

# Conteúdo

<b>1</b>	<b>Experimento 1 - Reconfiguração Dinâmica.....</b>	<b>1</b>
1.1	Definição do Teste . . . . .	1
1.1.1	Comportamento . . . . .	2
1.2	Estudo dos Requisitos . . . . .	2
1.2.1	Peculiaridades . . . . .	2
1.2.2	Fluxo de Ferramentas . . . . .	3
1.3	Implementação . . . . .	4
1.3.1	Síntese . . . . .	4
1.3.2	PlanAhead . . . . .	6
1.3.3	iMPACT . . . . .	8
1.4	Resultados . . . . .	9
1.4.1	Síntese . . . . .	9
1.4.2	PlanAhead . . . . .	9
1.4.3	iMPACT . . . . .	10
1.4.4	Possíveis Erros . . . . .	10
1.5	Conclusão . . . . .	12
<b>2</b>	<b>Experimento 2 - Memórias.....</b>	<b>13</b>
2.1	Definição do Teste . . . . .	13
2.2	Estudo dos Requisitos . . . . .	13
2.2.1	Tipos de Memória . . . . .	13
2.2.2	Escolha da Memória . . . . .	15
2.2.3	MicroBlaze . . . . .	15
2.2.4	Fluxo de Projeto . . . . .	16

2.3	Implementação . . . . .	17
2.3.1	XPS . . . . .	17
2.3.2	SDK . . . . .	18
2.4	Resultados . . . . .	19
2.4.1	XPS . . . . .	19
2.4.2	SDK . . . . .	20
2.5	Conclusão . . . . .	20
<b>3</b>	<b>Experimento 3 - <i>Bootloader</i> . . . . .</b>	<b>21</b>
3.1	Definição do Teste . . . . .	21
3.2	Estudo dos Requisitos . . . . .	21
3.2.1	Arquivo Binário . . . . .	21
3.2.2	Inicialização da Memória SPI Flash . . . . .	22
3.3	Implementação . . . . .	23
3.3.1	XPS . . . . .	23
3.3.2	SDK . . . . .	24
3.3.3	data2mem . . . . .	24
3.3.4	Programação Indireta da Memória Flash . . . . .	25
3.4	Resultados . . . . .	26
3.5	Conclusão . . . . .	26
<b>4</b>	<b>Experimento 4 - Autoreconfiguração com <i>MicroBlaze</i> e DDR3 . . . . .</b>	<b>28</b>
4.1	Definição do Teste . . . . .	28
4.2	Estudo dos Requisitos . . . . .	28
4.2.1	ICAP e ICAPE2 . . . . .	28
4.2.2	Fluxo de Projeto . . . . .	29
4.3	Resultados . . . . .	29
4.4	Conclusão . . . . .	30
<b>5</b>	<b>Experimento 5 - Autoreconfiguração com <i>MicroBlaze</i> e sem DDR3 . . . . .</b>	<b>31</b>
5.1	Introdução Teórica . . . . .	31
5.1.1	Periférico Reconfigurável . . . . .	31

5.1.2	Fluxo do Projeto . . . . .	32
5.2	Experimento . . . . .	32
5.2.1	XPS . . . . .	32
5.2.2	PlanAhead . . . . .	34
5.2.3	SDK . . . . .	36
5.2.4	data2mem . . . . .	37
5.2.5	iMPACT . . . . .	37
5.3	Resultados . . . . .	38
5.3.1	XPS . . . . .	38
5.3.2	PlanAhead . . . . .	38
5.3.3	SDK . . . . .	40
5.3.4	data2mem . . . . .	41
5.3.5	iMPACT . . . . .	41
5.4	Conclusão . . . . .	41

# Capítulo 1

## Experimento 1 - Reconfiguração Dinâmica

### 1.1 Definição do Teste

De forma a dar validade a todo o projeto, desenvolveu-se um experimento para entender o processo de desenvolvimento de sistemas reconfiguráveis dinamicamente e algumas peculiaridades do kit de desenvolvimento. Este experimento tem como objetivo a construção de um projeto que se utilize de reconfiguração dinâmica para reprogramar um FPGA.

Para se entender mais a fundo o fluxo de projeto, nada melhor que construir um projeto. Implementou-se para isto o sistema esquematizado na figura 1.1. Este sistema contém o “*Top*” para interfaceamento com a FPGA, o “*Clock\_Station*” e o “*Clocks*” para tratamento do sinal de relógio, a “*Static*”, que possui uma lógica estática para demonstrar que a reconfiguração de uma partição não interfere com outra, e a “*Dynamic*”, que possui a lógica a ser alterada dinamicamente. A interface “*Top*” possui conexões com os LEDs e o SYSCLK do dispositivo FPGA, bem como conexões de entrada e saída com os componentes instanciados nela. O componente “*Clock\_Station*” recebe a entrada de relógio diferencial e retorna 3 sinais de relógios simples, um com 1 Hz, outro com 2 Hz e o último com 5 Hz.

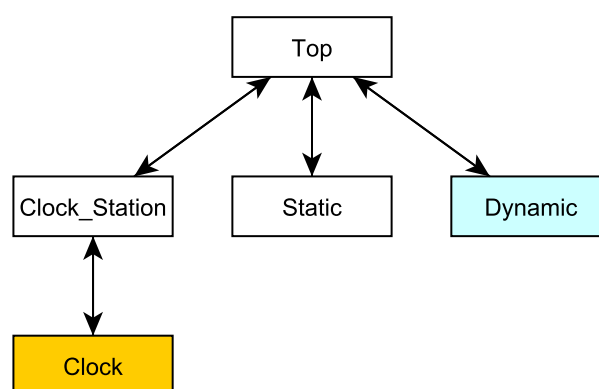


Figura 1.1: Foto ilustrativa do sistema desenvolvido para o teste de validação do experimento 1. Ele é composto por uma parte estática e uma parte dinâmica. Os elementos em branco são estáticos, os em azul são dinâmicos e o em amarelo corresponde a um componente gerado automaticamente com o através das ferramentas da Xilinx.

### 1.1.1 Comportamento

O projeto de um sistema parcialmente reconfigurável pode ser visto como vários projetos completos com partes em comum. Seguindo essa lógica, dois projetos com comportamentos diferentes foram construídos usando como base a figura 1.1. O comportamento individual de cada módulo ou componente será descrito a seguir. Este passo está ilustrado no fluxo de ferramentas da figura 1.2b como ISE, visto que esta ferramenta da Xilinx é a mais utilizada para projetos comuns.

O componente “*Static*” possui 3 entradas para relógios, um pulsando a 1 Hz, outro a 2 Hz e o terceiro a 5 Hz, e 3 saídas para LEDs. Ele simplesmente liga as entradas (relógios) às saídas (LEDs), criando assim uma forma visual de comprovar que a reconfiguração do componente “*Dynamic*” acontecerá dinamicamente.

O componente “*Dynamic*” possui dois comportamentos distintos. O primeiro deles é o de um simples contador crescente de 4 bits. O segundo é uma máquina de estados que alterna os 4 bits de saída entre "1100" e "0011" a cada pulso de relógio. Este componente possui uma entrada para um relógio pulsando a 1 Hz, que acionará as transições, e uma palavra de 4 bits de saída. A frequência de operação deste componente foi escolhida para, junto com as frequências do “*Static*”, permitir a visualização de que “*Static*” não para de funcionar quando “*Dynamic*” está sendo reconfigurado.

O componente “*Clocks*” recebe os sinais diferenciais de relógio e o transformam em um sinal comum. O bloco lógico utilizado para isso foi construído usando ferramentas presentes no ISE. Uma vez que a ferramenta permitia a construção de um relógio com divisor de frequência, a frequência do relógio da placa, que nesse caso é de 200 MHz, foi reduzida para 20 MHz.

O módulo “*Top*” instancia os componentes descritos acima e faz a interface dos mesmos com a FPGA. O componente dinâmico precisa de uma declaração de protótipo para ser instanciado corretamente. Utilizou-se o código abaixo para esta finalidade.

```
component dynamic
    port ( clk : in std_logic;
          leds : out std_logic_vector (3 downto 0));
end component;
```

O módulo “*Top*” possui também um divisor de frequência para reduzir a frequência devolvida por “*Clocks*” para 1 e 2 Hz.

## 1.2 Estudo dos Requisitos

Antes de se começar a construir o experimento, é necessário estudar as peculiaridades da placa e deste tipo de projeto, bem como seu fluxo de desenvolvimento. Este estudo é apresetado a seguir.

### 1.2.1 Peculiaridades

O kit de desenvolvimento utilizada apresenta várias peculiaridades com relação aos kits comuns. A única que afeta diretamente o experimento em questão é o uso de um relógio com sinal diferencial ao

invés do sinal comum, explicado a seguir. Outra peculiaridade é uma diferente estrutura de organização de arquivos, que permite um desenvolvimento mais limpo e de fácil entendimento.

#### 1.2.1.1 Relógio Diferencial

Diferentemente das FPGAs comuns, a que está presente neste kit contém um relógio diferencial, ou seja, ele é composto por dois sinais ao invés de um. A razão para tal é a presença de circuitos sensíveis a interferências, tais como *transceivers*, que são muito menores em sinais diferenciais. O kit disponibiliza duas opções de relógio: o SYSCLK e o USER\_CLOCK. O primeiro possui uma frequência fixa de oscilação de 200 MHz. O segundo possui uma frequência original de 156,250 MHz, mas pode ser programado através de uma interface I<sup>2</sup>C para ter frequências entre 10 MHz e 810 MHz. Nota-se porém que o uso do SYSCLK é bem mais simples que o do USER\_CLOCK, não sendo necessário nenhum circuito controlador/programador.

O Xilinx Design Tools disponibiliza uma ferramenta chamada *CORE Generator* para a construção de diversos tipos de circuitos de propriedade intelectual. Dentre eles se encontra o guia *Clocking Wizard*, que pode construir elementos para transformar o sinal do relógio em um sinal simples. Através dele também é possível utilizar as PLLs da placa para modificar a frequência deste sinal para valores entre 20 MHz e 800 MHz.

#### 1.2.1.2 Estrutura de Pastas

A questão da organização do projeto em pastas bem específicas foi bem reforçado na literatura (?????). Os manuais recomendam a estrutura de pastas apresentada abaixo.

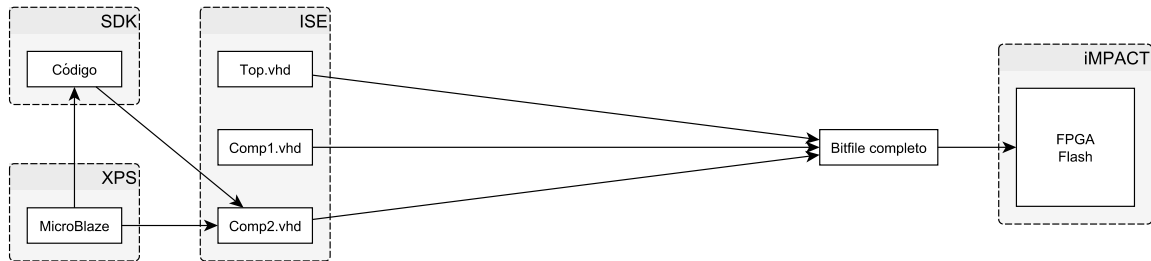
```
Projeto /
  Source /           // codigos-fonte organizados segundo particao
  Implementation /   // contem pastas para cada config. dinamica gerada
  Synth /            // contem pastas com os arquivos .xst e .prj
  Tools /            // ferramentas para automacao da sintese
  PlanAhead /        // pasta para o projeto do PlanAhead
```

A principal razão para esta recomendação é o uso de ferramentas variadas, tais como *scripts* em TCL, o XST e o PlanAhead. Esta estrutura de pastas foi obedecida por ajudar a manter o ambiente de desenvolvimento limpo.

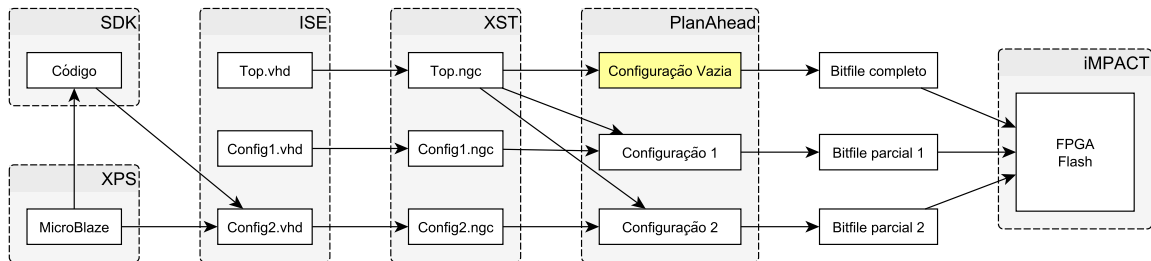
#### 1.2.2 Fluxo de Ferramentas

Uma das primeiras coisas que se destaca no desenvolvimento de dispositivos dinamicamente reconfiguráveis é a diferença no fluxo de ferramentas, também conhecido como *software tools flow*, em relação ao fluxo tradicional (??). Esta diferença é motivada pela necessidade de construção de diversos *bitfiles* parciais, ao contrário de outros projetos, onde objetivo final é um arquivo binário completo.

Como pode-se ver da figura 1.2a, o fluxo tradicional requer apenas o uso do programa ISE, e opcionalmente do XPS e do SDK, para a construção de um projeto de *hardware* e o iMPACT para a programação da FPGA. No fluxo para reconfiguração dinâmica, porém, mostrado na figura 1.2b, além das ferramentas do fluxo tradicional, faz-se necessário o uso da ferramenta XST para a síntese do *netlist* dos comportamentos das partições reconfiguráveis e do PlanAhead para a definição de partições e configurações. Note que estes fluxos não apresentam as únicas opções de fluxo de ferramentas, mas as que foram utilizadas neste projeto.



(a) Foto ilustrativa do fluxo de ferramentas tradicional. Note que o uso do microcontrolador MicroBlaze é opcional, tornando os primeiros blocos, SDK e XPS, também opcionais.



(b) Foto ilustrativa do fluxo de ferramentas para a reconfiguração dinâmica. Assim como no caso tradicional, o uso do SDK e do XPS são opcionais. Note que o bloco em amarelo indica a configuração padrão, que será programada inicialmente. A escolha da configuração padrão é arbitrária.

Figura 1.2: Comparação dos fluxos de ferramentas. Note que estes fluxos não apresentam as únicas opções de fluxo de ferramentas, mas as que foram utilizadas neste projeto.

Note que a reconfiguração parcial pede uma síntese utilizando o método “de baixo para cima” (*bottom-up*), mas uma implementação “de cima para baixo” (*top-down*) (?), ou seja, a síntese acontece primeiro nos elementos de mais baixo nível, mas a implementação deve começar pelos de mais alto nível. Isto acontece para que os requisitos de interfaceamento dos componentes reconfiguráveis sejam encontrados antes de se determinar como incluí-los no projeto, a partir de onde se continua o processo normal de implementação. Note também que a implementação deve garantir que a lógica em comum, ou estática, seja implementada da mesma forma para as diferentes configurações (?).

## 1.3 Implementação

### 1.3.1 Síntese

Com o comportamento do projeto definido na seção 1.1, o próximo passo segundo o fluxo de ferramentas é a síntese, identificada no fluxo de ferramentas da figura 1.2b como XST. Este passo é necessário uma vez que o próximo passo, referente ao PlanAhead, não aceita como entrada códigos-fonte, mas arqui-

vos conhecidos como *netlists*. Os códigos-fonte precisam passar por uma etapa de síntese separada para poderem ser importados no PlanAhead.

O comando XST (*Xilinx Synthesis Tool*), responsável por realizar a síntese, recebe tipicamente um *script* contendo o endereço dos códigos-fonte, o nome do arquivo de saída, o tipo do arquivo de saída, o modelo da FPGA utilizada e uma indicação do código-fonte principal. O comando para iniciar o processo é o seguinte.

```
| xst.exe -ifn Top.xst
```

O arquivo “Top.xst” contém os seguintes comandos.

```
| run
| -ifn Top.prj
| -ofn Top
| -ofmt NGC
| -p xc7k325t-2-ffg900
| -top top
```

O arquivo “Top.prj” contém os endereços dos arquivos, conforme a seguir.

```
| vhdl work "../Sources/static/top.vhd"
| vhdl work "../Sources/static/static.vhd"
| vhdl work "../Sources/static/clocks.vhd"
| vhdl work "../Sources/static/clock_station.vhd"
```

Note que estes comandos e arquivos indicados acima são para síntese dos componentes estáticos. Uma vez que não existe nenhuma restrição especial para tais componentes, eles podem ser sintetizados para um único arquivo de saída. O mesmo não pode ser dito para os elementos dinâmicos. Cada componente dinâmico precisa ser sintetizado em separado para depois ser incluído no projeto através do PlanAhead.

A síntese de componentes dinâmicos precisa ser realizada com um *script* “.xst” ligeiramente diferente. Como mostrado a seguir, faz-se necessária a inclusão do argumento “-iobuf NO”, que desabilita a inserção de componentes de Entrada/Saída (????).

```
| run
| -ifn DynFSM.prj
| -ofn DynFSM
| -ofmt NGC
| -p xc7k325t-2-ffg900
| -top dynamic
| -iobuf NO
```

Note que o arquivo “DynFSM.prj” contém informações sobre o código-fonte do componente dinâmico, como mostrado a seguir.

```
| vhdl work "../Sources/dynamic_fsm/dynamic.vhd"
```

Existe também a possibilidade de construção de um *script* para a síntese automática de todos os arquivos. Utilizou-se aqui uma adaptação do arquivo em linguagem TCL usado pela Xilinx em seus manuais e



exemplos (??????). A única função deste *script* é a construção dinâmica dos comandos com base em listas de arquivos pré-informados.

### 1.3.2 PlanAhead

Com os arquivos sintetizados, pode-se começar a etapa referente ao PlanAhead. Nela, importa-se os arquivos da etapa anterior, cria-se a partição reconfigurável, mapea-se esta partição no dispositivo, cria-se configurações alternativas, promove-se tais configurações e gera-se os *bitfiles* para a programação do dispositivo. Note que é preciso uma licença do ISE que permita o uso do PlanAhead e de reconfiguração parcial para a realização desta etapa.

O primeiro passo necessário no PlanAhead é a criação do projeto. Para isso, após a abertura do programa, clica-se no ícone superior esquerdo mostrado na figura 1.3, onde se lê “Create New Project”. Na janela que aparece, indica-se o nome do projeto e seu caminho, lembrando que foi criada uma pasta anteriormente especificamente para conter este projeto. Indica-se também que o projeto é do tipo “*Post-synthesis Project*” e que deseja-se habilitar a reconfiguração parcial, indica-se quais *netlists* compõe o comportamento estático do sistema e qual destes corresponde ao arquivo principal (“*Top*”), qual é o arquivo de restrições (*constrains*) e qual é o modelo da FPGA.

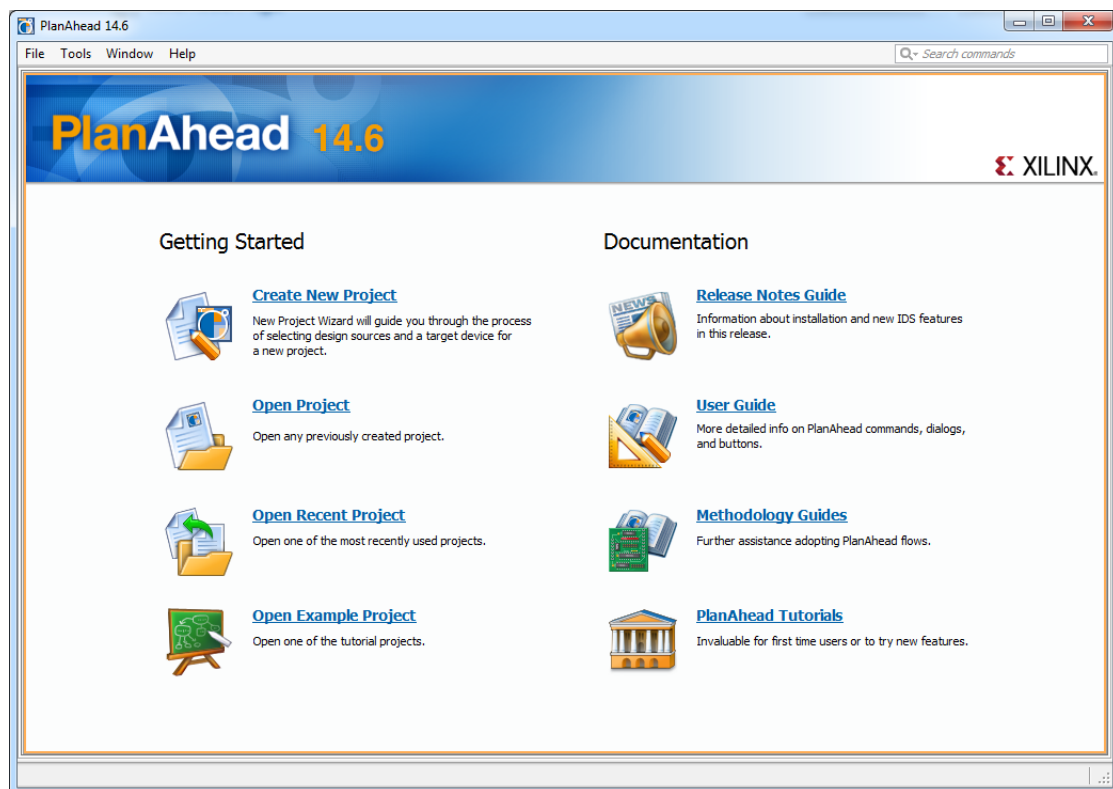


Figura 1.3: Imagem do PlanAhead logo que aberto.

Após a criação do projeto, carregam-se os arquivos sintetizados abrindo “*Open Synthesized Design*”, presente no painel “*Flow Navigator*” sob a opção “*Netlist Analysis*”. Duas janelas de avisos aparecem, informando que existe uma partição não implementada e aviso sobre alguns pinos. Estes avisos podem ser desconsiderados.

Pode-se agora definir a partição que armazenará o componente dinâmico. Isso é feito através do painel “*Netlist*”, selecionando-se a opção “*Set Partition*” do menu que aparece ao se clicar em “dynamic\_i” com o botão direito. Na janela que aparece, seleciona-se a opção referente à partição reconfigurável, nomea-se o módulo reconfigurável de acordo com o componente que será carregado e indica-se o arquivo (NGC) que corresponde ao arquivo sintetizado do componente reconfigurável. Não é necessário informar restrições, visto que o componente, seguindo recomendações, não acessa os pinos de entrada e saída diretamente. Note ainda que é recomendado que o primeiro módulo a ser incluído seja o mais complexo e sucetível a falhas, permitindo que erros e falhas na definição da região a seguir sejam identificados mais cedo.

Precisa-se definir também uma região para a partição recém criada. Isto pode ser feito pelo mesmo painel “*Netlist*”, selecionando-se a opção “*Set Pblock Size*” do menu que aparece ao se clicar em “dynamic\_i” com o botão direito. Nesse momento, precisa-se selecionar na aba “*Device*” uma região do dispositivo que contenha uma quantidade de recursos maior que a requerida pelo projeto. Note que o processo de escolha dessa região não é bem definido, o que abre espaço para diversos erros de alocação. Para testar se a região foi bem alocada, abre-se a aba “*Design Runs*” do painel inferior, clica-se com o botão direito na única configuração disponível no momento e seleciona-se “*Run Launch*”. Este processo pode demorar. Em tentativas subseqüentes, seleciona-se a opção de recomençar todo o processo do zero, evitando de se utilizar arquivos gerados em execução anteriores. Uma região já testada e comprovada para este experimento é a que contém as células (*slice*) de X80Y244 a X139Y205. Ela pode ser selecionada na aba “*Device*”, após se ampliar o mapa de recursos até que os nomes dos elementos lógicos, em cinza, fiquem visíveis. Outra possibilidade, um pouco mais determinística, é descrita na seção 1.4.4.3.

Ao final do “*Design Run*” aparece uma janela que pergunta o que fazer em seguida. Deve-se selecionar a opção “*Promote Partitions*” para exportar o resultado do mapeamento e roteamento da parte estática e informações da partição reconfigurável. Estas informações serão utilizadas nos passos seguintes para construir configurações alternativas compatíveis com a primeira. Na janela que se abre, deve-se marcar as configurações implementadas.

Após promover a primeira configuração, adiciona-se duas novas possibilidades de módulos reconfiguráveis para esta partição: um com comportamento definido e outro vazia. Para isso, clica-se em “*Synthesized Design*” novamente e seleciona-se a opção “*Add reconfigurable module*” do menu que aparece ao se clicar em “dynamic\_i” no painel “*Netlist*” com o botão direito. O processo é o mesmo da definição da partição, sendo a única mudança na seleção do arquivo sintetizado e do nome do módulo. O módulo vazio pode ser criado usando esta mesma opção, mas selecionando-se a opção que indica a inclusão de uma *black box* sem o uso de uma *netlist*.

Com as possibilidades de módulos reconfiguráveis definidas, pode-se construir várias configurações. Isso é feito através do clique com botão direito na “*Design Runs*” do painel inferior e selecionando-se a opção “*Create Runs...*”. A janela que abre possui um painel chamado “*Create Implementation Runs*”. Nesse painel, na coluna “*Partition Action*”, define-se, na coluna “*Module Variant*” da nova janela que aparece, o módulo desejado. Voltando para a primeira janela, clica-se em “*More*” para adicionar a última configuração desejada. Note que, quando definindo o “*Partition Action*”, a linha referente a “*Static Logic*” deve possuir “*Import*” na coluna “*Action*”, indicando que a parte estática não será implementada, mas sim importada de implementações anteriores.

Em seguida, uma janela aparece perguntando o que fazer com estas novas configurações. Deve-se selecionar a opção “*Launch runs on local host*” para iniciar suas implementações. Este processo é demorado, mas mais rápido que o da primeira configuração. Note que os avisos que aparecerem podem ser ignorados.

Também é necessário promover estas novas configurações. No menu de quando se clica com o botão direito sobre as configurações já existentes do painel “*Configurations*”, seleciona-se “*Promote Configuration...*”. A promoção de uma configuração é o equivalente a sua exportação (?). Promover a primeira configuração antes de implementar novas contribui para manter a parte estática, compartilhada, igual em todas as configurações, uma vez que elas não mais são sintetizadas e sim importadas.

Recomenda-se ainda fazer a verificação das configurações através do painel “*Configurations*”. Clicando-se com o botão direito, encontra-se a opção “*Verify Configuration...*”, que faz com que uma janela seja aberta. Selecionado-se todos os itens e clicando em “*OK*”, a verificação se inicia. Nenhum erro deve ser encontrado.

O último passo necessário para a criação dos *bitfiles* é o “*Generate Bitstream*”, localizado no menu a esquerda. Este passo recebe o resultado das sínteses e implementações e transforma-os em *bitfiles*. Ele precisa ser realizado com todas as configurações do “*Design Runs*” selecionados ou alguma delas não terá seus arquivos binários gerados. Após o termino deste processo, os tais *bitfiles* podem ser encontrados na pasta do projeto, dentro das pastas com nome de cada configuração que ficam dentro de da pasta “*\*.runs*”. Existem dois arquivos *.bit* dentro de cada pasta, um maior, que contém a configuração completa, e outro menor, que possui a configuração parcial.

### 1.3.3 iMPACT

Com os *bitfiles* em mãos, usa-se-os para programar a FPGA através da ferramenta iMPACT. A primeira coisa a se fazer após abrir o programa é permitir que o sistema crie um projeto automaticamente. Na janela que se abre, escolhe-se a opção “*Automatically connect to a cable and identify Boundary-Scan chain*” do item “*Configure devices using Boundary-Scan (JTAG)*”. Quando pergunta-se se deseja-se atribuir uma nova configuração, pode-se clicar que sim e escolher um arquivo binário completo gerado na etapa anterior. Normalmente escolhe-se a configuração vazia como configuração inicial para poupar energia.

Quando a configuração for completamente transmitida e implementada, observa-se que um LED está piscando com uma frequência de 2 Hz e todos os outros (acionados) estão acesos. Isto acontece uma vez que o sistema atribui sinal ativo para os elementos desconectados.

Para realizar a reconfiguração parcial dinâmica, clica-se com o botão direito no símbolo do dispositivo que aparece no iMPACT e seleciona-se a opção “*Assign New Configuration File...*”. Procura-se então pelos arquivos binários parciais localizados na pasta *PlanAhead* > *PlanAhead.runs* > “nome da configuração”. Este arquivo possui “*partial*” em seu nome, o que o diferencia do arquivo binário completo. Note que utilizar os arquivos binários completos não gera erro, mas constitui reconfiguração total, não parcial. Após a seleção da configuração desejada, o último passo necessário é a programação, que pode ser realizada clicando-se com o botão direito no dispositivo e selecionando-se a opção “*Program*”.

## 1.4 Resultados

### 1.4.1 Síntese

A síntese ocorreu como esperado, gerando os arquivos sintetizados (NGC). A figura 1.4 mostra o resultado do processo de síntese utilizando o *script* Tcl.

```
>make
Overriding with data file .\Tools\data_synth.tcl
Synthesis tool is set to xst
**** Synthesizing Reconfig Module named <DynCounter> ****
xst -ifn DynCounter.xst
**** Synthesizing Reconfig Module named <DynFSM> ****
xst -ifn DynFSM.xst
**** Synthesizing Reconfig Module named <Top> ****
xst -ifn Top.xst

**** Finished bottom-up synthesis of all RMs ****
```

Figura 1.4: Resultado da síntese.

### 1.4.2 PlanAhead

O passo do PlanAhead foi bem sucedido, tendo gerado os arquivos binários como esperado. O relatório de utilização do sistema, copiado abaixo, mostra que aproximadamente 1% dos recursos do dispositivo foi utilizado.

#### Slice Logic Utilization:

Number of Slice Registers:	2,734 out of 407,600	1%
Number used as Flip Flops:	2,696	
Number used as Latches:	3	
Number used as Latch-thrus:	0	
Number used as AND/OR logics:	35	
Number of Slice LUTs:	3,020 out of 203,800	1%
Number used as logic:	2,572 out of 203,800	1%
Number using O6 output only:	1,962	
Number using O5 output only:	114	
Number using O5 and O6:	496	
Number used as ROM:	0	
Number used as Memory:	338 out of 64,000	1%
Number used as Dual Port RAM:	160	
Number using O6 output only:	96	
Number using O5 output only:	0	
Number using O5 and O6:	64	
Number used as Single Port RAM:	0	
Number used as Shift Register:	178	
Number using O6 output only:	176	
Number using O5 output only:	1	
Number using O5 and O6:	1	

Number used exclusively as route-thrus:	110
Number with same-slice register load:	103
Number with same-slice carry load:	6
Number with other load:	1

Note porém que a quantidade de elementos lógicos ocupados é maior devido ao roteamento. O relatório que mostra este fenômeno foi copiado abaixo.

#### Slice Logic Distribution:

Number of occupied Slices:	1,445 out of 50,950	2%
Number of LUT Flip Flop pairs used:	3,851	
Number with an unused Flip Flop:	1,360 out of 3,851	35%
Number with an unused LUT:	831 out of 3,851	21%
Number of fully used LUT-FF pairs:	1,660 out of 3,851	43%
Number of slice register sites lost to control set restrictions:	0 out of 407,600	0%

A parte estática do dispositivo foi alocada como mostrado na figura 1.5.

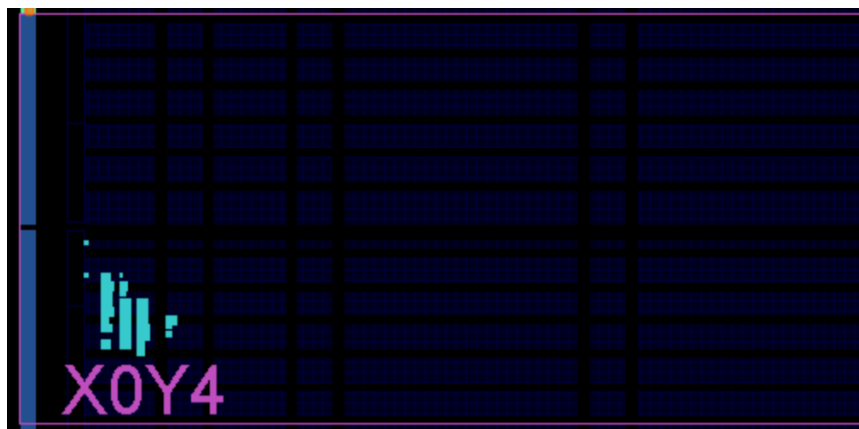


Figura 1.5: Alocação da parte estática do projeto no dispositivo. Em azul claro estão os elementos lógicos utilizados.

### 1.4.3 iMPACT

A etapa do iMPACT também foi bem sucedida. A programação da configuração completa levou 10 segundos e a programação da configuração parcial levou 1 segundo.

### 1.4.4 Possíveis Erros

Esta seção é destinada apenas para a solução de alguns erros encontrados durante a realização deste experimento. É muito provável que, se o experimento for desenvolvido exatamente como mostrado aqui, nenhum destes erros aconteça.

#### 1.4.4.1 Erros no código-fonte

Este é um dos erros mais comuns. A melhor forma de preveni-lo é através da construção dos diversos comportamentos/configurações individuais utilizando o ISE. Para acelerar o processo, realiza-se apenas a síntese. Outra possibilidade é o uso do XST diretamente reduzindo ainda mais o tempo desta verificação.

#### 1.4.4.2 Erro no carregamento do módulo reconfigurável

Este erro, do PlanAhead, em geral acontece quando o componente instanciado no “*Top*” possuem sinais diferentes do instanciado no “*Dynamic*”. Para corrigi-lo, deve-se verificar os códigos-fonte, sintetizá-los novamente, limpar a pasta “PlanAhead” e recomeçar este passo (PlanAhead) novamente.

#### 1.4.4.3 Erros na alocação de partições

Um erro bastante comum que aparece no PlanAhead é o de erro de alocação<sup>1</sup>. Existem duas possíveis formas de corrigi-lo: modificando-se o arquivo de restrições (UCF) ou alterando a região da partição. A primeira forma, que ajuda a garantir que todos os recursos reconfiguráveis estão incluídas na região da partição, é a inclusão de “INCLUSIVE=ROUTE” na linha que contém “INST “dynamic\_i” AREA\_GROUP = “pblock\_dynamic\_i””.

A segunda forma é simplesmente mudando a posição da região da partição para a direita, para a esquerda ou sua largura, de acordo com a mensagem de erro retornada. Esta método não é determinístico e pode ser necessárias várias tentativas antes de se conseguir uma partição mapeável. Este processo pode também ser realizado através do arquivo de restrições (UCF). O código abaixo contém uma alocação de partição funcional para este experimento, podendo ser copiada sobre a definição atualmente existente no arquivo UCF.

```
INST "dynamic_i" AREA_GROUP = "pblock_dynamic_i";
AREA_GROUP "pblock_dynamic_i" RANGE=SLICE_X80Y205:SLICE_X139Y244;
AREA_GROUP "pblock_dynamic_i" RANGE=DSP48_X3Y82:DSP48_X5Y97;
AREA_GROUP "pblock_dynamic_i" RANGE=RAMB18_X3Y82:RAMB18_X5Y97;
AREA_GROUP "pblock_dynamic_i" RANGE=RAMB36_X3Y41:RAMB36_X5Y48;
```

O arquivo UCF pode ser aberto pelo painel “*Sources*” sob a árvore “*Constrains*”. Em caso de mais de um arquivo UCF presente, o desejado pode ser identificado pela marcação “(*target*)” ao lado. Dois cliques são suficientes para abri-lo.

#### 1.4.4.4 Esquecer de promover a partição

A promoção da primeira configuração antes de se implementar as seguintes contribui para a implementação de configurações compatíveis. Esquecer de promover esta partição pode fazer com que erros sejam gerados na etapa de verificação das partições.

<sup>1</sup>AR# 53290: Partial Reconfiguration - 7 series device layout of tiles (CLB, DSP, BRAM, INT) and a shared clocking structure of vertical clock spines between interconnect (routing) tiles. Disponível em <<http://www.xilinx.com/support/answers/53290.htm>>

#### **1.4.4.5 O PlanAhead pode travar enquanto implementando uma configuração**

Apesar de mais raro, o PlanAhead pode travar quando implementando uma configuração. A melhor solução é o reinício do computador, visto que o programa bloqueia alguns arquivos durante a implementação e não os desbloqueia quando fechado forçadamente.

#### **1.4.4.6 Erros na detecção da placa**

Note que algum programa aberto pode interferir com a varredura realizada pelo iMPACT, fazendo-a falhar. Para prevenir este erro, deve-se fechar o XPS, o SDK e qualquer outro programa que possa se utilizar das interfaces USB. Note ainda que a placa deve estar ligada para poder ser detectada.

#### **1.4.4.7 Erro na geração do arquivo binário**

Na geração do arquivo binário, o erro BitGen:342 pode aparecer <sup>2</sup>. Este erro aparece devido ao uso de pinos não explicitados no arquivo UCF, fazendo com que o PlanAhead tenha que usar as configurações padrões nestes pinos. Uma forma de solucionar este problema é acrescentando a opção “-g Unconstrained-Pins:Allow” no comando do BitGen.

### **1.5 Conclusão**

O processo de desenvolvimento de partições reconfiguráveis e sua programação foi explorado e compreendido com sucesso. Observou-se os pontos críticos do desenvolvimento e o fluxo mais adequado para a construção deste tipo de projeto, bem como alguns erros comuns e suas soluções. O próximo passo segundo os objetivos mostrados anteriormente é a identificação de requisitos.

---

<sup>2</sup>“AR# 51813 14.2 BitGen - "ERROR:Bitgen:342 occurs after adding probes to the design in the case of 7 series devices"”, <<http://www.xilinx.com/support/answers/51813.html>>

## Capítulo 2

# Experimento 2 - Memórias

### 2.1 Definição do Teste

Seguindo o raciocínio apresentado no capítulo ??, o próximo passo natural no desenvolvimento deste projeto é o entendimento do funcionamento das memórias. Esta etapa abre caminho para que se armazene os arquivos binários de configurações parciais em uma memória embarcada, removendo a necessidade do computador para tal. Para que se valide o correto entendimento do funcionamento e uso das memórias, será realizado um teste que mostre que eles puderam ser lidos e escritos corretamente.

### 2.2 Estudo dos Requisitos

Para o desenvolvimento deste experimento, faz-se necessário o estudo dos diferentes tipos de memórias, citando seus pontos fracos e fortes. Também é necessário o estudo do MicroBlaze e das ferramentas XPS e SDK, visto que a forma mais direta de se trabalhar com memória em FPGAs é através do uso de microcontroladores embarcados.

#### 2.2.1 Tipos de Memória

No kit utilizado existem vários tipos de memórias que poderiam ser usados, cada um com suas peculiaridades (?). Este experimento foi dedicado à compreensão do funcionamento dos diversos tipos de memórias e à escolha da solução mais adequada.

##### 2.2.1.1 Memória *Block RAM*

A memória do tipo *Block RAM* é construída usando-se os blocos de memória RAM restantes da FPGA. Esta memória consegue alcançar velocidades de leitura na ordem de várias centenas de hertz, mas possui uma capacidade de armazenamento bem reduzida, de apenas 445 blocos de 36 Kb para este kit, totalizando um máximo de aproximadamente 2 MB (????). Note que a configuração total da FPGA necessita



de aproximadamente 11 MB, ou exatamente 91.548.896 bits, para sua programação total (??). Pode-se programá-la através do iMPACT, dentre outras formas <sup>1</sup>.

#### **2.2.1.2 Memória *Distributed RAM***

A memória *Distributed RAM* é construída utilizando-se das LUTs disponíveis nas células lógicas (????). Também são muito rápidas, apesar de síncronas, e também possuem pouca capacidade de armazenamento. Pode-se programá-la através do iMPACT, dentre outras formas.

#### **2.2.1.3 Memória *Linear BPI Flash***

A memória *Linear BPI Flash* disponibiliza 128 Mb de memória não-volátil (??), acessadas em palavras de 16 bits. Apesar disso, sua velocidade de leitura máxima é de 33 MHz. Convertendo tal velocidade para a leitura de 32 bits, tem-se uma velocidade de leitura de aproximadamente 16 MHz. Pode-se programá-la através do iMPACT.

#### **2.2.1.4 Memória *SPI Flash***

A memória SPI Flash é diferente na sua forma de acesso, que acontece através da interface SPI. Esta memória fornece 128 Mb de memória não volátil (??). Ela aceita três modos de operação, x1, x2 e x4, que corresponde a largura da palavra lida/escrita a cada pulso de relógio (??). A frequência de operação é de no máximo 50 MHz (??). Pode-se programá-la através do iMPACT.

#### **2.2.1.5 Cartão de Memória SD**

O cartão de memória SD dá acesso a uma memória não-volátil de tamanho arbitrário. Esta interface permite o uso de cartões de alto desempenho, lendo palavras de 4 bits a frequências de até 50 MHz (??). A limitação desta interface é a dificuldade de leitura e escrita devido ao sistema de arquivos inerente ao cartão. Note que também não existe a possibilidade de programação do cartão através do iMPACT, o que resolveria o problema do sistema de arquivos.

#### **2.2.1.6 *CompactFlash* e o *System ACE***

*System ACE* é um sistema que permite a programação de FPGAs e memórias voláteis a partir de um cartão *CompactFlash* (????). Este é bem robusto e popular em séries que possuem um leitor deste tipo de cartão, o que não é o caso desta.

---

<sup>1</sup>*Memory Initialization Methods*, escrito por Jim Wu em 31 de dezembro de 2011. Disponível em <<http://myfpgablog.blogspot.com.br/2011/12/memory-initialization-methods.html>>

### 2.2.1.7 Memória DDR3

A memória DDR3 é uma memória volátil com 1 GB de capacidade de armazenamento e possui uma frequência de operação da ordem dos 800 MHz (??). Só pode ser programada apenas em tempo de execução, fazendo-se necessário o uso de um *bootloader*. Ela possui restrições de temporização extremamente apertadas e necessita do uso de componentes bastante escassos no FPGA.

### 2.2.2 Escolha da Memória

Este experimento tem por objetivo o desenvolvimento de um sistema que instancie e acesse memórias capazes de armazenar as configurações parciais geradas no experimento anterior. Para isto, deve-se entender o processo de construção de um processador embarcado MicroBlaze, com periféricos para acesso às memórias, e de construção de programa embarcado. Deve-se ainda, antes de qualquer coisa, definir que tipo de memória é a mais apropriada para o objetivo traçado.

Nota-se do primeiro experimento que os *bitfiles* parciais gerados possuíam 645 KB cada, totalizando 1935 KB, ou 1,89 MB. Com isso o uso das memórias que se utilizam de recursos internos da FPGA fica comprometido. Observa-se também que a interface de reprogramação dinâmica, ICAPE2, utilizada em experimentos mais a frente, opera a uma frequência de até 100 MHz e pode receber palavras de até 32 bits, ou seja, 3200 Mb/s (??). Por causa disso, todas as interfaces não-voláteis disponíveis acabariam se tornando gargalos no tempo de programação.

A única memória com tamanho e velocidade suficientemente grandes para este projeto é a memória DDR3. Apesar disto, esta memória é volátil, necessitando ser inicializada sempre que o sistema é ligado. Uma solução para este problema é a implementação de um sistema que carregue as informações de uma memória não-volátil, tanto a Linear Flash quanto a SPI Flash seriam suficientes, para a memória DDR3 em um momento inicial, e depois permita que a memória DDR3 seja acessada para a reconfiguração dinâmica, ou seja, através do uso de um *bootloader*.

Sendo assim, optou-se por utilizar a memória QSPI Flash para uso na inicialização e a memória DDR3 para uso na execução.

### 2.2.3 MicroBlaze

O MicroBlaze é um microprocessador otimizado para implementação em FPGAs da Xilinx (??). Ele possui 32 registradores genéricos de 32 bits, instruções de 32 bits e endereços de 32 bits. Seu *pipeline* possui 3 ou 5 estágios e é construído em torno da arquitetura Harvard, como pode ser observado na figura 2.1. Todas as outras configurações, tais como o uso de *Big Endian* ou *Little Endian*, por exemplo, são opcionais (??).

O MicroBlaze permite o uso de diversas interfaces para comunicação com seus diversos periféricos, dentre elas a PLB, a LMB, a AXI e a ACE (??). A interface mais atual suportada, e que foi utilizada neste experimento, é a Advanced eXtensible Interface 4 (AXI4) (????). A AXI4 é uma interface mapeada em memória que oferece produtividade, flexibilidade e disponibilidade. Ela possui três tipos de interfaces, a

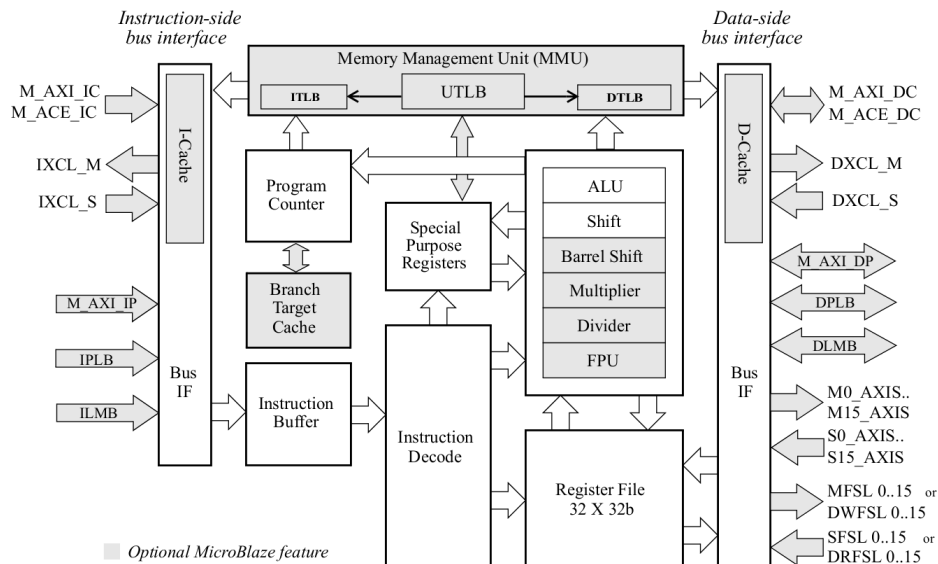


Figura 2.1: Diagrama de blocos do MicroBlaze.

AXI4, a AXI4-Lite e a AXI4-Stream, onde as duas primeiras são compostas de 5 canais de comunicação: de leitura de endereço, de escrita de endereço, de leitura de dados, de escrita de dados e de escrita de resposta.

## 2.2.4 Fluxo de Projeto

O procedimento para a construção de um projeto com MicroBlaze segue o fluxo descrito na figura 1.2a. A figura 2.2 apresenta o fluxo de ferramentas para este caso específico. Em outras palavras, o fluxo espera que primeiro se desenvolva o microprocessador com todos os seus periféricos para depois se desenvolver o programa que será embarcado. Tanto o programa quando o processador gerarão arquivos binários que serão carregados pelo iMPACT através do SDK.

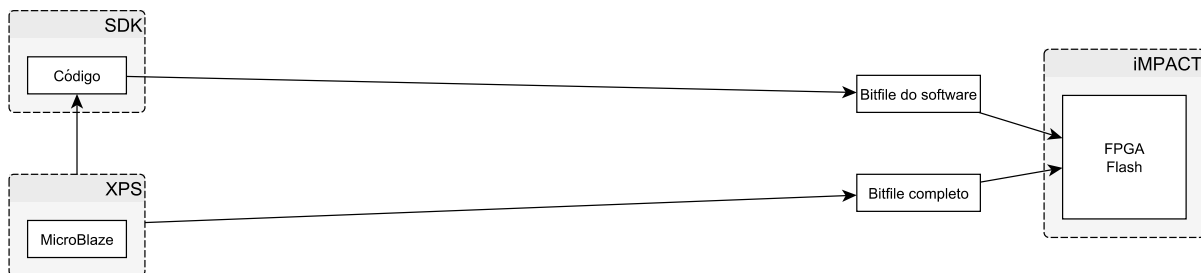


Figura 2.2: Foto ilustrativa do fluxo de ferramentas para o desenvolvimento de sistemas com MicroBlaze, extraído de (??).

## 2.3 Implementação

### 2.3.1 XPS

Para começar, deve-se abrir o programa e criar um novo projeto usando o *Base System Builder (BSB)*, opção que corresponde ao item superior esquerdo do menu da tela inicial. Na janela que aparece, escolheu-se a placa de desenvolvimento *Kintex-7 KC705 Evaluation Platform, Board Revision C*, e a opção de um só processador no sistema, para simplificar o projeto. Dando prosseguimento, escolheu-se os periféricos desejados segundo a figura 2.3 e o tamanho das memórias local, de intrusão e de dados, quíça, 128 KB, 8 KB e 8 KB respectivamente. Modificou-se ainda, como pode-se ver na figura 2.3, o *Baud Rate* da interface UART para 115200 bits/s e o *C\_SPI\_MODE* da *QSPI\_FLASH* para “*Quad SPI Mode*”.

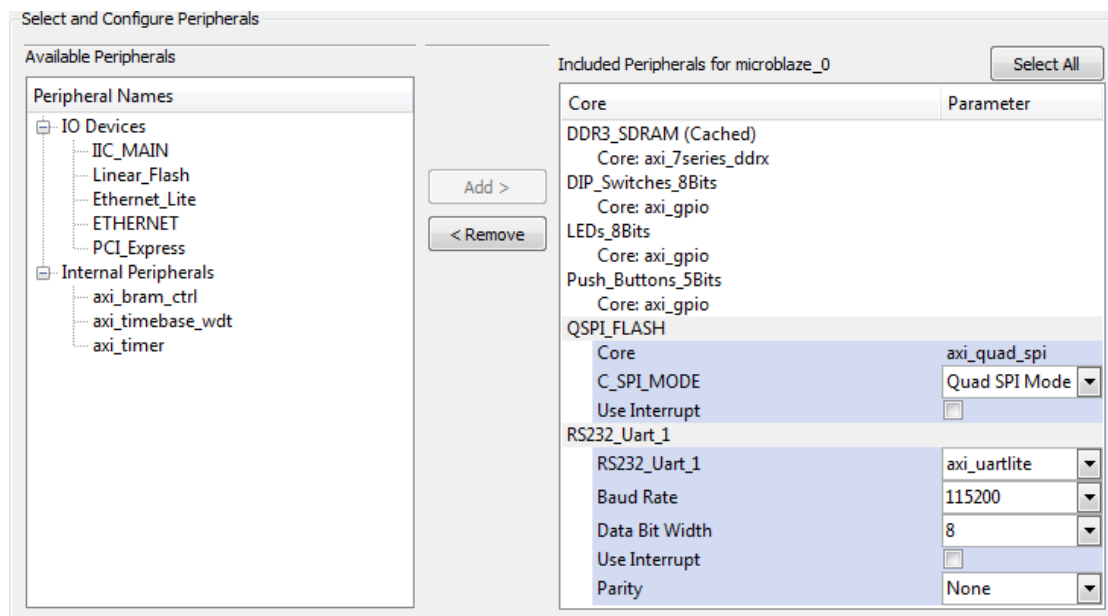


Figura 2.3: Escolha dos periféricos no BSB do XPS.

Antes de concluir o a construção do sistema, precisa-se ajustar os tamanhos das memórias e seus endereços, de forma que estes novos tamanhos possam ser corretamente acessados. Isto pode ser feito na aba *Addresses* do painel *System Assembly View*. Muda-se então o tamanho da memória SPI Flash para 128M, seu endereço base para 0x80000000, o tamanho da memória DDR3 para 1G e seu endereço base para 0xC0000000, conforme a figura 2.4. Note que o endereço-base da memória SPI Flash tem como única restrição de os 28 bits menos significativos iguais a zero e que o endereço-base da memória DDR3 tem esta mesma restrição para os 30 bits menos significativos. Estas restrições são devido aos tamanhos das memórias e o alinhamento dos espaços de dados na memória.

Precisa-se ainda modificar as posições de memória cobertas pela memória *cache*. Isto é feito clicando-se duas vezes sobre “microblaze\_0” na aba “*Bus Interfaces*” do painel “*System Assembly View*”. Na janela que se abre, clica-se “*Next*” 3 vezes para chegar página sobre *caches*. Modifique os endereços nas duas colunas para 0xc0000000 e 0xffffffff, indicando que a memória de instruções e memória de dados podem acessar os endereços da memória DDR3. Apenas os endereços contidos neste intervalo da memória de dados podem ser escritos. Note que o sistema reserva para uso próprio os primeiros 64K endereços das

Instance	Base Name	Base Address	High Address	Size
microblaze_0's Address Map				
microblaze_0_d_bram_ctrl	C_BASEADDR	0x00000000	0x0000FFFF	64K
microblaze_0_i_bram_ctrl	C_BASEADDR	0x00000000	0x0000FFFF	64K
LEDs_8Bits	C_BASEADDR	0x40000000	0x4000FFFF	64K
RS232_Uart_1	C_BASEADDR	0x40600000	0x4060FFFF	64K
debug_module	C_BASEADDR	0x41400000	0x4140FFFF	64K
QSPI_FLASH	C_BASEADDR	0x80000000	0x87FFFFFF	128M
DDR3_SDRAM	C_S_AXI_BASEA...	0xC0000000	0xFFFFFFFF	1G

Figura 2.4: Aba *Addresses* do *System Assembly View* indicando os ajustes dos endereços e tamanhos das memórias.

memórias, que correspondem às posições com finais entre 0x0000 a 0x3fff. A tentativa de uso destes endereços comprometerá o correto funcionamento do sistema.

Algumas outras configurações também podem ser ajustadas, mas não são estritamente necessárias para este experimento.

O projeto pode ser sintetizado através do botão “*Generate Netlist*” localizado no menu à esquerda e no menu “*Hardware*” da barra de menus. Este processo é demorado. Quando terminado, pode-se exportar o projeto através do botão “*Export Design*” para abrir o SDK com as informações deste processador. Na janela que aparecer, marque “*Include bitstream and BMM file*” e clique em “*Export & Launch SDK*”. O arquivo binário contém informações da configuração da FPGA enquanto o arquivo BMM apresenta um mapeamento das unidades de memória onde o programa embarcado pode ser inserido. Após a execução desta etapa, se nenhum erro tiver acontecido, a ferramenta SDK será aberta.

### 2.3.2 SDK

Quando a janela do SDK aparecer, ela perguntará onde se quer colocar o *workspace*. Uma boa opção é a pasta SDK criada dentro da pasta do projeto do sistema durante sua exportação, apesar desta escolha ser completamente arbitrária. Escolhida a pasta, o ambiente de trabalho é aberto.

Começa-se o processo criando um projeto do tipo “*Board Support Package*”. Este projeto compila automaticamente os drivers disponíveis para o projeto segundo as características do microcontrolador. Em seguida, pode-se criar os projetos dos *softwares* que irão embarcados. Para isto, cria-se um projeto do tipo “*Application Project*”. Na janela que se abre, nomeie o projeto e selecione a “*Board Support Package*” no *dropdown* do item “*Board Support Package*”. Na página seguinte, escolhe-se “*Empty Project*”.

Adiciona-se arquivos ao projeto tanto arrastando os códigos para ele quanto selecionando a opção “*New/Source File*” do menu que aparece quando se clica no projeto com o botão direito. Note que o acesso à memória é feito simplesmente dereferenciando um ponteiro para a posição de memória desejada. A escrita é feita do mesmo modo. Encontra-se em anexo códigos-exemplo para o teste das memórias DDR3 e SPI Flash.

Antes de executar o programa, é interessante habilitar a opção de depuração. Isto é feito através do menu “*Run*”, na opção “*Run Configurations...*”. Na janela que aparece, na aba “*STDIO Connection*”, habilita-se a conexão do STDIO com o console e modifica-se o “*Baud Rate*” para 115200. Clica-se então

em “Apply” e em seguida “Close”.

A programação do FPGA pode ser feita através do menu “Xilinx Tools”, na opção “Program FPGA”. Na janela que se abre, é padrão que as informações já estejam pré-preenchidas, mas caso isto não aconteça, procura-se pelos arquivos “system.bit” e “system\_bd.bmm” na pasta “implementations” na raiz do projeto do processador. Clica-se em “Program” para iniciar a programação.

Para se transferir o programa criado com o auxílio do SDK, seleciona-se o projeto deste programa, clica-se com o botão direito, seleciona-se o submenu “Run As...” e escolhe-se a opção “Launch on Hardware (GDB)”. No caso dos códigos-exemplo, algumas informações são imprimidas no console caso tudo tenha ocorrido conforme o esperado.

## 2.4 Resultados

O experimento foi bem sucedido. A seguir são apresentados alguns relatórios e amostras dos resultados.

### 2.4.1 XPS

O XPS, assim como o ISE, fornece relatórios que apresentam a utilização do dispositivo. Este relatório está mostrado a seguir. Note que mesmo com um sistema bem mais complexo, o grau de utilização do dispositivo não aumentou muito com relação ao experimento passado.

Slice Logic Utilization:

Number of Slice Registers:	7,775 out of 407,600	1%
Number used as Flip Flops:	7,720	
Number used as Latches:	0	
Number used as Latch-thrus:	0	
Number used as AND/OR logics:	55	
Number of Slice LUTs:	8,916 out of 203,800	4%
Number used as logic:	7,603 out of 203,800	3%
Number using O6 output only:	5,826	
Number using O5 output only:	173	
Number using O5 and O6:	1,604	
Number used as ROM:	0	
Number used as Memory:	1,014 out of 64,000	1%
Number used as Dual Port RAM:	572	
Number using O6 output only:	120	
Number using O5 output only:	12	
Number using O5 and O6:	440	
Number used as Single Port RAM:	0	
Number used as Shift Register:	442	
Number using O6 output only:	441	
Number using O5 output only:	1	

Number using O5 and O6:	0
Number used exclusively as route-thrus:	299
Number with same-slice register load:	268
Number with same-slice carry load:	30
Number with other load:	1

#### Slice Logic Distribution:

Number of occupied Slices:	3,944 out of 50,950	7%
Number of LUT Flip Flop pairs used:	11,005	
Number with an unused Flip Flop:	3,939 out of 11,005	35%
Number with an unused LUT:	2,089 out of 11,005	18%
Number of fully used LUT-FF pairs:	4,977 out of 11,005	45%
Number of slice register sites lost to control set restrictions:	0 out of 407,600	0%

### 2.4.2 SDK

Os resultados apresentados pelo SDK se resumem a uma compilação e uma programação de dispositivos bem sucedidas.

## 2.5 Conclusão

Conclui-se assim o experimento para o teste de programação das memórias. Notou-se que existe muita pouca literatura no assunto, forçando o programador a fazer uso dos fóruns de discussão e conhecimentos gerais de programação embarcada. Apesar disso, o objetivo do experimento, quiza conseguir ler/escrever de/em endereços das memória DDR3 e SPI Flash específicos, foi atingido com sucesso.

Uma forma interessante de dar continuidade a esta linha de pensamento é entender como funciona a programação da memória Flash através do computador, uma vez que este experimento apenas escreveu na memória através do próprio microcontrolador, e uma possível interpretação do arquivo binários, extraindo deles o máximo de informações possíveis.

## Capítulo 3

# Experimento 3 - *Bootloader*

### 3.1 Definição do Teste

O *bootloader*, como mencionado no experimento anterior, é um sistema que carrega as informações de uma memória lenta, como a SPI Flash, para uma memória rápida, como a DDR3. Usou-se um microcontrolador MicroBlaze com interfaces para as memórias SPI Flash e DDR3. O experimento 2 foi dedicado a aprender a utilizar estas memórias. Neste experimento, espera-se entender como é formado o arquivo binário, permitindo assim carregá-lo e interpretá-lo enquanto o transferindo para a memória DDR3.

Para se alcançar o objetivo proposto, faz-se necessário entender como é construído o arquivo binário e como se inicializa a memória não-volátil. O fluxo de projeto é o mesmo do experimento anterior.

### 3.2 Estudo dos Requisitos

#### 3.2.1 Arquivo Binário

O arquivo binário é formado por três partes: um cabeçalho, uma palavra para sincronia e a configuração propriamente dita (????). A figura 3.1 apresenta o início deste arquivo, contendo as partes mais importante do cabeçalho. Os bytes selecionados correspondem ao conteúdo legível do cabeçalho.

00000	00 09 0f f0 0f f0 0f f0 0f f0 00 00 01 61 00 23	...ð.ð.ð.ð...a.#
00010	42 6c 61 6e 6b 5f 72 6f 75 74 65 64 2e 6e 63 64	Blank_routed.ncd
00020	3b 55 73 65 72 49 44 3d 30 78 46 46 46 46 46 46	;UserID=0xFFFFFFFF
00030	46 46 00 62 00 0d 37 6b 33 32 35 74 66 66 67 39	FF.b..7k325tffg9
00040	30 30 00 63 00 0b 32 30 31 33 2f 31 31 2f 32 36	00.c..2013/11/26
00050	00 64 00 09 30 38 3a 35 34 3a 31 31 00 65 00 0a	.d..08:54:11.e..
00060	16 c4 ff ff ff ff ff ff ff ff ff ff ff ff ff	.Ayyyyyyyyyyyyyy

Figura 3.1: Cabeçalho do arquivo binário gerado no primeiro experimento para a configuração vazia.

O cabeçalho é formado por chaves e tamanhos, indicando os diversos campos deste. Ele contém pelo menos duas informações muito interessantes: o identificador do dispositivo alvo, que permite verificar a compatibilidade entre a configuração e o dispositivo que a está recebendo, e o tamanho da configuração, que permite que ela seja carregada de forma dinâmica sem necessidade de mais informações.



Na tabela 3.1 pode-se observar os tamanhos e campos apresentados na figura 3.1 <sup>1</sup>.

Tamanho	Chave	Significado
2 bytes	9 (0x00 09)	Tamanho em bytes do próximo campo
9 bytes	0x0f f0 0f f0 0f f0 0f f0 00	Indica que a configuração a seguir é válida.
2 bytes	1 (0x00 01)	Tamanho em bytes do próximo campo
1 byte	"a"(0x61)	Indica que os próximos campos conterão informações sobre o projeto e sobre a configuração.
2 bytes	35 (0x00 23)	Tamanho em bytes do próximo campo
35 bytes	Blank_routed.ncd; UserID=0xFFFFFFFF	Apresenta o nome do <i>netlist</i> e o identificador do usuário. 0x00 ao final indica o final da string.
1 byte	"b"(0x62)	Indica que o próximo campo é um indentificador do dispositivo-alvo.
2 bytes	13 (0x00 0d)	Tamanho em bytes do próximo campo
13 bytes	7k325tffg900	Identificador do dispositivo-alvo. 0x00 ao final indica o final da string.
1 byte	"c"(0x63)	Indica que o próximo campo é a data de síntese da configuração.
2 bytes	11 bytes (0x00 0b)	Tamanho em bytes do próximo campo
11 bytes	2013/11/26	Data da síntese da configuração.
1 byte	"d"(0x64)	Indica que o próximo campo é a hora de síntese da configuração.
2 bytes	9 bytes (0x00 09)	Tamanho em bytes do próximo campo
9 bytes	08:54:11	Hora de síntese da configuração.
1 byte	"e"(0x65)	Indica que os próximos 8 bytes contém o tamanho da configuração.
4 bytes	661188 (0x00 0a 16 c4)	Tamanho em bytes da configuração a partir desta posição.

Tabela 3.1: Descrição do cabeçalho dos arquivos binários.

Existe ainda no cabeçalho 32 bytes de espaçamento preenchidos por com “0xff” e bytes para autode-  
tecção de largura de banda (????). Estes bytes (“0x00 00 00 bb 11 22 00 44”) são usados no modo de  
configuração paralela para detectar automaticamente a largura de banda do arquivo de configuração. O  
modo serial ignora todos os bits anteriores a palavra de sincronia (??). Estes bits são então usados apenas  
para pré-processamento do arquivo binário.

A palavra de sincronia (“0xaa 99 ff 66”), encontrada a seguida, serve para indicar o início da configu-  
ração propriamente dita e para alinhar o fluxo de dados nos registradores internos.

### 3.2.2 Inicialização da Memória SPI Flash

A memória SPI Flash, assim como todas as outras, precisa de um procedimento especial para poder ser  
inicializada com informações arbitrárias (??). Em geral, as únicas informações que podem ser gravadas  
nas memórias não-voláteis são configurações para o FPGA e programas para algum MicroBlaze embarcado  
(??). Este processo é conhecido como programação indireta da memória Flash (??).

<sup>1</sup>“FAQ: Tell me about the format of the .BIT files please”, <[http://www.fpga-faq.com/FAQ\\_Pages/0026\\_Tell\\_me\\_about\\_bit\\_files.htm](http://www.fpga-faq.com/FAQ_Pages/0026_Tell_me_about_bit_files.htm)>

### 3.2.2.1 Compilação

Para a realização da programação indireta, o arquivo binário precisa ser compilado de forma a gerar um padrão de bits compatível. Este procedimento é realizado na etapa de construção do processador. Quando não especificado, o arquivo binário gerado utiliza a interface QSPI x1, que possui banda de transferência de 1 bit por leitura/escrita, e relógio de frequência 3 MHz. Com estas configurações, a programação do sistema demora apenas 1 minutos e 30 segundos (??), mas pode ser ainda mais reduzido.

No PlanAhead, as configurações de compilação podem ser modificadas através do arquivo “bitgen.ut” localizado na pasta “etc” do projeto. Através da opção “-g SPI\_buswidth:X”, onde X pode ser 1, 2 ou 4, pode-se alterar a interface utilizada neste tipo de programação (????), sendo a x4 a mais eficiente. Pode-se ainda forçar a opção “-g ConfigRate\_en:Y”, onde Y pode ser 3, 6, 9, 12, 16, 22, 26, 33, 40, 50 ou 66, para se utilizar de relógios de variadas frequências e obter assim o modo de configuração mais adequado para a memória em questão (?????). Existe também a opção “-g SPI\_Fall\_Edge:Yes”, que permite melhora margens de tempo e pode aumentar as taxas de leitura para configurações (????). Uma opção alternativa é utilizar um relógio externo através da opção “-g ExtMasterCclk\_en:Z”, onde Z pode ser “Disable”, “div-8”, “div-4”, “div-2” e “div-1”.

### 3.2.2.2 Arquivo de Memória PROM

Após compilado o projeto, seu arquivo binário e qualquer outra informação a ser programada na memória Flash precisa ser adicionada a um arquivo do tipo MCS. Este processo é necessário para que o iMPACT consiga carregar e programar a memória Flash de forma correta. Pode-se construir este arquivo de memória PROM através do próprio iMPACT, processo que será descrito mais a frente.

## 3.3 Implementação

O teste deste experimento compreende a construção de um sistema microprocessador que carregue os arquivos binários da memória SPI Flash para a memória DDR3. Este experimento contribuirá para a compreensão do processo de inicialização de memórias não-voláteis e do tratamento do cabeçalho dos arquivos binários. Usará-se os programas XPS, SDK, que fazem parte do Embedded Development Kit (EDK), data2mem e iMPACT.

### 3.3.1 XPS

Utilizou-se o mesmo microcontrolador MicroBlaze construído no experimento passado, não sendo necessária nenhuma modificação. A interface SPI utilizada aqui foi a padrão, x1, por motivos de simplificação do projeto. O uso de uma interface x4 diminuiria o tempo de programação em 4 vezes, o que não é fator crítico para este experimento, mas acarretaria na necessidade de recompilar todos os projetos desenvolvidos até agora.

### 3.3.2 SDK

Logo após a construção do microcontrolador, recomenda-se construir o projeto do programa embarcado. O procedimento para o SDK nesse caso é bem semelhante ao dos experimentos anteriores, mudando-se apenas os arquivos importados e a geração do *linker script*. Os arquivos a serem importados podem ser encontrados no CD de anexos.

O projeto do bootloader consiste apenas de uma máquina de estados para ler o cabeçalho do arquivo binário e interfaces com os drivers controladores das memórias DDR3 e SPI Flash e da interface de comunicação através da porta UART.

Desta vez, tem-se como objetivo fazer com que o sistema se carregue e entre em funcionamento de forma autônoma. Para isto, precisa-se, depois da inclusão dos devidos arquivos, presentes no anexo, gerar o *linker script*. Este arquivo descreve como o arquivo binário do *bootloader* deve ser armazenado na memória interna do FPGA para execução. Este *script* pode ser gerado clicando-se com o botão direito sobre o projeto do programa embarcado e selecionando-se a opção “*Generate Linker Script...*” ou selecionando-se o projeto, abrindo-se o menu “*Xilinx Tools*” e selecionando-se a opção do mesmo nome. Para o escopo deste experimento, as configurações apresentadas na janela que se abre são suficientes, bastando clicar em “*Generate*”. A criação deste *script* permite agora que se utilize o programa “*data2mem*” para construir um arquivo binário de configuração com um programa embarcado pré-programado.

### 3.3.3 data2mem

A ferramenta *data2mem* funciona através da linha de comando, mas normalmente é acionada através de programas com interfaces gráficas, tais como o ISE, o XPS, o SDK e o iMPACT (??). Sua função principal é o de mapear blocos contíguos de dados entre múltiplas *Block RAMs* distribuídas pelo FPGA mantendo um acesso lógico contínuo, i.e., dados em endereços de memória adjacentes podem estar em blocos completamente diferentes. No caso em questão, ela mapeará o *bootloader* desenvolvido nas *Block RAMs* de forma que no momento da programação, o MicroBlaze embarcado já possua um programa carregado.

Um comando típico do *data2mem* é construído com as opções “-bm” para indicar o caminho para o arquivo do tipo “*Block RAM Memory MAP*” (BMM), “-bt” para indicar o caminho para o arquivo binário do tipo BIT, “-bd” para indicar arquivos de programas do tipo ELF ou MEM, permitindo a inclusão de um identificador para associá-lo a algum dispositivo implementado, e “-o b” para indicar o caminho do arquivo binário (BIT) de saída. Um exemplo de uso do comando é apresentado abaixo.

```
data2mem \  
-bm SDK/XPS_QSPI_Final_hw_platform/system_bd.bmm \  
-bt SDK/XPS_QSPI_Final_hw_platform/system.bit \  
-bd SDK/bootloader/Release/bootloader.elf tag microblaze_0 \  
-o b SDK/XPS_QSPI_Final_hw_platform/download.bit
```

O comando acima também pode, como foi dito anteriormente, ser executado através do SDK. Para isto, deve-se acessar o menu “*Xilinx Tools*” e selecionar a opção “*Program Flash*”. Uma janela aparece, onde deve-se selecionar os arquivos BIT e BMM gerados pelo XPS na compilação do sistema e o arquivo ELF gerado na compilação do programa embarcado. Este procedimento pode ser feito com a placa de desen-

volvimento conectada ou não, gerando um erro que pode ser desprezado quando ela estiver desconectada. O arquivo gerado pode ser encontrado na pasta “SDK/\*\_hw\_platform” sob o nome “download.bit”.

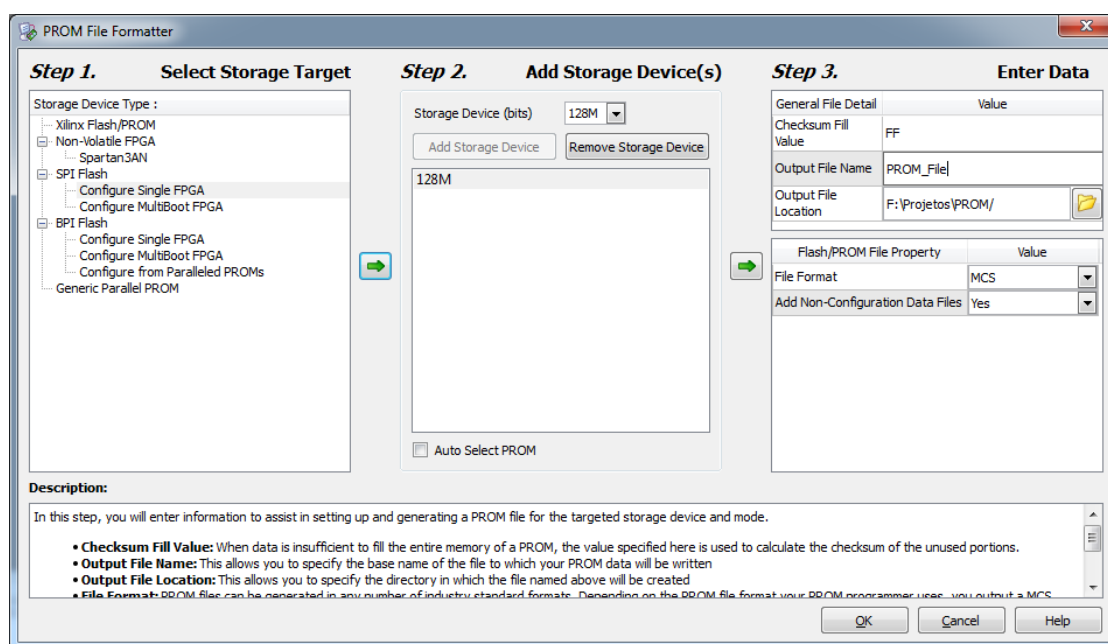


Figura 3.2: Janela para criação de um arquivo de memória PROM com as configurações devidamente ajustadas.

### 3.3.4 Programação Indireta da Memória Flash

Como todos os arquivos binários prontos, pode-se começar o processo de programação indireta da memória Flash. Este processo se inicia através da criação de um arquivo de memória PROM através da ferramenta iMPACT. Vale salientar apenas que faz-se necessário que todos os elementos (configurações totais e parciais) sejam previamente compilados para uma mesma interface SPI, seja ela x1, x2 ou x4. O uso de interfaces diferentes pode gerar erros durante a programação da memória Flash.

No momento da criação do novo projeto do iMPACT, seleciona-se a opção “*Prepare a PROM File*”. Na janela seguinte, seleciona-se “*SPI Flash/Configure Single FPGA*” no primeiro painel e “128M” no segundo e modifica-se “*Add Non-Configuration Data Files*” para “Yes”, conforme mostrado na figura 3.2.

Em seguida, adicionam-se os arquivos binários que se deseja programar na memória Flash. O primeiro arquivo a se adicionar é o de configuração total. Este arquivo é carregado durante o procedimento de início do FPGA. Apenas um arquivo deste tipo precisa ser carregado neste experimento, apesar de ser possível realizar um projeto com diversas revisões de configurações ou múltiplas possibilidades de configurações de *boot* (????). Para rejeitar a adição de outras configurações, deve-se clicar em “No” na mensagem de título “*Add Device*”.

A mensagem seguinte, “*Add Data File*”, faz referência a adição de arquivos de dados neste projeto. Clicando-se em “Yes”, uma janela aparece com informações de endereçamento. Pode-se aceitar os valores iniciais. Estes arquivos podem ter qualquer conteúdo, mas o programa espera arquivos gerados pelo SDK, sendo necessário mudar a configuração de arquivos apresentados para “*All Files (\*.\*)*”. Após adicionar-

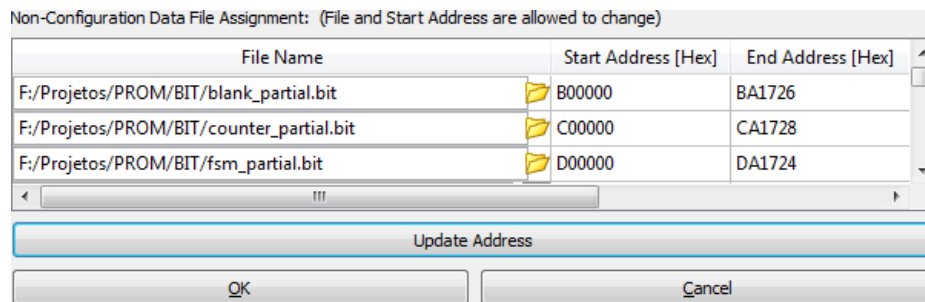


Figura 3.3: Janela para criação de um arquivo de memória PROM com as configurações devidamente ajustadas.

se todos os arquivos, deve-se clicar em “No” na janela de inclusão de novos arquivos de dados. Ao se fazer isto, uma janela para indicação dos endereços é mostrada. Recomenda-se mudar os endereços de início (“*Start Address*”) para valores arredondados, como 0xB00000, 0xC00000 e 0xD00000, obedecendo os endereços das revisões, de forma a facilitar o trabalho de programação do sistema embarcado. O botão “*Update Address*” deve ser clicado para ajustar os endereços de fim (“*End Address*”) antes de se prosseguir, obtendo-se algo similar a figura 3.3. Note que é possível incluir também um programa para o MicroBlaze na memória Flash, a ser carregado em tempo de execução, para controlar a mudança de configurações.

O último passo é gerar o arquivo, o que pode ser feito através do menu “*Operations*” ou do painel “*iMPACT Processes*”, selecionando-se a opção “*Generate File...*”. Este processo é rápido e resulta em um arquivo MCS gerado na pasta destino definida no passo da figura 3.2.

### 3.4 Resultados

Logo após a programação e após cada ciclo de energia (*power cycle*), o programa embarcado envia dados da sua execução através da porta UART para o computador, gerando a saída mostrada na figura 3.4. Pode-se observar que o programa, que quando compilado possuiu 104 KB, foi executado perfeitamente.

Note que a programação da FPGA através do iMPACT demorou sempre entre 800 e 1200 segundos, ou seja, entre 13 e 20 minutos, independente da largura de banda utilizada da interface utilizada. O motivo para esta demora não é informado, mas suspeita-se que seja devido a velocidade de transmissão da porta JTAG ou devido a velocidade de escrita na memória SPI Flash.

### 3.5 Conclusão

O experimento realizado funcionou como esperado, tendo carregado as informações do computador para a memória Flash e, em tempo de execução, da memória Flash para a memória DDR3. Ainda foi possível interpretar o cabeçalho do arquivo binário, extraindo dele informações importantes para o correto carregamento das configurações. O processo de programação da memória Flash através do computador é bem demorado, chegando a levar 20 minutos, mas durante o *boot* do sistema, ele é bem rápido, demorando apenas alguns poucos segundos.

É relativamente complicado trabalhar com os periféricos e *drivers* que os acompanham devido a documentação escassa e a grande dispersão das informações. Este experimento só pode ser realizado devido a uma imensa pesquisa e compilação de guias de usuário, relatórios de aplicações, *datasheets*, exemplos de projetos para diversos tipos de kits de desenvolvimento e comentários em fóruns de discussão.

```
Carregando configuracoes...LfCr
Processando cabecalho de Blank em QSPI0B00000LfCr
#T - T: 02 byte(s), C: 0x00 09 <9 byte(s)>LfCr
#T - T: 09 byte(s), C: 0x0F F0 0F F0 0F F0 0F F0 00 LfCr
#T - T: 02 byte(s), C: 0x00 01 <1 byte(s)>LfCr
#T - T: 01 byte(s), C: 0x61 <'a'>LfCr
#T - T: 02 byte(s), C: 0x00 23 <35 byte(s)>LfCr
#T - T: 35 byte(s), C: Blank_routed.ncd;UserID=0xFFFFFFFFLfCr
#T - T: 01 byte(s), C: 0x62 <'b'>LfCr
#T - T: 02 byte(s), C: 0x00 0D <13 byte(s)>LfCr
#T - T: 13 byte(s), C: 7k325tffg900LfCr
#T - T: 01 byte(s), C: 0x63 <'c'>LfCr
#T - T: 02 byte(s), C: 0x00 0B <11 byte(s)>LfCr
#T - T: 11 byte(s), C: 2013/11/26LfCr
#T - T: 01 byte(s), C: 0x64 <'d'>LfCr
#T - T: 02 byte(s), C: 0x00 09 <9 byte(s)>Lf
#T - T: 09 byte(s), C: 08:54:11LfCr
#T - T: 01 byte(s), C: 0x65 <'e'>Lf
#T - T: 04 byte(s), C: 00 0A 16 C4 <661188 bytes>Lf
Tamanho do cabecalho: 98 bytesLf
Carregando a configuracao Blank <QSPI0B00000 -> DDR3EC0104000>... Terminado!LfCr
Processando cabecalho de Counter em QSPI0C00000Lf
#T - T: 02 byte(s), C: 0x00 09 <9 byte(s)>LfCr
#T - T: 09 byte(s), C: 0x0F F0 0F F0 0F F0 0F F0 00 LfCr
#T - T: 02 byte(s), C: 0x00 01 <1 byte(s)>LfCr
#T - T: 01 byte(s), C: 0x61 <'a'>LfCr
#T - T: 02 byte(s), C: 0x00 25 <37 byte(s)>LfCr
#T - T: 37 byte(s), C: Counter_routed.ncd;UserID=0xFFFFFFFFLfCr
#T - T: 01 byte(s), C: 0x62 <'b'>LfCr
#T - T: 02 byte(s), C: 0x00 0D <13 byte(s)>LfCr
#T - T: 13 byte(s), C: 7k325tffg900LfCr
#T - T: 01 byte(s), C: 0x63 <'c'>LfCr
#T - T: 02 byte(s), C: 0x00 0B <11 byte(s)>LfCr
#T - T: 11 byte(s), C: 2013/11/26LfCr
#T - T: 01 byte(s), C: 0x64 <'d'>LfCr
#T - T: 02 byte(s), C: 0x00 09 <9 byte(s)>Lf
#T - T: 09 byte(s), C: 09:02:09LfCr
#T - T: 01 byte(s), C: 0x65 <'e'>Lf
#T - T: 04 byte(s), C: 00 0A 16 C4 <661188 bytes>Lf
Tamanho do cabecalho: 100 bytesLf
Carregando a configuracao Counter <QSPI0C00000 -> DDR3EC0204000>... Terminado!LfCr
Processando cabecalho de FSM em QSPI0D00000Lf
#T - T: 02 byte(s), C: 0x00 09 <9 byte(s)>LfCr
#T - T: 09 byte(s), C: 0x0F F0 0F F0 0F F0 0F F0 00 LfCr
#T - T: 02 byte(s), C: 0x00 01 <1 byte(s)>LfCr
#T - T: 01 byte(s), C: 0x61 <'a'>LfCr
#T - T: 02 byte(s), C: 0x00 21 <33 byte(s)>LfCr
#T - T: 33 byte(s), C: FSM_routed.ncd;UserID=0xFFFFFFFFLfCr
#T - T: 01 byte(s), C: 0x62 <'b'>LfCr
#T - T: 02 byte(s), C: 0x00 0D <13 byte(s)>LfCr
#T - T: 13 byte(s), C: 7k325tffg900LfCr
#T - T: 01 byte(s), C: 0x63 <'c'>LfCr
#T - T: 02 byte(s), C: 0x00 0B <11 byte(s)>LfCr
#T - T: 11 byte(s), C: 2013/11/26LfCr
#T - T: 01 byte(s), C: 0x64 <'d'>LfCr
#T - T: 02 byte(s), C: 0x00 09 <9 byte(s)>LfCr
#T - T: 09 byte(s), C: 09:02:02LfCr
#T - T: 01 byte(s), C: 0x65 <'e'>Lf
#T - T: 04 byte(s), C: 00 0A 16 C4 <661188 bytes>Lf
Tamanho do cabecalho: 96 bytesLf
Carregando a configuracao FSM <QSPI0D00000 -> DDR3EC0304000>... Terminado!LfCr
Fim!LfCr
```

Figura 3.4: Resultado da execução do programa embarcado gravado na memória QSPI Flash.

Para o próximo experimento, acredita-se já ter todas as ferramentas necessárias para a implementação da autorreconfiguração propriamente dita. Sendo assim, utilizará-se o mesmo comportamento do experimento 1 para definir uma partição reconfigurável e microcontroladores bem similares aos dos experimentos 2 e 3 para controlar a reconfiguração.

## Capítulo 4

# Experimento 4 - Autoreconfiguração com *MicroBlaze* e DDR3

### 4.1 Definição do Teste

Espera-se neste experimento projetar e implementar um sistema baseado em MicroBlaze que se reconfigure de forma autônoma, ou seja, sem a necessidade nenhum comando externo adicional. Para isto, deve-se entender o funcionamento de alguns periféricos e seus *drivers*, como integrar tudo de forma satisfatória, e o fluxo de projeto que permita as funcionalidades desejadas.

O projeto em questão pretende utilizar um microcontrolador para acionar a troca de comportamentos de uma partição reconfigurável totalmente independente. Tal feito pode, em teoria, ser alcançado facilmente através do periférico AXI4 HWICAP, que acessa as portas de configuração ICAP e ICAPE2, descritas a seguir.

### 4.2 Estudo dos Requisitos

#### 4.2.1 ICAP e ICAPE2

As interfaces ICAP e ICAPE2 (*Internal Configuration Access Port*) são formas de se acessar, tanto para leitura quanto para escrita, a configuração interna do FPGA. Algumas das aplicações mais comuns que a utilizam incluem sistemas *MultiBoot* (???), sistemas frequentemente atualizados e sistemas críticos que necessitam de frequente verificação do correto funcionamento do sistema (???), e.g. sistemas embarcados em aplicações espaciais suscetíveis a radiações capazes de alterar o conteúdo dos elementos lógicos. Estas portas de configuração se utilizam de um protocolo idêntico ao da interface SelectMAP (???), que se utiliza de uma série de comandos para iniciar um dos vários procedimentos de programação. Felizmente utiliza-se o controlador AXI4 HWICAP (??), que permite o controle da ICAP e ICAPE2 através do MicroBlaze.

Estas interfaces, tanto a SelectMAP quanto as ICAP e ICAPE2, possuem uma largura de banda de 32 bits

e frequência máxima de operação de 100 MHz, permitindo uma taxa de transferência de até 3.2 Gbps (??). Esta taxa de transferência permite que configurações parciais como a do experimento 1 sejam programadas em apenas 1.65 milissegundos ( $611288 \text{ bytes} \cdot 8 \frac{\text{bits}}{\text{byte}} \div 3.200.000.000 \frac{\text{bits}}{\text{segundo}} = 0,00165 \text{ segundo}$ ).

A única diferença entre as interfaces ICAP e ICAPE2 é a ausência de um sinal de espera na segunda. A ICAPE2, apesar de ter um comportamento mais determinístico no que diz respeito aos tempos de escrita e leitura, não está disponível para alguns dispositivos. A série de FPGAs 7 possui suporte a ICAPE2.

#### 4.2.1.1 Inversão dos bytes

A SelectMAP necessita de uma inversão na ordem dos bits de cada byte do arquivo de configuração (??), incluindo a palavra de sincronia, mencionada na seção 3.2.1. Esta inversão só é aplicável no uso das interfaces Serial, SelectMAP, e por consequência ICAP e ICAPE2, e BPI (??). Note que alguns tipos de arquivos sintetizados, como o MCS e o HEX, já podem ter estes bytes invertidos não sendo necessário invertê-los novamente. Não foi encontrada uma explicação para esta inversão.

#### 4.2.2 Fluxo de Projeto

Este projeto começará no *Project Navigator*, onde se instanciarão os componentes importados do experimento 1. O arquivo referente ao MicroBlaze, gerando através de ferramentas de auxílio a construção deste tipo de componentes, também será adicionado. Note, porém, que o fluxo de projeto não pode ser determinado com precisão, como mostrado na figura 4.1. Cada um dos passos ISE, SDK e XPS necessitam de arquivos que são gerados em um dos outros, tornando este projeto de implementação indefinida.

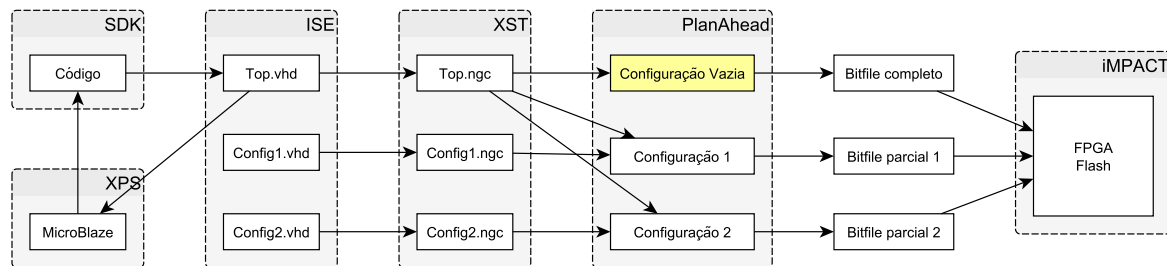


Figura 4.1: Fluxo de ferramentas para desenvolvimento do projeto.

Em busca por soluções, descobriu-se que é possível quebrar este ciclo através da construção manual de um arquivo de mapeamento de memória do tipo *Block RAM*. Sendo assim, o passo referente ao SDK não precisaria de informações do passo XPS. Apesar disso, este passo possui um grau de complexidade muito elevado, além do escopo deste projeto.

### 4.3 Resultados

Este experimento foi abordado de diversas formas diferentes, mas nenhuma levou em nenhum resultado palpável. Tentou-se implementar o microcontrolador com e sem memórias DDR3, visto que ela foi a



causadora de diversos erros, mas até sem ela observou-se a necessidade de construção manual de arquivos complexos, processo impraticável considerando o escopo deste projeto.

## **4.4 Conclusão**

Muitos erros, pouca documentação e pouco tempo impediram o atingimento completo dos objetivos propostos para este experimento. Apesar disto, muito conhecimento sobre o funcionamento do kit e suas limitações, em especial relacionadas aos seus recursos de relógio, foram adquiridos. Uma possibilidade de se contornar os problemas é o compartilhamento dos recursos de relógio entre o microprocessador e os componentes externos e a definição manual de restrições de localização de alguns elementos, de tal forma que eles atinjam os requisitos de temporização.

Uma forma possível de se simplificar este experimento, é a implementação do módulo reconfigurável como periférico do microcontrolador. Sendo assim, a reconfiguração seria iniciada pelo MicroBlaze para alterar o funcionamento deste periférico.

## Capítulo 5

# Experimento 5 - Autoreconfiguração com *MicroBlaze* e sem DDR3

Espera-se neste experimento projetar e implementar um sistema baseado em MicroBlaze que se reconfigure de forma autônoma, ou seja, sem a necessidade nenhum comando externo adicional. Para isto, deve-se entender o processo de criação de periféricos reconfiguráveis e seus *drivers*, o funcionamento do periférico AXI4 HWICAP e integrar tudo de forma satisfatória.

Ao contrário do experimento anterior, neste experimento não se utilizará o *Project Navigator*, eliminando assim várias complexidades introduzidas por ele.

### 5.1 Introdução Teórica

O desenvolvimento de sistemas que suportem a autorreconfiguração ainda não é totalmente suportado pelas ferramentas da Xilinx, como foi mostrado no experimento anterior. Algumas tarefas, como a inclusão de um executável em um processador (através do mapa de memória) instanciado em um sistema com módulos reconfiguráveis, precisam ser feitas a mão. Apesar disso, as ferramentas suportam algumas outras opções, como a inclusão direta de um módulo reconfigurável como periférico em um microcontrolador. Esta segunda opção será a abordada neste experimento.

#### 5.1.1 Periférico Reconfigurável

A ferramenta XPS permite a construção de periféricos com comportamentos completamente arbitrários. Estes periféricos podem ser utilizados para instanciar módulos reconfiguráveis (??). O PlanAhead deve ser usado em conjunto com o XPS para a definição das partições e configurações.

### 5.1.2 Fluxo do Projeto

O projeto se inicia desenvolvendo a lógica do periférico reconfigurável, processo similar ao passo ISE nos experimentos anteriores. Em seguida constrói-se o microprocessador utilizando este periférico. Com o processador pronto, pode-se programá-lo, mesmo que ainda não possa testá-lo. Por último, utiliza-se o PlanAhead para criar as diversas configurações e partições, sintetizando-as e implementando-as.

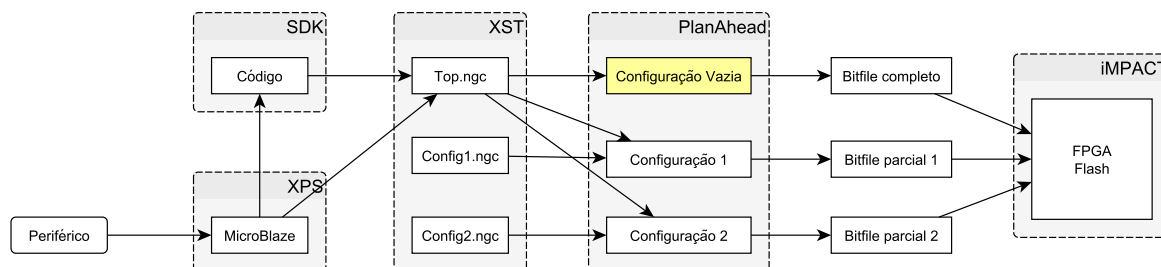


Figura 5.1: Fluxo de ferramentas para desenvolvimento do projeto.

## 5.2 Experimento

Este teste foi arquitetado para utilizar o máximo possível dos elementos já desenvolvidos neste trabalho. Utilizará-se os comportamentos reconfiguráveis desenvolvidos no experimento 1 para a modelagem do comportamento do periférico reconfigurável, o microprocessador desenvolvido no experimento 2 sem a memória DDR3 e o programa desenvolvido no experimento 3 considerando a ausência da memória DDR3, para se construir um sistema microprocessado que possa acionar a mudança de configurações de uma partição reconfigurável.

### 5.2.1 XPS

Ao se abrir o programa, deve-se construir um sistema utilizando o BSB, assim como no experimento 2. Quando questionado, seleciona-se os periféricos segundo a figura 5.2. Note que a interface UART tem *Baud Rate* de 115200 e o controlador da memória QSPI Flash tem modo de operação *Quad*.

Quando o programa termina de abrir, deve-se selecionar a opção “*Create or Import Peripheral...*” do menu “*Hardware*”. Pode-se deixar todas as opções como estão, com exceção da “*Software reset*” na página de “*IPIF (IP Interface) Services*”. Nota-se que foi criada a pasta “*pcores*” com uma pasta com nome igual ao periférico recém criado. Deve-se modificar os arquivos padrões criados para incluir a lógica do experimento 1. Um exemplo de como se modificar este arquivo está presente no anexo. Após modificado o periférico, deve-se reescanear a pasta de periféricos do usuário através da opção “*Rescan User Repositories*” do menu “*Project*”. Em seguida pode-se adicionar este periférico ao sistema.

Após a inclusão do periférico, as suas portas estarão toda desconectadas. Pode-se conectá-las através da aba “*Ports*”, expandindo a árvore referente a este periférico, clicando sobre a porta “*LEDs\_8Bits\_TRI\_O*” com o botão direito e selecionando a opção “*Make External*”. Com isto um pino, cria-se um pino conjunto

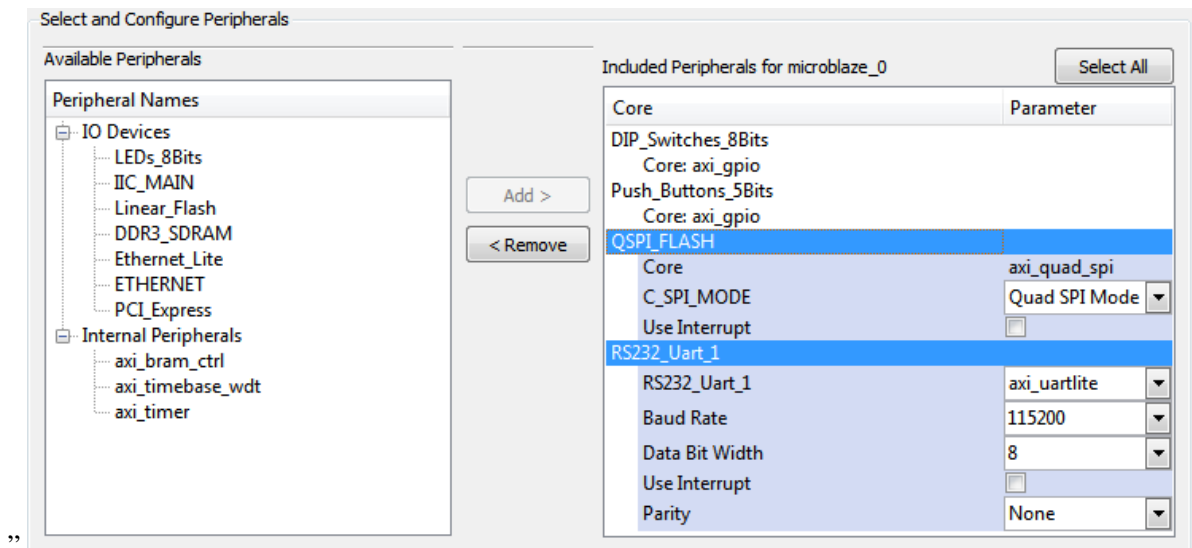


Figura 5.2: Fluxo de ferramentas para desenvolvimento do projeto.

de pinos de saída conforme necessitado pelo componente. Este pino pode ter seu nome modificado através da árvore “*External Ports*”, isto não é obrigatório. O pino “clk\_200MHz” será configurado mais a seguir.

É necessário também a inclusão do periférico “*FPGA Internal Configuraion Access Port*” (AXI\_HW-ICAP). Na adição, deve-se marcar a opção “*Instantiate STARTUP primitive in the HWICAP core*” para que o pino EOS (*End of Startup*) não seja mais necessário. Por causa disso, devido a presença de apenas uma primitiva STARTUP, também é necessário a mudança da opção “*Use Startup*” para falso nas configurações da memória QSPI Flash.

É preciso ainda executar o assistente de configuração de relógios, aberto através do menu “*Hardware*” selecionando-se a opção “*Launch Clock Wizard...*”. Na janela que se abre, deve ajustar todas as “*Clock Sources*” para “<AUTO>”, e a o relógio do periférico customizado “clk\_200MHz” para 200. Pode-se verificar a correta realização deste passo através da aba “*Ports*”.

Por último, deve-se ajustar os endereços e tamanhos das memórias. Para tal, deve-se ir a aba “*Addresses*” e alterar o tamanho da memória “QSPI\_FLASH” para 128M e seu endereço-base para 0x80000000. Pode-se surgir algum erro indicando a interseção de espaços de memória, fazendo-se necessário a mudança do endereço base de algum componente. Note que este passo não é obrigatório, apenas indicado.

### 5.2.1.1 Compilação e Exportação

Precisa-se agora gerar os arquivos NGC e BMM para que possam ser importados mais a frente pelo PlanAhead. Isto pode ser realizado clicando-se no botão “*Generate Netlist*” no painel “*Navigator*” ou no menu “*Hardware*”.

A exportação do projeto serve para que as informações deste projeto sejam transmitidas para o SDK. Ela pode ser feita pelo botão “*Export Design*” no painel “*Navigator*” ou pelo item “*Export Hardware Design to SDK...*” no menu “*Project*”. Note que a opção “*Include bistream and BMM file*”, presente na janela que se abre, deve ser desmarcado. A exportação do projeto para um arquivo binário acarretará em

erro, visto que o periférico reconfigurável possui componentes não definidos. Clica-se então no botão “*Export & Launch SDK*”.

Ao final deste processo de exportação, o SDK será aberto. Pode-se porém adiar este passo até que seja completamente necessário. A etapa do PlanAhead é mais urgente e importante, visto que permite a criação das diversas configurações.

### 5.2.2 PlanAhead

O procedimento do PlanAhead para este experimento é muito similar ao do experimento 1. Ao abrir o programa, cria-se um novo projeto, lembrando de se selecionar a opção de projeto pós-síntese (“*Post-synthesis Project*”) e de habilitar a reconfiguração parcial (“*Enable Partial Reconfiguration*”). Deve-se ainda incluir os arquivos NGC do projeto do microprocessador e suas restrições, presentes na presentes na pasta “*implementations*” e “*data*” do projeto do XPS, respectivamente.

Após a criação do projeto, deve-se abrir o o arquivo sintetizado através do botão “*Open Synthesized Design*”. Alguns avisos podem aparecer, mas pode-se ignorá-los. Este processo faz com que os componentes do projeto possam ser visualizados e manipulados.

A primeira coisa a se fazer com o projeto aberto é definir-se a partição reconfigurável que comportará o componente mutável. Para tal, expande-se o item referente ao periférico da lista no painel “*Netlist*” e clica-se com o botão direito sobre o componente identificado por “*USER\_LOGIC\_I*” e seleciona-se a opção “*Set Partition*”. Adiciona-se então uma partição reconfigurável com o *netlist* de alguma das configurações criadas no experimento 1. Deve-se verificar o experimento 1 para o procedimento para compilar os diferentes *netlists*.

Em seguida, aloca-se esta partição clicando-se o botão direito sobre mesmo item identificado por “*USER\_LOGIC\_I*” e selecionando-se o item “*Set Pblock Size*”. Com isso, a aba device deve ser posta em primeiro plano. Nela deve-se selecionar a partição desejada. Para este experimento, utilizou-se o bloco X1Y0, apesar de outros blocos poderem ser utilizados. Deve-se verificar o experimento 1 para técnicas de alocação de partições.

Antes de implementar a esta partição reconfigurável, deve-se habilitar a opção a função de resetar as células lógicas após reconfiguração. Isto pode ser feito selecionando-se a restrição física (*physical constrain*) e alterando seus atributos, acessados através das configurações desta restrição. Na aba atributos, dentro do painel Pblock Properties, clica-se no símbolo de adição localizado a esquerda. Da janela que aparece, seleciona-se a opção “*RESET\_AFTER\_RECONFIG*” e aceita-se a escolha. Note que o projeto deve ser salvo ou esta modificação será perdida.

Uma modificação que deve ser feita para evitar erros na geração dos arquivos binários é a inclusão dos pinos do periférico que acessam o dispositivo diretamente. Pode-se identificar estes pinos através da opção “*I/O Ports*” presente no menu “*Windows*”. Eles estão marcados em vermelho, significando que foram inicializados com valores aleatórios. Para corrigir este erro, deve-se ir ao painel “*Sources*”, árvore “*Constraints*” e procurar o arquivo identificado com “*(target)*”. Clica-se duas vezes sobre ele para abrí-lo. Deve-se adicionar os comandos apresentados abaixo, onde “*LEDs\_8Bits\_TRI\_O*” deve ser substituído pelo nome cuja marcação está em vermelho no painel “*I/O Ports*”.

```

NET "LEDs_8Bits_TRI_O[0]" LOC = "AB8" | IOSTANDARD = "LVCMOS15";
NET "LEDs_8Bits_TRI_O[1]" LOC = "AA8" | IOSTANDARD = "LVCMOS15";
NET "LEDs_8Bits_TRI_O[2]" LOC = "AC9" | IOSTANDARD = "LVCMOS15";
NET "LEDs_8Bits_TRI_O[3]" LOC = "AB9" | IOSTANDARD = "LVCMOS15";
NET "LEDs_8Bits_TRI_O[4]" LOC = "AE26" | IOSTANDARD = "LVCMOS25";
NET "LEDs_8Bits_TRI_O[5]" LOC = "G19" | IOSTANDARD = "LVCMOS25";
NET "LEDs_8Bits_TRI_O[6]" LOC = "E18" | IOSTANDARD = "LVCMOS25";
NET "LEDs_8Bits_TRI_O[7]" LOC = "F16" | IOSTANDARD = "LVCMOS25";

```

Os comandos apresentados acima apresentam restrições de alocação e de energia para estes pinos.

Precisa-se ainda, antes da implementação da configuração, informar ao programa que tal implementação deve incluir informações de memória do processador embarcado. Para isto, acessa-se o item “*Options*” do menu “*Flow*”. Na janela que se abre, escolhe-se a aba “*Strategies*”, onde muda-se o *droplist* “*Flow*” para ISE14. As opções de estratégia na árvore abaixo mudam, apresentando a opção “*ISE Defaults*”. Seleciona-se esta opção e clica-se no símbolo de adição localizado abaixo, de forma que uma nova estratégia seja criada nos moldes da selecionada. Deve-se então adicionar o comando “-bm ../../implementation/system.bmm” no campo “*More Options*”. Note que este endereço corresponde ao endereço relativo que liga a pasta “Projeto/PlanAhead/PlanAhead.runs/config\_1” à pasta “Projeto/implementation”. Precisa-se ainda indicar que a configuração a ser implementada deve usar a estratégia recém-definida, ao invés da padrão. Para isto, clica-se sobre a configuração no painel “*Design Runs*”, clica-se com o botão direito sobre a configuração em questão selecionando-se a opção “*Implementation Run Properties*”, vai-se até a aba “*Options*” e seleciona-se a nova estratégia. Deve-se lembrar de aplicar (*Apply*) esta mudança de estratégia ou ela será esquecida.

Pode-se agora implementar a configuração através do painel “*Design Runs*”. Clica-se na configuração desejada com o botão direito e seleciona-se “*Launch run...*”. Este processo é demorado, chegando a levar mais de 20 minutos em um computador com quatro núcleos e 12 GB de memória RAM. Ao final, deve-se selecionar a promoção da partição.

Note que o sistema desenvolvido neste exercício possui restrições de tempo que não puderam ser satisfeitas, mas isto não impede o correto funcionamento do sistema. A figura 5.3 mostra uma indicação de ocorrência deste erro.




Name	Part	Constraints	Strategy	Timing Score	Unrouted
 fsm (active)	xc7k325tffg900-2	constrs_1	ISE14_BM (ISE 14)	31839	0
 counter	xc7k325tffg900-2	constrs_1	ISE14_BM (ISE 14)		
 blank	xc7k325tffg900-2	constrs_1	ISE14_BM (ISE 14)		

Figura 5.3: Conteúdo da aba *Design Runs* após a implementação da primeira configuração e durante a implementação das seguintes. Note que o valor em vermelho indica que as restrições de tempo não foram atingidas, necessitando revisão.

Após a promoção da primeira configuração, outras podem ser adicionadas da mesma forma. Para tal, importa-se os arquivos NGC e cria-se novos *Design Runs*, conforme apresentado no experimento 1, atendendo apenas para a seleção da nova estratégia de compilação. Esta estratégia pode ser definida diretamente durante a criação dos novos *Design Runs*. Adicionou-se então um segundo módulo com comportamento

definido e um vazio.

Note que é interessante executar a verificação das configurações antes de qualquer outra coisa. Para isto, entra-se na aba “*Configurations*”, clica-se com o botão direito e seleciona-se a opção “*Verify Configuration...*”. Na janela que se abre, deve-se selecionar todas as configurações disponíveis antes de continuar. Nenhum erro deve ser encontrado.

O último passo a ser desenvolvido através da ferramenta PlanAhead é a geração de arquivos binários. Isto pode ser feito através da opção “*Generate Bitstream*” no menu lateral esquerdo. Este processo demorar alguns minutos e deve ser realizado para todas as configurações.

### 5.2.3 SDK

É recomendado abrir o SDK através do XPS, pelo comando de exportação apresentado anteriormente. Depois de aberto, o procedimento para programação é o mesmo dos experimentos passados.

Nesta etapa do experimento, desenvolveu-se, com o auxílio da API do *driver* para o periférico AXI HWICAP, um programa que carregasse informações da memória QSPI Flash e diretamente para o periférico controlador do ICAP. Este processo foi relativamente fácil, especialmente porque o ICAP não possui muitas formas de configuração. Sendo assim, o programa lê 32 bits da memória Flash e os envia diretamente para o periférico, que faz todo o controle necessário para que a reconfiguração seja realizada. A troca de configurações é controlada por um simples sistema de espera, que conta um número considerável de vezes depois que configuração anterior foi programada.

Várias formas podem ser utilizadas para a transferência deste programa para a FPGA para fins de teste, duas das quais foram exploradas. A primeira utiliza o comando “*Program FPGA*” do menu “*Xilinx Tools*” para programar a FPGA já com o programa embarcado. Para que este método funcione, deve-se selecionar o arquivo BMM utilizado no passo do PlanAhead, algum dos arquivos binários gerados por ele e o arquivo ELF gerado na compilação deste projeto. O segundo, mais fácil de se repetir mais vezes, utiliza o mesmo procedimento do anterior, a não ser a seleção do arquivo ELF. Neste caso, seleciona-se a opção “*bootloop*” como ELF para iniciar a memória *Block RAM*. Com isto, as opções “*Run As*” e “*Debug As*” apresentadas no experimento 2 e 3 são habilitadas, e a cada execução, apenas o novo arquivo ELF é enviado ao invés de toda a programação da FPGA.

Note que é interessante observar o tamanho dos arquivos binários gerados na etapa do PlanAhead. Estes arquivos influenciam diretamente nas posições acessadas pelo programa. É recomendado que estas posições sejam ser escolhidas de tal forma que sejam espaçadas uniformemente entre o final da configuração completa, que no caso deste dispositivo é a posição 0xAEA68C, e o final da memória Flash, que acontece em 0xFFFFFFFF. Caso as configurações não caibam na memória, pode-se alterar a região de alocação da partição reconfigurável, diminuindo assim a quantidade de elementos lógicos que precisariam ser programados dinamicamente, ou utilizar alguma outra forma de armazenamento, o que se mostrou muito complicado.

### 5.2.4 data2mem

Uma forma mais permanente de se inicializar um arquivo binário com um arquivo ELF é utilizando da ferramenta de linha de comando “*data2mem*”. Neste comando, indica-se qual configuração completa deve ser inicializada, qual é seu mapa de memória (arquivo BMM) e com que programa ela deve ser inicializada. Um exemplo de uso desta ferramenta está copiada abaixo.

```
data2mem -bm ..\implementation\system_bd.bmm  
-bt ..\PlanAhead\PlanAhead.runs\fsm\fsm.bit  
-bd ..\SDK\teste\Debug\teste.elf tag microblaze_0 -o b download.bit
```

Este comando foi usado enquanto na pasta PlanAhead, justificando os caminhos. Ao final, ele gerou o arquivo “download.bit”, composto pela configuração correspondente a máquina de estados desenvolvida no experimento 1 e o programa embarcado chamado teste. Note que este processo não precisa ser realizado nas configurações parciais, visto que elas não possuem um microcontrolador embarcado. De fato, apenas uma configuração completa, a que será carregada inicialmente, precisa passar por esta etapa, visto que na reconfiguração dinâmica, apenas a parte dinâmica precisa ser programada e ela não contém *Block RAMs*.

### 5.2.5 iMPACT

A última ferramenta necessária do fluxo de ferramentas é a iMPACT. Com ela, criar-se-á um arquivo MCS para inicialização da memória Flash e transmitir-se-á este arquivo para ela.

#### 5.2.5.1 Preparação da Memória Flash

Ao se abrir o iMPACT, deve-se aceitar a opção do sistema de criar e salvar um projeto automaticamente. Na janela que se abre, escolhe-se a opção “*Prepare a PROM File*”. Escolhe-se as opções conforme mostrado na figura 5.4. Deve-se atentar especialmente para o campo “*Add Non-Configuration Files*”, que deve estar setado como “*Yes*”, uma vez que ele é quem permite a inclusão das configurações parciais ao arquivo.

Em seguida, os arquivos são adicionados. O primeiro arquivo deve ser o “download.bit” gerado na etapa do “*data2mem*”. Após a inclusão deste, deve-se negar a inclusão de outros arquivos de configuração e aceitar a inclusão de arquivos de dados. Será solicitado o endereço-base de cada arquivo antes do seu carregamento, sendo importante tê-los em mãos. Ao final da inclusão de todas as configurações parciais, deve-se confirmar os endereços e gerar o arquivo MCS. Este processo pode ser iniciado pelo comando “*Generate File...*” do painel “*iMPACT Processes*” ou pelo comando de mesmo nome do menu “*Operations*”.

Com o arquivo MCS em mãos, pode-se abrir um novo projeto do iMPACT, selecionando-se desta vez a opção “*Configure devices using Boundary-Scan (JTAG)*”, com a opção de se conectar automaticamente ao cabo e identificar a cadeia de varredura habilitada. Note que a placa de desenvolvimento deve estar ligada e que não devem existir prorgamas que possam acessar as portas USB abertos. Não é necessário adicionar o arquivo de configuração completo da FPGA, visto que durante o processo de programação da memória Flash, ela também é inicializada. Sendo assim, pode-se rejeitar a primeira solicitação que for mostrada.



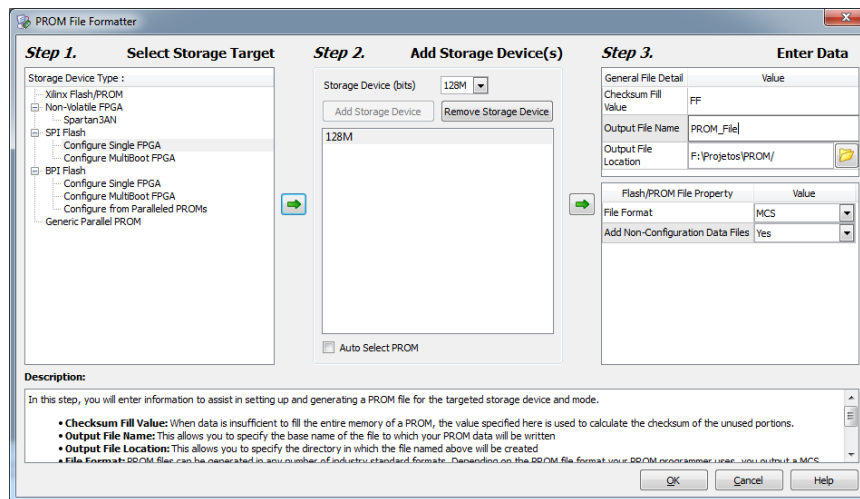


Figura 5.4: Janela para criação do arquivo de inicialização de memória Flash do tipo MCS.

Com o iMPACT esperando a próxima ação, deve-se clicar no ícone do dispositivo, mostrado na figura 5.5, com o botão direito e escolher a opção “Add SPI/BPI Flash...”. Na janela que se abre, procure o arquivo MCS criado a pouco. Depois de carregado, pode-se executar a programação do dispositivo clicando-se novamente com o botão direito sobre o símbolo da figura 5.5 e selecionando-se a opção “Program”.

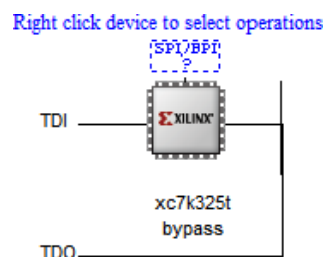


Figura 5.5: Imagem do ícone representativo do dispositivo a ser programado.

## 5.3 Resultados

Apresenta-se abaixo os resultados gerados em cada ferramenta utilizada.

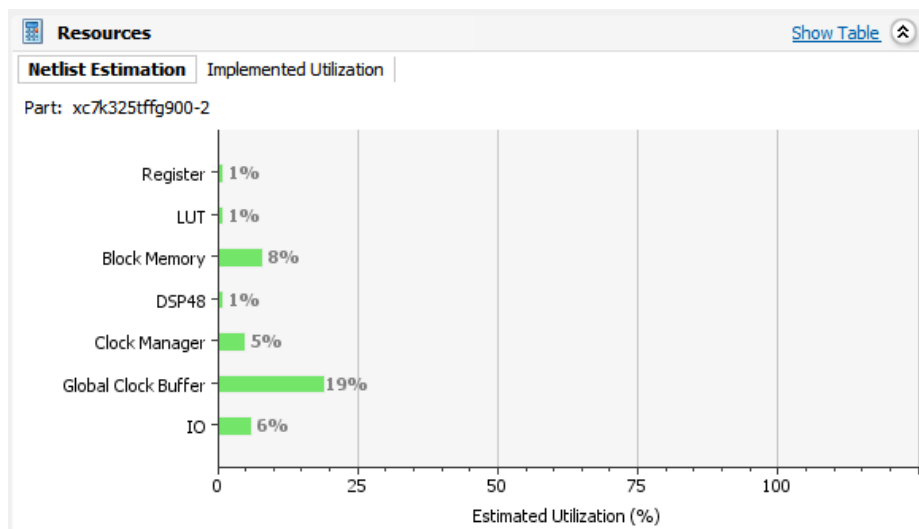
### 5.3.1 XPS

Esta ferramenta não gerou relatórios. Apesar disso, pode-se comentar que a execução da etapa referente a ela demorou 30 a 40 minutos, incluindo construção do sistema, modificações e síntese.

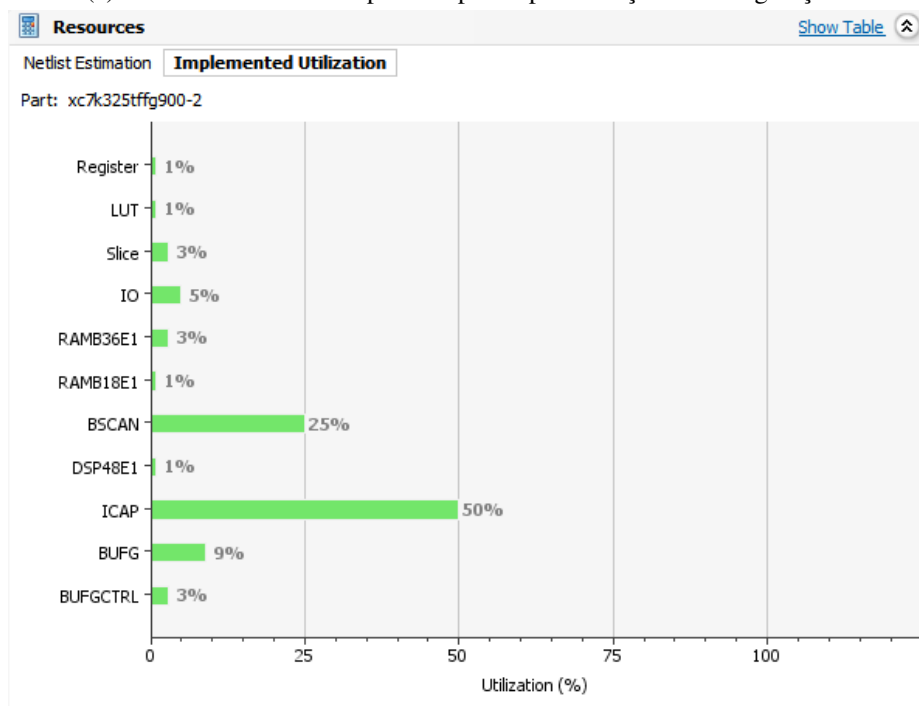
### 5.3.2 PlanAhead

Tendo sido a ferramenta que demandou mais tempo, devido a suas compilações extremamente complexas, o PlanAhead gerou relatórios de utilização do dispositivo. A figura 5.6a apresenta o relatório de

utilização com base apenas nas informações dos *netlists* e a figura 5.6b apresenta o relatório de utilização após a implementação da configurações.



(a) Relatório de uso do dispositivo pré-implementação de configurações.



(b) Relatório de uso do dispositivo pós-implementação de configurações.

Figura 5.6: Relatórios de uso do dispositivo.

Alguns pontos que valem ser comentados sobre o relatório dizem respeito a utilização do recurso “ICAP”. Este é um recurso escasso em FPGAs, existindo apenas 1 ou 2 por dispositivo. Neste caso, 1 ICAP de 2 foram utilizados, justificando a alta utilização deste.

A alocação de recursos no dispositivo ficou distribuída como mostrado na figura 5.7. Note que o MicroBlaze, que utiliza os elementos destacados em azul, não foi restrito a uma partição específica, ficando bastante espalhado no dispositivo. Este fato contribuiu para o não cumprimento de um dos requisitos de

tempo. Este requisito, porém, não afetou o funcionamento do sistema.

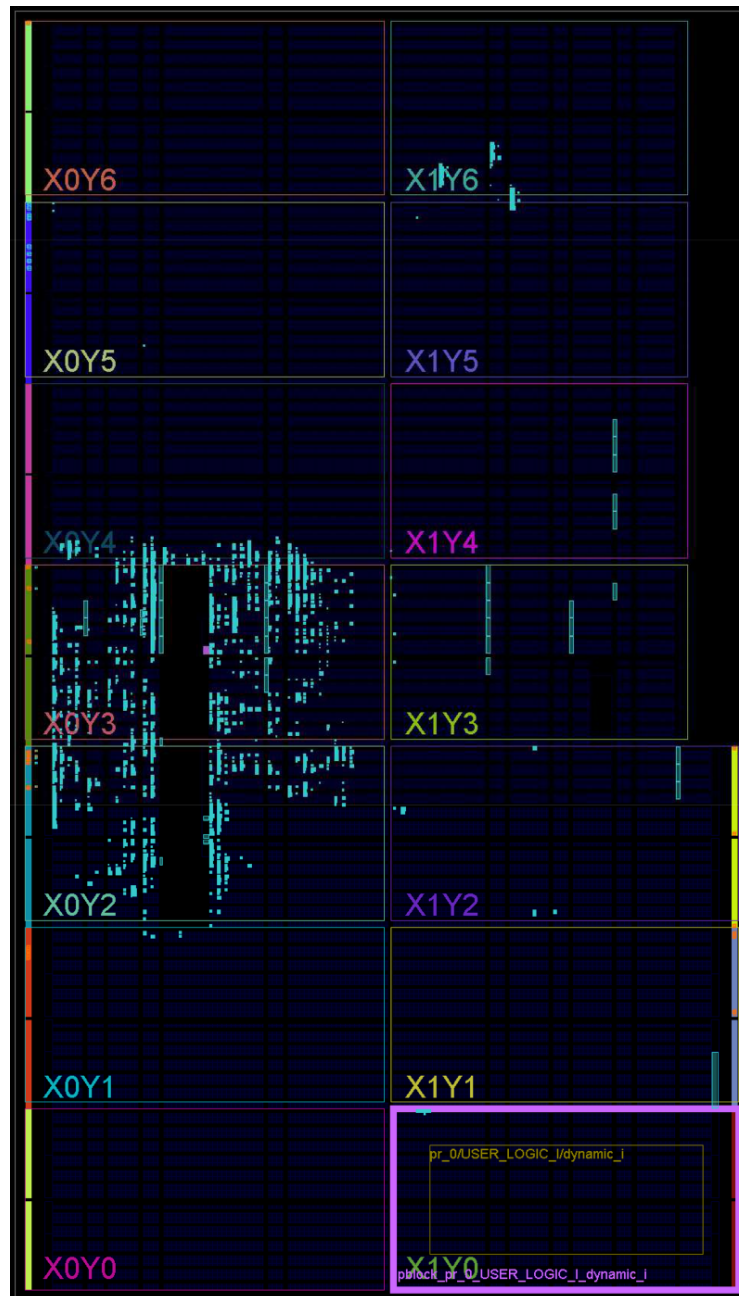


Figura 5.7: Resultado da implementação do sistema no dispositivo. Os pontos em azul claro representam os elementos lógicos utilizados. O retângulo vermelho no canto inferior direito delimita a partição reconfigurável.

### 5.3.3 SDK

Esta ferramenta não apresentou relatórios. Seu tempo de execução médio, considerando-se que o programa embarcado já tenha sido desenvolvido, é de 10 minutos.

### **5.3.4 data2mem**

Esta ferramenta não apresentou relatórios, tendo sido bem sucedida. Seu tempo de execução médio foi de 30 segundos.

### **5.3.5 iMPACT**

Esta ferramenta não apresentou relatórios, tendo sido bem sucedida. Seu tempo de execução médio foi de 5 minutos para a construção do arquivo MCS e de 15 minutos para a programação da memória QSPI Flash.

## **5.4 Conclusão**

Conclui-se este experimento verificando que ele foi bem sucedido. Conseguiu-se implementar a autorreconfiguração através de um microcontrolador MicroBlaze conforme esperado. Alguns dos elementos do fluxo de projeto, como o desenvolvimento de um periférico reconfigurável e o uso do mapa de memória (BMM) através do PlanAhead, estão muito mal explicados na literatura, aumentando os esforços necessários para o seu entendimento.

Acredita-se ter absorvido e transpassado os conhecimentos necessários para a construção de projetos mais complexos utilizando esta tecnologia. Apesar disto, nota-se uma clara limitação em projetos deste tipo, especialmente no que diz respeito a dificuldade de se utilizar a memória DDR3, reduzindo drasticamente o número de configurações possíveis de serem armazenadas em tempo de execução, o tamanho possível da memória local do processador (que com a DDR3 é virtualmente infinita), e reduzindo o desempenho geral do sistema.

## **Todo list**