Universidade Federal do Pampa

Autor: Mohamad El Abd Mohamad Badwan

PROJETO DE UMA UNIDADE LÓGICA ARITMÉTICA

Trabalho de Conclusão de Curso

BAGÉ
2011

MOHAMAD EL ABD MOHAMAD BADWAN

PROJETO DE UMA UNIDADE LÓGICA ARITMÉTICA

Trabalho de conclusão do curso, apresentado ao Curso de Graduação em Engenharia de Computação da Universidade Federal do Pampa, como requisito para a obtenção do Título de Bacharel em Engenharia de Computação.

Orientador: Prof. Dr. Edson Massayuki Kakuno

Co-Orientador: Prof. MSc. Reginaldo da Nóbrega Tavares

Bagé
2011

MOHAMAD EL ABD MOHAMAD BADWAN

PROJETO DE UMA UNIDADE LÓGICA ARITMÉTICA

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Computação.

Trabalho de Conclusão de Curso defendido e aprovado em: 20 de Dezembro de 2011.
Banca examinadora:

____

Prof. Dr. Edson Massayuki Kakuno
Orientador
Licenciatura em Física/Campus Bagé – UNIPAMPA

____

Prof. MSc. Bruno Silveira Neves
Engenharia de Computação/Campus Bagé – UNIPAMPA

____

Prof. Dr. Fabrício de Oliveira Ourique
Engenharia de Computação/Campus Bagé – UNIPAMPA

AGRADECIMENTOS

Agradeço primeiramente a Deus. Aos meus pais, à minha irmã e ao meu irmão, pela compreensão devido ao tempo dedicado aos estudos.

Ao meu orientador Edson Kakuno. E principalmente ao meu co-orientador Reginaldo Tavares que me incentivou no início desse projeto.

A todos meus amigos e aos colegas que compartilharam dos mesmos desafios enfrentados durante a graduação.

Agradeço a todos os professores do curso de Engenharia de Computação que fizeram parte da minha formação. Também agradeço aos demais professores que compartilharam seus conhecimentos durante minha formação.

"Obstáculos foram feitos para serem superados. Para chegar ao êxito precisamos respeitar princípios, regras, condutas e sempre valorizar as pessoas ao nosso redor. Sim, valorizar e respeitar cada pessoa, pois todas têm seu valor e de alguma forma nos ajudam. Respeitando isso e acreditando sempre, conseguimos superar os obstáculos e obter nossas conquistas."

Mohamad Badwan

RESUMO

Neste trabalho é apresentado um estudo de implementação em nível físico do projeto de uma Unidade Lógica Aritmética (ULA) de oito *bits*, usando a estrutura *bit-slice* e utilizando a tecnologia CMOS. Dois modelos de mapeamentos lógicos serão realizados e comparados em termos de área de ocupação do circuito, principalmente em relação ao número de transistores utilizados de cada modelo. A preocupação com a área está relacionada ao custo de fabricação do circuito e ao número de circuitos que podem ser fabricados em um mesmo *wafer* de silício. Adicionalmente a isto, o número de transistores afeta o consumo dinâmico e estático. Este tipo de preocupação é de particular interesse em projetos de dispositivos portáteis que utilizam baterias para o seu uso, pois precisam minimizar o consumo energético. O mapeamento lógico dos circuitos é um aspecto que tem grande impacto na estrutura do circuito. Neste projeto, a implementação inicial da ULA foi realizada com mapeamento para portas NAND. A partir da especificação do projeto para um *slice*, obtiveram-se as equações lógicas iniciais, que foram minimizadas, fatoradas e decompostas. Posteriormente foi feito o mapeamento para portas NAND e em seguida realizado a geração do layout do circuito com o uso da ferramenta MARTELO. Na segunda etapa, foi realizado o mapeamento do circuito para portas complexas, obtendo-se através da ferramenta ASTRAN o *layout* do circuito. Nesse processo de definição física do circuito, foram realizadas otimizações em termos de número de transistores, nos diversos níveis do fluxo do projeto.

Palavras-chave: ULA. Bloco operativo. Projeto de circuitos integrados. *Bit-slice*.

ABSTRACT

This work implements an 8 bits Arithmetic Logic Unit (ALU) using bit-slice structure from the through chip level physical design using CMOS technology. Two types of logical mappings models will be compared in terms of circuit area and number of transistors. Concern about the area is related to the manufacturing cost of the circuit and the number of circuits that can be manufactured on a single silicon wafer. Additionally to this, the number of transistors affects the power drain. This kind of concern is of particular interest in projects of portable devices that operate with batteries. The mapping of logical circuits is one aspect that has great impact on the structure of the circuit. In this project, the initial implementation of the ALU was performed using mapping for NAND gates. From the design specification for one slice, we obtained the initial logic equations, which were minimized, factored and decomposed. Following was done the mapping using NAND gates and then performed the design of the circuit layout using MARTELO software. In the second step was the mapping of circuit for complex gates, getting the layout of the circuit using ASTRAN software. In the process of obtaining the physical circuit, optimizations in number of transistors were done in several levels of project's workflow.

Keywords: ALU. Operating block. Integrated circuit design. Bit-slice.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

AMS – *AustriaMicroSystems*

ASIC – *Application Specific* Integrated Circuits

ASTRAN – *Automatic Synthesis of Transistor Networks*

CAD – *Computer Aided Design*

CMOS – *Complementary Metal Oxide Semiconductor*

DSP – *Digital Signal Processor*

EDA – *Eletronic Design Automation*

MIPS – *Microprocessor without Interlocked Pipeline Stages*

MOS – *Metal Oxide Semiconductor*

NMOS – *N-channel Metal Oxide Semiconductor*

PMOS – *P-channel Metal Oxide Semiconductor*

SOP – *Sum of Products*

ULA – Unidade Lógica Aritmética

SUMÁRIO

1 INTRODUÇÃO

1.1 Estrutura de um Processador

Em sistemas digitais, o processador é responsável pela computação de dados. O processador busca, decodifica e executa instruções sobre um conjunto de dados. As instruções e os dados a serem processados são armazenados na memória. A busca e a decodificação das instruções são realizadas por uma Unidade de Controle, que controla a execução das instruções. Já a execução é desempenhada por uma Unidade Operativa, que é o núcleo responsável pela computação efetiva dos dados. Dentro deste núcleo existe a Unidade Lógica Aritmética (ULA), que é responsável por executar as operações lógicas e aritméticas (STALLINGS, 2002). O processador também contém registradores internos para armazenamento temporário de dados durante a computação.

A comunicação entre o processador, memória e módulos de entrada e saída é feita através de barramentos do sistema, que são três: dados, endereço e controle, conforme Figura 1.1.

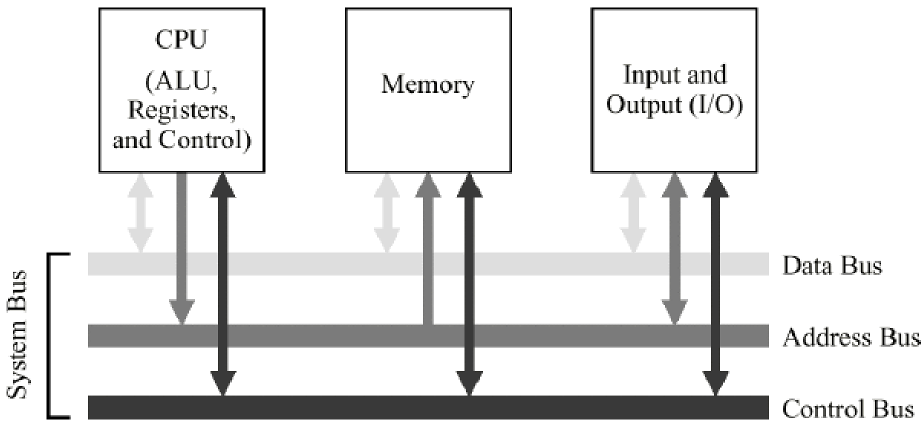

FIGURA 1.1 – Modelo de arquitetura de um sistema
Fonte: Murdoca (2000, p.93)

A operação a ser executada pela ULA é definida pelo sinal de controle vindo da Unidade de Controle, que também define quais são os registradores que fornecem os dados de entrada para a ULA e o registrador que armazenará o resultado da operação (MURDOCA; HEURING, 2000).

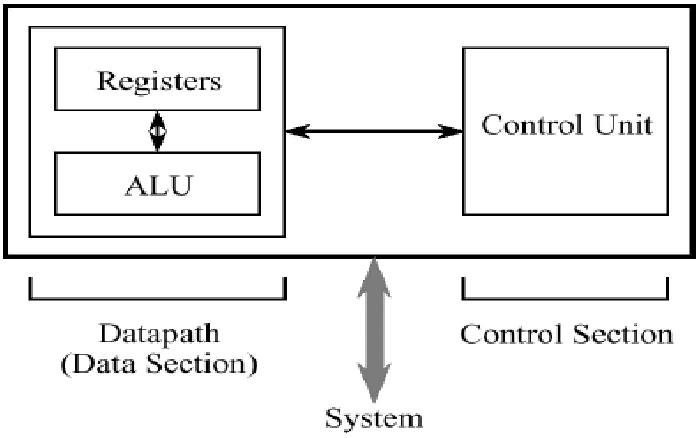

FIGURA 1.2 – Visão geral de um processador
Fonte: Murdoca (2000, p. 97)

1.2 Operações de uma ULA

Cada processador possui o seu conjunto de instruções e para executá-las, a ULA deve ser capaz de oferecer operações que dão suporte a maioria dessas instruções.

Uma ULA pode realizar operações lógicas e aritméticas. As operações dependem da concepção da ULA. Uma operação lógica ou aritmética é configurada durante a execução de uma instrução.

A Tabela 1.1 demonstra as operações da ULA de alguns processadores.

TABELA 1.1 – Comparativo entre operações de ULA's de processadores tradicionais

80x86	ARM	MIPS	SPARC
		Soma	Soma
Soma	Soma	Subtração	Subtração
Subtração	Subtração	And	Multiplicação
Multiplicação	Multiplicação	Or	Divisão
Divisão	Divisão	Not	And
And	And	Shift Right	Or
Or	Or	Shift Left	Xor
Xor	Xor	Igual	Not
Not	Not	Diferente	Shift Right
		Menor	Shift Left

Neste contexto, existem arquiteturas que possuem pseudo-instruções, que são instruções que não são implementadas por uma operação específica da ULA e sim modificadas para outra instrução com operação existente. Um exemplo de processador que possui pseudo-instruções é o MIPS (PATTERSON, 2005).

Dentre as operações aritméticas, a soma tem um papel muito importante, sendo utilizada em aplicações como multiplicadores, processadores digitais de sinais (DSPs), etc. O esforço de projetistas no desenvolvimento de somadores tem se intensificado, buscando otimizar os parâmetros de projeto (DEVI; GIRDHER; SINGH, 2010). Estes parâmetros podem ser definidos como os requisitos básicos para qualquer projeto em *hardware* de um sistema digital, dentre eles, os principais são: alta velocidade de operação, área física reduzida e baixo consumo de energia do circuito (SCHLOSSER, 2009).

1.3 Implementações de ULA's

Uma ULA pode ser implementada de diversas maneiras. Assim cada projeto apresenta diferentes características, ou seja, o *hardware* obtido apresenta características distintas em relação à velocidade de processamento, à área (dimensão física e número de transistores), ao consumo energético e à regularidade de cada circuito gerado. Ou seja, o desempenho de um circuito é altamente dependente da descrição estrutural. No entanto, encontrar a melhor arquitetura de um circuito, que alie o melhor desempenho dos parâmetros de projeto, não é um problema trivial (VERMA; BRISK; IENNE, 2009). Tendo em vista que os objetivos do projeto podem ser distintos, as descrições arquiteturais podem ser desenvolvidas com o objetivo de otimizar um ou mais parâmetros de projeto, dependendo da aplicação.

1.3.1 Estrutura *Bit-Slice*

Uma maneira de implementar os blocos da ULA é usando estruturas chamadas *bit-slice*. Essa estrutura consiste no projeto do circuito para apenas um *bit*, bastando replicar a estrutura para a quantidade de bits necessários (RABAEY, 1996). Esse método tem como característica a obtenção de um *layout* mais compacto e com a possibilidade de ter boas características de desempenho, pelo fato de possuir proximidade entre os módulos, que implica em área reduzida, conexões próximas e por conseqüência diminuição de consumo, além da diminuição do tempo de projeto devido à regularidade, que significa a padronização nos aspectos de projeto (MEINHARDT, 2006). Na Figura 1.3 é ilustrado um exemplo de estrutura *bit-slice*.

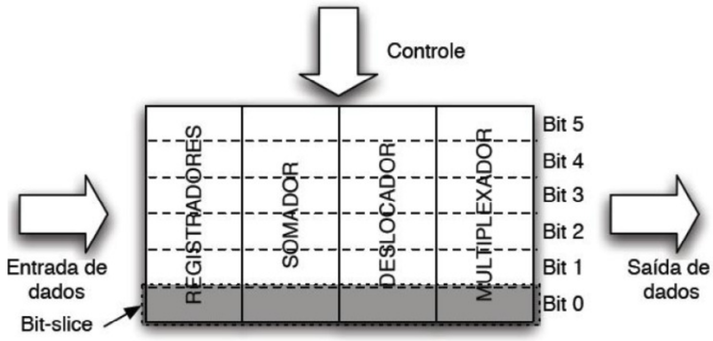

FIGURA 1.3 – Exemplo de uma estrutura *bit-slice*
Fonte: Rabaey (1996, p. 561)

1.3.2 Forma alternativa de implementar uma ULA

Outra maneira de implementar, seria projetar individualmente as operações da ULA para o número de *bits* necessários, acoplando um multiplexador entre as operações. Neste procedimento todas as operações são executadas paralelamente, e através de um sinal de controle, apenas o sinal da operação selecionada é levado em consideração na(s) saída(s) do circuito. Como vantagem dessa implementação, seria um ganho em velocidade de processamento e como desvantagem seria um circuito mais complexo, tornando mais trabalhoso e demorado o processo de depuração do circuito.

1.4 Fluxo de Projeto

Para implementar *hardware*, antes é necessário obter uma descrição física do circuito a partir de uma descrição lógica do mesmo. Esta transformação de uma descrição para outra é obtida através de um fluxo de projeto, que consiste em produzir o *layout* do circuito a partir de um *netlist* de portas lógicas, satisfazendo a uma série de restrições (WESTE; HARRIS, 2011).

O fluxo básico de um projeto físico de qualquer sistema digital consiste de uma descrição lógica, passando por uma etapa de síntese lógica, uma de síntese física, chegando a uma descrição do *layout* físico do circuito. Conforme diagrama apresentado na Figura 1.4.

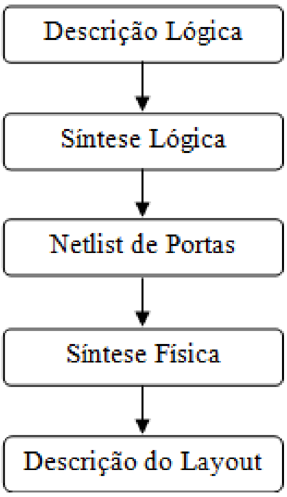

FIGURA 1.4 – Fluxo básico de um projeto físico

A descrição lógica consiste das equações lógicas obtidas a partir da especificação do projeto. O estágio de síntese lógica consiste em um conjunto de transformações das equações, tendo em vista produzir um *netlist* de portas lógicas que serão utilizadas na implementação física. O *netlist* corresponde às portas lógicas e às conexões pertencentes ao circuito. A síntese física consiste em obter a descrição do *layout* físico, que pode ser fornecida ao fabricante para a produção do circuito.

Na etapa de síntese lógica existe o mapeamento tecnológico, que é responsável por implementar o circuito na metodologia desejada, utilizando por exemplo blocos lógicos programáveis, biblioteca de células, geração automática de células e outros (LIMA, 1999). O mapeamento tecnológico pode ser uma das etapas mais críticas e de maior impacto no resultado final do desempenho do sistema em *hardware*, uma vez que ele define a construção do circuito e o desempenho, bem como o consumo de potência (MARQUES et al., 2009).

Nesse contexto, também podem ser feitos mapeamentos para portas simples e complexas. Portas simples são portas lógicas que implementam as funções básicas da álgebra booleana, de modo que na tecnologia CMOS (*Complementary Metal Oxide Semiconductor*) se restringe as portas NAND, NOR e inversor. Portas complexas são portas lógicas capazes de implementar funções com maior complexidade (SANTOS, 2005).

Na síntese física, uma sequência de etapas deve ser realizada, que são planejamento topológico (*floorplanning*), posicionamento, geração do *layout* e roteamento. No planejamento topológico é feita a alocação de área, posicionamento de sub-circuitos, planejamento das trilhas de alimentação e posicionamento dos pinos de entrada e saída do circuito; no posicionamento reserva-se um espaço físico para cada célula dentro da área destinada a cada sub-circuito; na etapa de geração e roteamento, o *layout* do circuito é gerado, juntamente com a realização das interconexões entre os componentes. Além de estas etapas alterarem os parâmetros de projeto, elas interferem nas etapas seguintes (SANTOS, 2005).

Nesta fase de síntese, pode-se explorar o espaço de projeto em relação ao estilo de *layout* e obter otimizações em termos de desempenho (MORAES et al., 1998). Um meio de se obter a descrição física é a partir de ferramentas de geração automática de síntese física de circuitos integrados, com isso obtém-se o *layout*, que pode ser utilizado para a fabricação do circuito integrado. Neste nível, a ferramenta deve respeitar regras de construção de *layout*, que são denominadas regras de projeto. Estas são definidas por um processo de fabricação e determinam principalmente as distâncias mínimas entre os componentes, assim como as dimensões mínimas dos mesmos (REIS, 2002).

1.5 Tecnologia CMOS

A tecnologia CMOS (*Complementary Metal Oxide Semiconductor*) utiliza transistores MOS na configuração complementar, i.e. a mínima unidade lógica é obtida através da conexão de um transistor MOS tipo N com outro tipo P. Esta combinação pode ser implementada em um único substrato, facilitando a integração de um sistema como todo.

Os transistores MOS são dispositivos semicondutores muito eficientes quando empregados em circuitos de lógica digital, pois funcionam como uma chave lógica, permitindo ou não a passagem de corrente elétrica de acordo com o sinal de entrada recebido (BOYLESTAD; NASHELSKY, 1998). Em circuitos integrados de aplicação específica (ASICs) cada transistor do projeto é levado em consideração no processo de fabricação. Neste contexto, o avanço da tecnologia, ou seja, a diminuição física da dimensão dos transistores, e a maior capacidade de integração destes numa mesma pastilha de silício (*chip*), permitiu um aumento no poder de processamento, uma diminuição no tamanho dos circuitos e um baixo consumo de energia nos circuitos integrados.

No projeto de circuitos integrados lógicos, a tecnologia CMOS é a que predomina. Esta é tema de muito estudo na área de computação, tendo em vista que a transição para novas tecnologias não vai ser imediata e, portanto se prevê sua utilização para muitas décadas. Em termos de energia, circuitos com tecnologia CMOS consomem somente em transições de estados, porém devido à diminuição dos componentes existe a preocupação devido ao consumo estático causado por correntes de fuga. Por isso, deve-se buscar otimizações em todos os níveis do fluxo, já que, além da redução do número de transições de estado dos transistores em um sistema, é cada vez mais importante uma redução no número de transistores (REIS, 2010).

1.6 Ferramentas de Projeto

No projeto de qualquer sistema digital se busca o melhor desempenho possível, com o menor custo de fabricação e um menor tempo de lançamento no mercado. O tempo de mercado se refere ao momento em que o produto estará disponível, este não pode ser elevado, pois pode enfrentar concorrência e sua margem de lucro deverá reduzir para ser competitivo. Para isto o tempo de projeto, que se refere ao tempo de sua idealização até a fabricação, também deve ser baixo.

Diante deste cenário, faz-se necessária cada vez mais a utilização de softwares de projeto, as chamadas ferramentas de EDA (*Eletronic Design Automation*), que são capazes de lidar com o projeto de circuitos integrados com um elevado número de componentes em um reduzido tempo. Outro fator que influencia a utilização de ferramentas é a complexidade de sistemas com alta escala de integração, pois podem conter centenas de milhares de transistores, inviabilizando que este tipo de projeto seja realizado manualmente (REIS; MORAES, 2000). Tradicionalmente, ferramentas têm sido usadas independentemente nos diversos níveis, de modo que em cada nível se tem um conjunto próprio de síntese, verificação e análise (CONG; SARRAFZADEH, 2000).

1.7 Finalização do Projeto

Com o *layout* pronto, é necessário fazer simulações elétricas para ser possível estimar atrasos, potência e ruídos que o circuito terá após sua fabricação. Caso algum problema seja detectado é necessário voltar às etapas iniciais do projeto para ser feita a correção. Terminando esse processo o circuito é considerado validado.

Após essa etapa já se poderia levar para um fabricante de circuitos integrados, que realizará a construção física do circuito projetado. No processo de fabricação o *layout* será transformado em máscaras que serão projetadas em uma lâmina de silício. O *layout* pode ter vários níveis, de modo que as máscaras representam os padrões geométricos de cada um desses níveis, que devem ser criados sobre as superfícies da lâmina. Desta forma, o *chip* vai sendo construído através de uma seqüencia de passos baseados em processos físico-químicos (REIS, 2002).

O custo de produção é dado por lâmina, assim quanto mais circuitos projetado na lâmina, menor será o custo de produção do circuito. Entre os custos de fabricação, tem-se o próprio processo de construção do circuito e adicionalmente tem os custos de teste elétrico, montagem, encapsulamento e testes (RABAEY, 1996).

2 DESCRIÇÃO DA UNIDADE LÓGICA ARITMÉTICA

2.1 Especificação da Unidade Lógica Aritmética

Com o objetivo de projetar uma ULA para aplicações de propósito geral, foi determinado que a implementação fosse para oito bits, de números inteiros não sinalizados, usando estrutura *bit-slice*. Dentre as operações típicas que uma ULA executa, foi escolhido um subconjunto de operações, que são:

- Soma
- Subtração
- Deslocamento à direita
- Deslocamento à esquerda
- Operação AND
- Operação OR
- Operação XOR
- Operação NOT
- Comparação (Maior, Igual e Menor)

2.2 Desenvolvimento Lógico de um *Slice* da ULA

Com o propósito de dimensionar o número de *bits* de controle da ULA, fator este que está relacionado com o número de operações, usa-se a seguinte relação:

$$2^n = \mathrm{op} \tag{2.1}$$

Onde:
n: representa o número de *bits* necessários, de modo que este deve ser um inteiro;
op: representa o número de operações da ULA.

Cada operação corresponde a uma combinação binária pré-definida na unidade de controle.

Tendo em vista, que neste projeto a ULA possui nove operações, são necessários quatro *bits* de controle.

O conjunto dos bits de controle, descritos como *c3*, *c2*, *c1* e *c0*, representa o código de cada operação. Desta forma, foram convencionados os códigos de cada operação, conforme Tabela 2.1.

TABELA 2.1 – Código das operações da ULA

c3	c2	c1	c0	Operação
0	0	0	0	Soma
0	0	0	1	Subtração
0	0	1	0	Deslocamento à direita
0	0	1	1	Deslocamento à esquerda
0	1	0	0	And
0	1	0	1	Or
0	1	1	0	Xor
0	1	1	1	Not
1	0	0	0	Comparação

2.3 Implementação das Operações da ULA para um *Slice*

2.3.1 Soma

Neste trabalho, esta operação consiste em somar duas entradas binárias de oito *bits* não sinalizados. O circuito somador fornece como resultado uma saída binária, e sinais de *carry-out* são propagados entre os *slices* que compõem a ULA (WAGNER, REIS e RIBAS, 2008).

Para um *slice*, a soma será implementada através de um somador completo, para isto, além dos operandos de entrada, são necessários *bits* adicionais (*carry-in* e *carry-out*). Estes sinais foram denominados *cin* e *cout*, respectivamente. O comportamento das saídas em função dos valores de entrada é mostrado na Tabela 2.2.

TABELA 2.2 – Tabela verdade para Soma

cin	e0	e1	s	cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Para a soma e demais operações, os sinais dos operandos de entrada são denominados *e0* e *e1*, já o sinal que representa o resultado foi denominado *s*.

2.3.2 Subtração

Este trabalho apresenta um modelo de um circuito subtrator completo. Nesse processo, a subtração é feita subtraindo uma entrada binária de outra entrada. O circuito subtrator fornece uma saída binária que representa o resultado da subtração e entre os *slices* que compõem a ULA é propagado um sinal (WAGNER, REIS e RIBAS, 2008).

Para um *slice*, esta operação é implementada subtraindo o operando *e1* do operando *e0*. Porém torna-se necessário pedir emprestado um dígito quando o operando *e1* for maior que *e0* ou quando um empréstimo foi feito no estágio anterior e não pode ser concluído no estágio atual, desta forma o pedido é propagado. Para poder sinalizar estas situações, adiciona-se um *bit* de entrada, que representa um empréstimo de um dígito feito no *slice* anterior, este *bit* foi denominado *emp1*. Outro *bit* de saída, denominado *pede1*, é adicionado com a função de indicar o pedido de empréstimo. A Tabela 2.3 descreve o comportamento da subtração em função dos valores de entrada.

TABELA 2.3 – Tabela verdade para Subtração

emp1	e0	e1	s	pede1
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	1

2.3.3 Deslocamento à Direita

Este circuito permite movimentar em um *bit* uma das entradas binárias para as saídas adjacentes com índice inferior, de modo que uma das entradas é desconsiderada.

Esta operação depende do *slice* de maior magnitude que está conectado ao *slice* analisado. Portanto o valor lógico da entrada *e1* do *slice* adjacente de maior magnitude é deslocado para a saída do *slice*. Para isso, foi necessário inserir um sinal de entrada no *slice*. Este sinal foi denominado *dir*. Neste caso a entrada *e0* foi desconsiderada.

TABELA 2.4 – Tabela verdade para Deslocamento à Direita

dir	s
0	0
1	1

2.3.4 Deslocamento à Esquerda

É semelhante ao deslocamento à direita, porém as entradas serão enviadas para as saídas de índice superior. O *slice* é conectado ao *slice* adjacente de menor magnitude. Um sinal de entrada também é inserido, este sinal foi denominado *esq*.

TABELA 2.5 – Tabela verdade para Deslocamento à Esquerda

esq	s
0	0
1	1

2.3.5 Operação AND

O circuito AND insere nas saídas um sinal alto quando as respectivas entradas tiverem nível lógico alto. Este processo é feito *bit* a *bit*.

Para um *slice*, a operação é executada sobre as entradas *e0* e *e1*, gerando o sinal de saída, conforme Tabela 2.6.

TABELA 2.6 – Tabela verdade para AND

e0	e1	s
0	0	0
0	1	0
1	0	0
1	1	1

2.3.6 Operação OR

O circuito OR insere nas saídas um sinal alto quando uma de suas respectivas entradas tiver sinal alto. Também é feito *bit* a *bit*.

Para um *slice*, executa a operação OR das entradas, dependendo apenas dos sinais *e0* e *e1* para gerar o sinal de saída, conforme Tabela 2.7.

TABELA 2.7 – Tabela verdade para OR

e0	e1	s
0	0	0
0	1	1
1	0	1
1	1	1

2.3.7 Operação XOR

O circuito XOR insere nas saídas um sinal alto quando suas respectivas entradas não possuírem sinais iguais. A Tabela 2.8 mostra a operação XOR para um *slice*.

TABELA 2.8 – Tabela verdade para XOR

e0	e1	s
0	0	0
0	1	1
1	0	1
1	1	0

2.3.8 Operação NOT

Neste circuito, o objetivo é inverter o estado lógico das entradas, inserindo nos oito *bits* de saída, os sinais invertidos de uma de suas respectivas entradas. Dessa forma, a outra entrada é desconsiderada.

Para um *slice*, a operação NOT gera o sinal de saída em função de apenas uma entrada. A entrada *e0* foi escolhida para esta operação, enquanto a entrada *e1* não é levada em consideração. O resultado da operação é levado à saída *s*, conforme mostrado na Tabela 2.9.

TABELA 2.9 – Tabela verdade para NOT

e0	s
0	1
1	0

2.3.9 Comparação

Utilizado em várias arquiteturas para instruções de condição. Neste projeto, o comparador compara duas entradas binárias e utiliza três sinais de saída para indicar se uma entrada é maior, igual ou menor que a outra.

A comparação tem como função definir se o conjunto de *bits* da entrada *e0* é maior, igual ou menor que o conjunto de *bits* da entrada *e1*. Nesse contexto, a verificação dessa condição na saída do *slice* depende da situação dos *slices* anteriores.

Nessa etapa, foi necessária a adição de *bits* de entrada, com a finalidade de indicar a situação do estado do *slice* anterior. Estes *bits* foram denominados de *maior_in*, *igual_in* e *menor_in*. Adicionalmente a isto, também foi necessário indicar o estado da comparação com três *bits* de saída, denominados de *maior_out*, *igual_out* e *menor_out*.

O comportamento da operação de comparação em função dos sinais de entrada pode ser verificado na Tabela 2.10.

TABELA 2.10 – Tabela verdade para Comparação

maior_in	igual_in	menor_in	e0	e1	maior_out	igual_out	menor_out
0	0	1	0	0	0	0	1
0	0	1	0	1	0	0	1
0	0	1	1	0	1	0	0
0	0	1	1	1	0	0	1
0	1	0	0	0	0	1	0
0	1	0	0	1	0	0	1
0	1	0	1	0	1	0	0
0	1	0	1	1	0	1	0
1	0	0	0	0	1	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	1	0	0
1	0	0	1	1	1	0	0

2.4 Estrutura do *Slice* da ULA

De acordo com os sinais necessários para atender a cada operação da ULA, o bloco que representa cada *slice* da ULA pode ser simbolizado conforme a Figura 2.1. Cada *slice* da ULA é composto por 13 *bits* de entrada e 6 *bits* de saída.

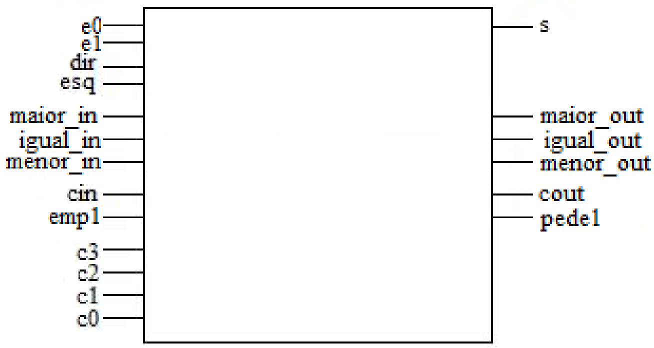

FIGURA 2.1 – Representação dos sinais de um *Slice* da ULA

A composição dos *bits* de entrada é dada da seguinte forma:

- *c3*, *c2*, *c1* e *c0*: são sinais que representam o código da operação a ser executada pela ULA;
- *maior_in*, *igual_in* e *menor_in*: são sinais que representam o estado dos estágios anteriores da comparação;
- *esq*: este sinal representa o valor vindo do *slice* adjacente de menor magnitude que será levado para a saída do *slice* quando for selecionada a operação de deslocamento à esquerda;
- *dir*: semelhante ao sinal anterior, porém o sinal de origem vem do *slice* adjacente de maior magnitude e quando a operação selecionada for o deslocamento à direita;
- *emp1*: sinal que representa o empréstimo de um *bit* realizado na subtração;
- *cin*: sinal que representa o estouro de representação do estágio anterior realizado na soma;
- *e0* e *e1*: representam os operandos de entrada.

Em relação aos *bits* de saída:

- *s*: este sinal representa o resultado das operações lógicas e aritméticas;
- *cout*: representa o sinal de propagação da soma;
- *pede1*: representa o sinal de pedido de um *bit* realizado na subtração;
- *maior_out*, *igual_out* e *menor_out*: são sinais que representam o estado atual da comparação das entradas.

2.5 Descrição Lógica

A partir das tabelas-verdade das operações, foi montada a tabela geral de um *slice* da ULA, que representa o comportamento das saídas em função dos *bits* de entrada. Esta descrição pode ser visualizada na Tabela 2.11:

TABELA 2.11 – Tabela verdade de um *Slice* do circuito

***Bits* de Entrada**													***Bits* de Saída**					
c3	**c2**	**c1**	**c0**	**maior_in**	**igual_in**	**menor_in**	**esq**	**dir**	**emp1**	**cin**	**e0**	**e1**	**s**	**cout**	**pede1**	**maior_out**	**igual_out**	**menor_out**
0	0	0	0	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0
0	0	0	0	-	-	-	-	-	-	0	0	1	1	0	0	0	0	0
0	0	0	0	-	-	-	-	-	-	0	1	0	1	0	0	0	0	0
0	0	0	0	-	-	-	-	-	-	0	1	1	0	1	0	0	0	0
0	0	0	0	-	-	-	-	-	-	1	0	0	1	0	0	0	0	0
0	0	0	0	-	-	-	-	-	-	1	0	1	0	1	0	0	0	0
0	0	0	0	-	-	-	-	-	-	1	1	0	0	1	0	0	0	0
0	0	0	0	-	-	-	-	-	-	1	1	1	1	1	0	0	0	0
0	0	0	1	-	-	-	-	-	0	-	0	0	0	0	0	0	0	0
0	0	0	1	-	-	-	-	-	0	-	0	1	1	0	1	0	0	0
0	0	0	1	-	-	-	-	-	0	-	1	0	1	0	0	0	0	0
0	0	0	1	-	-	-	-	-	0	-	1	1	0	0	0	0	0	0
0	0	0	1	-	-	-	-	-	1	-	0	0	1	0	1	0	0	0
0	0	0	1	-	-	-	-	-	1	-	0	1	0	0	1	0	0	0
0	0	0	1	-	-	-	-	-	1	-	1	0	0	0	0	0	0	0
0	0	0	1	-	-	-	-	-	1	-	1	1	1	0	1	0	0	0
0	0	1	0	-	-	-	-	0	-	-	-	-	0	0	0	0	0	0
0	0	1	0	-	-	-	-	1	-	-	-	-	1	0	0	0	0	0
0	0	1	1	-	-	-	0	-	-	-	-	-	0	0	0	0	0	0
0	0	1	1	-	-	-	1	-	-	-	-	-	1	0	0	0	0	0
0	1	0	0	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0
0	1	0	0	-	-	-	-	-	-	-	0	1	0	0	0	0	0	0
0	1	0	0	-	-	-	-	-	-	-	1	0	0	0	0	0	0	0
0	1	0	0	-	-	-	-	-	-	-	1	1	1	0	0	0	0	0
0	1	0	1	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0
0	1	0	1	-	-	-	-	-	-	-	0	1	1	0	0	0	0	0
0	1	0	1	-	-	-	-	-	-	-	1	0	1	0	0	0	0	0
0	1	0	1	-	-	-	-	-	-	-	1	1	1	0	0	0	0	0
0	1	1	0	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0
0	1	1	0	-	-	-	-	-	-	-	0	1	1	0	0	0	0	0
0	1	1	0	-	-	-	-	-	-	-	1	0	1	0	0	0	0	0
0	1	1	0	-	-	-	-	-	-	-	1	1	0	0	0	0	0	0
0	1	1	1	-	-	-	-	-	-	-	0	-	1	0	0	0	0	0
0	1	1	1	-	-	-	-	-	-	-	1	-	0	0	0	0	0	0
1	0	0	0	0	0	1	-	-	-	-	0	0	0	0	0	0	0	1
1	0	0	0	0	0	1	-	-	-	-	0	1	0	0	0	0	0	1
1	0	0	0	0	0	1	-	-	-	-	1	0	0	0	0	1	0	0
1	0	0	0	0	0	1	-	-	-	-	1	1	0	0	0	0	0	1
1	0	0	0	0	1	0	-	-	-	-	0	0	0	0	0	0	1	0
1	0	0	0	0	1	0	-	-	-	-	0	1	0	0	0	0	0	1
1	0	0	0	0	1	0	-	-	-	-	1	0	0	0	0	1	0	0
1	0	0	0	0	1	0	-	-	-	-	1	1	0	0	0	0	1	0
1	0	0	0	1	0	0	-	-	-	-	0	0	0	0	0	1	0	0
1	0	0	0	1	0	0	-	-	-	-	0	1	0	0	0	0	0	1
1	0	0	0	1	0	0	-	-	-	-	1	0	0	0	0	1	0	0
1	0	0	0	1	0	0	-	-	-	-	1	1	0	0	0	1	0	0

O símbolo “-“ mostrado na tabela verdade corresponde a um *don’t care*, que significa que este estado lógico não interfere o resultado de saída das funções.

2.6 Minimização Lógica

De uma forma geral, as informações de tabelas-verdade não proporcionam a forma mais eficiente de implementação, pois estas podem ter informações redundantes.

Para resolver o problema de redundância, é necessário minimizar as equações lógicas. O processo de minimização lógica consiste em obter a menor representação das equações lógicas de saída que descrevam o comportamento da tabela verdade. Para isso existem alguns métodos, como: minimização através de álgebra booleana, mapas de Karnaugh, método de Quine-McCluskey, método espresso.

Como metodologia para realizar a minimização lógica, foi utilizada a ferramenta Espresso (DE MICHELI, 1994). Essa ferramenta permite a transformação de uma representação de uma ou mais funções booleanas em outra representação minimizada equivalente. Essa ferramenta usa algoritmos e técnicas de minimização para obter uma representação de dois níveis das funções de saída (KATZ; BORRIELLO, 2005).

A descrição de entrada do Espresso é representada por uma tabela verdade codificada, que corresponde à tabela verdade original, conforme Apêndice A. A saída da ferramenta corresponde a uma tabela verdade minimizada, conforme Apêndice B.

2.6.1 Soma de Produtos (SOP)

A partir do resultado da ferramenta Espresso, cada função de saída pode ser representada em dois níveis de portas lógicas. Neste trabalho foi utilizada a representação na forma de soma de produtos, de modo que a função no primeiro nível compreende portas AND e no segundo nível portas OR.

Este procedimento é usualmente eficiente em circuitos com poucas variáveis de entrada, pois considerando a tecnologia CMOS, à medida que se aumenta o número de entradas em uma porta lógica, a porta se torna lenta. Isto ocorre, pelo fato que conexões em série aumentam a resistência da porta e por conseqüência geram o resultado em um maior tempo. O número de entradas da porta é usualmente denominado por *fan-in*. (BROWN; VRANESIC, 2004)

2.6.2 Equações Minimizadas do *Slice* da ULA

Para melhor escrever as equações lógicas, as descrições das entradas foram mapeadas para letras.

TABELA 2.12 – Mapeamento das descrições das entradas em letras

Descrição da entrada	Letra
c3	n
c2	m
c1	l
c0	j
maior_in	i
igual_in	h
menor_in	g
esq	f
dir	e
emp1	d
cin	c
e0	b
e1	a

Com o resultado obtido através da ferramenta foram extraídas as equações lógicas das funções de saída do circuito na forma de soma de produtos. As expressões são mostradas abaixo:

$$s = \bar{n}ml\bar{j}b\bar{a} + \bar{n}\bar{m}\bar{l}jd\bar{b}\bar{a} + \bar{n}\bar{l}j\bar{d}\bar{b}a + \bar{n}\bar{l}j\bar{d}b\bar{a} + \bar{n}\bar{m}\bar{l}jdba + \bar{n}\bar{m}\bar{l}\bar{j}\,\bar{c}b\bar{a} + \bar{n}\bar{m}\bar{l}\bar{j}c\bar{b}\bar{a} + \bar{n}mlj\bar{b} + \bar{n}\bar{m}\bar{l}\bar{j}\,\bar{c}\bar{b}a + \bar{n}ml\bar{b}a + \bar{n}\bar{l}\bar{j}cba + \bar{n}\bar{m}ljf + \bar{n}mj\bar{b}a + \bar{n}\bar{m}l\bar{j}e + \bar{n}m\bar{l}jb + \bar{n}m\bar{l}ba \qquad (2.2)$$

$$cout = \bar{n}\bar{m}\bar{l}\bar{j}cb + \bar{n}\bar{m}\bar{l}jca + \bar{n}\bar{m}\bar{l}\bar{j}ba \qquad (2.3)$$

$$\mathrm{pede1} = \bar{n}\bar{m}\bar{l}jd\bar{b}\bar{a} + \bar{n}\bar{m}\bar{l}jdba + \bar{n}\bar{m}\bar{l}j\bar{b}a \qquad (2.4)$$

$$\mathrm{maior\_out} = n\bar{m}\bar{l}\bar{j}\,\bar{i}\bar{h}gb\bar{a} + n\bar{m}\bar{l}\bar{j}\,\bar{i}h\bar{g}b\bar{a} + n\bar{m}\bar{l}\bar{j}i\bar{h}\bar{g}\bar{a} + n\bar{m}\bar{l}\bar{j}i\bar{h}\bar{g}b \qquad (2.5)$$

$$\mathrm{igual\_out} = n\bar{m}\bar{l}\bar{j}\,\bar{i}h\bar{g}\bar{b}\bar{a} + n\bar{m}\bar{l}\bar{j}\,\bar{i}h\bar{g}ba \qquad (2.6)$$

$$\mathrm{menor\_out} = n\bar{m}\bar{l}\bar{j}\,i\bar{h}\bar{g}\bar{b}a + n\bar{m}\bar{l}\bar{j}\,\bar{i}h\bar{g}\bar{b}a + n\bar{m}\bar{l}\bar{j}\,\bar{i}\bar{h}g\bar{b} + n\bar{m}\bar{l}\bar{j}\,\bar{i}\bar{h}ga \qquad (2.7)$$

2.7 Fatoração

Para contornar o problema do *fan-in*, é necessário transformar o circuito em vários níveis. A fatoração é uma técnica que permite transformar o circuito em multinível. Essa técnica permite várias alternativas na implementação final do projeto, o que impactua bastante no *hardware* do circuito.

A área ocupada por um circuito que implementa uma função lógica dentro de um circuito integrado é composta por transistores e por fios que fazem as conexões entre os transistores. Em uma expressão lógica cada literal corresponde a um fio no circuito que transportará um sinal lógico. A fatoração reduz o número de literais, o que proporciona uma redução na complexidade da fiação em um circuito integrado. Nesse processo de síntese, as ferramentas CAD (*Computer Aided Design*) fornecem muitos caminhos, incluindo o custo do circuito, o *fan-in* e a complexidade da fiação (BROWN; VRANESIC, 2004).

A partir das SOPs obtidas foi realizada manualmente a fatoração das funções. Juntamente com esse procedimento, foi objetivado ter um bom nível de compartilhamento entre todas as funções de saída, o que pode reduzir a área do circuito final.

O compartilhamento foi feito unindo todos os produtos das funções de saída. A partir desse ponto, buscou-se verificar sub-termos dos produtos que fossem comuns, pois desta forma o compartilhamento de *hardware* não se limita a apenas ao circuito de cada função, mas entre todas as funções de saída do circuito.

Entre os sub-termos escolhidos, foi procurado primeiramente agrupar os literais que correspondiam os sinais de controle *n*, *m*, *l* e *j*, pois estes se apresentam em quase todos os produtos. Também foi escolhido deixar os literais *b* e *a* agrupados, já que se apresentam na maioria dos produtos. Na fatoração, a maioria dos agrupamentos foi realizada com dois literais.

As expressões fatoradas são mostradas abaixo:

$$\mathrm{s} = \bar{\mathrm{n}}\bar{\mathrm{m}}(\bar{\mathrm{l}}\bar{\mathrm{j}}(\bar{\mathrm{c}}(\mathrm{b}\bar{\mathrm{a}} + \bar{\mathrm{b}}\mathrm{a}) + \mathrm{c}(\bar{\mathrm{b}}\bar{\mathrm{a}})) + \bar{\mathrm{l}}\mathrm{j}(\mathrm{d}(\bar{\mathrm{b}}\bar{\mathrm{a}} + \mathrm{ba})) + \mathrm{ljf} + \mathrm{l}\bar{\mathrm{j}}\mathrm{e}) \\ + \bar{\mathrm{n}}\mathrm{m}(\mathrm{l}\bar{\mathrm{j}}\mathrm{b}\bar{\mathrm{a}} + \mathrm{lj}\bar{\mathrm{b}} + \bar{\mathrm{b}}\mathrm{a}(\mathrm{l} + \mathrm{j}) + \bar{\mathrm{l}}\mathrm{jb} + \bar{\mathrm{l}}\mathrm{ba}) + \bar{\mathrm{n}}\bar{\mathrm{l}}(\mathrm{j}\bar{\mathrm{d}}(\bar{\mathrm{b}}\mathrm{a} + \mathrm{b}\bar{\mathrm{a}}) + \bar{\mathrm{j}}\mathrm{cba}) \tag{2.8}$$

$$\mathrm{cout} = \bar{\mathrm{n}}\bar{\mathrm{m}}(\bar{\mathrm{l}}\bar{\mathrm{j}}(\mathrm{c}(\mathrm{b} + \mathrm{a}) + \mathrm{ba})) \tag{2.9}$$

$$\mathrm{pede1} = \bar{\mathrm{n}}\bar{\mathrm{m}}(\bar{\mathrm{l}}\mathrm{j}(\mathrm{d}(\bar{\mathrm{b}}\bar{\mathrm{a}} + \mathrm{ba}) + \bar{\mathrm{b}}\mathrm{a})) \tag{2.10}$$

$$\mathrm{maior\_out} = \mathrm{n}\bar{\mathrm{m}}\bar{\mathrm{l}}\bar{\mathrm{j}}(\bar{\mathrm{i}}(\bar{\mathrm{h}}\mathrm{g}(\mathrm{b}\bar{\mathrm{a}}) + \mathrm{h}\bar{\mathrm{g}}(\mathrm{b}\bar{\mathrm{a}})) + \mathrm{i}(\bar{\mathrm{h}}\bar{\mathrm{g}}(\bar{\mathrm{a}} + \mathrm{b}))) \tag{2.11}$$

$$\mathrm{igual\_out} = \mathrm{n}\bar{\mathrm{m}}\bar{\mathrm{l}}\bar{\mathrm{j}}(\bar{\mathrm{i}}(\mathrm{h}\bar{\mathrm{g}}(\bar{\mathrm{b}}\bar{\mathrm{a}} + \mathrm{ba}))) \tag{2.12}$$

$$\mathrm{menor\_out} = \mathrm{n}\bar{\mathrm{m}}\bar{\mathrm{l}}\bar{\mathrm{j}}(\bar{\mathrm{i}}(\bar{\mathrm{h}}\mathrm{g}(\bar{\mathrm{b}} + \mathrm{a}) + \mathrm{h}\bar{\mathrm{g}}(\bar{\mathrm{b}}\mathrm{a})) + \mathrm{i}(\bar{\mathrm{h}}\bar{\mathrm{g}}(\bar{\mathrm{b}}\mathrm{a}))) \tag{2.13}$$

2.8 Decomposição

A decomposição é uma técnica para reduzir a complexidade do circuito em termos de fiação e portas lógicas. Nesse processo, a função é decomposta em funções menores, que representam partes do circuito. Desse modo, esses subcircuitos podem ser usados em vários pontos no circuito final.

Essa etapa foi realizada, transformando cada agrupamento das funções fatoradas em uma variável correspondente. O processo de decomposição é demonstrado através das equações mostradas no Apêndice C.

Com a decomposição realizada, as funções de saída podem ser expressas algebricamente como:

$$\mathrm{s} = \mathrm{x}_{93} + \mathrm{x}_{29} + \mathrm{x}_{33} \tag{2.14}$$

$$\mathrm{cout} = \mathrm{x}_{0}\mathrm{x}_{37} \tag{2.15}$$

$$\mathrm{pede1} = \mathrm{x}_{0}\mathrm{x}_{39} \tag{2.16}$$

$$\mathrm{maior\_out} = \mathrm{x}_{41}\mathrm{x}_{52} \tag{2.17}$$

$$\text{igual\_out} = x_{41}x_{54} \tag{2.18}$$

$$\text{menor\_out} = x_{41}x_{61} \tag{2.19}$$

Nessa fase do projeto algumas otimizações em termos de área foram feitas, tais como na equação $x_{17} = \overline{x_{13}}$, que inicialmente era $x_{17} = b\bar{a} + \bar{b}a$. Da álgebra booleana tem-se que $b\bar{a} + \bar{b}a = \overline{\bar{b}\bar{a} + ba}$, que é exatamente o complemento de $x_{13} = \bar{b}\bar{a} + ba$. Desta forma pode-se compartilhar *hardware* e tentar obter algum ganho em área.

3 PROJETO COM NAND’S

3.1 Mapeamento Tecnológico

Esse processo consiste em transformar todas as equações obtidas na decomposição em uma descrição equivalente, porém contendo somente portas de uma dada biblioteca. Nessa etapa do projeto, o circuito foi descrito manualmente para portas NAND de duas entradas e inversores. O motivo para implementar com tais portas, é que temos conexões em série na região N do transistor e assim por motivos elétricos tem-se uma porta com uma resposta mais rápida que uma porta NOR, que tem conexões em série na região P. Outro motivo seria a grande regularidade do circuito, por implementar células iguais para todo circuito.

A partir das descrições obtidas, foram feitas algumas otimizações em termos de área, como por exemplo, na equação:

$$x_{10} = \bar{b}\bar{a} \tag{3.1}$$

Transformando em NAND’s:

$$x_{10} = \overline{\overline{\bar{b}\bar{a}}} \tag{3.2}$$

A equação x_{10} é usada em x_{13}, de modo que:

$$x_{13} = x_{10} + x_7 \tag{3.3}$$

Transformando em NAND’s:

$$x_{13} = \overline{\overline{x_{10}}\,\overline{x_7}} \tag{3.4}$$

Se percebe que em $x_{13} = \overline{\overline{x_{10}}\,\overline{x_7}}$, necessita-se apenas do complemento de x_{10}, que já está implementado parcialmente na equação de x_{10} em NAND’s. Com isso, é possível reduzir o número de NAND’s.

Em outro exemplo, a equação $x_{11} = x_6 e$, transformada para NAND’s ficaria $x_{11} = \overline{\overline{x_6 e}}$, porém apenas o complemento de x_{11} faz parte do circuito final, logo $\overline{x_{11}} = \overline{x_6 e}$. Com isso, podemos reduzir o custo de um inversor na equação.

Essas situações se repetem em outras equações, o que permite uma diminuição de área no circuito final.

3.2 Verificação Lógica

Esta etapa do processo consiste em verificar se o conjunto de transformações das equações foi realizado corretamente, ou seja, se o fluxo de projeto mantém a consistência do projeto proposto.

Esse procedimento foi feito de maneira manual, por meio da extração do circuito através das equações mapeadas. A partir desse ponto foi verificado se o circuito estava de acordo com a tabela verdade. Esta verificação foi feita com o desenho do circuito mapeado para NAND's, desta forma foi inserido manualmente nas entradas do circuito os níveis lógicos da tabela verdade e comparado se os resultados das saídas correspondiam aos valores esperados. Nesse processo algumas transformações errôneas foram encontradas e corrigidas.

3.3 Circuito com Portas NAND

Das equações mapeadas foi descrito o circuito lógico em portas lógicas para um *slice* da ULA, conforme Figura 3.1. Essa estrutura apresenta 121 portas lógicas entre NAND's e inversores, e conforme citado anteriormente, apresenta treze sinais de entrada e seis de saída.

Com objetivo de produzir a ULA de oito *bits*, a estrutura apresentada foi replicada oito vezes, com um total de 968 portas lógicas.

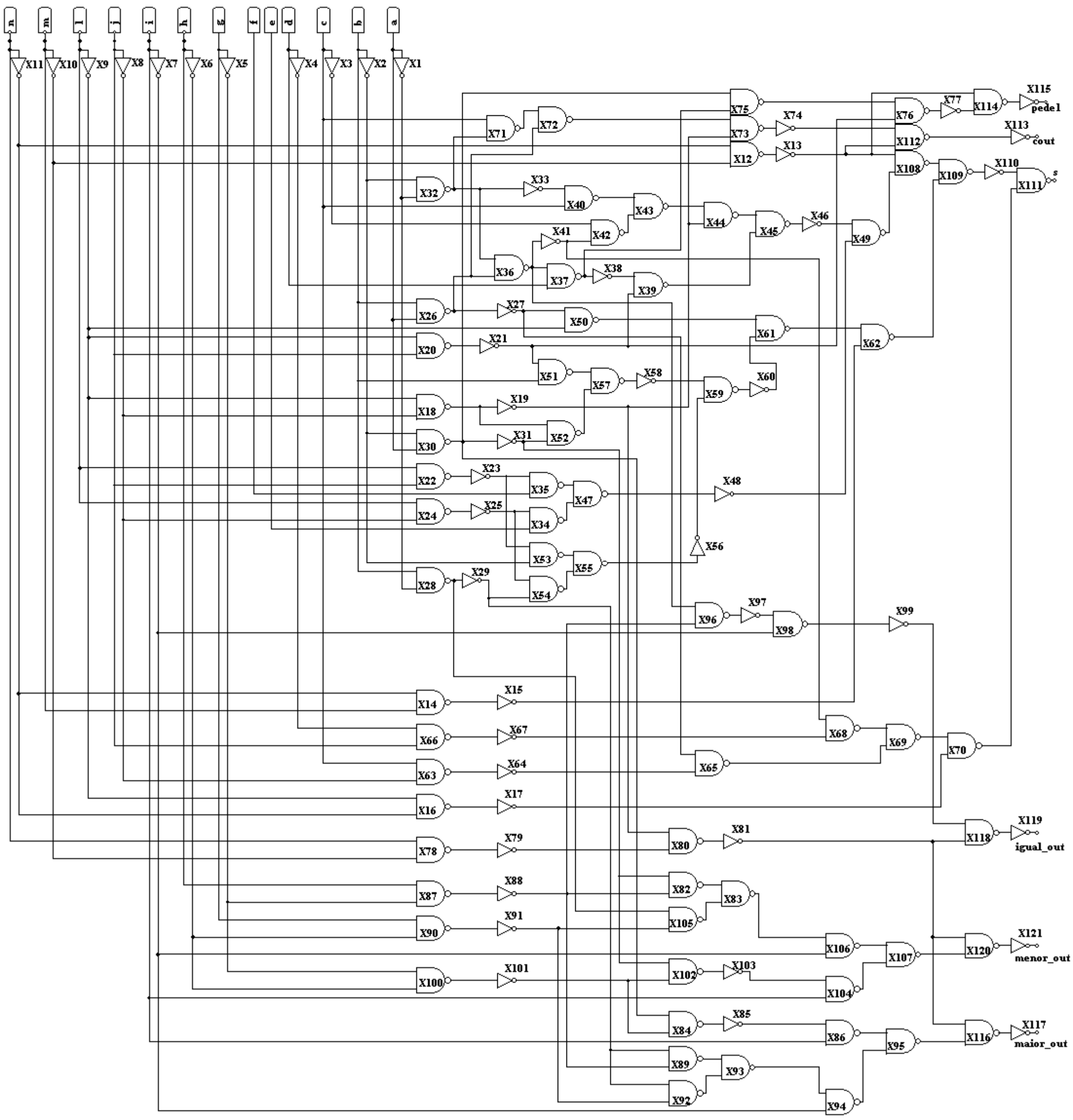

FIGURA 3.1 – Circuito com portas NAND de um *Slice* da ULA

3.4 Síntese Física

Nessa etapa do processo, o objetivo é obter o *layout* físico do circuito, tendo como ponto de partida as equações lógicas mapeadas para NAND's e inversores.

O caminho adotado para atingir este objetivo foi através de uma ferramenta de CAD, o MARTELO (MENEZES, 2003). Esta ferramenta é um gerador de matrizes regulares, que permite a geração de matrizes de portas NAND. A matriz é formada por bandas de células. Cada parte constituinte das bandas corresponde a uma célula básica para a tecnologia AMS

(*AustriaMicroSystems*) de 0,35µm, este valor representa o comprimento do canal do transistor. A célula básica é formada por um par de NAND's, conforme mostrado na Figura 3.2. Porém, em algumas situações é possível implementar um número ímpar de portas NAND. A ferramenta implementa as NAND's na tecnologia CMOS e a mesma permite dimensionar a largura dos transistores PMOS (*P-channel Metal Oxide Semiconductor*) e NMOS (*N-channel Metal Oxide Semiconductor*).

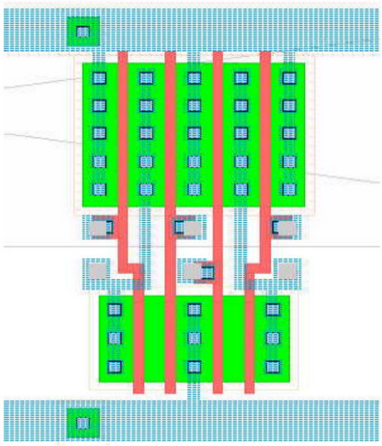

FIGURA 3.2 – *Layout* de um par de NAND's
Fonte: Menezes (2003, p. 31)

Com o objetivo de realizar as conexões entre as células, o MARTELO realiza iterações juntamente com uma ferramenta de roteamento. Caso o *software* não consiga gerar o roteamento na quantidade de níveis desejada, este gera um *layout* com trilhas verticais ou horizontais inseridas nos locais em que conflitos são detectados. Parâmetros de roteamento também podem ser definidos na ferramenta. Outro fator importante que pode ser configurado é o número de níveis de metal, que tem grande influência no roteamento e por conseqüência na geração do *layout*. Nesta etapa se deseja trabalhar com o menor número de níveis de metal, pois isso aumenta os passos de fabricação, além de estar sujeito a erros no processo e de aumentar os custos. No entanto é necessário evitar o afastamento das células permitindo fazer o roteamento nos níveis acima e por conseqüência não expandindo a área do circuito.

A partir das equações foi necessário descrevê-las no formato padrão da ferramenta, conforme mostrado no Apêndice D. Posteriormente, foi necessário realizar a descrição do posicionamento físico de cada porta lógica no circuito. Nesse estágio, são definidos o número de bandas da matriz e o local físico de cada NAND no circuito.

A etapa de posicionamento foi realizada de maneira manual, de modo que alguns aspectos foram levados em consideração. Dentre tais aspectos, cita-se a importância de situar as entradas e saídas nas extremidades físicas do circuito. Outro fator importante é posicionar as entradas e saídas de maneira que os *slices* adjacentes possam ser facilmente acoplados, tendo em vista que há conexões entre os *slices*. Nessa etapa é essencial posicionar portas lógicas com conexões entre si, de maneira que fiquem o mais próximo possível, tendo em vista que fisicamente facilita o roteamento das interconexões, além de contribuir para a redução do atraso geral do circuito. Adicionalmente a isto, é necessário definir o número de bandas de células e o número de células por banda, com o objetivo de obter uma matriz mais quadrada possível.

O posicionamento inicialmente foi feito para que a matriz do *slice* ficasse aproximadamente quadrada, porém interconectando os *slices* o *layout* apresentou espaços vazios, o que é indesejável. Com isto, o posicionamento manual foi feito novamente objetivando um layout mais horizontal por *slice*, para que o circuito completo fosse o mais quadrado possível sem espaços vazios. A descrição do posicionamento para um *slice* é mostrada no Apêndice E.

A última etapa do processo é a geração do *layout* propriamente dito. Por meio das descrições da lógica e de posicionamento, a ferramenta gera o *layout* fazendo o roteamento até concluir as interconexões do circuito.

O *layout* do roteamento de um *slice* da ULA é demonstrado na Figura 3.3. O *layout* foi feito para um número de três bandas de células, de modo que o *slice* apresentasse um formato horizontal e um aspecto quadrado conectando todos *slices*. Foi escolhido fazer o roteamento em três níveis de metal, pois com dois níveis a ferramenta precisou inserir trilhas no circuito para fazer as conexões do circuito, e por conseqüência expandindo a área do circuito. A largura escolhida dos transistores PMOS foi de 4μm e dos transistores NMOS foi de 2μm.

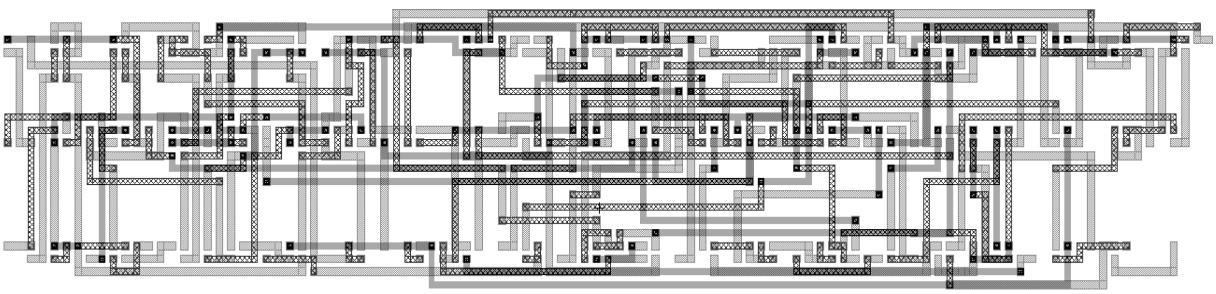

FIGURA 3.3 – *Layout* do roteamento de um *Slice* da ULA

Na Figura 3.4 é mostrado o *layout* completo do *slice* do circuito.

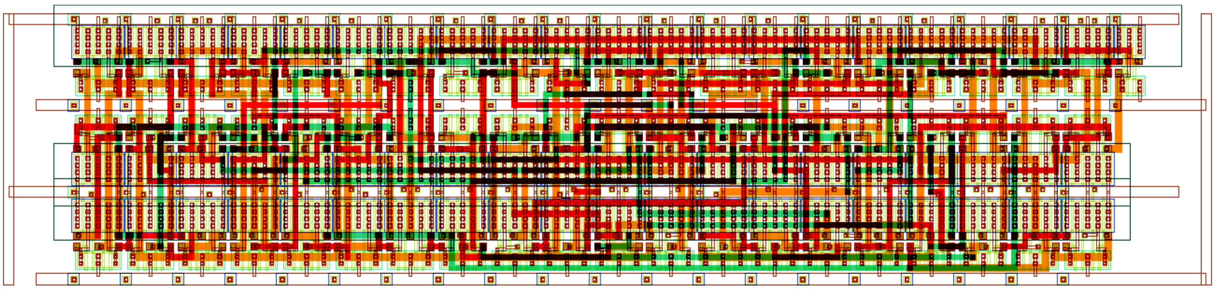

FIGURA 3.4 – *Layout* do *Slice* da ULA

3.5 Interconexão dos *Slices*

É importante salientar que como os *slices* são conectados, os sinais primários do circuito são oito *bits* de cada operando de entrada; o sinal de *carry* do somador e o de empréstimo do subtrator para o primeiro estágio; os sinais de entrada da comparação no primeiro *slice*; o sinal de entrada do deslocador à esquerda do *slice* menos significativo; e o sinal de entrada do deslocador à direita do *slice* mais significativo.

Para esta etapa, foi necessário replicar as descrições da lógica e do posicionamento do *slice*. De modo que os sinais propagados fossem conectados aos *slices* adjacentes. O *layout* do circuito com os oito *slices* conectados é mostrado na Figura 3.5.

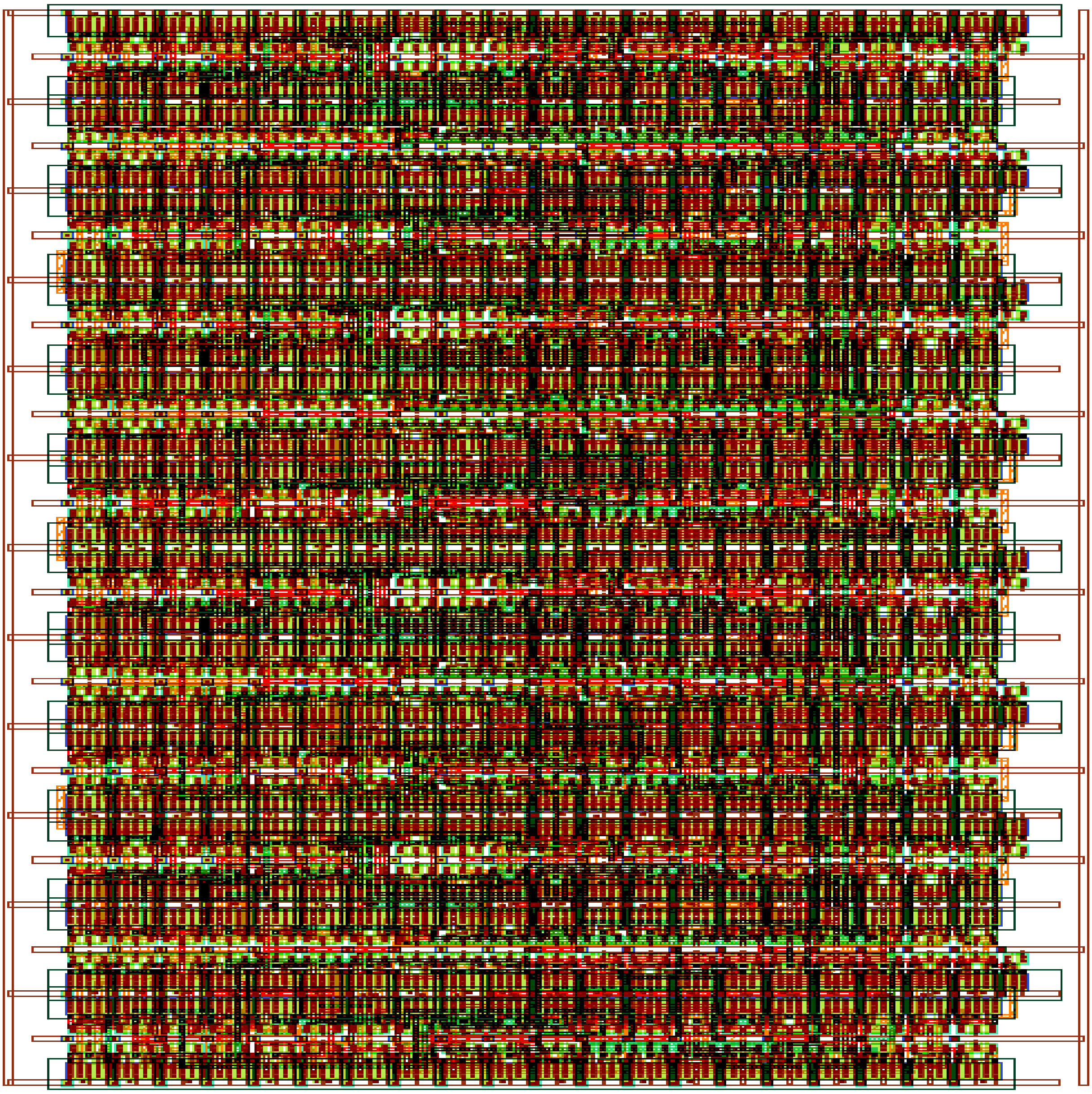

FIGURA 3.5 – *Layout* da ULA 8 *bits*

O circuito completo apresenta uma área de 50755,8 μm^2 (0,0507mm^2).

3.6 Resultados da Implementação com Mapeamento em NAND's

Um sumário das propriedades físicas da Unidade Lógica Aritmética é apresentado na Tabela 3.1.

TABELA 3.1 – Resultados obtidos através da implementação com NAND's

Métrica	**Valor**
Número de portas lógicas do *bit-slice*	121
Número de portas NAND do *bit-slice*	73
Número de portas de inversoras do *bit-slice*	48
Número de transistores do *bit-slice*	484
Número de portas lógicas do circuito	968
Número de transistores do circuito	3872
Área de uma NAND (μm^2)	60,75
Área de uma Banda (μm^2)	2349
Área do *slice* (μm^2)	6733,8
Área total do circuito (μm^2)	50755,8
Comprimento da fiação do circuito (µm)	21542

É importante salientar que a ferramenta gera apenas células NAND's, porém no circuito lógico, além das NAND's têm-se inversores. O custo de uma porta inversora corresponde a apenas dois transistores. Com isso, devido a essa limitação de se ter apenas células NAND's e ter que implementar os inversores com estas células, adiciona-se dois transistores por inversor, o que representa um aumento de transistores no circuito. Fator este que apresenta um impacto considerável em termos de área e consumo.

4 PROJETO COM PORTAS COMPLEXAS

As portas complexas são portas que representam uma função lógica complexa e são formadas por uma associação série/paralelo de transistores. Esta técnica de implementação permite obter um circuito otimizado, através de um número reduzido de transistores em comparação com um circuito implementado apenas com portas simples.

4.1 Mapeamento Tecnológico

A partir da fatoração obteve-se o circuito, desta forma agrupamentos complexos foram realizados. Na obtenção destes, teve-se a preocupação em não se ultrapassar um total de cinco transistores em série para cada porta complexa. Este fato deve-se por motivos físicos, que torna a porta complexa lenta, tendo em vista que conexões em série aumentam a resistência da porta e dificultam a passagem de corrente, aumentando a constante RC formada pela carga que sempre possui uma componente capacitiva.

Com o objetivo de fazer compartilhamento de *hardware* e por conseqüência obter diminuição do número de transistores, algumas portas simples foram implementadas, porém também limitadas por cinco transistores em série. Essa limitação foi aplicada baseada nos diagramas elétricos das portas.

Na Figura 4.1 é mostrado o circuito de um *slice* extraído das equações obtidas na fatoração e já com os agrupamentos complexos idealizados.

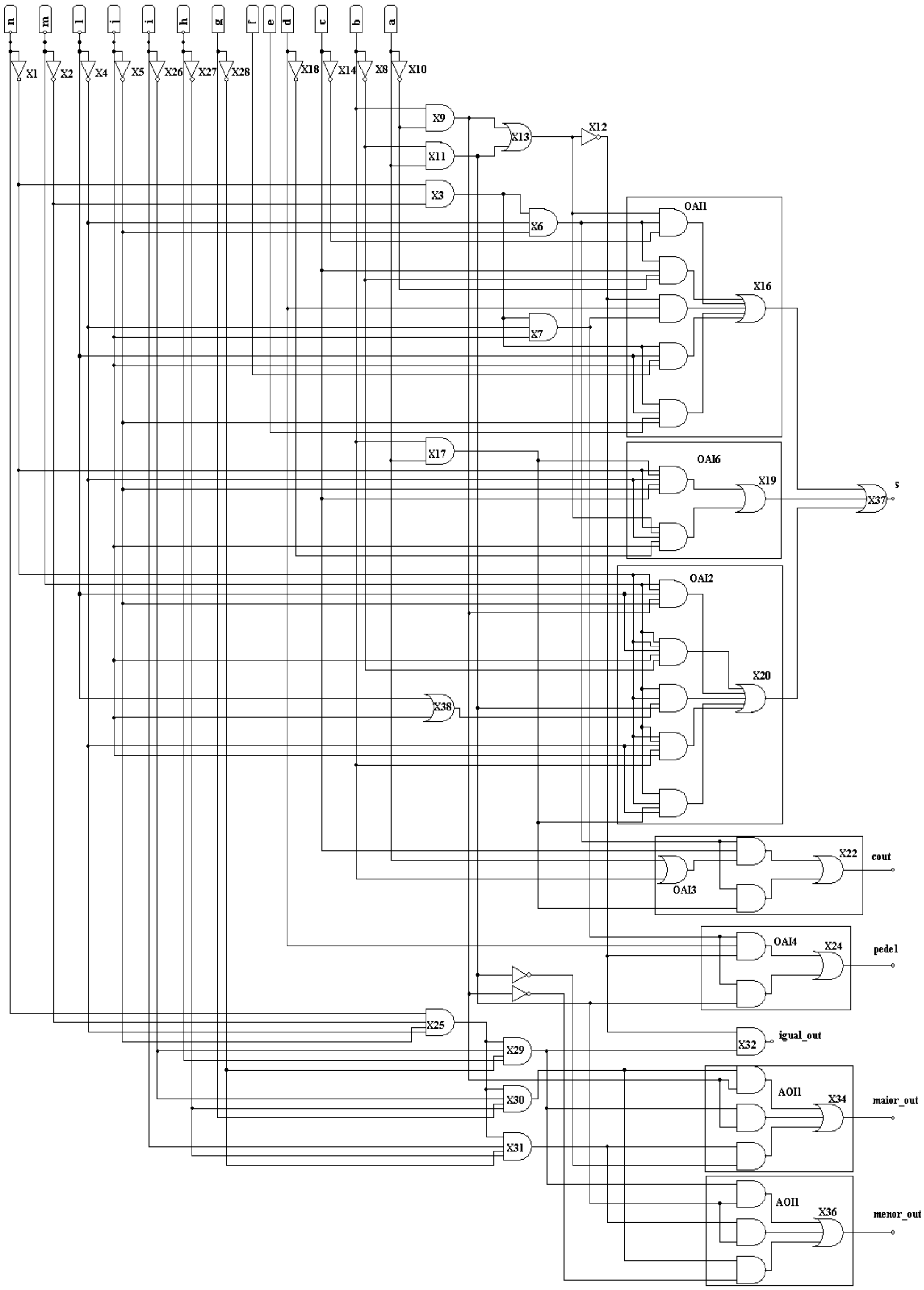

FIGURA 4.1 – Circuito lógico com portas complexas de um *Slice* da ULA

4.2 Síntese Física

No projeto com portas complexas, a etapa de síntese física foi feita com a utilização da ferramenta ASTRAN - *Automatic Synthesis of Transistor Networks* (ZIESEMER, 2007). Esta ferramenta permite a geração automática do *layout* de células MOS. A ferramenta gera uma matriz capaz de suportar células com diferentes redes de transistores.

A primeira etapa da síntese com esta ferramenta envolve a geração das células, já que neste caso as células se diferenciam umas das outras. Estas descrevem o comportamento de cada porta lógica do circuito. Nesse processo de geração de células, é feito o posicionamento dos transistores no interior das mesmas e o roteamento entre eles. Também é permitido dimensionar cada transistor das células projetadas. Com isso, a ferramenta gera as células de acordo com o tamanho dos transistores desejado, respeitando as regras de projeto da tecnologia especificada pelo projetista.

Após esta etapa, a ferramenta permite fazer automaticamente o posicionamento das células e também o roteamento das conexões entre elas. Estes processos são realizados interativamente com o projetista, de modo que é possível definir o número de bandas do circuito e também o número de níveis de metal. Após estas etapas a ferramenta permite gerar o layout físico do circuito.

4.2.1 Projeto das Células

Pelo fato da tecnologia CMOS possuir lógica negativa, as células implementadas não representam exatamente as portas descritas no circuito mostrado na Figura 4.1. Para fazer essa transformação sem alterar a lógica do circuito, duas técnicas foram adotadas baseadas na álgebra booleana. Dentre elas o teorema de De Morgan e a propriedade da complementação. As duas técnicas foram utilizadas e as escolhas foram realizadas tentando evitarem-se portas adicionais no circuito.

Primeiramente foram idealizadas as células complexas. Nessa etapa foi determinado se a saída de cada porta seria negada ou direta e também quais transformações no circuito seriam realizadas.

Nas portas *x16* (*oai1*), *x19* (*oai6*), *x20* (*oai2*), as saídas escolhidas foram de lógica negada. Neste caso aplicou-se a propriedade da complementação, assim as entradas dessas portas se mantêm diretas. Sabendo-se que as saídas destas portas são negadas e estão conectadas às entradas da porta *x37*, mantiveram-se as entradas negadas para não se adicionar inversores no circuito. Então a porta *x37* (*OR*) foi transformada em uma porta NAND com as entradas negadas, não havendo assim alteração de lógica.

Em relação às portas conectadas as entradas das portas *x16* (*oai1*), *x19* (*oai6*) e *x20* (*oai2*), foram projetadas para que suas saídas fossem diretas. Então se aplicou o teorema de De Morgan sobre as portas *x6, x7, x9, x11* e *x17*, que inicialmente eram portas AND e foram transformadas em portas NOR com as entradas negadas, já a porta *x38*, que era uma porta OR, foi transformada em uma NAND.

As portas *x22* (*oai3*) e *x24* (*oai4*) também foram projetadas para que as saídas fossem negadas. Porém, como se tratam de saídas diretas do circuito foi necessário inserir um inversor na saída de cada uma dessas portas.

As portas *aoi1* (*x34* e *x36*) foram projetadas com saídas diretas, já que são saídas reais do circuito, não sendo necessário adicionar inversores nas saídas. Para isso aplicou-se o teorema de De Morgan, logo as respectivas entradas devem ser negadas. Estas entradas são procedentes das portas AND's (*x29*, *x30*, *31*), que precisam ter as saídas negadas, portanto foram transformadas em NAND's. Já as entradas procedentes das portas *x9* e x11 tiveram que ser negadas com inversores. E as entradas que estavam conectadas aos inversores nas saídas das portas *x9* e x11, foram conectadas diretamente nas saídas de *x9* e x11.

O circuito conforme é implementado fisicamente é mostrado na Figura 4.2.

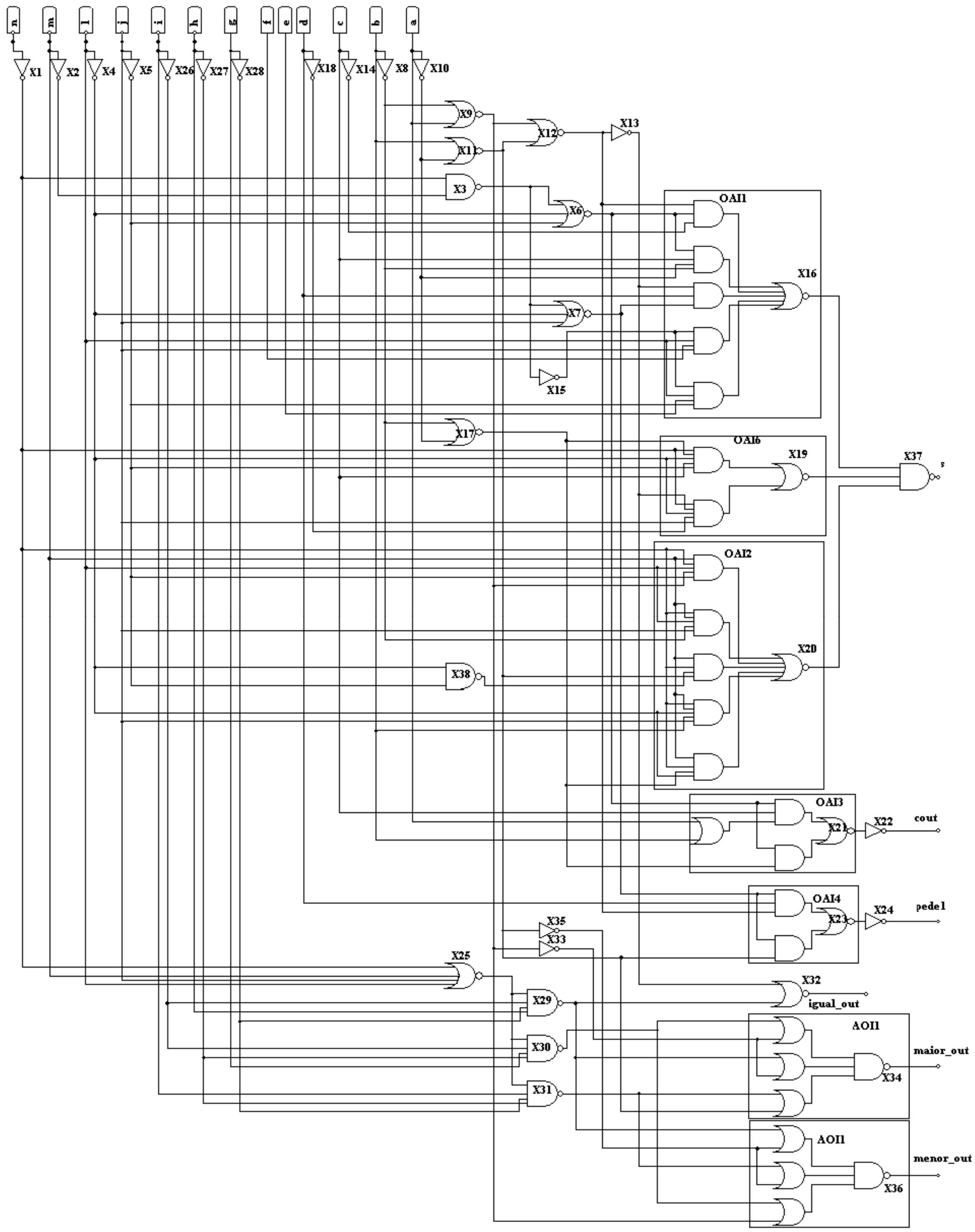

FIGURA 4.2 – Circuito lógico de um *Slice* descrito em portas negadas

4.2.2 Construção das células

As células foram criadas na tecnologia 0,35µm, com largura de 4µm para os transistores PMOS e de 2µm para os NMOS. Um exemplo de *layout* gerado pelo ASTRAN, que descreve uma célula inversora é mostrado na Figura 4.3 e seu respectivo diagrama elétrico é mostrado na Figura 4.4.

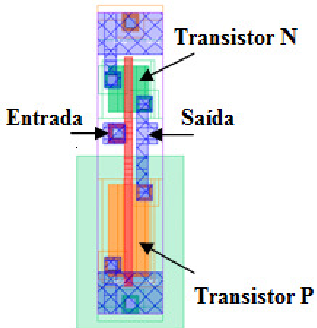

FIGURA 4.3 – Célula que descreve uma porta inversora

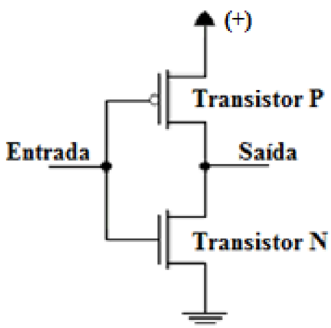

FIGURA 4.4 – Diagrama elétrico da porta inversora

Nesta etapa do projeto foi necessário descrever para cada célula como a rede de transistores está conectada no interior das mesmas.

Tendo como ponto de partida o circuito lógico e das transformações necessárias devido à tecnologia, foi realizada uma análise sobre quais células deveriam ser criadas. Para o *slice* da ULA foram criadas células que descrevem as portas complexas do circuito, bem como, as portas simples projetadas.

As células simples foram denominadas de *inv, nand2, nand3, nand4, nor2, nor3* e *nor4*. Já as complexas, denominadas de *oai1, oai2, oai3, oai4, oai6* e *aoi1*. As células citadas foram descritas no formato padrão da ferramenta, conforme mostrado no Apêndice F.

Os *layouts* das células projetadas são mostrados a seguir:

- Célula *inv*:

  Representa a função: out = $\bar{a}$

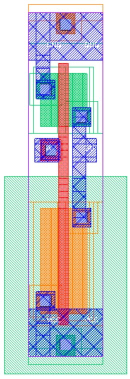

FIGURA 4.5 – *Layout* da célula *inv*

- Célula *nand2*:

  Representa a função: out = $\overline{ab}$

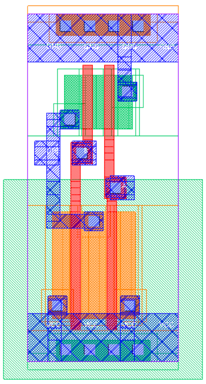

FIGURA 4.6 – *Layout* da célula *nand2*

- Célula *nand3*:

  Representa a função: out = $\overline{abc}$

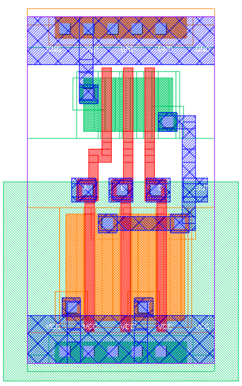

FIGURA 4.7 – *Layout* da célula *nand3*

- Célula *nand4*:

  Representa a função: $out = \overline{abcd}$

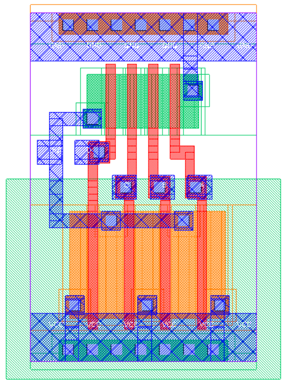

FIGURA 4.8 – *Layout* da célula *nand4*

- Célula *nor2*:

  Representa a função: $out = \overline{a + b}$

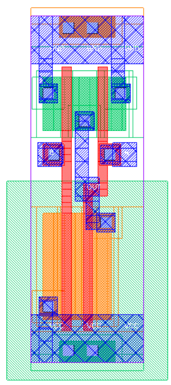

FIGURA 4.9 – *Layout* da célula *nor2*

- Célula *nor3*:

  Representa a função: $out = \overline{a + b + c}$

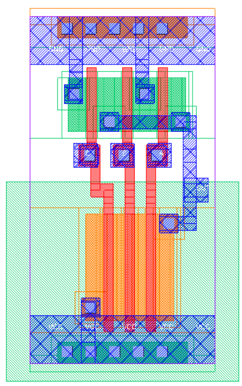

FIGURA 4.10 – *Layout* da célula *nor3*

- Célula *nor4*:

Representa a função: $out = \overline{a + b + c + d}$

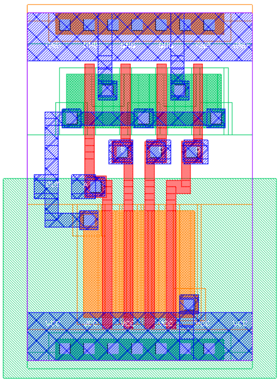

FIGURA 4.11 – *Layout* da célula *nor4*

- Célula *oai1*:

Representa a função: $out = \overline{a(bc + def) + gh\imath + \jmath l(mn + op)}$

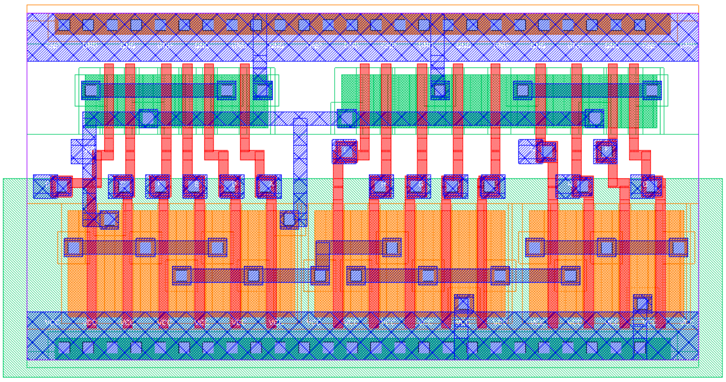

FIGURA 4.12 – *Layout* da célula *oai1*

- Célula *oai2*:

Representa a função: $out = \overline{ab(c(de + fg) + h\imath + \jmath(lm + n))}$

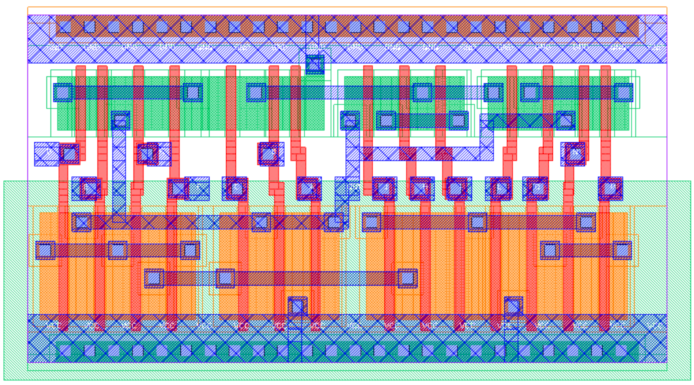

FIGURA 4.13 – *Layout* da célula *oai2*

- Célula *oai3*:

  Representa a função: $out = \overline{a(b(c + d) + e)}$

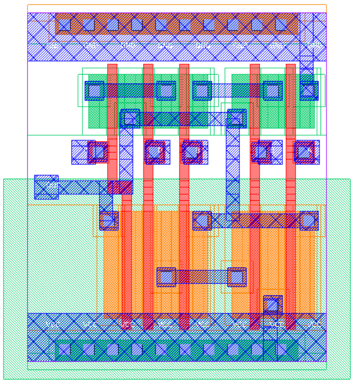

FIGURA 4.14 – *Layout* da célula *oai3*

- Célula *oai4*:

  Representa a função: $out = \overline{a(bc + d)}$

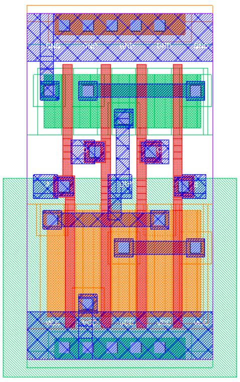

FIGURA 4.15 – *Layout* da célula *oai4*

- Célula *oai6*:

  Representa a função: $out = \overline{ab\,(cde + fgh)}$

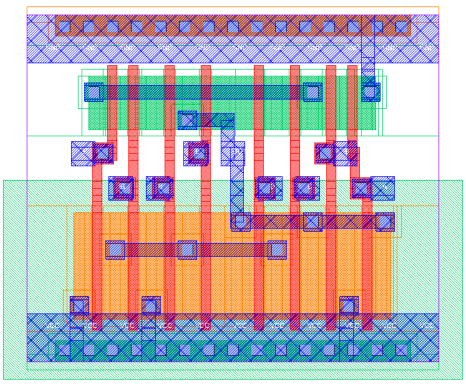

FIGURA 4.16 – *Layout* da célula *oai6*

- Célula *aoi1*:

  Representa a função: out = $\overline{(ab + c)\ (d + e)}$

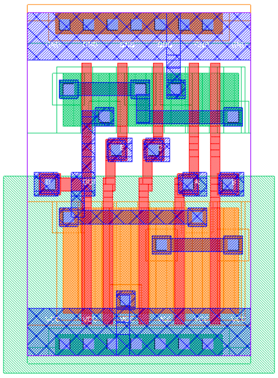

FIGURA 4.17 – *Layout* da célula *aoi1*

4.3 Interligação das Células e dos *Slices*

O próximo passo foi descrever as interconexões entre as células que representam o circuito de um *slice* da ULA. Em seguida foi necessário replicar essa descrição para obter o circuito de oito *bits*. Tendo em vista, que existe interdependência entre os *slices* também foi necessário descrever os sinais que são conectados entre os *slices* adjacentes. A descrição de um *slice* da ULA é mostrada no Apêndice G.

4.4 Posicionamento e Roteamento

A etapa de posicionamento é feita de maneira automática, porém permite interação com o projetista. Nesta fase, é possível definir o número de bandas do circuito, que são representadas por linhas no *layout,* de modo que cada banda é composta por células do circuito. Ao final do processo, o conjunto de bandas contém todas as células posicionadas. O número de bandas está diretamente relacionado com o formato do layout. Posteriormente a ferramenta permite fazer o roteamento das interconexões do circuito para o número de níveis de metal desejado. O layout do roteamento em três níveis de metal é mostrado na Figura 4.18. Foi escolhido trabalhar em três níveis, tendo em vista que no projeto com portas NAND's foi trabalhado com esse número de níveis, sendo assim, esse parâmetro ficasse igual nas duas implementações.

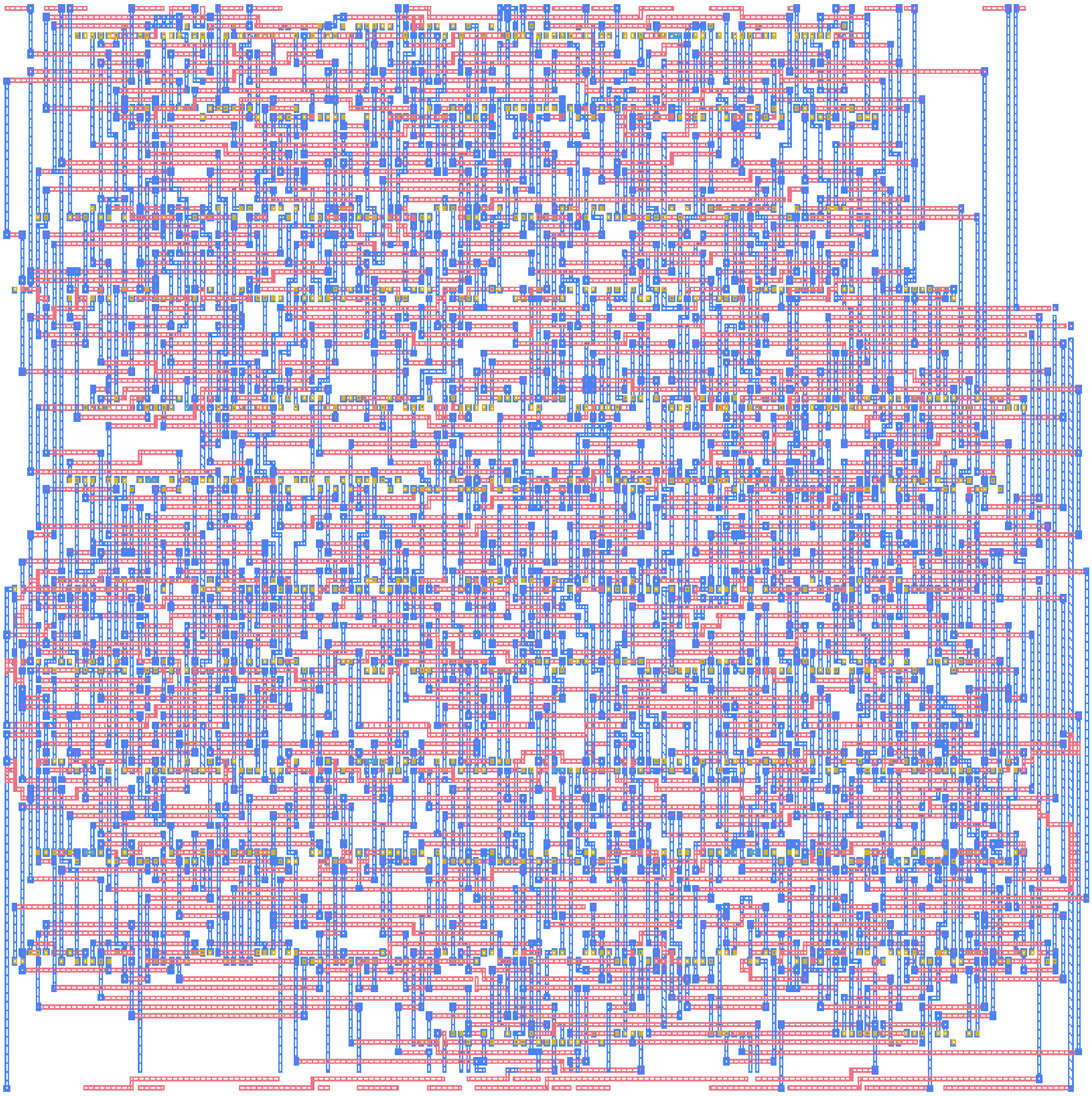

FIGURA 4.18 – *Layout* do roteamento do circuito

O layout que descreve o circuito completo é mostrado na Figura 4.19. Foi escolhido um número de doze bandas para que o circuito tivesse um formato mais quadrado possível.

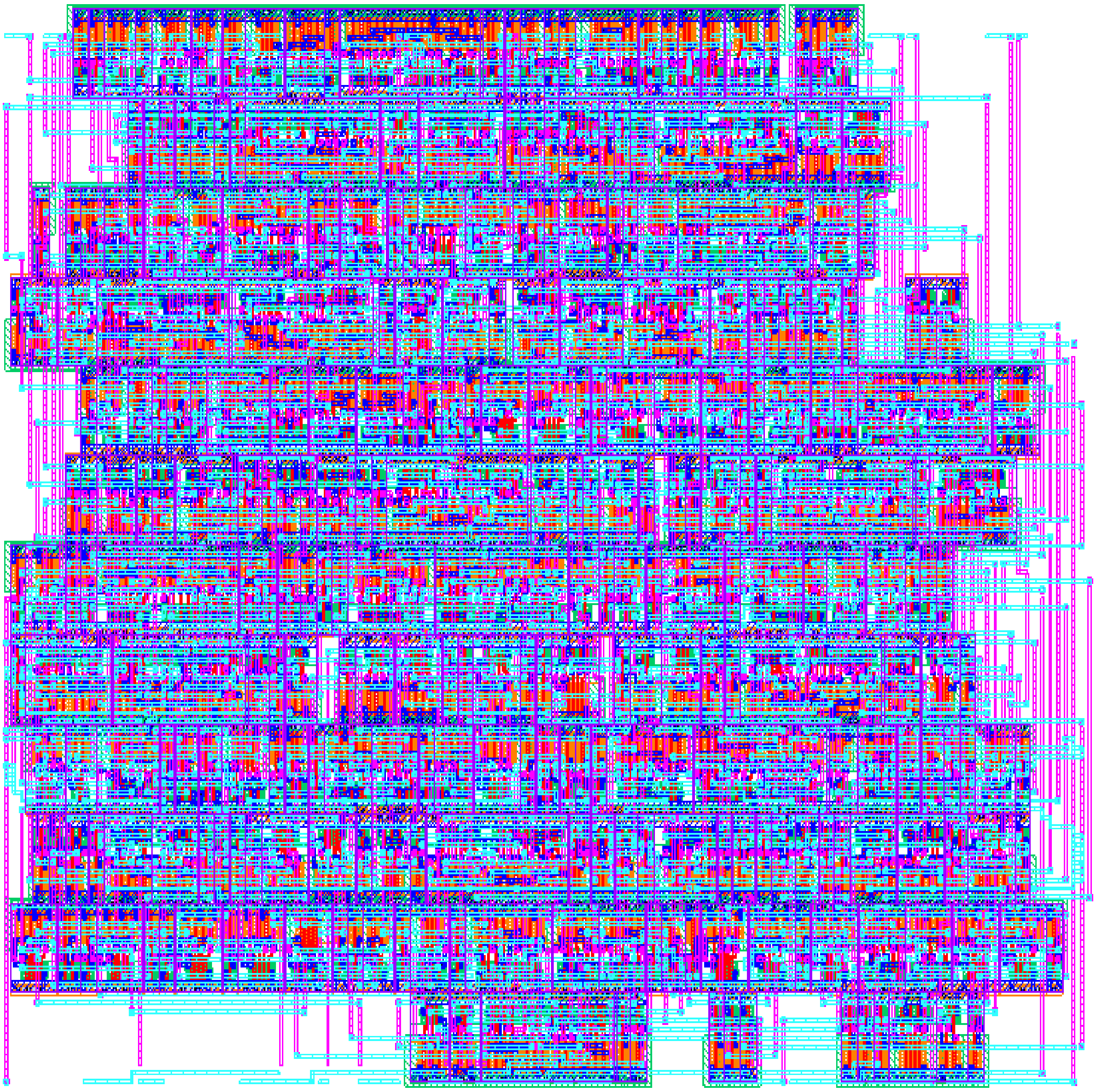

FIGURA 4.19 – *Layout* do circuito completo

4.5 Resultados da Implementação com Mapeamento em Portas Complexas

Do projeto utilizando portas complexas extraíram-se os resultados obtidos, conforme Tabela 4.1.

TABELA 4.1 – Resultados obtidos através da implementação com portas complexas

Métricas	**Valor**
Número de portas do *bit-slice*	38
Número de transistores do *bit-slice*	222
Número de portas do circuito	304
Número de transistores do circuito	1776
Área estimada de um *slice* (μm²)	3057,6
Área do circuito (μm²)	24460,8
Comprimento da Fiação (μm)	21665

O resultado do número de portas, transistores foram extraídos manualmente, já os resultados da área física e comprimento da fiação foram obtidos através da ferramenta.

5 COMPARAÇÕES DOS RESULTADOS

TABELA 5.1 – Métricas dos resultados obtidos através das implementações

Métrica	NAND's	Portas Complexas
Número de portas do *bit-slice*	121	38
Número de transistores do *bit-slice*	484	222
Área total do circuito *bit-slice* (µm²)	6733,8	3057,6
Comprimento total da fiação do *bit-slice* (µm)	2412	1942
Número de portas do circuito	968	304
Número de transistores do circuito	3872	1776
Área total do circuito (µm²)	50755,8	24460,8
Comprimento total da fiação do circuito (µm)	21542	21665
Fan-in médio	2	3
Fan-out médio	2	3

Os resultados da Tabela 5.1 mostram que se obteve no projeto com portas complexas, um circuito com um número menor de portas. Isso se deve pelo fato de ter portas com *fan-in* mais elevado que duas entradas, o que aumenta as possibilidades lógicas de cada porta. Porém a comparação com portas lógicas não é muito justa, já que no projeto complexo têm-se vários tipos de portas e com várias entradas. Em contrapartida, o projeto com NAND's só apresenta um tipo de porta. Em se tratando, do número de transistores o projeto com portas complexas apresenta uma redução de aproximadamente 54%. Essa redução impactou na dimensão física do circuito, reduzindo a área em aproximadamente 52%.

Em ambas implementações desse projeto trabalhou-se com transistores NMOS e PMOS, com largura de 2µm e 4µm, respectivamente. Porém, se formos levar em conta o desempenho elétrico das portas e considerarmos o *fan-out* das mesmas, provavelmente parte dos transistores do projeto com portas complexas devessem ter dimensão maior que o projeto com NAND's. Para ter certeza dessa situação, seria necessário fazer a simulação elétrica do circuito. Esse fato poderia acarretar um aumento na área no projeto com portas complexas.

6 CONSIDERAÇÕES FINAIS

Do resultado do projeto com mapeamento para portas NAND, obteve-se um *slice* da ULA que contém 121 portas, o que corresponde a 484 transistores. Tendo em vista que o projeto é para oito *bits*, este circuito necessitará de 3872 transistores. Em termos estruturais, o circuito apresenta bastante regularidade, como pode ser visto no *layout* gerado. Como característica da estrutura *bit-slice* percebe-se que o projeto para um *bit* pode ser facilmente replicado para qualquer quantidade de *bits.*

No projeto com portas complexas obteve-se um circuito com 38 portas e 222 transistores para um *slice,* com um total de 1776 transistores para o circuito completo, o que representa um circuito com praticamente metade do número de transistores, e por conseqüência em área do circuito. Porém, apesar da estrutura *bit-slice* ser regular, esta característica não se torna visível no *layout* com portas complexas.

Comparando-se as duas implementações, pode-se observar que houve uma grande redução no número de transistores no projeto com portas complexas. Grande parte dessa diferença é pelo fato de todas as portas implementadas no projeto com NAND's serem implementadas com células NAND's e como 39,67% do circuito são inversores, tem-se dois transistores adicionais nessas portas. Sendo assim, poderia se diminuir o número de transistores, se os inversores fossem implementados com células inversoras. Porém devido ao fato da ferramenta MARTELO usar uma estratégia para gerar layouts regulares, de modo que todo o circuito seja mapeado para NAND's, tem se a regularidade como aspecto positivo. Embora se tenha um número maior de transistores no circuito, em tecnologias muito pequenas, a regularidade é um aspecto muito importante, sendo menos suscetível a erros decorrentes da variabilidade do processo de fabricação (MENEZES, 2003). Além disso, circuitos regulares permitem um menor tempo de projeto e melhores estimativas de atraso e consumo (MENEZES, 2003).

É importante salientar que a área é o foco deste trabalho, e esta característica está relacionada diretamente com o número de transistores e também com as fiações pertencentes ao circuito. A área é um bom critério para o desempenho do circuito, pois com menos transistores pode se ter menos consumo dinâmico e estático no circuito. Reduzir o número de transistores tem forte impacto na área do circuito e no consumo de energia. No entanto, quando se projeta circuitos integrados, além da área é necessário levar em conta: os atrasos, o consumo de energia, a regularidade, entre outros. O projeto de um circuito integrado é uma tarefa complexa e executada em diferentes passos, e ao se projetá-lo, os parâmetros de projeto não podem ser desconsiderados.

Como sugestões, para trabalhos futuros, poderia se realizar a simulação elétrica do circuito e tentar estimar os atrasos e o consumo do circuito. Também poderia ser feito um trabalho voltado para o dimensionamento dos transistores, preocupando-se com os atrasos que isso implicaria. Por fim, existe um conjunto de preocupações que podem ser estudados, quando se fala em projeto de circuitos integrados.

REFERÊNCIAS

ASTRAN. Disponível em <http://www.inf.ufrgs.br/~amziesemerj/icpd/>. Acesso em: out, 2011.

BAKER, R. Jacob. **CMOS: Circuit Design, Layout, and Simulation**. 2. ed. New Jersey: Wiley, 2008.

BARROS, Edna. **Arquitetura de Sistemas Embarcados**. Centro de Informática - UFPE Disponível em <http://www.di.ufpe.br/~ensb/courses/public_html/slides/design-tech-arm-06-1.pdf> Acesso em: 27 abr. 2011.

BOYLESTAD, Robert; NASHELSKY, Louis. **Dispositivos Eletrônicos e Teoria de Circuitos**. 6. ed. Rio de Janeiro: LTC, 1998.

BRAYTON, Robert King; HACHTEL, Gary D.; MCMULLEN, Curtis T.; SANGIOVANNI-VINCENTELLI, Alberto L. **Logic Minimization Algorithms for VLSI Synthesis**. Norwell: Kluwer Academic Publishers, 1984.

BROWN, Stephen; VRANESIC, Zvonko. **Fundamentals of Digital Logic With VHDL Design**. 2. ed. New York: McGraw Hill, 2004.

CONG, Jason; SARRAFZADEH, Majid. **Incremental Physical Design**. International Symposium on Physical Design, 2000. **Proceedings**... [S.l.: s.n.], 2000. 84-92.

COUDERT, O. **Timing and Design Closure in Physical Design Flows**. International Symposium on Quality Eletronic Design (ISQED), 2002. **Proceedings**... [S.l.: s.n.], 2002. 511-516.

DE MICHELI, Giovanni. **Synthesis and Optimization of Digital Circuits**. Stanford: McGraw-Hill Science Engineering, 1994.

DEVI, Padma; GIRDHER, Ashima; SINGH, Balwinder. **Improved Carry Select Adder with Reduced Area and Low Power Consumption**. International Journal of Computer Applications, 2010. 14-18.

KATZ, Randy H.; BORRIELLO, Gaetano. **Contemporary Logic Design**. 2. ed. NJ, USA: Prentice Hall, 2005.

LIMA, Fernanda Gusmão de. **Projeto com Matrizes de Células Lógicas Programáveis**. 1999. 124 f. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre, 1999.

MARQUES, F. S.; MARTINELLO JR., O.; RIBAS, R. P.; ROSA JR, L. S.; REIS, A. I. **Mapeamento Tecnológico no Projeto de Circuitos Integrados Digitais**. In: MATTOS, J.; ROSA JR., L.; PILLA, M. (Org.). Desafios e Avanços em Computação: o Estado da Arte. Pelotas: Editora da Universidade Federal de Pelotas, 2009. p. 173-191.

MEINHARDT, Cristina. **Geração de Leiautes Regulares baseados em Matrizes de Células**. 2006. 131 f. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2006.

MENEZES, Claudio Carvalho. **Geração Automática de Leiaute Através de Matriz de Células - MARTELO.** 2003. 58 f. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2003.

MORAES, F.; TORRES, L.; ROBERT, M.; AUVERGNE, D. **Estimation of layout densities for CMOS digital circuits**. IEEE PATMOS, 1998. **Proceedings**... Denmark: [s.n], 1998. 61-70.

MURDOCA, Miles J.; HEURING, Vincent P. **Introdução à Arquitetura de Computadores**. Rio de Janeiro: Editora Campus/ Elsevier, 2000.

PATTERSON, David A.; HENNESSY, John L. **Organização e Projeto de Computadores. A Interface Hardware/Software**. 3. ed. Rio de Janeiro: Editora Campus, 2005.

RABAEY, Jan M.; CHANDRAKASAN, Anantha; NIKOLIC, Borivoje. **Digital Integrated Circuits - A Design Perspective.** 2. ed. Upper Saddle River, NJ, USA: Prentice-Hall, 1996.

REIS, Ricardo Augusto da Luz. **Concepção de Circuitos Integrados**. 2. ed. Porto Alegre: Bookman, 2002.

REIS, R. **Redução de Consumo pela Otimização de Componentes**. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 30, 2010, Belo Horizonte. **Anais**... Belo Horizonte: Sociedade Brasileira de Computação, 2010. 371-379.

REIS, R. L.; MORAES, F. G. **Sistemas Digitales - Metodologías de diseño VLSI**. Bogotá: CYTED, 2000.

SANTOS, Cristiano Lopes dos. **Verificação e Otimização de Atraso durante a Síntese Física de Circuitos Integrados CMOS**. 2005. 101 f. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2005.

SCHLOSSER, E. R.; PREDIGER, D. L.; GHISSONI, S.; GIRARDI, A. **Comparação de desempenho de somadores binários inteiros de 28 bits com aplicação em uma unidade aritmética em ponto flutuante**. XXIII Congresso Regional de Iniciação Cientifica e Tecnologia em Engenharia (XXII CRICTE). Joinville-SC: UDESC , 2009.

STALLINGS, W. **Arquitetura e Organização de Computadores**. 5. ed. São Paulo: Prentice Hall, 2002.

The SPARC Architecture Manual – Version 8. SPARC International, Inc. - U.S.A, 1992. Disponível em <http://www.sparc.org/standards/V8.pdf> Acesso em 29 abr. 2011.

VERMA, Ajay K.; BRISK, Philip; IENNE, Paolo. **Challenges in automatic optimization of arithmetic circuits**. In: Symposium on Computer Arithmetic, 19., 2009. **Proceedings**... [S.l.: s.n.], 2009. 213-218.

WAGNER, Flávio Rech; REIS, André Inácio; RIBAS, Renato Perez. **Fundamentos de Circuitos Digitais**. Porto Alegre: Bookman, 2008.

WESTE, Neil H. E.; HARRIS, David Money. **CMOS VLSI Design: A Circuits and Systems Perspective**. 4. ed. USA: Addison-Wesley, 2011.

ZIESEMER, Adriel Mota. **Geração Automática de Partes Operativas de Circuitos VLSI**. . 2007. 90 f. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2007.

APÊNDICE A – Descrição de entrada na ferramenta Espresso

```
.i 13
.o 6
.p 46

0000------000  000000
0000------001  100000
0000------010  100000
0000------011  010000
0000------100  100000
0000------101  010000
0000------110  010000
0000------111  110000
0001-----0-00  000000
0001-----0-01  101000
0001-----0-10  100000
0001-----0-11  000000
0001-----1-00  101000
0001-----1-01  001000
0001-----1-10  000000
0001-----1-11  101000
0010----0----  000000
0010----1----  100000
0011---0-----  000000
0011---1-----  100000
0100-------00  000000
0100-------01  000000
0100-------10  000000
0100-------11  100000
0101-------00  000000
0101-------01  100000
```

0101-------10 100000

0101-------11 100000

0110-------00 000000

0110-------01 100000

0110-------10 100000

0110-------11 000000

0111-------0- 100000

0111-------1- 000000

1000001----00 000001

1000001----01 000001

1000001----10 000100

1000001----11 000001

1000010----00 000010

1000010----01 000001

1000010----10 000100

1000010----11 000010

1000100----00 000100

1000100----01 000001

1000100----10 000100

1000100----11 000100

.e

APÊNDICE B – Descrição de saída na ferramenta Espresso

.i 13

.o 6

.p 30

1000010----00 000010

1000001----10 000100

1000010----10 000100

1000100----01 000001

1000010----11 000010

1000010----01 000001

1000100-----0 000100

1000001----0- 000001

1000100----1- 000100

1000001-----1 000001

0110-------10 100000

0001-----1-00 101000

0-01-----0-01 100000

0-01-----0-10 100000

0001-----1-11 101000

0000------010 100000

0000------100 100000

0111-------0- 100000

0000------001 100000

011--------01 100000

0-00------111 100000

0011---1----- 100000

0001-------01 001000

0000------11- 010000

01-1-------01 100000

0000------1-1 010000

```
0010----1----  100000
0000-------11  010000
0101-------1-  100000
010--------11  100000
.e
```

APÊNDICE C – Equações obtidas na decomposição

$x_0 = \bar{n}\bar{m}$

$x_1 = \bar{n}m$

$x_2 = \bar{n}\bar{l}$

$x_3 = \bar{l}\bar{j}$

$x_4 = \bar{l}j$

$x_5 = lj$

$x_6 = l\bar{j}$

$x_7 = ba$

$x_8 = b\bar{a}$

$x_9 = \bar{b}a$

$x_{10} = \bar{b}\bar{a}$

$x_{11} = x_6e$

$x_{12} = x_5f$

$x_{13} = x_{10} + x_7$

$x_{14} = dx_{13}$

$x_{15} = x_4x_{14}$

$x_{16} = cx_{10}$

$x_{17} = \overline{x_{13}}$

$x_{18} = \bar{c}x_{17}$

$x_{19} = x_{18} + x_{16}$

$x_{20} = x_3x_{19}$

$x_{21} = x_{20} + x_{15} + x_{12} + x_{11}$

$x_{22} = \bar{l}x_7$

$x_{23} = x_4b$

$x_{24} = \overline{x_3}$

$x_{25} = x_9x_{24}$

$x_{26} = x_5\bar{b}$

$x_{27} = x_6x_8$

$x_{28} = x_{27} + x_{26} + x_{25} + x_{23} + x_{22}$

$x_{29} = x_1 x_{28}$

$x_{30} = \bar{j} c x_7$

$x_{31} = j \bar{d} x_{17}$

$x_{32} = x_{31} + x_{30}$

$x_{33} = x_2 x_{32}$

$x_{34} = \overline{x_{10}}$

$x_{35} = c x_{34}$

$x_{36} = x_{35} + x_7$

$x_{37} = x_3 x_{36}$

$x_{38} = x_{14} + x_9$

$x_{39} = x_4 x_{38}$

$x_{40} = n \bar{m}$

$x_{41} = x_{40} x_3$

$x_{42} = \overline{x_9}$

$x_{43} = x_{46} x_9$

$x_{44} = x_{55} x_{42}$

$x_{45} = i x_{44}$

$x_{46} = h \bar{g}$

$x_{47} = x_{46} x_8$

$x_{48} = \bar{h} g$

$x_{49} = x_{48} x_8$

$x_{50} = x_{49} + x_{47}$

$x_{51} = \bar{i} x_{50}$

$x_{52} = x_{51} + x_{45}$

$x_{53} = x_{46} x_{13}$

$x_{54} = \bar{i} x_{53}$

$x_{55} = \bar{h} \bar{g}$

$x_{56} = x_{55} x_9$

$x_{57} = i x_{56}$

$x_{58} = \overline{x_8}$

$x_{59} = x_{48} x_{58}$

$x_{60} = \bar{i} x_{92}$

$$x_{61} = x_{60} + x_{57}$$

$$x_{92} = x_{59} + x_{43}$$

$$x_{93} = x_0 x_{21}$$

APÊNDICE D – Descrição das NAND's e Inversores de um *Slice* da ULA no formato de entrada da ferramenta MARTELO

```
.include libasic

X1 a n1 vcc inv
X2 b n2 vcc inv
X3 c n3 vcc inv
X4 d n4 vcc inv
X5 g n5 vcc inv
X6 h n6 vcc inv
X7 i n7 vcc inv
X8 j n8 vcc inv
X9 l n9 vcc inv
X10 m n10 vcc inv
X11 n n11 vcc inv
X12 n11 n10 n12 vcc nand2
X13 n12 n13 vcc inv
X14 n11 m n14 vcc nand2
X15 n14 n15 vcc inv
X16 n11 n9 n16 vcc nand2
X17 n16 n17 vcc inv
X18 n9 n8 n18 vcc nand2
X19 n18 n19 vcc inv
X20 n9 j n20 vcc nand2
X21 n20 n21 vcc inv
X22 l j n22 vcc nand2
X23 n22 n23 vcc inv
X24 l n8 n24 vcc nand2
X25 n24 n25 vcc inv
X26 b a n26 vcc nand2
X27 n26 n27 vcc inv
```

```
X28 b n1 n28 vcc nand2
X29 n28 n29 vcc inv
X30 n2 a n30 vcc nand2
X31 n30 n31 vcc inv
X32 n2 n1 n32 vcc nand2
X33 n32 n33 vcc inv
X34 n25 e n34 vcc nand2
X35 n23 f n35 vcc nand2
X36 n32 n26 n36 vcc nand2
X37 d n36 n37 vcc nand2
X38 n37 n38 vcc inv
X39 n21 n38 n39 vcc nand2
X40 c n33 n40 vcc nand2
X41 n36 n41 vcc inv
X42 n3 n41 n42 vcc nand2
X43 n42 n40 n43 vcc nand2
X44 n19 n43 n44 vcc nand2
X45 n44 n39 n45 vcc nand2
X46 n45 n46 vcc inv
X47 n35 n34 n47 vcc nand2
X48 n47 n48 vcc inv
X49 n46 n48 n49 vcc nand2
X50 n9 n27 n50 vcc nand2
X51 n21 b n51 vcc nand2
X52 n31 n18 n52 vcc nand2
X53 n23 n2 n53 vcc nand2
X54 n25 n29 n54 vcc nand2
X55 n54 n53 n55 vcc nand2
X56 n55 n56 vcc inv
X57 n52 n51 n57 vcc nand2
X58 n57 n58 vcc inv
X59 n56 n58 n59 vcc nand2
X60 n59 n60 vcc inv
```

X61 n60 n50 n61 vcc nand2

X62 n15 n61 n62 vcc nand2

X63 n8 c n63 vcc nand2

X64 n63 n64 vcc inv

X65 n64 n27 n65 vcc nand2

X66 j n4 n66 vcc nand2

X67 n66 n67 vcc inv

X68 n67 n41 n68 vcc nand2

X69 n68 n65 n69 vcc nand2

X70 n17 n69 n70 vcc nand2

X71 c n32 n71 vcc nand2

X72 n71 n26 n72 vcc nand2

X73 n19 n72 n73 vcc nand2

X74 n73 n74 vcc inv

X75 n37 n30 n75 vcc nand2

X76 n21 n75 n76 vcc nand2

X77 n76 n77 vcc inv

X78 n n10 n78 vcc nand2

X79 n78 n79 vcc inv

X80 n79 n19 n80 vcc nand2

X81 n80 n81 vcc inv

X82 n88 n31 n82 vcc nand2

X83 n82 n105 n83 vcc nand2

X84 n101 n30 n84 vcc nand2

X85 n84 n85 vcc inv

X86 i n85 n86 vcc nand2

X87 h n5 n87 vcc nand2

X88 n87 n88 vcc inv

X89 n88 n29 n89 vcc nand2

X90 n6 g n90 vcc nand2

X91 n90 n91 vcc inv

X92 n91 n29 n92 vcc nand2

X93 n92 n89 n93 vcc nand2

X94 n7 n93 n94 vcc nand2

X95 n94 n86 n95 vcc nand2

X96 n88 n36 n96 vcc nand2

X97 n96 n97 vcc inv

X98 n7 n97 n98 vcc nand2

X99 n98 n99 vcc inv

X100 n6 n5 n100 vcc nand2

X101 n100 n101 vcc inv

X102 n101 n31 n102 vcc nand2

X103 n102 n103 vcc inv

X104 i n103 n104 vcc nand2

X105 n91 n28 n105 vcc nand2

X106 n7 n83 n106 vcc nand2

X107 n106 n104 n107 vcc nand2

X108 n13 n49 n108 vcc nand2

X109 n108 n62 n109 vcc nand2

X110 n109 n110 vcc inv

X111 n110 n70 n111 vcc nand2

X112 n13 n74 n112 vcc nand2

X113 n112 n113 vcc inv

X114 n13 n77 n114 vcc nand2

X115 n114 n115 vcc inv

X116 n81 n95 n116 vcc nand2

X117 n116 n117 vcc inv

X118 n81 n99 n118 vcc nand2

X119 n118 n119 vcc inv

X120 n81 n107 n120 vcc nand2

X121 n120 n121 vcc inv

APÊNDICE E – Descrição do posicionamento para ferramenta MARTELO

bandas 3

41 X104 X99 X98 X7 X85 X101 X84 X90 X6 X87 X5 X82 X80 X78 X4 X66 X37 X38 X2 X30 X26 X28 X1 X32 X33 X40 X3 X71 X51 X63 X8 X22 X19 X18 X24 X20 X12 X9 X10 X11 X14
40 X107 X120 X106 X94 X86 X105 X83 X91 X92 X89 X88 X96 X81 X79 X68 X67 X46 X45 X39 X31 X27 X29 X36 X41 X42 X65 X64 X52 X57 X58 X50 X23 X49 X47 X25 X21 X13 X17 X16 X15
40 X121 X118 X119 X95 X116 X117 X100 X93 X115 X114 X97 X103 X102 X113 X112 X69 X75 X76 X77 X72 X73 X74 X44 X43 X60 X59 X56 X55 X54 X53 X35 X61 X62 X48 X34 X70 X108 X109 X111 X110

APÊNDICE F – Descrição das células criadas para implementação com portas complexas

```
.subckt inv a out vcc gnd
MN1 out a gnd gnd NMOS L=0.35U W=2.0U
MP1 out a vcc vcc PMOS L=0.35U W=4.0U
.ends inv

.subckt nand2 a b out vcc gnd
MN1  i1  a gnd gnd NMOS L=0.35U W=2.0U
MN2 out  b  i1 gnd NMOS L=0.35U W=2.0U
MP1 out  a vcc vcc PMOS L=0.35U W=4.0U
MP2 out  b vcc vcc PMOS L=0.35U W=4.0U
.ends nand2

.subckt nand3 a b c out vcc gnd
MN1  i1  a gnd gnd NMOS L=0.35U W=2.0U
MN2  i2  b  i1 gnd NMOS L=0.35U W=2.0U
MN3 out  c  i2 gnd NMOS L=0.35U W=2.0U
MP1 out  a vcc vcc PMOS L=0.35U W=4.0U
MP2 out  b vcc vcc PMOS L=0.35U W=4.0U
MP3 out  c vcc vcc PMOS L=0.35U W=4.0U
.ends nand3

.subckt nand4 a b c d out vcc gnd
MN1  i1  a gnd gnd NMOS L=0.35U W=2.0U
MN2  i2  b  i1 gnd NMOS L=0.35U W=2.0U
MN3  i3  c  i2 gnd NMOS L=0.35U W=2.0U
MN4 out  d  i3 gnd NMOS L=0.35U W=2.0U
MP1 out  a vcc vcc PMOS L=0.35U W=4.0U
MP2 out  b vcc vcc PMOS L=0.35U W=4.0U
MP3 out  c vcc vcc PMOS L=0.35U W=4.0U
MP4 out  d vcc vcc PMOS L=0.35U W=4.0U
.ends nand4
```

```
.subckt nor2 a b out vcc gnd
MN1 out  a gnd gnd NMOS L=0.35U W=2.0U
MN2 out  b gnd gnd NMOS L=0.35U W=2.0U
MP1 out  a  i1 vcc PMOS L=0.35U W=4.0U
MP2 i1   b vcc vcc PMOS L=0.35U W=4.0U
.ends nor2

.subckt nor3 a b c out vcc gnd
MN1 out  a gnd gnd NMOS L=0.35U W=2.0U
MN2 out  b gnd gnd NMOS L=0.35U W=2.0U
MN3 out  c gnd gnd NMOS L=0.35U W=2.0U
MP1 out  a  i1 vcc PMOS L=0.35U W=4.0U
MP2 i1   b  i2 vcc PMOS L=0.35U W=4.0U
MP3 i2   c vcc vcc PMOS L=0.35U W=4.0U
.ends nor3

.subckt nor4 a b c d out vcc gnd
MN1 out  a gnd gnd NMOS L=0.35U W=2.0U
MN2 out  b gnd gnd NMOS L=0.35U W=2.0U
MN3 out  c gnd gnd NMOS L=0.35U W=2.0U
MN4 out  d gnd gnd NMOS L=0.35U W=2.0U
MP1 out  a  i1 vcc PMOS L=0.35U W=4.0U
MP2 i1   b  i2 vcc PMOS L=0.35U W=4.0U
MP3 i2   c  i3 vcc PMOS L=0.35U W=4.0U
MP4 i3   d vcc vcc PMOS L=0.35U W=4.0U
.ends nor4

.subckt oai1 a b c d e f g h i j l m n o p out vcc gnd
MN1  i1   a gnd gnd NMOS L=0.35U W=2.0U
MN2  i2   b  i1 gnd NMOS L=0.35U W=2.0U
MN3  out  c  i2 gnd NMOS L=0.35U W=2.0U
MN4  i3   d  i1 gnd NMOS L=0.35U W=2.0U
```

```
MN5  i4   e  i3 gnd NMOS L=0.35U W=2.0U
MN6  out  f  i4 gnd NMOS L=0.35U W=2.0U
MN7  i5   g gnd gnd NMOS L=0.35U W=2.0U
MN8  i6   h  i5 gnd NMOS L=0.35U W=2.0U
MN9  out  i  i6 gnd NMOS L=0.35U W=2.0U
MN10 i7   j gnd gnd NMOS L=0.35U W=2.0U
MN11 i8   l  i7 gnd NMOS L=0.35U W=2.0U
MN12 i9   m  i8 gnd NMOS L=0.35U W=2.0U
MN13 out  n  i9 gnd NMOS L=0.35U W=2.0U
MN14 i10  o  i8 gnd NMOS L=0.35U W=2.0U
MN15 out  p i10 gnd NMOS L=0.35U W=2.0U
MP1  out  a i11 vcc PMOS L=0.35U W=4.0U
MP2  out  b i12 vcc PMOS L=0.35U W=4.0U
MP3  out  c i12 vcc PMOS L=0.35U W=4.0U
MP4  i12  d i11 vcc PMOS L=0.35U W=4.0U
MP5  i12  e i11 vcc PMOS L=0.35U W=4.0U
MP6  i12  f i11 vcc PMOS L=0.35U W=4.0U
MP7  i11  g i13 vcc PMOS L=0.35U W=4.0U
MP8  i11  h i13 vcc PMOS L=0.35U W=4.0U
MP9  i11  i i13 vcc PMOS L=0.35U W=4.0U
MP10 i13  j vcc vcc PMOS L=0.35U W=4.0U
MP11 i13  l vcc vcc PMOS L=0.35U W=4.0U
MP12 i13  m i14 vcc PMOS L=0.35U W=4.0U
MP13 i13  n i14 vcc PMOS L=0.35U W=4.0U
MP14 i14  o vcc vcc PMOS L=0.35U W=4.0U
MP15 i14  p vcc vcc PMOS L=0.35U W=4.0U
.ends oai1

.subckt oai2 a b c d e f g h i j l m n out vcc gnd
MN1  i1   a gnd gnd NMOS L=0.35U W=2.0U
MN2  i2   b  i1 gnd NMOS L=0.35U W=2.0U
MN3  i3   c  i2 gnd NMOS L=0.35U W=2.0U
MN4  i4   d  i3 gnd NMOS L=0.35U W=2.0U
```

```
MN5  out  e  i4 gnd NMOS L=0.35U W=2.0U
MN6  i5   f  i3 gnd NMOS L=0.35U W=2.0U
MN7  out  g  i5 gnd NMOS L=0.35U W=2.0U
MN8  i6   h  i2 gnd NMOS L=0.35U W=2.0U
MN9  out  i  i6 gnd NMOS L=0.35U W=2.0U
MN10 i7   j  i2 gnd NMOS L=0.35U W=2.0U
MN11 i8   l  i7 gnd NMOS L=0.35U W=2.0U
MN12 out  m  i8 gnd NMOS L=0.35U W=2.0U
MN13 out  n  i7 gnd NMOS L=0.35U W=2.0U
MP1  out  a vcc vcc PMOS L=0.35U W=4.0U
MP2  out  b vcc vcc PMOS L=0.35U W=4.0U
MP3  out  c  i9 vcc PMOS L=0.35U W=4.0U
MP4  out  d i10 vcc PMOS L=0.35U W=4.0U
MP5  out  e i10 vcc PMOS L=0.35U W=4.0U
MP6  i10  f  i9 vcc PMOS L=0.35U W=4.0U
MP7  i10  g  i9 vcc PMOS L=0.35U W=4.0U
MP8  i9   h i11 vcc PMOS L=0.35U W=4.0U
MP9  i9   i i11 vcc PMOS L=0.35U W=4.0U
MP10 i11  j vcc vcc PMOS L=0.35U W=4.0U
MP11 i11  l i12 vcc PMOS L=0.35U W=4.0U
MP12 i11  m i12 vcc PMOS L=0.35U W=4.0U
MP13 i12  n vcc vcc PMOS L=0.35U W=4.0U
.ends oai2

.subckt oai3 a b c d e out vcc gnd
MN1  i1   a gnd gnd NMOS L=0.35U W=2.0U
MN2  i2   b  i1 gnd NMOS L=0.35U W=2.0U
MN3  out  c  i2 gnd NMOS L=0.35U W=2.0U
MN4  out  d  i2 gnd NMOS L=0.35U W=2.0U
MN5  out  e  i1 gnd NMOS L=0.35U W=2.0U
MP1  out  a vcc vcc PMOS L=0.35U W=4.0U
MP2  out  b  i3 vcc PMOS L=0.35U W=4.0U
MP3  out  c  i4 vcc PMOS L=0.35U W=4.0U
```

```
MP4  i4   d  i3 vcc PMOS L=0.35U W=4.0U
MP5  i3   e vcc vcc PMOS L=0.35U W=4.0U
.ends oai3

.subckt oai4 a b c d out vcc gnd
MN1  i1   a gnd gnd NMOS L=0.35U W=2.0U
MN2  i2   b  i1 gnd NMOS L=0.35U W=2.0U
MN3  out  c  i2 gnd NMOS L=0.35U W=2.0U
MN4  out  d  i1 gnd NMOS L=0.35U W=2.0U
MP1  out  a vcc vcc PMOS L=0.35U W=4.0U
MP2  out  b  i3 vcc PMOS L=0.35U W=4.0U
MP3  out  c  i3 vcc PMOS L=0.35U W=4.0U
MP4  i3   d vcc vcc PMOS L=0.35U W=4.0U
.ends oai4

.subckt oai6 a b c d e f g h out vcc gnd
MN1  i1   a gnd gnd NMOS L=0.35U W=2.0U
MN2  i2   b  i1 gnd NMOS L=0.35U W=2.0U
MN3  i3   c  i2 gnd NMOS L=0.35U W=2.0U
MN4  i4   d  i3 gnd NMOS L=0.35U W=2.0U
MN5  out  e  i4 gnd NMOS L=0.35U W=2.0U
MN6  i5   f  i2 gnd NMOS L=0.35U W=2.0U
MN7  i6   g  i5 gnd NMOS L=0.35U W=2.0U
MN8  out  h  i6 gnd NMOS L=0.35U W=2.0U
MP1  out  a vcc vcc PMOS L=0.35U W=4.0U
MP2  out  b vcc vcc PMOS L=0.35U W=4.0U
MP3  out  c  i7 vcc PMOS L=0.35U W=4.0U
MP4  out  d  i7 vcc PMOS L=0.35U W=4.0U
MP5  out  e  i7 vcc PMOS L=0.35U W=4.0U
MP6  i7   f vcc vcc PMOS L=0.35U W=4.0U
MP7  i7   g vcc vcc PMOS L=0.35U W=4.0U
MP8  i7   h vcc vcc PMOS L=0.35U W=4.0U
.ends oai6
```

```
.subckt aoi1 a b c d e out vcc gnd
MN1  i1   a gnd gnd NMOS L=0.35U W=2.0U
MN2  i2   b  i1 gnd NMOS L=0.35U W=2.0U
MN3  i2   c gnd gnd NMOS L=0.35U W=2.0U
MN4  out  d  i2 gnd NMOS L=0.35U W=2.0U
MN5  out  e  i2 gnd NMOS L=0.35U W=2.0U
MP1  out  a  i3 vcc PMOS L=0.35U W=4.0U
MP2  out  b  i3 vcc PMOS L=0.35U W=4.0U
MP3  i3   c vcc vcc PMOS L=0.35U W=4.0U
MP4  out  d  i4 vcc PMOS L=0.35U W=4.0U
MP5  i4   e vcc vcc PMOS L=0.35U W=4.0U
.ends aoi1
```

APÊNDICE G – Descrição das conexões das portas de um *Slice* da ULA no formato de entrada da ferramenta ASTRAN

X1 n p1 vcc gnd inv

X2 m p2 vcc gnd inv

X3 p1 p2 p3 vcc gnd nand2

X4 l p4 vcc gnd inv

X5 j p5 vcc gnd inv

X6 p3 p4 p5 p6 vcc gnd nor3

X7 p3 p4 j p7 vcc gnd nor3

X8 b0 p8 vcc gnd inv

X9 p8 a0 p9 vcc gnd nor2

X10 a0 p10 vcc gnd inv

X11 b0 p10 p11 vcc gnd nor2

X12 p9 p11 p12 vcc gnd nor2

X13 p12 p13 vcc gnd inv

X14 c p14 vcc gnd inv

X15 p3 p15 vcc gnd inv

X16 p6 p14 p13 c p8 p10 p7 p12 d p15 l j f p5 a1 p16 vcc gnd oai1

X17 p8 p10 p17 vcc gnd nor2

X18 d p18 vcc gnd inv

X19 p1 p4 p5 c p17 j p18 p13 p19 vcc gnd oai6

X20 p1 m l p5 p9 j p8 p38 p11 p4 j b0 p17 p20 vcc gnd oai2

X21 p6 c b0 a0 p17 p21 vcc gnd oai3

X22 p21 p22 vcc gnd inv

X23 p7 d p12 p11 p23 vcc gnd oai4

X24 p23 p24 vcc gnd inv

X25 p1 m l j p25 vcc gnd nor4

X26 i p26 vcc gnd inv

X27 h p27 vcc gnd inv

X28 g p28 vcc gnd inv

X29 p25 p26 h p28 p29 vcc gnd nand4

X30 p25 p26 p27 g p30 vcc gnd nand4

X31 p25 i p27 p28 p31 vcc gnd nand4

X32 p13 p29 p32 vcc gnd nor2

X33 p9 p33 vcc gnd inv

X34 p30 p29 p33 p31 p11 p34 vcc gnd aoi1

X35 p11 p35 vcc gnd inv

X36 p29 p31 p35 p30 p9 p36 vcc gnd aoi1

X37 p16 p19 p20 p37 vcc gnd nand3

X38 p4 p5 p38 vcc gnd nand2