FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Departamento de Engenharia Electrotécnica e de Computadores

Processador dedicado para o teste em-circuito de blocos analógico-digitais em microssistemas integrados.

Francisco Xavier Fernandes Duarte

Licenciado em Engenharia Electrónica e Telecomunicações
pela Universidade de Aveiro

Dissertação submetida para a satisfação parcial
dos requisitos do grau de mestre em
Engenharia Electrotécnica e de Computadores
(Área de especialização em Informática e Sistemas Digitais)

Porto, Novembro de 2005

Dissertação realizada sob a supervisão do
Prof. Dr. José Alberto Peixoto Machado da Silva
Professor Auxiliar do
Departamento em Engenharia Electrotécnica e de Computadores da
Faculdade de Engenharia da Universidade do Porto

e do
Prof. Dr. José Carlos dos Santos Alves
Professor Auxiliar do
Departamento em Engenharia Electrotécnica e de Computadores da
Faculdade de Engenharia da Universidade do Porto

O trabalho aqui apresentado
foi realizado no âmbito do Projecto ASSOCIATE
do Programa Europeu MEDEA+
e do programa de Projectos de Investigação em Consórcio da
Agência da Inovação

Resumo

Palavras chave: *teste de blocos analógico-digitais, processador de teste, metodologia e geração automática do processador de teste, microssistema integrado, conversor AD.*

Nesta dissertação é proposta uma infra-estrutura e metodologia para o teste de núcleos analógicos e mistos (analógico-digitais) embebidos em SoC (*System on Chip*). Compreende um processador específico para o teste e *wrappers* que garantem o acesso a blocos a testar. O processador é responsável por realizar e escalonar as operações de teste, assim como controlar a interface com um testador externo. A geração de estímulos, controlo do teste, e a compressão de dados é efectuada *on-chip* de modo a permitir a minimização o número de interfaces com o testador e diminuir o tempo de teste associado.

Este processador possui a particularidade de poder ser implementado em lógica reconfigurável já existente no SoC, tirando-se partido dos recursos existentes no sistema. Como as operações de depuração a realizar dependem dos modos particulares de funcionamento e dos requisitos impostos pelos núcleos, sob depuração, embebidos no SoC, o processador de teste possui uma arquitectura flexível de modo a ir ao encontro das necessidades de teste impostos pelos diversos núcleos a testar.

Esta solução garante uma maior flexibilidade relativamente às propostas que recorrem a processadores genéricos adaptados (DSP, processadores, microcontroladores, …) para o teste ou a processadores específicos (*ASIP - Application Specific Instruction-set Processor*), com funcionalidade fixa, uma vez que, em cada instante, a FPGA pode ser configurada como processador específico de teste, de acordo com as operações a realizar. Deste modo, garante-se uma fácil reutilização e modificação de diversas técnicas de teste, evitando o risco de o processador se tornar obsoleto.

Define-se igualmente uma metodologia para a adaptação do processador de teste, de acordo com o tipo e os requisitos do núcleo sob depuração, demonstrando ser aplicável para a depuração de uma vasta gama de núcleos analógicos e mistos. Esta adaptação é efectuada por uma ferramenta computacional, desenvolvida para o efeito,

que gera automaticamente o processador de teste, a partir da informação presente no programa de teste. O processador é optimizado com a funcionalidade especificada pelas instruções inseridas no programa de teste, incluindo apenas os recursos necessários à sua execução.

Abstract

Keywords: *analogue and mixed-signal cores testing, test processor, methodology and test processor customization, system on Chip, A/D converter.*

This dissertation presents an infrastructure and methodology for testing analogue and mixed-signal cores embedded in *system-on-chip*. It comprises an application specific processor and wrappers around the cores under test. The processor is responsible for the scheduling and controls the test operations, as well as the interface with the external tester. On-chip stimuli generation, test control and data compression allow for minimizing external tester interface resources and test time.

The solution proposed here relies on reusing the logic reconfigurable block existing in the SoC, to implement an application specific instruction test processor.

Having a programmable processor for this task and implemented in a digital reconfigurable FPGA-like block has several advantages, when compared to test-specific controllers with fixed functionality. Depending on the size of the FPGA block, an adequate version of the test processor may be chosen and can be adapted for the precise test requirements of each SoC, avoiding the risk of the test processor becoming obsolete.

A methodology to develop the processor's architecture is proposed, to adapt to the specific test needs of each block under test, in order to fully exploit the reconfigurable resources available within the system, being flexible for the testing a wide range of analogue application circuits. To customize the test processor for a specific test program, an application was developed that configures the various parts of the processor in order to include only the resources necessary to run that test program.

Agradecimentos

Embora apenas um nome seja mencionado na capa desta dissertação, a realização do trabalho a que se refere não teria sido possível sem a colaboração de um grupo de amigos que me acompanharam ao longo dos últimos três anos. Não posso por isso deixar de exprimir aqui a minha profunda gratidão a todos os quantos, de diferentes modos, me ajudaram a levar a bom porto o trabalho aqui apresentado.

Gostaria de começar por agradecer aos meus orientadores, os Professores José Machado da Silva e José Carlos Alves, pela forma como asseguraram a satisfação de todas as condições necessárias à conclusão bem sucedida deste trabalho. Agradeço também a forma como me guiaram no decorrer do mesmo, os conhecimentos que me transmitiram, e todas as sugestões, opiniões e respostas que sempre se disponibilizaram a apresentar e que no seu conjunto representaram um contributo decisivo para o resultado deste trabalho. Por fim, manifesto o meu apreço pela paciência demonstrada ao longo das inúmeras revisões deste manuscrito.

Agradeço também aos Professores Hélio Mendonça e Isidro Vila Verde, não só pelo apoio e interesse manifestados mas principalmente pela sua ajuda, que em certas alturas foi indispensável.

Quase a terminar, gostaria de expressar o meu agradecimento às duas instituições que no seu conjunto tornaram o trabalho aqui descrito possível, o Instituto de Engenharia de Sistemas e de Computadores do Porto que generosamente me acolheu durante a sua duração, e a Agência de Inovação que, através do Programa ASSOCIATE, assegurou os meios financeiros necessários à sua execução.

Finalmente, gostaria de agradecer a toda a minha família e amigos, que de uma forma despercebida me ajudaram. Em particular, à *"Pincesa Bichana Patolas"*, minha companheira de viagem a quem furtei muitas das devidas atenções.

Nota ao leitor

Ao longo da escrita desta dissertação procurou-se recorrer o mais possível ao uso de termos portugueses. No entanto, em muitos casos esta opção acarreta o risco de no processo de tradução se perder a identificação imediata do conceito que se pretende transmitir. A fim de minimizar este risco e aumentar a clareza da exposição é em muitos casos usado o termo em língua inglesa, em itálico, optando-se pelo sacrifício da tradução em favor da clareza da apresentação.

O mesmo se passa quanto à utilização de acrónimos. Nos casos em que se julgou que a tradução não introduz dificuldades à leitura optou-se por usar o acrónimo resultante da tradução. Em outros casos, em que há uma maior familiaridade de utilização das versões inglesas, mantém-se essa forma. Uma lista de acrónimos é incluída para facilitar a sua identificação.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

AD	Analógico-digital
ATE	*Automatic Test Equipment*
BIST	*Built-In Self-Test*
BS	*Boundary-Scan*
CI	Circuito Integrado
DA	Digital-analógico
DfD	*Design for Debug*
DfT	*Design for Testability*
DNL	*Differential Non-Linearity*
DSP	*Digital Signal Processing*
FPGA	*Field Programmable Gate Array*
INL	*Integral Non-Linearity*
IP	*Intellectual Property*
JTAG	*Joint Test Action Group*
LSB	*Least significant bit*
RAM	*Random Access Memory*
ROM	*Read-Only Memory*
RTL	*Register Transfer Level*
TAM	*Test Access Mechanism*
THD	*Total Harmonic Distortion*
VHDL	*VHSIC Hardware Description Language*

CAPÍTULO 1

Introdução

1.1. Evolução Tecnológica

Os constantes avanços tecnológicos na área dos semicondutores, que têm sido confirmados pela lei de Moore [1], permitiram o desenvolvimento de sistemas completos num único circuito integrado (CI), vulgarmente denominado por *System-on-Chip* (SoC). Estes sistemas são tipicamente compostos por microprocessadores, memórias, conversores A/D e D/A, DSPs, entre outros. Esta integração permite um melhor desempenho, menor consumo de potência, menor volume e menor peso comparado com o projecto baseado em múltiplos CIs.

Contudo, os níveis de integração crescentes das tecnologias de microelectrónica impõem novos desafios ao teste e à depuração (*debug*) dos produtos, sendo necessário um maior esforço no que diz respeito à concepção e validação do projecto. Tal aumento de esforço implica um aumento do tempo e custo do projecto. Por outro lado, o rápido aumento tecnológico diminui o ciclo de vida dos produtos, fazendo com que o tempo para a introdução de um novo produto no mercado seja um dos parâmetros mais importantes em relação ao custo final do produto.

A figura seguinte ilustra que uma variação no tempo para o mercado, ΔT, conduz a uma redução total dos lucros, sendo de extrema importância encontrar uma solução de compromisso entre o atraso de lançamento de um produto no mercado e a qualidade associada.

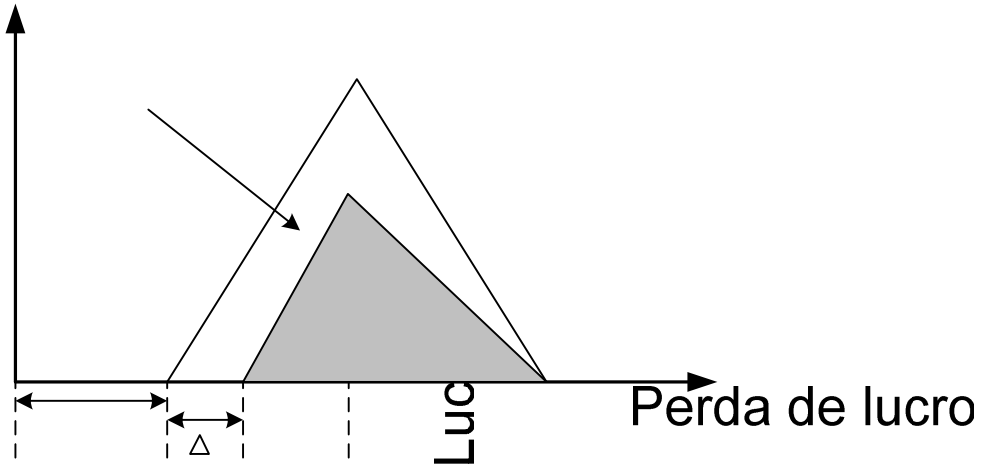


Figura 1.1: Modelo do time-to-market [4].

De acordo com o SIA (*Semiconductor Industry Association*), os custos associados ao teste excedem a 35% do custo total de um SoC, e continuam a aumentar. Isto implica que, para SoCs de elevada complexidade, o custo de teste desses componentes pode tornar o seu custo total muito pouco competitivo. Para além disto, os procedimentos de teste (geração e aplicação) são muito demorados, o que pode aumentar o tempo de introdução no mercado do novo produto (TTM – *Time-to-market*), podendo deste modo inviabilizar o seu lucro.

O recurso a módulos de hardware complexos, também denominados núcleos (*cores*), permite aumentar a produtividade e simultaneamente reduzir o tempo para o mercado, uma vez que estes núcleos se encontram pré-desenvolvidos e pré-verificados. De acordo com o SIA, em 2005, 90% dos CIs é composto por núcleos de hardware [3]. Este facto pode ser comprovado pelo estudo realizado pela agência In-Stat que estimava um crescimento médio de 31% de SoCs no mercado, tendo atingido em 2004 1.3 biliões de unidades. Outra prova encontra-se nos investimentos realizados pelos fabricantes, onde dois terços (2/3) dos seus recursos de I&D são capitalizados para área de SoCs [2].

Time to market T

1.2. Importância do teste

A testabilidade fornece às empresas uma firme vantagem competitiva no mercado da alta tecnologia e retira limitações no desenvolvimento do produto. Contudo, o teste de

microssistemas integrados constitui actualmente uma dificuldade para os fabricantes, devido aos elevados níveis de integração e à diversidade de circuitos a testar. O recurso às abordagens de teste convencionais para detectar e identificar falhas é praticamente impossível, sendo necessário desenvolver novas metodologias para a testabilidade (DfT – *Design for Test*), para a depuração (DfD – *Design for Debug*), aplicáveis tanto a sistemas digitais, como a analógicos e mistos.

Os problemas identificados originaram um esforço de normalização, resultando em normas como IEEE 1149.1 (digital) [5], IEEE 1149.4 (analógico) [6] e IEEE 1500 [7] para o teste de SoCs. O actual método de varrimento (*scan-based*) de teste para sistemas integrados complexos, incluindo circuitos mistos analógico-digitais, tem dificuldade em dar resposta às novas tecnologias de microelectrónica e aos níveis de integração mais elevados. Estes problemas são derivados do facto de ser necessário testar circuitos cada vez mais complexos, menos acessíveis e em cada vez menos tempo.

Da mesma maneira, as várias metodologias de teste e depuração não conseguem lidar de forma eficaz com os novos SoCs baseados cada vez mais em blocos funcionais de propriedade intelectual (IP – *Intellectual Property*) e com requisitos de utilizador constantemente utilizáveis, principalmente para SoCs com mais de um bloco microprocessador. O tempo necessário para a depuração de um SoC domina cada vez mais o tempo necessário a levar o produto para o mercado. Deste modo, um aspecto importante é assegurar a acessibilidade aos nós internos de um SoC, condição necessária para testar e depurar o dispositivo em si, como para efectuar o teste no ambiente do sistema completo.

Este problema pode ser minimizado recorrendo a fluxos de projecto que visam facilitar a etapa de teste de CI após a sua fabricação. O projecto estruturado para o teste (DfT) incorpora regras e técnicas no projecto de produtos para tornar o teste mais fácil e prático, possuindo influência em todas as etapas de vida do produto. Uma boa metodologia de teste, permite que uma função integrada seja utilizada durante o ciclo de desenvolvimento e que os testes criados nesta fase sejam reutilizados nas fases seguintes. O DfT pode ser usado de forma a controlar o desenvolvimento de produtos complexos, minimizando o tempo de desenvolvimento e reduzindo os custos associados à sua produção.

1.3. Contribuição apresentada nesta dissertação

Para ultrapassar as dificuldades inerentes ao teste de microssistemas integrados, este trabalho propõe o desenvolvimento de uma infra-estrutura e metodologia para o teste de núcleos analógicos e mistos (analógico-digitais) embebidos num SoC. Compreende um processador específico de teste e *wrappers* que garantem o acesso aos blocos a testar. Este processador tem a particularidade de poder ser implementado em lógica reconfigurável existente no SoC, tirando-se assim partido dos recursos já existentes no sistema.

Dentro deste objectivo central, podem-se identificar os seguintes objectivos específicos:

- Desenvolvimento de metodologia, aplicável para DfT e DfD, para análise e reforço da testabilidade e capacidade de depuração, de forma a identificar zonas do SoC difíceis de testar e guiar a reconfiguração para a testabilidade.
- Desenvolvimento de soluções BIST para blocos analógicos e mistos, compatíveis com as normas IEEE 1149.1, IEEE 1149.4 e IEEE 1500, em particular para ADCs, e suas estruturas de interface (*wrappers*).
- Definição de requisitos de utilizador para a integração no fluxo do projecto de metodologias de teste e depuração.

1.4. Organização da dissertação

Esta dissertação encontra-se organizada do seguinte modo:

- No capítulo 2 apresenta-se uma revisão de teste de dispositivos baseados em núcleos.
- No capítulo 3 é apresentada a arquitectura do processador de teste. A descrição desta arquitectura inicia-se com a apresentação dos requisitos de teste e a sua influência no(s) modo(s) de operação do processador, seguidas

da sua constituição interna, dos blocos principais, e do respectivo conjunto de instruções.

- No capítulo 4 descreve-se a metodologia para a geração automática do processador de teste, apresentando-se o fluxo de informação envolvido e descrevendo-se em detalhes cada elemento de entrada e saída.
- No capítulo 5 é efectuada a validação da solução proposta recorrendo a duas metodologias para o teste de núcleos analógicos e mistos.
- No capítulo 6 são apresentadas as considerações finais e as perspectivas de evolução futura, apresentando-se as vantagens, pontos de originalidade, desvantagens e limitações do trabalho desenvolvido.

CAPÍTULO 2

Teste de dispositivos baseados em núcleos de hardware

Os recentes avanços na tecnologia de CIs permitem o desenvolvimento de sistemas completos num único componente, isto é, as funcionalidades apresentadas por diversos blocos (interfaces de I/O, os tradicionais microprocessadores, memórias, …) que ainda há pouco tempo se encontravam implementadas em circuitos integrados (CI) isolados podem agora ser integradas num único substrato monolítico. Nestes encontramos os *Multi-Chip Modules* (MCM), os *System-on-Chip* (SoC), e os *System-in-Package* (SiP), sendo os SoC os que apresentam actualmente o maior mediatismo.

De modo a minimizar o tempo de acesso ao mercado (TTM), o projecto destes sistemas integrados recorre ao uso de núcleos de hardware, os quais apesar de se encontrarem pré-desenvolvidos e pré-verificados nem sempre se encontram testados, e envolve dois actores, o fornecedor do núcleo e o respectivo utilizador. Este último é responsável pelo teste e fabrico do sistema, incluindo o teste dos próprios núcleos.

Este capítulo inicia-se com a apresentação dos desafios impostos para o teste de microssistemas integrados, relativamente ao teste efectuado para as placas de circuito impresso. Seguidamente, é efectuada uma revisão bibliográfica de soluções propostas para o teste de microssistemas integrados e que tentam solucionar os desafios apresentados. Finalmente é efectuada a conclusão do capítulo.

2.1. Implicações impostas pelo teste de SoCs

Comparativamente ao teste efectuado para placas de circuito impresso, o teste de sistemas baseados em núcleos de hardware apresenta desafios adicionais [8], que se destacam de seguida:

- Teste interno de núcleos – com a redução constante do tamanho do transístor, o baixo consumo e o aumento da frequência de relógio, torna os tradicionais métodos de teste inadequados para o teste de núcleos, pelo que o desenvolvimento de novas metodologias de teste, de alta qualidade e de baixo custo, está a tornar-se num desafio cada vez maior.
- Transferência do conhecimento de teste de núcleo – dado que o teste envolve dois actores, é necessário uma padronização na transferência de informação relativa ao teste.
- Acesso aos núcleos integrados – como os núcleos são blocos internos ao *SoC*, é necessário definir um acesso aos respectivos terminais, de modo a aplicar o estímulo de teste e a capturar a respectiva resposta.
- Integração e optimização do teste do sistema – deve-se escolher um compromisso entre os diversos parâmetros relativos ao teste, dos quais se destacam o tempo, qualidade de teste, e degradação de desempenho.

Face ao descrito, a padronização da interface dos núcleos, a geração dos estímulos de teste e a captura das respectivas respostas são alguns dos desafios impostos para o teste de SoCs. Na secção seguinte apresenta-se a arquitectura genérica para o teste de SoCs.

2.2. Arquitectura genérica para teste de SoCs

A arquitectura genérica proposta para o teste de SoCs é constituída por três elementos: um gerador de estímulos e o respectivo receptor das respostas de teste, um mecanismo de acesso de teste (TAM – *Test Access Mechanism*) e pela interface do núcleo,

vulgarmente denominada por *wrapper*. Esta arquitectura encontra-se ilustrada na figura seguinte.

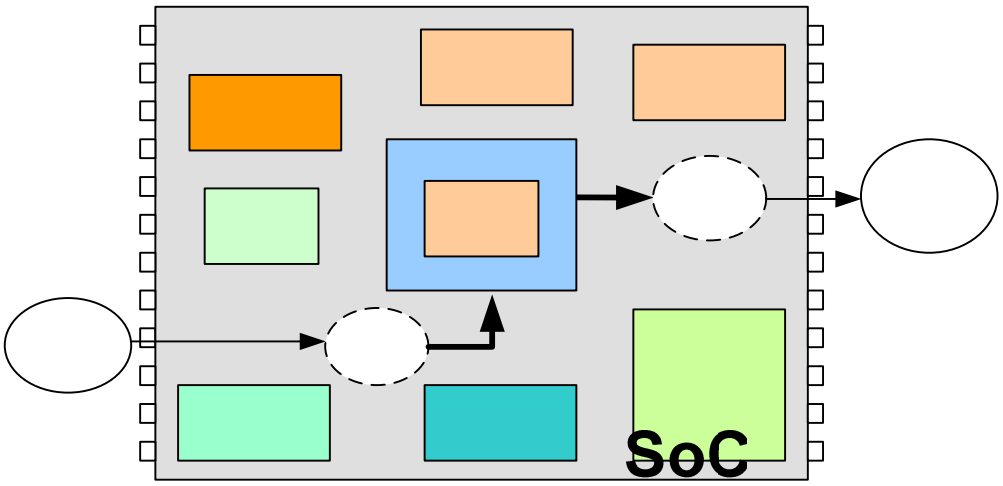


Figura 2.1: Arquitectura genérica para o teste de SoCs [9]

Tanto o gerador como o receptor podem ser implementados fora do SoC, recorrendo a um ATE (ATE – *Automatic Test Equipment*), ou dentro do SoC, através de BIST (BIST – *Built-In Self-Test*), ou então uma combinação das propostas anteriores, não sendo necessário que o gerador e o receptor sejam do mesmo tipo. Esta escolha é determinada por três factores [10]:

- Tipo de núcleo: o tipo de núcleo é classificado em três categorias lógica, memória e analógico (ou misto analógico-digital), os quais possuem funcionalidades e defeitos diferenciados pelo que requerem operações de teste distintas.
- Teste desenvolvido: os testes são classificados pelo tipo de circuito a testar, mas também pelas facilidades a realizar e pela adaptação necessária a efectuar.
- Qualidade e custo do teste: uma qualidade de teste elevada possui implicações no custo do teste. Comparativamente ao requerido pelo ATE, o recurso a BIST permite gerar com exactidão o estímulo de teste *on-chip*, efectuar o teste à frequência nominal (em tempo de execução – *at-speed*) do sistema e reduzir o tempo de teste. Porém o seu uso está associado a certos custos

adicionais, sendo necessário mais área de silício para a sua implementação o que pode acarretar uma degradação do desempenho através da introdução de hardware adicional, podendo inviabilizar a sua aplicação.

O mecanismo de acesso ao teste (TAM) é responsável por conduzir os estímulos de teste do gerador até ao núcleo, e as respostas obtidas deste até ao receptor. O projecto do TAM envolve compromissos entre a largura de banda adoptada para transportar os estímulos e o custo associado. Para a sua implementação é possível recorrer à partilha dos recursos existentes, como por exemplo barramentos de dados.

A interface (*wrapper*) entre o núcleo, o restante SoC e o TAM é necessária de modo a garantir um isolamento adequado do núcleo durante o teste, prevenindo eventuais conflitos entre o núcleo e o restante SoC. Deve obrigatoriamente garantir os seguintes modos de operação:

- Modo normal: o *wrapper* é transparente para o restante SoC durante o funcionamento normal do núcleo.
- Modo de teste do núcleo (teste interno): o *wrapper* garante ao núcleo isolamento necessário durante o seu teste e simultaneamente garante o acesso aos seus terminais. O TAM é ligado ao núcleo para transportar os estímulos de teste e as respectivas respostas.
- Modo de teste externo: o *wrapper* garante a ligação entre o TAM e as ligações externas do núcleo com o restante SoC para realizar o teste estrutural das respectivas ligações.
- Modo de bypass: *wrapper* permite que vários núcleos se encontrem ligados a um mesmo TAM.

Nas secções seguintes apresenta-se uma revisão do estado da arte.

2.3. Teste de núcleos de hardware

Nesta secção abordam-se algumas soluções propostas para o teste de dispositivos baseados em núcleos de hardware. A apresentação destas soluções baseia-se na classificação dos núcleos (lógica, memória e analógico) proposta por Zorian [10], dando especial relevo às soluções propostas para núcleos analógicos e mistos (analógico-digitais), no qual se enquadra a abordagem proposta nesta dissertação.

2.3.1. Teste de núcleos digitais

O teste de núcleos digitais já possui uma longa história no teste de componentes electrónicos e são muitas as soluções propostas para o seu teste. Algumas destas soluções recorrem a microprocessadores/microcontroladores integrados num SoC para efectuar a testabilidade dos diversos blocos que o compõem [11] [12]. Para garantir a exactidão das operações de teste é necessário efectuar em primeiro lugar a integridade funcional do microprocessador, mas esta verificação é uma tarefa difícil de realizar devido em grande parte ao número de dispositivos lógicos existentes, número limitado de pinos, arquitectura diversificada e complexa, entre outros.

Existem diversas abordagens para o problema de teste de processadores, mas apenas será apresentada uma abordagem baseada em software, pois o teste deste tipo de núcleos não se engloba no enquadramento proposto nesta dissertação.

Corno et. al. [13] recorrem ao conceito de macro para efectuar o autoteste de microprocessadores, sendo necessário gerar uma macro por cada instrução que se pretender testar. Uma macro é um conjunto de instruções agrupadas de forma a carregar os operandos da instrução que se pretende testar, executar a instrução e propagar os seus resultados para a memória. A versão final do programa de teste a executar é constituído por uma sequência de macros cujos operandos são seleccionados por um algoritmo genético.

```
MOV AX, K1   ; load register AX with K1
MOV BX, K2   ; load register BX with K2
ADD AX, BX   ; sum BX to AX
MOV RW, AX   ; write AX to RW
MOV RW2, PSW ;write status reg to RW
```

(a) ADD

```
MOV BX, 0    ; clear BX
CMP AX, K1   ; compare reg AX with K1
JG  IS_ABOVE ; jump if AX > BX
MOV BX, 1    ; move 1 to BX
    IS_ABOVE:
MOV RW, BX   ; write BX to RW
```

(b) JG

Figura 2.2: Extracto de código de uma macro relativa ao teste da instrução (a) ADD e da instrução (b) JG (Jump on Greater) [13]

A figura anterior ilustra excertos de código de uma macro relativa ao teste de uma instrução ADD (a) e para uma instrução JG (b). A macro relativa ao teste desta última instrução consiste em avaliar o conteúdo de um registo de acordo com o fluxo do programa. Encontra-se dividida em 3 fases, nas quais, se constrói a condição a avaliar, avalia-se a condição e efectua-se o salto, sendo escrito no registo o valor resultante da avaliação. Esta macro permite detectar possíveis faltas que afectem o módulo relativo à avaliação da condição.

Mas esta solução exige que sejam satisfeitos alguns requisitos. Em primeiro lugar, o processador deve ter acesso a uma memória; segundo, ela deve possuir tamanho suficiente para conter o programa de teste, finalmente esta deve-se encontrar acessível do exterior para carregar o programa de teste e ler os resultados produzidos pela sua execução.

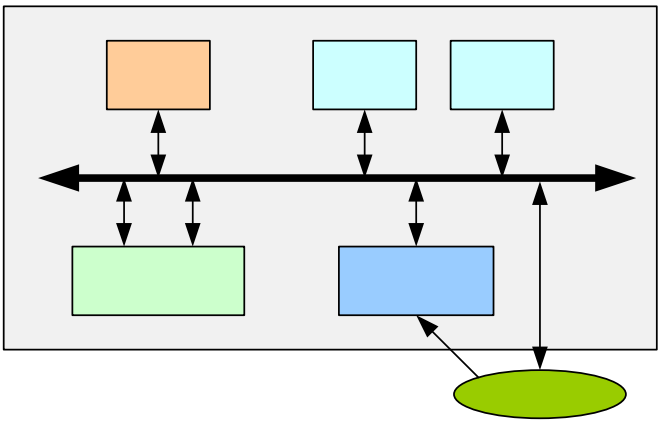


Figura 2.3: Possível arquitectura para um SoC [13]

A carga da memória é efectuada por um ATE através de um acesso directo à memória (DMA – *Direct Memory Access*), possibilitando a diminuição do tempo total

de teste através da sobreposição no tempo da etapa de carga e teste. A figura 2.3 ilustra esta situação. Além disso, o programa de teste pode ser transferido para a memória compactado, sendo o processador responsável pela descompactação do programa de teste antes da sua execução. Alternativamente, o programa de teste pode-se encontrar armazenado numa ROM, permitindo de um modo completamente autónomo a activação do processo de teste.

2.3.2. Teste de núcleos de memória

Os módulos de memória são as estruturas de hardware onde mais se utilizam técnicas de teste, em grande parte devido à estrutura regular que facilita o teste e consiste na aplicação de algoritmos complexos (March C, MSCAN, GALTAP, …) para garantir uma boa cobertura de falhas (falhas de endereçamento, falhas de transição, retenção de dados, …).

O teste de uma memória é mais simples de realizar num SoC por um processador, uma vez que este tipo de teste utiliza padrões determinísticos (por exemplo sequências 55h e AAh) sendo necessário apenas um segmento de código. Rajsuman [11] utiliza um código em *Assembly* do algoritmo March [14], para testar uma memória de 2Mbits. Argumenta que com esta solução diminui-se significativamente os requisitos de um ATE, realizando o teste em tempo de execução, sem degradação do desempenho do sistema e sem área de silício adicional, facilitando a alteração do algoritmo de teste sem a necessidade de alterar o hardware.

Zorian e Ivanov [15] apresentam um esquema para o teste de ROM. Consideram este tipo de memória um circuito combinacional, onde o endereço e o conteúdo da memória representam respectivamente a entrada e a saída do circuito combinacional. Deste modo recorre-se a um teste exaustivo, efectuando a leitura completa da ROM e compactando a saída. É utilizado um método de compactação baseado num MISR (MISR – *Multiple Input Sifht Register*) bidireccional e contador de 1's de modo a reduzir a probabilidade de mascaramento. Esta abordagem pode ser utilizada para efectuar testes de fabrico ou de manutenção.

2.3.3. Teste de núcleos analógicos e analógico-digitais

Ao contrário do que sucede para o teste de núcleos puramente digitais, onde os resultados de teste se encontram claramente definidos (sem necessidade de qualquer tipo de tolerância), o teste de núcleos analógicos e analógico-digitais é mais complexo, uma vez que o processamento de sinais com valores contínuos faz com que seja impossível avaliar o circuito em teste com base apenas num conjunto de valores de referência. Ligeiros desvios dos valores das características dos componentes dos circuitos impossibilitam a obtenção exacta dos valores esperados.

Rajsuman [11] recorre a um microprocessador para efectuar a testabilidade de um núcleo analógico-digital, mais concretamente de um conversor analógico-digital (DAC - *Digital/Analog Converter*).

As operações a efectuar, durante o teste, são especificadas num programa próprio de teste do DAC. A aplicação dos padrões de teste é efectuada através de um registo adicional, denominado por *Analog Test Register* (ATR), cujo conteúdo é modificado, durante o teste, através do acesso a um dos registos de endereços do microprocessador.

A figura seguinte ilustra a arquitectura proposta para o teste do DAC.

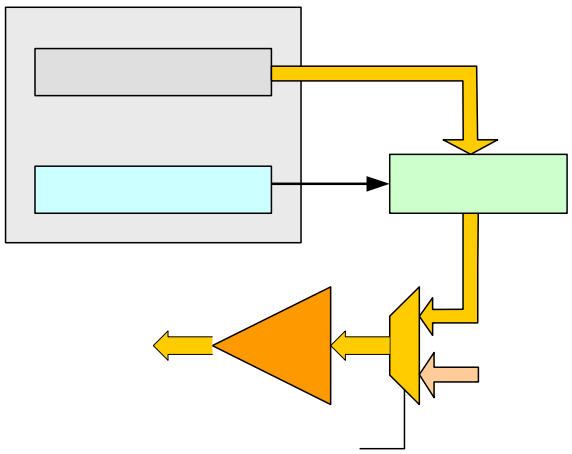


Figura 2.4: Arquitectura proposta para o teste de um DAC, recorrendo a um microprocessador [11].

O multiplexador (MUX), ilustrado na figura anterior, permite encaminhar para o DAC o sinal mais adequado (padrão de teste/sinal normal) de acordo com o seu modo de operação.

O programa apresentado na figura seguinte ilustra a geração dos padrões de teste adequados para caracterizar o desempenho do DAC. O desempenho é caracterizado através dos parâmetros estáticos gama de conversão, desvio na origem (*offset*), não-linearidade diferencial (DNL – *Differential Non-Linearity*) e não-linearidade integral (INL – *Integral Non-Linearity*), os quais permitem determinar a precisão DC do conversor e em aplicações em que o sinal medido varie lentamente tais como sensores de peso ou de temperatura.

```
MVI 0000H,(A1)      /*carrega o valor 0000H na posicao de memoria
                    especificada por A1 */
MVI FFFFH,(A1)      /*carrega o valor FFFFH na posicao de memoria
                    especificada por A1 */
Start MVI 0000H,D1 /* carrega o registo D1 com o valor 0000H */
MVI 0100H,D2
Cont  CMP D1,D2     /*compara conteúdo de D1 com D2*/
JZ      Stop
MOVE D1,(A1)        /*carrega o conteúdo de D1 na posicao de memoria
                    especificada por A1 */
INCR D1             /*gera próximo valor*/
JMP  Cont
Stop  HLT           /*termina teste*/
```

Figura 2.5: Programa desenvolvido para a caracterização do DAC através dos parâmetros estáticos [11].

O desvio na origem (*offset*) é determinado pela aplicação de um estímulo nulo ao DAC e capturando a respectiva resposta. A instrução **MVI 0000H,(A#)**, onde A# é um dos registo de endereços, é responsável pela geração do estímulo de teste.

A gama de conversão (FSR – *Full Scale Range*) do DAC é a diferença registada entre a resposta do DAC, quando estimulado com o valor de fim de escala (FFFFH), e a resposta obtida para o desvio na origem, isto é, $V_{FSR}=V_{FS}-V_{OS}$. O procedimento para obter estes valores é idêntico ao efectuado para o desvio na origem, mas neste caso são necessárias duas instruções (**MVI 000H,(A#)**, **MVI FFFFH,(A#)**) para gerar os estímulos adequados.

Para o cálculo da não-linearidade diferencial (DNL), que consiste no desvio da largura (diferença entre os níveis de tensão inferior e superior) de cada código em relação ao correspondente valor ideal (1 LSB), é necessário percorrer toda a gama de operação do DAC e capturar as respectivas respostas. As instruções situadas entre as linhas 3 a 9 (ilustradas na figura 2.5), garantem o estímulo necessário para se obter esta medida.

O procedimento anterior pode também ser usado para determinar a não-linearidade integral (INL), que consiste na diferença entre os níveis de transição reais e os ideais.

Apesar de a caracterização do desempenho do DAC ser efectuada através dos seus parâmetros estáticos, as não-idealidades existentes neste tipo de dispositivos, conversores A/D ou D/A, fazem com que o seu comportamento na presença de sinais dinâmicos seja diferente do observado em resposta de sinais estáticos, com desvios da característica estática causados por erros dinâmicos dependentes do sinal de entrada. Esta dependência faz com que uma dada especificação do desempenho de um conversor só seja válida para sinais de entrada semelhantes aos usados na sua obtenção, fazendo com que a completa caracterização de um conversor, mesmo para parâmetros estáticos, exija muitas vezes o seu teste a diferentes valores de frequência.

AbdEl-Halim [16] propõe um controlador para o teste de núcleos mistos, denominado por ATC (ATC – *Analog Test Controller*), constituído por uma infra-estrutura IEEE 1149.4 [6] e uma memória para suportar as operações de teste. A figura 2.6 ilustra a arquitectura interna do controlador proposto.

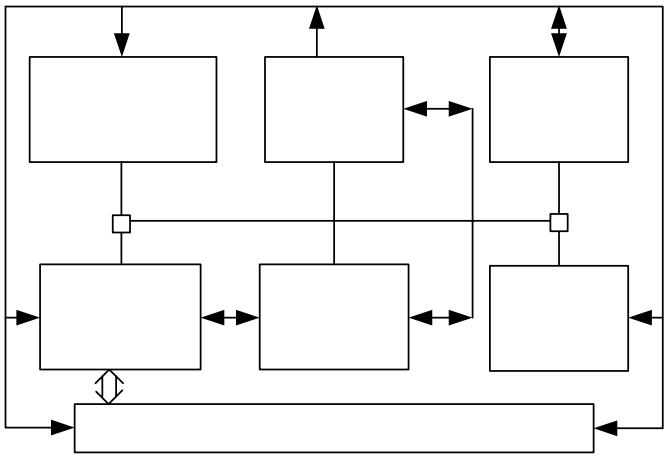


Figura 2.6: Arquitectura interna do controlador de teste proposto por AbdEl-Halim [16]

Para além dos módulos fundamentais para lidar com sinais analógicos (Controlador JTAG e *JTAG Bus Drivers/Buffers*) verifica-se a inclusão de módulos adicionais que garantem a funcionalidade do ATC. As operações a realizar são especificadas num programa de teste, armazenado na memória de programa, e a informação resultante da sua execução salvaguardada na memória de dados. A interface de comunicação garante a comunicação com um ATE, através do qual é carregada a memória de programa. A detecção de possíveis erros existentes entre a informação presente nos barramentos analógicos e os dados presentes no programa de teste é efectuada pela unidade de verificação, que contêm dispositivos de verificação adequados. Toda a actividade do ATC é gerida pela unidade de controlo.

Associado ao ATC existe uma ferramenta responsável pelo desenvolvimento do programa de teste, de acordo com as especificidades do teste a efectuar. Durante esta fase, entra-se num processo iterativo onde os nós a examinar são seleccionados, de acordo com os requisitos de teste impostos, e para cada nó calcula-se a tolerância do sinal e procedendo-se à geração do programa de teste adequado.

O ATC apresenta a versatilidade de efectuar diversos tipos de teste. Por exemplo, durante o funcionamento do sistema é possível recorrer ao ATC para monitorizar o comportamento do sistema através da infra-estrutura IEEE 1149.4 [6], comparando a informação presente no barramento com a definida no programa de teste. Com a observação de uma discrepância o ATC procede à sua sinalização, iniciando a recolha de informação relacionada com a falha, que poderá ser usada posteriormente para diagnóstico do sistema. Durante esta fase, a informação recolhida é processada para situar a existência das falhas ou quebras de desempenho do sistema. Durante a fase de produção, o ATC é simplificado uma vez que não é necessário recorrer ao uso de memórias, sendo administrado através de um equipamento de teste (ATE) ou de um computador que o sustenta com instruções de teste adequadas.

2.4. Arquitectura de teste *on-chip* vs *off-chip*

Na secção 2.2 foi apresentada uma arquitectura genérica para o teste de SoCs, constituída pelos módulos de *hardware* responsáveis pela geração dos estímulos de

teste, recepção das respostas de teste, mecanismo de acesso de teste e *wrapper*. Mas, dependendo das estratégias de teste, a arquitectura de teste poderá encontrar-se classificada em três níveis distintos [17], dependendo dos recursos de teste que são incorporados no SoC.

A figura seguinte ilustra uma arquitectura [18] para o teste de blocos analógico-digitais, onde todos os recursos de teste se encontram incorporados no SoC. Uma estrutura deste tipo é vulgarmente denominada por BIST (BIST – *Built-In Self-Test*).

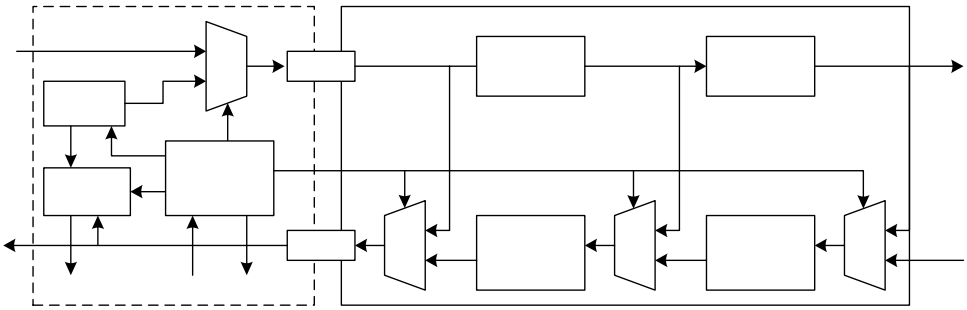


Figura 2.7: Arquitectura da estrutura BIST [18]

Toda a operação é controlada pelo bloco TC (*Test Controller*), sendo o bloco TPG (*Test Pattern Generator*) responsável pela geração dos estímulos de teste, permitindo gerar uma variedade de estímulos de teste distintos, num total de dezasseis estímulos (DC, sinusoidais, pseudo aleatórios, entre outros), o que facilita o teste de uma vasta gama de circuitos analógicos.

O processamento das respostas de teste é realizado pelo bloco ORA (*Output Response Analizer*). As operações realizadas permitem, por exemplo, efectuar compactação das respostas de teste, facilitando a sua comparação com um conjunto predefinido de valores, que definem gamas de tolerância aceitáveis para o resultado do teste, sendo possível deliberar, com base no resultado da comparação, se o bloco analógico em causa passou ou não no teste efectuado.

Arquitecturas de auto-teste semelhantes podem ser encontradas em [19], para o teste de conversores A/D e D/A, VCOs (*Voltage-Controlled Oscillators*) e PLLs (*Phase Locked Loops*), recorrendo à técnica HABIST (*Histogram-based Analog BIST*) onde os resultados do teste efectuado aos blocos analógicos são convertidos para resultados digitais. Estes são processados para determinar os parâmetros adequados.

Hafed e Roberts [20] abordam a viabilidade de realizar, em simultâneo, operações de teste sobre múltiplos núcleos analógico-digitais, recorrendo a estruturas BIST. Analisando as questões associadas com a medição *on-chip* de diversos níveis de desempenho, nomeadamente os aspectos relacionados com a integridade do sinal que surgem com o funcionamento em simultâneo de múltiplos circuitos de teste. As experiências realizadas, sobre um protótipo de teste descrito em [21], demonstram a viabilidade desta estratégia.

Porém, o facto de toda a infra-estrutura se encontrar dentro do SoC é necessário uma área de silício adicional para a sua implementação, podendo em algumas circunstâncias inviabilizar a adopção desta estratégia de teste. Mas o facto de o estímulo de teste ser gerado *on-chip* permite realizar testes com uma maior qualidade, comparativamente a soluções onde o estímulo de teste é gerado por um ATE. Isto deve-se ao facto de a distância entre a interface do SoC e a entrada do bloco a testar ser considerável, podendo em certas circunstâncias provocar uma degradação do sinal de teste. Com a incorporação do gerador no SoC, este encontra-se próximo do bloco a testar pelo que o estímulo pode ser aplicado directamente evitando a sua degradação e distorção.

Outras vantagens que advêm do facto de uma estrutura BIST se encontrar implementada no SoC, estão relacionadas com o tempo associado às operações de teste. Nesta situação é possível realizar o teste à velocidade nominal do bloco, diminuindo o tempo de teste, isto é, o tempo exigido para as operações de teste diminui à medida que a velocidade de operação do dispositivo aumenta.

Mas os custos associados à área de silício adicional necessária para implementar uma estrutura BIST podem ser reduzidos recorrendo a blocos de lógica reconfigurável para a sua implementação. Esta estratégia é proposta por Zeng et. al [22] e Huang et. al [23]. Em ambas as situações, durante a etapa de teste a FPGA é configurada com a estrutura BIST, incorporando os blocos relativos à geração, controlo e processamento das respostas de teste, encontrando-se apenas o barramento de teste implementado no SoC. Após a conclusão da etapa de teste, a FPGA é reconfigurada com a funcionalidade pretendida.

Devido à abundância de recursos de I/O, a FPGA também garante flexibilidade no projecto do barramento de teste, uma vez que pode ser configurada como interface

entre o SoC e o ATE simplificando os requisitos exigidos para este último, dispensando o recurso a equipamento dispendioso. Simultaneamente, sem a limitação do número de pinos de I/O e velocidade, são garantidos testes de melhor qualidade e mais rápidos.

Uma estratégia intermédia à adoptada nestas propostas, consiste em incorporar no SoC apenas o bloco responsável pela geração do(s) estímulo(s) de teste ou pelas operações de processamento sobre as amostras capturadas, sendo o controlo efectuado pelo ATE. De modo a diminuir significativamente as exigências impostas para os níveis de desempenho do ATE, Keezer e Zhou [24] propõe uma interface localizado entre o dispositivo a testar e o ATE, denominada por TSP (TSP – *Test Processor Support*). O objectivo do TSP consiste em dotar o sistema com uma interface de adaptação de elevado desempenho entre o DUT e o ATE, reduzindo significativamente os efeitos introduzidos pela linha de transmissão.

Na sua configuração mais simples, consiste numa matriz de blocos de transmissão e recepção (I/O), simplificando a interface com o ATE, e simultaneamente melhorando as características eléctricas do sinal e reduzindo os tempos associados à propagação do sinal. Benefícios adicionais podem ser obtidos acrescentando funcionalidades extras, como por exemplo, registos e/ou controlador IEEE 1149.1 [5], geradores pseudo-aleatórios, registos de assinaturas, multiplexadores e desmultiplexadores, entre outros, obtendo-se melhorias ao nível da testabilidade.

De acordo com as diversas estratégias de teste descritas, apresenta-se na tabela seguinte, os compromissos entre uma estratégia de teste *on-chip* relativamente a uma *off-chip*.

Tabela 2.1: Comparação entre o teste realizado on-chip vs teste off-chip.

Teste on-chip	***Teste off-chip***
👎 Área de silício adicional	👎 Exige ATE dispendiosos
👍 Tempos de teste reduzidos	👎 Tempos de teste longos
👍 Melhor qualidade de teste	👎 Difícil acesso aos nós internos
👍 Elevado desempenho	👎 Problemas de ruído

2.5. Mecanismo de acesso de teste (TAM)

Um dos problemas relacionados com o teste de núcleos encontra-se associado à acessibilidade, uma vez que o acesso directo à fronteira do núcleo é praticamente impossível, e para efectuar o teste de núcleos os padrões de teste devem viajar do nó de entrada para o núcleo a testar e deste para o nó de saída. Nesta secção apresentam-se algumas propostas para garantir a acessibilidade aos núcleos.

Zorian et. al. [25] propõem um mecanismo de acesso de teste, TAM, denominado por *Hierarchical-Distributed-Data BIST* (HD^2BIST), com o objectivo de solucionar os problemas relacionados com o teste de microssistemas integrados, dos quais se destacam a acessibilidade do núcleo e a reutilização das estratégias de teste, e simultaneamente permitir uma integração modular e a gestão de diversos núcleos com diferentes estratégias de teste e portos de acesso.

O HD^2BIST possui uma arquitectura hierárquica totalmente adaptável para arquitecturas hierárquicas de SoCs complexos. Possui como elemento principal um barramento de teste (TBus – *Test Bus*), que garante a efectiva solução para a acessibilidade do núcleo e a reutilização do TAM, através do qual circulam a informação de teste (controlo, estímulos de respectivas respostas, …). De modo a reflectir esta diversidade, o TBus é constituído por um barramento de controlo de teste (TCB – *Test Control Bus*), através do qual se transporta a informação para configurar e controlar todas as estruturas de teste HD^2BIST, e pelo barramento de dados de teste (TDB – *Test Data Bus*) por onde circulam os vectores de teste, gerados on-chip ou encaminhados a partir do exterior, para testar o SoC.

Esta diversidade reflecte-se também nos protocolos usados para o transporte da informação de teste. Enquanto que barramento TCB codifica a informação de controlo num conjunto predefinido de comandos denominadas por primitivas de teste, recorrendo a um protocolo de passagem por testemunho (*token-based*), o TDB recorre a um protocolo de comunicação série (*scan-chain-based*) para transportar os estímulos e respostas de teste.

De modo a garantir a eficiência do HD^2BIST como mecanismo de acesso de teste são implementados dois blocos, o processador de teste (TP – *Test Processor*) e o

bloco de teste (TB – *Test Block*). Enquanto que o primeiro é responsável pelo controlo de todas as estruturas HD^{2}BIST e escalonamento da operação de teste para cada núcleo no seu domínio, o bloco TB representa a interface dos núcleos a testar, sendo optimizado para suportar a solução de teste implementada para a estrutura interna do núcleo.

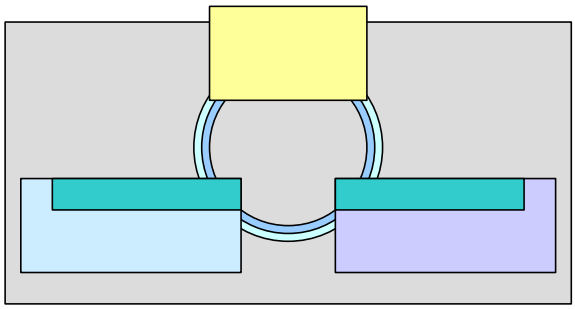


Figura 2.8: Integração do mecanismo de acesso HD2BIST num SoC [25].

O processo de teste consiste na execução de uma sequência de primitivas de teste, trocadas através do TCB entre o TP e o TB do núcleo a testar, que implementam as operações de teste definidas. São definidos dois tipos de primitivas, macroprimitivas e primitivas atómicas, cuja execução depende do bloco que se pretende testar. Enquanto que as macroprimitivas são usadas pelo TP para ligar diferentes níveis hierárquicos do TBus e a sua execução gere o teste de núcleos a níveis hierárquicos inferiores, as primitivas atómicas são usadas pelo TB para configurar a respectiva interface do núcleo a que se encontra associado. Deste modo, o processo de teste resume-se à execução de um conjunto de programas que definem as operações dos diversos núcleos.

Para garantir uma estratégia de teste flexível, o HD^{2}BIST permite que as primitivas sejam geradas dentro do SoC, pelo TP, ou sejam enviadas a partir do exterior para os núcleos a testar. Neste último caso, recorre-se às macroprimitivas *SETENV* (cria um atalho entre dois níveis hierárquicos adjacentes) e *UNSETENV* (que restaura o modo de funcionamento normal do TBus) para configurar os barramentos TCB e TDB de modo a permitir acesso directo a qualquer núcleo do SoC, independentemente da localização hierárquica.

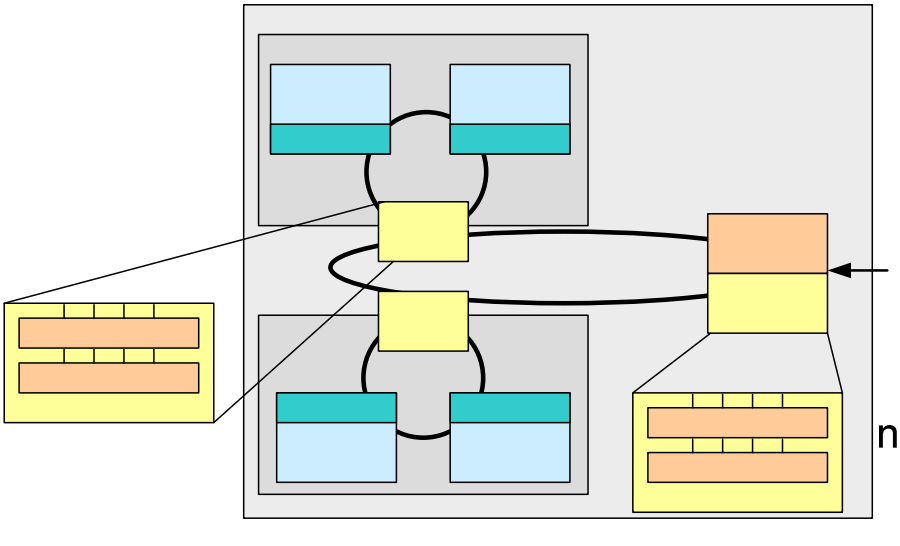


Figura 2.9: Arquitectura para garantir a acessibilidade a todos os níveis hierárquicos presentes num SoC [25].

O escalonamento das operações de teste é também garantido pelo HD^{2}BIST recorrendo a quatro primitivas usadas para gerir a sessão de teste, efectuando a sequência de configurações adequadas. Com a primitiva *START* inicia-se a execução da rotina de teste, sendo os resultados da sua execução capturados através da primitiva *COLLECT*. Estas primitivas usadas em conjunto com as primitivas *WAIT* e *JUMP* permitem flexibilizar o algoritmo de escalonamento das operações de teste de modo a reduzir o consumo de potência e o tempo de teste. Por exemplo, a primitiva *WAIT* permite suspender a execução de um programa de teste até que a testabilidade de outros núcleos se encontre concluída, possibilitando deste modo a redução do consumo durante o teste. Por outro lado, a primitiva *JUMP* permite saltar o teste de certos blocos caso seja detecta a existência de componentes defeituosos, reduzindo o tempo de execução.

A abordagem proposta por Wang et. al. [26] consiste num protocolo de teste para SoCs de dimensões razoáveis e cujos núcleos apresentam características de DfT, por exemplo BIST, *Boundary-Scan*, entre outras. A figura seguinte ilustra a arquitectura do protocolo proposto para o teste.

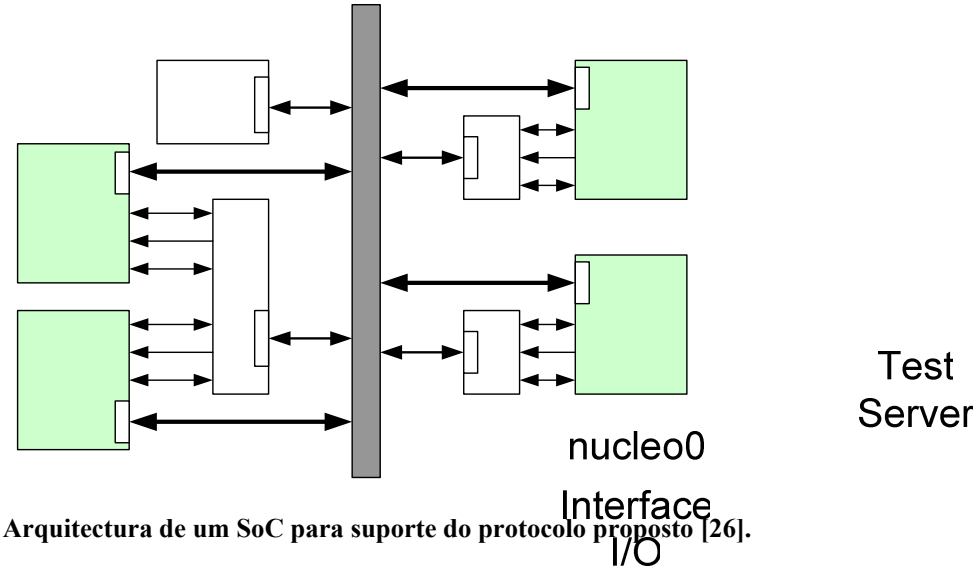


Figura 2.10: Arquitectura de um SoC para suporte do protocolo proposto [26].

Este protocolo é implementado recorrendo a dois módulos, *Test Server* (TS) e *Test Client* (TC). Este último (*Test Client*) é responsável pelo controlo das operações de teste e acomoda a interface de teste de um ou mais núcleos internos do *SoC*, podendo deste modo ser partilhado entre eles para diminuir a área ocupada, e cada TC possui um endereço único distinto do(s) núcleo(s) ao qual se encontra fixo. Assim, durante a execução da etapa de teste os dados necessários são entregues directamente ao núcleo respectivo, sem que seja necessário introduzir lógica adicional evitando deste modo uma degradação do desempenho.

O *Test Server* é responsável pela distribuição, através de pacotes, da informação de teste aos *Test Clients*, podendo ser implementado através de um co-processador de teste ou recorrendo a um microprocessador o qual pode ser programado para garantir as funcionalidades necessárias durante as operações de teste. Em ambas as situações é necessário que seja garantido o acesso à memória, na qual se encontra o programa de teste e a informação relativa a cada núcleo.

A troca de informação entre estes dois módulos (*Test Server* e o *Test Client*) é efectuada através de pacotes, cuja dimensão depende de acordo com o tipo de teste que se pretender efectuar. Na figura 2.11 ilustra-se o formato de cada pacote.

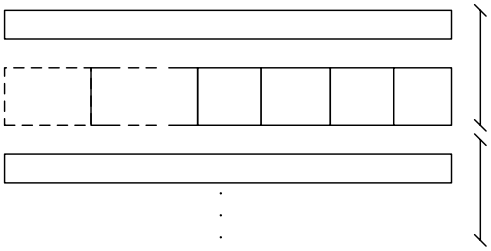


Figura 2.11: Formato de pacotes usados [26].

O cabeçalho (*Header*) é constituído pelo campo *Escape*, para efectuar a distinção entre novos pacotes (preenchido com "0") e a continuação de um pacote já iniciado, e pelo campo *Instruction* que contém a instrução de teste, encontrando-se dividido em 5 sub-campos. O campo *Stop_test* desabilita o modo de teste no *Test Client*, *JTAG_reset* especifica se é necessário efectuar o reset ao controlador *TAP*, *test_mode* especifica a metodologia de teste a aplicar ao núcleo, sendo as operações a efectuar durante o teste especificadas pelo campo *Test_code*. O campo *count* possui diferentes significados dependendo a sua interpretação do estado em que se encontra o *Test Client* quando recebeu o pacote com a operação de teste. Assim, pode ser interpretado como *scan_shift_count*, onde se especifica número de ciclos de relógio a aplicar ao núcleo, como *JTAG_shift_count*, referindo-se ao número de ciclos de relógio a aplicar durante uma operação Boundary-Scan, ou como *capture_count* onde o valor se refere ao número de ciclos durante o qual o *Test Client* captura informação de teste.

Sempre que o *Test Client* recebe um comando do *Test Server*, a operação de teste especificada no pacote é executada até que seja recebido um novo comando, sendo os subsequentes pacotes recebidos, entre os dois comandos, tratados como informação de teste. Deste modo, o *Test Server* pode entregar pacotes, que especificam instruções ou informação de teste, aos restantes núcleos enquanto a operação de teste a efectuar pelo núcleo não se encontrar concluída, permitindo deste modo a possibilidade de efectuar em paralelo operações de teste a diversos núcleos reduzindo significativamente o tempo de teste.

Seuren e Feige [27] propõem uma solução para o teste de núcleos analógico-digitais baseada no manuseamento dos seus sinais de um modo similar ao efectuado para o teste de núcleos digitais. De um modo geral, o teste de núcleos mistos, isto é, núcleos analógico-digitais, implica que todos os seus sinais se encontrem, directa ou

indirectamente, acessíveis ao contrário do que sucede para os núcleos digitais. Seuren e Feige distinguem os sinais, de acordo com os requisitos de acesso, classificando-os em sinais estáticos e sinais dinâmicos. Enquanto os primeiros são constantes ou variam ocasionalmente durante o teste, os últimos variam constantemente. Geralmente todos os sinais de controlo de teste são estáticos e digitais, necessitando de um acesso indirecto, enquanto que os sinais de informação de teste podem ser digitais, necessitando de acesso directo ou indirecto, ou analógicos sendo indispensável um acesso directo.

A solução proposta por Seuren e Feige consiste em implementar em cada núcleo uma interface que efectua a distribuição adequada dos sinais descritos. A figura seguinte ilustra a integração das interfaces nos núcleos de um SoC.

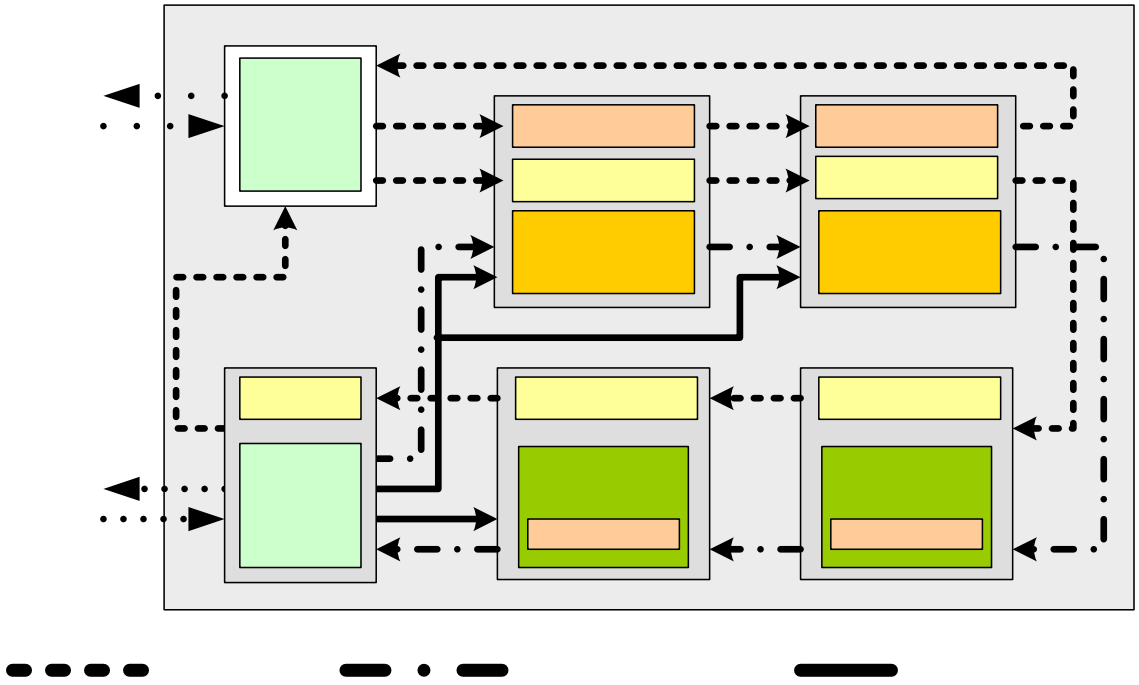


Figura 2.12: Integração do mecanismo de acesso num SoC de acordo com [27].

A interface de cada núcleo é composta por um *Test Control Module* (TCM) local, isto é, localizada no respectivo núcleo, existindo um TCM global responsável pelo controlo dos diversos TCMs locais. Durante a etapa de teste o TCM global e os locais encontram-se ligados através de uma cadeia série e carregados com os valores apropriados de controlo. OS TCM locais são também responsáveis pelo controlo dos sinais estáticos de teste no respectivo núcleo.

Os sinais estáticos de teste são mapeados através de um registo de deslocamento, denominado por *Test Point Register* (TPR), localizado entre a interface digital e o

núcleo analógico-digital, sendo também responsável por garantir outras funcionalidades como por exemplo, o acesso a núcleos localizados noutros módulos durante o teste das ligações ou então isolar o núcleo dos restantes durante a sua testabilidade. Por esta razão a sua configuração é dependente do modo de teste seleccionado pelo TCM.

A acessibilidade da cadeia de teste e dos sinais dinâmicos é garantida pelo *Test Multiplexer Block* (TMX), controlado através do TCM local ou pelo TCM global. A cadeia de teste dos diversos núcleos pode-se encontrar ligada através de uma cadeia série ou então em paralelo dependendo da estratégia de teste, enquanto que os sinais dinâmicos são encaminhados em paralelo pelo TMX.

Esta solução é extremamente flexível suportando o acesso ao teste de núcleos analógicos e mistos, minimizando a degradação do desempenho introduzida pela arquitectura. O tempo total de teste pode ser reduzido através de uma programação adequada do TPR de modo a permitir a execução de testes concorrentes. Pode ser facilmente integrável nas propostas existentes como IEEE 1500 [7], apresentado na secção seguinte, e JTAG permitindo o suporte para DfT.

A integração desta solução num ambiente *Boundary-Scan* encontra-se ilustrada na figura seguinte, sendo esta integração efectuada através da inclusão de registos *TCMs* e *TPRs* adicionais controlados através de duas instruções dedicadas, *PROGRAM_TCM* e *PROGRAM_TPR*, respectivamente.

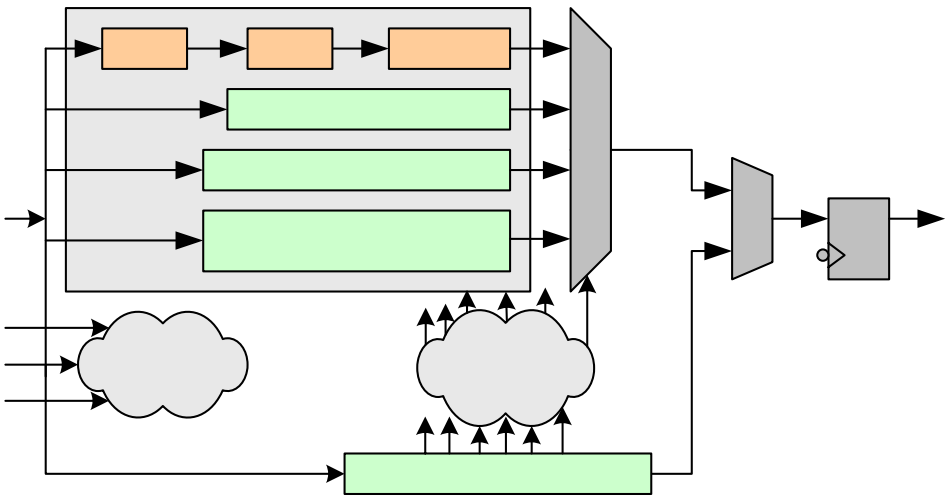


Figura 2.13: Integração do TCM num ambiente Boundary-Scan [27].

Porém, o recurso às tradicionais metodologias de teste *on-chip* para o teste de microssistemas integrados origina alguns problemas de visibilidade da informação de teste. Para contornar este problema de visibilidade, e simultaneamente acrescentar níveis de controlabilidade e observabilidade adequados, as empresas que desenvolvem blocos funcionais de propriedade intelectual (IP), estão actualmente a adoptar o protocolo OCP (OCP – *Open Chip Protocol*) para análise de depuração do SoC.

A potencialidade deste protocolo consiste no facto de recorrer ao OCI (OCI – *On-Chip Instrumentation*) [28], permitindo melhorar os níveis de controlabilidade e observabilidade proporcionalmente ao tamanho e complexidade do sistema. O OCI consiste essencialmente num subsistema dedicado para, de um modo eficiente, monitorizar em tempo real os sinais existentes no SoC. Para que tal seja possível, diversos blocos específicos são colocados em vários pontos do SoC melhorando assim a visibilidade da interface e das operações do sistema. Estes blocos são normalmente operados através de uma interface BS, configurada como porto dedicado à depuração. No entanto, com uma interface deste tipo pode apresentar algumas limitações ao nível de desempenho, uma vez que não foi projectado tendo em mente a depuração *on-chip*. Mas, soluções alternativas estão a ser adoptadas, nomeadamente a proposta pela norma Nexus IEEE5001 [29].

A figura seguinte ilustra a arquitectura para a depuração de um SoC baseado em barramentos OCP.

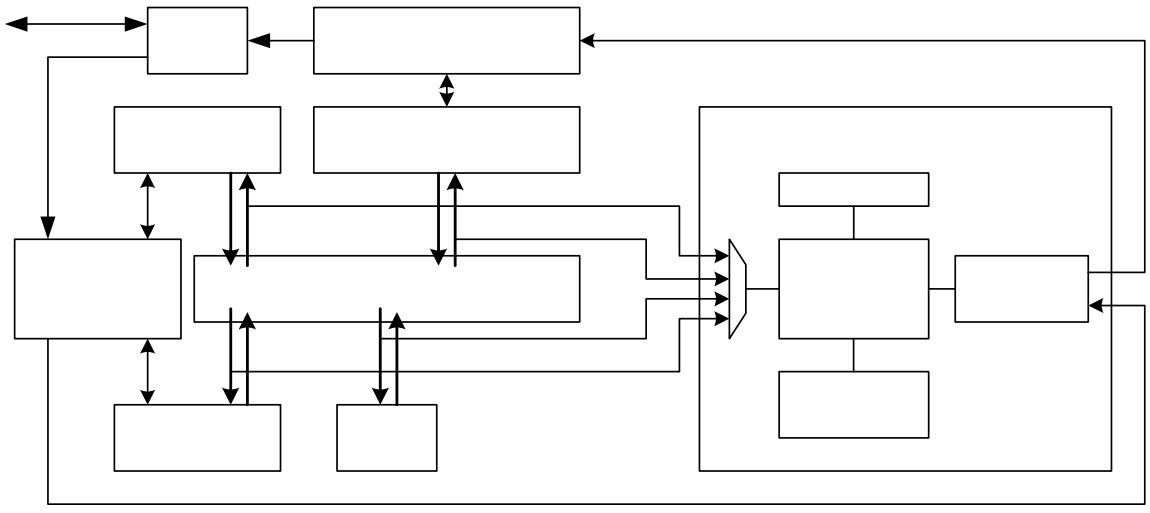


Figura 2.14: Arquitectura para a depuração de um SoC baseado em barramentos OCP [28].

Os módulos que constituem esta arquitectura (processadores, interface de memória, blocos IP, entre outros) partilham um barramento OCP comum, fornecendo-lhe (à arquitectura) uma estrutura adequada para as actividades de depuração, melhorando os níveis de controlabilidade e observabilidade. O bloco PDB (PDB – *Processor Debug Blocks*) garante o controlo da depuração do processador e de outros núcleos, enquanto que os blocos *On-chip Logic Analyzer* e *OCP Bus Analyzer* efectuam a monitorização do sistema, seguindo o rasto de todos os sinais envolventes. Este último bloco possui um multiplexador capturando para uma memória a informação presente nos diversos barramentos, a partir da qual é enviada para o exterior por intermédio de um barramento série.

2.6. Protocolo IEEE 1500

Uma vez que o projecto baseado em núcleos de hardware envolve dois actores, fornecedor do núcleo e o respectivo utilizador, e de modo a facilitar a integração de núcleos originários de fornecedores distintos há a necessidade de padronizar a interacção entre fornecedor do núcleo e o respectivo utilizador. O protocolo IEEE 1500 [7] define uma arquitectura flexível para a integração de diversos núcleos e uma linguagem conhecida como CTL (CTL – *Core Test Language*) baseada na linguagem STIL (STIL – *Standard Test Interface Language*) [30].

Este protocolo não cobre/preenche aspectos relacionados com: (1) método de teste interno dos núcleos, (2) integração do teste do sistema, (3) mecanismos de acesso de teste, (4) fonte e receptor do teste; uma vez que a sua padronização não é viável devido às restrições relacionadas com a diversidade de núcleos e sistemas, (5) e o teste de núcleos analógicos e mistos [8].

2.6.1. Linguagem de teste de núcleos

A linguagem de teste de núcleos, conhecida como *Core Test Language*, deve expressar todas as informações relacionadas com o teste de núcleos como por exemplo, métodos de teste, modos de teste e respectivos protocolos, padrões de teste (lista de vectores de teste, polinómios para BIST, algoritmos para teste de memórias, …) entre outros.

2.6.2. Arquitectura de teste de núcleos

A arquitectura de teste de núcleos possui como objectivo a definição de uma interface de hardware (*wrapper*) flexível, capaz de conduzir os padrões de teste para o núcleo assim como facilitar a integração dos mesmos, independentemente da sua funcionalidade. A sua definição deve garantir as seguintes características:

- Possuir múltiplos modos de operação
  - Normal, teste de ligações, teste interno, …
- Efectuar as ligações entre os núcleos e o mecanismo de acesso ao teste (TAM), independentemente do número de portas do núcleo.

A figura seguinte apresenta uma visão geral do *wrapper*, donde se ilustram os elementos que constituem a sua arquitectura.

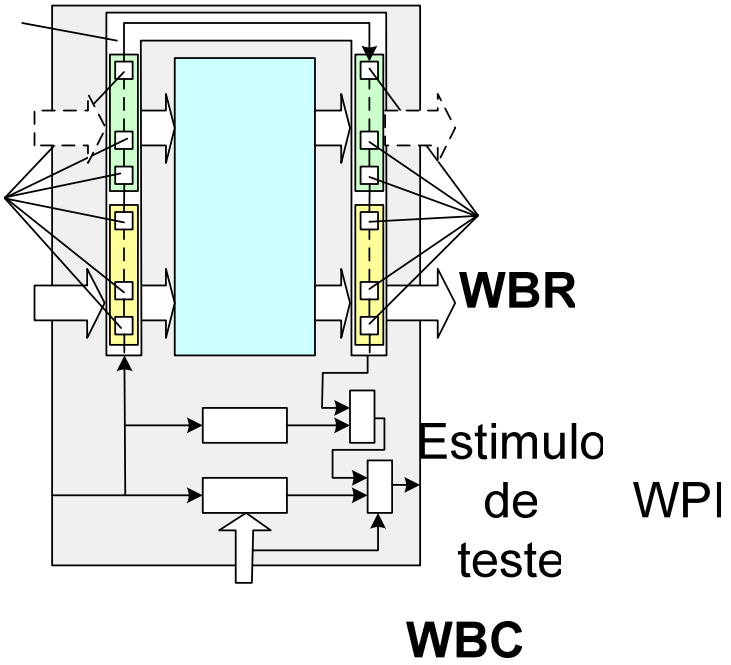


Figura 2.15: Arquitectura do wrapper proposto pela norma IEEE 1500 [7].

A sua arquitectura inclui entradas e saídas funcionais associadas às respectivas entradas e saídas do núcleo. Para além destes sinais a norma obriga a especificação de uma interface de entrada/saída série, respectivamente *Wrapper Serial Input* (WSI) e *Wrapper Serial Output* (WSO), e opcionalmente um conjunto de interfaces de entrada/saída série paralelas, denominadas por *Wrapper Parallel Input* (WPI) e *Wrapper Parallel Output* (WPO), com larguras parametrizáveis podendo variar para cada *wrapper* dentro do SoC.

Para o controlo do *wrapper* observa-se a inclusão da respectiva interface, *Wrapper Interface Port* (WIP), e de um registo de instrução, *Wrapper Instruction Register* (WIR), no qual são carregadas as instruções que definem o respectivo modo de operação do *wrapper*. O WIP é responsável pelo controlo do WIR e é constituído pelos sinais ilustrados na figura 2.16.

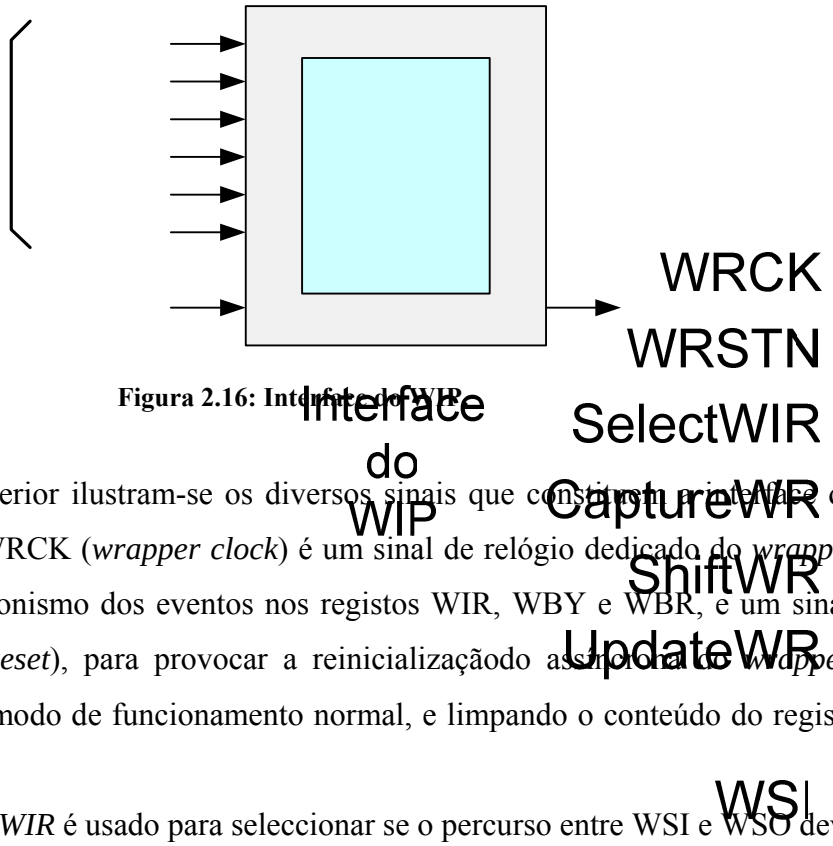
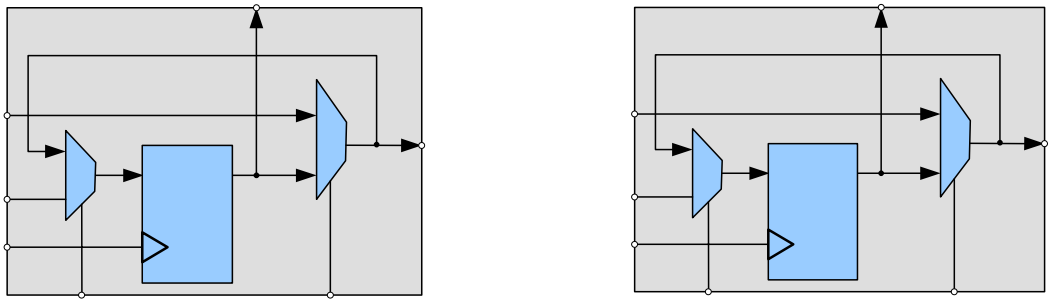


Figura 2.16: Interface do WIP

Na figura anterior ilustram-se os diversos sinais que constituem a interface do *wrapper* (WPI). O WRCK (*wrapper clock*) é um sinal de relógio dedicado do *wrapper* para efectuar o sincronismo dos eventos nos registos WIR, WBY e WBR, e um sinal, WRSTN (*wrapper reset*), para provocar a reinicializaçãodo assincrona do *wrapper*, colocando-o no seu modo de funcionamento normal, e limpando o conteúdo do registo de instrução.

O sinal *SelectWIR* é usado para seleccionar se o percurso entre WSI e WSO deve fornecer informação de teste ou instruções. Com este sinal no nível lógico ’1’, a instrução de teste é deslocada para o interior do WIR, colocado entre o WSI e WSO, caso contrário o registo para o qual a informação deve ser colocada é seleccionado pela instrução presente no WIR. Os restantes sinais, *CaptureWR*, *ShiftWR* e *UpdateWR*, são usados para capturar, deslocar e actualizar a informação para o registo seleccionado, respectivamente.

O acesso aos terminais do núcleo é garantido pelo *Wrapper Boundary Register* (WBR), o qual assegura a controlabilidade e a observabilidade dos respectivos terminais, através de células que garantem as funcionalidades necessárias, denominadas por *Wrapper Boundary Cells* (WBC), e que se ilustram na figura seguinte.

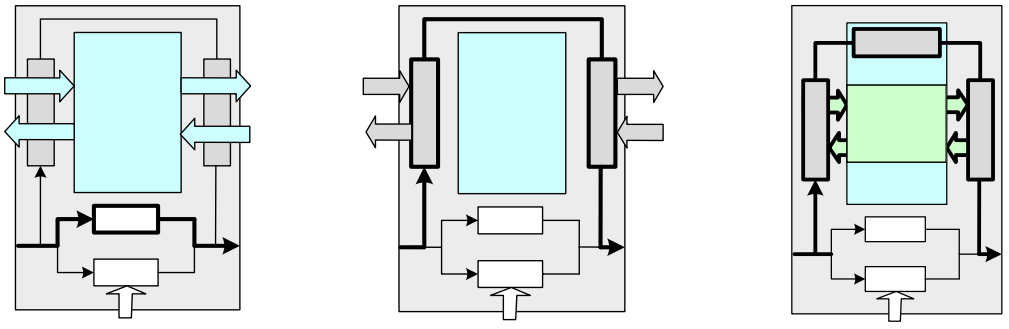


(a) Célula de entrada do WBR **(b) Célula de saída do WBR**

Figura 2.17: Células de entrada (a) e saída (b) que constituem o Wrapper Boundary Register [7].

O *Wrapper Bypass Register* (WBY) garante um percurso mínimo entre o WSI e WSO, permitindo uma movimentação de dados rápido entre os wrappers ligados através de uma cadeia série entre si.

A instrução, presente no WIR, em conjunto com os sinais do WIP determina um dos vários modos de operação (normal, bypass, serial intest, …) suportados pelo *wrapper*. A norma define três instruções obrigatórias (*wBypass, wExtTestS, wInTestS*) sendo as restantes, todas as que se encontram relacionadas com a interface paralela, opcionais. Nas figuras seguintes, encontram-se ilustrados os vários modos de operação suportados pelo *wrapper*.



(a) wBypass **(b) wExtTestS** **(c) wInTestS**

Figura 2.18: Modos de funcionamento do wrapper de acordo com as instruções obrigatórias [7].

A figura 2.18 apresenta as ligações dos núcleos ao nível do sistema, recorrendo a *wrappers* ligados por um TAM série (obrigatório) e uma TAM paralela (opcional).

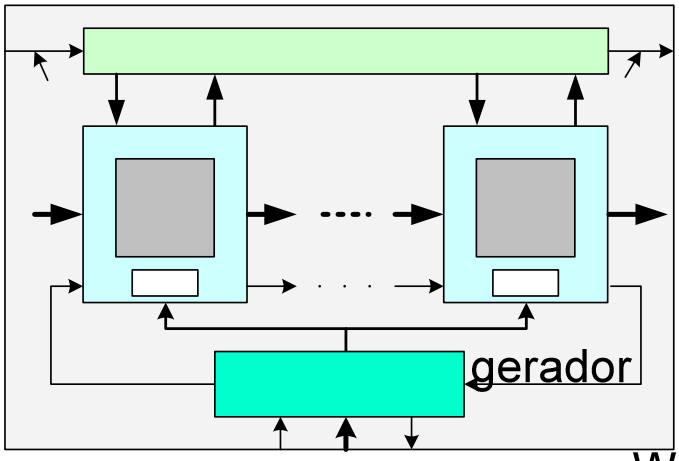


Figura 2. 19: Arquitectura geral de um SoC de acordo com a norma IEEE 1500 [7].

2.7. Conclusão

Neste capítulo foram apresentados alguns conceitos que visam facilitar o teste de microssistemas integrados, criando caminhos para a geração dos estímulos de teste e a captura das respectivas respostas. Porém, as soluções para a geração e captura podem ter origem externa ou interna ao microssistema, devendo ser definidas pelo utilizador.

Na revisão bibliográfica efectuada, foram apresentadas algumas propostas académicas e industriais para o teste de microssistemas integrados, que permitem solucionar os desafios relacionados com o teste de blocos, mecanismos de acesso e solucionar a padronização da interface dos blocos.

Ao longo do próximo capítulo apresenta-se uma arquitectura para um processador genérico de teste, que tenta solucionar os desafios impostos pelo teste de microssistemas integrados.

CAPÍTULO 3

Processador genérico para o teste de dispositivos analógicos e mistos em microssistemas integrados

Neste capítulo apresenta-se uma arquitectura proposta para um processador genérico para o teste de dispositivos analógicos e mistos em microssistemas integrados. A funcionalidade que o processador deve proporcionar é apresentada em primeiro lugar, discutindo-se os benefícios e requisitos que uma solução deste tipo deve suportar, efectuando-se, de seguida, a identificação das operações elementares. A arquitectura é apresentada de uma forma modular, de modo a permitir a identificação dos diversos blocos que constituem o processador. A especificação do conjunto de instruções a suportar é efectuada a partir das operações elementares identificadas, e inclui a descrição da sequência de transição de estados para a execução de cada instrução.

3.1. Benefícios e requisitos

O recurso às abordagens de teste convencionais para efectuar a testabilidade de núcleos de hardware em SoCs é praticamente impossível, uma vez que estas abordagens não cobrem na sua totalidade os requisitos de teste impostos pelas dezenas, ou mesmo centenas de núcleos profundamente integrados em SoCs.

A geração, transporte e a captura dos sinais analógicos é um dos aspectos críticos no teste de núcleos analógico-digitais, pelo que o recurso a estruturas BIST para levar a cabo estas operações é a escolha mais acertada. O BIST reduz o custo e o tempo associado ao teste, diminuindo o ciclo de concepção do dispositivo, minimizando o

volume de dados com algoritmos de compactação integrados e reduzindo significativamente os requisitos exigidos ao equipamento de teste externo (ATE). Uma outra razão que suporta o interesse pela existência de um BIST, consiste no facto de a sua aplicabilidade se estender a diversos cenários de teste (desenvolvimento e verificação, produção e manutenção), possibilitar o teste nos diversos níveis da hierarquia do SoC, simplificando o projecto de teste do sistema completo, para além de permitir o teste em tempo de execução cobrindo falhas de características temporais.

Porém, o uso de BIST está associado a custos adicionais. É necessário mais área de silício para os módulos de geração, avaliação e controlo de teste, o que pode conduzir a custos potencialmente elevados. Por outro lado, a presença de multiplexadores adicionais introduz degradação do desempenho, podendo mesmo inviabilizar o seu uso em arquitecturas que necessitem de um elevado desempenho.

No entanto, o BIST pode ser eficientemente implementado recorrendo a lógica reconfigurável ou a processadores incluídos em circuitos de elevada complexidade, que podem ser usados para realizar o teste dos diversos componentes do sistema.

Esta dissertação propõe o recurso a núcleos de lógica reconfigurável, por exemplo FPGAs (FPGA – *Field Programmable Gate Arrays*), PLDs (PLD – *Programmable Logic Devices*), disponíveis no SoC para a implementação do processador com arquitectura reconfigurável [31] dedicado ao teste de núcleos analógicos e mistos. O recurso a este tipo de dispositivos tem ganho uma aceitação crescente entre a comunidade internacional [32] [33] [34].

Esta solução garante uma maior flexibilidade relativamente às propostas que recorrem a processadores genéricos adaptados (DSP, processadores, microcontroladores, entre outros) para o teste ou a processadores específicos para teste (ASIP – *Application Specific Instruction-set Processor*), com funcionalidade fixa, uma vez que, em cada instante, a FPGA pode ser configurada como processador específico de teste, para controlar e escalonar as operações de teste, de acordo com as necessidades de teste a efectuar.

Deste modo, garante-se uma fácil reutilização e modificação de diversas técnicas de teste, evitando o risco de o processador se tornar obsoleto. Por exemplo, para diminuir a probabilidade de mascaramento associadas às rotinas de compressão pode-se optar por executar o teste duas vezes com polinómios diferentes. Por outro lado não

apresenta custos adicionais relativos à área necessária para a sua implementação uma vez que esta é efectuada em recursos disponíveis no SoC.

Uma outra razão que suporta o interesse por esta solução consiste no facto de que, dependendo do espaço disponível na FPGA, uma versão adequada do processador pode ser adoptada de modo a oferecer um conveniente compromisso entre o processamento e o espaço ocupado. De um modo geral, quanto maior for o processamento efectuado pelo processador sobre as respostas obtidas do teste, menor será a quantidade de tráfego de informação de teste entre o SoC e o ATE, reduzindo deste modo o tempo de teste.

Após a realização do teste dos diversos núcleos que compõem o SoC, a lógica reconfigurável poderá ser programada para a implementação de uma das funcionalidades do sistema, resultando na ausência de quaisquer custos associados à adição da capacidade de processamento de resultados do teste.

Este tipo de implementações da lógica reconfigurável pode ser levada a cabo usando barramentos de comunicação como o I^2C [35] ou barramentos de teste como o definido pela norma IEEE 1149.1 [5], o qual poderia ser usado para também para dar início ao processo de teste e enviar para o exterior os resultados do mesmo. Outra solução seria recorrer à norma IEEE 1532 [36] a qual estabelece um protocolo para a programação de lógica reconfigurável.

3.2. Especificação das características para um processador residente

As considerações efectuadas na secção anterior permitem concluir que os objectivos a atingir consistem fundamentalmente em explorar a combinação das potencialidades proporcionadas pelo recurso a um processador de teste e a sua implementação em núcleos de lógica reconfigurável. Nesta secção apresenta-se o conjunto de características que o processador desenvolvido para o teste de núcleos analógico-digitais deve possuir, tendo em vista os objectivos descritos.

3.2.1. Descrição da funcionalidade pretendida

A exploração das potencialidades proporcionadas pelo processador de teste deverá ser assegurada por um conjunto de instruções que permita implementar as operações elementares necessárias para a realização do teste, e cuja identificação terá lugar na próxima secção. As características de funcionalidade apresentadas nesta secção contribuem igualmente para a especificação desse mesmo conjunto de instruções.

Um primeiro factor a ser levado em consideração, consiste na possibilidade de garantir o teste a todos os núcleos (digitais, analógico e mistos) integrados no SoC independentemente da arquitectura exibida por este último e da localização dos respectivos núcleos.

Para este aspecto é possível distinguir duas situações distintas, nas quais os núcleos a testar se encontram directamente ligados ao à FPGA, conforme o ilustrado na figura 3.1, ou então encontram-se localizados em qualquer ponto do SoC. Este último caso é ilustrado na figura 3.2.

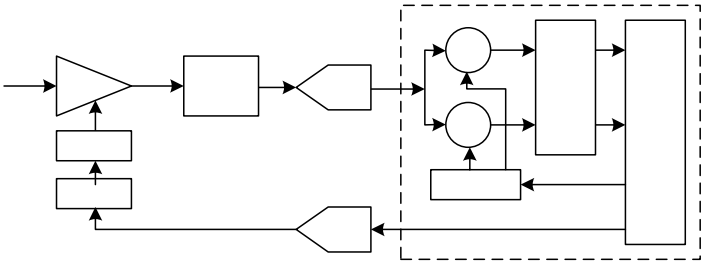


Figura 3.1: Arquitectura genérica de um receptor de comunicação *wireless*.

A figura anterior ilustra a situação onde o núcleo a testar (neste caso a interface de um receptor de radiofrequência) se encontra directamente ligado à FPGA, na qual será implementado o processador de acordo com as operações de teste a realizar. Nesta situação o receptor RF é facilmente isolado dos restantes núcleos do SoC envolventes durante a fase de teste, e os sinais necessários ao teste podem ser directamente aplicados sem ser necessário efectuar qualquer tipo de encaminhamento. O mesmo não se sucede na situação ilustrada na figura seguinte, onde o núcleo a testar é visto como um dispositivo periférico do SoC.

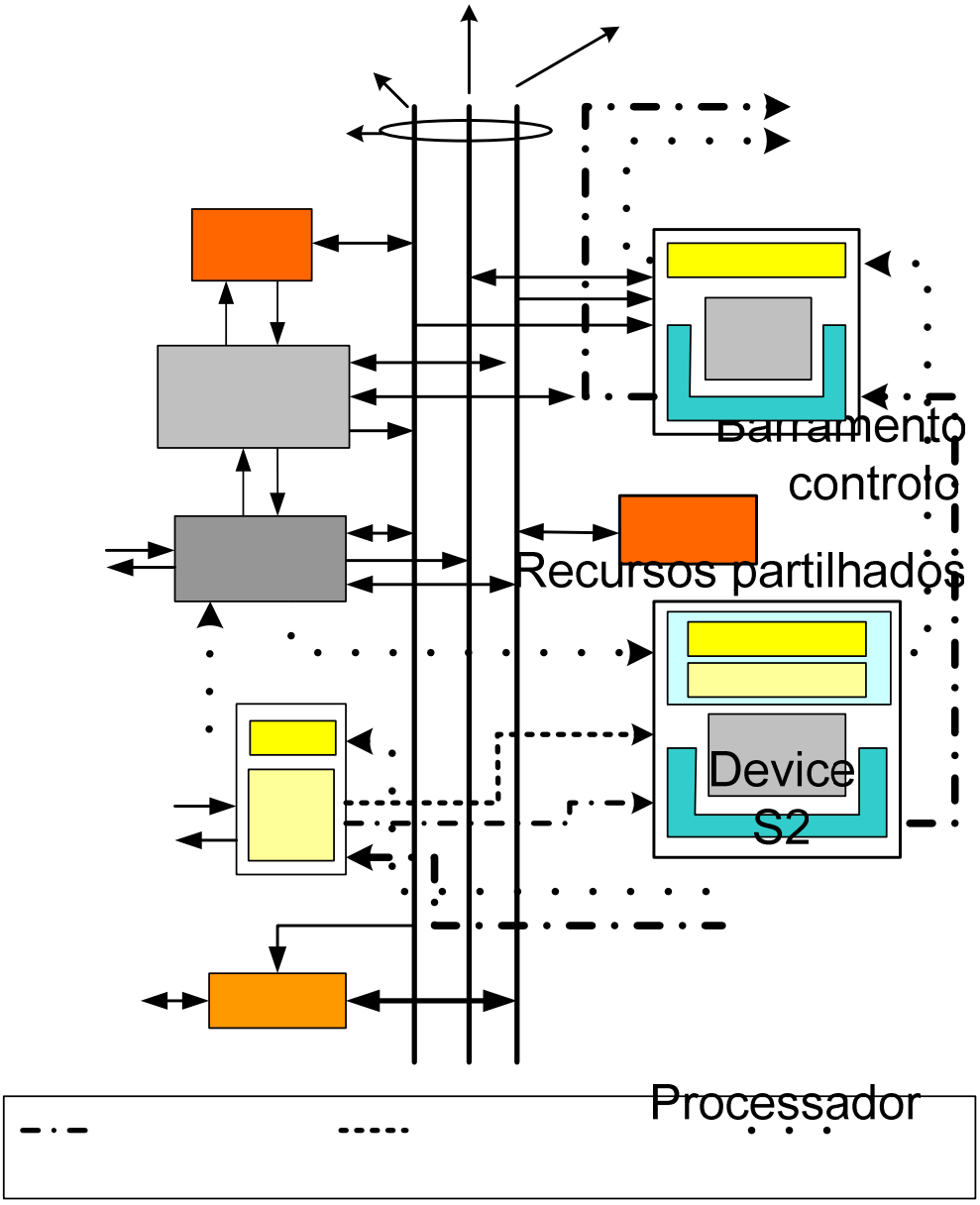


Figura 3.2: Arquitectura da infra-estrutura de teste.

Neste caso, o conversor A/D encontra-se ligado a um barramento do sistema através do qual é efectuado o acesso. O processador de teste, implementado na FPGA, é responsável pelo controlo da infra-estrutura de um modo semelhante ao descrito na secção 2.4 [27] e através da qual é efectuado o acesso e o isolamento do conversor durante a fase de testes. No entanto, caso seja necessário salvaguardar na memória as respostas obtidas do teste, para efectuar algum tipo de processamento, é fundamental garantir o acesso ao barramento do sistema. Nesta situação, o processador deve disputar

o acesso ao barramento do sistema com os recursos que o partilham, sem o risco de contenção. De modo a transferir os dados entre o conversor e a memória, é implementado pelo processador de teste um mecanismo de DMA, através dos sinais HRQ (HRQ – *Hold ReQuest*) e HLDA (HLDA – *HoLD Acknowledge*), que garante o acesso total ao barramento.

Uma arquitectura deste tipo é comum encontrar-se dividida em subdomínios, de modo a assegurar que os núcleos a testar se encontrem totalmente isolados, facilitando as operações de teste do *SoC*, de um modo idêntico ao que se sucede para o teste de estruturas hierárquicas.

Em ambas as situações descritas é necessário recorrer a interfaces (*wrappers*) adequados para assegurar o isolamento apropriado dos núcleos durante a fase de teste. Isto diz respeito em particular ao seu controlo e observação. Por outro lado é necessário distinguir os sinais usados durante o modo normal de funcionamento dos sinais de teste, os quais envolvem sinais estáticos e dinâmicos.

Conforme descrito anteriormente (secção 2.4), os sinais estáticos são sinais que permanecem constantes ou variam ocasionalmente durante a fase de teste, usados fundamentalmente na configuração do modo de operação, podendo ser encaminhados através de uma infra-estrutura série. O protocolo IEEE 1149.1 [5] pode ser usado para propagar este tipo de sinais sendo necessário associar ao processador de teste um controlador adequado responsável pela gestão do protocolo. É também possível implementar no FPGA células adequadas para a distribuição dos sinais, as quais não são fornecidas pelos *wrappers*.

O mesmo não sucede com os sinais dinâmicos, uma vez que estes variam constantemente durante o teste (sinal de relógio, estímulo de teste, …), sendo necessário garantir um acesso directo ao núcleo a testar, por parte dos sinais de teste gerados pelo processador e os provenientes do sistema. Para tal é fundamental a inclusão de células de entrada/saída adequadas que garantam esta comutação, as quais permitem que o acesso seja efectuado em série ou em paralelo. A figura seguinte ilustra esta situação.

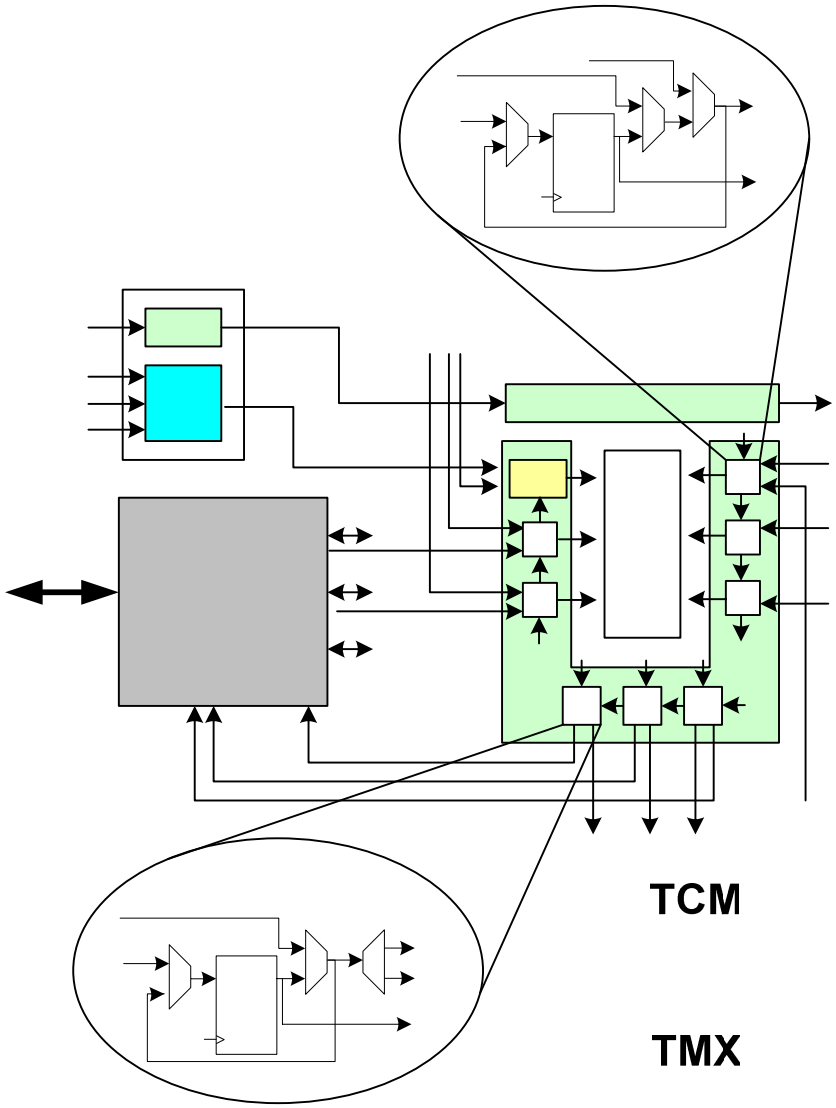


Figura 3.3: Interface entre o núcleo a testar e o processador de teste.

O *wrapper*, que envolve o ADC, inclui células adequadas para comutar as entradas do núcleo entre os sinais gerados durante a operação de teste, sejam eles estáticos ou dinâmicos, e os sinais que são envolvidos durante o modo de funcionamento normal. De um modo geral, estes sinais encontram-se ligados a outros núcleos do sistema, enquanto que os sinais de teste se encontram directamente ligados à FPGA. Sempre que necessário podem ser usados registos adicionais (*TCM* e *TMX*) [27] para garantir a acessibilidade de sinais de teste que provenham de algum ATE.

É também necessário ter em consideração que os diversos núcleos, disponíveis no SoC, possuem necessidades de teste distintas, pelo que normalmente requerem a

aplicação de estímulos de teste com características distintas, e estes devem-se encontrar bem caracterizados. É importante que os aspectos relacionados com o controlo temporal e a sincronização dos eventos analógicos e digitais. Por exemplo, no caso do teste de conversores A/D, é necessário que a relação entre a frequência do estímulo de teste, f_0, e a frequência de amostragem, f_a, seja tal que garanta a coerência da amostragem [37] [38], isto é, que o número de amostras capturadas seja um número inteiro de ciclos do estímulo e que as fases instantâneas dos ciclos do sinal de entrada se encontrem uniformemente distribuídas pelo intervalo [0, 2π]. Ambas as condições são asseguradas se a condição seguinte for satisfeita.

$$f_0 = \left(J \diagup M\right) \times f_a \qquad (3.1)$$

Onde,

M – é o número de amostras capturadas,

J – o número inteiro de ciclos do estímulo e que deverá ser primo relativamente a M.

Caso a amostragem não seja coerente, a probabilidade de ocorrência de todos os códigos não é constante existindo o risco das amostras serem capturadas sempre nos mesmos pontos em diferentes ciclos do sinal de entrada. A falta de coerência na amostragem tem implicações diferentes nos métodos de teste adoptados.

Um factor importante que o processador deve garantir é a capacidade de processamento das respostas obtidas do teste, de modo a minimizar a quantidade de informação transmitida para o exterior como resultado de teste. Esta deve ser a mais reduzida possível de modo a minimizar o tempo de transmissão dessa informação e a necessidade de operações de processamento posteriores, pelo equipamento externo. No limite, o resultado de teste poderá ser simplesmente uma classificação do funcionamento do circuito como correcto ou incorrecto.

Concluída a apresentação das funcionalidades que o processador deve garantir, procede-se, na secção seguinte, à identificação das operações elementares.

3.2.2. Operações elementares

As operações elementares requeridas determinam o conjunto de instruções a ser suportado pelo processador, o qual para além de garantir o controlo (aplicação de uma entrada conhecida, isto é, um estímulo adequado) e a observação (observar se a resposta obtida do núcleo é a desejada ou não) dos diversos núcleos durante o teste, deve possibilitar o teste das ligações entre os respectivos núcleos.

Deste modo, o processador deve suportar as operações adequadas ao teste de núcleos de hardware que compõem o SoC, das quais se destacam:

- **Geração dos padrões/estímulos de teste**

O processador deverá possuir a capacidade de gerar os estímulos de teste para caracterizar o desempenho do núcleo que se pretender testar. Para este efeito, a geração dos estímulos deve ser flexível uma vez que devido à variedade de núcleos analógicos e analógico-digitais integrados no SoC, é necessário aplicar um estímulo que vá ao encontro das necessidades de teste apresentadas por cada núcleo.

Por exemplo, para efectuar a caracterização completa de um ADC é necessário obter não só os seus parâmetros estáticos, que caracterizam a função de transferência do conversor, como também os parâmetros dinâmicos, que apresentam a informação relativa ao seu comportamento dinâmico. Enquanto que os primeiros são obtidos na presença de sinais de baixa frequência (muito menor que a sua frequência de amostragem), os parâmetros dinâmicos são obtidos para sinais de frequência relativamente elevada, próximas da frequência de amostragem do conversor.

- **Captura e avaliação das respostas de teste**

Ao contrário do que sucede com as técnicas de teste tradicionais, onde os valores de teste são comparados com um valor esperado pré-processado, em BIST isto é inviável, devido à grande quantidade de memória necessária para armazenar o volume de dados. A capacidade de efectuar operações de processamento adequadas, particularmente operações de compactação das respostas de teste, permite reduzir o

volume de dados a um tamanho aceitável que ainda seja capaz de detectar falhas existentes sem a ocorrência de fenómenos de erro.

- **Controlo da infra-estrutura de teste.**

O processador deve controlar os modos de operação básicos e opcionais da infra-estrutura de teste, definidos na norma IEEE 1149.1 [5] e IEEE 1149.4 [6]. Esta capacidade destina-se a permitir a implementação das operações de teste que recorrem à utilização dos referidos barramentos e aos seus modos de operação (teste externo, teste interno, …).

- **Configuração dos modos de teste e a propagação dos padrões de teste e respectivas respostas**

Independentemente da arquitectura do SoC o processador deverá garantir o acesso ao núcleo e a configuração dos respectivos modos de operação, quer o núcleo a testar se encontre directamente acoplado ao processador ou se encontre localizado em qualquer outro ponto.

Com base nestes requisitos é possível delinear um primeiro esboço da interface, ou seja, dos pinos de entrada e saída do controlador, conforme se ilustra na figura 3.4. Face ao desenvolvimento do processador, poderá resultar a necessidade de incluir mais pinos de entrada e saída.

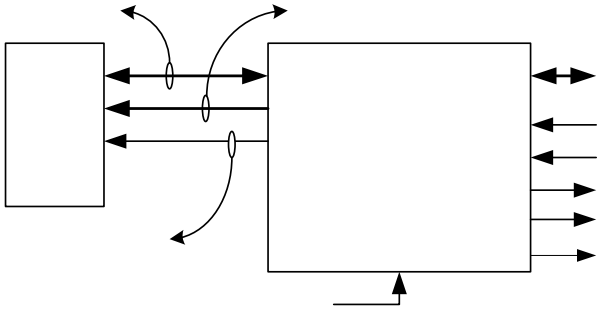


Figura 3.4: Definição inicial da interface do processador de teste.

A reutilização do processador para o teste de núcleos com características distintas pode implicar a inclusão de pinos, não identificados nesta secção, uma vez que não se referem aqui todos os pormenores da sua arquitectura e a interface com o exterior.

3.3. Arquitectura do processador

Esta secção descreve a arquitectura do processador de teste, de acordo com as especificações apresentadas nas secções precedentes. A apresentação da arquitectura inicia-se com a sua caracterização global, seguida da descrição detalhada de cada uma das unidades. A partir do conjunto de operações elementares, descritas nas secções anteriores, procede-se à apresentação do conjunto de instruções e finalmente de cada uma das instruções, ilustrando-se a respectiva sequência de transição de estados.

3.3.1. Caracterização Global

Esta secção tem por objectivo efectuar a caracterização global da arquitectura adoptada para a implementação do processador específico para o teste de núcleos analógicos e mistos que compõem o SoC.

De acordo com as especificações apresentadas nas secções precedentes, relativas à funcionalidade que o processador deve garantir, a arquitectura é constituída por três partes principais:

- Unidade de controlo

Este bloco constitui o núcleo do processador de teste. Suporta directamente o conjunto de operações elementares, descritas nas secções anteriores, sendo responsável pelo controlo dos recursos internos e controlo do fluxo do programa de teste.

- Blocos genéricos

Para além dos módulos (unidade de registos, controlador de acesso à memória, entre outros) que constituem o processador de teste, módulos adicionais com funcionalidades distintas são integrados no processador de modo a garantir a completa exploração das potencialidades proporcionadas pelo processador de teste. Como exemplo das funcionalidades suportadas, temos as destinadas ao controlo da infra-estrutura de teste, geração dos padrões de teste e a captura das respectivas respostas.

- Memória

A memória para além de conter o programa de teste a ser executado, inclui também os vectores responsáveis pela geração dos padrões de teste e pode ser usada para salvaguardar as respostas de teste. Esta memória poderá ser ou não exterior ao processador de teste. No caso de uma memória exterior, existe a vantagem de libertar espaço extra, o qual poderá ser explorado de modo a oferecer um compromisso adequado entre o processamento e o espaço ocupado pelo processador.

A figura 3.5 ilustra a arquitectura global descrita para o processador de teste, onde se visualizam as unidades identificadas anteriormente.

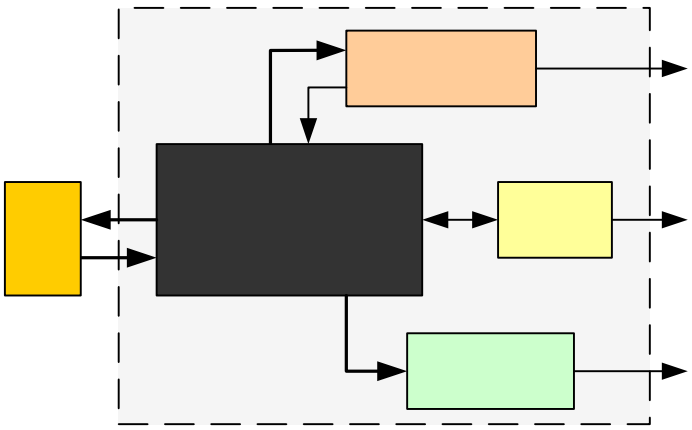


Figura 3.5:Arquitectura global do processador de teste.

A definição da largura para o barramento de dados do processador merece uma análise cuidada, em que sejam equacionados os diversos factores em jogo. Apesar de um barramento de dados com um maior número de bits proporcionar uma maior velocidade no deslocamento de dados, os seguintes factores devem ser considerados:

- O número de operações elementares identificadas é reduzido, pelo que o número de bits para codificar as respectivas instruções também será reduzido.
- Um barramento de dados com um menor número de bits implica um maior número de acessos à memória, o que pode implicar uma degradação do desempenho do processador em termos de velocidade. Esta degradação pode ser minimizada através da exploração do paralelismo.
- Um barramento de dados com um maior número de bits obriga a que interface do processador com a memória, quando esta lhe for exterior, possua um maior número de pinos.

Todos estes factores devem ser considerados da adopção do tamanho do barramento de modo a que as necessidades de teste impostas pelo núcleo a testar sejam satisfeitas. Como solução de compromisso satisfatória foi adoptado um barramento de dados com 8 bits.

A determinação da largura do barramento de endereços enfrenta mais dificuldades, sobretudo pela complexidade em quantificar a capacidade de memória necessária para o programa de teste. Esta capacidade depende de diversos factores, tais como o número de dispositivos a testar, a quantidade de estímulos a aplicar e o volume de repostas de teste a capturar. Este último factor é particularmente importante, podendo mesmo inviabilizar a utilização do processador se a capacidade de memória requerida para salvaguardar as respostas de teste não for demasiado elevada.

Face ao exposto, uma memória endereçada por um barramento de 16 bits constitui uma boa solução para conter o programa e estímulos de teste, para além de armazenar as respostas de teste.

Na figura seguinte apresenta-se o conjunto de pinos do processador de teste, cuja descrição se encontra sintetizada na tabela 3.1.

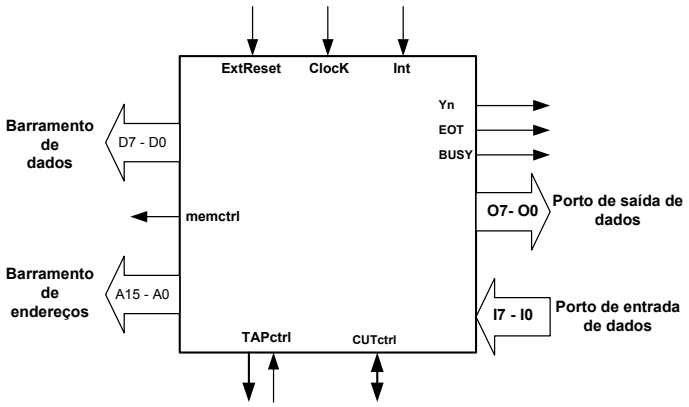


Figura 3.6: Conjunto de pinos do processador de teste.

Tabela 3.1: Funcionalidade definida para os pinos do processador de teste[1].

Funcionalidade definida para o conjunto de pinos do processador de teste		
Designação	**Nº de pinos**	**Funcionalidade**
A[15:0]	16	Barramento de endereços
D[7:0]	8	Barramento de dados
memctrl	2	Controlo de acesso à memória (escrita/leitura)
O[7-0]	8	Porto de saída de dados
I[7-0]	8	Porto de entrada de dados
CUTctrl	?	Porto de controlo do dispositivo a testar
TAPctrl	?	Porto de controlo da infra-estrutura de teste
Busy	1	Saída que indica o estado do processador (livre - '0', ocupado – '1')
EOT	1	Saída que indica o fim de uma operação de teste (EOT = 1).
Yn	1	Saída do estímulo de teste
Int	1	Entrada de interrupão/sincronismo
ExtReset	1	Entrada de inicialização do processador
Clock	1	Entrada de relógio do processador

[1] Na tabela 3.1 o carácter '?' sinaliza que a dimensão do barramento é desconhecida, dependendo dos requisitos impostos pelo dispositivo a testar.

3.3.2. Arquitectura

Esta secção apresenta uma arquitectura do processador de teste referido na figura 3.5, destinada a garantir a completa exploração das potencialidades proporcionadas pelo processador de teste. Para além dos blocos identificados na secção anterior procede-se à identificação dos restantes que constituem a arquitectura.

Na figura 3.7 ilustra-se a arquitectura definida para este processador, onde se identificam os blocos que integram a arquitectura interna, dos quais se destacam os seguintes elementos:

- Apontador de programa (PROGCNT),
- Apontador de segmento de dados (DSctrl),
- Apontador de segmento de vectores de teste (SSctrl),
- Bloco de controlo do barramento de dados (MDATActrl),
- Bloco de controlo do barramento de endereços (MADDRctrl),
- Bloco de controlo e descodificação das instruções (CTRLUNIT),
- Unidade de registos (REGS),
- Unidade aritmética e lógica (ALU),
- Bloco de controlo da infra-estrutura de teste (BSctrl),
- Unidade de geração dos estímulos de teste (STIMULUSctrl),
- Bloco de controlo do modo de operação (CUTctrl),

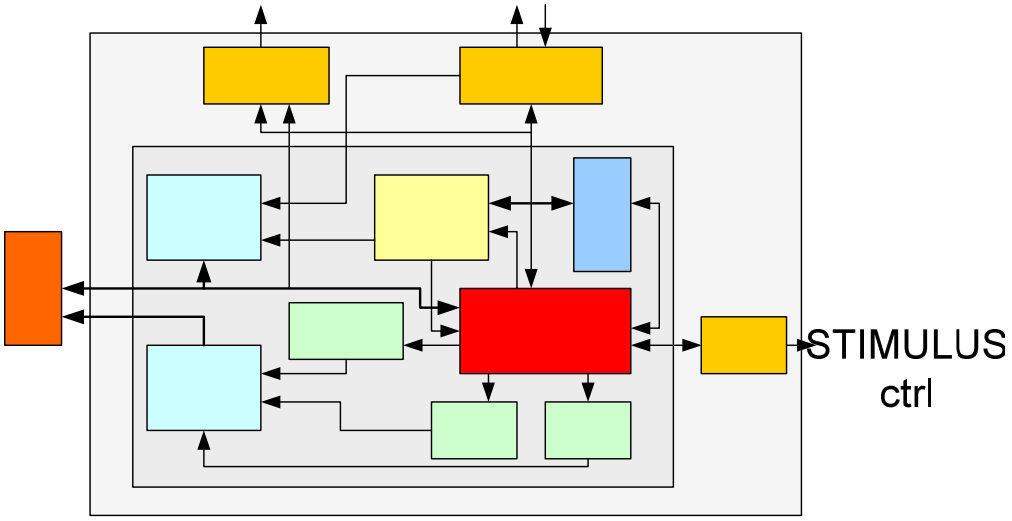


Figura 3.7: Arquitectura interna do processador de teste.

Efectua-se de seguida a caracterização de cada um dos blocos ilustrados na figura anterior.

3.3.2.1. Blocos principais

Nesta subsecção efectua-se a caracterização individual de cada um dos blocos identificados na figura 3.7. Esta caracterização é efectuada ilustrando-se a respectiva interface e descrevendo a funcionalidade implementada.

Apontador de programa (PROGCNT)

A figura 3.8 ilustra a interface deste bloco, responsável por manter actualizado o endereço para o segmento de memória que contém o programa de teste.

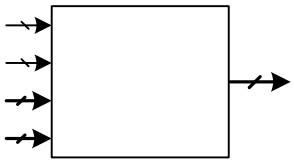


Figura 3.8: Interface do apontador de programa.

O apontador de programa é constituído por um registo síncrono de 8-bits, com possibilidade de suportar carga paralela, para suportar instruções de salto condicional.

Apontador de segmento de dados (DSctrl)

Na figura 3.9 ilustra-se a interface deste bloco, responsável por manter actualizado o endereço para o segmento da memória de dados, onde se armazenam as repostas obtidas do teste efectuado. É constituído por um contador síncrono de 16-bits.

Figura 3.9: Interface do apontador de segmento de dados.

Apontador de segmento de vectores de teste (SSctrl)

A figura 3.10 ilustra a interface deste bloco, responsável por manter actualizado o endereço para o segmento de memória onde se encontram os vectores digitais de teste usados para aplicar ao gerador de estímulos de teste. É constituído por um contador síncrono de 8-bits.

Figura 3.10: Interface do apontador de segmento de vectores de teste.

Bloco de controlo do barramento de dados (MDATActrl)

Na figura 3.11 ilustra-se a interface deste bloco, responsável pelo controlo do barramento de dados, administrando todos os ciclos de escrita da memória, por parte dos módulos que o requeiram.

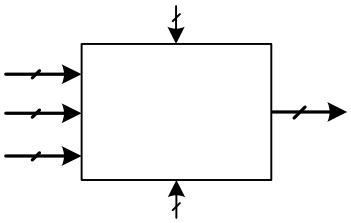


Figura 3.11: Interface do bloco de controlo do barramento de dados.

Bloco de controlo do barramento de endereços (MADDRctrl)

A figura 3.12 ilustra a interface deste bloco, responsável pelo controlo do barramento de endereços para a memória de programa e dados, isto é, controlo o acesso à memória de todos os ciclos, sejam estes de leitura/escrita de dados, leitura do código de instrução e dos vectores de teste.

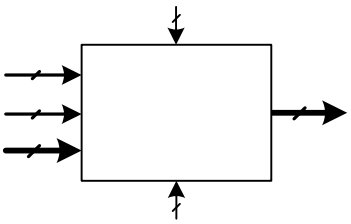
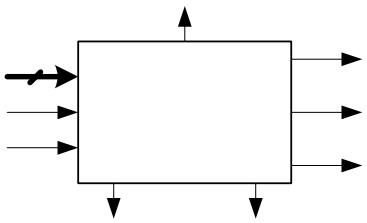


Figura 3.12: Interface do bloco de controlo do barramento de endereços.

Bloco de controlo e descodificação das instruções (CTRLUNIT)

É o núcleo do controlador de teste, uma vez que suporta directamente o conjunto de instruções. É constituído por um registo de instrução, destinado a receber o código de cada instrução a executar, por uma máquina de estados, que se responsabiliza por gerar a sequência de estados a percorrer de acordo com o código presente no registo de instrução, e por um circuito combinacional, responsável pela geração dos sinais que controlam os diversos blocos que integram a arquitectura do processador.



Controlo RE

Figura 3.13: Interface do bloco de controlo e descodificação das instruções.

Unidade de registos (REGS)

Barramento 8
dados

A figura 3.14 ilustra a interface da unidade de registos, que contém os registos internos do processador. Compreende 6 registos (R0,R1, …R5) de 8-bit para uso geral, sendo R0 um registo de leitura, e dois contadores (CNT1, CNT2) também de 8-bits específicos para operações de salto.

clk

CTRLUNI

reset

Controlo I/O Con

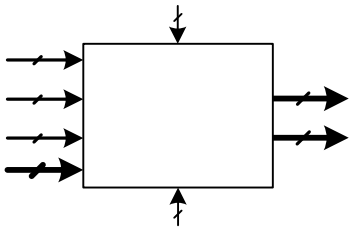


Figura 3.14: Interface da unidade de registos.

Unidade aritmética e lógica (ALU)

A unidade aritmética e lógica é responsável por realizar operações aritméticas simples (ADD, SUB, INC, …) e lógicas (AND, OR, NOT, …) sobre os dados contidos nos registos. De acordo com a operações necessárias a realizar podem ser implementadas funcionalidades adicionais para o suporte de operações mais complexas como multiplicação e divisão. A figura seguinte ilustra a interface deste bloco.

wreg 2

Figura 3.15: Interface da unidade aritmética e lógica.

Pode-se recorrer a esta unidade, caso de pretendam realizar operações sobre o resultado das respostas de teste.

Bloco de controlo da infra-estrutura de teste (BSctrl)

Este bloco destina-se a controlar uma infra-estrutura de teste compatível com o protocolo IEEE 1149.1 [5]. Possui dois registos de carga paralela, um deles destinado a controlar a máquina de estados do TAP através do sinal TMS, enquanto o outro se destina ao deslocamento para o interior da cadeia BS, através do sinal TDO, da sequência de bits necessária à programação do(s) registo(s) de instrução e de dados. Paralelamente à operação anterior, procede-se ao carregamento de um registo série com a sequência de bits deslocada do interior da infra-estrutura BS através do sinal TDI.

Figura 3.16: Interface do bloco de controlo da infra-estrutura de teste.

Unidade de geração dos estímulos de teste (STIMULUSctrl)

A figura 3.17 ilustra a interface deste bloco, responsável pela geração do(s) estímulo(s) de teste adequado(s) tendo em consideração o dispositivo 'alvo' a testar. Após a recepção do vector digital de estímulo este bloco gera continuamente o respectivo estímulo de teste sem interferência do processador de teste.

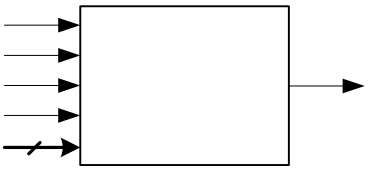


Figura 3.17: Interface da unidade geração dos estímulos de teste.

Bloco de controlo do modo de operação (CUTctrl)

A figura 3.18 ilustra uma interface deste bloco, responsável pelo controlo do dispositivo a testar.

Figura 3.18: Interface do bloco de controlo do modo de operação.

A interface ilustrada na figura anterior representa uma interface genérica, uma vez que o controlo a efectuar por este bloco depende dos requisitos impostos pelo tipo de teste a realizar e pelo dispositivo a testar. Este bloco pode variar de acordo com os objectivos de teste

No capítulo seguinte descreve-se a metodologia e a geração automática do processador de teste, que permite implementar o processador de teste de acordo com os requisitos impostos.

3.4. Especificação das instruções suportadas

Nesta secção apresenta-se o conjunto de instruções suportadas pelo processador de teste, de acordo com as operações elementares descritas, que permitem controlar a infra-estrutura de teste, gerar os estímulos de teste, efectuar operações de processamento, gerir os recursos internos, entre outras.

3.4.1. Definição do conjunto de instruções

Esta secção apresenta o conjunto de instruções mínimo que deve ser suportado pelo processador de teste. Este conjunto possui uma correspondência directa com as operações elementares identificadas nas secções precedentes, e são apresentadas nas tabelas 3.2 a 3.4.

Tabela 3. 2: Instruções para o controlo da infra-estrutura IEEE 1149.4.

Instruções para o controlo da infra-estrutura	
Instrução	**Operação elementar**
TRST	Aplica um impulso ao nível lógico '0' na linha TRST da infra-estrutura BS.
STATE TMSdata	Comanda as transições da máquina de estados do controlador TAP, de acordo com a sequência de bits especificada pelo operando **TMSdata**.
NSHF d2shf	Efectua o deslocamento para o interior da cadeia uma sequência de bits, especificada pelo operando **d2shf**.

Tabela 3.3: Instruções para suporte às operações de teste.

Instruções para suporte às operações de teste	
Instrução	**Operação elementar**
SRTEST	Inicia o procedimento de teste aplicando os sinais de controlo ao núcleo a testar.
EOTEST	Sinaliza o fim do procedimento de teste.
STIMULUS	Aplica um vector digital ao módulo responsável pela geração dos estímulos de teste.

Tabela 3.4: Instruções para o controlo dos recursos internos, canais de sincronismo, e do fluxo do programa de teste.

Instruções para o controlo dos recursos internos, canais de sincronismo, e do fluxo do programa de teste.	
Instrução	**Operação elementar**
LD Reg#, (Endereço)	Carrega o registo Reg# com o conteúdo da posição de memória especificada pelo endereço.
ST (Endereço), Reg#	Transfere o conteúdo do registo Reg# para a posição de memória especificada pelo endereço.
MOVC Reg#, N	Carrega o registo **Reg#** com o número N.
DJNZ Reg#, Endereço	Implementa um salto condicional, de acordo com o conteúdo do registo, para o endereço especificado.
JMP Endereço	Implementa um salto incondicional para o endereço especificado.
WAIT0, WAIT1	Espera até que a entrada de sincronismo se encontre no valor lógico pretendido.
NOP	Não realiza qualquer tipo de operação.
HLT	Termina a execução do programa

Além das operações descritas atrás, o conjunto de instruções contém operações aritméticas e lógicas e instruções adicionais, relativas à transferência de dados. Estas instruções encontram-se descritas no anexo A.

Cada uma das instruções apresentadas nas tabelas anteriores é descrita em seguida de uma forma mais pormenorizada, nomeadamente através do seu diagrama de transição de estados e, em alguns casos, com exemplos que ilustram a sua aplicação.

3.4.2. Caracterização individual

Nesta subsecção efectua-se a caracterização individual de cada uma das instruções identificadas na secção precedente.

Instruções para o controlo da infra-estrutura de teste

Apresenta-se de seguida a caracterização individual das instruções destinadas ao controlo da infra-estrutura de teste.

TRST

Esta instrução aplica um impulso ao nível lógico baixo '0' na linha TRST (TRST – *Test Reset*) da infra-estrutura BS, provocando uma inicialização assíncrona da sua máquina de estados. A inicialização da infra-estrutura de teste, por esta via, implica que todos os componentes compatíveis com a norma IEEE 1149.1 [5] disponham deste pino opcional, caso contrário será necessário aplicar a sequência de cinco impulsos em TCK (TCK – *Test Clock*), mantendo a linha TMS (TMS – *Test Mode Select*) ao nível lógico alto '1'.

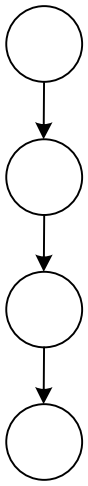


Figura 3.19: Diagrama de transição de estados da instrução TRST.

STATE TMSdata

Esta instrução comanda as transições da máquina de estados do controlador TAP, permitindo seleccionar o estado desejado. A sequência de bits para atingir o estado desejado, especificado pelo operando, encontra-se armazenada nas duas posições de memória seguintes à que contém o código da instrução. A primeira destas posições

contém os oito bits mais significativos a carregar e a segunda posição contém os oito bits menos significativos. A linha TMS é mantida no nível lógico desejado ('0'/'1') durante a transição ascendente do relógio de teste (TCK), ocorrendo a sua actualização nas transições descendentes no relógio de teste (TCK). A actualização do valor a deslocar para a cadeia consiste em efectuar o deslocamento de 1 bit para a direita do registo **tms_reg** que contém a sequência a deslocar.

Esta instrução deverá ser precedida por outra que carregue o contador com o número de transições pretendidas (**`MOVC CNT1, TCKval`**).

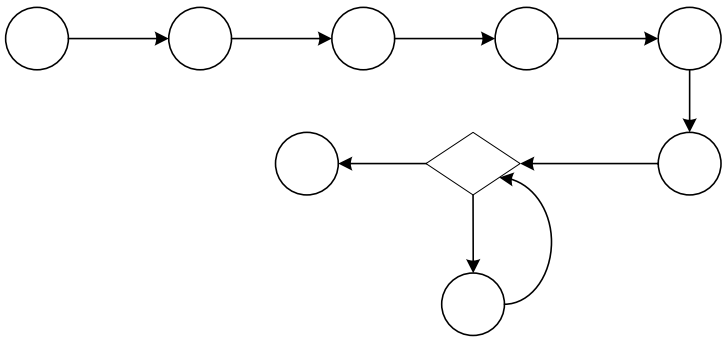


Figura 3.20: Diagrama de transição de estados da instrução STATE.

Os dados que compõem a sequência de bits para atingir o estado deverão estar organizados como se ilustra a figura 3.21, para a execução da instrução **`STATE $19F`**. Estes dados serão deslocados para da esquerda para a direita (do bit mais significativo para o menos significativo), pelo que o primeiro bit a deslocar será o bit menos significativo (D0) da palavra LSB.

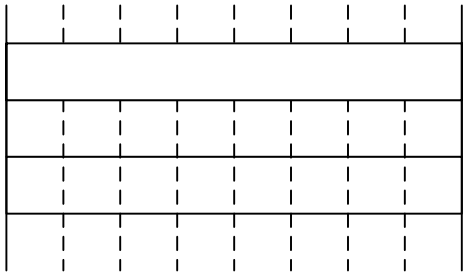


Figura 3.21: Armazenamento de dados na instrução STATE $19F.

A linha TMS é mantida no nível lógico baixo ‘0’ ou alto ‘1’ de acordo com o bit menos significativo da sequência a deslocar. A actualização da linha TMS é efectuada através do deslocamento de 1 bit da esquerda para a direita do registo **tms_reg** durante o flanco ascendente do relógio de teste (TCK).

A figura 3.22 apresenta o diagrama temporal para a execução da instrução **`STATE $19F`** (assumindo que o registo CNT1 tinha sido previamente carregado através da instrução **`MOVC CNT1,10`**) que permite atingir o estado *Shift-IR* da máquina de estados do controlador TAP, passando pelo estado *Test-Logic-Reset* provocando uma inicialização da infra-estrutura de teste.

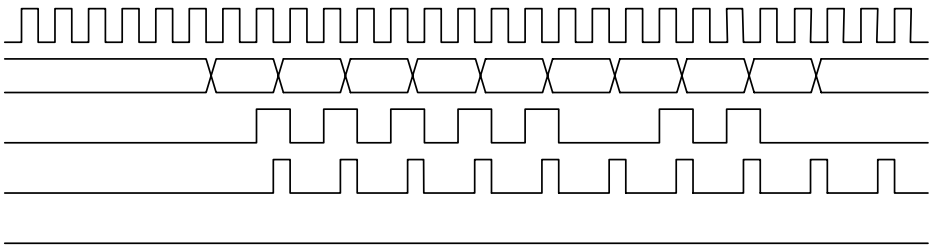


Figura 3.22: Diagrama temporal para a execução da instrução STATE $19F.

NSHF d2shf

Esta instrução desloca para o interior do TAP, através da sua saída série (TDO), uma sequência de N bits especificada pelo operando **d2shf** que se encontra armazenada nas três posições de memória seguintes à que contém o código de instrução. A primeira contém os oito bits mais significativos do valor a deslocar, a segunda posição contém os oito bits intermédios, e a terceira posição contém os oito bits menos significativos.

A actualização do valor a deslocar para a cadeia ocorre com as transições descendentes no relógio de teste, efectuando o deslocamento de 1 bit para a direita do registo **tdo_reg** que contém a sequência a deslocar. Durante esta fase, a linha TMS é mantida no nível lógico baixo ‘0’ durante os impulsos de relógio de teste, de modo a que a o controlador TAP se conserve no estado *Shift-IR* (*Shift-DR*). Esta instrução deverá ser precedida por outra que carregue o contador **CNT1** com o número de transições pretendidas (**`MOVC CNT1, TCKval`**).

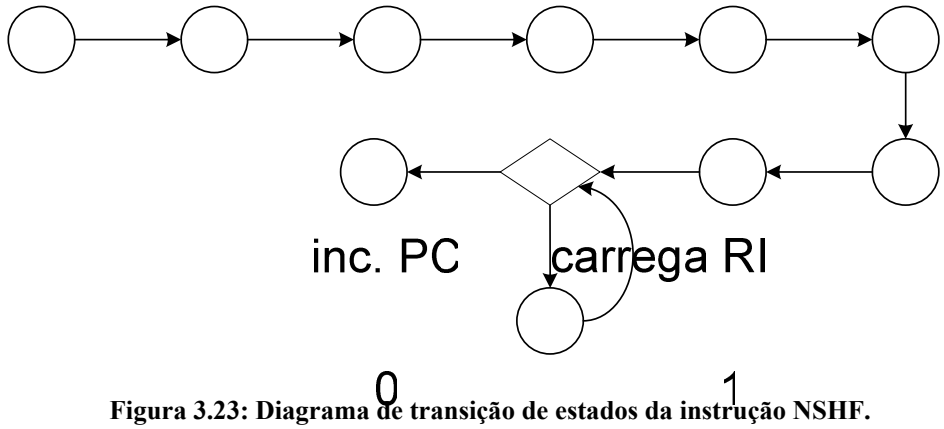


Figura 3.23: Diagrama de transição de estados da instrução NSHF.

Os dados que compõem sequência de bits a deslocar para o interior da cadeia BS deverão estar organizados como se ilustra a figura 3.24, para a execução da instrução **`NHSF $FFFFF`**. Estes dados serão deslocados para da esquerda para a direita (do bit mais significativo para o menos significativo), pelo que o primeiro bit a deslocar será o bit menos significativo (D0) da palavra LSB.

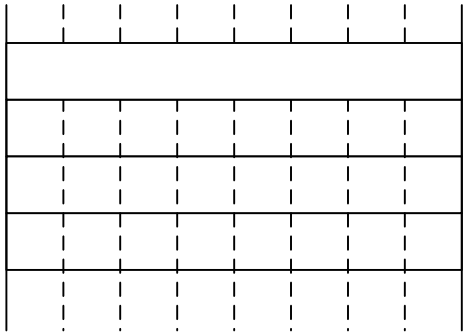


Figura 3.24: Armazenamento de dados na instrução NSHF.

Instruções para o controlo dos recursos internos, canais de sincronismo, e do fluxo do programa de teste.

Apresenta-se de seguida a caracterização individual das instruções que permitem controlar os recursos internos, os canais de sincronismo e o fluxo de execução do programa de teste.

LD Reg#, (Endereço)

Esta instrução transfere para o registo especificado pelo operando '**Reg#**' o conteúdo da posição de memória especificada pelo endereço de 8-bit. Este registo (**Reg#**) pode ser um dos registos de uso geral (R0, R1, R3, R3, R4 ou R5) ou um dos contadores (CNT1, CNT2).

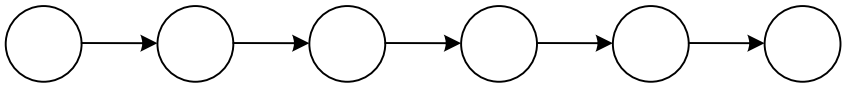


Figura 3.25: Diagrama de transição de estados da instrução LD.

ST (Endereço), Reg#

Esta instrução transfere para a posição de memória especificada pelo endereço de 8-bit o conteúdo do registo especificado pelo operando '**Reg#**'. Este registo ('**Reg#**') pode ser um dos registos de uso geral (R0, R1, R3, R3, R4 ou R5) ou um dos contadores (CNT1, CNT2).

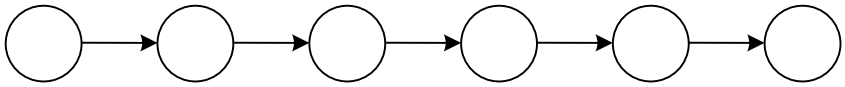


Figura 3.26: Diagrama de transição de estados da instrução ST.

MOVC Reg#, N

Esta instrução carrega no registo especificado pelo operando '**Reg#**' o conteúdo do armazenado na posição de memória seguinte à que contém o código de instrução,

especificado pelo segundo operando. Este registo (‘**Reg#**’) pode ser um dos registos de uso geral (R0, R1, R3, R3, R4 ou R5) ou um dos contadores (CNT1, CNT2).

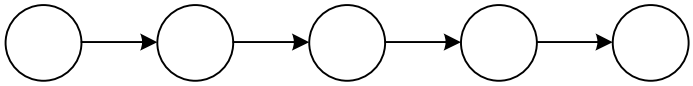


Figura 3.27: Diagrama de transição de estados da instrução MOVC.

DJNZ Reg#, Endereço

Esta instrução decrementa o registo especificado pelo operando ‘**Reg#**’ e verifica se o respectivo conteúdo chegou a 0. Caso não se verifique esta ocorrência efectua-se um salto absoluto para a posição especificada pelo operando que se segue ao código de instrução. Este registo (‘**Reg#**’) pode ser um dos registos de contagem (CNT1, CNT2).

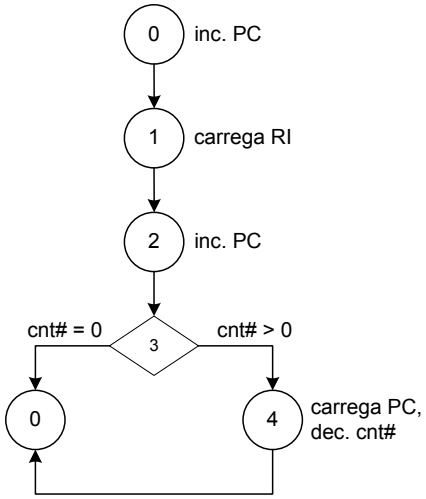


Figura 3.28: Diagrama de transição de estados da instrução DJNZ.

Esta instrução permite a implementação de ciclos de programa do tipo para i = 1 até N faça <conjunto de instruções>, em que N corresponde ao valor carregado no registo seleccionado, CNT#, através da instrução **MOVC CNT#, N**. **CNT#** representa um dos registos de contagem.

JMP Endereço

Esta instrução provoca um salto incondicional para o endereço de 8-bits que se encontra armazenado na posição a seguir ao código de instrução.

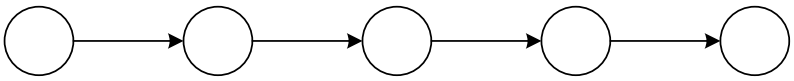


Figura 3.29: Diagrama de transição de estados da instrução JMP.

WAIT0, WAIT1

Estas instruções retêm o processador no mesmo estado por um número indefinido de ciclos de relógio, até que a entrada de sincronismo se encontre no valor lógico especificado (‘0’, ‘1’).

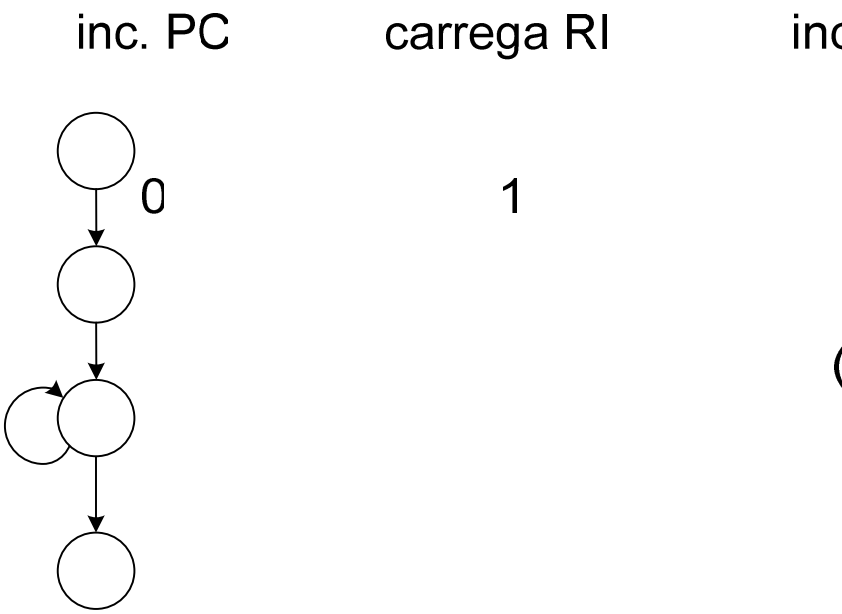


Figura 3.30: Diagrama de transição de estados da instrução WAIT.

NOP

Esta instrução não realiza qualquer tipo de operação apesar de ser descodificada. No entanto é possível recorrer a esta instrução para gerar atrasos temporais por software.

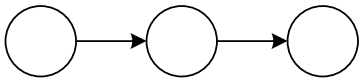


Figura 3.31: Diagrama de transição de estados da instrução NOP.

HLT

Esta instrução é usada para terminar a execução do programa, suspendendo todas as operações até que seja recebido um sinal de 'reset', e sinalizando que o processador terminou a execução o teste através da linha Busy.

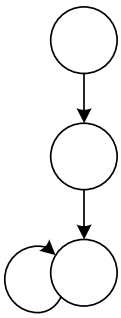


Figura 3.32: Diagrama de transição de estados da instrução HLT.

Instruções para suporte às operações de teste

Apresenta-se de seguida a caracterização individual das instruções destinadas ao suporte das operações de teste.

SRTEST

Esta instrução inicia o procedimento de teste escalonando para execução o módulo CUTctrl, responsável pelo controlo do dispositivo a testar (CUT – *Core Under Test*).

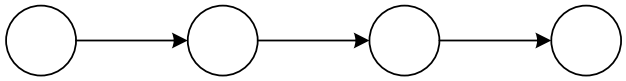


Figura 3.33: Diagrama de transição de estados da instrução SRTEST.

EOTEST

Esta instrução sinaliza o fim da operação de teste e reinicia os apontadores para o segmento de dados (DSctrl) e para o segmento de vectores de teste (SSctrl). A sinalização do fim de cada etapa de teste ocorre levando o sinal 'Eot' ao nível lógico alto '1'.

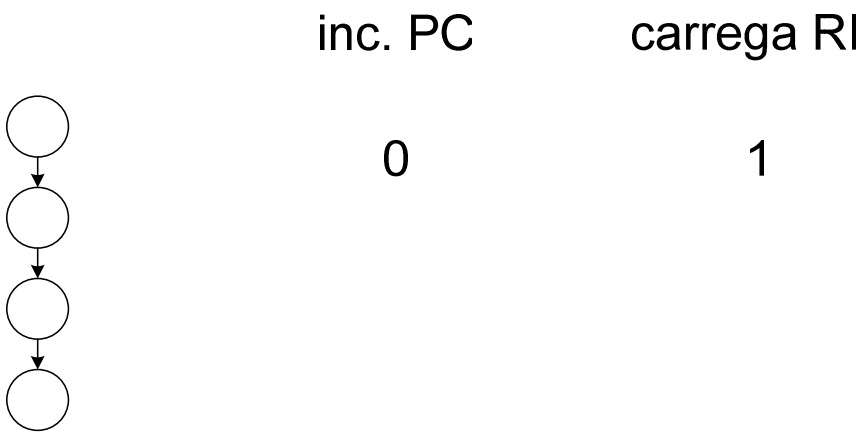


Figura 3.34: Diagrama de transição de estados da instrução EOTEST.

STIMULUS

Esta instrução aplica um vector de estímulo ao módulo STIMULUSctrl, responsável pela geração dos estímulos de teste. Sempre que esta instrução é usada, um novo vector é aplicado ao gerador e o apontador para o segmento de vectores de teste, SSctrl, é incrementado.

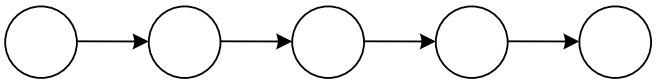


Figura 3.35: Diagrama de transição de estados da instrução STIMULUS.

inc. PC carrega RI

0 1

setsti

3.5. Conclusão

O processador genérico de teste descrito neste capítulo implementa os requisitos identificados, garantindo-se as funcionalidades básicas para as operações de teste dos blocos que compõem o microssistema integrado.

A implementação do processador num dispositivo de lógica programável possibilita, de acordo com os requisitos de teste, a adaptação da arquitectura e o conjunto de instruções descritas processador genérico, transformando-o num ASIP garantindo-se uma maior flexibilidade comparativamente às soluções baseadas no uso de BIST.

No capítulo seguinte apresenta-se a metodologia para a geração automática do processador de teste, que possibilita efectuar a adaptação do processador de teste de acordo com os requisitos de teste impostos pelo dispositivo a testar.

CAPÍTULO 4

Metodologia para a geração automática do processador de teste.

Neste capítulo descreve-se a metodologia para a geração automática do processador de teste, a partir da arquitectura apresentada no capítulo anterior, de modo a garantir a sua adaptação para o teste dos diversos blocos analógicos e mistos em microssistemas integrados.

Os objectivos estabelecidos para a metodologia são descritos em primeiro lugar, seguindo-se a apresentação da estratégia global adoptada com a descrição das etapas consideradas na geração/adaptação do processador e identificando-se o fluxo de dados utilizado pela ferramenta computacional, nomeadamente a informação de entrada e de saída. Este capítulo termina com uma apresentação mais detalhada do algoritmo adoptado para a adaptação e optimização do processador de teste.

4.1. Objectivos

Apesar da aparente simplicidade que caracteriza a aplicação de operações de teste a um determinado bloco, esta tarefa pode complicar-se substancialmente pela presença de dezenas ou mesmo centenas de núcleos em microssistemas integrados, uma vez que cada bloco a testar impõe requisitos de teste distintos, para além de, em certas circunstâncias, ser necessária a realização de testes que se baseiem em modos particulares de funcionamento.

As considerações apresentadas justificam a existência de uma solução flexível totalmente adaptável, de modo a ir ao encontro dos requisitos de teste impostos pelos diversos blocos que compõem o microssistema. O facto de nesta dissertação se propor o recurso a blocos de lógica reconfigurável para implementar o processador de teste, justifica a existência de uma metodologia que garanta a geração automática do processador de teste, de acordo com os requisitos de teste, a partir da arquitectura do processador genérico para teste, apresentado no capítulo anterior.

Na secção seguinte descreve-se a estratégia global adoptada para metodologia de geração automática do processador de teste, identificando-se as principais etapas.

4.2. Etapas e metodologia de geração e optimização

Nesta secção descreve-se o processo de adaptação e optimização da arquitectura do processador, apresentado no capítulo anterior, para garantir que a funcionalidade do processador seja a mais adequada para o teste em causa.

A metodologia adoptada para adaptação e optimização de um processador específico para o teste (ASIP) de blocos analógicos e mistos em microssistemas integrados, inicia-se com o levantamento das necessidades de teste impostas pela natureza do bloco a testar e pelo tipo de teste (teste estrutural ou funcional) que se pretender efectuar de modo a assegurar a funcionalidade necessária. É vital que esta etapa ocorra o mais cedo possível no ciclo de desenvolvimento, uma vez que a partir da informação reunida determina-se que etapas deverão ocorrer a seguir.

Do ponto de vista da metodologia assume particular importância a etapa de inserção de recursos de teste, os quais deverão ir ao encontro dos requisitos impostos pelo tipo de teste e pelo dispositivo a testar. Sendo este o veículo principal para garantir a flexibilidade dos diversos procedimentos de teste, torna-se vital que ocorra o mais cedo possível no ciclo de desenvolvimento do processador, o que depende da natureza das operações a realizar. Partindo do pressuposto de que os recursos adequados às necessidades de teste, já se encontram especificados após o levantamento dos requisitos, é possível iniciar o desenvolvimento do programa de teste, a partir do qual se procede à

adaptação e optimização do processador, de modo a incluir apenas os recursos necessários à execução do programa de teste desenvolvido.

Esta última etapa produz, ao nível RTL (RTL – *Register Transfer Level*), o modelo comportamental do processador organizado por um conjunto de módulos HDL (*HDL – Hardware Descritpion Language*) sintetizáveis, onde se incluem os blocos periféricos que garantem as funcionalidades pretendidas pelo processador de teste, assim como a informação relativa à satisfação das restrições associadas ao mapeamento dos diversos blocos na FPGA.

A partir do modelo comportamental do processador, procede-se à sua implementação, traduzindo a especificação do circuito num conjunto de dados representados como uma cadeia de bits (*bitstream*), que será usado para configurar a FPGA.

De seguida, analisa-se em maior detalhe cada uma das etapas mencionadas anteriormente.

Levantamento de requisitos

Independentemente do tipo de teste a efectuar e da natureza do componente a testar, é necessário efectuar o levantamento das necessidades de teste impostas de modo a caracterizar completamente a(s) funcionalidade(s) que o processador deve apresentar para efectuar uma determinada operação de teste. Sabe-se à partida que o processador deverá ser responsável pelo controlo da infra-estrutura de teste, aplicando o(s) estímulo(s) de teste adequado(s) e capturando as respectivas respostas, procedendo de seguida à sua avaliação. Mas as características apresentadas por estas operações dependem dos requisitos de teste impostos pela funcionalidade que o bloco a testar possui e pelo tipo de teste a efectuar. Por exemplo, o teste das ligações, é de um modo geral, realizado através da aplicação de estímulos de teste de baixa frequência, sendo possível recorrer aos mesmos módulos de controlo e observação para testar todas as ligações do circuito. O mesmo não se sucede para o teste funcional, onde as necessidades de teste impostas pelos diversos blocos obrigam à aplicação de estímulos de teste com características distintas, o que implica a existência de estruturas para o controlo e observação adequadas.

O levantamento realizado possibilita a caracterização do estímulo de teste adequado e das estruturas responsáveis pelo controlo e observação, permitindo avaliar se os recursos do processador genérico garantem ou não a funcionalidade pretendida. Estas considerações justificam a necessidade desta etapa, que deverá ocorrer o mais cedo possível, uma vez que determina se a etapa seguinte, relativa à inserção dos recursos de teste deve ou não ocorrer.

Inserção de recursos de teste

A ocorrência desta etapa é determinada a partir do levantamento da informação efectuada na etapa anterior, a qual avalia se os recursos disponíveis no processador são ou não os mais adequados aos requisitos impostos pelo teste a efectuar. Partindo do pressuposto de que os recursos presentes não são os mais adequados ao teste em causa, é necessário proceder à inserção dos recursos adicionais que garantam a funcionalidade pretendida. A utilização desta etapa implica que os recursos a inserir se encontrem construídos em modelos sintetizáveis em HDL, Verilog [39] ou VHDL [40], que devem descrever/caracterizar 'bem' o funcionamento do respectivo recurso. A inserção destes blocos na arquitectura genérica do processador deve-se encontrar num formato de entrada adequado, desenvolvido para este efeito, onde se especifica a instrução que actua sobre o módulo (recurso), definindo-se a sua interface com a unidade de controlo do processador e procedendo à respectiva instanciação no processador. Sempre que seja necessário é possível proceder à adaptação de algumas das instruções existentes, especificando os atributos pretendidos (por exemplo, número de operandos, ciclos de execução, entre outros).

A inserção manual de recursos de teste permite um maior grau de flexibilidade e liberdade, à custa de um maior dispêndio de tempo e trabalho, o que é inevitável. Repare-se que é possível a reutilização de alguns módulos, como por exemplo o controlador do TAP, a unidade de registos, ALU e outros, qualquer que seja o processador a desenvolver, procedendo-se, sempre que necessário à respectiva adaptação.

Desenvolvimento do programa de teste

Nesta etapa procede-se ao desenvolvimento do programa de teste, a executar pelo processador, que contém as instruções que reflectem a(s) funcionalidade(s) adequada(s) das operações de teste a realizar. É a partir da informação contida neste programa que se procede à optimização do processador, de modo a este conter apenas os recursos necessários à execução do programa desenvolvido.

Processo de optimização e adaptação

Esta etapa é responsável pela adaptação e optimização do processador de modo a garantir que este possua as funcionalidades adequadas à realização do teste. Este processo inicia-se com a leitura linha-a-linha do programa de teste, extraindo as instruções e os registos referenciados, efectuando-se de seguida a optimização do processador. Deste modo, garante-se que apenas os recursos necessários à execução do programa de teste se encontram incluídos na arquitectura do processador. A optimização é efectuada em duas fases, em primeiro lugar remove-se da arquitectura os recursos desnecessários ao teste em causa, inserindo-se os mais adequados. Seguidamente, a unidade de controlo do processador é simplificada para executar apenas as instruções referenciadas. Por exemplo, a unidade de registos e a ALU do processador são optimizadas para incluir apenas os registos e as instruções referenciados no programa de teste, respectivamente.

A saída desta etapa é o modelo comportamental, ao nível RTL, do processador organizado num conjunto de módulos HDL sintetizáveis, que descrevem as funcionalidades dos blocos periféricos. Nesta fase, é necessário definir as estratégias de optimização, restrições e opções de configuração impostas para guiar o processo de implementação do processador no bloco de lógica reconfigurável. Esta edição manual exige um conhecimento detalhado da organização interna do dispositivo.

A figura seguinte ilustra o processo de configuração (adaptação e optimização) do processador de teste para um determinado programa de teste. Tal como descrito previamente, é realizada uma análise ao programa de teste e a partir do seu resultado é efectuada a adaptação do processador, removendo e/ou adicionando recursos necessários (blocos funcionais), depois de optimizados, à sua arquitectura base.

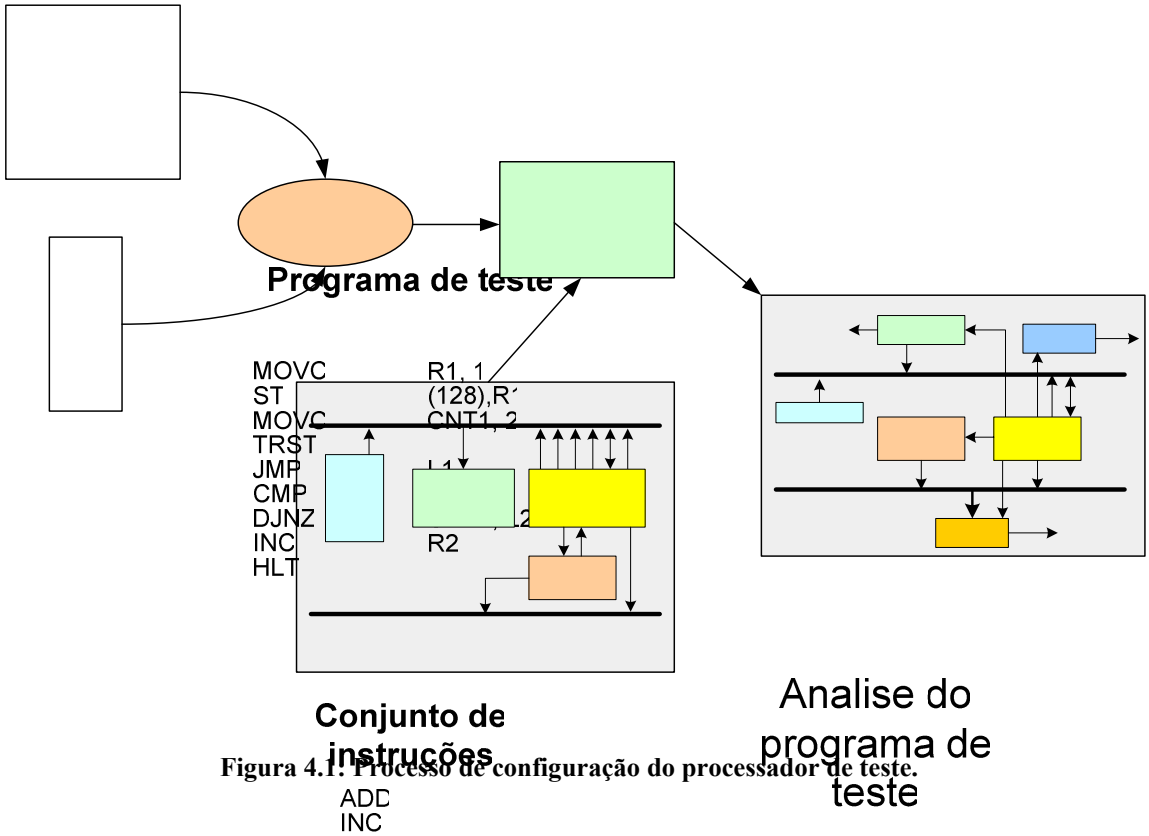


Figura 4.1: Processo de configuração do processador de teste.

Implementação do processador de teste

Esta etapa é a parte final do processo de adaptação do processador, que consiste na tradução da especificação do processador num conjunto de dados representados numa cadeia de bits (*bitstream*). Este *bitstream* é enviado para a FPGA, para a configurar com o circuito desenhado.

Este processo de implementação pode ser realizado em várias fases dependendo da ferramenta usada. Por exemplo, no sistema *Foundation*, da XILINX [41], é usado um programa com interface gráfica que invoca as ferramentas adequadas e exibe os relatórios emitidos durante a sua execução.

Na secção seguinte procede-se à identificação do fluxo de dados utilizado pela ferramenta de geração automática do processador de teste, procedendo-se igualmente à identificação da informação de entrada e saída deste processo.

4.3. Identificação do fluxo de dados

Nesta secção identifica-se o fluxo de dados existente no processo de adaptação do processador de teste, descrevendo-se a informação de entrada e saída deste processo.

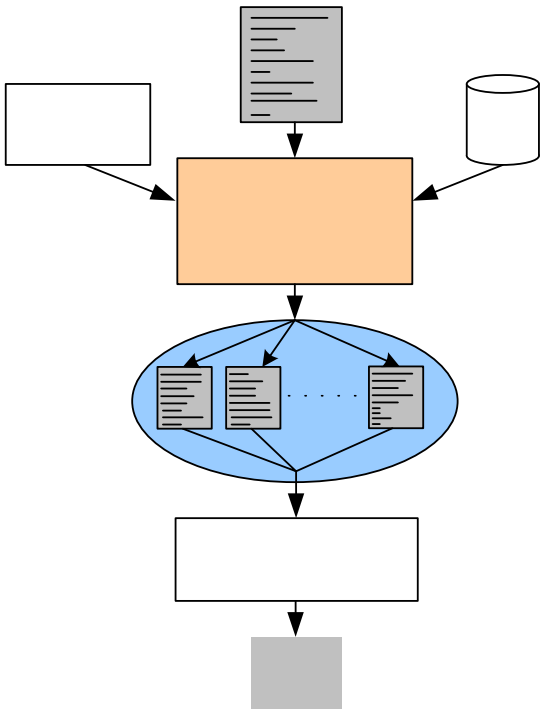


Figura 4.2: Diagrama de fluxos para a geração automática do processador de teste.

A figura 4.2 ilustra a representação simplificada do diagrama de fluxos existente no processo de adaptação de processador de teste, considerando as etapas identificadas na secção anterior. O processo de adaptação e optimização do processador é realizado através de *scripts*, desenvolvidos em linguagem PERL (PERL – *Pratical Extraction and Report Language*) [42], que invocados na sequência adequada garantem a correcta optimização do processador de teste.

A adaptação inicia-se com a execução do *script* '*compile-testprog.pl*', que actua sobre o programa de teste (*testprogram.asm*) procedendo à respectiva validação, identificando-se cada uma das instruções contidas no programa. Este *script* invoca a aplicação *shareware* denominada TASM [65], gerando os respectivos ficheiros em código-máquina (*tesprogram.obj*), com os respectivos códigos de instrução e

operandos, bem como um ficheiro-relatório (*tesprogram.lst*) que contêm os endereços, código fonte e o respectivo código-máquina, as *labels* usadas ao longo do programa e o respectivo valor. O ficheiro-relatório é, de seguida, processado pelo *script 'create-testproghex.pl'* para criar o ficheiro de inicialização da memória (*testprogram.hex*). A identificação de cada uma das instruções de um dado recurso encontra-se por sua vez armazenada num ficheiro em formato tabular (sufixo.tab), lido pelo TASM no início do processo de geração do código-máquina. Esta etapa encontra-se ilustrada na figura seguinte.

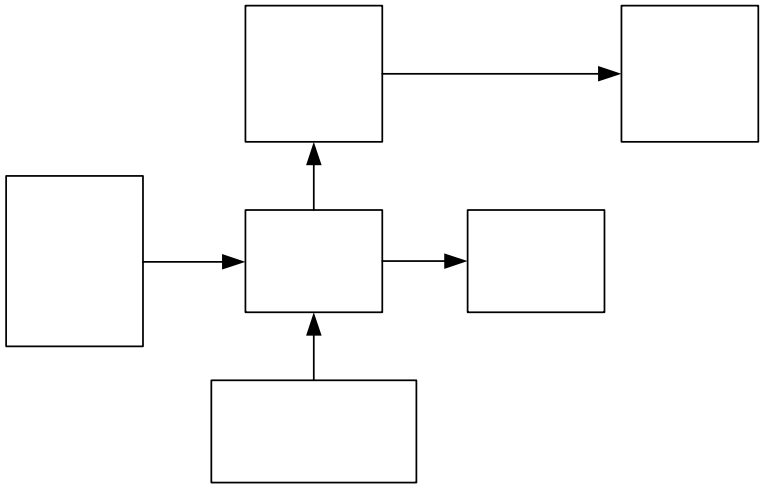


Figura 4.3: Validação do programa de teste e geração do ficheiro com o código a ser executado pelo processador.

Concluída a validação do programa de teste procede-se à extracção da informação necessária, a partir da base de dados que contêm os ficheiros HDL, para a configuração do processador de teste. Os *scripts 'extract-hdlinfo.pl'* e *'extract-regs.pl'*, quando invocados, criam ficheiros temporários que contém a informação relativa às instruções e registos referenciados, respectivamente. Estes ficheiros em conjunto com os que contêm a descrição base do processador, que inclui por exemplo o comprimento do barramento de dados e endereços, da unidade de controlo e da unidade de registos são por sua vez processados pelos *scripts 'update-ctrlunit.pl'*, *'update-regunit.pl'* e *'update-ucontroller.pl'*. Estes são responsáveis pela actualização da unidade de controlo, com a funcionalidade especificada pelas instruções contidas no programa de teste, da unidade

Ficheiro-relatór

.LST

Programa de teste

.ASM

TASM

de registos, incluindo apenas os registos referenciados, e do processador instanciando os recursos adequados para garantir a funcionalidade do programa de teste.

Após a adaptação do processador de teste procede-se à invocação do *script* *'create-ucf.pl'*, sendo gerado o ficheiro *ucontroller.ucf*, que contém a interface do processador adaptado. É neste ficheiro que se efectua a definição de restrições e opções de configuração, associando os PADs nos blocos lógicos de entrada e saída, especificando as restrições associadas ao mapeamento a lógica dentro dos componentes da FPGA (CLBs, IOBs, etc.) ou tempos de atraso mínimos ou máximos para ligações ou blocos de lógica.

A figura seguinte ilustra o processo de adaptação e optimização do processador de teste, visualizando-se a sequência pela qual os *scripts*, responsáveis por este processo, são invocados.

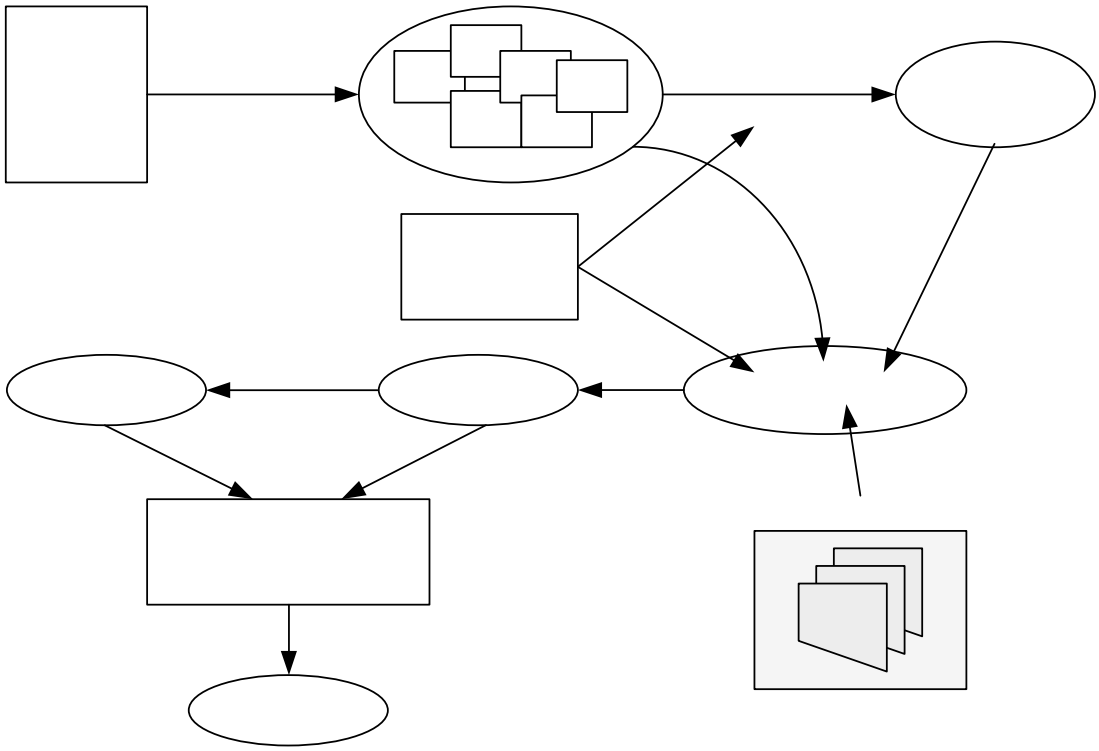


Figura 4.4: Geração automática do processador a partir do programa de teste.

A figura 4.2 identifica a informação de entrada e saída do processo de geração automática do processador de teste que será descrita com maior detalhe nas duas subsecções seguintes.

Programa de teste Ficheiros ter

.tmp

.ASM extract-hdlinfo.pl .tmp

extract-regs.pl .tmp

4.3.1. Informação de entrada

Com base na figura 4.2, é possível listar toda a informação de entrada utilizada pela ferramenta, que se encontra repartida por quatro componentes de acordo com as etapas identificadas na secção 4.2. A figura seguinte ilustra uma visão geral da informação de entrada necessária para a geração automática do processador de teste.

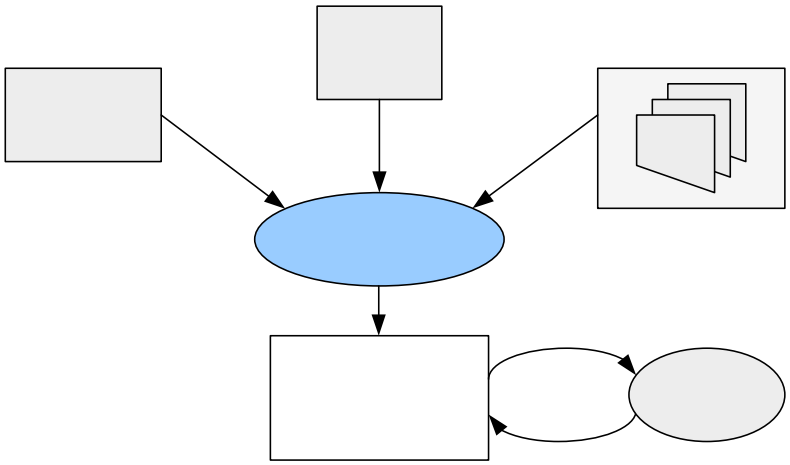


Figura 4.5: Informação de entrada para a geração automática do processador de teste.

Em seguida descreve-se, com maior detalhe, cada um destes componentes, referindo-se qual a sua contribuição para o processo de geração automática.

Programa de teste (*testprogram.asm*)

O programa de teste, *testprogram.asm*, escrito em linguagem *assembly* contém as instruções que garantem a funcionalidade pretendida para o processador de teste. É a partir do programa de teste que se extrai a informação adequada para a adaptação e optimização do processador de teste.

Arquitectura base do processador

Este ficheiro contém a descrição estrutural da arquitectura base do processador de teste apresentada no capítulo anterior. A dimensão dos barramentos de dados e de endereços, os módulos que garantem a funcionalidade básica do processador, entre outros, constitui a informação presente neste ficheiro.

Ficheiro(s) HDL

Sempre que o levantamento das necessidades de teste exigir a inserção de recursos adicionais, de modo a garantir a funcionalidade adequada do processador, é necessário efectuar a descrição do recurso de teste num formato adequado. Este ficheiro contém caracterização individual de cada recurso que compõe o processador. A correspondência entre o número / nome de pinos funcionais associados ao módulo encontra-se descrita neste ficheiro, que contém igualmente a indicação dos recursos comuns usados (barramento de dados/memória, etc.) e a instanciação do recurso no processador de teste. A especificação da instrução, definida pelo utilizador, que actua sobre o recurso faz também parte da informação contida neste ficheiro. Considera-se a existência de uma base de dados com este tipo de ficheiros. A figura seguinte ilustra a estrutura de dados de um ficheiro deste tipo.

```
# * * * *    Core Interfacing Signals * * * *  #
//ctrlunit_interface
                    opralu, Eflag, Zflag, Gtflag, Ltflag,  //ALU interface
//ctrlunit_interface_end
#    Core Output Signals    #
//ctrlunit_output_port
output [2:0] opralu;             //ALU outport
//ctrlunit_output_port_end

#    Core Input Signals    #
//ctrlunit_input_port
input Eflag, Zflag, Gtflag, Ltflag;    //ALU inport
//ctrlunit_input_port_end
#     New Core Required Logic    #
//ctrlunit_wire_decl
assign opralu   =   IR[27: 25];  //ALU outport
//ctrlunit_wire_decl_end

//ctrlunit_input_event
             Zflag or Eflag or Gtflag or Ltflag or  //ALU input event
//ctrlunit_input_event_end

#    Inter-operability with the ALU instructions    #
//ctrlunit_op2dec
                 4'b0000 : begin                     // ALU operation
                                en_wreg = 1;         // ALU operation
                                en_alu = 1;          // ALU operation
                                nextstate = INITIAL;      // ALU operation
                           end                       // ALU operation
//ctrlunit_op2dec_end
# * * * * Instantiation of the ALU module within the processor * * * * #
//mc_surround_blocks
//Aritmetic Logic Unit module
wire Eflag, Zflag, Gtflag, Ltflag;            //Core - ALU interface
wire [2:0] opralu;                            //Core - ALU interface
wire [7:0] aluout;
alu    alu_module    (    .aluout(aluout),    .Zflag(Zflag),    .Eflag(Eflag),
.Ltflag(Ltflag), .Gtflag(Gtflag), .a(abus), .b(bbus), .opralu(opralu) );
//mc_surround_blocks_end
# Interfacing with the core processor #
//mc_core_interface
.opralu(opralu),.Zflag(Zflag),.Eflag(Eflag), .Ltflag(Ltflag), .Gtflag(Gtflag),
//mc_core_interface_end
```

Figura 4.6: Extracto da estrutura de dados interna referente às instruções que actuam sobre a unidade aritmética e lógica (ALU).

Opções de implementação definidas pelo utilizador (*ucontroller.ucf*)

Este ficheiro, *ucontroller.ucf*, contém as estratégias de optimização do processador a implementar, através da definição de restrições e opções de configuração. É através deste ficheiro que se efectua a associação dos PADs, isto é, pinos da FPGA nos blocos lógicos de entrada e saída, especificando as restrições associadas ao mapeamento da lógica dentro dos componentes da FPGA (CLBs, IOBs, etc.) ou tempos

de atraso mínimos ou máximos para ligações ou blocos de lógica. A figura seguinte ilustra a estrutura de dados de um ficheiro deste tipo.

```
#************************************************************#
#                  ucontroller.ucf module                    #
#************************************************************#

#--------------------------------------
# T I M I N G   C O N S T R A I N T S
#--------------------------------------
# N/A

#--------------------------------------
# I P  C O R E  C O N S T R A I N T S
#--------------------------------------
# N/A

#-------------------------------------------------
# P L A C E  &  R O U T E  C O N S T R A I N T S
#-------------------------------------------------
# N/A

#-------------------------------------------------
# T I M I N G   I G N O R E  C O N S T R A I N T S
#-------------------------------------------------
# N/A

#-----------------------------------------------
#     P I N   A S S I G N M E N T S
#-----------------------------------------------
#---------------------------------------
#    INPUT PIN   A S S I G N M E N T S
#---------------------------------------

#---------------------------------------
#    OUTPUT PIN   A S S I G N M E N T S
#---------------------------------------
```

Figura 4.7: Extracto da estrutura de dados interna referente às opções de configuração do processador na FPGA.

4.3.2. Informação de saída

Esta subsecção descreve, em detalhe, a informação de saída produzida pela ferramenta responsável pela geração automática do processador de teste. Esta informação consiste em dois ficheiros: o ficheiro *bitstream*, que contém a informação necessária para programar a FPGA, e o ficheiro de inicialização da memória, que contém o código-máquina do programa de teste a ser executado pelo processador. A figura seguinte

ilustra uma visão geral da informação de saída gerada ao longo do processo de geração automática do processador de teste.

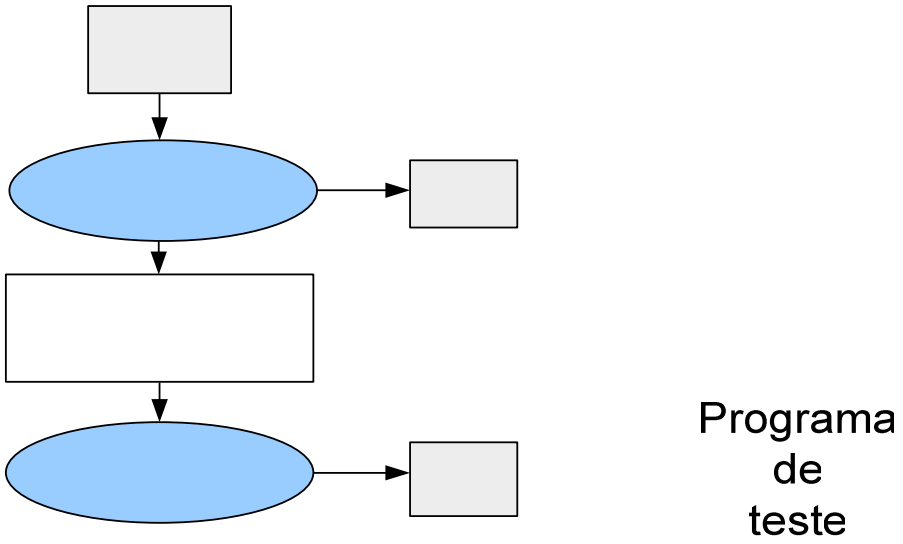


Figura 4.8: Informação de saída da geração automática do processador de teste.

Ficheiro de inicialização da memória (*testprogram.hex*)

Este ficheiro contém a informação de inicialização da memória de programa e de dados. É gerado a partir do programa de teste *'testprogram.asm'* através da aplicação TASM, que gera os respectivos ficheiros em código-máquina (*tesprogram.obj*), com os respectivos códigos de instrução e operandos, bem como um ficheiro-relatório (*tesprogram.lst*) que contêm os endereços, código fonte e o respectivo código-máquina, as *labels* usadas ao longo do programa e o respectivo valor. O ficheiro-relatório é posteriormente processado para criar o ficheiro de inicialização da memória.

Ficheiro de configuração (ucontroller.bit)

Este ficheiro consiste numa sequência de bits que deve ser enviada para a FPGA para implementar o processador de teste. Contém informação necessária à configuração da FPGA, definindo toda a lógica interna e interligações, a localização dos CLBs, IOBs e *buffers* especiais que a FPGA dispõe para distribuir sinais com fan-out elevado (tipicamente sinais de relógio) e implementar barramentos de 3 estados. A sua geração é efectuada pela tecnologia associada à FPGA a configurar.

4.4. Geração automática do processador de teste

Esta secção descreve o processo para a geração automática do processador de teste, nomeadamente a formação da estrutura de dados interna a partir da qual se procede à implementação do processador. De forma a ilustrar mais detalhadamente este processo, recorre-se a um exemplo de um programa de teste genérico, apresentado na figura seguinte.

```
NLEVELS             .EQU   5      ;no. of harmonics to determine
       SETDDS
        MOVC  R1, 1
       ST     (128),R1
       MOVC   R1, 2
       ST     (129),R1
       MOVC   R1, 3
       ST     (130),R1
       MOVC   R1, 4
       ST     (131),R1
       MOVC   R1, 5
       ST     (132),R1
       ; initialise BS infrastructure
       MOVC   CNT1, 11
       STATE  447          ;go to Shift-IR state
       MOVC   CNT1, 20
       NSHF   0            ;load EXTEST instruction
       MOVC   CNT1, 5
       STATE  7            ;go to Shift-DR state
       MOVC   CNT1, 47
       NSHF   8388609      ;load BS register
       MOVC   CNT1, 2
       STATE  3            ;update BS register
       ;start test execution
       MOVC   CNT1,NLEVELS
L1:    STIMULUS
       WAIT0
       DJNZ   CNT1,L1             ;get next sample
       EOTEST
       MOVC   CNT1, 6
       STATE  3F           ;go to Test-Logic-Reset state
       HLT
.END
```

Figura 4.9: Exemplo de um programa de teste para o qual o processador deve ser adaptado.

O programa, ilustrado na figura anterior, é responsável pelo controlo de uma infra-estrutura de teste, compatível com a norma IEEE 1149.4 [6], através da qual é encaminhado o estímulo de teste para o bloco a testar. O estímulo consiste num sinal sinusoidal de diferentes frequências de modo a caracterizar a resposta em frequência do

bloco a testar. A captura das respectivas respostas de teste é efectuada recorrendo a um ATE, sincronizado com o processador de teste através da instrução `WAIT0`.

O processo de adaptação inicia-se com a leitura linha-a-linha do programa de teste, extraindo a informação necessária que é armazenada em estruturas de informação temporária relativa aos registos e instruções referenciados. As figuras seguintes ilustram estas estruturas.

```
#instructions referenced
MOVC
ST
STATE
NSHF
SETDDS
STIMULUS
WAIT0
DJNZ
EOTEST
HLT
```

(a) Ficheiro com instruções referenciadas

```
#registers referenced
R1
CNT1
```

(b) Ficheiro com registos referenciadas

Figura 4.10: Conteúdo dos ficheiros temporários de registos e instruções.

A partir das instruções presentes no ficheiro ilustrado na figura 4.10 (a) procede-se à leitura dos respectivos ficheiros HDL associados às instruções referenciadas. Na figura 4.6, da secção 4.3.1, encontra-se ilustrada a estrutura de um ficheiro HDL para as instruções aritméticas e lógicas. Um directório específico contém todos os ficheiros deste tipo.

A informação contida nos ficheiros HDL é extraída e compilada em estruturas adequadas, algumas das quais se encontram ilustradas nas figuras seguintes. Na figura 4.11, ilustra-se o extracto da estrutura de dados temporária que contém a informação relativa à descodificação e execução das instruções referenciadas.

```
//ctrlunit_op2dec
    // ********* Move Instruction  *********
       4'b0111 : begin                          // MOVC instruction
                    en_mem = 1;                 // opcode : from  -> to
                   en_wreg = 1;                 //  71(h) -> 77(h)
                   nextstate = INITIAL;         //
                 end
    // ********* LOAD/STORE Operation  *********
       4'b0010 : begin                          //
                    enaddr = 1;                 //
                    if ( IR[27] == 1'b0 )       // LOAD Instruction
                         nextstate = LOAD2;     // opcode : from  -> to
                     else                       //        21(h) -> 27(h)
                     begin                      // STORE Instruction
                          selabus = 1;          // opcode : from  -> to
                          sel_wreg = 1;         //        29(h) -> 2D(h)
                          enfcore = 1;          //
                          nextstate = STORE2;   // continue STORE execution
                    end                         //
                  end                           //
       // ********* Jump operation ********  //
        4'b0001 : begin                         //
                    if (IR[27]==1'b1)           // JMP instruction
                    begin                       // opcode: 18(h)
                         enpc = 1;              //
                         incsel = 1;            //
                         nextstate = INITIAL;   //
                    end                         //
                    else                        //
                    begin                       //
                    if( IR[27]==1'b0 )          // DJNZ instruction
                     begin                      // opcode: 16(h), 17(h)
                         deccnt = 1;            //
                         nextstate = JUMP2;     // continue DJNZ inst
                     end                        //
                      end                       //
                 end                            //
       // ********* Boundary Scan operation *********  //
        4'b1010 : begin                         //
                    if( IR[27] )                // NSHF instruction
                          ldtdi = 1;            // opcode : AE(h)
                    else                        // STATE instruction
                          ldtms = 1;            // opcode : A6(h)
                    nextstate = BSCANop2;       //
                   end                          //
//ctrlunit_op2dec_end
```

Figura 4.11: Extracto da estrutura de dados interna referente à descodificação e execução das instruções referenciadas.

Na figura 4.12 ilustra-se o extracto da estrutura de dados temporária que contém os recursos a incluir na arquitectura do processador de modo a garantir funcionalidade descrita no programa de teste. É possível verificar a inclusão dos recursos adequados para a geração dos estímulos de teste (recursos dds_enable, dds12bit, sdm12bit) e para o controlo de uma infra-estrutura de teste, compatível com a norma IEEE 1149.4 [6] (recurso BSctrl).

```
//mc_surround_blocks
//************************************************************
//This section implements the test signal generator composed by one 12-bit DDS
//connected to an 12-bit SIgma-Delta Modulator 12-bit DDS
wire rstdds, endds_core, enddsoffset;
wire [11:0] ddsout;

dds_enable enabledds( .endds(endds_core), .clk(clock), .reset(reset));

dds12bit  DDS(  .ddsout(ddsout),  .clk(clock),  .reset(reset),  .rstdds(rstdds),
.endds(endds), . enoffset(enddsoffset), .ddsin({4'b0,stimuli}));

//12-bit Sigma-Delta Modulator module
sdm12bit SigmaDelta( .clock(clock), .areset(reset), .din(ddsout[11:0]),
.dout(Yn));
//*********************************************************
wire ldtms, ldtdi, shfBS;
wire [23:0] BSdata;
BScrtl1    BScrtl_module(    .tmsdata(tms),    .tdidata(tdi),    .clock(clock),
.reset(reset),  .BSdata(BSdata),  .shfBS(shfBS),  .ldtms(ldtms),  .ldtdi(ldtdi),
.tdo(tdo), .tck(tck) );

//mc_surround_blocks_end
```

Figura 4.12: Extracto da estrutura de dados interna referente aos recursos de teste a inserir no processador.

Para além da informação ilustrada nas figuras anteriores, é necessário compilar, em estruturas semelhantes, informação adicional relativa, por exemplo, à interface da unidade de controlo com os recursos associados às instruções, à arquitectura base do processador de teste, interface do processador de teste, entre outras.

Seguidamente, e a partir do ficheiro que contém os registos referenciados ilustrados na figura 4.10 (b), procede-se à optimização da unidade de registos do processador de teste, de modo a incluir na sua estrutura apenas os registos que foram referenciados. O mesmo ocorre para a unidade de controlo.

A partir da informação compilada nas estruturas adequadas, procede-se à adaptação do processador de teste, sendo gerado o modelo comportamental, ao nível RTL, do processador organizado num conjunto de módulos HDL sintetizáveis, onde cada um destes módulos descreve as funcionalidades dos diversos blocos que constituem o processador. A informação presente no modelo comportamental constitui uma base de informação adicional sobre a acessibilidade (em termos de controlabilidade / observabilidade) do bloco a testar. A informação relativa à interface do processador de teste é usada para gerar o ficheiro que contém as restrições associadas à colocação dos CLBs ou IOBs na FPGA ou os tempos de atraso mínimos ou máximos para ligações ou blocos de lógica para guiar o processo de implementação do processador. A figura 4.13

ilustra a estrutura de dados que poderá conter as restrições associadas à implementação do processador na FPGA.

```
#************************************************************#
#                  ucontroller.ucf module                    #
#************************************************************#
#--------------------------------------
# T I M I N G   C O N S T R A I N T S
#--------------------------------------
# N/A
#--------------------------------------
# I P  C O R E  C O N S T R A I N T S
#--------------------------------------
# N/A
#-------------------------------------------------
# P L A C E  &  R O U T E  C O N S T R A I N T S
#-------------------------------------------------
# N/A
#-----------------------------------------------------
# T I M I N G   I G N O R E  C O N S T R A I N T S
#-----------------------------------------------------
# N/A
#-----------------------------------------------
#     P I N   A S S I G N M E N T S
#-----------------------------------------------
#---------------------------------------
#    INPUT PIN   A S S I G N M E N T S
#---------------------------------------
NET "dbusout[0]" LOC = "";
          ..........
NET "dbusout[6]" LOC = "";
NET "dbusout[7]" LOC = "";
NET "reset" LOC = "";
NET "tdo" LOC = "";
NET "clock" LOC = "";
#---------------------------------------
#    OUTPUT PIN   A S S I G N M E N T S
#---------------------------------------
NET "eotest" LOC = "";
NET "addrbus[0]" LOC = "";
          ..........
NET "addrbus[15]" LOC = "";
NET "tms" LOC = "";
NET "mem_we" LOC = "";
NET "dbusin[0]" LOC = "";
          ..........
NET "dbusin[6]" LOC = "";
NET "dbusin[7]" LOC = "";
NET "tdi" LOC = "";
NET "Yn" LOC = "";
NET "tck" LOC = "";
NET "busy" LOC = "";
```

Figura 4.13: Extracto da estrutura de dados referente às opções de implementação do processador.

Estes últimos ficheiros são usados como informação de entrada pelo processo de implementação do processador, que poderá ser executado de forma completamente automática dependendo da tecnologia usada, gerando o ficheiro, *ucontroller.bit*, que

contém o resultado final da implementação representado como uma cadeia de bits (*bitstream*), usado para configurar a FPGA.

4.5. Conclusão

Neste capítulo descreveu-se a metodologia para a geração automática do processador de teste a partir do respectivo programa de teste. Este processador é optimizado de modo a garantir que a funcionalidade especificada pelas instruções inseridas no programa de teste é satisfeita, incluindo apenas os recursos necessários à sua execução.

A optimização efectuada ao processador, através da eliminação dos recursos desnecessários, poderá ser usada para melhorar a capacidade de processamento do processador, diminuindo deste modo a duração associada ao tempo de teste. De um modo geral, quanto maior for o processamento efectuado pelo processador sobre as respostas obtidas do teste, menor será a quantidade de tráfego de informação de teste entre o SoC e o ATE, reduzindo o tempo de teste.

Apesar desta estratégia conduzir à geração de processadores programáveis que não exibem um elevado grau de flexibilidade, é uma maneira eficiente de criar controladores dedicados optimizados para cada tarefa particular do teste, uma vez que somente os recursos necessários à execução de um programa são incluídos no processador. Adicionalmente, todos os programas de teste que recorrerem ao mesmo conjunto de instruções podem ser executados sem requerer a síntese de um diferente processador.

CAPÍTULO 5

Teste de núcleos analógicos e analógico-digitais

Este capítulo é dedicado ao teste de núcleos analógicos e mistos (analógico-digitais), recorrendo ao processador apresentado no capítulo anterior. De acordo com os requisitos de teste impostos pelo(s) núcleo(s) a testar, procede-se à adaptação do processador de teste que garanta a funcionalidade adequada para as metodologias de teste adoptadas. Para cada metodologia serão apresentados os detalhes de implementação e os resultados obtidos.

5.1. Teste de circuitos por correlação

Esta secção é dedicada à descrição do processo de concepção e implementação do processador de teste, baseado na aplicação da correlação, para determinar os diversos parâmetros que possibilitam a caracterização de um circuito misto que contém um conversor A/D, usando como veículo de demonstração a interface analógico de aquisição de sinais do contador de energia desenvolvido pela TECMIC [43].

5.1.1. Aplicação da correlação ao teste de circuitos mistos

A correlação é uma das funções comummente utilizadas na análise de dispositivos lineares e não-lineares. Recorrendo à operação de correlação é possível determinar as componentes de ganho e desvio de fase de qualquer sistema estimulado por um sinal

sinusoidal [44]. As mesmas operações permitem também, usando estímulos de frequências distintas, determinar a resposta em frequência do dispositivo em teste e diversos parâmetros de caracterização do respectivo desempenho.

Os valores associados ao ganho e ao desvio de fase, para cada um dos estímulos (senos) de diferentes frequências, são encontrados recorrendo ao cálculo das correlações da resposta do sistema com o correspondente estímulo de entrada e com um sinal em quadratura (co-seno), como se encontra ilustrado na figura seguinte.

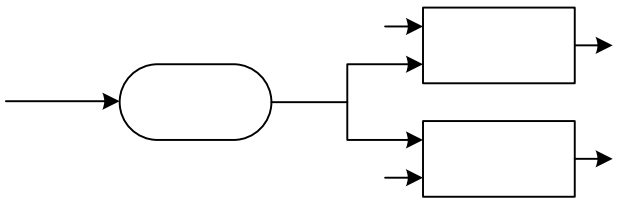


Figura 5.1: Cálculo da função de transferência por correlação.

Na figura 5.1, $Ryx_s(n)$ e $Ryx_c(n)$ representam respectivamente o resultado da operação de correlação entre a resposta do sistema, y, e o sinal de teste $X_s(n)$ e $X_c(n)$, às frequências de interesse, $n = 1, 2, 3,\ldots, m$. Estes resultados são usados para a obtenção do ganho e do desvio de fase introduzido pelo dispositivo às diversas frequências, de acordo com as seguintes equações:

$$|H_n| = \frac{2}{A^2}\sqrt{R_{yx_s}(n)^2 + R_{yxc}(n)^2} \qquad (5.1)$$

e

$$\angle H_n = arctg\left(R_{yx_s}(n) \Big/ R_{yxc}(n)\right) \qquad (5.2)$$

onde H_n representa o ganho da função de frequência à frequência $n\omega t$.

Esta operação pode também ser usada para a determinação de parâmetros adicionais como por exemplo THD, SINAD, SNR e IMD [45]. A distorção harmónica total, THD, é obtida efectuando a operação de correlação da resposta do sistema com o seno e o co-seno a essa frequência mas também com os senos e co-senos correspondentes aos harmónicos relevantes para o cálculo da THD, de acordo com a equação apresentada de seguida.

$$THD = 20\log\sqrt{\frac{\sum_{n=2}^{N}\left|H_n\right|^2}{\left|H_1\right|^2}} \qquad (5.3)$$

5.1.2. Sistema de implementação

A implementação desta solução foi efectuada recorrendo à interface analógica de aquisição de sinais do contador de energia desenvolvido pela TECMIC. A figura 5.2 ilustra o diagrama de blocos do protótipo de teste realizado numa placa de circuito impresso (PCB – *Printed Circuit Board*) que inclui uma FPGA, um conversor A/D e uma memória RAM. O sistema é dotado de uma infra-estrutura de teste IEEE 1149.4 [6] controlada pelo processador.

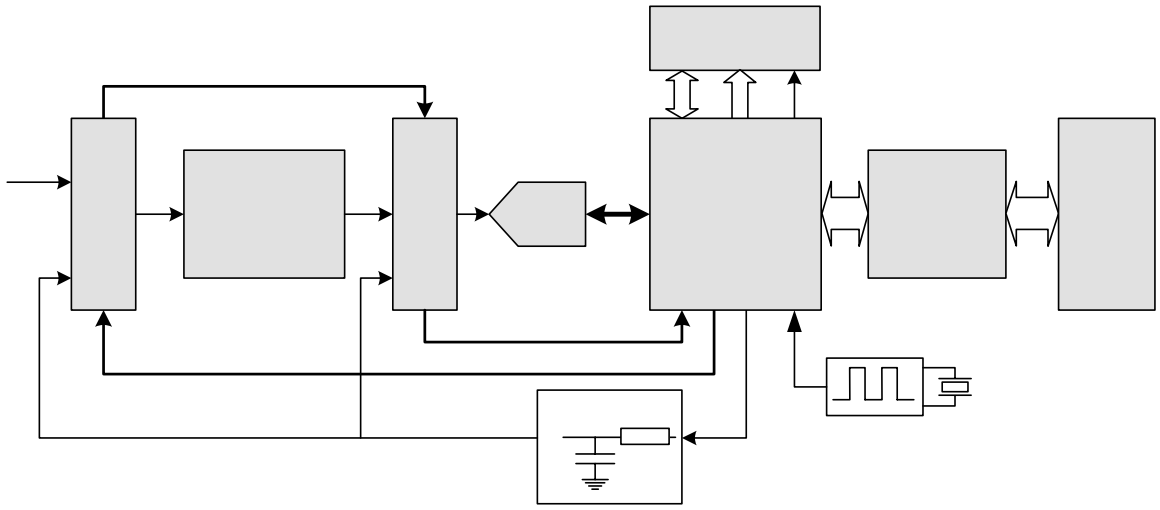


Figura 5.2: Sistema para implementação do teste por correlação de um ADC.

Esta infra-estrutura de teste, IEEE 1149.4 [6], é usada para injectar o estímulo de teste ao sistema a testar (lógica de condicionamento de sinal ou entradas do conversor A/D), seleccionando esse estímulo nas entradas primárias do circuito, garantindo-se a acessibilidade e a controlabilidade dos nós analógicos que se pretender testar. Esta infra-estrutura é implementada recorrendo a dois circuitos integrados SCANSTA400 [46], da National Semiconductor, controlados pelo processador de teste implementado

na FPGA, que configura o modo mais adequado para o funcionamento do STA. Uma solução alternativa, baseada em multiplexadores analógicos (por exemplo 4052 [47], 4066 [48] ou 4016 [49]), poderia ser equacionada para implementar o barramento de teste, dispensando o controlador e reduzindo o tempo necessário para programar e comutar entre as diferentes entradas (funcionamento normal e estímulo de teste).

O facto de o STA400 permitir trocar as entradas V e I ou partilhar no tempo um dos canais analógicos (formado pelo adaptador de impedâncias e A/D) pelos sinais, implementando deste modo um mecanismo de redundância no caso de ser detectada uma falha no outro canal, são argumentos a favor da sua utilização em detrimento dos multiplexadores analógicos.

A FPGA usada é o no dispositivo XC4013EPG223-4, da Xilinx [50], sendo a sua programação levada a cabo com recurso ao programa *lpx* (desenvolvido por José Carlos Alves) utilizando um cabo apropriado ligado à porta paralela do PC anfitrião para o envio de uma sequência de bits. Para enviar o *bitstream* para a FPGA, o programa *lpx* deve ser executado numa janela DOS, usando como argumento o nome do ficheiro a enviar. A edição, simulação e compilação dos esquemáticos das diferentes configurações da FPGA são levadas a cabo usando a ferramenta de CAD *Foundation*, da Xilinx [50], sendo a geração dos blocos efectuada usando a linguagem de descrição de hardware Verilog [39].

A interface de configuração e comunicação recorre a portas lógicas genéricas para assegurar o correcto condicionamento dos sinais de entrada e de saída. A inclusão de uma memória RAM [51] no sistema tem por objectivo permitir o armazenamento dos dados relativos ao programa de teste, operações de geração de estímulos e captura de resultados de teste.

5.1.3. Implementação do processador de teste

Todo o projecto deverá ser desenvolvido como um circuito síncrono com um único domínio de relógio (SYSCLK = 8MHz), derivado de um sinal de relógio externo de 40MHz. Os blocos que operam a cadências inferiores devem ser igualmente sincronizados com o SYSCLK e habilitados por sinais de *clock enable* produzidos por um módulo de gerador de relógio.

O dispositivo de FPGA utilizado no protótipo é ocupado pela aplicação (contador de energia) em cerca de 50%. Uma vez que durante o teste (ou pelo menos durante a aplicação do estímulo de teste) o contador de energia não pode funcionar, a FPGA é reconfigurada temporariamente com o processador de teste, o qual deve ser adaptado com o objectivo de caber na FPGA para acomodar a aplicação.

O processador a desenvolver deverá realizar, parcialmente ou na sua totalidade, as operações elementares descritas na secção 3.2.2 do capítulo 3. A figura 5.3 ilustra, em pseudo código, a funcionalidade que o processador de teste deve garantir para o teste de cada componente.

```
para ( cada componente ){
        configurar os respectivos modos de teste;
        gerar estímulo de teste;
        configurar infra-estrutura de teste (cadeia BST);
        aplicar estímulo de teste;
        capturar as respectivas respostas de teste ;
        efectuar operações de processamento;
}
```

Figura 5.3: Funcionalidade apresentada pelo processador de teste.

O processador é responsável pelo controlo adequado de cada componente a testar, gerando o respectivo estímulo de teste e procedendo à configuração da infra-estrutura de teste, através da qual é encaminhado o estímulo de teste para o componente a testar. Com o estímulo a ser aplicado ao componente procede-se à captura das respectivas respostas de teste, procedendo de seguida ao seu processamento. A figura 5.4 ilustra a arquitectura global do processador de teste, onde se visualizam as unidades responsáveis pelas operações identificadas na figura 5.3.

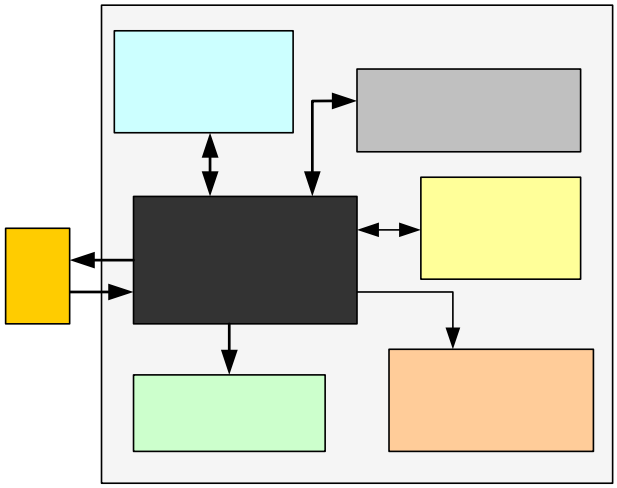


Figura 5.4: Arquitectura global do processador para o teste de circuitos por correlação.

Tal como foi descrito no capítulo 3, o processador é constituído por um núcleo rodeado pelas unidades adequadas que garantam as funcionalidades pretendidas pelo processador de teste. O núcleo, que corresponde à unidade central do processador, é responsável pelo escalonamento das diversas unidades que compõem o processador para a execução das operações pretendidas.

Nas secções seguintes procede-se à descrição detalhada dos blocos funcionais que implementam as operações identificadas como requeridas para este caso.

5.1.3.1. Controlador do dispositivo a testar

O dispositivo a testar consiste numa cadeia de condicionamento de sinal constituída por um conversor A/D de dois canais de 18 bits cada, com interface série (CS5330A da Crystal [52]), filtro/amplificador de entrada, um conversor A/D sigma-delta (ΣΔ) de 1 bit e um filtro passa-alto (para remover a componente DC), para cada entrada do A/D, AINL (*Left Analog Input*) e AINR (*Right Analog Input*). A operação de conversão gera amostras de ambos os canais (AINL e AINR), produzindo resultados em 18 bits, complemento para 2.

A figura seguinte ilustra a interface do conversor A/D com a FPGA e a infra-estrutura de teste.

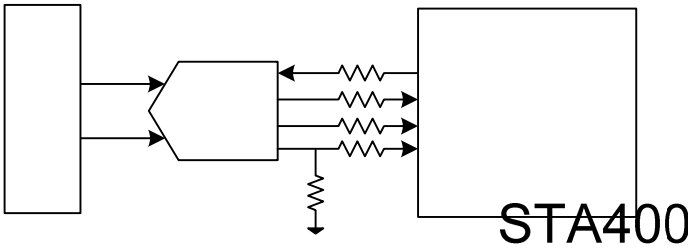


Figura 5.5: Interface do conversor A/D CS5330A.

A resistência de *pull-down* de 47kΩ colocada no sinal SDATA garantir o funcionamento no modo *Master*. Neste modo, a entrada MCLK (*Master Clock*) do conversor recebe, do controlador, um sinal de relógio usado para a inicialização do conversor e para a geração da frequência de amostragem do modelador sigma-delta (ΣΔ) presente no sistema. As interfaces LRCK (*Left/Right Clock*) e SCLK (*Serial Data Clock*) fornecem sinais de saída, derivados internamente a partir do sinal MCLK e definidos respectivamente pelas relações MCLK/4 e MCLK/256, que permitem sincronizar as duas palavras de 18 bits (canal AINL e AINR), resultantes do processo de conversão, enviadas para a FPGA pelo sinal SDATA (*Serial Data Output*). Na figura seguinte encontra-se ilustrado o diagrama temporal do conversor para o modo de operação seleccionado.

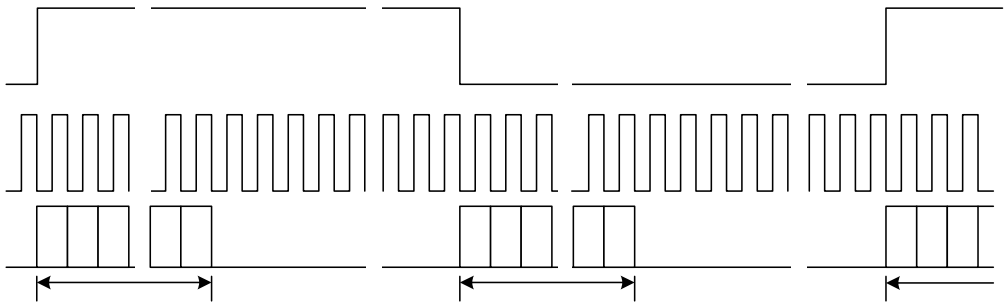


Figura 5.6: Diagrama temporal dos sinais produzidos pelo A/D CS5330A.

O sistema a implementar inclui um controlador responsável pelo correcto sequenciamento do processo de controlo, iniciando-o após a recepção de um sinal enviado com esse fim pelo processador. O sistema fornece um sinal MCLK responsável pela inicialização do conversor A/D e estabilização da sua resposta ao estímulo aplicado, procedendo-se de seguida à captura das respostas de teste pretendidas. Simultaneamente efectua-se a validação das amostras de teste capturadas, que consiste

numa comparação com as amostras anteriores verificando se a amostra 'cai' no intervalo esperado (ver bloco ***TRESPANAL – Test Response Analyser***). O diagrama de estados apresentado na figura seguinte ilustra a descrição comportamental do sistema a desenvolver (controlador do dispositivo a testar), e na figura 5.8 mostra-se o respectivo diagrama de blocos.

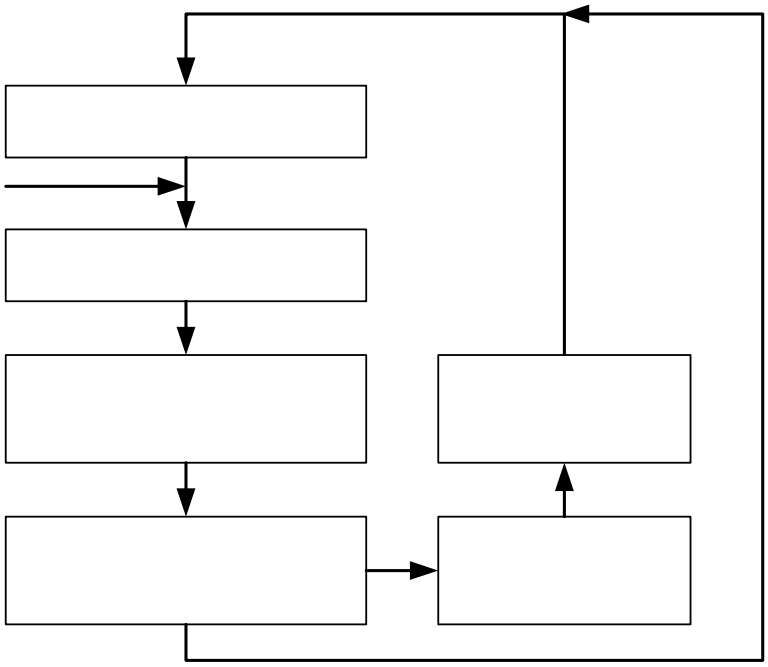


Figura 5.7: Diagrama de estados do controlador do dispositivo a testar.

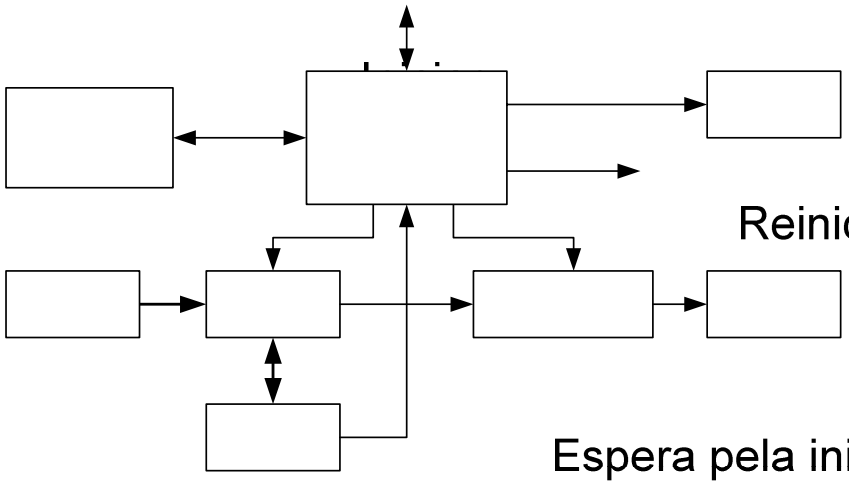


Figura 5.8: Diagrama de blocos do controlador do dispositivo a testar.

Procede-se de seguida à descrição detalhada dos blocos funcionais que compõem o sistema.

CS5330A_ctrl

A principal funcionalidade deste bloco consiste em controlar o sequenciamento das operações relacionadas com o controlo do conversor A/D e na captura das respostas de teste pretendidas.

CS5330A_init

Este bloco efectua a inicialização do conversor A/D, escalonando-o para entrar em funcionamento de acordo com o modo pretendido e fornecendo um sinal MCLK adequado. Após a ocorrência de 127 ciclos do MCLK surge o sinal LRCK e após a ocorrência de 11265 ciclos os dados série resultantes do processo de conservação surgem na interface SDATA. Simultaneamente espera pela estabilização da resposta do conversor A/D ao estímulo aplicado.

SPI – Serial-parallel interface

Este bloco realiza a conversão dos dados série enviados pelo conversor A/D para duas palavras de 18 bits em complemento para dois (canal direito, AINR, e canal esquerdo, AINL), juntamente com um sinal de sincronismo à frequência de amostragem (*NewSample*). O digrama temporal dos sinais produzidos por este módulo é ilustrado na figura 5.9.

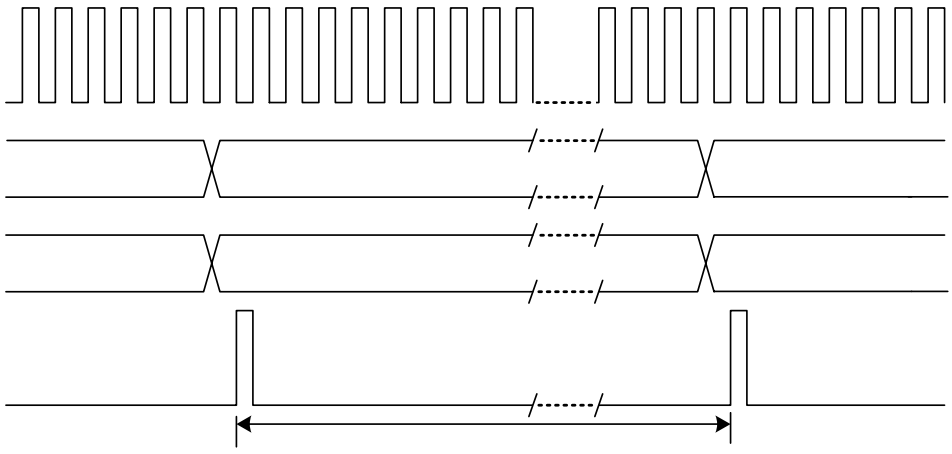


Figura 5.9: Saída da interface SPI.

TRESPANAL – Test Response Analyser

Este módulo avalia a validade das respostas capturadas do conversor A/D, permitindo detectar antecipadamente a ocorrência de falhas graves. Isto é, ao aplicar-se um sinal 'bem caracterizado' ao conversor A/D, é possível, estimar a variação esperada entre amostras consecutivas (o máximo ocorre na passagem pelo valor médio onde a derivada é máxima no caso de um sinal sinusoidal). Se para cada amostra capturada se efectuar uma comparação com as amostras anteriores pode-se ir avaliando se justifica continuar com a progressão do teste. Em anexo B encontra-se a descrição desta solução.

CS5330A_ss

Este bloco salvaguarda no segmento de dados da memória as amostras capturadas da resposta de teste do conversor A/D. Estas amostras correspondem a palavras de 16 bits, uma vez que pelo menos 2 LSBs (ou mesmo 4 ou 6 LSBs) podem ser omitidos pois correspondem a ruído, permitindo deste modo reduzir o tamanho do correlador e o volume de dados a armazenar e a processar.

5.1.3.2. Gerador do estímulo de teste

De modo a realizar a determinação das componentes de ganho e de desvio de fase, através da operação de correlação, de um qualquer sistema é necessário que este seja estimulado por um sinal sinusoidal adequado (ver secção 5.1.1.).

No presente caso o estímulo de teste adoptado consiste num sinal sinusoidal com amplitude de 4Vpp centrado em 2.4V e com uma frequência de 50 Hz. O valor de amplitude foi escolhido de modo a aproximar-se do valor de fim-de-escala do conversor, isto é, próxima do valor máximo admissível à sua entrada, enquanto o valor da frequência foi determinado pela própria funcionalidade pretendida para o circuito, isto é a aquisição de sinais a 50Hz.

A implementação para a geração de um sinal sinusoidal foi baseada numa arquitectura DDS (DDS – *Direct Digital Synthesizer*) [53], [54]. Um DDS permite gerar digitalmente um sinal analógico, com um controlo preciso de frequência, fase e amplitude.

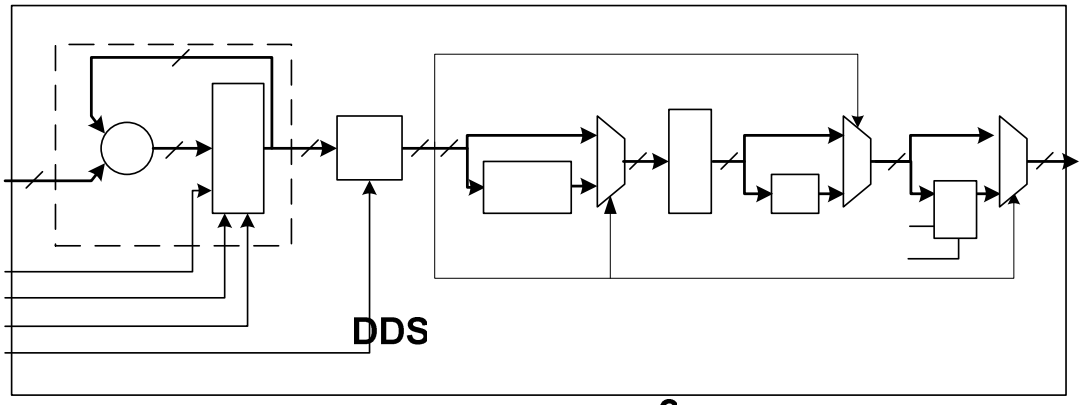


Figura 5.10: Arquitectura do DDS.

A figura 5.10 ilustra a arquitectura do DDS que se descreve a seguir, abordando a sua interface e os blocos que a constituem. Os sinais envolvidos na funcionalidade do DDS são:

CLK_{DDS} : sinal de relógio com frequência CLK_{DDS} = 8 MHz;

Reset : sinal de *reset* síncrono activo alto (nível lógico '1');

enDDS : sinal de *enable* activo alto (nível lógico '1');

enCOS : sinal de *enable* para gerar o sinal em quadratura, alto (nível lógico '1');

SineCos : saída digital do sinal gerado com 12 bits, ocupando a gama permitida por 12 bits [0, 4095];

Fseno : entrada de 9 bits que especifica a frequência a gerar, que corresponde ao incremento de fase.

Os blocos que a constituem são os seguintes:

Acumulador de fase

O acumulador consiste num somador e registo que produz o endereço de uma tabela que contém as amostras da função a sintetizar. O valor colocado na entrada é acumulado ao valor do registo e determina a rapidez de variação de fase (varrimento dessa tabela), o que se traduz na frequência do sinal periódico gerado.

Tabela de consulta (LUT – *Look-up table*)

É uma tabela que contém as amostras do sinal a gerar. A funcionalidade é semelhante a de uma memória ROM, cujas linhas de endereços são alimentadas pelas saídas do acumulador de fase e as saídas de dados apresentarão as amostras do sinal gerado. Estas amostras foram obtidas da função *sin(x)*, para um sinal com 512 amostras de 12 bits. A representação gráfica destes valores tabelados encontra-se ilustrada na figura seguinte.

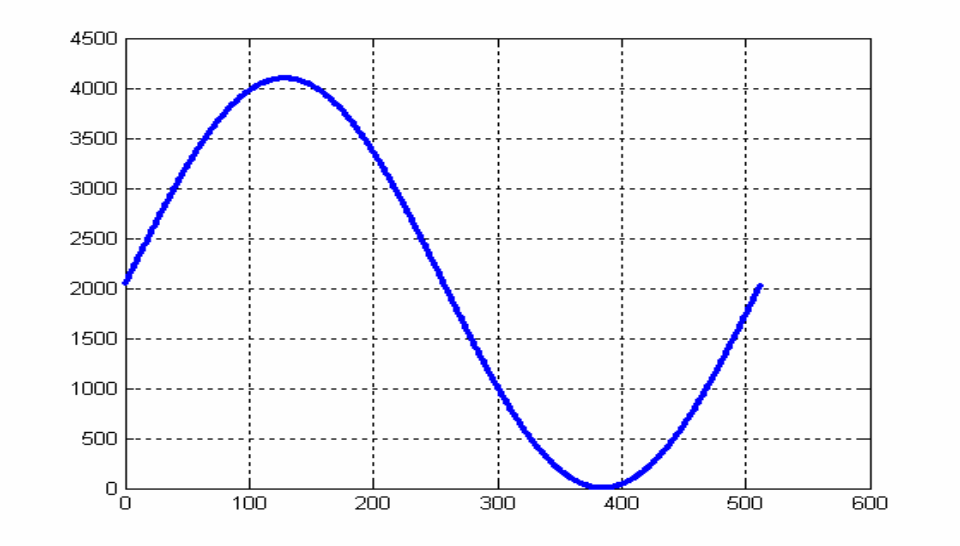


Figura 5.11: Representação gráfica dos valores tabelados no DDS.

A tabela de consulta foi reduzida para ¼ do período (128 amostras), uma vez que é suficiente armazenar as amostras da função seno no primeiro quadrante (intervalo [0, π/2[), obtendo-se o resto da função à custa de transformações aritméticas simples. A inclusão do bloco '+Pi/2' na arquitectura possibilita a geração do respectivo sinal em quadratura, enquanto que a lógica que envolve a tabela de consulta (blocos '1sCMPL' e '-1', multiplexers e o registo de saída), implementa as transformações aritméticas necessárias para obter as restantes amostras da função seno.

Para a geração de uma sinusóide com uma frequência, f_0, de 50 Hz, e com um período completo da função seno com 512 amostras, M, o acumulador necessita de percorrer os 512 endereços da LUT num tempo igual a 1/50. Deste modo, o acumulador

deverá trabalhar a uma frequência, f_{DDS}, e que determina a frequência de amostragem do sinal digital produzido pelo DDS, de acordo com a seguinte equação:

$$f_0 = \frac{Fseno \times f_{DDS}}{M} \rightarrow f_{DDS} = \frac{f_0 \times M}{Fseno} \qquad (5.4)$$

Substituindo na equação anterior os valores pretendidos para a frequência do sinal, f_0 = 50Hz, e incremento de fase, *Fseno* = 1, obtém-se o valor da frequência a que o acumulador deve trabalhar.

$$f_{DDS} = \frac{50 \times 512}{1} = 25600\,Hz \qquad (5.5)$$

Uma vez que o projecto é desenvolvido com um único domínio de relógio (SYSCLK), o DDS que opera a uma cadência inferior é habilitado pelo sinal de *clock enable*, *enDDS*, com período especificado pela equação anterior e que tem a duração exacta de um período de SYSCLK. A figura seguinte ilustra o diagrama temporal dos sinais envolvidos no DDS.

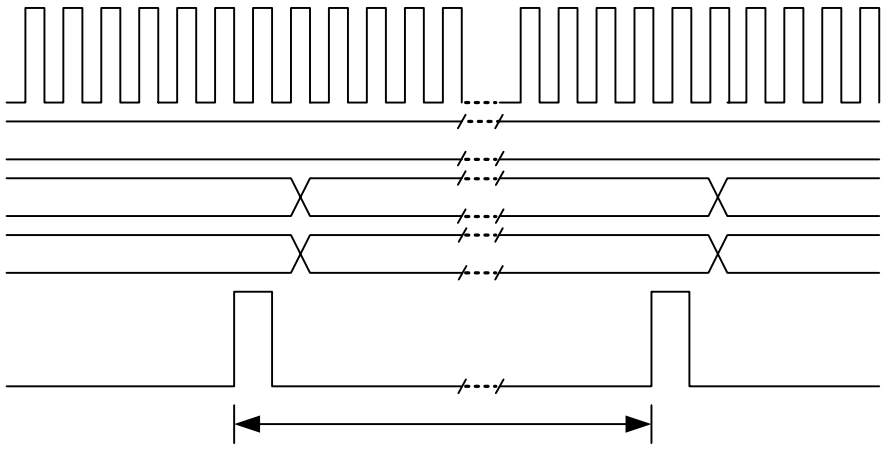


Figura 5.12: Diagrama temporal do DDS.

A conversão do sinal digital, gerado pelo DDS, para o domínio analógico pode ser efectuada recorrendo a um modulador sigma-delta (ΣΔ) [55] ou a um PWM (*Pulse Width Modulation*) [56]. A escolha recaiu sobre um modulador sigma-delta uma vez que apresenta uma complexidade reduzida, insensibilidade do desempenho a não-

linearidades de implementação, estabilidade inerente para todos os sinais admissíveis na sua entrada na ausência de atrasos na malha de realimentação e/ou ganho ou erro. A elevada frequência de amostragem do modulador, bastante superior à frequência de Nyquist do sinal a converter (igual ao dobro da largura de banda do sinal), permite a distribuição do ruído de quantificação por uma banda bastante superior à do sinal a converter. Colocando à saída do modulador um filtro passa-baixo com uma frequência de corte idêntica à largura de banda do sinal, a maior parte da potência do ruído é eliminado.

O modulador sigma-delta (ΣΔ) de primeira ordem é composto por um somador, um integrador e um quantificador colocados num anel de realimentação, o qual contém um conversor digital-digital (DD), de acordo com o esquema apresentado na figura 5.13.

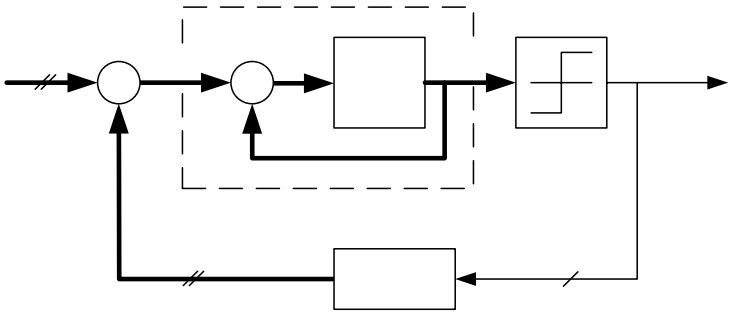


Figura 5.13: Modulador ΣΔ de primeira ordem.

A quantificação é realizada de forma completamente digital e consiste numa redução das amostras digitais do sinal obtidas a partir do acumulador. O conversor DD realiza a conversão da saída do quantificador na representação do sistema de numeração usado para os valores de entrada, sistema de numeração esse que é completamente arbitrário (por exemplo, complemento para dois).

O seguinte par de equações descreve o funcionamento do sistema ilustrado na figura 5.10.

$$e(nT) = x(nT) - y(n) \qquad (5.6)$$

$$f(nT) = \int_0^t e(nT)dt \qquad (5.7)$$

Integrador

x(nT) + e(nT) +

12-bit - Z

O funcionamento deste sistema é baseado na acumulação do erro verificado entre a entrada e a saída e na tentativa de anular esse valor. Este processo é descrito de seguida assumindo a presença de um valor constante à sua entrada e que a gama de valores admissíveis é simétrica em torno do valor 0. Sempre que o integral do erro, *f(nT)*, for igual ou superior a 0, o quantificador produz à sua saída o valor lógico 1, o que faz com que o conversor DD gere o valor +E (degrau de quantificação), levando o erro para um valor necessariamente negativo (porque +E é igual ou superior ao valor máximo do sinal de entrada) que por sua vez é aplicado à entrada do integrador. Este valor negativo, fará com que, em cada ciclo, o valor contido no integrador diminua progressivamente até descer abaixo de 0.

O inverso deste processo ao nível da polaridade dos sinais envolvidos tem lugar a partir do momento em que o valor contido no acumulador se torna negativo. Neste caso, o quantificador e o conversor DD passam a produzir nas suas saídas o valor lógico 0 e –E, respectivamente. O sinal de erro à entrada do integrador passa a ter um valor positivo, aumentando progressivamente até que o valor contido neste se tornar positivo.

Em ambas as situações anteriores, o número de ciclos necessários para que a saída do modulador, y(n), acompanhe o sinal de presente na sua entrada, x(n), e que o valor acumulado do erro se anule, depende do valor do sinal de entrada: quanto mais reduzido/elevado for este valor, maior será o número de ciclos durante os quais o valor contido no acumulador é negativo/positivo e a saída do integrador, f(nT), igual a 0/1.

Através deste processo, a diferença do sinal é mantida em oscilação em torno da referência de comparação e a saída modulada tenta acompanhar a entrada de modo a minimizar o erro. Deste modo, a densidade dos pulsos na saída irá variar de acordo com o sinal de entrada, de modo que a média dos pulsos de saída seja igual à média de entrada. O respectivo sinal analógico, cujo valor médio corresponde ao código digital aplicado à entrada do conversor, é obtido colocando à saída do modulador, y(n) um filtro passa-baixo analógico com uma frequência de corte igual à largura de banda do sinal.

A descrição comportamental do gerador de estímulo de teste, descrito anteriormente, foi efectuada em linguagem Verilog HDL, sendo a sua simulação efectuada através da ferramenta ModelSim [57]. Na figura 5.14 apresenta os resultados obtidos desta simulação, ilustrando o espectro para o sinal sinusoidal de 50 Hz.

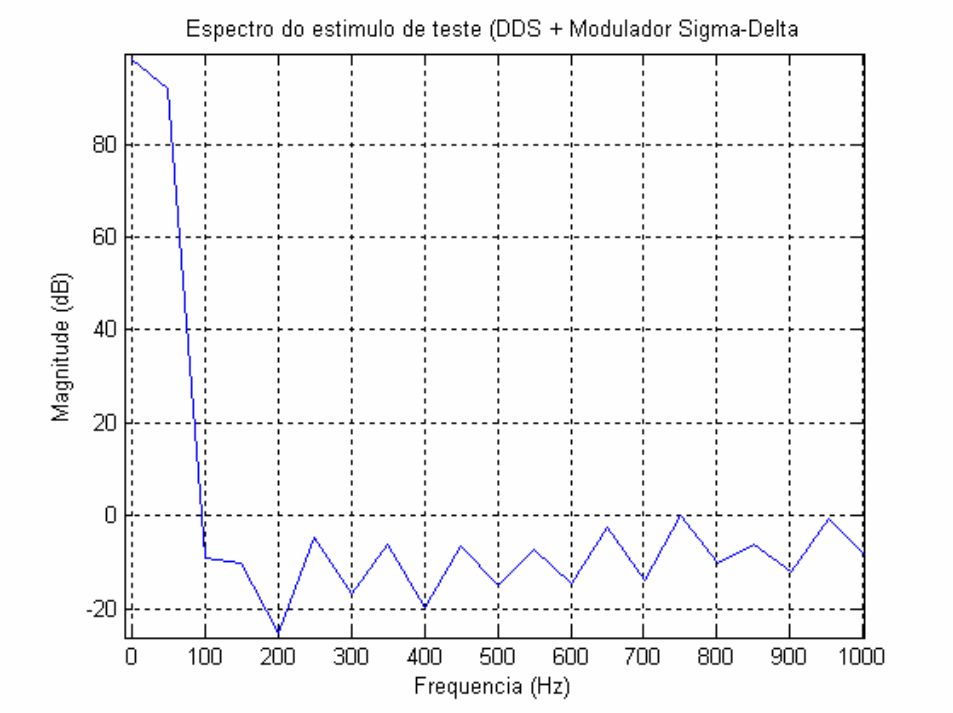


Figura 5.14: Espectro do sinal sinusoidal.

A unidade funcional descrita, constituída pelo DDS e modulador sigma-delta, é responsável pela geração de um sinal sinusoidal como estímulo de teste. As características (número de amostras por período, frequência e fase) exibidas por este sinal podem ser ajustadas caso exista a necessidade de proceder à sua adaptação para efectuar a testabilidade de um circuito que necessite de um estímulo de teste semelhante mas com características distintas. É então possível adaptar a unidade funcional descrita para produzir sinais com as características mais adequadas.

Esta adaptação consiste em ajustar o DDS para produzir sinais com as características (frequência, fase e número de amostras pretendidas por período) pretendidas, substituindo nas seguintes expressões os valores pretendidos.

$$f_{DDS} = \frac{f_0 \times M}{Fseno} \rightarrow Fseno = \frac{f_0 \times M}{f_{DDS}} \qquad (5.8)$$

$$n_{amostras} = \frac{M}{Fseno} = \frac{f_{DDS}}{f_0} \qquad (5.9)$$

$$\Delta\Phi = 2\pi \times \frac{Fseno}{M} = 2\pi \times \frac{f_0}{f_{DDS}} = \frac{2\pi}{n_{amostras}} \qquad (5.10)$$

A expressão 5.8 permite determinar a frequência de amostragem do sinal digital produzido pelo DDS, especificando a frequência pretendida pelo sinal a gerar, *f0*, o número de amostras da tabela, *M*, e a informação de entrada que especifica a frequência a gerar, *Fseno*. A expressão 5.9 determina o número de amostras que o sinal a gerar possui, enquanto que a expressão 5.10 indica o incremento de fase efectuado entre amostras consecutivas.

Uma vez que as expressões anteriores se encontram relacionadas entre si, a escolha do sinal a ser produzido pelo DDS deve efectuada assumindo uma solução de compromisso entre as características adequadas para o sinal, assumindo que a LUT contém um número de amostras fixo.

As tabelas seguintes ilustram as características de sinais sinusoidais que podem ser gerados por esta unidade, modificando apenas a frequência de referência, f_{DDS}, que determina a amostragem do sinal digital produzido pelo DDS, ou então o valor digital que especifica a frequência do sinal a gerar, *Fseno*.

Tabela 5.1: Características dos diversos sinais a gerar, mantendo fixo o valor da frequência de referência, f_{DDS}.

f_0 (Hz)	*Fseno*	f_{DDS} (Hz)	$n_{amostras}$	Δθ (graus)
50	1	25600	512	0,703
100	2		256	1,406
150	3		170	2,105
200	4		128	2,813
250	5		102	3,516

Tabela 5.2: Características dos diversos sinais a gerar, mantendo fixo o número de amostras por período completo do sinal e a informação que especifica a frequência.

f_0 (Hz)	*Fseno*	f_{DDS} (Hz)	$n_{amostras}$	Δθ (graus)
50	1	25600	512	0,703
100		51200		
150		76800		
200		102400		
250		128000		

Tabela 5.3: Características dos diversos sinais a gerar, mantendo fixo o número de amostras por período completo do sinal e a informação que especifica a frequência.

f_0 (Hz)	*Fseno*	f_{DDS} (Hz)	$n_{amostras}$	Δθ (graus)
30	1	15360	512	0,703
70		35840		
120		61440		
220		112640		
440		225280		

A tabela 5.1 apresenta os valores para adequados para gerar sinais sinusoidais com frequências e número de amostras por período completo do sinal variável, sendo apenas necessário modificar o valor digital que especifica a frequência do sinal a gerar, mantendo constante *o* valor da frequência de referência. Os valores presentes nas tabelas 5.2 e 5.3 permitem gerar sinais sinusoidais com frequências variáveis, mantendo constante o número de amostras por período completo do sinal e variando a frequência de referência, que determina a frequência de amostragem do sinal digital produzido pelo DDS.

É necessário ter algum cuidado na geração de sinais sinusoidais, de frequências distintas e número de amostras por período completo do sinal fixo, através da modificação da frequência de referência do DDS, uma vez que durante a conversão do sinal para o domínio analógico, efectuado através do modulador sigma-delta (ΣΔ), pode resultar numa má distribuição do ruído de quantificação introduzido pelo modulador. Deste modo, é necessário assegurar que o coeficiente de sobre-amostragem (CSA), isto é, razão entre a frequência de amostragem do modulador sigma-delta (ΣΔ) e a frequência do sinal de entrada, frequência de amostragem do sinal digital produzido pelo DDS (f_{DDS}), seja elevada de modo a resultar numa distribuição do ruído de quantificação por uma banda bastante superior à do sinal a converter. Esta relação é demonstrada pela expressão seguinte.

$$CSA = \frac{f_{\Sigma\Delta}}{f_{DDS}} \qquad (5.11)$$

5.1.3.3. Correlador

Este bloco é responsável por efectuar a operação de correlação de acordo com a descrição efectuada na secção 5.1.1. Compete ao bloco correlador o endereçamento e o controlo de acesso à RAM, onde se encontram as amostras capturadas da resposta de teste, e o controlo do DDS para se obter a sequência de amostras correspondentes ao seno ou ao co-seno. A figura seguinte ilustra o diagrama de blocos do circuito de correlação.

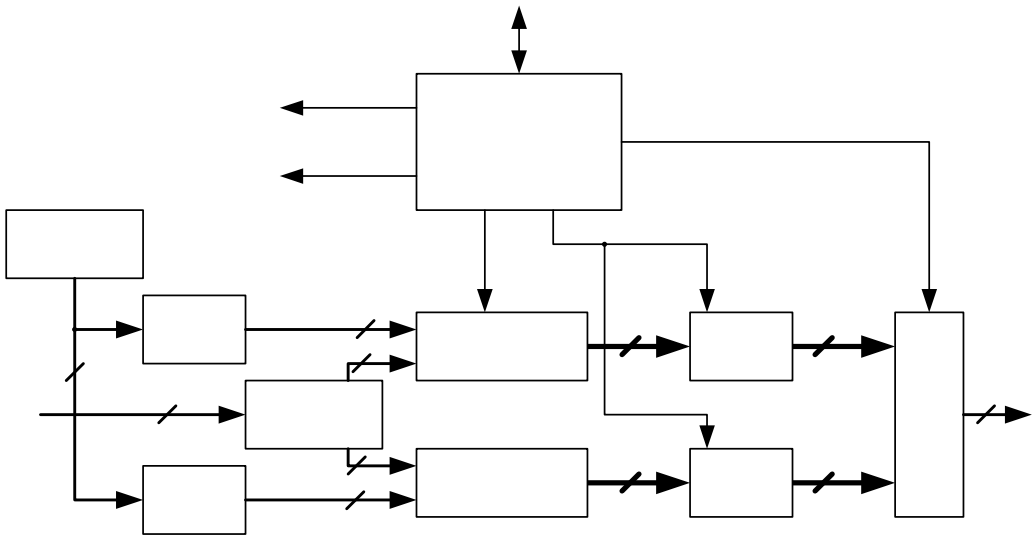


Figura 5.15: Diagrama de blocos do circuito responsável pela correlação.

A correlação propriamente dita consiste em sucessivas operações de multiplicação e acumulação (MAC – *Multiply and Accumulate operations*) pelo que é efectuada recorrendo a um multiplicador e a um acumulador, sendo todas as operações aritméticas efectuadas em formato de complemento para dois. A leitura dos valores da RAM é efectuada alternadamente, sendo em primeiro lugar lidos o par de amostras correspondente ao canal AINL do conversor A/D e em seguida o par correspondente ao canal AINR. Enquanto é lido o valor correspondentes às amostras capturadas ocorre o incremento de fase do DDS para se obter a amostra correspondente ao seno ou ao co-seno.

Controlo DDS

Controlo barramento de endereços

Interface RAM

O multiplicador, implementado através de um multiplicador série RRS (*Reduced-Register Sequential Multiplier*) [63], calcula o produto dos valores da sequência de amostras correspondente ao seno (16 bits) pelos valores das amostras (16 bits) capturadas do conversor A/D relativas ao canal AINL e AINR. Os resultados das sucessivas multiplicações são acumulados e posteriormente divididos por 2^{13} através da deslocação de 13 bits para a direita, desprezando-se os 13 LSBs. A sequência de saída passa assim a ser de um número de 32 bits, armazenado nas 4 posições consecutivas do segmento de memória que contém o resultado das operações de processamento efectuadas. As mesmas operações são efectuadas mas para o correspondente sinal em quadratura, isto é, para o co-seno. O diagrama de estados associado ao processo de correlação é apresentado na figura seguinte.

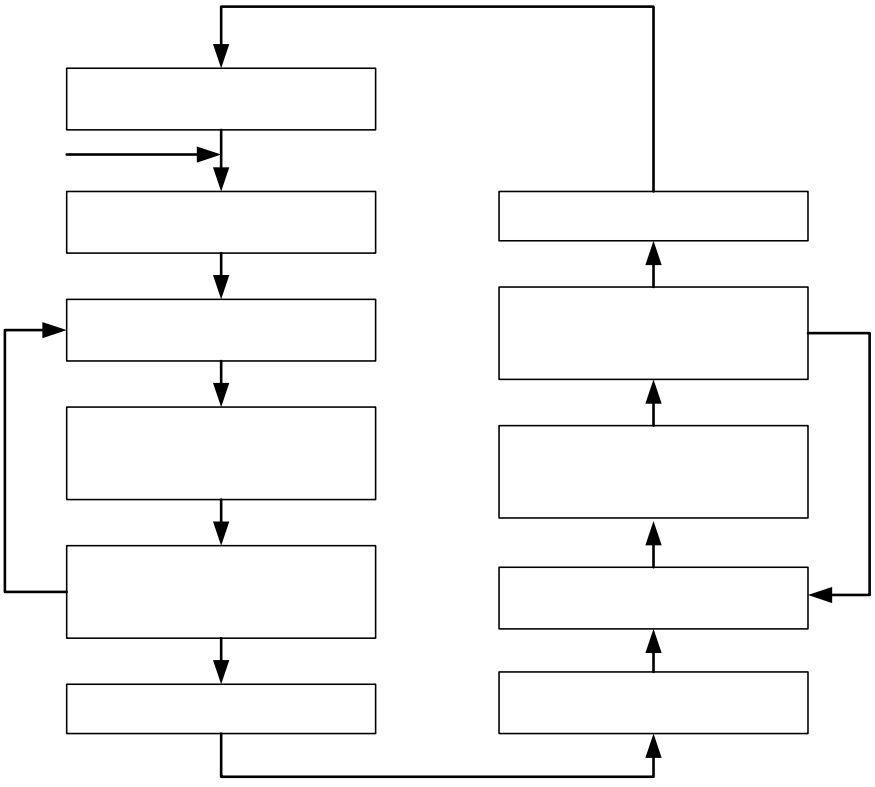


Figura 5.16: Diagrama de estados do processo de correlação.

Repouso

Iniciar

Reinicialização

5.1.4. Adaptação do processador de teste

A partir da descrição da metodologia de teste e do sistema de implementação, descritos nas secções anteriores, procede-se à adaptação do processador recorrendo à ferramenta computacional descrita no capítulo precedente. A adaptação inicia-se com a inserção dos recursos de teste que garantam a funcionalidade do processador, de acordo com a funcionalidade apresentada pelo algoritmo em pseudo código ilustrado na figura 5.3.

5.1.4.1. Inserção de recursos de teste

Nesta secção apresenta-se procedimento relativo à inserção dos recursos responsáveis pelo controlo do conversor A/D, geração do estímulo de teste, captura das respectivas respostas e pelas operações de processamento a realizar sobre as amostras capturadas. A funcionalidade pretendida por estes recursos foi apresentada na secção anterior.

Esta inserção é efectuada recorrendo a ficheiros HDL, apresentados no capítulo anterior, que contêm a caracterização individual de cada recurso e da instrução responsável pelo seu escalonamento. Cada um destes ficheiros é identificado por uma instrução responsável pela instanciação do recurso no processador e pelo seu escalonamento durante as operações de teste. Cada instrução é armazenada num ficheiro em formato tabular (*tasmv50.tab*), lido pelo TASM no início da geração do processo do código-máquina.

A instrução *TEST_CS5330A* identifica o recurso responsável pelo controlo do conversor A/D, enquanto que as instruções *SETDDS* e *XCORR* actuam sobre os recursos responsáveis pela geração do estímulo de teste e pelas operações de processamento sobre as amostras capturadas, respectivamente.

Tabela 5.4: Instruções para o controlo dos recursos do processador de teste.

Instruções para o controlo dos recursos do processador de teste.	
Instrução	**Operação elementar**
SETDDS	Efectua a instanciação do DDS como bloco responsável pela geração do estímulo de teste no processador, efectuando o seu escalonamento durante a execução do programa de teste.
XCORR	Efectua a instanciação do bloco correlador como recurso responsável pelas operações de processamento sobre as amostras capturadas.
TEST_CS5300A	Efectua a instanciação do bloco responsável pelo controlo do conversor A/D, configuração do modo de teste e captura das respostas de teste.

Os ficheiros HDL relativos aos recursos sobre os quais actuam as instruções definidas na tabela anterior são apresentados em anexo C.

A partir do conjunto de instruções apresentado no capítulo anterior e das que foram desenvolvidas nesta secção (tabela 5.4), procede-se ao desenvolvimento do programa de teste que traduza a funcionalidade pretendida para o processador de teste.

5.1.4.2. Desenvolvimento do programa de teste

Nesta subsecção efectua-se o desenvolvimento do programa para efectuar a testabilidade da interface analógico de aquisição de sinais do contador de energia apresentado na secção 5.1.2. O programa de teste é específico para o teste do conversor A/D inserido no circuito de condicionamento.

O programa deverá ser responsável pelo controlo de uma infra-estrutura de teste IEEE 1149.4 [5.4], através da qual é encaminhado o estímulo de teste, que consiste num sinal sinusoidal de 50 Hz e 4Vpp. Com a aplicação do estímulo de teste ao conversor A/D, procede-se ao seu controlo e à captura das respectivas respostas de teste. Terminada a captura das respostas de teste, procede-se ao seu processamento efectuando as operações de correlação conforme descrito na secção 5.1.1. Após a obtenção das assinaturas pretendidas é efectuada a reinicialização da infra-estrutura de teste e é sinalizado o fim da operação de teste.

A figura seguinte ilustra o programa desenvolvido para garantir as funcionalidades pretendidas, encontrando-se de acordo com o pseudo código ilustrado na figura 5.3.

```
NHARMONICS   .EQU  5      ;no. of harmonics to determine
       SETDDS             ;specify DDS as stimuli generator
                          ;sinusoidal test stimuli signal
       MOVC   R1, 1
       ST     (128),R1
       MOVC   R1, 2
       ST     (129),R1
       MOVC   R1, 3
       ST     (130),R1
       MOVC   R1, 4
       ST     (131),R1
       MOVC   R1, 5
       ST     (132),R1
       ; initialise BS infrastructure
       MOVC   CNT1, 11
       STATE  447         ;go to Shift-IR state
                          ;(through Test-Logic-Reset state)
       MOVC   CNT1,39
       NSHF   1048575            ;load EXTEST and BYPASS instruction
       MOVC   CNT1,5
       STATE  7           ;go to Shift-DR state
       MOVC   CNT1,48
       NSHF   4194561            ;load BS register
       MOVC   CNT1,2
       STATE  3           ;update BS register
       MOVC   CNT1,NHARMONICS
       ;start test execution
       STIMULUS
       TEST_CS5330A       ;schedules DUVctrl block for operation
L:     XCORR              ;performs cross-correlation operation
       STIMULUS
       DJNZ   CNT1,L      ;go to next harmonic
       MOVC   CNT1, 6
       STATE  3F          ;go to Test-Logic-Reset state
       EOTEST
       HLT
.END
```

Figura 5.17: Programa de teste.

O programa de teste inicia-se com a especificação do número de harmónicos a obter (*NHARMONICS .EQU 5*), procedendo-se de seguida ao armazenamento dos vectores digitais de teste correspondentes no segmento de memória apropriado. A instrução *SETDDS* efectua o escalonamento do DDS como o módulo responsável pela geração do estímulo de teste.

O controlo da infra-estrutura de teste é efectuado de seguida, programando o STA400 de entrada com a instrução *BYPASS*, desligando toda a lógica interna do integrado, e o adjacente ao conversor com a instrução *EXTEST*, programando o TBIC e os ABMs com a sequência de bits necessária de modo a permitir que o estímulo de teste

presente no pino AT1 (p9) seja encaminhado para o conversor pelos pinos A01 (p19) e A23 (p17), através do barramento interno AB1 do integrado, como se ilustra na figura 5.18.

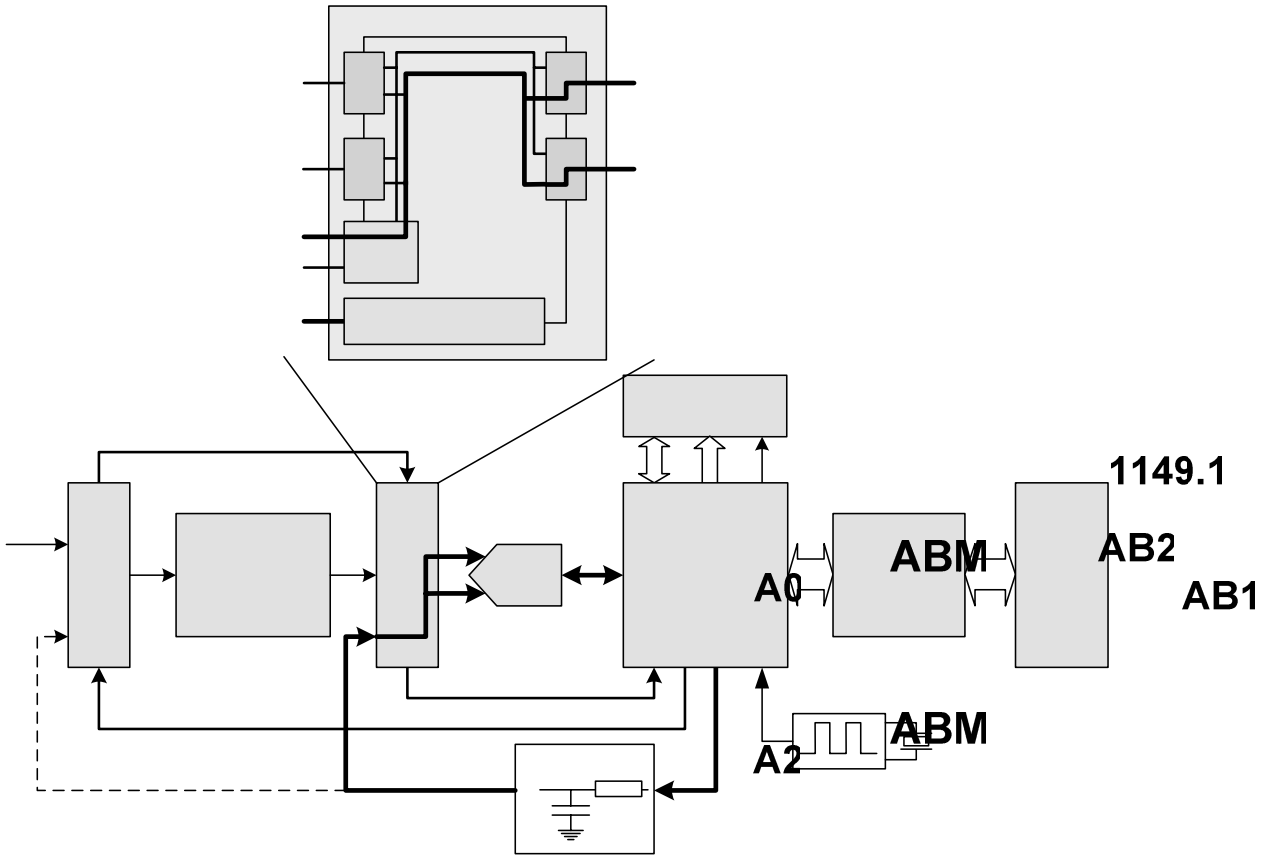


Figura 5.18: Percurso do estímulo de teste através da infra-estrutura de teste IEEE 1149.4.

Com a infra-estrutura de teste configurada, inicia-se a geração do estímulo de teste aplicando ao DDS o vector digital de teste armazenado na primeira posição do segmento de memória respectiva (*STIMULUS*). Com o estímulo a ser aplicado, o módulo responsável pelo controlo do conversor A/D é escalonado (*TEST_CS5330A*) para execução, esperando pela sua inicialização e estabilização da resposta perante a aplicação do estímulo de teste, procedendo-se à captura as amostras da resposta de teste.

O conjunto formado pelas três instruções seguintes (*XCORR, STIMULUS* e *DNJZ*) permitem a obtenção do ganho e do desvio de fase introduzido pelo conversor às frequências 50Hz, 100Hz, 150 HZ, 200Hz e 200Hz, que são os harmónicos pretendidos.

Com o processamento efectuado sobre as amostras da resposta de teste concluída, efectua-se o *reset* da infra-estrutura de teste e sinaliza-se o fim da operação de teste (*EOTEST* e *HLT*).

5.1.4.3. Geração automática do processador de teste

A partir do programa de teste, ilustrado na figura 5.17, procede-se à geração automática do processador de teste para garantir as funcionalidades definidas pelas instruções presentes no programa de teste. A implementação do programa de teste ocorre de acordo com a descrição efectuada no capítulo anterior.

A informação temporária que contém a informação relativa à geração do processador é apresentada de seguida. A figura 5.19 contém informação relativa à implementação das operações de teste, sendo possível visualizar a informação relativa à etapa de descodificação e execução das instruções associadas às operações, assim como os sinais envolvidos durante estas etapas e o código de operação (opcode) relativo a cada instrução.

Os recursos a serem incluídos na arquitectura proposta para o processador, para que se garanta a funcionalidade pretendida, encontram-se definidos no ficheiro ilustrado na figura 5.20.

Analisando estas ilustrações, é possível verificar a existência de uma correspondência directa entre as instruções definidas e os recursos sobre os quais actuam. Por exemplo, a instrução *XCORR*, definida no ficheiro ilustrado 5.19, actua sobre o recurso *xcorr_module* definido no ficheiro ilustrado na figura 5.20 através dos sinais *eofxcorr* e *enxcorr* (a negrito em ambas as figuras). O mesmo ocorre para a instrução TEST_CS5330A que actua sobre o recurso *DUVctrl_cs5330A*, responsável pelo controlo do conversor A/D, através dos sinais *eofduvtest* e *enduvtest* (a negrito em ambas as figuras).

```
//ctrlunit_testop
      // ********* Stimulus Instruction  *********
         4'b1000: begin
                        enstimuli = 1;      // opcode : 48(h)
                        enstimuli_addr = 1;
                        nextstate = STIMULop2;
                  end
                              .............

      // ********* TEST_CS5330A Instruction  *********
         4'b0011: begin
                        enduvtest = 1;            // opcode : 43(h)
                        if ( eofduvtest )
                              nextstate = INITIAL;
                        else
                              nextstate = DECODE;
                   end
      // ********* XCORR Instruction  *********
         4'b1001: begin          // opcode:49(h)
                        enxcorr = 1;
                        if ( eofxcorr )
                              nextstate = INITIAL;
                        else
                              nextstate = DECODE;
               end
                              .............
//ctrlunit_testop_end

//ctrlunit_op2Exec
    // ********** Continue LOAD Instruction  ********** //
            LOAD2 : begin                        // continue LOAD inst
                        en_mem = 1;              // continue LOAD inst
                        en_wreg = 1;             // continue LOAD inst
                        nextstate = INITIAL;     // continue LOAD inst
                     end                         // continue LOAD inst
                              .............

    // ********* Continue Stimulus Instruction  *********
            STIMULop2: begin                      //continue STIMULI
                        setstimuli = 1;          //continue STIMULI
                        nextstate = INITIAL;     //continue STIMULI
                      end                        //continue STIMULI
                              .............

    // ********** Continue HALT Instruction  ********** //
            HALT2: begin                        //continue HLT Instruction
                        busy = 0;               //continue HLT Instruction
                        nextstate = HALT2;      //continue HLT Instruction
                    end                         //continue HLT Instruction

//ctrlunit_op2exec_end
```

Figura 5.19: Extracto da estrutura de dados interna referente à descodificação e execução das instruções referenciadas.

```
//mc_surround_blocks
//************************************************************
//This section implements the test signal generator composed by
//one 12-bit DDS connected to an 12-bit SIgma-Delta Modulator
// 12-bit DDS
wire rstdds, endds_core, enddsoffset;
wire [11:0] ddsout;

dds_enable enabledds(.endds(endds_core),.clk(clock), .reset(reset) );

dds12bit DDS( .ddsout(ddsout), .clk(clock), .reset(reset), .rstdds(rstdds),
.endds(endds), . enoffset(enddsoffset), .ddsin({4'b0,stimuli}));

//12-bit Sigma-Delta Modulator module
sdm12bit SigmaDelta(.clock(clock),.areset(reset),.din(ddsout[11:0]),.dout(Yn)
);

//************************************************************
// The following controls the test operation to the Device Under Verification
// (DUV)

wire enladseg_duv, rstladseg_duv, enradseg_duv, rstradseg_duv, eofduvtest,
enduvtest;
wire [7:0] duvdata;

assign mclk = enduvtest ? clock : 0;

DUVctrl_cs5330A duvcrtl( .outsample(duvdata), .eofs2p(eofduvtest),
.mem_we(mem_we_duv), .enladseg(enladseg_duv), .enradseg(enradseg_duv),
.en_dataaddr(endsaddr_duv), .sels2p(selS2P), .mclk(clock), .sclk(sclk),
.lrck(lrck),.reset(reset),.ens2p(enduvtest),.sdata(sinput),.endds(endds));

//************************************************************************
// The following module implements a Correlator used as a processing unit

wire rstdds_xc, endds_xc, eofxcorr, enxcorr, lad, mem_we_xc, enINIxc,
endsaddr_xc, enladseg_xc, enradseg_xc, rstladseg_xc, rstradseg_xc;
wire [7:0] xcdata;

muxendds mux_endds( .endds(endds),.endds_core(endds_core), .enfxc(enxcorr),
.endds_xc(endds_xc));

muxrstdds mux_rstdds(.rstdds(rstdds),.rstdds_core(rstdds_core),
.enfcore(enfcore),.rstdds_xc(rstdds_xc),.enfxc(enxcorr));

xcorr_module xcmodule( .eofxcorr(eofxcorr), .clk(clock), .mem_we(mem_we_xc),
.enINI(enINIxc),.en_dataaddr(endsaddr_xc),.enladseg(enladseg_xc),
.reset(reset),.enradseg(enradseg_xc),.rstladseg(rstladseg_xc), .selxc(selxc),
.rstradseg(rstradseg_xc), .xcout(xcdata),.enDDS(endds_xc), .xcin1(dbusout),
.rstDDS(rstdds_xc),.enDDSoffset(enddsoffset),.enxcorr(enxcorr),
.xcin2(ddsout[11:0]));

//************************************************************************
                                ............
//mc_surround_blocks_end
```

Figura 5.20: Extracto da estrutura de dados interna referente aos recursos de teste a inserir no processador.

Na figura 5.21 apresenta-se a arquitectura do processador, gerada de acordo com o programa de teste ilustrado na figura 5.17, sendo o seu núcleo (*Test Processor Core*) rodeado pelas unidades funcionais que garantem a funcionalidade pretendida.

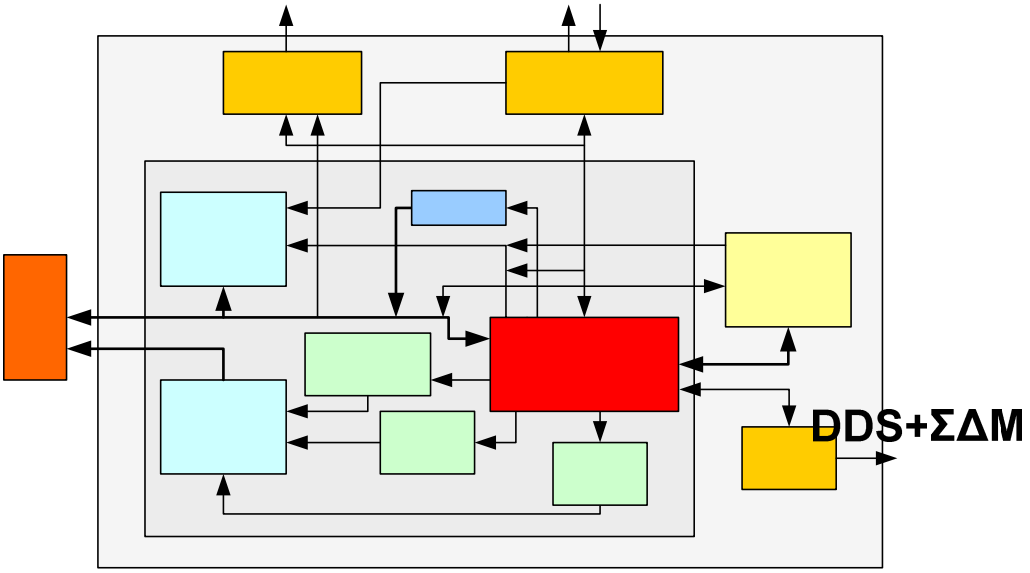


Figura 5.21: Arquitectura do processador para o teste do conversor CS5330A.

Uma vez que estas unidades partilham recursos comuns, apenas é permitido, em cada instante, o acesso a esses recursos à unidade que se encontra em execução, sendo o acesso das restantes unidades efectuado à medida que são escalonados para execução. Por exemplo, durante a fase de captura das respostas de teste apenas a unidade funcional responsável, CS5330Actrl, por esta operação possui acesso à memória de dados e programa, o mesmo se passa com a unidade responsável por efectuar operações de processamento sobre as respostas capturadas, *XCORR*.

A figura seguinte ilustra a interface do processador de teste para o teste do conversor CS5330A, obtido do editor de esquemático do *Foundation*.

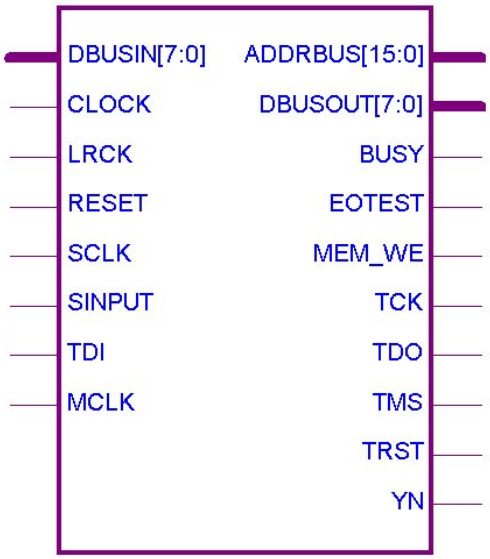


Figura 5.22: Interface do processador para o teste do conversor CS5330A.

A implementação do processador atrás descrito na FPGA ocupa um total de 558 CLBs de 576 disponíveis. A tabela 5.5 resume a contribuição espacial dos principais recursos na constituição do processador de teste.

Tabela 5.5: Contribuição dos principais recursos que constituem o processador de teste para a área total de implementação.

Recurso	Área (Flip-Flops)
STIMULIctrl	51
BScrtl	42
XCORR	231
CS5330Actrl	171
CTRLUNIT	36
REGS	16

5.1.5. Resultados obtidos

A tabela 5.6 resume os resultados obtidos para as correlações entre a resposta do conversor A/D com o correspondente estímulo de entrada e com um sinal em quadratura (co-seno), $Ryx_s(n)$ e $Ryx_c(n)$, respectivamente.

Tabela 5.6: Correlação entre a resposta capturada do conversor A/D com o estimulo de entrada, $Ryx_s(n)$, e sinal em quadratura, e $Ryx_c(n)$.

	AINL	**AINR**
$Ryx_s(0)$	31361077	31501633
$Ryx_c(0)$	-8396422	-8434425
$Ryx_s(1)$	-24230	-25253
$Ryx_c(1)$	22253	10820
$Ryx_s(2)$	-7148	-7425
$Ryx_c(2)$	-9490	-9706
$Ryx_s(3)$	-13431	-13246
$Ryx_c(3)$	1621	-31
$Ryx_s(4)$	-544	-631
$Ryx_c(4)$	-6616	-6470

Os valores presentes na tabela anterior são usados no cálculo para a obtenção do ganho e do desvio de fase introduzido pelo conversor A/D às diversas frequências, e para determinar a respectiva distorção harmónica total, THD, de acordo com as equações 5.1, 5.2 e 5.3, presentes na subsecção 5.1.1.

A tabela 5.7 resume os resultados obtidos para as amplitudes às frequências dos harmónicos pretendidos.

Tabela 5.7: Valores (em dB) obtidos para o ganho introduzido pelo conversor A/D e correspondente distorção harmónica total.

H_n/H_1	AINL	AINR
h2	-59.885	-61.489
h3	-68.732	-68.526
h4	-67.604	-67.826
h5	-73.787	-74.008
THD	-58.613	-59.768

Conforme se pode observar, estes valores afastam-se dos valores esperados, pois consoante o espectro obtido por simulação à saída do modulador sigma-delta (ΣΔ), figura 5.14, seriam esperados harmónicos com amplitudes inferiores a -88dB. Contudo, estes valores encontram-se de acordo com os espectros ilustrados nas figuras 5.23 e 5.24, obtidos a partir das respostas de teste capturadas do conversor A/D pelo controlador de teste.

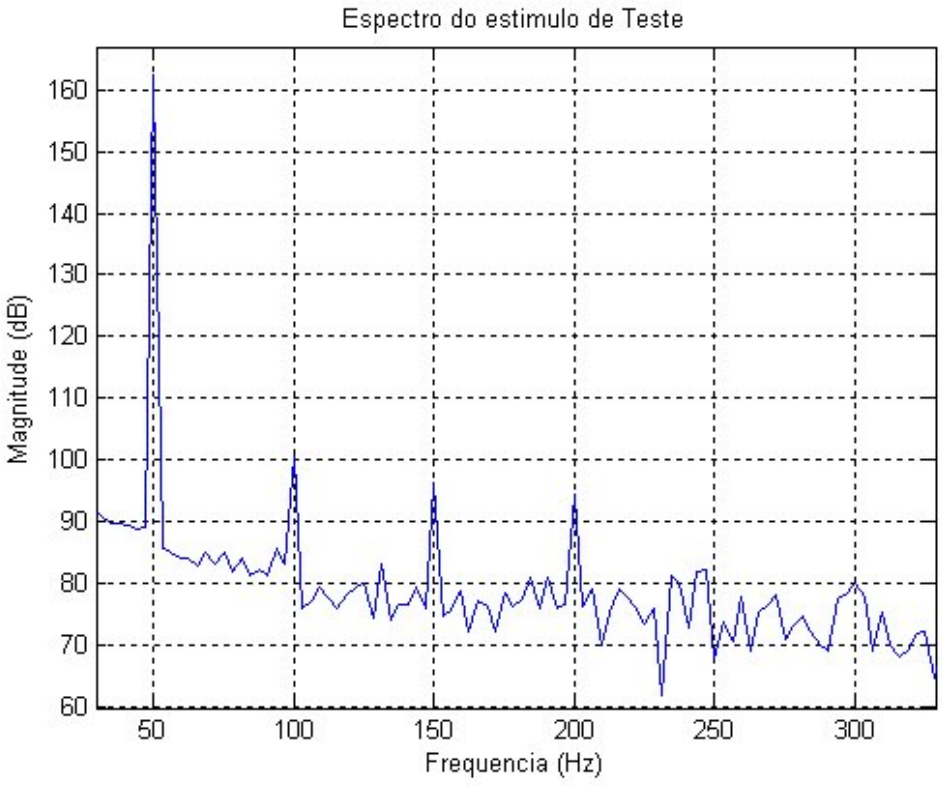


Figura 5.23: Espectro obtido para a resposta do estimulo de teste do canal AINR do conversor A/D.

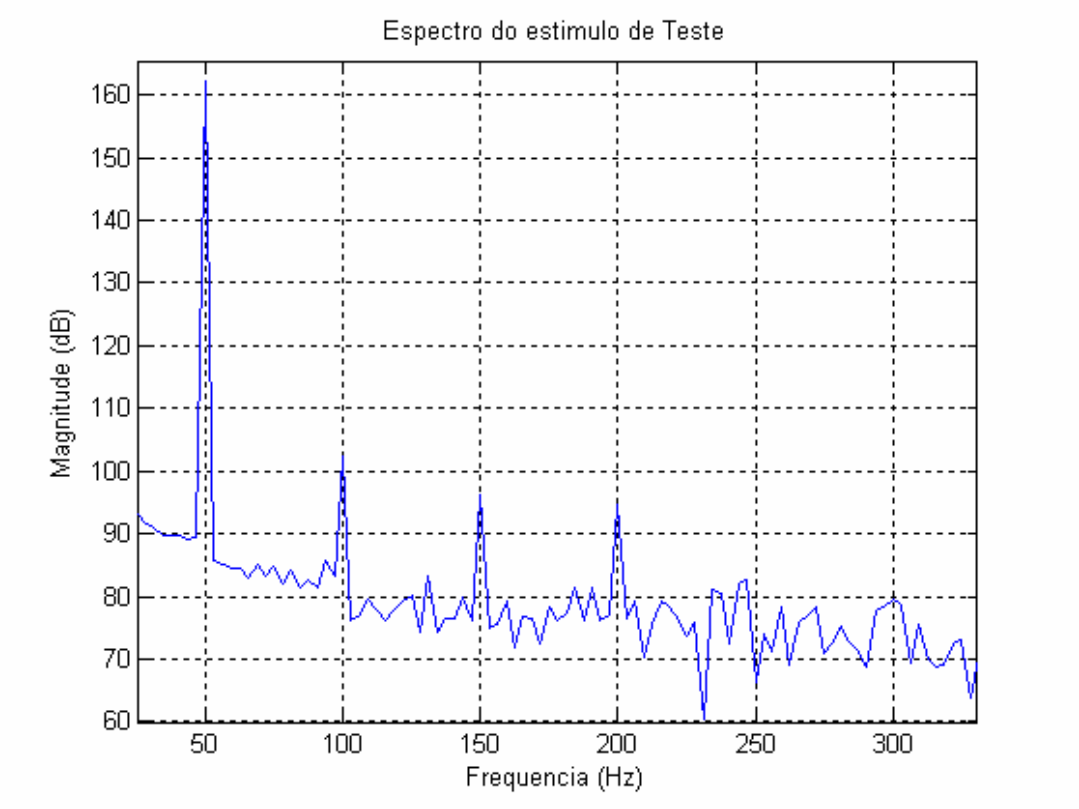


Figura 5.24: Espectro obtido para a resposta do estimulo de teste do canal AINL do conversor A/D.

Como os valores da tabela 5.7 se encontram de acordo com o espectro ilustrado na figura anterior, será razoável, numa primeira hipótese, considerar que a distorção registada é introduzida pelo conversor ou então pela infra-estrutura, através da qual é encaminhado o estímulo de teste. Deste modo, procedeu-se à obtenção do espectro do estímulo de teste isolando-o do conversor e da infra-estrutura de teste. A figura seguinte ilustra o estímulo de teste e o espectro obtido após o filtro RC passa-baixo, por intermédio de um osciloscópio [57], enquanto que na figura 5.26 encontra-se ilustrado o espectro obtido recorrendo a uma placa de aquisição de dados [58].

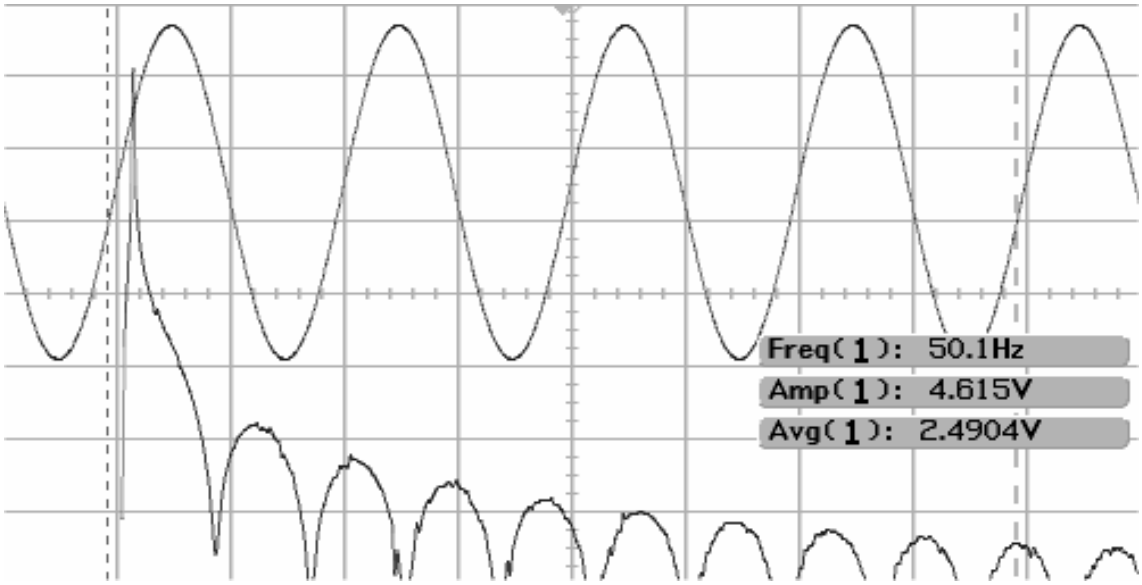


Figura 5.25: Estímulo de teste e respectivo espectro obtido por um osciloscópio.

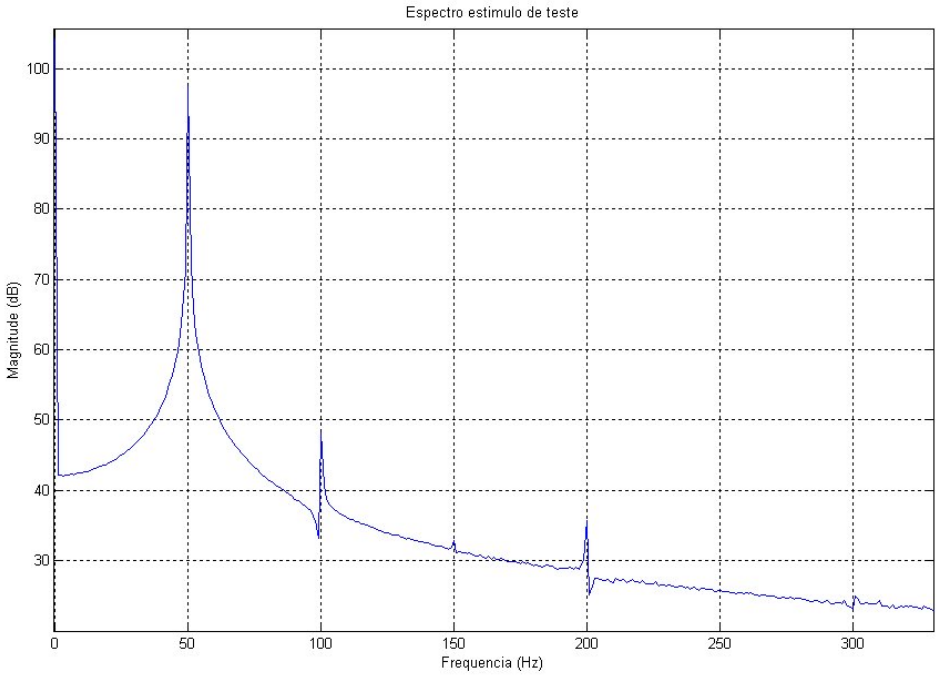


Figura 5.26: Espectro do estímulo de teste 'isolado' após o filtro passa-baixo.

Verifica-se, através das figuras anteriores, a presença de harmónicos de amplitude elevada às frequências múltiplas da fundamental, pelo que se conclui que a distorção registada é introduzida pelo estímulo de teste.

Uma vez que se verifica que o próprio estímulo de teste possui harmónicos de amplitudes elevadas às frequências múltiplas da fundamental, é necessário considerar a hipótese de que é a unidade funcional responsável pela geração do estímulo de teste,

descrita na secção 5.1.3.2, responsável por esta distorção. Isto é, não se encontra a gerar correctamente o sinal de teste pretendido, apesar dos resultados obtidos das simulações efectuadas no ModelSim e Matlab (ver figura 5.14). Por este facto, procedeu-se à aquisição, na FPGA, de dados relativos ao estímulo de teste no DDS e no modulador sigma-delta (ΣΔ), sendo salvaguardados na memória disponível.

As figuras seguintes (5.27 e 5.28) ilustram o espectro do estímulo de teste, nas situações descritas, obtido do processamento efectuado pelo Matlab das amostras capturadas.

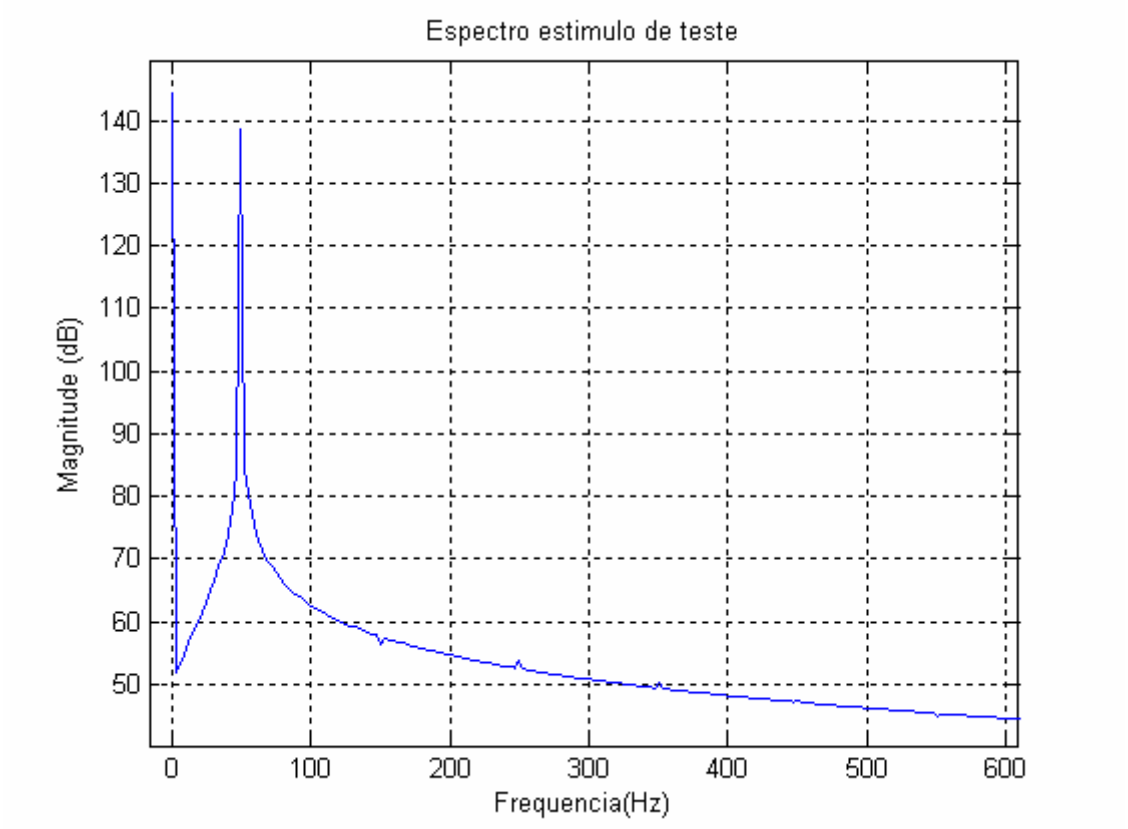


Figura 5.27: Espectro do estímulo de teste na FPGA após DDS.

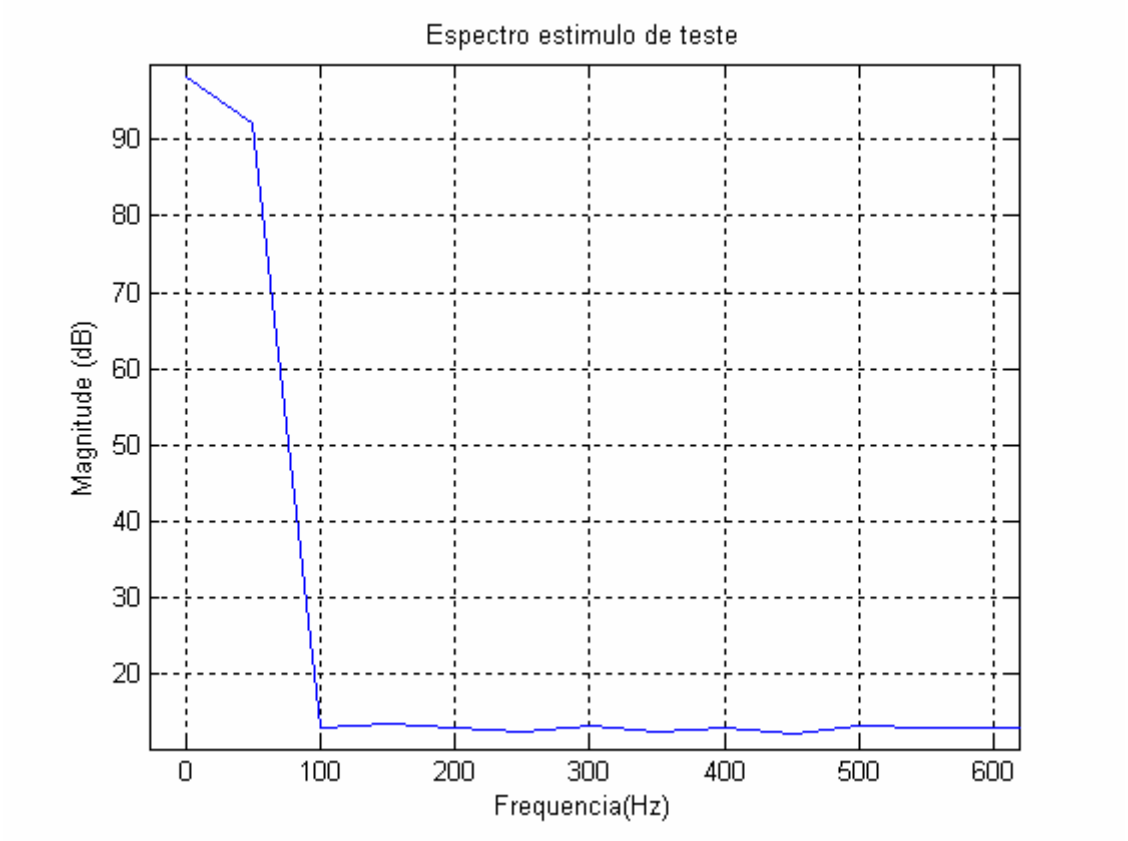


Figura 5.28: Espectro do estímulo de teste na FPGA após modulador sigma-delta (ΣΔ).

Os espectros obtidos para o estímulo de teste, nas situações especificadas, confirmam que se encontram dentro da gama de valores pretendidos, de acordo com o espectro obtido pela simulação na subsecção 5.1.3.2. Até ao momento não foi possível identificar a origem da distorção presente no estímulo de teste, podemos ainda considerar as hipóteses que se apresentam de seguida.

A distorção presente no estímulo de teste pode-se dever às flutuações na amplitude do *bitstream*, a fornecer ao filtro RC passa-baixo, na saída do FPGA, sendo necessário efectuar uma adaptação de tensão deste sinal. Justifica-se então a introdução de um *buffer* para adaptar a saída da FPGA à carga que está sendo dirigida. Esta adaptação foi efectuada colocando à saída da FPGA uma porta inversora do tipo *Schmitt-trigger* [59], sendo a sua saída encaminhada para o filtro passa-baixo. Com esta adaptação procedeu-se à aquisição de novos espectros através da placa de aquisição de dados. A figura seguinte ilustra um dos espectros obtidos, verificando-se novamente que a distorção continua presente no estímulo de teste.

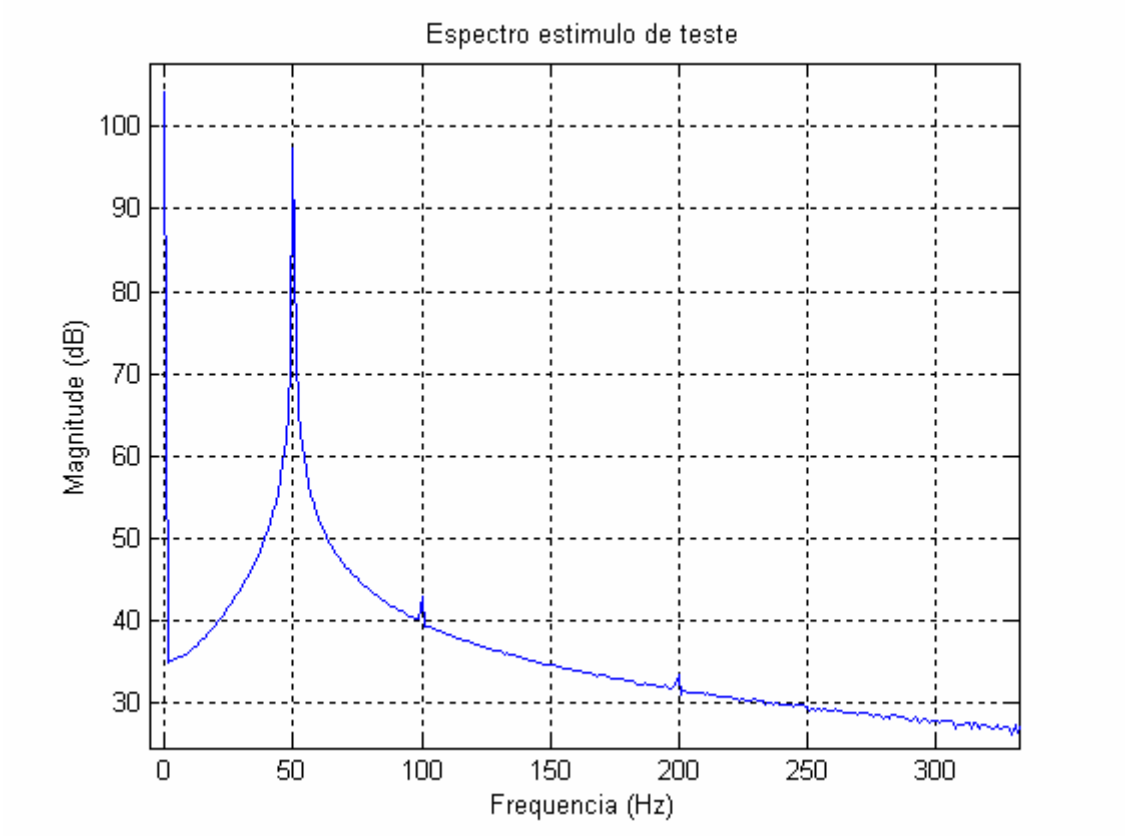


Figura 5.29: Espectro do estímulo de teste após se ter efectuado a adaptação da saída da FPGA.

Deste modo, pode-se considerar, como última hipótese, que a distorção é introduzida pela rede de alimentação, uma vez que a frequência do estímulo de teste (determinado pela própria funcionalidade pretendida para o circuito) possui o mesmo valor da frequência da rede eléctrica, isto é, 50Hz.

Para verificar esta hipótese, a frequência de amostragem do sinal digital produzido pelo DDS, foi modificada, de acordo com a equação 5.4, de modo a obter um sinal com uma frequência que não se encontre correlacionada com a frequência da rede de alimentação. A frequência de amostragem de 35840 Hz (ver tabela 5.3) permite gerar um sinal de 70 Hz com as mesmas características do estímulo de teste. A figura seguinte ilustra o espectro obtido para este sinal, capturado pela placa de aquisição de dados, Continuando-se a verificar a presença de harmónicos de amplitude elevada às frequências múltiplas da fundamental.

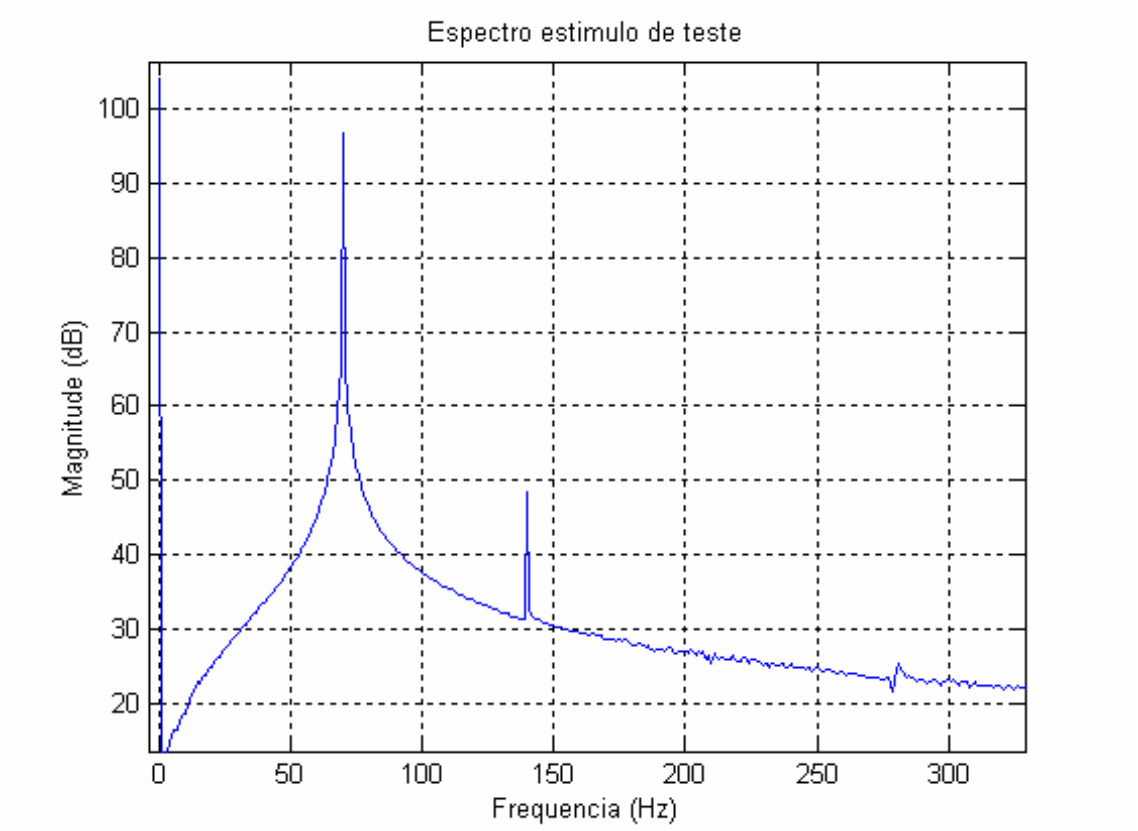


Figura 5.30: Espectro para um sinal sinusoidal de 70 Hz.

Apesar das diversas hipóteses consideradas até ao momento não foi possível identificar a origem da introdução da presença da distorção no sinal de teste. Apenas se pode concluir que a unidade funcional desenvolvida para o processamento das respostas de teste capturadas pelo processador se encontra a operar correctamente.

5.2. Teste de circuitos por estimação do conteúdo harmónico.

Esta secção é dedicada à descrição do processo de concepção e implementação do processador de teste, baseado na estimação do conteúdo harmónico, para determinar os diversos parâmetros que possibilitam a caracterização de um circuito que contém um conversor A/D.

5.2.1. Estimação do conteúdo harmónico para o teste de circuitos

A metodologia de teste [60] a implementar, permite estimar um número genérico de harmónicos introduzidos por um conversor A/D a partir de um conjunto reduzido de assinaturas/valores, que representam as coordenadas da sua função de transferência. Para além da avaliação do conteúdo harmónico, permite igualmente determinar outros parâmetros, como por exemplo, o ganho, desvio na origem (*offset*), a distorção harmónica total (THD), que possibilitam a caracterização de conversores A/D.

Este algoritmo baseia-se no facto de os harmónicos gerados por um conversor A/D se encontrarem matematicamente relacionados com os coeficientes do polinómio que define a sua função de transferência. Num conversor não-ideal, a sua função de transferência é definida por uma função polinomial com coeficientes não nulos, p_i, de acordo com a seguinte equação:

$$y(t) = p(t) = \sum_{i=0}^{k-1} p_i x^i(t) \qquad (5.12)$$

em que *y(t)* corresponde ao sinal de saída do conversor, *x(t)* o respectivo sinal de entrada.

O polinómio que caracteriza a função de transferência do conversor é obtido recorrendo à função *polyfit (x,y,k)*, presente no Matlab [64], que retorna o polinómio de grau *k-1* que melhor se ajusta ao conjunto de coordenadas *(x, y)*. O par de coordenadas (x, y) relativas a *k* pontos da função de transferência, são obtidas aplicando um estímulo de teste constituído por *K* níveis DC (coordenada *x*), uniformemente distribuídos, sobrepostos por um sinal alternado de valor médio nulo, como se encontra ilustrado na figura seguinte.

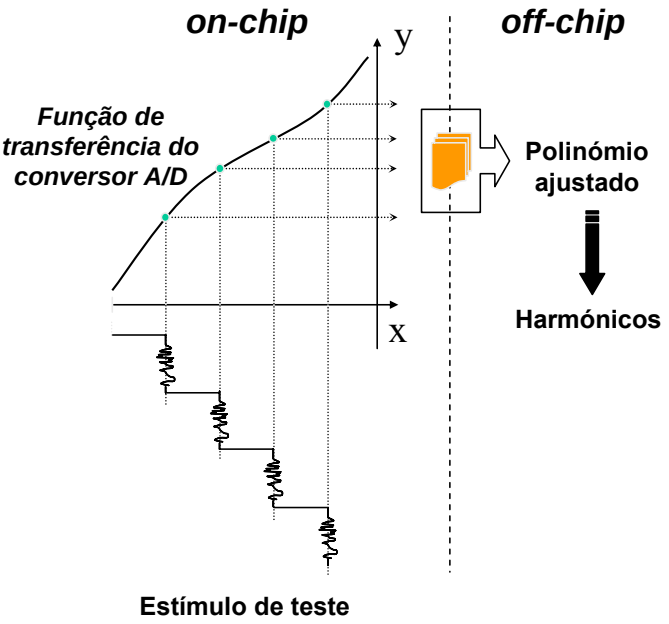


Figura 5.31: Representação gráfica do algoritmo de teste.

A captura de um determinado número de amostras, *M*, para cada código de saída correspondente ao nível DC aplicado, permite obter a coordenada *y*, de acordo com a seguinte equação:

$$y_n = \frac{\sum_{i=1}^{M} p(x_n)}{M}, n = 1...k \qquad (5.13)$$

Conhecido o conjunto de coordenadas da função de transferência e o polinómio que melhor se ajusta, é possível obter os harmónicos introduzidos pelo conversor. Para além da estimação do conteúdo harmónico, é possível determinar parâmetros adicionais que permitem caracterizar estaticamente e dinamicamente o conversor AD, determinando o valor correspondente ao ganho (*G*), desvio na origem (*offset*, V_{off}), a distorção harmónica total (THD), de acordo com as seguintes equações:

$$V_{off} = h_0 \qquad (5.14)$$

$$G = 20\log_{10}\left(\frac{h1Q}{V}\right) \qquad (5.15)$$

$$THD = 2o\log_{10}\left(\frac{\sum_{n=2}^{k-1} h_n^2}{h_1^2}\right) \qquad (5.16)$$

5.2.2. Sistema de implementação

A implementação desta solução foi efectuada recorrendo a uma plataforma reconfigurável construída em torno de uma FPGA (*Field Programmable Gate Array*), XC4013EPG223-4, da Xilinx [50], montado numa placa de circuito impresso (PCB – *Printed Circuit Board*). O sistema inclui um conversor A/D [61], uma memória RAM [51], sendo a infra-estrutura de teste baseada em multiplexadores analógicos [62]. Na 5.32 apresenta-se o diagrama de blocos do protótipo de teste.

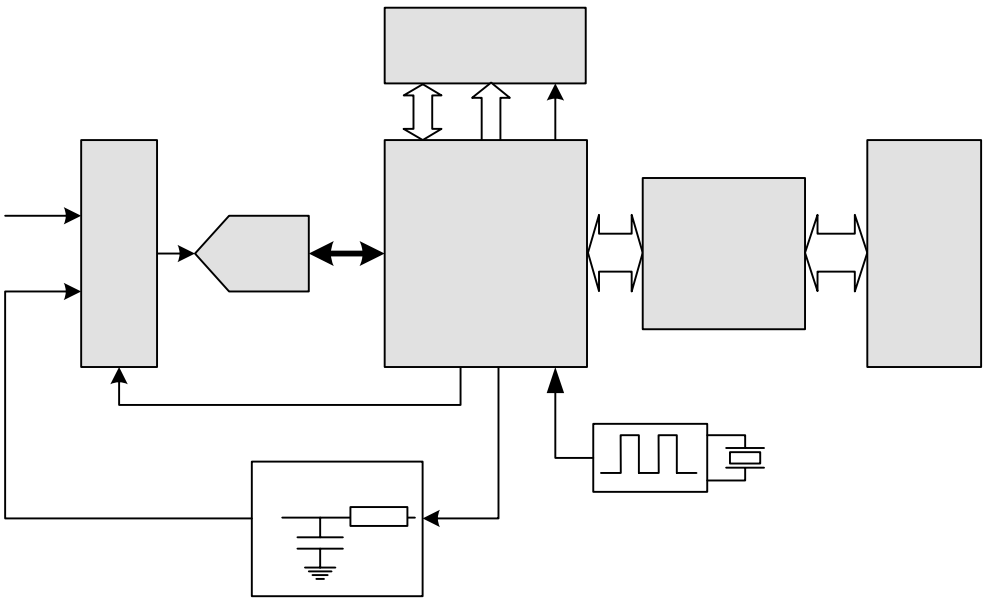


Figura 5.32: Placa de expansão para PC de suporte à implementação do processador de teste para a estimação do conteúdo harmónico de um ADC.

Esta infra-estrutura de teste apresenta vantagens face a uma infra-estrutura compatível com a norma IEEE 1149.4 [6], pois dispensa a implementação do controlador na FPGA e reduz o tempo necessário para programar e comutar entre as diferentes entradas (funcionamento normal e estímulo de teste) dos circuitos integrados

SCANSTA400 [46], da National Semiconductors, para injectar o estímulo de teste ao sistema a testar.

Os circuitos destinados ao controlo da configuração da FPGA foram previamente descritos na secção 5.1.2.

5.2.3. Implementação do processador de teste

Como foi descrito previamente, subsecção 5.1.2, todo o projecto deverá ser desenvolvido como um circuito síncrono com um único domínio de relógio (SYSCLK = 8MHz), sendo as secções que operam a cadências inferiores sincronizadas com o SYSCLK e habilitadas por sinais de *clock enable* produzidos por um módulo de gerador de relógio.

O dispositivo de FPGA é reconfigurado com o processador de teste, o qual deve ser adaptado com o objectivo de caber na FPGA para acomodar a aplicação. O processador a desenvolver deverá realizar algumas das operações elementares descritas na subsecção 3.2.2 do capítulo 3. Na figura 5.33 apresenta-se, em pseudo código, a funcionalidade apresentada pelo processador de teste para o algoritmo de teste descrito na subsecção 5.2.1.

```
para (componente ){
        configurar infra-estrutura de teste;
        configurar modo de teste;
        aplicar K níveis DC{  /* coordenadas x*/
                para (nível DC ){
                        capturar M amostras;
                        obter os respectivos valores médios; /* coordenadas y*/
                };
        };
};
```

Figura 5.33: Funcionalidade apresentada pelo processador de teste.

O processador é responsável pelo controlo adequado do componente a testar, configurando a infra-estrutura de teste, através da qual é encaminhado o estímulo de teste para o componente a testar. Para cada estímulo aplicado ao conversor A/D procede-se à captura de um determinado número de respostas de teste, sendo o valor médio obtido salvaguardado para a memória de dados. A figura 5.34 ilustra a arquitectura global do processador de teste, onde se visualizam as unidades responsáveis pelas operações identificadas na figura 5.33.

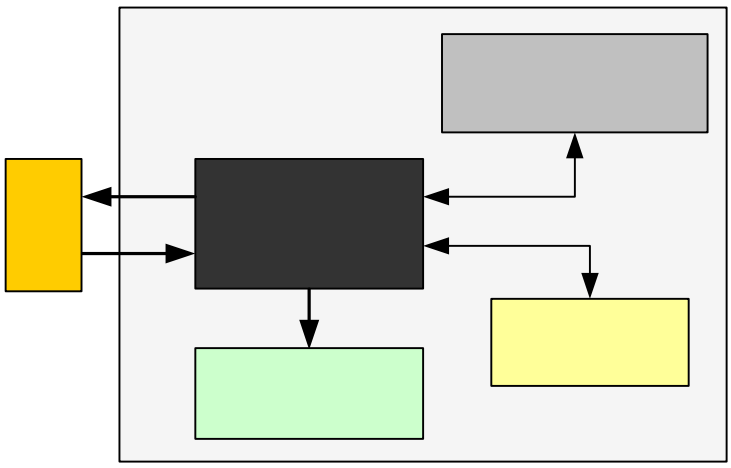


Figura 5.34: Arquitectura global do processador para o teste de circuitos por estimação do conteúdo harmónico.

Processador de tes

5.2.3.1. Controlador do dispositivo a testar

O dispositivo a testar consiste num conversor A/D de 8 bits, com interface paralela (TLC0820AC da Texas Instruments [61]), que recorre a uma combinação de técnicas *flash* modificadas para permitir a uma rápida conversão de dados. A figura seguinte ilustra a interface do conversor A/D com a FPGA.

M
E
M

Gerador
estímulo de te

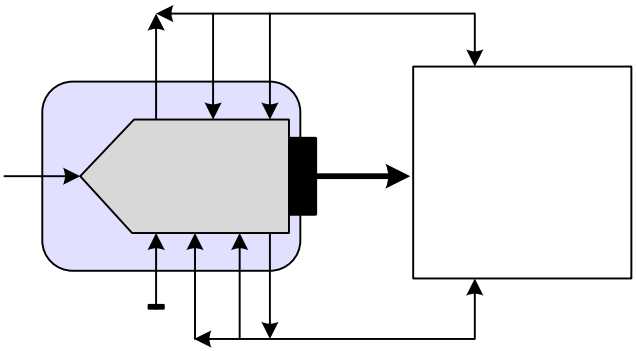


Figura 5.35: Interface do conversor A/D.

Os sinais envolvidos na funcionalidade do conversor são:

Ain

ADC

A_{in} : entrada do sinal analógico;

/EC : sinal de fim de conversão;

/OE : sinal de *enable*, activo baixo (nível lógico '0'), que coloca o resultado na saída digital;

Ref± Md /CS /Of

/SC : sinal de início de conversão, activo baixo (nível lógico '0');

/CS : sinal de selecção do conversor A/D, activo baixo (nível lógico '0');

/Oflw : sinal de *overflow*, activo baixo (nível lógico '0'), que sinaliza que o sinal analógico excedeu o valor de referência V_{REF+};

Ref+/- : sinais de referência que definem a gama de valores de entrada do sinal analógico ($V_{REF-} = 0V$, $V_{REF+} = 5V$);

Md : sinal de selecção do modo de operação do conversor;

D_{0-7} : saída digital de 8 bits que contém o resultado do processo de conversão.

A operação de teste ao conversor A/D inicia-se forçando o sinal Md ao nível lógico '1'. Neste modo de operação, a conversão é iniciada aplicando um impulso negativo no sinal /SC, encontrando-se concluída com a ocorrência de um impulso negativo no sinal /EC. Neste ponto o resultado do processo encontra-se armazenado num registo interno do conversor, sendo necessário aplicar um impulso negativo no sinal /OE para colocar respectivo resultado na saída digital, que de outro modo se encontra em alta impedância. Na figura seguinte encontra-se ilustrado o diagrama temporal do conversor para o modo de operação seleccionado.

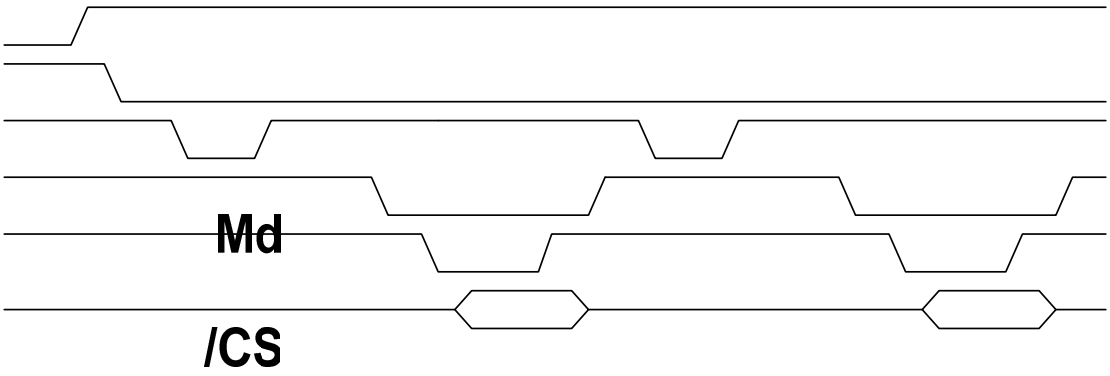


Figura 5.36: Diagrama temporal dos sinais envolvidos no processo de conversão.

O sistema a implementar inclui um controlador responsável pelo correcto sequenciamento do processo de controlo, iniciando-o após a recepção de um sinal enviado com esse fim pelo processador. O controlador selecciona o modo de conversão desejado para o conversor A/D, de acordo com a figura 5.36, e inicia o processo de conversão para um determinado número de níveis DC, capturando para cada um destes níveis um determinado número de amostras, sendo o resultado das capturas, para cada nível, acumulado sucessivamente num registo interno. Após se ter capturado o número de amostras pretendido, procede-se à obtenção do respectivo valor médio, sendo o seu resultado salvaguardado na memória de dados. O diagrama de estados apresentado na figura seguinte ilustra a descrição comportamental do sistema a desenvolver (controlador do dispositivo a testar), e na figura 5.38 mostra-se o respectivo diagrama de blocos.

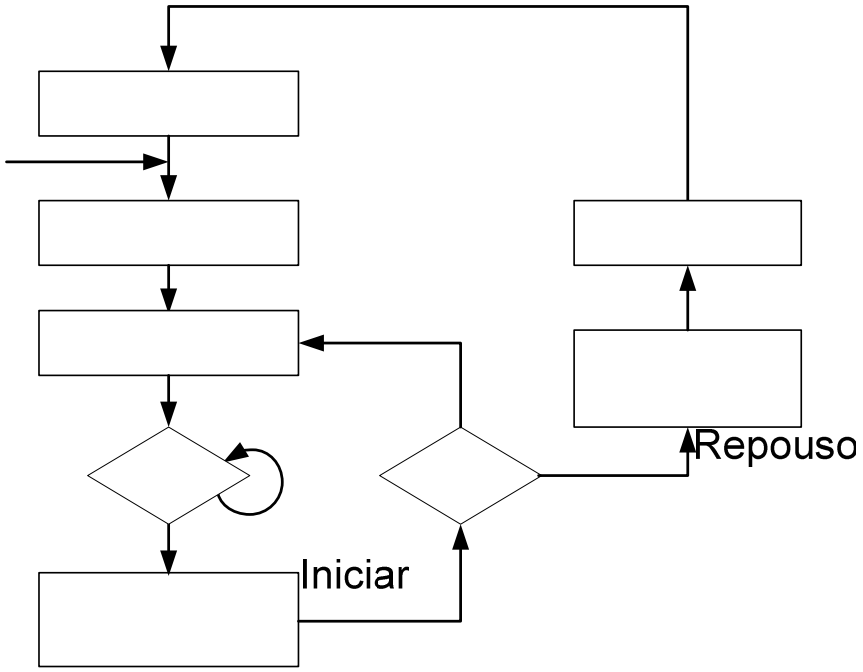


Figura 5.37: Diagrama de estados do controlador do dispositivo a testar.

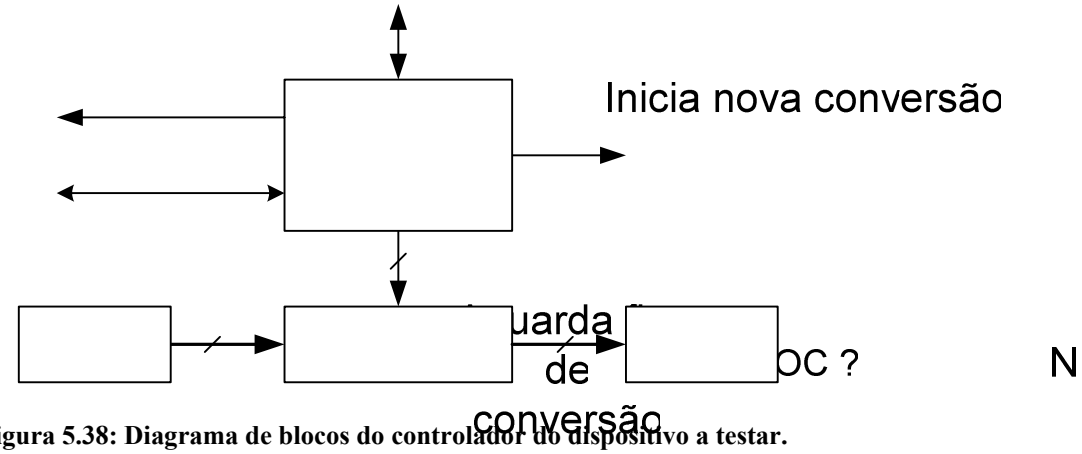


Figura 5.38: Diagrama de blocos do controlador do dispositivo a testar.

O módulo **TLC0820AC_ctrl** possui como principal funcionalidade o sequenciamento das operações relacionadas com o controlo do conversor A/D de acordo com modo de operação ilustrado na figura 5.36, enquanto que o módulo **ACC_reg** consiste num acumulador ao qual serão adicionadas as respostas de teste obtidas do conversor. O valor médio, de cada estímulo aplicado ao conversor, é obtido através de uma deslocação para a direita do número de bits equivalente ao número de respostas de teste capturadas.

5.2.3.2. Gerador do estimulo de teste

De modo a realizar a estimação do conteúdo harmónico introduzido pelo conversor A/D, é necessário aplicar um estímulo de teste constituído por um determinado número de níveis DC uniformemente distribuídos ao longo da gama de entrada do conversor, de acordo com equação seguinte:

$$x_n = \frac{n}{k+1} V_{FS}, n = 1...k \qquad (5.17)$$

onde k corresponde ao número de níveis DC a aplicar, x_n nível DC a aplicar (coordenada x).

Como gerador de estímulo de teste, a escolha recaiu sobre um modulador sigma-delta (ΣΔ) [55] em detrimento de um PWM (*Pulse Width Modulation*) [56], pelas mesmas razões descritas na secção5.1.3.2. Na mesma secção descreveu-se o princípio de funcionamento de um modulador deste tipo. A figura seguinte ilustra a arquitectura para um modulador sigma-delta (ΣΔ) de 8 bits de primeira ordem.

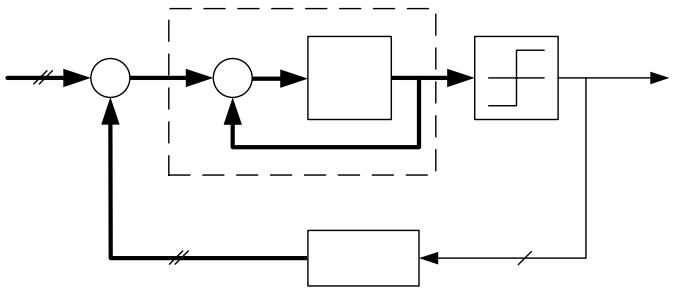


Figura 5.39: Modulador ΣΔ 8 bits de primeira ordem.

No presente caso o estímulo de teste consiste na aplicação de 16 níveis DC, uniformemente espaçados na gama de entrada do conversor A/D, isto é 0 – 5V. Na tabela seguinte apresenta-se os valores correspondentes aos níveis DC, coordenada x, a usar como estímulo de teste.

Tabela 5.8: Estímulo de teste a aplicar.

n	U(n) (hex)	Xn (V)	n	U(n) (hex)	Xn (V)
1	0F	0.2933	9	87	2.6535
2	1E	0.5879	10	96	2.9542
3	2D	0.8828	11	A5	3.2352
4	3C	1.1778	12	B4	3.5363
5	4C	1.4765	13	C3	3.8376
6	5B	1.7755	14	D0	4.1156
7	6A	2.0749	15	DF	4.4151
8	79	2.3744	16	EF	4.7138

Na tabela anterior encontram-se também os vectores digitais a aplicar ao modulador sigma-delta (ΣΔ) para gerar os respectivos estímulos de teste, isto é, níveis DC.

5.2.4. Adaptação do processador de teste

A partir da descrição da metodologia de teste e do sistema de implementação, descritos nas secções anteriores, procede-se à adaptação do processador recorrendo à ferramenta computacional descrita no capítulo precedente, e de um modo semelhante ao efectuado na secção 5.1.4. A adaptação inicia-se com inserção dos recursos de teste que garantam a funcionalidade do processador, de acordo com a funcionalidade apresentada pelo algoritmo em pseudo código ilustrado na figura 5.33.

5.2.4.1. Inserção de recursos de teste

Nesta secção apresenta-se procedimento relativo à inserção dos recursos responsáveis pelo controlo do conversor A/D, geração do estímulo de teste e captura das respectivas respostas, de acordo com a funcionalidade descrita na secção anterior.

Esta inserção é efectuada recorrendo a ficheiros HDL que contêm a caracterização individual de cada recurso e da instrução responsável pelo seu escalonamento. Cada ficheiro HDL é identificado por uma instrução responsável pela instanciação do respectivo recurso no processador e pelo seu escalonamento durante as operações de teste. Cada instrução é armazenada num ficheiro em formato tabular (*tasmv50.tab*), lido pelo TASM no início da geração do processo do código-máquina.

A instrução *TEST_TLC0820AC* identifica o recurso responsável pelo controlo do conversor A/D, sendo o recurso responsável pela geração do estímulo de teste identificado pela *SIGMADELTA8B*. As instruções *SRTEST*, *ACCTRESP* e *SVTRESP* actuam sobre o recurso responsável pelo controlo do conversor A/D, para iniciar a operação de conversão, acumular a resposta obtida e salvaguardar o valor médio para cada nível DC, respectivamente. O sincronismo entre o conversor A/D e o processador de teste é efectuado pela instrução *WAIT0*.

Tabela 5.9: Instruções para o controlo dos recursos do processador de teste.

Instruções para o controlo dos recursos do processador de teste.	
Instrução	**Operação elementar**
SIGMADELTA8B	Efectua a instanciação do modulador sigma-delta (ΣΔ) de 8bits como unidade responsável pela geração do estímulo de teste no processador.
TEST_TLC0820AC	Efectua a instanciação do bloco responsável pelo controlo do conversor A/D.
SRTEST	Sinaliza o controlador a iniciar uma nova conversão
WAIT0	Efectua o sincronismo entre o fim-de-conversão e a captura da respectiva resposta de teste.
ACCTRESP	Acumula as sucessivas respostas de teste capturadas para um determinado nível DC.
SVTRESP	Obtém o valor médio e salvaguarda-o na memória de programa e dados.

Os ficheiros HDL relativos aos recursos sobre os quais actuam as instruções definidas na tabela anterior são apresentados em anexo C.

A partir do conjunto de instruções apresentado no capítulo anterior e das que foram desenvolvidas nesta secção (tabela 5.9), procede-se ao desenvolvimento do programa de teste que traduza a funcionalidade pretendida para o processador de teste.

5.2.4.2. Desenvolvimento do programa de teste

Identificadas as instruções que garantem as operações de acordo com o algoritmo apresentado na figura 5.33, desenvolve-se, nesta subsecção, o programa para efectuar a testabilidade do conversor A/D.

O programa deverá ser responsável pelo controlo do multiplexador analógico [62], que constitui a infra-estrutura de teste, através da qual é encaminhado o estímulo de teste, constituído por um conjunto de níveis DC uniformemente distribuídos ao longo da gama de entrada do conversor. Para cada nível DC captura um determinado número de amostras, obtendo-se cada um dos níveis o respectivo valor médio, que será salvaguardado na memória de dados.

A figura 5.34 ilustra o programa desenvolvido para garantir as funcionalidades pretendidas, encontrando-se de acordo com o pseudo código ilustrado na figura 5.33.

O programa de teste inicia-se com a especificação do número de níveis DC (*NLEVELS .EQU 16*) aplicar e do número de amostras a capturar em cada nível (*NSAMPLES .EQU 255*), procedendo-se de seguida ao armazenamento dos vectores digitais de teste correspondentes aos respectivos níveis. A instrução *SIGMADELTA8B* efectua o escalonamento do modulador sigma-delta (ΣΔ) como o módulo responsável pela geração do estímulo de teste.

A instrução *TEST_TLC0820AC* efectua o escalonamento da unidade responsável pelo controlo do conversor A/D, e selecciona o multiplexador analógico para encaminhar o estímulo de teste para o conversor A/D, do pino NC1 (p5) para o pino COM1 (p3) através do barramento interno. Para que tal seja possível selecciona a interface IN1 do conversor ao nível lógico ‘0’.

O conjunto formado pelas três instruções *SRTEST*, *WAIT0* e *ACCTRESP*, permitem capturar, para cada nível DC, o número de amostras pretendidas. O valor médio obtido para as amostras capturas é salvaguardado através da execução da

instrução *SVTRESP*. Este processo é repetido para os restantes níveis (*DJNZ CNT2,L2; DJNZ CNT1,L1*).

Por fim, sinaliza-se o fim da operação de teste através das instruções *EOTEST* e *HLT*, sendo a primeira também usada para efectuar o *reset* da infra-estrutura de teste.

```
NSAMPLES     .EQU  255    ;no. of harmonics to determine
NLEVELS      .EQU  16     ;no. of harmonics to determine
       SIGMADELTA8B       ;specify 8-bit SigmaDelta as stimuli generator
       TEST_TLC0820AC     ;specify DUV to test
       MOVC  R1, $0F
       ST    (128),R1
       MOVC  R1, $1E
       ST    (129),R1
       MOVC  R1, $2D
       ST    (130),R1
       MOVC  R1, $3C
       ST    (131),R1
       MOVC  R1, $4C
       ST    (132),R1
       MOVC  R1, $5B
       ST    (133),R1
       MOVC  R1, $6A
       ST    (134),R1
       MOVC  R1, $79
       ST    (135),R1
       MOVC  R1, $87
       ST    (136),R1
       MOVC  R1, $96
       ST    (137),R1
       MOVC  R1, $A5
       ST    (138),R1
       MOVC  R1, $B4
       ST    (139),R1
       MOVC  R1, $C3
       ST    (140),R1
       MOVC  R1, $D0
       ST    (141),R1
       MOVC  R1, $DF
       ST    (142),R1
       MOVC  R1, $EF
       ST    (143),R1
       ;start test execution
       MOVC  CNT1, NLEVELS
L1:    MOVC  CNT2, NSAMPLES
       STIMULUS
L2:    SRTEST
       WAIT0
       ACCTRESP
       DJNZ  CNT2,L2            ;get next sample
       SVTRESP
       DJNZ  CNT1,L1            ;go to next level
       EOTEST
       HLT
.END
```

Figura 5.40: Programa de teste

5.2.4.3. Geração automática do processador de teste

A partir do programa de teste, ilustrado na figura 5.40, procede-se à geração automática do processador de teste para garantir as funcionalidades do algoritmo de teste adoptado, definidas pelas instruções presentes no programa de teste. Mais uma vez, a implementação do programa de teste ocorre de acordo com a descrição efectuada no capítulo 4.

A informação temporária que contém a informação relativa à geração do processador de teste é apresentada de seguida. Na figura 5.41, que contém informação relativa à implementação das operações de teste, é possível visualizar a informação relativa à etapa de descodificação e execução das instruções associadas às operações, assim como os sinais envolvidos durante estas etapas e o código de operação (opcode) relativo a cada instrução.

Os recursos a serem incluídos na arquitectura proposta para o processador, para que se garanta a funcionalidade pretendida, encontram-se definidos no ficheiro ilustrado na figura 5.42.

Verifica-se igualmente a existência de uma correspondência directa entre as instruções definidas e os recursos sobre os quais actuam. Por exemplo, a instrução *SRTEST*, definida no ficheiro ilustrado 5.41, actua sobre o recurso *duvtlc0820ac_ctrl* definido no ficheiro ilustrado na figura 5.42, através do sinal **srtest** (a negrito em ambas as figuras). O mesmo ocorre para as instruções *ACCTRESP* e *SVTRESP*, que actuam sobre o recurso *duvtlc0820ac_svtresp*, responsável pela captura das respostas de teste do conversor A/D, através dos sinais *enduvcaptresp, enduvsvtresp e* eoduvsvtresp, respectivamente.

```
//ctrlunit_testop
             // ********* SigmaDelta8B Instruction  *********
                         4'b0001: nextstate = INITIAL;    // opcode : 41(h)

             // ********* TEST_TLC0820AC Instruction  *********
                         4'b0011: begin                    // opcode : 43(h)
                                      setpath = 1;
                                      nextstate = INITIAL;
                                  end
             // ********* SRTEST Instruction  *********
                         4'b0100: begin                  // opcode : 44(h)
                                      srtest = 1;
                                      nextstate = INITIAL;
                                  end

             // ********* WAIT0 Instruction  *********
                         4'b0101: begin                    // Opcode : 45(h)
                                      if ( eoc == 1'b1 )
                                            nextstate = DECODE;
                                      else
                                            nextstate = INITIAL;
                                  end

             // ********* ACCTRESP Instruction  *********
                         4'b1011: begin                  // opcode : 4B(h)
                                      enduvcaptresp = 1;
                                      nextstate = INITIAL;
                                  end

             // ********* SVTRESP Instruction  *********
                         4'b1100: begin                  // opcode : 4C(h)
                                      enduvsvtresp = 1;
                                      if(eoduvsvtresp)
                                            nextstate = INITIAL;
                                      else
                                            nextstate = DECODE;
                                  end

             // ********* Stimulus Instruction  *********
                         4'b1000: begin                      // opcode : 48(h)
                                      enstimuli = 1;
                                      enstimuli_addr = 1;
                                      nextstate = STIMULop2;
                                  end
                                .............
//ctrlunit_op2exec_end
```

Figura 5.41: Extracto da estrutura de dados interna referente à descodificação e execução das instruções referenciadas.

```
//mc_surround_blocks
//*************************************************************************
// The following module implements a 8-bit Sigma-Delta Modulator
wire [7:0] din;
assign din = stimuli;

sdm8bit SigmaDelta( .clock(clock), .areset(reset), .din(din), .dout(Yn));

//*************************************************************************
// TLC0820AC control blocks
wire setpath, enduvtest,enduvcaptresp, enduvsvtresp, eoduvsvtresp, enINIduv,
enOUTDduv, mem_we_duv, endsaddr_duv;
assign enOUTDduv = 0;
wire [7:0] duvdata;

duvtlc0820ac_ctrl duv_new_conversion ( .srtconv(srtconv), .chipsel(chipsel),
.clk(clock), .reset(reset), .srtest(srtest) );

duvtlc0820ac_svtresp      duv_svtresp( .clock(clock), .reset(reset),
.duv_in(inport),.duv_out(duvdata), .enduvtresp(enduvcaptresp),
.enduvsvtresp(enduvsvtresp), .eoduvsvtresp(eoduvsvtresp), .enfduv(enduvtest),
.enINIduv(enINIduv), .mem_we_duv(mem_we_duv), .endsaddr_duv(endsaddr_duv));

always @( posedge clock or posedge reset )
begin
       if (reset)
              selpath = 0;
       else
       begin
              if (setpath)
                     selpath = 1;
              if (eotest)
                     selpath = 0;
       end
end
//*************************************************************************
                                  ............
//mc_surround_blocks_end
```

Figura 5.42: Extracto da estrutura de dados interna referente aos recursos de teste a inserir no processador.

Na figura 5.43 apresenta-se a arquitectura do processador, gerado de acordo com o programa de teste ilustrado na figura 5.40, onde se visualiza o núcleo (*Test Processor Core*) do processador de teste, responsável pela descodificação e execução das instruções, rodeado pelas unidades funcionais que garantem a funcionalidade pretendida.

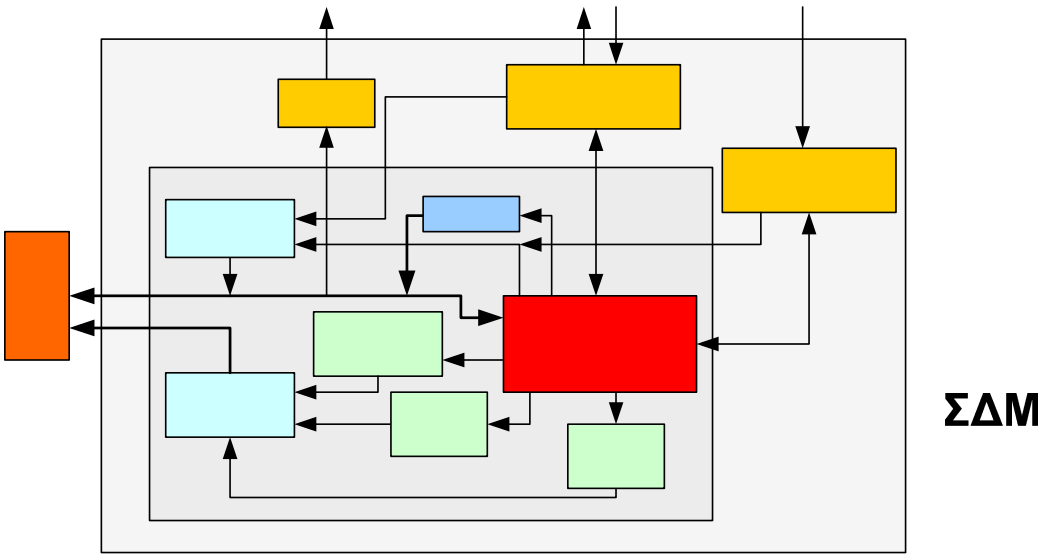


Figura 5.43: Arquitectura do processador para o teste do conversor TLC0820AC.

De um modo idêntico ao descrito na subsecção anterior 5.1.4.3, o acesso aos recursos comuns por parte das unidades funcionais efectuado à medida que são escalonados para execução.

A figura seguinte ilustra a interface do processador de teste para o teste do conversor TLC0820AC, obtido do editor de esquemático do *Foundation*.

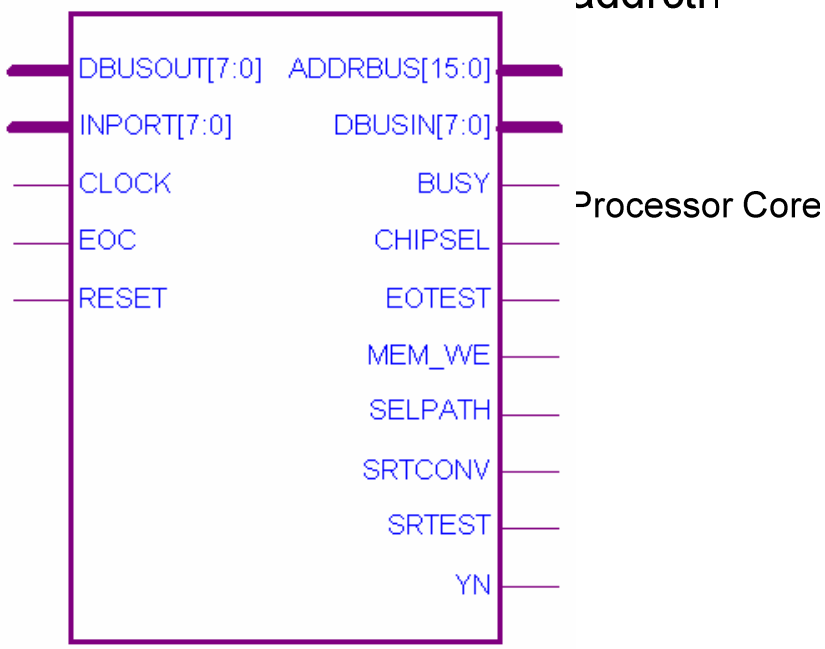


Figura 5.44: Interface do processador para o teste do conversor TLC0820AC.

A tabela 5.10 resume a contribuição espacial dos principais recursos na constituição do processador de teste.

Tabela 5. 10: Contribuição dos principais recursos que constituem o processador de teste para a área total de implementação.

Recurso	Área (Flip-Flops)
STIMULIctrl	26
TLC0820ACctrl	48
TLC0820ACacctresp	27
CTRLUNIT	20
REGS	24

5.2.5. Resultados obtidos

Na tabela 5.11 apresentam-se os resultados correspondentes ao conjunto de coordenadas (x,y), que caracteriza a função de transferência do conversor A/D. Cada par de coordenadas (x, y) corresponde ao nível DC aplicado e ao valor médio obtido para as amostras capturadas, respectivamente.

Tabela 5.11: Conjunto de coordenadas (x, y) que definem a F.T. do conversor A/D.

n	Xn (V)	Yn (V)	n	Xn (V)	Yn (V)
1	0.2933	0,4070	9	2.6535	2,6714
2	0.5879	0,6916	10	2.9542	2,9602
3	0.8828	0,9767	11	3.2352	3,2292
4	1.1778	1,2570	12	3.5363	3,5174
5	1.4765	1,5440	13	3.8376	3,8096
6	1.7755	1,8288	14	4.1156	4,0834
7	2.0749	2,1176	15	4.4151	4,3741
8	2.3744	2,4069	16	4.7138	4,6658

Os valores presentes na tabela anterior são usados para determinar o polinómio que melhor se ajusta à função de transferência do conversor A/D, sendo possível obter os harmónicos introduzidos pelo conversor, assim como parâmetros adicionais que permitam caracterizar estaticamente e dinamicamente o conversor AD, de acordo com a subsecção 5.2.1. A figura seguinte ilustra a função de transferência do conversor A/D para o conjunto de coordenadas (x, y) apresentado na tabela 5.10.

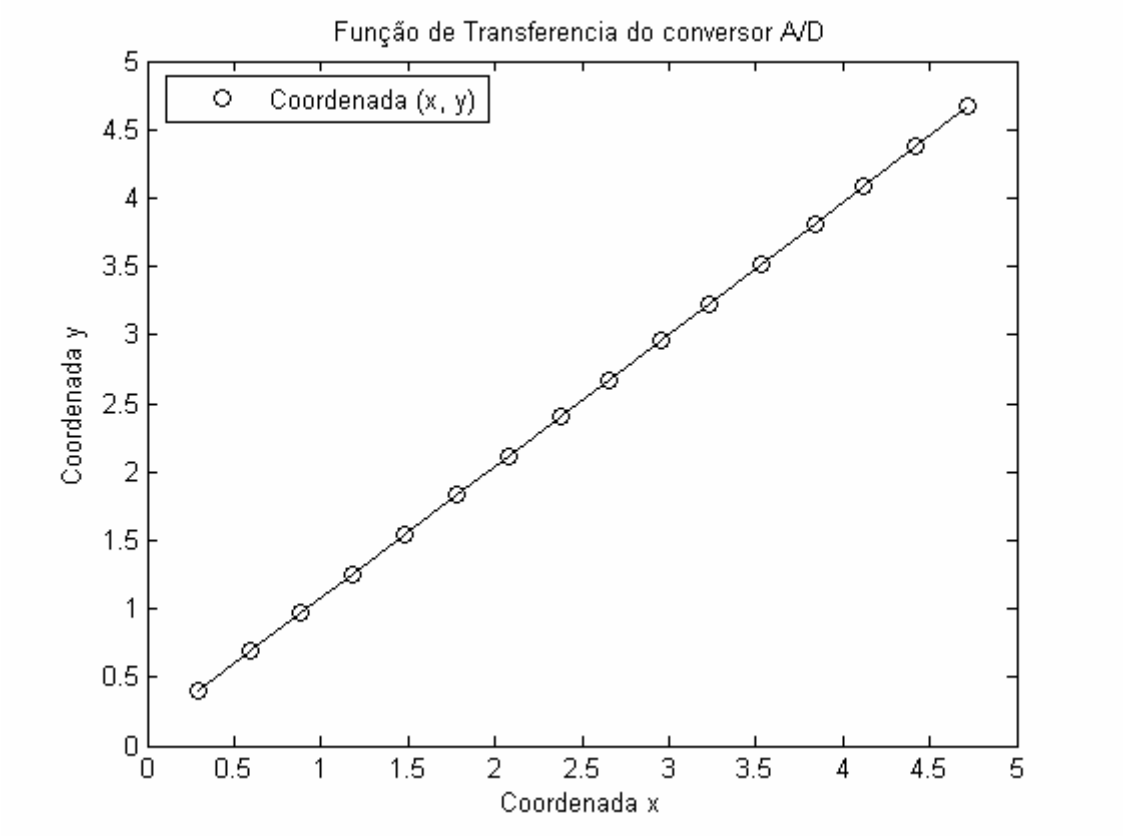


Figura 5.45: Função de transferência do conversor A/D TLC0820AC.

Recorrendo à recorrendo à função *polyfit (x,y,k)*, presente no MATLAB, é possível encontrar o polinómio que melhor se ajusta à função de transferência ilustrada na figura anterior. Na expressão seguinte apresenta-se o polinómio obtido, enquanto que na tabela 5.12 encontram-se resumidos os resultados obtidos para as amplitudes dos harmónicos até à quarta ordem.

$$p(x) = 0.9139e^{-4}x^4 + 0.2895e^{-3}x^3 - 0.4137e^{-2}x^2 + 0.9672x + 0.1241 \qquad (5.18)$$

Tabela 5.12: Valores (em dB) para os harmónicos obtidos até à quarta ordem e a distorção harmónica total correspondente.

H2	-46.6345 (dB)
H3	-54.3687 (dB)
H4	-56.3031 (dB)
THD	-45.5745 (dB)

Conforme se pode observar, os valores obtidos através do circuito implementado na FPGA, encontram-se da gama de valores esperados para um conversor de 8 bits. Este facto pode ser comprovado pela expressão 5.19 que apresenta o limite teórico da SNR do conversor A/D de 8 bits.

$$SNR_{dB} = 6.02 \times n + 1.76 = 6.02 \times 8 + 1.76 = 49.92\text{dB} \qquad (5.19)$$

Verifica-se que o valor registado para a distorção harmónica total, do conversor de 8-bits testado, se situa abaixo do limite teórico da SNR de um conversor equivalente.

5.3. Conclusão

No capítulo anterior foi apresentada a metodologia para a geração automática do processador de teste a partir do respectivo programa de teste. Neste capítulo, essa metodologia é usada para desenvolver processadores distintos, de acordo com os métodos de teste adoptados. Cada processador de teste é optimizado de modo a garantir que apenas a funcionalidade especificada pelo programa de teste é satisfeita, incluindo somente os recursos necessários à sua execução.

Na tabela seguinte apresenta-se a ocupação da FPGA para três configurações do processador de teste.

Tabela 5. 13: Ocupação da FPGA para diferentes configurações do processador.

Modelo processador	**4-input LUT**	**Flip-Flops**
Processador completo	1575	749
Processador adaptado para método de correlação	821	540
Processador adaptado para método de estimação de conteúdo harmónico.	418	257

Os valores presentes na primeira linha resumem-se ao processador que suporta o completo conjunto de instruções, incluindo os blocos periféricos, as operações sobre a ALU e a unidade de registos completa. As linhas seguintes correspondem a duas arquitecturas distintas do processador de teste. A segunda linha ilustra o processador de teste adaptado para o método de teste descrito na secção 5.1, enquanto que os valores presentes na última linha correspondem ao processador que garante as funcionalidades descritas na secção 5.2. Como se pode observar, existe uma variação significativa da ocupação da FPGA quando o processador é configurado apenas com os recursos necessários.

Deste modo, dependendo das necessidades de teste e do espaço disponível na FPGA, um processador de teste adequado pode ser adoptado de modo a que se garanta uma solução de compromisso entre o espaço ocupado pelo processador e a sua capacidade de processamento do processador. Isto é, quanto maior for o processamento efectuado pelo processador sobre as respostas obtidas do teste, menor será a quantidade de tráfego de informação de teste entre o SoC e o ATE, reduzindo o tempo de teste.

CAPÍTULO 6

Conclusão e perspectivas de trabalho futuro

Nesta dissertação propôs-se uma infra-estrutura e metodologia para o teste de núcleos analógicos e mistos (analógico-digitais) embebidos em SoC (*System on Chip*).

Os constantes avanços tecnológicos na área dos semicondutores dá origem ao aparecimento de sistemas de elevada complexidade funcional. A implementação destes sistemas é cada vez mais efectuada recorrendo à integração de blocos funcionais de propriedade intelectual provenientes de diferentes origens, o que, aliado à natureza analógica e analógico-digital desses blocos dificulta o desenvolvimento de estrutura de teste/depuração eficientes.

No capítulo 2 foram apresentados alguns conceitos relacionados com o teste em SoCs, exibindo-se de seguida algumas propostas para o teste de SoCs que contribuem para ultrapassar as lacunas existentes nas actuais estratégias de teste.

Ao longo do capítulo 3 foram discutidos os benefícios e requisitos, identificando-se as operações básicas e opcionais definidos para a infra-estrutura de teste proposta nesta dissertação, que se resume a um processador de teste residente. Este permite controlar a actividade da lógica de teste, implementando as operações de depuração adequadas. A implementação do processador de teste em dispositivos de lógica programável possibilita que seja adaptável de modo a ir ao encontro dos requisitos impostos pelo teste em causa. Deste modo, garante-se uma maior flexibilidade relativamente às soluções de teste baseadas no uso de estrutura BIST, algumas das quais foram apresentadas no capítulo 2.

Devido ao facto de o processador de teste ser implementado numa área de lógica reconfigurável, já existente no circuito, levou ao desenvolvimento de uma metodologia

para a adaptação do processador de acordo com as necessidades impostas pelo núcleo a testar. O processador de teste é gerado de forma automática, a partir do conjunto de informação de entrada, que inclui o modelo do sistema, programa de teste e o conjunto de instruções. Este conjunto de informação permite configurar o processador apenas com os recursos necessários para implementar a funcionalidade especificada no programa de teste.

Esta estratégia, de gerar automaticamente processadores programáveis, é uma maneira eficiente de criar controladores dedicados optimizados para cada tarefa particular do teste, comparativamente às soluções apresentadas no capítulo 2, o que pode ser comprovado pelos resultados obtidos no capítulo 5. Neste capítulo, procedeu-se à optimização do processador para duas metodologias de teste distintas

6.1. Contribuição do trabalho apresentado

A estratégia adoptada nesta dissertação relativamente ao recurso a núcleos de lógica reconfigurável, disponível no SoC, para a implementação de um processador de teste de dedicado ao teste de núcleos analógicos e mistos, apresenta algumas vantagens relativamente às soluções de teste apresentadas.

Quando implementado numa FPGA garante-se uma diminuição associada aos custos de teste, uma vez que os recursos necessários para a implementação da solução de teste são significativamente reduzidos. Para além disso, dependendo do espaço disponível na FPGA, uma versão adequada do processador pode ser adoptada de modo a oferecer um conveniente compromisso entre o processamento e o espaço ocupado. Em casos extremos, onde os recursos exigidos pelo processador de teste ultrapassam os recursos disponibilizados pela FPGA, é possível efectuar uma divisão do processo de teste em várias etapas. Numa primeira fase a FPGA é configurada com os recursos relacionados com a geração do estímulo de teste e captura das respectivas respostas, sendo posteriormente reconfigurada com os recursos necessários para as actividades de processamento.

Para além das vantagens referidas, existe uma metodologia para a adaptação do processador, garantindo-se uma fácil reutilização e modificação de diversas técnicas de

teste, evitando o risco de se tornar obsoleto, uma vez que em cada instante é possível adoptar pela solução de teste mais eficiente para o núcleo em causa.

Esta solução apresenta ainda a vantagem de ser aplicável a outros tipos de núcleos, permitindo o desenvolvimento de técnicas de teste uniformizadas de sistemas complexos, com eventual partilha de recursos de teste para a implementação do auto teste de diferentes núcleos.

6.2. Perspectivas de trabalho futuro

A estratégia adoptada neste trabalho para a metodologia de geração automática do processador de teste é uma das perspectivas de modificação possível, uma vez que apresenta algumas limitações. Uma destas limitações encontra-se relacionada com recursos que acedem à memória, uma vez que apenas um número limitado de recursos pode efectuar o acesso durante a execução do programa de teste. Uma possível solução para este problema poderá passar pelo desenvolvimento de uma instrução adequada, que quando associada a um determinado recurso efectua adaptação do(s) recurso(s) responsáveis por controlar o acesso aos recursos comuns.

Neste momento a ferramenta computacional associada à metodologia para a geração automática do processador de teste, responsável pela sua adaptação de acordo com a funcionalidade presente no programa de teste, exige ao utilizador final um profundo conhecimento da metodologia e da arquitectura do processador de teste. De modo a abstrair o utilizador final destes detalhes, é necessário efectuar o desenvolvimento da ferramenta a um nível de interface mais alto. Deste modo, é possível desviar a atenção do utilizador para o estudo do dispositivo a testar e da metodologia de teste adequada.

Uma forma de complementar o estudo realizado seria o desenvolvimento de um método para gerar o programa de teste a partir da arquitectura é uma das perspectivas de evolução possível.

Referências

[1] Schaller, R. R. “Moore’s Law: Past, Present and Future” IEEE Spectrum, vol. 34-6, pp.52-59, 1997.

[2] Kamath, U.; Kaundin, R.; “System-on-Chip Designs: Strategy for Success”, Wipro Technologies, 2001.

[3] SIA – Semiconductor Industry Association, “The national technology roadmap *for semiconductors*”, 1999.

[4] Mourad, S; Zorian, Y., “Principles of Testing Electronic Systems”, 420 pages, Wiley & Sons, 2000.

[5] IEEE Std 1149.1-2001, “IEEE Standard Test Access Port and Boundary-Scan Architecture”, IEEE, 2001.

[6] IEEE 1149.4 – Standard for mixed-signal test bus, Test Technology Technical Committee of the IEEE Computer Society, June 1999

[7] IEEE Std 1500-2005, “IEEE Standard Testability Method for Embedded Core-based Integrated Circuits”, IEEE, 2005.

[8] Marinissen, E.J.; Zorian, Y.; Kapur, R.; Taylor, T.; Whetsel, L.;”Towards a standard for embedded core test: An example”, IEEE International Test Conference, pp.616 – 627, 28-30 Sept. 1999.

[9] Marinissen, E.J.; Zorian, Y.; “Challenges in testing core-based system ICs”, IEEE Communications Magazine, Vol.37-6, pp.104 – 109, 1999.

[10] Zorian, Y.; Marinissen, E.J.; Dey, S.; “Testing Embedded-Core Based System Chips”, IEEE International Test Conference, pp. 130 - 143, 18-23 Oct. 1998.

[11] Rajsumam, R.; “Testing a System-On-a-Chip with Embedded Microprocessor”, IEEE International Test Conference, , pp: 499 – 508, 28-30 Sept. 1999.

[12] Galke, C.; Pflanz, M.; Vierhaus, H. T.; “A Test Processor Concept for Systems-on-a-Chip”, *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors* (ICCD’02), Germany, September 2002.

[13] Corno, F.; Sonza Reorda, M.; Squillero, G.; Violante, M.; “On the Test of Microprocessor IP Cores”, Proceedings of the IEEE Design Automation and Test in Europe, pp. 209-213, 2001.

[14] Van de Goor, A. J.; “Using March Tests to Test SDRAMs”, IEEE Design & Test of Computers, Volume 10, Issue 1, pp. 8 – 14, March 1993.

[15] Zorian, Y; Ivanov, A.; “An effective BIST scheme for ROM’s”, IEEE transactions on Computers, Vol. 41, Issue 5, pp. 646 – 653, May 1992.

[16] AbdEl-Halim, M.A.; "An Analogue Mixed-Signal Test Controller", 45th *Midwest Symposium on Circuits and Systems*, Oklahoma, Vol.1, pp. 384 – 387, Aug. 2002.

[17] Matos, José S.; Silva, José M.; "Mixed-Signal Board Level DfT Techniques Using IEEE P1149.4", Proc. 5th IEEE International Conference on Electronics, Circuits and Systems, pp. 441-446, 1998.

[18] Stroud, C.; Morton, J.; Islam, T.; Alassaly, H.; "A mixed-signal built-in self-test approach for analog circuits", Southwest Symposium on Mixed-Signal Design, pp. 196-201, 2003.

[19] "Built-In Self-Test Streamlines Testing Of Mixed-Signal SoCs", Artigo anónimo de Electronic Design, disponível em www.elecdesign.com/Articles/ArticleID/3842/3842.html

[20] Hafed, Mohamed; Roberts, Gordon W.;"Test and Evaluation of multiple embedded mixed-signal test cores"; Proc. IEEE International Test Conference, pp. 1022-1030, 2002.

[21] Hafed, Mohamed; Abaskharoun, N.; Roberts, Gordon W.;"A Stand-Alone Integrated Test Core for Time and Frequency Domain Measurements", Proc. IEEE International Test Conference, pp. 1031-1040, 2000.

[22] Huang, J.; Cheng, K.; "A delta-sigma modulation based BIST scheme for mixed-signal systems", Southwest Symposium on Mixed-Signal Design, pp. 147-152, 2000.

[23] Zeng, G.; Ito, H.; "Hybrid BIST for system-on-a-chip using an embedded FPGA core", Proc. 22nd VLSI Test Symposium, pp. 353-358, 2004.

[24] Keezer, D.C.; Zhou, Q.; "Test support processors for enhanced testability of high performance circuits", Proc. International Test Conference, pp. 801-809, 1999.

[25] Benso, A.; Di Carlo, S.; Prinetto, P.; Zorian, Y.; "A Hierarchical Infrastructure for SoC Test Management", IEEE Design & Test of Computers, Volume 20, Issue 4, pp. 32 – 39, July-Aug. 2003.

[26] Wang, S.; Chakradhar, S.T.; Kedarnath, B.,"Re-configurable Embedded Core Test Protocol", Design Automation Conference, 2004. Proceedings of the ASP-DAC 2004, Asia and South Pacific, pp. 234 – 237, 27-30 Jan. 2004.

[27] Feige, C.; Seuren, G., "Extending core test methodology to the analog/mixed-signal domain", Proceedings of the IEEE European Test Workshop (ETW 2001), Stockholm, May 2001

[28] Stollon, Neal;"On-chip instrumentation aids OCP debugging", Artigo publicado em EETimes Online, disponível em http://www.eet.com/news/design/showArticle.jhtml?articleID=171100311

[29] IEEE-ISTO 5001™-2003, "The Nexus 5001™ Forum Standard for a Global Embedded Processor Debug Interface", www.nexus5001.org

[30] IEEE 1450 - Standard Test Interface Language (STIL), http://grouper.ieee.org/groups/1450/

[31] Densmore, D.; "Reconfigurable Architectures: Overview and commercial Examples", Outubro 2003, disponível em http://network.ku.edu.tr/~stasiran/ ecoe560/Lectures/cpus/CommercialReconfig.pdf

[32] Nicklin, David; "Utilising FPGAs in re-configurable basestations and software radios", Xilinx Inc., Electronics Engineering Magazine, Page 1 DNL – April 20th Rev2, disponível em htt://cegt201.bradley.edu/projects/ proj2000/prjfpga/xilinx/basestat.pdf

[33] Korhonen, T.; Lawrence, A.; Page, I.; Urakawa, J.; "Using Reconfigurable Processors as Control System Elements", International Conference on Accelerator and Large Experimental Physics Control Systems, Nov. 3-7, 1997, Beijing

[34] Lewis, B.; Bolsens, I.; Lauwereins, R.; Wheddon, C.; Gupta, B.; Tanurhan, Y.; "Reconfigurable SoC--What Will It Look Like", 2002 Design, Automation and Test in Europe Conference and Exposition, pp. 660 - 663, 4-8 March 2002, Paris, France.

[35] "The I2C Bus Specification version 2.1", Philips Semiconductors, Janeiro 2000, disponível no site http://www.semiconductors.philips.com/markets/mms/protocols/i2c/

[36] IEEE Std. 1532-2002, IEEE Standard for In-System Configuration of Programmable Devices (Revision of IEEE Std. 1532-2001), IEEE, 2003.

[37] Mendonça, Hélio S.; Silva, J. Machado da; Matos, J. S.; "A Comparison of ADC Test Methods", Proceedings of the XV Design of Circuits and Integrated Systems Conference, pp. 102-107, Montpellier, Novembro, 2000.

[38] DYNAD – "Methods and draft standards for the DYNamic characterisation and testing of Analogue to Digital converters", disponível em http://paginas.fe.up.pt/~hsm/dynad/

[39] IEEE Std 1364-2001 "IEEE standard Verilog hardware description language", 2001

[40] IEEE Std 1076-2000 "IEEE Standard VHDL Language Reference Manual", 2000

[41] Xilinx web site, http://www.xilinx.com

[42] O'Reilly, The Source of Perl; http://www.perl.com

[43] TECMIC – Tecnologias de Microelectrónica, http://www.tecmic.pt

[44] Elsden, C.; Ley, A.; "A digital Transfer Function Analyser Based on Pulse Rates Techniques", Automatica, vol.5, pp.51, 1969.

[45] Duarte, Jorge S.; Silva, José M.; Matos, José S.; "Mixed-Signal BIST in a SoC Environment", Proceedings of the XV Design of Circuits and Integrated Systems Conference, pp. 459-463, Montpellier, Novembro, 2000.

[46] STA400EP IEEE 1149.4 Analog Test Access Device, National Semiconductor

[47] MC74HC4052AN – Analog Multiplexer/demultiplexer, On Semiconductor

[48] MC74HC4066AN – Quad Analog Switch/multiplexer/demultiplexer, On Semiconductor

[49] MC54HC4016 – Quad Analog Switch/multiplexer/demultiplexer, Motorola

[50] http://www.xilinx.com

[51] CY7C109-15VC - 128K x 8 Static RAM, Cypress

[52] CS5330A-KS, "8-Pin, Stereo A/D Converter for Digital Audio", Cirrus Logic

[53] DDS theory and simulator, disponivel em http://www.geocities.com/CapeCanaveral/5611/dds.html

[54] Essenwanger, K.A.; Reinhardt, V.S.; "Sine output DDSs A survey of the state of the art", Proceedings of the 1998 IEEE International Frequency Control Symposium, pp. 370 – 378, 27-29 May 1998

[55] Roberts, Gordon W.; Lu, Albert K.; "Analog Signal Generation for Built-In-Self-Test of Mixed-Signal Integrated Circuits", Kluwer Academic Publishers, Norwell, MA, 1995.

[57] ModelSim XE/Starter 5.6a, http://www.model.com/default.asp

[56] Barr, Michael; "Introduction to Pulse Width Modulation", September 2001, disponível em http://www.embedded.com/story/OEG20010821S0096

[57] "54622D 2+16 Channel, 100 MHz Mixed-Signal Oscilloscope", Agilent Technologies.

[58] "SCB-68 - Shielded I/O Connector Block for DAQ Devices with 68-Pin Connector", National Intruments

[59] 74AC14 - Hex Inverter with Schmitt Trigger Input, Fairchild Semiconductor

[60] Mendonça, H.; Silva, J. M.; Matos, J. S.; "Computing ADC Harmonic Content from a Reduced Number of Values", IEEE Instrumentation and Measurement Technology Conference (IMtc/2003), Vail, Colorado, USA, 20-22 May, 2003.

[61] TLC0820AC, "Advanced LinCMOS™ High-Speed 8-Bit Analog-to-Digital Converters Using Modified Flash Techniques", Texas Instruments

[62] MAX4717/18, "4.5Ω/20Ω, 300MHz Bandwidth, Dual SPDT Analog Switches in UCSP", Maxim Integrated Products.

[63] Ciletti, Michael D.; "Advanced Digital Design With Verilog HDL", Prentice Hall Xilinx Design Series, 2003.

[64] http://www.mathworks.com/products/matlab

[65] http://info.borland.com/borlandcpp/cppcomp/tasmfact.html

ANEXO A

Conjunto de Instruções

Neste Anexo descreve-se o conjunto de instruções do processador de teste, relativamente à sua operação e operandos, incluindo detalhes como o número de bytes, ciclos de execução, código de operação e *flags* afectadas. Este conjunto encontra-se dividido nas categorias seguintes, as quais possuem correspondência directa com os tipos de operações elementares necessárias:

1. Instruções aritméticas e lógicas – instruções que efectuam operações aritméticas (incremento/decremento e complemento) e lógicas (comparação, deslocamento, …) sobre os operandos de 8-bit especificados na instrução.
2. Instruções de salto – instruções que alteram condicionalmente ou incondicionalmente a sequência de execução do programa de teste.
3. Instruções de cópia e transferência de dados – instruções que efectuam a cópia e transferência de entre dois operandos da instrução, sejam estes registos ou endereços de memória.
4. Instrução de teste – instruções que controlam a execução do teste a efectuar.
5. Instruções de controlo do processador de teste – instruções que controlam as operações do controlador de teste.

As instruções aparecem descritas de acordo são descritas de acordo com a sua classificação e algumas são descritas com exemplos concretos. As seguintes abreviaturas e símbolos são usados na descrição do conjunto de instruções.

r	: Registo do processador de teste
r_s	: Registo fonte
rd	: Registo destino
( )	: Conteúdo de
d	: deslocamento de 8-bit
?	: Indeterminado
CR	: Ciclos de relógio
√	: *Flag* seleccionada

Flags

Eflag	: *Flag* igualdade
Zflag	: *Flag* zero
Ltflag	: *Flag* menor que
Gtflag	: *Flag* maior que

Instruções Aritméticas e lógicas

CPL r

Opcode	Operando	Bytes	CR	Código de operação (hex)						
				R1	**R2**	**R3**	**R4**	**R5**	**CNT1**	**CNT2**
CPL	r	2		0A 41	0A 82	0A C3	0B 04	0B 45	0B 86	0B C7

Descrição: Esta instrução converte o conteúdo do registo no respectivo complemento.

Mnemónica	Operação
CPL r	**r = ~r**

Flags: Nenhuma flag é afectada.

INC r

Opcode	Operando	Bytes	CR	Código de operação (hex)						
				R1	**R2**	**R3**	**R4**	**R5**	**CNT1**	**CNT2**
INC	r	2		02 41	02 82	02 C3	03 04	03 45	03 86	03 C7

Descrição: Esta instrução incrementa em uma unidade o conteúdo presente no registo 'r'.

Mnemónica	Operação
INC r	**r = r + 1**

Flags: Nenhuma flag é afectada.

DEC r

Opcode	Operando	Bytes	CR	Código de operação (hex)						
				R1	**R2**	**R3**	**R4**	**R5**	**CNT1**	**CNT2**
DEC	r	2		04 41	04 82	04 C3	05 04	05 45	05 86	05 C7

Descrição: Esta instrução decrementa em uma unidade o conteúdo presente no registo 'r'.

Mnemónica	Operação
DEC r	**r = r - 1**

Flags

Eflag	Zflag	Ltflag	Gtflag
	√		

CMP r, r

Opcode	Operando	Bytes	CR	Código de operação (hex)
CMP	r1, r2	2		0050 – 01F0

Descrição: Esta instrução compara o conteúdo dos operandos. De modo a reflectir o resultado da comparação, as flags (*Eflag, Gtflag* e *Ltflag*) são afectadas do seguinte modo:

- Se r1 = r2 : *Eflag* = 1, *Gtflag* = 0, *Ltflag* = 0;
- Se r1 > r2 : *Eflag* = 0, *Gtflag* = 1, *Ltflag* = 0;
- Se r1 < r2 : *Eflag* = 0, *Gtflag* = 0, *Ltflag* = 1;

Flags

Eflag	Zflag	Ltflag	Gtflag
√		√	√

SHR rd, rs1, rs2

Opcode	Operando	Bytes	CR	Código de operação (hex)
SHR	rd, r1, r2	2		0851 – 09F7

Descrição: Esta instrução desloca para direita o conteúdo presente no registo r1, sendo número de posições a deslocar especificadas no registo r2. O resultado desta operação é salvaguardado no registo rd.

Mnemónica	Operação
SHR rd, r1, r2	**rd = r1 >> r2**

Flags: Nenhuma flag é afectada.

SHL rd, rs1, rs2

Opcode	Operando	Bytes	CR	Código de operação (hex)
SHL	rd, r1, r2	2		0651 – 07F7

Descrição: Esta instrução desloca para esquerda o conteúdo presente no registo r1, sendo o número de posições a deslocar especificado no registo r2. O resultado desta operação é salvaguardado no registo rd.

Mnemónica	Operação
SHL rd, r1, r2	**rd = r1 << r2**

Flags: Nenhuma flag é afectada.

AND rd, rs1, rs2

Opcode	Operando	Bytes	CR	Código de operação (hex)
AND	rd, r1, r2	2		0C99 – 0D1D

Descrição: Esta instrução executa a operação lógica AND entre o conteúdo dos registos r1 e r2. O resultado desta operação é salvaguardado no registo rd.

Mnemónica	Operação
AND rd, r1, r2	**rd = r1 & r2**

Flags: Nenhuma flag é afectada.

OR rd, rs1, rs2

Opcode	Operando	Bytes	CR	Código de operação (hex)
OR	rd, r1, r2	2		0E99 – 0F1D

Descrição: Esta instrução executa a operação lógica OR entre o conteúdo dos registos r1 e r2. O resultado desta operação é salvaguardado no registo rd.

Mnemónica	Operação
OR rd, r1, r2	**rd = r1 | r2**

Flags: Nenhuma flag é afectada.

Instruções de salto

JMP d

Opcode	Operando	Bytes	CR	Código de operação (hex)
JMP	d	2	4	18

Descrição: Esta instrução provoca um salto para o endereço de 8-bit especificado, transferindo deste modo a execução do programa.

Flags: Nenhuma flag é afectada.

DJNZ r, d

Opcode	Operando	Bytes	CR	Código de operação (hex)	
				CNT1	**CNT2**
DJNZ	r, d	2	5	16	17

Descrição: Esta instrução provoca um salto para o endereço de 8-bit especificado, se a flag de controlo não tiver sido actuada anteriormente. Caso contrário, o próximo código de instrução será lido da primeira posição que se segue ao operando desta instrução.
O registo seleccionado é decrementado, e se r ≠0,a execução do programa é transferida para a posição de memória especificada. O registo 'r' pode ser um dos contadores internos CNT1 ou CNT2.

Flags

Eflag	Zflag	Ltflag	Gtflag
	√		

Instruções de transferência de dados

MOVC r, 8-bit

Opcode	**Operando**	**Bytes**	**CR**	**Código de operação (hex)**						
				R1	**R2**	**R3**	**R4**	**R5**	**CNT1**	**CNT2**
MOVC	r, 8-bit	2	4	71	72	73	74	75	76	77

Descrição: Esta instrução carrega o registo especificado no primeiro operando com o valor de 8-bit presente no segundo operando.

Flags: Nenhuma flag é afectada.

LD r, 8-bit

Opcode	**Operando**	**Bytes**	**CR**	**Código de operação (hex)**						
				R1	**R2**	**R3**	**R4**	**R5**	**CNT1**	**CNT2**
LD	r,(8-bit)	2	5	21	22	23	24	25	26	27

Descrição: Esta instrução transfere o conteúdo da posição de memória para o registo pretendido, sendo posição de memória é especificada por um endereço de 8-bit. O registo 'r' pode ser qualquer um dos registos de uso geral.

Flags: Nenhuma flag é afectada.

ST `(8-bit), r`

Opcode	Operando	Bytes	CR	Código de operação (hex)				
				R1	**R2**	**R3**	**R4**	**R5**
ST	(8-bit), r	2	6	29	2A	2B	2C	2D

Descrição: Esta instrução transfere o conteúdo do registo para a posição de memória especificada pelo endereço de 8-bit.

Flags: Nenhuma flag é afectada.

IN r

Opcode	Operand	Bytes	CR	Código de operação (hex)				
				R1	**R2**	**R3**	**R4**	**R5**
IN	r	2	8	31	32	33	34	35

Descrição: Esta instrução lê o valor presente no porto I/O e copia-o para o registo especificado no primeiro operando.

Flags: Nenhuma flag é afectada.

OUT r

Opcode	Operando	Bytes	CR	Código de operação (hex)				
				R1	**R2**	**R3**	**R4**	**R5**
OUT	r	2	8	39	3A	3B	3C	3D

Descrição: Esta instrução copia o conteúdo do registo 'r' para o porto I/O.

Flags: Nenhuma flag é afectada.

OUTI r

Opcode	Operando	Bytes	CR	Código de operação (hex)				
				R1	**R2**	**R3**	**R4**	**R5**
OUTI	r	2	7	69	6A	6B	6C	6D

Descrição: Esta instrução copia o conteúdo da posição de memória especificada pelo ponteiro para o porto de I/O. O conteúdo do ponteiro de memória é incrementado de uma unidade.

Flags: Nenhuma flag é afectada.

INI

Opcode	Operando	Bytes	CR	Código de operação (hex)
INI		2		60

Descrição: Esta instrução lê o valor presente no porto I/O, sendo o resultado desta leitura armazenado na posição de memória especificada pelo conteúdo do ponteiro de memória. O conteúdo do ponteiro de memória é incrementado de uma unidade.

Flags: Nenhuma flag é afectada.

Instruções de teste

Instruções para o controlo da infra-estrutura BST

NSHF 48-bit

Opcode	Operando	Bytes	CR	Código de operação (hex)
NSHF	48-bit	8		AE

Descrição: Esta instrução desloca para o TAP seleccionado, através da saída série (TDO), uma sequência de 48-bits especificada pelo primeiro operando. O número de impulsos no relógio de teste (TCK), aplicados durante a operação a deslocamento é especificado pelo operando, de 8-bit. A actualização do valor a deslocar para a cadeia ocorre com as transições descendentes no relógio de teste.
A linha TMS é mantida a 0 durante os impulsos de relógio de teste, de modo a que a o controlador TAP se conserve no estado *Shift-IR* (*Shift-DR*).

Flags: Nenhuma flag é afectada.

STATE 16-bit

Opcode	Operando	Bytes	CR	Código de operação (hex)
STATE	16-bit	4		A6

Descrição: Esta instrução permite seleccionar qualquer estado do controlador TAP seleccionado. O primeiro operando especifica a sequência de 16-bits para atingir o estado desejado. A actualização da linha TMS ocorre com as transições descendentes no relógio de teste (TCK).

O número de impulsos no relógio de teste (TCK), aplicados durante a operação de selecção é especificado pelo segundo operando.

Flags: Nenhuma flag é afectada.

Ambas as instruções o conteúdo do operando **nTCKs** é carregado no registo CNT1 (8-bit).

Instruções de controlo do processador de teste

HLT

Opcode	Operando	Bytes	CR	Código de operação (hex)
HLT		1	?	4F

Descição: Esta instrução é usada para terminar a execução de um programa. Suspende todas as operações até que um sinal de 'reset' seja recebido.

Flags: Nenhuma flag é afectada.

NOP

Opcode	Operando	Bytes	CR	Código de operação (hex)
NOP		1	4	5C

Descrição: Não realiza qualquer tipo de operação. A instrução é descodificada no entanto nenhuma operação é realizada.

Flags: Nenhuma flag é afectada.

WAIT0, WAIT1

Opcode	Operando	Bytes	CR	Código de operação (hex)
WAIT0		1	?	45
WAIT1		1	?	46

Descrição: Estas instruções retêm o processador no mesmo estado por um número indefinido de ciclos de relógio, até que a entrada de sincronismo se encontre no valor especificado.

Flags: Nenhuma flag é afectada.

Instruções de controlo de teste

EOTEST

Opcode	Operando	Bytes	CR	Código de operação (hex)
EOTEST		1	4	44

Descrição: Esta instrução é usada para especificar o fim da operação de teste, levando o sinal 'busy' ao nível baixo '0' de modo a indicar que o processador se encontra livre.

Flags: Nenhuma flag é afectada.

SRTEST

Opcode	Operando	Bytes	CR	Código de operação (hex)
SRTEST		1	4	40

Descrição: Esta instrução é usada para forçar a ADC iniciar a conversão de um estímulo de teste. Apresentando à saída a resposta ao estimulo aplicado.

Flags: Nenhuma flag é afectada.

STIMULUS

opcode	operand	Bytes	Cycles	Hex Code
STIMULUS		2		48

Descrição: Esta instrução força o módulo de geração de vectores de teste a gerar um estímulo de teste adequado ao dispositivo que se pretende testar.

Flags: Nenhuma flag é afectada.

ANEXO B

Neste anexo apresenta-se a descrição da solução proposta na subsecção 5.1.3.1 do capítulo 5, que permite interromper a execução da operação de teste caso seja detectada antecipadamente, isto é, durante a captura das respostas de teste, a ocorrência de falhas graves.

Ao recorrer a um sinal 'bem caracterizado matematicamente' para estímulo de teste é possível estimar a variação esperada entre amostras consecutivas. No caso de um sinal sinusoidal, o máximo da variação ocorre na passagem pelo valor médio onde a derivada é máxima.

Considere-se o seguinte sinal sinusoidal:

$$v(t) = A \cdot sen(\varpi t) \qquad \text{(B1)}$$

em que A é a amplitude do sinal sinusoidal e ω é a velocidade angular ($\omega = 2 \cdot \pi \cdot f_1$)

Para uma dada frequência e amplitude, a derivada máxima é dada pela seguinte equação:

$$dv/dt = \varpi \cdot A \cdot \cos(\varpi t) \qquad \text{(B2)}$$

Substituindo nesta equação a expressão relativa à velocidade angular e desprezando o termo periódico relativo ao coseno, temos a derivada máxima é:

$$dv/dt = \pm 2 \cdot \pi \cdot f_1 \cdot A \qquad \text{(B3)}$$

A uma dada taxa de amostragem, a variação de tensão máxima entre amostras consecutivas é:

$$\Delta V_{MAX} = \frac{dv}{dt} \times \Delta T = \pm 2 \cdot \pi \cdot f_I \cdot A \cdot \Delta T = \pm \frac{2 \cdot \pi \cdot f_I \cdot A}{f_S} \qquad \text{(B4)}$$

em que f_I é a frequência do sinal de entrada, A corresponde à respectiva amplitude e ΔT é a distância temporal entre amostras consecutivas, isto é, corresponde à frequência de amostragem do sinal, f_S. Estes valores são todos conhecidos à partida.

Deste modo, a variação entre amostras consecutivas, ΔV, é então especificada pela seguinte equação:

$$\Delta V = 0 \text{ ou } |\Delta V| \geq 2 \times \Delta V_{MAX} \qquad \text{(B5)}$$

Esta variação é positiva/negativa, bastando definir uma janela de por exemplo ±5 ΔV (ou seja 10 ΔV) por 5 ΔT, ou seja, em que se considera que a ocorrência de uma falta se o valor do declive de entre N consecutivas amostras for diferente de 0 ou maior que o valor especificado para a janela KΔV.

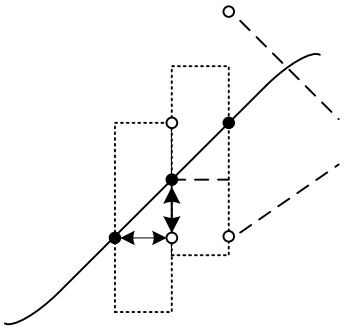


Figura B.1: Pré-selecção de amostras faltosas para uma janela ΔV.

Atendendo a esta especificação o teste é considerado inválido caso se detectem a ocorrência de um determinado número de amostras que não caiam na janela especificada.

ANEXO C

Ficheiros HDL

Neste anexo são apresentados os ficheiros HDL relativos aos recursos para a adaptação do processador, de acordo com as metodologias para o teste de núcleos analógicos e mistos (analógico-digitais) apresentados no capítulo 5. Estes ficheiros contêm caracterização individual de cada recurso que compõe o processador, especificando a correspondência entre o número / nome de pinos funcionais associados ao módulo, a instanciação do recurso no processador de teste. A especificação da instrução, definida pelo utilizador, que actua sobre o recurso faz também parte da informação contida neste ficheiro.

Em primeiro lugar são apresentados os ficheiros que contem os recursos para a aplicação da correlação ao teste de circuitos mistos, seguidamente dos que contem os recursos que possibilitam a estimação do conteúdo harmónico do conversor A/D.

C.1 Ficheiros HDL relativas à operação de correlação

```
# * * * *    Core Interfacing Signals * * * *  #
//ctrlunit_interface
                    eofduvtest, enduvtest,    //DUVctrl interface
//ctrlunit_interface_end

//ctrlunit_output_port #    Core Output Signals    #
                    output enduvtest;   //DUVctrl outport
                       reg enduvtest;   //DUVctrl outport
//ctrlunit_output_port_end

//ctrlunit_input_port #    Core Output Signals    #
                    input eofduvtest;   //DUVctrl inport
//ctrlunit_input_port_end

#     New Core Required Logic    #
//ctrlunit_input_event
             eofduvtest or //DUVctrl input event
//ctrlunit_input_event_end

# Initialization of type reg signals
//ctrlunit_output_init
                      enduvtest  = 0;   //DUV init signals
//ctrlunit_output_init_end

#    Inter-operability with the DUV instructions    #
//ctrlunit_testop
// ********* TEST_DUV Instruction  *********
             4'b0011: begin            // opcode : 41(h)
                         enduvtest = 1;
                         if ( eofduvtest )
                                 nextstate = INITIAL;
                           else
                                 nextstate = DECODE;
                      end
//ctrlunit_testop_end

# External interfacing signals
//mc_interface
                    sclk, lrck, mclk, sinput,
//mc_interface_end
//mc_output_port
                    output mclk;
//mc_output_port_end
//mc_input_port
                    input sinput, sclk, lrck;
//mc_input_port_end

# Instantiation of the block within the processor
//mc_surround_blocks
wire enladseg_duv, rstladseg_duv, enradseg_duv, rstradseg_duv, eofduvtest,
enduvtest;
wire [7:0] duvdata;
assign mclk = enduvtest ? clock : 0;
DUVctrl_cs5330A duvcrtl( .outsample(duvdata), .eofs2p(eofduvtest), .sclk(sclk)
, .mem_we(mem_we_duv), .enladseg(enladseg_duv), .enradseg(enradseg_duv),
.en_dataaddr(endsaddr_duv), .sels2p(selS2P), .mclk(clock), .lrck(lrck),
.reset(reset), .ens2p(enduvtest), .sdata(sinput), .endds(endds));
//mc_surround_blocks_end

# Interfacing with the core processor
//mc_core_interface
.eofduvtest(eofduvtest), .enduvtest(enduvtest),     //DUVctrl interface
//mc_core_interface_end
```

Figura C.1: Ficheiro HDL que contém a descrição do recurso responsável pelo controlo do conversor A/D.

```
# * * * *    Core Interfacing Signals * * * *  #
//ctrlunit_interface
                    eofxcorr, enxcorr,  //XCORR interface
//ctrlunit_interface_end

//ctrlunit_output_port #    Core Output Signals    #
                    output enxcorr;     //XCORR outport interface
                       reg enxcorr;     //XCORR outport interface
//ctrlunit_output_port_end

//ctrlunit_input_port #    Core input Signals    #
                    input eofxcorr;     //XCORR inport interface
//ctrlunit_input_port_end

//ctrlunit_input_event #     New Core Required Logic    #
                    eofxcorr or
//ctrlunit_input_event_end

//ctrlunit_output_init # Initialization of type reg signals
                    enxcorr = 0;
//ctrlunit_output_init_end

#    Inter-operability with the XCORR instructions    #
//ctrlunit_testop
// ********* XCORR Instruction  *********
            4'b1001: begin // opcode : opcode:49(h)
                        enxcorr = 1;
                        if ( eofxcorr )
                                nextstate = INITIAL;
                        else
                                nextstate = DECODE;
                     end
//ctrlunit_testop_end

# Instantiation of the block within the processor
//mc_surround_blocks
// The following module implements a Correlator used as a processing unit
wire rstdds_xc, endds_xc, eofxcorr, enxcorr, lad, mem_we_xc, enINIxc,
endsaddr_xc, enladseg_xc, enradseg_xc, rstladseg_xc, rstradseg_xc;
wire [7:0] xcdata;

muxendds mux_endds(.endds(endds),.endds_core(endds_core),.endds_xc(endds_xc),
.enfxc(enxcorr));

muxrstdds mux_rstdds(.rstdds(rstdds),.rstdds_core(rstdds_core),.enfxc(enxcorr)
.enfcore(enfcore), .rstdds_xc(rstdds_xc));

xcorr_module xcmodule(.eofxcorr(eofxcorr), .mem_we(mem_we_xc), enINI(enINIxc),
.clk(clock), .en_dataaddr(endsaddr_xc), .enladseg(enladseg_xc), .selxc(selxc),
.enradseg(enradseg_xc), .xcout(xcdata), .rstladseg(rstladseg_xc),
.rstradseg(rstradseg_xc),.enxcorr(enxcorr), .enDDS(endds_xc), .reset(reset),
.rstDDS(rstdds_xc), .enDDSoffset(enddsoffset), .xcin1(dbusout),
.xcin2(ddsout[11:0]));
//mc_surround_blocks_end

# Interfacing with the core processor
//mc_core_interface
                    .eofxcorr(eofxcorr), .enxcorr(enxcorr),
//mc_core_interface_end
```

Figura C.2: Ficheiro HDL que contém a descrição do recurso responsável pela operação de correlação.

```
# * * * *    Core Interfacing Signals * * * *  #
//ctrlunit_interface
                    rstdds,  //DDS interface
//ctrlunit_interface_end

//ctrlunit_output_port #    Core output Signals    #
                    output rstdds;
                       reg rstdds;
//ctrlunit_output_port_end
# Initialization of type reg signals
//ctrlunit_output_init
                    rstdds = 0;
//ctrlunit_output_init_end

#    Inter-operability with the DDS instructions    #
//ctrlunit_testop
// *************** SETDDS operation **************
                    4'b0001 : begin
                                 enfcore = 1;
                                 rstdds = 1;
                                 nextstate = INITIAL;
                              end
//ctrlunit_testop_end
#*********************************************************************#
# External interfacing signals
//mc_interface
                          Yn,    // Stimulus Signal
//mc_interface_end

//mc_output_port
output Yn;                // Stimulus Signal

//mc_output_port_end

# Instantiation of the block within the processor
//mc_surround_blocks
//************************************************************
//This section implements the test signal generator composed by
//one 12-bit DDS connected to an 12-bit SIgma-Delta Modulator
// 12-bit DDS
wire rstdds, endds_core, enddsoffset;
wire [11:0] ddsout;

dds_enable enabledds(.endds(endds_core), .clk(clock), .reset(reset));

dds12bit DDS( .ddsout(ddsout), .clk(clock), .reset(reset), .rstdds(rstdds),
.endds(endds), .enoffset(enddsoffset), .ddsin({4'b0,stimuli}));

//12-bit Sigma-Delta Modulator module
sdm12bit SigmaDelta( .clock(clock), .areset(reset), .din(ddsout[11:0]),
.dout(Yn));
//mc_surround_blocks_end

# Interfacing with the core processor
//mc_core_interface
                       .rstdds(rstdds_core),  // Interface between ctrlunit
//mc_core_interface_end
```

Figura C.3: Ficheiro HDL que contém a descrição dos recursos responsáveis pela geração dos estímulos de teste a aplicar ao conversor A/D.

C.2 Ficheiros HDL relativas à operação de estimação de conteúdo harmónico

```
# * * * *    Core Interfacing Signals * * * *  #
//ctrlunit_interface
          setpath, srtest, enduvcaptresp, //TEST_TLC0820AC interface
          enduvsvtresp,eoduvsvtresp,
//ctrlunit_interface_end

//ctrlunit_output_port #    Core Output Signals    #
output setpath, srtest, enduvcaptresp, enduvsvtresp; //TEST_TLC0820AC outport
   reg setpath, srtest, enduvcaptresp, enduvsvtresp; //TEST_TLC0820AC outport
//ctrlunit_output_port_end

//ctrlunit_input_port #    Core Input Signals    #
//ctrlunit_input_port_end

#     New Core Required Logic    #
//ctrlunit_input_event
             eoduvsvtresp or
//ctrlunit_input_event_end
# Initialization of type reg signals
//ctrlunit_output_init
  setpath           = 0;  //TEST_TLC0820AC init signal
  srtest            = 0;
  enduvcaptresp     = 0;
  enduvsvtresp      = 0;
//ctrlunit_output_init_end

#    Inter-operability with the DUV instructions    #
//ctrlunit_testop
       // ********* TEST_TLC0820AC Instruction  *********
                4'b0011: begin           // opcode : 43(h)
                                setpath = 1;
                                nextstate = INITIAL;
                         end
       // ********* SRTEST Instruction  *********
                4'b0100: begin           // opcode : 44(h)
                                srtest = 1;
                                nextstate = INITIAL;
                         end
       // ********* ACCTRESP Instruction  *********
                4'b1011: begin           // opcode : 4B(h)
                                enduvcaptresp = 1;
                                nextstate = INITIAL;
                        end
       // ********* SVTRESP Instruction  *********
              4'b1100: begin             // opcode : 4C(h)
                                enduvsvtresp = 1;
                                if(eoduvsvtresp)
                                        nextstate = INITIAL;
                                else
                                        nextstate = DECODE;
                       end
//ctrlunit_testop_end
                                .............
```

Figura C.4: Ficheiro HDL que contém a descrição do recurso responsável pelo controlo do conversor A/D.

```
                                  ............

# External interfacing signals
//mc_interface
               srtconv, chipsel, selpath, //TLC0820AC_duv interface
//mc_interface_end

//mc_output_port
output  srtconv, chipsel, selpath;
   reg  selpath;
//mc_output_port_end

//mc_input_port
//mc_input_port_end

# Instantiation of the block within the processor
//mc_surround_blocks
// TLC0820AC control blocks
wire setpath, enduvtest,enduvcaptresp, enduvsvtresp, eoduvsvtresp, enINIduv,
enOUTDduv, mem_we_duv, endsaddr_duv;
assign enOUTDduv = 0;
wire [7:0] duvdata;

duvtlc0820ac_ctrl duv_new_conversion ( .srtconv(srtconv), .chipsel(chipsel),
.clk(clock), .reset(reset), .srtest(srtest) );

duvtlc0820ac_svtresp      duv_svtresp( .clock(clock), .reset(reset),
.duv_in(inport),.duv_out(duvdata), .enduvtresp(enduvcaptresp),
.enduvsvtresp(enduvsvtresp), .eoduvsvtresp(eoduvsvtresp), .enfduv(enduvtest),
.enINIduv(enINIduv), .mem_we_duv(mem_we_duv), .endsaddr_duv(endsaddr_duv));

always @( posedge clock or posedge reset )
begin
       if (reset)
              selpath = 0;
       else
       begin
              if (setpath)
                     selpath = 1;
              if (eotest)
                     selpath = 0;
       end
end
//mc_surround_blocks_end

# Interfacing with the core processor
//mc_core_interface
           .setpath(setpath), .srtest(srtest), .enduvcaptresp(enduvcaptresp),
          .eoduvsvtresp(eoduvsvtresp),.enduvsvtresp(enduvsvtresp),
//mc_core_interface_end
```

Figura C.5: Ficheiro HDL que contém a descrição do recurso responsável pela operação de estimação do conteúdo harmónico.

```
//mc_surround_blocks
//*****************************************************************************
// The following module implements a 12-bit Sigma-Delta Modulator

wire [7:0] din;
assign din = stimuli;

sdm8bit SigmaDelta( .clock(clock),.areset(reset), .din(din), .dout(Yn));
//*****************************************************************************

// TLC0820AC control blocks
wire setpath, enduvtest,enduvcaptresp, enduvsvtresp, eoduvsvtresp, enINIduv,
enOUTDduv, mem_we_duv, endsaddr_duv;
assign enOUTDduv = 0;
wire [7:0] duvdata;

duvtlc0820ac_ctrl duv_new_conversion ( .srtconv(srtconv), .chipsel(chipsel),
.clk(clock), .reset(reset), .srtest(srtest) );

duvtlc0820ac_svtresp      duv_svtresp( .clock(clock), .reset(reset),
.duv_in(inport),.duv_out(duvdata), .enduvtresp(enduvcaptresp),
.enduvsvtresp(enduvsvtresp), .eoduvsvtresp(eoduvsvtresp), .enfduv(enduvtest),
.enINIduv(enINIduv), .mem_we_duv(mem_we_duv), .endsaddr_duv(endsaddr_duv));

always @( posedge clock or posedge reset )
begin
       if (reset)
              selpath = 0;
       else
       begin
              if (setpath)
                     selpath = 1;
              if (eotest)
                     selpath = 0;
       end
end
//*********************************************************
// This the following module controls the access to the Cbus
wire en_mem, en_inport;

// Bus control module
zbus2 zbus_module(
                     .from_mem(dbusout), .from_inport(inport),
                     .bus(cbus),
                     .en_mem(en_mem), .en_inport(en_inport)
                );
//*****************************************************
```

Figura C.6: Ficheiro HDL que contém a descrição dos recursos responsáveis pela geração dos estímulos de teste a aplicar ao conversor A/D.