 (16)
in i2 (16)
in i3 (16)
in i4 (16)
in i5 (16)
in i6 (16)
in i7 (16)
in i8 (16)
out o2 (16)
out o3 (16)
out o4 (16)
out o5 (16)
out o6 (16)
out o7 (16)
out o8 (16)
out o9 (16)

main:
o2 <: 126*i1+125*i2+112*i3+56*(i4+i7+i8)
o3 <: 160*(i1+i2)+152*i3+9*i5+80*(i4+i7+i8)
o4 <: 7*(i1+i2+i3+i7+i8)+6*i4
o5 <: 140*(i1+i2)+133*i3+8*i5+70*(i4+i7+i8)
o6 <: 144*(i1+i2+i3+i4)+9*i6+232*i7+240*i8
o7 <: 162*(i1+i2+i3+i4)+10*i6+261*i7+270*i8
o8 <: 150*(i1+i2+i3+i4)+250*i7+269*i8
o9 <: 135*(i1+i2+i3+i4)+225*i7+243*i8
end
    
```

FIGURA B.5 – Arquivo de Entrada. EWF. MRRT



FIGURA B.6 – Simulação. EWF. MRRT

4) FAT – Versão Inicial:

```

in n (8)
in x (8)
out fat1(32)
var i (8)
var fat (32)
out cmp (1)

main:
i<:1
fat<:1

while[cmp<: i<=n]
  fat <: fat * i
  i <: i+1
  fat1 <: fat
endwhile
end

```

FIGURA B.7 – Arquivo de Entrada. FAT V.I. MRRT

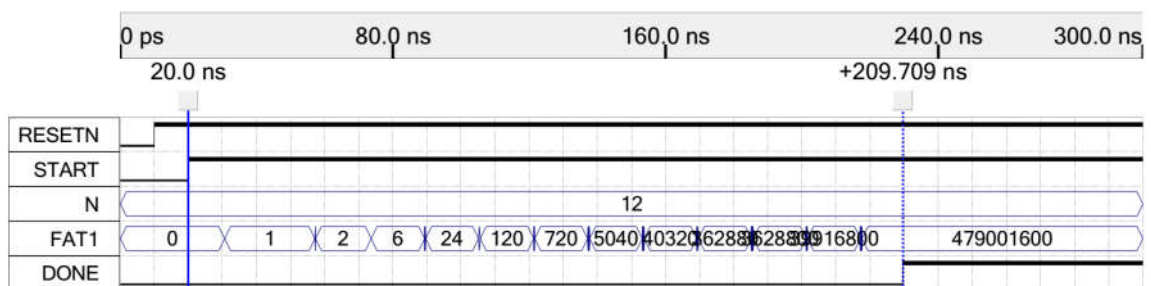


FIGURA B.8 – Simulação. FAT V.I. MRRT

5) FAT – Versão Final:

```

in n (8)
in x (8)
out fat1(32)
var i (8)
var fat (32)
out cmp (1)

main:
i<:1
fat<:1

while[cmp<: i<=n]
  fat <: fat * i
  i <: i+1
  fat1 <: fat
endwhile
end

```

FIGURA B.9 – Arquivo de Entrada. FAT V.F. MRRT

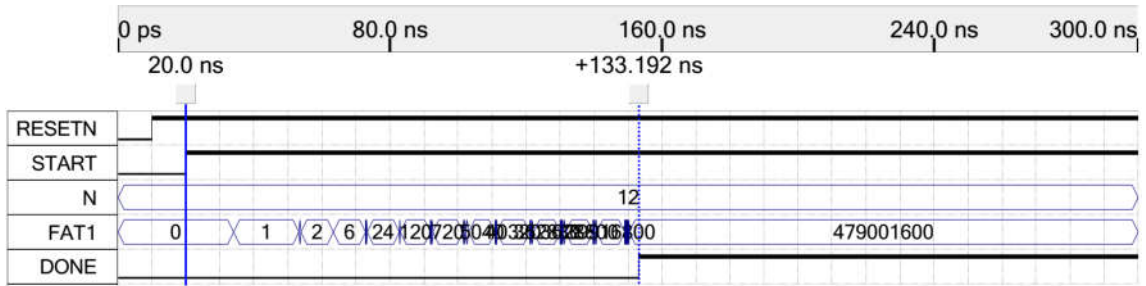


FIGURA B.10 – Simulação. FAT V.F. MRRT

6) SQR:

```

in y (8)
out xout(8)
var gi (8)
var sgi (8)
var x (8)
out cmp (1)
out cmp2 (1)
out sucess(1)

main:
gi<:1
sgi<:0
x<:0

while[cmp<: sgi<y]
    sgi <: sgi+gi
    gi <: gi+2
    x <: x+1
    xout <: x
endwhile
if[cmp2<: sgi=y]
    sucess<: 1
else
    sucess<: 0
endif
end
    
```

FIGURA B.11 – Arquivo de Entrada. SQR. MRRT

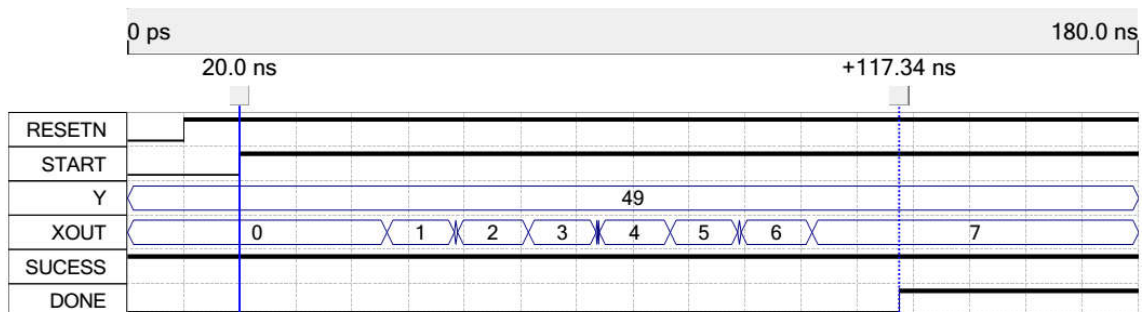


FIGURA B.12 – Simulação. SQR. MRRT

7) GCD:

```

in a (8)
in b (8)
out c (8)
var ai (8)
var bi (8)
out cmp (1)
out cmp2 (1)

main:
ai<:a
bi<:b

while[cmp<: ai!bi]
c <: ai
if[cmp2<: ai>bi]
ai <: ai - bi
else
bi <: bi - ai
endif
endwhile
end

```

FIGURA B.13 – Arquivo de Entrada. GCD. MRRT

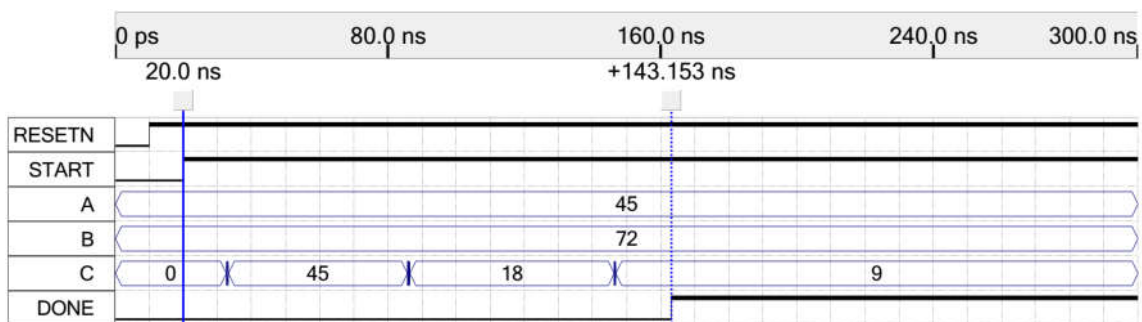


FIGURA B.14 – Simulação. GCD. MRRT

8) FIR:

```

in h0 (16)
in xt (16)
in h1 (16)
in xt1 (16)
in h2 (16)
in xt2 (16)
in h3 (16)
in xt3 (16)
in h4 (16)
in xt4 (16)

out y (16)

main:
y <: (h0*xt)+(h1*xt1)+(h2*xt2)+(h3*xt3)+(h4*xt4)
end

```

FIGURA B.15 – Arquivo de Entrada. FIR. MRRT

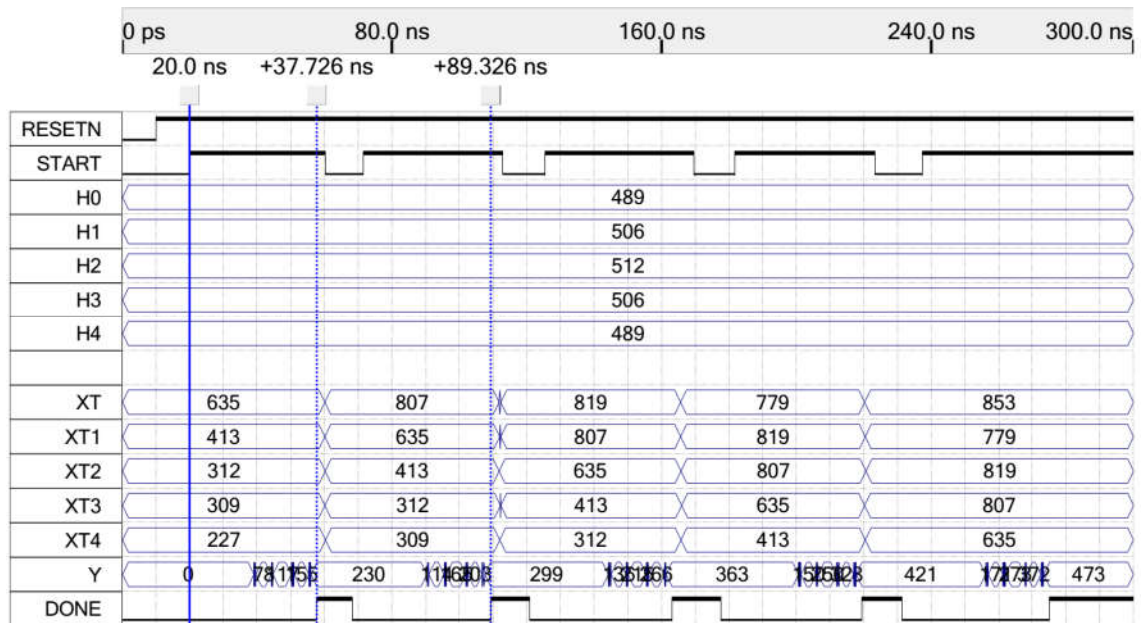


FIGURA B.16 – Simulação. FIR. MRRT

9) IIR:

```

in xn (16)
in xn1 (16)
in xn2 (16)
in yn1 (16)
in yn2 (16)
in b0 (16)
in b1 (16)
in b2 (16)
in a1 (16)
in a2 (16)

out y (16)

main:
y <: ((xn*b0)+(xn1*b1)+(xn2*b2)+(yn1*a1)+(yn2*a2))
end

```

FIGURA B.17 – Arquivo de Entrada. IIR. MRRT

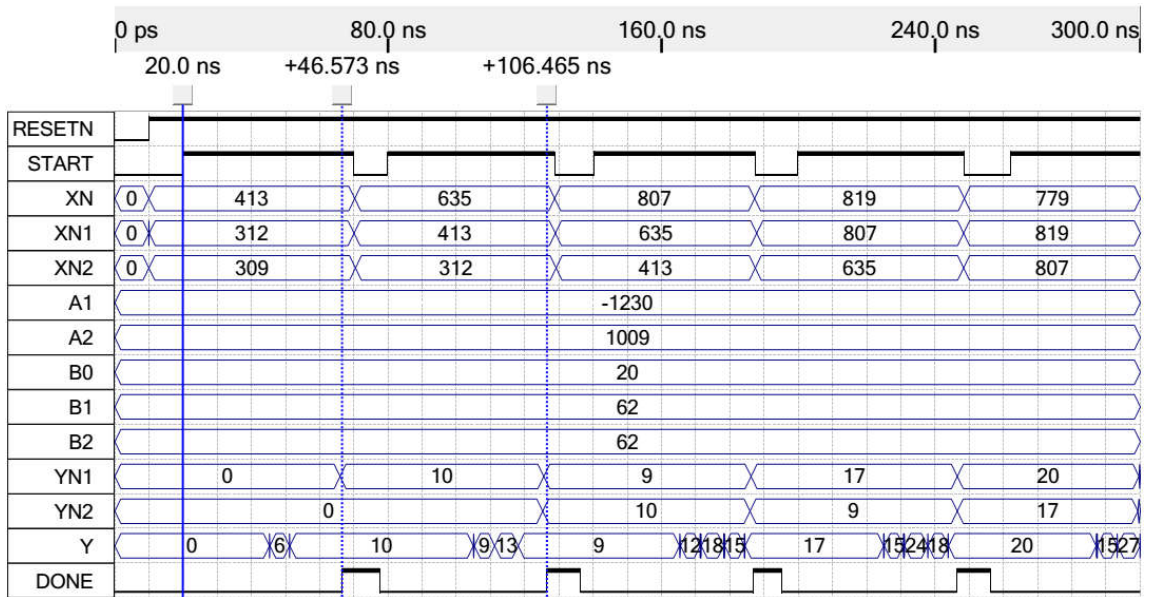


FIGURA B.18 – Simulação. IIR. MRRT

10) TEA-E – Versão Inicial:

```

in v0 (32)
in v1 (32)
var y (32)
var z (32)
var sum (32)
var n (8)
out cmp (1)
out y1 (32)
out z1 (32)

main:
n <: 8
y <: v0
z <: v1
sum <: 2654435769

while[cmp <: n > 0]
  n <: n - 1
  sum <: sum + 2654435769
  y <: (y + (((z << 4) + 10) ^ (z + sum) ^ ((z >> 5) + 15)))
  z <: z + (((y + (((z << 4) + 10) ^ (z + sum) ^ ((z >> 5) + 15))) << 4) + 20) ^ ((y + (((z << 4) + 10) ^ (z + sum) ^ ((z >> 5) + 15))) + sum) ^ (((y + (((z << 4) + 10) ^ (z + sum) ^ ((z >> 5) + 15))) >> 5) + 25))
  y1 <: y
  z1 <: z
endwhile
end

```

FIGURA B.19 – Arquivo de Entrada. TEA-e V.I. MRRT

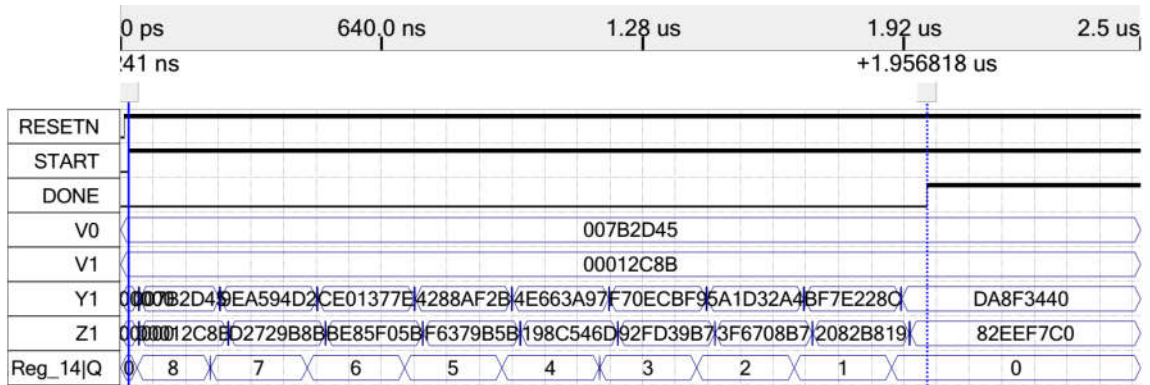


FIGURA B.20 – Simulação. TEA-e V.I. MRRT

11) TEA-E – Versão Final:

```

in v0 (32)
in v1 (32)
var y (32)
var z (32)
var sum (32)
var n (8)
out cmp (1)
out y1 (32)
out z1 (32)

main:
n <: 8
y <: v0
z <: v1
sum <: 2654435769

while[cmp<: n > 0]
n <: n - 1
sum <: sum + 2654435769
y <: (y + (((z << 4) + 10) ^ (z + sum) ^ ((z >> 5) + 15)))
z <: z + (((y + (((z << 4) + 10) ^ (z + sum) ^ ((z >> 5) + 15))) << 4) + 20) ^ ((y + (((z << 4) + 10) ^ (z + sum) ^ ((z >> 5) + 15))) + sum) ^ (((y + (((z << 4) + 10) ^ (z + sum) ^ ((z >> 5) + 15))) >> 5) + 25))
y1 <: y
z1 <: z
endwhile
end
    
```

FIGURA B.21 – Arquivo de Entrada. TEA-e V.F. MRRT

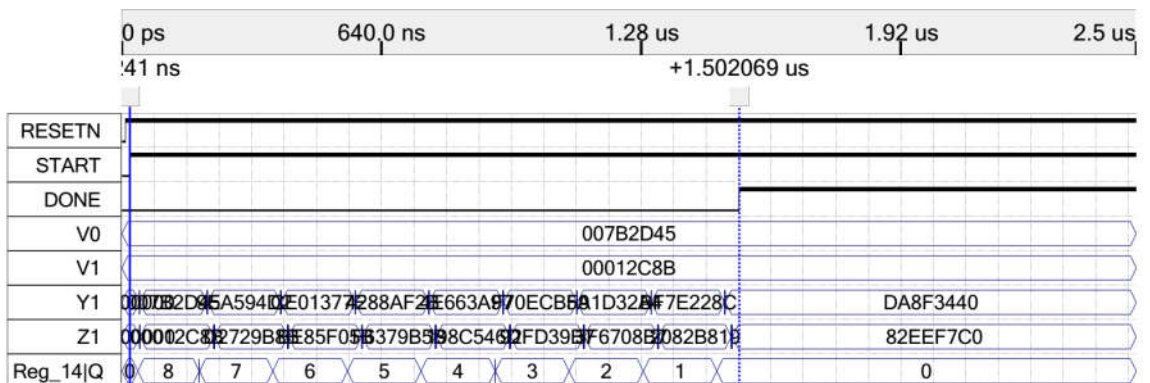


FIGURA B.22 – Simulação. TEA-e V.F. MRRT

12) TEA-D – Versão Inicial:

```

in v0 (32)
in v1 (32)
var y (32)
var z (32)
var sum (32)
var n (8)
out cmp (1)
out y1 (32)
out z1 (32)

main:
n <: 8
y <: v0
z <: v1
sum <: 4055616968

while[cmp<: n > 0]
  n <: n - 1
  sum <: sum - 2654435769
  z <: (z - (((y < 4) + 20) ^ (y + sum) ^ ((y > 5) + 25)))
  y <: (y - (((z - (((y < 4) + 20) ^ (y + sum) ^ ((y > 5) + 25))) < 4) + 10) ^ ((z - (((y < 4) + 20) ^ (y + sum) ^ ((y > 5) + 25))) + sum) ^ ((z - (((y < 4) + 20) ^ (y + sum) ^ ((y > 5) + 25))) >> 5) + 15)))
  y1 <: y
  z1 <: z
endwhile
end
    
```

FIGURA B.23 – Arquivo de Entrada. TEA-d V.I. MRRT

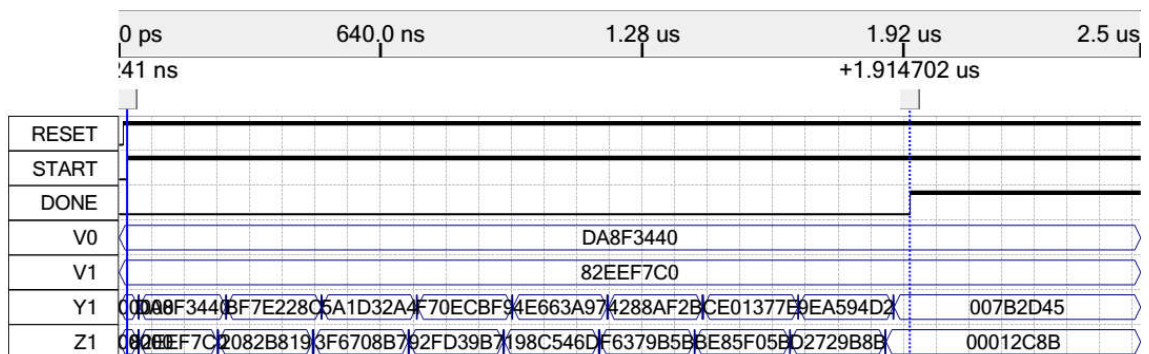


FIGURA B.24 – Simulação. TEA-d V.I. MRRT

13) TEA-D – Versão Final:

```

in v0 (32)
in v1 (32)
var y (32)
var z (32)
var sum (32)
var n (8)
out cmp (1)
out y1 (32)
out z1 (32)

main:
n <: 8
y <: v0
z <: v1
sum <: 4055616968

while[cmp<: n > 0]
  n <: n - 1
  sum <: sum - 2654435769
  z <: (z - (((y << 4) + 20) ^ (y + sum) ^ ((y >> 5) + 25)))
  y <: (y - (((z - ((y << 4) + 20) ^ (y + sum) ^ ((y >> 5) + 25))) << 4) + 10) ^ ((z - ((y << 4) + 20) ^ (y + sum) ^ ((y >> 5) + 25))) + sum) ^ (((z - ((y << 4) + 20) ^ (y + sum) ^ ((y >> 5) + 25))) >> 5) + 15)))
  y1 <: y
  z1 <: z
endwhile
end

```

FIGURA B.25 – Arquivo de Entrada. TEA-d V.F. MRRT

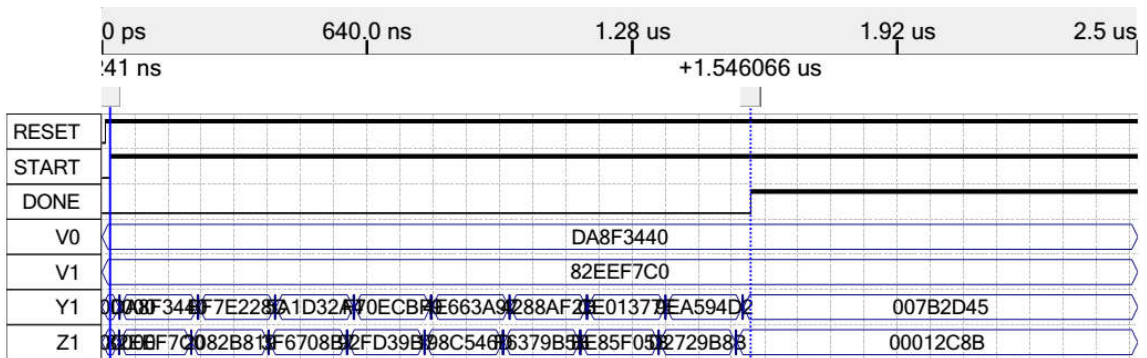


FIGURA B.26 – Simulação. TEA-d V.F. MRRT

14) Divisão:

```

in y (16)
in x (16)
var y1 (16)
var q1 (16)
out q (16)
out cmp(1)

main:
y1 <: y
q1 <: 0

  while[cmp <: y1 >= x]
    y1 <: y1 - x
    q1 <: q1 + 1
  endwhile
q <: q1
end

```

FIGURA B.27 – Arquivo de Entrada. Divisão. MRRT

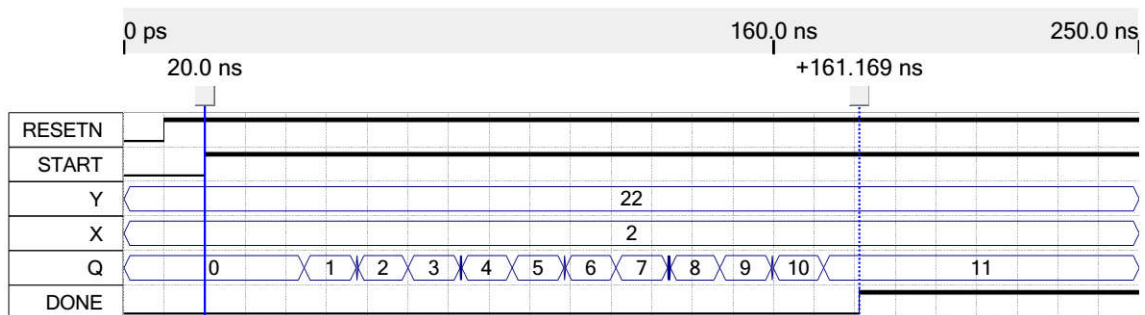


FIGURA B.28 – Simulação. Divisão. MRRT

15) DotProd – Versão Inicial:

```

in m (8)
in n (8)
in tam(4)
var cont (4)
var sum (64)
out summ (64)
out count (4)
out cmp(1)

main:
cont <: 0
sum <: 0

  while[cmp <: cont < tam]
    cont <: cont + 1
    sum <: sum + (m * n)
    count <: cont
  endwhile
summ <: sum

end

```

FIGURA B.29 – Arquivo de Entrada. DotProd V.I. MRRT

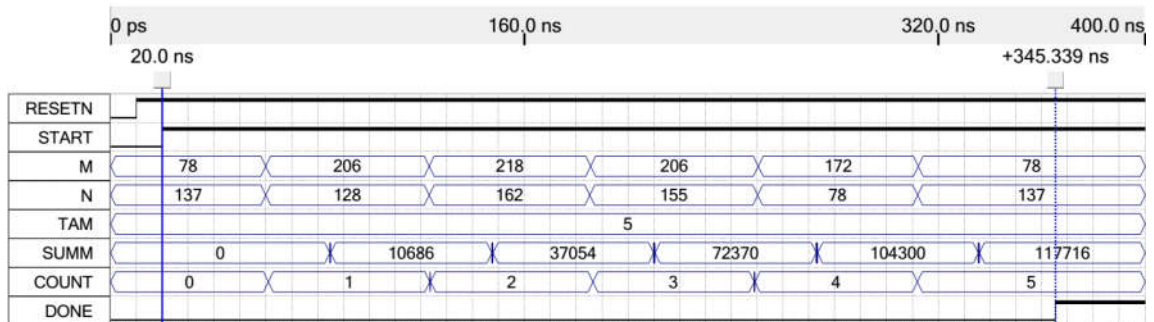


FIGURA B.30 – Simulação. DotProd V.I. MRRT

16) DotProd – Versão Final:

```

in m (8)
in n (8)
in tam(4)
var cont (4)
var sum (64)
out summ (64)
out count (4)
out cmp(1)

main:
cont <: 0
sum <: 0

while[cmp <: cont < tam]
  cont <: cont + 1
  sum <: sum + (m * n)
  count <: cont
endwhile
summ <: sum

end

```

FIGURA B.31 – Arquivo de Entrada. DotProd V.F. MRRT

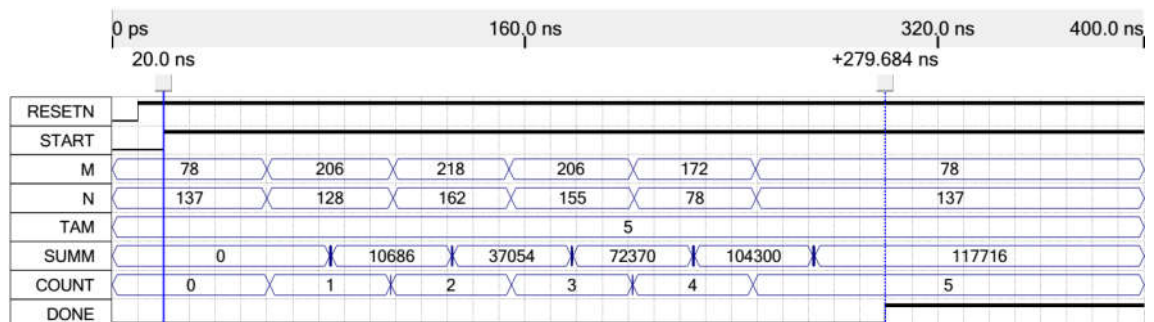


FIGURA B.32 – Simulação. DotProd V.F. MRRT

17) Fibonacci – Versão Inicial:

```

in ent (5)
out res(32)
var x (32)
var z (32)
var n (5)
out cmp (1)
out cmp2 (1)

main:
x<:0
z<:1
n<:ent

while[cmp<: n>1]
  x <: x + z
  z <: x + z + z
  n <: n - 2
endwhile

if[cmp2<: n = 0]
  res <: x
else
  res <: z
endif
end

```

FIGURA B.33 – Arquivo de Entrada. Fibonacci. V.I. MRRT

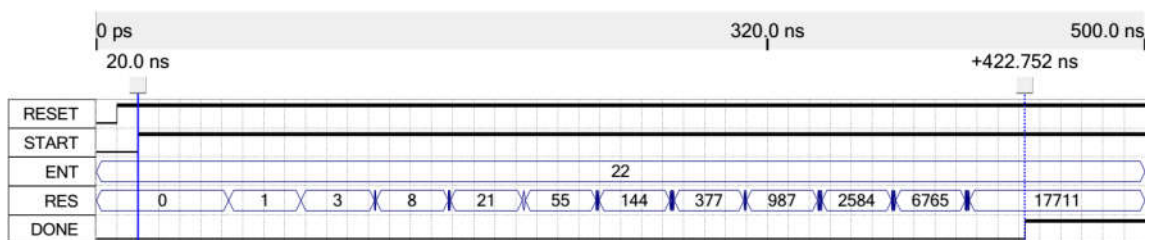


FIGURA B.34 – Simulação. Fibonacci. V.I. MRRT

18) Fibonacci – Versão Final:

```

in ent (5)
out res(32)
var x (32)
var z (32)
var n (5)
out cmp (1)
out cmp2 (1)

main:
x<:0
z<:1
n<:ent

while[cmp<: n>1]
  x <: x + z
  z <: x + z + z
  n <: n - 2
endwhile

if[cmp2<: n = 0]
  res <: x
else
  res <: z
endif
end

```

FIGURA B.35 – Arquivo de Entrada. Fibonacci. V.F. MRRT

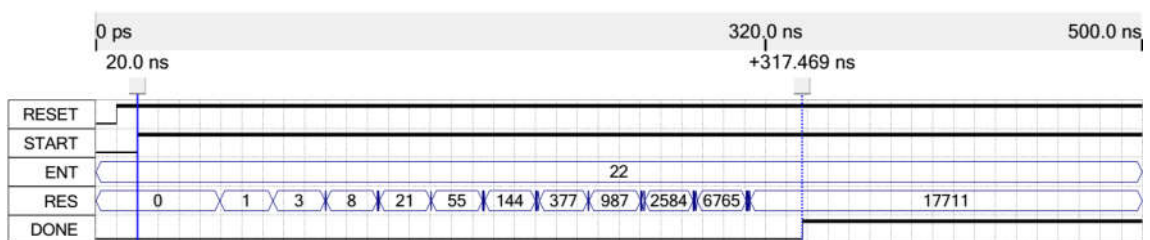


FIGURA B.36 – Simulação. Fibonacci. V.F. MRRT

19) Wavelet:

```

;a0 = 0,3327 -> 0,3327*4096 = 1363
;a1 = 0,8069 -> 0,8069*4096 = 3305
;a2 = 0,4598 -> 0,4598*4096 = 1883
;a3 = 0,1350 -> 0,1350*4096 = 553
;a4 = 0,0854 -> 0,0854*4096 = 350
;a5 = 0,0352 -> 0,0352*4096 = 144
in xn (32)
in xn1 (32)
in xn2 (32)
in xn3 (32)
in xn4 (32)
in xn5 (32)

out an (32)
out dn (32)

main:
an <: ((1363*xn)>>12)+((3305*xn1)>>12)+((1883*xn2)>>12)+((553*xn3)>>12)+((350*xn4)>>12)+((144*xn5)>>12)
dn <: ((1363*xn)>>12)-((3305*xn1)>>12)+((1883*xn2)>>12)-((553*xn3)>>12)+((350*xn4)>>12)-((144*xn5)>>12)
end
    
```

FIGURA B.37 – Arquivo de Entrada. Wavelet. MRRT

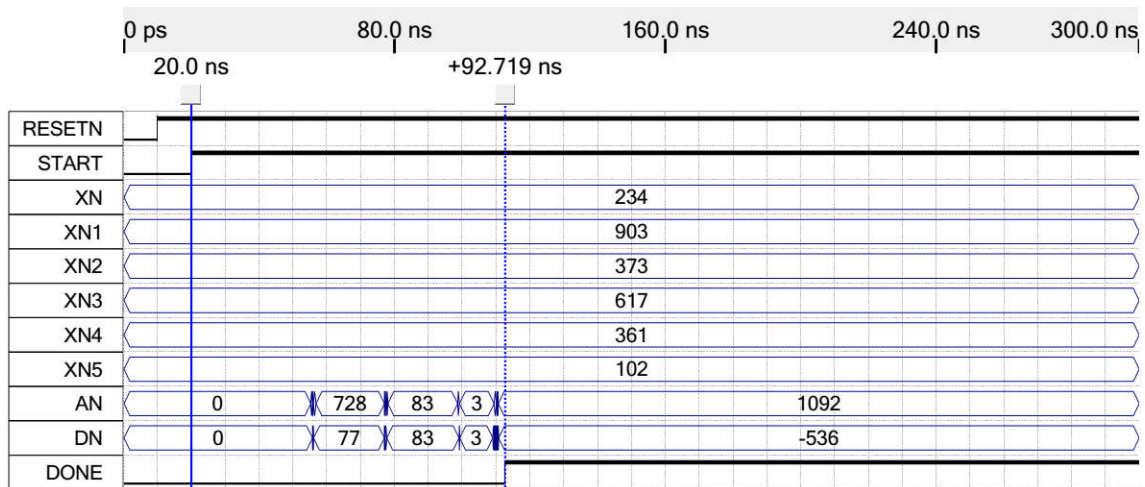


FIGURA B.38 – Simulação. Wavelet. MRRT

B.2 Simulações Utilizando Escalonamento MTRR

Os benchmarks simulados nesta seção utilizam o escalonamento MTRR, e correspondem aos dados apresentados no Capítulo 6.

1) Diffeq:

```

in x      (16)
in dx     (16)
in y      (16)
in u      (16)
in a      (16)
var x1    (16)
var y1    (16)
var u1    (16)
out comp  (1)
out saida (16)

main:
x1<:x
y1<:y
u1<:u

while [comp <: x1<a]
  x1 <: x1+dx
  u1 <: u1-(3*x1*u1*dx)-(3*y1*dx)
  y1 <: y1+u1*dx
  saida <: y1
endwhile
end

```

FIGURA B.39 – Arquivo de Entrada. Diffeq. MTRR

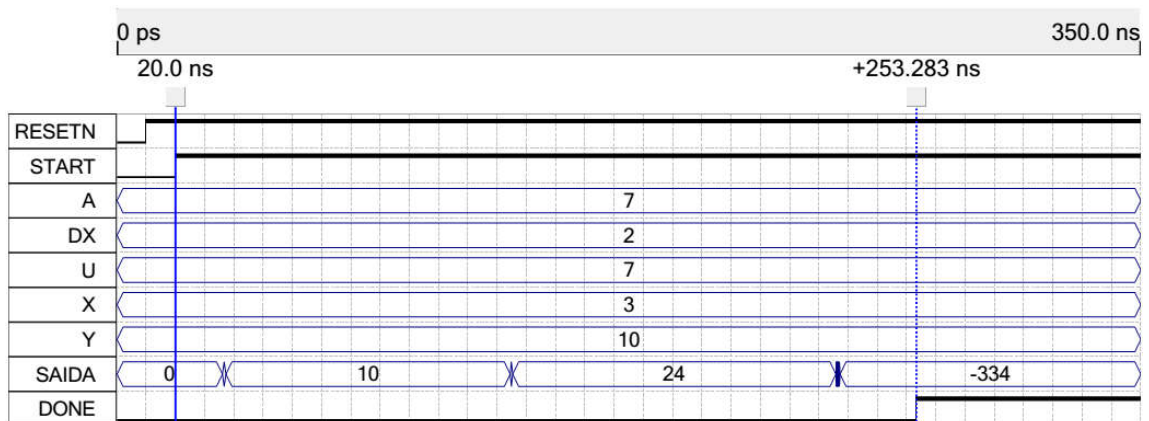


FIGURA B.40 – Simulação. Diffeq. MTRR

2) EWF:

```

in i1 (16)
in i2 (16)
in i3 (16)
in i4 (16)
in i5 (16)
in i6 (16)
in i7 (16)
in i8 (16)
out o2 (16)
out o3 (16)
out o4 (16)
out o5 (16)
out o6 (16)
out o7 (16)
out o8 (16)
out o9 (16)

main:
o2 <: 126*i1+125*i2+112*i3+56*(i4+i7+i8)
o3 <: 160*(i1+i2)+152*i3+9*i5+80*(i4+i7+i8)
o4 <: 7*(i1+i2+i3+i7+i8)+6*i4
o5 <: 140*(i1+i2)+133*i3+8*i5+70*(i4+i7+i8)
o6 <: 144*(i1+i2+i3+i4)+9*i6+232*i7+240*i8
o7 <: 162*(i1+i2+i3+i4)+10*i6+261*i7+270*i8
o8 <: 150*(i1+i2+i3+i4)+250*i7+269*i8
o9 <: 135*(i1+i2+i3+i4)+225*i7+243*i8
end
    
```

FIGURA B.41 – Arquivo de Entrada. EWF. MTRR

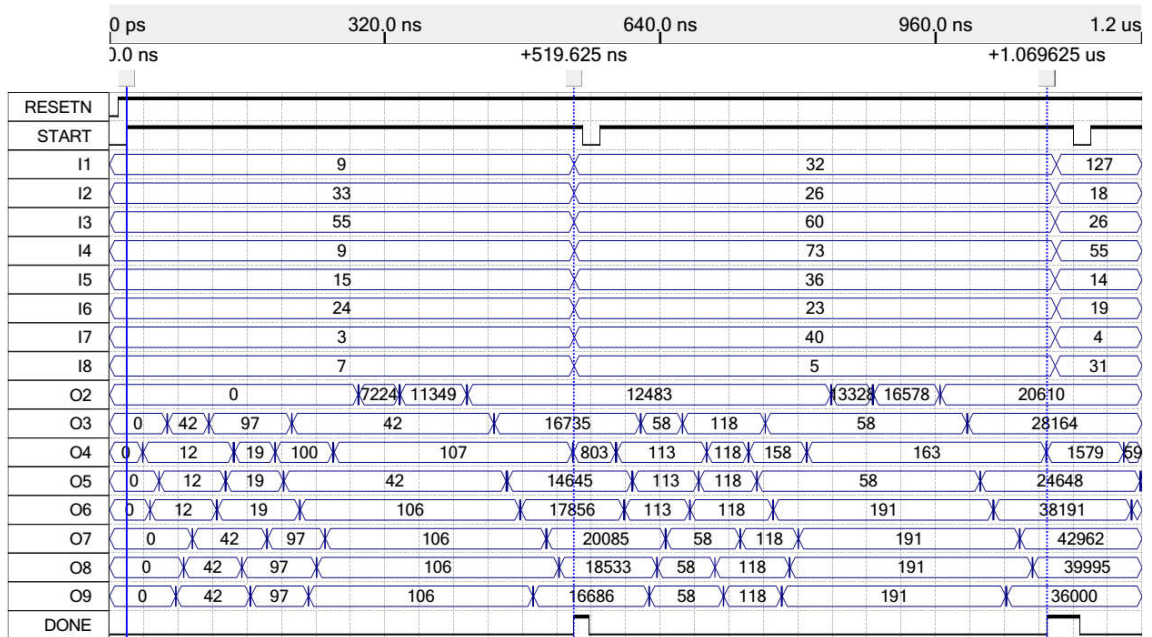


FIGURA B.42 – Simulação. EWF. MTRR

3) FAT – Versão Inicial:

```

in n (8)
in x (8)
out fat1(32)
var i (8)
var fat (32)
out cmp (1)

main:
i<:1
fat<:1

while[cmp<: i<=n]
fat <: fat * i
i <: i+1
fat1 <: fat
endwhile
end

```

FIGURA B.43 – Arquivo de Entrada. FAT. V.I. MTRR

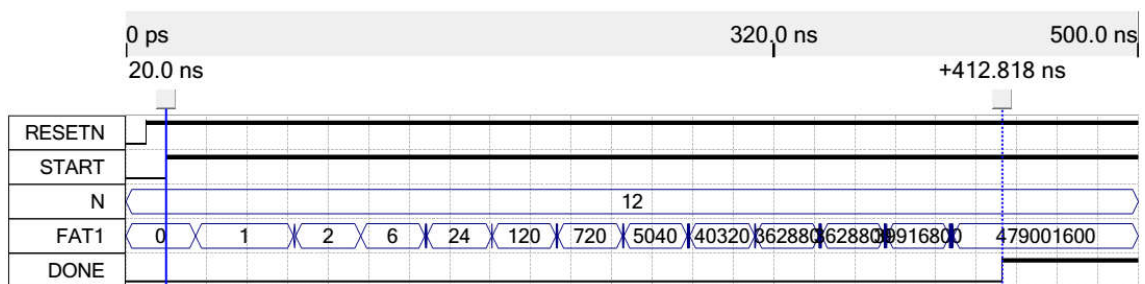


FIGURA B.44 – Simulação. FAT. V.I. MTRR

4) FAT – Versão Final:

```

in n (8)
in x (8)
out fat1(32)
var i (8)
var fat (32)
out cmp (1)

main:
i<:1
fat<:1

while[cmp<: i<=n]
fat <: fat * i
i <: i+1
fat1 <: fat
endwhile
end

```

FIGURA B.45 – Arquivo de Entrada. FAT. V.F. MTRR

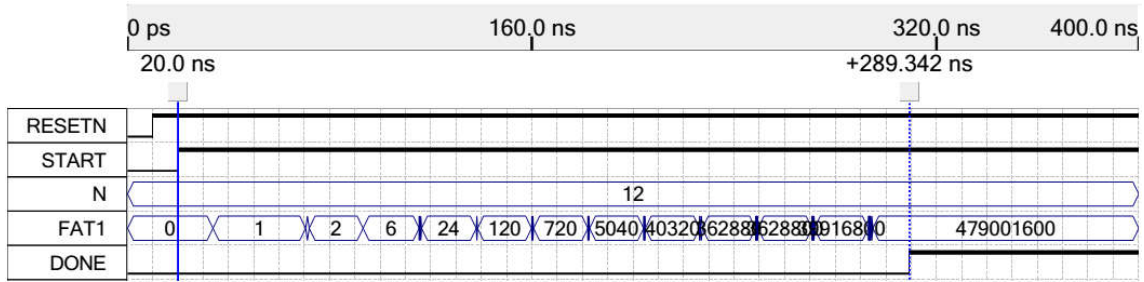


FIGURA B.46 – Simulação. FAT. V.F. MTRR

5) SQR – Versão Inicial:

```

in y (8)
out xout(8)
var gi (8)
var sgi (8)
var x (8)
out cmp (1)
out cmp2 (1)
out sucess(1)

main:
gi<:1
sgi<:0
x<:0

while[cmp<: sgi<y]
  sgi <: sgi+gi
  gi <: gi+2
  x <: x+1
  xout <: x
endwhile
if[cmp2<: sgi=y]
  sucess<: 1
else
  sucess<: 0
endif
end
    
```

FIGURA B.47 – Arquivo de Entrada. SQR. V.I. MTRR

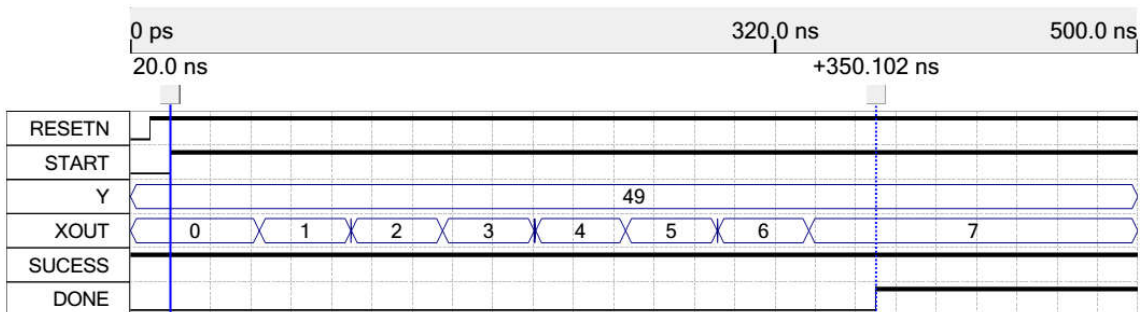


FIGURA B.48 – Simulação. SQR. V.I. MTRR

6) SQR – Versão Final:

```

in y (8)
out xout(8)
var gi (8)
var sgi (8)
var x (8)
out cmp (1)
out cmp2 (1)
out sucess(1)

main:
gi<:1
sgi<:0
x<:0

while[cmp<: sgi<y]
  sgi <: sgi+gi
  gi <: gi+2
  x <: x+1
  xout <: x
endwhile
if[cmp2<: sgi=y]
  sucess<: 1
else
  sucess<: 0
endif
end

```

FIGURA B.49 – Arquivo de Entrada. SQR. V.F. MTRR

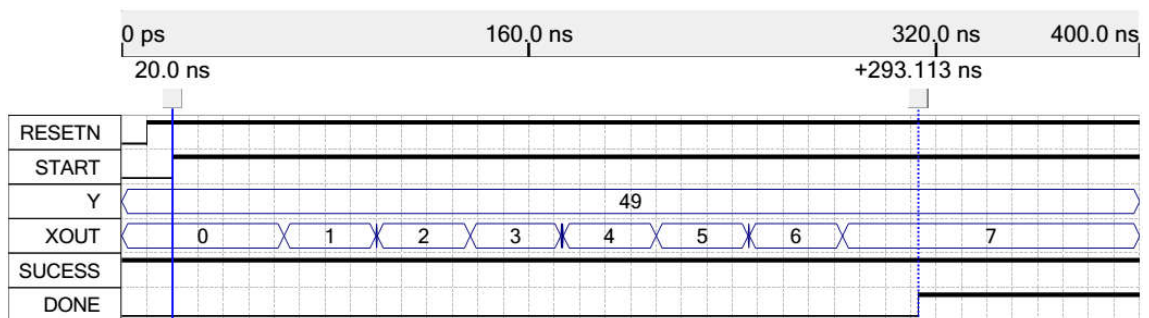


FIGURA B.50 – Simulação. SQR. V.F. MTRR

7) GCD:

```

in a (8)
in b (8)
out c (8)
var ai (8)
var bi (8)
out cmp (1)
out cmp2 (1)

main:
ai<:a
bi<:b

while[cmp<: ai!bi]
  c <: ai
  if[cmp2<: ai>bi]
    ai <: ai - bi
  else
    bi <: bi - ai
  endif
endwhile
end

```

FIGURA B.51 – Arquivo de Entrada. GCD. MTRR

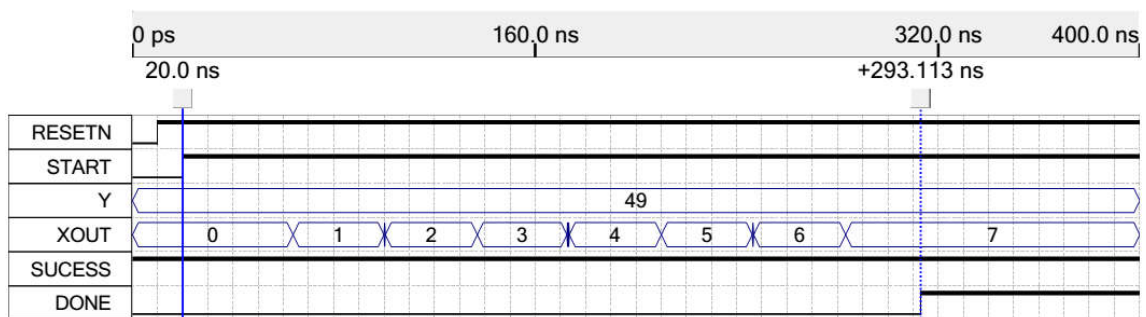


FIGURA B.52 – Simulação. GCD. MTRR

8) FIR:

```

in h0 (16)
in xt (16)
in h1 (16)
in xt1 (16)
in h2 (16)
in xt2 (16)
in h3 (16)
in xt3 (16)
in h4 (16)
in xt4 (16)

out y (16)

main:
y <: (h0*xt)+(h1*xt1)+(h2*xt2)+(h3*xt3)+(h4*xt4)
end

```

FIGURA B.53 – Arquivo de Entrada. FIR. MTRR

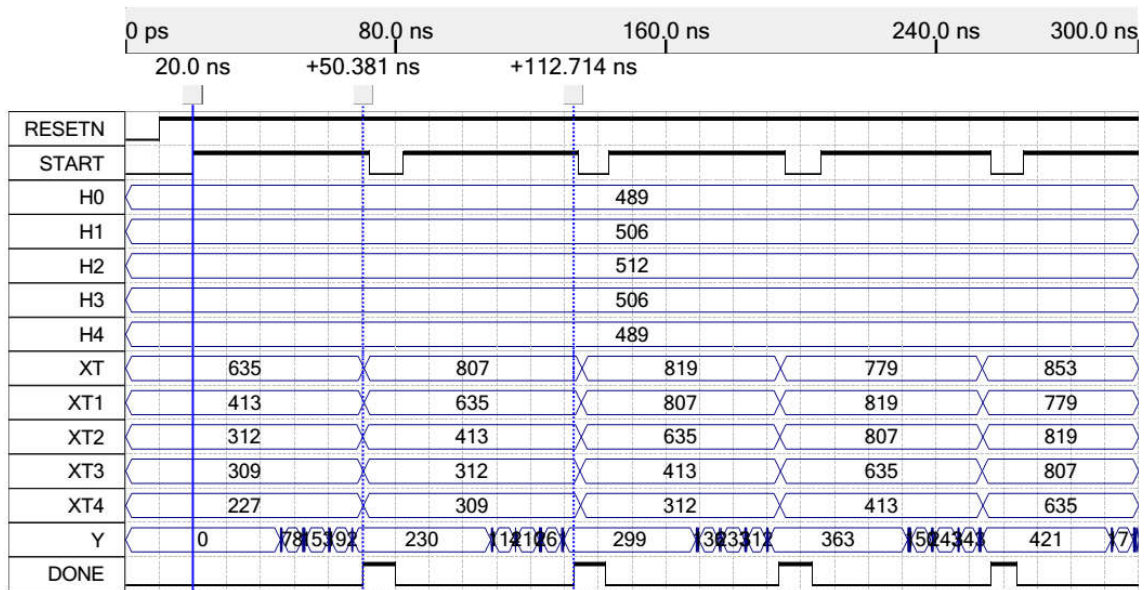


FIGURA B.54 – Simulação. FIR. MTRR

9) IIR:

```

in xn (16)
in xn1 (16)
in xn2 (16)
in yn1 (16)
in yn2 (16)
in b0 (16)
in b1 (16)
in b2 (16)
in a1 (16)
in a2 (16)

out y (16)

main:
y <- ((xn*b0)+(xn1*b1)+(xn2*b2)+(yn1*a1)+(yn2*a2))
end

```

FIGURA B.55 – Arquivo de Entrada. IIR. MTRR

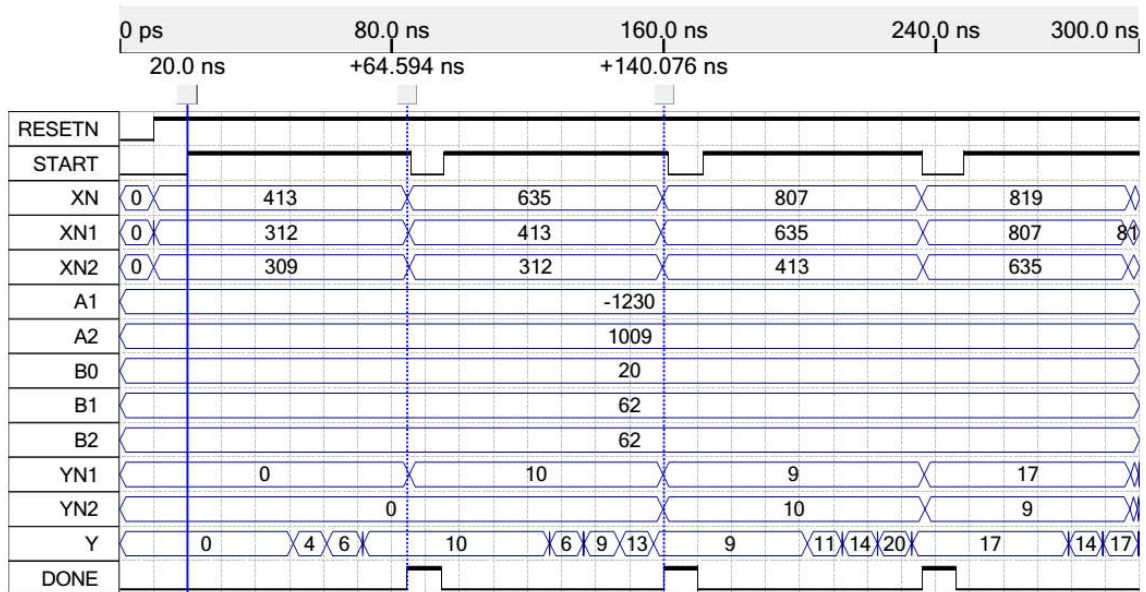


FIGURA B.56 – Simulação. IIR. MTRR

10) TEA-E:

```

in v0 (32)
in v1 (32)
var y (32)
var z (32)
var sum (32)
var n (8)
out cmp (1)
out y1 (32)
out z1 (32)

main:
n <: 8
y <: v0
z <: v1
sum <: 2654435769

while[cmp<: n > 0]
  n <: n - 1
  sum <: sum + 2654435769
  y <: (y + (((z << 4) + 10) ^ (z + sum) ^ ((z >> 5) + 15)))
  z <: z + (((y + (((z << 4) + 10) ^ (z + sum) ^ ((z >> 5) + 15))) << 4) + 20) ^ ((y + (((z << 4) + 10) ^ (z + sum) ^ ((z >> 5) + 15))) + sum) ^ (((y + (((z << 4) + 10) ^ (z + sum) ^ ((z >> 5) + 15))) >> 5) + 25))
  y1 <: y
  z1 <: z
endwhile
end

```

FIGURA B.57 – Arquivo de Entrada. TEA-e. MTRR

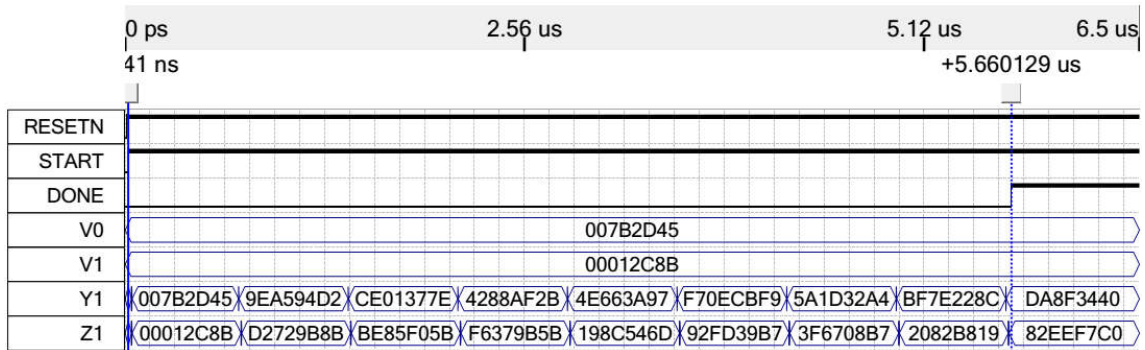


FIGURA B.58 – Simulação. TEA-e. MTRR

11) TEA-D:

```

in v0 (32)
in v1 (32)
var y (32)
var z (32)
var sum (32)
var n (8)
out cmp (1)
out y1 (32)
out z1 (32)

main:
n <: 8
y <: v0
z <: v1
sum <: 4055616968

while[cmp<: n > 0]
  n <: n - 1
  sum <: sum - 2654435769
  z <: (z - (((y < 4) + 20) ^ (y + sum) ^ ((y > 5) + 25)))
  y <: (y - (((z - (((y < 4) + 20) ^ (y + sum) ^ ((y > 5) + 25))) << 4) + 10) ^ ((z - (((y < 4) + 20) ^ (y + sum) ^ ((y > 5) + 25))) + sum) ^ (((z - (((y < 4) + 20) ^ (y + sum) ^ ((y > 5) + 25))) >> 5) + 15)))
  y1 <: y
  z1 <: z
endwhile
end
    
```

FIGURA B.59 – Arquivo de Entrada. TEA-d. MTRR

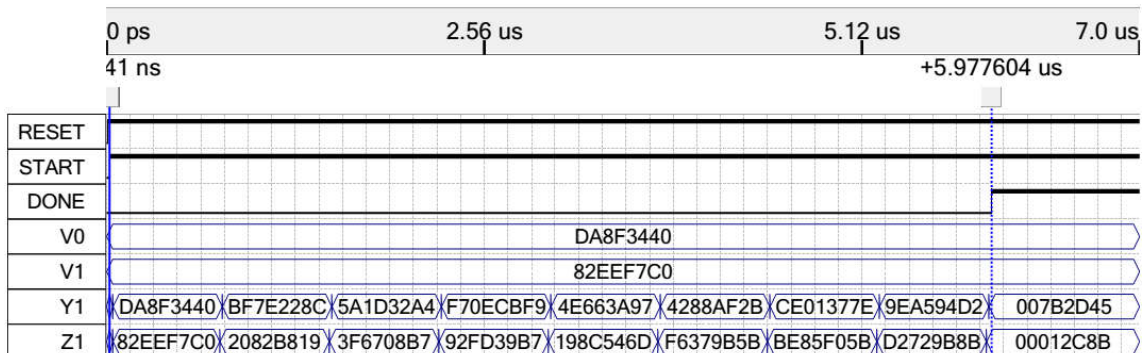


FIGURA B.60 – Simulação. TEA-d. MTRR

12) Divisão:

```

in y (16)
in x (16)
var y1 (16)
var q1 (16)
out q (16)
out cmp(1)

main:
y1 <: y
q1 <: 0

while[cmp <: y1 >= x]
  y1 <: y1 - x
  q1 <: q1 + 1
endwhile
q <: q1
end

```

FIGURA B.61 – Arquivo de Entrada. Divisão. MTRR

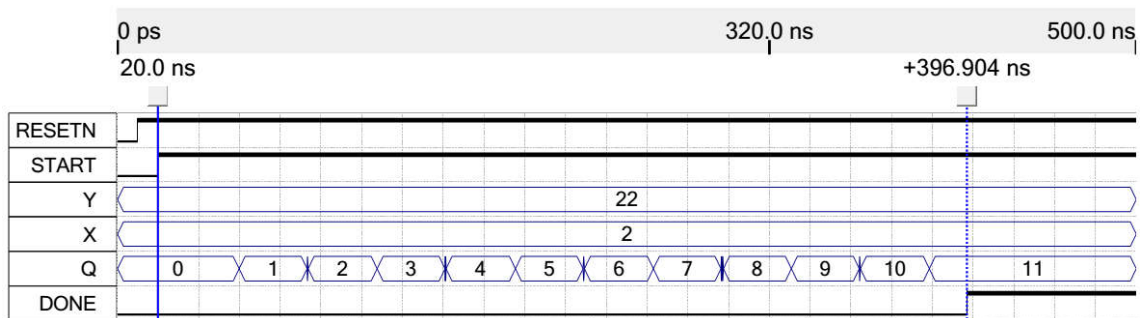


FIGURA B.62 – Simulação. Divisão. MTRR

13) DotProd Versão Inicial:

```

in m (8)
in n (8)
in tam(4)
var cont (4)
var sum (64)
out summ (64)
out count (4)
out cmp(1)

main:
cont <: 0
sum <: 0

while[cmp <: cont < tam]
  cont <: cont + 1
  sum <: sum + (m * n)
  count <: cont
endwhile
summ <: sum

end

```

FIGURA B.63 – Arquivo de Entrada. DotProd. V.I. MTRR

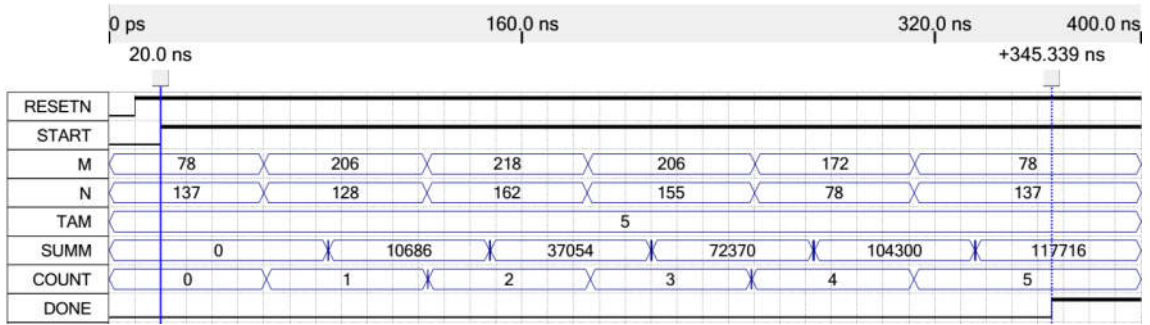


FIGURA B.64 – Simulação. DotProd. V.I. MTRR

14) DotProd Versão Final:

```

in m (8)
in n (8)
in tam(4)
var cont (4)
var sum (64)
out summ (64)
out count (4)
out cmp(1)

main:
cont <: 0
sum <: 0

while[cmp <: cont < tam]
  cont <: cont + 1
  sum <: sum + (m * n)
  count <: cont
endwhile
summ <: sum

end
    
```

FIGURA B.65 – Arquivo de Entrada. DotProd. V.F. MTRR

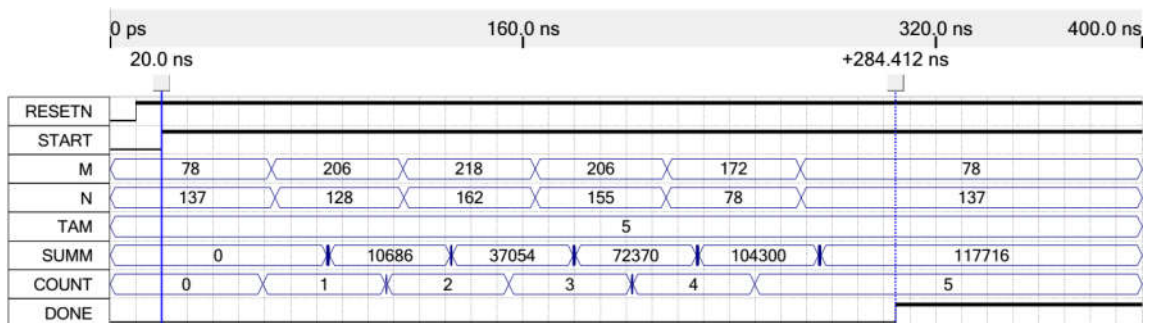


FIGURA B.66 – Simulação. DotProd. V.F. MTRR

15) Fibonacci:

```

in ent (5)
out res(32)
var x (32)
var z (32)
var n (5)
out cmp (1)
out cmp2 (1)

main:
x<:0
z<:1
n<:ent

while[cmp<: n>1]
  x <: x + z
  z <: x + z + z
  n <: n - 2
endwhile

if[cmp2<: n = 0]
  res <: x
else
  res <: z
endif
end

```

FIGURA B.67 – Arquivo de Entrada. Fibonacci. MTRR

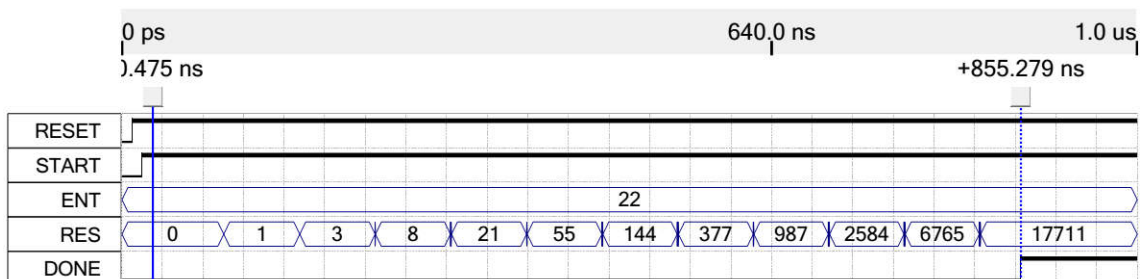


FIGURA B.68 – Simulação. Fibonacci. MTRR

16) Wavelet:

```

;a0 = 0,3327 -> 0,3327*4096 = 1363
;a1 = 0,8069 -> 0,8069*4096 = 3305
;a2 = 0,4598 -> 0,4598*4096 = 1883
;a3 = 0,1350 -> 0,1350*4096 = 553
;a4 = 0,0854 -> 0,0854*4096 = 350
;a5 = 0,0352 -> 0,0352*4096 = 144
in xn (32)
in xn1 (32)
in xn2 (32)
in xn3 (32)
in xn4 (32)
in xn5 (32)

out an (32)
out dn (32)

main:
an <: ((1363*xn)>>12)+((3305*xn1)>>12)+((1883*xn2)>>12)+((553*xn3)>>12)+((350*xn4)>>12)+((144*xn5)>>12)
dn <: ((1363*xn)>>12)-((3305*xn1)>>12)+((1883*xn2)>>12)-((553*xn3)>>12)+((350*xn4)>>12)-((144*xn5)>>12)
end

```

FIGURA B.69 – Arquivo de Entrada. Wavelet. MTRR

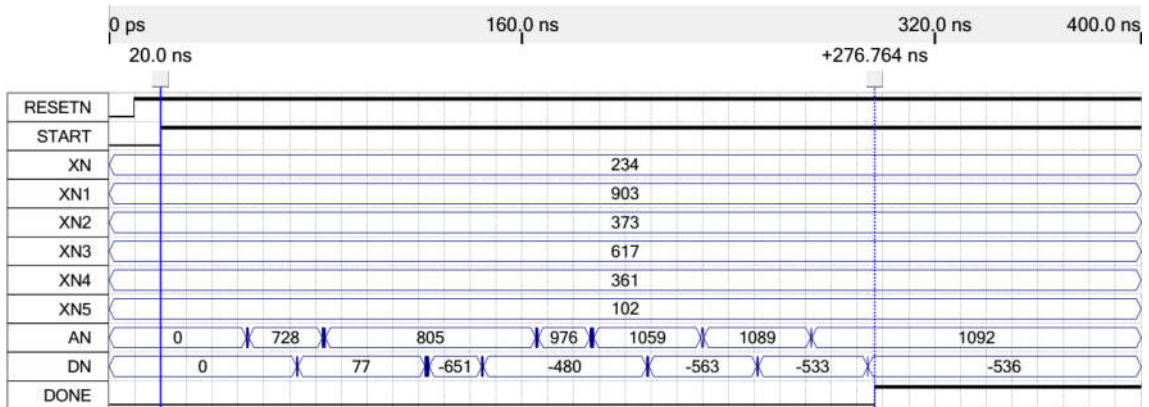


FIGURA B.70 – Simulação. Wavelet. MTRR

FOLHA DE REGISTRO DO DOCUMENTO

1. CLASSIFICAÇÃO/TIPO DM	2. DATA 23 de dezembro de 2015	3. REGISTRO N° DCTA/ITA/DM-087/2015	4. N° DE PÁGINAS 145
5. TÍTULO E SUBTÍTULO: Síntese comportamental de circuitos assíncronos otimizados.			
6. AUTOR(ES): Kledermom Garcia			
7. INSTITUIÇÃO(ÕES)/ÓRGÃO(S) INTERNO(S)/DIVISÃO(ÕES): Instituto Tecnológico de Aeronáutica – ITA			
8. PALAVRAS-CHAVE SUGERIDAS PELO AUTOR: 1. Síntese de alto nível. 2. Decomposição. 3. Bundled-data. 4. Escalonamento.			
9. PALAVRAS-CHAVE RESULTANTES DE INDEXAÇÃO: Circuitos assíncronos; Sistemas digitais; Circuitos lógicos; Algoritmos; VHDL (linguagem de programação); Engenharia elétrica; Engenharia eletrônica.			
10. APRESENTAÇÃO: ITA, São José dos Campos. Curso de Mestrado. Programa de Pós-Graduação em Engenharia Eletrônica e Computação. Área de Dispositivos e Sistemas Eletrônicos. Orientador: Roberto D'Amore; coorientador: Duarte Lopes de Oliveira. Defesa em 09/12/2015. Publicada em 2015.			
11. RESUMO: O projeto de circuitos digitais assíncronos é uma alternativa atraente ao estilo clássico, síncrono, pois, a eliminação do sinal de <i>clock</i> soluciona os problemas relacionados ao <i>clock skew</i> , oferece um potencial ganho em desempenho, redução do consumo de potência além de facilitar o roteamento, visto que a implementação é feita por módulos. Este trabalho propõe um método para geração de circuitos assíncronos no estilo <i>bundled-data</i> , que é constituído de um controlador assíncrono e um <i>datapath</i> que utiliza componentes convencionais síncronos. A otimização proposta para o controlador busca eliminar transições de estado inseridas para satisfazer as propriedades de polaridade dos sinais de controle, mas que não executam nenhuma operação de dados. Essas transições ocorrem frequentemente em laços de repetição, o que gera uma queda expressiva no desempenho do circuito. Para avaliar o método proposto, foi desenvolvida uma ferramenta para a síntese do circuito assíncrono que gera o circuito correspondente descrito em linguagem VHDL a partir de uma descrição textual contendo as entradas, saídas e expressões. Os testes experimentais da ferramenta foram executados em dispositivos FPGAs. Os resultados demonstram a viabilidade do método proposto, pois foi obtida uma sensível melhora no desempenho do circuito quando comparado ao análogo assíncrono sem as otimizações propostas.			
12. GRAU DE SIGILO: (X) OSTENSIVO () RESERVADO () SECRETO			