

# Todo list

Terminar iMPACT . . . . .	8
Passo a passo do experimento 3 . . . . .	14
Explicar como fui para o próximo experimento . . . . .	14
Descrever o experimento 4 . . . . .	15
Explicar como fui para o próximo experimento . . . . .	15
Descrever o experimento 5 . . . . .	15
Explicar como fui para o próximo experimento . . . . .	15

# Capítulo 1

## Desenvolvimento

*Este capítulo trata da concepção dos experimentos realizados. Nele serão descritos com detalhes cada um dos experimentos, ficando a parte de análise reservada ao capítulo ??.*

### 1.1 Introdução

Devido ao caráter experimental e exploratório do objetivo proposto na seção ??, decidiu-se dividir o projeto em vários experimentos menores. Desta forma, além de garantir algum material mesmo que tudo dê errado, consegue-se simplificar o processo de pesquisa e desenvolvimento através dos pequenos passos e análises frequentes.

Como o objetivo final do projeto é a familiarização com as ferramentas e processos envolvidos na autoreconfiguração, decidiu-se começar estudando os elementos necessários para se realizar a reconfiguração dinâmica. O passo seguinte mais lógico é o de estudar como funciona as memórias dos sistema e de que jeito elas seriam melhor utilizadas. O último passo seria entender como funciona a autoreconfiguração em baixo nível, ou seja, como os dados devem ser entregues aos devidos componentes para que ela aconteça. Para cada um destes experimentos foi proposto um teste que validasse o completo entendimento do mesmo.



Figura 1.1: Foto ilustrativa do kit de desenvolvimento Kintex-7 KC705 extraída do site da Xilinx.

Para o desenvolvimento desse projeto, escolheu-se utilizar o kit de desenvolvimento da Xilinx® chamado Kintex-7 KC705. O único critério utilizado foi a disponibilidade dos equipamentos no início do projeto e a capacidade do dispositivo de realizar a reconfiguração parcial dinâmica. Este kit possui FPGA modelo XC7K325T-2FFG900C, leitor de cartão de memória, conector PCIe®, memória DDR3, visor de 7-segmentos e porta ethernet, dentre outros.

Escolheu-se ainda, de forma arbitrária, o uso da linguagem VHDL para a descrição de *hardware* ao invés da Verilog.

## 1.2 Experimento 1 - Reconfiguração Dinâmica

De forma a dar validade a todo o projeto, foi preciso desenvolver um experimento para se entender o processo de desenvolvimento de sistemas reconfiguráveis dinamicamente e algumas peculiaridades do kit de desenvolvimento.

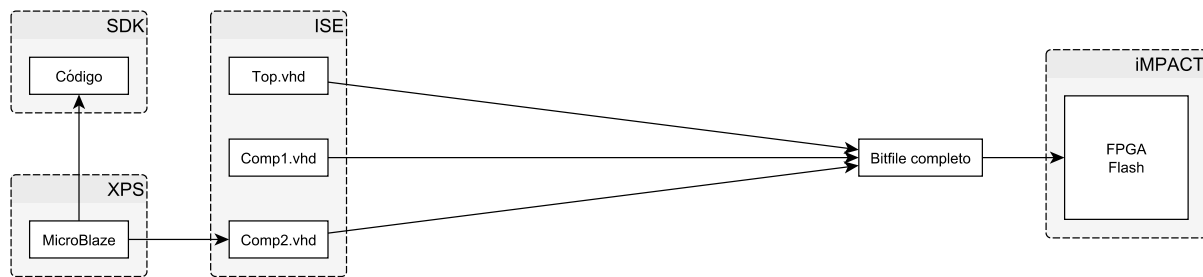
### 1.2.1 Fluxo de Ferramentas

A primeira coisa 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. 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 mostrado na figura 1.2b, além das ferramentas do fluxo tradicional, faz-se necessário o uso da ferramenta XST para a síntese do *netlist* 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.

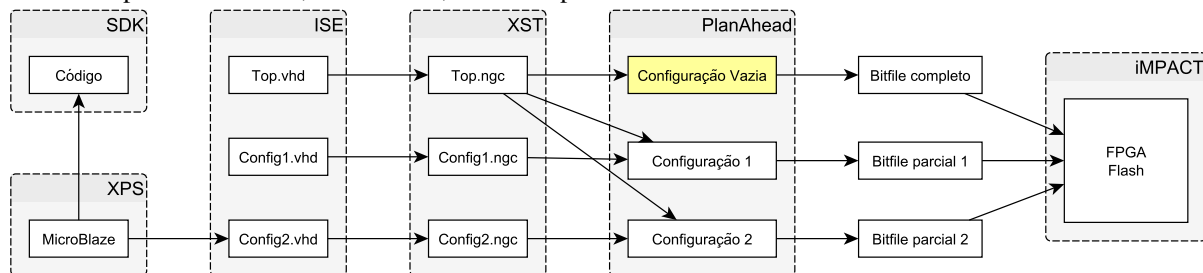
Reconfiguração parcial pede uma síntese utilizando o método “de baixo para cima” (*bottom-up*), mas uma implementação “de cima para baixo” (*top-down*) (??), ou seja, a implementação acontece construindo primeiro a interface com o sistema e depois os componentes auxiliares, mas a síntese precisa ser realizada no sentido oposto. Esta implementação é equivalente a se construir diversos projetos tradicionais com alguma lógica em comum, onde a síntese deve garantir que esta lógica em comum seja implementada da mesma forma para as diferentes configurações (??).

### 1.2.2 Peculiaridades

O kit de desenvolvimento utilizada apresenta algumas peculiaridades com relação aos kits comuns. A seguir serão apresentadas algumas destas peculiaridades.



(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.

### 1.2.2.1 Relógios

Diferentemente das FPGAs comuns, a que está presente neste kit contém um relógio diferencial, ou seja, dois sinais compoem tal relógio. A razão para tal é a presença de circuitos sensíveis a interferência, 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. Por motivos de simplicidade, utilizou-se o SYSCLK. Para poder se trabalhar com o sinal diferencial, construiu-se, utilizando as ferramentas do ISE, um componente para tratamento do sinal de relógio. Este componente recebe o sinal diferencial, reduz sua frequência para 20 MHz, que corresponde ao menor valor suportado pelas PLLs da placa, e emite um sinal convencional.

### 1.2.3 Teste

Para se entender mais a fundo o fluxo de projeto, nada melhor que construir um projeto. Para isso, implementou-se o sistema esquematizado na figura 1.3. Este sistema contém o “Top” para interfaceamento com a FPGA, o “Clocks” para tratamento do sinal de relógio, a “Static”, que possui um 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.



Figura 1.3: 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 e os em azul são dinâmicos.

**Estrutura de Pastas** A questão da organização do projeto em pastas bem específicas é sempre bem mencionado na literatura (?????). Os manuais recomendam a seguinte estrutura de pastas.

```

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

```

Esta estrutura de pastas foi obedecida por ajudar a manter o ambiente de desenvolvimento limpo.

### 1.2.3.1 Comportamento

Como explicado anteriormente, 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 1.3. 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.

O componente “Static” possui uma entrada para um relógio pulsando a 2 Hz e uma saída para um LED. Seu comportamento apenas faz com que o LED pisque a uma frequência de 2 Hz, o que permite observar seu funcionamento durante a reconfiguração do componente “Dynamic”.

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 e uma palavra de 4 bits de saída. A frequência de operação deste componente foi escolhida para ser a metade da frequência da “Static” para poder ser visualmente comprovado 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.

### 1.2.3.2 Síntese

Com o comportamento do projeto definido, o próximo passo segundo o fluxo de ferramentas é a síntese. Este passo é necessário uma vez que o próximo passo, referente ao PlanAhead, não aceita como entrada códigos-fonte. Os códigos-fonte precisam passar por uma etapa de síntese separada para poderem ser importados no PlanAhead. Este passo está ilustrado no fluxo de ferramentas da figura 1.2b como XST.

O comando XST 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"
```

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 (?????). A única coisa que se faz neste *script* é a construção dinâmica dos comandos com base em listas de arquivos pré-informados.

### 1.2.3.3 PlanAhead

Com os arquivos sintetizados, pode-se começar a etapa referente ao PlanAhead. Nela, importaremos os arquivos da etapa anterior, criaremos a partição reconfigurável, mapearemos esta partição no dispositivo, criaremos configurações alternativas, promoveremos tais configurações e geraremos 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.

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 1.4, onde se lê “Create New Project”. Na janela que aparece, indicamos o nome do projeto e seu caminho, lembrando que foi criada uma pasta anteriormente especificamente para conter este projeto. Indicamos também que o projeto é do tipo “*Post-synthesis Project*” e que desejamos habilitar a reconfiguração parcial, indicamos quais *netlists* compõe o comportamento do projeto e qual corresponde ao arquivo principal, qual é o arquivo de restrições (*constraints*) e qual é o modelo da FPGA.

Após a criação do projeto, carregam-se os arquivos sintetizados abrindo “*Open Synthesized Design*”, mostrado na figura 1.5. Duas janelas de avisos aparecem, informando que existe uma partição não implementada e aviso sobre alguns pinos.

Pode-se agora definir a partição que armazenará o componente dinâmico. Isso é feito através do painel “*Netlist*”, selecionando a opção “*Set Partition*” do menu que aparece ao se clicar em “dynamic\_i” com o botão direito. Na janela que aparece, selecionamos a opção referente à partição reconfigurável, nomeamos o módulo reconfigurável de acordo com o componente que será carregado e indicamos o arquivo 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 suscetível a falhas, permitindo que erros e falhas na definição da região a seguir sejam identificados mais cedo.



Figura 1.4: Imagem do PlanAhead logo que aberto.



Figura 1.5: Imagem do menu “Open Synthesized Design”.

Precisa-se definir também uma região para a partição recém definida. Isto pode ser feito pelo mesmo painel “Netlist”, selecionando 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 subsequentes, seleciona-se a opção de recomençar todo o processo do zero, evitando erros gerados nos estágios iniciais. Uma região possível, que foi utilizada nesse projeto, foi a que contém as células (*slice*) de X80Y244 a X139Y205. Os nomes das células ficam visíveis através da ampliação do dispositivo.

Uma vez terminado o “Design Run”, 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. Um módulo vazio pode ser criado usando esta mesma opção, mas selecionando a opção que indica a inclusão de uma *blackbox*



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. Em seguida, as duas novas configurações serão criadas através de “*Design Runs*”. É necessário promover neste momento a primeira configuração implementada. O processo de promoção será comentado a seguir. Note que os avisos que aparecerem podem ser ignorados.

Ao final dos “*Design Runs*”, recomenda-se 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.

Devemos agora promover as 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.

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*. Esta etapa precisa ser realizada com cada uma das configurações realizadas, ou alguma delas não terá seus *bitfiles* gerados. 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 *bitfile* dentro de cada pasta, um maior, que contém a configuração completa, e outro menor que possui a configuração parcial.

#### 1.2.3.4 iMPACT

Com os *bitfiles* em mãos, usa-se-os para programar a FPGA através auxílio da ferramenta iMPACT.

A primeira coisa a se fazer é inicializar a cadeia de dispositivos, fazendo com que um símbolo representando a FPGA apareça na tela.

Terminar iMPACT

#### 1.2.4 Possíveis Erros

**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.

**Erros no 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.

**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.

**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.

### 1.2.5 Conclusão

O processo de desenvolvimento de partições reconfiguráveis e sua programação foi explorado com sucesso. Observou-se os pontos críticos do desenvolvimento e o fluxo mais adequado para a construção deste tipo de projeto.

## 1.3 Experimento 2 - Teste de Memórias

Seguindo o raciocínio apresentado no início do 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 *bitfiles* de configurações parciais em uma memória embarcada, removendo a necessidade do computador para tal.

### 1.3.1 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.

---

<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>>

### 1.3.1.1 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>2</sup>.

### 1.3.1.2 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.

### 1.3.1.3 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.

### 1.3.1.4 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.

### 1.3.1.5 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.

**CompactFlash e o System ACE** *SystemAce* 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.

---

<sup>2</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>>

### 1.3.1.6 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*.

### 1.3.2 Teste

Notou-se no primeiro experimento que os *bitfiles* parciais gerados possuem 645 KB, totalizando 1935 KB, ou 1,89 MB. Com isso o uso das memórias que se utilizam de recursos internos da FPGA fica comprometido. Caso se escolha esta opção, apenas os *bitfiles* parciais poderiam ser armazenados, necessitando da ajuda do computador para a programação inicial. Este tipo de memória é mais útil como memória de dados e de programa em microcontroladores embarcados na FPGA, devido a sua altíssima velocidade de acesso e capacidade de acesso de vários canais em paralelo.

Observa-se que a interface de reprogramação dinâmica, ICAPE2, 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 apresentadas acima acabariam por ser o gargalo do tempo da programação. Uma solução 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. A forma mais fácil de implementar tal funcionalidade é através do uso de um microcontrolador MicroBlaze funcionando como *bootloader*.

Antes de implementar o *bootloader*, precisamos entender como funcionam as memórias do escolhidas. Este teste, então, tem por objetivo entender o fluxo de projeto para o uso do MicroBlaze, o funcionamento deste microcontrolador e as formas corretas de se acessar as memórias do sistema.

#### 1.3.2.1 Fluxo de Projeto

O procedimento para a construção de um projeto com MicroBlaze segue o fluxo descrito na figura 1.2a, no que diz respeito ao XPS e ao SDK. A figura 1.6 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.

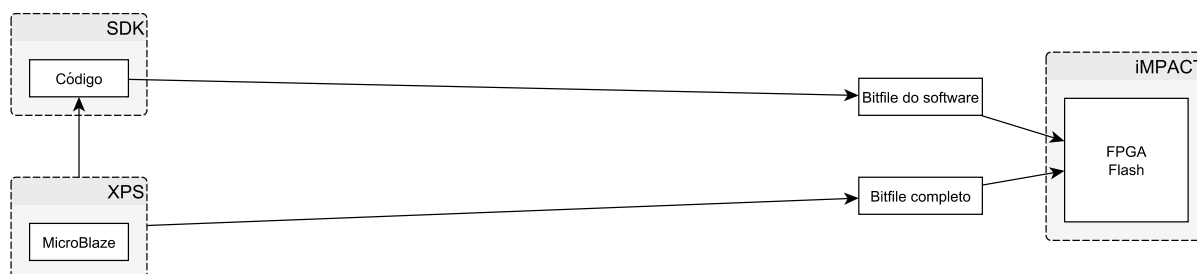


Figura 1.6: Foto ilustrativa do fluxo de ferramentas para o desenvolvimento de sistemas com MicroBlaze, extraído de (??).

#### 1.3.2.2 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 1.7. Todas as outras configurações, tais como o uso de *Big Endian* ou *Little Endian*, por exemplo, são opcionais (??).

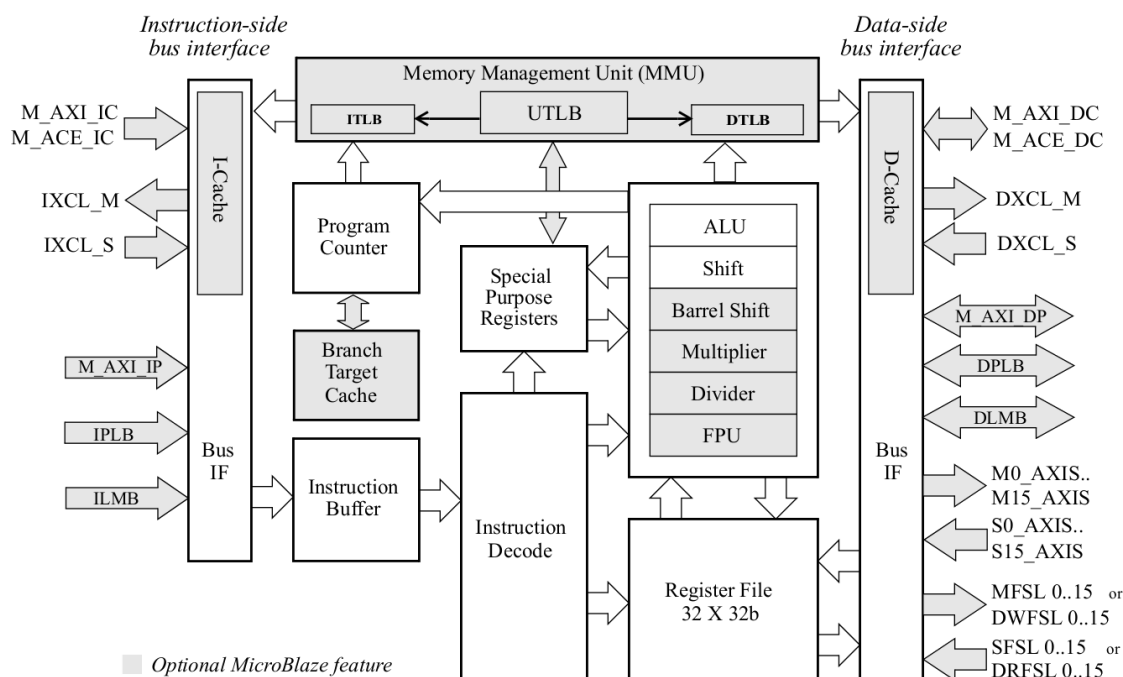


Figura 1.7: Diagrama de blocos do MicroBlaze.

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 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.

### 1.3.2.3 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 figura 1.8. Na janela que aparece, escolheu-se a placa de desenvolvimento que se escolheu utilizar e a opção de um só processador no sistema, para simplificar o projeto. Dando prosseguimento, escolheu-se os periféricos desejados segundo a figura 1.9 e o tamanho das memórias local, de intrução e de dados, quíça 64 KB.

Antes de concluir o a construção do sistema, precisa-se ajustar os tamanhos das memórias e seus

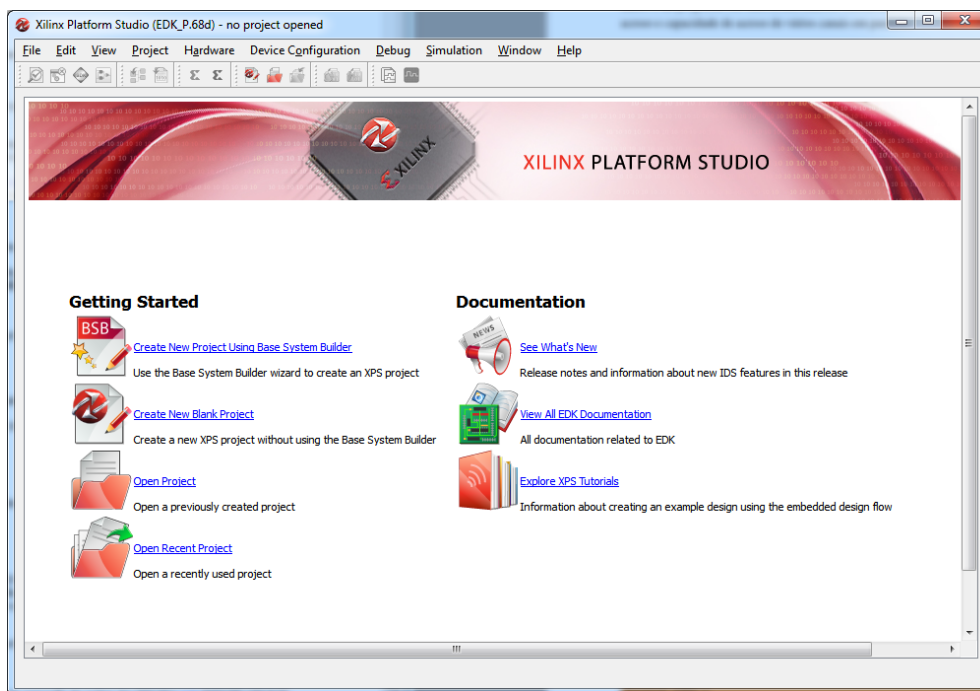


Figura 1.8: Imagem do XPS logo que aberto.

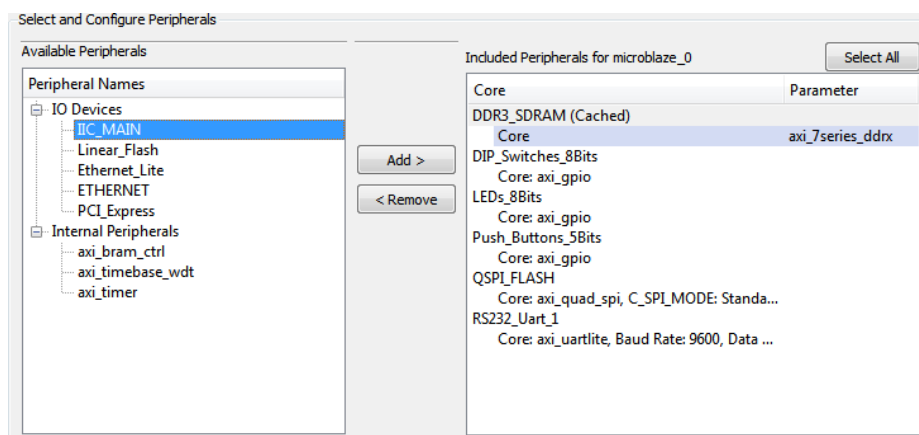


Figura 1.9: Escolha dos periféricos no BSB do XPS.

Instance	Base Name	Base Address	High Address	Size	Bus Interface(s)	Bus Name	Lock
microblaze_0's Address Map							
microblaze_0_d_bram_ctrl	C_BASEADDR	0x00000000	0x0000FFFF	64K	SLMB	microblaze_0_dl...	<input type="checkbox"/>
microblaze_0_i_bram_ctrl	C_BASEADDR	0x00000000	0x0000FFFF	64K	SLMB	microblaze_0_il...	<input type="checkbox"/>
Push_Buttons_5Bits	C_BASEADDR	0x40000000	0x4000FFFF	64K	S_AXI	axi4lite_0	<input type="checkbox"/>
LEDs_8Bits	C_BASEADDR	0x40020000	0x4002FFFF	64K	S_AXI	axi4lite_0	<input type="checkbox"/>
DIP_Switches_8Bits	C_BASEADDR	0x40040000	0x4004FFFF	64K	S_AXI	axi4lite_0	<input type="checkbox"/>
RS232_Uart_1	C_BASEADDR	0x40600000	0x4060FFFF	64K	S_AXI	axi4lite_0	<input type="checkbox"/>
debug_module	C_BASEADDR	0x41400000	0x4140FFFF	64K	S_AXI	axi4lite_0	<input type="checkbox"/>
QSPI_FLASH	C_BASEADDR	0x70000000	0x77FFFFFF	128M	S_AXI	axi4lite_0	<input type="checkbox"/>
DDR3_SDRAM	C_S_AXI_BASEA...	0xC0000000	0xFFFFFFFF	1G	S_AXI	axi4_0	<input type="checkbox"/>

Figura 1.10: Aba Addresses do System Assembly View indicando os ajustes dos endereços e tamanhos das memórias.

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 0x70000000, o tamanho da memória DDR3 para 1G e seu endereço base para 0xC0000000, conforme a figura 1.10. Note que o endereço da memória SPI Flash pode ser alterado, tendo como única restrição de ter os 28 bits menos significativos iguais a zero. O endereço base da memória DDR3, porém, não pode ser alterado visto que o endereço de 0xC0000000 a 0xC7FFFFFF é utilizado como memória *cache* pela interface AXI4.

O projeto pode ser sintetizado através do botão “*Generate Netlist*” localizado no menu a 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*”. Após a implementação do arquivo binário, a ferramenta SDK será aberta.

#### 1.3.2.4 SDK

Quando a janela do SDK aparecer, ela perguntará onde se quer colocar o *workspace*. Seleciona-se a pasta SDK, criada dentro da pasta do projeto do sistema durante sua exportação.

### 1.3.3 Conclusão

## 1.4 Experimento 3 - Teste do *Bootloader*

O *bootloader*, como mencionado acima, é um sistema que carrega as informações de uma memória lenta, a SPI Flash foi escolhida, para uma memória rápida, ou seja, a memória DDR3. Usou-se um micro-controlador MicroBlaze com controladores para a memória SPI Flash, DDR3 e GPIO, que dá acesso aos pinos da placa. Note que é interessante que este subsistema possa sinalizar para os demais que o carregamento foi completado e informar algumas informações dos elementos transferidos, tais como tamanhos e posições iniciais.

#### 1.4.1 Teste

O teste deste experimento compreende a construção de um subsistema microprocessado que carregue os arquivos binários da memória SPI Flash para a memória DDR3 e indique para o resto do sistema o final do processo. Este subsistema, que aqui é mestre, será escravo quando incluído no sistema principal. Usará-se os programas XPS e SDK, que fazem parte do Embedded Development Kit (EDK).

Passo a passo do experimento 3

#### 1.4.2 Conclusão

Explicar como foi para o próximo experimento

## 1.5 Experimento 4 - Teste da Autoreconfiguração com *Bootloader* Dedicado

Descrever o experimento 4

Explicar como fui para o próximo experimento

## 1.6 Experimento 5 - Teste da Autoreconfiguração

Descrever o experimento 5

Explicar como fui para o próximo experimento