Boletim Técnico da Escola Politécnica da USP
Departamento de Engenharia de Telecomunicações
e Controle

ISSN 1517-3550

BT/PTC/0011

Estudo de Caso: Tornando um Projeto Testável Utilizando Ferramentas Synopsys

Reinaldo Silveira

José Roberto A. Amazonas

São Paulo – 2000

FICHA CATALOGRÁFICA

Silveira, Reinaldo
Estudo de caso : tornando um projeto testável utilizando ferramentas
Synopsys / R. Silveira, J.R.A. Amazonas. -- São Paulo : EPUSP, 2000.
16 p. -- (Boletim Técnico da Escola Politécnica da USP, Departamento de Engenharia de Telecomunicações e Controle, BT/PTC/0011)

1. Testabilidade 2. Síntese automática I. Amazonas, José Roberto de Almeida II. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Telecomunicações e Controle III. Título IV. Série

ISSN 1517-3550

CDD 005.14
621.3815

Estudo de Caso: Tornando um Projeto Testável utilizando Ferramentas Synopsys.

Reinaldo Silveira,
Prof. José Roberto. A. Amazonas

Abstract— **Este trabalho é um estudo de caso que visa mostrar os passos de implementação que são necessários para tornar um projeto testável. Tomamos como exemplo dois projetos de média complexidade, a fim de demonstrar as condições reais de utilização das ferramentas de testabilidade. Ambos os projetos foram concebidos sem considerar nenhum aspecto de testabilidade, portanto as coberturas de falhas obtidas serão muito próximas às condições médias encontradas em diversas aplicações.**

Keywords— **Testabilidade, Synopsys,** *Scan*, *Boundary Scan*, **DFT.**

I. Introdução

ESTE trabalho tem por objetivo apresentar o *design flow* de síntese de um projeto escrito em VHDL[1], visando especialmente a testabilidade do mesmo. Será basicamente um trabalho exploratório dos recursos de síntese automática disponíveis no pacote da Synopsys [2]. Foram escolhidos dois blocos funcionais de complexidade e funcionalidade diferentes, ambos utilizados em sistemas SDH[3].

O sistema SDH (*Synchronous Digital Hierarchy*) é um sistema de multiplexação síncrona utilizado principalmente para transmissão em fibras ópticas. O modelo de multiplexação foi concebido para ser simples e facilmente configurável. Entretanto, para ser possível a transmissão de sinais provenientes de outros sistemas que não possuem as mesmas características, foram criados esquemas de mapeamento que permitem absorver as disparidades de sincronismo e alinhamento. Esses esquemas receberam o nome de *Virtual Containers* de ordem inferior (VC). Os VCs de ordem inferior são posteriormente agrupados em *Containers* de ordem superior e assim por diante, até compor o quadro completo a ser transmitido. Os *Containers* de ordem inferior são localizados em relação à hierarquia imediatamente superior através de ponteiros. Ao conjunto VC mais os ponteiros a ele associados damos o nome de *Tributary Unit* (TU).

Um dos circuitos que utilizaremos neste trabalho é um mapeador/demapeador de TU12. TU12 é a unidade tributária associada ao *virtual container* VC12, que por sua vez encapsula um sinal do tipo E1 de 2,048Mb/s definido na recomendação ITU-T G-703[4].

O outro circuito é um realinhador de TU12. A função do realinhador é, como o nome sugere, a de realinhar um tributário do tipo TU12 a fim de inseri-lo numa dada hierarquia. O processo de realinhamento é necessário para diminuir a latência do sinal E1 que está sendo transmitido.

Na seção II, será feita uma apresentação dos circuitos para que seja possível o entendimento dos procedimentos utilizados na implementação. Na seção III serão apresentados os resultados das implementações sem utilizar testabilidade. Estes resultados serão tomados como referência para avaliar o impacto da síntese automática de testabilidade sobre o sistema. Na seção IV o procedimento para inserção de cadeias de *scan* será apresentado, bem como os resultados obtidos. Na seção V será abordado o problema do *Boundary Scan* e finalmente na seção VI será apresentado a comparação dos resultados e a apresentação dos testes finais.

II. Apresentação dos Blocos

O sistema SDH emprega um esquema de multiplexação simplificada para tributários[1] de ordem mais elevada, e um esquema mais complexo para tributários de ordem inferior, pois muitas vezes estes são provenientes de outros tipos de hierarquia, apresentando características diversas do SDH. Os tributários de um modo geral precisam ser formatados convenientemente para poderem ser inseridos na hierarquia; este "envelope" inicial é chamado normalmente de *virtual container*(VC). Os *virtual containers* já possuem todas as características normais à hierarquia SDH; estes são divididos em octetos com funções específicas, contendo informações de manutenção da conexão e da informação a ser transmitida. Ao conjunto de octetos que contém a informação a ser transmitida dá-se o nome de *Payload*, ao conjunto de octetos que contém informações de gerenciamento e manutenção da conexão da-se o nome de *Path Overhead*. O *virtual container* é inserido na hierarquia SDH através de um ponteiro que assinala o primeiro octeto do tributário. Este mecanismo permite que sejam compensadas pequenas variações de sincronismo entre os diversos equipamentos que compõem o sistema. Quando um *virtual container* é associado a um ponteiro, esta estrutura recebe o nome de *Tributary Unit* (TU).

TU12 é a unidade tributária que encapsula um sinal do tipo E1 de 2,048Mb/s definido na recomendação ITU-T G.703, através de um VC12. A estrutura de um VC12 pode ser vista na figura 1.

silveira@lsi.usp.br.
jra@lcs.poli.usp.br

[1] Um tributário, falando de forma simplificada, é a informação que trafega na rede devidamente formatada nos padrões do SDH.

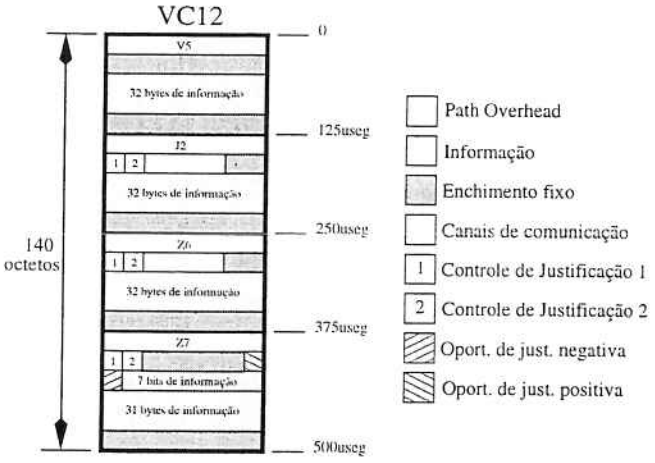

Fig. 1. Esquema de mapeamento de um E1 em um VC12.

Normalmente, os ponteiros estão localizados numa posição fixa bem determinada em relação ao nível hierárquico imediatamente superior, enquanto o VC fica "flutuando" no espaço a ele reservado. No caso do TU12 o *overhead* dos ponteiros é de quatro *bytes* sobre os 140 *bytes* do VC12, que são inseridos na sua estrutura a cada 125μseg. Na realidade nem todo este *overhead* é usado para os ponteiros, neste espaço também se encontram *bits* de sinalização, espaço para justificação e reserva. Além de apontar para o início de VC12 o *overhead* de TU12 também permite ajustar pequenas variações de sincronismo entre o equipamento que gerou o VC12 e o equipamento de hierarquia superior, através do processo de justificação. Os *bytes* de *overhead* de TU12 são denominados de V1, V2, V3 e V4. Quando há necessidade de justificação, os *bytes* V1 e V2 sinalizam que o *frame* corrente vai receber um *byte* a mais ou a menos dependendo da necessidade que foi detectada. Caso haja necessidade de receber um *byte* de informação adicional, este ocupará a posição de V3, caso a necessidade seja um *byte* a menos, o *byte* vago será a primeira posição após V3. A estrutura de um TU12 bem como o significado dos *bytes* de *overhead* pode ser vista na figura 2.

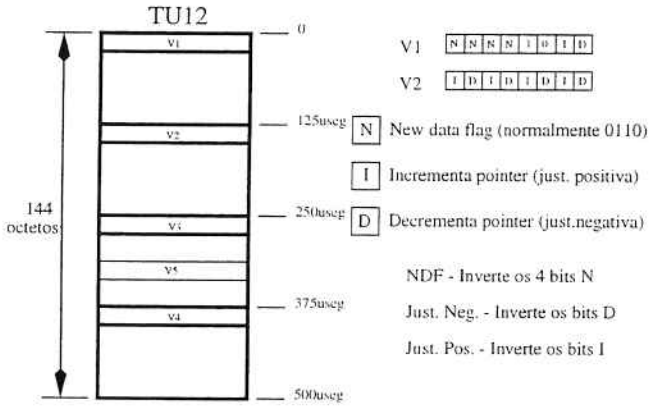

Fig. 2. Estrutura de um TU12.

Na multiplexação de tributários de ordem inferior, como é o caso do TU12, octetos de cada um dos tributários são intercalados sucessivamente até ocupar todo o espaço disponível no tributário de ordem imediatamente superior. Como já foi dito anteriormente, os tributários que estiverem sendo multiplexados precisam estar alinhados entre si e sincronizados com o tributário superior para que os ponteiros sejam posicionados sempre em posições conhecidas. Entretanto, se a implementação for um multiplexador do tipo *add-drop*[2], os TU12 obtidos de outras hierarquias não apresentarão necessariamente sincronismo coerente com a hierarquia local. Desta forma há a necessidade da implementação de um bloco que realinhe os ponteiros de um TU12 para um novo valor especificado. Uma forma trivial de resolver o problema é passar o TU12 por um *buffer* do tipo FIFO inserindo tantos atrasos quantos forem necessários até obter o sincronismo desejado. No entanto, esta solução é inaceitável pois implicaria no aumento da latência do sinal transportado em até 125μseg em cada ponto onde a operação for necessária. Uma outra solução, um pouco mais complexa mas muito mais inteligente, implica no recálculo dos ponteiros de TU12 de forma a introduzir o mínimo de latência necessário aos dados transportados. Esta função pode ser melhor compreendida através da figura 3. Se for tomado o eixo vertical como eixo de tempo, pode-se observar que apesar do realinhamento, a posição temporal de V5 praticamente não se altera. Será visto portanto qual é o circuito que implementa essa função.

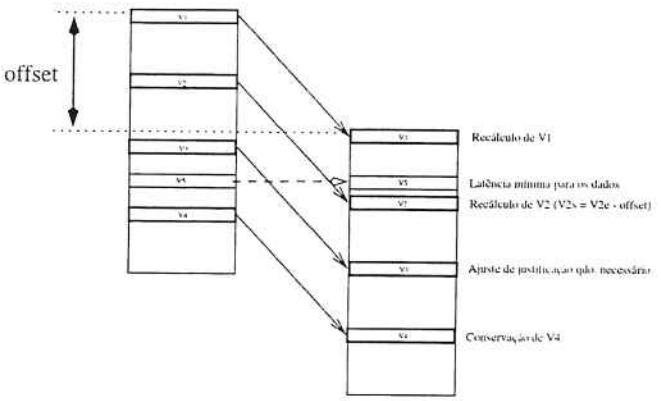

Fig. 3. Representação ilustrativa da função de realinhamento de TU12.

A. *Realinhador de TU12*

A idéia básica por trás do circuito realinhador consiste em separar o *payload* do *overhead*, guardando a informação de alinhamento, e inserir um novo *overhead* quando a informação de sincronismo de saída estiver disponível. Uma pequena FIFO é utilizada para absorver eventuais processos de justificação do *virtual container* e a latência de recálculo de ponteiros. O diagrama de blocos do circuito realinhador pode ser visto na figura 4.

Para efeito didático o circuito será dividido em quatro partes principais: a fifo, o controle de entrada, o controle

[2]Um multiplexador do tipo *add-drop* é um equipamento que insere(*add*) e retira(*drop*) tributários, sem contudo demultiplexar toda a hierarquia. Esta é, aliás, uma das grandes vantagens apregoadas pelo SDH.

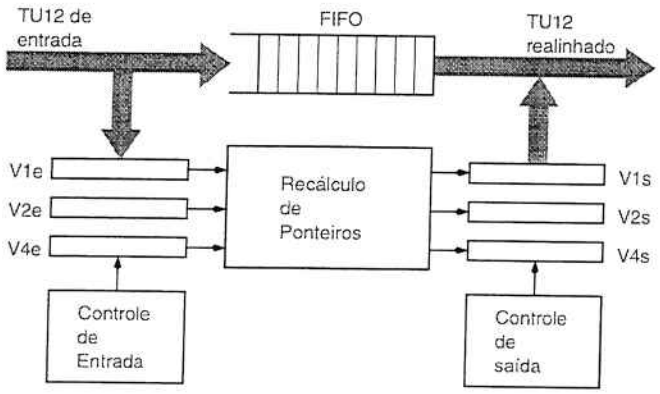

Fig. 4. Diagrama de blocos do circuito realinhador de TU12.

de saída e o bloco de recálculo de ponteiros. O controle de entrada implementa a parte do algoritmo que separa o *overhead* do TU12 do VC12. Esta função é implementada, basicamente, através de uma máquina de estados que conta os *bytes* do tributário independentemente do estado dos seus apontadores. Evidentemente para que a funcionalidade seja completa é preciso levar em consideração as condições de exceção, como justificações, mudanças voluntárias de ponteiros, e etc. A informação de *offset* também é armazenada em duas etapas: através do armazenamento de V2 e do cálculo do *offset* entre os dois alinhamentos. Estas informações são utilizadas pelo bloco de recálculo de ponteiros para o cálculo dos novos valores de V1 e V2.

O controle de saída passa a funcionar no momento em que existe requisições de saída. Evidentemente, a informação de sincronismo deve ter sido fornecida. Nesse momento os dados são fornecidos à saída à medida que são requisitados. A montagem do novo TU12 é feita nesse momento, também levando-se em consideração possíveis justificações, desta vez ditadas pelo nível de ocupação da FIFO. O recálculo de ponteiros, por sua vez, é feito no momento em que o sincronismo de saída é fornecido. Como todos os subsistemas funcionam sob um mesmo *clock* e ainda assim seu funcionamento está sujeito a eventos totalmente assíncronos, são utilizados circuitos para arbitrar o acesso a determinados elementos.

B. Mapeador/Demapeador de TU12

O sinal E1 é um sinal serial cujas características são definidas pela recomendação ITU-T G.703. Este sinal é normalmente codificado em HDB3, mas para efeito do exemplo vamos supor que o sinal E1 já passou por um decodificador de HDB3 para NRZ. O processo de mapeamento no VC12 está representado na figura 1. Nela podemos ver a porção destinada ao *overhead* do tributário, a área de informação, enchimentos fixos, canais de comunicação e controle. Considerando que não haja justificação, pode-se ver que é possível alojar $1024bits$ de informação em *frames* que se repetem a cada 500μseg, resultando na taxa nominal do sinal E1 (2,048Mbit/s). Entretanto, de acordo com a recomendação ITU-T G.703, o sinal E1 pode apresentar uma variação nominal de ±50 ppm (partes por milhão, ±102,4 bit/s), ou seja, a cada segundo pode-se apresentar uma diferença de 100 *bits* (quase 13 *bytes*) entre o equipamento fonte e o destino de E1. É necessário portanto que o tributário VC12 ofereça a possibilidade de absorver eventuais variações na taxa nominal de transmissão de E1 sem que haja perda de dados, isto é conseguido através do processo de justificação.

É possível ajustar a taxa de recepção do sinal E1 alocando um lugar dentro do *frame* de VC12 para que a região de dados possa crescer ou diminuir. Esta região está representada na figura pelos *bits* hachurados diagonalmente. Quando há um aumento nominal na taxa de transmissão, o *bit* assinalado como oportunidade de justificação positiva é utilizado para transportar um *bit* de informação. Quando há uma diminuição na taxa nominal, o *bit* assinalado como oportunidade de justificação negativa e que normalmente é utilizado para transporte de informação, é deixado de lado, sendo preenchido com um enchimento qualquer. A ocupação dos *bits* de oportunidade de justificação é assinalada através dos *bits* 1 e 2 da figura 1. Os *bits* aparecem três vezes dentro do *frame* para garantir a correta leitura dos mesmos através da simples redundância[3].

A forma usual de resolver este problema é através de uma FIFO, uma vez que os *bits* seriais de E1 são escritos na sua taxa nominal e são posteriormente processados numa taxa muito maior, intercalada por períodos de inatividade. O principal problema num sistema digital que trabalha com *clocks* diferentes, é o problema de meta-estabilidade que pode ocorrer nos elementos de armazenamento. A leitura dos *bits* de E1, que é seguramente o canal de mais baixa velocidade, poderia ser feita através da amostragem na freqüência mais elevada. Entretanto, esta técnica não é a mais conveniente pois apresenta a desvantagem de aumentar o consumo de potência e o projeto ficar atrelado à freqüência de amostragem, ou seja, se uma outra freqüência de operação precisasse ser utilizada o circuito precisaria ser redesenhado. Como o intuito do projeto, além de funcionar de maneira eficiente, é poder ser reutilizado em vários subsistemas SDH, foi adotada uma outra técnica descrita a seguir.

A solução adotada pode ser vista na figura 5. Nela é possível ver a implementação como um banco de memórias de um *bit*, arranjadas como uma RAM (memória de acesso aleatório). A escrita dos *bits* de entrada é feita na posição apontada por um contador e é controlada por uma base de tempo do próprio sinal de entrada. A saída, de forma semelhante, é feita lendo-se a posição apontada por um outro contador segundo a base de tempo do sistema que irá mapear o E1. A meta-estabilidade nunca ocorrerá se for garantido que os apontadores nunca apontem para uma mesma posição simultaneamente, ou que nunca um apontador "rode" sobre o outro, o que acarretaria também a perda de dados. Se for escolhido um tamanho conveniente para a FIFO e os valores iniciais (de *reset*) dos apontadores, esta condição é plenamente satisfeita.

[3]É fácil ver que um esquema assim caracterizado é capaz de adequar variações de até ±2000 bit/s, muito superior a necessária para acomodar o E1. Daí pode-se concluir que a freqüência de justificações é bem inferior à freqüência do VC12.

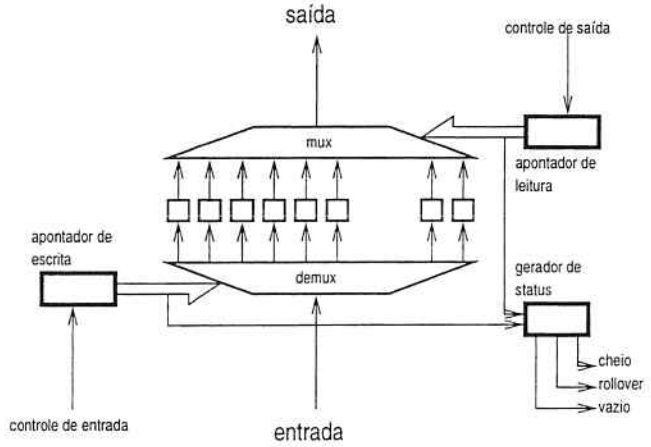

Fig. 5. Diagrama de blocos da FIFO.

A montagem do TU12, segue uma seqüência bastante parecida com aquela usada no circuito realinhador. Uma diferença significativa em relação à operação do realinhador é que neste último manipula-se 8 *bits* de dados, enquanto no mapeador a fifo é de *bits*. Utiliza-se, então, um bloco *shifter* especial que além de fazer a conversão serial/paralelo, pode ser configurado para fornecer palavras de 1, 7 e 8 bits. Isto é necessário para o processo de mapeamento com justificação, que será função do índice de utilização da fifo.

O processo inverso, o de demapear o TU12 e recuperar o sinal E1, segue procedimento semelhante. Através do mesmo procedimento utilizado no bloco realinhador, separa-se o *overhead* de TU12 do VC12, armazenando somente as informações relevantes. Em seguida procede-se a separação do *overhead* de VC12 do tributário E1, de maneira inversa a descrita anteriormente, através de um conversor paralelo/serial que admite palavras de 1, 7 e 8 bits. Os dados seriais são então inseridos numa FIFO semelhante à da figura 5. A saída se dá através de um bloco chamado `pscalerv` (vide fig. 7). Este bloco sintetiza, a partir do *clock* do sistema, um *clock* de aproximadamente 2,048Mhz que esvaziará os bits da FIFO de saída. O *clock* será levemente acelerado se a FIFO estiver muito cheia e retardado se a FIFO estiver muito vazia. Este procedimento produz um E1 com um pequeno *jitter* de saída que pode ser facilmente eliminado com os procedimentos convencionais.

III. Síntese dos Blocos

A síntese dos circuitos não apresentou grandes dificuldades, uma vez que todo o projeto visava a síntese automática. Algumas modificações foram necessárias para compatibilizar os *packages* de VHDL da ferramenta de síntese da Synopsys[4].

A ferramenta básica para síntese no ambiente Synopsys é o *Design Compiler*. O *Design Compiler* sintetiza uma descrição RTL num *netlist* de portas lógicas dependentes de tecnologia, ou seja, o processo completo de síntese requer a existência de uma biblioteca de células descritas e caracterizadas sob uma dada tecnologia de fabricação. Sobre o *Design Compiler* agregam-se todas as outras ferramentas Synopsys. O fluxo de trabalho genérico recomendado para a ferramenta pode ser visto na figura 6.

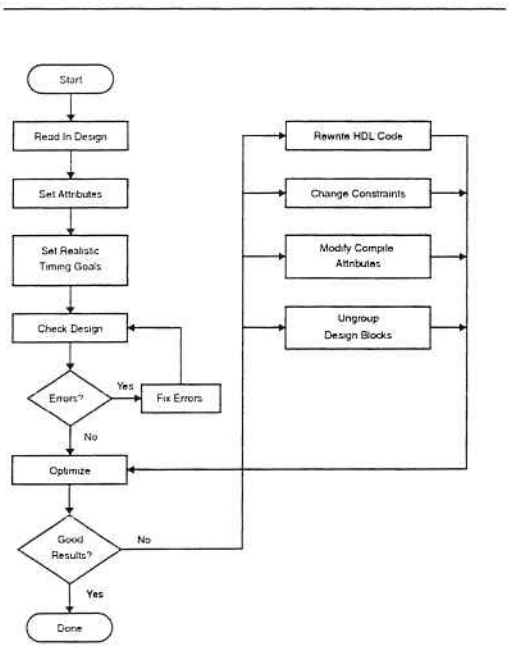

Fig. 6. Processo de Síntese.

O *Design Compiler* possui duas interfaces, o `dc_shell` e o `design_analyzer`. O `dc_shell`, como o nome sugere, é um *shell* onde os comandos do *Design Compiler* são ativados e as mensagens e informações do processo são enviadas. O `design_analyzer` é a contrapartida gráfica das ferramentas de síntese. Nele pode-se acessar todos os comandos e variáveis da ferramenta, com a vantagem de oferecer representações gráficas (esquemas elétricos) dos sistemas em desenvolvimento correlacionados com os eventos de interesse. Por exemplo, durante uma verificação, ao "clicar" sobre uma indicação de erro o ponto correspondente na representação esquemática é imediatamente ressaltada. Esse recurso é muito útil na fase de desenvolvimento, tendo sido utilizado com grande freqüência durante este trabalho.

Apesar de ter sido utilizado com maior freqüência o `design_analyzer`, os comandos serão apresentados como se tivessemos utilizado o `dc_shell`, simplesmente por ser uma representação mais cômoda, a exemplo aliás do que é feito nos próprios tutoriais da Synopsys.

A. Síntese do Bloco Realinhador

O projeto original é composto por onze módulos: `count144`, `en_just`, `fifo`, `inpctrl`, `ofcounter`, `outctrl`, `pcalc`, `pointer`, `pointer_gen`, `realin` e `reqarbt`, sendo o de mais alta hierarquia o módulo `realin`. Como foi dito anteriormente, alguns blocos precisaram de algumas adaptações devido à incompatibilidade entre funções de conversão

[4]O projeto foi desenvolvido utilizando-se basicamente as ferramentas da Mentor Graphics [5], algumas funções de conversão de tipos utilizadas nesta não são compatíveis com as ferramentas Synopsys.

de dados do VHDL da Synopsys e o da Mentor Graphics[5]. Uma destas adaptações é mostrada a seguir:

```
[autel@cynthia] diff begin/ofcounter.vhd scanbsd/ofcounter.vhd
10c13,14
< LIBRARY ARITHMETIC;
---
> --LIBRARY ARITHMETIC;
>
12c16,17
< USE ARITHMETIC.STD_LOGIC_ARITH.ALL;
---
> --USE ARITHMETIC.STD_LOGIC_ARITH.ALL;
> USE IEEE.STD_LOGIC_ARITH.ALL;
36,37c41,43
<   offset <= to_stdlogicvector(istate,8);
<
---
> --  offset <= to_stdlogicvector(istate,8);
>   offset <= conv_std_logic_vector(istate,8);
```

Os blocos que precisaram de modificações semelhantes formam: `ofcounter`, `pcalc`, `pointer` e `realin`. Por serem modificações apenas de caráter específico da ferramenta, ou seja, não introduzindo nenhuma mudança na arquitetura original, não serão detalhadas as demais modificações.

Seguindo a seqüência sugerida pelo diagrama da figura 6, é executada a seguinte seqüência de comandos para carregar o projeto:

```
dc_shell> read -f vhdl count144.vhd
dc_shell> read -f vhdl en_just.vhd
dc_shell> read -f vhdl fifo.vhd
dc_shell> read -f vhdl inpctrl.vhd
dc_shell> read -f vhdl ofcounter.vhd
dc_shell> read -f vhdl outctrl.vhd
dc_shell> read -f vhdl pcalc.vhd
dc_shell> read -f vhdl pointer.vhd
dc_shell> read -f vhdl pointer_gen.vh
dc_shell> read -f vhdl reqarbt.vhd
dc_shell> read -f vhdl realin.vhd
dc_shell> uniquify
dc_shell> remove_constraint -all
dc_shell> remove_clock -all
```

O comando `uniquify` cria, para blocos que são usados mais de uma vez, instâncias de blocos distintas para cada uma das chamadas. Como o projeto opera em uma velocidade relativamente baixa e também não ser objetivo deste trabalho explorar os recursos de síntese e otimização particulares do Synopsys foram removidos quaisquer *constraints*.

O processo de síntese com *Design Compiler* funciona da seguinte forma: toda vez que um elemento é lido, num dos formatos admitidos, o bloco é traduzido para uma representação interna, onde os elementos de *hardware* principais já estão identificados. Elementos seqüenciais como banco de registradores e flip-flops são inferidos e separados dos elementos puramente combinatórios, e primitivas de caráter combinatório, como somadores, incrementadores, etc..., são inferidas e usadas com todos os demais para compor o bloco recém lido. Estes componentes constituem a representação estrutural da descrição de entrada e é a partir dela que são feitas as etapas posteriores de síntese, mapeamento tecnológico e otimização.

Neste ponto basta indicar a tecnologia e proceder à síntese. A indicação da tecnologia depende do ajuste de uma série de variáveis internas do *Design Compiler*. Por comodidade estas variáveis podem ser ajustadas automaticamente através de um arquivo de configuração. Um exemplo do arquivo de configuração de nome "`.synopsys_dc.setup`" é mostrada a seguir:

```
search_path = {. ams_synopsys_lib/cub33
/usr/local/tools/synopsys/1998.08/libraries/syn
/home/projects/bia/fuzzy/bibliotecas/synopsys/synopsys/cub33/};
link_library = {cub33.db};
target_library = {cub33.db};
symbol_library = {cubs33.sdb};
define_design_lib work -path WORK;
synthetic_library = {dw_foundation.sldb}
link_library = target_library + synthetic_library
search_path = search_path + {synopsys_root + "/dw/sim_ver"}
synlib_wait_for_design_license = {"DesignWare-Foundation"}
```

O comando `compile` é usado para fazer a síntese, transformando a representação estrutural de componentes genéricos, numa representação estrutural otimizada de componentes da tecnologia de trabalho.

`dc_shell > compile`

O comando `compile` possui diversos argumentos para direcionar a síntese e otimização, por exemplo:

```
dc_shell > compile [-no_map] [-map_effort low | medium | high]
                   [-area_effort low | medium | high]
                   [-incremental_mapping] [-exact_map]
                   [-in_place] [-routability [-add_porosity]]
                   [-scan] [-verify] [-verify_hierarchically]
                   [-verify_effort low | medium | high]
                   [-ungroup_all] [-boundary_optimization]
                   [-no_design_rule | -only_design_rule]
                   [-background run_name] [-host machine_name]
                   [-arch architecture] [-xterm] [-top]
```

É possível agora tirar alguns relatórios com o resultado da síntese.

```
dc_shell > report_area
****************************************
Report : area
Design : realin
Version: 1999.05
Date   : Thu Dec  2 14:59:38 1999
****************************************

Library(s) Used:

    cub33 (File: ~fuzzy/bibliotecas/synopsys/synopsys/cub33/cub33.db)

Number of ports:                22
Number of nets:                220
Number of cells:               160
Number of references:           28

Combinational area:        365.290009
Noncombinational area:     427.500000
Net Interconnect area:    1424.875610

Total cell area:           792.790039
Total area:               2217.665527
```

Tomando o tamanho de uma porta `Nand` de duas entradas como uma célula padrão[6], conclui-se que o componente possui $\frac{792,8}{0,41} = 1933,7$ portas equivalentes. Em seguida pode-se executar o comando `check_test` para verificar o estado em relação à síntese de cadeias de *scan*.

[5]Ferramenta onde foi desenvolvido originariamente o projeto.

[6]É possível discriminar o tamanho de cada célula da biblioteca usada para a síntese, desta forma elegemos uma `Nand` de duas entradas como célula de tamanho padrão, cujo tamanho é igual 0,41. A documentação da biblioteca não é clara quanto à unidade de área usada para esta medida, mas tudo indica que a unidade seja $10^{-3}mm^2$, uma vez que a biblioteca em questão é da technologia AMS cub33 de $0.6\mu m$

```
dc_shell > check_test
...
*************************************************
  Test Design Rule Violation Summary

  Total violations: 17
*************************************************

MODELING VIOLATIONS
      4 Cell has no scan equivalent violations (TEST-120)
TOPOLOGY VIOLATIONS
     10 Unconnected input pin violations (TEST-332)
PROTOCOL VIOLATIONS
      3 Asynchronous pins uncontrollable violations (TEST-116)

*************************************************
```

O *Design Compiler* possui um gerador de vetores de teste automático (ATPG), capaz de gerar um conjunto de vetores baseando-se exclusivamente na topologia do circuito. Para isso a ferramenta identifica todos os elementos seqüenciais do netlist e assinala-os como elementos da cadeia de *scan*: desta forma suas entradas podem ser caracterizadas como saídas padrão (nós observáveis) e suas saídas como entradas padrão (nós controláveis), ficando o resto da lógica caracterizado como puramente combinatório. O processo de geração dos vetores calcula uma cobertura baseada nestas premissas, e continua até que uma cobertura desejada seja alcançada ou que um certo tempo de CPU seja excedido. A geração é disparada através do seguinte comando:

```
dc_shell > create_test_patterns
...
  Combinational Test Pattern Generation starts:

    Start random pattern generation...

       60.47% faults processed ; cumulative fault coverage =  60.47%
       70.26% faults processed ; cumulative fault coverage =  70.26%
       71.40% faults processed ; cumulative fault coverage =  71.40%
       71.84% faults processed ; cumulative fault coverage =  71.84%
       72.54% faults processed ; cumulative fault coverage =  72.54%
       72.79% faults processed ; cumulative fault coverage =  72.79%
       72.92% faults processed ; cumulative fault coverage =  72.92%
       73.30% faults processed ; cumulative fault coverage =  73.30%
       73.39% faults processed ; cumulative fault coverage =  73.39%
       73.60% faults processed ; cumulative fault coverage =  73.60%
       73.64% faults processed ; cumulative fault coverage =  73.64%
       73.74% faults processed ; cumulative fault coverage =  73.74%
       74.10% faults processed ; cumulative fault coverage =  74.10%
       74.25% faults processed ; cumulative fault coverage =  74.25%

    ...End random pattern generation

    Start deterministic pattern generation...

       98.54% faults processed ; cumulative fault coverage =  84.37%
       99.62% faults processed ; cumulative fault coverage =  85.56%
      100.00% faults processed ; cumulative fault coverage =  85.92%

    ...End deterministic pattern generation\


                              Non-collapsed     Collapsed
    No. of detected faults    5602              3642
    No. of abandoned faults   0                 0
    No. of tied faults        482               482
    No. of redundant faults   21                17
    No. of untested faults    1101              597
    Total no. of faults       7206              4738
    Fault coverage            83.57             85.92

    No. of test patterns       136

    Test Generation Time (CPU)  7.41 sec

    Start compaction...

    ...End compaction

    No. of compacted patterns    120

    Compaction Time (CPU)       0.65 sec
```

```
...Writing test program realin (without vectors) to file
      ~autel/Synopsys/realinhador/xwork/begin/realin.vdb
```

Este resultado dá uma visão da potencialidade do teste com cadeias de *scan* se estas forem utilizadas com um mínimo de esforço. Note no final do *report* que os vetores não são salvos no arquivo de vetores, obviamente o resultado da geração não faz o menor sentido se a cadeia não existir de fato. Por isso, a ferramenta Synopsys **nunca** salva os vetores gerados se o bloco analisado não possuir cadeias de *scan*, caso possua elementos seqüenciais.

B. Síntese do Bloco Mapeador/Demapeador de TU12

O bloco mapeador/demapeador é composto por dezenove módulos. As adaptações de conversão de tipos abstratos, mencionadas no ítem anterior foram necessárias em nove blocos. A síntese foi conduzida seguindo o seguinte *script*:

```
dc_shell> read -f vhdl count144.vhd
dc_shell> read -f vhdl E164mem.vhd
dc_shell> read -f vhdl bip2.vhd
dc_shell> read -f vhdl crc7.vhd
dc_shell> read -f vhdl justm.vhd
dc_shell> read -f vhdl loopback.vhd
dc_shell> read -f vhdl pscalerv.vhd
dc_shell> read -f vhdl shifter_in.vhd
dc_shell> read -f vhdl shifter_out.vhd
dc_shell> read -f vhdl tm12.vhd
dc_shell> read -f vhdl tu12c.vhd
dc_shell> read -f vhdl tu12dm.vhd
dc_shell> read -f vhdl vc12c.vhd
dc_shell> read -f vhdl vd12.vhd
dc_shell> read -f vhdl fifotu12.vhd
dc_shell> read -f vhdl tu12mp.vhd
dc_shell> read -f vhdl fifotu12_out.vhd
dc_shell> read -f vhdl vc12dm.vhd
dc_shell> read -f vhdl tu12.vhd
dc_shell> uniquify
dc_shell> remove_constraint -all
dc_shell> remove_clock -all
```

As mesmas condições iniciais utilizadas no bloco realinhador foram usadas na síntese do mapeador. Nestas condições temos:

```
dc_shell> compile
dc_shell> report_area
****************************************
Report : area
Design : tu12
Version: 1999.05
Date   : Mon Dec  6 15:22:15 1999
****************************************

Library(s) Used:

    cub33 (File: ~fuzzy/bibliotecas/synopsys/synopsys/cub33/cub33.db)

Number of ports:              44
Number of nets:              344
Number of cells:             224
Number of references:         33

Combinational area:      1359.389893
Noncombinational area:   1494.360107
Net Interconnect area:   5264.945801

Total cell area:         2853.750000
Total area:              8118.695801
```

Mais uma vez tomando uma porta equivalente como referência, temos: $\frac{2853.8}{0.41} = 6960$ portas equivalentes. Em seguida pode-se verificar o estado do componente.

```
dc_shell> check_test
...
```

```
*************************************************
  Test Design Rule Violation Summary

  Total violations: 130
*************************************************

MODELING VIOLATIONS
     60 Cell has no scan equivalent violations (TEST-120)
TOPOLOGY VIOLATIONS
      8 Improperly driven three-state net violations (TEST-115)
     37 Unconnected input pin violations (TEST-332)
PROTOCOL VIOLATIONS
     14 Uncontrollable clock violations (TEST-169)
CAPTURE VIOLATIONS
     11 Clock used as data violations (TEST-131)


*************************************************
  Sequential Cell Summary

  74 out of 745 sequential cells have violations
*************************************************

SEQUENTIAL CELLS WITH VIOLATIONS
   *  74 cells are black boxes
SEQUENTIAL CELLS WITHOUT VIOLATIONS
   * 671 cells are valid scan cells
```

Note que o número de violações das regras de testabilidade da ferramenta é muito maior neste bloco. Provavelmente a estimativa de cobertura também será pior, como pode ser vista a seguir:

```
...

  Combinational Test Pattern Generation starts:

    Start random pattern generation...

       20.09% faults processed ; cumulative fault coverage =  20.09%
       21.77% faults processed ; cumulative fault coverage =  21.77%
       23.00% faults processed ; cumulative fault coverage =  23.00%
       23.53% faults processed ; cumulative fault coverage =  23.53%
       24.22% faults processed ; cumulative fault coverage =  24.22%
       24.54% faults processed ; cumulative fault coverage =  24.54%
       24.78% faults processed ; cumulative fault coverage =  24.78%
       25.02% faults processed ; cumulative fault coverage =  25.02%
       25.87% faults processed ; cumulative fault coverage =  25.87%
       25.95% faults processed ; cumulative fault coverage =  25.95%
       26.09% faults processed ; cumulative fault coverage =  26.09%
       26.17% faults processed ; cumulative fault coverage =  26.17%
       26.27% faults processed ; cumulative fault coverage =  26.27%
       26.36% faults processed ; cumulative fault coverage =  26.36%
       26.43% faults processed ; cumulative fault coverage =  26.43%
       26.47% faults processed ; cumulative fault coverage =  26.47%
       26.53% faults processed ; cumulative fault coverage =  26.53%
       26.57% faults processed ; cumulative fault coverage =  26.57%
       26.61% faults processed ; cumulative fault coverage =  26.61%

    ...End random pattern generation

    Start deterministic pattern generation...

       72.09% faults processed ; cumulative fault coverage =  29.15%
...
       73.59% faults processed ; cumulative fault coverage =  29.15%
       76.19% faults processed ; cumulative fault coverage =  31.28%
...
       79.88% faults processed ; cumulative fault coverage =  31.28%
       81.25% faults processed ; cumulative fault coverage =  32.54%
       82.15% faults processed ; cumulative fault coverage =  33.52%
...
       89.05% faults processed ; cumulative fault coverage =  33.52%
       89.43% faults processed ; cumulative fault coverage =  33.90%
...
       92.52% faults processed ; cumulative fault coverage =  33.90%
       92.83% faults processed ; cumulative fault coverage =  34.12%
...
       99.56% faults processed ; cumulative fault coverage =  34.12%
      100.00% faults processed ; cumulative fault coverage =  34.23%

    ...End deterministic pattern generation


                                Non-collapsed     Collapsed
    No. of detected faults      8335              5616
    No. of abandoned faults     0                 0
    No. of tied faults          1504              1504
    No. of redundant faults     0                 0
    No. of untested faults      16871             10792
    Total no. of faults         26710             17912
    Fault coverage              33.07             34.23

    No. of test patterns        339

    Test Generation Time (CPU)  44.29 sec

    Start compaction...

    ...End compaction

    No. of compacted patterns   310

    Compaction Time (CPU)       3.35 sec
...Writing test program tu12 (without vectors) to file
       ~autel/Synopsys/tu12/xwork/begin/tu12.vdb
```

Pode-se notar um esforço muito maior para conseguir alguma cobertura, obviamente em função do número de violações encontrado anteriormente. O que será feito a seguir é, sistematicamente, verificar uma a uma as violações encontrada nos projetos e corrigí-las. Este processo será detalhado na próxima seção.

IV. Inserindo *Scan*

Antes de começar a corrigir as violações de testabilidade indicadas na análise anterior, foi feita a inserção do *scan* no bloco realinhador sem nenhuma modificação e verificado se o resultado correspondeu a estimativa.

A. *Bloco Realinhador*

Sem introduzir nenhuma modificação, o bloco realinhador foi compilado com a seguinte seqüência de comandos:

```
dc_shell> read -f vhdl count144.vhd
dc_shell> compile -scan
dc_shell> insert_scan
dc_shell> read -f vhdl en_just.vhd
dc_shell> compile -scan
dc_shell> insert_scan
dc_shell> read -f vhdl fifo.vhd
dc_shell> compile -scan
dc_shell> insert_scan
dc_shell> read -f vhdl inpctrl.vhd
dc_shell> compile -scan
dc_shell> insert_scan
dc_shell> read -f vhdl ofcounter.vhd
dc_shell> compile -scan
dc_shell> insert_scan
dc_shell> read -f vhdl outctrl.vhd
dc_shell> compile -scan
dc_shell> insert_scan
dc_shell> read -f vhdl pcalc.vhd
dc_shell> compile -scan
dc_shell> insert_scan
dc_shell> read -f vhdl pointer.vhd
dc_shell> compile -scan
dc_shell> insert_scan
dc_shell> read -f vhdl pointer_gen.vhd
dc_shell> compile -scan
dc_shell> insert_scan
dc_shell> read -f vhdl reqarbt.vhd
dc_shell> compile -scan
dc_shell> insert_scan
dc_shell> read -f vhdl realin.vhd
dc_shell> uniquify
dc_shell> remove_constraint -all
dc_shell> remove_clock -all
dc_shell> compile -scan
dc_shell> insert_scan
```

Note que desta vez foi utilizado o argumento `-scan` junto com o comando `compile`. Este argumento instrui o com-

pilador a utilizar em todos os elementos seqüenciais (Flip-flops), os seus equivalentes de *scan*. Desta forma todas as otimizações, de área e espaço, levam em consideração o novo componente. Outra particularidade é que cada módulo foi compilado separadamente, assim como a inserção da cadeia de *scan*. Isto foi necessário pois a inserção global produzia um erro desconhecido que abortava a ferramenta. O resultado da síntese é visto a seguir:

```
dc_shell> report_area
...
Combinational area:        436.300049
Noncombinational area:     328.100006
Net Interconnect area:    1495.801636

Total cell area:           764.400024
Total area:               2260.201660
```

Pode-se ver que o aumento de área é extremamente pequeno, apesar da área das células ter diminuído. Entretanto, ao se verificar as reais condições de testabilidade tem-se:

```
dc_shell> create_test_patterns
...
  Combinational Test Pattern Generation starts:

    Start random pattern generation...

       40.44% faults processed ; cumulative fault coverage =  40.44%
       41.32% faults processed ; cumulative fault coverage =  41.32%
       41.44% faults processed ; cumulative fault coverage =  41.44%
       41.50% faults processed ; cumulative fault coverage =  41.50%
       41.52% faults processed ; cumulative fault coverage =  41.52%

    ...End random pattern generation

    Start deterministic pattern generation...

       92.45% faults processed ; cumulative fault coverage =  41.52%
       93.45% faults processed ; cumulative fault coverage =  41.52%
       94.45% faults processed ; cumulative fault coverage =  41.52%
       95.31% faults processed ; cumulative fault coverage =  41.70%
       96.48% faults processed ; cumulative fault coverage =  41.70%
       97.94% faults processed ; cumulative fault coverage =  41.70%
       98.86% faults processed ; cumulative fault coverage =  42.62%
      100.00% faults processed ; cumulative fault coverage =  42.64%

    ...End deterministic pattern generation


                                Non-collapsed     Collapsed
    No. of detected faults      2786              2135
    No. of abandoned faults     0                 0
    No. of tied faults          0                 0
    No. of redundant faults     0                 0
    No. of untested faults      5058              2872
    Total no. of faults         7844              5007
    Fault coverage              35.52             42.64

    No. of test patterns         44

    Test Generation Time (CPU)  5.74 sec

    Start compaction...

    ...End compaction

    No. of compacted patterns    35

    Compaction Time (CPU)       0.25 sec
...Writing test program realin to file
       ~autel/Synopsys/realinhador/xwork/begin/realin.vdb
```

Note que um retrabalho é necessário a fim de alcançar os níveis desejados de cobertura. Dois procedimentos podem ser utilizados para corrigir as violações de um projeto. A primeira é utilizar o relatório produzido pelo comando `check_test`. Este comando quando chamado de dentro do `design_analizer`, permite uma ligação entre a mensagem/violação apresentada e seu correspondente no esquema elétrico gerado. Entretanto, a menos que se conheça muito bem o *design* e as possíveis variações sintetizadas, este procedimento pode ser confuso. É útil na verificação dos elementos inferidos, ou na determinação de inferências indesejadas por erro de codificação.

A segunda técnica é mais simples, porém parece ser mais trabalhosa. Consiste em compilar, checar as violações e corrigí-las, inserir o *scan*, módulo a módulo. Nesta técnica é possível manter um controle rígido da síntese e do estado geral do circuito. Foi usada esta segunda técnica onde foram identificados problemas que acabavam por induzir muitos outros.

O primeiro problema foi encontrado no módulo `fifo`. Neste módulo foi utilizado um elemento seqüencial (Flip-flop) com *set* e *reset* assíncrono. Nestes casos a ferramenta não pode utilizar o componente na cadeia de *scan*, pois uma dada combinação de teste poderia modificar o valor interno do *flip-flop* e portanto corrompendo a cadeia de *scan*. Por outro lado, se o componente for deixado de fora um nó incontrolável aparecerá no circuito, propagando essa incontrolabilidade para vários outros nós do sistema. No módulo `fifo` este recurso foi utilizado inicialmente, conforme pode ser visto no trecho VHDL, a seguir:

```
...
    IF ((rst = '0') or (NDF_rst = '1')) THEN
      pointer <= 3;
      first <= '1';
    ELSIF V1s_req = '1' THEN
      first <= '0';
    ELSIF rising_edge(clk) THEN
      IF (ctin = '1' and first = '0') THEN
         IF pointer = 7 THEN
...
```

A correção pode ser vista a seguir:

```
...
    IF rising_edge(clk) THEN
      IF ((rst = '0') or (NDF_rst = '1')) THEN
          pointer <= 3;
          first <= '1';
      ELSIF V1s_req = '1' THEN
          first <= '0';
      END IF;
      IF (ctin = '1' and first = '0') THEN
          IF pointer = 7 THEN
...
```

Após a modificação o modulo passou a apresentar 100% de cobertura. Um outro problema ocorreu no módulo `outctrl`. Alguma diferença entre a síntese Synopsys e a síntese Mentor Graphics, induz numa diferente inferência dos elementos seqüenciais, fazendo com que alguns *latches* sejam sintetizados indevidamente com a codificação utilizada. Na verdade este é um tipo de problema que pode passar despercebido, uma vez que a utilização destes *latches* não modifica a funcionalidade do circuito.

Revendo a funcionalidade do circuito, foram explicitadas as condições irrelevantes, principalmente quando era utilizado o comando VHDL `CASE`, solucionando o problema.

Após estas pequenas modificações o bloco foi sintetizado com a mesma seqüência de comandos apresentada anteriormente e o resultado pode ser visto a seguir:

```
dc_shell> report_area
...
Combinational area:        435.890045
Noncombinational area:     327.839996
Net Interconnect area:    1494.743042
```

```
Total cell area:          763.730042
Total area:              2258.473145
...
dc_shell> create_test_patterns
...
  Combinational Test Pattern Generation starts:

    Start random pattern generation...

       82.86% faults processed ; cumulative fault coverage =  82.86%
       91.67% faults processed ; cumulative fault coverage =  91.67%
       94.24% faults processed ; cumulative fault coverage =  94.24%
       95.10% faults processed ; cumulative fault coverage =  95.10%
       95.46% faults processed ; cumulative fault coverage =  95.46%
       95.70% faults processed ; cumulative fault coverage =  95.70%
       95.89% faults processed ; cumulative fault coverage =  95.89%
       96.12% faults processed ; cumulative fault coverage =  96.12%
       96.39% faults processed ; cumulative fault coverage =  96.39%
       96.49% faults processed ; cumulative fault coverage =  96.49%
       96.55% faults processed ; cumulative fault coverage =  96.55%
       96.57% faults processed ; cumulative fault coverage =  96.57%

    ...End random pattern generation

    Start deterministic pattern generation...

       98.31% faults processed ; cumulative fault coverage =  98.31%
       99.22% faults processed ; cumulative fault coverage =  99.22%
      100.00% faults processed ; cumulative fault coverage = 100.00%

    ...End deterministic pattern generation


                              Non-collapsed     Collapsed
    No. of detected faults    7807              5142
    No. of abandoned faults   0                 0
    No. of tied faults        0                 0
    No. of redundant faults   23                17
    No. of untested faults    0                 0
    Total no. of faults       7830              5159
    Fault coverage            100.00            100.00

    No. of test patterns       148

    Test Generation Time (CPU)  6.07 sec

    Start compaction...

    ...End compaction

    No. of compacted patterns    132

    Compaction Time (CPU)       0.63 sec
...Writing test program realin to file
       ~autel/Synopsys/realinhador/xwork/begin/realin.vdb
```

Note que após a inserção da cadeia de *scan* os vetores de teste são efetivamente escritos no arquivo de saída. Posteriormente este arquivo pode ser convertido para VHDL, Verilog e alguns outros formatos de testadores comerciais. Os vetores em formato VHDL/Verilog, são normalmente utilizados para simulação e verificação dos tempos do circuito em condições de teste.

B. *Mapeador/Demapeador*

Foi visto no ítem anterior que não basta a pura e simples inserção do *scan* no circuito sintetizado. Se não forem corrigidas as violações indicadas, somente coberturas muito baixas são conseguidas. Portanto, neste segundo circuito será omitida a síntese preliminar e serão analisados diretamente os problemas do projeto.

O bloco mapeador/demapeador além de maior é também mais complexo que o realinhador. Uma diferença básica é a utilização de diferentes *clocks* em diferentes partes do circuito. A presença de *ports* bidirecionais também é um agravante. Foi utilizado então, o procedimento mencionado anteriormente, cada submódulo foi analisado individualmente, hierarquicamente dos mais inferiores até a *topcell* e onde foram identificados os seguintes problemas.

Os módulos problemáticos foram: `fifotu12_out`, `vc12dm` e `tu12`, respectivamente seguindo a ordem hierárquica, sendo `tu12` a *topcell*. De fato, esta é a listagem dos módulos que necessitam de modificações; na realidade os problemas se restringem a `fifotu12_out` e `tu12`.

O módulo `fifotu12_out` (fig. 7) é muito parecido com a fifo de entrada, com exceção da geração local do sinal `clk2` que estará regendo a saída dos bits da fifo. A utilização deste *clock* sintetizado faz com que, durante o teste, este e os demais nós dependentes fiquem incontroláveis, tornando o teste muito difícil.

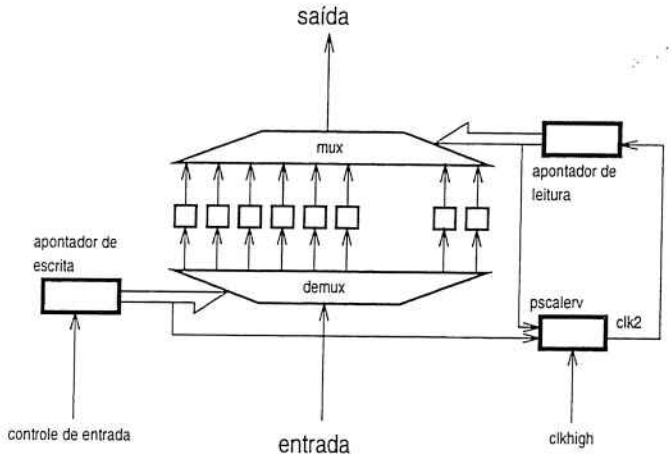

Fig. 7. Diagrama de blocos da FIFO de saída.

A solução nestes casos é introduzir artificialmente sinais que controlem o nó `clk2`, conforme pode ser visto na figura 8. Um *mux* é introduzido entre o ponto de geração do sinal `clk2` e a sua utilização. O sinal `clk3` é multiplexado entre `clk2` e o sinal `clktest`, através do sinal `test_mode`. Estes novos sinais são utilizados somente durante o teste, portanto em operação normal `test_mode` deve ser "0". A modificação melhora em muito as condições de teste do bloco, evidentemente a testabilidade não chega a 100% pois, da forma que ficou, o bloco `pscalerv` não é observável. O sinal `clk2` poderia ser levado a uma saída, entretanto concluiu-se que o *overhead* produzido não justificaria a cobertura.

Os sinais adicionais introduzidos na modificação da figura 8, obviamente, precisaram ser encaminhados para fora do módulo e hierarquicamente para fora do circuito. Portanto, foi necessário acrescentar pinos extras no módulo `fifotu12_out`, no seu superior imediato o módulo `vc12dm` e na *topcell* o módulo `tu12`. Esta foi a única modificação do módulo `vc12dm`.

O módulo `tu12`, por sua vez, apresentou uma série de problemas. A primeira diz respeito à uniformidade dos elementos seqüenciais. Para se consolidar uma cadeia de *scan* é necessário que todos os elementos seqüenciais sejam compatíveis, ou seja, não é desejável que se misture *flip-flops* sensíveis a borda com *latches* sensíveis a nível. Algumas ve-

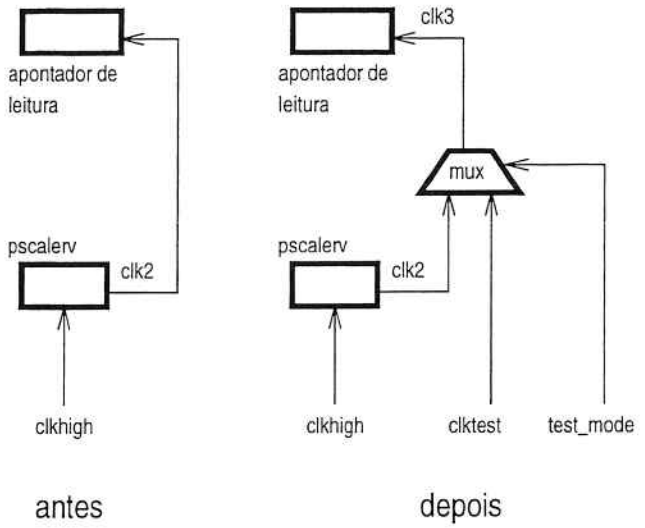

Fig. 8. Solução utilizada no bloco fifotu12_out.

zes *latches* são inferidos indesejavelmente, outras, são utilizados indevidamente. Tem-se os dois exemplos no módulo tu12: na primeira, é implementado um interface para controlador semelhante ao "8051", onde dados e endereços são multiplexados num mesmo barramento. Na fase de endereçamento, os endereços são armazenados internamente num registro especial. Foi utilizado inicialmente, um *latch* de 8 bits para esta função. A correção imediata foi a substituição do *latch* por elementos síncronos, conforme pode ser visto no código VHDL a seguir:

```
... <<antes>>
  -- Latch de enderecos
  adr_latch : PROCESS (clk, ale, csl, ad)
  BEGIN
    IF (ale = '1' and csl = '0') THEN
  madr(0) <= ad(0);
  madr(1) <= ad(1);
... <<depois>>
  -- Latch de enderecos
  adr_latch : PROCESS (clk)
  BEGIN
    IF rising_edge(clk) THEN
      IF (ale = '1' and csl = '0') THEN
madr(0) <= ad(0);
madr(1) <= ad(1);
...
```

O segundo caso, o de inferência indevida, foi detetado na implementação de um multiplexador de acesso a registros internos. Sendo o barramento de 8 bits, e alguns registros de 7 bits, um *latch* indevido foi inferido quando era feita a atribuição de valores de bits ao sinal de saída. Este é um daqueles erros que não alteram a funcionalidade do circuito, porém atrapalham em muito os esquemas de testabilidade. Uma vez detetado o problema a solução é trivial.

Um outro problema encontrado refere-se à inferência de *tristate*. Parece mais uma vez que os esquemas de inferência não são rigidamente idênticos entre a síntese Mentor Graphics e Synopsys. Foi preciso reescrever a implementação do *tristate* mesmo porque a geração dos vetores de teste é muito sensível aos barramentos bidirecionais. Pensando nisso concluiu-se que a interface do tipo "8051" seria problemática e foi decidido adaptar para uma do tipo "68HC11". A diferença que norteou esta decisão refere-se aos sinais de *read* e *write* na interface tipo "8051". Na geração automática de teste os sinais recebem, numa primeira instância, atribuição aleatória de valores. Isto pode fazer com que o estado do barramento bidirecional seja indeterminado quando a atribuição não corresponder nem a *read*, nem a *write*. Por outro lado, se atribuição ativar ambos os sinais, pode-se ter contenção no barramento. O modelo "68HC11" usa por sua vez um único sinal para sinalizar *read* e *write*. O sentido da transação é determinada em todo o ciclo de acesso, e a temporização ditada por um sinal de *strobe* separado. Esta forma de operação mostrou-se mais conveniente como poderá ser vista nos resultados seguintes.

```
dc_shell> report_area
****************************************
Report : area
Design : tu12
Version: 1999.05
Date   : Wed Dec  8 16:23:16 1999
****************************************

Library(s) Used:

    cub33 (File: ~fuzzy/bibliotecas/synopsys/synopsys/cub33/cub33.db)

Number of ports:                53
Number of nets:                377
Number of cells:               246
Number of references:           43

Combinational area:       1495.089966
Noncombinational area:    1311.549927
Net Interconnect area:    5406.796387

Total cell area:          2806.639893
Total area:               8213.436523

dc_shell> set_test_hold 1 test_mode
Performing set_test_hold on port 'test_mode'.
dc_shell> set_test_hold 0 csl
Performing set_test_hold on port 'csl'.
dc_shell> create_test_patterns
...
  Combinational Test Pattern Generation starts:

    Start random pattern generation...


    ...End random pattern generation

    Start deterministic pattern generation...

       48.24% faults processed ; cumulative fault coverage =  48.18%
       56.12% faults processed ; cumulative fault coverage =  56.06%
       58.19% faults processed ; cumulative fault coverage =  58.14%
       65.62% faults processed ; cumulative fault coverage =  65.58%
       77.87% faults processed ; cumulative fault coverage =  77.82%
       78.61% faults processed ; cumulative fault coverage =  78.56%
       79.03% faults processed ; cumulative fault coverage =  78.98%
       79.34% faults processed ; cumulative fault coverage =  79.29%
       79.52% faults processed ; cumulative fault coverage =  79.47%
       79.73% faults processed ; cumulative fault coverage =  79.68%
       79.81% faults processed ; cumulative fault coverage =  79.76%
       79.98% faults processed ; cumulative fault coverage =  79.93%
       80.19% faults processed ; cumulative fault coverage =  80.14%
       80.34% faults processed ; cumulative fault coverage =  80.29%
       80.39% faults processed ; cumulative fault coverage =  80.34%
       81.82% faults processed ; cumulative fault coverage =  81.77%
       84.30% faults processed ; cumulative fault coverage =  84.21%
       85.88% faults processed ; cumulative fault coverage =  85.78%
       87.07% faults processed ; cumulative fault coverage =  86.98%
       87.93% faults processed ; cumulative fault coverage =  87.84%
       89.42% faults processed ; cumulative fault coverage =  89.32%
       90.09% faults processed ; cumulative fault coverage =  89.97%
       90.84% faults processed ; cumulative fault coverage =  90.71%
       91.33% faults processed ; cumulative fault coverage =  91.19%
       92.04% faults processed ; cumulative fault coverage =  91.77%
       92.79% faults processed ; cumulative fault coverage =  92.51%
       93.70% faults processed ; cumulative fault coverage =  93.38%
       94.19% faults processed ; cumulative fault coverage =  93.85%
       94.54% faults processed ; cumulative fault coverage =  94.16%
       94.72% faults processed ; cumulative fault coverage =  94.34%
       95.12% faults processed ; cumulative fault coverage =  94.75%
       96.27% faults processed ; cumulative fault coverage =  95.90%
       96.53% faults processed ; cumulative fault coverage =  96.16%
```

```
    96.79% faults processed ; cumulative fault coverage =  96.42%
    97.36% faults processed ; cumulative fault coverage =  96.99%
    97.65% faults processed ; cumulative fault coverage =  97.28%
    97.85% faults processed ; cumulative fault coverage =  97.47%
    98.15% faults processed ; cumulative fault coverage =  97.77%
    98.44% faults processed ; cumulative fault coverage =  97.98%
    98.62% faults processed ; cumulative fault coverage =  98.15%
    98.88% faults processed ; cumulative fault coverage =  98.37%
    99.11% faults processed ; cumulative fault coverage =  98.61%
    99.34% faults processed ; cumulative fault coverage =  98.83%
    99.57% faults processed ; cumulative fault coverage =  99.06%
    99.75% faults processed ; cumulative fault coverage =  99.24%
    99.98% faults processed ; cumulative fault coverage =  99.40%
    99.98% faults processed ; cumulative fault coverage =  99.40%

...End deterministic pattern generation


                          Non-collapsed     Collapsed
No. of detected faults    27414             18410
No. of abandoned faults   42                22
No. of tied faults        6                 6
No. of redundant faults   0                 0
No. of untested faults    114               90
Total no. of faults       27576             18528
Fault coverage            99.43             99.40

No. of test patterns       647

Test Generation Time (CPU)  50.80 sec

Start compaction...

...End compaction

No. of compacted patterns    591

Compaction Time (CPU)       7.78 sec
...Writing test program tu12 to file
          ~autel/Synopsys/tu12/xwork/begin/tu12.vdb
```

Note que antes da geração dos vetores de teste, foi utilizado um novo comando "*set_test_hold*". Este comando "seta" um determinado sinal num valor fixo durante o teste. O sinal `test_mode` foi colocado em nível "1", indicando teste interno, e o sinal `csl` (*chip select*) foi colocado em "0", indicando componente selecionado. Algumas informações interessantes podem ser obtidas:

```
dc_shell> report_test -scan_path
****************************************
Report : test
         -scan_path
Design : tu12
Version: 1999.05
Date   : Wed Dec  8 16:32:07 1999
****************************************

  (*) indicates change of polarity in scan data
  (c) indicates cell is scan controllable only
  (o) indicates cell is scan observable only
  (x) indicates cell cannot capture

Complete scan chain #1 (test_si1 --> tu12_out(7)) contains 650 cells:
... (lista de sinais)
Complete scan chain #2 (test_si2 --> test_so2) contains 8 cells:
... (lista de sinais)
Complete scan chain #3 (test_si3 --> test_so3) contains 6 cells:
... (lista de sinais)
Complete scan chain #4 (test_si4 --> test_so4) contains 70 cells:
... (lista de sinais)
```

Veja que este bloco apresenta quatro cadeias de *scan*. Isto acontece por que o circuito utiliza quatro *clocks* diferentes, são eles: `clk`, `clkin`, `clkhigh` e o recém criado `clktest`.

V. *Boundary-Scan*

O processo de inserção de *Boundary-Scan* é independente da inserção de cadeias de *scan* internas ao circuito. Para sintetizar a lógica de *Boundary-Scan*, basta executar o comando `insert_jtag`. Como resultado, o sistema será reestruturado da seguinte forma:

- A antiga `topcell` será renomeada para `Core`.
- Será criado o bloco `JTAG_BR`, contendo o registrador de *bypass*.
- Serão criados os blocos `JTAG_BSRINBOTH`, que serão usados como os registros de *boundary-scan* para todas as entradas não *clock*.
- Serão criados os blocos `JTAG_BSRINCLKOBS`, que serão usados como os registros de *boundary-scan* para todas as entradas do tipo *clock*.
- Serão criados os blocos `JTAG_BSROUTBOTH`, que serão usados como os registros de *boundary-scan* para todas as saídas.
- Será criado o bloco `JTAG_IR2`, contendo um registro de instrução de dois bits[7].
- Será criado o bloco `JTAG_TAP`, contendo o controlador `TAP`, com *reset* assíncrono.

A. *Boundary-Scan no Mapeador/Demapeador*

Acompanhe os comando principais do processo:

```
dc_shell> insert_scan
...
dc_shell> report_area
...
****************************************
Report : area
Design : tu12
Version: 1999.05
Date   : Fri Dec 10 18:42:03 1999
****************************************

Library(s) Used:

    cub33 (File: ~fuzzy/bibliotecas/synopsys/synopsys/cub33/cub33.db)

Number of ports:                 58
Number of nets:                 216
Number of cells:                 82
Number of references:            15

Combinational area:       1602.220215
Noncombinational area:    1482.209961
Net Interconnect area:    5815.416016

Total cell area:          3084.430176
Total area:               8899.845703
...
dc_shell> report_test -port

****************************************
Report : test
         -port
Design : tu12
Version: 1999.05
Date   : Fri Dec 10 18:43:35 1999
****************************************

                                                          Test Clocks
                                                    ---------------------
Port            Signal Type                  Index  Rise   Fall  Period
--------------- ---------------------------- -----  ------ ------ ------
jtag_tck        jtag_tck
jtag_tdi        jtag_tdi
jtag_tdo        jtag_tdo
jtag_tms        jtag_tms
jtag_trst       jtag_trst
test_se         test_scan_enable
test_si1        test_scan_in                    1
test_si2        test_scan_in                    2
test_si3        test_scan_in                    3
test_si4        test_scan_in                    4
test_so2        test_scan_out                   2
test_so3        test_scan_out                   3
test_so4        test_scan_out                   4
tu12_out(7)     test_scan_out                   1
```

[7] É possível definir-se novas instruções, além das exigidas pela norma IEEE 1149.1, como veremos mais adiante.

```
                ----- Port ------ ---------- Associated BSR Cell(s) ----------
                In   JTAG   Data  -- Routing --- ------ Implementation -------
Port            BSR  Mode   Type   In  Out  Ctl  (  In / Out / Ctl  )
--------------- --- ------- ----- ---- ---- ---- ----------------------------
LOS             Yes both    Data    1    -    - (JTAG_BSRINBOTH / - / - )
ad(0)           Yes both    Data   46   30   32 (JTAG_BSRINBOTH / JTAG_BSROUTBOTH / JTAG_BSRCTL)
ad(1)           Yes both    Data   45   31   32 (JTAG_BSRINBOTH / JTAG_BSROUTBOTH / JTAG_BSRCTL)
ad(2)           Yes both    Data   44   33   39 (JTAG_BSRINBOTH / JTAG_BSROUTBOTH / JTAG_BSRCTL)
ad(3)           Yes both    Data   43   34   39 (JTAG_BSRINBOTH / JTAG_BSROUTBOTH / JTAG_BSRCTL)
ad(4)           Yes both    Data   42   35   39 (JTAG_BSRINBOTH / JTAG_BSROUTBOTH / JTAG_BSRCTL)
ad(5)           Yes both    Data   41   36   39 (JTAG_BSRINBOTH / JTAG_BSROUTBOTH / JTAG_BSRCTL)
ad(6)           Yes both    Data   40   37   39 (JTAG_BSRINBOTH / JTAG_BSROUTBOTH / JTAG_BSRCTL)
ad(7)           Yes both    Data   47   38   39 (JTAG_BSRINBOTH / JTAG_BSROUTBOTH / JTAG_BSRCTL)
ale             Yes both    Data    2    -    - (JTAG_BSRINBOTH / - / - )
byte_sync       Yes both    Data    -   48    - ( - / JTAG_BSROUTBOTH / - )
clk             Yes both    Data    3    -    - (JTAG_BSRINBOTH / - / - )
clkhigh         Yes both    Data    4    -    - (JTAG_BSRINBOTH / - / - )
clkin           Yes both    Data    5    -    - (JTAG_BSRINBOTH / - / - )
clkout          Yes both    Data    -   49    - ( - / JTAG_BSROUTBOTH / - )
clktest         Yes both    Data    6    -    - (JTAG_BSRINBOTH / - / - )
csl             Yes both    Data    7    -    - (JTAG_BSRINBOTH / - / - )
enable_lb       Yes both    Data    8    -    - (JTAG_BSRINBOTH / - / - )
frame_sync      Yes both    Data    -   50    - ( - / JTAG_BSROUTBOTH / - )
intr            Yes both    Data    -   51    - ( - / JTAG_BSROUTBOTH / - )
jtag_tck        No  -       -       -    -    -  -
jtag_tdi        No  -       -       -    -    -  -
jtag_tdo        No  -       -       -    -    -  -
jtag_tms        No  -       -       -    -    -  -
jtag_trst       No  -       -       -    -    -  -
nrzin           Yes both    Data    9    -    - (JTAG_BSRINBOTH / - / - )
nrzout          Yes both    Data    -   52    - ( - / JTAG_BSROUTBOTH / - )
req_in          Yes both    Data   10    -    - (JTAG_BSRINBOTH / - / - )
req_out         Yes both    Data   11    -    - (JTAG_BSRINBOTH / - / - )
rst             Yes both    Data   12    -    - (JTAG_BSRINBOTH / - / - )
rwl             Yes both    Data   13    -    - (JTAG_BSRINBOTH / - / - )
test_mode       Yes both    Data   14    -    - (JTAG_BSRINBOTH / - / - )
test_se         Yes both    Data   15    -    - (JTAG_BSRINBOTH / - / - )
test_si1        Yes both    Data   16    -    - (JTAG_BSRINBOTH / - / - )
test_si2        Yes both    Data   17    -    - (JTAG_BSRINBOTH / - / - )
test_si3        Yes both    Data   18    -    - (JTAG_BSRINBOTH / - / - )
test_si4        Yes both    Data   19    -    - (JTAG_BSRINBOTH / - / - )
test_so2        Yes both    Data    -   53    - ( - / JTAG_BSROUTBOTH / - )
test_so3        Yes both    Data    -   54    - ( - / JTAG_BSROUTBOTH / - )
test_so4        Yes both    Data    -   55    - ( - / JTAG_BSROUTBOTH / - )
tu12_in(0)      Yes both    Data   20    -    - (JTAG_BSRINBOTH / - / - )
tu12_in(1)      Yes both    Data   21    -    - (JTAG_BSRINBOTH / - / - )
tu12_in(2)      Yes both    Data   22    -    - (JTAG_BSRINBOTH / - / - )
tu12_in(3)      Yes both    Data   23    -    - (JTAG_BSRINBOTH / - / - )
tu12_in(4)      Yes both    Data   24    -    - (JTAG_BSRINBOTH / - / - )
tu12_in(5)      Yes both    Data   25    -    - (JTAG_BSRINBOTH / - / - )
tu12_in(6)      Yes both    Data   26    -    - (JTAG_BSRINBOTH / - / - )
tu12_in(7)      Yes both    Data   27    -    - (JTAG_BSRINBOTH / - / - )
tu12_out(0)     Yes both    Data    -   56    - ( - / JTAG_BSROUTBOTH / - )
tu12_out(1)     Yes both    Data    -   57    - ( - / JTAG_BSROUTBOTH / - )
tu12_out(2)     Yes both    Data    -   58    - ( - / JTAG_BSROUTBOTH / - )
tu12_out(3)     Yes both    Data    -   59    - ( - / JTAG_BSROUTBOTH / - )
tu12_out(4)     Yes both    Data    -   60    - ( - / JTAG_BSROUTBOTH / - )
tu12_out(5)     Yes both    Data    -   61    - ( - / JTAG_BSROUTBOTH / - )
tu12_out(6)     Yes both    Data    -   62    - ( - / JTAG_BSROUTBOTH / - )
tu12_out(7)     Yes both    Data    -   63    - ( - / JTAG_BSROUTBOTH / - )
v1_sync_in      Yes both    Data   28    -    - (JTAG_BSRINBOTH / - / - )
v1_sync_out     Yes both    Data   29    -    - (JTAG_BSRINBOTH / - / - )

                                TOTAL: 63 BSR cells
```

```
dc_shell> report_test -jtag
```

```
****************************************
Report : test
         -jtag
Design : tu12
Version: 1999.05
Date   : Fri Dec 10 18:44:47 1999
****************************************

   JTAG Logic Inserted:                          Yes
   Pads Inserted:                                No
   Asynchronous TAP Reset:                       Yes
   Gated clockDR Signal:                         No
   BSR Cell Port Drive Limit:                    6
   Instruction Register Bit Width:               2
   Total Number of JTAG instructions:            3
     Number of 1149.1-Specified Instructions:    3
     Number of User-Specified Instructions:      0

   Device Identification Register:               No

        TAP Port       Port Name
      ---------------------------------------
          TCK          jtag_tck
          TDI          jtag_tdi
          TDO          jtag_tdo
          TMS          jtag_tms
          TRST         jtag_trst

       Instruction       Code
      --------------------------
       BYPASS            11
       EXTEST            00
       SAMPLE/PRELOAD    01

                  Target            Capture       Load        Enable
 Instruction      Register            Pin          Pin          Pin
-------------------------------------------------------------------------------
BYPASS           JTAG_BYPASS_REG    ---           ---         ---
EXTEST           JTAG_BSR           ---           ---         ---
SAMPLE/PRELOAD   JTAG_BSR           ---           ---         ---
```

```
dc_shell> check_bsd
```

```
  Loading design 'tu12'
...Starting IEEE 1149.1 Compliance Checking.
...Finding set of sequential elements.
...Analyzing TAP and TAP Controller.
.....Analyzing TAP.
.....Finding the set of TAP controller sequential elements.
.....Pruning set of TAP Controller sequential Elements
...Checking the TAP controller initialization.
...Analyzing the TAP controller reset condition.
...Traversing the TAP controller states.
...Inferring TAP controller clock outputs.
...Analyzing TRST port.
Warning: Undriven input port TRST is floating. When undriven,
     this port should behave as though it was
     driven by logic one. (TEST-819)
...Analyzing TMS Port.
Warning: Undriven input port TMS is floating. When undriven,
     this port should behave as though it was
     driven by logic one. (TEST-819)
...Analyzing TCK halt state.
Warning: Undriven input port TDI is floating. When undriven,
     this port should behave as though it was
     driven by logic one. (TEST-819)
...Analyzing the instruction register.
.....Finding the update flops.
...Analyzing the BYPASS register.
...Analyzing the DIR register.
Device Identification Register doesn't exists
...Analyzing the boundary scan register.
.....Finding the update flops.
.....Finding the BSR cells controlling the design ports.
.....Finding the BSR cells sensing the design ports.
.....Finding the BSR cells driving the design ports.
Warning: Logic cannot exist between boundary scan cell
     U488/SHADOW and design port ad(7). (TEST-843)
Information: There are 7 other ports with the same violation. (TEST-299)
...Finding the set of IR decoding pins.
...Analyzing the different signatures at the decoding pins.
...Inferring the different test data register (TDR)s selected by instructions.
...Checking output conditioning of the implemented instructions.
...Inferring the implemented SAMPLE/PRELOAD instructions.
...Finding the BSR controlling cells PIs.
...Finding the BSR controlling cells POs.
...Finding the BSR non-controlling cells PIs.
...Finding the BSR cells POs.
...Modifying the BSR controlling/Output cells PIs.
...Inferring the implemented INTEST instruction.
...Inferring the implemented CLAMP instruction.
...Inferring the implemented HIGHZ instruction.
...Inferring the implemented IDCODE & USERCODE instruction.
...Inferring the implemented RUNBIST instruction.
...Analyzing the EXTEST instruction.
...Analyzing the BYPASS instruction.
...Analyzing the SAMPLE/PRELOAD instruction.
...Analyzing the INTEST instruction.
...Analyzing the CLAMP instruction.
...Analyzing the RUNBIST instruction.
...Analyzing the IDCODE and USERCODE instructions.
...Analyzing Test-Logic-Reset Tap Controller State.
...Finished IEEE 1149.1 Compliance Checking.

*****************************************************

          IEEE 1149.1 Summary

*****************************************************
  4 state elements found in the TAP controller
  2 cells found in the Instruction Register
  3 standard instructions found.
  0 user defined instructions found.
  1 cells in BYPASS register
  63 cells in boundary scan register

*****************************************************

          IEEE 1149.1 Violation Summary

*****************************************************
  2 Violations found in extraction of TAP Controller
Violates Rule: 3.3.1b Corresponds to Errors: TEST-819
Violates Rule: 3.6.1b Corresponds to Errors: TEST-819
  1 Boundary scan design Violations found
Violates Rule: 10.4.1e Corresponds to Errors: TEST-843
```

Como pôde ser visto o comando `report_test -port` provê informações sobre os *ports* do circuito e quais estão utilizando células de *boundary-scan*. O comando `report_test -jtag` informa as opções de síntese do BS, e as instruções implementadas. Finalmente o comando `check_bsd`, verifica se o projeto está de acordo com a especificação IEEE 1149.1. Note que foram encontradas três violações: as duas primeiras ocorrem por não existir resistores de *pull-up* nos *ports* do controlador TAP, a última ocorre por estar-se utilizando barramentos bidirecionais, e células de *tristate* foram colocadas entre os blocos BSR e os pinos de saída.

B. Boundary-Scan no Realinhador

Ao contrário do Mapeador/Demapeador, o Realinhador possui apenas uma cadeia de *scan*. É possível acessá-la através do controlador de *Boundary-Scan*, definindo uma

instrução adicional ao que normalmente é sintetizado. Isto é possível executando o comando:

```
dc_shell> set_jtag_instruction instruction_name [-code code]
```

Antes de invocar a síntese BS, por exemplo, executando o comando:

```
dc_shell> set_jtag_instruction INTSCAN
```

adiciona-se uma nova instrução de nome `INTSCAN`. Executando o comando `insert_jtag`, o *Test Compiler* irá adicionar o controle necessário para a decodificação e a lógica de controle da cadeia de *scan* e então criará dois *pseudo-ports*, `INTSCANCAPTURE` de entrada, e `INTSCANENABLE` de saída. A ligação entre estes *pseudo-ports* e a cadeia interna de *scan* deve ser feita manualmente.

- O sinal `INTSCANCAPTURE` deve ser ligado no *port* `test_so` do bloco `Core`.
- O sinal `INTSCANENABLE` deve ser ligado ao `test_se` do bloco `Core`.
- E o sinal `TDI` deve ser ligado ao `test_si` do `Core`.

Obviamente, alguma lógica adicional precisará ser inserida. Aplicando ao realinhador, temos:

```
dc_shell> set_jtag_instruction INTSCAN
dc_shell> insert_jtag
dc-shell> current_design realin
dc_shell> write -format vhdl -hierarchy -output ''realin_bsd.vhd''
```

Como o trabalho é todo feito em VHDL, a hierarquia em VHDL é "salva" para efetuar as modificações necessárias. Veja na figura 9 as modificações acrescentadas.

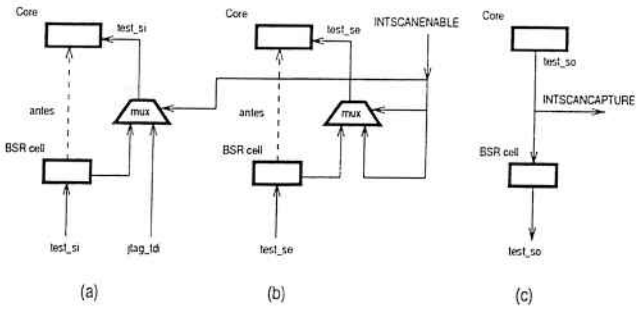

Fig. 9. Modificações necessárias para se ter acesso à cadeia de *scan* interna, através do controlador de *Boundary-scan*.

```
dc_shell> read -f vhdl realin_bsd_intscan.vhd
dc_shell> uniquify
dc_shell> compile -map_effort medium -verify -verify_effort low
    -boundary_optimization
dc_shell> create_schematic -size mentor_maximum    -gen_database
dc_shell> create_schematic -size mentor_maximum  -symbol_view
dc_shell> create_schematic -size mentor_maximum -schematic_view
dc_shell> report_area
...
****************************************
Report : area
Design : realin
Version: 1999.05
Date   : Mon Dec 13 14:12:23 1999
****************************************

Library(s) Used:

    cub33 (File: ~fuzzy/bibliotecas/synopsys/synopsys/cub33/cub33.db)

Number of ports:                30
Number of nets:                106
Number of cells:                43
Number of references:           38

Combinational area:         519.519836
Noncombinational area:      409.450043
Net Interconnect area:     1712.814941

Total cell area:            928.969849
Total area:                2641.784668
...
dc_shell> check_bsd

  Loading design 'Core'
...Starting IEEE 1149.1 Compliance Checking.
...Finding set of sequential elements.
...Analyzing TAP and TAP Controller.
......Analyzing TAP.
Error: NO IEEE 1149.1 test ports have been identified. (TEST-812)
...Finished IEEE 1149.1 Compliance Checking.

************************************************

          IEEE 1149.1 Summary

************************************************
  0 cells found in the Instruction Register
  0 standard instructions found.
  0 user defined instructions found.
  NO TEST DATA  REGISTER
  No boundary scan register

************************************************

          IEEE 1149.1 Violation Summary

************************************************
  2 Violations found in extraction of TAP Controller
Violates Rule: 3.1.1a Corresponds to Errors: TEST-812
Violates Rule: 4.1.1a.i Corresponds to Errors: TEST-812
0
design_analyzer> current_design "~autel/Synopsys/realinhador/xwork/begin/realin.db:realin"
Current design is 'realin'.
{"realin"}
design_analyzer> check_bsd
  Loading design 'realin'
...Starting IEEE 1149.1 Compliance Checking.
...Finding set of sequential elements.
...Analyzing TAP and TAP Controller.
......Analyzing TAP.
......Finding the set of TAP controller sequential elements.
......Pruning set of TAP Controller sequential Elements
...Checking the TAP controller initialization.
...Analyzing the TAP controller reset condition.
...Traversing the TAP controller states.
...Inferring TAP controller clock outputs.
...Analyzing TRST port.
Warning: Undriven input port TRST is floating. When undriven,
     this port should behave as though it was driven by logic one. (TEST-819)
...Analyzing TMS Port.
Warning: Undriven input port TMS is floating. When undriven,
     this port should behave as though it was driven by logic one. (TEST-819)
...Analyzing TCK halt state.
Warning: Undriven input port TDI is floating. When undriven,
     this port should behave as though it was driven by logic one. (TEST-819)
...Analyzing the instruction register.
......Finding the update flops.
...Analyzing the BYPASS register.
...Analyzing the DIR register.
Device Identification Register doesn't exists
...Analyzing the boundary scan register.
......Finding the update flops.
......Finding the BSR cells controlling the design ports.
......Finding the BSR cells sensing the design ports.
......Finding the BSR cells driving the design ports.
...Finding the set of IR decoding pins.
...Analyzing the different signatures at the decoding pins.
...Inferring the different test data register (TDR)s selected by instructions.
Warning: No Register is selected to connect between
     TDI and TDO during instruction opcode 10. (TEST-837)
...Checking output conditioning of the implemented instructions.
...Inferring the implemented SAMPLE/PRELOAD instructions.
...Finding the BSR controlling cells PIs.
...Finding the BSR controlling cells POs.
...Finding the BSR non-controlling cells PIs.
...Finding the BSR cells POs.
...Modifying the BSR controlling/Output cells PIs.
...Inferring the implemented INTEST instruction.
...Inferring the implemented CLAMP instruction.
...Inferring the implemented HIGHZ instruction.
...Inferring the implemented IDCODE & USERCODE instruction.
...Inferring the implemented RUNBIST instruction.
...Analyzing the EXTEST instruction.
...Analyzing the BYPASS instruction.
...Analyzing the SAMPLE/PRELOAD instruction.
...Analyzing the INTEST instruction.
...Analyzing the CLAMP instruction.
...Analyzing the RUNBIST instruction.
...Analyzing the IDCODE and USERCODE instructions.
...Analyzing Test-Logic-Reset Tap Controller State.
...Finished IEEE 1149.1 Compliance Checking.

************************************************

          IEEE 1149.1 Summary

************************************************
  4 state elements found in the TAP controller
  2 cells found in the Instruction Register
  3 standard instructions found.
  0 user defined instructions found.
  1 cells in BYPASS register
  25 cells in boundary scan register

************************************************
```

```
IEEE 1149.1 Violation Summary

**************************************************
  2 Violations found in extraction of TAP Controller
Violates Rule: 3.3.1b Corresponds to Errors: TEST-819
Violates Rule: 3.6.1b Corresponds to Errors: TEST-819
  2 Boundary scan design Violations found
Violates Rule: 7.1.1a Corresponds to Errors: TEST-837
Violates Rule: 7.1.1c Corresponds to Errors: TEST-837
```

O circuito pôde então ser recarregado e reotimizado. Vê-se na figura 10 que a modificação apresentada na figura 9 foi implementada e simplificada corretamente. O multiplexador de `test_si` foi preservado e mapeado no componente `MU2`, enquanto o multiplexador de `test_so` foi simplificado e substituído pelo componente `A021`.

Neste ponto, o procedimento parece ter sido completado com sucesso, entretanto quando se procede às verificações costumeiras nota-se um grande número violações. As violações são decorrentes do fato do sistema não ser capaz de reconhecer os elementos do *boundary-scan* inseridos automaticamente no passo anterior. O *netlist* gerado nos processos de síntese automático são armazenados numa base de dados, que entre outras coisas possui informações de quais elementos foram inseridos automaticamente e quais as suas finalidades. As cadeias de *scan* e os blocos do *boundary-scan* são rápidamente destacáveis nesta etapa. Para se efetuar as modificações necessárias para a utilização da instrução adicional, foi necessário a conversão da base de dados interna da ferramenta Synopsys num formato que pudesse ser manipulado, neste caso a linguagem VHDL. Obviamente, estas propriedades úteis de *ports*, *nets* e *gates* foram perdidas, provocando o elevado número de violações observado quando o *netlist* foi novamente reintroduzido na ferramenta.

Apesar dos problemas, o circuito foi sintetizado corretamente. A dificuldade entretanto consiste na elaboração dos vetores de teste, uma vez que a ferramenta não permite acoplar os testes gerados nas etapas anteriores ao circuito com *boundary-scan* interligado à cadeia de *scan* interna.

VI. Conclusões

Foi visto nestes exemplos, que não basta que um projeto seja concebido para síntese automática, para que a inserção automática de circuitos de teste dê bons resultados. Foi constatado, e já era de se esperar, que a natureza do projeto pode torná-lo mais fácil ou mais difícil de ser testado. O projeto do circuito realinhador, por exemplo, é um circuito todo síncrono, usa um único sinal de *clock*. Portanto tudo indica que a inserção de uma cadeia de *scan* pode resultar numa cobertura extremamente interessante, como acabou se confirmando. O mapeador/demapeador, por outro lado, possui vários sinais de *clock*, *ports* bidirecionais, além de ser um circuito mais complexo, exigindo muito mais trabalho. Mesmo assim foi possível conseguir uma cobertura muito interessante. Um resumo dos resultados deste exemplo podem ser vistos na tabela I.

Na tabela II foram calculados os *overheads* das implementações, tomando-se como base o circuito já com todas as modificações de testabilidade, porém sintetizado sem as estruturas de DFT.

Note que o impacto da inserção das cadeias de *scan* é mínimo, mesmo considerando circuitos pequenos como o Realinhador. Curiosamente, o *overhead* foi negativo no caso do mapeador/demapeador, talvez devido aos diferentes algoritmos usados nos dois casos. Portanto, pode se concluir que a ferramenta é capaz de produzir resultados satisfatórios a custos relativamente baixos.

Em relação ao *boundary-scan*, entretanto, a ferramenta apresentou-se muito mais limitada. Como já era esperado, a inserção dos circuitos de *boundary-scan*, apresentou *overhead* tanto maior, quanto menor o circuito. Entretanto, as limitações apareceram quanto tentou-se acessar a cadeia de *scan* interna, através da interface de *boundary-scan*. O procedimento descrito nos manuais é pouco detalhado, limita-se a apresentar os passos seguidos na seção anterior, sem fazer referência de como fazer para recuperar as informações do *scan* interno. Isto torna o procedimento difícil e trabalhoso. Outra limitação bastante incômoda é que a ferramenta não permite que seja escolhida a interface de *boundary-scan* como meio de injeção dos vetores de teste internos. Para isso, a ferramenta sugere a geração de uma descrição adequada dos circuitos de *boundary-scan*, através da linguagem BSDL, e em seguida, através de ferramentas *third-part* formatar os vetores apropriadamente.

Neste aspecto, pode-se concluir que, a menos que tenhamos disponíveis estas ferramentas de formatação de vetores utilizando as estruturas de *boundary-scan*, o procedimento é incompleto, e portanto inútil para efeitos práticos.

Como conclusão do trabalho, deve-se lembrar que um projeto bem sucedido no aspecto testabilidade, além de se

TABLE I
Resumo dos resultados

	realin	tu12	cob.
Síntese simples, sem DFT			
área	2217,67	8118,69	-
nº de portas	1933,70	6960,00	-
Síntese simples, após as modificações			
área	2206,84	8309,23	-
nº de portas	1924,02	7119,12	-
Após o *scan*			
área	2258,47	8213,44	100%
nº de portas	1862,76	6845,46	99,40%
Após o *boundary-scan*			
área	2641,78	8899,84	-
nº de portas	2265,78	7523,00	-

TABLE II
Overhead das implementações

overhead	realin	tu12
Após o *scan*		
área	2,3%	-1,15%
nº de portas	-3,19%	-3,84%
Após o *boundary-scan*		
área	19,71%	7,11%
nº de portas	17,76%	5,67%

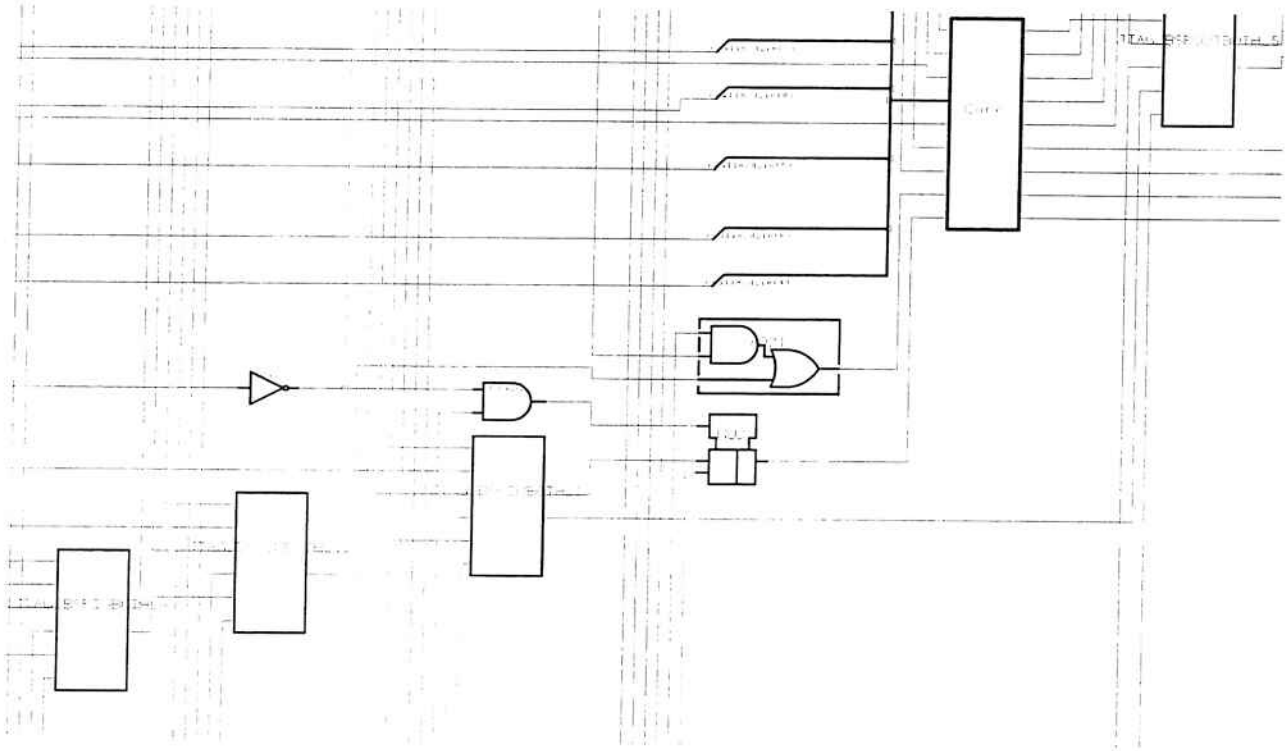

Fig. 10. Modificação após a otimização.

adequar a todos os aspectos da especificação tanto funcionais quanto econômicos, é necessário que o projetista tome especial cuidado quanto aos seguintes aspectos:

- O *Test Compiler*, ao gerar os vetores de teste, tenta testar *ports* bidirecionais em ambas as direções. No caso da geração aleatória de vetores, primeira fase da geração, é possível que os sinais habilitantes sejam "setados" em uma condição contraditória, causando contenção. Por outro lado, se não for possível para o gerador o teste nos dois sentidos, o nó pode acabar sendo considerado como inalcançável, prejudicando a cobertura. Sinais habilitantes gerados em componentes seqüenciais podem também causar contenção, se estes elementos pertencerem ao *scan* de teste.
- Muito cuidado deve ser tomado na codificação do projeto. *Latches* podem ser gerados inadvertidamente, e por serem considerados como elementos seqüenciais podem ter um efeito indesejado sobre a cobertura final. Caso o uso seja inevitável, existem meios de aumentar a cobertura utilizando estes componentes, mas em favor da simplicidade é recomendável evitá-los.
- Quando o uso de múltiplos *clocks* é inevitável, e isto inclui também os *gated-clocks*, deve-se observar o compromisso entre a minimização desejada e a dificuldade na geração dos vetores de teste. Viu-se num dos exemplos que foi necessário adicionar estruturas de teste para que a cobertura pudesse ser aumentada.
- Problema semelhante aos sinais bidirecionais: quando se usa *tristate* deve-se ter em mente a possibilidade de algum sinal tornar-se inalcançável durante a geração dos vetores de teste. A ocorrência de condições de alta impedância também provoca condições de imprevisibilidade, inviabilizando o teste. Em alguns casos resistores de *pull-up/pull-down* são recomendados.
- É preciso considerar, que em alguns casos, o projeto deve ser adaptado para melhor responder às condições de teste. Quando considerada esta possibilidade nas etapas iniciais do projeto é possível incorporar estas modificações com um mínimo de *overhead* para o circuito e para o teste propriamente dito.

References

[1] Douglas J. Smith, *HDL Chip Design: A practical guide for designing, synthesizing and simulating ASICs and FPGAs using VHDL or Verilog*, Doone Publications, 1997.

[2] Synopsys Inc., "Synopsys home page," http://www.synopsys.com/.

[3] Sistema de Documentação Telebrás, "Especificações gerais de sistemas SDH," out 1995, 225-100-722 (padrão).

[4] CCITT, *Características Físicas y Eléctricas de los Interfaces Digitales Jerárquicos.*

[5] Mentor Graphics Inc, "Eda software from mentor graphics," http://www.mentorg.com/.
[6] Synopsys Inc., *Scan Synthesis User Guide*, 1999, online documentation.
[7] Synopsys Inc., *Test Compiler Reference Manual*, 1999, online documentation.
[8] Synopsys Inc., *Design Analyzer Reference Manual*, 1999, online documentation.

BOLETINS TÉCNICOS - TEXTOS PUBLICADOS

BT/PTC/9901 – Avaliação de Ergoespirômetros Segundo a Norma NBR IEC 601-1- MARIA RUTH C. R. LEITE, JOSÉ CARLOS TEIXEIRA DE B. MORAES

BT/PTC/9902 – Sistemas de Criptofonia de Voz com Mapas Caóticos e Redes Neurais Artificiais – MIGUEL ANTONIO FERNANDES SOLER, EUVALDO FERREIRA CABRAL JR.

BT/PTC/9903 – Regulação Sincronizada de Distúrbios Senodais – VAIDYA INÉS CARRILLO SEGURA, PAULO SÉRGIO PEREIRA DA SILVA

BT/PTC/9904 – Desenvolvimento e Implementação de Algoritmo Computacional para Garantir um Determinado Nível de Letalidade Acumulada para Microorganismos Presentes em Alimentos Industrializados – RUBENS GEDRAITE, CLÁUDIO GARCIA

BT/PTC/9905 - Modelo Operacional de Gestão de Qualidade em Laboratórios de Ensaio e Calibração de Equipamentos Eletromédicos – MANUEL ANTONIO TAPIA LÓPEZ, JOSÉ CARLOS TEIXEIRA DE BARROS MORAES

BT/PTC/9906 – Extração de Componentes Principais de Sinais Cerebrais Usando Karhunen – Loève Neural Network – EDUARDO AKIRA KINTO, EUVALDO F. CABRAL JR.

BT/PTC/9907 – Observador Pseudo-Derivativo de Kalman Numa Coluna de Destilação Binária – JOSÉ HERNANDEZ LÓPEZ, JOSÉ JAIME DA CRUZ, CLAUDIO GARCIA

BT/PTC/9908 – Reconhecimento Automático do Locutor com Coeficientes Mel-Cepstrais e Redes Neurais Artificiais – ANDRÉ BORDIN MAGNI, EUVALDO F. CABRAL JÚNIOR

BT/PTC/9909 – Análise de Estabilidade e Síntese de Sistemas Híbridos – DIEGO COLÓN, FELIPE MIGUEL PAIT

BT/PTC/0001 – Alguns Aspectos de Visão Multiescalas e Multiresolução – JOÃO E. KOGLER JR., MARCIO RILLO

BT/PTC/0002 – Placa de Sinalização E1: Sinalização de Linha R2 Digital Sinalização entre Registradores MFC- PHILLIP MARK SEYMOUR BURT, FERNANDA CARDOSO DA SILVA

BT/PTC/0003 – Estudo da Técnica de Comunicação FO-CDMA em Redes de Fibra Óptica de Alta Velocidade – TULIPA PERSO, JOSÉ ROBERTO DE A. AMAZONAS

BT/PTC/0004 – Avaliação de Modelos Matemáticos para Motoneurônios – DANIEL GUSTAVO GOROSO, ANDRÉ FÁBIO KOHN

BT/PTC/0005 – Extração e Avaliação de Atributos do Eletrocardiograma para Classificação de Batimentos Cardíacos – ELDER VIEIRA COSTA, JOSÉ CARLOS T. DE BARROS MORAES

BT/PTC/0006 – Uma Técnica de Imposição de Zeros para Auxílio em Projeto de Sistemas de Controle – PAULO SÉRGIO PIERRI, ROBERTO MOURA SALES

BT/PTC/0007 – A Connected Multireticulated Diagram Viewer – PAULO EDUARDO PILON, EUVALDO F. CABRAL JÚNIOR

BT/PTC/0008 – Some Geometric Properties of the Dynamic Extension Algorithm – PAULO SÉRGIO PEREIRA DA SILVA

BT/PTC/0009 – Comparison of Alternatives for Capacity Increase in Multiple-Rate Dual-Class DS/CDMA Systems – CYRO SACARANO HESI, PAUL ETIENNE JESZENSKY

BT/PTC/0010 – Reconhecimento Automático de Ações Faciais usando FACS e Redes Neurais Artificiais – ALEXANDRE TORNICE, EUVALDO F. CABRAL JÚNIOR