PM

Sistema de Replicação de Circuitos Integrados Digitais Baseado em Análise Comportamental

PROJETO DE MESTRADO

Rodolfo Henrique Silva Rodrigues

MESTRADO EM ENGENHARIA DE TELECOMUNICAÇÕES
E REDES DE ENERGIA

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

dezembro | 2014

Sistema de Replicação de Circuitos Integrados Digitais Baseado em Análise Comportamental

PROJETO DE MESTRADO

Rodolfo Henrique Silva Rodrigues
MESTRADO EM ENGENHARIA DE TELECOMUNICAÇÕES
E REDES DE ENERGIA

ORIENTAÇÃO
João Dionísio Simões Barros

CO-ORIENTAÇÃO
Luis Elias Ribeiro Rodrigues

CONSTITUIÇÃO DO JÚRI

PRESIDENTE
Prof. Doutor Joaquim Amândio Rodrigues Azevedo,
Professor Auxiliar da Universidade da Madeira

ARGUENTE
Prof. Doutor Fernando Manuel Rosmaninho Morgado Ferrão Dias,
Professor Auxiliar da Universidade da Madeira

ORIENTADOR
Prof. Doutor João Dionísio Simões Barros,
Professor Auxiliar da Universidade da Madeira

Funchal, 26 de novembro de 2014

RESUMO

Nesta dissertação de mestrado é desenvolvido um sistema de replicação de circuitos integrados digitais (combinatórios e sequenciais), por observação do seu normal funcionamento.

O sistema desenvolvido carateriza-se pela capacidade de extrair e descrever na linguagem VHDL o comportamento de um circuito integrado digital em funcionamento, utilizando técnicas não invasivas e automatizadas, suportado por um vasto conjunto de algoritmos de aquisição e análise de dados.

O sistema desenvolvido assenta em dois módulos principais: um módulo de *software* que consiste numa plataforma de algoritmos de análise, controlo e gestão do sistema (alojada num computador) e um módulo de aquisição de dados (*hardware*) que consiste num circuito capaz de realizar as medições necessárias para o funcionamento do sistema, comandado pelo módulo de *software*. A comunicação entre os dois módulos é efetuada via porta série.

Os algoritmos desenvolvidos realizam uma análise da correspondência entre entradas e saídas procurando aplicar uma aproximação a um circuito combinatório se possível, caso contrário são utilizados métodos heurísticos para efetuar uma aproximação a um circuito sequencial através de uma máquina de estados. Entradas ou saídas constantes são previamente identificados e excluídos do processo de análise, para efeitos de simplificação.

Os resultados obtidos demonstram que é possível replicar o comportamento observado em circuitos digitais (combinatórios e sequenciais) desde que o número de amostras recolhidas seja adequado. Verifica-se ainda que o método desenvolvido replica a funcionalidade do circuito integrado nas condições onde o circuito está inserido.

Palavras-Chave:

Engenharia inversa, replicação de circuitos, aquisição não invasiva, correspondência entrada-saída, aproximação heurística.

ABSTRACT

In this master thesis, is developed a system to extract the behaviour of digital integrated circuits (combinational and sequential), by observing its normal operation.

The developed system is characterized by the capacity to extract and describe the behaviour of a digital integrated circuit in VHDL language, during its normal operation, using non-invasive and automated techniques, supported by algorithms for data acquisition and analysis.

The developed system is based on two main modules: a software module which contains analysis and control algorithms and also makes the management of the system (hosted in a computer) and a data acquisition module (hardware) consisting of a circuit able to perform the measurements required for system operation, commanded by the software module. Communication between two modules is made via serial port.

The developed algorithms perform a input-output correspondence analysis, seeking to apply an approach to a combinatorial circuit if possible. Otherwise, heuristic methods are used to do an approach to a sequential circuit, by description of a equivalent finite state machine. Constants inputs or outputs are previously identified and excluded from the analysis process, for simplification purposes.

The results demonstrate that it is possible to replicate the observed behavior in digital circuits (combinational and sequential) if the number of samples collected is adequate. It is also observed that the replica behaviour depends on the stimulus provided by the surronding environment.

Keywords:

Reverse engineering, circuits replication, non-invasive acquisition, input-output correspondence, heuristic approach.

AGRADECIMENTOS

Em primeiro lugar quero agradecer ao meu orientador, o Prof. Dionísio Barros, por todo o tempo despendido na orientação deste trabalho, pelo apoio prestado e ideias sugeridas que contribuíram para a conclusão com sucesso deste projeto e todo o conhecimento que me transmitiu ao longo do meu percurso académico.

Quero igualmente agradecer ao Prof. Elias Rodrigues, co-orientador desta dissertação, pelos diferentes pontos de vista que proporcionou, muito úteis em diversas situações bem como pelo apoio e disponibilidade manifestados ao longo da realização ao longo do projeto.

À minha família, em particular aos meus pais, pela oportunidade que me concederam de ter um curso superior e uma esperança de um futuro melhor.

Ao Ricardo Sousa pelas trocas de ideias construtivas partilhadas ao longo do trabalho e aos colegas com quem tive a honra de partilhar o laboratório ao longo do trabalho e que me proporcionaram momentos inesquecíveis.

Ao engenheiro Filipe Santos pelo apoio prestado no laboratório ao longo do projeto.

Um agradecimento especial à Andreia Correia que me proporcionou as energias necessárias para a realização deste trabalho.

A todos os que de alguma forma contribuíram para a conclusão com sucesso desta dissertação, o meu obrigado.

LISTA DE ABREVIATURAS

ADC – Conversor de analógico para digital (*Analog to Digital Converter*)

ALU – Unidade Lógica e Aritmética (*Arithmetic Logic Unit*)

ASIC – *Application-Specific Integrated Circuit*

AVMG – *Automatic Verilog Model Generator*

BCD – Codificação binária decimal (*Binary-Coded Decimal)*

CAD – Desennho assistido por computador (*Computer Aided Design)*

CMOS – *Complementary Metal-Oxide Semiconductor*

COM – *Component Object Model*

CPLD – C*omplex Programmable Logic Device*

DAC – Conversor de analógico para digital (*Digital to Analog Converter*)

EEPROM – *Electrically-Erasable Programmable Read-Only Memory*

EI – Engenharia Inversa

FPGA – *Field-Programmable Gate Array*

FSM – Máquina de estados finita (*Finite State Machine*)

GND – *Ground*

HDL – Linguagem de descrição de *hardware (Hardware Description Language)*

IC – *Integrated Circuit*

LED – Díodo emissor de luz (*Light Emitting Diode*)

LUT – *Look Up Table*

OCR – Reconhecimento ótico de carateres (*Optical Character Recognition)*

PCB – Placa de circuito impresso (*Printed Circuit Board*)

PLD – Dispositivo Lógico programável (*Programmable Logic Device)*

RAM – *Random Access Memory*

RTL – *Register Transfer Level*

SEM – *Scanning Eletron Microscope*

TBJ – Transístor Bipolar de Junção

TTL – *Transístor-Transistor Logic*

VCC – *Voltage Common Colector*

VHDL – *VHSIC Hardware Description Language*

VHSIC – Circuito Integrado de Muito Alta Velocidade (*Very High Speed Integrated Circuit*)

ÍNDICE

1. Introdução

1.1. Motivação

Atualmente os circuitos eletrónicos são fundamentais no suporte da tecnologia tal como a conhecemos. Os circuitos integrados estão presentes em quase todos os aparelhos que utilizamos, tendo crescido exponencialmente nas últimas décadas e com previsão de crescimento nas próximas décadas [1].

A substituição de um circuito integrado (IC – *Integrated Circuit*) é um procedimento simples quando o mesmo se encontra perfeitamente identificado ou o seu funcionamento é conhecido. No entanto, quando não existe conhecimento do IC avariado (por estratégia do fabricante, degradação do encapsulamento, …) nem do seu funcionamento a substituição do mesmo assume contornos difíceis.

A abordagem que fundamenta esta dissertação assenta no princípio que o componente que substitui o IC avariado não necessita de ser exatamente igual, mas sim um componente que realize as mesmas funcionalidades. Assim sendo, se as funcionalidades de um determinado IC forem observadas durante o seu funcionamento normal, é possível construir um componente alternativo com funcionalidades semelhantes, capaz de substituir o IC observado, no sistema em que se encontra inserido (ou num sistema igual).

A engenharia inversa de circuitos eletrónicos já é utilizada há algumas décadas, porém a maioria dessas técnicas baseiam-se em análises estruturais, manuais e são invasivas, ou seja, inutilizam o IC alvo de engenharia inversa [2]. Mais recentemente a investigação nesta área está direcionada para a automatização destes processos bem como o desenvolvimento de técnicas não invasivas [2].

As razões mencionadas motivaram ao desenvolvimento de algoritmos capazes de extrair o comportamento de circuitos integrados digitais e replicá-los (obter esquema de um circuito capaz de realizar as mesmas funções), de uma forma automatizada e não invasiva, baseado num sistema de aquisição de dados compatível.

Desta forma é pretendido contribuir para o conhecimento nesta área e aplicar o conteúdo desta dissertação a situações reais, tais como:

- Recuperação de circuitos: quando um circuito integrado se encontra danificado e não existe informação disponível sobre o mesmo ou o circuito onde se encontra inserido, a recuperação do circuito pode ser conseguida através da extração do comportamento de um circuito integrado igual em funcionamento num meio idêntico;
- Deteção de anomalias: quando um circuito de funcionamento bem conhecido se encontra a operar incorretamente a aplicação de engenharia inversa comportamental aos componentes desse circuito

pode permitir encontrar o componente que se encontra em funcionamento deficiente;

- Atualização de tecnologia: quando um circuito é submetido ao processo de engenharia inversa comportamental, a tecnologia de construção da réplica pretende-se que seja recente. Assim sendo, o sistema de replicação de circuitos integrados pode ser utilizado para atualizar a tecnologia de circuitos mais antigos.

Em suma, as caraterísticas e possíveis aplicações de um sistema de replicação de circuitos integrados acima enunciadas motivaram a realização desta dissertação.

1.2. Objetivos

A aplicação de engenharia inversa ao nível estrutural, por vezes, já não corresponde ao paradigma atual onde se pretende a redução da componente manual (automatização) e a migração para novas tecnologias, através das linguagens de descrição de *hardware*. Assim sendo, a evolução para uma análise comportamental automatizada e replicação numa linguagem de descrição de *hardware* permite a realização de manutenção de um circuito eletrónico sem conhecimento do circuito ou do sistema onde se encontra inserido, bem como inferir a contribuição de um IC num sistema para análise e possível melhoramento.

Neste âmbito, os objetivos a concretizar nesta dissertação são:

1) Revisão das técnicas de engenharia inversa atualmente existentes para circuitos eletrónicos;
2) Desenvolvimento de algoritmos capazes de, a partir de um conjunto de medições, extrair o comportamento do circuito e descrever esse comportamento em linguagem VHDL, numa estrutura sintetizável;
3) Projeto de um sistema não invasivo de aquisição de dados compatível com os algoritmos desenvolvidos;
4) Criação de um protótipo do sistema completo capaz de replicar circuitos integrados digitais de uma forma maioritariamente automatizada;
5) Análise da capacidade de replicação obtida, bem como identificação das limitações e possibilidades de expansão do sistema.

Os algoritmos desenvolvidos e o sistema de aquisição de dados projetado devem ter em conta as técnicas de defesa contra a engenharia inversa atualmente utilizadas (ver Anexo A) e operar de forma a possuir o maior grau de imunidade possível às referidas técnicas.

1.3. Organização e conteúdos

Esta dissertação encontra-se organizada em seis capítulos: Introdução, Estado de arte, Arquitetura do sistema e algoritmos de replicação de circuitos digitais, Sistema de aquisição de dados, Resultados e Conclusões. No final da dissertação encontram-se as Referências e Anexos.

No Capítulo 1, Introdução, foram apresentadas as motivações para realização deste projeto e objetivos a atingir. Neste capítulo é também apresentada a estruturação da tese.

No Capítulo 2, Estado de arte, é realizada uma abordagem às aplicações da engenharia inversa de diferentes tipos de circuitos e por diferentes métodos. Na parte final do capítulo é efetuada uma pequena revisão sobre a sintetização de *hardware*.

No Capítulo 3, Arquitetura do sistema e algoritmos de replicação de circuitos digitais, é realizada uma descrição da arquitetura geral do sistema, bem como detalhada a implementação dos algoritmos desenvolvidos para replicação do comportamento de circuitos digitais por observação do funcionamento.

No Capítulo 4, Sistema de aquisição de dados, é descrito todo o sistema de aquisição de dados que irão suportar e fornecer as medições necessárias aos algoritmos de replicação, sendo detalhadas as opções de encaminhamento dos sinais a medir bem como o respetivo isolamento.

No Capítulo 5, Resultados, é avaliado o comportamento do sistema perante vários circuitos de teste e realizado um estudo estatístico do desempenho do sistema.

No Capítulo 6, Conclusões, são inferidas as conclusões mais relevantes desta dissertação e sugeridas algumas ideias para realização de trabalhos futuros, que consistem essencialmente na otimização e expansão do sistema desenvolvido.

1.4. Contribuições originais

As contribuições originais desta dissertação de mestrado sobre o desenvolvimento de um sistema de replicação de circuitos integrados digitais baseado em análise comportamental são:

1. Análise comportamental e obtenção de uma réplica sem comparação com uma biblioteca de componentes ou qualquer réplica anteriormente analisada.
2. Desenvolvimento de um sistema de aquisição de dados para medição das tensões nos pinos de circuitos integrados durante o seu funcionamento, com isolamento elétrico entre a unidade de aquisição e a unidade de controlo.

2. Estado da arte

2.1. Introdução

O funcionamento de um circuito pode ser facilmente compreendido se existir documentação sobre o mesmo. Porém quando não existe qualquer documentação ou essa documentação não está disponibilizada ao público, há um ramo de engenharia que estuda a recuperação ou duplicação de circuitos: a engenharia inversa [3]. Neste capítulo serão descritas técnicas utilizadas para realização de engenharia inversa.

O conhecimento sobre o funcionamento dos circuitos digitais (combinatórios e sequenciais) é necessário para compreensão de algumas técnicas descritas nas subseções seguintes existindo vasta documentação nessa área, como são exemplos [4], [1], [5] e [6].

A engenharia inversa (EI) consiste em extrair o conhecimento ou esquema de qualquer coisa feita pelo Homem. A EI começou por ser uma atividade amadora que surgiu com a eletrónica moderna, quando as pessoas impressionadas com os aparelhos (rádio, televisão,…) os desmontavam para compreender o seu interior [7].

A EI é geralmente conduzida de forma a obter conhecimento em falta, ideias e filosofias de *design*, quando tal informação não está disponível, ou quando a informação é propriedade de alguém que não está disposto a compartilhá-la ou quando simplesmente a informação foi perdida ou destruída [7].

2.2. Aplicações da engenharia inversa

Especialistas na área da EI defendem que enquanto a Lei de Moore [5] provocar a diminuição do tamanho dos transístores abaixo dos 30 nm os desafios enfrentados pela EI serão cada vez maiores e as técnicas cada vez mais aperfeiçoadas, para que a EI possa continuar a ser aplicada nas áreas descritas nas subseções seguintes [8].

2.2.1. Fins militares e de segurança

As unidades militares de um país são um ponto sensível onde fugas de informação podem comprometer seriamente a segurança dos militares e inclusive da nação. Por isso, alguns países como os Estados Unidos da América ou com capacidade militar semelhante, utilizam a EI para verificar o conteúdo dos circuitos eletrónicos presentes em todas as viaturas militares e aparelhos de comunicação, de modo a detetar eventuais escutas ou dispositivos dissimulados capazes de armazenar e/ou transmitir informação para o inimigo [9].

A agência americana DARPA (*Defense Advanced Research Projects Agency*) tem vindo a estudar técnicas de engenharia inversa baseada em algoritmos, através do programa IRIS (*Integrity and Reliability of Integrated Circuits*) [9].

2.2.2. Recuperação de circuitos obsoletos

A perda ou ausência de informação sobre um determinado circuito é um problema que pode ser solucionado pela engenharia inversa, principalmente quando [3]:

- O fabricante original já não existe ou já não fabrica o produto por ser obsoleto, mas ainda assim o circuito é fundamental num equipamento de longa vida ou de elevado custo;
- A documentação original nunca foi divulgada pelo fabricante ou foi perdida;
- Extração de dados do circuito quando não existe nenhum desenho assistido por computador do circuito.

Este tipo de recuperação de circuitos para substituição é mais comum em aviões, navios e submarinos que se encontram ao serviço há anos suficiente para o *hardware* se tornar obsoleto, todavia abaixo do tempo de vida útil do aparelho [3], [10].

2.2.3. Controlo de qualidade e/ou inspeção

A EI pode ser utilizada para verificar se o produto está de acordo com a descrição dada pelo fabricante através da comparação com o esquema ou por comparação com um circuito *standard*. Esta aplicação da EI pode ser realizada pelo próprio fabricante no sentido de garantir a qualidade do seu produto ou por uma entidade fiscalizadora no sentido de defender os consumidores [3].

2.2.4. Competição inteligente

A EI é uma técnica que pode também ser utilizada para descobrir segredos e técnicas de concorrentes e utilizar os dados extraídos para analisar as ideias contidas no circuito concorrente e realizar melhorias sem ter de percorrer todo o caminho de conceção realizado pelos competidores [10].

As técnicas de EI podem permitir identificar pontos fracos do circuito, como por exemplo pontos de maior desgaste, sendo uma ferramenta ótima de otimização de circuitos [3].

A competição inteligente pode assumir contornos de pirataria comercial, quando violadas as leis de propriedade intelectual [10] (ver Anexo B).

2.3. Técnicas de engenharia inversa

A EI é um ramo da engenharia praticado já há muito tempo, ainda assim é um processo sobretudo manual, sendo que o desafio para o futuro é a automatização da EI [11].

Para efeitos de análise vamos agrupar as diversas técnicas de EI para *hardware* em cinco métodos:

- Desassemblagem do produto;
- Análise manual do sistema;
- Engenharia inversa de circuitos integrados;
- Engenharia inversa de placas de circuito impresso;
- Engenharia inversa baseada em algoritmos para extração do comportamento.

As quatro primeiras técnicas enunciadas podem considerar-se técnicas manuais (ou com uma forte componente manual) orientadas para a compreensão do *hardware*, ao passo que as técnicas baseadas em algoritmos tendem a reduzir a componente manual e aumentar a automatização (não implica que algumas técnicas baseadas em algoritmos não necessitem de etapas manuais), sendo principalmente orientadas para o comportamento do circuito.

Nas subsecções seguintes cada um dos métodos enunciados será descrito com maior detalhe.

2.3.1. Engenharia inversa por desassemblagem do produto

Esta técnica é uma das técnicas mais simples na área da EI de *hardware*. Consiste meramente na desmontagem do produto, onde os componentes, placas e subconjuntos são fotografados e posteriormente inventariados [10], [12].

Esta técnica é útil quando apenas é pretendido conhecer os componentes que se encontram no dispositivo/circuito e compor uma lista de materiais necessários para o fabrico do mesmo, permitindo ainda a elaboração de uma estimativa do custo de produção do circuito em função dos componentes [10], [12].

2.3.2. Engenharia inversa por análise manual do sistema

Existe uma grande diversidade de circuitos eletrónicos (podem consistir em *hardware*, *software*, *firmware*, comunicação, transdutores, etc) e, consequentemente, vários métodos de análise [10], [12].

A EI por análise do sistema pode ser considerada um melhoramento da técnica anterior, pois numa fase inicial segue o mesmo procedimento. No entanto esta técnica é mais completa pois contempla ainda a análise de operações, caminhos de sinal, interconexões bem como examina a estrutura e a constituição dos materiais. Existem três formas de análise: por obtenção do esquema elétrico, análise funcional e/ou análise de software [10], [12].

2.3.2.1. Obtenção do esquema elétrico

A primeira fase desta técnica assemelha-se à técnica descrita na secção 2.2.2.1 (engenharia inversa por desassemblagem do produto), onde todos os componentes são identificados. Posteriormente o circuito é fotografado e são anotadas todas as ligações entre componentes/subsistemas (em casos de sistemas com várias camadas, todas as camadas são separadas e as conexões entre camadas registadas). Por fim todos os componentes são catalogados e seletivamente removidos [10], [12].

Por fim é reconstituído o esquema elétrico (manualmente ou com recurso a *software* de desenho/simulação de circuitos) baseado nos componentes e ligações entre componentes. Em alternativa, ou para melhor precisão, podem ser utilizadas pontas de prova de um medidor de grandezas elétricas para encontrar conexões entre módulos [10], [12].

Em suma, uma vez obtido o esquema elétrico é possível inferir o funcionamento do mesmo e efetuar uma cópia do circuito em análise.

2.3.2.2. Análise funcional

A análise funcional consiste em monitorizar o sistema durante o seu funcionamento. Esta técnica consiste em colocar pontas de prova em pontos estratégicos do circuito e analisar o sistema com base na visualização e análise dos sinais medidos [10], [12].

Para auxiliar a análise podem ser aplicados estímulos ao circuito de forma a observar o circuito em diferentes modos de funcionamento (quando o circuito já é parcialmente conhecido). A utilização de geradores de sinais, osciloscópios e analisadores lógicos são muito usuais neste tipo de técnica [10], [12].

Esta técnica pode ser aplicada com complemento à técnica de desassemblagem do produto, principalmente em sistemas multicamada, sendo nesse caso realizada a separação de camadas mantendo (ou reconstruindo) as ligações elétricas entre camadas e utilizando posteriormente pontas de prova para medir e analisar o sistema [10], [12].

A análise funcional descrita anteriormente baseia-se na análise manual por visualização dos sinais em pontos estratégicos do circuito, porém este tipo de análise pode ser muito mais vasta e efetuar uma análise comportamental do sistema, muito útil na engenharia inversa baseada em algoritmos. Outras formas de análise funcional serão descritas em seções seguintes.

2.3.2.3. Engenharia inversa de software[1]

Tal como o *hardware* também o *software* pode ser revertido, isto é, em dispositivos programáveis existem processos para converter o código de máquina numa forma estruturada e legível, de forma a compreender o comportamento do dispositivo [10], [12].

Uma tarefa típica consiste na extração código incorporado numa memória. Existem várias técnicas praticadas, como por exemplo, a monitorização dos sinais utilizados para programar o dispositivo durante o *upload* do código. Existem mecanismos de segurança tais como encriptação dos dados, segurança baseada em *hardware* ou bloqueio da leitura do *chip*, que necessitam de ser considerados. Em casos de código encriptado é necessária uma análise da encriptação e desencriptação da informação [10], [12].

Em alternativa, quando fisicamente possível, os dispositivos podem ser reprogramados de forma a realizarem as mesmas tarefas que o dispositivo original sendo uma tarefa muito especializada, realizada maioritariamente por programadores de memórias EEPROM[2] [10], [12].

A engenharia inversa de *software* é uma vasta área da engenharia inversa, mas com técnicas e processos distintos da engenharia inversa de *hardware*. A engenharia inversa de *software* e *hardware* podem ser complementares (principalmente em circuitos eletrónicos com dispositivos programáveis embebidos), porém a recuperação de código fonte não é o objetivo principal do caso em estudo. Técnicas e processos acerca de engenharia inversa de *software* estão descritos em [13],[14], [15],[16] e [17].

2.3.3. Engenharia inversa de circuitos integrados (ICs)

A engenharia inversa de um circuito integrado é por vezes denominada de técnica de extração do circuito e inclui um conjunto de técnicas usualmente aplicadas

[1] No ramo da engenharia inversa a investigação e literatura sobre engenharia inversa de *sotftware* (obtenção de código de alto nível a partir de código de baixo nível) são muito mais abundantes que no caso da engenharia inversa de *hardware* (ou engenharia inversa de eletrónica) [18].

[2] EEPROM (***E****lectrically-****E****rasable* ***P****rogrammable* ***R****ead-****O****nly* ***M****emory*) é uma memória de armazenamento não volátil que permite que a memória seja eletricamente reprogramada ou ainda atualização de palavra(s) de dados sem necessidade de reprogramação total [19].

para recuperar o esquema elétrico/lógico do IC. As técnicas habituais de extração do esquema elétrico/lógico de um IC seguem várias etapas: desencapsulamento do IC, separação das camadas (*layers*), recolha de imagens, deteção/anotação dos componentes e interconexões, obtenção do esquema elétrico/lógico e por fim a verificação e análise do circuito obtido [10], [12], [20].

As metodologias mais comuns aplicadas na engenharia inversa de ICs geralmente baseiam-se em processos altamente invasivos que tornam o IC analisado em estado não funcional (muitas vezes são necessários vários ICs iguais para se obter o esquema elétrico/lógico). Atualmente, investigadores procuram desenvolver técnicas não invasivas de efetuar engenharia inversa em ICs [2].

2.3.3.1. Desencapsulamento e separação das camadas

A primeira etapa do processo de extração do circuito de um IC é o seu desencapsulamento, isto é, a remoção da camada exterior do IC para um acesso mais fácil as camadas onde estão os semicondutores (camadas de interesse). A forma mais comum de remoção da camada exterior é a imersão do IC numa solução ácida que dissolve o invólucro mas não danifica o material semicondutor ou dielétrico. Circuitos com encapsulamento cerâmico ou hermético podem necessitar de técnicas diferentes, tais como tratamento mecânico e/ou térmico ou até mesmo o polimento da camada de cerâmica [10], [12].

A separação das camadas do circuito (camadas na ordem dos µm) é feita na maioria dos casos com recurso a um dos seguintes métodos: decapagem a seco (corrosão), decapagem húmida e polimento. Em ICs desconhecidos ou pouco habituais um corte transversal pode ser útil numa primeira análise, pois permitirá obter informação sobre a composição e espessura das camadas [10], [12]. As metodologias mecânicas (polimento, por exemplo) apresentam o inconveniente de suscitar irregularidades nas camadas, devido a zonas mais polidas que outras. Esta não uniformidade, entre outros fatores, podem acentuar erros na etapa de recolha de imagens [20].

2.3.3.2. Recolha de imagens

O processo de recolha de imagens tem por objetivo a sua ampliação para que possa ser analisada (manualmente ou através de *software* especializado) e recuperado o circuito do IC, tendo sido os sensores óticos os primeiros a ser utilizados para captura de imagens. A captura das imagens pode ser efetuada de forma manual ou automática, embora para os atuais ICs o método manual já não seja indicado [10], [12].

Em paralelo com a evolução dos ICs também o tamanho dos componentes se tornou significativamente menor que o comprimento de onda típico da luz utilizada pelos

sensores óticos [21]. Para contornar este problema podem ser utilizadas técnicas de melhoramento da resolução [21] ou utilizados microscópios [10], [12]. Para semicondutores de dimensões inferiores a 0,18 µm os sensores óticos já não possuem resolução suficiente e é necessário utilizar um microscópio eletrónico de varrimento (SEM – *Scanning Eletron Microscopes*) [3] ou outros tipos de microscópios de alta resolução [10], [12] e [22].

2.3.3.3. Deteção/anotação dos componentes e interconexões e obtenção do esquema elétrico

Com as imagens recolhidas e alinhadas a etapa seguinte é identificar todos os componentes (transístores, condensadores, díodos, portas lógicas, etc) e todas as ligações entre componentes e entre camadas [10], [12]. A tarefa de identificar os componentes pode ser feita manualmente por engenheiro especializado, porém em circuitos com dezenas de milhar de componentes considerando a carga de trabalho a que seria sujeito o engenheiro a automatização é quase uma obrigação, com impacto muito significativo na redução do tempo de análise [20].

Para realizar esta etapa automaticamente as empresas especializadas na área da EI possuem *software* personalizado para efetuar estes procedimentos e ainda exportar o esquema elétrico/lógico, como é exemplo o *Chipworks ICWorks Tool Suite* [23].

Projetos recentes conduziram a novos algoritmos para localizar e reconhecer portas lógicas [20]. A localização pode ser realizada através de um algoritmo que se baseia no gradiente (variação) do nível energia da imagem, para encontrar regiões que provavelmente contêm portas lógicas, de acordo com a Figura 2.1. Como o *background* de cada camada é homogéneo, a variação do nível de energia da imagem é baixo, logo as zonas sem componentes são interpretadas pelo algoritmo como zonas sem interesse [20].

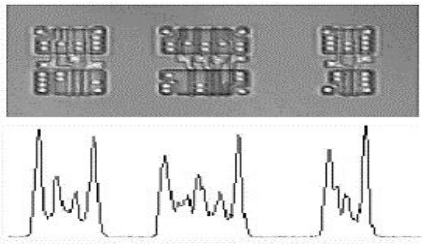

Figura 2.1 – Região com portas lógicas e sua energia [20].

Após a localização a etapa seguinte é efetuar o reconhecimento de padrões que correspondam a portas lógicas nas regiões previamente definidas na etapa de localização. Quando encontrados os padrões é comparada a sua semelhança com o

[3] Os microscópios SEM utilizam um feixe focalizado de eletrões de alta energia para gerar uma variedade de sinais na superfície das amostras sólidas, que permitem determinar a morfologia externa, composição química/estrutura cristalina e orientação dos materiais que a compõem [22].

padrão típico recorrendo à correlação de Pearson[4] [24]. Alguns erros nesta fase podem advir da irregularidade da superfície provocada na separação de camadas [20].

Neste último método descrito a obtenção do esquema elétrico/lógico é obtida em dois passos: navegação e definição de nós [20]. A navegação consiste na utilização de um algoritmo de alinhamento de camadas adjacentes para formar um módulo tridimensional. Devido ao ampliamento, as dimensões das imagens aumentam e o custo computacional deste algoritmo é elevado. A função de alinhamento utiliza pontos âncora definidos pelo utilizador para facilitar procedimentos de deslocação das camadas durante o processo de alinhamento [20]. A definição de nós é realizada para definir as ligações entre componentes do IC e baseiam-se na identificação de quatro tipos de nós diferentes [20]:

- *Beginning*: ponto utilizado para uma saída de uma porta lógica;
- *Corner*: utilizados para unir duas pistas condutoras;
- *Divergent*: utilizados para unir mais de duas pistas condutoras;
- *Terminal:* onde o nó termina, usualmente numa entrada de uma porta lógica.

Uma vez encontradas todas as conexões, o esquema elétrico/lógico é gerado automaticamente num formato profissional, como por exemplo *Verilog* [20].

2.3.3.4. Verificação e análise do circuito obtido

A etapa final do processo de recuperação do circuito interno de um IC é a verificação e análise do circuito obtido. A verificação consiste em detetar curto-circuitos, portas flutuantes, circuitos sem entradas ou saídas, entre outras violações das regras dos circuitos elétricos. A organização dos esquema elétrico/lógico pode ser importante para que o circuito fique compreensível, pois mesmo para engenheiros com experiência na área pode ser difícil compreender um esquema elétrico/lógico com os componentes distribuídos de forma pouco usual [10], [12].

2.3.3.5. Técnica não invasiva de reconhecimento de elementos de um IC

Como já referido anteriormente neste documento as técnicas tradicionais de EI são altamente invasivas e destrutivas. Recentemente foi desenvolvida uma técnica não invasiva para detetar os componentes de um IC, baseada numa varredura com um microscópio de varrimento laser (resolução limite inferior a 0,065 μm) do IC e numa aproximação apoiada num algoritmo de aprendizagem, de acordo com a Figura 2.2 [2].

[4] O coeficiente de correlação de Pearson varia no intervalo $[-1,1]$ e é uma medida de dependência linear entre duas variáveis aleatórias de valores reais. Historicamente é a primeira medida de correlação e ainda é uma das mais utilizadas [24].

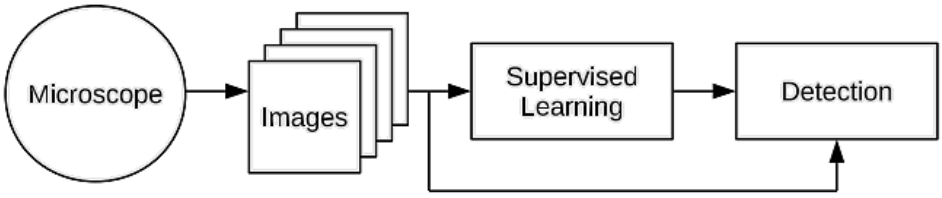

Figura 2.2 – Diagrama geral do funcionamento de um método não invasivo [2].

Apesar de inovador, este método ainda possui uma limitação técnica importante: é necessário um conhecimento prévio do IC, nomeadamente do *layout*, para que a navegação e "interrogação" do IC tenha sucesso. Assim sendo, nos casos onde essa informação não está disponível, é necessária a utilização das técnicas tradicionais da EI (técnicas invasivas) [2].

O reconhecimento de componentes pode ser realizado em dois modos: janela deslizante ou deteção de componentes. No caso da janela deslizante o reconhecimento envolve a digitalização densa do IC e localização dos pontos de interesse, ao passo que no modo de deteção de componentes é realizada uma primeira varredura para deteção de pontos de interesse e posteriormente realizada uma digitalização densa das áreas de interesse, poupando assim recursos de memória [2].

A aprendizagem consiste num detetor treinado de exemplos positivos (aqueles que geralmente correspondem a um ponto de interesse) e exemplos negativos (aqueles que geralmente não correspondem a um ponto de interesse). O detetor ao longo da sua operação pode "memorizar" mais exemplos positivos e negativos, aumentando assim o seu "conhecimento" [2].

Este método ainda apresenta as suas limitações, todavia é um passo importante no estudo de métodos não invasivos na engenharia inversa de IC, que permitirá que o IC analisado se mantenha funcional após sujeito a análise [2].

2.3.4. Engenharia inversa de placas de circuito impresso

A engenharia inversa de placas de circuito impresso (PCB – *Printed Circuit Board*) é uma atividade já praticada há algumas décadas, sendo um tipo de circuito onde as técnicas manuais são mais simples de executar. No entanto, do ponto de vista da engenharia, o objetivo é automatizar os processos e já existem processos automáticos (ou semi-automáticos) associados a programas CAD (*Computer Aided Design*) para obtenção do circuito de uma PCB. Os métodos mais comuns são para PCB com cobre em uma ou duas camadas e são compostos pelas seguintes etapas (Figura 2.3) [25], [26]:

- Aquisição de dados e digitalização da imagem da PCB;
- Processamento da imagem e vetorização para utilização em programas CAD;

- Deteção de ligações e componentes;
- Geração do esquema elétrico;
- União dos dados das camadas (caso a PCB tenha camada de cobre em ambos os lados)
- Fabricação, teste e análise ao circuito.

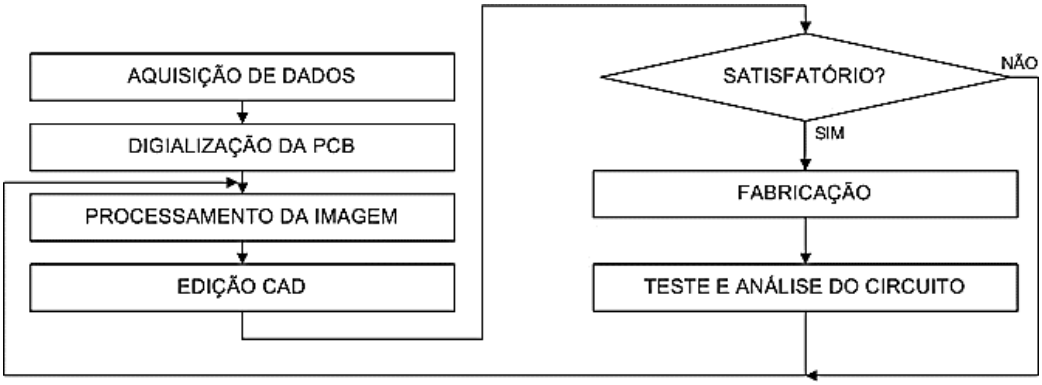

Figura 2.3 – Etapas para extração do circuito de uma camada de uma PCB através de técnicas automáticas de EI [26].

Existem ainda métodos para obter o circuito de uma PCB e converter para *Verilog* HDL (*Hardware Description Language*) baseado num gerador automático de um modelo de *Verilog* HDL (AVMG – *Automatic Verilog HDL Model Generator*) [27].

2.3.4.1. Aquisição de dados e digitalização de imagem da PCB

A primeira fase da recuperação de um circuito de uma PCB consiste na preparação e varrimento com sensor ótico da PCB para obtenção de uma imagem de cada camada no formato digital.

A preparação consiste em colocar a placa pronta a ser digitalizada, existindo duas variantes do mesmo método: digitalização da PCB com todos os componentes [25] ou remoção dos componentes e limpeza da solda após primeira digitalização detalhada para localização e recolha de informação sobre os componentes [26]. A última etapa da preparação consiste em colocar a PCB sobre um fundo preto, de forma a evitar reflexões de luz [26] e permitir distinguir com facilidade os furos da PCB, através da diferença de cor entre o fundo preto e o cobre [25].

A digitalização da PCB (com ou sem componentes) é efetuada geralmente com sensores óticos (não é necessário resolução tão alta como no caso dos circuitos integrados) embora as imagens recolhidas sejam de alta resolução (por exemplo em [26] é referida a utilização de um *scanner* com resolução de 3200x6400 DPI e profundidade de cor de 48 *bits*, permitindo 281 triliões de cores possíveis) [25], [26].

2.3.4.2. Processamento de imagem e edição em programas CAD

Tal como no caso dos circuitos integrados, a etapa de processamento de imagem é fundamental na extração do circuito. No caso da EI para recuperação de circuitos de PCB existem três passos importantes: conversão da imagem da cor real para escala de cinzentos, filtro de ruído e deteção de bordas [26].

A conversão[5] para uma escala de cinzentos consiste, como o nome indica, na conversão de cada *pixel* colorido para uma tonalidade de cinzento, reduzindo de um conjunto de cores para um valor de intensidade numa escala única (escala de cinzentos) [25], [26].

A filtragem de ruído é um processo preliminar em qualquer aplicação de processamento de imagem e visa corrigir alguns defeitos ocorridos na recolha de imagem, usualmente devido a perda de dados durante a transmissão, reflexão imprópria ou absorção da luz em superfícies irregulares ou onduladas [26].

Após a filtragem a etapa seguinte é a deteção de bordas (tradução da expressão inglesa *edge detection*) que consiste na deteção de variações bruscas na intensidade da imagem e é essencial na deteção de pistas e buracos [25]. O *software MatLab* [28] é uma ferramenta importante e poderosa para a deteção de bordas[6] [25].

A vetorização das imagens processadas é uma transformação importante para efetuar posteriormente o reconhecimento de pistas e/ou componentes. A vetorização das imagens é fundamental ainda para futuras análises ou compatibilidade com programas CAD [26].

2.3.4.3. Deteção de ligações e componentes

Esta etapa é apenas significativa no método onde os componentes não são removidos.

As ligações são detetadas de forma quase instantânea, isto é, correspondem às zonas onde existe cobre. Os componentes são encontrados por comparação de um componente com uma biblioteca de componentes padrão (por vezes existem algoritmos de rotação para permitir maior eficácia na comparação). É igualmente importante que as duas imagens do componente a ser comparadas tenham a mesma luminosidade, situação que implica que às vezes sejam adicionados pixéis pretos a uma das imagens [25].

[5] Na conversão é comum ser utilizado o algoritmo *K-means* [29] para realizar a conversão para escala de cinzentos e o método de *Otsu* [29] para estabelecer um limiar que permita classificar cada *pixel* como cobre ou substrato (material não condutor) [25].

[6] Os métodos mais utilizados para deteção de bordas são: aproximação de Sobel [30], aproximação de Prewitt [31], aproximação de Robert [32], método de Canny [33] e o Laplaciano do método Gaussiano [34].

2.3.4.4. Geração de esquema elétrico

No método onde os componentes são removidos, o resultado do processamento de imagem em escala de cinzentos consiste na melhor aproximação ao circuito contido na PCB, dado que os componentes são removidos e adicionados manualmente nas novas cópias do circuito analisado [26].

No caso onde os componentes não são removidos o esquema elétrico é gerado após a identificação dos componentes e ligação de cada pino à pista condutora correspondente [25]. Em alguns casos o resultado final pode ser apresentado na forma apresentada na Figura 2.4

$$nome - da - ligação:\ \ componente_A - pista_A \quad componente_B - pista_B$$

Figura 2.4 – Formato possível de saída de dados [25].

Uma saída de dados na forma de esquema elétrico pode igualmente ser possível, dependendo das características do *software* utilizado.

2.3.4.5. União das camadas

Esta etapa é exclusiva para as PCB com duas camadas de material condutor.

Tendo sido obtido, de forma independente, o esquema elétrico de cada camada, a relação entre duas camadas da mesma PCB é encontrada através dos furos presentes no esquema elétrico de cada camada, através de quatro etapas [25]:

- Encontrar na camada superior a pista que contem o furo;
- Encontrar o furo correspondente na camada inferior;
- Encontrar na camada inferior a pista que contem o furo;
- Mover para a camada superior a(s) pista(s) conectada(s) ao furo, de forma a formar um esquema elétrico único.

Nos casos onde um furo não tem correspondência na outra camada, é usualmente assumido que a deteção do furo foi falsa e é ignorada essa informação [25].

2.3.4.6. Fabricação, teste e análise do circuito

Esta etapa é a etapa final no processo de engenharia inversa de uma PCB. O fabrico de uma cópia da placa analisada pode ser realizado utilizando métodos comuns

de fabrico em função do resultado final fornecido pelo *software* utilizado. Se o resultado final for o circuito elétrico de uma forma gráfica o circuito é transferido para a nova PCB com recurso a técnicas de remoção do material condutor, baseadas no uso de persulfato de amónio ($(NH_4)_2S_2O_8$) e cloreto de ferro ($FeCl_2$) [26]. Caso o formato de saída seja um ficheiro do tipo GERBER[7] então o fabrico será automático, sendo apenas necessário transferir o ficheiro para a máquina de fabrico [26].

No processo de fabrico é importante garantir que os componentes utilizados possuem todos os requisitos dos componentes originais. Após o fabrico é testada a funcionalidade do circuito para verificar se o circuito cópia opera da mesma forma que o circuito original. Se o resultado do teste for negativo devem ser repetidos os passos descritos desde o ponto B desta secção [26].

2.3.4.7. Transformação do circuito de uma PCB em linguagem HDL

Atualmente existem técnicas para transformar um circuito contido numa PCB em código *Verilog* HDL baseado num gerador automático de modelo *Verilog* HDL, ou seja, consiste numa metodologia para converter dados para um formato tecnológico mais recente [27].

Em [27] são referidos dois métodos para efetuar esta conversão: conversão direta para a nova tecnologia (HDL) do circuito existente ou para recuperação do esquema elétrico e funcionamento do circuito através de técnicas de EI análise e melhoramento do circuito antes da conversão para a linguagem de descrição de *hardware* (reengenharia). A Tabela 2.1 apresenta de forma mais detalhada os requisitos necessários para cada um dos métodos.

Tabela 2.1 – Métodos de conversão/migração para Verilog HDL [27].

	MÉTODO I	**MÉTODO II**
Resumo	**Consiste numa conversão automática para a nova tecnologia (HDL)**	**Realizada engenharia inversa e reengenharia antes da migração para HDL**
Requerimentos	- Informação técnica completa necessária (esquemático requerido); - Ligações ou código ABEL [8] necessários; - Conversão a partir de TTL ou PLD	- Nenhuma informação técnica requerida (esquemático não requerido); - Informação incompleta, sistemas modificados ou reparados não são um problema; - Conversão a partir de TTL

[7] O formato GERBER [35] é um formato *standard* e contém dados sobre a informação a ser transferida para uma PCB baseado num sistema de coordenadas. Toda a informação relativa à forma e tamanho das pistas, furos e outras caraterísticas está contida no ficheiro GERBER.
[8] A linguagem de descrição de *hardware* ABEL permite escrever projetos digitais com equações, tabelas de verdade, diagramas de estado, ou qualquer combinação dos três, otimizar e simular o projeto sem especificar um dispositivo ou a atribuição de pinos [36].

A Figura 2.5 apresenta a grande diferença entre os métodos I e II sob a forma de diagrama de blocos.

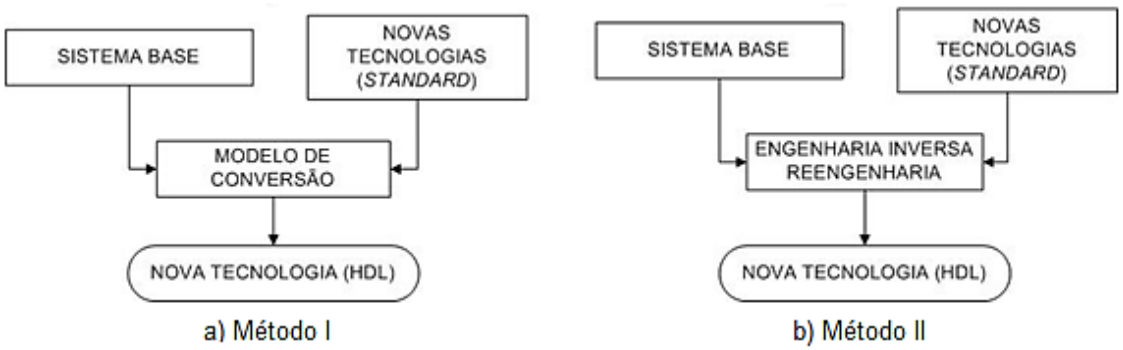

Figura 2.5 – Diagrama de blocos da conversão para linguagem HDL [27].

Um sistema AVMG (*Automatic Verilog Model Generator*) é constituído basicamente por quatro subsistemas (ver Figura 2.6) [27]:

- Análise de imagem;
- Gerador gráfico;
- Gerador de ligações;
- Gerador de código HDL.

Nesta técnica é assumido que a PCB sob análise apenas possui camada condutora de um ou dois lados e nunca PCB do tipo *waffer* [27].

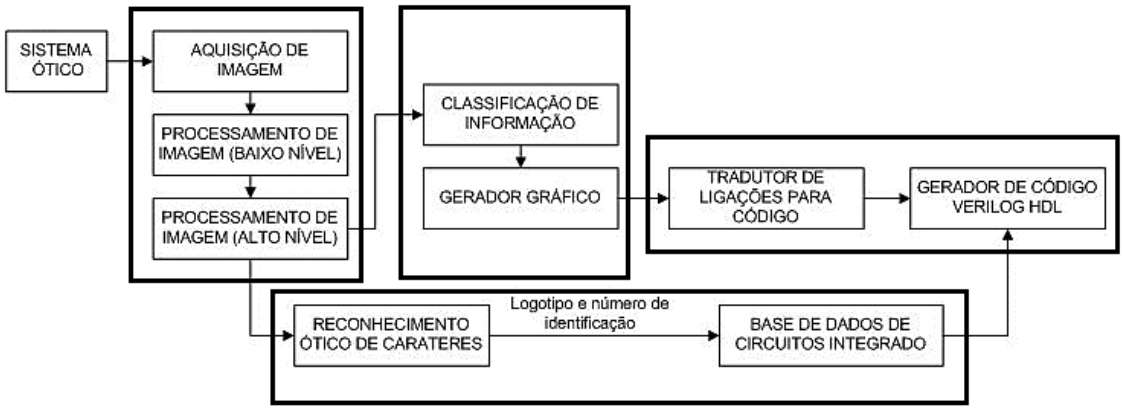

Figura 2.6 – Arquitetura de um gerador automático de modelo Verilog HDL (AVMG) [27]

As técnicas de aquisição, processamento e análise de imagem utilizadas num AVMG são as técnicas tradicionais, já anteriormente descritas, à exceção da análise dos ICs contidos na PCB. Este sistema possui uma base de dados com informação de uma vasta gama de ICs e bibliotecas TTL. O sistema utiliza um *software* de reconhecimento de carateres (OCR – *Optical Character Recognition*) para identificar o número, tipo e/ou família de cada IC, sendo auxiliado por fontes externas de identificação em casos de maior dificuldade [27].

O gerador gráfico sintetiza de forma gráfica a informação extraída da análise da imagem (essencialmente as ligações), formando um circuito primitivo [27].

Após um circuito primitivo definido, o gerador de ligações elabora um relatório onde descreve todas as ligações do sistema, como exemplificado na Figura 2.7 a), tendo já nesta fase a capacidade de transformar os dados contidos no gráfico num formato GERBER [27].

A geração do código é feita em duas etapas: a reunião das informações necessárias (relatório de ligações e informação dos ICs) e a conversão para código *Verilog* HDL, como apresentado na Figura 2.7 b) [27].

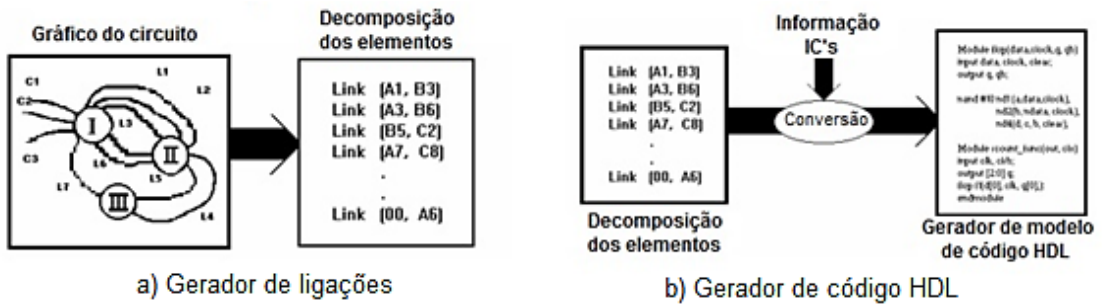

Figura 2.7 – Gerador de código Verilog HDL: a) Gerador de ligações b) Gerador de código HDL [27].

No produto final são utilizadas variáveis globais de controlo do tempo de funcionamento, incluídas no *Verilog*, de forma a que tempos de atraso interno dos ICs sejam tidos em conta [27].

2.3.5. Engenharia inversa baseada em algoritmos para extração do comportamento

A EI baseada em algoritmos é uma técnica que favorece a automatização e é menos invasiva que a maioria das técnicas descritas anteriormente. A bibliografia existente nesta área de estudo mostra que as técnicas de EI baseadas em algoritmos destinam-se atualmente a circuitos digitais [9], [11], [37].

As técnicas de EI baseadas em algoritmos, de uma forma geral, utilizam três métodos de reprodução do circuito original: descrição do circuito numa linguagem de descrição de *hardware* formando componentes de alto nível [11], comparação com biblioteca de circuitos típicos [9] e conversão do circuito numa máquina de estados finitos (FSM – *Finite State Machine*) [11], [37].

A abordagem para circuitos digitais combinatórios e sequenciais difere, sendo seguidamente apresentados métodos para os dois tipos de circuitos. No entanto alguns algoritmos descritos na seção dos circuitos sequenciais podem ser aplicados também a circuitos combinatórios, dado que é possível transformar um circuito combinatório num circuito sequencial (quando a tabela de verdade do circuito de atribuição de estados é igual à tabela de verdade de atribuição das saídas) [38].

2.3.5.1. Circuitos digitais combinatórios

A. Algoritmo I: identificação de sub-circuitos

Um dos métodos utilizados atualmente para identificação de circuitos combinatórios é a aproximação por correspondência funcional e consiste numa primeira fase em identificar os nós do circuito (sub-circuitos) que coincidem com componentes de uma biblioteca de circuitos conhecidos [9]. Por exemplo se em um determinado nó for encontrada a função $f(a,b,c) = ab + bc + ca$, que corresponde a um somador, pode indicar a existência de somadores *multibit* [9]. A análise da função lógica (análise comportamental) em detrimento da análise estrutural do circuito representa uma diferença significativa relativamente a outros métodos apresentados anteriormente.

A técnica descrita em [9] utiliza técnicas de corte[9] e correspondência booleana (*Boolean matching)* inicialmente utilizadas em tecnologias de mapeamento. Para que a técnica *cut-enumeration* seja eficiente, um sub-circuito (*cut*) diz-se k-viável (k*-feasible)* quando o número de *inputs* por porta lógica é inferior a k [39]. Esta técnica é eficiente para $k < 6$, sendo uma limitação desta técnica, embora circuitos mais complexos possam ser usualmente decompostos em sub-circuitos com menos de 6 *inputs*. Uma vez dividido em sub-circuitos os circuitos são agrupados em classes equivalentes utilizando correspondência booleana por permutação independente. Por exemplo, o(s) nó(s) que correspondem à função $f(a,b,c) = ab + c$ e o(s) nó(s) que correspondem à função $f(a,b,c) = bc + a$, podem ser agrupados na mesma classe equivalente. Cada classe equivalente pode corresponder a um circuito conhecido existente na biblioteca [9].

B. Algoritmo II: agregação multi-bit

Sendo as funções de cada nó conhecidas, a etapa seguinte é utilizar um algoritmo para identificar nós com interligações específicas entre si, nomeadamente: agrupar sub-circuitos com sinal de seleção comum ou agrupar sub-circuitos onde a saída é a entrada de outro sub-circuito e assim sucessivamente. Estes dois tipos de análises permitem identificar *multiplexers/desmultiplexers* ou árvores de paridade, respetivamente [9].

A agregação de nós com funções desconhecidas, mas com sinais comuns, podem ser classificados como módulos desconhecidos e posteriormente sujeitos a outros algoritmos ou análise por especialista humano [9].

[9] Os algoritmos de corte (*cut-enumeration)* são, de uma forma genérica, algoritmos que dividem circuitos mais complexos em sub-circuitos mais simples baseados numa filosofia "dividir e conquistar" [40]. Este tipo de algoritmos é também utilizado por grande parte das FPGAs para sintetização de *hardware* [41].

C. Algoritmo III: identificação e propagação de palavras

A agregação de nós que operam simultaneamente pode indicar a existência de uma palavra (dados multibit de tamanho fixo). Uma vez encontrada uma palavra, mais palavras podem ser geradas por propagação da mesma pelas portas lógicas, ou seja, sendo conhecida uma palavra w é possível encontrar as condições para que o conteúdo de w seja propagado numa nova palavra w', para todos os valores de w [9]. O circuito da Figura 2.8 mostra um exemplo de propagação de uma palavra.

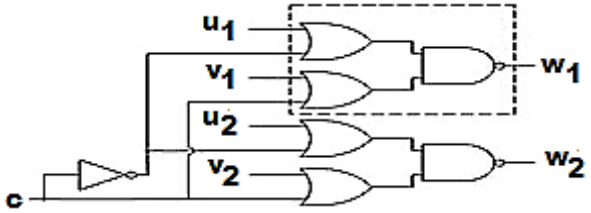

Figura 2.8 – Exemplo de propagação de palavra [9].

O circuito da Figura 2.8 consiste num seletor controlado pelo sinal c, onde a negação dos valores de u_1 e u_2 é propagada para as saídas w_1 e w_2 (respetivamente) quando $c = 0$, caso contrário a negação dos valores de v_1 e v_2 é propagada para as saídas w_1 e w_2. Este tipo de propagação é o objeto de análise dos algoritmos de propagação de palavra [9].

O algoritmo de análise de propagação de palavra segue uma aproximação do tipo "previsão e verificação". Considerando uma palavra inicial w a etapa de previsão consiste em encontrar um conjunto S formado pelas potenciais palavras propagadas, através do agrupamento das saídas das portas acionadas por w. De seguida, para cada nova palavra w' ($w' \in S$) existe um conjunto C de sinais de controlo, que são processados, e conectados às entradas (para um número pequeno de entradas) para os quais a saída corresponde a w' [9].

A etapa de verificação consiste na realização de simulação simbólica[10] de um sub-circuito relevante para a propagação que está a ser analisada. Para efeitos de simulação as entradas do sub-circuito são inicializados da seguinte forma [9]:

- Cada *bit* de w é convertido para um valor simbólico D;
- Para cada combinação de interesse entre os sinais de controlo, todos os valores binários são avaliados;
- As restantes entradas do sub-circuito são consideradas sem interesse (atribuído X).

Uma simulação com um valor particular σ ($\sigma \in C$) atribuído aos sinais de controlo é considerado bem-sucedida se foi propagado para a saída D ou $\overline{D}$. Nesse caso, w propaga w' quando a combinação σ ocorre, permitindo que w' seja testado em futuras

[10] Por razões de eficiência é utilizada a representação simbólica onde as portas lógicas operam no domínio $\{0,1,D,\overline{D},X\}$, onde D representa um valor do conjunto $\{0,1\}$, $\overline{D}$ é a negação de D e X representa um valor desconhecido [9].

propagações. Uma aproximação análoga pode ser efetuada no sentido inverso, isto é, conhecida a palavra w' determinar qual a palavra w e a combinação σ que a originaram, permitindo assim testar propagação no sentido inverso [9].

D. *Algoritmo IV: Identificação e correspondência de módulos*

Os algoritmos baseados em divisão por sub-circuitos e análise dos nós possuem duas limitações principais: cada nó está limitado a um máximo de seis entradas e a dificuldade de identificar estruturas sem um padrão interno de interconexões. O algoritmo IV que será descrito ultrapassa as limitações dos algoritmos referidos anteriormente, pois é baseado numa aproximação através da construção de módulos e posteriormente relacionamento com componentes conhecidos de uma biblioteca [9].

Este algoritmo baseia-se na premissa que um circuito combinatório opera de tal forma que para cada palavra de entrada corresponde uma palavra de saída. Desta forma, se for descartado todo o circuito intermédio a identificação de módulos pode ser efetuada com recurso a um algoritmo que efetua a comparação da palavra de entrada e verifica a palavra de saída comparando com uma referência predefinida existente numa biblioteca de circuitos, averiguando se o módulo desconhecido e o módulo referência efetuam a mesma operação lógica (ϕ), de acordo com a Figura 2.9 [9].

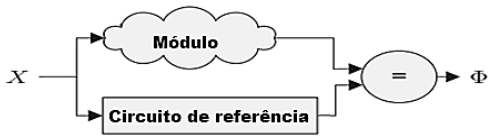

Figura 2.9 – Algoritmo de identificação e correspondência de módulos [9].

Este algoritmo é útil na deteção de somadores, subtratores, operações booleanas e deslocação/rotação de *bits* [9].

E. *Algoritmo V: Análise baseada em suporte comum*

Este algoritmo deteta módulos que não têm necessariamente entradas ou saídas de palavras ou consistam em pequenos sub-circuitos replicados. Esta técnica de análise pode ser utilizada para detetar combinações de módulos com uma propriedade específica: cada uma das saídas do módulo depende do mesmo conjunto de entradas, como apresentado na Figura 2.10.

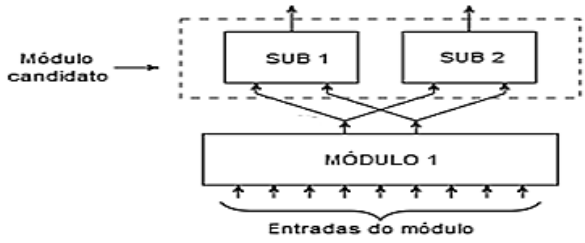

Figura 2.10 – Circuito típico para aplicação do algoritmo baseado em suporte comum [9].

A identificação de nós de saída com suportes idênticos começa com o agrupamento dos mesmos em classes equivalentes, agrupando dois (ou mais) nós de saída se e só se as todas as entradas lógicas que influenciam os nós agrupados forem as mesmas. Um algoritmo para encontrar nós é aplicado aos nós da mesma classe e origina um módulo candidato, que posteriormente é comparado com os componentes existentes na biblioteca [9].

Este tipo de algoritmo permite detetar descodificadores, *desmultiplexers* e/ou contadores, visto que em cada um dos casos as diferentes saídas dependem do mesmo conjunto de entradas [9].

F. Algoritmo VI: Pós-processamento de módulos combinatórios

O algoritmo de pós-processamento tem por objetivo fundir vários módulos combinatórios encontrados por outros algoritmos, formando módulo maiores, com nível de abstração maior. Os módulos resultantes da fusão são armazenados na biblioteca de componentes para futuras comparações (tal como os componentes constituintes da fusão) [9].

Um caso onde este algoritmo pode ser aplicado com sucesso é quando as saídas de um descodificador são as entradas de seleção de um *multiplexer*, então ambos os módulos podem ser agrupados, formando um módulo de roteamento de sinal [9].

2.3.5.2. Circuitos digitais sequenciais

A. Algoritmo I: identificação de contadores, registos de deslocamento e memórias RAM

Em circuitos digitais sequenciais existem três tipos de circuitos muito comuns: contadores, registos de deslocamento e memórias RAM. A identificação deste tipo de circuitos pode ser importante pois cobre uma quantidade significativa de circuitos existentes e baseia-se em primeiro lugar numa análise topológica e posteriormente em algoritmos de verificação comportamental [9].

Para que possam ser aplicados os algoritmos descritos nesta secção é necessário um conhecimento prévio de uma descrição de alto-nível do circuito e interface de entradas e saídas [9].

A identificação de contadores é realizada em duas etapas: numa primeira fase é aplicado um algoritmo que efetua uma análise topológica e identifica conjunto de *latches* que cuja topologia pode corresponder a de um contador. Em segundo lugar um algoritmo de análise baseada numa análise SAT (satisfatibilidade booleana) é aplicado para verificar se funcionalmente o módulo candidato a contador cumpre as duas condições seguintes [9]:

- um *bit* i transita para o nível alto quando todos os *bits* de ordem inferior se encontram no nível alto (contador crescente) ou transita para o nível baixo quando todos os *bits* menos significativos estão no nível baixo (contador decrescente);
- os sinais de controlo do contador (*habilitação* e *reset*) são os mesmos para todos os *bits* do contador;

De uma forma analítica cada *bit* de um contador, representado por C_i, tem de satisfazer as condições lógicas da equação [9]

$$\begin{aligned} C_i = \quad & \neg r \wedge e \ \wedge (q_1 \wedge q_2 \ldots \wedge q_{i-1}) \wedge \neg q_i \ \vee \\ & \neg r \wedge e \ \wedge (\neg q_1 \vee \neg q_2 \ldots \vee \ \neg q_{i-1}) \wedge q_i \ \vee \\ & \neg r \wedge \neg e \wedge q_i \vee \ s \end{aligned} \tag{2.1}$$

onde $q_1, q_2, \ldots, q_{i-1}$ representam as *latches* do contador e os sinais r, s e e representam os sinais de *reset*, *preset* e habilitação, respetivamente. Pela análise da equação (2.1) verifica-se que cada bit do contador se encontra no estado ativo se o algoritmo de análise funcional detetar uma das seguintes condições [9]:

- Sinal de *reset* inativo, sinal de habilitação ativo, todos os *bits* de ordem inferior a i estão no nível alto e o *bit* de ordem i está no nível baixo;
- Sinal de *reset* inativo, sinal de habilitação ativo, algum *bit* de ordem inferior a i está no nível baixo e o *bit* de ordem i está no nível alto;
- Sinal de *reset* inativo, sinal de habilitação inativo e o *bit* de ordem i está no nível alto;
- Sinal de *preset* ativo.

Algoritmicamente são processados, para cada *bit* do contador, os cofatores[11] d_i de acordo com a equação

$$\begin{aligned} f_i &= cofactor\,(d_i, q_1 \wedge \ldots \wedge q_{i-1} \wedge \neg q_i)\,, \\ h_i &= cofactor\,(d_i, q_1 \wedge \ \ldots \wedge q_{i-1} \wedge q_i)\,, \\ g_i &= cofactor\,(d_i, (\neg q_1 \vee \ \ldots \vee \neg q_{i-1}\,) \wedge \neg q_i) \end{aligned} \tag{2.2}$$

onde pode constatar-se que f e g representam as situações onde houve a alteração e manutenção do valor de q_i, respetivamente [9].

Pela relação entre as expressões definidas em (2.1) e (2.2) f_i, g_i e h_i podem ser reduzidos à equação [9].

$$\begin{aligned} f_i &= (\neg r \wedge e) \vee s\ , \\ h_i &= (\neg r \wedge \neg e) \vee s\ , \\ g_i &= \neg r \vee s\,. \end{aligned} \tag{2.3}$$

[11] Dadas as funções booleanas f e g, $cofactor(f, g)$ é a função obtida quando f é avaliado sobre um domínio especificado por $g = 1$.

Os sinais r, s e e devem ser comuns a todos os *bits*, logo as funções f_i, h_i e g_i devem ser equivalentes. Assim sendo, o algoritmo que implementa uma análise topológica pode determinar que um conjunto de *latches* não formam um contador se as funções f_i, h_i e g_i não forem equivalentes para todos os i-bites [9].

Os algoritmos de identificação de registos de deslocamento seguem um procedimento igual ao utilizado para deteção de contadores, diferindo apenas nas condições. No caso de um registo de deslocamento, o estado seguinte de cada *bit*, s_i, é definido pelas condições da equação [9]

$$S_i = \quad \neg r \wedge (e \wedge q_{i-1} \vee \neg e \wedge q_i) \vee s \tag{2.4}$$

onde q_{i-1} representa as *latches* associadas aos *bits* de ordem anterior e os sinais r, s e e representam os sinais de *reset*, *preset* e habilitação, respetivamente. Pela análise da equação (2.4) um *bit* de um registo de deslocamento encontra-se no nível alto quando se verifica uma das seguintes condições: [9]

- Sinal de *reset* inativo, sinal de habilitação ativo e *bit* anterior $(i - 1)$ no estado alto;
- Sinal de *reset* inativo, sinal de habilitação inativo e *bit* i no estado alto;
- Sinal *preset* ativo.

No caso dos registos de deslocamento os cofatores d_i são dados pela equação

$$\begin{aligned} f_i &= cofactor\ (d_i, q_{i-1} \wedge \neg q_i), \\ h_i &= cofactor\ (d_i, \neg q_{i-1} \wedge q_i) \end{aligned} \tag{2.5}$$

onde pode constatar-se que f e g representam as situações onde houve ou não deslocamento, respetivamente [9].

Pela relação entre as expressões definidas em (2.4) e (2.5) então f_i e g_i podem ser reduzidos à equação (2.6) [9].

$$\begin{aligned} f_i &= \neg r \wedge e \vee s, \\ h_i &= \neg r \wedge \neg e \wedge s. \end{aligned} \tag{2.6}$$

De forma semelhante ao descrito relativamente aos contadores, o algoritmo verifica se as funções f_i e g_i são iguais para cada *bit* do candidato a registo de deslocamento.

Por fim, existe mais um circuito comum em circuitos digitais sequenciais: memórias RAM. A identificação de memórias RAM baseia-se na deteção de *latches/flip-flop*s associados numa lógica de leitura e escrita [9].

A técnica de identificação de memórias RAM é baseada no facto de uma memória RAM ter uma estrutura análoga a uma árvore onde as células de memória são as folhas e os *outputs* são as raízes. O algoritmo de identificação de lógica de leitura efetua a marcação das *latches/flip-flops* que satisfaçam as duas condições que se seguem: pelo

menos uma das portas se encontra marcada e a porta apenas possui uma saída [9]. Após o algoritmo efetuar a marcação, é realizada uma análise funcional. Considerando que os *flip-flops* representadas por l_i e s_i correspondem aos *flip-flops* de armazenamento da informação e endereço, respetivamente, é identificada lógica de leitura numa memória RAM se forem verificadas as condições [9]:

- Se $y = f(s_1, \dots, s_k, l_1, \dots, l_n)$ então $y = l_i$ ou $y = \neg l_i$ para qualquer valor de s_i $(i \in \{1, \dots, k\})$. Por outras palavras para cada sinal de seleção s_i apenas um *flip-flop* de dados (l_i) é propagado para a saída;
- Todas os *flip-flops* são propagadas para a saída ($y = l_i$ ou $y = \neg l_i$) para todos os valores de i apropriados.

A identificação de lógica de escrita baseia-se na estrutura apresentada na Figura 2.11 baseada num *multiplexer* 2:1 que em função do sinal de habilitação de escrita guarda o último valor ou associa um novo valor [9].

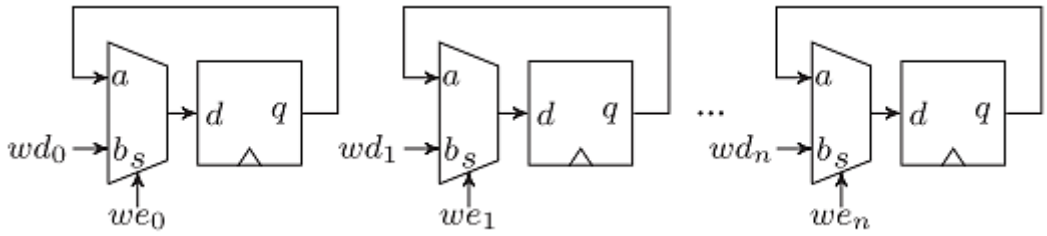

Figura 2.11 – Lógica de escrita de memória RAM, onde $\mathbf{we_i}$ e $\mathbf{wd_i}$ representam a habilitação de escrita e dados para escrita, respetivamente [9].

Um algoritmo é utilizado para identificação de lógica de escrita que verifica as seguintes propriedades [9]:

- Todos os sinais de habilitação de escrita podem ser ativados: $we_i \neq 0$;
- Para dois sinais de habilitação de escrita da mesma memória RAM não existem dois sinais diferentes, isto é, é necessário verificar que $we_i = 0 \wedge we_j = 0, \forall i \neq j$.

Se as duas condições anteriores forem verificadas, é identificada lógica de escrita de memórias RAM. Note-se que neste caso foi considerado que a escrita era efetuada quando o sinal de habilitação de escrita se encontra no nível alto, contudo o algoritmo pode ser adaptado para a situação inversa [9].

B. Algoritmo II: Algoritmo de comparação de padrões

O algoritmo descrito nesta secção possui algumas técnicas semelhantes ao Algoritmo I descrito na secção de circuitos combinatórios. Considerando um circuito desconhecido $C = (I, O, S)$ onde I, O e S representam as entradas (*inputs*), saídas (*outputs*) e S a definição formal do comportamento do circuito, respetivamente, e uma biblioteca de componentes abstratos $\mathcal{L} = \{\alpha_1, \alpha_2, \dots, \alpha_n\}$. O objetivo é encontrar um

circuito $C' = (I, O, \Gamma)$ onde C' é equivalente a C e Γ os componentes abstratos de $\mathcal{L}$. A aplicação deste algoritmo requer várias etapas [11]:

- Definição da biblioteca: construção de uma biblioteca de componentes abstratos $\mathcal{L}$;
- Identificação de blocos funcionais: Divisão de um circuito desconhecido C em blocos funcionais (sub-circuitos) $b_1, b_2, \dots, b_k$, onde cada b_i é um candidato para comparação com os componentes da biblioteca $\mathcal{L}$;
- Comparação com a biblioteca de componentes: dado um bloco funcional candidato b_i determinar se existe em $\mathcal{L}$ um componente α_j que ultrapasse com sucesso uma das verificações descritas em seguida.

Uma das verificações que é realizada consiste em analisar se todas as entradas e saídas de um componente abstrato α se encontram na especificação S do circuito desconhecido C. Se essa correspondência existir é efetuada a verificação se C satisfaz S e caso a verificação seja concluída com sucesso é reportada a correspondência entre C e α, caso contrário é prosseguida à comparação com outro componente α da biblioteca $\mathcal{L}$ [11].

Em alguns casos o número de entradas e saídas pode diferir entre C e α, contudo podem realizar a mesma função lógica, sendo utilizado então um procedimento heurístico. Considerando que os eventos $\Delta a, \Delta b, \dots, \Delta n$ provocam a alternância entre estados de funcionamento (Figura 2.12 a)) por testes aos nível do *hardware* é possível extrair o padrão de respostas a diferentes eventos, contruindo um gráfico de padrão do circuito, como apresentado na Figura 2.12 b) [11].

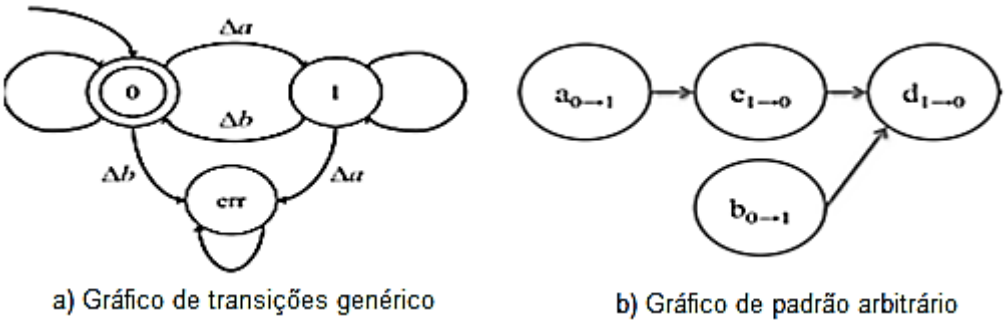

Figura 2.12 – Diagramas de transições e padrões [11].

A comparação com os componentes abstratos α passa a ser realizada ao nível dos gráficos de padrão que no caso dos componentes da biblioteca $\mathcal{L}$ poderão ser obtidos por *software* de simulação ou já se encontrarem armazenados de comparações prévias. Assim sendo quando o circuito desconhecido C e o componente abstrato α possuem o mesmo gráfico de padrão, mesmo que o número de entrada e saídas não seja idêntico, é considerado que C corresponde a α[12] [11].

[12] Mais informação sobre diferentes formas de construção gráficos padrão e compatibilidade entre diferentes gráficos padrão podem ser consultadas em [11], [42], [43].

C. Algoritmo III: Algoritmo de extração de máquina de estados de um circuito

O algoritmo de extração de uma máquina de estados finitos de um circuito baseia-se na aplicação de seis sub-algoritmos distintos, que operam em duas estratégias distintas de análise: identificação da propagação dos sinais de habilitação e identificação de componentes fortemente conectados [37].

A função de cada sub-algoritmo encontra-se na Tabela 2.2:

Tabela 2.2 – Sub-algoritmos para extração de máquina de estados finita [37].

ALGORITMO	FUNÇÃO
1	Identifica potenciais registos de estado
2	Distinção entre os registos de estado reais dos potenciais registos de estados
3	Agrupamento de registos de acordo com a partilha de sinais de habilitação
4	Agrupa registos de estado de acordo com os grupos identificados
5	Identifica circuitos de lógica combinatória que formam realimentação dos registos de estado
6	Análise de componentes fortemente conectados

O algoritmo geral pode ser melhor compreendido pelo fluxograma da Figura 2.13.

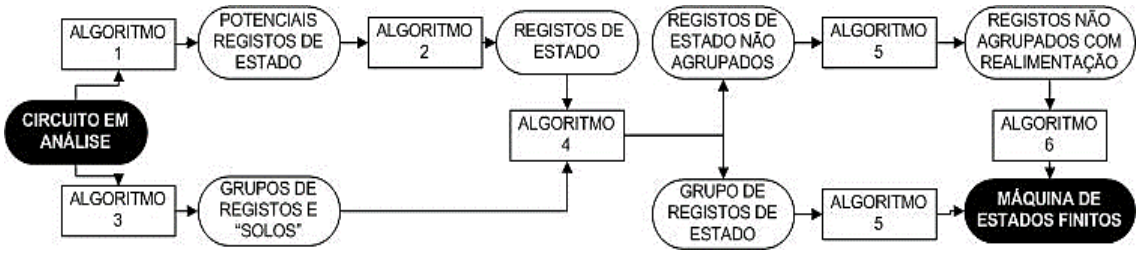

Figura 2.13 – Funcionamento do algoritmo de extração de máquinas de estado finitas [37].

A identificação de potenciais registos de estado e distinção entre registos de estado reais de registo de estados potenciais (Algoritmos 1 e 2) baseia-se nos registos onde a saída retorna a entrada através de circuitos combinatórios (realimentação) [37].

No entanto, nem todos os registos realimentados devem ser considerados registos de estado uma vez que existem circuitos tais como acumuladores que estruturalmente são realimentados, porém têm funções distintas de uma máquina de estados finita [37]. Assim sendo, é aplicado um algoritmo de eliminação de potenciais registos de estado (Algoritmo 2) baseado nas aplicações típicas de uma máquina de estados finita (controlo de circuitos). O Algoritmo 2 examina cada potencial registo de estado assinalado pelo Algoritmo 1 e identifica quais os registos que se encontram conectados a um (ou mais) sinal de controlo de um circuito e declarando-os como registos de estado e eliminando os que não satisfazem a referida verificação [37].

O Algoritmo 3 identifica quais os registos que são controlados pelo mesmo sinal de habilitação, agrupando os registos que partilham o mesmo sinal de habilitação e identificando os registos restantes como “solos”. No contexto da extração de uma

máquina de estados finita, é pouco provável que registos controlados por sinais de habilitação diferentes pertençam à mesma máquina de estados [37].

O Algoritmo 4 intersecta os registos de estado identificados de formas distintas pelos Algoritmos 2 e 3, de forma a separar registos de estados de máquinas de estados finitas distintas (cruzamento de informação dos dois algoritmos permite maior fiabilidade na distinção) separando novamente em grupos de registos de estado e registo não agrupados [37].

O Algoritmo 5 identifica a lógica combinatória com realimentação presente no circuito, de forma a inferir de que as diferentes combinações irão influenciar a saída da máquina de estados finita [37].

O Algoritmo 6 deteta componentes fortemente conectados (Figura 2.14) permitindo identificar registos da mesma máquina de estados que não tinham sido identificados nas etapas anteriores [37].

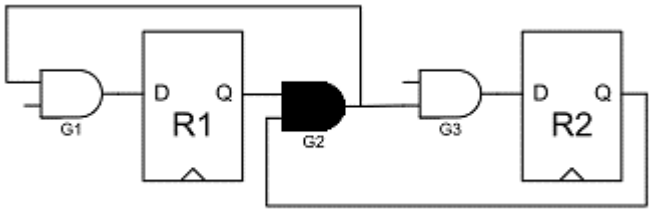

Figura 2.14 – Dispositivos fortemente conectados [37].

Se dois registos são afetados pelo mesmo circuito combinatório então os dois registos podem ser agrupados como pertencentes à mesma máquina de estados finita [37]. Pela análise da Figura 2.14 é possível constatar que o registo R1 é realimentado pelo circuito combinatório que contém as portas G1 e G2 e o registo R2 é realimentado pelo circuito combinatório que contém as portas G2 e G3, isto é, a porta G2 é comum ao circuito de ambos os registos, logo são considerados componentes fortemente conectado e por conseguinte R1 e R2 são agrupados como pertencentes à mesma máquina de estados finita [37].

2.4. Síntese de circuitos digitais

A estrutura do código em linguagem de descrição de *hardware* tem influência na obtenção de um esquema lógico que reproduza o comportamento descrito no código (síntese).

A revisão das ferramentas já existentes para síntese de circuitos através de código VHDL e as estruturas de código mais indicadas para diferentes situações, podem ser consultadas no Anexo C deste documento.

3. Arquitetura do sistema e algoritmos de replicação de circuitos digitais

3.1. Introdução

Neste capítulo será apresentada a arquitetura do sistema desenvolvido bem como descritos os algoritmos desenvolvidos desde a aquisição e processamento dos dados (medições ao circuito sob análise) até obtenção da réplica.

O circuito, a descrever, deve encontrar-se em funcionamento durante o processo de análise e descrição, sendo replicadas unicamente as funcionalidades observadas, ou seja, este sistema apenas garante uma réplica capaz de substituir o circuito analisado para operação num sistema igual ao aquele onde foi analisado.

Os algoritmos desenvolvidos não utilizam qualquer biblioteca de circuitos conhecidos ou registo de circuitos previamente analisados, visto que nos moldes de operação do sistema (dependência do ambiente onde está inserido o circuito a replicar) o mesmo circuito integrado pode originar diferentes réplicas. A não utilização de bibliotecas é também a inovação que se pretende introduzir comparativamente a sistemas de engenharia inversa baseados em algoritmos existentes, desenvolvendo um sistema com maior capacidade de análise ao invés de comparação.

3.2. Funcionamento geral e arquitetura do sistema

Atualmente, existe uma grande variedade de circuitos integrados, pelo que o sistema de replicação deverá efetuar operações de triagem e análise. De uma forma genérica, o sistema executa sequencialmente as seguintes ações:

- Identificação dos pinos destinados a alimentação: identifica os pinos destinados a alimentação do circuito e sinais constantes, excluindo-os da restante análise e determinando uma referência para as medições;
- Despiste de sinais analógicos: executa uma procura por sinais analógicos de forma a garantir que não está a replicar circuitos para os quais não foi projetado. Quando encontrados sinais analógicos durante a execução é enviado um aviso ao utilizador (no entanto a replicação prossegue sem garantias de sucesso);
- Análise de circuitos digitais combinatórios: efetua uma análise para verificar se o circuito sob análise é combinatório. Em caso afirmativo aplica algoritmos de análise combinatória, caso contrário inicia procedimentos de replicação de circuitos sequenciais;
- Deteção do sinal de relógio: Aplicável apenas a circuitos sequenciais síncronos, esta etapa destina-se a descobrir o pino do sinal de relógio;
- Deteção do sinal de *reset*: Aplicável apenas a circuitos sequenciais, esta etapa destina-se a descobrir o pino do sinal de *reset*;
- Análise de circuitos digitais sequenciais: Aplicável apenas a circuitos sequenciais, esta etapa destina-se à execução de algoritmos de análise sequencial;

- Escrita do ficheiro VHDL: após determinado o comportamento do circuito é escrito um ficheiro de descrição do circuito na linguagem VHDL.

O funcionamento genérico do sistema é sintetizado pelo fluxograma da Figura 3.1.

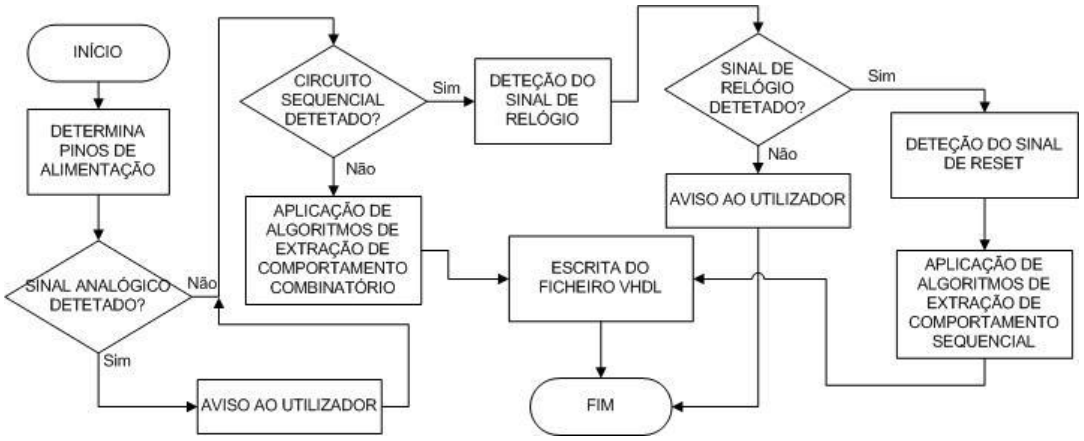

Figura 3.1 – Funcionamento geral do sistema de replicação de circuitos integrados digitais.

Pela observação do fluxograma verifica-se que numa primeira fase são efetuadas ações de triagem sendo apenas aplicados os algoritmos de análise após identificado o tipo de circuito, agilizando o processo de análise desta forma.

O sistema desenvolvido assenta numa estrutura modular, onde cada módulo desempenha uma função específica. Os diferentes módulos são independentes e apenas partilham a estrutura de dados existindo um módulo responsável por realizar a gestão de todos os módulos durante a execução dos algoritmos, garantindo a sequencialidade pretendida para as operações e o acesso coordenado à estrutura de dados.

O armazenamento dos dados é realizado em ficheiros de texto, com estrutura e organização dos dados definidas, de tal forma que os diferentes módulos realizam a leitura e escrita dos dados baseados exclusivamente na organização definida. Em alternativa os dados poderiam ser guardados numa base de dados. No entanto como os dados recolhidos têm um interesse apenas temporário (durante a execução) e destinam-se a uso local exclusivamente o armazenamento em ficheiros é uma solução que satisfaz as necessidades do sistema.

A estrutura modular implementada consiste em diferentes ficheiros executáveis (um para cada módulo). A utilização de uma estrutura modular permite o sistema beneficiar de algumas vantagens, tais como:

- Concebido para fácil otimização: o sistema pode ser otimizado substituindo apenas um módulo por um outro que execute a mesma função. Para o efeito não é necessário qualquer conhecimento do código fonte do módulo a otimizar, apenas é necessário conhecimento da estrutura de dados e função desempenhada pelo módulo;
- Versatilidade de conceção: Cada módulo pode ser programado na linguagem mais adequada à função de cada módulo, dado que não existe partilha de variáveis entre módulos e do ponto de vista do coordenador do sistema apenas o ficheiro executável é relevante ao processo;
- Controlo do fluxo de dados: Módulos mais pequenos permitem melhor controlo do fluxo de dados;

- Facilidade de análise e evolução: a implementação modular facilita a análise e compreensão do sistema, fator importante na otimização. A inclusão de novas funcionalidades no sistema – evolução – pode ser efetuada através da junção de novos módulos e adaptação do módulo coordenador.

Na Figura 3.2 encontra-se apresentada a estrutura modular do sistema implementado.

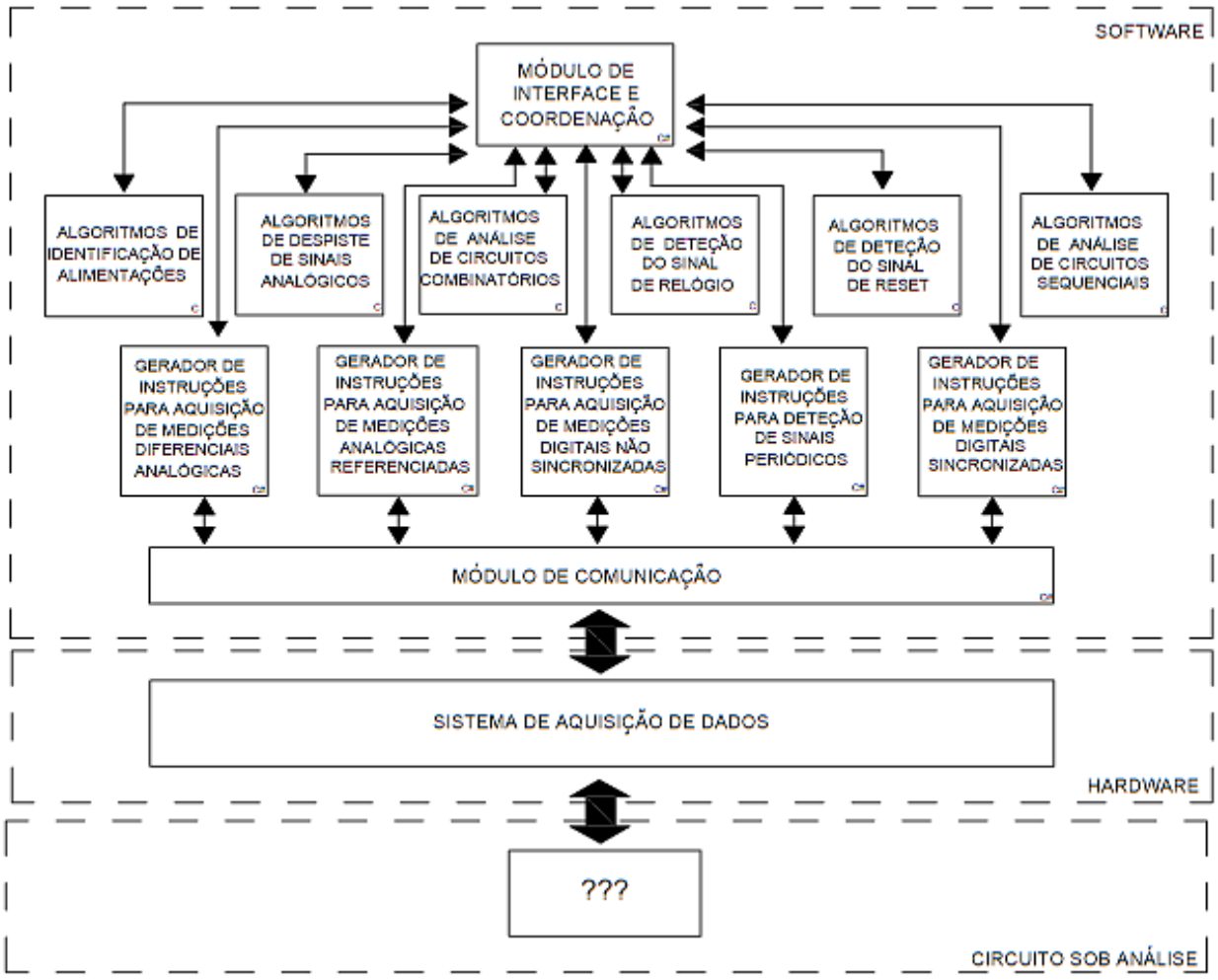

Figura 3.2 – Arquitetura modular do sistema de replicação de circuitos integrados digitais.

Nos módulos de interface com o utilizador e implementação de funcionalidades de comunicação com o sistema de aquisição de dados a linguagem utilizada foi o C# (C *Sharp*) de forma a beneficiar da orientação a objetos que esta linguagem disponibiliza. Nos módulos de análise e processamento massivo de dados foi utilizada a linguagem C, dado que o baixo nível de implementação utilizado nesta linguagem permite um controlo e gestão mais precisos, nomeadamente na gestão da memória.

A comunicação entre o *software* e o *hardware* foi implementada via porta COM (também designada de porta série) e foram desenvolvidos algoritmos para envio e receção de dados.

Nas subseções seguintes deste capítulo serão descritos os algoritmos implementados (*software*). No capítulo 4 será descrito o sistema de aquisição de dados (*hardware*).

3.3. Parâmetros de configuração inicial do sistema

O sistema implementado possui a capacidade de identificar e ajustar automaticamente vários parâmetros de operação sem intervenção do utilizador, mas existem parâmetros que têm necessariamente de ser definidos pelo utilizador no início do processo de replicação. Na Tabela 3.1 encontram-se apresentados e descritos os parâmetros configuráveis pelo utilizador.

Tabela 3.1 – Parâmetros de configuração inicial do sistema.

Parâmetro	Descrição
Diretoria de destino	Define a diretoria onde serão armazenados todos os ficheiros contendo dados temporários gerados pelo sistema, bem como a localização do ficheiro VHDL gerado.
Nome do ficheiro	Define o nome do ficheiro VHDL.
Número de pinos	Define o número de pinos do circuito a analisar.
Tolerância (V)	Define a tolerância máxima admissível em torno de uma medição (válido para medições analógicas).
Número de amostras digitais	Define o número de amostras a realizar em cada medição digital.
Número de amostras analógicas	Define o número de amostras a realizar em cada medição analógica.
Porta COM	No caso da existência de vários dispositivos ligados ao computador via porta COM o utilizador deve especificar qual das portas se encontra conectada ao sistema de aquisição de dados.

Estes dados são armazenados num ficheiro de configurações onde todos os módulos podem aceder e extrair as configurações necessárias para o seu funcionamento. A implementação das inicializações pode ser consultada no Anexo D (seção D.1).

3.4. Algoritmos de identificação dos pinos destinados a alimentação

3.4.1. Função do algoritmo

A primeira etapa do processo de replicação de um circuito integrado digital é detetar pinos cuja tensão seja sempre constante o que irá permitir determinar o pino que será utilizado como referência (GND) e remover os pinos com comportamento constante do restante processo de análise.

A tensão nos pinos destinados a alimentação carateriza-se por ser constante (desprezando as pequenas variações) e não tem influência no comportamento lógico do circuito. Da mesma forma, infere-se que os pinos onde a tensão é sempre constante, mesmo não sendo pinos destinados à alimentação do circuito, não interferem no comportamento lógico do circuito. Baseado neste princípio, os algoritmos desenvolvidos

nesta etapa classificam todos os pinos onde a tensão é constante como pinos de alimentação, ou seja, um pino de tensão constante não destinado a alimentação é classificado como pino de alimentação mesmo não o sendo no circuito em análise, pois a sua influência no comportamento lógico é similar a um pino de alimentação. Ao(s) pino(s) com a tensão(ões) constante(s) mais baixa(s) é atribuída a designação de referência – GND.

Para deteção dos pinos destinados a alimentação são utilizados dois módulos de *software,* de entre os módulos apresentados na Figura 3.2:

- Módulo gerador de instruções para aquisição de medições diferenciais analógicas;
- Módulo de algoritmos de identificação de alimentações.

3.4.2. Dados requeridos pelo algoritmo

A deteção dos pinos destinados a alimentação é efetuada com base na análise entre a relação da tensão entre cada par de pinos do circuito integrado sob análise (medições diferenciais). Considerando que a cada pino o circuito sob teste é atribuído um número e a tensão diferencial V_{XY} dada por

$$V_{XY} = V_X - V_Y \tag{3.1}$$

onde X e Y representam dois pinos do circuito sob teste, e sabendo que

$$V_{XY} = -V_{YX} \tag{3.2}$$

então para a execução do algoritmo de deteção de tensões constantes é necessário conhecer todas as tensões V_{XY} para todos os valores de X e Y definidos por

$$X \geq 1 \wedge Y \geq 2 \wedge Y > X \wedge X \leq (N-1) \wedge Y \leq N \tag{3.3}$$

onde N representa o número de pinos do circuito integrado.

As condições definidas em (3.3) representam todas as combinações dos valores de X e Y para os quais se obtêm todas as tensões diferenciais entre cada par possível de pinos, com o mínimo de combinações. O algoritmo utiliza sempre o menor número de combinações possíveis que garantam toda a informação, logo o número de medições necessárias para execução dos algoritmos de deteção de alimentações é dado por

$$Nº\ Medições = \sum_{i=1}^{N-1} M \tag{3.4}$$

onde M e N representam o número de amostras por cada par de pinos e o número de pinos do circuito, respetivamente.

3.4.3. Geração de instruções para aquisição de medições diferenciais analógicas

O módulo gerador de instruções para aquisição de medições diferenciais analógicas é o primeiro a ser executado e tem por objetivo analisar as configurações

iniciais e converter essa informação em instruções formatadas de forma que o módulo de comunicação seja capaz enviar essas mesmas instruções ao sistema de aquisição de dados e assim sejam obtidas as medições descritas na subsecção 3.4.2. Este módulo possui ainda a função de converter os dados recebidos do sistema de aquisição de dados e efetuar uma conversão para que as medições fiquem com uma organização compatível com as requeridas pelos algoritmos de identificação de alimentações. O fluxograma da Figura 3.3 ilustra o funcionamento geral deste módulo.

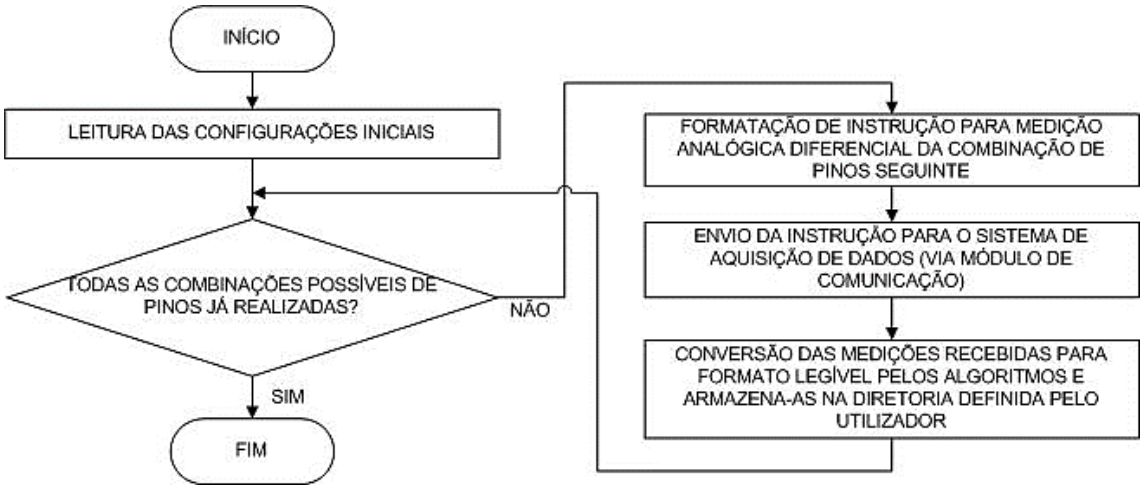

Figura 3.3 – Funcionamento do módulo gerador de instruções para aquisição de medições diferenciais analógicas.

A conversão dos valores recebidos (V_{REC}) consiste em aplicar um fator de correção de forma a recuperar o valor real da medição, V_{REAL}, de acordo com a expressão (3.5) e ajustes no encaminhamento do sinal detalhados no Anexo D (seção D.2).

$$V_{REAL} = (V_{REC} + F_{AJ}) \times F_{ADC} \times F_{SAD} \tag{3.5}$$

onde F_{AJ}, F_{ADC} e F_{SAD} representam o fator de ajuste, fator de correção do conversor analógico para digital e fator aplicado pelo circuito de aquisição de dados[13]. As medições são armazenadas em ficheiros cujo nome reflete a combinação de pinos das medições nele contidas (*measurementsPXPY.med*). O módulo dos algoritmos para deteção de alimentações utiliza a mesma metodologia para selecionar os ficheiros que contém os dados pretendidos, isto é, o nome completo do ficheiro é o fator identificativo que permite os módulos envolvidos na identificação de pinos de alimentação comunicar através de uma estrutura de dados vasta e baseada em ficheiros.

Em suma, o gerador de instruções para aquisição de medições diferenciais analógicas (desenvolvido na linguagem C#) é um módulo de ligação entre um módulo de análise e processamento (algoritmos de identificação de pinos com tensão constante) e o sistema de aquisição de dados. Este módulo foi desenvolvido na linguagem C# e o código fonte encontra-se disponível no Anexo D (seção D.3).

[13] No sistema implementado têm-se que $F_{AJ} = 2{,}5$, $F_{ADC} = 5/1024$ e $F_{SAD} = 6$. Mais detalhes sobre os fatores de conversão serão apresentados no capítulo 4 deste documento.

3.4.4. Análise e processamento dos dados

O princípio utilizado para detetar tensões constantes baseia-se no valor médio da diferença de tensão entre pinos, e na distribuição das amostras recolhidas durante o processo de medição. A Figura 3.4 mostra a distribuição típica das amostras em função de diferentes combinações de pinos.

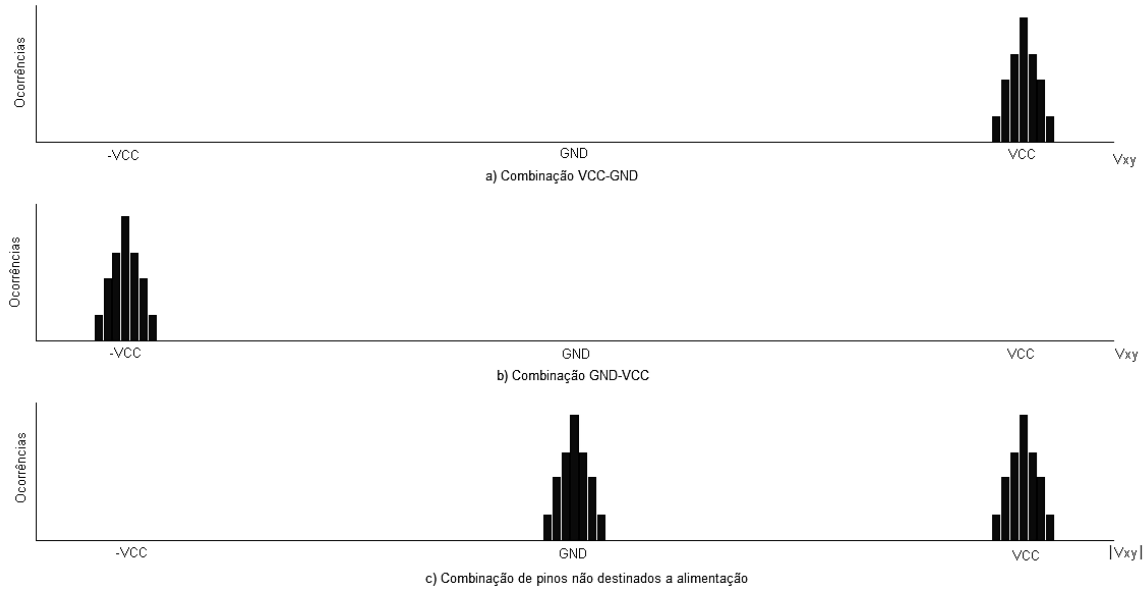

Figura 3.4 – Distribuição típica das amostras analógicas diferenciais para diferentes pares de pinos.

Pela análise da Figura 3.4 é possível verificar que quando são efetuadas medições a um par de pinos destinados à alimentação do circuito a concentração de amostras é em torno de um único valor (±VCC), ou seja, nesta situação temos

$$V_{MÉDIO} \approx \pm VCC \tag{3.6}$$

onde $V_{MÉDIO}$ representa o valor médio das amostras recolhidas para a combinação de pinos em análise. Se considerarmos o caso onde pelo menos um dos pinos da combinação medida não se destina a alimentação (ou não for constante), então o valor médio das amostras é dado por.

$$V_{MÉDIO} \approx \pm \frac{VCC}{2} \tag{3.7}$$

considerando que o circuito analisado é digital.

Pelas análises que conduziram às equações (3.6) e (3.7) é possível inferir que perante medições de combinações de pinos destinados a alimentação (ou constantes) as amostras se localizam em torno do valor médio da diferença de tensão entre os pinos. Assim sendo, para que uma combinação de pinos seja classificada pelo algoritmo como pinos destinados a alimentação, todas as amostras da medição devem respeitar a condição

$$V_{MÉDIO} - \Delta v \ \leq \ V_i \leq \ V_{MÉDIO} + \Delta v, \qquad i = \{0, 1, \dots, M\} \tag{3.8}$$

onde $V_{MÉDIO}$ representa a diferença de tensão média entre o par de pinos em análise, Δv é o desvio máximo permitido e M o número de amostras disponíveis, ou seja, quando

analisados dois pinos destinados a alimentação todas as amostras devem estar contidas no intervalo $V_{médio} \pm \Delta v$.

O desvio máximo permitido, Δv, é definido pelo utilizador (parâmetro "Tolerância" da Tabela 3.1) e guardado nas configurações iniciais. Para que o algoritmo funcione corretamente, o valor de Δv definido pelo utilizador deve respeitar a expressão

$$\Delta v < \left|\frac{VCC}{2}\right| \tag{3.9}$$

para um bom funcionamento do algoritmo e considerando valor típicos para o VCC.

O algoritmo desenvolvido aplica o princípio de funcionamento descrito nesta seção: calcula o valor médio das amostras da combinação de pinos em análise e compara com o intervalo $V_{MÉDIO} \pm \Delta v$. O algoritmo determina ainda qual a tolerância mínima que pode ser utilizada sem afetação dos resultados (para utilização futura) bem como normaliza os resultados no final do procedimento. O fluxograma da Figura 3.5 sintetiza o funcionamento do algoritmo de deteção de alimentações implementado.

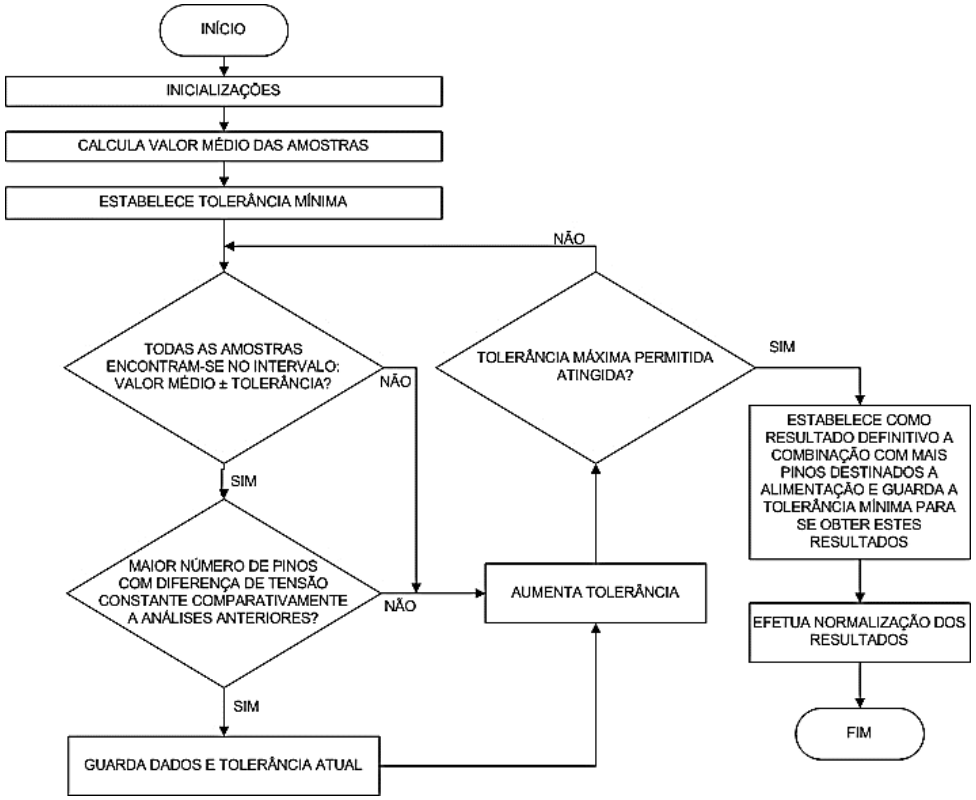

Figura 3.5 – Funcionamento geral do algoritmo de deteção de pinos destinados à alimentação do circuito ou de tensão constante.

As inicializações efetuadas pelo algoritmo consistem na leitura das configurações iniciais estabelecidas pelo utilizador, de forma a conhecer que tipo de informação deve procurar na estrutura de dados para análise.

O cálculo do valor médio é realizado para cada combinação de pinos, após a leitura das medições de um ficheiro, correspondente à respetiva combinação. O cálculo do valor médio para cada combinação de pinos é realizado de acordo com a expressão

$$V_{MÉDIO} = \frac{1}{M}\sum_{i=1}^{M} V_i \tag{3.10}$$

onde M e V_i representam o número de amostras e o valor da amostra obtida em i-gésimo lugar, respetivamente.

As amostras são armazenadas num vetor, calculado o valor médio das amostras e posteriormente verificado se cada amostra pertence ao intervalo $V_{MÉDIO} \pm \Delta v$. Quando todas as amostras se encontram nesse intervalo, podem ocorrer as situações apresentadas na Tabela 3.2, considerando que os pinos denominados por X e Y originam a medição descrita na equação (3.1).

Tabela 3.2 – Classificação dos pinos em função do valor médio.

V_{XY}	Situação
$V_{MÉDIO} < 0$	A tensão no pino Y é superior à tensão no pino X. Os pinos X e Y podem ser classificados como GND e VCC, respetivamente, ou em alternativa –VCC e GND[14], respetivamente.
$V_{MÉDIO} \approx 0$	A tensão nos pinos X e Y são aproximadamente iguais, isto é, correspondem ao mesmo sinal. Assim sendo, se algum dos pinos já foi previamente classificado como VCC ou GND essa classificação é estendida ao pino não classificado, caso contrário nada se pode concluir e nenhuma classificação é efetuada.
$V_{MÉDIO} > 0$	A tensão no pino X é superior à tensão no pino Y. Os pinos X e Y podem ser classificados como VCC e GND, respetivamente, ou em alternativa GND e –VCC[14], respetivamente.

O processo é repetido para diferentes valores de Δv, começando num valor próximo de zero e que é progressivamente incrementado até ao valor máximo definido pelo utilizador. Quando é verificado que para um valor diferente de Δv é obtido um maior número de pinos com tensão constante os dados anteriores são descartados e armazenado o novo resultado. Quando o valor máximo de Δv é atingido e a respetiva análise é efetuada é procurado o menor valor de Δv para o qual se obtêm resultados iguais aos obtidos com Δv máximo, sendo esse o valor mínimo de Δv para o qual os resultados são fiáveis. Essa informação é adicionada ao ficheiro de configurações para utilização futura.

Durante o processo de análise cada pino é analisado várias vezes, devido à comparação com diferentes pinos, de forma que a ordem de análise pode influenciar o resultado, pois quando um pino é definido como GND toda a análise posterior é referenciada ao GND previamente detetado. Acontece que em circuitos com múltiplas tensões constantes (vamos considerar V_1 e V_2, sempre com $V_2 > V_1 > GND$) se uma primeira análise coincidir com a análise da relação entre V_1 e V_2 então V_1 será considerado com GND e posteriormente quando o GND for analisado será definido como uma tensão constante negativa (-VCC). De forma a uniformizar o resultado obtido no final da análise é realizada uma normalização dos resultados, que consiste em encontrar a média mais baixa das tensões constantes e adicionar esse valor à média das restantes tensões constantes, ou seja, transformar a tensão constante mais baixa no GND, garantindo assim a operação na gama das tensões positivas.

Em suma, o resultado final obtido ao nível de pinos com tensões constantes e a respetiva atribuição da classificação VCC e GND pode não ser igual à estipulada pelo

[14] Esta classificação alternativa apenas é atribuída se em alguma análise anterior o pino de maior tensão da análise atual já tenha sido classificado como GND.

fabricante do circuito integrado em análise, mas é equivalente e capaz de fornecer a referência para o sistema de replicação poder progredir para as etapas seguintes.

O código fonte (linguagem C) utilizado para implementar os algoritmos de deteção de pinos destinados a alimentação encontra-se listado no Anexo D (seção D.4).

3.4.5. Sensibilidade a erros

Durante a implementação dos algoritmos para identificação de pinos com tensões constantes ou dedicadas à alimentação é necessário ter em conta a sensibilidade a erros, nomeadamente erros de medição.

Um pino que é classificado como destinado à alimentação é ignorado por todas as etapas de análise consequentes. Assim sendo, uma classificação errada nesta fase pode implicar uma replicação incorreta do circuito integrado em análise, pelo que a gestão de possíveis erros de medição deve ser realizada cautelosamente.

Neste módulo, existem duas formas de gerir os possíveis erros de medição:

- Eliminação das amostras consideradas como erros de medição através de análise estatística (distribuição de frequência) [44];
- Diminuir a sensibilidade do algoritmo de forma a integrar medições menos precisas sem afetar a análise do algoritmo.

A eliminação de amostras consideradas como erros de medição através de uma função de densidade de probabilidade permitiria identificar as medições que tem uma ocorrência residual em relação às restantes amostras e descartá-las da análise. Considerando que é possível que existam pinos que se encontram maioritariamente num estado e com variações esporádicas para um outro estado durante um período curto de tempo, como por exemplo um sinal de *reset*; a aplicação deste método de gestão de erros de medição poderia conduzir à identificação de falsos erros de medição em algumas amostras, que sendo descartadas originariam a classificação incorreta de pinos destinados à alimentação.

Consequentemente, a forma mais viável de despistar erros de medição é diminuir a sensibilidade do algoritmo através do aumento da tolerância máxima, Δv. Esta técnica não evita que erros de medição superiores à tolerância máxima provoquem erros na análise. No entanto, enquanto na aplicação de uma função de densidade de probabilidade amostras duvidosas seriam ignoradas e mais pinos com tensão constante iriam ser erradamente detetados (podendo provocar a perda do sinal de *reset*, por exemplo), ao diminuir a sensibilidade irá ocorrer a situação inversa, isto é, menos pinos com tensão constante serão detetados. Porém, nesta última situação desde que existam outros pinos com tensões constantes o algoritmo seria capaz de definir outra referência (GND) para o circuito e continuar sem afetar a possibilidade de um resultado equivalente, pois em fases mais avançadas novas medições serão recolhidas e o(s) erro(s) de medição nesta fase não serão críticos. Tendo em conta as implicações de cada um dos dois métodos de gestão de erros apresentados, conclui-se que neste módulo a diminuição da sensibilidade é a forma mais adequada de efetuar o despiste de erros de medição sem afetação significativa do resultado final.

Quando um circuito integrado analógico é submetido a análise os pinos de alimentação serão igualmente identificados, pois respeitam a distribuição apresentada

na Figura 3.4. Os pinos analógicos não destinados a alimentação como não apresentam distribuição uniforme em torno de um valor médio não seriam identificados como pinos destinados a alimentação, à semelhança do que acontece com circuitos digitais, sendo analisados em etapas seguintes para extração do comportamento do circuito. Em suma, este módulo é capaz de detetar pinos com tensão constante em qualquer circuito integrado, analógico ou digital.

3.5. Algoritmos de identificação de sinais analógicos

3.5.1. Função do algoritmo

O módulo de identificação de sinais analógicos têm por objetivo verificar se existe algum sinal analógico em algum dos pinos do circuito integrado, isto é, se algum dos sinais de entrada ou saída do circuito sob análise apresenta mais do que dois níveis de tensão distintos. Caso seja detetado um sinal analógico, o utilizador recebe uma mensagem informativa que o circuito a replicar apresenta um sinal analógico em algum dos pinos de entrada ou saída, e irá efetuar uma aproximação a um sistema digital. Como o sistema de replicação de circuitos implementado foi concebido para circuitos puramente digitais, quando a mensagem de deteção de sinal analógico é apresentada o utilizador é informado que o resultado da replicação obtida é uma aproximação e não garante fiabilidade. A replicação não é interrompida quando é detetado um sinal analógico, porque os parâmetros de configuração iniciais inseridos pelo utilizador influenciam esta etapa e em caso de inserção de parâmetros inadequados o sistema pode reajustar automaticamente os parâmetros e obter uma réplica digital. Porém, a avaliação da fiabilidade da réplica obtida após mensagem informativa de deteção de sinais analógicos é sempre da responsabilidade do utilizador.

Esta etapa do circuito pode ser considerada como uma etapa de triagem, ainda assim não é uma etapa impeditiva da continuação do algoritmo em caso de deteção de sinal analógico. Nessas situações o resultado final não tem fiabilidade garantida, sendo da responsabilidade do utilizador a avaliação da réplica obtida. Para deteção de sinais analógicos são utilizados dois módulos de *software* de entre os módulos apresentados na Figura 3.2:

- Módulo gerador de instruções para aquisição de medições analógicas referenciadas;
- Módulo de algoritmos de despiste de sinais analógicos.

3.5.2. Dados requeridos pelo algoritmo

A deteção de sinais analógicos é realizada com base em medições analógicas de cada pino onde a referência das medições é o pino definido como referência (GND) pelo algoritmo de deteção de sinais destinados a alimentação. As medições necessárias para execução deste algoritmo podem ser descritas pela expressão

$$V = V_i - V_{REF} \approx V_i, \qquad i \in \mathbb{N} \wedge i \leq N \tag{3.11}$$

onde i, N e V_{REF} representam o número atribuído ao pino, o número de pinos do circuito integrado e a tensão no pino de referência, respetivamente.

Para esta etapa o número de medições necessárias é dado por

$$Nº\ Medições = \sum_{i=1}^{N-R} M \tag{3.12}$$

onde N, M e R representam o número de pinos, o número de amostras por cada pino e o número de pinos destinados a alimentação, respetivamente.

3.5.3. Geração de instruções para aquisição de medições analógicas referenciadas

O módulo gerador de instruções para aquisição de medições analógicas referenciadas tem por objetivo analisar as configurações iniciais e classificação atribuída pelo algoritmo de deteção de sinais destinados a alimentação e converter essa informação em instruções formatadas de forma que o módulo de comunicação seja capaz enviar essas mesmas instruções ao sistema de aquisição de dados e assim sejam obtidas as medições descritas na subsecção 3.5.2. À semelhança do módulo definido na subseção 3.4.3 este módulo possui ainda o função de converter os dados recebidos do sistema de aquisição de dados e efetuar uma conversão para que as medições fiquem com uma organização compatível com as requeridas pelos algoritmos de despiste de sinais analógicos. O fluxograma da Figura 3.6 ilustra o funcionamento geral deste módulo.

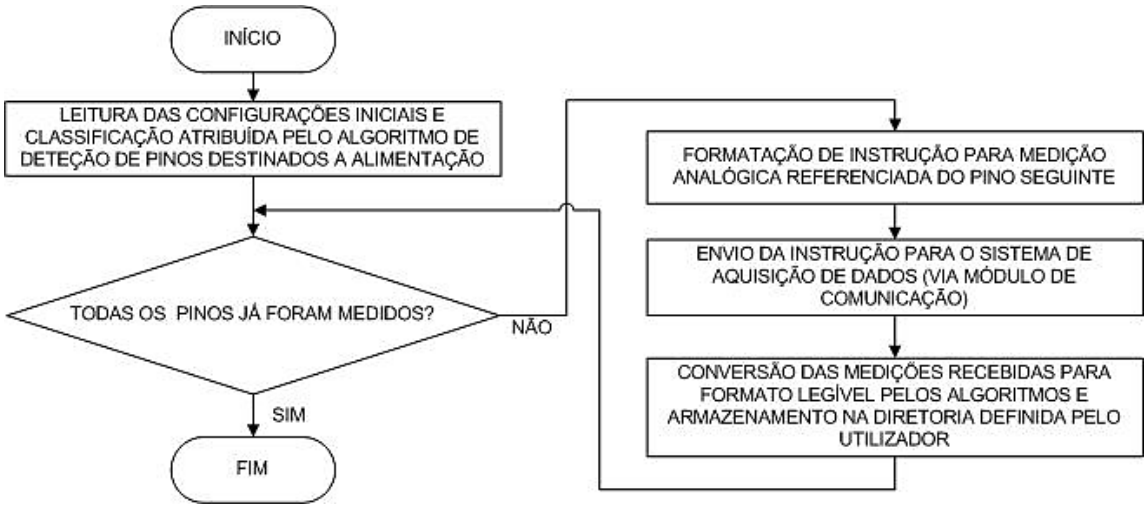

Figura 3.6 – Funcionamento do módulo gerador de instruções para aquisição de medições analógicas referenciadas.

Este módulo é muito similar ao módulo gerador de instruções para a aquisição de medidas analógicas diferenciais, diferindo unicamente no facto deste módulo definir automaticamente a referência, ou seja, este módulo compõe instruções para realização de medições diferenciais entre dois pinos onde um dos pinos é sempre o pino de referência (na etapa anterior ambos os pinos eram variáveis), sendo proveniente desse facto a designação de medições analógicas referenciadas. A utilização de um módulo independente ao invés do reaproveitamento do módulo gerador de instruções para

medições analógicas já existente tem por objetivo diminuir a dependência de diferentes módulos em torno de um único módulo, de forma a facilitar uma futura alteração/otimização na forma de detetar sinais analógicos, reaproveitando a arquitetura já implementada.

Considerando que o sistema de aquisição de dados utilizado é o mesmo que na etapa anterior, o fator de conversão das medições obtidas é igualmente definido pela equação (3.5). As medições são igualmente armazenadas em ficheiros cujo nome reflete o pino das medições nele contidas (*measurementsPX.med*) e o nome do ficheiro é o fator identificativo na estrutura de dados baseada em ficheiros, à semelhança da descrição apresentada na subseção 3.4.3.

Em suma, o gerador de instruções para aquisição de medições analógicas referenciadas é um módulo de ligação entre um módulo de análise e processamento (algoritmos de despiste de sinais analógicos) e o sistema de aquisição de dados. Este módulo foi desenvolvido na linguagem C# e o código fonte encontra-se disponível no Anexo D (seção D.3).

3.5.4. Análise e processamento dos dados

A análise dos níveis de tensão em cada pino consiste de forma genérica na verificação se os pinos possuem no máximo dois níveis de tensão. Em função dessa análise é possível inferir se o circuito é completamente digital/analógico ou misto. O funcionamento geral do algoritmo encontra-se sintetizado no fluxograma da Figura 3.7.

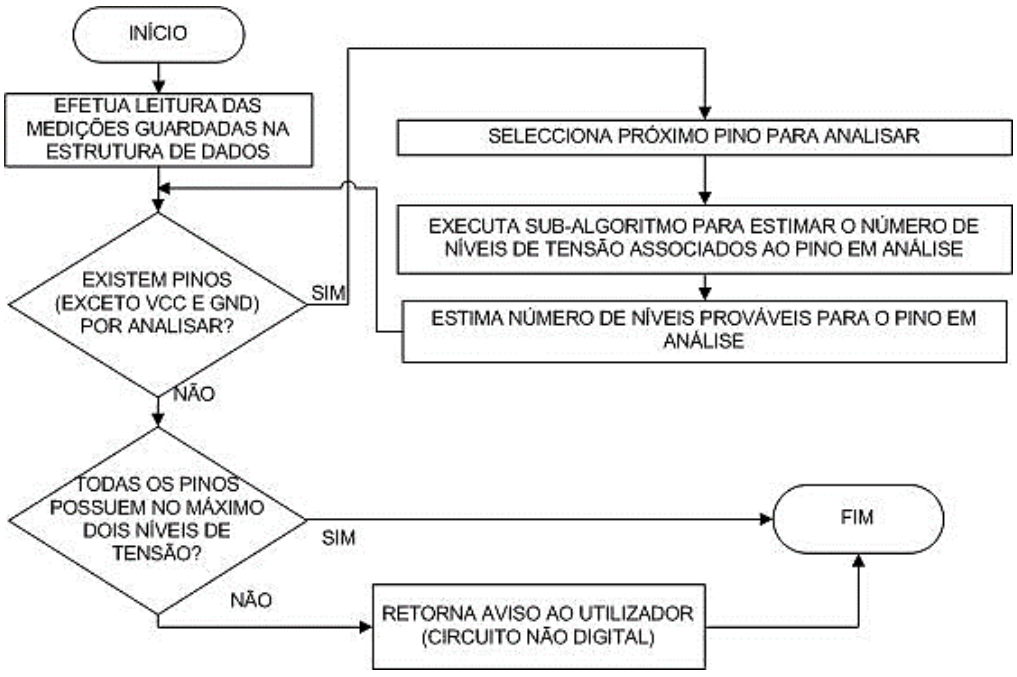

Figura 3.7 - Funcionamento geral do algoritmo de análise dos níveis de tensão em cada pino.

O fluxograma apresentado na Figura 3.7 representa um funcionamento geral de coordenação e decisão, sendo a real análise dos níveis efetuada num sub-algoritmo. Durante a execução dos algoritmos para despiste de sinais analógicos é aplicada uma análise estatística (função de distribuição de frequência [44]) de forma a detetar eventuais erros de medição que possam originar falsos níveis.

A análise do número de níveis consiste em percorrer todas as amostras recolhidas para um pino e detetar vários conjuntos de amostras agrupados em intervalos do tipo $V_{NÍVEL(i)} \pm \Delta v$ perfeitamente definidos, onde $V_{NÍVEL(i)}$ representa o valor médio da tensão do nível i e Δv representa o desvio máximo permitido (igual ao desvio máximo utilizado na deteção de pinos destinados a alimentação). Para cada pino são armazenados os valores médios de cada nível e o número de amostras encontrado para cada nível e sendo utilizado o número de amostras contabilizado para inferir da fiabilidade do nível. O sub-algoritmo que realiza a análise do número de níveis de cada pino pode ser sintetizado pelo fluxograma da Figura 3.8:

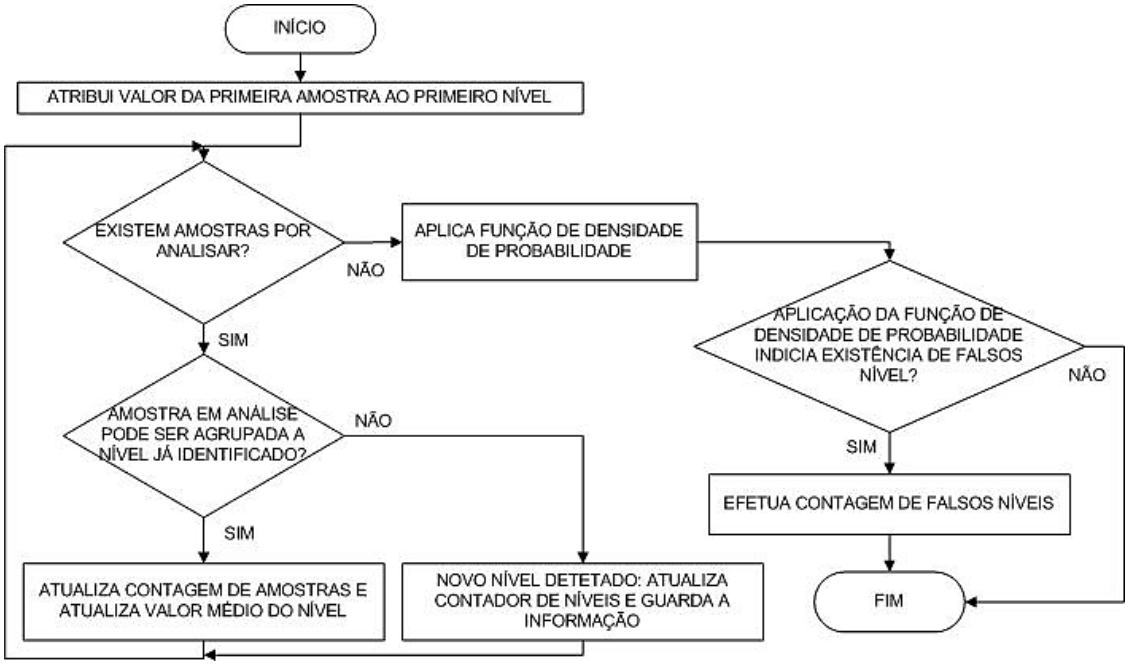

Figura 3.8 – Funcionamento do sub-algoritmo de análise dos níveis de tensão em cada pino.

A deteção de níveis é realizada baseado no princípio do agrupamento em torno de um valor médio. No entanto, a abordagem a utilizar nesta etapa tem de ser diferente da utilizada na deteção de tensões constantes pois nesta fase pretende-se determinar de vários níveis de tensão em vez de um único valor constante. Nesta etapa quando o algoritmo analisa as amostras relativas a um novo pino efetua os seguintes procedimentos: atribui o valor da primeira amostra ao valor médio do primeiro nível e posteriormente compara as amostras com os níveis já identificados anteriormente; caso a amostra em análise se enquadre no intervalo $V_{MÉDIO\ (NÍVEL_i)} \pm \Delta v$ de algum dos níveis já identificados a amostra é agrupada a esse nível, caso contrário é designado um novo nível a cujo valor médio é atribuído o valor da amostra em análise. Quando uma nova amostra é associada a um nível já existente é efetuada uma atualização do valor médio e da contagem de amostras associadas a esse nível.

Ao valor médio das amostras associadas a um nível é inicialmente atribuído o valor da primeira amostra e é atualizado cada vez que uma nova amostra é agrupada ao nível, de forma que a influência da ordem de análise das amostras seja minimizada. A atualização do valor médio de um nível é calculado a partir da expressão

$$\overline{V_k} = \frac{\overline{V_{k-1}} \times (k-1) + V_k}{k} \tag{3.13}$$

onde k representa o número de amostras associado ao nível, V_k é a última amostra associada ao nível, $\overline{V_{k-1}}$ e $\overline{V_k}$ representam a média antes e após a associação da última amostra, respetivamente.

O despiste de falsos níveis de tensão devido a possíveis erros de medição é realizado através da aplicação de uma função de distribuição de frequência que pode ser representada por [44]

$$\sum_{i=1}^{N}(x_i, y_i) \tag{3.14}$$

onde N, x_i e y_i representam o número de pinos sujeitos a análise, o valor médio do nível i e o número de amostras associadas ao nível i, respetivamente. É definido um número mínimo de amostras (em função do número total de amostras) associadas a um nível, que deve ser atingido, caso contrário esse nível é considerado um falso nível. Foi aplicada uma análise de frequência em termos absolutos pois o número de cálculos necessários era menor comparativamente a uma análise em termos relativos, sem prejuízo dos resultados obtidos. Assim sendo, um nível é considerado falso quando

$$Nº\ amostras < \frac{k \times M}{100}, \qquad (0 < k < 100) \tag{3.15}$$

sendo k um valor percentual e M o número total de amostras associadas ao pino em análise[15]. Na Figura 3.9 encontra-se ilustrado um exemplo da distribuição de frequência associado a um pino, onde pode ser constatada a existência de dois falsos níveis (X_2 e X_3) dado que se encontram abaixo do limiar (a tracejado) definido pela equação (3.15).

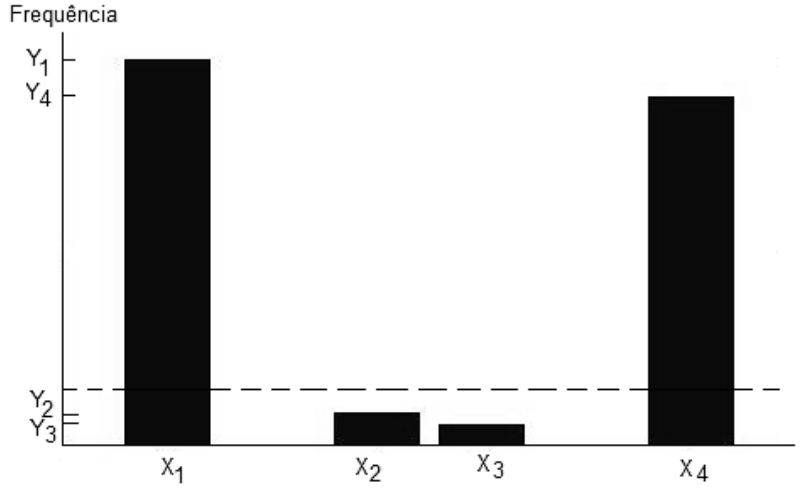

Figura 3.9 – Exemplo de distribuição de frequência associado a um pino.

Aplicada a análise da distribuição de frequência o algoritmo já dispõe de informação sobre os níveis associados a cada pino, nomeadamente o número total de níveis e falsos níveis. Assim sendo, o número provável de níveis reais,N_p, é dado por

$$N_P = N_T - N_F \tag{3.16}$$

sendo N_T e N_F o número de total de níveis e o número de falsos níveis, respetivamente.

[15] Na implementação do algoritmo utilizado o valor $k = 10$, ou seja, um nível é considerado falso quando tem associadas menos de 10% das amostras relativas ao pino em análise.

Com o número de níveis prováveis para cada pino o algoritmo atribui ao circuito integrado em análise uma das seguintes classificações:

- Circuito puramente digital, se nenhum pino possuir mais do que dois níveis de tensão;
- Circuito analógico, se todos os pinos analisados possuírem mais do que dois níveis tensão;
- Circuito misto, se alguns pinos com mais e menos do que dois níveis de tensão, simultaneamente.

Quando o circuito sob análise é classificado pelo algoritmo como analógico ou misto é enviado ao utilizador uma mensagem informativa que o circuito em análise não é completamente digital, e o resultado produzido pelo sistema será uma aproximação a um circuito digital, que pode não corresponder ao circuito integrado analisado.

O código fonte utilizado para implementação dos algoritmos de despiste de sinais analógicos encontra-se listado no Anexo D (seção D.5).

3.6. Algoritmos de análise de circuitos combinatórios

3.6.1. Função do algoritmo

Após detetados os pinos com tensão constante e realizado o despiste de sinais analógicos a etapa seguinte consiste na aquisição de dados e análise dos mesmos para efeitos de extração do comportamento do circuito integrado em análise.

Os algoritmos de análise de circuitos combinatórios realizam uma primeira análise com o objetivo de verificar se uma abordagem combinatória permite obter uma boa aproximação ao circuito sob teste. Se os algoritmos inferirem que uma análise combinatória permite descrever o comportamento do circuito então, com os resultados da análise efetuada, será automaticamente gerado um ficheiro na linguagem VHDL com a descrição comportamental do circuito e o processo de replicação termina, caso contrário é retornada a indicação que é necessária uma nova abordagem (abordagem sequencial) e o processo de replicação prossegue para as etapas subsequentes.

Para análise combinatória do circuito sob teste são utilizados dois módulos de *software* de entre os módulos apresentados na Figura 3.2:

- Módulo gerador de instruções para aquisição de medições digitais não sincronizadas;
- Módulo de algoritmos de análise de circuitos combinatórios.

3.6.2. Dados requeridos pelo algoritmo

A análise combinatória é executada com base em medições da tensão dos pinos que influenciam o comportamento do circuito, ou seja, os pinos de tensão não constante. Nesta etapa as medições são requeridas num formato digital (0/1) ao invés do formato analógico, uma vez que num circuito digital combinatório apenas são considerados dois níveis de tensão.

Para um correto funcionamento do algoritmo de análise, é essencial que as medições utilizadas na análise sejam referentes ao mesmo instante temporal, isto é, os dados requeridos pelo algoritmo de análise combinatória consistem nas tensões de todos os pinos de interesse nos instantes $t_1, t_2, t_3, \dots, t_M$ onde M representa o número de amostras definidas pelo utilizador. A precisão do algoritmo é severamente afetada se os dados para análise não respeitarem a condição temporal, pois o algoritmo irá considerar, erradamente, relações entre entradas e saídas que podem não estar relacionadas devido a um desfasamento temporal. As medições nesta fase são realizadas assincronamente de forma a evitar a medições que permitam análises incorretas de sinais periódicos e visam obter todas as combinações ao longo do tempo.

O número de medições necessárias para a execução deste algoritmo é dada por

$$Nº\ Medições = \sum_{i=1}^{N-R} M \tag{3.17}$$

onde N, M e R representam o número de pinos, o número de amostras por cada pino e o número de pinos destinados a alimentação, respetivamente.

3.6.3. Geração de instruções para aquisição de medições digitais não sincronizadas

O módulo gerador de instruções para aquisição de medições digitais não sincronizadas tem por objetivo analisar as configurações iniciais e classificação atribuída pelo algoritmo de deteção de sinais destinados a alimentação e converter essa informação em instruções formatadas de forma que o módulo de comunicação seja capaz de enviar essas mesmas instruções ao sistema de aquisição de dados e assim sejam obtidas as medições descritas na subsecção 3.6.2. À semelhança dos módulos definidos nas subseções 3.4.3 e 3.5.3, este módulo possui ainda a função de converter os dados recebidos para que as medições fiquem com uma organização compatível com as requeridas pelos algoritmos de análise combinatória. O fluxograma da Figura 3.10 ilustra o funcionamento geral deste módulo.

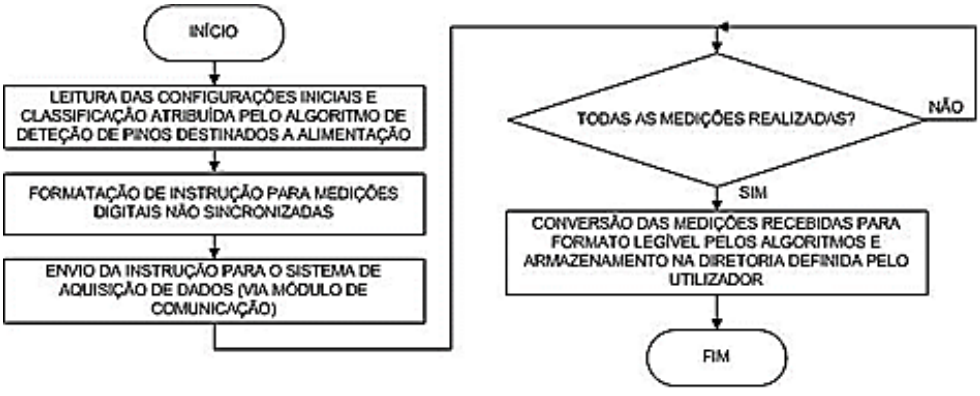

Figura 3.10 – Funcionamento do módulo gerador de instruções para aquisição de medições digitais não sincronizadas.

De forma a minimizar o número de comunicações entre o módulo de comunicação e o sistema de aquisição de dados, este módulo encontra-se preparado para receber as medições de todos os pinos, referentes ao mesmo instante temporal,

numa única palavra contendo a sequência ordenada de zeros e uns, correspondentes à ordem de leitura dos pinos. A conversão aplicada pelo módulo consiste em separar da palavra recebida o valor correspondente a cada pino e acrescentando-o no ficheiro de dados correspondentes a cada um dos pinos. Após a conversão da palavra recebida para um valor inteiro equivalente, o valor de cada pino é obtido através da expressão

$$V_i = Int\left(Resto\left(\frac{V_{INT}}{10^i}\right) \times \frac{1}{10^{i-1}}\right), \;\; 1 \leq i \leq N \tag{3.18}$$

onde N, i, V_i e V_{INT} representam o número de pinos, o número do pino, a tensão no pino i e valor inteiro equivalente à palavra recebida, respetivamente. As funções $Int()$ e $Resto()$ representam o valor inteiro e resto de uma divisão, respetivamente.

Para melhor compreensão da aplicação da equação (3.18) no processo de conversão, na Tabela 3.3 encontra-se exemplificado um caso de conversão de palavras recebidas de um sistema com três pinos.

Tabela 3.3 – Exemplo de conversão de dados recebidos de um sistema com três pinos.

Equivalente inteiro	V_3	V_2	V_1
0	0	0	0
1	0	0	1
10	0	1	0
11	0	1	1
100	1	0	0
101	1	0	1
110	1	1	0
111	1	1	1

Os valores convertidos encontram-se no formato requerido pelos algoritmos de análise combinatória e são armazenados em ficheiros. Quando o número de pinos do circuito integrado em análise é inferior ao número de pinos que o sistema de dados admite é necessário aplicar ajustes no nome dos ficheiros, à semelhança com os processos descritos no Anexo D.2 O código fonte utilizado para implementação do módulo gerador de instruções para aquisição de medições digitais não sincronizadas encontra-se apresentada no Anexo D (seção D.6).

3.6.4. Análise e processamento dos dados

A análise combinatória realizada aos dados disponíveis tem por objetivo verificar se para cada combinação dos sinais de entrada ocorre uma única saída. Se esta condição é verificada então é escrito o código VHDL que descreve o funcionamento do circuito, caso contrário esta etapa é abortada e são iniciados os procedimentos para analisar o circuito sob o ponto de vista de lógica sequencial.

A abordagem do algoritmo de análise combinatória consiste em efetuar a leitura dos dados de ficheiros e respetivo armazenamento em nova estrutura de dados de maior facilidade de acesso e exclusão das medições repetidas, com o objetivo de construir uma tabela de verdade descritiva do comportamento do circuito. Após a construção da tabela de verdade é realizada uma análise para identificação de circuitos

sequenciais (pela verificação da singularidade da combinação de saída para a mesma combinação de entrada). O funcionamento do algoritmo pode ser sintetizado pelo fluxograma apresentado na Figura 3.11.

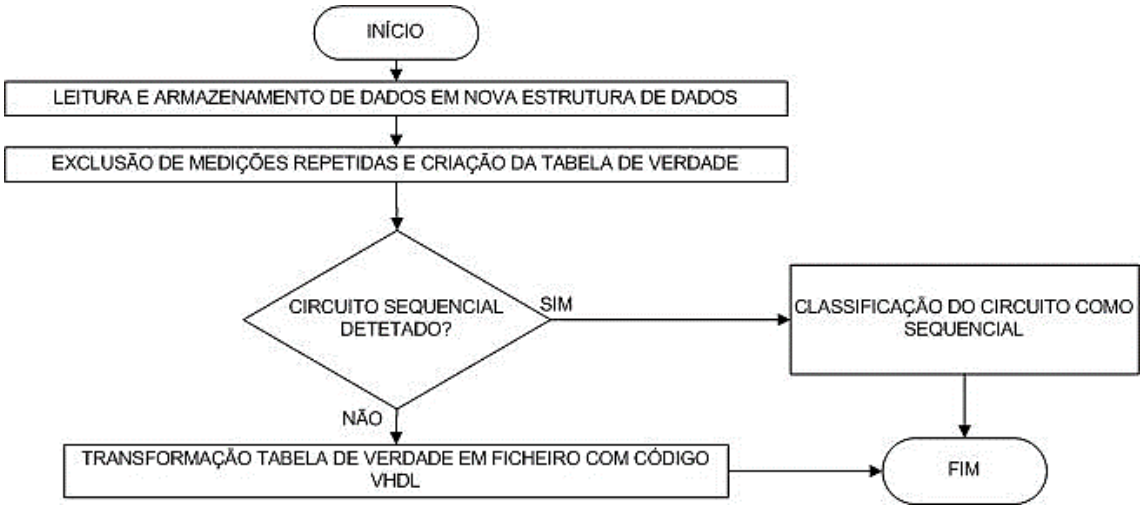

Figura 3.11 – Funcionamento geral do algoritmo de análise combinatória.

O código fonte completo, em linguagem C, utilizado na implementação do algoritmo de análise combinatória encontra-se listado no Anexo D.7.

3.6.4.1. Estruturas de dados utilizada

Inicialmente os dados encontram-se armazenados em ficheiros, porém durante a execução do algoritmo estes devem ser guardados numa estrutura de dados local do algoritmo de forma a aumentar a rapidez no acesso aos dados, sendo necessária a leitura dos ficheiros no início da execução do algoritmo de análise.

No algoritmo de análise combinatória os principais dados a armazenar são as medições digitais não sincronizadas efetuadas ao circuito e o pino a que se encontram associadas. Os dados são guardados num vetor bidimensional (matriz) de dimensões $N \times M$, onde N e M representam o número de pinos do circuito e o número de amostras utilizadas na análise, respetivamente. Graficamente, a estrutura de dados pode ser representada pela matriz apresentada na Figura 3.12.

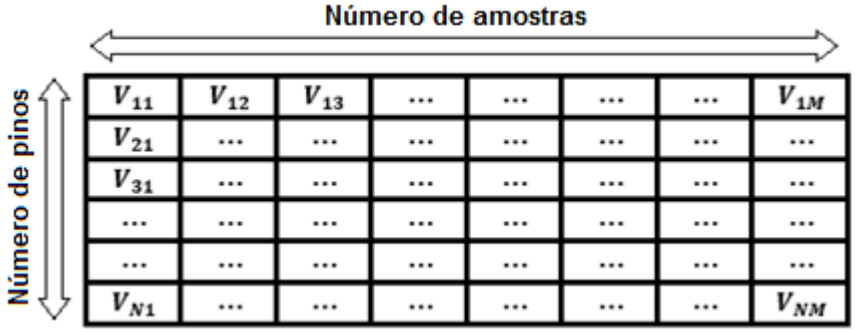

Figura 3.12 – Estrutura de dados para armazenamento das medições.

Com a estrutura de dados apresentada na Figura 3.12 cada coluna da matriz armazena as amostras, referentes a um instante temporal, dos N pinos do circuito em

análise. Cada linha da matriz armazena as M amostras obtidas referentes ao pino cujo número é igual ao número da linha da matriz. Esta organização dos dados permite um rápido acesso a qualquer medição ou conjunto de medições. No entanto, a matriz de armazenamento das medições não contém toda a informação necessária para construção de uma tabela de verdade pois não fornece informação acerca dos pinos de entrada e saída do circuito em análise. Para esse efeito é utilizado um vetor de comprimento N, onde em cada posição se encontra armazenada a informação sobre o tipo de pino e onde o índice do vetor indica a que pino se refere a informação contida na posição. Nesta fase do processo de replicação é possível encontrar quatro tipos de pino: entrada, saída, VCC ou GND. A Figura 3.13 ilustra graficamente a estrutura de dados que guarda informação sobre o tipo de cada pino, onde o valor apresentado entre parenteses corresponde ao índice do vetor e também ao pino a que se refere a informação contida na respetiva posição.

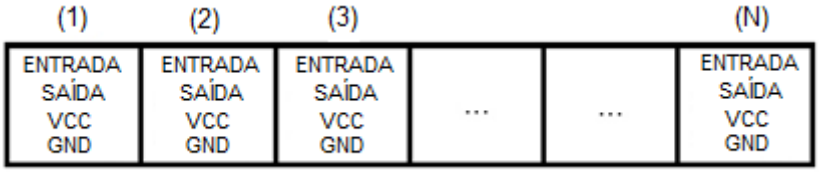

Figura 3.13 – Estrutura de dados para armazenamento de informação sobre o tipo de pinos.

Pelo cruzamento dos dados contidos nas estruturas de dados apresentadas na Figuras 3.11 e 3.12, é possível saber exatamente que conjuntos de medições correspondem a entradas ou saídas, pois cada posição do vetor apresentado na Figura 3.13 indica o tipo de dados contido na linha de igual índice da matriz da Figura 3.12.

3.6.4.2. Construção da tabela de verdade

A construção da tabela de verdade do circuito em análise consiste, em grande parte, na eliminação de medições repetidas. Como cada coluna da matriz da Figura 3.12 armazena um conjunto de medições referentes ao mesmo instante temporal, a eliminação de medições repetidas é realizada através da remoção das colunas repetidas na matriz. Sem medições repetidas, a tabela de verdade pode ser obtida selecionando os dados pelas linhas correspondentes a entradas e saídas do circuito. A tabela de verdade obtida apenas irá conter as combinações observadas, não garantindo a cobertura de todas as combinações de entrada possíveis.

3.6.4.3. Deteção de lógica sequencial

O despiste de lógica sequencial consiste em verificar se na tabela de verdade nenhuma combinação de entrada se encontra repetida. Por outras palavras, quando uma combinação de entrada se encontra repetida significa que existe uma combinação de saída diferente para cada uma das combinações de entrada repetidas encontradas, caso contrário apenas uma das medições teria sido mantida e a(s) restante(s) eliminada(s) anteriormente. Quando esta situação é detetada, a análise combinatória é terminada e retornada a indicação que foi detetada lógica sequencial, sendo essa

indicação interpretada pelo módulo de coordenação que irá iniciar os algoritmos para análise sequencial.

Após a confirmação do circuito como sendo um circuito digital combinatório apenas é necessária a conversão do conteúdo da tabela de verdade para linguagem VHDL, assente em uma estrutura de código sintetizável por um compilador já existente.

3.6.5. Estrutura do código (linguagem VHDL) da réplica

Conhecida a tabela de verdade de um circuito combinatório a conversão da mesma para linguagem VHDL permite que a síntese do circuito réplica seja realizada pelo compilador de um *software* já existente (no caso em estudo foi utilizado o compilador associado ao *software* ISE *Design*, da *Xilinx* [45].). Uma vez sintetizado o circuito, pode ser utilizada uma FPGA para reproduzir o funcionamento da réplica.

O esquema do circuito sintetizado é influenciado pela estrutura do código gerado. As FPGAs encontram-se otimizadas para implementação de tabelas de verdade (de acordo com [46]), de modo que a estrutura do código VHDL segue a apresentada na Listagem 3.1.

Listagem 3.1 – Estrutura do código VHDL implementado para descrição de circuitos combinatórios.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity nome is
port(
     pin1: in       STD_LOGIC;
     pin2: out      STD_LOGIC;
     pin3: out      STD_LOGIC;
     pin4: in       STD_LOGIC;
     (...)
     pinN: in/out   STD_LOGIC
);
 end entity;

architecture rtl of nome is
signal variável_1 : STD_LOGIC_VECTOR ((Nº entradas-1) downto 0);
begin
    process(pin1,pin4,...) --todas as entradas
    begin
        variável_1<=pin1 & pin4 & ... ; --todas as entradas
        case variável_1
            --when "entradas" => combinação de saída;
            when "01..."=> pin2 <= '0'; pin3 <= '0';...;
            when "10..."=> pin2 <= '0'; pin3 <= '1';...;
            when "11..."=> pin2 <= '1'; pin3 <= '0';...;
            when "00..."=> pin2 <= '1'; pin3 <= '1';...;
            when others => pin2 <= '0'; pin3 <= '0';...;
        end case;
    end process;
end architecture;
```

Neste tipo de estrutura é utilizada uma conversão dos sinais de entrada associados a um instante transformando N sinais de um *bit* em um único sinal de N *bits*, que por sua vez originam a combinação de saída verificada no mesmo instante a que se referem as entradas. Através do comando *case* da linguagem VHDL é possível associar um valor lógico a cada saída dependendo da combinação de entrada. Neste tipo de estrutura é obrigatório garantir que nenhuma combinação de entrada se repete no comando *case*, pois originará erro no compilador.

No que respeita à declaração de entradas e saídas, nomeadamente aos pinos classificados como VCC ou GND, não influência o funcionamento do circuito: podem ser declarados (*in, out ou inout*) e nunca serem utilizados ou simplesmente não serem declarados, uma vez que se destinam apenas à alimentação do circuito original e o seu nível lógico é conhecido.

3.6.6. Taxa de sucesso da réplica obtida

De forma a quantificar a percentagem do comportamento do circuito integrado que foi possível extrair com a observação efetuada é estimada uma taxa de sucesso para a réplica obtida. A réplica obtida reflete todas as combinações observadas, todavia nunca é possível ter mais conhecimento do circuito do que o observado. Assim sendo, quando não são observadas todas as combinações de entrada possíveis, esta taxa não reflete a probabilidade do circuito funcionar corretamente no sistema onde foi observado, pois não é possível saber se as amostras não observadas não ocorrem no sistema onde o circuito integrado se encontra inserido ou se apenas não ocorreram durante o período de observação.

Na estimativa são considerados o número de entradas do circuito e o número de diferentes combinações de entrada encontradas durante o período de observação. Deste modo, a taxa de sucesso da replicação de um circuito combinatório é dada por

$$T(\%) = \frac{M_{NR}}{2^N} \times 100 \tag{3.19}$$

onde T, M_{NR} e N representam a taxa de sucesso, o número de combinações de entrada não repetidas observadas e o número de entradas do circuito, respetivamente.

3.7. Algoritmos de deteção do sinal de relógio

3.7.1. Função do algoritmo

Quando o circuito sob análise é identificado como sequencial, toda a análise combinatória e respetivas medições são descartadas. A primeira etapa do processo de análise de um circuito sequencial síncrono é encontrar o sinal de relógio uma vez que o comportamento de circuito um circuito sequencial está diretamente relacionado ao sinal de relógio.

Os algoritmos de deteção do sinal de relógio têm por objetivo identificar qual dos pinos corresponde ao sinal de relógio e sincronizar as medições subsequentes ao sinal de relógio detetado. A determinação do intervalo de tempo entre duas transições do sinal de relógio (meio período) é importante para efeitos de sincronismo durante a aquisição de medições digitais sincronizadas.

Para deteção do sinal de relógio do circuito sob teste são utilizados dois módulos de *software* de entre os módulos apresentados na Figura 3.2:

- Módulo gerador de instruções para deteção de sinais periódicos;

- Módulo de algoritmos de deteção do sinal de relógio.

3.7.2. Dados requeridos pelo algoritmo

Para o algoritmo efetuar com sucesso a deteção do sinal de relógio, é necessário conhecer o intervalo de tempo entre M transições consecutivas de cada possível sinal de relógio. Na Figura 3.14 encontra-se representado graficamente os instantes temporais importantes para o algoritmo ($t_1, t_2, \dots t_7$) assim como os intervalos temporais de interesse para o algoritmo ($T_{12}, T_{23}, \dots, T_{67}$).

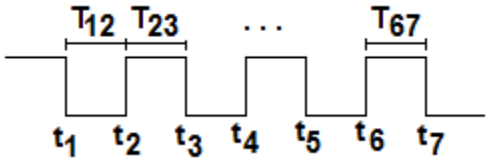

Figura 3.14 – Dados temporais importantes para o algoritmo de deteção do sinal de relógio.

Genericamente, os intervalos temporais de interesse para o algoritmo são dados por

$$T_{ij} = t_j - t_i \tag{3.20}$$

onde $i = j - 1$ para quaisquer valores de j pertencente ao conjunto dos números naturais compreendidos no intervalo $[2, N]$, onde M representa o número de transições observadas.

Estas medições são realizadas para cada pino de entrada do circuito, perfazendo o total de medições dado pela equação

$$Nº\ Medições = \sum_{i=1}^{E} M \tag{3.21}$$

onde E e M representam o número de pinos de entrada do circuito e o número de transições observadas, respetivamente.

3.7.3. Geração de instruções para deteção de sinais periódicos

O módulo gerador de instruções para deteção de sinais periódicos tem por objetivo analisar a classificação atribuída a cada pino e gerar instruções formatadas para aquisição das medições indicadas na subseção 3.7.2. As instruções geradas são transmitidas ao módulo de comunicação que por sua vez irá estabelecer comunicação com o sistema de aquisição de dados e receber as medições efetuadas.

Este módulo possui ainda a função de converter os dados recebidos do sistema de aquisição de dados para que as medições fiquem com uma organização compatível com as requeridas pelo algoritmo de deteção do sinal de relógio. As medições recebidas

consistem nos instantes em que ocorrem as transições e a conversão realizada por este módulo consiste na aplicação da equação (3.20).

O fluxograma da Figura 3.15 ilustra o funcionamento geral deste módulo.

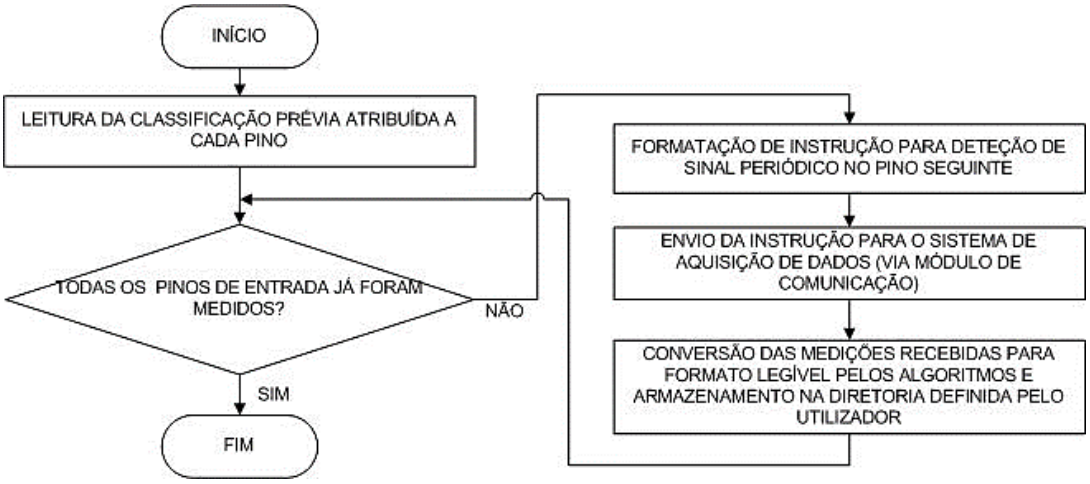

Figura 3.15 – Funcionamento do módulo gerador de instruções para aquisição de medições para deteção de sinais periódicos.

Este módulo foi desenvolvido na linguagem C# e o código fonte encontra-se disponível no Anexo D (seção D.8).

3.7.4. Análise e processamento dos dados

A deteção do pino de relógio é efetuada em duas etapas: deteção de sinais periódicos e escolha do sinal periódico com menor período (maior frequência), dado que em regra o sinal de relógio é o sinal com maior frequência nos circuitos sequenciais. A deteção de sinais periódicos é realizada com base na distribuição das amostras em torno de um valor médio, sendo também efetuado o despiste de erros de medição.

O funcionamento geral do algoritmo encontra-se sintetizado na Figura 3.16.

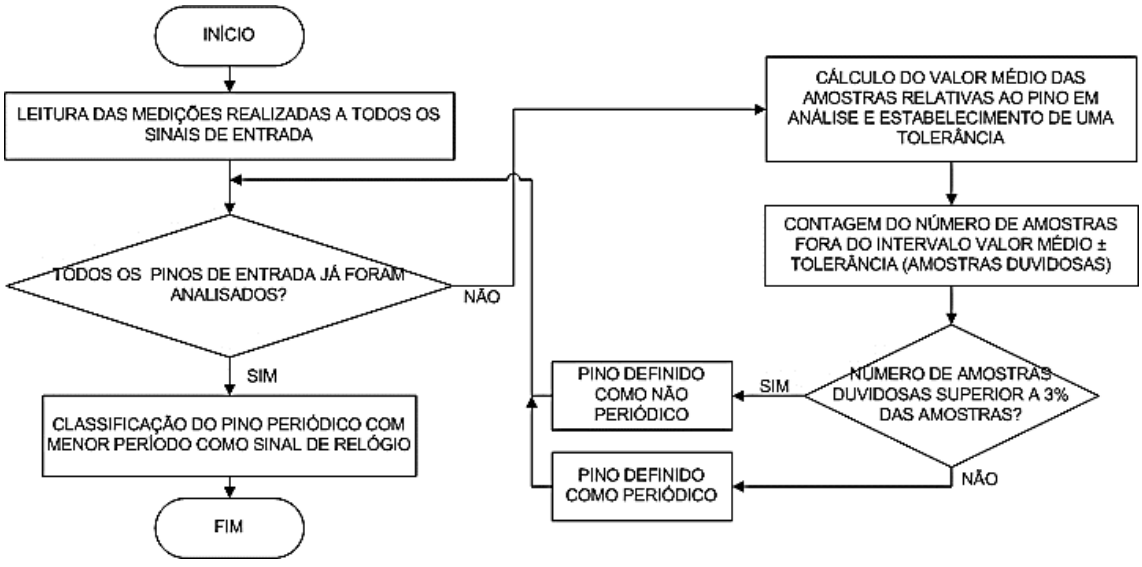

Figura 3.16 – Funcionamento geral do algoritmo de deteção do sinal de relógio.

Numa primeira etapa é realizado o cálculo do valor médio das amostras relativas a um pino, que consiste na média aritmética simples, de acordo com a expressão

$$T_{MÉDIO} = \frac{1}{M}\sum_{i=1}^{M} T_i \tag{3.22}$$

onde $T_{MÉDIO}, T_i$ e M representam o valor médio das amostras, o valor da i-gésima amostra e o número de amostras, respetivamente. A tolerância na análise de cada pino, $T_{TOLERÂNCIA}$, é atribuída em função do valor médio e é calculada através da expressão

$$T_{TOLERÂNCIA} = \frac{k.T_{MÉDIO}}{100} \tag{3.23}$$

onde k representa um fator percentual entre 0 e 100, porém, baseado em testes realizados através do método de aproximações sucessivas, seja aconselhável a utilização de valor pequenos[16].

Sendo conhecido o valor médio a etapa seguinte é realizar o despiste de erros de medição e verificação de periodicidade do sinal de cada pino de entrada. Neste contexto de análise comportamental de um sinal, a distinção entre um comportamento atípico e um erro de medição é uma tarefa difícil. A contagem da percentagem de amostras fora do intervalo $V_{MÉDIO} \pm T_{TOLERÂNCIA}$ permite em simultâneo despistar situações de não periodicidade e erros de medição.

A influência de um erro de medição no valor médio é baixa quando o número de amostras é muito superior aos erros de medição, ou seja, um sinal periódico com esporádicos erros de medição irá conter uma percentagem muito reduzida de amostras fora do intervalo $V_{MÉDIO} \pm T_{TOLERÂNCIA}$, logo é definido como periódico e o valor médio é guardado.

Um sinal não periódico, devido à não concentração das amostras em torno do valor médio irá conter uma maior percentagem de amostras fora do intervalo de interesse, logo será definido como não periódico. A percentagem máxima[17] de amostras fora do intervalo de interesse é definida no código fonte ao invés de ser definida pelo utilizador, pois a utilização de parâmetros inadequados nesta etapa pode afetar drasticamente as etapas seguintes.

Um sinal não periódico com erros de medição associados será naturalmente definido como não periódico, assim como um sinal periódico com demasiados erros de medição (quando os erros de medição têm influência no valor médio).

Quando a análise à periodicidade dos sinais de todos os pinos de entrada é concluída o pino com sinal periódico de maior frequência (menor período) é classificado com sinal de relógio e é atualizada a classificação dos pinos na estrutura de dados, de forma a incluir a informação do sinal de relógio.

O código fonte completo, em linguagem C, utilizado na implementação do algoritmo de deteção do sinal de relógio encontra-se listado no Anexo D (seção D.9).

[16] Na implementação do algoritmo foi utilizado $k = 1\%$.
[17] Na implementação do algoritmo o valor máximo definido foi 3%.

3.8. Algoritmos de deteção do sinal de *reset*

3.8.1. Função do algoritmo

O sinal de *reset* é, à semelhança do sinal de relógio, um sinal importante na tarefa de extrair o comportamento do circuito sequencial, bem como na sintetização de *hardware*. O objetivo dos algoritmos de deteção do sinal de *reset* é encontrar uma relação comportamental entre um pino de entrada e as combinações do(s) pino(s) saída, estabelecendo um estado inicial dos pinos de saída perante atuação no pino de entrada candidato a pino de *reset*. Todos os pinos de entrada, excluindo pino de relógio e pinos de tensão constante, são candidatos a pino de *reset*. É objetivo também estabelecer uma relação entre o sinal de *reset* e o sinal de relógio, de modo a aferir se o *reset* é realizado assincronamente ou sincronamente numa das transições do sinal de relógio.

Para deteção do sinal de *reset* do circuito sob teste são utilizados dois módulos de *software* de entre os módulos apresentados na Figura 3.2:

- Módulo gerador de instruções para aquisição de medições digitais sincronizadas;
- Módulo de algoritmos de deteção do sinal de *reset*.

3.8.2. Dados requeridos pelo algoritmo

Os dados requeridos pelo algoritmo de deteção do sinal de *reset* consistem nos mesmos dados requeridos pelo algoritmo de análise combinatória e descritos na subseção 3.6.2, apenas diferindo no instante de recolha dos dados pretendidos. Enquanto no caso do algoritmo de análise combinatória os instantes de recolha das amostras são aleatórios, o algoritmo de deteção do sinal de *reset* requer que as amostras sejam recolhidas após as transições do sinal de relógio.

3.8.3. Geração de instruções para aquisição de medições digitais sincronizadas

O módulo gerador de instruções para aquisição de medições sincronizadas é muito idêntico ao módulo gerador de instruções para aquisição de medições não sincronizadas. A diferença das instrução enviada ao sistema de aquisição de dados (via módulo de comunicação) reside na inclusão da informação sobre o pino e período do sinal de relógio.

O formato dos dados recebidos e conversões necessárias são iguais aos descritos na subseção 3.6.3.

Este módulo foi desenvolvido na linguagem C# e o código fonte encontra-se disponível no Anexo D (seção D.6).

3.8.4. Análise e processamento dos dados

O algoritmo de deteção do sinal de *reset* analisa todos os pinos de entrada com o objetivo de encontrar o pino que realiza a função de *reset*. Após ler as configurações iniciais e transferir as medições para a estrutura de dados interna do algoritmo, a cada pino são aplicados sub-algoritmos para verificar se o pino em análise pode ser o pino de *reset*, quer seja o seu funcionamento em lógica direta ou inversa, isto é, o *reset* é realizado na transição ascendente ou descendente do sinal. Quando um sinal de *reset* é detetado, a relação entre o sinal de *reset* e o sinal de relógio é analisada, permitindo verificar se o *reset é* síncrono ou assíncrono.

O funcionamento geral do algoritmo de deteção do sinal de *reset* encontra-se sintetizado no fluxograma da Figura 3.17.

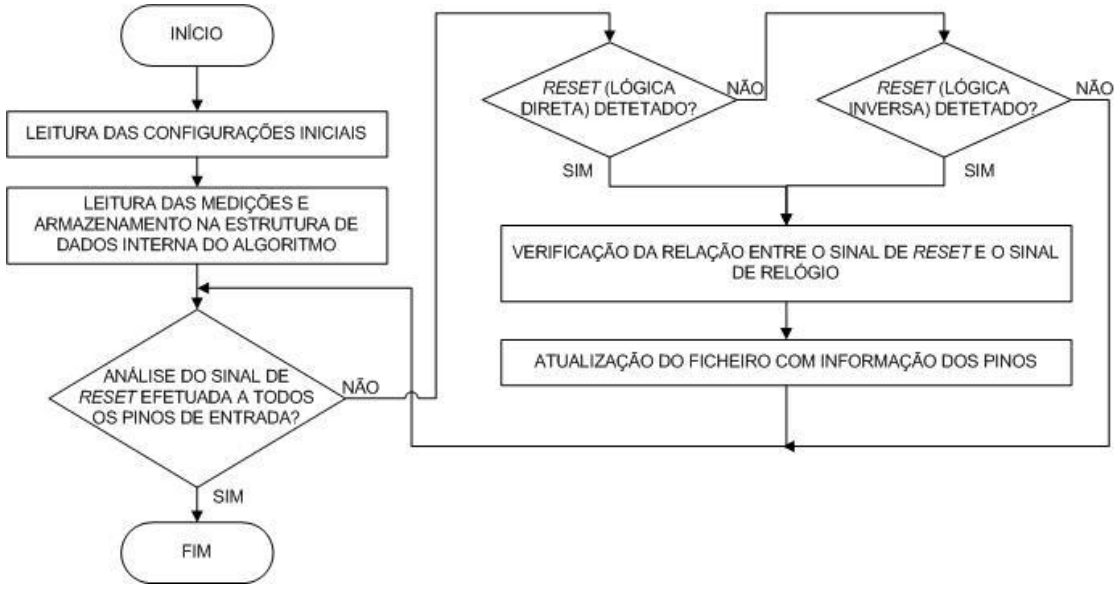

Figura 3.17 – Funcionamento geral do algoritmo de deteção do sinal de *reset*.

3.8.4.1. Sub-algoritmo de deteção do sinal de *reset*

O sub-algoritmo de deteção do sinal de *reset* consiste num algoritmo que analisa a influência de um pino de entrada em todos os pinos de saída, bem como armazena a informação relevante para posterior corelacionamento com o sinal de relógio. O sub-algoritmo permite a deteção de múltiplas situações, através das configurações dos argumentos do algoritmo:

- Pino em análise: define a que pino será aplicado a análise;
- Pino de relógio: define o pino de relógio do circuito;
- Lógica pretendida (direta ou inversa): define se é pretendida a deteção na transição ascendente ou descendente (equivalente a nível alto ou nível baixo, respetivamente, para efeitos de análise algorítmica);
- Deteção pretendida: na implementação deste sistema só foi pretendido detetar sinais de *reset*, todavia o sub-algoritmo encontra-se preparado para fácil expansão que permita ao algoritmo detetar sinais de *preset* entre outros, através da configuração deste parâmetro.

O funcionamento do sub-algoritmo de deteção do sinal de *reset* encontra-se apresentado no fluxograma da Figura 3.18.

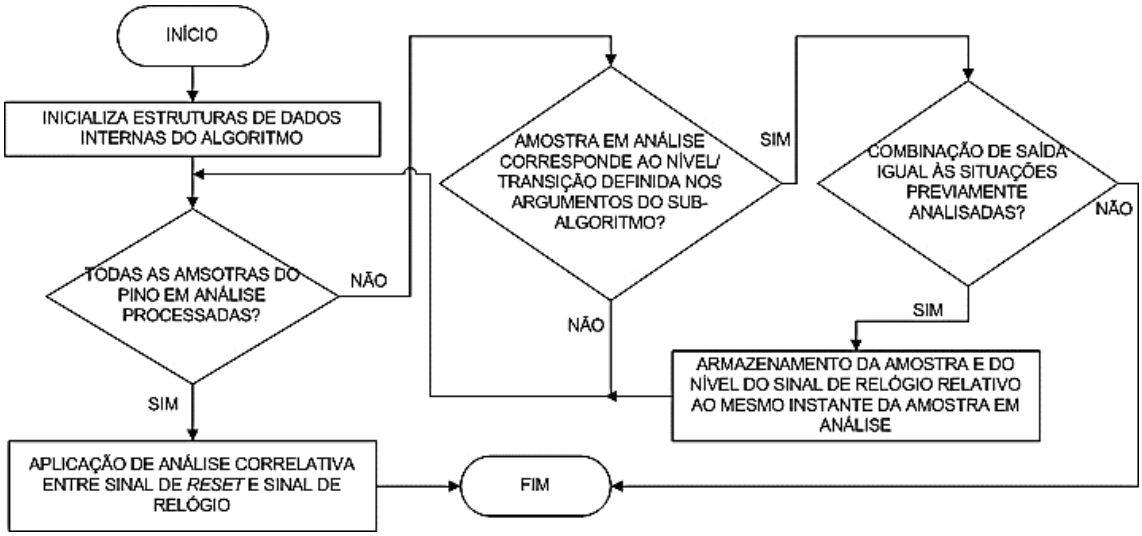

Figura 3.18 – Funcionamento do sub-algoritmo de análise de deteção do sinal de *reset*.

O fluxograma da Figura 3.18 mostra que a deteção do sinal de *reset* é efetuada através de uma análise individual das amostras relativa ao pino em análise, sendo apenas alvo de análise as amostras correspondentes ao nível definido (amostras de interesse) no argumento *lógica pretendida*, sendo as restantes descartadas. Quando uma amostra de interesse é encontrada é armazenada a amostra e o nível de sinal do pino de relógio relativo ao mesmo instante da amostra de interesse. A análise prossegue até que a combinação de saída seja diferente de uma combinação verificada anteriormente durante a análise de uma amostra de interesse anterior. Se todas as amostras são analisadas, então implica que não tenha sido detetada nenhuma combinação de saída diferente para as amostras de interesse, logo o pino é declarado como pino de *reset*. Um outro sub-algoritmo é utilizado para realizar uma análise correlativa entre o pino em análise e o sinal de relógio.

3.8.4.2. Sub-algoritmo de correlação com o sinal de relógio

O sub-algoritmo de correlação com sinal de relógio é executado quando um pino é classificado como pino de *reset* e tem por objetivo analisar o sinal de *reset* relativamente ao sincronismo com o sinal de relógio. A análise efetuada baseia-se no conjunto de amostras do nível de relógio armazenadas, relativas aos instantes das amostras de interesse.

Com as amostras do sinal de relógio previamente armazenadas são efetuadas duas operações: a multiplicação e a soma, de todas as amostras, de acordo com as expressões;

$$V_{SOMA} = \sum_{i=1}^{M} V_i \,; \tag{3.24}$$

$$V_{MULT} = \prod_{i=1}^{M} V_i; \tag{3.25}$$

onde M e V_i representam o número de amostras de interesse e a i-gésima amostra do sinal de relógio armazenada, respetivamente.

Considerando que nesta fase toda a análise é digital, os níveis possíveis do sinal de relógio correspondem a uma variável binária (0/1). Assim sendo a equação (3.24) resulta num valor igual ou superior a zero, enquanto a equação (3.25) resulta numa variável binária. Pela análise dos valores obtidos nas equações (3.24) e (3.25) é possível inferir uma relação do sinal de *reset* com o sinal de relógio, de acordo com a Tabela 3.4.

Tabela 3.4 – Relação entre o sinal de *reset* e sinal de relógio em função das equações (3.24) e (3.25).

V_{SOMA}	V_{MULT}	Análise	Classificação do *reset*
0	0	Para que o valor da soma seja zero, então todas as amostras do sinal de relógio armazenadas correspondem ao nível baixo, ou seja, todas as situações de *reset* ocorreram no nível baixo do sinal de relógio.	Síncrono (Transição descendente ou nível baixo)
> 0	0	O valor da soma (superior a zero) indica que ocorreu, no mínimo, um *reset* no nível alto do sinal de relógio. O valor nulo da multiplicação indica a ocorrência de pelo menos uma situação de *reset* no nível baixo do sinal de relógio. Pela associação de ambas as informações verifica-se que ocorreram situações de *reset* em ambos os níveis do sinal de relógio.	Assíncrono
0	1	Combinação impossível. O valor da soma indica que todas as situações de *reset* ocorreram no nível baixo do sinal de relógio enquanto o valor da multiplicação indica o oposto. Esta situação nunca ocorre.	------
> 0	1	O valor da soma indica a existência de, no mínimo, uma amostra no nível alto do sinal de relógio e o valor da multiplicação demonstra que todas as amostras se encontram no nível alto do sinal de relógio. Conclui-se que todas as situações de *reset* ocorreram no nível alto do sinal de relógio.	Síncrono (Transição ascendente ou nível alto)

Como a replicação é realizada através da observação do funcionamento do circuito e não é possível provocar externamente nenhum estímulo, verificações adicionais são necessárias. Quando um *reset* é classificado como síncrono um aspeto importante a considerar na classificação é a verificação da não ocorrência da situação inversa (contra-análise), de forma a garantir que a classificação atribuída não resulta de uma observação incompleta. Exemplificando, para garantir que o sinal de *reset* funciona apenas na transição ascendente do sinal de relógio é necessário verificar uma situação de não funcionamento na transição descendente do sinal de relógio (e vice-versa). Quando o resultado da contra-análise demonstra que a classificação atribuída carece de observações da situação inversa, o sinal de *reset* é reclassificado para assíncrono. Esta opção de implementação visa abranger situações mais gerais e o código fonte encontra-se disponível no anexo D (seção D.10).

3.9. Algoritmo de análise de circuitos sequenciais

3.9.1. Função do algoritmo e dados requeridos

Conhecidos os pinos aos quais se encontram associados os sinais de relógio e *reset*, o algoritmo de análise sequencial tem por objetivo analisar a sequência de combinações de entrada e respetivas combinações de saída ao longo do tempo e efetuar uma aproximação heurística a uma máquina de estados. Como a informação disponível consiste unicamente nas entradas, saídas e sinais de relógio e *reset*, o algoritmo de análise sequencial é responsável pela atribuição de estados e transições entre estados que descrevam o funcionamento observado. Por fim, o diagrama de estados e respetivas transições são convertidos pelo algoritmo para linguagem VHDL, utilizando uma estrutura de código sintetizável.

Na análise sequencial são reutilizadas as medições dos algoritmos de deteção do sinal de *reset*, dado que a análise do sinal de *reset* enquadra-se no âmbito da análise comportamental e é essencial possuir as medições dos restantes pinos de entrada e saída, referentes aos mesmos instantes temporais, de forma a permitir que ambas as análises sejam associadas.

3.9.2. Aproximação de um circuito sequencial a uma máquina de estados desenvolvida através de métodos heurísticos

A aproximação do comportamento do circuito a uma máquina de estados finitos é baseada na análise das combinações de entrada e saída atuais, anteriores e seguintes. Devido à complexidade, quantidade de dados e ao facto da replicação ser efetuada por observação (não há garantia que todas as situações são observadas) o resultado da replicação consiste na melhor aproximação com base nos dados obtidos. Como os estados internos do circuito sob análise não estão acessíveis ou, porventura, o circuito sob análise não foi concebido como máquina de estados, a forma mais adequada de obter a melhor aproximação é através de um método heurístico.

Os métodos heurísticos englobam estratégias, procedimentos e métodos aproximativos com o objetivo de encontrar uma boa solução, mesmo que não seja a solução ótima, em um tempo computacional razoável [47]. Métodos heurísticos são usualmente aplicados na área da inteligência artificial e aproximam-se da forma de pensar humana [48], [49]. A utilização de métodos heurísticos é apropriada em situações onde não existe um método exato para a resolução do problema ou o mesmo requer elevado tempo de processamento, sendo nesses casos a obtenção de uma boa (mas não ótima) solução melhor do que não obter qualquer solução [47].

No caso em estudo, o principal problema da análise sequencial consiste na distinção entre dois estados diferentes cuja combinação de saída é a mesma. Para dois instantes diferentes, com os dados disponíveis, nunca é possível garantir que para a mesma combinação de saída está associada a estados diferentes ou ao mesmo estado, sendo a decisão do algoritmo baseada na análise de situações observadas anteriormente, o que evidência um procedimento associativo/heurístico. Comparativamente à análise combinatória, é verificada uma maior incerteza na

aproximação, ou por outras palavras, a aproximação efetuada pelos algoritmos de análise combinatória representam na maioria dos casos uma solução ótima enquanto os algoritmos de análise sequencial produzem maioritariamente uma solução possível.

Na análise das situações que os circuitos sequenciais proporcionam e respetiva aproximação a uma máquina de estados, surgem três casos relevantes, cujas implicações se encontram detalhadas na Tabela 3.5.

Tabela 3.5 – Caraterísticas de circuitos sequenciais em função das entradas e tipo de estados.

	Situação	Implicações
1	**Inexistência de pinos de entrada (excetuando sinais de relógio e *reset*)**	Devido à inexistência de pinos de entrada cada estado só pode realizar uma transição. Assim sendo, existe uma única sequência de estados (apenas interrompida pelo sinal de *reset*) que sendo detetada algoritmicamente permite esquematizar todos os estados e respetiva transição, permitindo replicar o comportamento do circuito em análise. Os contadores são um exemplo deste tipo de circuitos.
2	**Existência de pinos de entrada e cada combinação de saída distinta e associada a um único estado**	A existência de pinos de entrada origina que cada estado tenha várias transições possíveis. Como a cada combinação de saída apenas está associado um único estado, o diagrama de estados e transições pode ser obtido através do relacionamento das saídas (estados) com as entradas (transições).
3	**Existência de pinos de entrada e vários estados associados à mesma combinação de saída**	A existência de pinos de entrada origina que cada estado tenha várias transições possíveis. A não existência de uma relação explícita entre a combinação de saída e o estado associado obriga a uma análise baseada na observação de estados anteriores e transições já ocorridas de forma a determinar o estado mais provável associado a cada combinação de medida e respetivas transições, de modo a construir algoritmicamente um diagrama de estados e transições.

Na Tabela 3.5 foram descritas três situações distintas que podem ser alvo de abordagens distintas para efeitos de aproximação. Porém, a distinção entre as situações 2 e 3 apresentadas na Tabela 3.5 é impossível pois o circuito é desconhecido e não são conhecidos os estados internos. Assim sendo, a utilização de um algoritmo genérico com capacidade de efetuar a aproximação de um circuito sequencial a uma máquina de estados em qualquer uma das situações referidas é a solução mais viável [37].

A aproximação implementada visa a procura da melhor solução. Uma máquina de estados finitos é tão mais otimizada quanto menor for o número de estados, sem que a redução do número de estados afete o funcionamento da máquina de estados (estados redundantes). A aproximação implementada constrói um diagrama de estados e transições que se autocorrige à medida que mais casos vão sendo observados. Quando uma nova combinação de saída é encontrada um novo estado é criado. Quando ocorre uma combinação de saída já observada anteriormente é construída uma transição para um estado já existente (que possua a mesma combinação de saída), visando a aproximação da solução ótima. Quando esta situação acontece são verificadas as transições seguintes do estado ao qual a nova transição foi associada e verificado se as próximas transições são compatíveis com as N transições seguintes do

estado associado, onde N representa um critério de profundidade de análise. Quando não é conseguida a associação a um estado já existente então um novo estado é criado. Na Figura 3.19 encontra-se representado um caso exemplificativo de uma situação onde a associação a um estado anterior é realizada com sucesso.

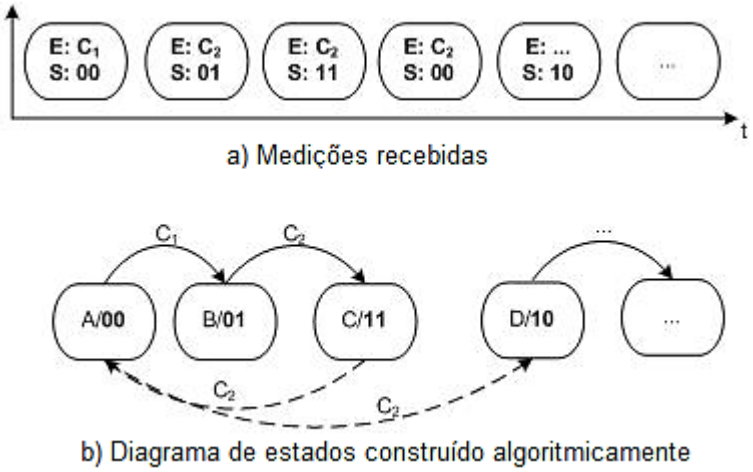

Figura 3.19 – Associação, com sucesso, a estado já existente com profundidade de análise unitária.

Pela análise da Figura 3.19 é verificado que nas primeiras três medições são encontradas novas combinações de saída e são criados novos estados (A,B e C). Na quarta medição a combinação de saída (00) coincide com uma medição já existente logo é criada uma nova transição para o estado A. Para simplificar a análise será considerada uma profundidade de análise unitária para exemplificação, ou seja, vamos verificar a compatibilidade da transição seguinte: com a informação já conhecida, o estado A quando recebe a combinação de entrada C_1 transita automaticamente para o estado B (saída 01), e como a medição seguinte consiste na combinação de entrada C_2, como ainda não tinha sido observada nenhuma ocorrência desta combinação não há incompatibilidade entre as transições em análise e as previamente observadas (a combinação de entrada e saída C_1 e 01, respetivamente, também não indicavam incompatibilidade). Porém, na Figura 3.20 é apresentado um novo exemplo onde existe incompatibilidade entre a associação efetuada e o comportamento observado previamente. Esta análise pode ser generalizada para profundidade N.

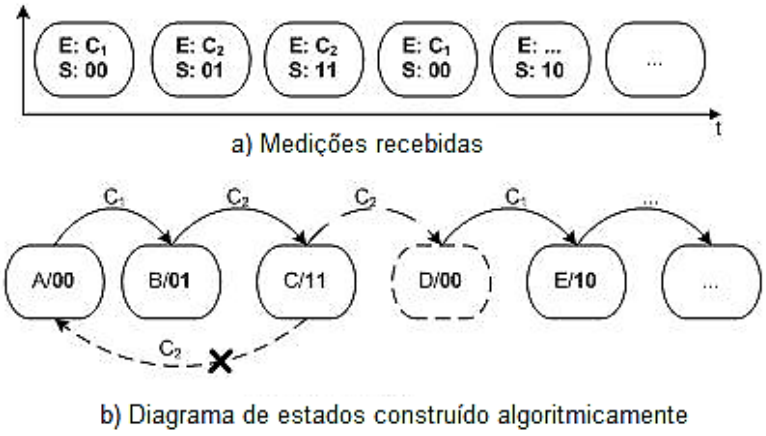

Figura 3.20 – Associação, sem sucesso, a estado já existente com profundidade de análise unitária.

Pela análise da Figura 3.20 verifica-se uma situação de associação a um estado existente similar ao apresentado na Figura 3.19. Porém na medição a combinação de entrada é C_1 originando uma combinação de saída 10. O estado já existente ao qual se pretende associar a medição em análise efetua a transição para o estado B, de saída 01, quando recebe a combinação de entrada C_1 logo verifica-se que existe uma incompatibilidade na transição C_1. Desta forma a associação a um estado anterior não pode ser efetuada e um novo estado é criado (Estado D).

Generalizando para uma profundidade de análise de valor N, não pode existir qualquer transição incompatível nas N transições seguintes do estado a que é feita a associação. Quando existem vários estados anteriores com a mesma combinação de saída que a medição em análise só é criado um novo estado se fracassarem as associações a todos os estados com a mesma saída.

No pior caso, quando nenhuma associação é conseguida, o diagrama de estado consistirá numa sequência em M estados, onde M representa o número de medições, que não sendo uma aproximação ótima é a aproximação possível. No caso onde o circuito sob análise não possui pinos de entrada, a mesma aproximação pode ser assumindo que a combinação de entrada mantém-se constante.

3.9.3. Configurações iniciais para análise de circuitos sequenciais

Quando um circuito sequencial é detetado o utilizador tem a possibilidade de reajustar o número de amostras a utilizar na análise e definir novas configurações iniciais, apresentadas na Tabela 3.6.

Tabela 3.51 – Parâmetros de configuração para circuitos sequenciais.

Parâmetro	Descrição
Seleção automática dos parâmetros	Permite escolher se os parâmetros a utilizar são definidos de forma manual ou automática.
Precisão	Define o grau de certeza para que uma nova transição seja estabelecida. Pode assumir os níveis baixo, normal e alto.
Profundidade	Define a profundidade de análise. O valor pode ser definido de forma manual ou automática.
Conversão do sinal de *reset*	O algoritmo possibilita a conversão do sinal de *reset* para síncrono, assíncrono ou manter o sinal original.

3.9.4. Análise e processamento

3.9.4.1. Implementação da aproximação heurística

O algoritmo de análise sequencial efetua um conjunto de três operações que permitem a replicação de circuitos digitais sequenciais: análise da transição de relógio de interesse, realização da aproximação descrita na seção 3.9.2. e escrita de ficheiro com descrição comportamental em VHDL.

A identificação da transição de interesse do sinal de relógio consiste em verificar se as combinações de entrada e saída são alteradas quando o sinal relógio varia do

nível baixo para o nível alto (transição ascendente) ou vice-versa (transição descendente). Quando é conhecida a transição de interesse, as medições associadas às transições sem interesse são descartadas.

A implementação da aproximação do circuito sequencial a uma máquina de estados encontra-se sintetizada na Figura 3.21.

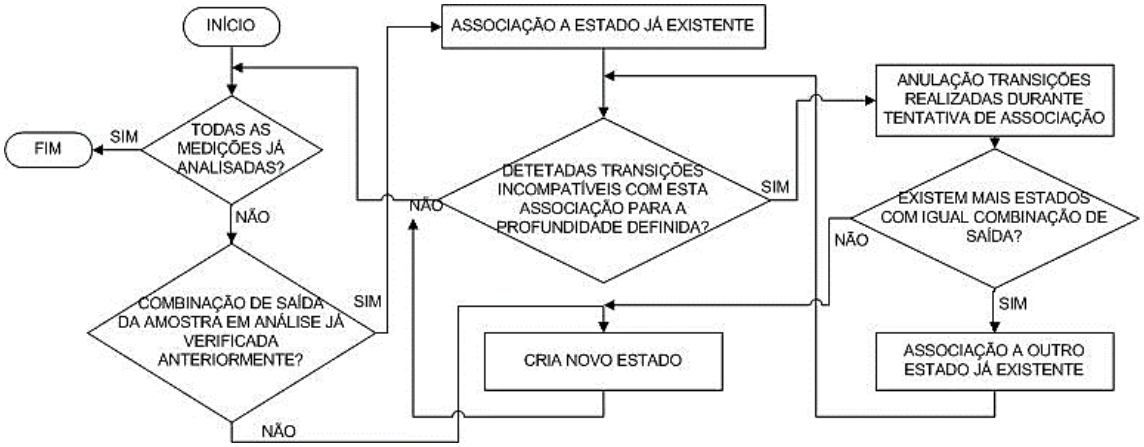

Figura 3.21 – Fluxograma da implementação da aproximação de circuito sequencial a uma máquina de estados.

A implementação da aproximação consiste unicamente em converter o princípio de aproximação descrito na seção 3.9.2 para código (linguagem C), que se encontra disponibilizado no Anexo D (seção D.11). Em suma, a cada nova combinação de saída encontrada é atribuído um novo estado e as combinações de saída já existentes são realizadas tentativas de associação a estados já existentes se não for observada qualquer incompatibilidade com comportamento já verificado anteriormente, e criação de novo estado em caso contrário. Quando todas as medições já foram sujeitas a análise, o diagrama de estados e transições obtido é convertido para linguagem VHDL.

3.9.4.1. Estrutura do código (linguagem VHDL) da réplica

A estrutura do código VHDL utilizado na descrição comportamental de circuitos sequenciais é distinta da utilizada na descrição de circuitos combinatórios, partilhando apenas a estrutura de código utilizada para declaração de entradas e saídas. As variáveis utilizadas na descrição de circuitos sequenciais consistem numa estrutura contendo os diferentes estados possíveis e variáveis para armazenamento do estado atual, estado seguinte, combinação de entrada e sinal de *reset*, de acordo com a Listagem 3.2.

Listagem 3.2 – Variáveis utilizadas na descrição de circuitos sequenciais.

```
type estados_replica is (     |  shared variable estado_atual: estados_replica;
    Estado_0,                 |  shared variable estado_seguinte: estados_replica;
    Estado_1,                 |  signal reset : STD_LOGIC;
    ...                       |  signal entradas: STD_LOGIC_VECTOR ((Nº entradas-1) downto 0);
    Estado_N  );              |  --(ou apenas STD_LOGIC se Nº entradas é igual a 1)
```

O corpo do código VHDL possui três módulos de processamento paralelo (declaração *process)* sincronizados com o sinal de relógio: deteção de *reset*, transição

de estado e um módulo combinatório de atualização das saídas e aplicação do diagrama de estados.

A estrutura da implementação da deteção do *reset* difere na dependendo da relação com o sinal de relógio (síncrono ou assíncrono), mas em qualquer um dos casos é realizada uma conversão do valor do pino de *reset* (definido como entrada) para uma variável interna, como apresentado na Listagem 3.3.

Listagem 3.3 – Blocos de deteção e conversão do sinal de *reset* para variável interna.

```
process (Relógio, Pino_reset) is
begin
    if(Pino_reset ='nível de interesse') then
        reset <= '1';
    else
        reset <='0';
    end if;
end process;
```

a) Reset assíncrono

```
process (Relógio, Pino_reset) is
begin
    if(Relógio ='nível de interesse') then
        if(Pino_reset ='nível de interesse') t
            reset <= '1';
        else
            reset <='0';
        end if;
    end if;
end process;
```

b) Reset síncrono

O bloco de transição de estado efetua a verificação se o sinal de *reset* se encontra ativo e a transição para o estado seguinte dependendo do sinal de *reset*: quando se encontra ativo atribui transição para estado inicial, caso contrário transita para o próximo estado em função do diagrama de estados do circuito em análise. A estrutura gerada pelo algoritmo para este bloco encontra-se apresentada na Listagem 3.4.

Listagem 3.4 – Bloco de atribuição do estado seguinte.

```
process (Relógio, reset) is
begin
    if (reset = '1') then
        estado_atual := Estado_0;
    elsif(X_edge(CLK)) then --rising_edge ou falling_edge dependendo do circuito
        estado_atual := estado_seguinte;
    end if;
end process;
```

A atribuição das saídas e aplicação do diagrama de estados e transições tem estrutura diferente dependendo da existência ou não de pinos de entrada. Quando não existem entradas (estado seguinte é apenas influenciado pelo estado anterior), é utilizada a estrutura apresentada na Listagem 3.5.

Listagem 3.5 – Bloco de atribuição das saídas e aplicação do diagrama de estados para circuitos sem entradas.

```
process (Relógio) is
begin
    case estado_atual is
        when Estado_0 => estado_seguinte:="Atribui estado"; "Atribui sinais de saída";
        (...)
        when Estado_N => estado_seguinte:="Atribui estado"; "Atribui sinais de saída";
        when others =>   estado_seguinte := Estado_0;
    end case;
end process;
```

Quando existem pinos de entrada o estado seguinte é influenciado pelo estado anterior e pela combinação de entrada, logo existe a necessidade de efetuar uma dupla

seleção, ao invés de uma seleção simples como no caso anterior. A estrutura de código gerada pelo algoritmo encontra-se apresentada na Listagem 3.6.

Listagem 3.6 – Bloco de atribuição das saídas e aplicação do diagrama de estados para circuitos com entradas.

```
process (Relógio) is
begin
  entradas<= "pino entrada 1" & ... & "pino entrada N";
  case estado_atual is
      when Estado_0 =>
        case (entradas) is
           when "combinação 1" => estado_seguinte:="Atribui estado"; "Atribui sinais de saída";
           (...)
           when "combinação M" => estado_seguinte:="Atribui estado"; "Atribui sinais de saída";
           when others => estado_seguinte:=estado_atual;
        end case;
      (...)
      when Estado_N =>
        case (entradas) is
           when "combinação 1" => estado_seguinte:="Atribui estado"; "Atribui sinais de saída";(...)
           (...)
           when "combinação M" => estado_seguinte:="Atribui estado"; "Atribui sinais de saída";
           when others => estado_seguinte:=estado_atual;
        end case;
      when others =>   estado_seguinte := Estado_0;
   end case;
 end process;
```

3.9.5. Taxa de sucesso da réplica obtida

De forma a quantificar a percentagem do comportamento circuito integrado que foi possível extrair com a observação efetuada é estimada uma taxa de sucesso para a réplica obtida, à semelhança do realizado para os circuitos combinatórios.

Na estimativa são considerados o número de estados, as transições detetadas durante o período de observação e as transições possíveis (em função do número de entradas). Deste modo, a taxa de sucesso da replicação de um circuito sequencial é dada por

$$T(\%) = \frac{T_D}{S \times 2^E} \times 100 \tag{3.26}$$

onde T, T_D, S e E representam a taxa de sucesso, o número total de transições detetadas, o número de estados encontrados e o número de entradas, respetivamente.

3.10. Módulo de comunicação

O módulo de comunicação tem a simples tarefa de transmitir informações entre os módulos alojados no computador e o sistema de aquisição de dados.

O módulo de comunicação efetua a leitura da instrução e respetivos parâmetros a partir de um ficheiro e formata os dados de forma a enviar uma trama que o sistema de aquisição de dados seja capaz de interpretar. O formato das tramas, instruções e respetivos argumentos são descritos no capítulo 4 deste documento. Os dados enviados

pelo sistema de aquisição são recebidos e armazenados na estrutura de dados, numa diretoria lida de um ficheiro de configuração.

O código fonte (linguagem C#) utilizado para implementar este módulo encontra-se na seção D.12 do Anexo D.

3.11. Módulo de interface e coordenação

O módulo de interface e coordenação tem duas funções: fornecer uma interface ao utilizador que permita introduzir os parâmetros e configurações a utilizar na replicação bem como retornar o estado da replicação e realizar a gestão da execução dos diferentes módulos.

O módulo de interface é o único módulo que se encontra sempre em execução durante todo o processo de replicação e garante o funcionamento geral do sistema, estabelecendo a ordem de execução dos diferentes módulos ao longo do tempo para que o sistema execute o funcionamento apresentado na Figura 3.1. O módulos são colocados em execução de acordo com o fluxograma apresentado na Figura 3.22

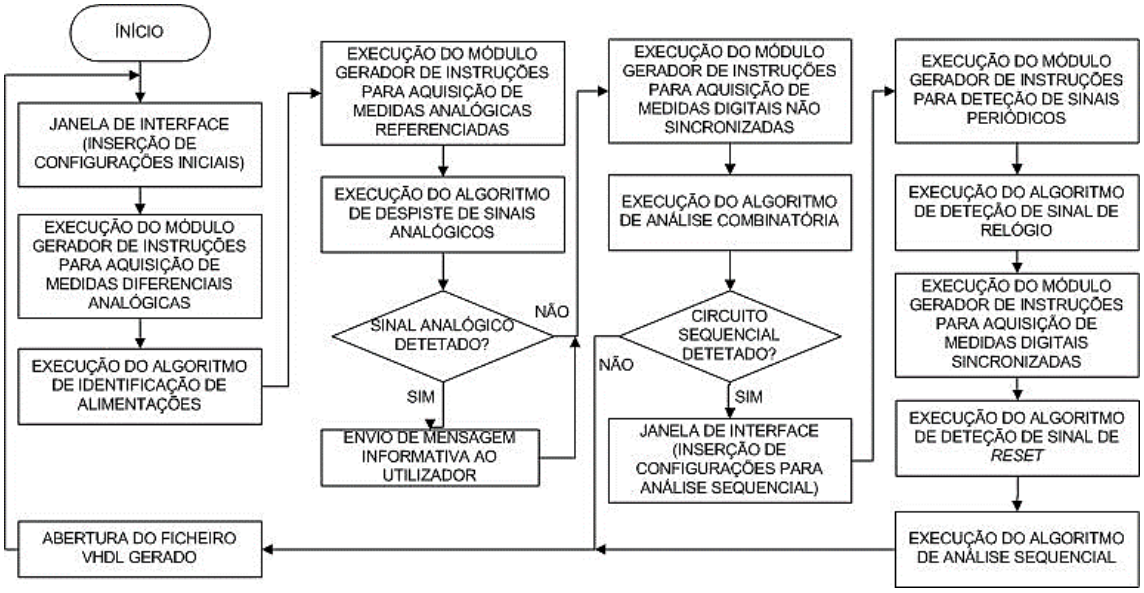

Figura 3.22 – Funcionamento do módulo de interface e coordenação.

O código fonte (linguagem C#) utilizado na implementação encontra-se listado no Anexo D (seção D.13). Um tutorial sobre a interface desenvolvida está disponível no Anexo E.

3.12. Análise matemática da probabilidade de observar o comportamento completo de um circuito

Como a precisão das réplicas dependem da observação e da quantidade de amostras recolhidas será demonstrado matematicamente que, para um grande número de amostras recolhidas ($R \gg 2^n$), a probabilidade de serem observadas todas as

combinações (no caso dos circuitos combinatórios) é um acontecimento muito provável, quando todas as combinações de entrada possuem a mesma probabilidade de ocorrer. A probabilidade de em M amostras recolhidas todas serem referentes a uma combinação de entrada diferente é dada pela equação (3.27).

$$P(M) = \frac{M}{M} \times \frac{M-1}{M} \times \frac{M-2}{M} \times \ldots \times \frac{1}{M} = \frac{(M-1)!}{M^{M-1}} . \tag{3.27}$$

Como é possível constatar que o

$$\lim_{M \to +\infty} P(M) = 0 \tag{3.28}$$

ou seja, com o aumento do número de combinações de entrada a probabilidade de recolher M combinações diferentes em M amostras torna-se muito improvável. A probabilidade de obter todas as combinações de entrada para um número de amostras recolhidas superior a M é dado pela equação (3.29).

$$\begin{gathered} P(R) = P(N \times M) = P(M) + P(2.M) + \cdots + P(N \times M) = \\ = P(M) + \big(1 - P(M)\big).P(M) + \big(1 - P(M)\big)^2.P(M) + \cdots + \big(1 - P(M)\big)^{N-1}.P(M) = \\ = \sum_{i=1}^{N} P(M).\big(1 - P(M)\big)^{i-1} = \end{gathered} \tag{3.29}$$

A equação (3.29) apresenta a estrutura de uma progressão geométrica [50] e a soma dos termos de uma progressão geométrica é dada por [50]

$$\sum_{i=1}^{\infty} a_1.q^{n-1} = \frac{a_1}{1-q} \tag{3.30}$$

e utilizando a expressão (3.31) é possível simplificar a expressão (3.29) em

$$P(N \times M) = \frac{P(M)}{1 - \big(1 - P(M)\big)} = 1 , \tag{3.31}$$

que converge para o valor um, ou seja, quando o valor de N cresce a probabilidade de adquirir todas as combinações de entrada possíveis tende para a unidade, logo é um acontecimento muito provável.

Se um circuito sequencial for considerado uma aproximação a um circuito combinatório onde o estado atual e as entradas constituem o equivalente às combinações de entrada e o estado seguinte equivale às combinações de saída, sendo as combinações apenas verificadas sincronamente com o sinal de relógio, a dedução apresentada nas equações (3.27) à (3.21) pode ser generalizada para circuitos sequenciais. Enquanto num circuito combinatório a análise do número de amostras é realizada em função do número de entradas, nos circuitos sequenciais têm de ser considerados o número de estados e número de entradas (que correspondem ao número de transições possíveis que cada estado pode ter). Assim sendo, pelo paralelismo descrito entre circuitos combinatórios e sequenciais, pode ser inferido que se for realizada uma observação prolongada e o número de amostras recolhidas for suficientemente grande, a probabilidade de todos os estados e transições serem observados tende para o valor unitário de acordo com a equação (3.31).

4. Sistema de aquisição de dados

4.1. Introdução

Neste capítulo será descrito o protótipo do sistema de aquisição de dados implementado para demonstrar experimentalmente o funcionamento dos algoritmos descritos no capítulo anterior. O sistema de aquisição de dados é o responsável por fornecer aos algoritmos de replicação todos os dados necessários, através da realização de medições no circuito alvo de replicação.

Devido ao desconhecimento do circuito sob teste e da localização da referência (GND) do mesmo, precauções adicionais terão de ser contempladas no projeto do sistema de aquisição de dados de forma a impedir a ocorrência de curto-circuitos, ou seja, é necessário isolamento entre o circuito sob teste e o sistema de aquisição de dados. A velocidade de aquisição das medições é outro fator importante a ponderar no projeto do sistema. Dado que os circuito-alvo são digitais, a minimização das conversões de sinais analógicos para digitais é um fator, entre outros, que permite aumentar a velocidade de aquisição.

Na primeira fase de aquisição de dados, são adquiridas medições diferenciais entre cada par de pinos do circuito sob teste na forma analógica para identificação das tensões de alimentação e despiste de circuitos não digitais. Estas aquisições podem ser realizadas a uma velocidade baixa dado que as tensões de alimentação são aproximadamente constantes. Na segunda fase de aquisição, na forma digital para extração do comportamento, a velocidade de aquisição é, essencialmente nos circuitos sequenciais, fundamental na precisão dos resultados. Esta distinção permite restringir a escolha de componentes mais rápidos, em geral mais caros, aos blocos do sistema destinados à segunda fase de aquisição.

Entre o sistema de aquisição e o computador onde são executados os algoritmos de replicação foi implementado um canal de comunicação via porta série.

4.2. Critérios de escolha de componentes

No projeto e dimensionamento de um circuito eletrónico existem vários fatores a serem considerados, tais como: caraterísticas dos componentes, custo dos componentes, componentes em *stock*, entre outros.

Na implementação do sistema de aquisição de dados o principal critério utilizado na escolha dos componentes foi o *stock* de componentes existentes no laboratório [51]. A frequência máxima de operação do circuito sob teste é influenciada pela resposta em frequência dos componentes do circuito de aquisição de dados. No entanto, como os componentes eletrónicos com maior largura de banda são geralmente mais

dispendiosos, com o protótipo desenvolvido pretende-se comprovar o funcionamento do sistema para baixas frequências assente numa arquitetura preparada para operação em frequências maiores.

4.3. Arquitetura do sistema de aquisição de dados

Num alto nível de abstração, a arquitetura do sistema pode ser descrita em quatro módulos distintos: encaminhamento, isolamento, aquisição e controlo. Na Figura 4.1 representado o diagrama de blocos geral do circuito de aquisição de dados.

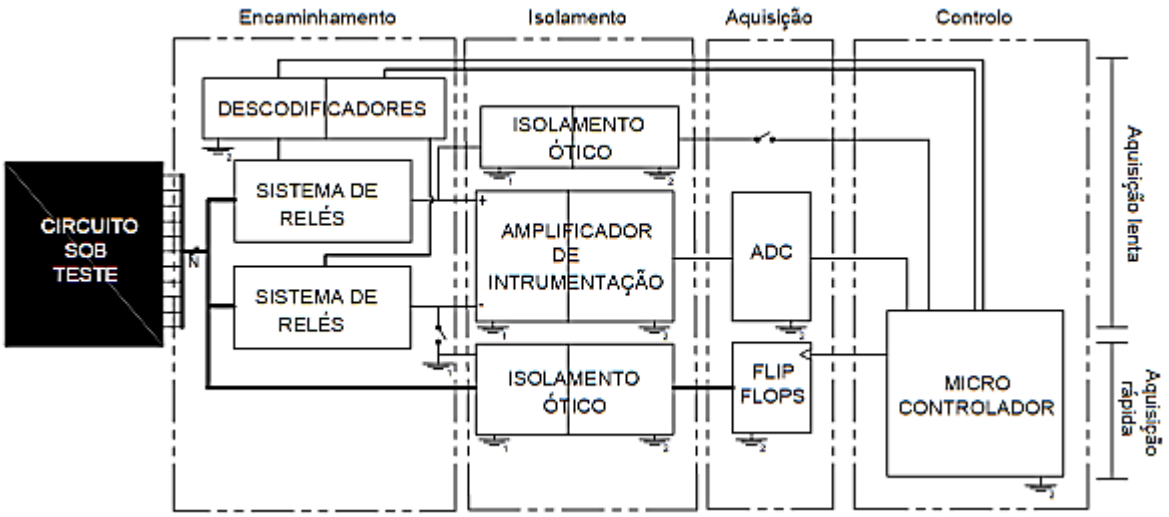

Figura 4.1 – Diagrama de blocos do sistema de aquisição de dados.

A função do módulo de encaminhamento é realizar o encaminhamento dos sinais do circuito sob teste para realização de medições diferenciais entre quaisquer dois pinos do circuito sob teste, numa fase onde não são conhecidas alimentações do circuito.

O módulo de isolamento é responsável por transmitir a informação para os blocos seguintes, de uma forma eletricamente isolada, evitando assim a ocorrência de curto-circuitos durante a fase onde a referência do circuito é desconhecida. Quando a referência do circuito é determinada (algoritmicamente), esta é conectada à componente não isolada do circuito através de um interruptor ou componente similar. O isolamento é garantido na saída do bloco de isolamento, mesmo após conhecida a referência, providenciando assim um mecanismo de segurança na camada física.

A aquisição de dados consiste em recolher as medições, analógicas e digitais, já previamente isoladas do circuito sob teste. Na aquisição de medições analógicas e digitais são utilizados conversores de analógico para digital (ADC – *Analog* to *Digital Converter*) e *flip-flops*, respetivamente.

O sistema de controlo, baseado num microcontrolador, tem a missão de em cada instante interpretar as necessidades de aquisição de dados e aplicar os sinais de

controlo aos diferentes componentes do circuito bem como fazer a recolha e transmissão da informação para o computador onde se encontram alojados os algoritmos. Neste módulo existe ainda um detetor de transições, destinado a identificar as transições do sinal de relógio com objetivo de gerar uma interrupção no microcontrolador para efetuar a aquisição de medições, essencialmente nos circuitos sequenciais. Dois interruptores (ou componentes similares) controlam a proveniência do sinal de início de aquisição de dados, isto é, se os instantes das aquisições são determinados pelo microcontrolador ou diretamente pelo detetor de transições.

Nesta dissertação foi implementado um sistema capaz de analisar circuitos integrados com um máximo de 16 pinos, contudo o circuito pode ser generalizado para maior número de pinos. Tendo por base o diagrama de blocos da Figura 4.1 existem várias implementações possíveis. Nas subseções que se seguem será descrita a implementação realizada nesta dissertação.

4.4. Implementação do sistema de aquisição de dados

4.4.1. Módulo de encaminhamento

O módulo de encaminhamento é essencial na aquisição de medições diferenciais entre os vários pinos do circuito sob teste. Este bloco é constituído por dois sistemas de relés idênticos, onde todos os pinos do circuito são conectados a cada um dos sistemas de relés e através de variáveis de seleção apenas um dos pinos é conectado à saída do sistema de relés. Assim sendo, através do controlo das variáveis de seleção dos dois sistemas de relés é possível efetuar o encaminhamento do sinal de forma a medir a tensão entre qualquer par de pinos do circuito sob teste, como ilustrado na Figura 4.2.

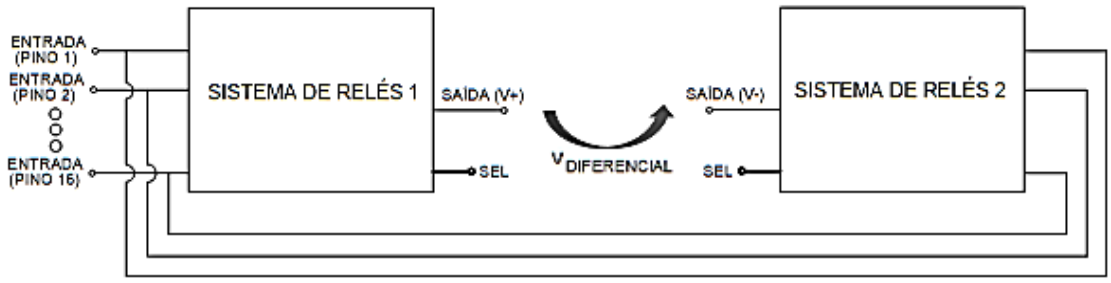

Figura 4.2 – Diagrama de blocos de alto nível do módulo de encaminhamento.

A escolha de relés em detrimento de outros comutadores, nomeadamente MOSFETs, é devida às especificidades do circuito:

- Necessidade de isolamento entre o sinal a analisar e os sinais de controlo (necessidade de menor número fontes de tensão isoladas necessárias com o uso de relés);

- Minimização das quedas de tensão no comutador;
- Frequência de comutação pouco relevante (o intervalo de tempo entre duas medições não é decisivo nas medições que este módulo suporta, logo a frequência de comutação pode ser baixa);
- Capacidade de condução de corrente em qualquer sentido.

Escolhido o comutador, é necessário dimensionar o restante circuito do sistema de relés. A escolha do relé que deve estar ativo é realizada em função dos sinais de seleção, porém a utilização de 16 sinais de controlo (um para cada relé) para cada sistema de relés implica uma utilização massiva de portas do microcontrolador. A inserção de um descodificador permite reduzir o número de sinais de seleção para

$$S = \log_2(N) \tag{4.1}$$

onde S e N representam o número de sinais de seleção necessários e o número máximo de pinos que o sistema admite, respetivamente.

O esquema elétrico de um sistema de relés com descodificador (DG526CJ) implementado encontra-se representado na Figura 4.3.

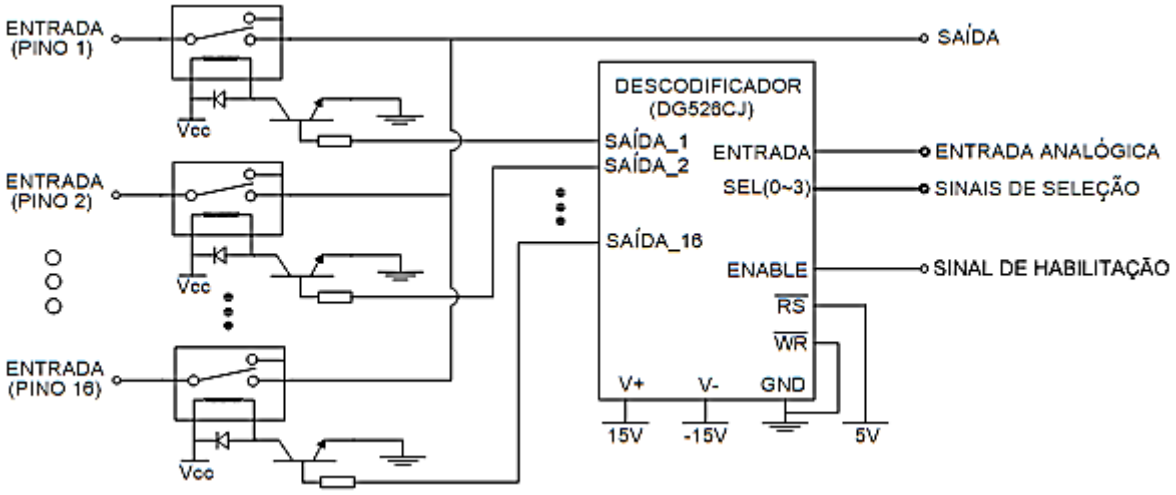

Figura 4.3 – Esquema elétrico de sistema de encaminhamento baseado em relés, com descodificador.

A utilização de um descodificador analógico permite o controlo da tensão de ativação dos relés através do ajuste da entrada analógica do descodificador, possibilitando assim uma maior versatilidade do circuito e independência do relé escolhido (limitado à gama de operação do descodificador). A corrente de ativação do relé não é fornecida pelo descodificador sendo recomendável a utilização de um amplificador de corrente na configuração de emissor comum, onde a tensão de saída do descodificador irá determinar o modo de operação do transístor bipolar de junção (TBJ), fornecendo corrente ao relé quando a tensão de saída é suficiente para polarizar o TBJ e vice-versa.

O díodo aos terminais da bobine do relé permite que a corrente de desmagnetização da bobine flua pelo díodo, evitando danos no circuito de controlo do relé.

O controlo de um sistema de relés igual ao descrito é efetuado através dos sinais de seleção e habilitação. Uma análise do tempo de comutação dos relés comparativamente ao tempo de comutação do descodificador é necessária, de forma a garantir que dois relés não se encontram ativos em simultâneo no mesmo sistema de relés, causando um possível curto-circuito no circuito sob teste. Quando ocorre uma mudança do relé ativo o sinal de habilitação do descodificador deve desativar o mesmo (nenhuma saída ativa) durante a abertura do relé anteriormente ativo. A Figura 4.4 apresenta a evolução dos sinais de controlo em situações de mudança do relé ativo,

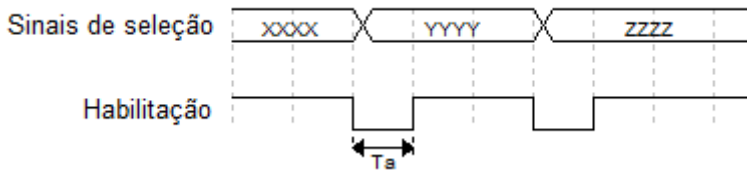

Figura 4.4 – Diagrama Temporal dos sinais de controlo de um sistema de relés.

onde T_a é dado por

$$T_a = T_{ab} + M_s \tag{4.2}$$

sendo T_{ab} e M_s o tempo de abertura do relé e margem de segurança, respetivamente.

Na implementação do módulo de encaminhamento foram utilizados os componentes apresentados na Tabela 4.1.

Tabela 4.1 – Componentes utilizados na implementação dos módulos de encaminhamento.

Componente	Referência	Caraterísticas
Relé	EC2-5NU	[52]
Transístor	2N2222	[53]
Díodo	1N4448	[54]
Descodificador	DG526CJ	[55]

São utilizados dois circuitos iguais ao da Figura 4.3 para formação do módulo de encaminhamento completo.

4.4.2. Módulo de isolamento

O módulo de isolamento permite que seja efetuada a passagem de informação de uma forma eletricamente isolada. No sistema de aquisição são aplicados dois tipos de isolamento: por elevada impedância de entrada e ótico, para medições analógicas e digitais respetivamente.

4.4.2.1. Isolamento por elevada impedância de entrada

As medições de sinais analógicos são necessárias na fase de deteção dos pinos de alimentação e incidem na medição da tensão entre as saídas dos dois blocos de encaminhamento baseados em relés. Como nesta fase, a medição pretendida é diferencial, isto é, não está referenciada ao GND, é utilizado um amplificador de instrumentação.

Na implementação do isolamento de sinais analógicos foi utilizado o amplificador de isolamento de referência AD622 [56], com a montagem apresentada na Figura 4.5.

Figura 4.5 – Isolamento de sinal analógico com o amplificador de instrumentação AD622.

Com a montagem apresentada na Figura 4.5, é possível a leitura de diferenças de tensão na entrada de módulo 15V. Os pinos RG permitem a configuração de um ganho de tensão (entre 2 e 1000) através da utilização de uma resistência [56]. Como a saída do amplificador se destina a leitura numa ADC e dada a gama de tensões admitidas na entrada foi configurado o ganho mínimo permitido pelo amplificador (ganho unitário) através da não utilização de qualquer resistência entre os pinos RG [56]. Assim sendo, a tensão na saída é dada pela expressão (4.3).

$$V_{SAÍDA} = V_{(+IN)} - V_{(-IN)} \tag{4.3}$$

Tendo a gama de tensões de saída (igual à gama de tensões de entrada) e a gama de tensões de entrada tipicamente admitidas pelas ADC de baixa potência a aplicação de uma redução proporcional na saída do amplificador será necessária.

4.4.2.2. Isolamento ótico

As medições de sinais digitais são necessárias na fase de aquisição de amostras para análise comportamental. Nesta fase a despistagem de circuitos não digitais já foi efetuada, ou seja, o objetivo das medições é determinar o nível, em vez do valor exato da tensão entre dois pontos. Assim sendo, a utilização de opto-acopladores permite a transmissão de informação de uma forma eletricamente isolada através de um conjunto

LED-fotodetetor, onde a polarização, ou não, do LED depende do nível do sinal de entrada, em particular da corrente.

Na implementação do bloco de isolamento ótico no sistema de aquisição de dados desenvolvido, é necessário ter em conta os requisitos da Tabela 4.2.

Tabela 4.2 – Caraterísticas e requisitos do bloco de isolamento ótico.

	Caraterísticas	Requisitos
Entrada	Ligação: pinos do circuito sob teste	- Alta impedância de entrada - Conversão para escala mais baixa
	Gama de tensões de entrada: 0 V - 15 V	
	Nível alto: 3,3 V / 5 V	
Saída	Ligação: flip-flop	- Saída compatível com lógica TTL/CMOS
	Gama de tensões de saída: 0 V - 5 V	
	Nível alto: 5 V	
Opto acoplador	6N136 [57]	

A impedância de entrada de um opto-acoplador é baixa, logo o aumento da impedância de entrada deverá ser feito externamente, por exemplo, com recurso a um amplificador operacional, a funcionar como seguidor de tensão (ganho unitário), como ilustrado na Figura 4.6. O aumento da impedância de entrada é fundamental para não afetar o funcionamento do circuito sob teste devido ao consumo de corrente.

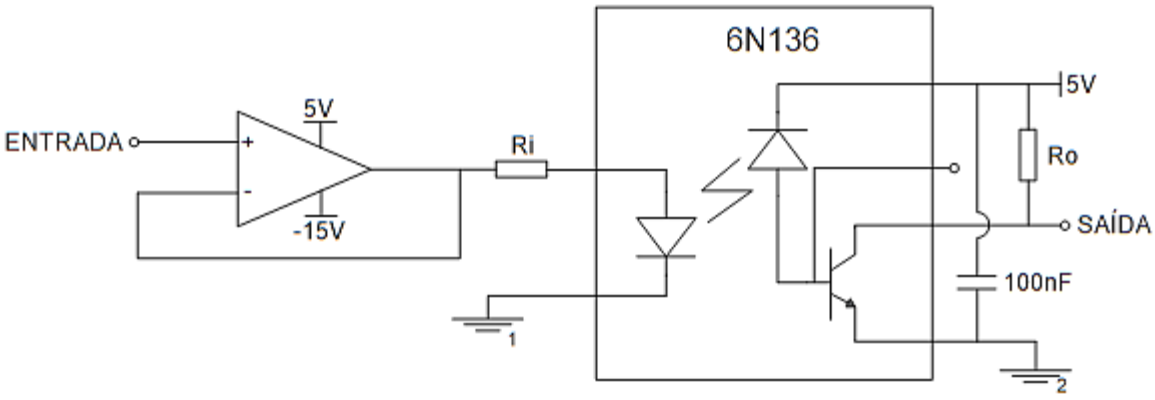

Figura 4.6 – Circuito de isolamento ótico com opto-acoplador 6N136 e alta impedância de entrada.

O dimensionamento da resistência R_i tem em conta a tensão de entrada e a corrente no LED. Pelas regras dos circuitos elétricos a corrente no LED, I_{LED}, é dada por

$$I_{LED} = \frac{V_{ENTRADA} - V_{LED}}{R_i} \tag{4.4}$$

Onde $V_{ENTRADA}$ e V_{LED} representam a tensão de entrada e queda de tensão no LED, respetivamente.

Para $V_{LED} = 1{,}45$ [57] e polarização do LED quando $V_{ENTRADA} \geq 3{,}3V$ mantendo a corrente I_{LED} no intervalo indicado pelo fabricante (entre 5mA e 25mA [57]), e de acordo com a equação (4.4) foi fixado o valor de R_i em 180 Ω.

A resistência de saída, R_O, tem por função regular a corrente na saída. Note-se que existe uma inversão no nível do sinal pois quando o LED se encontra polarizado (nível alto na entrada) devido à polarização do foto-transístor a tensão na saída corresponde ao nível baixo e vice-versa. Quando o foto-transístor não se encontra polarizado, a corrente no coletor do transístor é nula, logo a queda de tensão na resistência é mínima e apenas vai influenciar o tempo de carga do condensador de entrada do bloco seguinte (flip-flop) [58]. Uma resistência muito alta vai amentar o tempo de carga do referido condensador, limitando a velocidade de operação do circuito. Quando o foto-transístor se encontra polarizado a corrente máxima na saída não deverá ser superior a 8 mA, logo pela lei de Ohm [5] temos

$$R_O = \frac{V_{CC}}{I_O} = \frac{5\ V}{8\ mA} = 625\ \Omega \tag{4.5}$$

onde I_O representa a corrente de saída. Para efeitos de implementação, de forma a não operar no limite do componente, foi regulada a corrente I_O para um valor máximo de 5 mA, ou seja, pela equação (4.5) foi utilizada uma resistência R_O de valor igual a 1 kΩ.

4.4.3. Módulo de aquisição

O módulo de aquisição tem a função de converter os sinais a medir para um formato compatível com o microcontrolador, garantindo a fiabilidade das aquisições. À semelhança do módulo de isolamento a aquisição de sinais analógicos e digitais são distintas.

4.4.3.1. Aquisição de sinais analógicos

A aquisição de sinais analógicos é efetuada com recurso a um conversor de analógico para digital (ADC). Apesar de na arquitetura do sistema a ADC ser representada como um elemento externo ao microcontrolador, a grande maioria dos microcontroladores já possuem ADC internas. No entanto, a ADC é um elemento relevante na aquisição de dados e para efeitos de análise e dimensionamento do circuito a ADC é representada externamente.

A ADC do microcontrolador utilizado permite a conversão de sinais numa amplitude de 5V. Pela Figura 4.1 verifica-se que o sistema de aquisição deverá medir a tensão na saída do amplificador de instrumentação que varia numa gama de [-15;15] V. Assim sendo, é necessário um sistema de conversão proporcional do intervalo de entrada para o intervalo [0;5] V, suportado pela ADC.

Numa primeira fase, o sinal original é convertido para uma escala compreendida entre [-2,5; 2,5] V (redução proporcional de seis vezes) através de um divisor resistivo e posteriormente efetuada a soma de um valor constante de 2,5V convertendo assim o sinal para o intervalo de interesse. Na Figura 4.7 é apresentado o circuito de condicionamento do sinal para leitura na ADC.

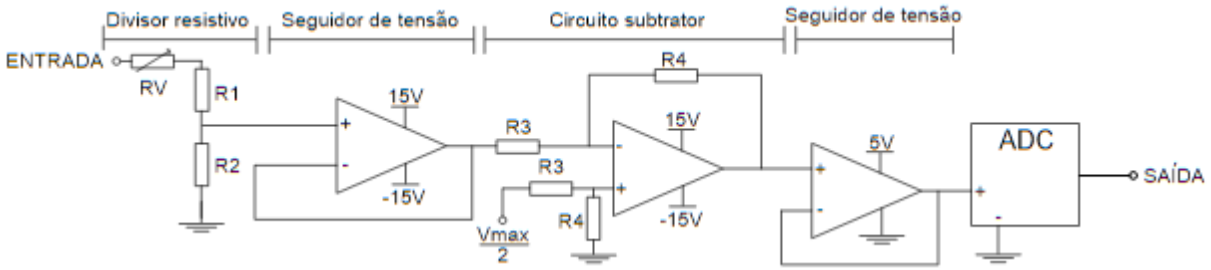

Figura 4.7 – Circuito de aquisição de sinais analógicos numa gama de entrada de módulo 15V.

A relação entre as tensões de entrada e saída do divisor resistivo é dada pela expressão

$$V_{SAÍDA} = \frac{R_2}{R_1 + R_2 + R_V} \cdot V_{ENTRADA} \tag{4.6}$$

onde a resistência variável, R_V, têm por função permitir um ajuste fino.

Pela análise da Figura 4.7 pode constatar-se que a soma do valor constante de 2,5V é efetuada através de um circuito subtrator (subtraindo o valor simétrico), pois permite utilizar um circuito mais pequeno comparativamente a um circuito somador inversor e de mais simples dimensionamento. A relação entre a entrada e saída do circuito subtrator (ou amplificador diferencial) é dada pela expressão

$$V_{SAÍDA} = \frac{R_4}{R_3} \cdot (V_+ - V_-) \tag{4.7}$$

onde V_+ e V_- representam as tensões aos terminais não-inversor e inversor do amplificador operacional, respetivamente. Para $R_3 = R_4$ é obtido um amplificador diferencial de ganho unitário, ou seja, um circuito subtrator.

Por fim é utilizado um seguidor de tensão para evitar que a impedância de entrada do ADC tenha influência no ganho do circuito subtrator, assim como limitar a tensão no intervalo [0;5] V através das tensões de saturação.

A relação entre a tensão de entrada no ADC e a tensão de entrada no bloco de aquisição é então obtida pela expressão

$$V_{ADC} = 2{,}5 + \frac{1}{6} \times V_{ENTRADA} \ , \forall\, V_{ENTRADA} \in [-15; 15] \tag{4.8}$$

onde V_{ADC} e $V_{ENTRADA}$ representam as tensões no ADC e circuito de condicionamento de sinal, respetivamente.

O circuito de condicionamento foi implementado com as resistências apresentadas na Tabela 4.3.

Tabela 4.3 – Resistências utilizadas no circuito de condicionamento de sinal.

Resistência	Valor (kΩ)
R_1	270
R_2	56
R_3	1
R_4	1
R_V	100

Em caso de redução da gama de tensões de entrada do ADC o ajuste da resistência variável e da tensão $V_{MAX}/2$ permite explorar alguma flexibilidade do circuito em se ajustar para diferentes intervalos. O circuito completo de aquisição a baixa velocidade consiste no agrupamento dos circuitos dos blocos de encaminhamento, isolamento por elevada impedância de entrada e aquisição de dados analógicos de acordo com a Figura 4.1 e encontra-se disponível no Anexo F (seção F.1).

4.4.3.2. Aquisição de sinais digitais

O circuito de aquisição utiliza um *flip-flop* (tipo D) para cada pino, com sinal de relógio partilhado por todos os flip-flops, como ilustrado na Figura 4.8.

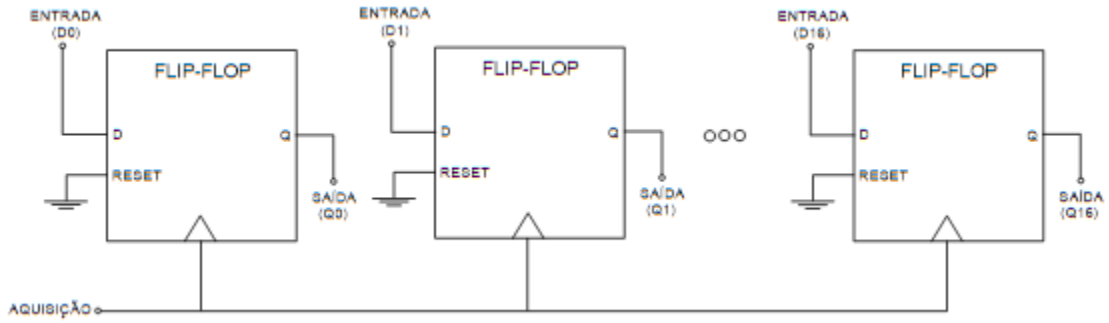

Figura 4.8 – Circuito de aquisição de sinais digitais.

A utilização de *flip-flops* na configuração apresentada na Figura 4.8 garante que a aquisição de todos os sinais são no mesmo instante temporal, condição essencial para garantir a fiabilidade do produto final. O sinal de aquisição é controlado pelo módulo de controlo e deve ser garantido que entre dois sinais de aquisição o microcontrolador efetua a leitura de todos os *flip-flops*.

Na implementação do módulo de aquisição de dados digitais são utilizados flip-flops 74HCT174N, cujas caraterísticas [59] são compatíveis com as necessidades do circuito. As caraterísticas dos *flip-flops* dependem em grande parte dos isoladores óticos utilizados no bloco anterior. No circuito implementado, os isoladores óticos 6N136 são

compatíveis com lógica TTL e CMOS, compatível com a generalidade dos componentes de eletrónica digital.

O circuito completo de aquisição de sinais de frequência normal de funcionamento do circuito digital consiste no agrupamento dos circuitos dos blocos de isolamento ótico e aquisição de dados digitais de acordo com a Figura 4.1 e encontra-se disponível no Anexo F (seção F.2). O circuito completo do sistema de aquisição de dados encontra-se disponível na seção F.3 do mesmo anexo.

4.4.4. Módulo de controlo

O módulo de controlo é responsável por gerir os sinais de controlo de todo o sistema de aquisição de dados, efetuar o registo das medições e comunicar com o computador onde estão alojados os algoritmos de replicação. A Tabela 4.4 apresenta os requisitos necessários do microcontrolador para desempenhar as funções deste módulo, no contexto do protótipo desenvolvido.

Tabela 4.4 – Requisitos mínimos do microcontrolador do módulo de controlo.

Requisito	Tipo ou quantidade
Entradas analógicas:	1
Entradas/saídas digitais:	28
Interrupções externas:	1
Comunicação:	Porta COM ou *Ethernet*
Frequência de operação:	16MHz[18]
Memória interna:	248 KB (ver subseção 4.5)

Considerando as caraterísticas do sistema a implementar o microcontrolador escolhido foi o *Arduíno Mega* 2560 [60], cujas caraterísticas satisfazem os requisitos da Tabela 4.4.

4.4.4.1. Funcionamento geral do módulo de controlo

Quando existe a necessidade de conhecer mais dados do circuito sob análise o computador envia uma instrução ao módulo de controlo, sendo realizadas as tarefas pedidas na instrução e retornadas as medições obtidas, aguardando posteriormente até nova instrução, tal como ilustrado na Figura 4.9.

[18] Na implementação de sistemas de aquisição de dados com maiores velocidades de operação estas caraterísticas assumem grande relevância na escolha do microcontrolador.

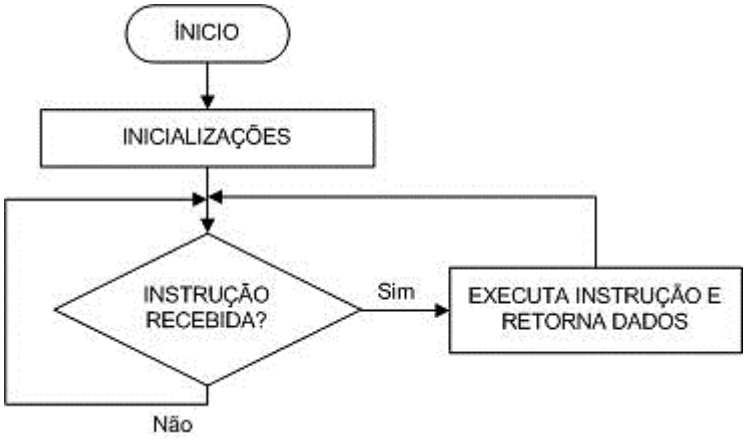

Figura 4.9 – Funcionamento geral do módulo de controlo do sistema de aquisição de dados.

As instruções permitidas e respetivas implementações encontram-se descritas nas subseções seguintes.

4.4.4.2. Sinais de controlo

Os sinais de controlo permitem que múltiplas instruções sejam realizadas sobre o mesmo circuito, permitindo a existência de circuitos mais flexíveis e minimizando a utilização de circuitos dedicados a uma única operação. Neste sistema de aquisição de dados os sinais de controlo desempenham maioritariamente tarefas de encaminhamento de sinal.

O sistema de aquisição de dados possui os sinais de controlo descritos na Tabela 4.5.

Tabela 4.5 – Sinais de controlo do sistema de aquisição de dados.

Sinal de controlo	Designação	Descrição
Seleção Descodificador A[19]	*Sel_A*	Seleciona o relé a ativar do conjunto de relés A
Seleção Descodificador B[19]	*Sel_B*	Seleciona o relé a ativar do conjunto de relés B
Habilitação Descodificador A	*Hab_A*	Habilita o funcionamento do descodificador A (lógica direta)
Habilitação Descodificador B	*Hab_B*	Habilita o funcionamento do descodificador B (lógica direta)
Conexão do sinal de relógio	*Rel_Con*	Ativa o interruptor que encaminha o sinal de relógio
Conexão da referência	*Ref_Con*	Ativa o interruptor que encaminha a referência (GND)
Aquisição de dados digitais	*Dg_Aq*	Sinal de aquisição (relógio) dos flip-flops

[19] Sinal constituído por 4 *bits*

4.4.4.3. Instruções

O módulo de controlo tem a capacidade de realizar cinco instruções distintas, enumeradas na Tabela 4.6.

Tabela 4.6 – Instruções reconhecidas pelo módulo de controlo.

Instrução (Parâmetros)	Código	Descrição
Medição Analógica $(\boldsymbol{N, A, B})$	0001	Realiza N medições analógicas de tensão entre os pinos A e B.
Medição Digital Assíncrona $(\boldsymbol{N, Ref})$	0010	Realiza N medições digitais de todos os pinos em simultâneo para efeitos de análise de circuitos combinatórios. Utiliza o pino indicado em Ref como referência.
Medição Digital Síncrona $(\boldsymbol{N, Rel, Ref, Periodo})$	0011	Realiza N medições digitais de todos os pinos em simultâneo a cada transição do pino Rel, para efeitos de análise de circuitos sequenciais. Utiliza o pino indicado em Ref como referência.
Deteção de Transição $(\boldsymbol{N, Rel, Ref})$	0100	Regista o intervalo de tempo entre transições durante N transições do sinal do pino Rel. Utiliza o pino indicado em Ref como referência.
Confirmação de ligação ()	1111	Confirma a existência de uma ligação entre o computador e o sistema de aquisição de dados.

Cada instrução possui um código identificativo único, e é enviada para o microcontrolador com a estrutura da Figura 4.10.

CÓDIGO : Nº MEDIÇÕES : ARG 1 : ARG 2 : ... : ARG N:-

Figura 4.10 – Estrutura de uma instrução.

O fim da instrução é indicado pelo carater "-" e os parâmetros de cada instrução encontram-se separados pelo carater ":", descritos abaixo:

- Código: identificador único de cada instrução;
- Medições: define o número de medições a realizar na execução da instrução;
- Argumento 1, Argumento 2: definem parâmetros de encaminhamento de sinal;
- Argumento 3: Define o período do sinal de relógio na instrução de medição digital síncrona. Nas restantes instruções são necessários no máximo dois argumentos.

Definidas as instruções e respetivos argumentos, o módulo de controlo faz aplicar os sinais de controlo conforme a instrução recebida de acordo com a Tabela 4.7.

Tabela 4.7 – Relação entre as instruções e os sinais de controlo.

Instrução				Sinais de controlo						
Código	**Medições**	**Arg1**	**Arg2**	**Sel_A**	**Sel_B**	**Hab_A**	**Hab_B**	**Rel_Con**	**Ref_Con**	**Dg_aq**
0001	N	A	B	A	B	1	1	0	0	0
0010	N	A	-	-	A	0	1	0	1	0
0011	N	A	B	A	B	1	1	1	1	
0100	N	A	B	A	B	1	1	1	1	0

4.4.4.4. Implementação das instruções

Nas subseções seguintes será descrita a implementação das instruções suportadas pelo microcontrolador. O código fonte completo utilizado para programação do microcontrolador encontra-se listado no Anexo F (seção F.4).

A. <u>Medição analógica</u>

Considerando a arquitetura implementada, a aquisição de medições analógicas implica a identificação de um par de pinos, selecionáveis através dos argumentos da instrução. O valor resultante da medição, $V_{ANALÓGICO}$, é dado pela expressão

$$V_{ANALÓGICO} = V_{ARG1} - V_{ARG2} \tag{4.9}$$

onde V_{ARG1} e V_{ARG2} representam as tensões nos pinos definidos nos argumentos 1 e 2 da instrução, respetivamente. Esta instrução realiza medições diferenciais, permitindo também a realização de medições associadas a uma referência (indicando a referência no argumento 2). A execução desta instrução encontra-se ilustrada na Figura 4.10.

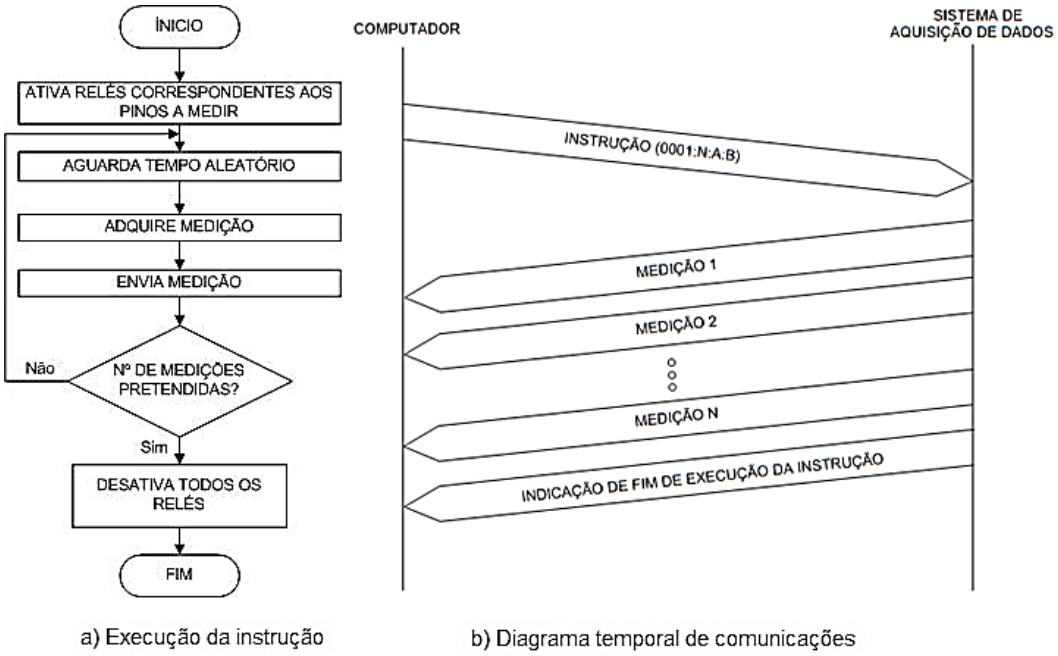

Figura 4.10 – Execução da instrução de medição analógica.

Pela análise da Figura 4.10 verifica-se a existência de um tempo aleatória entre medições. A instrução para realização de medições analógicas é requerida ao sistema de aquisição de dados nas etapas de deteção das alimentações e despiste de sinais não digitais, logo é necessário que o intervalo entre aquisições não seja constante de forma a evitar que a aquisição de medições de um sinal periódico ocorram sempre no mesmo instante em diferentes ciclos. A utilização de um intervalo de tempo constante entre medições de um sinal periódico cujo período é muito próximo ou múltiplo do intervalo de tempo entre medições, pode resultar num conjunto de medições que conduzirão a uma interpretação errada do sinal medido, originando a classificação de um sinal variável no tempo como constante.

B. Medição digital assíncrona

Esta instrução realiza a medição de todos os pinos do circuito integrado sob análise relativamente a um pino de referência indicado no segundo argumento da instrução. A execução desta instrução consiste essencialmente em atribuir os sinais de controlo para encaminhamento do sinal de referência e controlo do sinal de aquisição, como ilustrado na Figura 4.11.

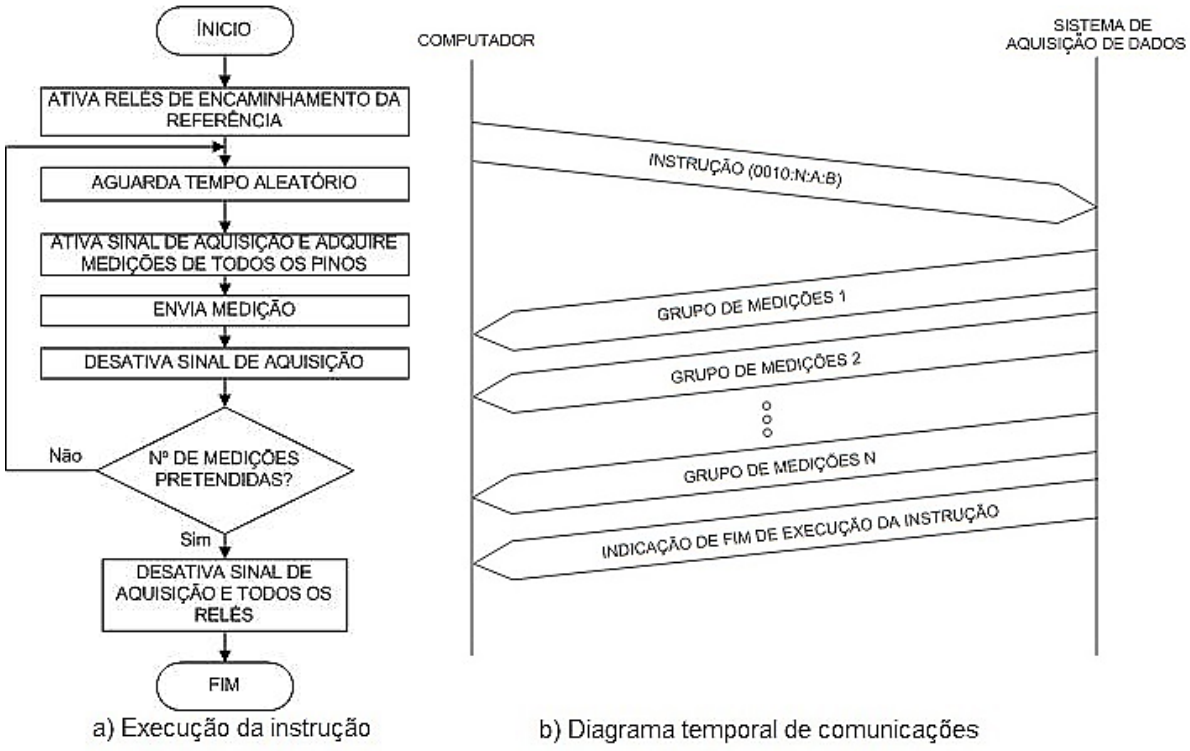

Figura 4.11 – Execução da instrução de medição digital assíncrona.

Pela análise da Figura 4.11 verifica-se que a aquisição é realizada em intervalos de tempos não periódicos. Este assincronismo destina-se a evitar interpretações erradas

das medições de sinais periódicos, à semelhança do referido anteriormente na execução de medições analógicas.

As medições efetuadas na execução desta instrução são digitais e devido ao isolamento ótico aplicado (já descrito na secção 4.4.2.2 deste documento) as medições obtidas e enviadas pelo sistema de aquisição de dados encontram-se em lógica negada.

O sinal de aquisição de dados consiste no sinal de relógio dos *flip-flops* que efetuam a aquisição dos sinais. Como os *flip-flops* utilizados (74HCT174 [59]) realizam a aquisição na transição ascendente do sinal de relógio o sinal de aquisição funciona em lógica direta, contudo pode ser necessário ajustar o sinal de aquisição em função dos componentes utilizados.

C. *Medição digital síncrona*

A instrução de aquisição de medições digitais síncronas realiza as mesmas medições que a instrução descrita na subsecção anterior, com a variante que os instantes de aquisição são definidos pelas transições de um sinal de relógio (pino de relógio é um dos argumentos da instrução) ao invés de ser realizada a intervalos temporais não constantes. A deteção das transições do sinal de relógio é efetuada através da utilização de uma interrupção externa (ativada em cada transição), e a execução desta instrução encontra-se esquematizada na Figura 4.12.

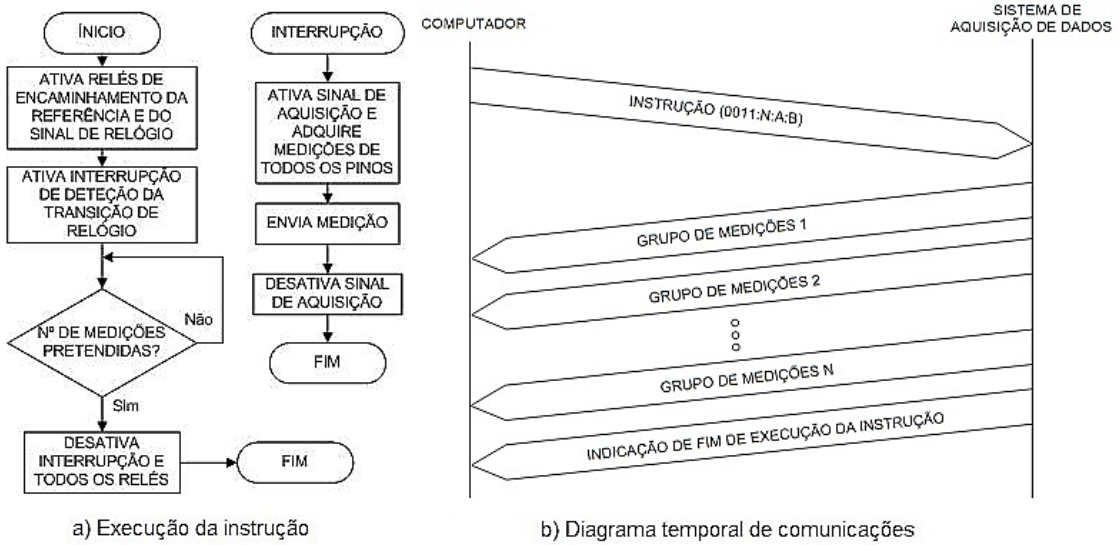

Figura 4.12 – Execução da instrução de medição digital síncrona.

Um fator a ter em conta na aquisição sincronizada é o tempo de estabilização dos sinais após a transição do sinal de relógio, pois a fiabilidade das medições pode ser afetada pela aquisição de um sinal durante a sua transição. Na Figura 4.13 encontra-se apresentado um diagrama onde é considerado que o tempo de subida e descida dos sinais é muito superior ao tempo de transição do sinal de relógio. Após a transição do sinal de relógio é aguardado um tempo, T_e, e posteriormente realizada a aquisição (A_q).

O tempo de espera (aguarda estabilização dos sinais) para aquisição é calculado a partir do período do sinal de relógio, que é um dos argumentos da instrução.

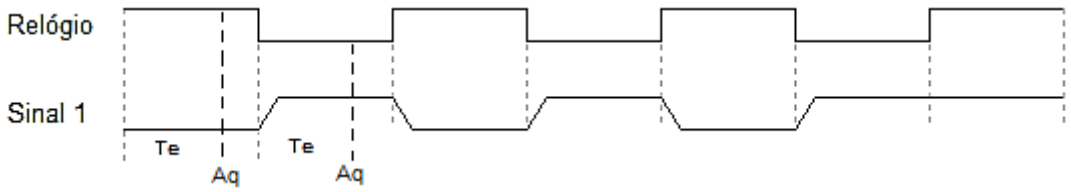

Figura 4.13 – Aquisição de medições após uma transição do sinal de relógio.

O intervalo de tempo T_e é dado por

$$T_e = n.\frac{T}{2} \ , n \in]0,1[\tag{4.10}$$

onde T e n representam o período do sinal de relógio e um fator de ajuste. Para efeitos de implementação foi utilizado $n = 0{,}9$ uma vez que os sinais já se encontram estabilizados devido à proximidade da transição do sinal de relógio.

Para uma correta realização desta instrução é necessário que todas as medições sejam realizadas antes da transição seguinte do sinal de relógio, caso contrário as medições obtidas não serão fidedignas. Na implementação desta instrução é necessário ter em conta o número de comandos que o microcontrolador tem de executar a cada transição, nomeadamente quando o número de pinos do circuito sob análise é inferior a 16 (número máximo de pinos possíveis). Foi analisado e decidido que, independentemente do número de pinos a medir, é realizada a medição de todos pinos pois os comandos para excluir medições sem significado implicariam um maior custo computacional, tendo então sido tomada a opção em adquirir todas as medições e posteriormente descartar as medições sem significado para o algoritmo de replicação.

D. <u>Deteção de transição</u>

A deteção consiste na execução de uma instrução que deteta a transição de um sinal e regista o tempo decorrido até ao momento em que a transição ocorre. No caso implementado o microcontrolador (*Arduíno* Mega) tem a função que retorna o tempo que passou desde o início do funcionamento do sistema. De forma a minimizar as operações no microcontrolador entre transições são enviados os dados brutos (tempo desde o início do funcionamento até à ocorrência da transição) para o computador sendo todo o processamento (subtração dos tempos de duas transições consecutivas para obter intervalo entre transições) realizados fora do microprocessador quando existe mais tempo disponível para processamento. O sistema implementado é modular e permite o melhoramento de diferentes partes do sistema, porém é necessário ter em conta a forma como os dados são disponibilizados pelo sistema de aquisição de dados, como é exemplo esta instrução.

Esta instrução é utilizada para fornecer dados para detetar algoritmicamente qual dos pinos corresponde ao sinal de relógio, quando detetado um circuito sequencial. O

sinal a analisar é indicado num dos argumentos da instrução e cada transição do mesmo irá ser tratada através de uma interrupção externa, como ilustrado na Figura 4.14.

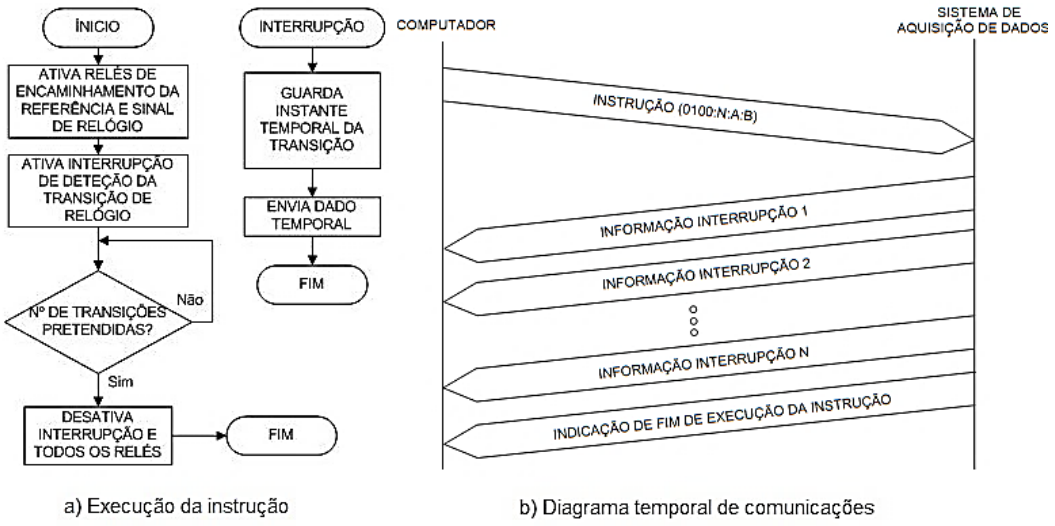

Figura 4.14 – Execução da instrução de deteção de transições.

Esta instrução foi concebida para ser executada continuamente até o número de transições pretendidas ser atingido. No caso de ser aplicada esta instrução a um sinal constante o número de transições nunca será atingido, pelo que é necessário adicionar um mecanismo de proteção, através da existência de um *timer* que limite o tempo máximo em que é possível ficar à espera de uma transição, garantindo assim que a instrução nunca fica em funcionamento infinito.

Para sistemas de maior frequência de operação onde o intervalo entre transições é menor ou utilização de microcontrolador cujas interrupções apenas sejam ativadas por uma transição ascendente ou descendente (aos invés de ambas) a utilização de um circuito detetor de transições (ver seção F.5 do Anexo F) pode ser importante no melhoramento do desempenho do sistema.

E. Confirmação da ligação

Quando o sistema recebe um pedido de confirmação de ligação, envia uma resposta afirmativa, como apresentado na Figura 4.14

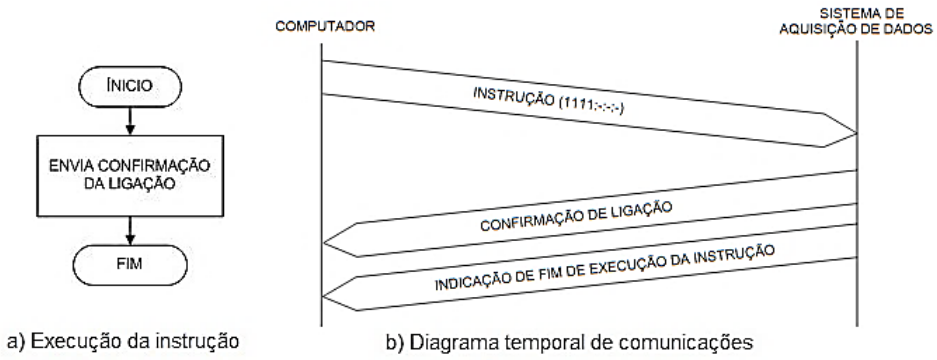

Figura 4.15 – Execução da instrução de confirmação de ligação.

A confirmação de ligação é uma instrução realizada exclusivamente para verificar a conexão entre o sistema de aquisição de dados e o computador onde estão alojados os algoritmos.

Esta instrução pode ser interpretada como a instrução equivalente ao comando *ping* [61] utilizados em redes de comunicação de dados.

No Anexo G encontra-se apresentado sistema de montagem implementado experimentalmente.

4.5. Considerações para operação em alta frequência

Para um sistema de aquisição de dados com capacidade de adquirir dados a alta frequência são necessárias algumas alterações ao nível dos componentes utilizados, nomeadamente a utilização de componentes com largura de banda compatível com a frequência de aquisição pretendida. Ao implementar o circuito numa placa de circuito impresso é também aconselhável a utilização de um roteamento de sinal compatível com a frequência a que o circuito vai operar.

Ao nível da arquitetura do sistema de aquisição de dados, podem ser necessárias alterações de forma a adicionar ao sistema uma memória, dado que com o aumento da frequência a quantidade de dados recolhidos por segundo aumenta e a capacidade de transmissão de dados do sistema pode não ser suficientemente elevada. A utilização de uma memória é necessária quando

$$M_{DADOS} - (V_{LIG} + M_{INT}) > 0 \tag{4.10}$$

onde M_{DADOS}, V_{LIG} e M_{INT} representam a memória necessária para armazenar os dados, a taxa de transmissão de dados entre o sistema de aquisição de dados e o computador e a memória interna do microcontrolador, respetivamente. As parcelas da equação (4.10) devem são medidas em *bytes*/s ou múltiplos. Quando uma memória é adicionada ao sistema a capacidade mínima necessária da memória é dada pela manipulação da equação (4.10), sendo dada por

$$M_{ADICIONAL} > M_{DADOS} - (V_{LIG} + M_{INT}) \tag{4.11}$$

onde $M_{ADICIONAL}$ representa a capacidade da memória em *bytes* ou múltiplos (dependendo da ordem e grandeza utilizada nas restantes parcelas). A velocidade de transferência de dados para a memória adicionada deverá ser compatível com a velocidade de operação do sistema.

Quando V_{LIG} não é um valor constante, deve ser considerado o valor mínimo.

5. Resultados

5.1. Introdução

Neste capítulo serão apresentados os resultados obtidos experimentalmente através da replicação de circuitos integrados digitais (combinatórios e sequenciais) utilizando o protótipo implementado, de forma a validar os algoritmos desenvolvidos. Serão ainda analisados os fatores que influenciam as réplicas obtidas e comparados os resultados obtidos para circuitos combinatórios e sequenciais.

5.2. Resultados obtidos para circuitos combinatórios

5.2.1. Análise do impacto do número de amostras recolhidas

No caso dos circuitos combinatórios a capacidade de replicação do sistema desenvolvido é altamente influenciada pelo número de amostras recolhidas dado que a fonte de dados do sistema é baseada na observação do funcionamento. Assim sendo, nenhuma combinação pode ser forçada por ação externa, logo toda a análise será baseada na probabilidade de todas as combinações ocorrerem naturalmente. De forma a tornar a análise independente do número de entradas do circuito, o impacto das amostras recolhidas é analisado através da razão entre o número de amostras recolhidas, R, e o número de combinações de entrada possíveis, M ($M = 2^n$, onde n representa o número de pinos de entrada). A verificação experimental da influência do número de amostras foi realizada através da replicação de circuitos combinatórios com variação do número de amostras recolhidas e combinações de entrada possíveis e registo da taxa de sucesso estimada pelo sistema. Os resultados obtidos encontram-se apresentados na Figura 5.1 e podem ser consultados com maior detalhe no Anexo H.1.

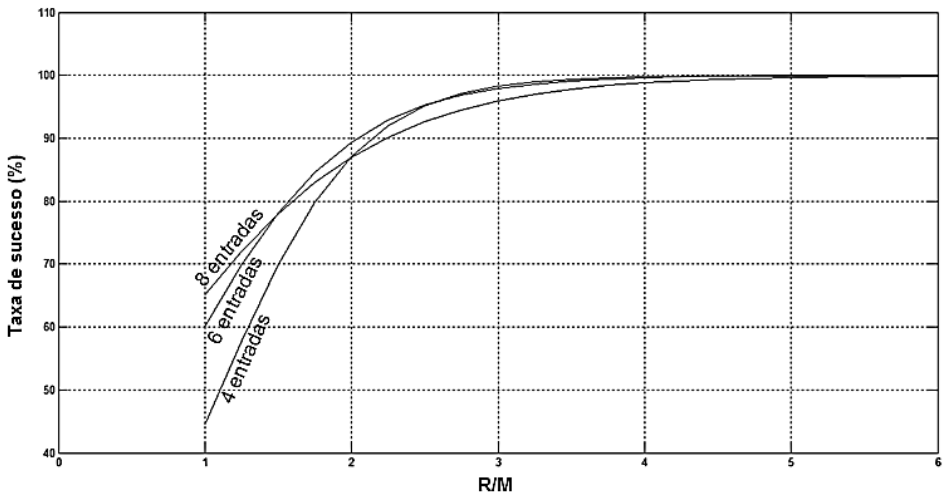

Figura 5.1 – Evolução da taxa de sucesso em função do número de amostras.

Pela análise da Figura 5.1 é possível verificar que a taxa de sucesso tende para 100% com o aumento da relação R/M, coincidente com o evolução demonstrada matematicamente pela equação (3.31). Esta convergência confirma experimentalmente que quando o número de amostras recolhidas é muito superior ao número de combinações de entrada possíveis, a replicação completa do circuito combinatório é um acontecimento muito provável. Quantitativamente, através dos dados obtidos experimentalmente, quando o número de amostras recolhidas é, no mínimo, cinco vezes superior ao número de combinações de entrada possíveis ($R = 5M$), é muito provável que o circuito tenha sido replicado na totalidade, independentemente do número de combinações de entrada.

5.2.2. Análise das réplicas obtidas

De forma a analisar a capacidade de replicação do sistema desenvolvido, no que respeita a circuitos combinatórios, foram submetidos a replicação os circuitos integrados apresentados na Tabela 5.1. O sistema de replicação implementado apenas tinha conhecimento (fornecido pelo utilizador) dos pinos de saída do circuito sob teste, não sendo conhecida qualquer outra caraterística.

Tabela 5.1 – Circuitos integrados combinatórios utilizados para teste do sistema.

Teste	Tipo de circuito	Componentes [20]		Referência	Tipo de lógica
		PL	Mult.		
#1	Porta *AND*	4	0	74LS08 [62]	TTL (lógica direta)
#2	Porta *OR*	4	0	74HCT32 [63]	CMOS (lógica direta)
#3	Porta *XOR*	4	0	74LS86 [64]	TTL (lógica direta)
#4	Descodificador	17	0	74HC139 [65]	CMOS (lógica negada)
#5	Conversor BCD - 7 segmentos	37	0	74LS48 [66]	TTL (lógica direta)
#6	Porta NOR[21]	4	0	CD4001[67]	CMOS (lógica direta)

Utilizando o *software ISE Design* da *Xilinx* [45], foi sintetizado o código VHDL gerado pelo sistema de replicação e realizadas simulações para aferir o comportamento da réplica. Nas verificações experimentais realizadas os circuitos sob análise foram estimulados artificialmente utilizando um microcontrolador[22] auxiliar para efetuar a geração de sinais aleatórios nas entradas do circuito integrado, ou seja, apesar dos circuitos sob teste não se encontrarem inseridos num sistema maior, a estimulação provocada intencionalmente cria um ambiente de funcionamento equivalente para o qual o sistema foi concebido.

[20] Legenda: PL- Portas Lógicas; *Mult*: *Multiplexers*.
[21] Teste realizado a circuito montado por terceiros (teste cego).
[22] Foi utilizado o Arduino Uno [68]

Os resultados obtidos encontram-se apresentados na Tabela 5.2, onde é estabelecida uma relação entre a taxa de sucesso, o número de componentes sintetizados e a correspondência estrutural e comportamental entre a réplica e as especificações técnicas dos circuitos integrados.

Tabela 5.2 – Resultados obtidos para circuitos combinatórios.

Taxa (%)	Teste	Componentes		Correspondência estrutural	Correspondência comportamental	
		PL	*Mult*.		Observada	Completa
100	#1	4	0	Total[23]	Total	Total
	#2	4	0	Total[22]	Total	Total
	#3	20	0	Equivalente[22]	Total	Total
	#4	24	0	Equivalente [22]	Total	Total
	#5	112	10	Inexistente[22]	Total	Total
	#6[24]	4	0	Equivalente	Total	Total
90-99	#1	≥100	≥9	Inexistente	Total	Parcial
	#2	≥100	≥2	Inexistente	Total	Parcial
	#3	≥100	≥5	Inexistente	Total	Parcial
	#4	≥100	≥10	Inexistente	Total	Parcial
	#5	≥115	≥10	Inexistente	Total	Parcial
<90	#1	≥150	≥12	Inexistente	Total	Parcial
	#2	≥150	≥8	Inexistente	Total	Parcial
	#3	≥150	≥10	Inexistente	Total	Parcial
	#4	≥150	≥10	Inexistente	Total	Parcial
	#5	≥150	≥10	Inexistente	Total	Parcial

Pela análise da Tabela 5.2 verifica-se que quando a taxa de sucesso é igual a 100%, a correspondência comportamental é absoluta, isto é, corresponde aos comportamentos descritos nas folhas de caraterísticas do circuitos e, por conseguinte, ao comportamento observado. Por outras palavras, foi verificado experimentalmente que quando todas as combinações possíveis são observadas num circuito combinatório a réplica obtida corresponde (funcionalmente) à totalidade do circuito, inferindo-se desta forma que estas réplicas podem substituir o circuito em qualquer circunstância. Ao invés, quando a taxa de sucesso é inferior a 100%, foi confirmado experimentalmente que a correspondência comportamental existe apenas em relação ao comportamento que foi observado, ocorrendo uma correspondência parcial das funcionalidades do circuito integrado. Nestas situações, como a réplica obtida não corresponde à totalidade do circuito, não existe garantia que a réplica funcione corretamente em qualquer circunstância. Quando a réplica se destina a ser utilizada inserida no sistema onde foi observada é provável que funcione corretamente, contudo não pode ser excluída a hipótese de ocorrer uma combinação não observada, ou seja, é impossível quantificar com exatidão a probabilidade de funcionamento de uma réplica no sistema onde foi observada quando a taxa de sucesso é inferior a 100%.

[23] O circuito resultante após sintetização pode ser consultado no Anexo H.
[24] A montagem do teste #6 foi realizada por terceiros sendo o objetivo descobrir o máximo possível sobre funcionamento reproduzindo situação de não conhecimento prévio do utilizador sobre o circuito

Estruturalmente, só é obtida total correspondência em alguns casos onde a taxa de sucesso é igual a 100% e depende das caraterísticas do sintetizador. À medida que a taxa de sucesso diminui correspondência estrutura diminui, pois devido à menor quantidade de dados disponíveis não é possível simplificar ao máximo a circuito resultante. O tipo de portas utilizadas pelo sintetizador influência a correspondência estrutural como, por exemplo, no caso das portas *XOR* (teste #3), onde a réplica foi gerada com portas do tipo *NOT*, *AND* e *OR* devido às caraterísticas do sintetizador, o que provocou um aumento do número de portas lógicas utilizadas. A disparidade entre o número de portas lógicas do circuito original e a réplica verificada no caso do descodificador (teste #4) e mais acentuadamente no conversor de BCD para 7 segmentos (teste #5) está associado à maior complexidade dos circuitos e em particular ao maior número de entradas e saídas com dependência entre si. Resumidamente, pelos resultados apresentados na Tabela 5.2, devido à baixa correspondência estrutural, é possível inferir que o sistema desenvolvido é orientado à replicação do comportamento (ver simulações no Anexo H) em detrimento do esquema lógico, de acordo com o planeamento realizado e com os objetivos estabelecidos para o sistema.

Pela análise das Tabela 5.1 e 5.2 é possível verificar que o sistema replicou diferente famílias lógicas (TTL e CMOS) bem como diferentes tipos de lógica (direta e negada), sem implicação nas taxas de sucesso estimadas, uma vez que em qualquer caso o sistema de replicação alcançou a taxa de sucesso máxima, quando a monitorização efetuada o permite. Este facto permite também concluir que o sistema de aquisição de dados funciona corretamente para uma gama genérica de circuitos estruturalmente distintos.

5.3. Resultados obtidos para circuitos sequenciais

5.3.1. Análise do impacto do número de amostras recolhidas

Ao contrário do verificado nos circuitos combinatórios o número de amostras recolhidas não tem influência direta na taxa de sucesso estimada para circuitos sequenciais. Foi verificado experimentalmente que a precisão das réplicas obtidas aumenta com o aumento do número de amostras recolhidas, embora a precisão depende essencialmente da relação entre o número de estados gerados internamente pelo algoritmo e o número de amostras recolhidas.

5.3.2. Caraterísticas do protótipo desenvolvido

As caraterísticas do sistema desenvolvido que influenciam os circuitos sequenciais encontram-se apresentadas na Tabela 5.3.

Tabela 5.3 – Limitações do sistema desenvolvido.

Caraterística	Aplicação	Limite
Frequência máxima do sinal de relógio	Para deteção	31,25 kHz
	Para aquisição	2 kHz
Número máximo de amostras para análise sequencial	Para aquisição	2000
	Para processamento	ilimitado

5.3.3. Análise das réplicas obtidas

De forma a analisar a capacidade de replicação do sistema desenvolvido, no que respeita a circuitos sequenciais, foram sujeitos a replicação circuitos integrados de funcionamento conhecido, apresentados na Tabela 5.4. À semelhança dos testes realizados aos circuitos combinatórios, o sistema de replicação implementado apenas tinha conhecimento (fornecido pelo utilizador) dos pinos de saída do circuito sob teste, não sendo conhecida qualquer outra caraterística. Ao longo desta secção serão analisadas as réplicas obtidas e a influência do sistema onde se encontram inseridos na réplica obtida através do sistema desenvolvido.

Tabela 5.4 – Circuitos integrados sequenciais utilizados para teste do sistema de replicação.

Teste	Tipo de circuito	Referência	Tipo de lógica
#1	Contador	74LS393 [69]	TTL (lógica direta)
#2	Conversor série-paralelo	74HCT164 [70]	CMOS (lógica direta)
#3	Flip-Flop (D)	74HCT174 [59]	CMOS (lógica direta)
#4	Conversor série-paralelo	74HC595 [71]	CMOS (lógica direta)
#5	Flip-Flop (JK)	74LS76 [72]	TTL (lógica direta)

Os testes aos circuitos integrados apresentados na Tabela 5.4 foram realizados de forma idêntica aos circuitos combinatórios e sintetizados utilizando o *software ISE Design* da *Xilinx* [45].

5.3.3.1. Análise do impacto da ocorrência do sinal de *reset*

Num circuito sequencial a ocorrência do sinal de *reset* é importante para a obtenção da réplica, visto que é uma funcionalidade importante que a maioria dos circuitos integrados possui. Porém, situações de *reset* ocorrem, regra geral, poucas vezes durante o funcionamento do circuito e uma vez que a replicação efetua é baseada em observação a não observação de situações de *reset* durante a replicação é possível.

A não ocorrência do sinal de *reset* durante a observação tem impacto na réplica obtida, como pode ser demonstrado através de testes efetuados ao circuito integrado 74LS393[25].

De acordo com [69], o circuito integrado 74LS393 consiste em dois contadores de 4 *bits* e para efeitos de teste foram utilizadas as montagens de contador simples e duplo, apresentadas na Figura 5.2.

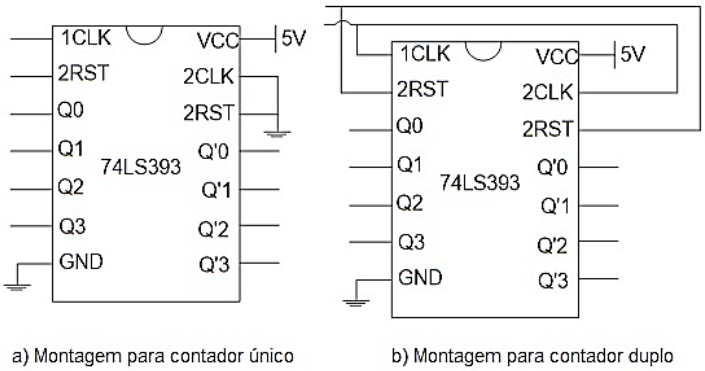

Figura 5.2 – Montagens para estimulação do circuito integrado 74LS393.

Para efeito da análise do sinal de *reset* foi utilizada a montagem de contador simples apresentada na Figura 5.2. Os diagramas de transições das máquinas de estados replicadas em função da ocorrência do sinal de *reset* estão representados na Figura 5.3 (os estados com fundo preto representam o estado inicial).

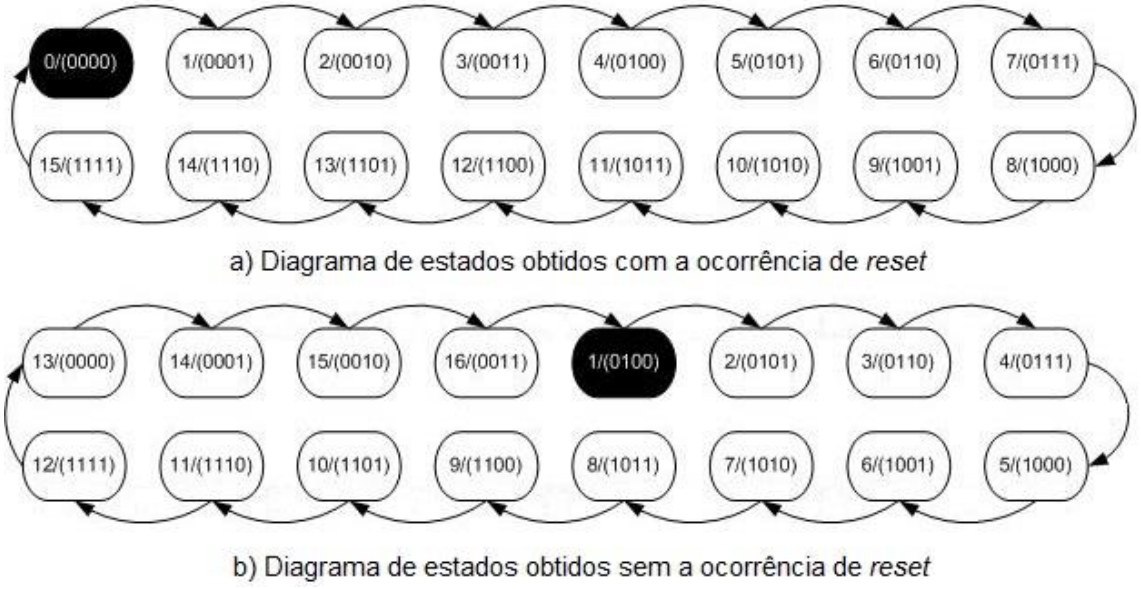

Figura 5.3 – Diagramas de transição de estado das réplicas obtidas em função da ocorrência do sinal de *reset*. (Legenda: Estado/ (Saída)).

Pela análise da Figura 5.3 verifica-se que em ambas as situações o número de transições e estados identificados permanece inalterado, bem como a sequência de estados. No entanto, verifica-se que o estado inicial (assinalado com fundo negro) é

[25] A não ocorrência de situações de *reset* têm impacto semelhante em todos os circuitos integrados sequenciais, no entanto o impacto é no circuito 74LS393 é facilmente visível.

alterado em função da ocorrência do sinal de *reset*: quando ocorre, pelo menos uma situação de *reset,* o estado para onde é realizada a transição é definido como estado inicial e é realizada uma transição para este estado sempre que o pino de *reset* é ativado, contrariamente ao que ocorre quando nenhuma situação de *reset* é detetada onde o primeiro estado identificado é definido como estado inicial, sendo definido estaticamente e não existindo, após sintetização, qualquer relação entre o pino de *reset* e um estado inicial.

O código VHDL gerado é influenciado pela ocorrência de *reset* como pode ser observado nos excertos de código apresentados na Figura 5.4.

```
shared variable curr_reply_state: reply_states;
shared variable next_reply_state: reply_states;
signal reset : STD_LOGIC ;
begin
process (CLK) is
begin
    if (reset = '1') then
        curr_reply_state:= REPLY_STATE_0;
    elsif (falling_edge(CLK)) then
        curr_reply_state := next_reply_state;
    end if;
end process;
```

a) Com ocorrência de *reset*

```
shared variable curr_reply_state: reply_states:=REPLY_STATE_1;
shared variable next_reply_state: reply_states;
signal reset : STD_LOGIC :='0';
begin
process (CLK) is
begin
    if (reset = '1') then
        curr_reply_state:= REPLY_STATE_1;
    elsif (falling_edge(CLK)) then
        curr_reply_state := next_reply_state;
    end if;
end process;
```

b) Sem ocorrência de *reset*

Figura 5.4 – Código VHDL gerado em função da ocorrência de *reset*.

Com a não ocorrência do sinal de *reset* o pino de *reset* é definido como GND ou VCC (dependendo se o *reset* é ativado no nível alto ou baixo) e é atribuída à variável *reset*[26] um valor inicial (zero) que nunca será alterado, uma vez que não foi detetado pino de *reset*, isto é, estruturalmente a réplica não estará preparada para reproduzir situações de *reset*. Este comportamento foi igualmente verificado em outros circuitos integrados sujeitos a teste. Para melhor compreensão do impacto da ocorrência de reset, os códigos VHDL gerados em função da ocorrência de *reset* para a montagem de contador único apresentada na Figura 5.2 encontram-se disponíveis no Anexo I (seção I.1). Em suma, quando não é observada a ocorrência de uma situação de *reset* as transições e os estados ocorridos durante a observação serão igualmente detetados e reproduzidos, porém não será possível retornar a um estado inicial por via de uma ação de *reset*.

5.3.3.2. Análise do impacto do sistema envolvente

Como a descrição da réplica em linguagem VHDL é comportamental em função da observação realizada, o sistema onde o circuito integrado sob teste se encontra inserido influencia a réplica obtida. Por outras palavras, diferentes estimulações podem ser aplicadas ao mesmo circuito integrado, por exemplo, quando este se encontra

[26] A variável *reset* está associada ao pino de *reset* e funciona sempre em lógica direta (ativo no nível alto) e é atualizada por um bloco *process* dedicado a verificar as variações do pino de *reset*.

inserido em sistemas diferentes, podendo existir estados que não são atingidos devido à não ocorrência dos sinais de estimulação necessários.

Para efeitos de análise do impacto do sistema envolvente na réplica obtida, foram efetuados testes ao circuito integrado 74HCT164 (conversor série-paralelo), com diferentes montagens (ver Figura 5.5) que permitem inferir sobre o impacto causado pelo sistema envolvente ao circuito no diagrama de estados da réplica.

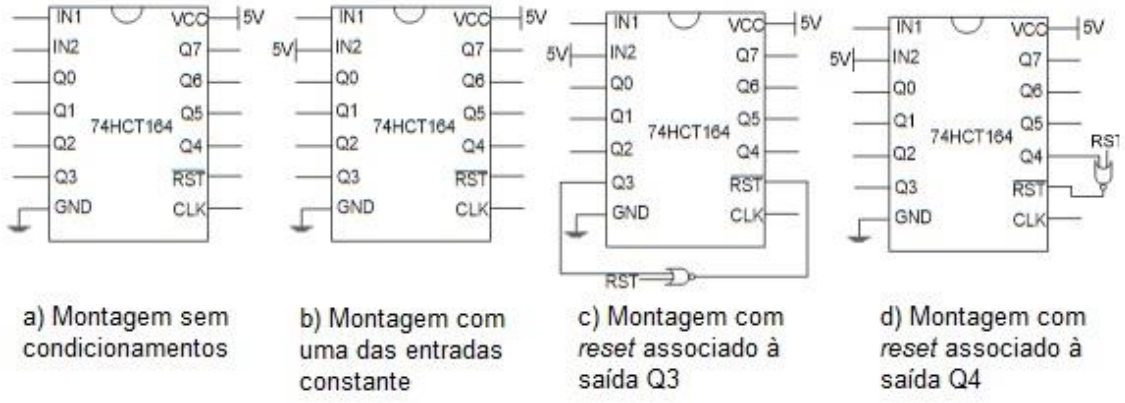

Figura 5.5 – Montagens para estimulação do circuito integrado 74HCT164 (conversor série paralelo).

De acordo com [70], o circuito integrado 74HCT164 realiza o deslocamento lógico para a esquerda (de Q0 para Q7) sendo o *bit* menos significativo (Q0) dado pelo produto das entradas 1 e 2 (IN1 e IN2). O *reset* ocorre quando o pino $\overline{RST}$ se encontra no nível baixo e o deslocamento é realizado na transição ascendente do sinal de relógio.

Para análise da influência do sistema envolvente serão consideradas as réplicas obtidas para as montagens c) e d) da Figura 5.5. Na montagem c) verifica-se que o *reset* ocorre quando o sinal RST se encontra no nível alto ou quando o pino de saída Q3 assume o valor '1'. A montagem d) realiza um condicionamento similar ao da montagem c), diferindo apenas no pino de saída que provoca *reset*, neste caso o pino Q4. Os testes realizados à montagem c) com geração de sinais aleatórios e ocorrência de *reset* resultaram numa réplica onde os pinos Q3, Q4, Q5, Q6 e Q7 foram classificados como GND[27] e o diagrama de transições obtido encontra-se apresentado na Figura 5.6.

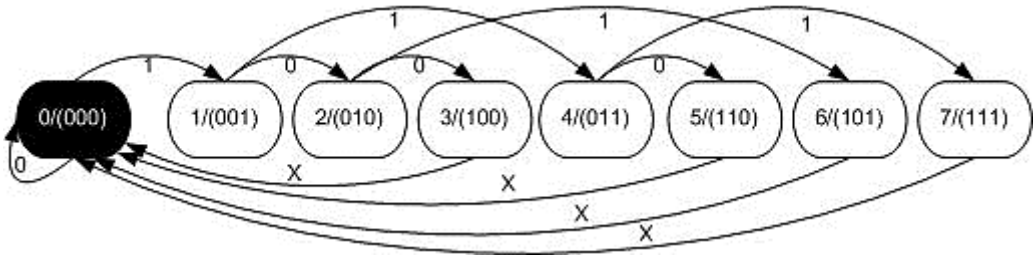

Figura 5.6 – Diagrama de estados obtido para a montagem com *reset* associado à saída Q3 (Legenda: Estado/ (Saída)).

[27] O pino Q3 é classificado como GND pois o tempo em que permanece no nível alto é insignificante.

Testes realizados à montagem d), com as mesmas condições de teste (embora possam ter sido observados estados por ordem diferente), resultaram numa réplica onde os pinos Q4, Q5, Q6 e Q7 foram classificados como GND[28] e o diagrama de transições obtido encontra-se apresentado da Figura 5.7.

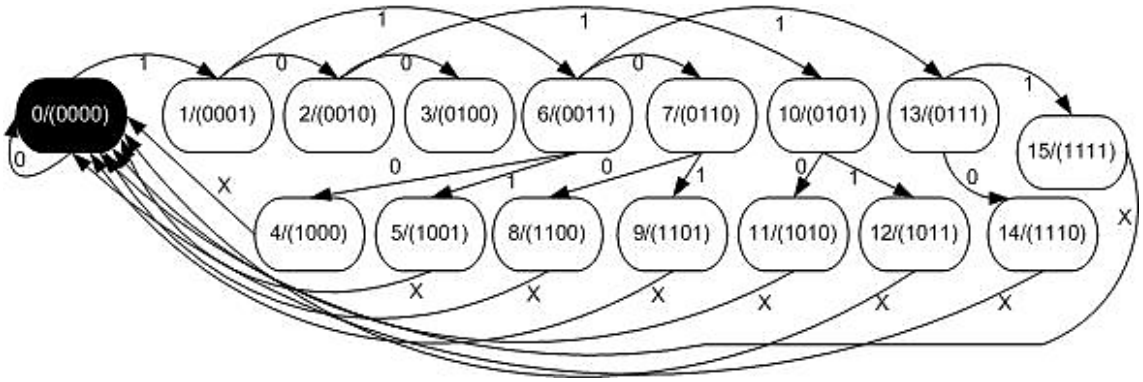

Figura 5.7 – Diagrama de estados obtido para a montagem com *reset* associado à saída Q4 (Legenda: Estado/ (Saída)).

Pela comparação entre as Figuras 5.6 e 5.7 pode facilmente inferir-se que para o mesmo circuito integrado podem obter-se réplicas com diferentes diagramas de transições. Diagramas de transições/estados diferentes ocorrem igualmente para diferentes montagens dos circuitos integrados apresentados na Tabela 5.3.

Isto ocorre porque a réplica reflete o comportamento do circuito integrado ao invés da sua estrutura interna, logo quando o meio envolvente impõe diferentes estímulos ao circuito, diferentes transições e/ou estados podem ser observados ou ainda podem impedir que alguns estados e transições sejam observados. Por outras palavras, as condicionantes impostas pelo sistema onde o circuito integrado se encontra inserido não podem ser ultrapassadas pelo sistema de replicação desenvolvido que se limita a efetuar a observação do funcionamento do circuito integrado, logo a réplica irá refletir unicamente as transições e estados observados, que podem ser distintos para diferentes em função do meio onde o circuito integrado está inserido.

5.3.3.3. Análise das taxas de sucesso obtidas

Nas análises efetuadas às réplicas de circuitos combinatórios, em subseções anteriores, foram utilizados testes onde a taxa de sucesso estimada pelo algoritmo é igual a 100%. Porém, no caso dos circuitos sequenciais, a taxa de sucesso estimada pelo algoritmo reflete a percentagem de transições encontradas em função do número de estado detetados e o número de transições possíveis para cada estado. Assim sendo, esta taxa de sucesso não reflete a percentagem do circuito integrado que foi descoberta tal como sucedia nos circuitos combinatórios, ou por outras palavras, mesmo

[28] O pino Q4 é classificado como GND pois o tempo em que permanece no nível alto é insignificante.

que a réplica de um circuito sequencial tenha uma taxa de sucesso estimada em 100% não é garantido que tenha sido replicada a totalidade das funcionalidades do circuito integrado, como pode ser comprovado com os testes realizados ao circuito integrado 74HCT164 e apresentados nas Figuras 5.6 e 5.7.

Na Figura 5.8 encontram-se as montagens utilizadas para teste dos circuitos integrados 74HCT174 (*flip-flop* D) e 74LS76 (*flip-flop* JK).

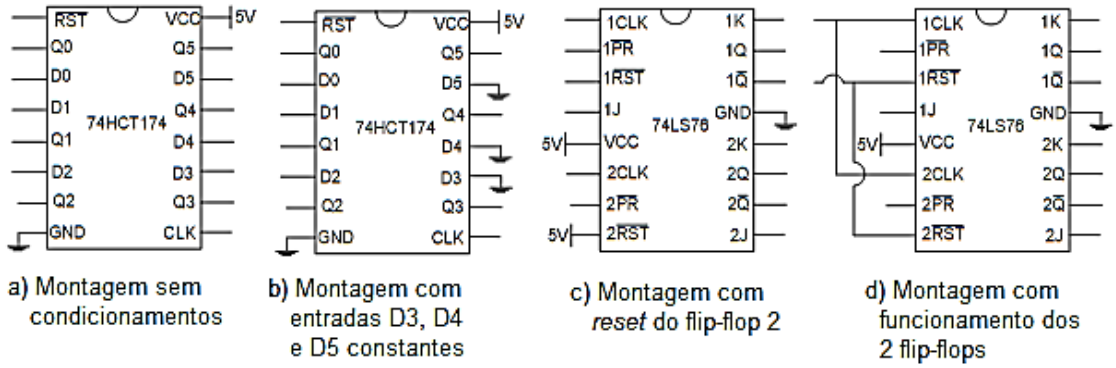

Figura 5.8 – Montagens para estimulação dos circuitos integrados 74HCT174 e 74LS76.

O número de transições possíveis para cada estado depende do número de pinos classificados como pinos de entrada, e é dado por

$$T_E = 2^N \tag{5.6}$$

onde T_E e N representam o número de transições possíveis por estado e o número de pinos de entrada excluindo pinos de relógio e *reset*, respetivamente. O número de transições possíveis depende de T_E e do número de estados identificados, dado por

$$T = T_E \times E \tag{5.7}$$

onde T e E representam o número total de transições e o número de estados detetados.

Analogamente ao ocorrido com os circuitos combinatórios, o aumento de T implica um maior número de medições efetuadas para observação da totalidade do comportamento. Como não foi verificada uma relação entre a taxa de sucesso e o número de entradas do circuito, para efeitos de teste, dos circuitos sequenciais listados na Tabela 5.5 foram recolhidas 2000 amostras por teste (máximo suportado pelo protótipo implementado).

Tabela 5.5 – Taxas de sucesso obtidas em função da montagem e número mínimo de estados/transições necessários para definir comportamento do circuito.

Circuito integrado	Montagem	Nº mínimo de estados	Nº mínimo de transições	Taxa de sucesso obtida (%)
74LS393	Figura 5.2 a)	16	16	100
	Figura 5.2 b)	16	64	25
74HCT164	Figura 5.5 a)	256	1024	56
	Figura 5.5 b)	256	512	67
	Figura 5.5 c)	8	16	100
	Figura 5.5 d)	16	32	100
74HCT174	Figura 5.8 a)	64	4096	16

Circuito integrado	Montagem	Nº mínimo de estados	Nº mínimo de transições	Taxa de sucesso obtida (%)
74HCT174	Figura 5.8 b)	8	64	100
74HC595	Sem condicionamentos	512	8192	13
74LS76[29]	Figura 5.8 c)	2	16	100
	Figura 5.8 d)	4	128	86

Analisando os resultados obtidos para o circuito integrado 74LS393 (contador) verifica-se que o número de estados detetados em cada montagem permanece inalterado, ao passo que o número de transições quadruplica. Este facto ocorre porque o sistema de replicação desenvolvido apenas considera um pino de relógio ou *reset*, ou seja, na montagem da Figura 5.2 b) os pinos de relógio e *reset* do segundo contador são considerados como pinos de entrada. De acordo com a equação (5.6) para dois pinos de entrada existem 4 transições possíveis por estado, justificando assim a quadruplicação do número e transições entre montagens. A taxa de sucesso é reduzida para 25% neste caso, pois os pinos de entrada são constantes nos instantes de interesse e só é observada uma transição em quatro possíveis para cada estado, reduzindo assim a taxa de sucesso, pois devido ao circuito envolvente 75% das transições que o sistema de replicação admite que possam existir (em função do número de estados e entradas) nunca vão ocorrer, logo nunca serão observadas. Apesar de na montagem da Figura 5.2 b) as restantes transições nunca ocorrerem fisicamente, essa informação é desconhecida do sistema de replicação, logo não é refletida na taxa de sucesso. Porém todas as transições observadas são replicadas.

Relacionando a taxa de sucesso obtida com o número mínimo de estados e transições necessários para descrever o circuito, verifica-se que a taxa de sucesso decresce com o aumento do número de transições. O aumento do número de estados não influência diretamente a taxa de sucesso (apenas influencia pela relação de proporcionalidade direta entre aumento de número de estados e transições), pois verifica-se que para o mesmo número de estados a taxa de sucesso é mais baixa no caso onde existem mais transições (montagens da Figura 5.5 a) e b)) e para o mesmo número de transições e igual número de estados (montagens da Figura 5.5 c) e Figura 5.8 c)) foi verificado que a taxa de sucesso permanece inalterada. Assim sendo, pode concluir-se que o aumento do número de transições tem maior impacto na taxa de sucesso que o aumento do número de estados, como pode ser constatado a partir dos valores da Tabela 5.5. O decréscimo da taxa de sucesso com o aumento do número de transições deve-se à não observação de todas as transições. No entanto, todas as transições observadas são replicadas podendo concluir-se que quando não são replicados os estados e transições na totalidade deve-se à não ocorrência dos mesmos durante a observação, inferindo-se assim que os algoritmos desenvolvidos funcionam de acordo com a sua conceção.

[29] Teste realizado a circuito montado por terceiros (teste cego).

5.3.3.4. Aproximação estrutural das réplicas ao circuito original

As réplicas obtidas para circuitos sequenciais não apresentam, em geral, qualquer semelhança estrutural ao circuito integrado original, contrariamente ao verificado em alguns casos nos circuitos combinatórios. Diagramas lógicos obtidos para circuitos sequenciais encontram-se disponíveis no Anexo I (seção I.2). No entanto, a síntese depende da estrutura da descrição de *hardware* e do compilador utilizado, não tendo influência na funcionalidade reproduzida.

5.4. Análise comparativa entre resultados obtidos para circuitos combinatórios e sequenciais

Considerando os resultados obtidos para circuitos combinatórios e sequenciais é possível verificar experimentalmente que existem menos fatores que condicionam a replicação de circuitos combinatórios.

O impacto do meio envolvente na réplica obtida é muito mais significativo nos circuitos sequenciais (comparativamente aos circuitos combinatórios) uma vez que nos circuitos sequenciais para o mesmo estado podem ser observadas diferentes transições para diferentes estímulos do circuito sob teste (como verificado nas Figuras 5.6 e 5.7) enquanto no caso dos circuitos combinatórios, para o mesmo circuito, uma combinação de entrada corresponde a uma única combinação de saída, independentemente da estimulação do circuito. Em síntese, nos circuitos sequenciais o meio envolvente pode originar a observação de diferentes transições para o mesmo estado ao passo que nos circuitos combinatórios o meio envolvente pode permitir observar um número maior/menor de combinações de entrada, sem alteração da relação entre combinação de entrada e saída, devido ao sistema envolvente.

No que respeita a otimização das réplicas obtidas os circuitos sequenciais são moderadamente otimizados pela aproximação heurística desenvolvida, que visa reduzir o número de estados, ao passo que no caso dos circuitos combinatórios não é realizada qualquer otimização pelos algoritmos desenvolvidos. Durante o processo de sintetização, otimização pode ser aplicada pelo compilador.

6. Conclusões

6.1. Conclusões gerais

A engenharia inversa é um procedimento realizado desde há décadas com o objetivo de descobrir como um produto foi construído sendo que os procedimentos mais comuns para este efeito consistem em análises manuais ou semi-manuais do(s) circuito(s) através da separação das diferentes camadas e análise das mesmas com auxilio de sensores óticos e *software* de processamento de imagem. No entanto estas técnicas são direcionadas para uma recuperação do esquema elétrico/lógico do(s) circuito(s) e geralmente danificam o(s) circuito(s) tornando(s)-os inutilizáveis. Mais recentemente técnicas de engenharia inversa não invasivas têm sido desenvolvidas assim como técnicas baseadas em algoritmos com objetivo de identificar o(s) circuito(s) através da análise funcional do(s) mesmo(s). Atualmente alguns sistemas de engenharia inversa apresentam a descrição do esquema lógico ou funcionalidade obtida em linguagens de descrição de *hardware* pois desta forma é fornecido um nível de abstração muito maior, o que torna a réplica obtida independente do *hardware*. Esta dissertação contribuiu para o conhecimento científico no âmbito das técnicas de engenharia inversa não invasivas, baseadas em algoritmos e com conversão do circuito obtido para linguagem de descrição de *hardware* que é uma área cuja investigação científica é recente e onde é possível a existência de desenvolvimentos significativos nas próximas décadas. Atualmente os algoritmos aplicáveis em engenharia inversa são essencialmente algoritmos de reconhecimento de padrões e processamento de imagem (análise estrutural) ou por comparação com uma biblioteca de circuitos conhecidos (análise comportamental).

Os algoritmos desenvolvidos nesta dissertação visaram a obtenção de uma capacidade de análise genérica, sem comparação com qualquer biblioteca de circuitos conhecidos. Algoritmos diferentes foram desenvolvidos para circuitos combinatórios e sequenciais. No caso dos circuitos combinatórios (quando é verificado que para cada combinação de entrada só existe uma combinação de saída) são observadas as combinações e a correspondência entrada-saída é refletida em código VHDL através da utilização dos comandos *case* e *when*. Quando o circuito é sequencial uma aproximação a uma máquina de estados é realizada através da utilização de métodos heurísticos, onde a solução obtida pode não ser a solução ótima mas a uma solução possível em função do comportamento observado. Em qualquer um dos casos, a descrição em linguagem VHDL é automaticamente gerada e o código pode ser sintetizado (utilizando *software* destinado para o efeito) e obtido o esquema lógico da réplica, dado que a estrutura do código gerado é otimizada para a sintetização.

O sistema de aquisição de dados implementado visa retirar medições sem afetar o funcionamento do circuito e ser facilmente controlado pelo computador através de instruções estabelecidas entre o módulo de controlo do sistema de aquisição de dados e algoritmos dedicados à geração de instruções através de porta COM. O protótipo

obtido resultou da ligação entre os algoritmos e o sistema de aquisição de dados desenvolvidos e visa demonstrar experimentalmente o correto funcionamento dos algoritmos desenvolvidos.

A capacidade de replicação obtida para circuitos combinatórios depende exclusivamente do número de amostras recolhidas e das combinações de entrada ocorridas durante o período de observação. O comportamento da réplica corresponde ao comportamento observado e a taxa de sucesso estimada pelo sistema para circuitos combinatórios reflete a percentagem do circuito integrado que foi replicada. Assim sendo, quando a taxa de sucesso estimada é igual a 100%, foi replicada a totalidade do funcionamento do circuito combinatório, permitindo a substituição do circuito integrado original pela réplica independentemente do sistema envolvente do circuito a substituir. Quando a taxa de sucesso estimada é inferior a 100% é apenas aconselhável que o circuito seja substituído no sistema onde foi observado, podendo ainda assim ocorrer situações não observadas, às quais a réplica não irá reagir. No caso dos circuitos sequenciais a capacidade de replicação obtida depende do número de transições possíveis no circuito em análise e do número de transições observadas. O sistema envolvente ao circuito integrado sob teste influencia a réplica obtida, isto é, diferentes aproximações a uma máquina de estados podem ser obtidas para o mesmo circuito integrado quando o sistema envolvente provoca estímulos diferentes no circuito sob teste. Em qualquer dos casos, combinatórios ou sequenciais, apenas as combinações/transições observadas são replicadas, sendo recomendado que as réplicas obtidas sejam substituídas no meio onde foram observados os circuitos integrados originais.

Com base no sistema desenvolvido foi redigido um artigo aceite em conferência sobre computação aplicada [73] (ver Anexo J).

6.2. Trabalhos futuros

Como o projeto desenvolvido foi um protótipo para demonstração do funcionamento dos algoritmos, futuramente é possível trabalhar na otimização do mesmo e/ou construção de um protótipo capaz de operar a frequências mais altas para criação de um produto comercial. Uma partilha do processamento inteligente entre o computador e o sistema de aquisição de dados assim como a aplicação técnicas de compressão de dados podem representar inovações tecnológicas que permitam obter uma capacidade de replicação ainda maior do que a apresentada neste documento.

No caso do protótipo implementado, o utilizador no início do processo de replicação tem de indicar ao sistema quais os pinos de saída. Futuramente, a deteção automática de entrada e saídas através da análise da impedância de cada pino seria uma forma de enriquecer o sistema de replicação.

Futuramente pode ainda ser estudada a expansão da capacidade de replicação do sistema para circuitos analógicos, utilizando o núcleo de processamento já desenvolvido, adicionando várias ADC na entrada e DAC na saída.

Generalizando é possível expandir a capacidade de replicação do sistema desenvolvido é através da utilização de sensores na entrada capazes de converter uma grandeza qualquer num sinal digital e atuadores na saída que realizam a operação inversa, permitindo que o processamento é análise continue a ser digital (já desenvolvido). Segundo esta abordagem, os circuitos analógicos primeiramente abordados podem ser vistos como um caso particular onde os sensores são ADC e os atuadores são DAC.

Referências

[1] J. Wakerly, *Digital Design - Principles & Practices*, 2ª Edição. New Jersey: Prentice-Hall, 1994.

[2] E. Matlin, M. Agrawal, and D. Stoker, “Non-Invasive Recognition of Poorly Resolved Integrated Circuit Elements,” *IEEE Trans. Inf. Forensics Secur.*, vol. 9, no. 3, pp. 354–363, Mar. 2014.

[3] V. Raja, *Reverse engineering : an industrial perspective*, First Edit. London: Springer, 2008, pp. 1–10.

[4] A. Padilla, *Sistemas Digitais*, 2ª Edição. McGraw-Hill Companies, 1993.

[5] P. Scherz, *Pratical Eletronics For Inventors*, 2ª Edição. McGraw-Hill Companies, 2007.

[6] J. Millman and C. Halkias, *Integrated Eletronics*, 2ª Edição. McGraw-Hill Companies, 1972.

[7] E. Eilam, *Reversing: Secrets of Reverse Engineering*. Indianapolis: Wiley Publishing Inc., 2005.

[8] P. A. Lal, J. Hoople, and S. Ardanuç, “Integrated Chipscale Ultrasonic Communication Links,” PCT/US2013/0507722009.

[9] P. Subramanyan, N. Tsiskaridze, W. Li, A. Gascon, W. Y. Tan, A. Tiwari, N. Shankar, S. Seshia, and S. Malik, “Reverse Engineering Digital Circuits Using Structural and Functional Analyses,” *IEEE Trans. Emerg. Top. Comput.*, vol. 2, no. 1, pp. 63–80, 2014.

[10] R. Torrance and D. James, “The state-of-the-art in semiconductor reverse engineering,” in *48th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2011, pp. 333–338.

[11] Z. Wasson, “Reverse Engineering Circuits Using Behavioral Pattern Mining,” in *IEEE Conference on Hardware-Oriented Security and Trust (HOST)*, 2012.

[12] R. Torrance and D. James, “Reverse Engineering in the Semiconductor Industry,” *2007 IEEE Cust. Integr. Circuits Conf.*, no. Cicc, pp. 429–436, 2007.

[13] D. Tang, R. Zhu, and R. Xu, “Functional reverse design: Method and application,” *Comput. Support. Coop. Work Des. (CSCWD), 2010 14th Int. Conf.*, no. 50775111, pp. 723–727, 2010.

[14] E. Riccobene, “A Model-driven Design Environment for Embedded Systems,” in *Design Automation Conference- 43rd ACM/IEEE*, 2006, pp. 915–918.

[15] S. A. Fahmi and H.-J. Choi, “Software Reverse Engineering to Requirements,” *2007 Int. Conf. Converg. Inf. Technol. (ICCIT 2007)*, no. May 1998, pp. 2199–2204, Nov. 2007.

[16] L. Chouambe, N. Ag, B. Klatt, and K. Krogmann, “Reverse engineering Software-Models of Component-Based Systems,” in *In 12th European Conference on Software Maintenance and Reengineering*, 2008, pp. 93–102.

[17] P. Zhang and Y. Su, “Understanding the aspects from various perspectives in aspects-oriented software reverse engineering,” *2010 Int. Conf. Comput. Appl. Syst. Model. (ICCASM 2010)*, no. Iccasm, pp. V11–311–V11–314, Oct. 2010.

[18] P. O. B. A. Jordan, “Reverse Engineering in Mechatronics Education,” *Mohammed Bani Younis Philadelphia University Department of Computer Engineering Tarek A . Tutunji Department of Mechatronics Engineering*, pp. 1–5, 2010.

[19] F. G. Capuano and I. V. Idoeta, *Elementos de Eletrónica Digital*, 30th ed., vol. 1. Érica Editora, 2001.

[20] R. Quijada, A. Raventós, F. Tarrés, R. Durà, and S. Hidalgo, “The Use of Digital Image Processing for IC Reverse Engineering,” in *11th International Multi-Conference on Systems, Signals & Devices (SSD)*, 2014, pp. 1–4.

[21] D. James, “Design-for-manufacturing features in nanometer processes - A reverse engineering perspective,” *2009 IEEE/SEMI Adv. Semicond. Manuf. Conf.*, pp. 56–61, May 2009.

[22] J. Goldstein, C. E. Lyman, D. E. Newbury, and D. C. Joy, *Scanning Electron Microscopy and X-Ray Microanalyis*, 3ª Edição. New York: Springer, 1984.

[23] “Chipworks.” [Online]. Available: http://www.chipworks.com/.

[24] P. Di Lena and L. Margara, “Optimal global alignment of signals by maximization of Pearson correlation,” *Inf. Process. Lett.*, vol. 110, no. 16, pp. 679–686, Jul. 2010.

[25] B. Johnson, “EE368 : Reverse Engineering of Printed Circuit Boards,” *IEEE*, pp. 1–4, 1995.

[26] R. Che, S. Azmi, R. Daud, A. Nasir, and F. K. Ahmad, “Reverse Engineering for Obsolete Single Layer Printed Circuit Board (PCB),” in *Computing & Informatics, ICOCI ’06. International Conference*, 2006, pp. 1–7.

[27] C. Koutsougeras, N. Bourbakis, and V. J. Gallardo, “Reverse Engineering of Real PCB Level Design Using VERILOG HDL,” *Intl. J. Eng. Intell. Syst.*, vol. 10, no. 2, pp. 63–68, 2002.

[28] MathWorks, “MathWoks Documentation Center,” 2014. [Online]. Available: http://www.mathworks.com/help/vision/ref/edgedetection.html. [Accessed: 24-May-2014].

[29] D. Liu and J. Yu, “Otsu Method and K-means,” *2009 Ninth Int. Conf. Hybrid Intell. Syst.*, no. 2, pp. 344–349, 2009.

[30] E. Aybar, “Sobel edge detection method for MatLab,” 2006.

[31] W. Dong and Z. Shisheng, “Color Image Recognition Method Based on the Prewitt Operator,” *2008 Int. Conf. Comput. Sci. Softw. Eng.*, pp. 170–173, 2008.

[32] A. Zamani and M. S. Moslehian, *Approximate Roberts orthogonality*. Springer Basel, 2013.

[33] J. Canny, “A computational approach to edge detection,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 8, no. 6, pp. 679–98, Jun. 1986.

[34] J. Wan, X. He, and P. Shi, “An Iris Image Quality Assessment Method Based on Laplacian of Gaussian Operation,” in *MVA2007 IAPR Conference on Machine Vision Applications*, 2007, vol. 3, pp. 3–6.

[35] S. DiBartolomeo, “D-Codes, Apertures & GERBER Plot Files,” *Artwork Converion Software, Inc.*, 1991. [Online]. Available: http://www.artwork.com/gerber/appl2.htm. [Accessed: 25-May-2014].

[36] T. S. Line, *ABEL-HDL Reference Manual (Version 8.0)*. Lattice Semiconductor Corporation, 2003.

[37] Y. Shi, C. W. Ting, B.-H. Gwee, and Y. Ren, “A highly efficient method for extracting FSMs from flattened gate-level netlist,” *Proc. 2010 IEEE Int. Symp. Circuits Syst.*, pp. 2610–2613, May 2010.

[38] V. Pedroni, *Circuit Design With VHDL*, 1st ed. Massachussets Institute of Technology, 2004.

[39] S. C. A. M. R. Brayton, U. C. Berkeley, and X. W. T. Kam, “Reducing Structural Bias in Technology Mapping,” in *Computer-Aided Design. ICCAD-2005. IEEE/ACM International Conference*, 2005, pp. 518–525.

[40] A. Balcioglu and R. K. Wood, “Enumerating Near-Min S-T Cuts,” in *Naval Postgraduate School -Operations Research Dept.*, California: Springer, 2003, pp. 21–49.

[41] V. Pedroni, *Digital Electronics and Design With VHDL*. Burlington: Elsevier, 2008.

[42] E. Balas and J. Xue, “Finding a maximum clique in an arbitrary graph,” *SIAM J. Comput.*, vol. 15, no. 4, pp. 1054–1068, 1986.

[43] M. R. Garey and D. S. Johnson, *Computer and intractbility*, 1st ed. New York: W. H. Freeman and Company, 1979.

[44] L. Farber, *Estatística Aplicada*, 4ª Edição. São Paulo: Pearson Education Inc, 2010.

[45] Xilinx, “Xilinx - ISE WebPACK Design Software,” 2014. [Online]. Available: http://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.htm. [Accessed: 12-Nov-2013].

[46] L. C. Figueiredo, “Implementação em FPGA de algoritmos computacionais paralelos,” Universidade de Aveiro, 2010.

[47] A. Z. Cordenonsi, “Ambientes, objetos e dialogicidade: uma estratégia de ensino superior em heurísticas e metaheurísticas,” Universidade Federal do Rio Grande do Sul, 2008.

[48] D. Kahneman, *Rápido e Devagar: Duas Formas de Pensar*, 1ª Edição. New York: Editora Objetiva, 2011.

[49] U. F. Rural, J. R. Br, and R. Br, “Heurísticas e Vieses de Decisão: a Racionalidade Limitada no Processo Decisório Marcelo Alvaro da Silva Macedo.”

[50] R. Larson and R. P. Hosteller, “Sequences, Series and Probability,” in *Precalculus*, 7th Editio., Cengage Learning, 2006, pp. 643–714.

[51] F. Santos, “SIGLAB - Lista de componentes,” 2014. [Online]. Available: http://www.cee.uma.pt/inventario/index.php?op=listcomponents. [Accessed: 17-Jul-2014].

[52] “Folha de caraterísticas EC2-5NU,” 1999. [Online]. Available: http://pdf.datasheetcatalog.com/datasheet/nec/EC2-3TNJ.pdf. [Accessed: 17-Jul-2014].

[53] “Folha de caraterísticas 2N2222A.” [Online]. Available: http://pdf.datasheetcatalog.com/datasheet/on_semiconductor2/2N2222A-D.PDF. [Accessed: 17-Jul-2014].

[54] “Folha de caraterísticas 1N4448,” 2004. [Online]. Available: http://www.nxp.com/documents/data_sheet/1N4148_1N4448.pdf. [Accessed: 17-Jul-2014].

[55] “Folha de caraterísticas DG526CJ,” 1999. [Online]. Available: http://pdf.datasheetcatalog.com/datasheets2/31/316139_1.pdf. [Accessed: 17-Jul-2014].

[56] “Folha de caraterísticas AD622,” 2013. [Online]. Available: http://www.analog.com/static/imported-files/data_sheets/AD622.pdf. [Accessed: 17-Jul-2014].

[57] “Folha de caraterísticas 6N136,” 2008. [Online]. Available: http://www.fairchildsemi.com/ds/6N/6N136.pdf. [Accessed: 17-Jul-2014].

[58] “Folha de caraterísticas 74HCT174,” 1998. [Online]. Available: http://pdf.datasheetcatalog.com/datasheet/philips/74HCT174N.pdf. [Accessed: 17-Jul-2014].

[59] “Folha de caraterísticas 74HCT174N,” 1998. [Online]. Available: http://pdf.datasheetcatalog.com/datasheet/philips/74HCT174N.pdf. [Accessed: 17-Jul-2014].

[60] “Folha de caraterísticas Arduino Mega 2560,” 2014. [Online]. Available: http://arduino.cc/en/Main/arduinoBoardMega2560. [Accessed: 17-Jul-2014].

[61] W. Stallings, *Redes e Sistemas de Comunicação de Dados - Teoria e Aplicações Corporativas*, 5ª edição. Rio de Janeiro: Editora Campus, 2005.

[62] “Folha de caraterísticas 74LS08,” 2000. [Online]. Available: http://pdf.datasheetcatalog.com/datasheets/70/375337_DS.pdf. [Accessed: 17-Jul-2014].

[63] “Folha de caraterísticas 74HCT32,” 1990. [Online]. Available: http://pdf.datasheetcatalog.com/datasheet/philips/74HC_HCT32_CNV_2.pdf. [Accessed: 17-Jul-2014].

[64] “Folha de caraterísticas 74LS86,” 1995. [Online]. Available: http://pdf.datasheetcatalog.com/datasheets2/11/115612_1.pdf. [Accessed: 17-Jul-2014].

[65] “Folha de caraterísticas 74HC139,” 1993. [Online]. Available: http://pdf.datasheetcatalog.com/datasheet/philips/74HC_HCT139_CNV_2.pdf. [Accessed: 15-Aug-2014].

[66] “Folha caraterísticas 74LS48,” 1992. [Online]. Available: http://pdf.datasheetcatalog.com/datasheets/120/375650_DS.pdf. [Accessed: 15-Aug-2014].

[67] “Folha caraterísticas CD4001,” 1988. [Online]. Available: http://pdf.datasheetcatalog.com/datasheets/166/108518_DS.pdf. [Accessed: 22-Sep-2014].

[68] “Folha de caraterísticas do Arduíno Uno,” 2014. [Online]. Available: http://arduino.cc/en/Main/arduinoBoardUno. [Accessed: 25-Jul-2014].

[69] “Folha de caraterísticas 74LS393,” 1998. [Online]. Available: http://pdf.datasheetcatalog.com/datasheets/400/334868_DS.pdf. [Accessed: 03-Sep-2014].

[70] “Folha caraterísticas 74HCT164,” 2013. [Online]. Available: http://www.nxp.com/documents/data_sheet/74HC_HCT164.pdf. [Accessed: 03-Sep-2014].

[71] “Folha de caraterísticas 74HC595,” 2011. [Online]. Available: http://www.nxp.com/documents/data_sheet/74HC_HCT595.pdf. [Accessed: 03-Sep-2014].

[72] “Folha de caraterísticas 74LS76,” 1999. [Online]. Available: http://www.futurlec.com/74LS/74LS76.shtml. [Accessed: 20-Sep-2014].

[73] IADIS, “11th International Conference applied Computing,” 2014. [Online]. Available: http://www.computing-conf.org/. [Accessed: 20-Aug-2014].

[74] S. Guilley, J. Danger, R. Nguyen, and P. Nguyen, “System-Level Methods to Prevent Reverse-Engineering , Cloning , and Trojan Insertion,” 2012.

[75] *Decreto-lei n.º 36/2003, de 05 de Março do Ministério da Economia*. 2003.

[76] *Decreto-lei n.º143/2008 de 25 de julho do Ministério da Justiça*. Portugal, 2008.

[77] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, 1st ed. McGraw-Hill Companies, 1994.

[78] D. J. Smith, *HDL-CHIP-design.pdf*, 1st ed. Madinson: Doone Publications, 1996.

[79] C. J. Roth, *Digital Systems Design Using VHDL.pdf*. Boston: PWS Publishing Company, 1998.

[80] “Folha de caraterísticas 74HCT04,” 2012. [Online]. Available: http://www.nxp.com/documents/data_sheet/74HC_HCT04.pdf. [Accessed: 17-Jul-2014].

[81] “Folha caraterísticas 74LS04,” 1989. [Online]. Available: http://pdf.datasheetcatalog.com/datasheets/134/231533_DS.pdf. [Accessed: 17-Jul-2013].

ANEXOS

A. Técnicas de defesa contra a engenharia inversa

A engenharia inversa apesar das vantagens que apresenta, pode ser considerada uma ameaça, nomeadamente em circuitos eletrónicos de alto valor [74]. As técnicas de defesa contra a engenharia inversa foram inicialmente desenvolvidas para proteção de *software* e posteriormente estendidas aos circuitos eletrónicos [74]. As soluções mais utilizadas para proteção de circuitos contra a aplicação de engenharia inversa encontram-se descritas na Tabela A.1, bem como o impacto no sistema desenvolvido:

Tabela A.1 – Técnicas de defesa contra engenharia inversa [74], [8].

Técnica	Descrição	Impacto
Funções fisicamente inclonáveis	Consistem em dispositivos que produzem uma chave interna, essencial para funcionamento, única e não conhecida por elementos exteriores ao dispositivo. É possível a extração de informação encriptada.	Como o mecanismo de defesa é intrínseco ao dispositivo e o sistema desenvolvido efetua a análise de entradas e saída, logo será capaz de replicar o funcionamento interno e obter uma réplica com o mesmo comportamento do ponto de vista das entradas e saídas.
Ofuscação	Baseia-se na implementação de um esquema simples de uma forma complexa, mais difícil de descodificar, utilizando técnicas tais como o roteamento em “esparguete” ou a cobertura com uma camada de metal, para evitar a observação do circuito.	Estas técnicas fornecem proteção ao nível da obtenção da estrutura do circuito por análise visual (manual ou computorizada) porém como a análise comportamental através da observação das entradas e saídas não é afetada por estas técnicas de defesa.
Dissimulação ótica	Consiste na adição de componentes que dificultem a recuperação do esquema do circuito, mas não produzem efeito no circuito, como por exemplo fios fictícios, componentes sem ligação, entre outros.	À semelhança do que ocorre na técnica de ofuscação, a dissimulação ótica protege os circuitos contra engenharia inversa que utiliza métodos visuais para extração do esquema elétrico, não tendo impacto no método desenvolvido para extração do comportamento.
Migração para circuitos óticos e acústicos	Esta técnica consiste na substituição dos sinais elétricos por sinais óticos ou acústicos no interior dos circuitos integrados.	O sistema implementado foi concebido para analisar sinais elétricos. Se os sinais de entrada e saída forem elétricos (convertidos para óticos no interior do circuito) o sistema desenvolvido é capaz de extrair o comportamento, caso contrário devido à impossibilidade de adquirir medições elétricas o sistema não conseguirá extrair o comportamento o circuito.

B.Lei da propriedade industrial

A aplicação da engenharia inversa, e em particular do sistema desenvolvido, deve sempre ter em conta a legislação vigente, nomeadamente a lei da propriedade industrial. No caso de Portugal as leis aplicáveis neste âmbito encontram-se no Código de Propriedade Industrial (adiante CPI) [75] e em particular no Decreto-Lei n.º143/2008 [76], onde foram aprovadas medidas de simplificação e acesso à propriedade industrial, alterando o CPI original.

A propriedade industrial desempenha a função de garantir a lealdade da concorrência, pela atribuição de direitos privativos sobre os diversos processos técnicos de produção e desenvolvimento da riqueza (Artigo 1º do CPI) [75]. Assim sendo a aplicação do sistema de replicação de circuitos integrados desenvolvido a circuito patenteados implica o cumprimento de normas legais. A aplicação do sistema desenvolvido pode ser aplicada quando em conformidade com o abaixo disposto [75], [76]:

- Realização em âmbito privado e sem fins comerciais (Artigo 102º do CPI);
- Realização a título experimental (Artigo 145º do CPI);
- Realização em situações de interesse público[30] (Artigo 110º do CPI).

Quando não aplicáveis as situações acima descritas, e de acordo com o Artigo 322º, constituem violações do Código de Propriedade Industrial [75], [76]:

- Reproduzir ou imitar, totalmente ou em alguma das suas partes caraterísticas do modelo registado;
- Explorar um modelo registado, mas pertencente a outrem.

A aplicação do sistema a circuitos não patenteados ou cuja patente tenha caducado (Artigo 37º do CPI) não constitui qualquer violação legal e pode ser aplicada sem qualquer restrição.

[30] A concessão da licença por motivo de interesse público é da competência do Governo [75], [76].

C. Síntese de circuitos

C.1. Introdução à síntese de *hardware*

Atualmente uma grande parte dos sistemas digitais são projetados utilizando uma linguagem de descrição de *hardware*, sendo o VHDL e o *Verilog* as linguagens mais usadas [41].

A síntese consiste na geração (e otimização) de um modelo de circuito a partir de um circuito mais abstrato, isto é, o produto de uma sintetização possui sempre mais detalhe que o circuito ou diagrama original [41] [77].

Este tipo de linguagens permitem que o circuito seja sintetizado e totalmente simulado por *software* antes de qualquer implementação física, permitindo um desenvolvimento e teste maior sem incremento dos custos em material, uma vez que após escrito o código este pode ser implementado fisicamente numa FPGA ou CPLD. Após todo o desenvolvimento concluído e testado, pode ser criado um circuito integrado (também designado por ASIC – *Application-Specific Integrated Circuit*) com a funcionalidade sintetizada [41].

Acrescem ainda as vantagens destas linguagens permitirem que a tecnologia seja independente do fabricante, pois código previamente concebido pode ser facilmente incorporado em novos projetos ou reutilizáveis com tecnologias diferentes [41].

C.2. FPGAs e ASICs *Application-Specfic Integrated Circuit*

Em geral as funcionalidades dos circuitos integrados são fixas e definidas no processo de fabrico. No entanto, as FPGAs e ASICs (*Application-Specfic Integrated Circuit*) são circuitos integrados onde a funcionalidade não é definida na fábrica, mas sim pelo utilizador [78]. No caso das FPGAs a arquitetura interna difere entre fabricantes, mas ainda assim a generalidade das FPGAs são constituídas por milhões de blocos de lógica programável e conexões programáveis, que tornam estes dispositivos versáteis como ilustrado na Figura C.1 [41].

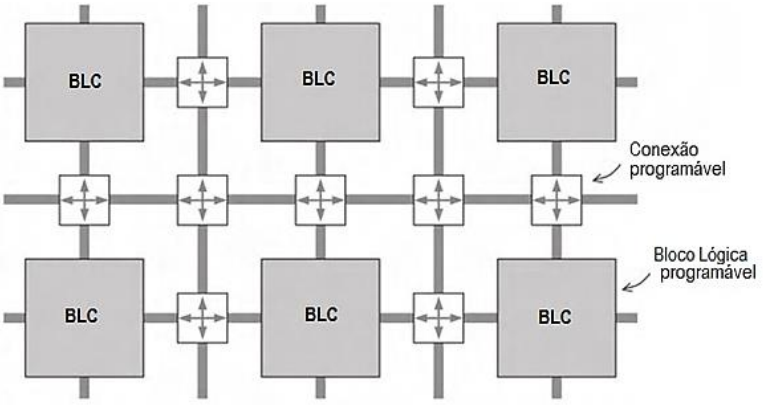

Figura C.1 – Diagrama interno parcial de uma FPGA [41].

A estrutura interna dos blocos de lógica programável é complexa e a evolução das FPGAs reside em grande parte no melhoramento destes blocos, tornando-os maiores e mais complexos, aumentando assim a capacidade de síntese de circuitos das FPGAs [41].

Os ASICs são circuitos integrados concebidos para executar uma função específica. A construção de um ASIC é realizada em duas etapas: fabrico genérico e especificação da funcionalidade [78].

A primeira etapa, fabrico genérico, é realizada na fábrica e é a etapa mais complexa, demorada e dispendiosa consistindo na construção de um *chip* com múltiplos transístores não conectados. A segunda etapa consiste na interligação dos transístores de forma a realizar a função pretendida [78].

Existem dois tipos de ASICs: conjuntos de células (*gate array*) e células *standard,* onde uma célula básica consiste num conjunto de transístores [78].

Os ASICs constituídos por conjuntos de células possuem colunas de células distribuídas ao longo do *chip*, existindo duas variantes: com células dedicadas para interconexões entre células (apenas 70% a 90% disponível para utilização personalizada) ou sem células dedicadas para interconexões [78].

No caso dos ASICs de células *standard* o conceito de célula básica não existe, nem existem componentes pré fabricados. Neste tipo de ASICs o fabricante cria máscaras personalizadas para cada etapa de fabrico, utilizando assim o sílicio mais eficientemente [78].

Os ASICs possuem também algumas bibliotecas que agrupam células de forma a formar componentes mais complexos, tais com *multiplexers*, controladores, ALU, entre outros [78].

C.3. Linguagens e metodologias de descrição de *hardware*

As linguagens de descrição de *hardware* (HDL – *Hardware Description Languages*) são linguagens que permitem criar um modelo com uma função específica ou uma peça de *hardware*, através de *software*. Essencialmente estas linguagens permitem dois tipos principais de descrição: comportamental e estrutural [78].

A descrição estrutural efetua uma modelação do *hardware* a utilizar na síntese, tendo um menor controlo no comportamento do circuito, comparativamente a uma descrição comportamental [78]. Uma descrição estrutural providencia a interconexão entre componentes, podendo ser considerada equivalente a um esquema lógico [77].

A descrição comportamental é uma modelação mais abstrata (pouco controlo ao nível de *hardware*) e facilita a especificação de objetivos do circuito a sintetizar [78]. Descrições comportamentais são fundamentais na modelação de circuitos de grau de dificuldade acrescido ou com elevado número de entradas e saídas [77]. A mesma função lógica descrita a partir de descrição comportamental e estrutural pode resultar em circuitos diferentes após a sintetização, uma vez que diferentes circuitos podem ser obtidos para diferentes níveis de otimização, tendo o sintetizador maior liberdade de otimização da síntese nas descrições comportamentais [77].

O nível de abstração ou detalhe de um sistema encontra-se exemplificado na Figura C.2.

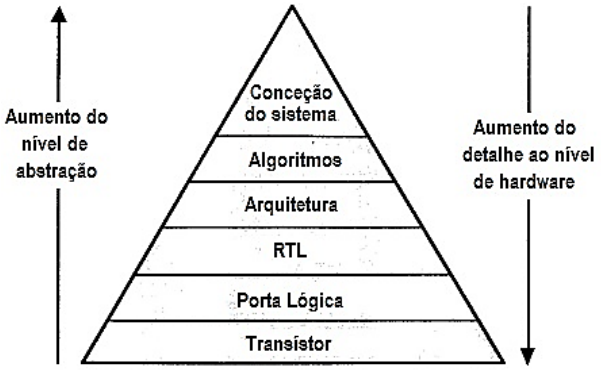

Figura C.2 – Nível de abstração ou detalhe de um sistema [78]

As principais linguagens de descrição de *hardware* são o Verilog e o VHDL. Ambas são capazes de descrever circuitos lógicos sintetizáveis, possuindo apenas pequenas diferenças. As principais diferenças encontram-se na Tabela C.1 [78].

Tabela C.1 – Principais diferenças entre as linguagens Verilog e VHDL [78].

Verilog	VHDL
- Linguagem mais simples, logo mais fácil de aprender; - Maior facilidade de descrição em baixo nível; - Menor precisão necessária na descrição do modelo.	- Mais tipos de dados e funções de conversão disponíveis; - Maior facilidade de descrição em alto nível; - Melhor desempenho em circuitos de maiores dimensões.

Atualmente existem várias ferramentas de *software* que permitem a geração automática de um circuito digital a partir de código VDHL. No entanto, mesmo que o código VHDL compile sem erros e seja simulado corretamente, não é garantido que esse mesmo código seja sintetizado corretamente, ou o resultado dessa implementação pode ser ineficiente [79].

Como as ferramentas de sintetização não são capazes de avaliar a intenção do utilizador, por vezes é necessário promover algumas alterações no código VHDL para que a síntese seja corretamente e/ou eficientemente implementada, principalmente nos circuitos onde é exigida sincronização [79].

A sintaxe, tipos de dados e outras informações úteis para escrita de código VHDL estão descritas em [38], [41], [78], [79].

C.3.1. Recomendações para modelação de circuitos sintetizáveis em VHDL

Para evitar que o circuito resultante de uma sintetização seja ineficiente, existem recomendações para melhorar vários parâmetros do circuito sintetizado essencialmente a precisão e velocidade de funcionamento [78]. A Tabela C.2 apresenta algumas recomendações gerais para modelação de circuitos sintetizáveis.

Tabela C.2 – Recomendações gerais para modelação de circuitos sintetizáveis [78].

	Recomendações
Precisão	- Assegurar que todos os sinais de interesse se encontram na lista de sensibilidade do $process;$ - Não utilizar tipos de dados sem comprimento definido; - A utilização de parenteses em expressões permite algum controlo estrutural.
Velocidade	- Utilizar sempre que possível declarações concorrenciais (reduz o número de sinais que o simulador tem de monitorizar em simultâneo); - Minimizar o número de sinais na lista de sensibilidade do $process;$ - Não modelar demasiados pequenos $process$ (existe um pequeno atraso para os ativar/desativar).

Outras recomendações mais específicas serão descritas nas subsecções seguintes.

C.3.2. Síntese de circuitos sem memória (combinatórios)

Assumindo que a declaração $entity$ (declaração de entradas e saídas) é necessária para um código VHDL compilável a análise para efeitos de sintetização vai recair sobre o conteúdo do $process$. Um $process$ para circuitos digitais combinatórios apresenta a estrutura da Figura C.3.

```
process ( lista sensibilidade )
    -- declaração de variáveis
begin
    -- declaração sequencial 1
    -- declaração sequencial 2
    -- ...
    -- declaração sequencial N
end process;
```

Figura C.3 – Estrutura de um $process$ de um circuito combinatório [78].

Nos circuitos puramente combinatórios é essencial que todas as combinações de entradas estejam abrangidas no código, caso contrário a sintetização será ineficiente [78].

No caso dos circuitos sem memória (combinatórios) a estruturação do código não é tão influente na sintetização comparativamente a circuitos com memória (sequenciais), residindo a principal diferença na utilização das declarações if e $case$ [38], [78].

Na síntese a partir de código VHDL a declaração $case$ é entendida pelo sintetizador como uma codificação sem prioridades ao passo que a declaração $if/else$ representa uma implementação com prioridade, podendo originar a síntese de descodificadores prioritários ou *multiplexers* [38], [78]. A utilização do comando $else$ é importante para reduzir a síntese de *multiplexers* e se todas as combinações de entradas estiverem abrangidas o resultado obtido a partir de declarações $if/else$ será similar ao obtido com a declaração $case$, após o processo de otimização durante a síntese [38]. No entanto, a utilização da declaração $case$ garante na maioria dos casos uma síntese mais eficiente [78].

C.3.3. Síntese de máquinas de estados finitos em VHDL

A descrição em linguagem de *hardware* de uma máquina de estados finitos assenta em dois blocos lógicos: um bloco sequencial de atribuição do próximo estado e um bloco combinatório de atualização das saídas [38], [41], [78]. Um exemplo genérico do código de cada bloco encontra-se apresentado na Figura C.4.

```
process (reset, clock)
begin
    if("reset ativo")
        --Atribui estado reset ao proximo estado
    else if ("transição de clock de interesse")
        --Atribui proximo estado em função do estado atual
    end if;
end process;
```

a) Bloco sequencial

```
process(proximo_estado)
begin
    --Atualiza saidas
    --Indica próximo estado
        --(com base no estado atual e/ou entradas)
end process;
```

b) Bloco combinatório

Figura C.4 – Estrutura genérica dos blocos de uma máquina de estados [38].

O bloco sequencial efetua a atribuição do próximo estado bem como verificação e atuação perante uma situação de *reset*. A deteção de *reset* apresentada na Figura C.4 é assíncrona (não depende do sinal de clock). A utilização de um *reset* assíncrono garante que a máquina é sempre inicializada num estado conhecido antes da primeira transição ativa do sinal de *clock* [78]. A utilização de *reset* síncrono também é possível mas aumenta a possibilidade da máquina de estados ficar permanentemente presa num estado não definido (quando o circuito é ligado não são conhecidos os estados dos *flip-flops*) ou implicar uma utilização de mais lógica combinatória para prevenir esta situação [78].

O bloco combinatório efetua a atualização das saídas e indica qual o próximo estado (baseado no estado atual e entradas do circuito) e não está sujeito aos problemas de sincronização referidos anteriormente e, sendo um bloco combinatório, o código deste módulo segue as mesmas regras descritas na subseção de circuitos combinatórios: a utilização da declaração $case$ é preferível à declaração if [78],[38]. A Figura C.5 apresenta duas formas diferentes de estruturar o código VHDL do bloco combinatório.

```
process (estado_anterior)
begin
    case (estado_anterior) is
        when estado_0 =>
            --Atualiza saidas
            próximo_estado <= "atribui estado"
        when estado_1 =>
            --Atualiza saidas
            próximo_estado <= "atribui estado"
        (...)
        when others =>
            --Atualiza saidas
            próximo_estado <= "atribui estado"
    end case;
end process;
```

a) Sem entradas

```
process (estado_anterior)
begin
    case (estado_anterior) is
        when estado_0 =>
            if("entradas"="valor") then
                --Atualiza saidas
                próximo_estado <= "atribui estado"
            else
                --Atualiza saidas
                próximo_estado <= "atribui estado"
            end if;
        (...)
        when others =>
            --Atualiza saidas
            próximo_estado <= "atribui estado"
    end case;
end process;
```

b) Com entradas

Figura C.5 – Exemplo de estruturação do bloco combinatório [38].

Na Figura C.3 b) verifica-se que a avaliação das entradas foi realizada através de uma declaração if , contudo a declaração $case$ também pode ser utilizada,

principalmente quando existem mais do que duas combinações de entrada possíveis [38].

Existem variados estilos alternativos para implementação de máquinas de estados finitas que podem ser consultados em [38] e [78].

D. Algoritmos implementados

D.1. Inicializações

A leitura dos parâmetros de configuração do circuito são fundamentais aos diversos módulos. As funções que implementam a leitura das configurações iniciais e reconfigurações encontram-se apresentadas nas Listagens D.1 e D.2, respetivamente. A implementação foi efetuada na linguagem C.

Listagem D.1 – Função para leitura das configurações iniciais.

```
void initialize(){
  int k, nr_measurementsVCCGND=0;
  int i, j = 0;
  FILE *fpInit;
  char fileLineContent[300];
  fpInit = fopen("settings.REP","r");                    //_open configuration file_
  if (fpInit==NULL )    printf("Error opening settings.REP\n");
  fgets(fileLineContent,300,fpInit);                     //_Read setting from configuration file_
  strcpy(chipDirectory,fileLineContent);                 //_save diretory_
  fgets(fileLineContent,300,fpInit);                     //_Read setting from configuration file_
  strcpy(destinantionDirectory,fileLineContent);         //_save replica location_
  fgets(fileLineContent,300,fpInit);                     //_Read setting from configuration file_
  PINS_NR=atoi(fileLineContent);                         //_save number of pins_
  fgets(fileLineContent,300,fpInit);                     //_Read setting from configuration file_
  j=0;
  while (fileLineContent[j]!= '\0'){                     //_Replace comma by decimal point_
     if (fileLineContent[j]==',')       fileLineContent[j]='.';
     j++;
  }
  MAX_TOLERANCE=atof(fileLineContent);                   //_save maximum tolerance_
  fgets(fileLineContent,300,fpInit);                     //_Read setting from configuration file_
  NON_USED_MEASUREMENTS_NR=atoi(fileLineContent);        //_non-interest data_
  fgets(fileLineContent,300,fpInit);                     //_Read setting from configuration file_
  MEASUREMENTS_NR=atoi(fileLineContent);                 //_save number of measurements
  fclose(fpInit);                                        //_Close file_
  j=0;
  while (chipDirectory[j]!= '\0') j++;                   //_remove last character: '\n'_
  chipDirectory[j-1]='\0';
  for (j=0;j<MAX_PIN_NR;j++){                            //_initilize structures_
     pinsType[j]=0;
     pinsV[j]=0;
  }
}
```

Listagem D.2 – Função para leitura das reconfigurações.

```
void sequentialSettingsInitialization(){          //*This function load new settings (sequential settings)
      char fileLineContent[50];
      FILE *settings=fopen("settingsSeq.rep","r");          //_Open settings file_
       if(settings==NULL){
            printf("File Error");                           //_File opening failure_
      }else{
            rewind(settings);                               //_rewind cursor in file_
            fgets(fileLineContent,50,settings);             //_Read setting from configuration file_
            MEASUREMENTS_NR=atoi(fileLineContent);          //_save number of sequential measurements_
```

```
            fgets(fileLineContent,50,settings);                          //_Read setting from configuration file_
            ACCURACY=atoi(fileLineContent);                              //_save accuracy_
            fgets(fileLineContent,50,settings);                          //_Read setting from configuration file_
            RESET_CONVERT_TYPE=atoi(fileLineContent);                    //_Select reset conversion type_
            fgets(fileLineContent,50,settings);                          //_Read setting from configuration file_
            ANALYSIS_DEPTH=atoi(fileLineContent);                        //_save sequential anaysis depth_
    }
   fclose(settings);                                                     //_Close file_
}
```

D.2. Adaptação a circuitos com número de pinos inferior ao máximo suportado pelo sistema de aquisição de dados

O sistema de aquisição de dados permite a realização de medições a circuitos integrados com um máximo de 16 pinos. Quando o número de pinos do circuito integrado é igual a 16, fisicamente só existe uma forma possível para realização das medições, porém quando o número de pinos é inferior a 16 o circuito sob teste deverá sempre ser colocado junto ao topo, como ilustrado na Figura D.1.

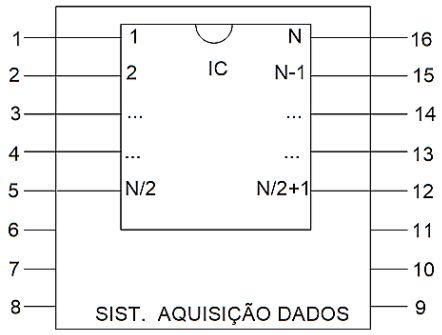

Figura D.1 – Adaptação de circuitos integrados com menos de 16 pinos.

Pela observação da Figura D.1 pode verificar-se que a numeração dos pinos do circuito integrado apenas coincide com o número dos pinos do sistema de aquisição de dados quando o número de pinos do circuito integrado é igual a 16. Os algoritmos de replicação utilizam no seu funcionamento a numeração de pinos do circuito integrado (variável) e o sistema de aquisição de dados utiliza a numeração própria (fixa), logo para compatibilizar ambos os sistemas é necessário realizar a conversão, de acordo com a expressão

$$P_{SAD}(i) = \begin{cases} i, & 1 \leq i \leq \dfrac{N}{2} \\ \left(16 - \dfrac{N}{2}\right) + \left(i - \dfrac{N}{2}\right), & \dfrac{N}{2} < i \leq 16 \end{cases} \quad \text{(D.1)}$$

onde $P_{SAD}(i)$ representa o pino do sistema de aquisição de dados correspondente ao pino i do circuito integrado.

Esta conversão é realizada sempre que existe necessidade de realizar medições em circuitos com menos de 16 pinos, de forma a compatibilizar o sistema de aquisição de dados com os algoritmos. Durante o processo de replicação, alguns os ficheiros são renomeados de forma a corresponder à numeração do circuito integrado utilizando a fórmula de conversão apresentada na equação (D.1).

D.3. Geradores de instruções para medições analógicas

As principais funções dos módulos geradores de instruções para medições analógicas (diferenciais e referenciadas) descritos nas seções 3.4.3 e 3.5.3 deste documento respetivamente, encontram-se apresentados nas Listagens D.3 e D.4. A implementação foi efetuada na linguagem C#.

Listagem D.3 – Função para gerar instruções de aquisição de medições analógicas.

```
    private void selectMeasurements() {
        int i, j,k;
        Decimal [] temporaryData=new Decimal [measurementsNR];
        string pathComplete;
        if ((pinsNR % 2) == 1) pinsNR++;          //_Re-configure pins to select pins of physical socket to read_
        int[] pins = new int[pinsNR];
        int halfPins = pinsNR / 2;
        for (i = 0; i < pinsNR; i++) {                //_(the chip must be always placed on top of the socket)_
            if (i < (halfPins)) {             //_Make the conversion of pins number to postion of pin in the socket_
                pins[i] = i;
            }
            else {
                pins[i] = (16 - halfPins) + (i - halfPins);
            }
        }
        for (i = 0; i < pinsNR; i++)  {                                  //_Generate measurements instructions_
            for (j = i + 1; j < pinsNR; j++) {
    //_File name_
    pathComplete = path+ "\\measurementsP"+(i+1).ToString() + "P" + (j+1).ToString() + ".MED";
    writeCOMfile(pins[i], pins[j], pathComplete);                        //_Write communication settings_
                Process runningProcess = new Process();                        //_Create process to communicate_
                runningProcess.StartInfo.CreateNoWindow = true;
                runningProcess = System.Diagnostics.Process.Start("moduleCOM.exe");
    //_Communicate with data acquisition system (DAS)_
                int idprocess = runningProcess.Id;
                while (runningProcess.HasExited == false) {                    //_wait conclusion of communication_
                }
                StreamReader file2 = new StreamReader(pathComplete);      //_Open file with stored measurements _
                for (k = 0; k < measurementsNR; k++)  {
                    temporaryData[k] = Convert.ToDecimal(file2.ReadLine()); //_Read measurements provided by DAS_
                }
                file2.Close();                                                  //_Close file_
                for (k = 0; k < measurementsNR; k++)  {
                    temporaryData[k] = ((temporaryData[k] * 5 / 1024) - 2.5m) * 6;          //_Apply conversion factor_
                }
                StreamWriter  file3 = new StreamWriter(pathComplete);
                for (k = 0; k < measurementsNR; k++) {
                    file3.WriteLine(temporaryData[k].ToString());              //_Write converted data into file_
                }
                file3.Close();
            }
        }
    }
```

Listagem D.4 – Função de escrita no ficheiro utilizado para transferir dados para a comunicação com o sistema de aquisição de dados.

```
public void writeCOMfile(int i, int j, string storagePath) {
    StreamWriter fileCOM = new StreamWriter("comdata.rep");          //_Open communication settings file_
    fileCOM.WriteLine(portName);                                     //_Write COM port name_
    fileCOM.WriteLine(storagePath);                                  //_Write storage path_
    fileCOM.WriteLine("0001");                  //_Write instruction opcode: diferential measurements
    fileCOM.WriteLine(measurementsNR);          //_Write the number of measurements requested_
    fileCOM.WriteLine(i.ToString());            //_Write positive pin to measure_
    fileCOM.WriteLine(j.ToString());            //_Write negative pin to measure_[31]
    fileCOM.Close();                            //_Close file_
}
```

[31] Quando a referência para as medições analógicas já é conhecida, o pino GND deve ser atribuído aqui.

D.4. Algoritmo de análise dos pinos destinados a alimentação

As principais funções do algoritmo de análise dos pinos destinados a alimentação descritos na seção 3.4.4 deste documento, encontram-se apresentadas nas Listagens D.5, D.6 e D.7. A implementação foi efetuada na linguagem C.

Listagem D.5 – Funções de deteção dos pinos destinados a alimentação.

```
void detectionVCC_GND(){
   char strX[20], strY[20];
   for (x=1;x<=PINS_NR;x++)                    //_Build file name_
       sprintf(strX, "%i", x);
      for (y=x+1;y<=PINS_NR;y++){
        sprintf(strY, "%i", y);
        strcpy(fileName,chipDirectory);
        strcat(fileName,"\\MeasurementsVCCGND\\measurementsP");
        strcat(fileName,strX);
        strcat(fileName,"P");
        strcat(fileName,strY);
        strcat(fileName,".MED");
        verifyPins();                                          //_Verify VCC and GND situation_
        }
   }
}

void verifyPins(){                                  //*Verify if selected file contains constant values
   float measurements[MEASUREMENTS_NR];
   float sum=0, MeasurementsAdjust=MEASUREMENTS_NR;
   char* fileOpen;
   char lineContent[50];
   int i,violationDeteted=0,counter=0;
   fileOpen=fileName;
   fp0=fopen(fileName,"r");
    if (fp0 == NULL) {
      printf ("File opening error.\n");
            exitValue=10;
    }
   while (fgets(lineContent,50,fp0)){
     if (counter<MEASUREMENTS_NR){                  //_read and store file measurements_
          measurements[counter]=atof(lineContent);
          sum+=measurements[counter];
          counter++;
     }
   }
   MeasurementsAdjust=counter;
   fclose(fp0);
   mean=sum/MeasurementsAdjust;                     //_Calculates average_
   MeasurementsAdjust=MEASUREMENTS_NR;
   counter--;
   while (counter>=0 && violationDeteted==FALSE){   //_verify if samples are in confidence interval_
     if ( measurements[counter]>=(mean+TOLERANCE) || measurements[counter]<=(mean-TOLERANCE)){
       violationDeteted=1; break;
     }
     counter--;
   }
    if(mean>=0){
           meanCorrected=mean;
    }else{                                                 //_Obtain absolute value_
            meanCorrected=mean*(-1);
```

```
    }
   if (violationDeteted==0 && meanCorrected>2*TOLERANCE){        //_Success case (constant value)_
      if (pinsType[x-1]!=0 || pinsType[y-1]!=0){                //_Verify previous assignements_
        specialCaseVCCGND();                                    //_Analysis previous assignements_
      }else{
         if (mean>0){                                           //_ VCC and GND atribution: case 1_
            pinsType[x-1]=VCC_PIN;
            pinsType[y-1]=GND_PIN;
            pinsV[x-1]=meanCorrected;
         }else{                                                 //_ VCC and GND atribution: case 2_
            pinsType[x-1]=GND_PIN;
            pinsType[y-1]=VCC_PIN;
            pinsV[y-1]=meanCorrected;
         }
      }
   }
}
```

Listagem D.6 – Função de tratamento de casos especiais.

```
void specialCaseVCCGND(){          //*analyses when previous GND or VCC was assigned
                                   //*re-adjustment of VCC and GND
    if  (pinsType[x-1]==GND_PIN && pinsType[y-1]==NORMAL_PIN){
            pinsType[y-1]=VCC_PIN;
            pinsV[y-1]=mean*(-1);
    }
   if  (pinsType[x-1]==NORMAL_PIN && pinsType[y-1]==GND_PIN){
            pinsType[x-1]=VCC_PIN;
            pinsV[x-1]=mean;
   }
   if  (pinsType[x-1]==VCC_PIN && pinsType[y-1]==NORMAL_PIN){
            pinsType[y-1]=VCC_PIN;
            pinsV[y-1]=pinsV[x-1]- mean;
   }
   if  (pinsType[x-1]==NORMAL_PIN && pinsType[y-1]==VCC_PIN){
            pinsType[x-1]=VCC_PIN;
            pinsV[x-1]=mean-pinsV[y-1];
   }
}
```

Listagem D.7 – Funções para normalização dos resultados.

```
void addOffset(){                          //*get the minimum VCC deteted. That value is added to all VCC
   float offset=getMinimum();              //_find minimum_
   int j;
   for (j=0;j<PINS_NR;j++){
     if (pinsV[j]!=0 || pinsType[j]==2){
       pinsV[j]+=offset;
       if (pinsV[j]<1.7)                   //_Level correction: re-adjust false VCC_
         pinsType[j]= GND_PIN;
       else
          pinsType[j]= VCC_PIN;
     }
   }
}
```

```
float getMinimum(){                          //*This function find the minimum value of VCC values
    float minimum=pinsV[0];
    int i;
    for (i=1;i<PINS_NR;i++){
      if (pinsV[i]<minimum)                  //_Compare with previous lower value_
        minimum=pinsV[i];                    //_Store if presente value is the lowest_
    }
    if (minimum<0)                           //_Get absolute value_
      minimum=minimum*(-1);
  return minimum;
}
```

D.5. Algoritmo de despiste de sinais analógicos

O algoritmo de despiste de sinais analógicos foi descrito na seção 3.5.4 deste documento. A implementação foi realizada em linguagem C e a alocação de memória para armazenamento dos dados encontra-se apresentada na Listagem D.8.

Listagem D.8 – Excerto de código fonte para alocação de memória.

```
(…)
//_allocate memory to data structures_
//» measurements -> store  original measurements. Matrix form: [Nr Pins] [Nr Measurements]«
//» temporaryMeasurements -> store semi-processed measurements. Matrix form: [Nr Pins] [Nr Measurements]«
//» levelMeanValues -> store voltage levels detected for each pin. Matrix form: [Nr Pins] [Nr Measurements]
//»(probably de 2nd dimension of array will not be ully used)«
measurements = malloc(sizeof *measurements *PINS_NR);
temporaryMeasurements=malloc(sizeof *temporaryMeasurements *PINS_NR);
 if (measurements && temporaryMeasurements){
     for (i = 0; i < PINS_NR; i++){
       measurements[i] = malloc(sizeof *measurements[i] * MEASUREMENTS_NR);
       temporaryMeasurements[i]=malloc(sizeof *temporaryMeasurements[i] *MEASUREMENTS_NR);
       for(j=0;j<MEASUREMENTS_NR;j++)
            temporaryMeasurements[i][j]=0;
     }
  }
  levelsMeanValues = malloc(sizeof *levelsMeanValues *PINS_NR);
  if (levelsMeanValues) {
     for (i = 0; i < PINS_NR; i++) {
       levelsMeanValues[i] = malloc(sizeof *levelsMeanValues[i] * MEASUREMENTS_NR);
       for (j=0;j<MEASUREMENTS_NR;j++)
          levelsMeanValues[i][j]=-99;
     }
   }
(…)
```

As funções que implementam a verificação do número de níveis de tensão de cada pino encontram-se apresentadas nas Listagens D.9 e D.10, nomeadamente as funções coordenadora e de análise de níveis, respetivamente.

Listagem D.9 – Função coordenadora da verificação de níveis.

```
void numberOfLevelsVerification(){     //*verify if all input and output pins are digital (1 bit), analog or mixed.
   int pinOut,probableLevelsNr, basicDigital=0, analogConfirm=1;
   for (pinOut=0;pinOut<PINS_NR;pinOut++){
     if(pinsType[pinOut]!=0 && pinsType[pinOut]!=3){
            searchLevels(pinOut);
            probableLevelsNr=levelsCounter-ghostLevels;
            if (probableLevelsNr<=2){                                   //_Set as digital or analog_
                  basicDigital=1;
            } else{
                  analogConfirm=0;
            }
     }
   }
   int circuitType=basicDigital+analogConfirm;              //_digital =2, mixed=1; analog =0_
  switch(circuitType){                                  //_select replication type or error code: digital, analog or mixed_
            case 0: exitValue=3; break;
            case 1: exitValue=4 ;break;
            case 2: exitValue=2;break;
```

```
            default: exitValue=0;
    }
}
```

Listagem D.10 – Função de análise de níveis.

```
void searchLevels(int pin){
//*detect how many diferent tension levels found. Applied to the indicated pin on function argument
    int i,j,k, endIf,newLevelDetected,level;
    float analysingValue, newMean,auxCounter;
    samplesLevelCounter = malloc(sizeof *samplesLevelCounter *MEASUREMENTS_NR);
    ghostLevels=0;   levelsCounter=1;
   for (i=0;i<MEASUREMENTS_NR;i++) samplesLevelCounter[i]=0;              //_initialize_
   levelsMeanValues[pin][0]=measurements[pin][0];                        //_copy the first measurement: first level_
   samplesLevelCounter[0]=1;
//_increases/initialize nr of samples to the first level_
   for(i=1;i<MEASUREMENTS_NR;i++){                                       //_Start levels detection_
            newLevelDetected=1;
            analysingValue=measurements[pin][i];            //_copy to analysingValue the sample in analysis_
            endIf=0;
            for(j=0;j<levelsCounter;j++){
if (analysingValue<levelsMeanValues[pin][j]+TOLERANCE &&            analysingValue>levelsMeanValues[pin][j]-
TOLERANCE && endIf==0){
                                    newLevelDetected=0;          //_current sample attached to previous level_
                                    level=j; endIf=1;
                            }
            }
            if (newLevelDetected==1){                                          //_new level detected_
                            levelsMeanValues[pin][levelsCounter]=analysingValue;      //_save the new level_
                             levelsCounter++;
             }else{
                            samplesLevelCounter[level]++;                              //_increases level counter_
                            newMean=levelsMeanValues[pin][level]*(samplesLevelCounter[level]-1)+analysingValue;
                            //_re-calculates de mean (Step 1)_
                            levelsMeanValues[pin][level]=newMean/samplesLevelCounter[level];
                            //_re-calculates de mean (Step 2)_
            }
   }
    for (k=0;k< levelsCounter;k++){
if(samplesLevelCounter[k]<maxMeanDeviationSamples*MEASUREMENTS_NR){
            //_Detect level with a low number of measurements_
                            ghostLevels++;
                            globalGhostLevels++;
             }
    }
}
```

D.6. Gerador de instruções para aquisição de medições digitais

As principais funções utilizadas na implementação dos módulos geradores de instruções para aquisição de medições digitais (sincronizadas e não sincronizadas), descritos nas seções 3.6.3 e 3.8.3 deste documento, respetivamente encontram-se apresentados nas Listagens D.11, D.12 , D.13 e D.14. A implementação foi realizada na linguagem C#.

Listagem D.11 – Função para gerar instruções de aquisição de medições digitais não sincronizadas.

```
public void selectMeasurements(){
    writeCOMfile();                                                    //_Write communication setings_
     Process runningProcess = new Process();                           //_Create process to communicate_
     runningProcess.StartInfo.CreateNoWindow = true;
     runningProcess = System.Diagnostics.Process.Start("moduleCOM.exe"); //_Start communication_
     int idprocess = runningProcess.Id;
     while (runningProcess.HasExited == false) {                       //_Wait end of communication_
     }
    dataConversion();                                                  //_Convert received data_
}
```

Listagem D.12 – Função de escrita no ficheiro utilizado para transferir dados para a comunicação com o sistema de aquisição de dados: medição digital não sincronizada.

```
    public void writeCOMfile(int i, int j, string storagePath) {
       StreamWriter fileCOM = new StreamWriter("comdata.rep");        //_Open communication settings file_
       fileCOM.WriteLine(portName);                                   //_Write COM port name_
       fileCOM.WriteLine(storagePath);                                //_Write storage path_
       fileCOM.WriteLine("0010");                                     //_Write instruction opcode
       fileCOM.WriteLine(measurementsNR);          //_Write the number of measurements requested_
       fileCOM.WriteLine(pinsNR.ToString());       //_Write number of pins pin to measure_
       fileCOM.WriteLine(pinGND.ToString());       //_Write reference (GND) pin to measurements_
       fileCOM.Close();                            //_Close file_
    }
```

Listagem D.13 – Função de escrita no ficheiro utilizado para transferir dados para a comunicação com o sistema de aquisição de dados: medição digital sincronizada.

```
    public void writeCOMfile(int i, int j, string storagePath) {
       StreamWriter fileCOM = new StreamWriter("comdata.rep");        //_Open communication settings file_
       fileCOM.WriteLine(portName);                                   //_Write COM port name_
       fileCOM.WriteLine(storagePath);                                //_Write storage path_
       fileCOM.WriteLine("0011");                                     //_Write instruction opcode
       fileCOM.WriteLine(measurementsNR);          //_Write the number of measurements requested_
       fileCOM.WriteLine(pinCLK.ToString());       //_Write clock pin to synchronize _
       fileCOM.WriteLine(pinGND.ToString());       //_Write reference (GND) pin to measurements_
       fileCOM.WriteLine(clkPeriod_4.ToString());  //_Write ¼ of period from CLK signal_
       fileCOM.Close();                            //_Close file_
    }
```

Listagem D.14 – Funções de conversão dos dados recebidos e escrita na estrutura de dados.

```
public void dataConversion(){
UInt64 lineData;
        int i;
        createFiles();                                  //_Create files to store converted measurements_
        //_Open non-processed measurements file_
        StreamReader fileResult = new StreamReader(path + "\\unprocessedMeasurements.rep");
        for (i = 0; i < measurementsNR; i++){                                   //_Read all lines of data_
                lineData = Convert.ToUInt64(fileResult.ReadLine());     //_Convert line of data_
                writeIntoFiles(lineData);                                       //_Write converted data into files_
        }
        fileResult.Close();                                                     //_Close files_
}

public void writeIntoFiles(UInt64 data){                //*Receives data into a 16 bits array and serializes it
        int i;
        UInt64  excess;
        uint [] dataSplit=new uint [16];
        for (i = 0; i < 16; i++){
                excess = data % 10;                             //_Uses rest of integer division by 10 to get LSB_
                data = data / 10;                               //_Exclude last bit to next operation_
                dataSplit[i] = excess;                          //_Store LSB_
        }
        //_Open files to append data_
        StreamWriter fileM01 = File.AppendText(path + "\\MeasurementsP1.MED");
        StreamWriter fileM02 = File.AppendText(path + "\\MeasurementsP2.MED");
        (...)
        StreamWriter fileM16 = File.AppendText(path + "\\MeasurementsP16.MED");
        //_Append data and close the file_
        fileM01.WriteLine(dataSplit[0].ToString()); fileM01.Close();
        fileM02.WriteLine(dataSplit[1].ToString()); fileM02.Close();
        (...)
        fileM16.WriteLine(dataSplit[15].ToString()); fileM16.Close();
}
```

A função de ajuste do nome dos ficheiros, de acordo com o a seção D.2, encontra-se apresentada na Listagem D.15.

Listagem D.15 – Função de reajustamento do nome dos ficheiros.

```
public void renameFiles(){
        int i;
        String file;
        if ((pinsNR % 2) == 1) pinsNR++;        //_Re-configure pins to select pins of physical socket to read_
        int halfPins = pinsNR / 2;
        int factor = (16 - pinsNR) / 2;
        for (i = (9 - factor); i < (9 + factor); i++){                          //_Range of files to delete_
            try{
                file = path + "\\MeasurementsP" + i.ToString() + ".MED";
                System.IO.File.Delete(@file);                                   //_Delete non-interest files_
            }
            catch (Exception) {
            }
        }
        int[] pins = new int[pinsNR];
        for (i = 0; i < pinsNR; i++){           //_(the chip must be always placed on top of the socket)_
             if (i < (halfPins)){               //_Make the conversion of pins number to postion of pin in the socket_
                pins[i] = i;
             }
```

```
            else{
                pins[i] = (16 - halfPins) + (i - halfPins);
            }
        }
        String file2;
        int j = 9-factor;
        for (i = (9+factor); i <=16; i++){          //_Range of pins to re-adjust_
            try{
                file = path + "\\MeasurementsP" + i.ToString() + ".MED";     //_Original file name_
                file2 = path + "\\MeasurementsP" + j.ToString() + ".MED";   //_Readjusted file name_
                System.IO.File.Move(@file, @file2);                          //_Re-adjust files names_
                j++;
            }
            catch (Exception){
            }
        }
}
```

D.7. Algoritmo de análise combinatória

O algoritmo de análise combinatória foi descrito na seção 3.6.4 deste documento. As funções para deteção de lógica sequencial e escrita do corpo do ficheiro VHDL (em linguagem C) encontram-se disponíveis nas Listagens D.16 e D.17, respetivamente.

Listagem D.16 – Função de deteção de lógica combinatória.

```
int sequencialLogicDetection(){                    //*_This function determine if there are sequential logic involved_
int i,j,k,inputIndex,outputIndex;
int *storeInputs, *storeOutputs,*storeInputs2, *storeOutputs2;
int a1,a2,b1,b2;
if (inputPins==0) return 1;                                  //_if there are no input the circuit must be sequential_
//_Memory allocation_
 storeInputs=malloc(sizeof *storeInputs *inputPins);
 storeOutputs=malloc(sizeof *storeOutputs *(PINS_NR-inputPins));
 storeInputs2=malloc(sizeof *storeInputs2 *inputPins);
 storeOutputs2=malloc(sizeof *storeOutputs2 *(PINS_NR-inputPins));
 for(i=0;i<newMeasurementsNr-1;i++){
     for(j=i+1;j<newMeasurementsNr;j++){         //_Compare with all data_
            inputIndex=0;outputIndex=0;
            for(k=0;k<PINS_NR;k++){
                 switch(pinsType[k]){     //_Build input and output combinations_
                      case INPUT_PIN: storeInputs[inputIndex]=processedMeasurements[k][i];
                                                  storeInputs2[inputIndex]=processedMeasurements[k][j];
                                                   inputIndex++; break;
                      case OUTPUT_PIN: storeOutputs[outputIndex]=processedMeasurements[k][i];
                                                 storeOutputs2[outputIndex]=processedMeasurements[k][j];
                                                 outputIndex++; break;
                      }
            }
            a2=1;
             for (a1=0;a1<inputPins;a1++){
                      if(storeInputs[a1]!=storeInputs2[a1]) a2=0;                  //_Compare input combination_
            }
            if(a2){                                                                          //_if input combination is the same_
                      a2=1;
                      for (b1=0;b1<(PINS_NR-inputPins);b1++){
                      //_Compare output combination when they have same input_
                           if(storeOutputs[b1]!=storeOutputs2[b1])  return 1;       //_Sequential logic detected_
                     }
           }
 }
return 0;                                                                                  //_Sequential logic: not detected_
}
```

Listagem D.17 – Função de escrita do corpo do ficheiro VHDL para circuitos combinatórios.

```
void writeBodyBasicDigitalFileVHDL(int pin){                                    //* write the body of VHDL file
    int i,j,comma;
    int secondCondition=0;
    int intAux;
    char strAux1[500], strAux2[500],strAux[50];
    highProb=100*(newMeasurementsNr)/(power(2,inputPins));                  //_Write estimated success rate_
   fprintf(ficheiroVHDL,"--Probability of success(%%): %f\n", highProb);
   //_creates auxiliar signal_
   fprintf(ficheiroVHDL,"signal concatenate : STD_LOGIC_VECTOR (%i downto 0);\nbegin\n\tprocess(",inputPins-1);
```

```
for (i=0;i<PINS_NR;i++){
      if (pinsType[i]==1){
        sprintf(strAux, "%i", i+1);
        if (comma==1)
          fprintf(ficheiroVHDL,",");
        fprintf(ficheiroVHDL,"pin%s",strAux);
        comma=1;
      }
  }
  int readPins=1;
  char strAux3[1000];
  //_Concatenate inputpins_
  fprintf(ficheiroVHDL,")\n\tbegin\n\t\tconcatenate<=");
  for (i=0;i<PINS_NR;i++){
                        if(pinsType[i]==1){
                                strcpy(strAux3,"pin");
                                sprintf(strAux, "%i", i+1);
                                strcat(strAux3,strAux);
                                if (readPins!=inputPins) strcat(strAux3," & ");
                                readPins++;
                                fprintf(ficheiroVHDL,strAux3);
                        }
  }
  fprintf(ficheiroVHDL,";\n\n\t\tcase concatenate is\n");                    //_creates case statement_
   for (i=0;i<newMeasurementsNr;i++){                                       //_create case options_
      strcpy(strAux1,"\t\t\twhen ");
      fprintf(ficheiroVHDL,strAux1);
      fprintf(ficheiroVHDL,"%c",34);
      for (j=0;j<PINS_NR;j++){
            if (pinsType[j]==1) {
                        sprintf(strAux1, "%i", processedMeasurements[j][i]);
                        fprintf(ficheiroVHDL,strAux1);
            }
      }
                        fprintf(ficheiroVHDL,"%c",34);              //_Add special character_
                        fprintf(ficheiroVHDL,"=> ");
      for (j=0;j<PINS_NR;j++){                                      //_create the events when the specified case happens_
            if (pinsType[j]==2) {
                        strcpy(strAux1,"pin");
                        sprintf(strAux2, "%i", j+1);
                        strcat(strAux1,strAux2);
                        fprintf(ficheiroVHDL,strAux1);
                        fprintf(ficheiroVHDL," <= '%i'; ",processedMeasurements[j][i]);
            }
      }
      fprintf(ficheiroVHDL,"\n");
   }
   //_write the dafault case_
   fprintf(ficheiroVHDL,"\t\t\twhen others => ");
            for (j=0;j<PINS_NR;j++){
            if (pinsType[j]==2) {
                        strcpy(strAux1,"pin");
                        sprintf(strAux2, "%i", j+1);
                        strcat(strAux1,strAux2);
                        fprintf(ficheiroVHDL,strAux1);
                        fprintf(ficheiroVHDL," <= '0';");
            }
   }
   fprintf(ficheiroVHDL,"\n\t\tend case;\n");
}
```

D.8. Gerador de instruções para deteção de sinais periódicos

As principais funções utilizadas na implementação do módulo gerador de instruções para deteção de sinais periódicos (descrito na seção 3.7.3 deste documento) encontram-se apresentadas na Listagem D.18.

Listagem D.18 – Função de deteção de lógica combinatória.

```
private void selectMeasurements() {
        int i, j,k;
        Decimal [] temporaryData=new Decimal [measurementsNR];
        string pathComplete;
        if ((pinsNR % 2) == 1) pinsNR++;          //_Re-configure pins to select pins of physical socket to read_
        int[] pins = new int[pinsNR];
        int halfPins = pinsNR / 2;
        for (i = 0; i < pinsNR; i++) {                  //_(the chip must be always placed on top of the socket)_
           if (i < (halfPins)) {              //_Make the conversion of pins number to postion of pin in the socket_
              pins[i] = i;
           }
           else {
              pins[i] = (16 - halfPins) + (i - halfPins);
           }
        }
        int k;
        Decimal[] temporaryData = new Decimal[measurementsNR];
        for (i = 0; i < pinsNR; i++) {       //_Evaluate all pins_
           if (pinsType[i]==INPUT_CODE){    //_Select all input pins_
              pathComplete = path + "\\measurementsP" + (i + 1).ToString() + ".MED";
              writeCOMfile();                                                    //_Write communication setings_
              Process runningProcess = new Process();                            //_Create process to communicate_
              runningProcess.StartInfo.CreateNoWindow = true;
              runningProcess = System.Diagnostics.Process.Start("moduleCOM.exe"); //_Start communication_
              int idprocess = runningProcess.Id;
             while (runningProcess.HasExited == false) {                         //_Wait end of communication_
             }
             StreamReader file2 = new StreamReader(pathComplete);                        //_Open received data file_
             for (k = 0; k < measurementsNR; k++){
                temporaryData[k] = Convert.ToDecimal(file2.ReadLine());                  //_Read data_
             }
             file2.Close();                                                              //_Close file_
             for (k = 1; k < measurementsNR; k++)  {
                temporaryData[k - 1] = temporaryData[k] - temporaryData[k - 1];          //_Convert data_
             }
             StreamWriter file3 = new StreamWriter(pathComplete);                        //_Open new file_
             for (k = 0; k < measurementsNR-1; k++){
                file3.WriteLine(temporaryData[k].ToString());                            //_Write converted data_
             }
             file3.Close();                                                              //_CLose file_
           }
        }
     }
```

Nesta etapa, o ficheiro de comunicações escrito é igual ao descrito na Listagem D.4, alterando apenas o *opcode* para o valor “0100”, correspondente à instrução de deteção de sinais periódicos.

D.9. Algoritmo de deteção do sinal de relógio

As principais funções utilizadas na implementação do algoritmo de deteção do sinal de relógio (descrito na seção 3.7.4 deste documento) encontram-se apresentadas nas Listagens D.19 e D.20.

Listagem D.19 – Função de análise da distribuição dos dados e classificação dos mesmos em função da periodicidade.

```
void checkDataDistribution(){
    //*Check distribution of data to detect periodical signals and the respective period
    //*Analyzes possible sampling errors
    //*Negative values means constant signal
    int i,j,samplesOutOfRange;
    float sum;
    for(i=0;i<PINS_NR;i++){
        sum=0; pinsEvaluation[i]=NON_PERIODICAL;                    //_Variables initialization_
        if(pinsType[i]==1){
            //_Mean calculation for each input pin and respective tolerance_
            for(j=0;j<MEASUREMENTS_NR;j++) sum+=measurements[i][j];
                clkMeanValue[i]=sum/MEASUREMENTS_NR;
                clkTolerance[i]=PERCENTUAL_TOLERANCE*clkMeanValue[i];
                if(clkTolerance[i]<SENSIBILITY_COMPENSATION) clkTolerance[i]=SENSIBILITY_COMPENSATION;
                //_Counting possible sampling errors_
                samplesOutOfRange=0;
                for(j=0;j<MEASUREMENTS_NR;j++){
                   if(measurements[i][j]>clkMeanValue[i]+clkTolerance[i] || measurements[i][j]<clkMeanValue[i]-
                                                                                                clkTolerance[i])
                        samplesOutOfRange++;
  }
            //_Classifies  pins which has samples in correct range as periodical_
            if(samplesOutOfRange<=SAMPLE_PERCENTUAL_TOLERANCE*CYCLES_NUMBER){
                 pinsEvaluation[i]=PERIODICAL;
            }
        }

    }
}
```

Listagem D.20 – Função de identificação do pino de relógio.

```
int detectClockPin(){                    //*identtifies the pin which has the periodical signal with higher frequency
    int i, clkPin=17, PreviousValueStored=FALSE;
    for(i=0;i<PINS_NR;i++){                                         //_Analyzes all pins_
        if(pinsEvaluation[i]==PERIODICAL){                          //_Select periodical pins_
            if(PreviousValueStored==FALSE){                         //_Atributes initial values_
                if(clkMeanValue[i]>0){
                    minimumPeriod=clkMeanValue[i]; PreviousValueStored=TRUE, clkPin=i;
                }
            } else{                                                 //_Select lower value for signal período_
                if(clkMeanValue[i]<minimumPeriod && clkMeanValue[i]>0){
                    minimumPeriod==clkMeanValue[i]; clkPin=i;
                }
            }
        }
    }
  return clkPin;                                                    //_returns the number of clock pin_
}
```

D.10. Algoritmo de deteção do sinal de *reset*

As principais funções utilizadas na implementação do algoritmo de deteção do sinal de *reset* (descrito na seção 3.8.3 deste documento) encontram-se apresentadas na Listagem D.21.

Listagem D.21 – Funções para deteção do sinal de *reset* (concebidas para extensão à deteção de sinal de *preset* se necessário).

```
void resetDetection(){                                          //*This function detect reset pins
    int i, RST_DL, pinCLK;
    for (i=0;i<PINS_NR;i++) {                                   //_find and store de clock pin_
            if(pinsType[i]==4){
                  pinCLK=i;
                  break;
            }
   }
   clockPIN=pinCLK;
   for(i=0;i<PINS_NR;i++){                                  //_coordenates a reset detection for all input pins_
        if(pinsType[i]==1){
            RST_DL=detectRST_byLogicType(i,pinCLK,DIRECT_LOGIC,0);          //_verify reset (direct logic case)_
            if(RST_DL==0)
                detectRST_byLogicType(i,pinCLK,REVERSE_LOGIC,0);            //_verify reset (reverse logic case)_
       }
    }
}


int detectRST_byLogicType (int pin, int clockPIN, int logicType, int RST_or_PST){
            //*call the routine to analyzes reset and synchronism. Update the data in a file
   int i,j;
   int typeToWritte;
    int PST_RST_D;
    //_the function 'detectSynchronismRST' detects reset to a selected pin, according to a logic type_
    PST_RST_D=detectSynchronismRST(pin,clockPIN,logicType,RST_or_PST);
    if (PST_RST_D!=NO_RST_DETECTED){                            //_verify if a reset was detected_
       inputPins--;
       if(logicType==DIRECT_LOGIC){                             //_Assign pin in direct logic case_
            switch(PST_RST_D){
                        case 1: typeToWritte=RST_D_SF;break;
                        case 3: typeToWritte=RST_D_A;break;
                        case 4: typeToWritte=RST_D_SR;break;
            }
       } else {                                                 //_Assign pin in reverse logic case_
            switch(PST_RST_D){
                      case 1: typeToWritte=RST_R_SF;break;
                      case 3: typeToWritte=RST_R_A;break;
                      case 4: typeToWritte=RST_R_SR;break;
            }
       }
      updatePinsTypeFile(pin,typeToWritte);
      return 1;                                                 //_indicates a successful reset detection_
     }
    return 0;                                                   //_indicates a failed reset detection_
}
```

```
int detectSynchronismRST(int pin, int clockPIN, int logicType,int RST_or_PST){
    //*Detect if a selected pin could be a reset pin, by logic type(direct or reverse).
    //* Uses operations with clock signal to determinate if the reset could be assynchronus or syncronus.
    //_-----Table 1--------_
    // PLUS/TIMES          Description
    //   0   /    0        synchronus, with falling edge clock
    //   0   /    1        Nothing detected
    //   1   /    0        Assynchronus
    //   1   /    1        Synchronus
    int i,j,sum;
    int clockPlusVerification=0,clockTimesVerification=1;
    int clockPlusCounterVerification=0,clockTimesCounterVerification=1;
    int result, counterVerificationSamples=0;
    int     counterVerificationsResults[4]={1,0,1,1};
    //Index: 0-Not reverse CLK sync  1-nothing detected  2-not assyncronus  3-not CLK sync
    float accuracyRST=0;
    for(i=0;i<MEASUREMENTS_NR;i++){
        if(measurements[pin][i]==logicType){                //_verify if the possible reset pin in analysis is active _
            sum=0;
            for(j=0;j<PINS_NR;j++){
                if(pinsType[j]==2){                                 //_select all outputs _
                    sum+=measurements[j][i];                //_sum all outputs when possible reset pin is active _
                }
            }
        if(sum==RST_or_PST) {                                       //_ when reset is detected_
            clockPlusVerification+=measurements[clockPIN][i];       //_sum all clock values when reset is active_
            clockTimesVerification*=measurements[clockPIN][i];      //_multiply all clock values when reset is active_
        } else{                                                     //_make a counter analysis in failure cases_
            clockPlusCounterVerification+=measurements[clockPIN][i];
            clockTimesCounterVerification*=measurements[clockPIN][i];
counterVerificationsResults[ASSYNCRONUS-1]=0;       //_Initialize exception analysis structure_
            counterVerificationSamples++;
        }
        //_Analysis the failure cases according with table 1_
        switch(clockPlusCounterVerification){                       //_Counter verification analysis_
            case 0: if(clockTimesCounterVerification==0)
                    counterVerificationsResults[SYNCHRONUS_CLK_FALLING_EDGE-1]=0;
        break;
            default: if(clockTimesCounterVerification==1)
                    counterVerificationsResults[SYNCHRONUS_CLK_RISING_EDGE-1]=0;
        }
    }
}
```

D.11. Algoritmo de aproximação de circuitos sequenciais a uma máquina de estados

As principais funções utilizadas na implementação do algoritmo de análise de circuitos sequenciais (descrito na seção 3.9.4 deste documento) encontram-se apresentadas nas Listagens D.22, D.23 e D.24.

Listagem D.22 – Excerto da função principal do algoritmo de análise sequencial.

```
(...)
        statesNumber=0;                         //_initializes states number_
        resetInfo=resetStateAtribution();       //_Generate intial state (reset)_
        analysis_and_decision (resetInfo);              //_sequential analysis_
        transitionsTableBuilder();              //_Build transistions table_
        metric=analysisSuccess();               //_Success analysis_
        if(inputPins==0){                       //_Write VHDL code with or without inputs_
                create_VHDL_file_seq_No_inputs();
        }else{
                create_VHDL_file_seq_with_inputs();
        }
        (...)
```

Listagem D.23 – Função de análise e tomada de decisão.

```
void analysis_and_decision(int start){
  int verificationResult;
  int RST_statePreviouslyAssigned, matchingResetStatus;
  for(outputUnderAnalysis=start;outputUnderAnalysis<(NEW_MEASUREMENTS_NR-1);outputUnderAnalysis++){
    if(sequentialTable[outputUnderAnalysis+1].resetON==FALSE) {          //_non-reset situations_
    verificationResult=verifyOutputOccurrence(start,
outputUnderAnalysis,sequentialTable[outputUnderAnalysis].output);
       switch (verificationResult){
         case NEW: assign_NewState(outputUnderAnalysis);      //new state to assign; DECISION NOT DOUBFUL
break;
         default: assign_previousState(outputUnderAnalysis,start); //try assign to previous state DECISION DOUBFUL;
break;
       }
    } else{                                                  //_Establish transition to reset state_
        sequentialTable[outputUnderAnalysis].decision=DISCARDED;
        sequentialTable[outputUnderAnalysis].assignedState=-2;
        outputUnderAnalysis++;
        while(sequentialTable[outputUnderAnalysis].resetON==TRUE){
                sequentialTable[outputUnderAnalysis].decision=NOT_DOUBFUL;
                outputUnderAnalysis++;
        }
    outputUnderAnalysis--;
   }
  }
}
```

Listagem D.24 – Função de aproximação heurística a estado já existente.

```
void assignTemporaryState(int index_,int positionStart){
  int atributionVerification=FALSE;
  int preAtributionState;
  while(sequentialTable[index_].indexTransition<sequentialTable[index_].possibleTransitionsNumber &&
```

```
    atributionVerification==FALSE){
            preAtributionState=sequentialTable[index_].possibleTransitions[sequentialTable[index_].indexTransition];
            atributionVerification=verifySeletedAtribution(preAtributionState,index_,positionStart);
            if(atributionVerification==TRUE){
                    sequentialTable[index_].assignedState=preAtributionState;
            }
            sequentialTable[index_].indexTransition++;
    }
    if(atributionVerification==FALSE){
            sequentialTable[index_].decision=EXCLUDED;
            applyCorretionAlghoritm(index_,positionStart);
    }
}
```

D.12. Módulo de comunicação

O código fonte utilizado para implementar o módulo de comunicação descrito na seção 3.10 deste documento encontra-se apresentado na Listagem D.25.

Listagem D.25 – Código fonte completo (linguagem C#) do módulo de comunicação.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
using System.Threading;
using System.Security.Permissions;
using System.Security.Principal;
using System.Diagnostics;

namespace moduleCOM
{
   public partial class Form1 : Form
   {
      string portName, path;
      int opcode, measurementsNR;
      int arguments = 0;
      int[] argumentsArray ;
      public Form1()
      {
         InitializeComponent();
      }

      private void Form1_Load(object sender, EventArgs e)
      {
         executeCommunication();
      }

      public void executeCommunication(){
         StreamReader file = new StreamReader("comdata.rep");
         String communicationString;
         portName = file.ReadLine();
         serialPort1.PortName = portName;
         path = file.ReadLine();
         String opcodeStr = file.ReadLine();
         opcode = Convert.ToInt32(opcodeStr);
         measurementsNR = Convert.ToInt32(file.ReadLine());
         communicationString = opcodeStr + ":" + measurementsNR.ToString();
         switch (opcode){
            case 1: arguments = 2; break;
            case 10: arguments = 2; break;
            case 11: arguments = 3; break;
            case 100: arguments = 2; break;
            default: arguments = 0; communicationString = communicationString + ":0:0"; break;
         }
         if (arguments > 0){
            int i;
            argumentsArray = new int[arguments];
```

```
            for (i = 0; i < arguments; i++){
                argumentsArray[i] = Convert.ToInt32(file.ReadLine());
                communicationString = communicationString + ":" + argumentsArray[i].ToString();
            }
        }
        communicationString = communicationString + ":-\n";
        communicate(communicationString);
        receiveData();
        Close();
    }

    public void receiveData(){
        String receivedData;
        Boolean finish = false;
        StreamWriter file = new StreamWriter(path);
        while (finish==false){
            receivedData = serialPort1.ReadLine();
            if (receivedData.CompareTo("END")==1){
                finish = true;
            } else{
                file.Write(receivedData);
            }
        }
        serialPort1.Close();
        file.Close();
    }

    public void communicate(string sendData){
        serialPort1.Open();
        serialPort1.DiscardInBuffer();
        serialPort1.DiscardOutBuffer();
        try{
            serialPort1.Write(sendData);
        }
        catch{
            MessageBox.Show("Cabo desconectado", "Erro");
        }
    }

    private void Form1_FormClosing(object sender, FormClosingEventArgs e)
    {
        if (serialPort1.IsOpen == true){
            serialPort1.Close();
        }
    }

  }
}
```

D.13. Módulo de coordenação e interface

O código fonte utilizado para implementar o módulo de coordenação e interface (descrito na seção 3.11 deste documento) baseia-se na criação de processos e respetiva execução, para os diferentes módulos e encontra-se apresentado na Listagem D.26.

Listagem D.26 – Função de criação de um novo processo pelo módulo de coordenação e interface.

```
private int createProcessAndRunning(int stepNr, string processPath) {
        startTestTime = DateTime.Now;
        labelStartingTimeTest.Text = startTestTime.ToString();
        updateConsoleTextboxs();
        labelStatusTest.Text = "RUNNING";
        labelStatusTest.ForeColor = Color.Green;
        labelStage.Text = stepNr.ToString() + "/" +totalSteps.ToString();
        Process runningProcess = new Process();
         runningProcess.StartInfo.CreateNoWindow = true;
        runningProcess = System.Diagnostics.Process.Start(processPath);
        int idprocess = runningProcess.Id;
        while (runningProcess.HasExited  == false){
        }
        labelStatusTest.Text = "FINISHED";
        return runningProcess.ExitCode;
}
```

E. Utilização da interface gráfica do sistema desenvolvido

O sistema desenvolvido possui uma interface gráfica para que seja possível o utilizador inserir os parâmetros de operação e acompanhar as diferentes etapas do processo.

Ao iniciar a aplicação será exibida a janela principal onde o utilizador poderá inserir as configurações iniciais (ver Figura E.1): diretoria de armazenamento e nome do ficheiro VHDL, número de pinos do circuito integrado, tolerância máxima admitida, indicação dos pinos de saída e o número de amostras associadas a cada medição (com diferenciação entre medições analógicas e digitais.)

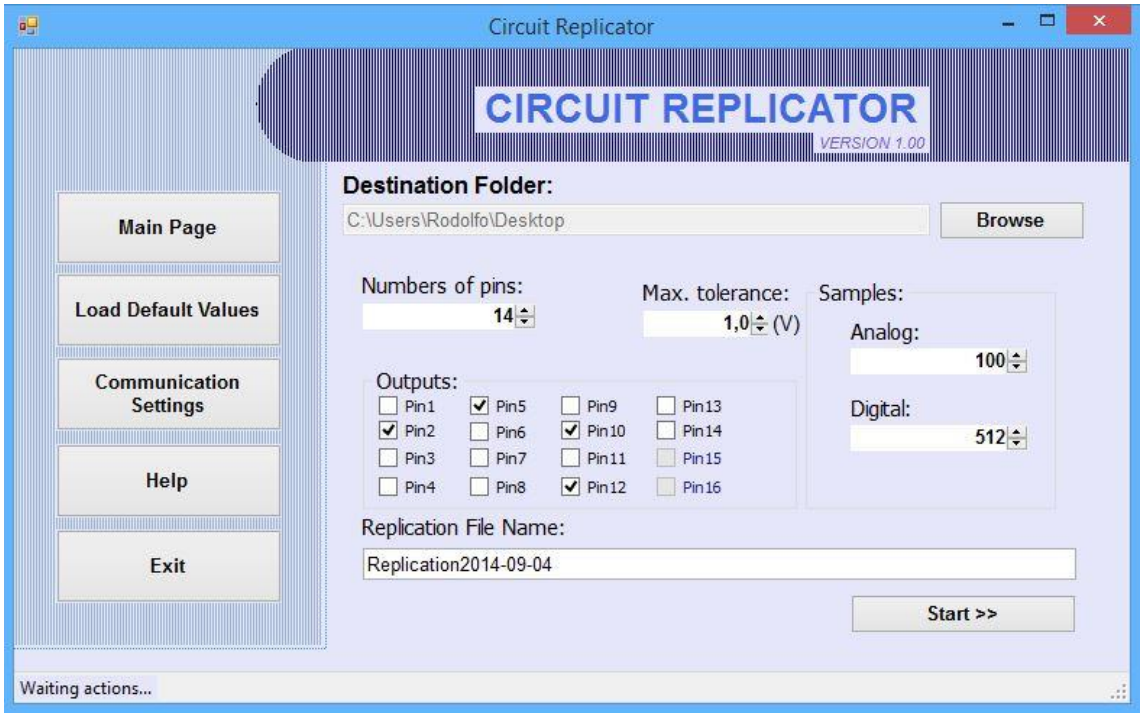

Figura E.1 – Interface para configuração introdução dos parâmetros iniciais.

No lado esquerdo da janela principal da interface encontra-se um menu com opções, sendo cada menu descrito na Tabela E.1.

Tabela E.1 – Descrição dos menus da interface gráfca.

Menu	Descrição
Main Page	Apresenta a janela principal (Figura E.1)
Load Default Values	Reconfigura os parâmetros, atribuindo as predefinições.
Communication Settings	Permite definir a porta série a utilizar na ligação com o sistema de aquisição de dados[32]. Apresenta ainda a taxa de transmissão de dados (Ver Figura E.2).
Help	Apresenta a descrição de cada parâmetro, permitindo ao utilizador efetuar uma melhor configuração dos parâmetros (ver Figura E.3).
Exit	Fecha a aplicação.

[32] A deteção de dispositivos conectados a portas série é feita de forma automática. No caso de existirem múltiplos dispositivos conectados pode ser necessário um reajuste manual.

As interfaces gráficas dos diferentes menus encontram-se apresentadas nas Figuras E.2 e E.3.

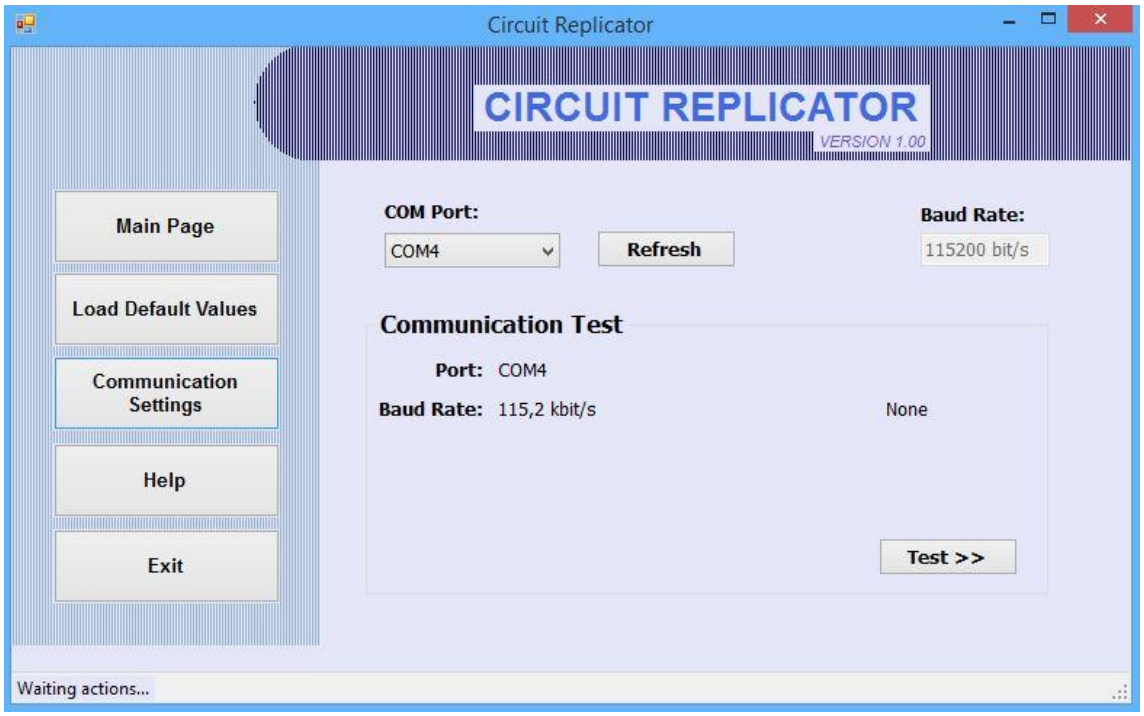

Figura E.2 – Menu de configuração das definições de comunicação (*Communication Settings*).

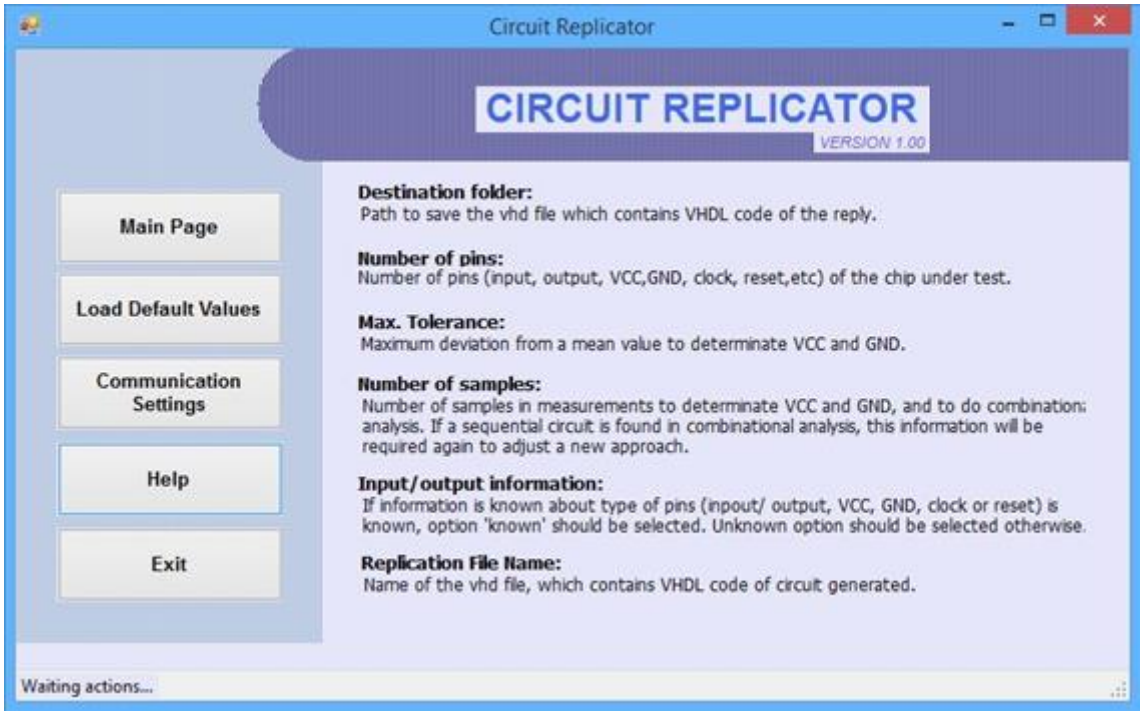

Figura E.3 - Menu de ajuda (*Help*).

Após inserção dos parâmetros de configuração, o utilizador deve carregar no botão "S*tart"* da janela apresentada na Figura E.1 para que o processo de replicação automática tenha início.

Durante a replicação é apresentada ao utilizador informação, em tempo real, sobre a etapa em que o processo se encontra, como apresentado na Figura E.4.

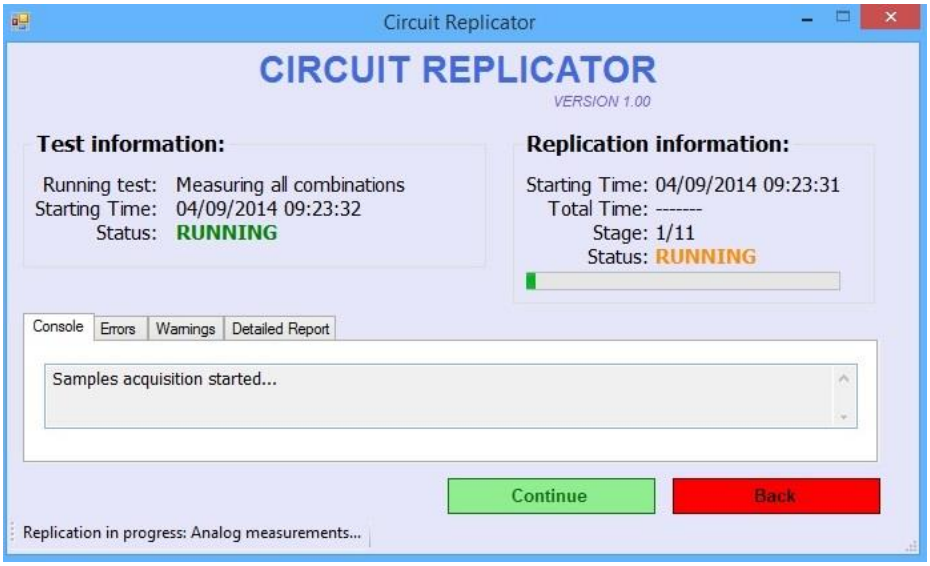

Figura E.4 – Janela de apresentação do progresso das etapas de replicação.

Quando o processo de replicação estiver indicado como concluído ("*FINISHED*") o utilizador deve clicar no botão "*Continue*" e o ficheiro VHDL gerado durante o processo de replicação irá abrir automaticamente. Quando circuito sequencial é detetado, uma nova janela de configurações é exibida, apresentada na Figura E.5.

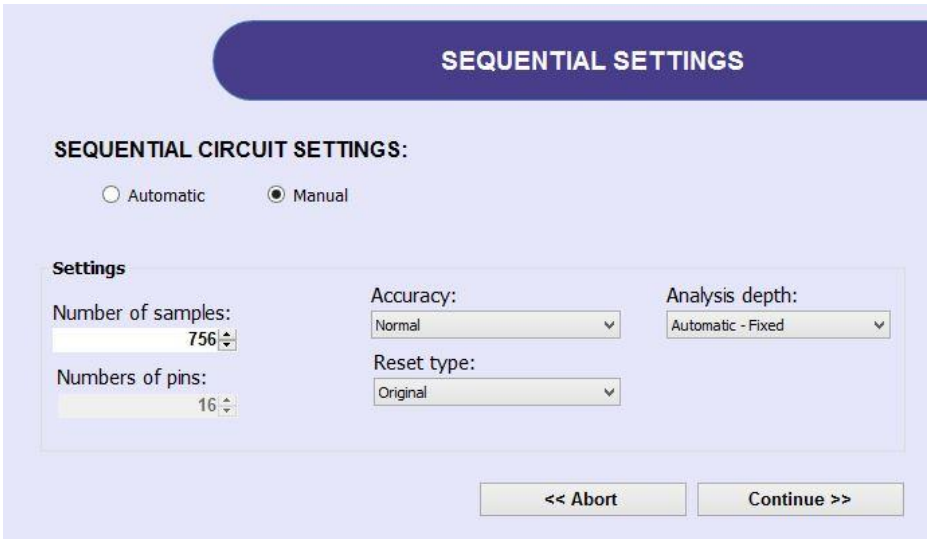

Figura E.5 – Interface para configuração dos parâmetros para análise sequencial.

Esta interface permite a configuração automática ou manual dos parâmetros. Um reajuste do número de medições é permitido bem como requerido. A precisão (baixa, média ou alta), tipo de *reset* (utilizar o observado, conversão para síncrono ou assíncrono) e a profundidade de análise pretendida (automático ou manual) são os parâmetros necessários para que o sistema prossiga a replicação.

F. Sistema de aquisição de dados

F.1. Circuito de aquisição sinais analógicos

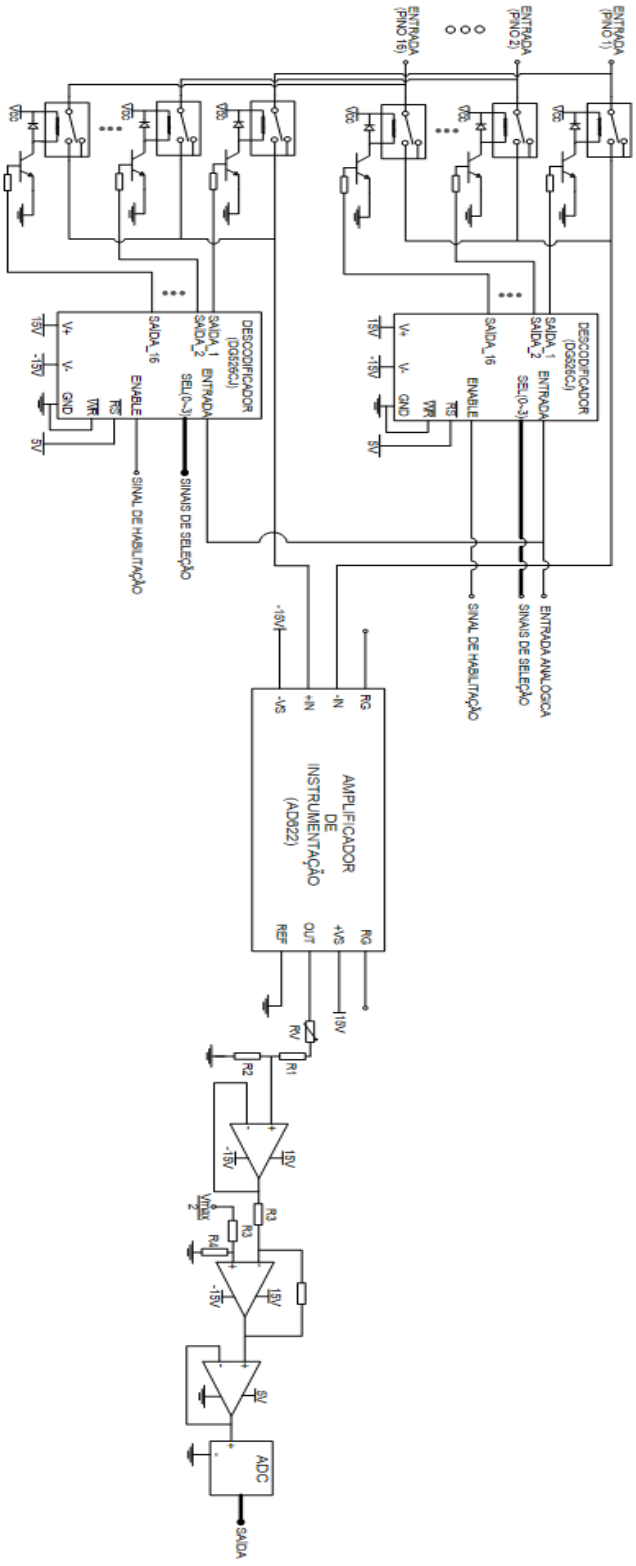

Figura F.1 – Circuito de aquisição de medições analógicas.

F.2. Circuito de aquisição de sinais digitais

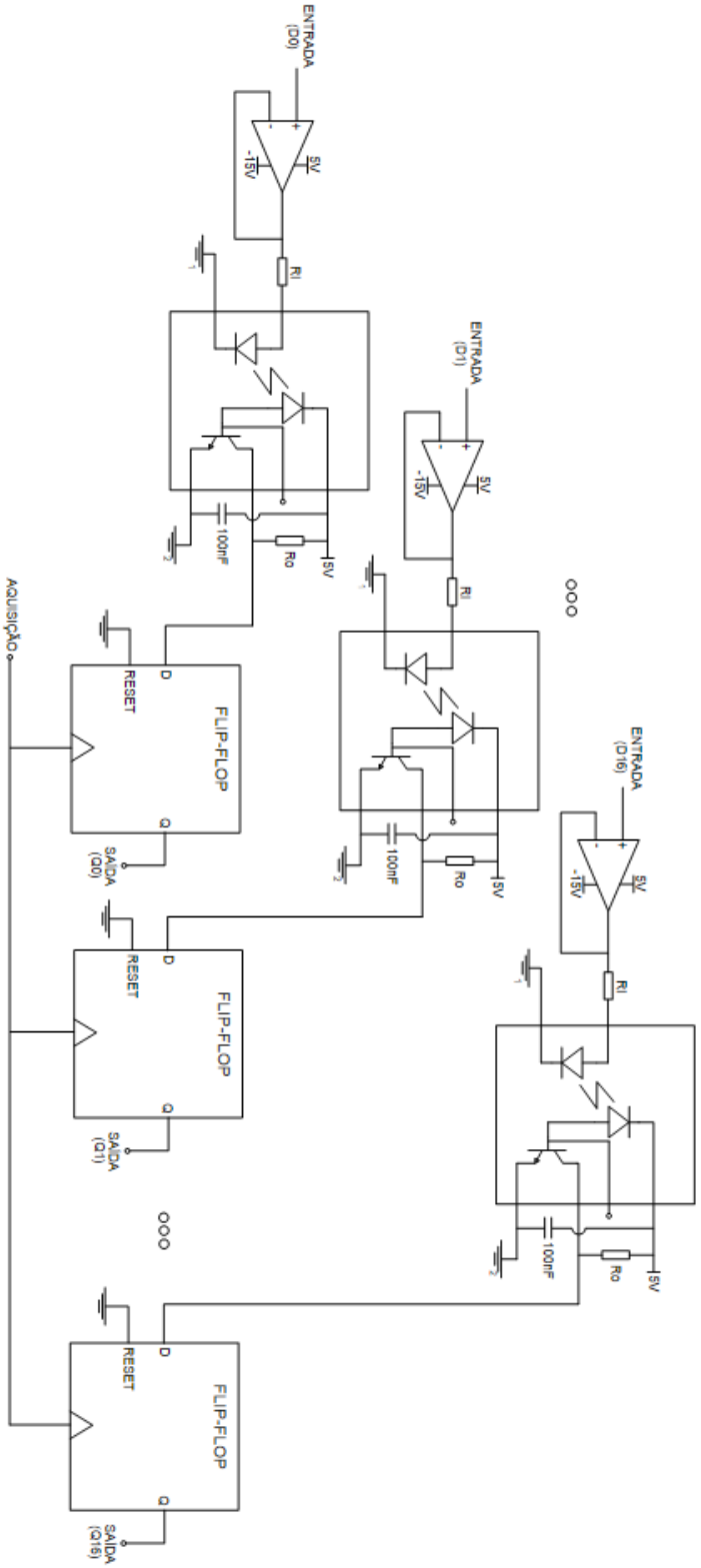

Figura F.2 – Circuito de aquisição de medições digitais.

F.3. Esquema elétrico do de aquisição de dados

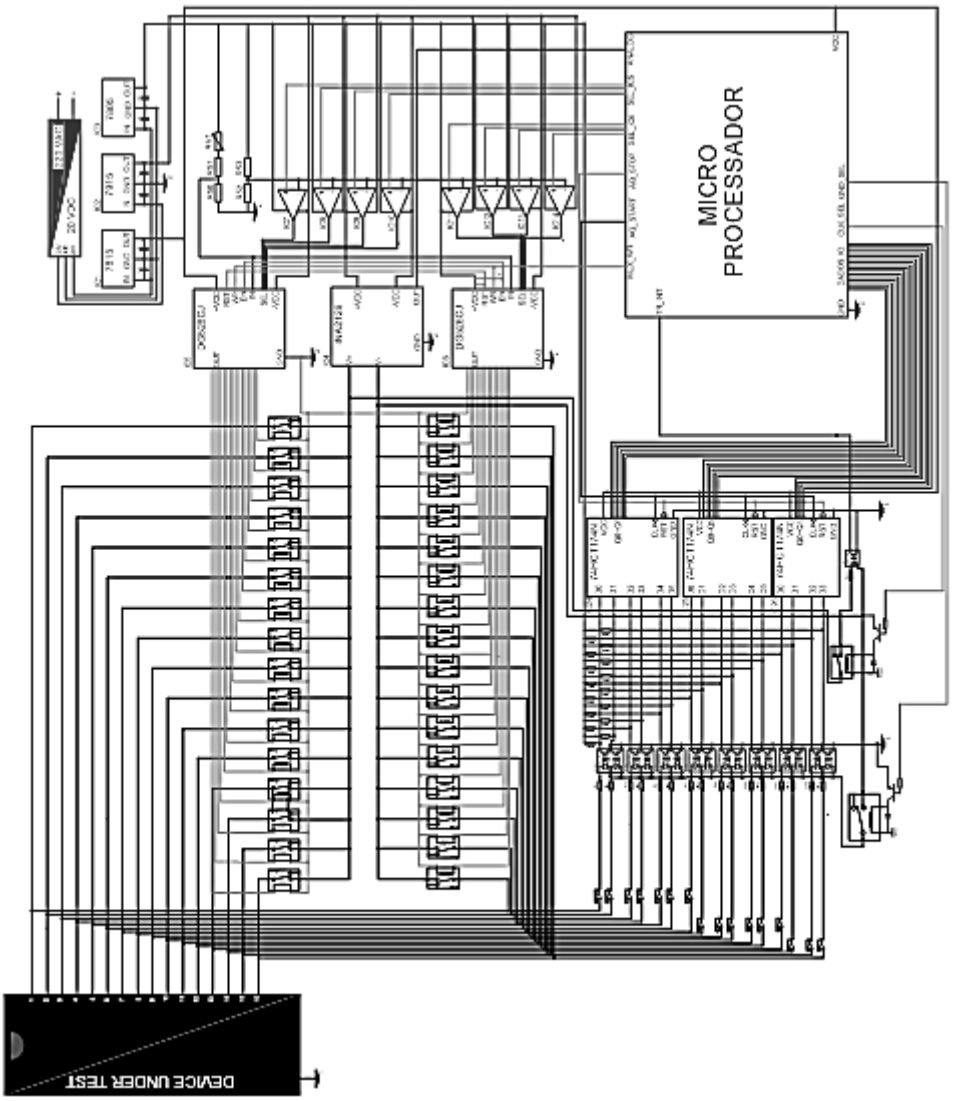

Figura F.3 – Esquema elétrico do sistema de aquisição de dados.

F.4. Código fonte do microcontrolador do sistema de aquisição de dados

Listagem F.1 – Código fonte do microcontrolador (*Arduíno* Mega 2560) do sistema de aquisição de dados

```
#include <string.h>
#include <math.h>

const int positiveMux_A3 = 22;
const int positiveMux_A2 = 24;
const int positiveMux_A1 = 26;
const int positiveMux_A0 = 28;
const int EnablePositiveMux = 30;
const int negativeMux_A3 = 23;
const int negativeMux_A2 = 25;
const int negativeMux_A1 = 27;
const int negativeMux_A0 = 29;
const int EnableNegativeMux = 31;
const int portADC = 1;
const int flipFlopAcq=37;
const int relayGND = 34;
const int relayCLK = 35;
//const int digitalPin 1 to 8 =>38 to 45;
//const int digitalPin 9 to 16 =>46 to 53;
// Variables will change:
boolean stringComplete=false;
boolean dataFound=false;
boolean finishedDataCLK=false,finishedSeqMeasurements=false;
String inputString = "";
int MEASUREMENTS_NR=-1;
int clkMeasurements=0,seqMeasurements=0;
unsigned int digitalSeq[1000];
unsigned long timeValues[105];
unsigned long periodCLK;

void setup() {
  pinMode(EnablePositiveMux, OUTPUT);
  pinMode(positiveMux_A3, OUTPUT);
  pinMode(positiveMux_A2, OUTPUT);
  pinMode(positiveMux_A1, OUTPUT);
  pinMode(positiveMux_A0, OUTPUT);
  pinMode(EnableNegativeMux, OUTPUT);
  pinMode(negativeMux_A3, OUTPUT);
  pinMode(negativeMux_A2, OUTPUT);
  pinMode(negativeMux_A1, OUTPUT);
  pinMode(negativeMux_A0, OUTPUT);
  pinMode(38,INPUT);
  pinMode(39,INPUT);
  pinMode(40,INPUT);
  pinMode(41,INPUT);
  pinMode(42,INPUT);
  pinMode(43,INPUT);
  pinMode(44,INPUT);
  pinMode(45,INPUT);
  pinMode(46,INPUT);
  pinMode(47,INPUT);
  pinMode(48,INPUT);
  pinMode(49,INPUT);
```

```
  pinMode(50,INPUT);
  pinMode(51,INPUT);
  pinMode(52,INPUT);
  pinMode(53,INPUT);
  pinMode(ADC,INPUT);
  pinMode(flipFlopAcq,OUTPUT);
  pinMode(relayGND,OUTPUT);
  pinMode(relayCLK,OUTPUT);
  // initialize serial communication:
  Serial.begin(115200);
  Serial.flush();
  initialization();
}

void initialization(){                    //_initializes controler pins
  int i;
  for(i=22;i<32;i++) digitalWrite(i,LOW);
  digitalWrite(flipFlopAcq,LOW);
  digitalWrite(relayGND,LOW);
  digitalWrite(relayCLK,LOW);
}

void transitionDetection(){                          //_INT 0: Save current time (uS) of INT 0 execution_
    timeValues[clkMeasurements]=micros();
    clkMeasurements++;
    if(clkMeasurements>MEASUREMENTS_NR){             //_Verify if measurements are enough_
        finishedDataCLK=true;                        //_Ativate flag to send data_
        detachInterrupt(0);                          //_Disable INT 0_
    }
}

void sequentialAcquisition(){                        //_Do digital sinchronized measures_
     int i, value;
     unsigned int sum=0;
     delayMicroseconds(periodCLK);                   //_Wait 1/5 of CLK period_
     digitalWrite(flipFlopAcq,HIGH);                 //Load data to flip flop_
     delayMicroseconds(1);
     digitalWrite(flipFlopAcq,LOW);
     for(i=0;i<16;i++){
      value=digitalRead(38+i);                       //_Read data from flip-flops_
      sum=sum+value*pow(2,i);                        //_Convert data to int_
      }
    digitalSeq[seqMeasurements]=sum;                 //_Store data_
    seqMeasurements++;
    if(seqMeasurements>MEASUREMENTS_NR){
        finishedSeqMeasurements=true;                //_Ativate flag to send data_
        detachInterrupt(0);                          //_Disable INT 0_
    }
}

void sendSeqdata(){
    int i;
    for(i=0;i<MEASUREMENTS_NR;i++){                  //_Send sequential data_
      Serial.println(digitalSeq[i],BIN);
    }
    Serial.println("END");
  digitalWrite(EnableNegativeMux, LOW);              //_Release reference relay_
  digitalWrite(EnablePositiveMux, LOW);              //_Release clock relay_
}

void sendCLKdata(){
```

```
   int i;
   for(i=0;i<MEASUREMENTS_NR;i++){                    //_Send clock data_
    Serial.println(timeValues[i]);
   }
   Serial.println("END");
  digitalWrite(EnableNegativeMux, LOW);               //_Release reference relay_
  digitalWrite(EnablePositiveMux, LOW);               //_Release clock relay_
}

void clockDetection(char *arguments){
  char *argA, *argB;
  int pinsNR,referenceMux,acquiredData=0;
  argA=strtok(arguments,":");                         //_Break function argument 1 from main string_
  argB=strtok(NULL,":");                              //_Break function argument 2 from main string_
  pinsNR=atoi(argA);                                  //_Identify number of pins to measure_
  referenceMux=atoi(argB);                            //_Select reference pin to measure_
  changeNegativeMux(referenceMux);                    //_Ativate reference relay_
  changePositiveMux(pinsNR);                          //_Ativate clock relay_
  digitalWrite(relayGND,HIGH);                        //_Ativate reference routing relay_
  digitalWrite(relayCLK,HIGH);                        //_Ativate clock routing relay_
  delay(5);
  attachInterrupt(0, transitionDetection, CHANGE);    //_Ativate intrruption_
}

void convertPositiveMuxIndex(int index){    //_Serializes positive mux selection signal_
   digitalWrite(positiveMux_A3, LOW);digitalWrite(positiveMux_A2, LOW);
   digitalWrite(positiveMux_A1, LOW);digitalWrite(positiveMux_A0, LOW);
   if(index>7){
     digitalWrite(positiveMux_A3, HIGH);
     index=index-8;
   }
   switch(index){
      case 1: digitalWrite(positiveMux_A0, HIGH); break;
      case 2: digitalWrite(positiveMux_A1, HIGH); break;
      case 3: digitalWrite(positiveMux_A1, HIGH); digitalWrite(positiveMux_A0, HIGH);break;
      case 4: digitalWrite(positiveMux_A2, HIGH); break;
      case 5: digitalWrite(positiveMux_A2, HIGH); digitalWrite(positiveMux_A0, HIGH);break;
      case 6: digitalWrite(positiveMux_A2, HIGH); digitalWrite(positiveMux_A1, HIGH);break;
      case 7: digitalWrite(positiveMux_A2, HIGH); digitalWrite(positiveMux_A1, HIGH);
     digitalWrite(positiveMux_A0, HIGH); break;
   }
}

void changePositiveMux(int relayIndex){
  digitalWrite(EnablePositiveMux, LOW);               //_Unable positive mux relay_
  delay(25);                                          //_Wait relay opening time_
  convertPositiveMuxIndex(relayIndex);                //_Select relay_
  digitalWrite(EnablePositiveMux, HIGH);              //_Enable positive mux relay_
  delay(25);                                          //_Wait relay closing time_
}

void convertNegativeMuxIndex(int index){   //_Serializes negative mux selection signal_
   digitalWrite(negativeMux_A3, LOW);digitalWrite(negativeMux_A2, LOW);
   digitalWrite(negativeMux_A1, LOW);digitalWrite(negativeMux_A0, LOW);
   if(index>7){
     digitalWrite(negativeMux_A3, HIGH);
     index=index-8;
   }
   switch(index){
      case 1: digitalWrite(negativeMux_A0, HIGH); break;
      case 2: digitalWrite(negativeMux_A1, HIGH); break;
```

```
      case 3: digitalWrite(negativeMux_A1, HIGH) ;digitalWrite(negativeMux_A0, HIGH);break;
      case 4: digitalWrite(negativeMux_A2, HIGH); break;
      case 5: digitalWrite(negativeMux_A2, HIGH); digitalWrite(negativeMux_A0, HIGH);break;
      case 6: digitalWrite(negativeMux_A2, HIGH);digitalWrite(negativeMux_A1, HIGH);break;
      case 7: digitalWrite(negativeMux_A2, HIGH);digitalWrite(negativeMux_A1, HIGH);
              digitalWrite(negativeMux_A0, HIGH);
   }
}

void changeNegativeMux(int relayIndex){
  digitalWrite(EnableNegativeMux, LOW);                 //_Unable positive mux relay_
  delay(25);                                            //_Wait relay opening time_
  convertNegativeMuxIndex(relayIndex);                  //_Select relay_
  digitalWrite(EnableNegativeMux, HIGH);                //_Enable positive mux relay_
  delay(25);                                            //_Wait relay closing time_
}

void analogReading(){                                   //*Provides N analog measurements
  int acquiredData=0;
  int analogValue;
  int waitingTimeUs, totalDelayUs;
  randomSeed(millis());
  while(acquiredData<MEASUREMENTS_NR){                  //_Acquire N samples_
    delay(5);                                           //_Stabilization time for ADC (fixed time)_
    waitingTimeUs=random(1,2000);                       //_Generate random time(us)_
    delayMicroseconds(waitingTimeUs);                   //_Wait random time to avoid periodical measurements_
    analogValue=analogRead(portADC);                    //_Acquire data_
    Serial.println(analogValue);                        //_Send data_
    acquiredData++;                                     //_Increment acquired samples_
  }
  Serial.println("END");                          //_Send message which indicates the end of measurements_
}

void digitalReading(int numberPins){
  int i;
  unsigned int sum=0;
  int value;
  digitalWrite(flipFlopAcq,HIGH);                       //Load data to flip flop_
  delayMicroseconds(1);
   digitalWrite(flipFlopAcq,LOW);
   for(i=0;i<16;i++){
    value=digitalRead(38+i);                            //_Read data from flip-flops_
    sum=sum+value*pow(2,i);                             //_Convert data to int_
  }
  Serial.println(sum,BIN);                              //_Send data_
}

void runDifferentialAnalogMeasurements(char *arguments){
  //*function to execute the instruction to make analog measurements
  char *argA, *argB;
  int positiveMux,negativeMux;
  argA=strtok(arguments,":");                           //_Break function argument 1 from main string_
  argB=strtok(NULL,":");                                //_Break function argument 2 from main string_
  positiveMux=atoi(argA);                               //_Select positive pin to measure_
  negativeMux=atoi(argB);                               //_Select negative pin to measure_
  changePositiveMux(positiveMux);                       //_Ativate positive relay_
  changeNegativeMux(negativeMux);                       //_Ativate negative relay_
  analogReading();                                      //_Do differential analog measurements_
  digitalWrite(EnablePositiveMux, LOW);                 //_Release all positive relays_
  digitalWrite(EnableNegativeMux, LOW);                 //_Release all negative relays_
}
```

```
void runCombinatorialMeasurements(char *arguments){
  char *argA, *argB;
  int pinsNR,referenceMux,acquiredData=0;
  int waitingTimeUs;
  randomSeed(millis());
  argA=strtok(arguments,":");                          //_Break function argument 1 from main string_
  argB=strtok(NULL,":");                               //_Break function argument 2 from main string_
  pinsNR=atoi(argA);                                   //_Identify number of pins to measure_
  referenceMux=atoi(argB);                             //_Select reference pin to measure_
  changeNegativeMux(referenceMux);                     //_Ativate reference relay_
  digitalWrite(relayGND,HIGH);
  delay(5);
  while(acquiredData<MEASUREMENTS_NR){
    digitalReading(pinsNR);
    acquiredData++;
    waitingTimeUs=random(100,2000);                    //_Generate random time(us)_
    delayMicroseconds(waitingTimeUs);                  //_Wait random time to avoid periodical measurements_
  }
  digitalWrite(relayGND,LOW);
  digitalWrite(EnableNegativeMux, LOW);                //_Release reference relay_
  delay(5);
  Serial.println("END");
}

void  runSequentialMeasurements(char *arguments){
  char *argA, *argB, *argC;
  int referenceMux,clkMux;
  argA=strtok(arguments,":");                          //_Break function argument 1 from main string_
  argB=strtok(NULL,":");                               //_Break function argument 2 from main string_
  argC=strtok(NULL,":");                               //_Break function argument 3 from main string_
  clkMux=atoi(argA);                                   //_Identify number of pins to measure_
  referenceMux=atoi(argB);                             //_Select reference pin to measure_
  periodCLK=atoi(argC)/5;                              //_Select  clk 1/4 period to measure_
  changeNegativeMux(referenceMux);                     //_Ativate reference relay_
  changePositiveMux(clkMux);                           //_Ativate clk relay_
  digitalWrite(relayGND,HIGH);
  digitalWrite(relayCLK,HIGH);
  delay(5);
  attachInterrupt(0, sequentialAcquisition, CHANGE);       //_Ativate intrruption_
}

void generalSerialReading() {
  char *opcode, *measurementsStr, *arguments;
  char instructionReceived[100];
  while (Serial.available()>0) {                       //_Reading input buffer_
    char inChar = (char)Serial.read();
     inputString += inChar;
    if (inChar == '\n') {                              //_Identify the end of string_
      stringComplete = true;
      dataFound=true;
    }
  }
  if(dataFound==true){                                 //_String received_
    stringComplete=false; dataFound=false;
    Serial.flush();                                    //_Clean input buffer_
    inputString.toCharArray(instructionReceived, 100) ;//_Convert to char array_
    inputString="";                                    //_Clean string for next communication_
    opcode=strtok(instructionReceived,":");            //_Breaking instruction: extrating opcode_
```

```
    measurementsStr=strtok(NULL,":");                   //_Breaking instruction: extrating measurements number_
    MEASUREMENTS_NR=atoi(measurementsStr);              //_Convert data type_
    arguments=strtok(NULL,"-");                         //_Breaking instruction: extrating arguments_
    //Execute instructions_
    if(strcmp(opcode,"0001")==0){
      runDifferentialAnalogMeasurements(arguments);     //_Run analog measurements_
    }
    if(strcmp(opcode,"0010")==0){
      runCombinatorialMeasurements(arguments);          //_Run digital measurements for combinatorial analysis_
    }
    if(strcmp(opcode,"0011")==0){
      runSequentialMeasurements(arguments);             //_Run digital measurements for combinatorial analysis_
    }
   if(strcmp(opcode,"0100")==0){
      clockDetection(arguments);                        //_Run transition detector_
    }
    if(strcmp(opcode,"1111")==0){
       Serial.println("1");
       Serial.println("END");                          //_Send message which indicates the end of measurements_
    }
  }
}

void loop() {
    generalSerialReading();                             //_Read instructions from serial port_
    if(finishedDataCLK==true){                          //_Verify if there are clock data to send_
       clkMeasurements=0;                               //_Re-intialize variables_
       sendCLKdata();                                   //_Send data_
       finishedDataCLK=false;                           //_Re-intialize variables_
    }
    if(finishedSeqMeasurements==true){                  //_Verify if there are sequential data to send_
       seqMeasurements=0;                               //_Re-intialize variables_
       sendSeqdata();                                   //_Send data_
       finishedSeqMeasurements=false;                   //_Re-intialize variables_
    }
}
```

F.5. Circuito de deteção de transições

O circuito detetor de transições é um circuito que a cada transição do sinal de entrada, ascendente ou descendente, reproduz na saída um impulso de curta duração. O circuito é baseado no atraso das portas lógicas e pode ser implementado como ilustrado na Figura F.4.

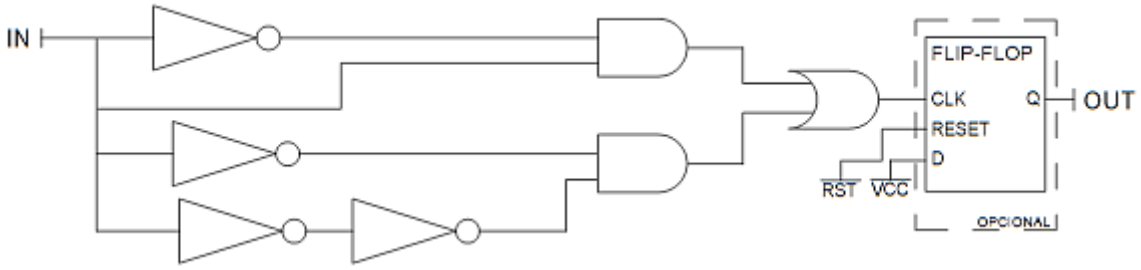

Figura F.4 – Circuito detetor de transições.

O circuito possui dos sub-detetores incorporados, que se encontram ligados à porta OR: à entrada superior está ligado o circuito de deteção transições ascendentes e na entrada inferior o circuito de deteção de transições descendentes. A utilização de um fl*ip-flop* na saída é opcional e destina-se a aumentar a duração do impulso de saída, que tipicamente é abaixo de uma dezena de nano-segundos [80], [81], para situações onde a largura do impulso necessite de ser aumentada para permitir a deteção. Nestas situações deve ser utilizada a entrada *reset* do *flip-flop* para determinar a largura do impulso.

A deteção da transição ascendente é baseada no atraso do sinal à saída da porta NOT comparativamente ao sinal de entrada, originando na entrada da porta AND um intervalo de tempo onde ambos os sinais se encontram no nível alto, logo após as transições ascendentes, provocando desta forma um curto impulso após as transições ascendentes do sinal de entrada, como apresentado na Figura F.5.

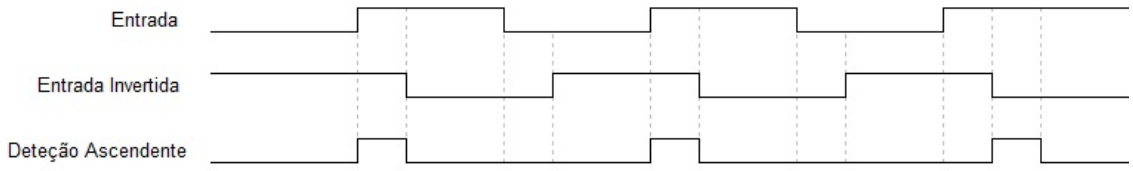

Figura F.5 – Diagrama temporal ilustrativo do funcionamento da deteção de uma transição ascendente.

A deteção da transição descendente é baseada no mesmo princípio descrito na deteção da transição ascendente, com a variante que nesta situação é necessário atrasar o sinal de entrada através de duas portas NOT de tal forma que o sinal original atrasado (pela via de dupla inversão) se encontra em atraso comparativamente ao sinal invertido, permitindo assim a existência de um curto intervalo de tempo onde existem dois sinais no nível alto na entrada da porta AND, originando um curto impulso após as transições descendentes. Note-se que devido ao maior atraso imputado ao circuito, o impulso na saída do circuito surge com um maior atraso após a ocorrência da transição

comparativamente à transição ascendentes, contudo é um atraso na ordem dos nano-segundos. Na Figura F.6 encontra-se ilustrado um diagrama temporal relativo à deteção de uma transição descendente.

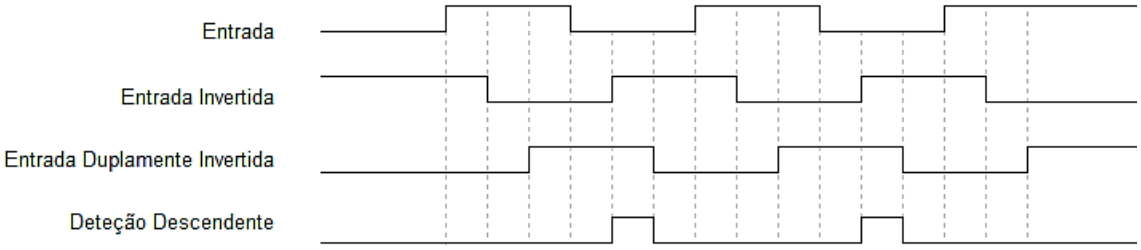

Figura F.6 – Diagrama temporal ilustrativo do funcionamento da deteção de uma transição ascendente.

A colocação de uma porta OR na saída que recebe nas entradas os sinais de deteção das transições ascendentes e descendentes irá permitir obter um sinal na saída que indique a ocorrência de qualquer transição, através de um impulso como ilustrado na Figura F.7.

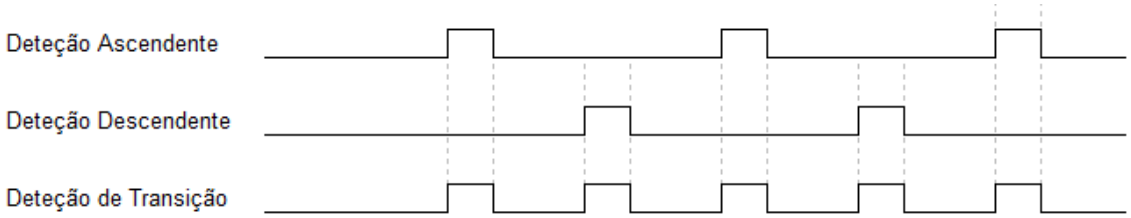

Figura F.7 – Deteção de todas as transições.

G. Esquema de montagem

A montagem do protótipo para verificação experimental do sistema proposto encontra-se apresentada na Figura G.1.

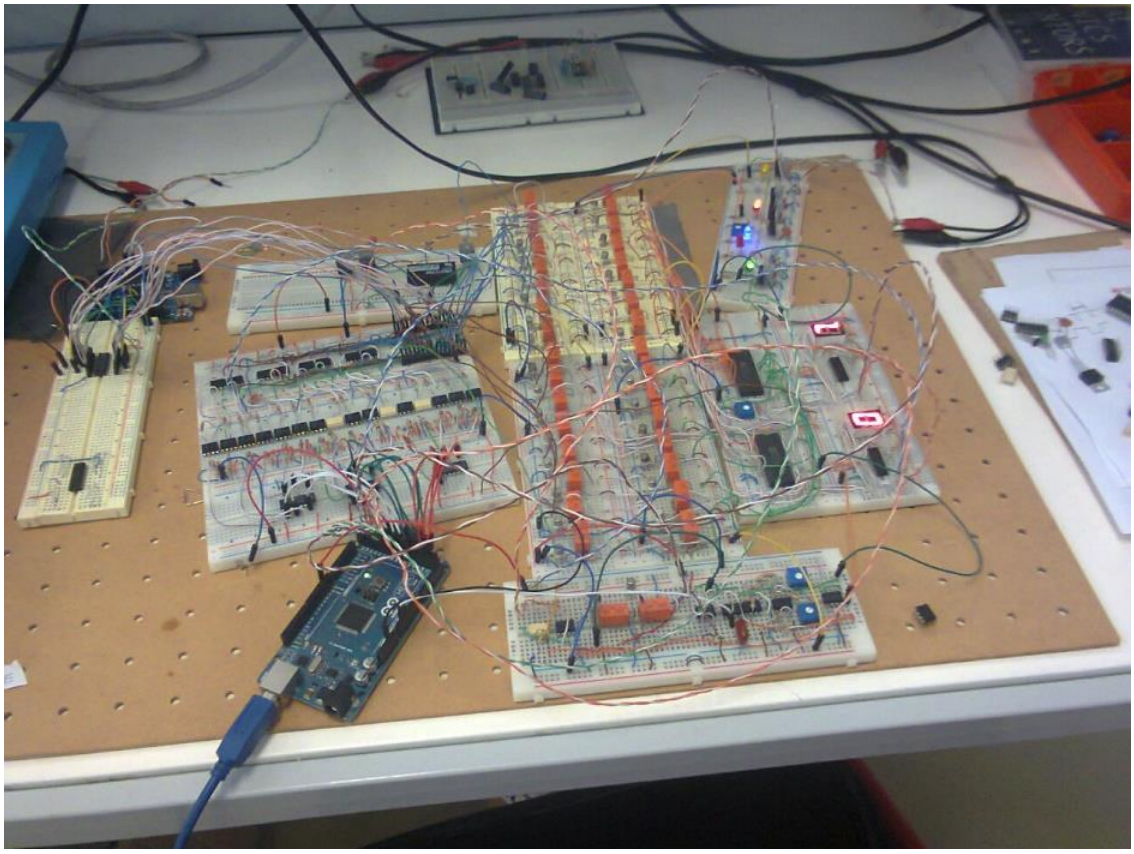

Figura G.1 – Montagem do protótipo experimental do sistema de replicação de circuitos integrados.

H. Resultados experimentais obtidos para circuitos combinatórios

H.1. Análise do impacto do número de amostras recolhidas

Para analisar o impacto do número de amostras na taxa de sucesso foram registadas as taxas de sucesso para diferentes situações, através da variação da relação entre amostras recolhidas e combinações de entrada possíveis. Para cada situação foram realizados três testes e selecionada a mediana das taxas de sucesso estimadas pelo sistema. Os resultados encontram-se apresentados na Tabela H.1.

Tabela H.1 – Medições realizadas com variação do número de amostras e combinações de entrada (R e M representam os números de amostras recolhidas e combinações de entrada, respetivamente).

R/M	Taxa de sucesso (%)								
	8 Entradas ($M = 256$)			6 Entradas ($M = 64$)			4 Entradas ($M = 16$)		
1	63	±	1	57	±	1	43	±	1
1,25	73	±	1	73	±	1	50	±	1
1,5	77	±	1	76	±	1	81	±	1
1,75	81	±	1	87	±	1	81	±	1
2	88	±	1	85	±	1	81	±	1
2,25	90	±	1	85	±	1	87	±	1
2,5	91	±	1	94	±	1	93	±	1
2,75	92	±	1	92	±	1	87	±	1
3	95	±	1	95	±	1	93	±	1
3,25	97	±	1	98	±	1	100	±	1
3,5	94	±	1	98	±	1	100	±	1
3,75	94	±	1	95	±	1	100	±	1
4	99	±	1	98	±	1	100	±	1
4,5	98	±	1	100	±	1	100	±	1
5	100	±	1	100	±	1	100	±	1
6	100	±	1	100	±	1	100	±	1

Com base nos valores da Tabela H.1, com recurso a *software ISE Design* foi possível obter a função com melhor aproximação aos pontos registados (regressão logística), tendo sido obtida a expressão

$$\begin{cases} \dfrac{98{,}7}{1 + 1{,}9e^{-1{,}27x}} & , x = 8 \\ \dfrac{99}{1 + 3{,}7e^{-1{,}72x}} & , x = 6 \\ \dfrac{98{,}9}{1 + 10{,}49e^{-2{,}13x}} & , x = 4 \end{cases} \quad \text{(H.1)}$$

onde x representa o número de entradas do circuito.

H.2. Esquemas lógicos obtidos pela sintetização do código VHDL gerado pelo sistema de replicação

Nas Figuras H.1, H.2 e H.3 encontram-se apresentadas os esquemas lógicos obtidos nos testes aos circuitos combinatórios referidos na seção 5.2.2 deste documento, para confirmação do comportamento. Após a sintetização, o *software* utilizado apresenta o resultado agrupado por LUTs (*Look Up Tables),* que por sua vez contêm as funções lógicas que descrevem o circuito sintetizado.

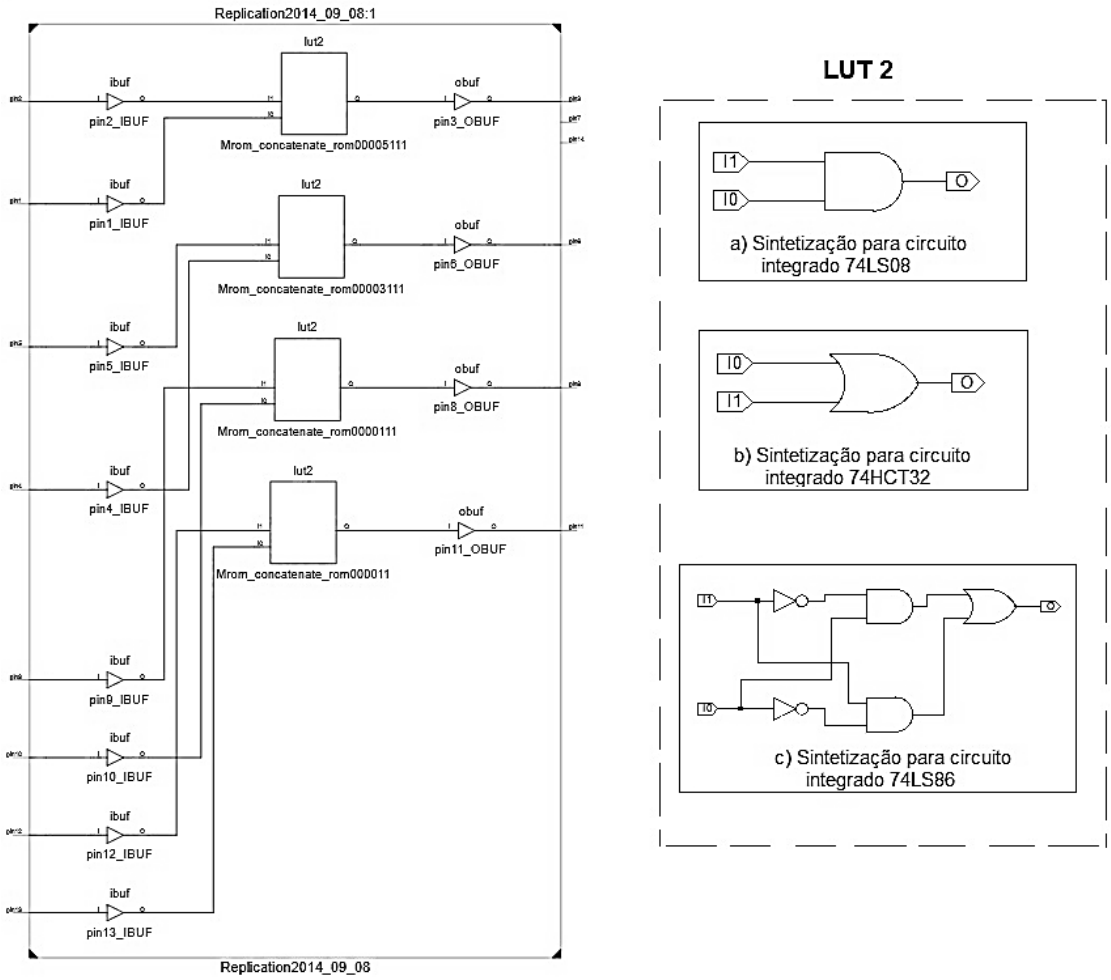

Figura H.1 – Esquema lógico obtido após sintetização das réplicas (testes: 74LS08 (#1), 74HCT32 (#2) e 74LS86 (#3)).

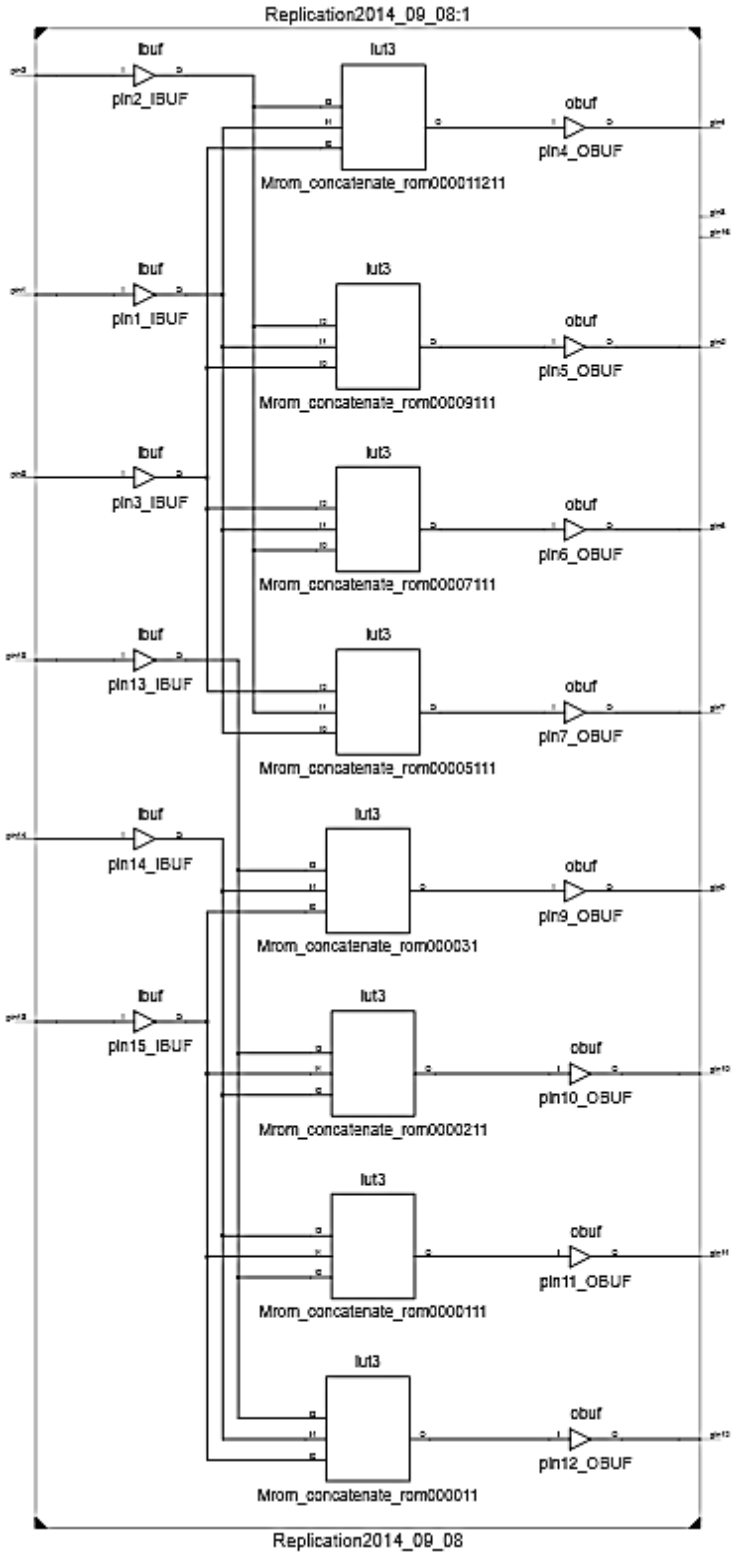

Figura H.2 – Esquema lógico obtido após sintetização da réplica (Descodificador: 74HC139 (teste #4)).

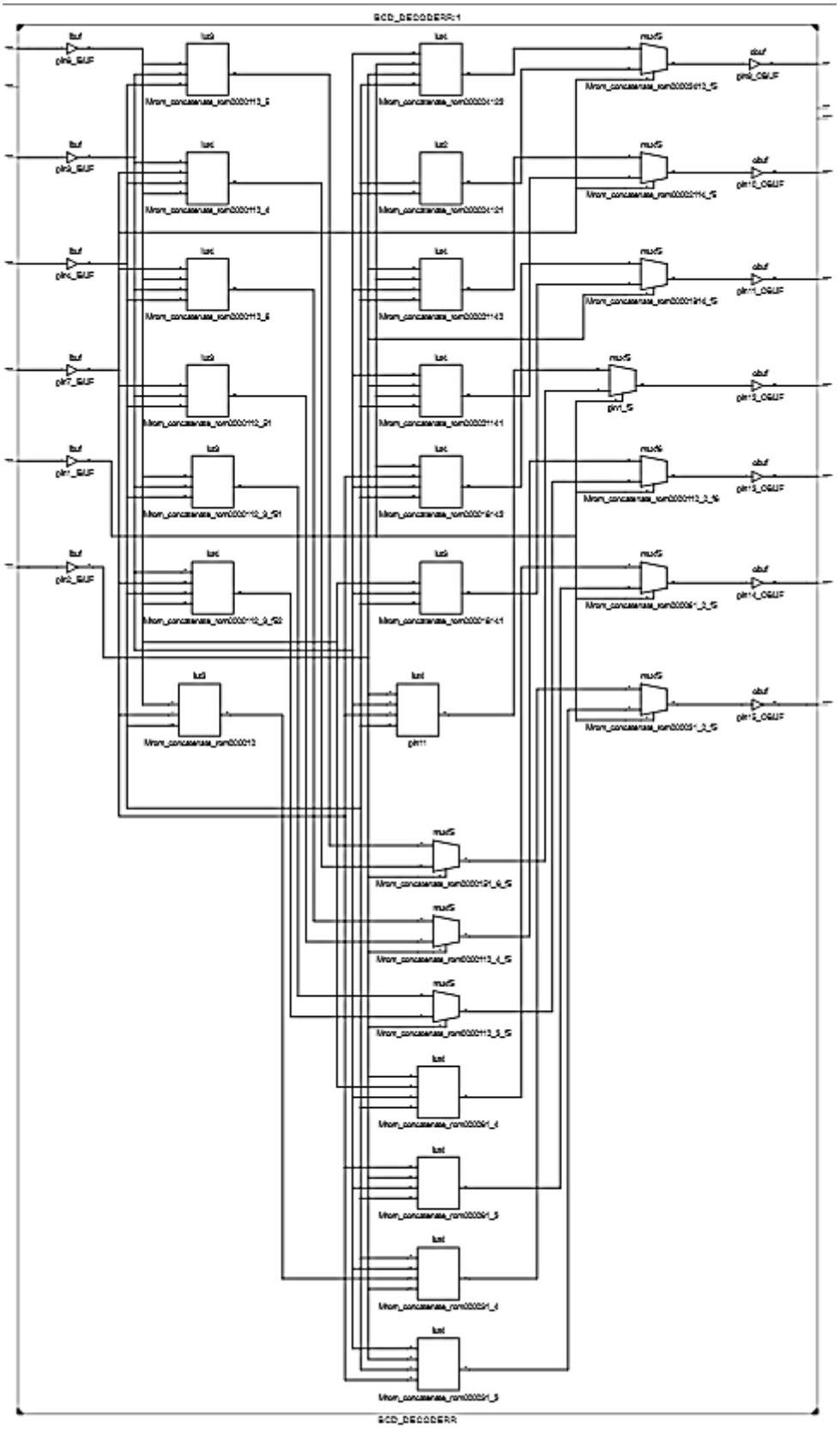

Figura H.3 – Esquema lógico obtido após sintetização da réplica (Conversor de BCD para 7 segmentos (teste: #5)).

H.3. Simulações efetuadas às réplicas obtidas

Nas Figuras H.4, H.5, H.6, H.7 e H.8 encontram-se apresentadas as simulações realizadas às réplicas obtidas nos testes aos circuitos combinatórios referidos na seção 5.2.2 deste documento, para confirmação do comportamento.

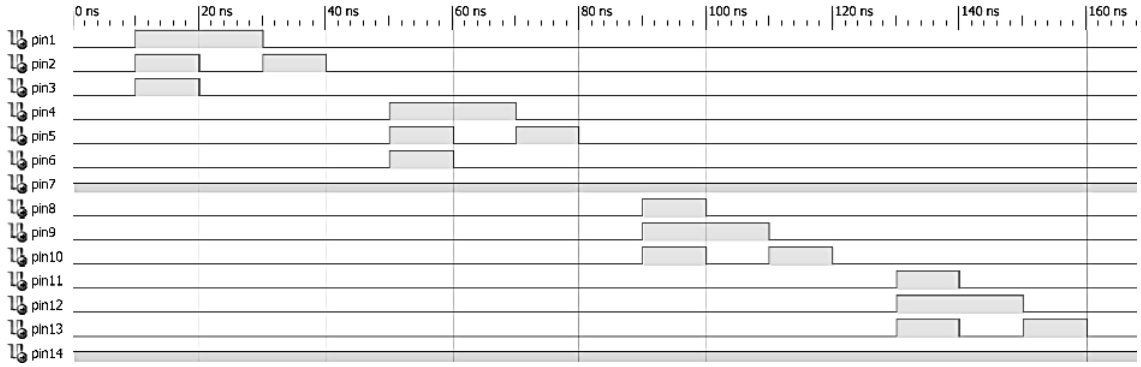

Figura H.4 – Simulação obtida para a réplica do circuito 74LS08 [62]: quatro portas lógicas do tipo *AND* (teste #1).

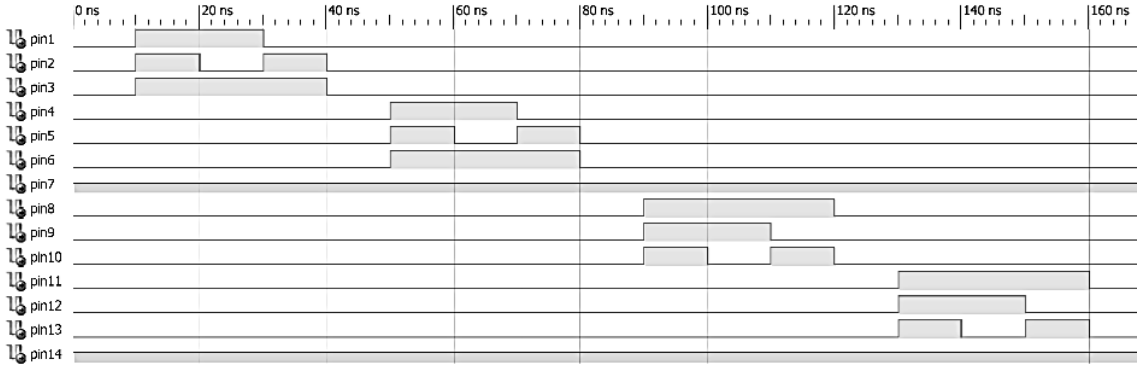

Figura H.5 – Simulação obtida para a réplica do circuito 74HCT32 [63]: quatro portas lógicas do tipo *OR* (teste #2).

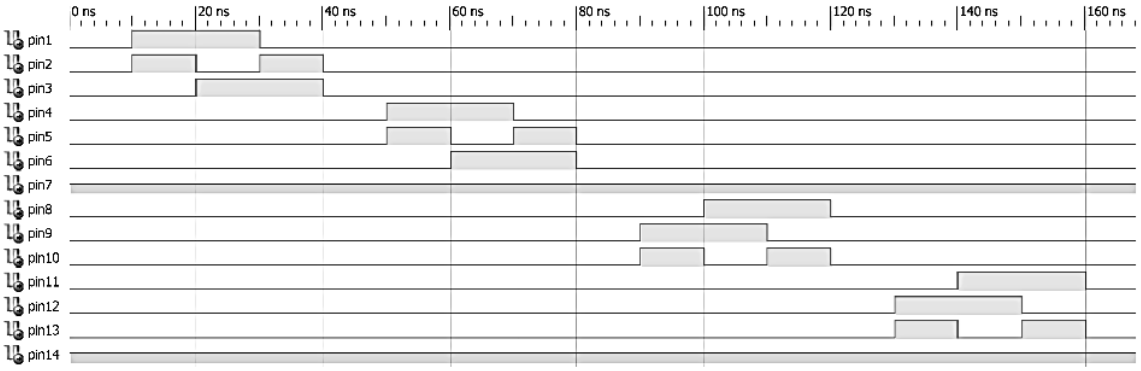

Figura H.6 – Simulação obtida para a réplica do circuito 74LS86 [64]: quatro portas lógicas do tipo *XOR* (teste #3).

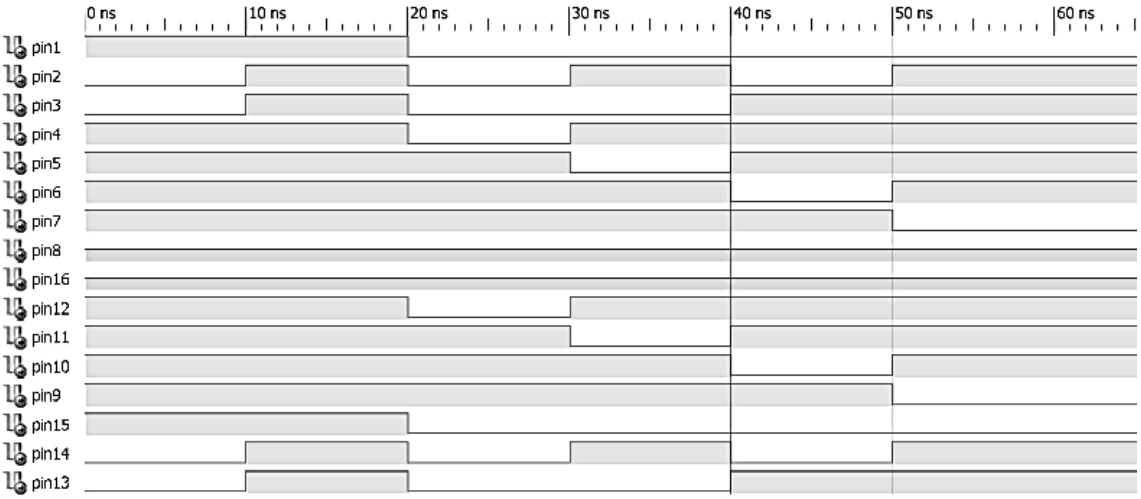

Figura H.7 – Simulação obtida para a réplica do circuito 74HC139 [65]: descodificador 2:4 duplo (teste #4).

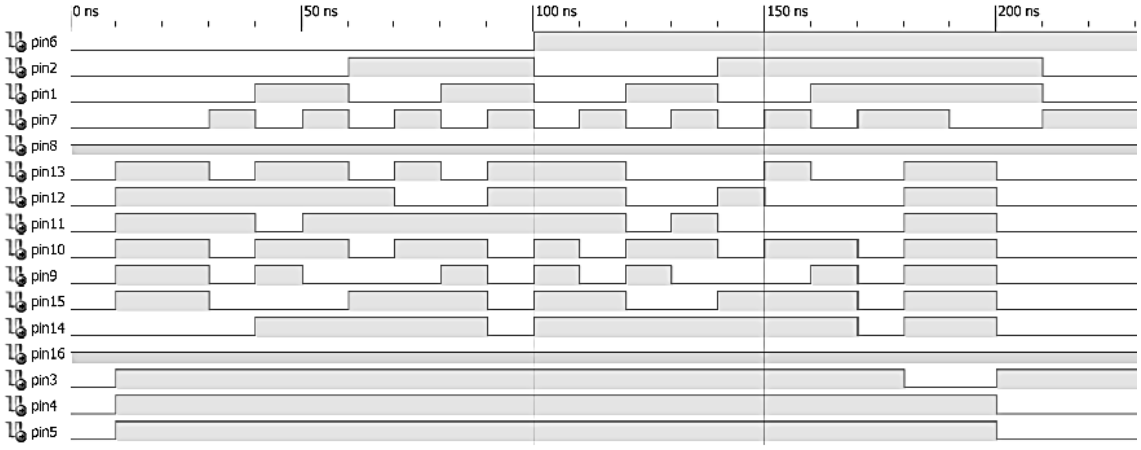

Figura H.8 – Simulação obtida para a réplica do circuito 74LS48 [66]: descodificador de BCD para sete segmentos (teste #5).

F.Resultados experimentais obtidos para circuitos sequenciais

I.1. Impacto da ocorrência de situação de *reset*

A ocorrência ou não de uma situação de *reset* durante o período de observação, afeta o código VHDL gerado para a réplica. Os códigos VHDL gerados para a réplica do circuito integrado 74LS393 com ocorrência e não ocorrência de *reset* encontram-se apresentados nas Listagens I.1 e I.2, respetivamente.

Listagem I.1 – Código VHDL gerado para réplica do circuito integrado 74LS393 (com ocorrência de *reset* durante a observação).

```
--Created by Circuit Replicator
--VHDL language replication file
--Author: Rodolfo Rodrigues
--Created on Mon Sep 15 14:59:08 2014

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity Replication2014_09_15 is
port(
        CLK:            in      STD_LOGIC;      --Clock signal
        RST_D_A :       in      STD_LOGIC;      --Reset: direct logic, assynchronus
        pin3:           out     STD_LOGIC;      --Digital port
        pin4:           out     STD_LOGIC;      --Digital port
        pin5:           out     STD_LOGIC;      --Digital port
        pin6:           out     STD_LOGIC;      --Digital port
        pin7:           inout   STD_LOGIC;      --No connection needed: GND
        pin8:           inout   STD_LOGIC;      --No connection needed: GND
        pin9:           inout   STD_LOGIC;      --No connection needed: GND
        pin10:          inout   STD_LOGIC;      --No connection needed: GND
        pin11:          inout   STD_LOGIC;      --No connection needed: GND
        pin12:          inout   STD_LOGIC;      --No connection needed: VCC
        pin13:          inout   STD_LOGIC;      --No connection needed: VCC
        pin14:          inout   STD_LOGIC       --No connection needed: VCC
);
 end entity;

architecture rtl of Replication2014_09_15 is

        --Structure with states
        type reply_states is (
                REPLY_STATE_0,
                REPLY_STATE_1,
                REPLY_STATE_2,
                REPLY_STATE_3,
                REPLY_STATE_4,
                REPLY_STATE_5,
                REPLY_STATE_6,
                REPLY_STATE_7,
```

```
                REPLY_STATE_8,
                REPLY_STATE_9,
                REPLY_STATE_10,
                REPLY_STATE_11,
                REPLY_STATE_12,
                REPLY_STATE_13,
                REPLY_STATE_14,
                REPLY_STATE_15
        );

        shared variable curr_reply_state: reply_states;
        shared variable next_reply_state: reply_states;

        signal reset: STD_LOGIC;                -- synchronus reset

begin
        process (CLK, RST_D_A) is
        --process to apply reset
        begin
                if(RST_D_A ='1') then
                        reset <= '1';
                else
                        reset <='0';
                end if;
        end process;


        process (CLK, reset) is
        --this process select next state
        begin
                if (reset = '1') then
                        curr_reply_state := REPLY_STATE_0;
                elsif(falling_edge(CLK)) then
                        curr_reply_state := next_reply_state;
                end if;
        end process;

        process (CLK) is
        begin   case curr_reply_state is
when REPLY_STATE_0 => pin3 <='0';pin4 <='0';pin5 <='0';pin6 <='0';next_reply_state := REPLY_STATE_0;
when REPLY_STATE_1 => pin3 <='1';pin4 <='0';pin5 <='0';pin6 <='0';next_reply_state := REPLY_STATE_2;
when REPLY_STATE_2 => pin3 <='0';pin4 <='1';pin5 <='0';pin6 <='0';next_reply_state := REPLY_STATE_3;
when REPLY_STATE_3 => pin3 <='1';pin4 <='1';pin5 <='0';pin6 <='0';next_reply_state := REPLY_STATE_4;
when REPLY_STATE_4 => pin3 <='0';pin4 <='0';pin5 <='1';pin6 <='0';next_reply_state := REPLY_STATE_5;
when REPLY_STATE_5 => pin3 <='1';pin4 <='0';pin5 <='1';pin6 <='0';next_reply_state := REPLY_STATE_6;
when REPLY_STATE_6 => pin3 <='0';pin4 <='1';pin5 <='1';pin6 <='0';next_reply_state := REPLY_STATE_7;
when REPLY_STATE_7 => pin3 <='1';pin4 <='1';pin5 <='1';pin6 <='0';next_reply_state := REPLY_STATE_8;
when REPLY_STATE_8 => pin3 <='0';pin4 <='0';pin5 <='0';pin6 <='1';next_reply_state := REPLY_STATE_9;
when REPLY_STATE_9 => pin3 <='1';pin4 <='0';pin5 <='0';pin6 <='1';next_reply_state := REPLY_STATE_10;
when REPLY_STATE_10 => pin3 <='0';pin4 <='1';pin5 <='0';pin6 <='1';next_reply_state := REPLY_STATE_11;
when REPLY_STATE_11 => pin3 <='1';pin4 <='1';pin5 <='0';pin6 <='1';next_reply_state := REPLY_STATE_12;
when REPLY_STATE_12 => pin3 <='0';pin4 <='0';pin5 <='1';pin6 <='1';next_reply_state := REPLY_STATE_13;
when REPLY_STATE_13 => pin3 <='1';pin4 <='0';pin5 <='1';pin6 <='1';next_reply_state := REPLY_STATE_14;
when REPLY_STATE_14 => pin3 <='0';pin4 <='1';pin5 <='1';pin6 <='1';next_reply_state := REPLY_STATE_15;
when REPLY_STATE_15 => pin3 <='1';pin4 <='1';pin5 <='1';pin6 <='1';next_reply_state := REPLY_STATE_0;
when others => next_reply_state := REPLY_STATE_0;
        end case;
        end process;

end rtl;
```

Listagem I.2 – Código VHDL gerado para réplica do circuito integrado 74LS393 (sem ocorrência de *reset* durante a observação).

```
--Created by Circuit Replicator
--VHDL language replication file
--Author: Rodolfo Rodrigues
--Created on Sat Sep 15 17:49:24 2014

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity counterNoRST is
port(
	CLK:		in	STD_LOGIC;	--Clock signal
	pin2:		inout	STD_LOGIC;	--No connection needed: GND
	pin3:		out	STD_LOGIC;	--Digital port
	pin4:		out	STD_LOGIC;	--Digital port
	pin5:		out	STD_LOGIC;	--Digital port
	pin6:		out	STD_LOGIC;	--Digital port
	pin7:		inout	STD_LOGIC;	--No connection needed: GND
	pin8:		inout	STD_LOGIC;	--No connection needed: GND
	pin9:		inout	STD_LOGIC;	--No connection needed: GND
	pin10:		inout	STD_LOGIC;	--No connection needed: GND
	pin11:		inout	STD_LOGIC;	--No connection needed: GND
	pin12:		inout	STD_LOGIC;	--No connection needed: VCC
	pin13:		inout	STD_LOGIC;	--No connection needed: VCC
	pin14:		inout	STD_LOGIC	--No connection needed: VCC
);
 end entity;

architecture rtl of counterNoRST is

	--Structure with states
	type reply_states is (
		REPLY_STATE_0,
		REPLY_STATE_1,
		REPLY_STATE_2,
		REPLY_STATE_3,
		REPLY_STATE_4,
		REPLY_STATE_5,
		REPLY_STATE_6,
		REPLY_STATE_7,
		REPLY_STATE_8,
		REPLY_STATE_9,
		REPLY_STATE_10,
		REPLY_STATE_11,
		REPLY_STATE_12,
		REPLY_STATE_13,
		REPLY_STATE_14,
		REPLY_STATE_15,
		REPLY_STATE_16
	);

	shared variable curr_reply_state: reply_states:=REPLY_STATE_1;
	shared variable next_reply_state: reply_states;
	signal inputs : STD_LOGIC;
	signal reset : STD_LOGIC :='0';
	begin
```

```
	process (CLK) is
	--this process select next state
	begin
		if (reset = '1') then
			curr_reply_state:= REPLY_STATE_0;
		elsif (falling_edge(CLK)) then
			curr_reply_state := next_reply_state;
		end if;
	end process;


	process (CLK) is
	begin

		case curr_reply_state is
when REPLY_STATE_0 => next_reply_state:=curr_reply_state;
when REPLY_STATE_1 => pin3 <='0'; pin4 <='0'; pin5 <='1'; pin6 <='0'; next_reply_state:=REPLY_STATE_2;
when REPLY_STATE_2 => pin3 <='1'; pin4 <='0'; pin5 <='1'; pin6 <='0'; next_reply_state:=REPLY_STATE_3;
when REPLY_STATE_3 => pin3 <='0'; pin4 <='1'; pin5 <='1'; pin6 <='0'; next_reply_state:=REPLY_STATE_4;
when REPLY_STATE_4 => pin3 <='1'; pin4 <='1'; pin5 <='1'; pin6 <='0'; next_reply_state:=REPLY_STATE_5;
when REPLY_STATE_5 => pin3 <='0'; pin4 <='0'; pin5 <='0'; pin6 <='1'; next_reply_state:=REPLY_STATE_6;
when REPLY_STATE_6 => pin3 <='1'; pin4 <='0'; pin5 <='0'; pin6 <='1'; next_reply_state:=REPLY_STATE_7;
when REPLY_STATE_7 => pin3 <='0'; pin4 <='1'; pin5 <='0'; pin6 <='1'; next_reply_state:=REPLY_STATE_8;
when REPLY_STATE_8 => pin3 <='1'; pin4 <='1'; pin5 <='0'; pin6 <='1'; next_reply_state:=REPLY_STATE_9;
when REPLY_STATE_9 => pin3 <='0'; pin4 <='0'; pin5 <='1'; pin6 <='1'; next_reply_state:=REPLY_STATE_10;
when REPLY_STATE_10 => pin3 <='1'; pin4 <='0'; pin5 <='1'; pin6 <='1'; next_reply_state:=REPLY_STATE_11;
when REPLY_STATE_11 => pin3 <='0'; pin4 <='1'; pin5 <='1'; pin6 <='1'; next_reply_state:=REPLY_STATE_12;
when REPLY_STATE_12 => pin3 <='1'; pin4 <='1'; pin5 <='1'; pin6 <='1'; next_reply_state:=REPLY_STATE_13;
when REPLY_STATE_13 => pin3 <='0'; pin4 <='0'; pin5 <='0'; pin6 <='0'; next_reply_state:=REPLY_STATE_14;
when REPLY_STATE_14 => pin3 <='1'; pin4 <='0'; pin5 <='0'; pin6 <='0'; next_reply_state:=REPLY_STATE_15;
when REPLY_STATE_15 => pin3 <='0'; pin4 <='1'; pin5 <='0'; pin6 <='0'; next_reply_state:=REPLY_STATE_16;
when REPLY_STATE_16 =>  pin3 <='1'; pin4 <='1'; pin5 <='0'; pin6 <='0'; next_reply_state:=REPLY_STATE_1;
when others => next_reply_state := REPLY_STATE_1;
		end case;
	end process;

end rtl;
```

O diagrama temporal da simulação realizada na situação de não ocorrência de *reset* apresentado na Figura I.1, permite verificar a existência de um estado inicial.

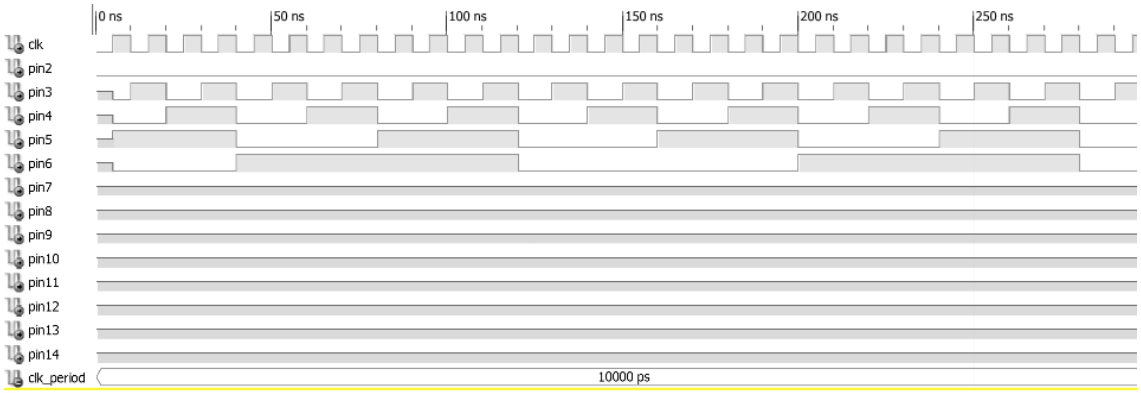

Figura I.1 – Diagrama temporal da simulação temporal realizada à réplica do circuito integrado 74LS393 sem ocorrência de *reset*.

I.2. Esquema lógico das réplicas obtidas

Como referido na secção 5.3.3.4 deste documento, não existe replicação estrutural das réplicas, quando comparados os esquemas lógicos obtidos com os esquemas lógicos definidos nas folhas de caraterísticas, como pode ser atestado nas Figuras I.2, I.3 e I4.

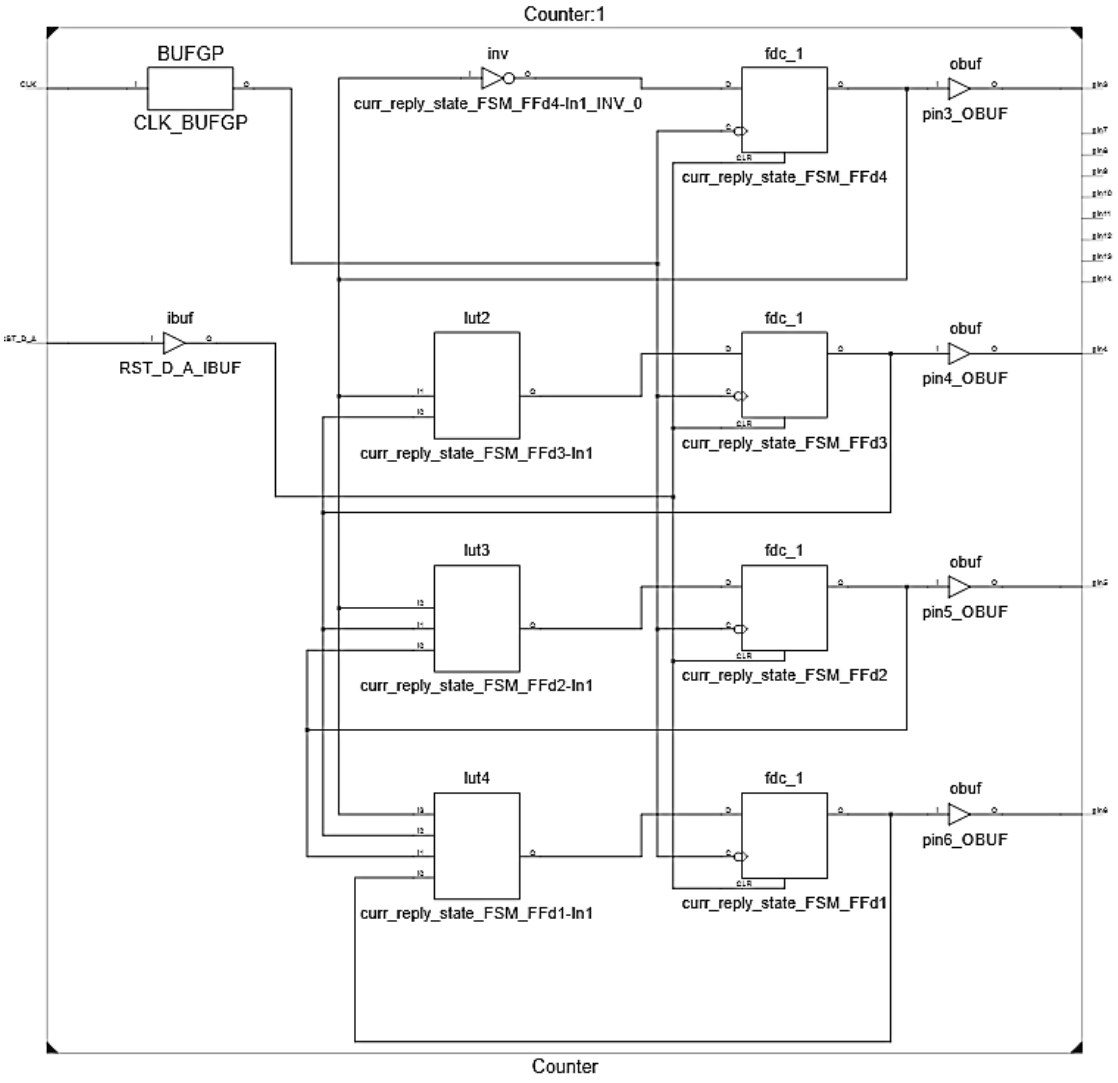

Figura I.2 – Esquema lógico obtido para a réplica do circuito integrado 74LS393 (contador) com ocorrência de *reset* durante a observação.

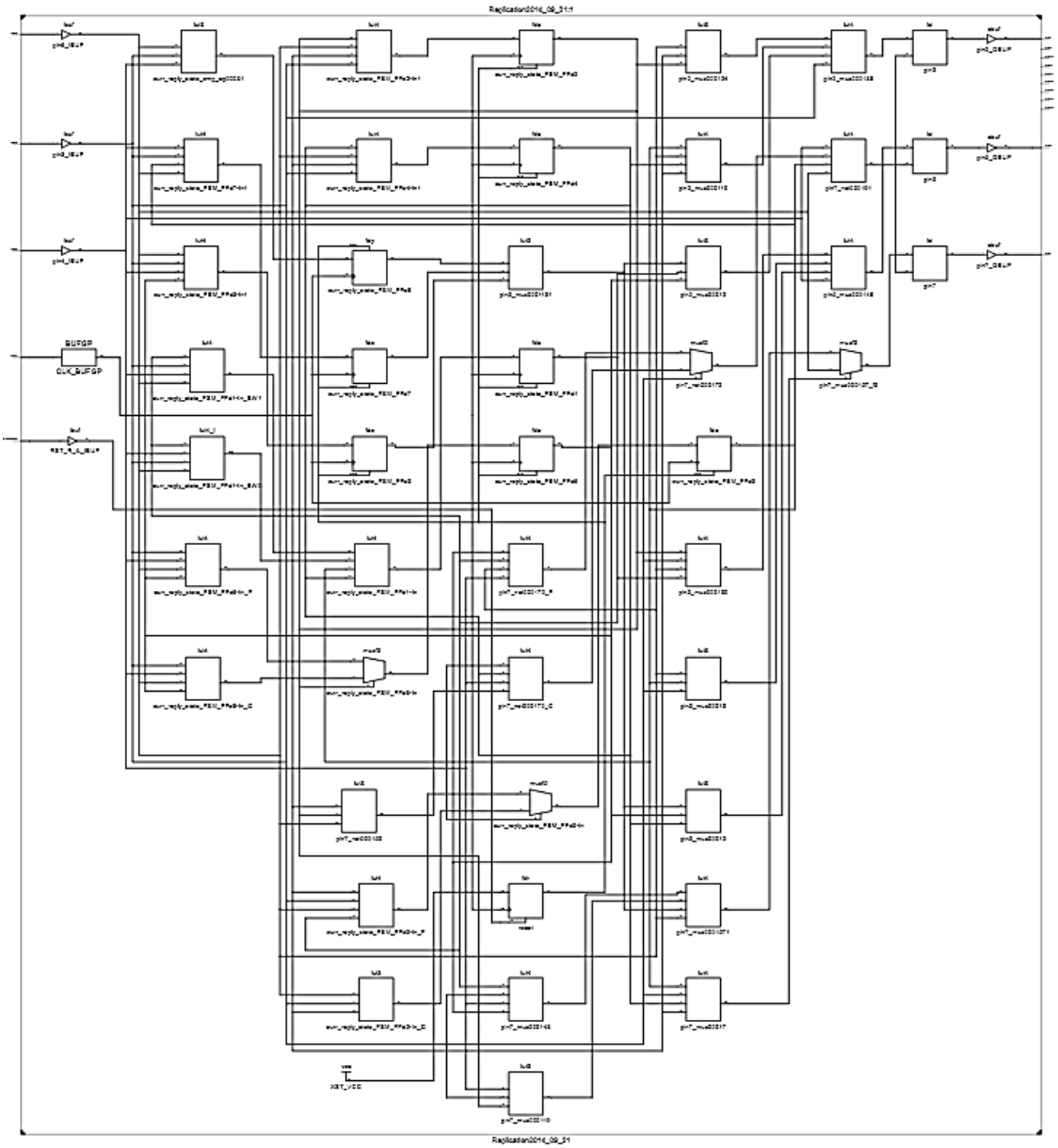

Figura I.3 – Esquema lógico obtido para a réplica do circuito integrado 74HCT174 (*flip-flop* tipo D, com montagem da Figura 5.8 b)) com ocorrência de *reset* durante a observação.

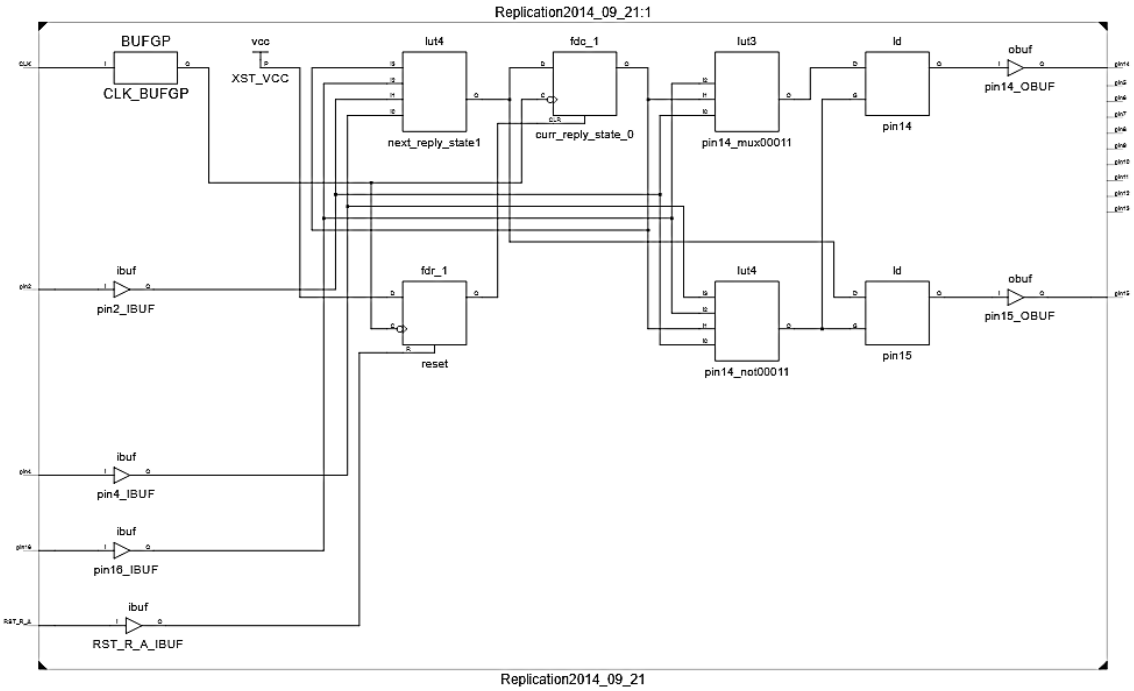

Figura F.4 – Esquema lógico obtido para a réplica do circuito integrado 74LS76 (*flip-flop* tipo JK, com montagem da Figura 5.8 c)) com ocorrência de *reset* durante a observação.

J. Artigo Científico aceite para publicação na 11th International Conference Applied Computing

REVERSE ENGINEERING OF DIGITAL INTEGRATED CIRCUITS BASED ON BEHAVIORAL ALGORITHMS

Rodolfo Rodrigues
CCCEE
University of Madeira
Funchal, Portugal
e-mail: a2051509@uma.pt

Dionísio Barros
CCCEE
University of Madeira
Funchal, Portugal
e-mail: dbarros@uma.pt

Elias Rodrigues
CCCEE
University of Madeira
Funchal, Portugal
e-mail: elias@uma.pt

ABSTRACT

This paper proposes a novel system to extract the behavior from digital integrated circuits (ICs), combinational and sequential, by observing its normal operation. An acquisition data system controlled by a computer send measurements of voltage values on the pins of the IC, that is subsequently analyzed on the computer by several algorithms to obtain its behavioral. When an input-output correspondence is found a combinatorial function is made, otherwise heuristically methods are applied to achieve a sequential circuit approach. Extracted behavior is automatically converted into VHDL language description, to synthetize the extracted schematic. Thus, the replica obtained by the system could be structurally improved, assuming the same behavior, so an equivalent circuit has been generated. Due to abstraction level, provided by behavioral description, the obtained replica has a lower dependence of hardware compared to structural approaches.

KEYWORDS

Reverse engineering, integrated circuits replication, non-invasive techniques, input-output correspondence, heuristic approach

INTRODUCTION

Nowadays, electronic circuits dependence is enormous and ICs are present into a large amount of devices, so the maintenance of electronic circuits is so important as its development. The maintenance of electronic circuits, which contains ICs, have two problems: (1) older circuits could have obsolete ICs which are no longer produced and technical information may be not available and (2) detect ICs with anomalies could be difficult with visual analysis because it appears perfect, so when a circuit is working badly, behavioral reverse engineering could be applied to all ICs contained on circuit to find the origin of the malfunctioning. Definitely, behavioral reverse engineering is a potential solution for circuits recovering, even with few information.

In the recent years reverse engineering has gained popularity in activities such as performance and security benchmarking, control quality certifications and support patent licensing (Torrance & James 2011). However, many techniques of reverse engineering still semi-manual (Raja 2008) focused on logical schematic extraction, from a disassembled chip and using image analysis algorithms (Quijada et al. 2014). This kind of processes are invasive and usually after reverse engineering analysis, the IC is no longer useable. The developing of non-invasive techniques of reverse engineering is currently a research area (Matlin et al. 2014). The increasingly complexity of microchips using a greater number of layers and logic gates makes this process unaffordable when using traditional methods that rely on human inspection and analysis (Quijada et al. 2014). Despite modern computers have greater processing capacity than a human being, fully algorithmic reverse engineering is a relatively new field of research (Subramanyan et al. 2014). Actually, mostly of reverse engineering techniques based on algorithms uses libraries with components information, and reverse engineering is made by matching the circuit under analysis against libraries component to find any correspondence (structural or functional) (Subramanyan et al. 2014). Many algorithms try to accomplish an approach to typical circuits, as counters, shift registers, and others (Subramanyan et al. 2014). Our purpose is develop a system capable of replicate digital integrated circuits by learning its behavior, using non-invasive, automatic and fully algorithmic techniques, without the support of any library. The used approach tries to create a generic learning method independently if circuits under test has a typical behavior or not. Usual applications of reverse engineering aim to discover the exact content of IC (mainly security and control applications), but the proposed system is aimed to create an equivalent replica, independently of structural similarity between original circuit and the obtained replica.

METHODOLGY

2.1 Developed system

The proposed system consists of a set of algorithms (hosted in a computer) and an acquisition circuit measurement (controlled by computer) capable of performing measurement of voltages on the pins of the IC and send the collected information to the computer. After the analysis processes the extracted behavior is described into a VHDL file, into a synthesizable structure as shown into Figure 1.

Figure 1. Illustrative diagram of proposed system.

In this paper, we will focus algorithms of behavior extraction. Generically, the analysis made by the algorithms consists into three steps: find constant signals and discard them, input-output correspondence analysis (combinational approach) and heuristic approach is applied to generate a customized state machine, which describes circuit behavior (when combinational approach failed). This sequence of procedures could be diagrammed by flowchart of Figure 2.

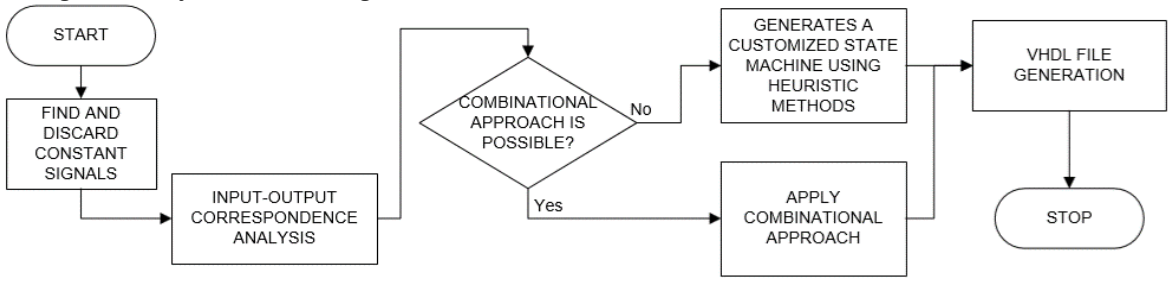

Figure 2. Flowchart of general analysis.

Sub-algorithms, such as communication with data acquisition system, are used to provide all data required by the main algorithms.

2.2 Algorithms

2.2.1 Detection of constant signals

Constant signals have no influence in IC behavior, or by others words, the influence is always the same during circuit normal operation and consequently could be discarded. This analysis allow us to find pins intended to supply the IC (designated as VCC and GND) and eventually other constant signals, whose discarding will simplify analysis. The measurements required by this algorithm are differential measurements between each combination of pins, in analog format. For each combination of pins, the differential voltage average can be obtained using the expression

$$\bar{V} = \frac{1}{M}\sum_{i=1}^{M} V_i \tag{1}$$

where $\bar{V}$ and M means voltage mean value and number of samples, respectively. Assuming only digital ICs are being analyzed, differential voltage average should be near to zero or VCC for two constant signals and VCC/2 when, at least, one of the signals is not constant. Two pins are classified as constants when all samples measured between those pins are included in range $[\bar{V} - \delta;\ \bar{V} + \delta]$, where δ represents the maximum tolerance allowed. This step provides reference pin (GND) for the next measurements.

2.2.2 Input-output correspondence and combinatorial approach

This step requires M measurements (binary format) of all non-constant pins related to the same instant, with random interval between samples to avoid wrong analysis of periodic signals. Input-output correspondence analysis consists into check if for each input combination a single output combination is observed (repeated combinations are discarded). When this condition is verified a combinatorial approach is possible, otherwise, heuristic methods are needed. Combinational behavior is converted to VHDL language using *case* statement where all input combinations observed and respective outputs combinations are listed inside *case*, using *when* statement. Using proper software (*ISE Design Suite* for example) VHDL code could be synthetized and an equivalent schematic (replica) will be generated. The percentage of extracted knowledge of IC (success rate) under analysis is estimated by algorithm according the expression

$$S(\%) = 100 \times C/2^N \tag{2}$$

where S, C and N means the success rate, number of input combinations observed and number of input pins, respectively.

2.2.3 Heuristic approach method to generate a customized state machine

The approximation of the behavior of the circuit into a customized finite state machine is based on analysis of the current and previous inputs/outputs, and is used when a sequential circuit is under analysis. Before heuristic methods execution, sub-algorithms to detect clock and reset signal are applied, due to the importance of those signals into analyses. Clock signal detection is based on detection of periodical signal with higher frequency and reset is detected when an input has always the same influence on output combination. Considering that the system only has access to the input/output combinations, the main problem of a sequential approach is "guess" the internal state of the IC. The purpose of heuristic methods is finding a good solution in a reasonable computational time, even if it is not the optimal solution (Cordenonsi 2008). The proposed approach assign a new state when an output combination is found by first time. Optimal state machines have the minimum number of possible states, then when a repeated output combination is detected, a transition to a previous state (candidate state) in order to get the best approach. However, a compatibility verification is needed to ensure that transition is compatible to previous occurrences. Compatibility verification consists into check N transitions after previous occurrences of the candidate state and do not find different transitions to the same states and input combination. The parameter N represents the depth of analysis, and is set by user. A successful example of a transition to a previous state is shown in Figure 3.

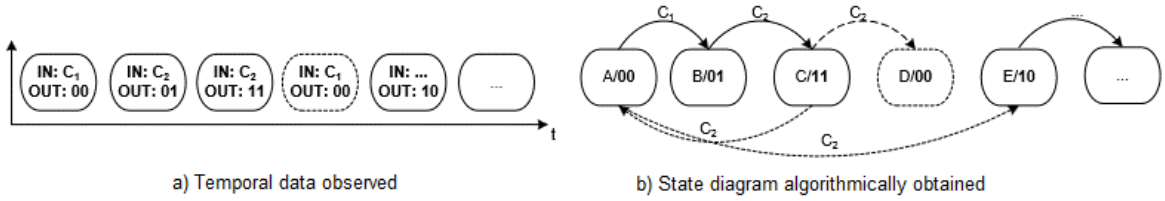

Figure 3. Successful transition to a previous state ($N = 1$).

When the transition to a state, previously detected, causes two different transitions for the same combination of state and input transitions a mismatch is found: Incorrect transition is reversed and a new attempt is made to perform a new transition to another state with the same output, already identified. If all previous states with the same output combinations are incompatible, a new state is generated. An unsuccessful example of a transition to a previous state is shown in Figure 4.

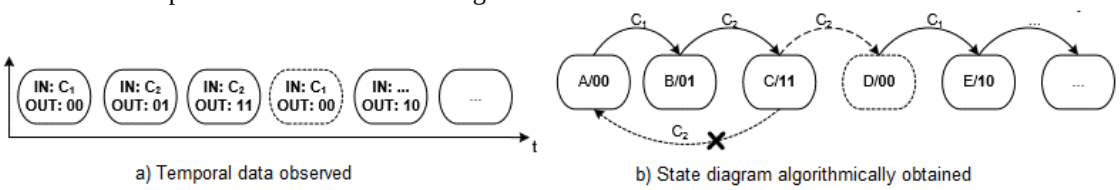

Figure 4. Unsuccessful transition to a previous state ($N = 1$).

From Figure 4 it appears that the attempt to make the transition from state C to state A is not correct because in the next transition there are two different transitions ("01" and "10") to the same input combination (C_1). Therefore, as there are no other states with the same output, a new state is created (state D). The accuracy of the approximation rises with the increasing depth analysis. The implementation of the algorithm is summarized in the flowchart of Figure 5.

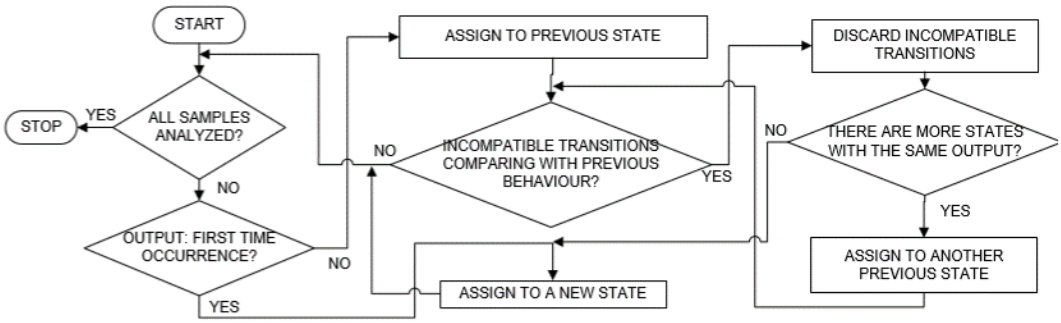

Figure 5. Flowchart of the heuristic method to generate a customized finite state machine.

The percentage of extracted knowledge of IC (success rate) under analysis is estimated by algorithm according the expression

$$S(\%) = 100 \times T_O / T_P \tag{3}$$

where S, T_O and T_P means the success rate, number of transitions observed and number of possible transitions, respectively.

EXPERIMENTAL RESULTS

Experimentally, it was observed that the success rate of the replica, the number of samples collected and the number of combinations / transitions are directly related. Table 1 shows the experimental results for different tested ICs (in case of sequential ICs, different mounting schemes affect the number of transitions).

Table 1. Results for replication of combinatorial and sequential circuits (M – Number of possible input combinations, T – Number of possible transitions, R – Samples collected).

Combinatorial					Sequential				
Function	**Reference**	**M**	**R**	**Success rate (%)**	**Function**	**Reference**	**T**	**R**	**Success rate (%)**
Decoder	74HC139	64	128	84	**Counter (#16)**	74LS393	16	2000	100
			192	94					
			320	100					
Decoder binary to BCD	74LS48	128	256	79	**Shift Register**	74HCT164	32		100
							512		67
			384	92	**Flip-flop D**	74LS174	64		64
							4096		16
			640	100	**Flip-flop JK**	74LS76	16		100
							128		86
AND gate	74LS08	256	512	81	**Shift Register**	74HC595	8192		13
			768	95					
			1280	100					

In Table 1 we can see that the higher the ratio between the number of samples collected and the number of combinations / possible transitions, higher success rates are achieved because more situations are observed. The implemented prototype allows a maximum of collecting 2000 samples.

CONCLUSION

This paper proposed a novel system to extract the behavior from digital ICs (combinational and sequential), using input-output correspondence and heuristic methods to achieve combinational and sequential circuits replica, after observing circuit during its operation. The main advantage of the proposed system is the level of abstraction provided by the VHDL code automatically generated, allowing the simulation and synthesis of the replica. Recovery circuits and possible technological upgrades (when applied to old circuits) are possible applications to the proposed system. Results showed the replication of some circuits, with success rates between 79% and 100% (combinatorial) or between 13% and 100% (sequential) depending on the observed situations. To improve the performance of the system, a new prototype with capacity to collect more samples must be developed (using the same algorithms), because more situations will be observed and consequently better success rates will be achieved.

REFERENCES

Cordenonsi, A.Z., 2008. *Ambientes, objetos e dialogicidade: uma estratégia de ensino superior em heurísticas e metaheurísticas*. Universidade Federal do Rio Grande do Sul.

Matlin, E., Agrawal, M. & Stoker, D., 2014. Non-Invasive Recognition of Poorly Resolved Integrated Circuit Elements. *IEEE Transactions on Information Forensics and Security*, 9(3), pp.354–363.

Quijada, R. et al., 2014. The Use of Digital Image Processing for IC Reverse Engineering. In *11th International Multi-Conference on Systems, Signals & Devices (SSD)*. Barcelona, Spain, pp. 1–4.

Raja, V., 2008. *Reverse engineering : an industrial perspective* First Edit. V. Raja & K. J. Fernandes, eds., London: Springer.

Subramanyan, P. et al., 2014. Reverse Engineering Digital Circuits Using Structural and Functional Analyses. *IEEE Transactions on Emerging Topics in Computing*, Vol. 2, No. 1, pp.63–80.

Torrance, R. & James, D., 2011. The state-of-the-art in semiconductor reverse engineering. In *48th ACM/EDAC/IEEE Design Automation Conference (DAC)*. San Diego, USA, pp. 333–338.