Todo list

Conteúdo

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 extraida 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.

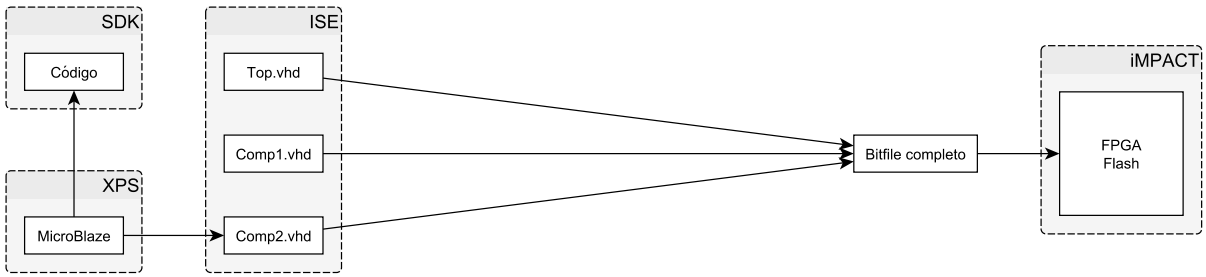
1.2.1 Fluxo de Ferramentas

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.

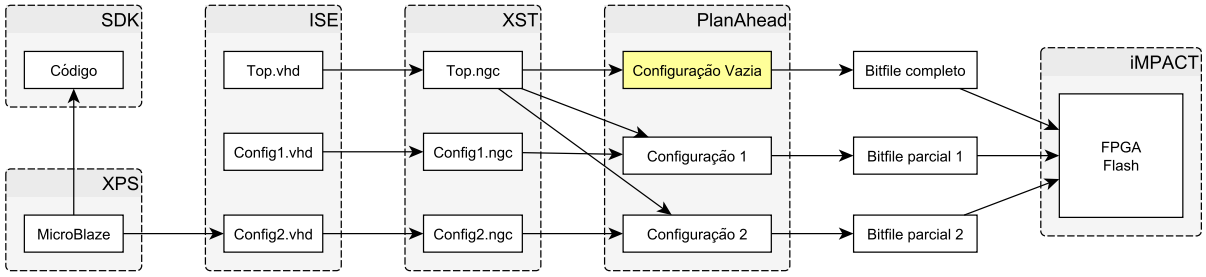
Reconfiguração parcial pede uma síntese utilizando o método “de baixo para cima” (*bottom-up*), mas uma implementação “de cima para baixo” (*top-down*) (**??**), ou seja, a implementação acontece construindo primeiro a interface com o sistema e depois os componentes auxiliares, mas a síntese precisa 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^2C 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 dinâmicamente.

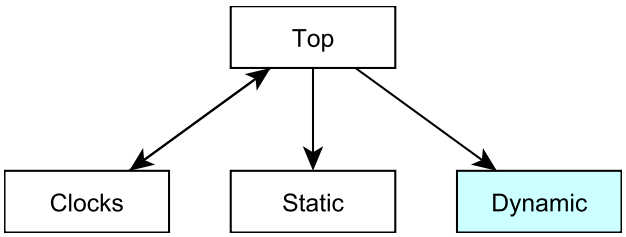


Figura 1.3: Foto ilustrativa do sistema desenvolvido para o teste de validação do experimento 1. Ele é composto por uma parte estática e uma parte dinâmica. O elementos em branco são estáticos e os em azul são dinâmicos.

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 saida, 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 síntetizados 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 cógigo-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 síntetisados, 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, promoremos 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 (*constrains*) 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 sucetível a falhas, permitindo que erros e falhas na definição da região a seguir sejam identificados mais cedo.

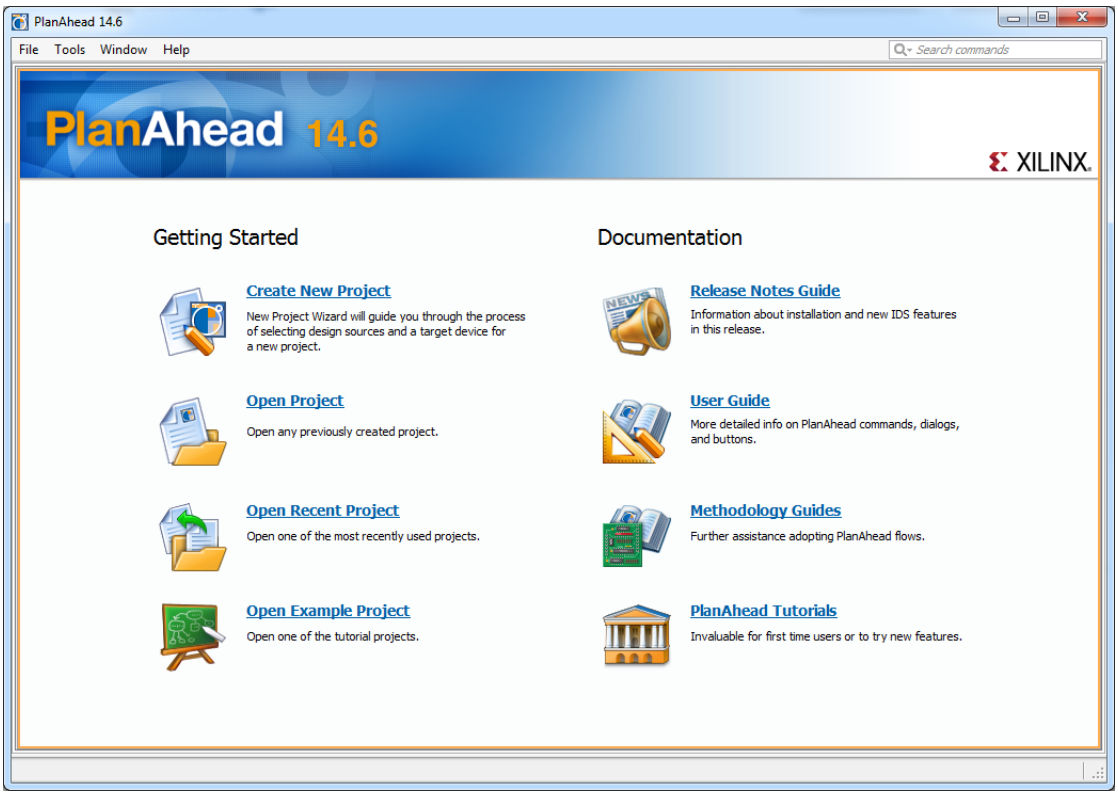


Figura 1.4: Imagem do PlanAhead logo que aberto.

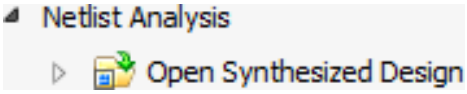


Figura 1.5: Imagem do menu "*Open Synthesized Design*".

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 recomeçar 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 auxilio da ferramenta iMPACT. A primeira coisa a se fazer após abrir o programa é permitir que o sistema automaticamente crie um projeto. 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 sitema 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*".

1.2.4 Possíveis Erros

Erros no código-fonte Este é um dos erros mais comuns. A melhor forma de previní-lo é através da construção dos diversos comportamentos/configurações individuais utilizando o ISE. Para acelerar o processo, realiza-se apenas a síntese.

Erros na alocação de partições Um erro bastante comum que aparece no PlanAhead é o de erro de alocação[1]. Existem duas possíveis formas de corrigí-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 incluidas 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 simplemente 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.

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.

1.2.5 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] *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 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 Mémoria

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

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 [2].

1.3.1.2 Memória *Distributed RAM*

A memória *Distributed RAM* é construida utilizando-se das LUTs disponíveis nas célula 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

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

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.

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 típo 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.

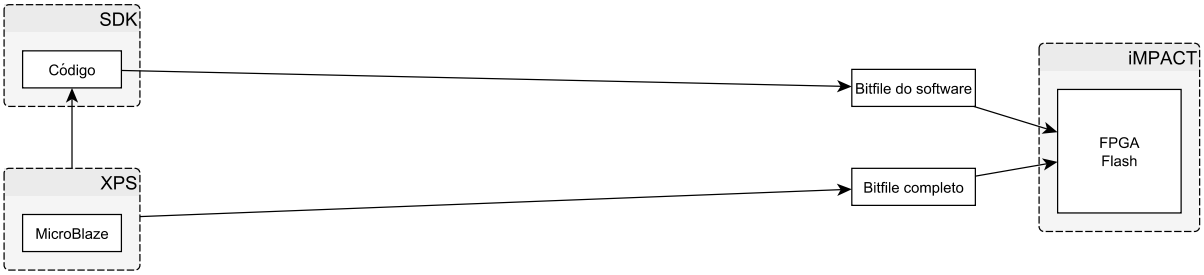


Figura 1.6: Foto ilustrativa do fluxo de ferramentas para o desenvolvimento de sistemas com MicroBlaze, extraído de (**??**).

1.3.2.2 MicroBlaze

O MicroBlaze é um microprocessador otimizado para implementação em FPGAs da Xilinx (**??**). Ele possui 32 registrados genéricos de 32 bits, instruções de 32 bits e endereços de 32 bits. Seu *pipeline* possui 3 ou 5 estágios e é construido 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 (**??**).

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 compostam 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 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 1.8 e o tamanho das memórias local, de intrução e de dados, quiça 64 KB. Modificou-se ainda, como pode-se ver na figura 1.8, o *Baud Rate* da interface UART para 115200 bits/s.

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

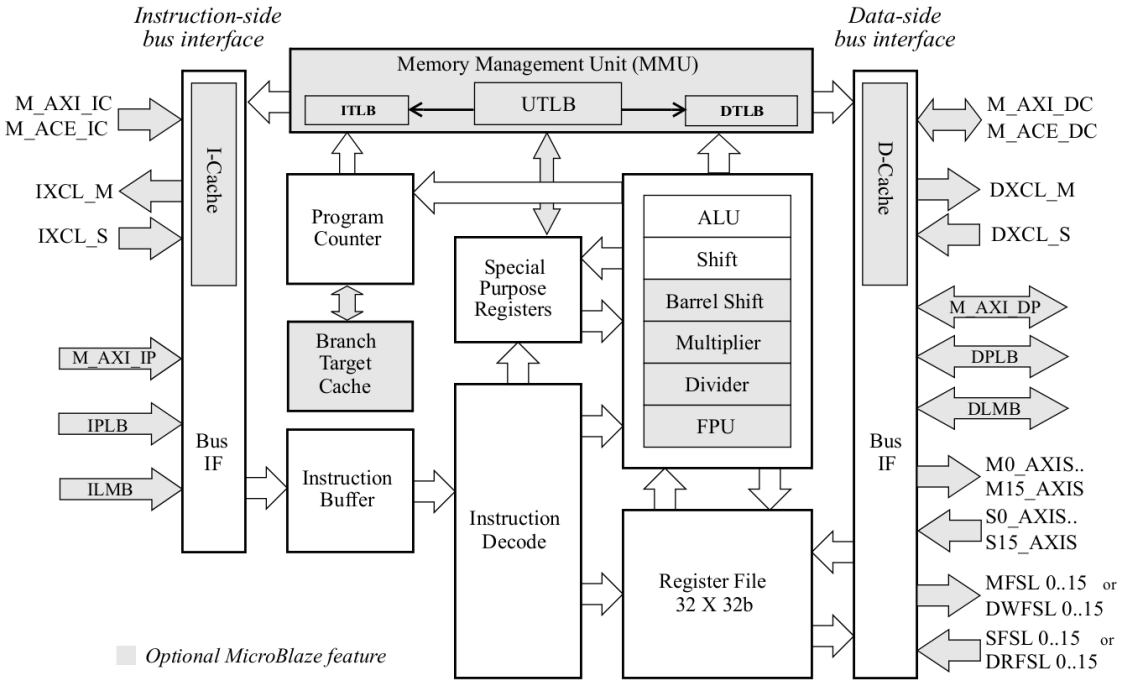


Figura 1.7: Diagrama de blocos do MicroBlaze.

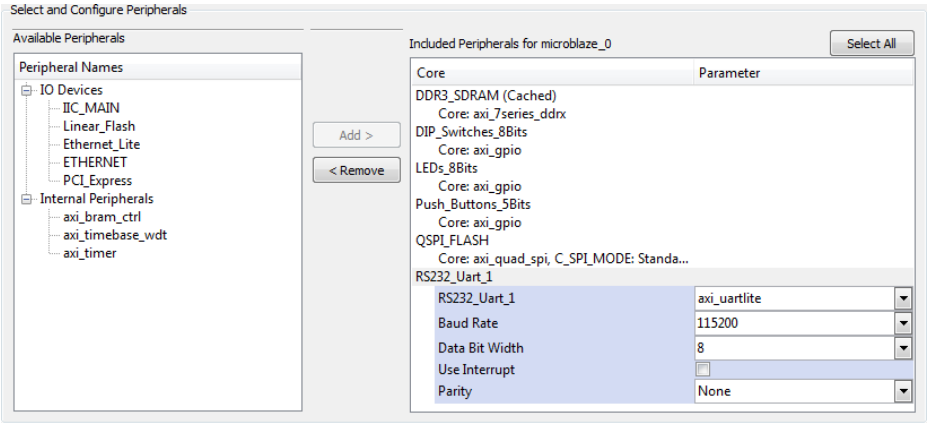


Figura 1.8: Escolha dos periféricos no BSB do XPS.

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 1.9. 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 agora 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 da primeira coluna para 0xc0000000 e 0xffffffff e os da segunda para 0x80000000 e 0xffffffff, indicando que a memória

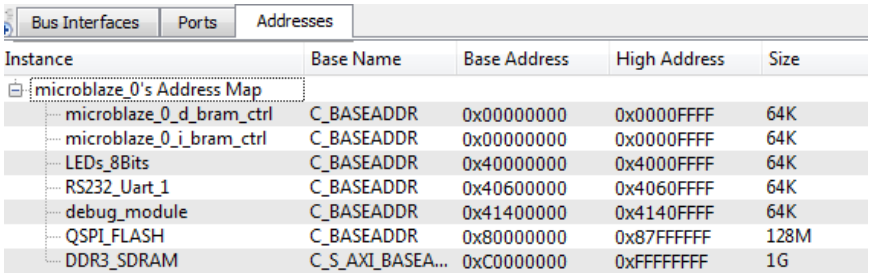


Figura 1.9: Aba *Addresses* do *System Assembly View* indicando os ajustes dos endereços e tamanhos das memórias.

de instruções pode acessar apenas os endereços da memória DDR3, mas que a memória de dados pode acessar toda a faixa de endereços entre 0x80000000 e 0xffffffff, que engloba as memórias SPI Flash e 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 memórias, que correspondem às posições com finais entre 0x0000 a 0x3fff. A tentativa de uso destes endereços comprometerá o correto funcionamento do sistema.

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*. 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 a memória é feito simplesmente dereferenciando um ponteiro para a posição de memória desejada. A escrita é feita do mesmo modo. Encontra-se emanexo códigos-exemplo para o teste das memórias DDR3 e SPI Flash.

Antes de executar o programa, é interessante habilitar a opção de debugação. 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 raíz do projeto do processador. Clica-se em "*Program*" para inciar 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 "*1 Launch on Hardware (GDB)*". No caso dos códigos-exemplo, algumas informações são imprimidas no console caso tudo tenha ocorrido confome o esperado.

1.3.3 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 forúns de discussão e conhecimentos gerais de programação embarcada. Apesar disso, o objetivo do experimento, quiça conseguir ler/escrever de/em endereços das memória DDR3 e SPI Flash específicos, foi atingido com sucesso.

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, a memória 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 para assim carregá-lo e interpretá-lo enquanto o transferindo para a memória DDR3.

Utilizará-se também o GPIO, que dá acesso aos pinos da placa, visto 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.

```
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=0xFFFFFF
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 ff  .Äÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
```

Figura 1.10: Cabeçalho do arquivo binário gerado no primeiro experimento para a configuração vazia.

1.4.1 Arquivo Binário

O arquivo binário é formado por três partes: um cabeçalho, uma palavra para sincronia e a configuração propriamente dita (**????**). O cabeçalho é formado por chaves e tamanhos, indicando diversos campos deste. Na tabela 1.1 pode-se observar os tamanhos e campos apresentados na figura 1.10. A figura 1.10 possui os primeiros 112 bytes, dos quais os primeiros 98 estão selecionados, da configuração parcial vazia construida

no experimento 1. O primeiro conjunto tamanho/chave indica, através da sequência de de palavras 0x0f, que a configuração é válida. Se estas palavras fossem 0x00 indicariam que a configuração não é mais válida, e se contivessem 0xff indicariam que a configuração está vazia.

O cabeçalho descrito na tabela 1.1 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.

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 *netlist* 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 sintese 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 1.1: Descrição do cabeçalho dos arquivos binários.

Ainda no cabeçalho, tem-se um 32 bytes de espaçamento preenchidos por com “0xff”. Em seguida, tem-se os bytes para autodetecção de largura de banda (**????**). Estes bytes (“0x00 00 00 bb 11 22 00 44”) são usados no modo de configuração paralelo 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 inicio da configuração proprimamente dita e para alinhar o fluxo de dados nos registradores internos.

Inversão dos bytes A interface ICAP e a ICAPE2, similares a interface SelectMAP (**??**), precisa de uma inversão na ordem dos bits nos bytes da configuração, incluindo a palavra de sincronia (**??**). Esta inversão é necessária apenas no uso destas três interfaces. Note que alguns tipos de arquivos sintetisados, como o MCS e o HEX, já podem ter estes bytes invertidos não sendo necessário invertê-los novamente (**??**).

1.4.2 Inicialização da Memória SPI Flash e a interface UART

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 a FPGA e programas para algum MicroBlaze embarcado (**??**). Este processo custoso (**??**) é pouco documentado e achou-se de difícil implementação. A solução encontrada mais interessante para se contornar este problema é o uso da interface UART para transmissão de dados entre o computador, do MicroBlaze e das característica não-voláteis desta memória para sua programação em um estágio preliminar de preparação da memória Flash.

A interface UART (*Universal Asynchronous Receiver/Transmitter*) é normalmente utilizada para a transmissão de informações de debugação do programa em execução e para a implementação das funções de leitura/escrita no MicroBlaze. Apesar disto, ela é aberta para ser utilizada de qualquer forma que se achar conveniente. Neste caso, pretende-se realizar a transmissão dos arquivos de configuração parciais para escrita em memória não-volátil. Em uma situação ideal, esta interface pode transferir os arquivos binários parciais do primeiro experimento em 46 segundos ($661284\ \text{bytes} \cdot \frac{8\ \text{bit}}{\text{byte}} \div 115200 \frac{\text{bits}}{\text{segundo}}$). Note que a transmissão de dados através da porta serial do computador pode ser filtrada para evitar o envio de pacotes vazios [3], poupando o canal de transmissão. Para solucionar este problema, deve-se modificar os pacotes enviados para que não sejam nulos, mas sem modificar o conteúdo do mesmo. Pode-se alcançar este objetivo introduzindo um bit alto no inicio de cada palavra enviada. Uma vez que a interface UART permite o envio de palavra de até 8 bits, pode-se, com esta modificação, enviar 7 bits de dados por pacote, aumentando o tempo de envio em aproximadamente 14%. O algoritmo proposto captura 7 bytes por vez e constrói 8 pacotes de dados com 1 na posição mais significativa (MSB) e 7 bits de dados nos seguintes, como mostrado na figura 1.11.

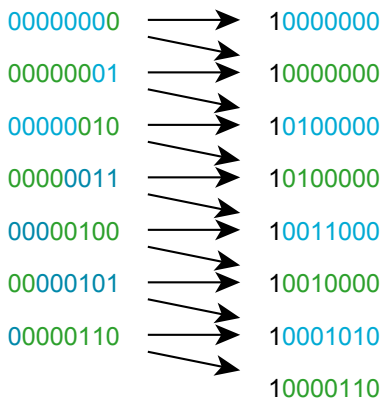


Figura 1.11: Transformação nos bytes do arquivo binário para que possam ser transmitidos. À esquerda tem-se os bytes originais e à direita tem-se os bytes transformados.

[3] *Re: UART Problem*, <http://forums.xilinx.com/t5/New-Users-Forum/UART-Problem/m-p/383265/highlight/true#M6901>

Após o recebimento de cada pacote da configuração parcial, o programa embarcado o processa fazendo a transformação inversa dos dados e os salva na memória SPI Flash. Assim como na transformação direta, este processamento é realizado em blocos de 8 bytes, gerando vetores de 7 bytes de dados.

1.4.3 Teste

O teste deste experimento compreende a construção de um sistema microprocessado que carregue os arquivos binários da memória SPI Flash para a memória DDR3, capture as informações relevantes e indique através de GPIO o final do processo. Este experimento contribuirá para a compreenção do processo de inicialização de memórias não-voláteis e do tratamento do cabeçalho dos arquivos binários. Usará-se os programas XPS e SDK, que fazem parte do Embedded Development Kit (EDK).

1.4.3.1 MicroBlaze

Utilizou-se o mesmo microcontrolador MicroBlaze construido no experimento passado, não sendo necessária nenhuma modificação.

1.4.3.2 SDK

Passo a passo do experimento 3 na SDK

1.4.4 Conclusão

Explicar como fui para o próximo experimento

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

Descrever o experimento 4

1.5.0.1 Registrador de *Status*

Este registrador, acessado através de uma sequência predefinida de comandos pela interface ICAPE2 (**??**), informa

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