UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
CAMPUS MOSSORÓ
CURSO DE CIÊNCIAS DA COMPUTAÇÃO

ÁLAMO GONÇALVES DA SILVA

ESTUDO E IMPLEMENTAÇÃO DE TÉCNICAS DE MULTIPLICAÇÃO EM HARDWARE E ANÁLISE COMPARATIVA DE DESEMPENHO EM FPGAS

MOSSORÓ-RN
2014

ÁLAMO GONÇALVES DA SILVA

ESTUDO E IMPLEMENTAÇÃO DE TÉCNICAS DE MULTIPLICAÇÃO EM HARDWARE E ANÁLISE COMPARATIVA DE DESEMPENHO EM FPGAS

Monografia apresentada à Universidade Federal Rural do Semi-Árido - UFERSA, campus Mossoró, para a obtenção do título de Bacharel em Ciências da Computação.
Orientador(a): Profº. Dr. Leonardo Augusto Casillo – UFERSA.

MOSSORÓ-RN
2014

Dados Internacionais de Catalogação na Publicação (CIP)
Biblioteca Central Orlando Teixeira (BCOT)
Setor de Informação e Referência

S586e Silva, Álamo Gonçalves da

Estudo e implementação de técnicas de multiplicação em *hardware* e análise comparativa de desempenho em FPGAS./ Álamo Gonçalves da Silva-- Mossoró, 2014.
61f.: il.

Orientador: Prof. Dr. Leonardo Augusto Casillo

Monografia (Graduação em Ciência da Computação) – Universidade Federal Rural do Semi-Árido. Pró-Reitoria de Graduação.

1. Metodologia de multiplicação. 2. FPGA. 3. Dispositivos reconfiguráveis. I. Título.

RN/UFERSA/BCOT /405-14 CDD: 004

Bibliotecária: Vanessa Christiane Alves de Souza Borba
CRB-15/452

ÁLAMO GONÇALVES DA SILVA

ESTUDO E IMPLEMENTAÇÃO DE TÉCNICAS DE MULTIPLICAÇÃO EM HARDWARE E ANÁLISE COMPARATIVA DE DESEMPENHO EM FPGAS

Monografia apresentada à Universidade Federal Rural do Semi-Árido – UFERSA, campus Mossoró, para a obtenção do Título em Bacharel de Ciências da Computação.

Aprovado em : 26/02/2014

BANCA EXAMINADORA

Profº. Dr. Leonardo Augusto Casillo - UFERSA
Presidente

Profº. Dr. Danniel Cavalcante Lopes - UFERSA
Primeiro Membro

Dênis Freire Lopes Nunes
Segundo Membro

RESUMO

No contexto atual de projeto e desenvolvimento de circuitos integrados, uma das abordagens mais difundidas em campo é a utilização de dispositivos reconfiguráveis em partes ou em todo o ciclo do projeto. Dentre os mais utilizados está o FGPA (*Field Programmable Gate Array*), o qual, através de linguagens de descrição de hardware, pode ter sua matriz de elementos lógicos internos organizada – e reorganizada a qualquer tempo – para atuar como um circuito específico, direcionado à uma determinada função. Através dessa abordagem é possível criar, implantar e testar sistemas completos antes de sua produção em massa, diminuindo por exemplo, o custo de projeto sobre *chips* defeituosos e o tempo para que determinada tecnologia chegue ao mercado, denominado *time-to-market*. Inseridos em meio a uma gama comumente utilizados em projetos de *hardware*, os circuitos multiplicadores exercem papel importante nos sistemas atuais, cada vez mais robustos e com maior capacidade de processamento. Entretanto, esses mesmos sistemas têm por muitas vezes restrições de projeto no que diz respeito à área ocupada em *chip*, velocidade, tempo e energia consumida, para citar os mais comuns. Sendo assim, pesquisas direcionadas à essas partes específicas são essenciais para o contínuo aperfeiçoamento dos sistemas modernos. Este trabalho mostra a análise metodológica, bem como o ciclo completo de projeto, de três metodologias voltados à circuitos multiplicadores. Serão estudadas características intrínsecas das mesmas, o que resultará em conclusões acerca de quais metodologias são compatíveis aos vários tipos de sistemas de *hardware* mediante suas restrições de projetos. Ainda, serão relatados aspectos relevantes ao controle de circuitos desenvolvidos nos FPGAs da fabricante Altera®, uma das maiores empresas de dispositivos reconfiguráveis atualmente.

Palavras-chaves: Metodologias de multiplicação. FPGA. Dispositivos reconfiguráveis. Controle de circuitos.

ABSTRACT

In the current context of the design and development of integrated circuits, one of the most diffused approaches in the field is the use of reconfigurable devices in parts or in the project as a whole. Among the most used is the FPGA (Field Programmable Gate Array), which can be programed through hardware description languages having its array of logic elements internally organized - and reorganized at any time - to act as a specific circuit, directed to a particular function. Through this approach it is possible to create, deploy and test complete systems before mass production, reducing the project cost about defective chips and the time for that particular technology reaches the market, called time-to-market. Inserted in a bank of a commonly used hardware designs, multiplier circuits play an important role in current systems, more robust and with more powerful processing capability. However, these systems frequently have design constraints in the project concerning the area occupied in chip, speeds, time and consumed energy – to mention the most common ones. Thus, directly pointed research towards these specific parts are essential to the continuous improvement of modern systems. This work shows the methodological analysis, as well as the full project cycle, of three methodologies focused on multiplier circuits. Will be studied intrinsic features of them, which will result in conclusions about which methods are compatible with various types of hardware systems regarding its project restrictions. In addition, relevant aspects about the control circuits found in Altera® FPGAs will be reported.

Key words: Multiplier methodologies. FPGA. Reconfigurable devices. Control circuits.

LISTA DE ABREVIATURAS E SIGLAS

CPLD – Complex Programmable Logic Device

FPGA – Field Programmable Gate Array

HDL – Hardware Description Language

VHDL – VHSIC Hardware Description Language

EDA – Eletronic Design Automation

ASIC – Application Specific Integrated Circuit

RAM – Random Access Memory

SRAMs – Static Random Access Memory

LUT – Lookup Table

PLL – Phased Locked Loop

ULA – Unidade Lógica e Aritmética

RTL – Resgister Transfer Level

LISTA DE FIGURAS

Figura 1 – Ilustração da estrutura geral de um antifusível.....14
Figura 2 – Célula de memória flash.....15
Figura 3 – Estrutura de uma célula SRAM 16
Figura 4 – Arquitetura de Blocos Lógicos em Ilha 17
Figura 5 – Elemento lógico mínimo genérico de um FPGA.....17
Figura 6 – Elemento lógico da família de dispositivos CYCLONE II.....18
Figura 7 – Diagrama de blocos da família CYCLONE II, modelos EP2C35.....19
Figura 8 – Exemplo de multiplicação binária.....20
Figura 9 – Exemplo de completo de um.....21
Figura 10 – Exemplo de complemento de dois 21
Figura 11 – Algoritmo de multiplicação generalizado para números de 4 bits 24
Figura 12 – Diagrama de blocos do circuito somador para números de 4 bits.....25
Figura 13 – Diagrama parcial de blocos do circuito implementado 26
Figura 14 – Diagrama de blocos do circuito multiplicador gerado pela ferramenta EDA utilizada.....26
Figura 15 – Localização dos PLLs em dispositivos CYCLONE II.....29
Figura 16 – Diagrama de blocos resultante da descrição da metodologia proprietária 29
Figura 17 – Algoritmo de Booth para multiplicação de números inteiros.....31
Figura 18 – Diagrama de blocos de um PLL em configuração mínima.....35
Figura 19 – Simulação do caso de teste 1 no circuito proprietário.....36
Figura 20 – Simulação do caso de teste 2 no circuito proprietário.....36
Figura 21 – Diagrama de blocos do circuito proprietário.....37
Figura 22 – Diagrama de blocos do circuito que implementa a multiplicação do Booth 38

LISTA DE TABELAS

Tabela 1 – Números binários na forma de complemento de 2 possíveis com 4bits..21
Tabela 2 – Passo a passo da execução do algoritmo de Booth para a operação.....32
Tabela 3 – Relatório de recursos utilizados na descrição do circuito multiplicador...34
Tabela 4 – Relatório de recursos utilizados na descrição dos circuitos proprietários37
Tabela 5 – Relatório de recursos utilizados na descrição do circuito que implementa o algoritmo de Booth 38

Sumário

1 INTRODUÇÃO 8
1.1 OBJETIVOS 11
1.2 ORGANIZAÇÃO DO TRABALHO 12
2 REFERENCIAL TEÓRICO 13
2.1 FPGA’s 13
2.2 MULTIPICAÇÃO 19
2.3 APLICAÇÕES 22
3 METODOLGIA 23
3.1 CIRCUITO MULTIPLICADOR 23
3.2 ALTERA® 27
3.3 MULTIPLICADOR DE BOOTH 30
4 RESULTADOS E DISCUSSÕES 33
4.1 CIRCUITO MULTIPLICADOR 33
4.2 ALTERA® 34
4.3 BOOTH 37
5 CONSIDERAÇÕES FINAIS 39
REFERÊNCIAS 41
APÊNDICES

1 INTRODUÇÃO

Nos primórdios do desenvolvimento dos sistemas digitais, empresas e pesquisadores voltaram seus esforços em busca de soluções para acelerar a execução dos dispositivos. Altos investimentos foram dispendidos para que se pudesse evoluir os circuitos projetados, principalmente no que se diz respeito aos materiais utilizados para a fabricação dos mesmos. Após o surgimento dos transistores, na década de 1960, Gordon E. Moore projetou – o que depois de alguns anos viria a tornar-se a Lei de Moore – que os sistemas digitais teriam o dobro de transistores ao mesmo custo a cada período de dezoito meses (PATTERSON; HENNESSY, 2005).

Após alguns anos, depois de a projeção de Moore confirmar-se, empresas tomaram essa lei como uma diretriz de desenvolvimento em seus projetos. Circuitos integrados tornaram-se cada vez mais robustos e potentes, fornecendo base para o desenvolvimento de sistemas com componentes cada vez mais rápidos. Ainda, a difusão de sistemas computacionais – apoiada por pesquisas de novos componentes e otimização – diminuía o custo do *hardware* utilizado. Diante da diminuição dos custos, projetos externos às grandes empresas podiam ser executados. Tais projetos tinham um cunho mais específico, voltados às necessidades de cada cliente (PATTERSON; HENNESSY, 2012).

Para preencher a lacuna recém surgida foram desenvolvidos dispositivos de lógica programável. Ao contrário dos *chips* ordinários, esses poderiam ser programados – e reprogramados – de acordo a necessidade do cliente. Os primeiros circuitos dessa natureza eram basicamente matrizes de portas lógicas, programáveis a partir fórmulas booleanas e inseridas através de *softwares*. Dessa forma os transistores eram interconectados de forma a produzir a função pretendida (HAUCK; DEHON, 2008).

A partir dos primeiros dispositivos de lógica programáveis, surgiram inúmeras linhas de pesquisa que resultaram em uma gama de circuitos ainda mais complexos e multiuso, os quais poderiam ser alterados quantas vezes fosse necessário ao cliente ou projetista. CPLDs (*Complex Programmable Logic Devices*) e FPGAs (*Field Programmable Gate Array*), por exemplo, assumiram papel importantíssimo no desenvolvimento de *chips* com lógica complexa, pois tornou-se rápido e barato alterar, embarcar e testar um projeto de *hardware* comparado método de projeto

conhecido, o qual envolvia um grande custo de fabricação do chip para que a lógica do mesmo pudesse ser avaliada (TOCCI; WIDMER; MOSS, 2007).

Voltando a Lei de Moore, existem projeções diferentes a respeito da evolução dos circuitos integrados nos dias atuais. Não basta apenas aumentar o números de transistores em um *chip* para torná-lo melhor e mais rápido. Há também limitações de área, calor dissipado, fonte de energia que alimenta o sistema – principalmente baterias – e a especificidade do circuito são pontos essenciais a serem considerados ainda na fase de projeto do *hardware*. Ainda mais, a demanda por aplicações dedicadas – sistemas desenvolvidos para executar tarefas específicas e que geralmente carregam alguma restrição de projeto, seja pela necessidade de rapidez de execução, energia consumida, área ocupada e tempo de execução (sistemas de tempo real) – obriga um estudo prévio do impacto a ser causado pelos circuitos a serem desenvolvidos. Arquiteturas diferentes bem como a organização dos componentes internos do *chip* devem ter suas funcionalidades testadas a analisadas para que se possam atender as limitações impostas.

Dentre os vários tipos de dispositivos de *hardware* que permitem a programação e reprogramação de suas interconexões estão os FPGAs – que em tradução livre significa matriz de portas programáveis em campo. Basicamente, este tipo de dispositivo é organizado a partir de unidades mínimas programáveis, os elementos lógicos. Esses elementos são organizados em forma de uma matriz, as quais são comandadas por unidades controle locais. Por fim, na periferia desses chips, estão as conexões externas – portas de entrada e saída. Por fim, a denominação “programáveis em campo” refere-se ao fato de o ser possível alterar a configuração dos *chips* (conexões dos elementos lógicos) a qualquer tempo, inclusive se o *hardware* já estiver sendo utilizado em campo de produção, como uma plataforma automatizada (CARDOSO; FERNANDES, 2007).

A definição da sigla “FPGA” pode dar a entender que há somente a matriz de elementos lógicos e que a mesma deve ser configurada com todos os circuitos necessários ao projeto. Entretanto, existem plataformas de desenvolvimento para FPGAs que contém, por exemplo, processadores, portas USB, conexões de memória externa, dentre outros, transformando-as em sistemas completos e permitindo ao projetista preocupar-se somente com o desenvolvimento do circuito pretendido. A relativa facilidade de conceber e implantar um circuito com lógica específica tornou-se um dos principais fatores para que os FPGAs ganhassem mais

espaço a cada dia. Através de uma linguagem de descrição de *hardware*, as HDLs (*Hardware Description Languages*) é possível descrever como os elementos lógicos e de controle serão interconectados. Dentre as mais conhecidas e utilizadas, estão o VHDL e o Verilog. Essa relativa facilidade de “criar” um dispositivo apenas escrevendo – e descrevendo – seu funcionamento e abstraindo conceitos e habilidades práticas de microeletrônica na fase de projeto faz os estudantes, projetistas e empresas olharem a tecnologia do FPGA com bons olhos (HAUCK; DEHON, 2008).

A utilização de linguagens de descrição, assim como as linguagens de programação, trazem uma série de benefícios aos projetistas e empresas desenvolvedoras. A facilidade de reuso de código para circuitos comumente utilizados (caso dos registradores, por exemplo) e a criação de bibliotecas de códigos, os quais em última análise são uma sorte de dispositivos de *hardware* que podem ser implantados, editados ou otimizados a qualquer tempo sem a necessidade de ter que produzir outro *chip*. Diminuindo o custo de projeto, consequentemente o custo de produção em massa e a pesquisa por novas estruturas e tipos de organização de *hardware* fazem o produto final ser mais atrativo financeiramente aos consumidores.

Normalmente, linguagens de descrição são utilizadas em conjunto com ferramentas EDA (Eletronic Design Automation), as quais auxiliam consideravelmente durante todo o ciclo do projeto. Antes do advento desse tipo de *software*, projetistas de *hardware* esboçavam o projeto dos circuitos, fabricavam-no – artesanalmente em alguns casos – e só então poderiam testá-lo. Hoje existe a possibilidade de projetar e simular o circuito em um ambiente completamente virtual antes de embarcar o sistemas no dispositivo físico.

Apesar de serem largamente utilizados, os circuitos desenvolvidos em dispositivos programáveis não tem o mesmo desempenho que *chips* ASIC (*Application Specific Integrated Circuit*), os quais são fabricados somente com os circuitos especificados em seus projetos. Apesar disso, *chips* ASIC são geralmente projetados e testados em FPGAs antes de iniciar sua produção. Esses *chips* específicos, maioria nos dias atuais, são geralmente utilizados em ambientes controlados e bem definidos, o que gera limitações de projetos para esses sistemas, sejam elas a área ocupada, organização dos componentes internos, fonte de energia, desempenho, dentre outros (LANDIS, 1996).

A busca por sistemas capazes de processar uma maior quantidade de dados em menos tempo – apesar de suas naturezas específicas, na maioria dos casos – passa necessariamente pela análise dos componentes internos. O mesmo resultado pode ser alcançado de diferentes formas. Apesar disso, as restrições de projetos impõem que se deva analisar cuidadosamente qual estrutura empregar levando-se em consideração fatores como área ocupada em *chip*, velocidade, energia necessária e tempo, para citar os mais comuns. E dentre os circuitos comumente empregados em projetos de *hardware* estão os multiplicadores, essenciais no tratamento de operações complexas, as quais são indispensáveis em sistemas robustos (STALLINGS, 2010).

Esta trabalho apresentará a análise de três metodologias empregadas no desenvolvimento de circuitos multiplicadores, bem como todas as etapas de projeto e os resultados obtidos em simulações. Do confronto dessas metodologias, já conhecidas e utilizadas em projetos recentes, são esperados dados capazes de nortear as decisões durante o desenvolvimento do sistema.

Os resultados advindos dessa análise poderão produzir avanços consideráveis na forma de implementar circuitos utilizados no processamento de sinais. Sendo esse um dos usos mais frequentes para FPGAs, entende-se que avanços são necessários a fim de torná-los mais robustos – capazes de processar entradas mais fiel e rapidamente.

1.1 OBJETIVOS

Este trabalho tem como objetivo principal analisar e comparar aspectos relevantes de três metodologias distintas aplicadas à multiplicação em *hardware*. Para tanto, serão utilizados conceitos de sistemas digitais para a implementação de um circuito multiplicador baseado somente em operações lógicas, um segundo operando algoritmicamente e uma terceira implementação, esta última através de uma função proprietária fornecida pela fabricante do FPGA utilizado neste trabalho. Serão comparadas características como área ocupada em *chip* e desempenho dos mesmos, além da profunda investigação a respeito da redução de ciclos de relógio quando utilizados circuitos descritos a partir de funções proprietárias.

1.2 ORGANIZAÇÃO DO TRABALHO

O seguinte trabalho está organizado com segue: o Capítulo 2 examina o referencial teórico necessário para o entendimento das sessões posteriores; o Capítulo 3 mostra como foram desenvolvidas as metodologias bem como seus circuitos resultantes; o Capitulo 4 analisa e discute os dados obtidos a partir de testes. Logo após apresenta-se a Conclusão seguida da sessão de Referências. Por fim, serão listados os casos de teste utilizados no Apêndice A. No Apêndice B estão contidos os testes para o circuito da sessão 3.1. Os Apêndices C e D contém as simulações da sessão 3.3.

2 REFERENCIAL TEÓRICO

2.1 FPGA’s

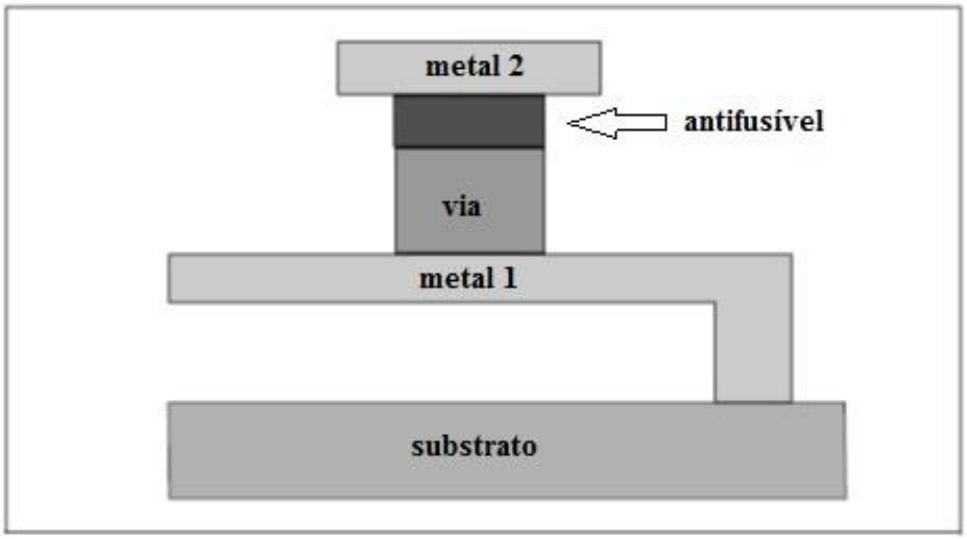
Da sessão de introdução deste trabalho, temos que FPGAs são basicamente constituídos por uma matriz de elementos lógicos capazes de receber configurações diversas ao longo de sua vida útil. Mais ainda, FPGAs convergem o melhor dos dois mundos dentro da computação: *hardware* e o *software*. Ao longo dos anos esses dispositivos tiveram suas arquiteturas e elementos constituintes aprimorados a tal ponto que são capazes de ter embarcados em suas matrizes, guardadas as devidas proporções de área, sistemas computacionais completos e suficientemente robustos para executar milhões de operações por segundo. Além disso, a flexibilidade - antes somente encontrada à nível de *software* – fornecida pela reconfiguração dos mesmos tornam o projeto, implementação e principalmente a atualização de circuitos desenvolvidos nessas plataformas tarefas relativamente simples e consideravelmente menos maçante do que a programação de ASIC, por exemplo.

Ter a possibilidade de programar o circuito de acordo com as especificações do cliente e ainda poder refazer esse procedimento após iniciado o funcionamento do sistema é o que define os FPGAs. A fim de possibilitar essa característica, existem vários métodos de configuração que podem ser aplicados aos elementos programáveis para que os blocos lógicos sejam devidamente conectados, de acordo com o circuito projetado. Um dos métodos utilizados para a configuração dessas conexões é o emprego dos antifusíveis (do inglês, *antifuse*), os quais atuam inversamente aos fusíveis.

Ao ser fabricado, o FPGA tem todas as suas interconexões normalmente abertas com antifusíveis entre as extremidades de todos os caminhos de dados que possam vir tornarem-se vias físicas após a programação. Para programar esse tipo de dispositivo, uma alta corrente é fornecida nos pontos de interconexão, resultando no derretimento do filamento interno do antifusível e unindo as extremidades de forma a criar uma via física entre as mesmas – como ilustrado na Figura 1. Após a repetição desse procedimento para cada célula programável o circuito projetado estará descrito no FPGA. Esse tipo de abordagem tem como principal vantagem a maior velocidade do circuito resultante bem como um menor consumo e dispersão de energia - uma vez que as conexões são físicas – e portanto, não precisam ser

mantidas em um determinado nível lógico constantemente. Claramente, não há possibilidade de reprogramação nesse tipo de dispositivo, motivo o qual os mesmos são raramente utilizados, salvo projetos nos quais as vantagens obtidas desse tipo de FPGA são indispensáveis (HAUCK; DEHON, 2008).

Figura 1 – Ilustração da estrutura geral de um antifusível



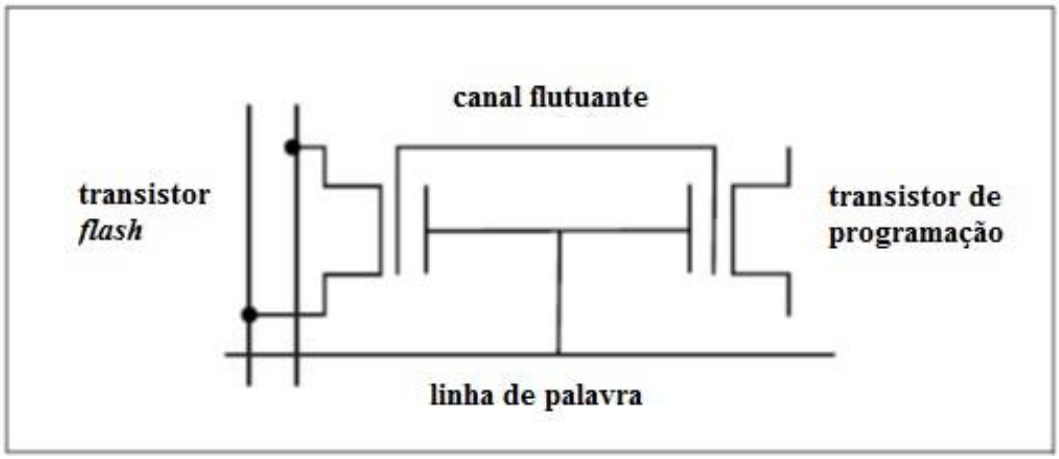
FONTE: HAUCK; DEHON, 2008

Tendo em vista que a possibilidade de reprogramação dos FPGAs é indispensável na maioria dos projetos, tecnologias alternativas que permitam essa característica compõem a maior parte dos dispositivos programáveis presentes no mercado. Ao utilizar memórias do tipo *flash*, por exemplo, garante-se a reprogramação e ainda um baixo consumo de energia interna, dado que as mesmas são produzidas com uma pequena quantidade de transistores. Entretanto, a potência relativamente alta do processo de gravação aliada a uma baixa velocidade durante o mesmo processo, além da limitação dos números de vezes em que se pode reescrever nesse tipo de memória fazem com que os FPGAs com essa abordagem também não sejam comumente utilizados (HAUCK; DEHON, 2008).

A Figura 2 exibe um modelo esquemático do tipo de abordagem supracitado. Nele podem ser identificados três elementos principais: o transistor *flash* - que controla operações de leitura e escrita na célula de memória, localizado no lado oposto ao transistor de programação – utilizado somente durante a programação no

dispositivo – e um canal flutuante capaz de controlar a atuação dos dois transistores descritos.

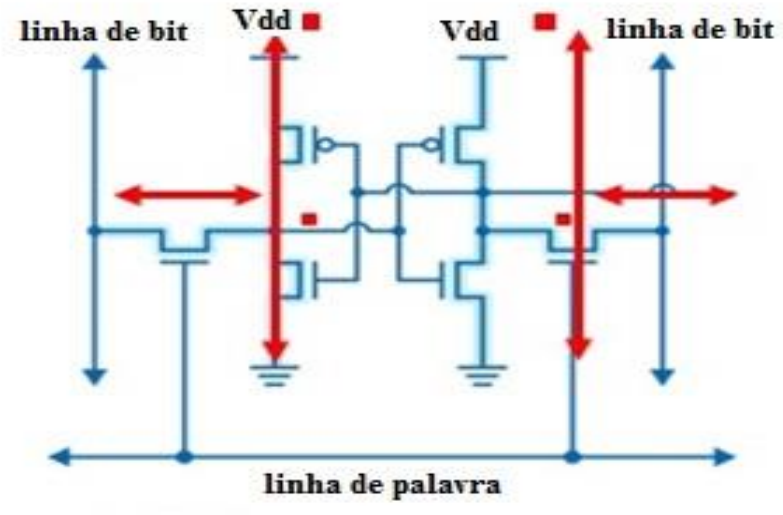
Figura 2 – Célula de memória flash



FONTE: HAUCK; DEHON, 2008

Por último, existem os dispositivos que utilizam memórias RAM (*Ramdom Access Memory*) estáticas – SRAMs (*Static Ramdom Access Memory*) – como elementos constituintes do FPGA. Apesar do maior número de transistores necessários em sua fabricação – aumentando o consumo de energia – as SRAMs possibilitam infinitas reconfigurações à maiores velocidades se comparadas às memórias *flash*. Entretanto, por serem voláteis, as SRAMs necessitam de uma corrente energética constante a fim de manter o estado lógico configurado – fato evidenciado pelas linhas vermelhas do esquema de uma SRAM, mostrado na Figura 3. Apesar disso, o fato de FPGAs modernos não serem constituídos apenas por *chips* reconfiguráveis, mas também por vários circuitos anexos que sustentam a autoconfiguração, tornam os FPGAs baseados em SRAMs os mais difundidos e utilizados atualmente. Sendo assim, é razoável que os dispositivos utilizados nesse trabalho fossem pertencentes a esse grupo (HAUCK; DEHON, 2008).

Figura 3 – Estrutura de uma célula SRAM

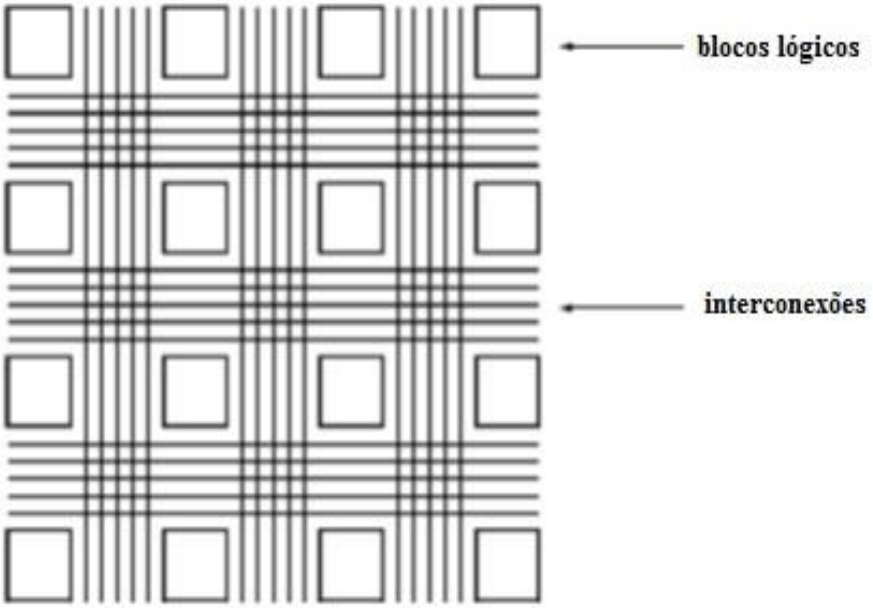


FONTE: <http://www.microsemi.com/products/fpga-soc/low-power>

O FPGA utilizado no projeto e simulações deste trabalho foi o modelo EP2C35, presente na plataforma de desenvolvimento DE2 da Altera®. Em sessões posteriores, abordar-se-ão mais detalhes específicos da arquitetura desde dispositivo. Por ora, apenas alguns de seus digramas de bloco com características gerais análogas a maioria dos FPGAs serão utilizados para auxiliar a visualização das descrições.

A arquitetura mais utilizada atualmente em dispositivos FPGA é a do tipo “ilha” como mostra a Figura 4, na qual os blocos lógicos são arranjados em uma matriz bidimensional e flutuam em um “mar” de conexões. Vale-se ressaltar a figura apenas apresenta didaticamente o tipo de arquitetura em questão. Em uma arquitetura real, os blocos lógicos são conectados fisicamente aos blocos conexões além de serem necessários elementos de entrada e saída de dados (HAUCK; DEHON, 2008).

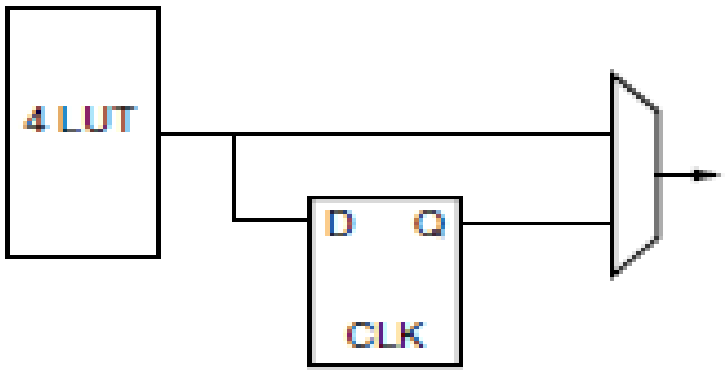
Figura 4 – Arquitetura de Blocos Lógicos em Ilha



FONTE: HAUCK; DEHON, 2008

Dentro de cada bloco lógico estão contidas as partes mínimas programáveis dos FPGAs, os elementos lógicos. Cada elemento desse tipo possui, no mínimo, uma LUT (*Lookup Table*) e uma célula de memória (geralmente um *flip-flop*). A Figura 5 ilustra esse conceito. LUT é um tipo de *hardware* o qual implementa uma tabela-verdade – com quatro entradas, normalmente (4-LUT) – e funciona basicamente como um multiplexador e, juntamente com a célula de memória, forma o elemento computacional básico para o funcionamento do FPGA.

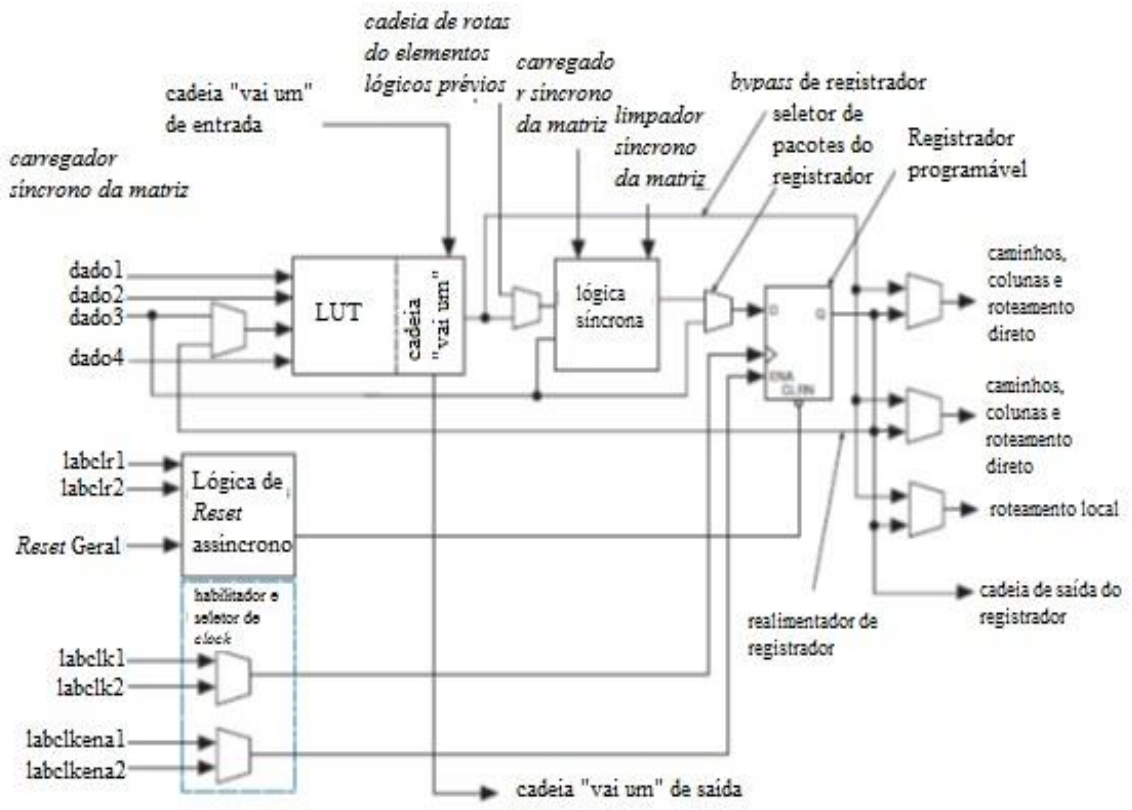
Figura 5 – Elemento lógico mínimo genérico de um FPGA



FONTE: HAUCK; DEHON, 2008

Ao contrário do que se possa inferir a partir das ilustrações simplórias das figuras anteriores, os elementos lógicos dos FPGAs modernos têm muito mais do que somente LUTs e *flip-flops* internamente. A Figura 6 mostra o elemento lógico do dispositivo utilizado neste trabalho.

Figura 6 – Elemento lógico da família de dispositivos CYCLONE II

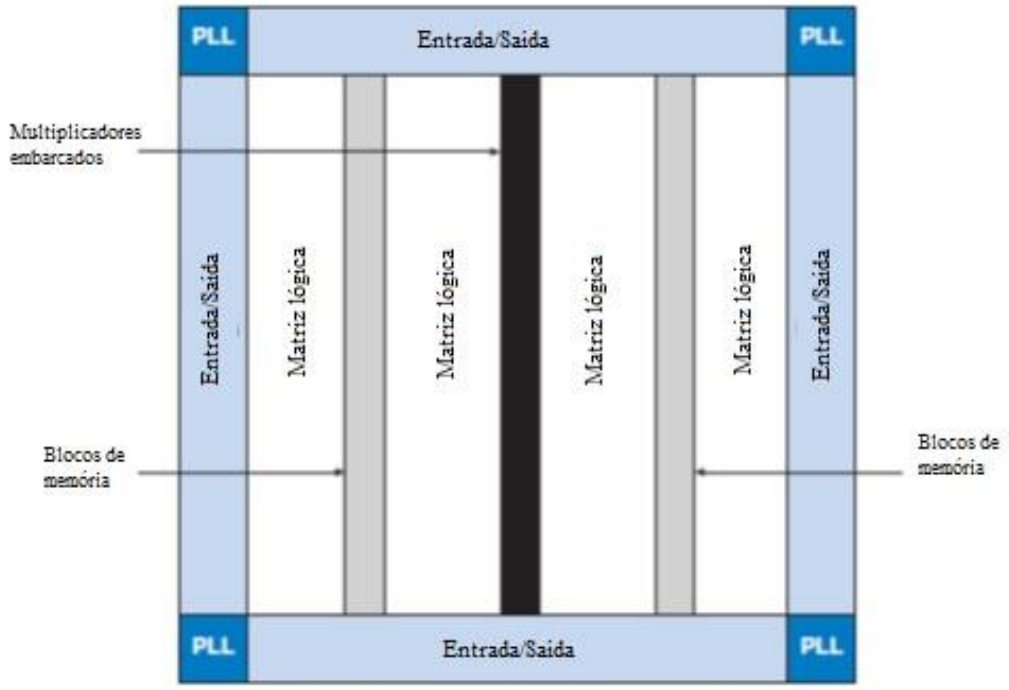


FONTE: CYCLONE..., 2008

Percebe-se agora uma complexidade exponencialmente maior do que a apresentada na Figura 5. Circuitos específicos dedicados aos sinais de controle interno como roteamento e cadeias de “vai um” além das estruturas programáveis, como *clock* e *reset*, os quais são individualmente acessados para cada elemento lógico. Ainda, a Figura 7 mostra o diagrama de blocos geral do FPGA utilizado. Levando-se em consideração que o modelo apresentado é de baixo custo e direcionado à área acadêmica – e, portanto, não tem incorporadas as tecnologias mais avanças de seu tempo – percebe-se que, além das estruturas básicas de blocos lógicos e de conexões margeados por entradas e saídas, há também blocos de memória, multiplicadores locais e blocos PLL (*Phased Locked Loops*) – estes

serão abordados de forma mais aprofundada na sessão que apresenta a metodologia proprietária utilizada. Apesar de não embarcar tecnologia de ponta, mesmo os FPGAs de baixo custo permitem uma grande flexibilidade e performance (CYCLONE..., 2008).

Figura 7 – Diagrama de blocos da família CYCLONE II, modelos EP2C35



FONTE: CYCLONE..., 2008

2.2 MULTIPICAÇÃO

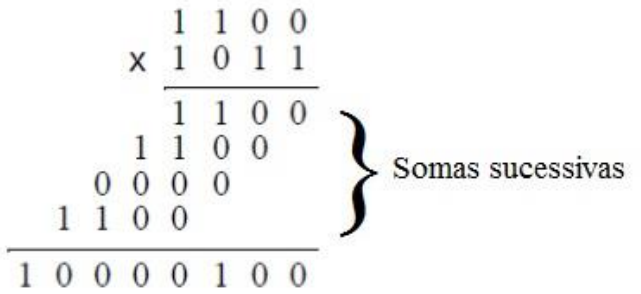
Como discutido anteriormente, os FPGAs modernos têm sistemas completos anexados às matrizes programáveis. Sendo assim, o projetista pode apegar-se somente ao circuito específico a ser utilizado e valer-se de processadores e memórias embarcadas para agilizar o tempo despendido no projeto. Além dos circuitos citados há também multiplicadores dedicados, os quais são geralmente utilizados em circuitos desenvolvidos para o processamento de sinais – que na maioria dos casos usam cálculos complexos para determinar os parâmetros estudados.

Levando em consideração que atualmente os FPGAs são empregados, na maior parte das vezes, em algum tipo de processador de sinais, entende-se

necessário o estudo acerca dos multiplicadores de forma a encontrar a metodologia – bem como a arquitetura – ideal para cada tipo de objetivo, dadas as restrições de projeto. Todas essas metodologias partem de um mesmo ponto comum: a multiplicação binária.

A Figura 8 exemplifica o método base de multiplicação, o qual é geralmente ensinado nas disciplinas básicas de cursos que envolvem sistemas digitais. O algoritmo envolve operações E (*AND*) entre o multiplicando e cada bit do multiplicador além de somas sucessivas dos produtos parciais resultantes das operações lógicas anteriores (TOCCI; WIDMER; MOSS, 2007).

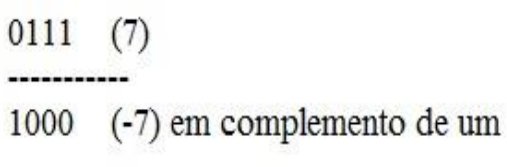
Figura 8 – Exemplo de multiplicação binária



FONTE: Autoria própria

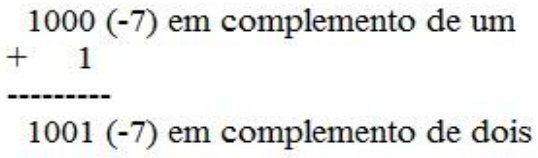
Seguindo o algoritmo apresentado, muitas variantes podem ser implementadas ainda obtendo o mesmo resultado. Entretanto, quando os projetos necessitam suportar operações de multiplicação com números negativos deve-se alterar o circuito a fim de que o mesmo possa utilizar a forma de “complemento de dois” – forma de representar o valor negativo de um dado número binário – e obter o produto correto. Para se chegar ao “complemento de dois”, primeiramente deve-se aplicar o “complemento de um” ao valor binário inicial, como mostra a Figura 9, que nada mais é do inverter todos os bits do número original – sendo esse procedimento um caso particular para o sistemas de numeração de base 2. Por último, como mostra a Figura 10, deve-se somar “1” ao “complemento de um” calculado anteriormente (TOCCI; WIDMER; MOSS, 2007).

Figura 9 – Exemplo de completo de um



FONTE: Autoria própria

Figura 10 – Exemplo de complemento de dois



FONTE: Autoria própria

Após os processos exibidos nas Figuras 9 e 10, tem-se o número binário negativo na forma de “complemento de dois”, a qual será utilizada neste trabalho. A Tabela 1 mostra os números de quatro bits possíveis na forma de “complemento de dois”.

Tabela 1 – Números binários na forma de complemento de 2 possíveis com 4bits

Decimal (positivo)	Binário (se o número é positivo, não há alteração)	Decimal (negativo)	Binário (C2)
0	0000	-1	1111
1	0001	-2	1110
2	0010	-3	1101
3	0011	-4	1100
4	0100	-5	1011
5	0101	-6	1010
6	0110	-7	1001
7	0111	-8	1000

Fonte: Autoria própria

2.3 APLICAÇÕES

Por vezes, projetos de processadores reconfiguráveis necessitam de abordagens diferentes do algoritmo de multiplicação da sessão anterior para que se possa implementar satisfatoriamente as restrições de projeto. Para tanto, tempo deve ser despendido para estudar qual o melhor método – envolvendo implementação e testes dos mesmos. Seria de bom uso uma base de comparação – como a gerada por este trabalho – entre as metodologias usuais a fim servir como base de conhecimento para projetos de *hardware*, podendo diminuir o tempo dos mesmos (CASILLO, 2013).

Mais uma vez, circuitos utilizados para o processamento de sinais devem ter, idealmente, área reduzida e grande poder de multiplicação – principalmente os circuitos de tempo real – além de grande velocidade nos ciclos de relógio. Isso pode também ser estendido para projetos que requerem ULAs (Unidade Lógica e Aritmética) especiais, modificadas de forma a cumprir requisitos específicos.

3 METODOLGIA

O presente trabalho tem o intuito principal de desvendar alguns aspectos particulares a respeito de metodologias conhecidas e bastante utilizadas na implementação de circuitos multiplicadores. Para tanto, foram selecionadas três vertentes aparentemente bastante distintas em aspectos importantes como área ocupada em *chip*, velocidade de execução, quantidade de ciclos de relógio – quando possível. Além disso, detalhes de configuração da tecnologia proprietária do FPGA utilizado são analisadas e descritas pormenorizadamente a fim de servir de base para projetos que apliquem tal tecnologia.

Ambos os tipos de ferramentas, *hardware* e *software*, empregados nos projetos das metodologias apresentas são fabricadas e desenvolvidas pela empresa Altera®, uma das maiores fabricantes de dispositivos reconfiguráveis do mundo. A suíte de desenvolvimento por *software* utilizada foi o *Quartus II® versão 13.0sp1* juntamente com o *ModelSim-Altera® versão 10.1d* – ferramenta de testes voltada ao programa EDA utilizado.

Sabendo-se do poder dos FPGAs e levando em consideração que projetos atuais utilizam um barramento de dados de pelo menos 32 bits, todos os circuitos resultantes têm entradas para números dentro de tal magnitude e saídas compatíveis com o maior resultado possível dadas entradas dessa proporção, 64 bits. Dessa forma, os resultados obtidos neste trabalho podem realmente ter impacto prático nesses projetos que utilizem multiplicação para números inteiros.

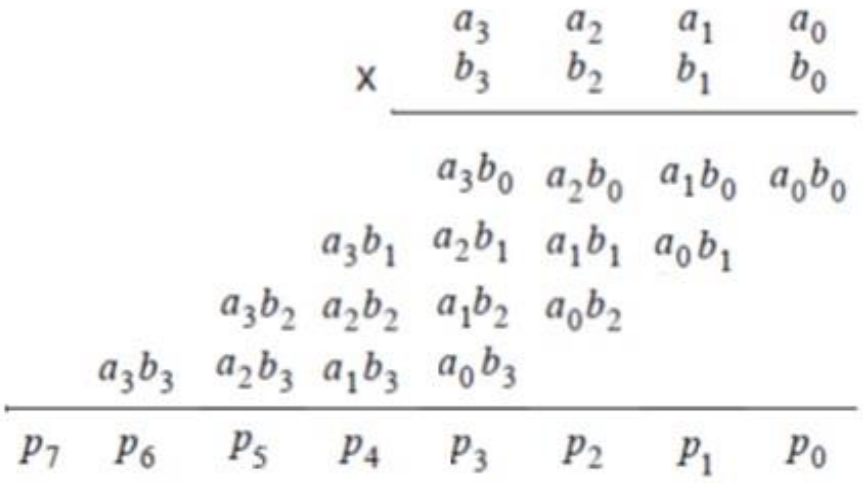
3.1 CIRCUITO MULTIPLICADOR

Partindo-se do algoritmo de multiplicação apresentado na sessão “Multiplicadores” deste trabalho, foi possível implementar um circuito multiplicador utilizando apenas expressões lógicas, o que em suma traduz-se em portas lógicas e propagação de sinais – tal metodologia é considerada a tradução daquele algoritmo.

Levando-se em conta que o diagrama de blocos para a metodologia apresentada nesta subseção seria particularmente grande, fato esse que levaria à uma má estruturação de seu conteúdo, podendo ocasionar dificuldade na identificação e interpretação das imagens, serão utilizadas imagens análogas utilizando números representados em 4 bits apenas – sendo uma questão de escala

inferir o tamanho real de um circuito com números de 32 bits. A Figura 11 mostra a articulação geral da execução da multiplicação previamente discutida.

Figura 11 – Algoritmo de multiplicação generalizado para números de 4 bits

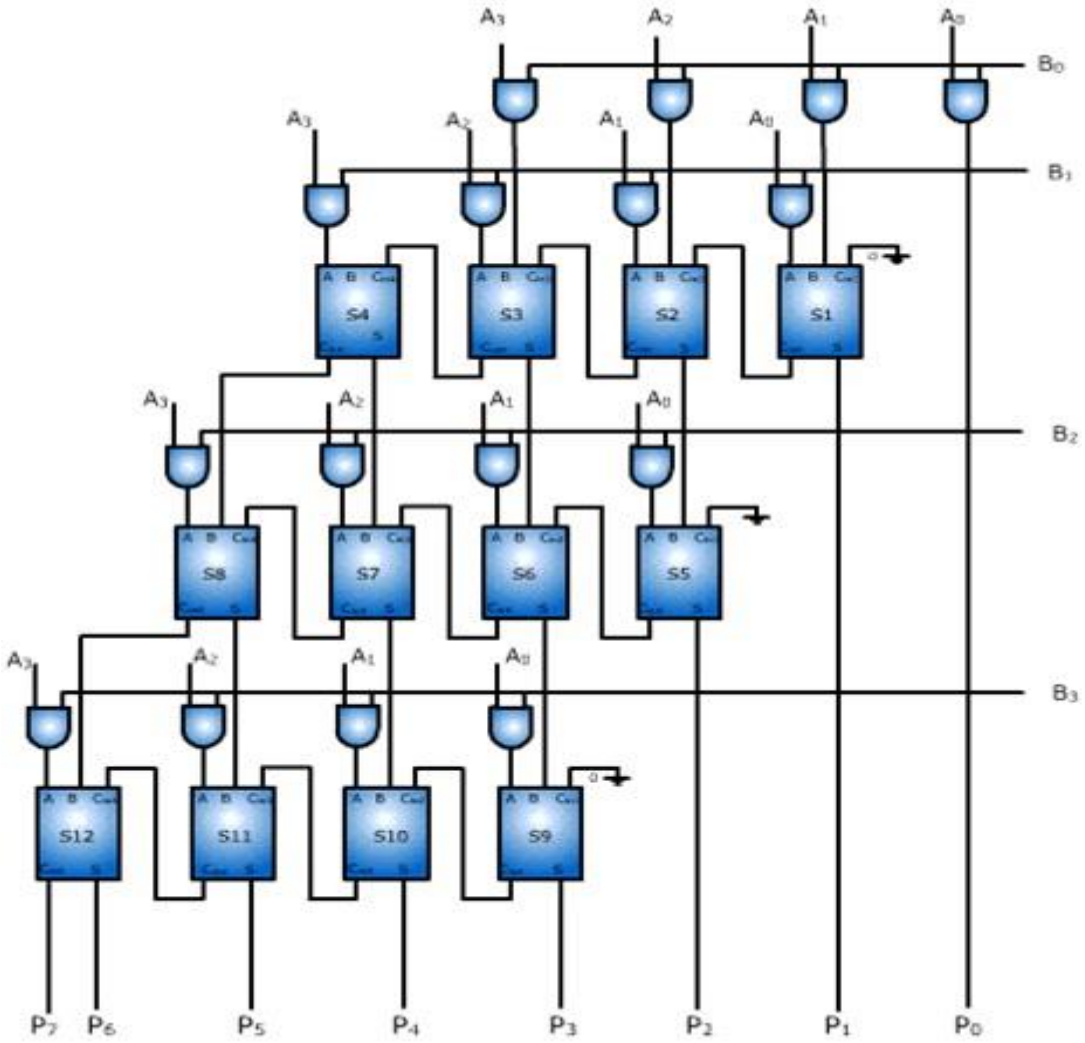


FONTE: Autoria própria

A figura mostrada acima exemplifica didaticamente o que seria necessário para implementar-se o circuito multiplicador capaz de lidar corretamente com todas as operações lógicas e propagações de sinais envolvidas. Para tanto, são utilizados somadores completos para efetuar as sucessivas somas executadas tendo os produtos parciais como entrada.

Ao executarmos o algoritmo para os valores generalizados da figura anterior, tem-se que o primeiro produto parcial é somado ao segundo e o resultado dessa operação serve como entrada para a próxima operação envolvendo o terceiro produto parcial. Dessa forma, serão encadeados tantos somadores quantos forem os produtos parciais e um último meio somador para efetuar a operação entre a última soma parcial e o produto da multiplicação. Cada elo da cadeia será também responsável por passar o “vai um” (em inglês, *carry*) gerado pelas somas anteriores, como pode ser visto na Figura 11. Para tornar a visão dos blocos mais prática e clara, é exibido na Figura 12 o diagrama de blocos parcial do circuito descrito.

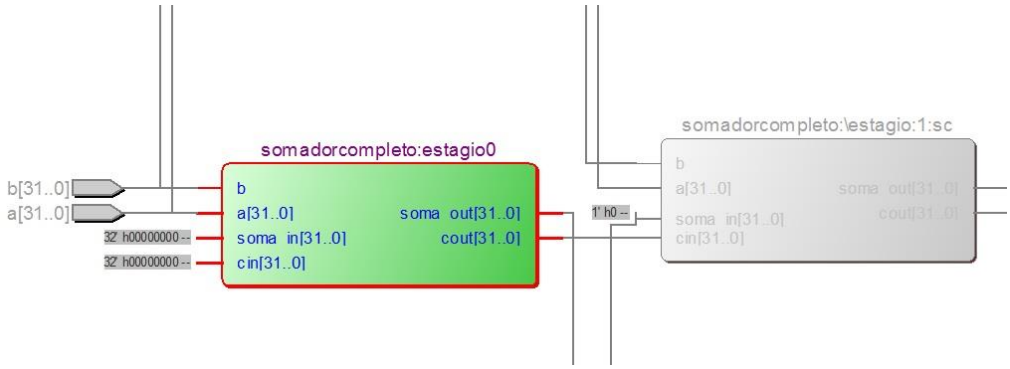
Figura 12 – Diagrama de blocos do circuito somador para números de 4 bits



FONTE: Autor Desconhecido

A descrição do circuito multiplicador foi efetuada tomando como base as figuras anteriores e efetuando-se um aumento na escala para que se pudesse utilizar números de 32 bits. O diagrama de blocos gerado pelo *Quartus II*® é exibido na Figura 13.

Figura 13 – Diagrama parcial de blocos do circuito implementado

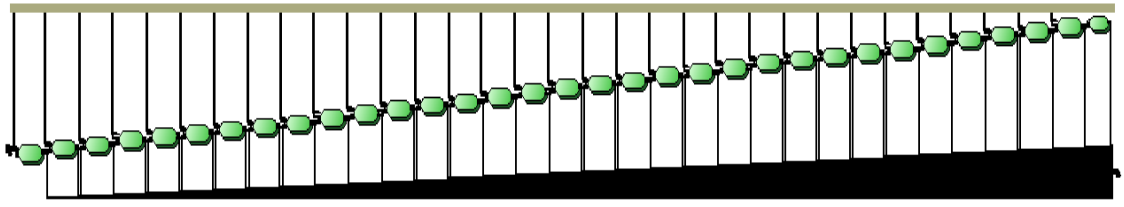


FONTE: Quartus II – versão 13.0.1

Nota-se que não existe controle por meio de ciclos de relógio (*clock*, em inglês), pois as operações lógicas são executadas concorrentemente a nível de *hardware*. De fato, o controle nesse tipo de abordagem é feito externamente ao multiplicador. Sendo assim, o módulo que encapsular o circuito precisa ter algum tipo de controle no que diz respeito à chegada dos números nas entradas do mesmo.

Nota-se, na Figura 14, a presença de um bloco ligeiramente menor – localizado à direita na imagem, no canto superior – a presença do último estágio de soma, esta última sendo necessária para que o produto parcial geral seja somado ao resultado da última soma sucessiva gerada pelo circuito.

Figura 14 – Diagrama de blocos do circuito multiplicador gerado pela ferramenta EDA utilizada



FONTE: Quartus II – versão 13.0.1

Por valer-se de operações concorrentes, espera-se que essa abordagem execute os casos de teste em um menor tempo. Entretanto, a área ocupada pode vir a tornar-se um problema dependendo das restrições do projeto no qual esse circuito

seja aplicado. Por não conter ciclos de relógio, apenas fatores não relacionados a esse tipo de controle serão analisados no capítulo Resultados.

3.2 ALTERA®

Um dos motivos da grande difusão no uso dos FPGAs é a possibilidade de reuso de código na criação de circuitos empregados em mais de um projeto. O usuário pode facilmente armazenar arquivos em linguagem de descrição de *hardware* e criar uma biblioteca de circuitos para uso futuro. Entretanto, existem alguns circuitos de uso comum os quais têm suas descrições não tão triviais quanto o de uma unidade lógica e aritmética simples, por exemplo. Por esse motivo, as fabricantes de FPGAs têm incorporado em seus programas de desenvolvimento bibliotecas de propriedade intelectual as quais contém códigos de circuitos que podem ser utilizados em qualquer projeto, aumentando a rapidez de desenvolvimento e livrando o projetista da tarefa de descrever circuitos secundários aos de seu interesse. E dentre esses vários tipos de circuitos, existem multiplicadores proprietários que podem ser facilmente configurados – dentro do *software* de desenvolvimento – em formato “caixa-preta”, no qual apenas as entradas, saídas e sinais de controle externos são especificados visualmente através da interface de usuário sem que o mesmo tenha que preocupar-se com a forma que o circuito final será descrito.

A segunda metodologia estudada neste trabalho foi a utilização de multiplicadores baseados em funções distribuídas pela fabricante do FPGA através de sua ferramenta EDA. Utilizando a função LPM_MULTI, foi configurado um bloco multiplicador de números de 32 bits controlado por ciclos de relógio externo. A referida função utiliza internamente um circuito multiplicador por somas sucessivas com propagação de “vai um” análogo ao circuito descrito e implementado na sessão anterior. Entretanto, agora é possível utilizar números negativos – com bit de sinal – nas multiplicações efetuadas (LPM..., 1996).

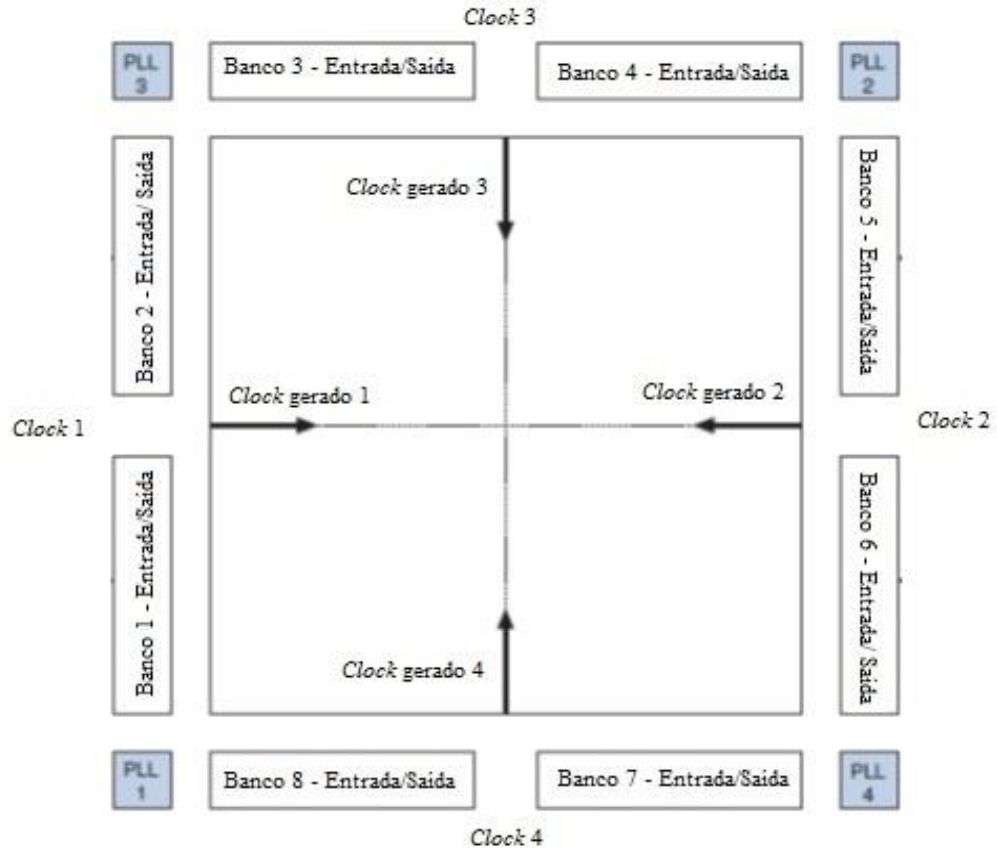
Apesar da aparente facilidade, a utilização de funções fornecidas pelo fabricante do FPGA envolve cuidados especiais na parte de controle. Ao aplicar esse tipo de abordagem, o *software*, a partir de sua biblioteca, cria o circuito especificado. Entretanto, existe um relógio geral para todas as partes proprietárias

do FPGA que opera à uma frequência padrão, sendo o mesmo aplicado à todas as funções que utilizem esse tipo de controle. Nesse caso, por ser um circuito bastante simples – basicamente um circuito multiplicador controlado por um relógio – talvez a configuração padrão seja suficiente, mas na maioria dos casos o projetista deseja alterar a frequência de controle a fim de tornar as medições e testes mais precisas. O modo de como mudar essa configuração – pouco divulgada nos documentos fornecidos pela fabricante vem impedindo que projetistas de várias instituições alterem as frequências dos sinais de controle internos, fato esse que motivou a pesquisa direcionada para o método de controle de relógio em circuitos proprietários.

Há, na maioria dos FPGAs modernos, blocos específicos para o controle de sinais, os *Phased-Locked Loops* (em tradução livre, “Elo Travado em Fase”), capazes de modular sinais a partir de uma entrada fornecida mantendo a fase de ambos – sinais fornecido e gerado – sincronizadas. Constituídos internamente por pelo menos um oscilador e um detector de fase, o circuito PLL é essencialmente utilizado em processamento de sinais.

A partir das informações do parágrafo anterior e de estudos realizados nos documentos de especificação do FPGA utilizado – os *datasheets* – além de discussões em fóruns internacionais de pesquisadores de *hardware*, descobriu-se a existência de uma função proprietária específica capaz de configurar os PLLs e gerar sinais de controle internos que podem ser usados em qualquer módulo do projeto, seja ele proprietário ou de autoria própria do projetista. Para o caso dos dispositivos *Cyclone II*® da Altera®, a Figura 15 mostra a localização dos PLLs (CYCLONE..., 2007).

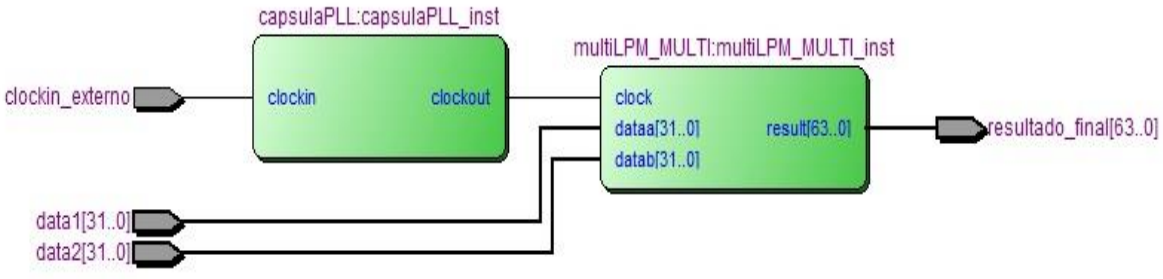
Figura 15 – Localização dos PLLs em dispositivos CYCLONE II



FONTE: CYCLONE..., 2007

O diagrama de blocos do circuito resultante da aplicação da metodologia proprietária é exibido na Figura 16. Além de ter a parte multiplicativa – por assim dizer – existe também o bloco que faz o acesso aos circuitos PLL do FPGA, o qual gera um relógio interno com frequência diferente do sinal de entrada externo.

Figura 16 – Diagrama de blocos resultante da descrição da metodologia proprietária



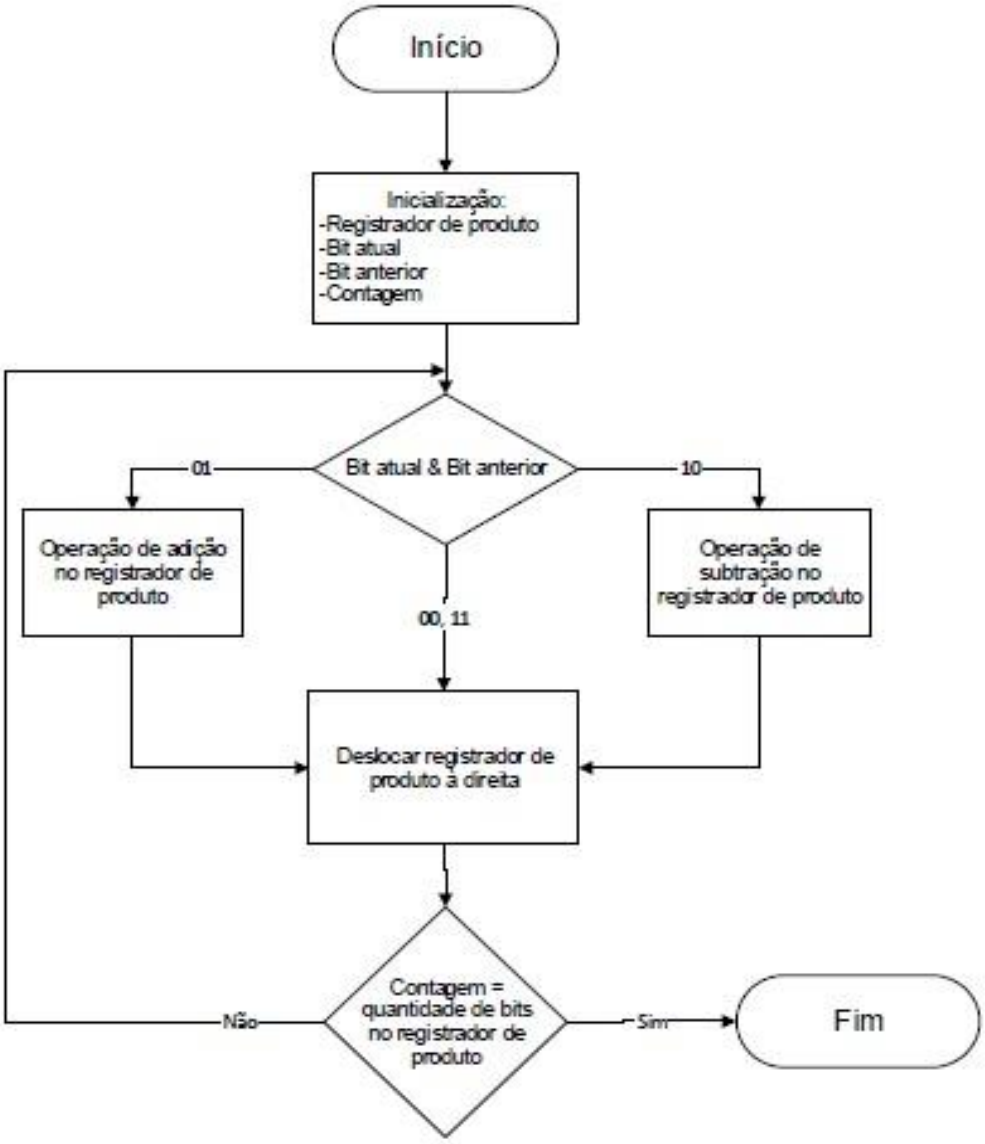
FONTE: Quartus II – versão 13.0.1

3.3 MULTIPLICADOR DE BOOTH

O algoritmo de multiplicação de Booth é uma das metodologias comumente empregadas em ULAs atualmente. Esse método aborda a operação aritmética em questão do ponto de vista que em que há somente operações de soma e deslocamentos durante a execução do algoritmo base (da sessão Multiplicadores), mais precisamente sobre os produtos parciais. Dessa forma, alterando a abordagem ao algoritmo base, é possível utilizar apenas um acumulador juntamente com um módulo de deslocamento operando sempre no registrador de produto, obtendo a princípio um circuito menor que os anteriormente analisados neste trabalho (PATTERSON; HENNESSY, 2005).

O fluxograma apresentado na Figura 17 ilustra a lógica empregada na implementação da metodologia em discussão. Inicialmente acontece a inicialização dos registradores com os valores as serem multiplicados, especialmente o que vai armazenar o produto – este é dividido ao meio. A este também são adicionados um bit a mais em cada metade para indicar sua positividade ou negatividade. A metade mais significativa do registrador de produto é inicialmente composta por zeros e o multiplicando na metade menos significativa. Existem ainda alguns bits especiais para controle, sendo o “bit atual” e o “bit anterior”. A união desses bits lado a lado deverá fornecer o código de operação a ser efetuada no registrador de produto durante as iterações do algoritmo. A saber, os códigos “00” e “11” indicam que o será operado um deslocamento à direita em todo o registrador de produto; “01” informa ao circuito que execute uma soma entre a parte mais significativa do produto com o multiplicando; e por fim, “10” faz com que ocorra uma subtração entre o multiplicando e o produto. Na primeira iteração, o bit anterior é “0” – dado que não houve iterações anteriores. O bit atual sempre armazenará o bit menos significativo do registrador de produto.

Figura 17 – Algoritmo de Booth para multiplicação de números inteiros



FONTE: Autoria própria

Para que a didática na explicação do algoritmo seja completa, a Tabela 2 mostra um exemplo de execução do algoritmo de Booth. Mais uma vez, por motivos de estruturação do trabalho, será apresentado um exemplo com números formados por menos bits do que o circuito desenvolvido. Entretanto, esse mesmo exemplo deve ser suficiente para elucidar definitivamente a lógica da multiplicação de Booth.

Tabela 2 – Passo a passo da execução do algoritmo de Booth para a operação

Iteração	Registrador de produto	Multiplicando	Bit atual	Bit anterior	Operação executada no registrador de produto
0	00000 11100	00010	0	0	Deslocamento
1	00000 01110	00010	0	0	Deslocamento
2a	00000 00111	00010	1	0	Subtração entre a metade mais significativa do registrador de produto e o multiplicando
2b	11110 00111	Deslocamento			
3	01111 00011	00010	1	1	Deslocamento
4	00111 10001	00010	1	1	Deslocamento
5a	00011 11000	00010	0	1	Adição do multiplicando à metade mais significativa do registrador de produto
5b	00101 11000	Resultado final da multiplicação			

FONTE: Autoria própria

Utilizando a tabela anterior, e seguindo os passos do fluxograma da Figura 17, é possível identificar a lógica utilizada na descrição do circuito resultante. De todas as metodologias apresentadas até então, espera-se que esta seja a abordagem que necessita do maior tempo para calcular o resultado final apropriado. Isso se deve ao fato de que o circuito precisa de um ciclo de relógio para cada bit do produto final, o qual é composto por 66 bits – além dos 64 bits padrão são adicionados 2 bits extras para o tratamento da magnitude numérica. Entretanto, levando-se em consideração que esse mesmo circuito utiliza poucos módulos, espera-se que a área ocupada seja a menor dentre todas as metodologias analisadas.

A partir da análise dos casos de teste será possível avaliar e discutir sobre a utilização pontual de metodologias voltadas à projetos específicos levando em consideração as restrições dos mesmos. Em suma, será criada uma base de conhecimento para metodologias multiplicativas em *hardware* a qual servirá de alicerce em implementações futuras.

4 RESULTADOS E DISCUSSÕES

No intuito de avaliar as metodologias propostas neste trabalho em seus pontos principais, foram desenvolvidos dois casos de teste. Primeiramente foram analisadas operações para um grupo de números inteiros positivos de 32 bits – aleatórios – simulados dois a dois – multiplicando e multiplicador. Em seguida, foram avaliadas as metodologias que suportam números negativos – na forma de complemento de 2 – igualmente testadas com multiplicações dois a dois envolvendo números negativos. Ambos os testes, apesar de simplórios, cumprem o papel de mostrar as diferenças entre os circuitos implementados.

Outro fator que motivou a aplicação de tais métodos de teste foi a falta de um controle total dos circuitos. Partindo-se da premissa de que seria estudada a essência de cada abordagem, a inserção de módulos de controle resultaria em um aumento do número de elementos lógicos empregados em cada circuito – salvo o modelo proprietário, o qual usa intrinsecamente os módulos de controle dedicados do FGPA – gerando um acréscimo de área desnecessária e ponto em cheque as metodologias.

Após a fase de testes dos circuitos implementados, percebeu-se o considerável número de simulações geradas pela ferramenta EDA. Portanto, com o objetivo de prezar pela organização e fácil interpretação deste trabalho, todos os casos de teste estão presentes no final do mesmo, inseridos no Apêndice A.

4.1 CIRCUITO MULTIPLICADOR

Por tratar-se de uma abordagem que utiliza apenas expressões lógicas para efetuar os cálculos dos produtos, e sabendo-se que a presença de módulos de controle afetaria a essência desta metodologia, o circuito em questão fica sujeito somente à vazão padrão dos barramentos internos do FPGA – o que poderia tornar a operação mais lenta em alguns modelos. Além disso, a área em chip ocupada foi consideravelmente maior do que nas outras duas implementações.

A Tabela 3 mostra os parâmetros colhidos a partir dos relatórios das ferramentas EDA utilizadas. É possível observar que o FPGA utilizado, apesar de ter um baixo custo, tem robustez suficiente para ser programado com circuitos maiores, os quais utilizariam um mais elementos lógicos disponíveis. Pouco mais de 2.017

unidades lógicas passíveis de programação e um total de 128 pinos foram utilizados para conceber o circuito.

Tabela 3 – Relatório de recursos utilizados na descrição do circuito multiplicador

Total de elementos lógicos (33.216)	Total de registradores (0)	Total de pinos (475)	Total de pinos virtuais (0)	Total de memoria bits (483.840)	Elementos de multiplicadores de 9-bits embarcados (70)	Total de PLL's (4)
2.017	0	128	0	0	0	0

FONTE: Relatório de compilação – Quartus II – versão 13.0.1

Para que se possa obter resultados realmente comparativos, é necessário que os casos de teste sejam igualmente aplicados à todos os circuitos. Por esse motivo, os números aleatórios obtidos encontram-se entre 1 e 2.147.483.647 em base decimal. A limitação à esse intervalo deveu-se ao fato de que os outros circuitos interpretariam como negativos os números que superassem tal limite.

Seguindo as previsões, o circuito multiplicador que utiliza apenas expressões lógicas ocupa mais área em *chip* por encadear vários somadores para calcular os produtos parciais. Sua velocidade de execução manteve-se constante à 32,02ns em todas as multiplicações efetuadas – essa velocidade pode variar entre modelos de FPGA dependendo da vazão natural das conexões internas – como pode ser verificado no Apêndice B.

4.2 ALTERA®

A segunda metodologia a ser posta em prova foi a que utiliza uma função de multiplicação proprietária fornecida pela fabricante de dispositivo, a Altera®. A "LPM_MULT" como é chamada, utiliza multiplicadores dedicados do FPGA para executar os cálculos. Neste trabalho, a função citada foi configurada para operar de acordo com as especificações previamente explicitadas, com entradas de 32 bits e saída com o dobro de tamanho, além de suportar um relógio para o controle dos circuitos dedicados. Entretanto, como discutido em sessões anteriores, a utilização de funções proprietárias infere que o relógio do circuito funciona sempre à uma

frequência padrão, a qual não pode ser alterado salvo quando utilizado recursos – igualmente proprietários – específicos voltados ao controle de sinais presentes no FPGA.

Existem no dispositivo utilizado, o *Cyclone II EP2C35*, um total de quatro blocos dedicados a operar sobre sinais diversos. Tais blocos podem ser configurados pelo usuário e incorporados ao projeto, tornando possível a alteração da frequência padrão dos dispositivos. Tais blocos são chamados PLL e são capazes de gerar e modular sinais a partir de outros. A Figura 18 exibe como apresenta-se um bloco PLL em sua configuração mais simples.

Figura 18 – Diagrama de blocos de um PLL em configuração mínima

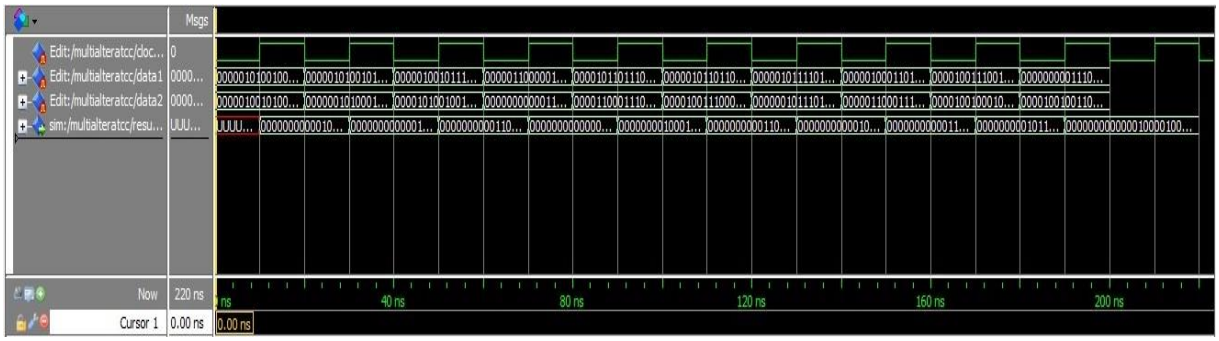
FONTE: Quartus II – versão 13.0.1

O impacto resultante da inserção desse PLL no projeto que abordava a metodologia proprietário foi mínimo. Uma vez que o circuito atuante encontra-se fora da matriz programável do FPGA, nenhum elemento lógico tampouco registradores necessitam ser utilizados. Sendo assim, torna-se salutar empregar tal artifício a fim de tornar os sinais utilizados coerentes com o projeto. Apesar disso, analogamente ao que foi explicado na subseção anterior, a utilização da PLL nesse circuito poderia causar uma interpretação errônea dos casos de testes – além de ser necessário a inserção do mesmo em todas as implementações deste trabalho. Portanto, o circuito no qual os testes foram executados não fazia uso da PLL, estando o mesmo sujeito ao relógio padrão para execução dos casos.

Adicionalmente ao Caso de Teste 1, também foi executado o Caso de Teste 2 – localizado também no Apêndice A - o qual faz uso de números negativos. A Figura 19 mostra a execução do primeiro seguida do segundo caso, exibido na Figura 20.

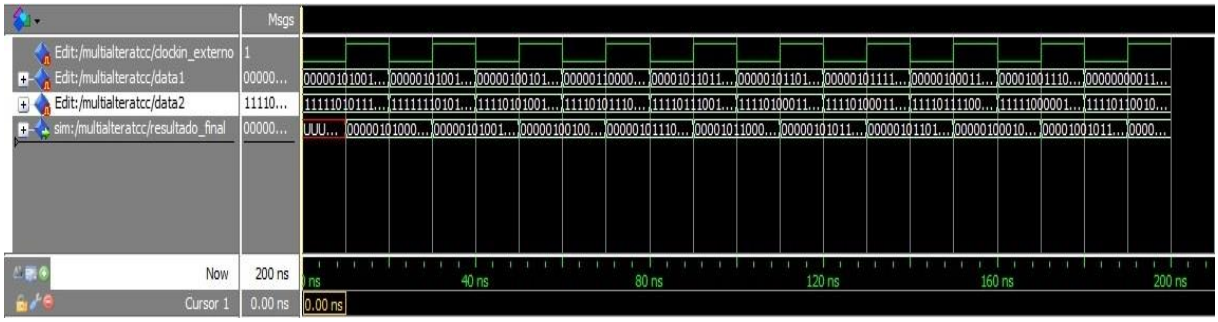
Em ambas percebe-se que as multiplicações são executadas em apenas um ciclo de relógio. Sendo assim, o tempo para o cálculo de um produto entre números – positivos ou negativos – de 32bits será igual ao tempo do período do relógio que rege o circuito.

Figura 19 – Simulação do caso de teste 1 no circuito proprietário



FONTE: ModelSim Altera – versão 10.1d

Figura 20 – Simulação do caso de teste 2 no circuito proprietário



FONTE: ModelSim Altera – versão 10.1d

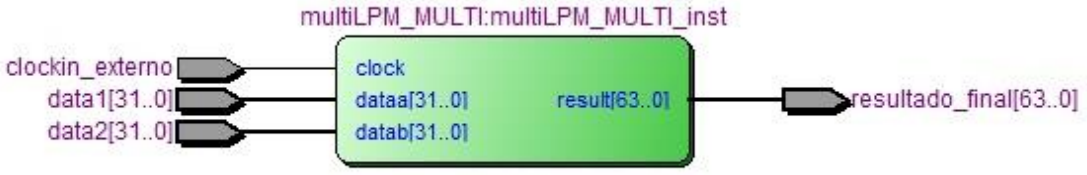
A velocidade de execução das multiplicações deve-se à utilização de multiplicadores embarcados, próximos à matriz de elementos lógicos e estruturada internamente para uma maior rapidez de execução. A implementação da metodologia proprietária – como mostra a Tabela 4 – utilizou 93 elementos lógicos e 192 pinos externos, o que deveu-se principalmente aos módulos dedicados empregados. O diagrama de blocos do circuito resultante apresenta-se na Figura 21.

Tabela 4 – Relatório de recursos utilizados na descrição dos circuitos proprietários

Total de elementos lógicos (33.216)	Total de registradores (0)	Total de pinos (475)	Total de pinos virtuais (0)	Total de memoria bits (483.840)	Elementos de multiplicadores de 9-bits embarcados (70)	Total de PLL's (4)
93	0	129	0	0	8	0

FONTE: Relatório de compilação – Quartus II – versão 13.0.1

Figura 21 – Diagrama de blocos do circuito proprietário



FONTE: Quartus II – versão 13.0.1

4.3 BOOTH

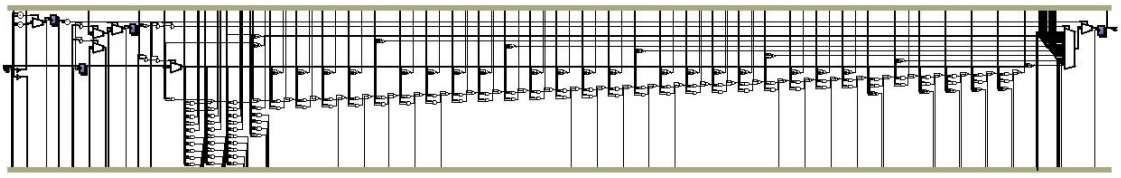
Tomando como base o fluxograma da multiplicação de Booth apresentado no capítulo anterior, a terceira metodologia foi implementada seguindo os conceitos do algoritmo. Foram utilizados apenas deslocadores e somadores para efetuar as operações.

Seguindo com a organização e estruturação do trabalho, os testes executados no circuito resultante dessa metodologia estão também localizados no Apêndice A. Além disso, há a impossibilidade de controlar perfeitamente um circuito que emprega essa abordagem sem que haja meios de sincronizar – um habilitador, por exemplo – a chegada de novas entradas somente depois que todos os estágios forem executados. Dito isto, o relógio presente no circuito tem o intuito apenas de controlar as iterações internas – os estágios – da execução. É possível também verificar a existência de um sinal interno chamado "estado booth" que auxilia na identificação das iterações do algoritmo quando da execução do mesmo. Após o 64º estágio o circuito oferece o resultado da multiplicação no registrador de produto. As

simulações dos testes do circuito apresentado nesta sessão estão contidos nos Apêndices C e D.

Apesar de gerar um circuito visualmente complexo – como mostra o digrama de blocos na Figura 22 – o mesmo apresentou a menor área ocupada em *chip* – como pode ser visto na Tabela 5 – dentre todos os outros, utilizando apenas 80 elementos lógicos e 33 pinos, além de 38 registradores.

Figura 22 – Diagrama de blocos do circuito que implementa a multiplicação do Booth



FONTE: Quartus II – versão 13.0.1

Tabela 5 – Relatório de recursos utilizados na descrição do circuito que implementa o algoritmo de Booth

Total de elementos lógicos (33.216)	Total de registradores	Total de pinos (475)	Total de bits de memória (483.840)	Multiplicadores embarcados (70)	Total de PLL's (4)
60	38	33	0	0	0

FONTE: Relatório de compilação – Quartus II – versão 13.0.1

Também é possível afirmar que o a abordagem de Booth levou a uma quantidade de tempo consideravelmente alta em comparação com os circuitos citados até então. O fato de o algoritmo valer-se apenas de deslocamentos e somas, ambos operados nos próprios registradores de resultado, fez o cálculo de uma multiplicação entre dois números levar cerca de 129ns na maioria dos casos – esse resultado pode ser pior caso aconteçam muitas operações de soma no decorrer da execução.

5 CONSIDERAÇÕES FINAIS

Confirmando as previsões iniciais, o circuito multiplicador apresentou uma área consideravelmente maior do que as demais metodologias. Por outro lado, a abordagem que mostrou-se mais lenta no cálculo das operações foi a que seguia um modelo algorítmico – o da multiplicação de Booth. Por fim, valendo-se de módulos específicos embarcados no FPGA, o circuito resultante da aplicação da função "LPM_MULT" da fabricante mostrou-se ser suficientemente balanceada, podendo ser considerada uma abordagem aplicável a qualquer projeto cujo fator área possa ser minimamente comprometido em troca de uma velocidade maior de execução.

Voltando o foco para as vantagens específicas de cada circuito descrito, o emprego de um modelo que vale-se essencialmente expressões lógicas e elementos básicos – as portas lógicas – não necessariamente teve o pior desempenho no cálculos de multiplicações. Contrariando as expectativas, o circuito apresentou um tempo baixo e constante durante a execução dos testes. Entretanto, além da necessidade de implementação um controle externo para sincronizar a chegada de novos dados às entradas e o desempenho mediano não justifica a utilização desse tipo de circuito em projetos reais.

Seguindo por outra via, a metodologia que utiliza o algoritmo de Booth para implementar a multiplicação claramente deve ser empregada em projetos nos quais a área em *chip* é um fator chave de restrição. No entanto é preciso avaliar se o mesmo projeto admite o fraco desempenho durante os cálculos além do fato de haver variação do tempo de cada operação dependendo do número de operações aritméticas efetuadas antes de cada deslocamento do registrador de produto. Outro ponto de impacto negativo é a necessidade de um controle externo ao circuito aritmético gerado para que se possa aplicá-lo a um ambiente de operações contínuas.

Por fim, a abordagem proprietária mostrou-se como a melhor opção para a maioria dos projetos pelo fato de ocupar apenas um pouco mais de espaço em *chip* – se comparada a melhor metodologia nesse aspecto – e ter o melhor desempenho mediante todas as outras estudadas. Ainda, por utilizar partes dedicadas para efetuar as multiplicações, o controle do circuito é regido pelos ciclos de relógio – diminuindo o tempo de desenvolvimento de projetos que utilizem essa abordagem.

Os resultados práticos obtidos somados a resolução do problema de configuração do relógio (*clock*) utilizado nas funções proprietárias terão impactos realmente positivos nos projetos executados posteriormente, uma vez que os mesmos poderão ser configurados como realmente seriam utilizados em campo.

REFERÊNCIAS

CARDOSO, Fabbryccio A. C. M.; FERNANDES, Marcelo Augusto Costa. **Fluxo de Projetos.** *In:* ARANTES, Dalton Soares. Rapid Prototyping for DSP and Digital Communication. 2010. Disponível em: <http://www.decom.fee.unicamp.br/~cardoso/ie344b.html>. Acesso em: 14, Jan 2014.

CASILLO, Leonardo. **Metodologia para adaptação de microarquiteturas microprogramadas soft-core à uma ISA padrão: e**studo do Impacto sobre a complexidade de hardware para o padrão MIPS. Tese de Doutorado. Universidade Federal do Rio Grande do Norte – UFRN, Natal, 2013.

CYCLONE II Device Handbook. **Altera®,** San Jose – California: Altera Corporation, 2008.

HAUCK, Scott; DEHON, André. **Reconfigurable Computing:** the theory and practice of FPGA-based computation. Burlington – Massachusetts: Elsevier, 2008.

LANDIS, Dave. **Programmable Logic and Application Specific Integrated Circuits.** p. 01 – p. 54, Cap. 02. *In:* HARPER, Charles A; JONES, H. Handbook of Components for Electronics. 2. ed. [S.l.]: McGraw-Hill, Inc., 1996.

LPM Quick Reference Guide. **Altera®,** San Jose – California: Altera Corporation, 1996.

PATTERSON, David A.; HENNESSY, John L. **Computer Organization and Design:** the hardware/software interface. San Francisco – California: Elsevier, 2005.

_____. **Computer Organization and Design:** the hardware/software interface. 4. ed. San Francisco – California: Elsevier, 2012.

PLLs in Cyclone II Devices. p. 01 – p.34, Cap. 07. 2007. *In:* CYCLONE II Device Handbook. **Altera®,** San Jose – California: Altera Corporation, 2008.

STALLINGS, William. **Arquitetura e Organização de Computadores.** 8. ed. São Paulo: Pearson Education do Brasil, 2010.

TOCCI, Ronald. J; WIDMER, Neal S.; MOSS, Gregory L. **Digital Systems:** principles and applications. 10. ed. Upper Saddle River – New Jersey: Pearson, 2007.

APÊNDICE A – **Registro de casos de testes**

- **Tabela 1** – Caso de teste 1

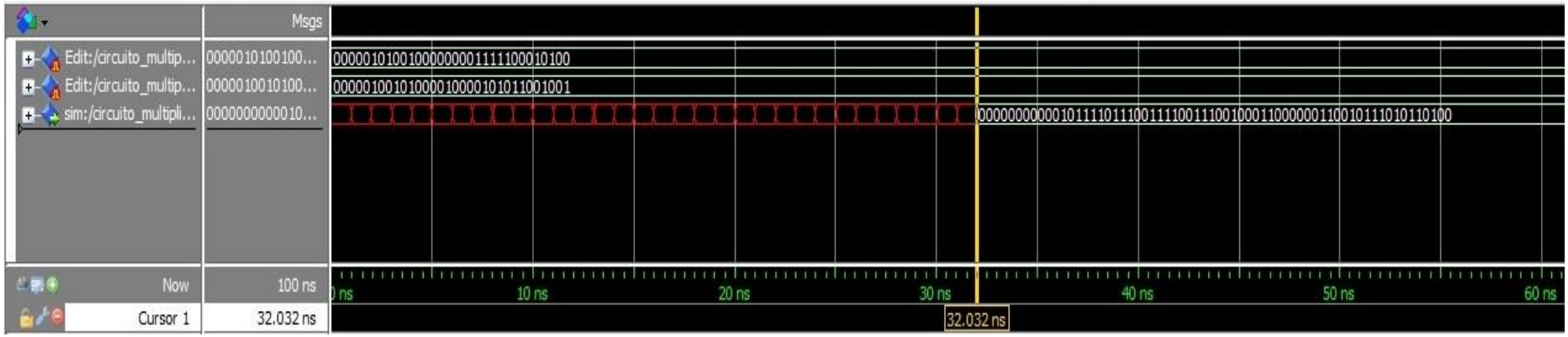
1	85991188	00000101001000000001111100010100	X	77662921	00000100101000010000101011001001
2	86630911	00000101001010011110000111111111	X	42893239	00000010100011100111111110110111
3	79549475	00000100101111011101010000100011	X	172651078	00001010010010100111001001000110
4	101518729	00000110000011010000110110001001	X	1739125	00000000000110101000100101110101
5	192183191	00001011011101000111101110010111	X	209164751	00001100011101111001100111001111
6	95688496	00000101101101000001011100110000	X	163716600	00001001110000100001110111111000
7	99500233	00000101111011100100000011001001	X	48827274	00000010111010010000101110001010
8	74238596	00000100011011001100101010000100	X	104667443	00000110001111010001100100110011
9	164245984	00001001110010100011000111100000	X	152287132	00001001000100111011011110011100
10	66257986	00000000011100110000010001000010	x	154158361	00001001001100000100010100011001

- **Tabela 2** – Caso de teste 2

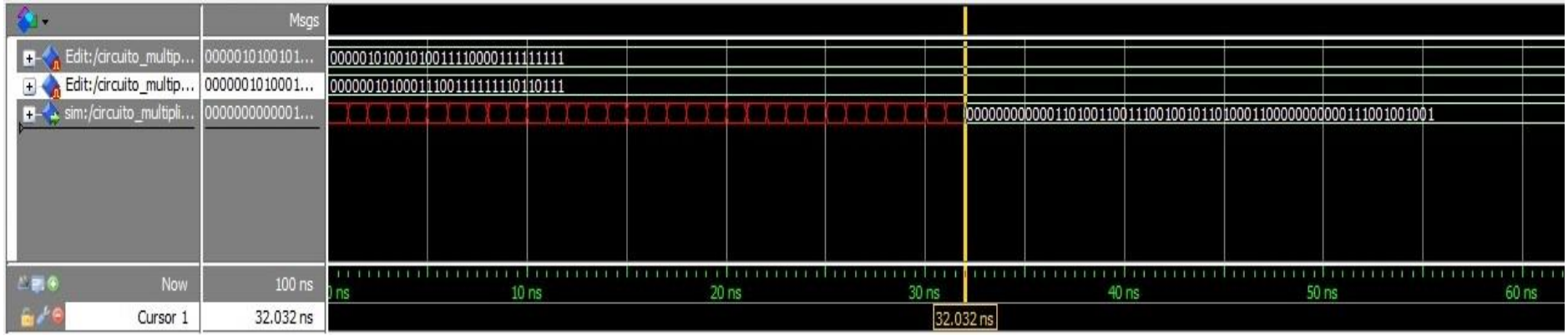
1	85991188	00000101001000000001111100010100	X	-85221459	11111010111010111001111110101101
2	86630911	00000101001010011110000111111111	X	-21910554	11111110101100011010101111100110
3	79549475	00000100101111011101010000100011	X	-180846518	11110101001110001000000001001010
4	101518729	00000110000011010000110110001001	X	-171746443	11110101110000110101101101110101
5	192183191	00001011011101000111101110010111	X	-147759500	11110111001100010101111001110100
6	95688496	00000101101101000001011100110000	X	-194941479	11110100011000010110110111011001
7	99500233	00000101111011100100000011001001	X	-192943712	11110100011111111110100110100000
8	74238596	00000100011011001100101010000100	X	-141547747	11110111100100000010011100011101
9	164245984	00001001110010100011000111100000	X	-131437759	11111000001010100110101101000001
10	66257986	00000000011100110000010001000010	x	-162905161	11110110010010100100001110110111

APÊNDICE B – Simulações do caso de teste 1 no circuito multiplicador

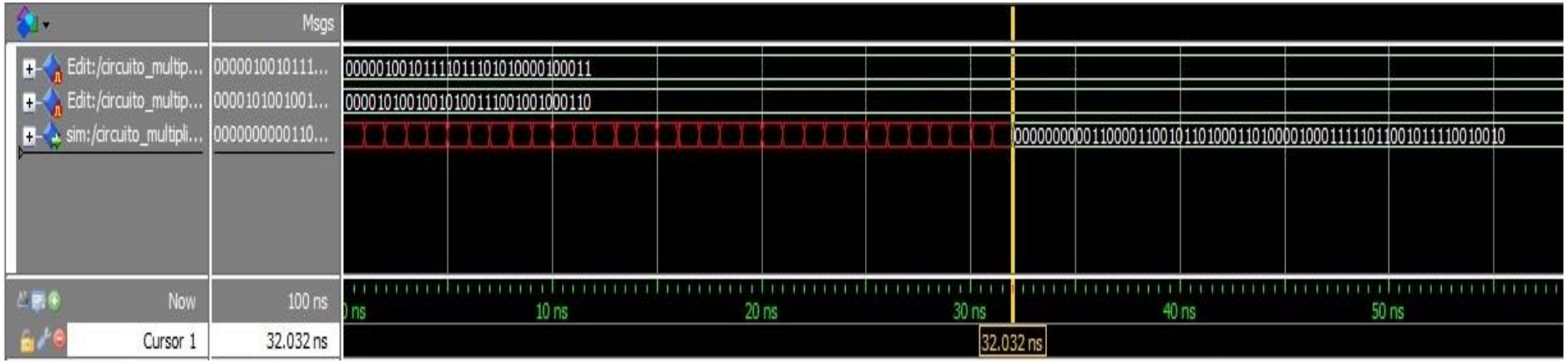
- **Imagem 1** – Simulação dos números da linha 1 da Tabela 1 do Apêndice A



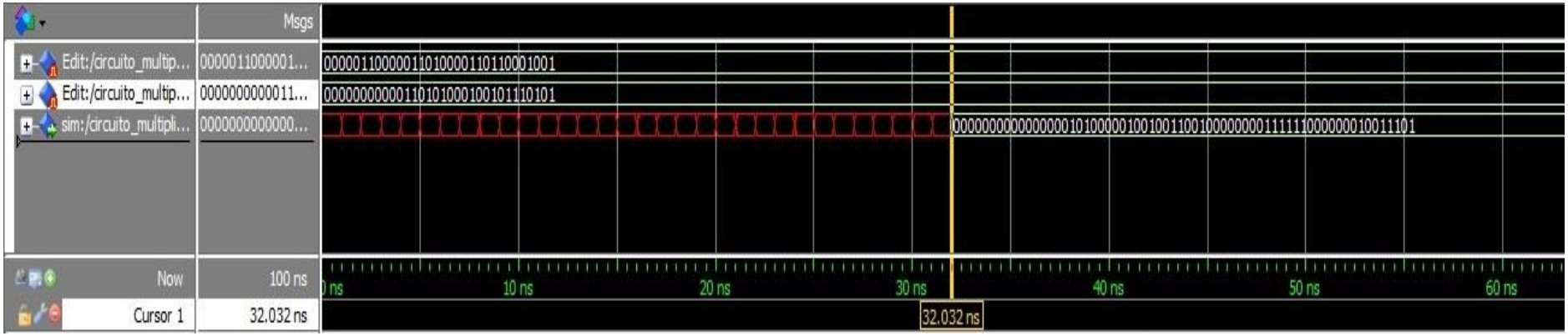
- **Imagem 2** – Simulação dos números da linha 2 da Tabela 1 do Apêndice A



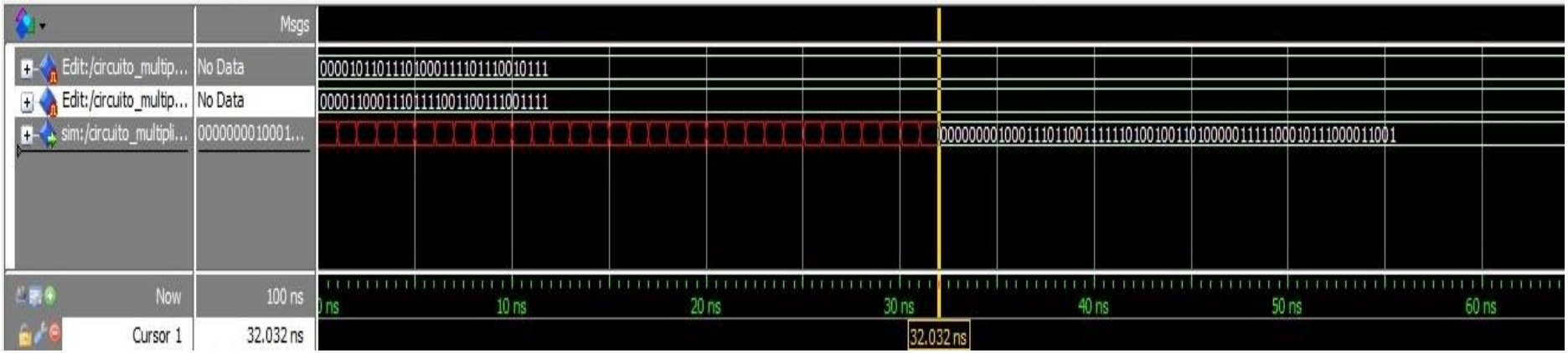
- **Imagem 3** – Simulação dos números da linha 3 da Tabela 1 do Apêndice A



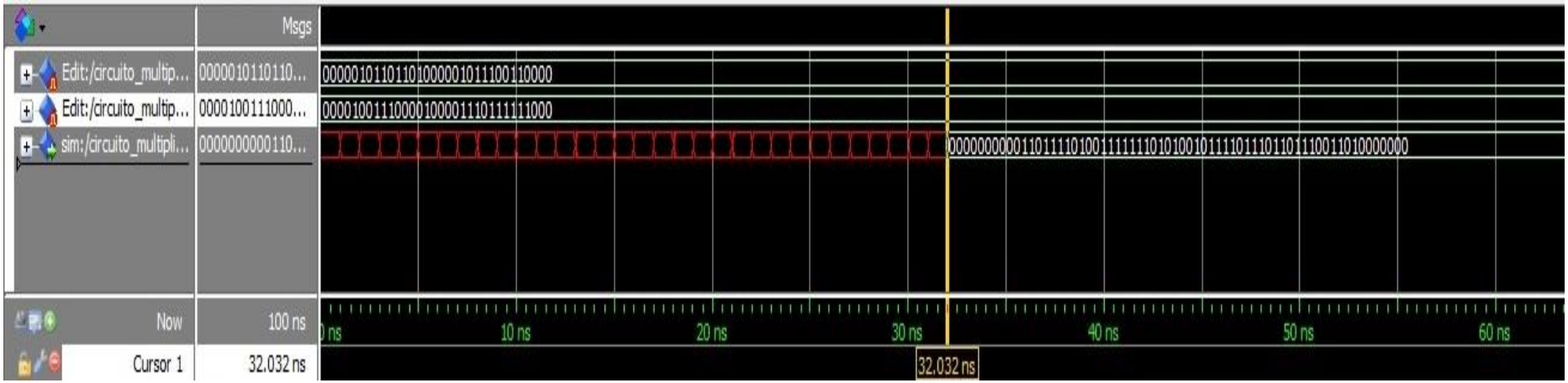
- **Imagem 4** – Simulação dos números da linha 4 da Tabela 1 do Apêndice A



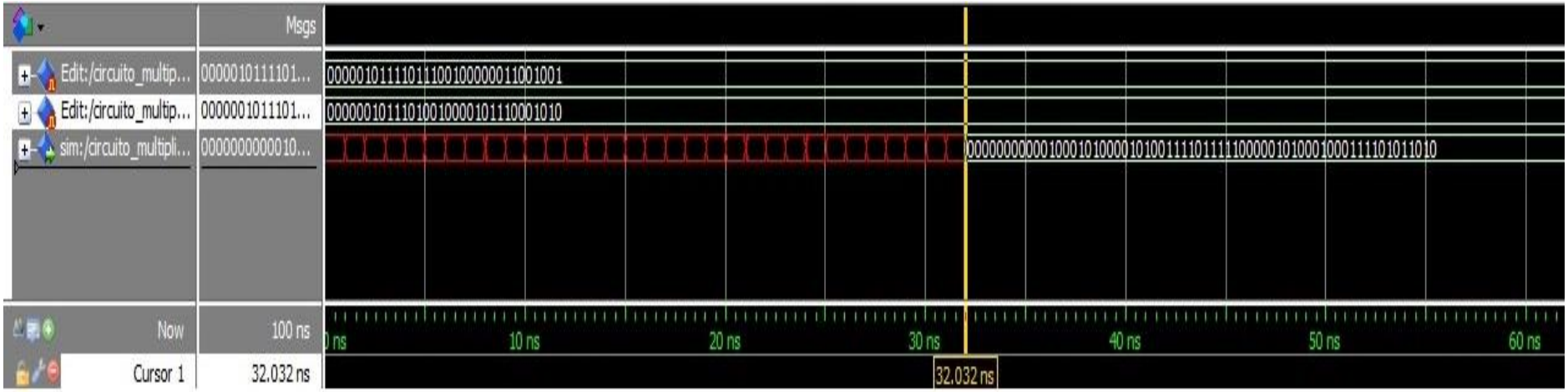
- **Imagem 5** – Simulação dos números da linha 5 da Tabela 1 do Apêndice A



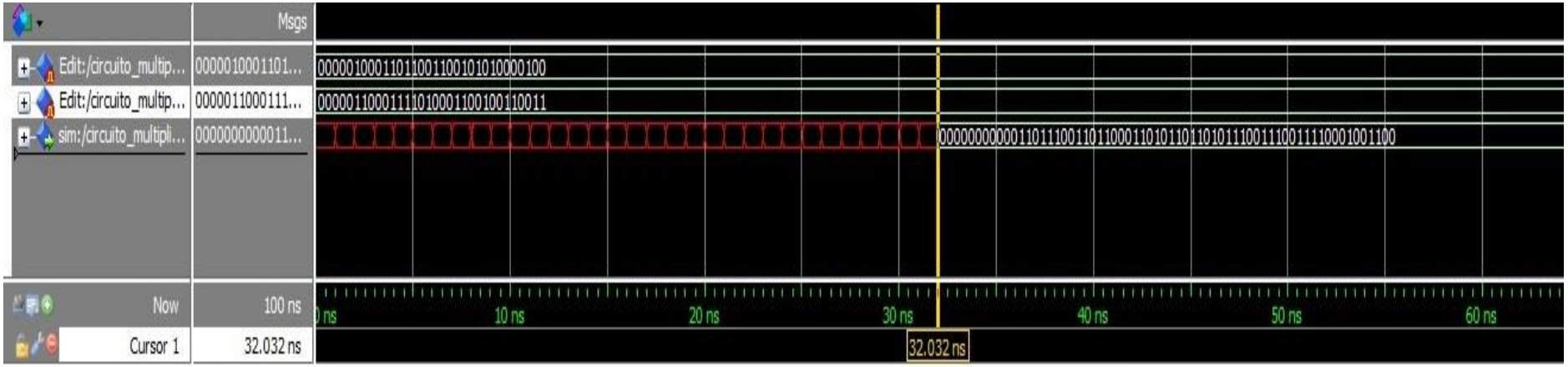
- **Imagem 6** – Simulação dos números da linha 6 da Tabela 1 do Apêndice A



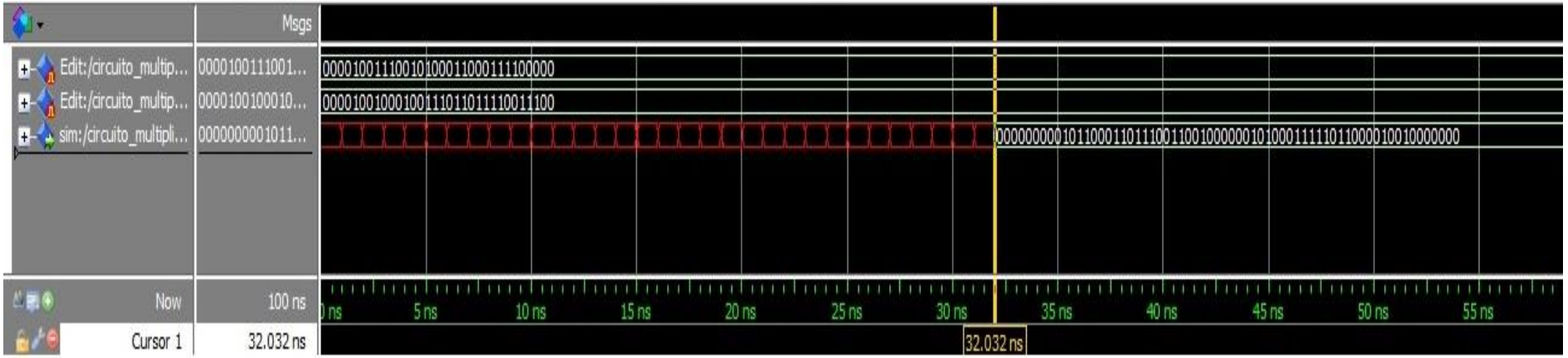
- **Imagem 7** – Simulação dos números da linha 7 da Tabela 1 do Apêndice A



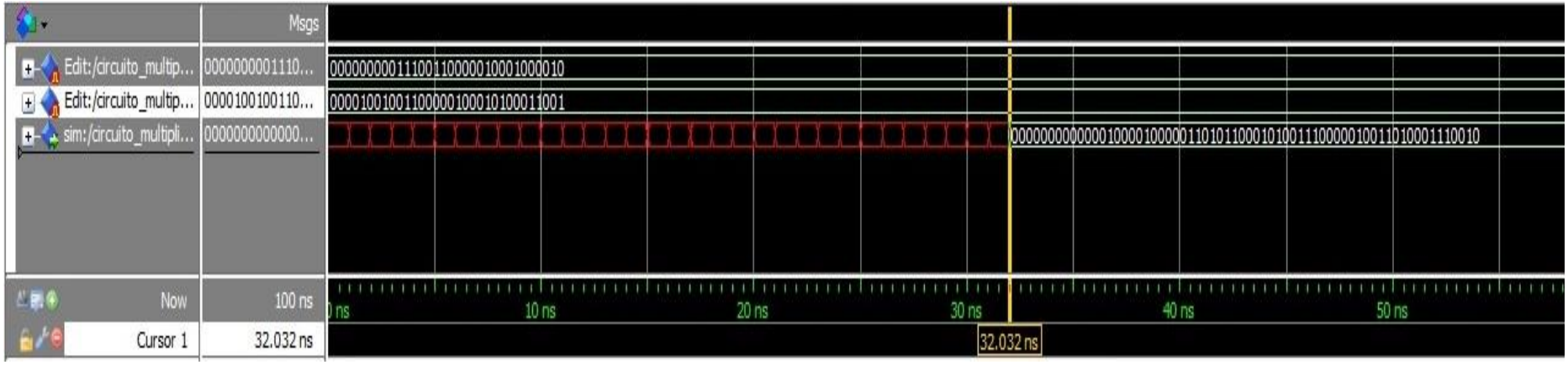
- **Imagem 8** – Simulação dos números da linha 8 da Tabela 1 do Apêndice A



- **Imagem 9** – Simulação dos números da linha 9 da Tabela 1 do Apêndice A

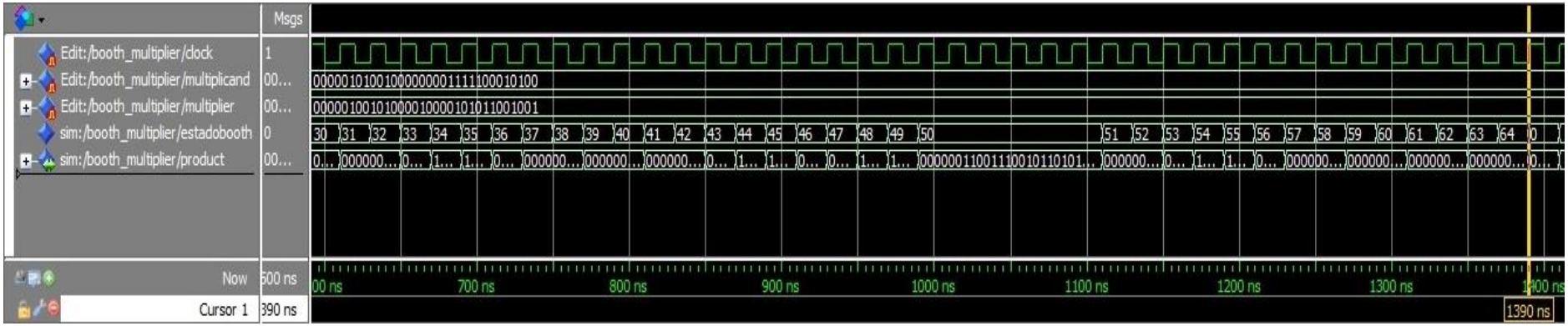


- **Imagem 10** – Simulação dos números da linha 10 da Tabela 1 do Apêndice A

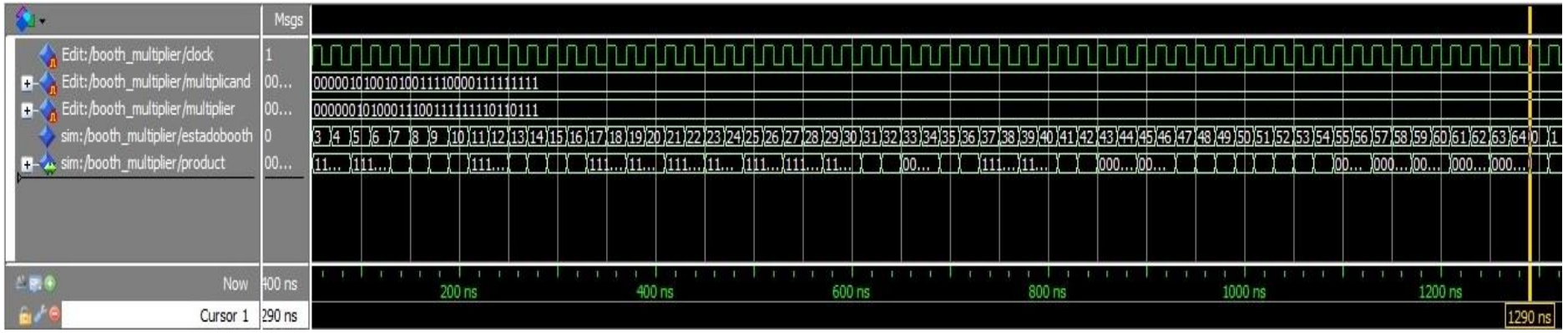


APÊNDICE C - Simulações do caso de teste 1 no circuito que implementa o algoritmo de Booth

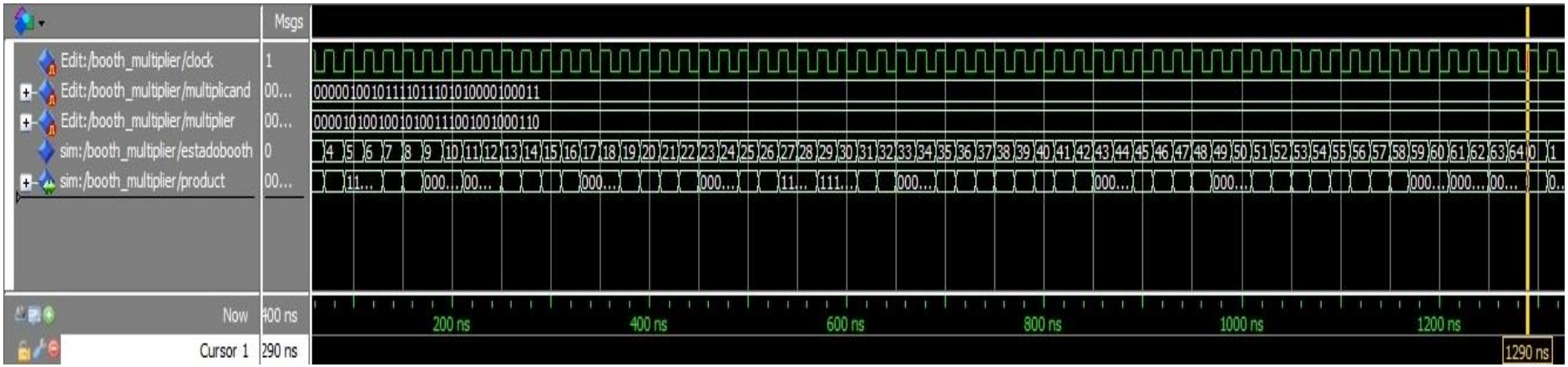
- **Imagem 1** – Simulação dos números da linha 1 da Tabela 1 do Apêndice A



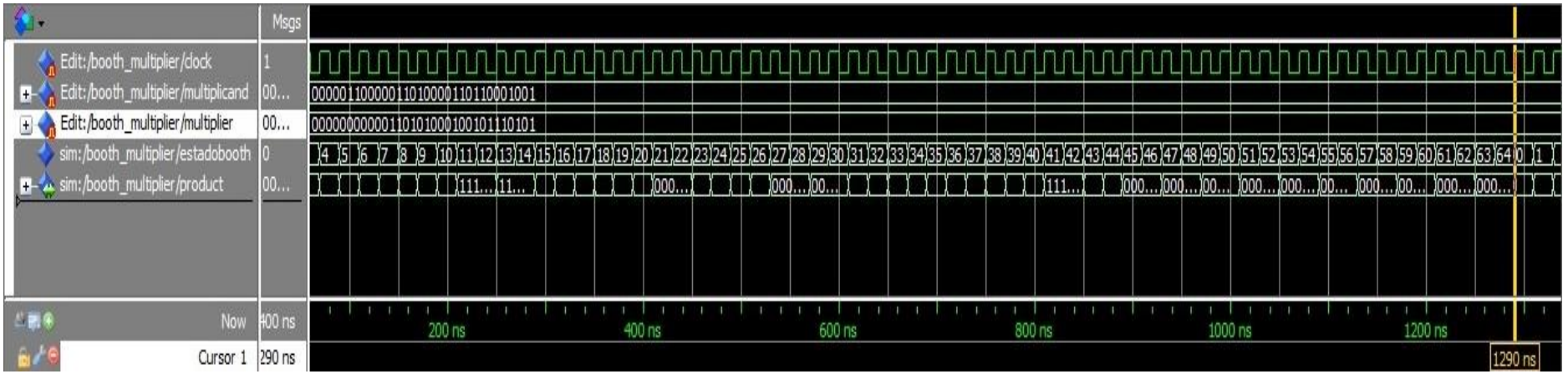
- **Imagem 2** – Simulação dos números da linha 2 da Tabela 1 do Apêndice A



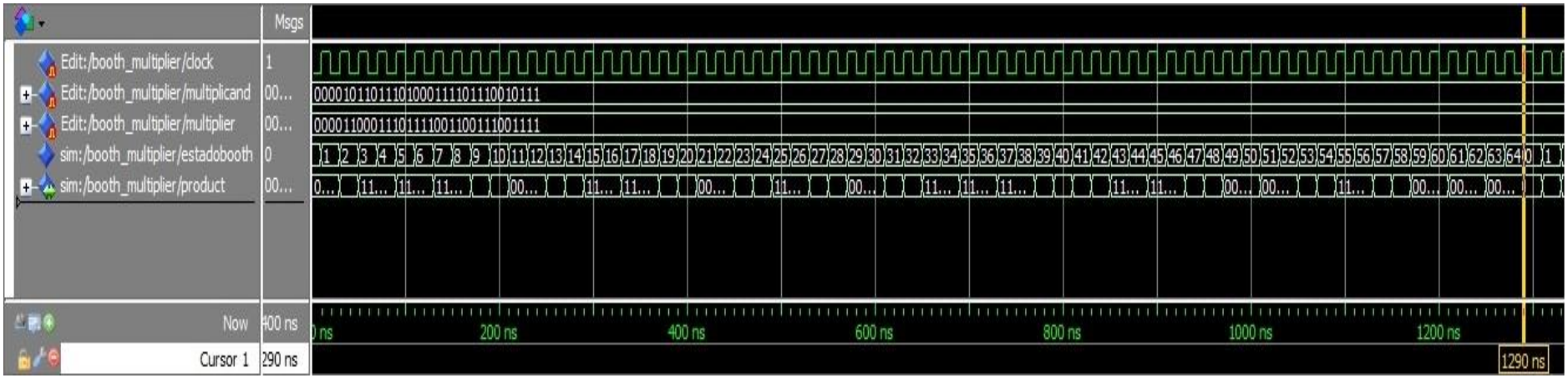
- **Imagem 3** – Simulação dos números da linha 3 da Tabela 1 do Apêndice A



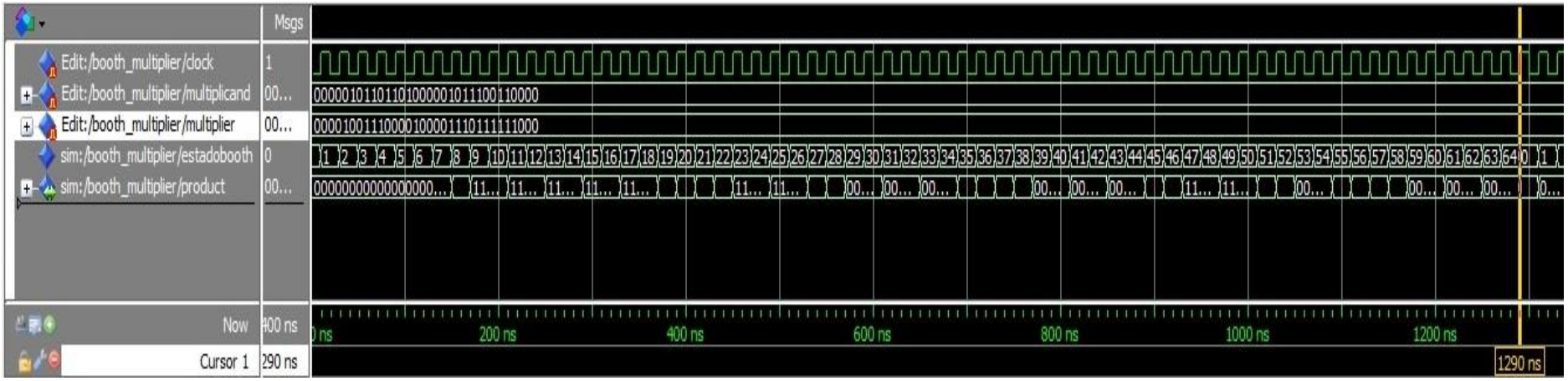
- **Imagem 4** – Simulação dos números da linha 4 da Tabela 1 do Apêndice A



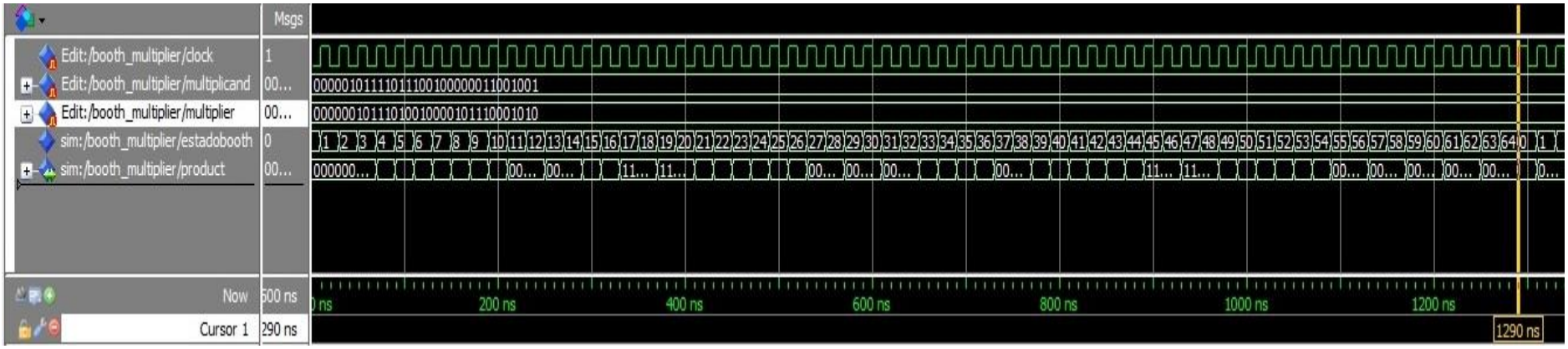
- **Imagem 5** – Simulação dos números da linha 5 da Tabela 1 do Apêndice A



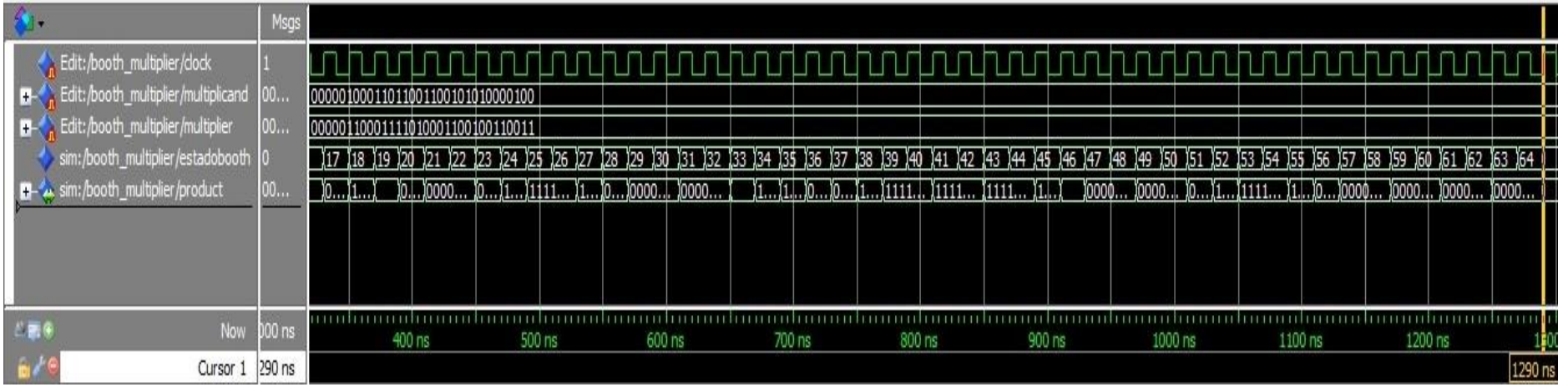
- **Imagem 6** – Simulação dos números da linha 6 da Tabela 1 do Apêndice A



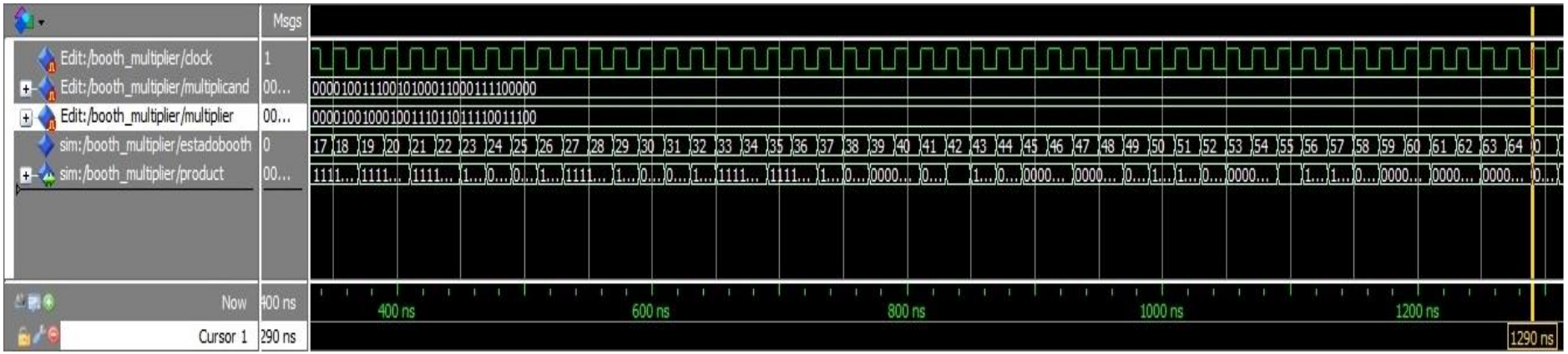
- **Imagem 7** – Simulação dos números da linha 7 da Tabela 1 do Apêndice A



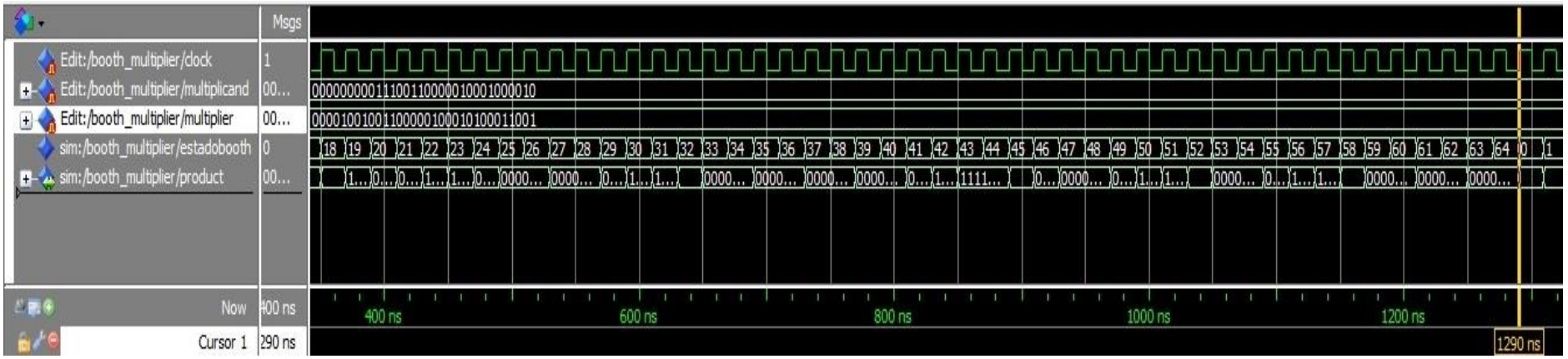
- **Imagem 8** – Simulação dos números da linha 8 da Tabela 1 do Apêndice A



- **Imagem 9** – Simulação dos números da linha 9 da Tabela 1 do Apêndice A

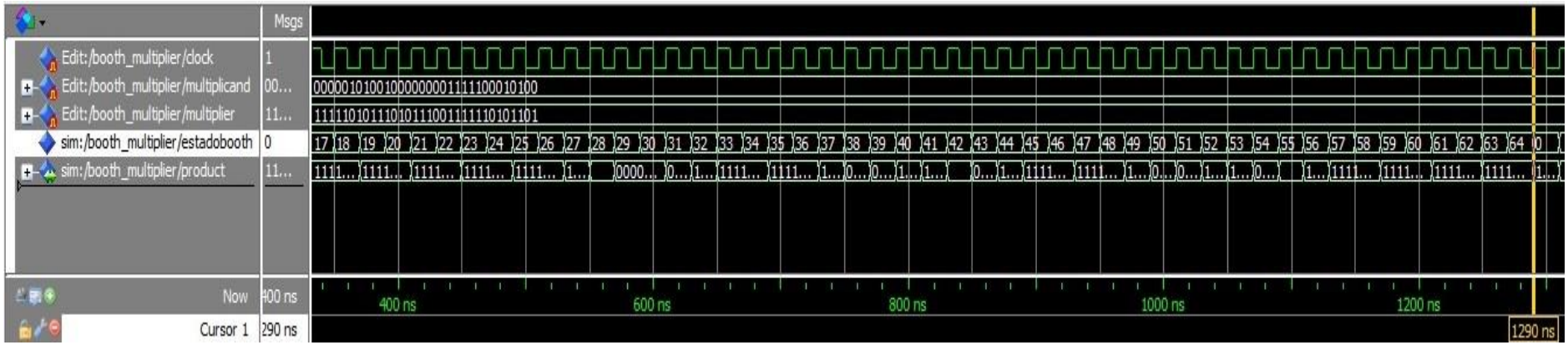


- **Imagem 10** – Simulação dos números da linha 10 da Tabela 1 do Apêndice A

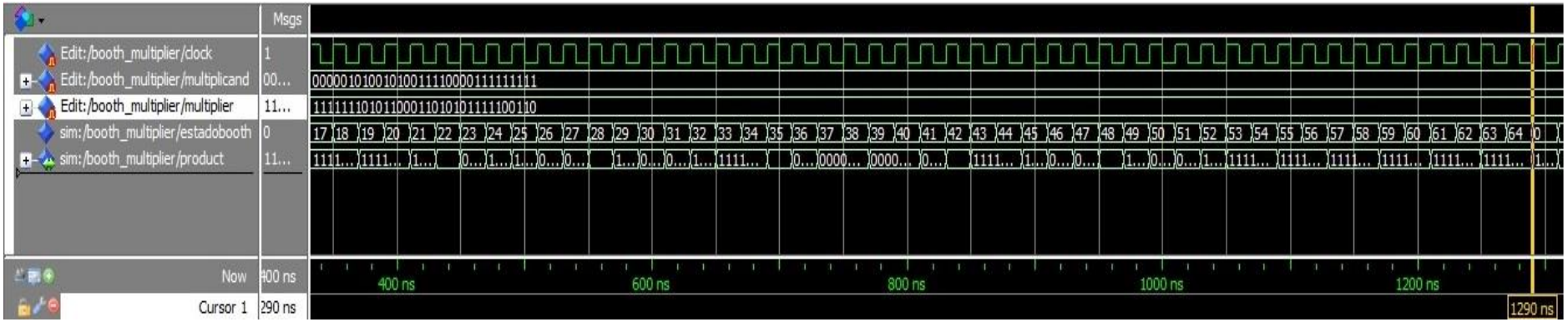


APÊNDICE D - Simulações do caso de teste 2 no circuito que implementa o algoritmo de Booth

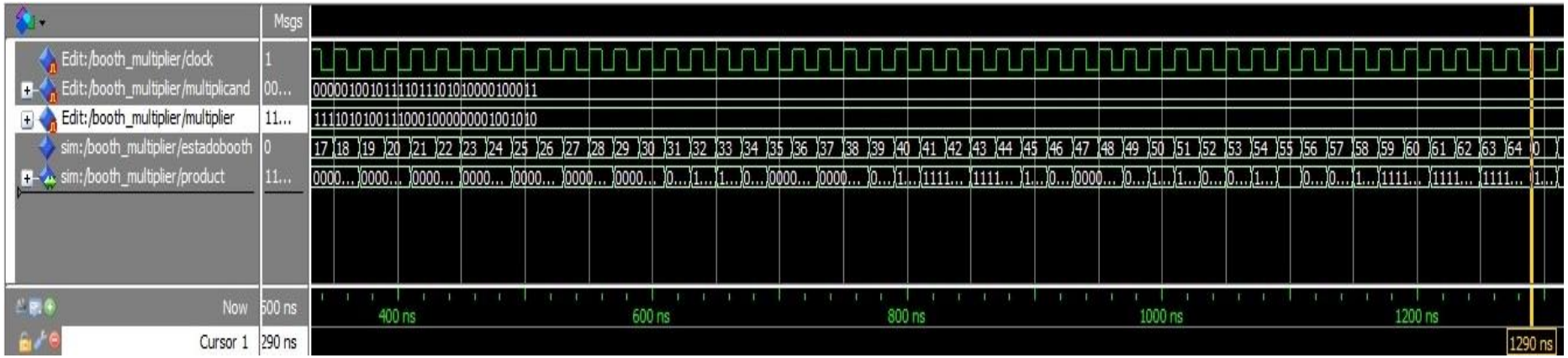
- **Imagem 1** – Simulação dos números da linha 1 da Tabela 2 do Apêndice A



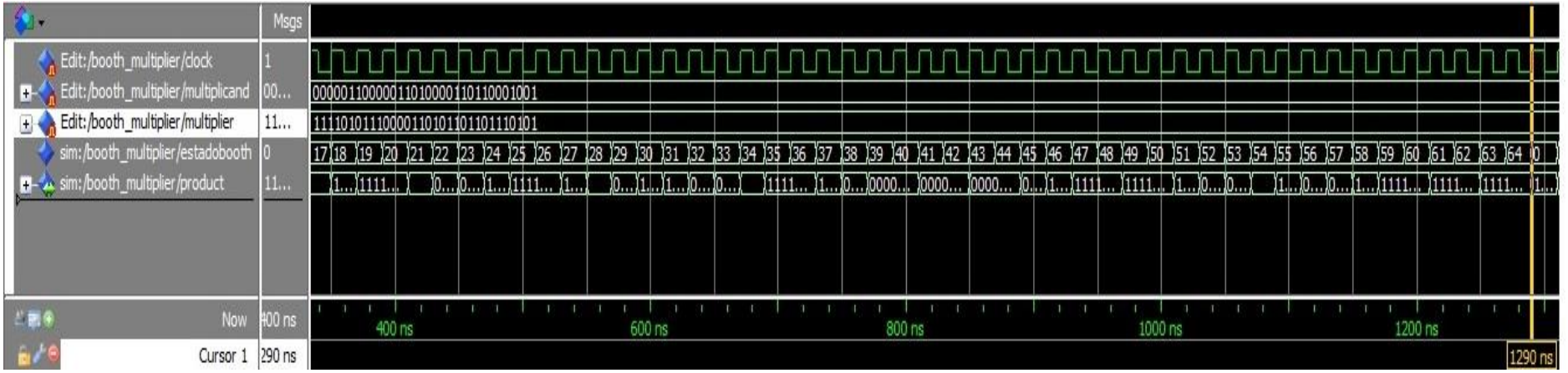
- **Imagem 2** – Simulação dos números da linha 2 da Tabela 2 do Apêndice A



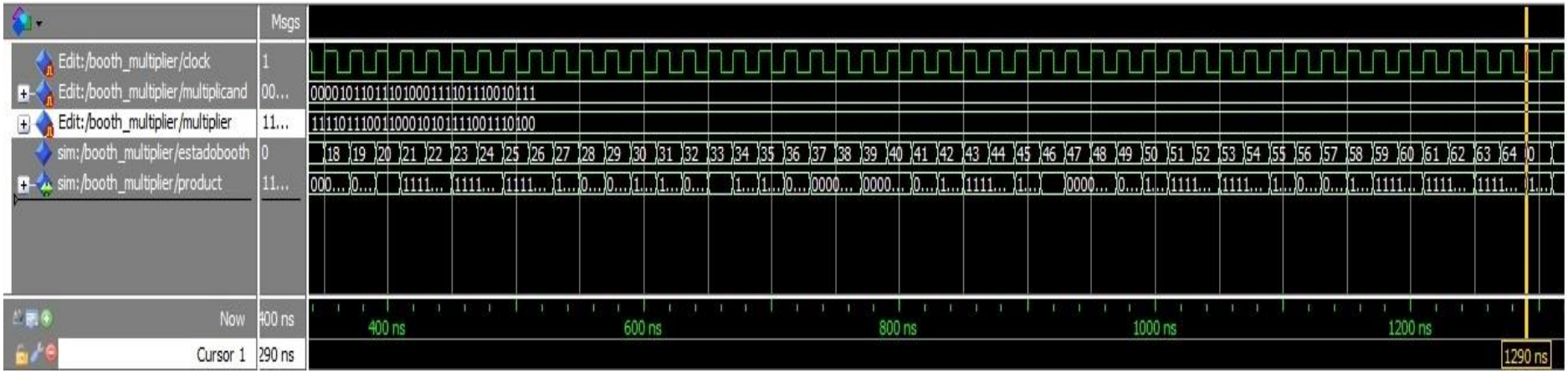
- **Imagem 3** – Simulação dos números da linha 3 da Tabela 2 do Apêndice A



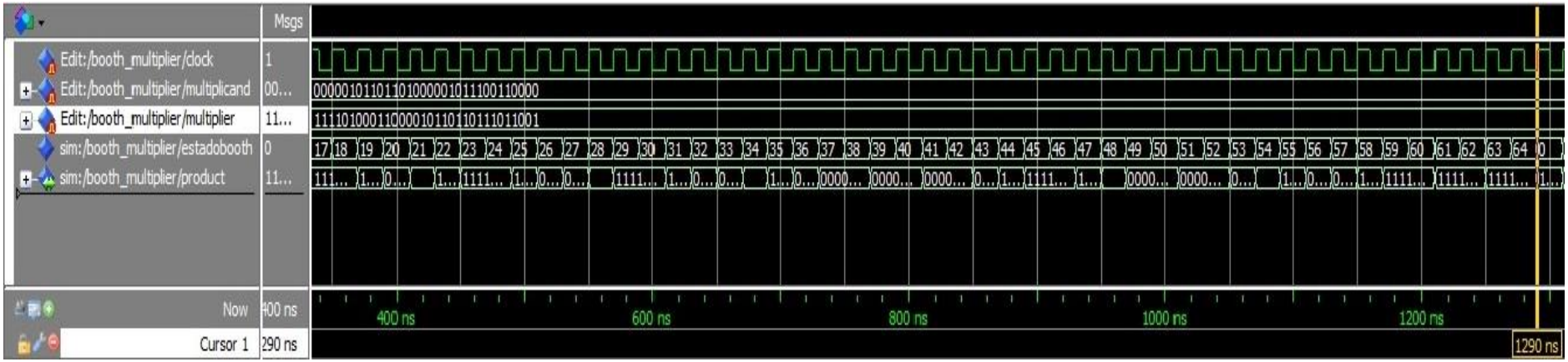
- **Imagem 4** – Simulação dos números da linha 4 da Tabela 2 do Apêndice A



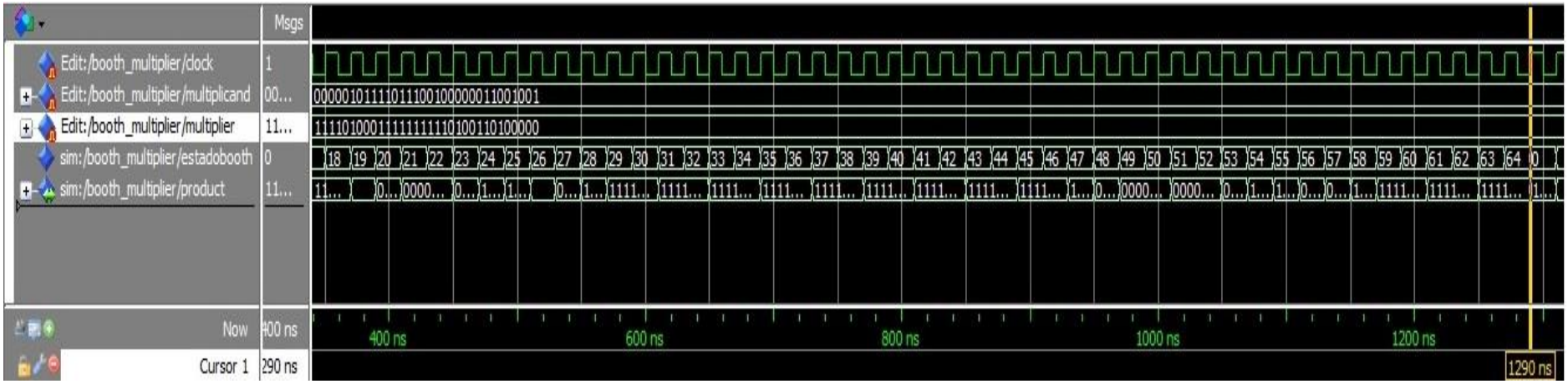
- **Imagem 5** – Simulação dos números da linha 5 da Tabela 2 do Apêndice A



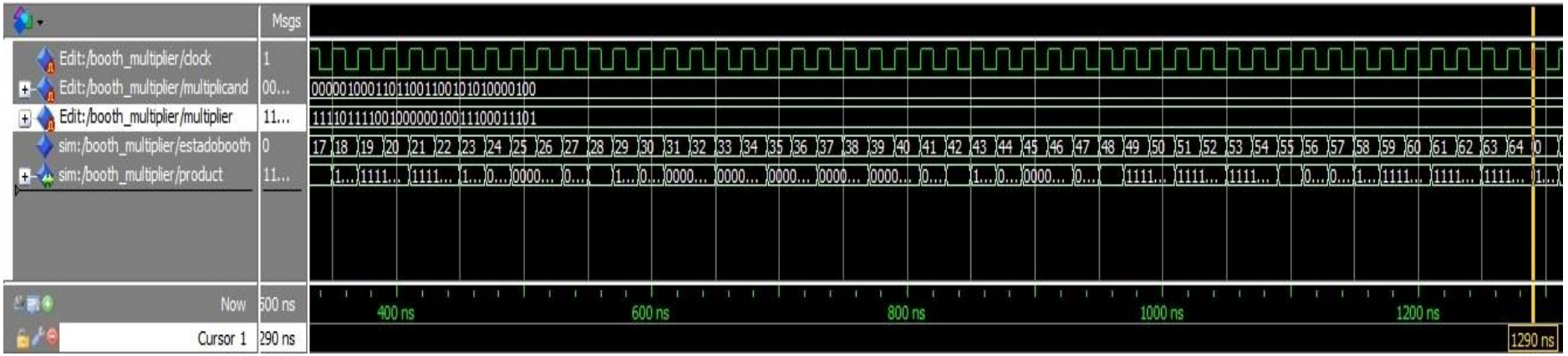
- **Imagem 6** – Simulação dos números da linha 6 da Tabela 2 do Apêndice A



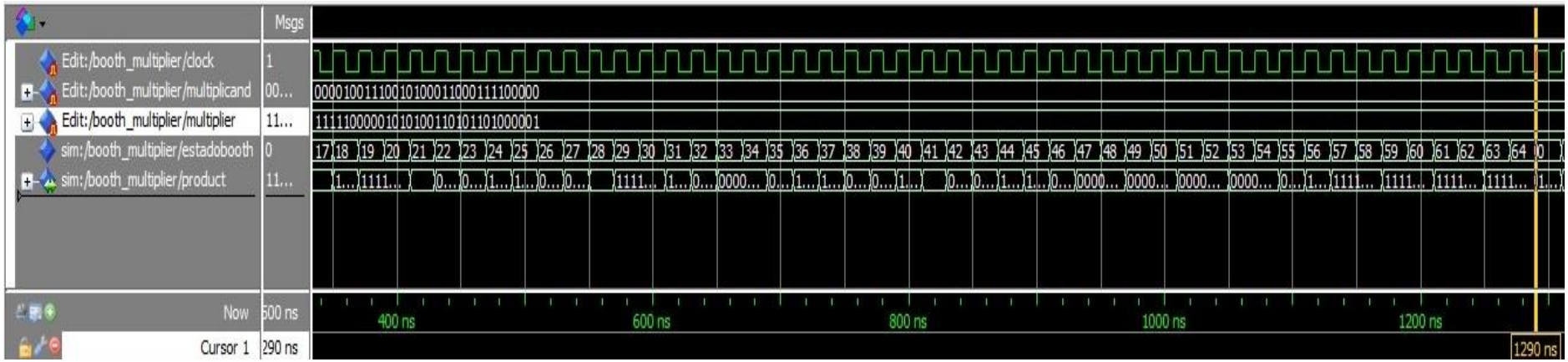
- **Imagem 7** – Simulação dos números da linha 7 da Tabela 2 do Apêndice A



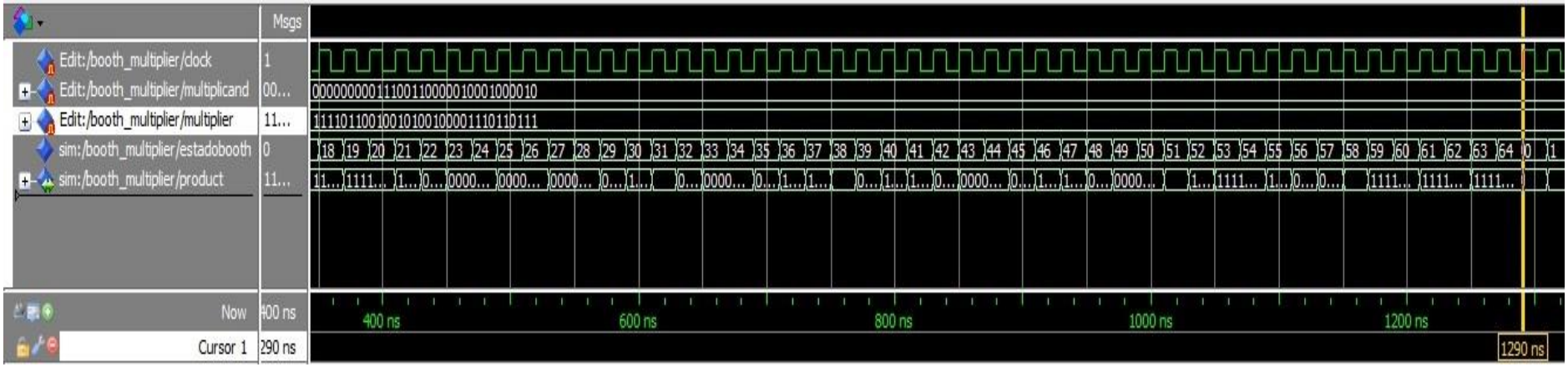
- **Imagem 8** – Simulação dos números da linha 8 da Tabela 2 do Apêndice A



- **Imagem 9** – Simulação dos números da linha 9 da Tabela 2 do Apêndice A

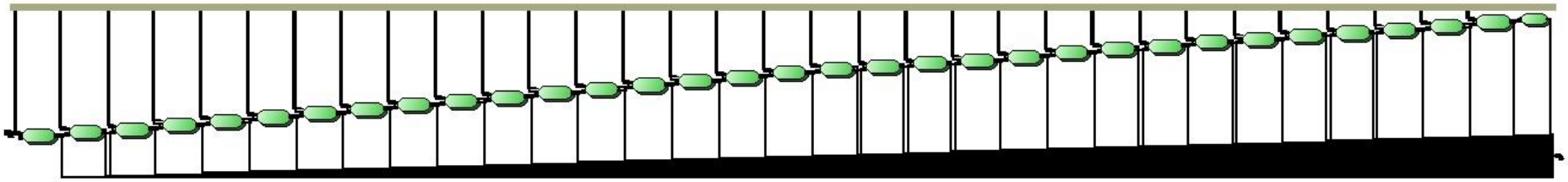


- **Imagem 10** – Simulação dos números da linha 10 da Tabela 2 do Apêndice A

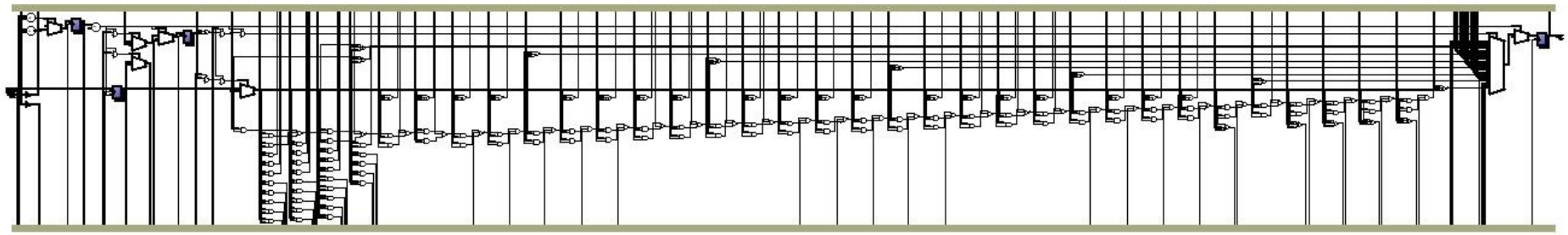


APÊNDICE E – Diagramas a nível de registradores – RTL (*Register Transfer Level*) – gerados pelo Quartus II versão 13.0.1

- **Imagem 1** – RTL do circuito multiplicador



- **Imagem 2** – RTL do circuito que implementa o algoritmo de Booth



- **Imagem 3** – RTL do circuito que implementa a função de multiplicação proprietária fornecida pela Altera®