Henrique Gonçalves Guerra

PROCESSAMENTO BINAURAL EM TEMPO REAL PARA LOCALIZAÇÃO

Dissertação de Mestrado

5 de Setembro de 2011

· U C ·

UNIVERSIDADE DE COIMBRA

Universidade de Coimbra
FACULDADE DE CIÊNCIAS E TECNOLOGIA
Departamento de Engenharia Electrotécnica e de Computadores

Mestrado Integrado em Engenharia Electrotécnica e de Computadores
2010-2011

Dissertação de Mestrado

PROCESSAMENTO BINAURAL EM TEMPO REAL PARA LOCALIZAÇÃO

Henrique Gonçalves Guerra

Júri:
Presidente: Prof. Doutor Jorge Manuel Miranda Dias
Orientador: Prof. Doutor Jorge Nuno de Almeida e Sousa Almada Lobo
Co-Orientador: Prof. Doutor João Filipe de Castro Cardoso Ferreira
Vogal: Prof. Doutor Lino José Forte Marques

Coimbra, 5 de Setembro de 2011

Abstract

This work presents two hybrid implementations – with a FPGA (Field Programming Gate Array) or a CPU (Central Processing Unit), and a GPU (Graphic Processing Unit) – of an algorithm that detects interaural cues for spatial localization of sound sources in natural auditory scenes (i.e. in the presence of reverberation and background noise). Sound waves arising from a source on our left will arrive at the left ear first, provoking a small, but perceptible, difference in arrival time (known as an ITD, interaural time difference). Similarly, for intensity, the far ear lies in the head's "sound shadow", giving rise to interaural level differences (ILDs). ITDs and ILDs thus constitute binaural cues, which are important cues used by the human brain for sound source localization in space. Since ITDs are essentially frequency independent, they only provide information regarding azimuth localization; however, there is a tangible dependence of ILDs on frequency, caused by the different effects of the head's sound shadow on each frequency band for different elevations and distances (the latter only noticeable within a range between 1 and 2 meters). Therefore, binaural cues can be used for full localization sound sources in 3D. Binaural cue detection algorithms usually involve a robust correlation detector applied in parallel to several frequency bands of a pre-filtered stereo signal. This often leads to demanding computational resources. However, the parallel trait of the localization process can be developed by using SIMD (Single Instruction Multiple Data) technologies. In the last decade, GPUs and reconfigurable hardware such as FPGAs have evolved significantly in terms of processing power. GPUs, in particular, have been used for more generic applications, where graphical hardware is used beyond graphical manipulation, using GPGPU (General-purpose computing on Graphics Processing Units); more recently, GPU computing has been developed (i.e. a parallel programming environment that is no longer based on the graphics processing pipeline), bringing forth tools such as NVIDIA's CUDA (Compute Unified Device Architecture). In this work, the robust model of Faller and Merimaa for binaural cue detection has been implemented using a FPGA Altera DE2 Cyclone II and a NVIDIA 9800GTX+ GPU with 128 CUDA cores. Although functional, the foreseen FPGA processing, were also done in a CPU, due to problems with the speed and latency of the communications with the FPGA.
Results show that the implementation robustly detects binaural cues in real-time (with a satisfactory tradeoff between latency and processing rate), and shows an improvement of 86x in processing time when comparing to the processing cycle of the original MATLAB implementation. The implementation presented herewith is ready for use as a low-level module in localization algorithms, including extensive documentation for replication, usage and customization.

- Key-Words: FPGA; GPU; parallelization; localization; real-time

Resumo

Neste trabalho apresenta-se duas implementações híbridas – com uma FPGA (Field Programming Gate Array) ou CPU, e GPU (Graphic Processing Unit) – de um algoritmo que detecta sinais interaurais para localização espacial de fontes sonoras em cenários acústicos naturais (i.e. na presença de reverberância e ruído de fundo). As ondas sonoras provenientes de uma fonte à nossa esquerda irão chegar primeiro ao ouvido esquerdo, provocando uma pequena, mas perceptível, diferença de chegada (conhecido como ITD, interaural time difference). Da mesma forma, para a intensidade, o ouvido mais distante ficará na "sombra sonora" da cabeça, provocando um aumento de diferenças de intensidade (ILD, interaural level difference). Os ITDs e ILDs constituem assim os sinais binaurais, sinais esses importantes para o cérebro humano proceder à localização da fonte sonora no espaço. Como os ITDs são essencialmente independentes da frequência, apenas fornecem a informação referente à localização em azimute; no entanto, existe a dependência tangível da frequência para os ILDs, devido aos diferentes efeitos da sombra sonora da cabeça em cada banda de frequência para diferentes elevações e distâncias (este último apenas perceptível num intervalo de 1 a 2 metros. Por consequência, os sinais binaurais podem ser utilizados para localizar de forma eficiente a fonte sonora em 3D. Os algoritmos de sinais binaurais envolvem normalmente um poderoso detector de correlação aplicado paralelamente em varias bandas de frequência de um sinal estéreo pré-filtrado, necessitando por isso de recursos computacionais exigentes. No entanto, a característica paralela do processo de localização pode ser implementada usando tecnologias SIMD (Single Instruction Multiple Data). Na última década, GPUs e o hardware reconfigurável como FPGAs evoluíram significativamente em termos de poder de processamento. GPUs, em particular, têm sido desenvolvidos para aplicações mais genéricas, onde o hardware gráfico é usado além do processamento gráfico, usando GPGPU(General-purpose computing on Graphics

Processing Units); mais recentemente foi desenvolvida a computação em GPU (i.e. o ambiente de programação paralela deixou de ser baseado em pipeline gráfico), originando ferramentas como o CUDA da NVIDIA (Compute Unified Device Architecture). Neste trabalho, o modelo de Faller and Merimaa de detecção de sinais binaurais é implementado usando uma FPGA Altera DE2 Cyclone II e uma NVIDIA 9800GTX+ GPU com 128 núcleos CUDA.

Embora funcional, o processamento previsto para FPGA acabou por ser feito numa CPU, devido a problemas com a velocidade e a latência da comunicação com a FPGA.

Os resultados da implementação CPU-GPU demonstram que os sinais binaurais são detectadas com eficácia (com um equilíbrio satisfatório entre a latência e taxa de processamento), e revela melhorias de 86x no tempo de processamento, quando comparado com o ciclo de processamento na implementação em MATLAB. A implementação aqui apresentada possui características para ser usada como um módulo de baixo nível em algoritmos de localização, incluindo uma extensa documentação para replicação, uso e personalização.

- Palavras-chave: FPGA; GPU; paralelização; localização; tempo-real

Agradecimentos

Ao Prof. Doutor Jorge Nuno de Almeida e Sousa Almada Lobo e o Prof. Doutor João Filipe de Castro Cardoso Ferreira , orientadores da dissertação, agradeço o apoio, incentivo, a partilha do saber e as valiosas contribuições para o trabalho.

Sou muito grato a todos os meus familiares pelo incentivo recebido ao longo destes anos.

O meu profundo e sentido agradecimento a todos os meus amigos que contribuíram para a concretização desta dissertação, estimulando-me intelectual e emocionalmente.

Conteúdo

Lista de Figuras

Lista de Tabelas

LISTA DE TABELAS

Abreviaturas

- API — Application Programming Interface
- ASIC — Application-Specific Integrated Circuit
- CPU — Central Processing Unit
- CUBLAS — Compute Unified Basic Linear Algebra Subprograms
- CUDA — Compute Unified Device Architecture
- CUFFT — Compute Unified Fast Fourier Transform
- DC — Direct Current
- DDR — Double-Data-Rate
- DMA — Direct Memory Access
- DSP — Digital Signal Processor
- ERB — Equivalent Rectangular Bandwidth
- FFT — Fast Fourier Transform
- FIR — Finite Impulse Response
- FPGA — Field-Programmable Gate Array
- GPGPU — General-Purpose Computing on Graphics Processing Units
- GPU — Graphics Processing Unit
- HDL — Hardware Description Language

- IC — Interaural Coherence
- IFFT — Inverse Fast Fourier Transform
- IHC — Inner Hair Cells
- IIR — Infinite Impulse Response
- ILD — Interaural Level Difference
- IMPEP — Integrated Multimodal Perception Experimental Platform
- IORD — I/O Read strobe pin
- IOWR — I/O Write strobe pin
- IP — Intellectual Property
- ITD — Interaural Time Difference
- JTAG UART — Joint Test Action Group Universal Asynchronous Receiver/-Transmitter
- NRE — Non-Recurring Engineering
- OSI — Open Systems Interconnection
- PDF — Probability Density Function
- POP — Perception on Purpose
- RAM — Random-Access Memory
- RS-232 — Recommended Standard 232
- RTL — Register Transfer Level
- SBT — Software Build Tools
- SDRAM — Synchronous Dynamic Random-Access Memory
- SIMD — Single Instruction Multiple Data
- SIMT — Single Instruction Multiple Threads

- SOPC — System on a Programmable Chip Builder
- SRAM — Static Random Access Memory
- UDP — User Datagram Protocol
- USB — Universal Serial Bus
- TCP/IP — Transmission Control Protocol Internet Protocol
- VHDL — Very high speed integrated circuits Hardware Description Language

1

Introdução

1.1 Motivação

As ondas sonoras resultantes de uma fonte sonora localizada à nossa esquerda chegam primeiro ao nosso ouvido esquerdo, e só depois ao nosso ouvido direito. Esta pequena diferença, mas perceptível, no tempo de chegada do som (denominada de diferença temporal interaural – ITD) é usada pelo nosso cérebro, que funciona como uma matriz de detectores de correlação depois dos sinais auditivos terem sido pré-processados pelo ouvido interno, como uma pista importante em termos de localização espacial da fonte sonora. De forma similar, para a intensidade sonora, o ouvido mais distante encontra-se no lado oposto da cabeça que actua como um atenuador dependente da frequência (a que se dá o nome de "sombra auditiva"), provocando diferenças de intensidade binaurais (ILDs) (King et. al) (20) e (Kapralos et. al) (19), demonstrado na figura 1.1.

Os ITDs variam sistematicamente com o ângulo de incidência da onda sonora relativamente ao eixo interaural e, em teoria, são independentes da frequência, representando a pista de localização mais importante para baixas frequências (abaixo dos 1500 Hz em humanos). As ILDs são mais complexas que os ITDs no sentido em que variam significativamente com a frequência sonora. Sons de baixa frequência viajam facilmente em volta da cabeça, causando ILDs quase imperceptíveis. Em compensação, valores de ILD produzidos por sons de frequência mais elevada são bastante mais significativos, e são crescentemente influenciados pelas propriedades de filtragem do ouvido externo, que impõe picos e vales no espectro sonoro que chega ao tímpano.

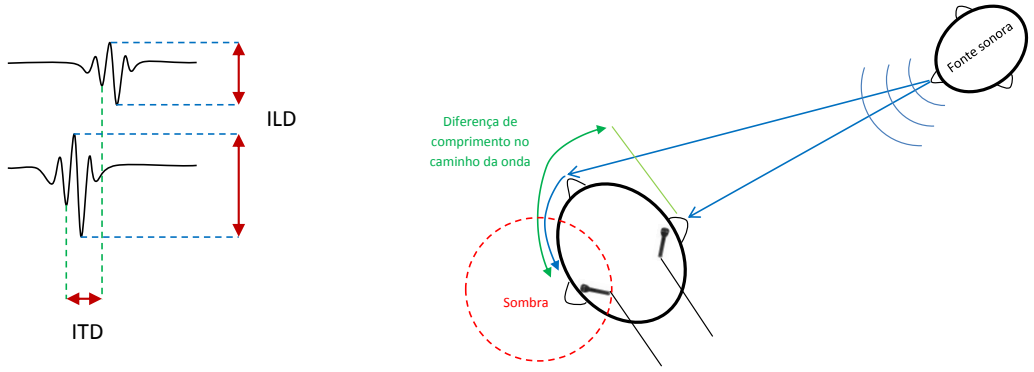

Figura 1.1: Origem do ITD e do ILD. Os tempos de chegada de uma onda sonora de uma fonte não são exactamente iguais no tímpano esquerdo e direito, devido ao comprimento dos caminhos até aos ouvidos. Esta diferença temporal de chegada entre ambos os ouvidos é denominada de ITD. O ITD máximo é medido quando a onda de som chega ao longo do eixo que cruza os tímpanos. Neste caso, o ITD pode ser estimado conforme a distância dos ouvidos, $\approx 18cm$, dividindo por a velocidade do som, $\approx 340m/s$, originado o valor $529\mu s$. A existência da cabeça entre os dois ouvidos causa a atenuação(sombra) da onda sonora do tímpano contra-lateral, dando origem ao ILD.

Adicionalmente, quando considerando fontes sonoras a uma distância de 1 a 2 metros do ouvinte, as pistas binaurais são suficientes por si só para localizar de forma completa uma fonte sonora no espaço tridimensional (isto é, em termos de azimute, elevação e distância) (Shinn-Cunningham et. al) (36), exceptuando quando se trata de uma fonte localizada no plano medial (azimute igual a zero) – que apresentará ITDs e ILDs nulas, qualquer que seja a sua distância ou elevação – e descontando o efeito omnipresente a que se dá o nome de "confusão frente-trás" (por exemplo, ignorando que sons possam ser resultado de fontes atrás do ouvinte, ou desfazendo a confusão através do movimento dos ouvidos relativamente à fonte).

Assim como para os humanos e uma boa parte dos representantes do reino animal, também para os robôs o ambiente sonoro circundante pode oferecer informação extremamente pertinente, ainda para mais porque a audição é um processo perceptual panorâmico, ao contrário, por exemplo, dos sistemas visuais mais comuns, além de que pode oferecer uma resolução temporal substancial e é menos susceptível de sofrer efeitos de oclusão. Localizar fontes sonoras no espaço pode, portanto, oferecer enormes

vantagens nas mais diversas aplicações robóticas.

Em 2004, Christof Faller e Juha Merimaa (13) formularam um modelo de detecção de ITDs e ILDs para localização espacial de fontes auditivas. Este modelo permite a detecção robusta destas pistas binaurais na presença de ruído, de várias fontes sonoras e em cenários ecoantes. O algoritmo, portanto, amplifica a razão sinal-ruído e facilita a análise de cenas auditivas, nomeadamente no seguimento de múltiplos objectos sonoros.

O modelo acima descrito possui potencial para que seja aplicado o princípio de "instrução única para múltiplos dados " (SIMD – Single Instruction Multiple Data) com vista a aumentar o desempenho através da programação paralela, utilizando para isso um dispositivo denominado de FPGA (Field-Programmable Gate Array – lógica programável) e uma GPU (Graphics Processing Units – Unidades de Processamento Gráfico).

As FPGAs existem desde 1983 e têm evoluído em desempenho e eficiência, sendo utilizadas na implementação de exigentes algoritmos através do seu enorme conjunto de portas programáveis, podendo estes algoritmos ser programados em vários percursos paralelos.

As GPUs foram originalmente concebidas para processamento gráfico, mas hoje em dia são utilizadas para qualquer tipo de aplicação (GPGPU – General Purpose GPU programming),ou seja, para realizar cálculos tradicionalmente tratados em CPU, utilizando estágios programáveis e aritmética de precisão paralelamente.

1.2 Objectivos

Esta dissertação tem como objectivo a implementação em tempo real do algoritmo de detecção de pistas binaurais para localização proposta por Faller e Merimaa (13) explorando várias vertentes de paralelização, com o objectivo imediato de servir de suporte do sistema apresentado na figura 1.2, mas com a perspectiva de ser usado e/ou adaptado para ser utilizado por qualquer sistema de localização auditiva. Com esse intuito, esta dissertação representa um documento suficientemente completo para reprodução, uso e adaptação desta implementação.

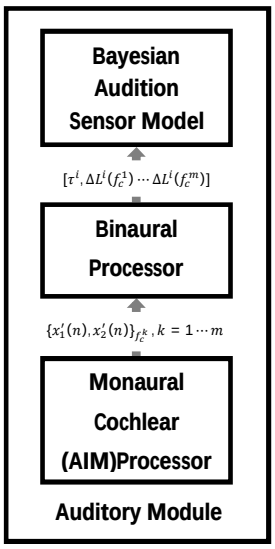

Figura 1.2: Sistema binaural Bayesiano do IMPEP. Adaptado de Ferreira et. al (9)

1.3 Estado da arte

No último século, tem vindo a ser realizada a pesquisa para perceber como as pistas acústicas são realizadas no sistema auditivo de forma a localizar e separar sons concorrentes. O termo audição binaural é referente ao modo de funcionamento do sistema auditivo humano ou animal usando dois ouvidos. Os órgãos auditivos servem como pré-processadores e condicionadores de sinal e separam as pistas acústicas ajudando o cérebro a resolver tarefas relacionadas com a localização auditiva, detecção e reconhecimento. Uma das características do sistema auditivo humano é, por exemplo, a capacidade de reagir a sinais de alerta vindos de uma direcção fora do alcance da visão.

O interesse na localização espacial binaural com precisão cresceu rapidamente nos últimos anos, principalmente devido à necessidade crescente de soluções realistas relacionadas com áudio e acústica, isto é, síntese de som 3D, tecnologia de próteses auditivas, vigilância e navegação baseadas em acústica. Têm sido propostos vários modelos, a maioria deles baseados em vectores de microfones, em muitas situações necessitando de um poder de processamento exaustivo. No entanto, existem poucas soluções de localização binaural em que apenas dois microfones identificam a posição tridimensional da fonte

sonora, satisfazendo a localização em tempo real de ambientes acusticamente adversos.

Para o problema de localização da posição espacial da fonte sonora, existem diversos modelos propostos como o de Handzel e Krishnaprasad (14). Os modelos matemáticos de propagação de ondas sonoras revelaram uma dependência significativa de características específicas das fontes e do ambiente, sendo os mesmos altamente complexos e difíceis de optimizar de acordo com o modelo de vectores de microfones descrito em Duraiswami et. al (33).

O ouvido humano é um pré-processador de sinal que estimula o sistema nervoso central, dando origem a um excelente processador de sinal que consiste em mecânica, acústica, hidroacústica e componentes eléctricos que, em conjunto, dão origem a um sensível receptor e analisador de espectro de alta-resolução. Da perspectiva de processamento de sinal, os princípios físicos constituintes de um sistema preciso, como o do ouvido humano, é pretendido na forma de algoritmo. Desta forma, existem métodos que realizam uma mímica dos mecanismos de localização de fontes sonoras baseados na biologia humana, construindo modelos do ouvido externo, médio e interno, usando para isso conhecimentos de como os eventos acústicos são traduzidos e transformados pela biologia do sistema auditivo (Grassi e Shamma, (15)).

1.3.1 Localização sonora com dois microfones

Em contraste aos ITDs, os ILDs são fortemente afectados pela frequência da onda de chegada, tornando-os altamente dependentes da frequência. Num intervalo de baixa-frequência, a cabeça humana é menor que o comprimento da onda de chegada e, desta forma, a difracção tem um impacto reduzido. Considerando um intervalo de alta-frequência o comprimento de onda é menor quando comparado com as dimensões da cabeça e nesta gama de frequências os ILDs não são apenas determinados pela forma da cabeça, mas também por influência da forma do ouvido externo.

1.4 Contribuição

O sistema implementado possibilita a adaptação a diferentes ambientes, através da alteração das 16 bandas de frequência, configuração do máximo do ITD e do máximo do ILD, a alteração da frequência de amostragem e o limiar de energia combinada dos dois microfones.

Esta implementação pretende ser integrada num sistema de maior dimensão, como por exemplo o robô IMPEP, desenvolvido no Instituto de Sistemas e Robótica, de forma a diminuir a sua dependência da CPU e consequentemente aumentando o desempenho e diminuindo o consumo energético.

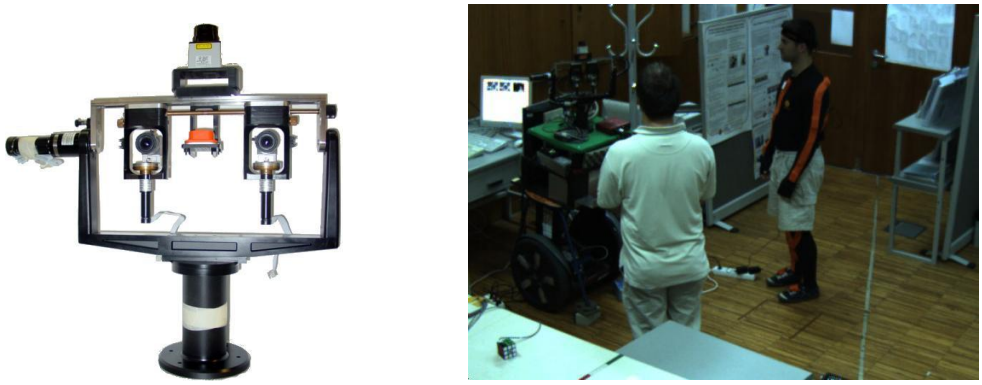

Figura 1.3: Exemplo de aplicação em robôs sociais. O sistema robótico IMPEP (Integrated Multimodal Perception Experimental Platform, à esquerda) desenvolvido no Instituto de Sistemas e Robótica no âmbito do projecto europeu POP (Perception on Purpose; projecto FP6-IST-2004-027268), observa dois interlocutores num contexto de interacção homem-robô (à direita) (18)

Alternativamente, usando um sistema de segurança ou de vigilância, dirigindo o eixo da câmara automaticamente na direcção da fonte de som detectada (uma pessoa a falar alto, ou a gritar), semelhante à reacção dos humanos quando ouvem uma fonte sonora de alta energia, toda a sua atenção é orientada na direcção do acontecimento de forma instintiva, representado nas figuras 1.4, 1.5 e 1.6.

Figura 1.4: Exemplo de Situação 1. Uma discussão entre duas ou várias pessoas sucede em redor da câmara de vigilância e como esta não possui dois microfones, desconhece a ocorrência da situação.

Figura 1.5: Exemplo de Situação 2. A mesma câmara da situação anterior, mas equipada com um sistema binaural, detecta a ocorrência.

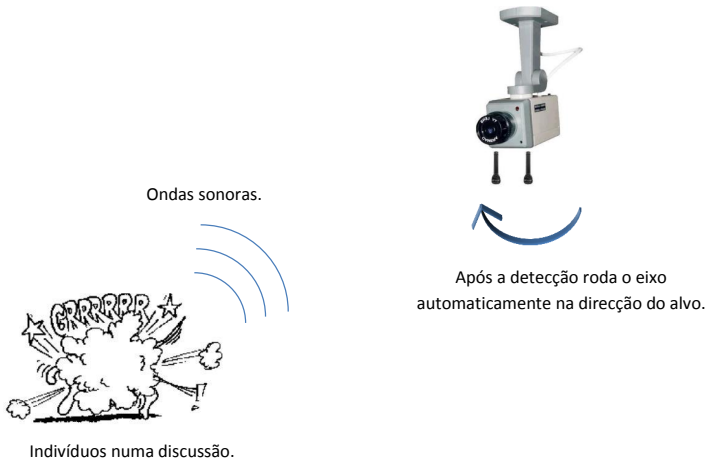

Figura 1.6: Exemplo de Situação 2(continuação). Após a detecção, a câmara roda o eixo na direcção da fonte sonora e assim pode emitir um aviso ao segurança que se encontra mais próximo do local.

1.5 Estrutura da dissertação

Na primeira secção do capítulo 2 é apresentado o modelo de indicações binaurais proposto por Faller e Merimaa (13). Na secção seguinte é apresentada a estrutura conceptual da FPGA e as ferramentas existentes de apoio à programação deste dispositivo. Ainda no segundo capítulo é demonstrada a linguagem de programação CUDA que permite programar unidades de processamento gráfico da NVIDIA.

No capítulo 3 é explicado como foi efectuada a implementação do modelo de indicações binaurais em modo híbrido FPGA e GPU, sendo mostrado o fluxo de dados geral e detalhado das duas plataformas, evidenciando como foi aplicada a paralelização dos cálculos.

No capítulo 4 são apresentados os resultados temporais e experimentais das implementações híbridas em FPGA-GPU e CPU-GPU, demonstrando o desempenho e a eficácia do algoritmo.

Por fim no capítulo 5 apresentam-se as conclusões e propõe-se trabalho futuro.

2

Teoria de suporte

2.1 Modelo de indicações binaurais

Os humanos desenvolveram a captação de ondas sonoras há muito tempo atrás. Esta percepção é uma habilidade necessária não só para comunicação verbal e prazer musical, mas também para analisar em termos espaciais o cenário circundante. Nesta secção explica-se o modelo proposto por Faller and Merimaa (13), onde, através de equações matemáticas, se obtém uma aproximação ao sistema auditivo humano, utilizando como refêrencia o ouvido externo, intermédio e interno, sendo representada na figura 2.1.

2.1.1 Periferia auditiva

A principal função do sistema auditivo humano é converter variações das ondas de pressão sonoras propagáveis tanto pelo ar como pela água ou outros materiais em impulsos neuronais, fornecendo esta informação ao córtex cerebral.

No ouvido interno existe a cóclea que é constituída, entre outros, pela membrana basilar. Esta vibra em diferentes pontos para frequências características distintas. O movimento oscilatório demonstrado na figura 2.2 é transmitido pelas células ciliadas (*Inner Hair Cells IHC*) e é convertido em actividade neuronal.

O banco de filtros gammatone foi proposto originalmente por Patterson (35) como modelo de análise da frequência existente na cóclea. É um modelo relativamente popular por duas razões: fornece uma boa correspondência fisiológica e é computacionalmente eficiente.

No domínio do tempo este banco de filtro é representado pela seguinte equação:

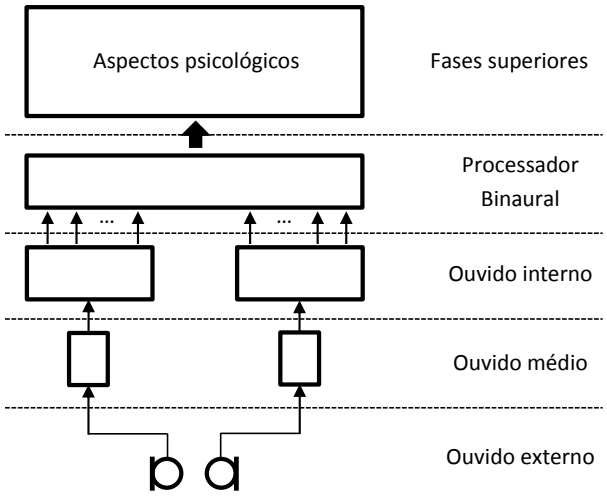

Figura 2.1: Modelo de audição espacial. São considerados os aspectos físicos, fisiológicos e psicológicos do sistema auditivo, começando pelo ouvido externo, seguindo-se para o ouvido médio e interno, saída multifrequência do processador binaural e finalmente a implementação de níveis superiores (adaptado a partir (13)).

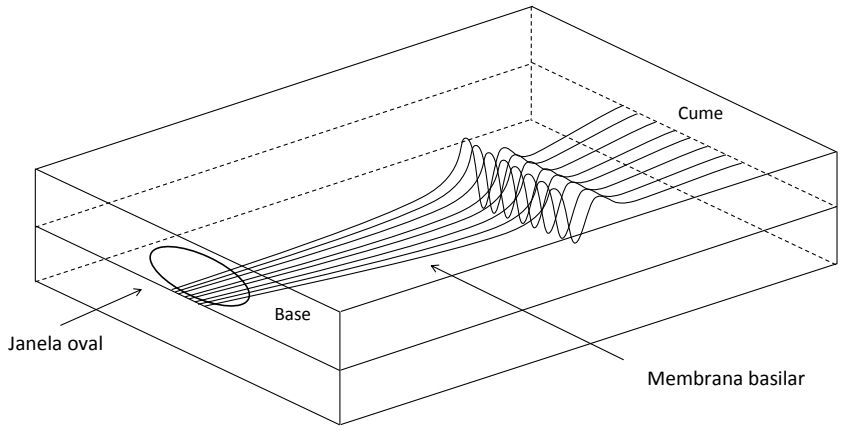

Figura 2.2: Onda sinusoidal a viajar através de um breve modelo de caixa de uma cóclea desenrolada. (adaptado a partir (11, Lyon and Mead))

$$gt(t) = t^{N-1}e^{-2\pi bt}\cos(2\pi f_0 t + \delta) \tag{2.1}$$

Sendo N de ordem 2, b a largura de banda escolhida de acordo com a largura de banda rectangular equivalente (ERB) dos filtros auditivos humanos correspondentes. Os valores são medidos de forma experimental com uma correspondência semelhante ao sistema auditivo humano, de acordo com Glasberg e Moore (10) (23, Moore) :

$$ERB(f) = 24.7 + 0.108f \tag{2.2}$$

Sendo a largura de banda calculada através:

$$b(f) = 1.019ERB(f) \tag{2.3}$$

Inicialmente são capturados dois blocos de som originados por dois microfones, sendo estes blocos processados separadamente de forma a emular o funcionamento da cóclea de cada ouvido. Este processo é seguido aproximadamente utilizando um banco de filtros ERB e a simulação da transformação neural.

Como descrito anteriormente, o movimento da membrana basilar é induzido nos IHCs que convertem o movimento em actividade neural (30, Pickles).

Este modelo tem como entrada o banco de filtros gammatone convertendo em probabilidade instantânea flutuante de um evento de pico pós-sináptico em uma fibra do nervo auditivo (34) apresentado na figura 2.3. A taxa a que o neuro-transmissor é excitado está relacionada com o deslocamento instantâneo da membrana basilar (banco de filtros gammatone ou estímulo acústico). A quantidade de transmissão numa fissura determina a probabilidade instantânea de uma ocorrência de pico. Assim a taxa de libertação assemelha-se a uma saída de meia onda rectificada dos filtros auditivos.

Passos importantes na tradução neuronal (ilustrados na figura 2.4):

- A cada saída do banco de filtros gammatone, comprime-se em envelope através da transformada de Hilbert aumentando a sua energia exponencialmente por 0.23 seguido do seu quadrado.

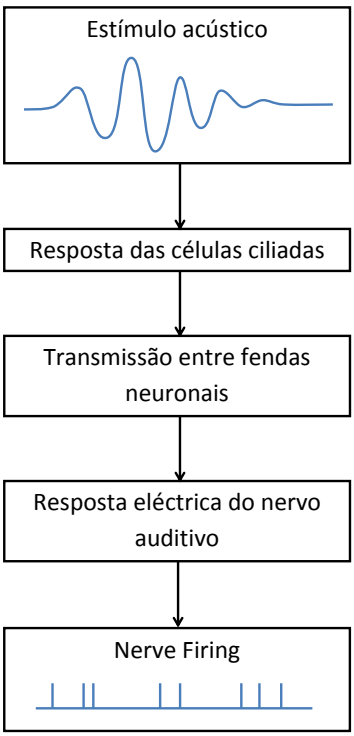

Figura 2.3: Resposta à estimulação do nervo auditivo acústico. (adaptado a partir (32, Meddis))

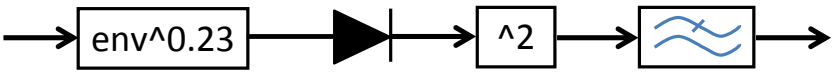

Figura 2.4: Modelo funcional dos elementos fisiológicos neuronais que foi proposto por Bernstein and Trahiotis (12), sendo constituído por uma compressão em envelope, uma rectificação de meia onda quadrática, seguindo-se uma filtragem passa-baixo de ordem 4 com uma frequência de corte igual a 425Hz.

- Este envelope é modificado através de uma rectificação de meia onda e recombinado com a saída do banco de filtros gammatone.
- É efectuada uma filtragem de resposta a impulso finita (FIR), passa-baixo de quarta ordem, com frequência de corte de 425Hz ao envelope resultante .

2.1.2 Processador binaural

O processador binaural é identificado pela a diferença temporal interaural (ITD), diferença de amplitude interaural (ILD), a coerência interaural (IC) e a energia combinada dos dois microfones.

A cada instância temporal quando o IC e a energia são superiores a um certo limiar distinto o ITD e o ILD são considerados.

O ITD e IC são estimado através da função de correlação cruzada normalizada:

$$\gamma(n,m) = \frac{a_{12}(n,m)}{\sqrt{a_{11}(n,m)a_{22}(n,m)}} \tag{2.4}$$

onde

$$a_{12}(n,m) = \alpha x_1(n - maxm, 0)x_2(n - max\{-m, 0\}) + (1-\alpha)a_{12}(n-1,m), \tag{2.5}$$

$$a_{11}(n,m) = \alpha x_1(n - maxm, 0)x_1(n - max\{m, 0\}) + (1-\alpha)a_{11}(n-1,m), \tag{2.6}$$

$$a_{22}(n,m) = \alpha x_2(n - max{-}m, 0)x_2(n - max\{-m, 0\}) + (1-\alpha)a_{22}(n-1,m). \tag{2.7}$$

x_1 e x_2 são os valores de saída da tradução neuronal do ouvido esquerdo e do direito, n é o índex de cada amostra do sinal e m o atraso (em amostras) α é uma constante temporal que determina o valor de decadência entre amostras do sinal, ou seja, à medida que são percorridas, o peso das amostras anteriores vai diminuindo. O α é determinado por:

$$T = \frac{1}{\alpha f_s} \tag{2.8}$$

f_s é a frequência de amostragem do sinal de entrada. $\gamma(n,m)$ é avaliado em janelas temporais de $\frac{m}{f_s}$ normalmente entre $[-1,1]ms$. O ITD em amostras é determinado com o atraso de m através do cálculo do máximo da função de correlação cruzada normalizada:

$$\tau(n) = \arg\max_m \gamma(n,m) \tag{2.9}$$

sendo o ITD limitado pelo tamanho da janela temporal (intervalo de amostragem).

O IC é estimado através do valor máximo instantâneo da função de correlação cruzada normalizada:

$$c_{12}(n) = \max_m \gamma(n,m) \tag{2.10}$$

O IC fornece a coerência entre os sinais provenientes do lado esquerdo e do direito. Tem variação de $[0,1]$ devido a x_1 e x_2 serem provenientes de uma rectificação de meia onda em que no caso de 1 os sinais são perfeitamente coerentes. Por outro lado devido a um *offset* DC existente na tradução neuronal, os valores de IC são maiores que zero mesmo para x_1 e x_2 independentes.

O ILD é calculado da seguinte forma:

$$\Delta L(n) = 10\log_{10}(\frac{L_2(n,\tau(n))}{L_1(n,\tau(n)}) \tag{2.11}$$

onde

$$L_1(n,\tau(n)) = \alpha x_1^2(n - max\{m,0\}) + (1-\alpha)L_1(n-1,m), \tag{2.12}$$

$$L_2(n,\tau(n)) = \alpha x_2^2(n - max\{m,0\}) + (1-\alpha)L_2(n-1,m). \tag{2.13}$$

Devido à compressão em envelope existente na tradução neuronal, os valores resultantes do ILD serão menores que as diferenças de amplitude existentes entre os sinais esquerdo e direito capturados pelos microfones.

Para sinais coerentes o ILD estimado é 0.46 vezes o sinal físico.

A energia do sinal é calculada através da soma da energia de x_1 e x_2, sendo utilizada para estimar o ITD de índex n:

$$p(n) = L_1(n,\tau(n)) + L_2(n,\tau(n)). \tag{2.14}$$

2.1.3 Selecção de amostras

A selecção de amostras ITD e ILD é feita da seguinte forma:

$$\{\Delta L(n), \tau(n) \mid c_{12}(n) > c_0\}. \tag{2.15}$$

O IC é próximo de 1 (sinais perfeitamente coerentes) quando a primeira onda da fonte sonora é capturada. Num ambiente reverberante onde são capturados ecos, estes vão produzir ITDs e ILDs de direcções diferentes da fonte sonora, fazendo diminuir o IC para valores menores que 1. Deste modo o IC é um bom indicador de ITDs e ILDs provenientes de ecos, podendo estes ser eliminados através do limiar c_0.

As funções de densidade de probabilidade (PDF) são estimadas a partir do cálculo dos histogramas do ITD e dos ILDs.

As PDFs mostram a probabilidade do ITD e dos ILDs, sendo possível identificar onde se encontram as fontes sonoras espaciais.

2.2 FPGA

Hoje em dia é comum muitas aplicações desenvolvidas em software serem convertidas em aceleradores de hardware. Para isto, podem ser usados Circuitos Integrados para Aplicações Específicas como ASICs *(Application-Specific Integrated Circuit)* ou sistemas reconfiguráveis baseados em FPGAs *(Field Programmable Gate Array)*.

As FPGA's são dispositivos que oferecem a potencialidade de dispositivos ASIC's, tais como:

- redução de tamanho, peso e consumo de energia.
- alto desempenho.
- maior segurança contra cópias não autorizadas.
- baixo custo.

Além disto, em comparação com as ASIC's, as FPGA's revelam menor tempo de prototipagem, reprogramação do circuito, menor custo NRE (Non-recurring engineering), resultando em projectos mais económicos para soluções que precisam de poucas

quantidades de circuitos. Comparando com DSPs (Digital Signal Processing) e processadores de propósito geral, as implementações em FPGAs podem explorar mais amplamente o paralelismo interno dos algoritmos (22).

2.2.1 Visão geral do dispositivo

A FPGA Altera Cyclone II é composta por uma matriz de blocos de elementos lógicos. Cada um realiza pequenas funções e operações lógicas, para isso são programados, configurados e interligados de acordo com a função lógica que se deseja empenhar. Na figura 2.5 pode-se visualizar a estrutura de uma FPGA.

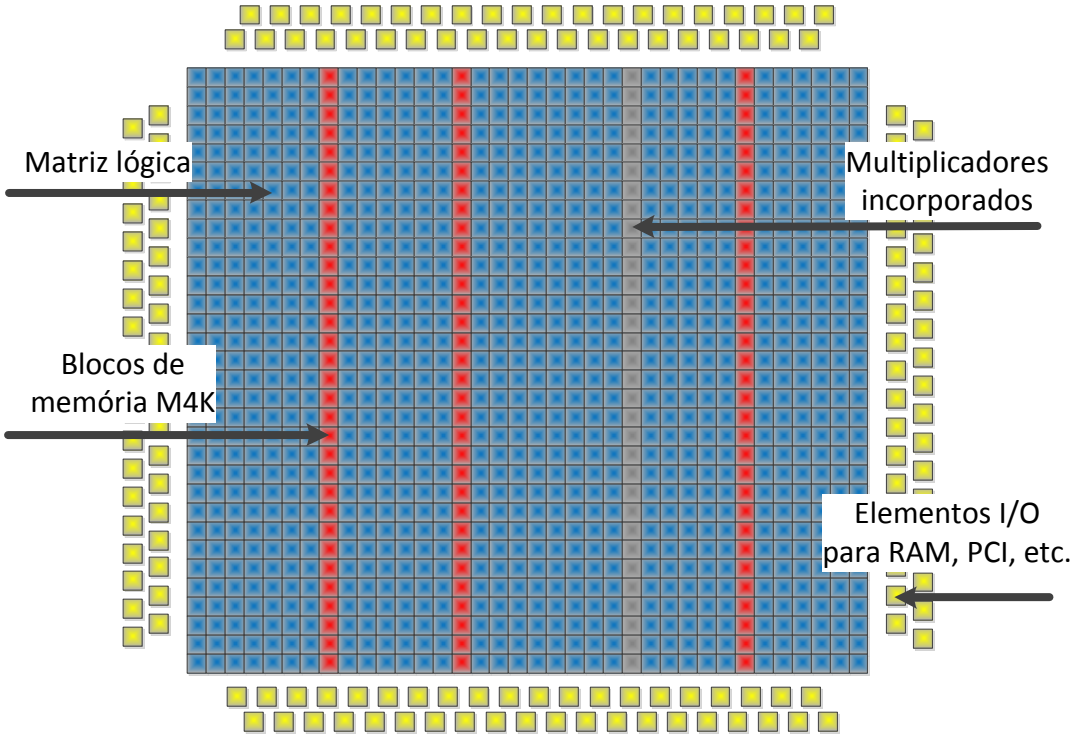

Figura 2.5: Estrutura conceptual de uma FPGA Altera Cyclone II. Contém uma lógica personalizável de duas dimensões, sendo que estes elementos conectam e fornecem sinais de interconexão à matriz de blocos de elementos lógicos, aos blocos de memória embebidos e aos multiplicadores incorporados. Este dispositivo fornece uma rede global de relógio e até 4 Malhas de Captura de Fase (PLLs), permitindo gerir o relógio do dispositivo. Os multiplicadores incorporados são organizados em colunas ao longo do dispositivo.

2.2.2 Quartus II

Na elaboração de projectos em FPGA existem três principais etapas: a descrição, síntese e posicionamento e interligação, sendo as etapas realizadas com o software Quartus II (5). Na etapa de descrição, são escritas as especificações do projecto em linguagem de hardware como VHDL *(Very high speed integrated circuits Hardware Description Language)* ou Verilog. O código gerado na prototipagem é compilado e simulado e posteriormente interpretado por a ferramenta de síntese, sendo transcrito o código num circuito equivalente utilizando elementos genéricos como registos, portas lógicas, somas ou comparações. Concluída a síntese o circuito está num nível RTL (*Register Transfer Level*). Na sequência, já na etapa de posicionamento e interligação, é definido quais os dispositivos que são utilizados na FPGA, e em detalhe, define-se a localização e posicionamento dentro da FPGA (*place*) e como são realizadas as interligações (*route*).

Quartus II (5) é a base de todo o projecto em FPGA, onde se pode visualizar todo o processo esquemático, efectuar ligações entre diferentes primitivas ou módulos prontos para uso (*MegaFunction*) (2).

2.2.3 SOPC Builder

O SOPC (7) é uma interface intuitiva com a finalidade de gerar sistemas integrados num único dispositivo, permite parametrizar, seleccionar componentes e definir as suas ligações. Inclui vários componentes prontos para uso, um deles, e essencial, é o processador Nios II (4), classificado como um *soft-core processor* (implementado através de síntese lógica). Outros componentes existentes são a interface para memória (SDRAM, DDR, DDR2), ethernet, RS-232 e outro periféricos. Além disso, é possível adicionar periféricos personalizados e para isso é criado o HDL em quartus e adicionado no SOPC.

O processador Nios II e outras interfaces necessárias para ligar circuitos na DE2 são implementados no circuito da FPGA Cyclone II. Estes componentes são interconectados por uma rede de conecções múltiplas. Os blocos de memória interna do dispositivo Cyclone II podem ser usados para disponibilizar memória ao processador Nios II. A SRAM, SDRAM e memória Flash são acedidas por interfaces apropriadas. A interface JTAG UART é usada para providenciar uma ligação com o computador anfitrião através do USB (Universal Serial Bus). Outro módulo, chamado JTAG Debug module, permite o computador anfitrião controlar o sistema Nios II, sendo possível carregar programas

na memória, iniciar e parar a execução, definir pontos de interrupção e recolha de dados em tempo real. Como todos os constituintes do processador Nios II são implementados no circuito da FPGA usando uma linguagem de descrição de hardware (HDL), é possível implementar código suplementar para desenvolver uma componente externa ao SOPC. Na figura 2.6 é ilustrada uma FPGA que inclui um sistema SOPC Builder e módulos de lógica personalizados.

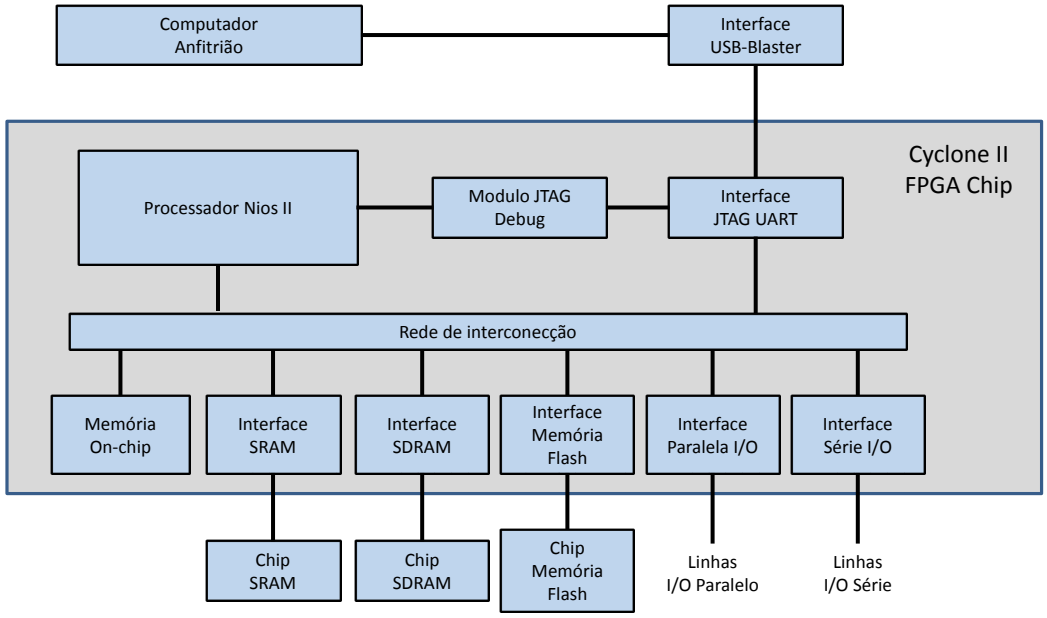

Figura 2.6: Exemplo de uma FPGA com um sistema gerado em SOPC Builder. Através da rede de interconecção (Avalon-MM master) é feita a comunicação entre o sistema SOPC Builder e os outros módulos. A SRAM, SDRAM e memória Flash são acedidas por interfaces apropriadas.A interface JTAG UART é usada para providenciar uma ligação com o computador anfitrião através do USB.

Para programar o processador Nios II é utilizada uma ferramenta do Quartus II, o NIOS II SBT para Eclipse (*Software Build Tools*), sendo uma plataforma de desenvolvimento de software para o NIOS II.

O Nios II SBT para Eclipse é uma GUI (*Graphical User Interface*) baseada na estrutura do Eclipse (16) C/C++, com esta plataforma é possível realizar todas as tarefas de desenvolvimento do software Nios II, incluindo criar, editar, compilar, executar e depurar programas, podendo ainda interagir com módulos personalizados em hardware, através da função de escrita IOWR (I/O de escrita) e de leitura IORD (I/O de leitura). É disponibilizado no Anexo B a implementação de controlo dos Leds a partir do software Nios II.

2.3 CUDA

Na última década as GPUs têm sido desenvolvidas de forma a permitir utilizações mais genéricas, onde o hardware gráfico é usado para cálculos além daqueles de natureza estritamente gráfica em que uma das ferramentas para GPGPU (*General-Purpose Computing on Graphics Processing Units*) é o CUDA(Compute Unified Device Architecture).

O CUDA é uma plataforma de software para computação paralela de alto desempenho que utiliza o poder de processamento das unidades de processamento gráfico (GPUs) da NVIDIA. As GPUs têm avançado rapidamente, passando de unidades de processamento de uma função específica para serem dispositivos altamente programáveis e muito poderosos em computação paralela. Para tirar partido do poder de computação paralela destes dispositivos a NVIDIA introduziu em 2007 a linguagem CUDA, onde é possível executar múltiplas operações de uma instrução a múltiplos dados (SIMD).

A ferramenta CUDA tem ganho utilizadores nos campos científico, biomédico, da computação, da análise de risco e da engenharia devido às características presentes nestes campos, as quais são altamente paralelizáveis.

2.3.1 Visão geral do dispositivo

Na terminologia CUDA a CPU é chamada anfitriã (*host*) e a GPU é chamada dispositivo (*device*).

A razão da discrepância na capacidade de cálculo entre uma CPU e uma GPU é que as GPUs são especializadas em computação altamente paralela e mais transístores são dedicados ao processamento em vez de cache de dados e controlo de fluxo, como

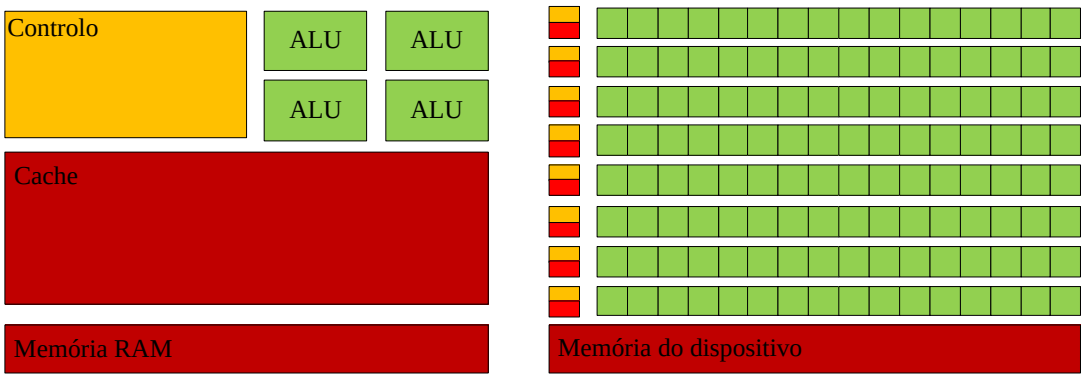

Figura 2.7: Comparação entre a aplicação dos transístores em CPUs e em GPUs. Nas GPUs mais transístores são dedicados para o processamento de dados, sendo estas especializadas na computação paralela. Nas CPUs existem transístores dedicados ao controlo de fluxo e à cache de dados. Adaptado de (NVIDIA *programming guide* (28))

representado na figura 2.7. As aplicações gráficas são extremamente paralelas, sendo a mesma operação aplicada a múltiplos dados e a latência de acesso à memória pode ser ocultada através dos cálculos em vez do uso de grandes caches de dados.

Uma GPU NVIDIA com a capacidade de executar código compilado em CUDA cria, gere, agenda e executa *threads* automaticamente em grupos de 32 *threads* paralelas com zero *overhead* de agendamento, sendo este conjunto de 32 *threads* o chamado *warp*. Quando um multiprocessador recebe um bloco com mais de 32 *threads* para processar, o bloco é quebrado em *warps* e as *threads* são organizadas de acordo com o seu número de identificação. Com esta arquitectura a NVIDIA introduziu um novo conceito na computação paralela: *single-instruction multiple thread* (SIMT).

De acordo com a figura 2.8 uma GPU da NVIDIA consiste num array de multiprocessadores de threads.

Cada multiprocessador consiste em 8 processadores escalares com 16384 registos de 32 bits de uso exclusivo, tendo cada um 2048 registos, cada um possui uma memória partilhada (*shared memory*) com 16kB de tamanho e acesso extremamente rápido e é acessível apenas por *threads* de um determinado bloco. Além da memória partilhada, todos os multiprocessadores têm acesso a três espaços de memória comuns:

- Memória de constantes (*constant memory*): Memória de acesso rápido apenas para leitura

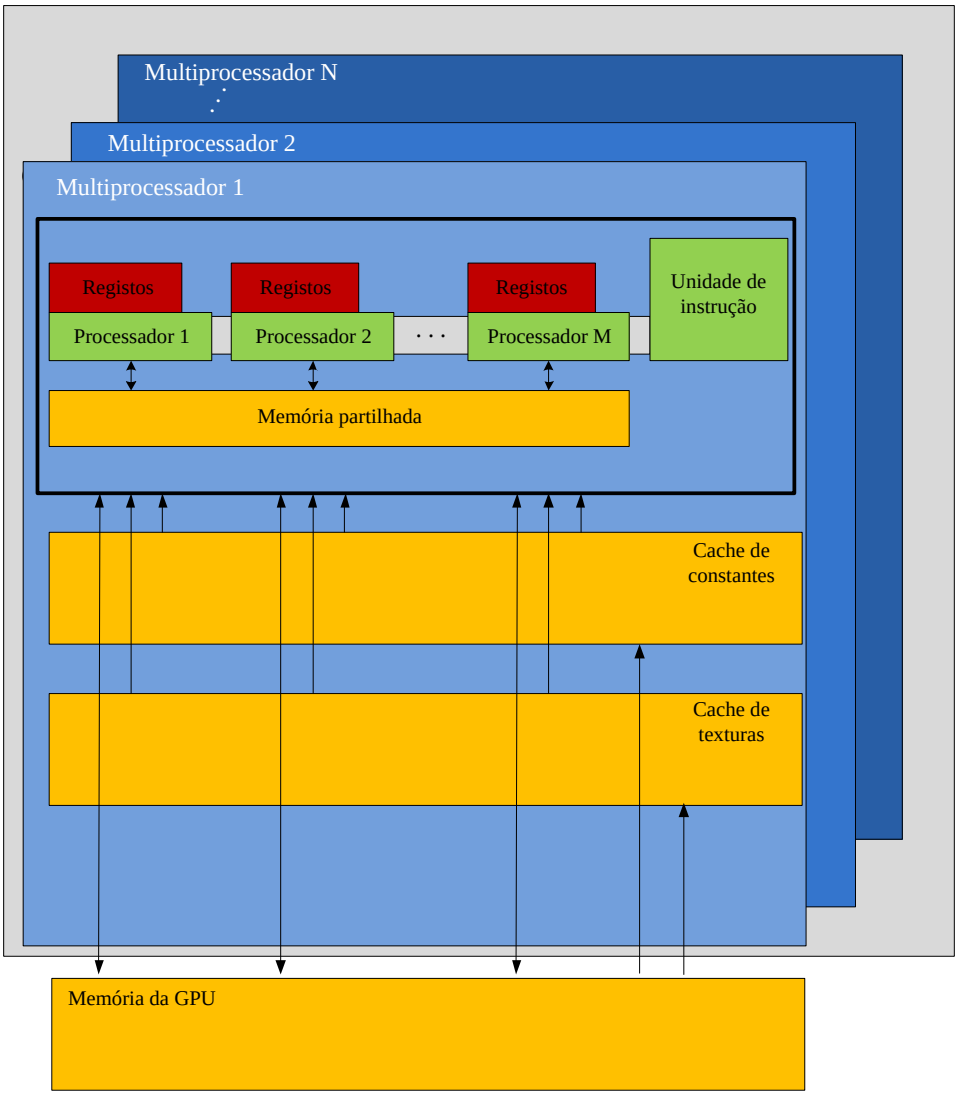

Figura 2.8: Disposição de multiprocessadores numa GPU da NVIDIA. Cada multiprocessador tem 8 núcleos SIMD(Single-Instruction Multiple-Data), onde os 8 núcleos executam uma mesma instrução em dados diferentes. Adaptado de (NVIDIA *programming guide* (28))

- Memória de texturas (*texture memory*): Memória de acesso rápido apenas para leitura que é optimizada para operações com texturas.
- Memória global (*global memory*): Memória de leitura e escrita com a maior latência de acesso.

2.3.2 API

Para executar uma aplicação escrita em CUDA é necessário cumprir requisitos de software e de hardware, no que diz respeito ao hardware, uma lista de unidades de processamento gráfico capazes de processar CUDA pode ser encontrada em (25) Depois de ultrapassado o requisito de hardware, é necessário cumprir os requisitos de software: instalar um driver específico que contém um compilador e algumas ferramentas adicionais que são fornecidas pela NVIDIA (26).

A pilha de software da plataforma CUDA é representada na figura 2.9, composta pelo driver de acesso ao hardware, um componente de execução e duas bibliotecas matemáticas: CUBLAS (*Compute Unified Basic Linear Algebra Sub-programs*) e CUFFT (*Compute Unified Fast Fourier Transform*) e no topo desta pilha encontra-se a API da plataforma CUDA.

A execução é baseada em *threads*, que são escalonadas através de grupos de blocos e formando uma grelha. Com o conceito de bloco e grelha é que se organiza a repartição de dados entre as *threads*, bem como a sua organização e distribuição em *hardware*. Uma grelha pode ter uma ou duas dimensões e os blocos podem ter uma, duas ou três dimensões. Uma função que executa em CUDA é chamada *kernel* e para se efectuar a chamada desta função é necessário especificar as dimensões da grelha e de cada bloco. Para se efectuar a chamada da função utiliza-se entre o nome da função e os argumentos a ela passados, uma tabela bidimensional onde constam as dimensões da grelha e de cada bloco, sendo este delimitado pelos caracteres <<< e >>>. Dentro da função *kernel* é possível aceder aos valores dos índices das *threads* através da variável *threadIdx*, dos valores dos índices dos blocos através da variável *blockIdx* e aos valores das dimensões dos blocos através da variável *blockDim*. Por fim, é possível aceder aos valores das dimensões da grelha através da variável *gridDim*.

A API CUDA fornecida pela NVIDIA introduz extensões à linguagem C, como qualificadores do tipo de função para definir a unidade lógica de execução do código

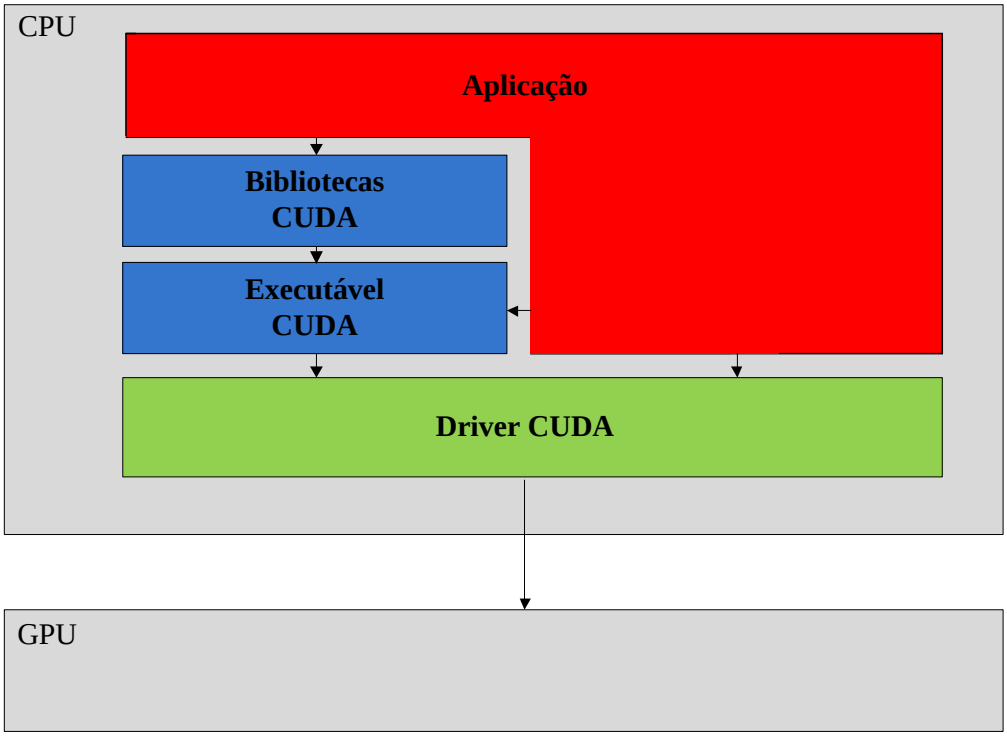

Figura 2.9: Pilha de software da plataforma CUDA. No topo da pilha encontra-se a API da plataforma CUDA. É composta por duas bibliotecas matemáticas prontas para uso: CUBLAS (*Basic Linear Algebra Subroutines*) e CUFFT (*Fast Fourier Transform*), por um componente de execução e pelo driver de acesso ao hardware.

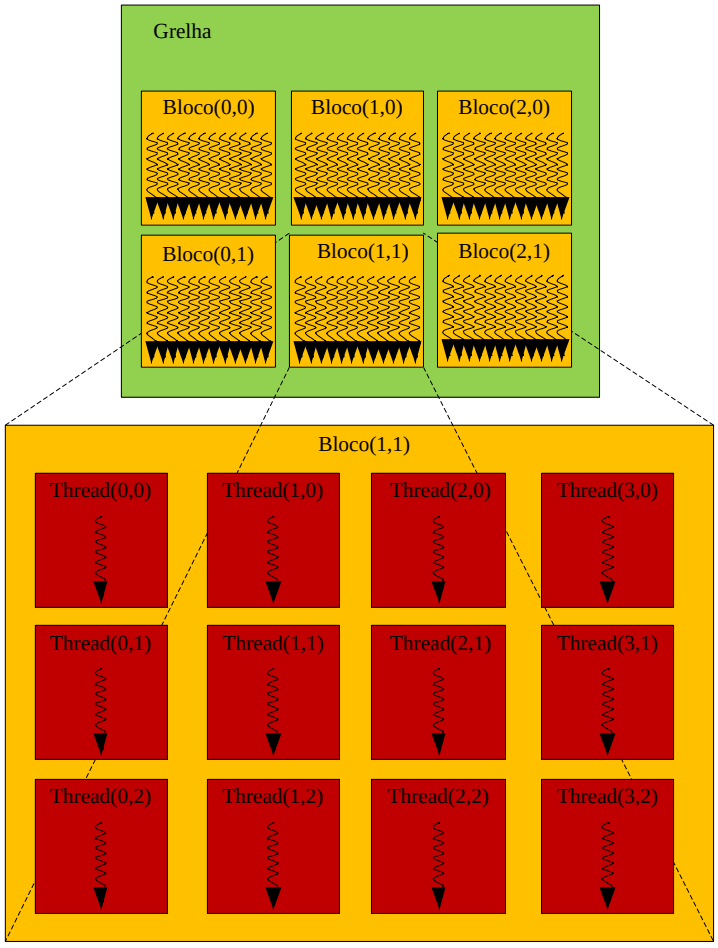

Figura 2.10: Exemplo de uma grelha com os seus blocos de *threads*. Os blocos são distribuídos numa grelha. Nesta figura, exemplifica-se uma grelha de dimensões (2,3) tendo cada bloco um tamanho de (3,4) *threads*.

CPU ou GPU, onde o qualificador *__device__* define que uma função executa na GPU e só pode ser chamada a partir da GPU, o qualificador *__global__* define o que a plataforma CUDA chama de *kernel* e esta função executa na GPU e é chamada a partir da CPU e finalmente o qualificador *__host__* especifica que função vai executar em CPU e só pode ser chamada a partir da CPU. Esta API também introduz qualificadores do tipo de variável, como o *__device__* que define uma variável que reside na memória global da GPU, podendo estas ser acedidas por todas as *threads* de uma grelha e também a partir da CPU através do uso da biblioteca de execução do CUDA e o tempo de vida desta variável é o da aplicação, o qualificador *__constant__* define uma variável que reside no espaço da memória das constantes da GPU e finalmente o qualificador de variável *__shared__* especifica que a variável reside na memória partilhada da GPU e só é acessível pelas *threads* de um determinado bloco, sendo o seu tempo de vida o do bloco.

A API CUDA apresenta algumas restrições, tais como:

- As funções *__device__* e *__global__* não suportam recursividade, não podem ser declaradas variáveis estáticas dentro das mesmas e não podem ter um número variável de argumentos;
- As funções *__device__* não fornecem o seu endereço, mas ponteiros para as funções *__global__* são suportados;
- As funções *__global__* têm de retornar void;
- Os parâmetros de uma função *__global__* estão limitados a 256 bytes;
- Só pode ser usada em GPUs da NVIDIA;

A figura 2.11 ilustra o processamento de um programa escrito em CUDA. Basicamente, o programa inicia copiando os dados a serem processados da memória RAM para a memória da GPU. Após, configura-se uma grelha e é realizada uma chamada da função *kernel* para executar na GPU. Por fim, os dados resultantes são copiados para a memória RAM. Desta forma, o ciclo de execução de uma aplicação utilizando a tecnologia CUDA alterna entre execuções na CPU e na GPU.

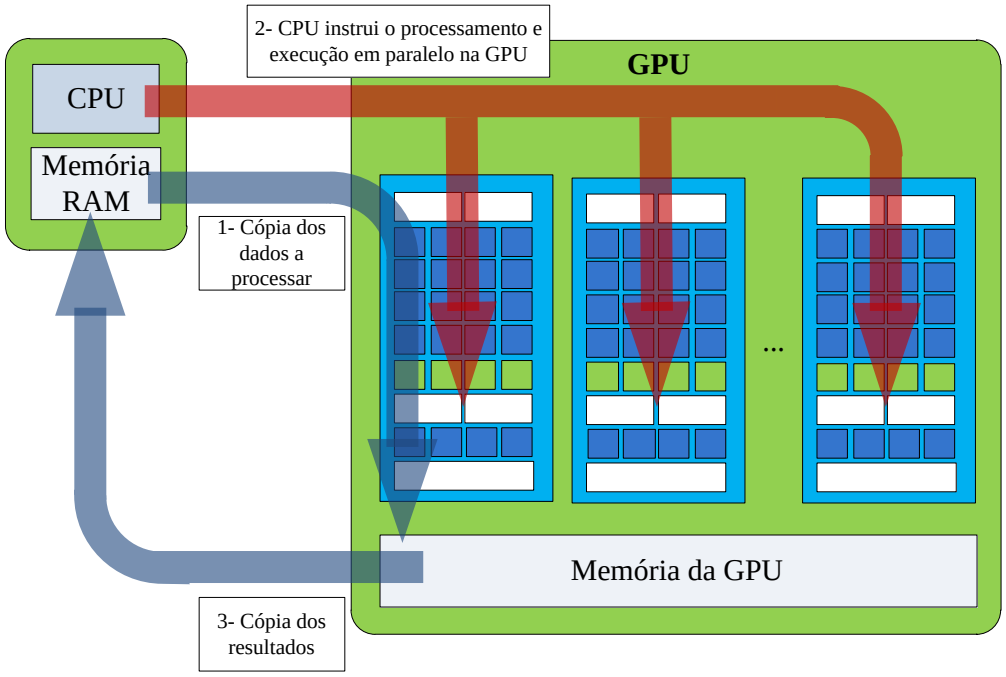

Figura 2.11: Fluxo de execução de um programa escrito em linguagem CUDA. Quando inicia os dados são copiados da memória RAM para a memória da GPU, é configurado o tamanho da grelha, com o número de blocos e a dimensão de cada bloco de *threads*, e é chamado o kernel (função que executa na GPU). No final os dados processados na GPU são copiado para a memória RAM.

3

Implementação

Inicialmente o modelo foi implementado numa versão em Matlab proposta por (13). Esta versão implementa apenas uma frequência crítica sequencialmente, sendo o objectivo primordial conseguir uma implementação paralela das 16 bandas, em tempo real, que à saída forneça um ITD, devido a este ser independente da frequência, 16 ILDs e 16 ICs e ao contrário do ITD estes são dependentes da frequência.

O modelo foi implementado em modo híbrido em FPGA-GPU e CPU-GPU.

O mapa global do fluxo de dados da implementação FPGA-GPU está representado na figura 3.1.

Conseguidos os tempos de cálculo em Matlab do banco de filtros gammatone, tradução neuronal, processador binaural e selecção de amostras, estabeleceu-se que o banco de filtros gammatone fosse implementado numa FPGA, pois mostrou ter um peso computacional mais leve, sendo dos três o mais rápido e em CUDA a tradução neuronal, o processador binaural e a selecção de amostras. Foi ainda implementada uma versão da filtragem gammatone em CPU.

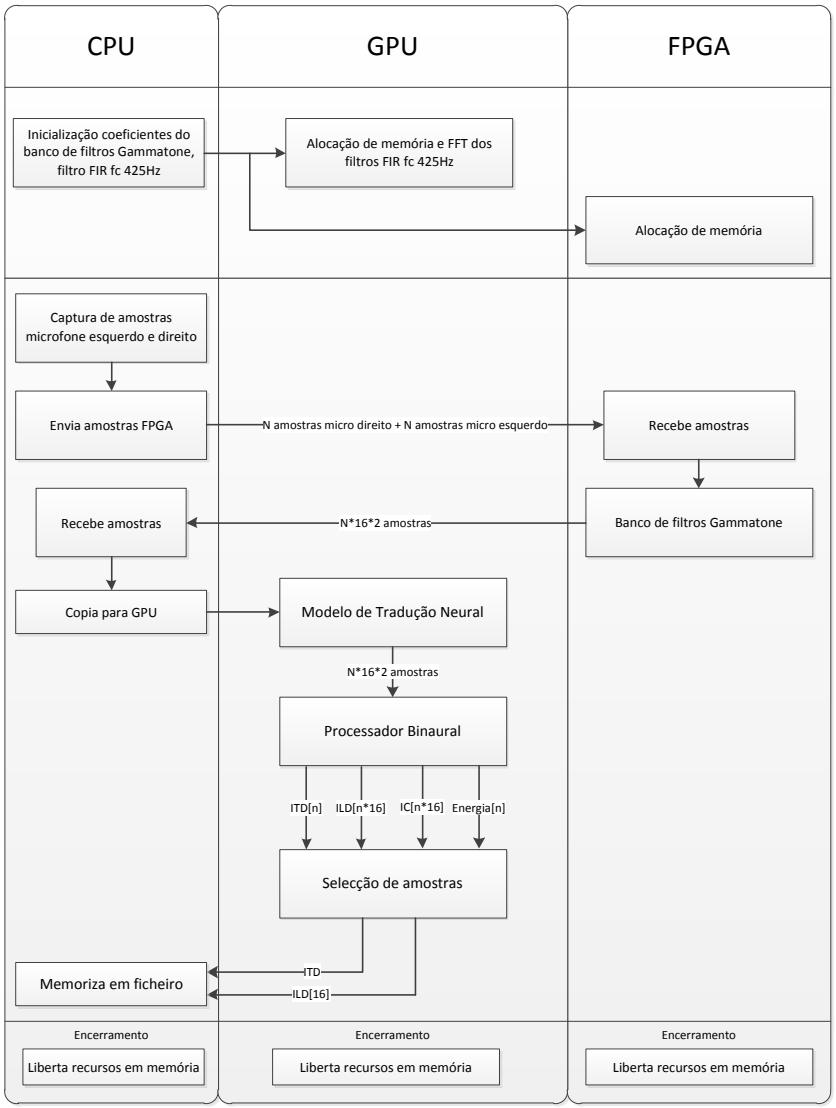

Figura 3.1: Diagrama de fluxo de dados do sistema. No inicio é reservada toda a memória necessária para funcionamento do programa, no CPU, GPU, e FPGA. São criados os coeficientes do banco de filtros gammatone das 16 bandas de frequências e os coeficientes do filtro FIR são convertidos para o domínio da frequência através da FFT. Após captura das amostras de ambos os microfones, é efectuada a filtragem gammatone em FPGA para as 16 bandas aos dados obtidos dos microfones, em CUDA é aplicada a tradução neuronal, o processador binaural e finalmente é realizada a selecção de amostras, resultando num ITD e 16 ILD's.

3.1 FPGA

3.1.1 Implementação Gammatone em hardware FPGA

Primeiramente foi implementada a interface Ethernet utilizando para isso um servidor de sockets (6). Neste *design* é utilizado um processador Nios II (4) e o sistema operativo Micro/OS-II (3), porém o SOPC *builder* fornecido por este *design* não fornece o controlador DM9000a utilizado na DE2, sendo este fornecido através do (1). O diagrama do sistema de comunicação entre o cliente remoto e a FPGA está representado na figura 3.2.

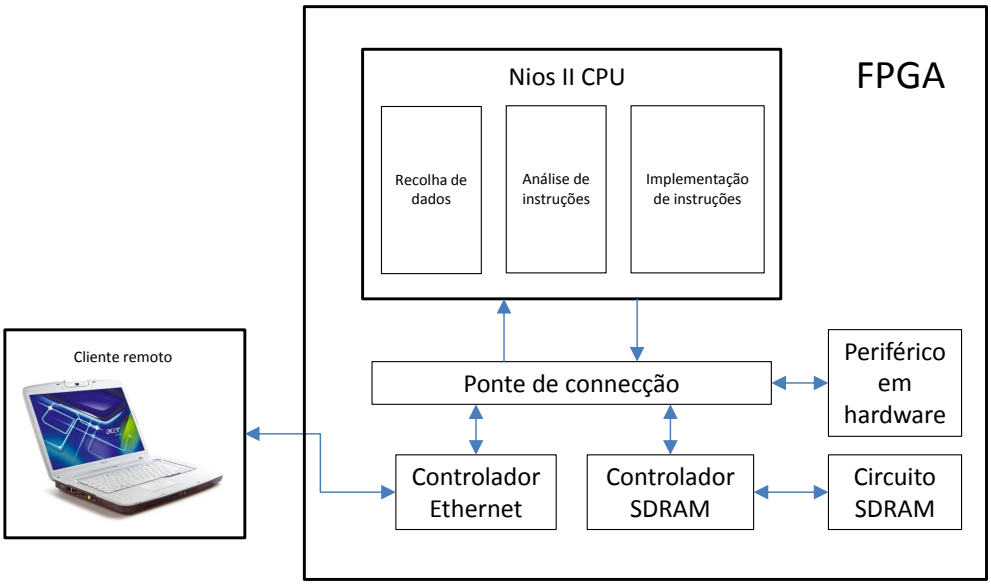

Figura 3.2: Comunicação entre o cliente remoto e a FPGA. O cliente remoto envia as amostras para a FPGA, o processador Nios II processa os dados nas diferentes camadas OSI (*Open Systems Interconnection*) e guarda os valores recebidos na memória SDRAM através do controlador associado. As amostras são enviadas para processamento no periférico em hardware. Quando o cálculo no periférico em hardware termina o processo inverte-se, o NIOS II recebe as amostras do periférico em hardware e envia para o cliente remoto. Ilustração modificada de (17)

Quando a FPGA é programada são iniciados os coeficientes das 16 bandas de frequência do banco de filtros gammatone através da equação 2.3. O banco de fil-

tros é constituído por 4 filtros de resposta a impulso infinita (IIR) em cascata de ordem 2, sendo implementado primeiramente em software Nios II. O processador Nios II tem uma frequência associada de 50MHz, como realizar os cálculos em software Nios II resultaria num tempo de computação elevado e, de forma a acelerar o processo, é feita uma implementação paralela em hardware.

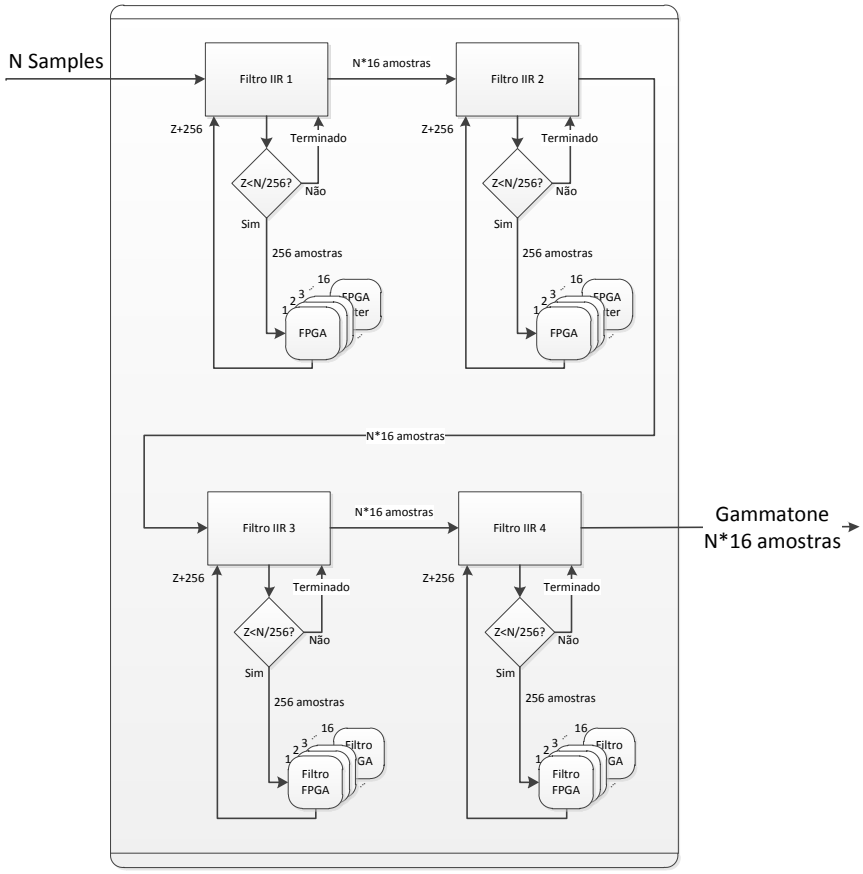

Figura 3.3: Diagrama demostrativo do modelo gammatone em hardware FPGA e como é feita a sua computação. Máximo de amostras possíveis memorizar na FPGA é de 250. Processados N amostras provenientes do microfone esquerdo e N do microfone direitos em blocos de 250 amostras para 4 filtros IIR em sequência. Resultando na saída 16*N amostras da entrada esquerda e 16*N amostras da entrada direita.

O diagrama 3.3 explica a forma como a FPGA processa as amostras. O número máximo de amostras possíveis de memorizar na FPGA é de 250. Neste exemplo são

determinadas N amostras em blocos de 250 amostras para cada um dos 4 filtros IIR, devido à FPGA não conseguir memorizar mais que 250 amostras. Como os filtros IIR possuem *feedback* é necessário memorizar as amostras de saída (y[i-2] e y[i-1]) do bloco de 250 amostras processado antes do novo bloco chegar do Host, continuando o processamento do novo bloco com estas duas amostras como feedback. Este processo é repetido até o último bloco de 250 amostras chegar do *host*.

3.1.2 Periférico em Hardware

Para realizar comunicações com o periférico são utilizadas as funções IOWR e IORD em software Nios II, em que, a cada amostra do sinal de entrada são escritas com a função IOWR (x[i], x[i-1] e x[i-2]) os coeficientes do sinal (b[0], b[1], b[2], a[1] e b[2]) e o feedback (y[i-1] e y[i-2]) demonstrado na figura 3.4. Enquanto o *hardware* calcula está a ser realizada uma leitura através da função IORD da *flag ready* no software. Quando a *flag ready* fica a 1 o sinal de saída pode ser lido.

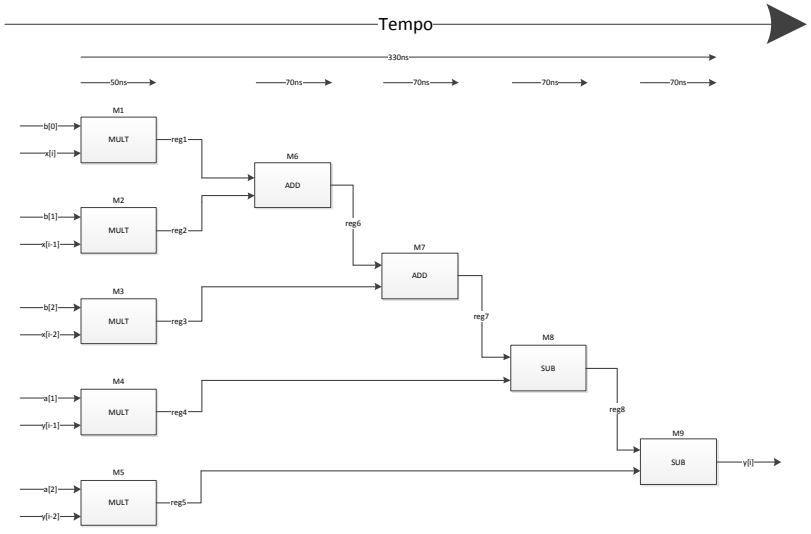

Figura 3.4: Diagrama demostrativo do módulo em hardware. São executadas 5 multiplicações em paralelo, duas adições e duas subtracções correspondendo a uma iteração de um filtro IIR.

O periférico em hardware consiste em 5 multiplicações em paralelo, duas adições e

duas subtracções de vírgula flutuante de 32 bits denominados de módulos (IP) *intellectual property* da Altera (2).

O módulo em hardware apresentado na figura 3.5 é invocado pelo sistema operativo Micro/OS-II (3).

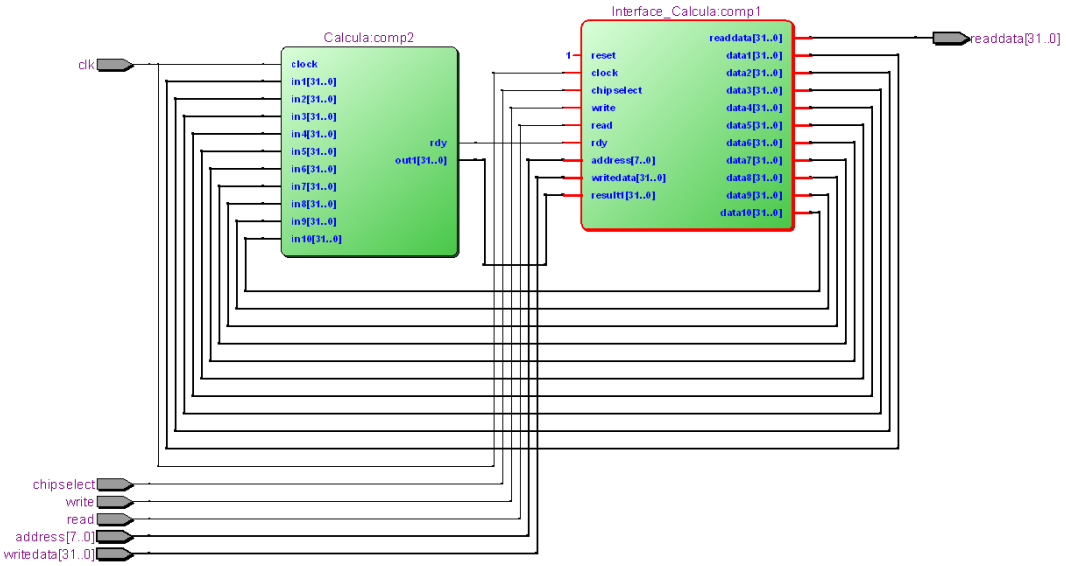

Figura 3.5: Diagrama demostrativo do modulo em hardware. O *chipselect* está a 1 quando a função IORW ou IORD é invocada em *software* e a 0 nos outros casos. O *clock* é colocado a 50Mhz. O *read* está a 1 no caso da função invocada ser IORD e o *write* no caso da função ser IOWR. O endereço irá definir em que zona (do in[1] ao in[10]) do periférico se deseja ler ou escrever.

Por fim quando o cálculo de todas as amostras enviadas pelo *host* é finalizado, o resultado é enviado da FPGA para o *host* através do controlador Ethernet.

3.2 CUDA

Nesta secção explica-se a implementação da tradução neuronal, processador binaural e escolha de amostras e como foi efectuada a sua paralelização em CUDA.

O modelo proposto por (13) possuí a implementação em Matlab, que se revelou lenta.

A princípio é definido o tamanho da janela de processamento, a frequência de amostragem do sinal de entrada, a frequência das 16 frequências criticas, o ITD máximo, o ILD máximo, o limiar c_0 e o limiar de energia.

Através do tamanho da janela de processamento n e do tamanho do atraso m é alocada a memoria na CPU e na GPU. n e m são directamente proporcionais ao uso de memória.

De seguida são inicializados em CPU os coeficientes da transformada de Hilbert, os coeficientes do filtro FIR passa-baixo com frequência de corte 425Hz, referentes à tradução neuronal e realizada a cópia para a memória global da GPU. É realizada a FFT dos coeficientes através da biblioteca (24, CUFFT), de forma a esta ser realizada apenas uma vez no decorrer do programa. Aqui são também inicializados os coeficientes do banco de filtros gammatone.

3.2.1 Tradução Neuronal

Nesta secção é explicada a implementação do módulo de tradução neuronal de acordo com a figura 3.6 baseado na figura 2.4.

O processo de paralelização em linguagem CUDA é realizado através de uma grelha, representada na equação 3.1 e cada bloco contem o número de *threads* especificados na equação 3.2.

$$grelha = (\frac{n}{64}, 32) \tag{3.1}$$

$$bloco = (64, 1) \tag{3.2}$$

Onde 64 é o numero de *threads*, n o tamanho da janela pré-definida e 32 os 16 canais x_1 e 16 canais x_2.

Os dados de saída do banco de filtro gammatone são copiados para a memória global da GPU. De seguida é aplicada a transformada de Hilbert, onde se recorre ao domínio da frequência, utilizando a FFT, pois uma convolução consome muitos recursos. No domínio da frequência é aplicada a multiplicação de n em n e por fim a IFFT. A IFFT origina valores reais e imaginários, sendo preciso extrair a magnitude ponto a ponto.

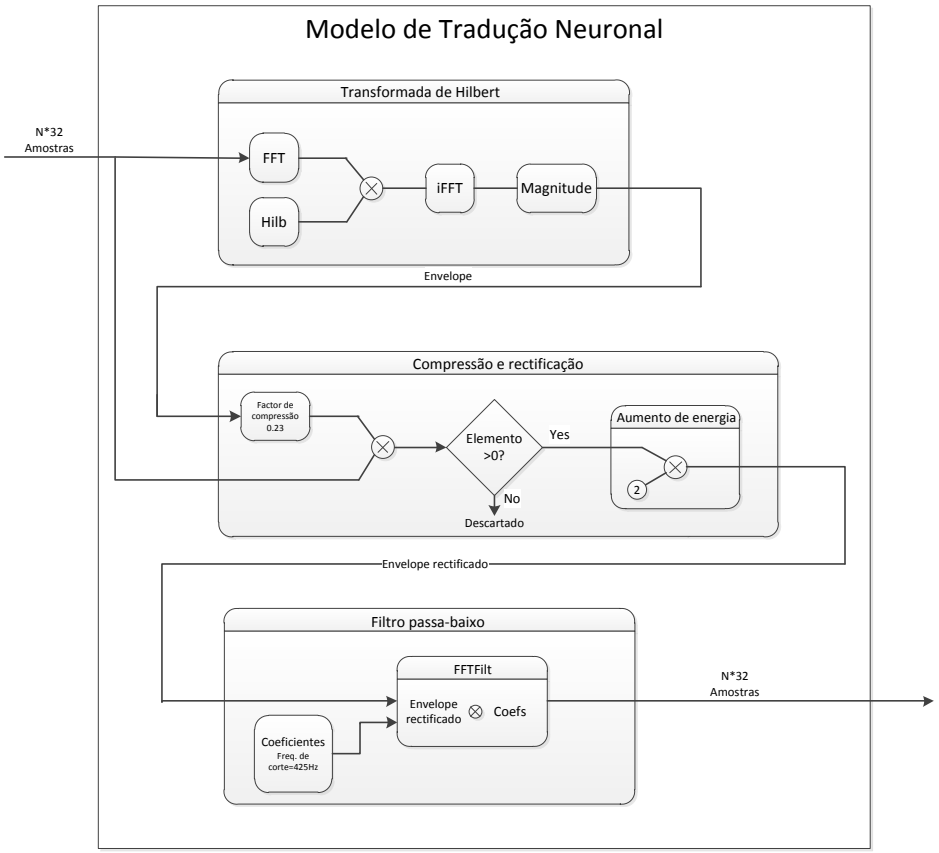

Figura 3.6: Diagrama demostrativo do módulo tradução neuronal. Aplicada à saída do banco de filtro gammatone a transformada de Hilbert, realizada a compressão em envelope e finalmente aplicado o filtro FIR passa baixo de frequência de corte 425Hz.

Para se efectuar a compressão é efectuado o exponencial a todas as amostras dos sinais por um factor de 0.23 e ao resultado é multiplicado o sinal proveniente do banco de filtros gammatone. De seguida é realizada uma rectificação de meia onda, onde são verificadas as amostras que possuem valor superior a zero e, em caso de existirem, é reposto o seu valor original a zero. Posteriormente é aplicado um exponencial de factor 2 de forma a aumentar a energia.

Finalmente é aplicado o filtro FIR passa-baixo com frequência de corte 425Hz. É utilizado o domínio da frequência pelo mesmo motivo a quando da transformada de Hilbert. São realizadas as 32 FFT's dos envelopes resultantes da compressão e rectificação e é aplicada a multiplicação ponto a ponto com a FFT pré-calculada na inicialização do programa, e de seguida a IFFT e cálculo da magnitude. Obtêm-se 16 *Nerve Firing* x_1 e 16 *Nerve Firing* x_2.

3.2.2 Processador Binaural

Os operadores $a_{12}(n,m)$, $a_{11}(n,m)$ e $a_{22}(n,m)$ contêm valores de decadência α da amostra anterior em relação à amostra actual a ser calculada. Este processo é simples em processamento sequencial, sendo algo complexo em computação paralela. A solução encontrada para o cálculo em paralelo foi dividir as equações 2.5, 2.6 e 2.7, nas equações 3.3, 3.4 3.5, 3.6, 3.7 e 3.8 de forma a realizar os cálculos sem e com dependências separadamente sendo as equações resultantes as seguintes:

$$a_{12_{prefix}}(n,m) = \alpha x_1(n - maxm, 0)x_2(n - max\{-m, 0\}), \tag{3.3}$$

$$a_{11_{prefix}}(n,m) = \alpha x_1(n - maxm, 0)x_1(n - max\{m, 0\}), \tag{3.4}$$

$$a_{22_{prefix}}(n,m) = \alpha x_2(n - max{-m}, 0)x_2(n - max\{-m, 0\}). \tag{3.5}$$

$$a_{12}(n,m) = a_{12_{prefix}}(n,m) + (1-\alpha)a_{12_{prefix}}(n-1,m), \tag{3.6}$$

$$a_{11}(n,m) = a_{11_{prefix}}(n,m) + (1-\alpha)a_{11_{prefix}}(n-1,m), \tag{3.7}$$

$$a_{22}(n,m) = a_{22_{prefix}}(n,m) + (1-\alpha)a_{22_{prefix}}(n-1,m). \tag{3.8}$$

As equações 3.3, 3.4 e 3.5 são implementadas com uma grelha representada na equação 3.9 e cada bloco tem o número de *threads* representado na equação 3.10.

$$grelha = (\frac{n}{64}, 2m + 1) \tag{3.9}$$

$$bloco = (64, 1) \tag{3.10}$$

Para efectuar a paralelização do cálculo do ILD as equações 2.12 e 2.13 são alteradas da seguinte forma:

$$L_{1_{prefix}}(n, m) = \alpha x_1^2(n - max\{m, 0\}) + (1 - \alpha)L_1(n - 1, m), \tag{3.11}$$

$$L_{2_{prefix}}(n, m) = \alpha x_2^2(n - max\{m, 0\}) + (1 - \alpha)L_2(n - 1, m). \tag{3.12}$$

$$L_1(n, m) = L_{1_{prefix}}(n, m) + (1 - \alpha)L_{1_{prefix}}(n - 1, m), \tag{3.13}$$

$$L_2(n, m) = L_{2_{prefix}}(n, m) + (1 - \alpha)L_{2_{prefix}}(n - 1, m). \tag{3.14}$$

Com a grelha 3.9, e cada bloco com o numero de *threads* representado na equação 3.10 são realizados os cálculos da equação 3.11 e 3.12.

Através da biblioteca *Thrust* (37) é feita a soma de prefixos existente nas equações 3.6, 3.7, 3.8, 3.13 e 3.14.

Finalmente utilizando uma grelha representada na equação 3.15 e cada bloco com o número de *threads* representado na equação 3.16 é calculada a correlação cruzada normalizada através da equação 2.4 extraindo aqui o ITD através da equação 2.9 e o IC através da equação 2.10. Após o cálculo da equação 2.9 é conhecida a posição m onde se encontra o máximo da função de correlação cruzada normalizada 2.4 sendo este utilizado para extrair o ILD e a energia de acordo com as equações 2.11 e 2.14.

$$grelha = (\frac{n}{64}, 16) \tag{3.15}$$

$$bloco = (64, 1) \tag{3.16}$$

Onde 64 é numero de *threads* em cada bloco e 16 o número de canais.

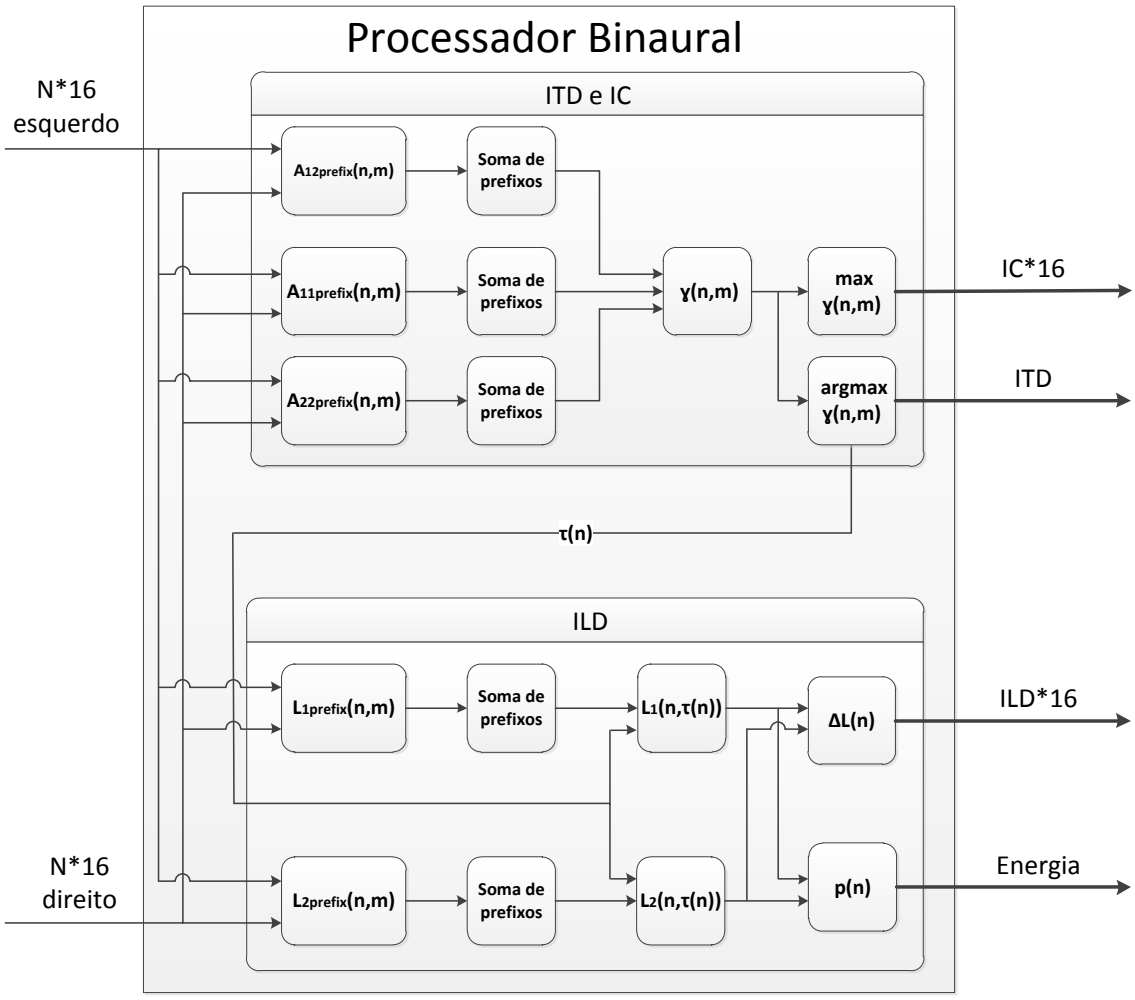

Figura 3.7: Diagrama demostrativo do processador binaural. Cálculo de prefixos das equações 3.6, 3.7, 3.8, 3.13 e 3.14, soma de prefixos, cálculo da função de correlação cruzada normalizada, extracção do IC e ITD, seguido do cálculo do ILD e da energia.

3.2.3 Selecção de amostras

De acordo com a equação 2.15 são eliminados os ecos através do limiar c_0 definido inicialmente. No início do programa o utilizador pode introduzir um valor de limiar de energia, onde as amostras são consideradas a partir desse limiar.

Utiliza-se uma grelha representada na equação 3.15 e cada bloco tem um número de *threads* representado na equação 3.16 e é feita a selecção do seguinte modo:

- Através da equação da energia 2.14, é verificado se cada amostra é superior limiar ao definido no início do programa e, se o IC é superior ao limiar c_0, no caso de o IC ou o $p(n)$ menor que os limiares, a amostra não é considerada e o seu valor original é reposto a zero. Este processo é repetido paralelamente para todas as amostras.

- As amostras resultantes após as condições consideradas anteriormente são usadas para estimar o histograma do ITD e determinar os 16 histogramas dos ILDs. Para a implementação dos histogramas em linguagem CUDA é utilizada a biblioteca *Histogram* (31). Cada histograma possui 256 intervalos discretos e onde é possível distinguir a frequência de ocorrências existentes no ITD e nos ILD's em cada intervalo discreto.

- A cada bloco de n amostras é criada uma PDF para o ITD e 16 PDFs para os ILDs. Posteriormente é identificado o máximo de cada uma das PDFs através da uma função existente na biblioteca *Thrust* (37) que calcula o máximo global, extraindo assim 1 ITD e 16 ILDs a cada bloco de n amostras processadas. Por fim estes valores são copiados para a memória da CPU e armazenados em disco.

4

Resultados e Discussão

Neste capítulo apresentam-se os resultados da implementação do modelo de indicações binaurais, e faz-se a comparação entre os resultado obtidos com a implementação em Matlab (fallaer).

Para obter um ITD e 16 ILD's são executados os seguintes passos:

- Obtenção dos sinais do microfone esquerdo e microfone direito através do servidor de áudio de baixa latência (29, JACK Connection Kit).
- Cópia de amostras em CPU com o cliente existente em (29).
- Aplicação de filtro Gammatone (4 filtros IIR) para as 16 bandas críticas.
- Cópia de amostras para memória global da GPU.
- Transformada de Hilbert a 32 bandas (16 para cada microfone) aos dados de saída do banco de filtros Gammatone.
- Compressão em envelope e rectificação de 32 bandas.
- Filtragem passa-baixo de 32 bandas.
- Cálculo de 16 bandas de elementos $a12_{prefix}(n,m)$, $a11_{prefix}(n,m)$, $a22_{prefix}(n,m)$, $L1_{prefix}(n,m)$, $L2_{prefix}(n,m)$.
- Soma de prefixos de 16 bandas.
- Função de correlação cruzada normalizada (extracção de ITD e IC) para 16 bandas.

- ILD e energia combinada para 16 bandas.
- Com o limiar c_0 e o limiar da energia as amostras são rectificas para uma banda ITD e 16 bandas ILD, no último apenas considerado o limiar c_0.
- Estimação de 17 histogramas (uma banda ITD e 16 bandas ILD).
- Máximos globais dos 17 histogramas.
- Obtenção de 1 ITD e 16 ILD's.

A implementação foi testada usando um Pentium Dual Core 3.4Ghz com uma GPU NVIDIA 9800 GTX+ com 512MB de memória dedicada e uma FPGA DE2 com 35000 Logic Elements, 8 MB de SDRAM e controlador Ethernet de 10/100Mb.

Nesta implementação para que se obtenha um resultado em tempo real, os blocos de processamento necessitam ser maiores que 2048 amostras, ou seja, dividindo por a frequência de amostragem de 44,1KHz, origina 34.83 ms. Um número de amostras reduzido tem impacto na qualidade nos histogramas finais, pois existe menos informação por bloco. Por fim, de forma a avaliar a eficácia da implementação são comparados os tempos das implementações FPGA-GPU e CPU-GPU com a versão em Matlab.

A alocação de memória ocorre apenas uma vez, tanto na CPU, como GPU e FPGA. São inicialmente criados os coeficientes gammatone através de 2.3 e os coeficientes do filtro passa-baixo.

4.1 Implementação FPGA e GPU

De acordo com os objectivos a implementação é realizada em FPGA e GPU, onde o banco de filtros gammatone é implementado em FPGA, a tradução neuronal, processador binaural e selecção de amostras é realizada em linguagem CUDA.

Na tabela 4.1 são demonstrados os tempos de cálculo da implementação híbrida FPGA e linguagem CUDA em função do tamanho do bloco de processamento.

Como se pode verificar pela tabela 4.1 o peso temporal da FPGA é elevado. Na figura 4.1 é demonstrada a proporção temporal em FPGA para um bloco de 3072 amostras, estando os valores discriminados na tabela 4.2. Na figura 4.1 representa-se o gráfico de proporção temporal da implementação FPGA.

Número de amostras por bloco	Tempo de cálculo por bloco FPGA	Tempo de cálculo por bloco GPU	Total FPGA+GPU
512	2,3s	0,017s	2,317s
1024	4,58s	0,022s	4,602s
2048	9,14s	0,034s	9,172s
3072	13,69s	0,045s	13,735s
4096	18,21s	0,056s	18.266s
8192	36,41s	0,102s	36.512s

Tabela 4.1: Tempos de cálculo da implementação híbrida FPGA e GPU. O tempo de cálculo com a FPGA revelou-se bastante lento devido ao tempo despendido nas comunicações Ethernet e o tempo de escrita/leitura para o *hardware*.

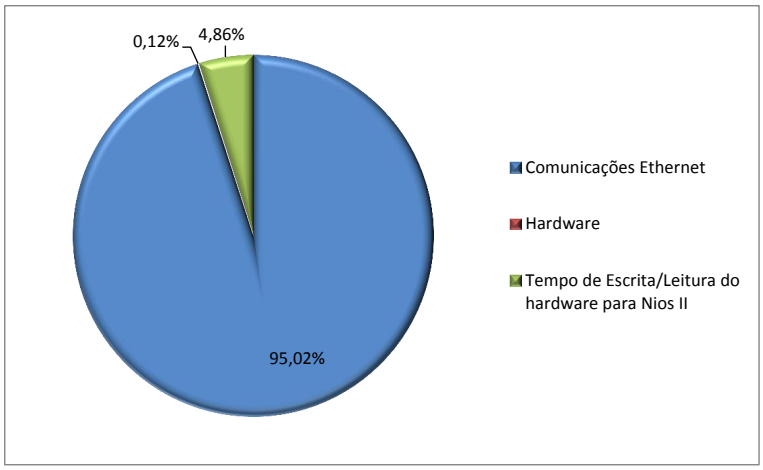

Figura 4.1: Proporção temporal para o cálculo de um bloco de 3072 amostras. Os tempos de comunicação Ethernet revelam um consumo temporal de 95%, a escrita/leitura no hardware 5%. Os cálculos em hardware revelaram um peso, em percentagem, diminuto quando comparado com os outros dois pesos.

Comunicações Ethernet	12,89s
Tempo de Escrita/Leitura entre o hardware e o NIOS II	0,66s
Hardware	0,016s

Tabela 4.2: Tempos de cálculo discriminado da implementação híbrida FPGA e GPU para um bloco de 3072 amostras. Devido ao tempo despendido nas comunicações Ethernet o tempo de escrita e leitura para o *hardware*, esta implementação revelou-se lenta.

Hardware FPGA	0,016s
CPU	0,00842s

Tabela 4.3: Comparação dos tempos de cálculo em Hardware FPGA e o modelo homólogo em CPU.

Considerando que não existem comunicações Ethernet e leituras/escrita entre o hardware e o Nios II, a implementação em FPGA a 50Mhz revelou-se mais lenta que num CPU a 3,4Ghz. Foram realizados testes de desempenho na troca de pacotes, revelando que o servidor de sockets possuía taxas de transferência de 100KB/s, sendo este valor inferior do que é necessário ($2canais * 16 * bandas * 44100fs = 1.35MB/s$). Devido à indisponibilidade de uma FPGA de alto desempenho não se obtiveram os resultados esperados, no entanto em caso de uso de uma FPGA com características superiores (8) com um frequência de relógio de 400MHz (8 vezes superior à utilizada no presente trabalho) integrada no computador através de uma interface *PCI-Express*, permitiria um tempo de cálculo substancialmente inferior.

4.2 Implementação CPU e GPU

Na presente secção é demonstrada a implementação do banco de filtros gammatone em CPU e os restantes módulos em GPU.

Na figura 4.2 é possível visualizar o gráfico de tempos de computação em função do número de amostras por bloco de processamento realizado a uma frequência de amostragem de 44,1 KHz.

Na tabela 4.4 são demonstrados os tempos de cálculo da implementação CPU e GPU para blocos com diferentes números de amostras, onde se verifica que a partir de

2048 amostras por bloco se obtém um resultado em tempo real. O tempo de cálculo por bloco é adquirido através da média do número de blocos utilizados, o tempo de necessário para o tempo real é a adquirido conforme o tamanho do bloco, e por último a taxa de processamento é o tempo necessário para tempo real em hertz.

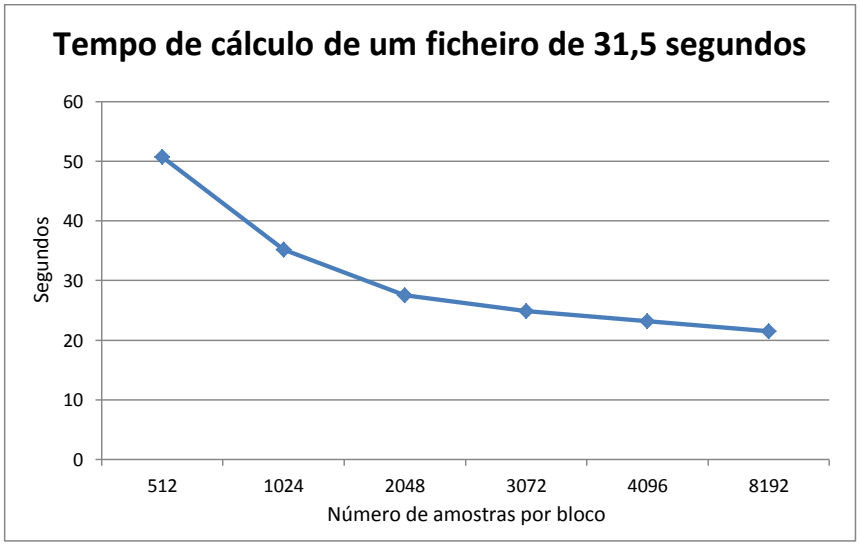

Figura 4.2: Tempos de cálculo da implementação CPU e GPU em função do número de amostras por bloco de um ficheiro com 31,5 segundos.

Conforme pode ser observado na figura 4.3, o consumo de tempo de maior dimensão refere-se à soma de prefixos, com um tempo total de 31,6ms, representando 55% do tempo total da computação do ITD e dos 16 ILDs.

Da figura 4.4 observa-se que existe um *speedup* elevado da implementação em CPU-GPU em relação à implementação sequencial em Matlab. Devido ao tempo consumido pelas comunicações Ethernet e o tempo de escrita/leitura entre o hardware e o NIOS II a implementação FPGA-GPU revelou-se mais lenta que a implementação em Matlab. Considerando nulas as comunicações *ethernet* e latência de escrita/leitura na implementação FPGA-GPU, o *speedup* poderia atingir 71x, 15x menor que a implementação

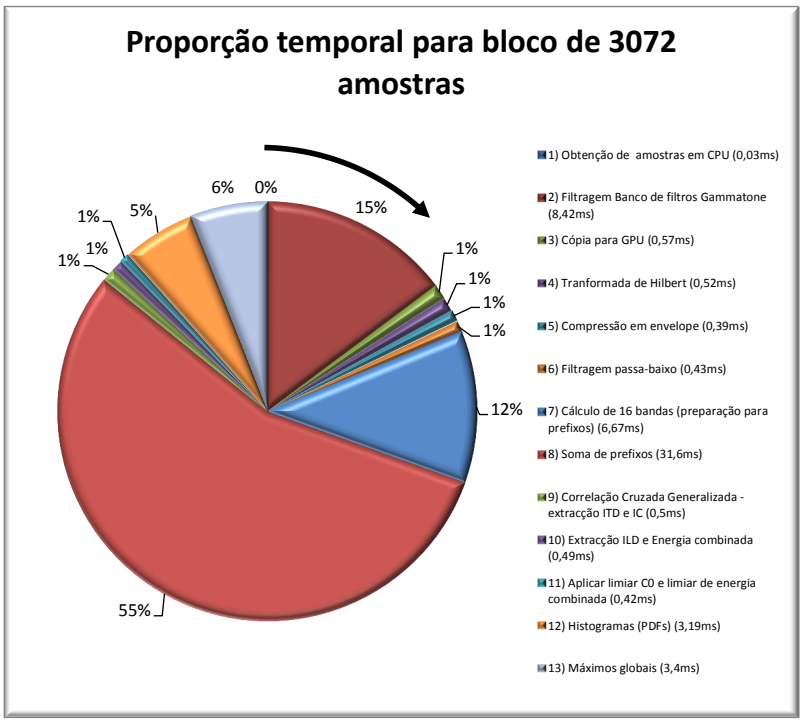

Figura 4.3: Proporção temporal entre os vários estágios do cálculo do ITD e dos ILDs de 16 bandas de frequência. Neste gráfico é demonstrado o peso de cada passo, em percentagem. O oitavo passo é o que tem mais peso no tempo de computação, representando 55% do tempo total.

Número de amostras por bloco	Número de blocos	Tempo de cálculo	Tempo de cálculo por bloco	Tempo necessário para tempo-real	Taxa de processamento
512	2752	50,700s	18,42ms	11,61ms	86,13Hz
1024	1376	35,170s	25,56ms	23,22ms	43,06Hz
2048	688	27,549s	40,04ms	46,43ms	21,53Hz
3072	458	24,883s	53,33ms	69,65ms	14,35Hz
4096	344	23,189s	67,41ms	92,87ms	10,76Hz
8192	172	21,497s	124,98ms	185,76ms	5,38Hz

Tabela 4.4: Tempos de processamento para blocos com diferentes números de amostras, utilizando um ficheiro com 31.95 segundos de som. O tempo de cálculo por bloco é adquirido através da média do número de blocos utilizados, o tempo de necessário para o tempo real é a adquirido conforme o tamanho do bloco, e por último a taxa de processamento é o tempo necessário para tempo real em hertz.

CPU-GPU, porém com uma FPGA de 400MHz em vez de 50MHz o *speedup* seria superior à implementação CPU-GPU.

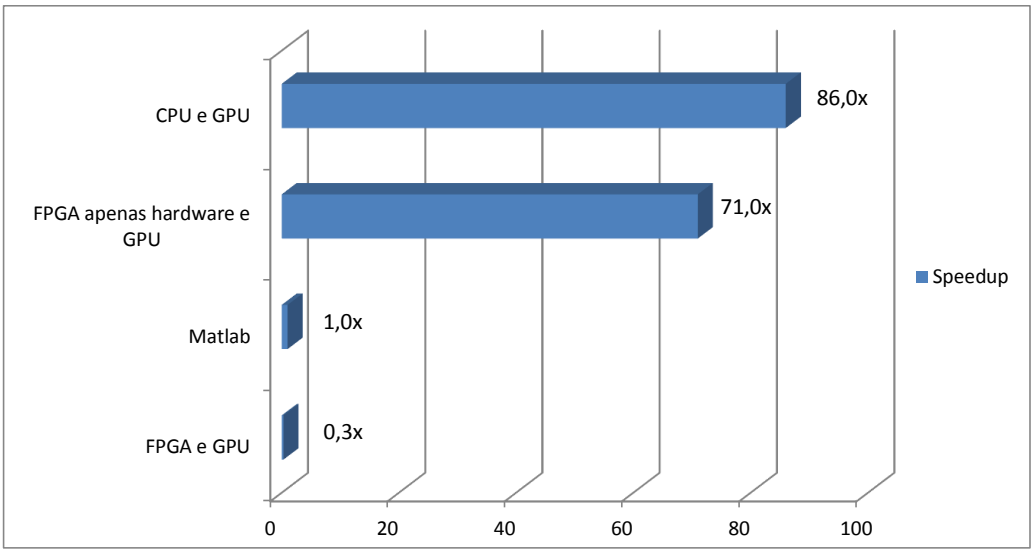

Figura 4.4: Speedup das diferentes implementações realizadas para o modelo. A implementação em CPU e GPU possui um speedup de 86,0x em relação à implementação em Matlab. Não sendo consideradas as comunicações *ethernet* e latência de escrita/leitura na implementação FPGA-GPU, um *speedup* poderia atingir 71x.

4.3 Resultados experimentais

É lido um ficheiro áudio estéreo com aproximadamente 30 segundos onde se encontra registada uma voz a repetir "Nicole, look at me" aproximadamente a cada 3 segundos, sendo também registadas outras vozes próximas da cabeça robótica, vindas de diferentes lugares do laboratório. O orador encontra-se posicionado no lado direito e move-se lentamente para o centro, continuando a mover-se para a esquerda e finalmente voltando ao centro.

Utilizando blocos de 3072 amostras, correspondendo aproximadamente a 69,65 ms a uma frequência de amostragem de 44100Hz. O valor máximo do ITD pode ser estimado conforme a distância dos microfones, $\approx 18cm$, dividindo pela velocidade do som, $\approx 340m/s$, originando um valor de ITD máximo de $0.529ms$.

Seguem-se os resultados detalhados de um bloco de processamento para a frequência crítica de 500Hz, onde é evidenciado o método do algoritmo para estimar a localização.

Na figura 4.5 apresenta-se os gráficos do quinto bloco do ficheiro de áudio contendo 3072 amostras, é escolhido este número de amostras para que se obtenha uma taxa de

aquisição de dados de $\approx 15Hz$.

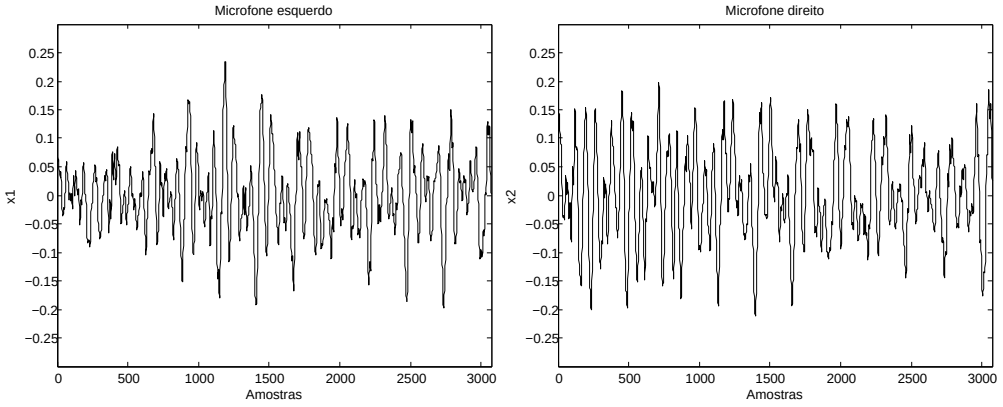

Figura 4.5: Som de entrada esquerdo e direito para um bloco de 3072 amostras.

Aos dados de entrada apresentados na figura 4.5 é aplicado o banco de filtros gammatone com frequência crítica de 500 Hz. Ao aumentar a frequência crítica, a saída do banco de filtros gammatone terá uma amplitude menor e uma frequência mais acentuada.

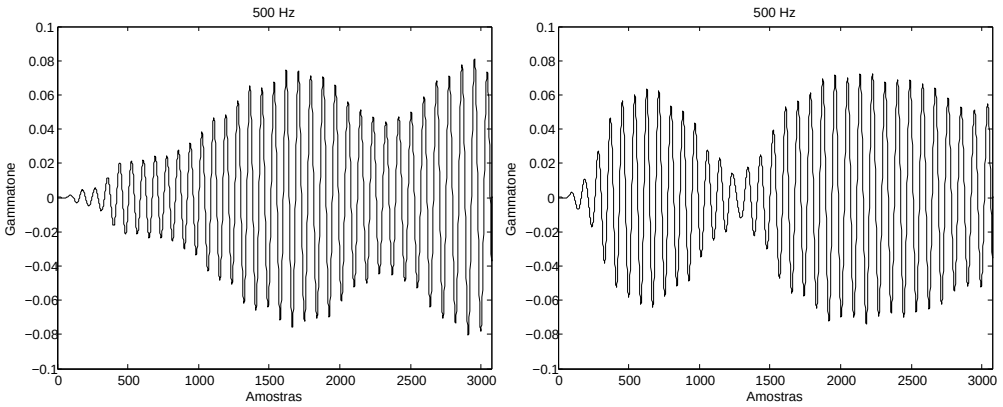

Figura 4.6: Aplicação dos filtros gammatone aos dados de entrada. O gráfico do lado esquerdo representa a saída da filtragem gammatone dos dados do microfone esquerdo e gráfico do lado direito representa a saída da filtragem gammatone dos dados do microfone direito.

Na figura 4.7 são apresentados os resultados ao aplicar-se a tradução neuronal ao sinal de saída da filtragem gammatone 4.6.

Após a tradução neuronal, realiza-se a correlação cruzada generalizada, extraindo

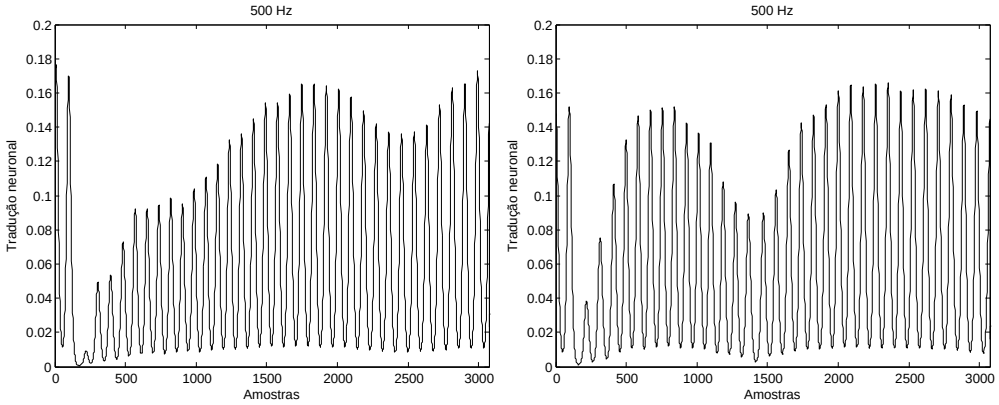

Figura 4.7: Resultado da tradução neuronal.

o seu máximo instantâneo 2.10, os resultados obtidos são apresentados na figura 4.8. Daqui obtém-se a coerência interaural, onde o limiar $c_0 = 0.9$ e considera-se o limiar de energia = 0.006 2.14.

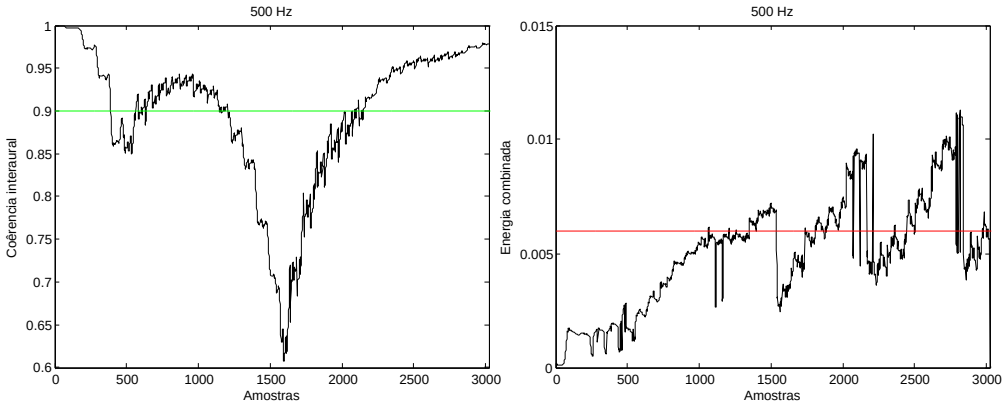

Figura 4.8: Gráfico da coerência interaural e energia combinada. No gráfico da esquerda a linha a verde corresponde ao limiar de $c_0 = 0.9$ e no gráfico da direita a linha a vermelho representa o limiar de energia = 0.006 2.14.

Com os dados calculados da correlação cruzada normalizada aplicam-se as equações 2.9 e 2.11 e obtém-se o ITD e o ILD para uma banda de frequência de 500Hz, sendo os gráficos com o resultado apresentados na figura 4.9.

Com o ITD e o ILD obtido e os limiares de $c_0 = 0.9$ e energia = 0.006 aplica-se a equação 2.15 para se efectuarem a selecção de amostras. Neste exemplo só são consideradas as amostras, tanto para o ITD, como para o ILD que ultrapassem os

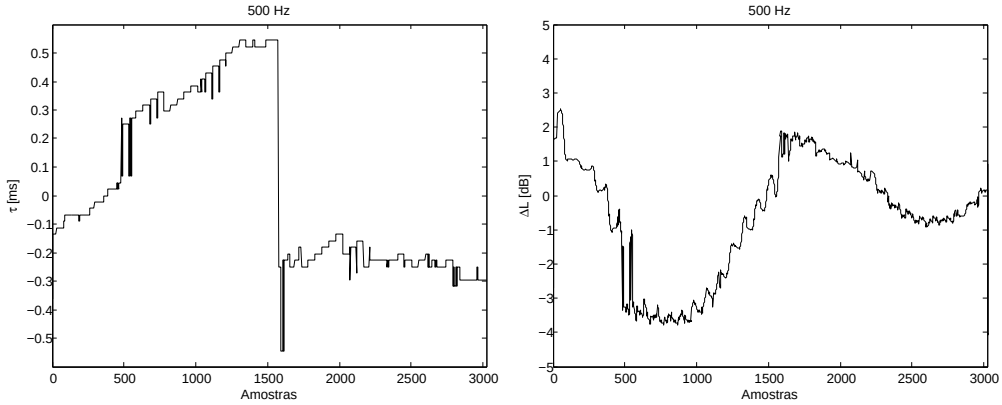

Figura 4.9: Resultado do ITD apresentado no gráfico do lado esquerdo e do ILD apresentado no gráfico do lado direito.

limiares $c_0 = 0.9$ de coerência interaural (IC) e de energia = 0.006; caso não ultrapassem o seu valor é reposto a zero e não contando para a função de densidade de probabilidade.

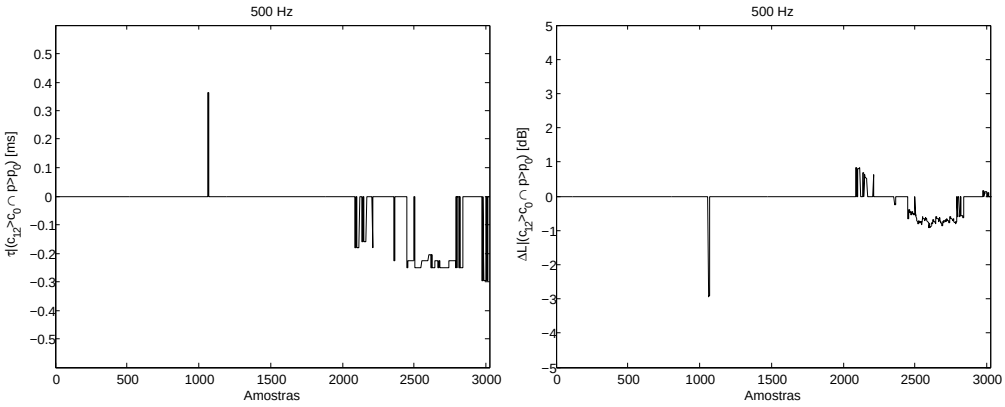

Figura 4.10: ITD e ILD rectificados. Depois de efectuada a selecção de amostras obtém-se o ITD apresentado no gráfico do lado esquerdo e o ILD apresentado no lado direito. Apenas são consideradas as amostras que ultrapassem os dois limiares obtidos conforme demonstrado na figura 4.8.

Dos dados obtidos da selecção de amostras é realizado o histograma do ITD e do ILD de resolução 256 intervalos discretos, sendo os histogramas apresentados na figura 4.11.

Para verificar a precisão do modelo implementado os resultados para o ITD e o ILD obtidos em Matlab são demonstrados na figura 4.12 para os mesmos parâmetros

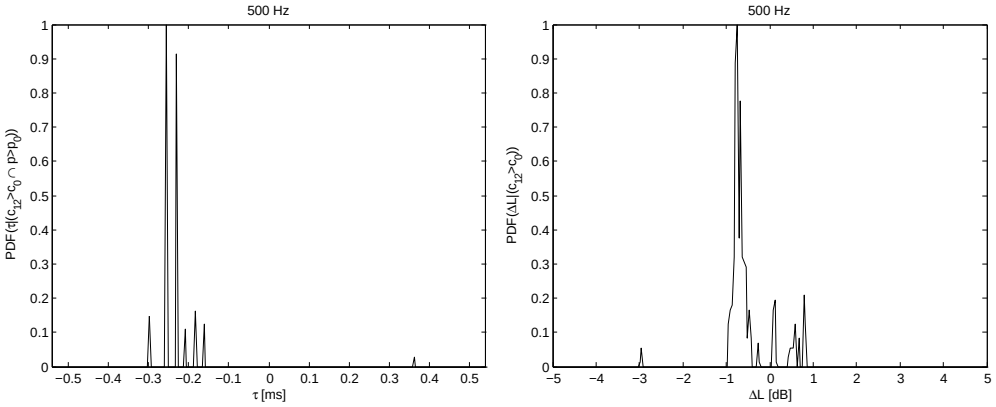

Figura 4.11: Histograma do ITD representado do lado esquerdo e o do ILD apresentado do lado direito, utilizando a implementação proposta neste trabalho.

e comparando com a figura 4.11 onde se apresentam os resultados da solução apresentada nesta dissertação, verifica-se que tanto o ITD como o ILD são semelhantes. Estes histogramas não possuem características exactamente iguais, em comparação implementação em Matlab, devido à saída da soma de prefixos em GPU ser realizada como múltiplas subsomas em paralelo (27), estas subsomas podem alterar ligeiramente o sinal, porém não existe uma influência acentuada no sinal final (visível na figura 4.16), existindo uma localização próxima da posição real.

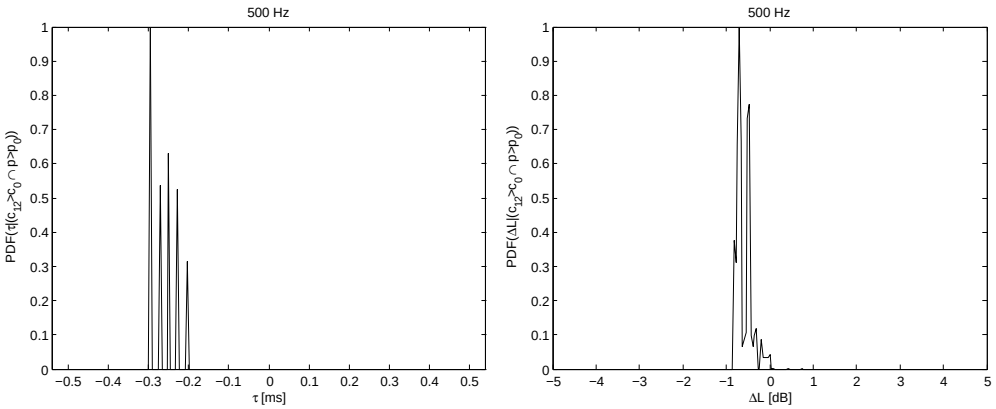

Figura 4.12: Histograma do ITD representado do lado esquerdo e o do ILD apresentado do lado direito, para a versão em Matlab.

Após a selecção de amostras, são calculados os máximos do histograma do ITD, um por cada bloco processado através dos dados obtidos da implementação CPU-GPU

e desta forma adquirir a localização em função da diferença de fase do sinal dos dois microfones. É possível visualizar o padrão de deslocamento do orador, onde ele começa na esquerda, desloca-se para a centro, continua para a direita e finalmente volta ao centro, demonstrado na figura 4.13 os valores reais para os primeiros 21 segundos do teste, sendo o gráfico com o resultado apresentado na figura 4.14.

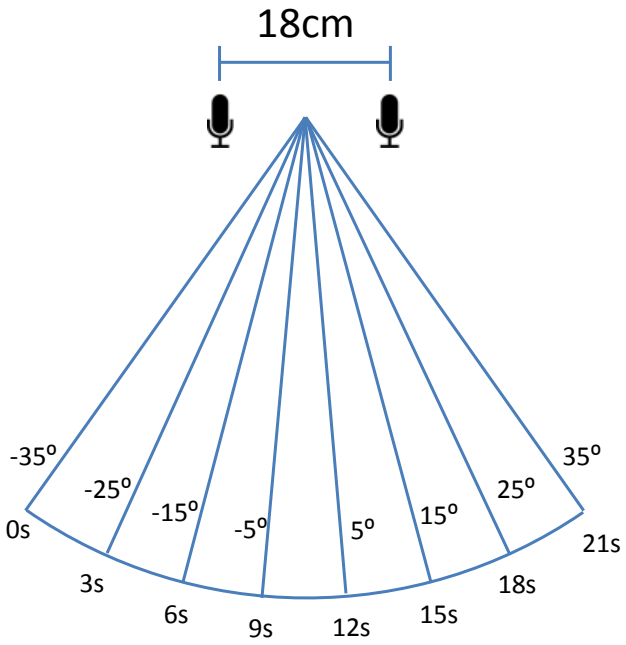

Figura 4.13: Vista superior referente à posição do orador em função do tempo decorrido, de 0 segundos a 21 segundos. A cada 3 segundos o orador desloca-se aproximadamente 15 graus para direita.

Segue-se a formula para calcular o azimute em graus:

$$Azimute(graus) = \frac{90 * ITD_{Hist_{max}}}{ITDmax} \tag{4.1}$$

onde 90 é o azimute máximo em graus, o $ITD_{Hist_{max}}$ o máximo do histograma ITD para cada bloco processado e o $ITDmax = 0.529$.

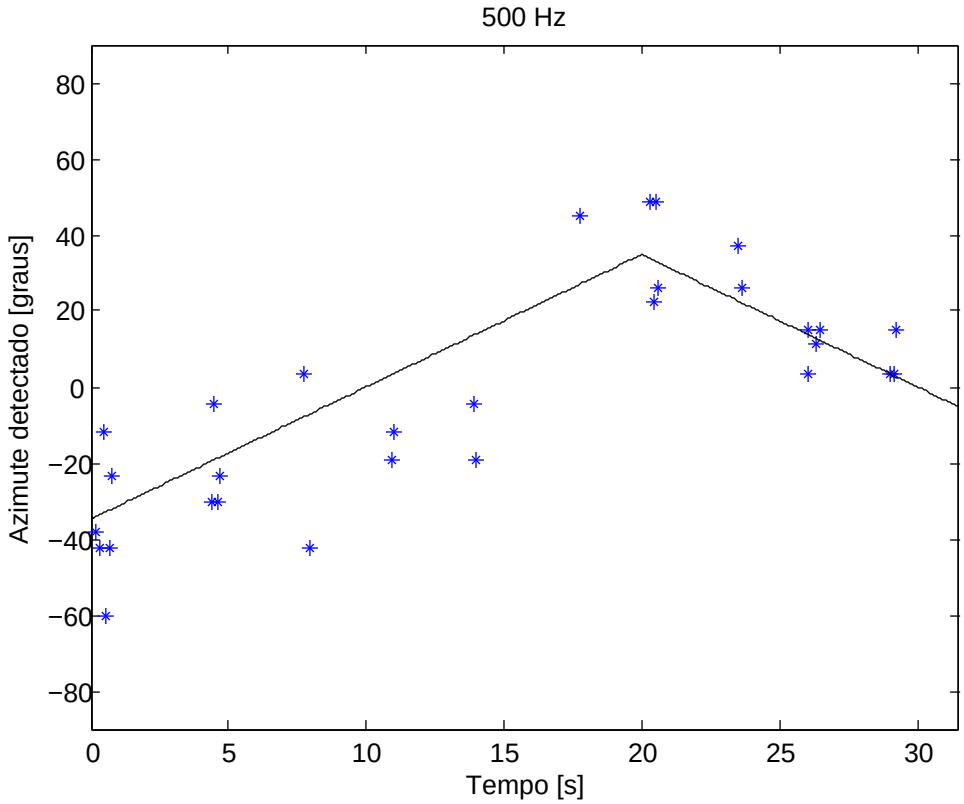

Figura 4.14: Azimutes detectados correspondentes ao máximo dos histograma ITD, por cada bloco processado ao longo do tempo do ficheiro, para os limiares $c_0 = 0.9$ e energia= 0.006. É convertido o máximo do histograma ITD para azimute através da formula 4.1. Os pontos azuis correspondem ao resultado da implementação e a linha preta os valores reais da posição. Verifica-se que o orador desloca-se do lado esquerdo para o centro, do centro para o lado direito e depois volta ao centro.

Da figura 4.15 onde $c_0 = 0.9$ e o limiar de energia= 0.001, verifica-se que com estes valores a precisão do cálculo do ITD diminui, pois mais amostras ultrapassam os limiares, ou seja, vão ser considerados ecos para o cálculo do ITD.

Da figura 4.16 onde $c_0 = 0.998$ e o limiar de energia= 0.001, verifica-se que com estes valores a precisão do cálculo do ITD aumenta. Através deste gráfico verifica-se que o orador fala a cada 3 segundos, começa do lado esquerdo, desloca-se para o centro,

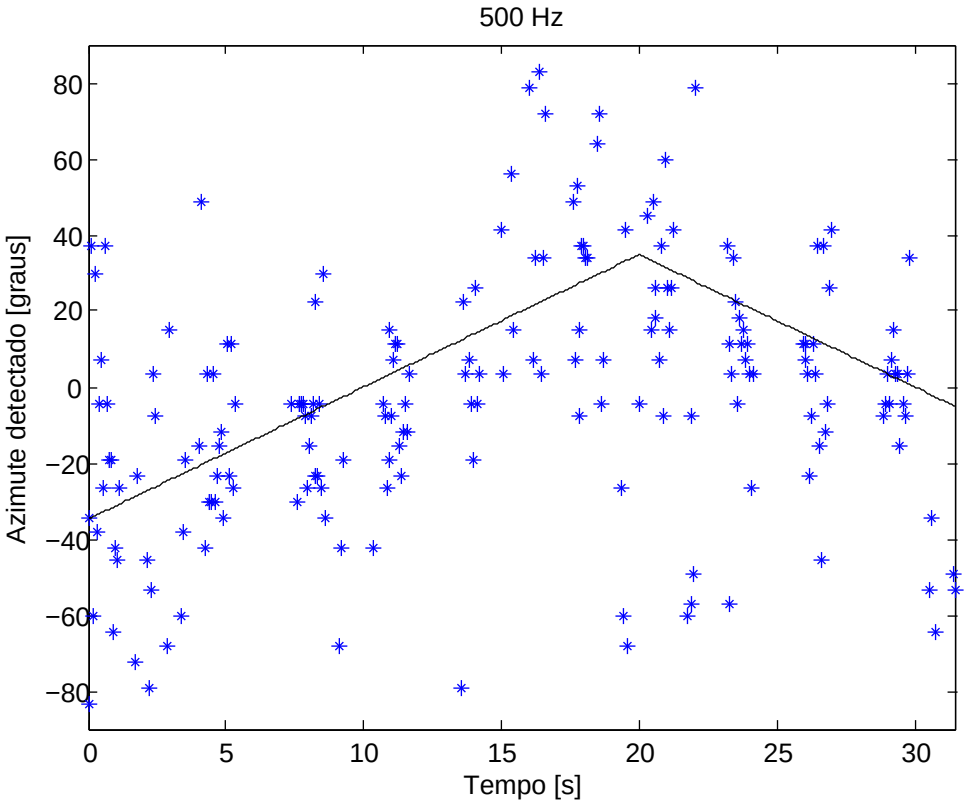

Figura 4.15: Azimutes detectados correspondentes ao máximo dos histograma ITD, por cada bloco processado ao longo do tempo do ficheiro, para os limiares $c_0 = 0.9$ e energia= 0.001. É convertido o máximo do histograma ITD para azimute através da formula 4.1. Os pontos azuis correspondem ao resultado da implementação e a linha preta os valores reais da posição. Com estes valores dos limiares verifica-se que a precisão diminui e vão ser consideradas amostras que representam ecos.

continuando até atingir o lado direito e finalmente volta para o centro, com isto se mostra que os ITD estimados pelo sistema reflectem os valores verdadeiros de azimute, demonstrando assim a robustez do sistema face à posição real do orador.

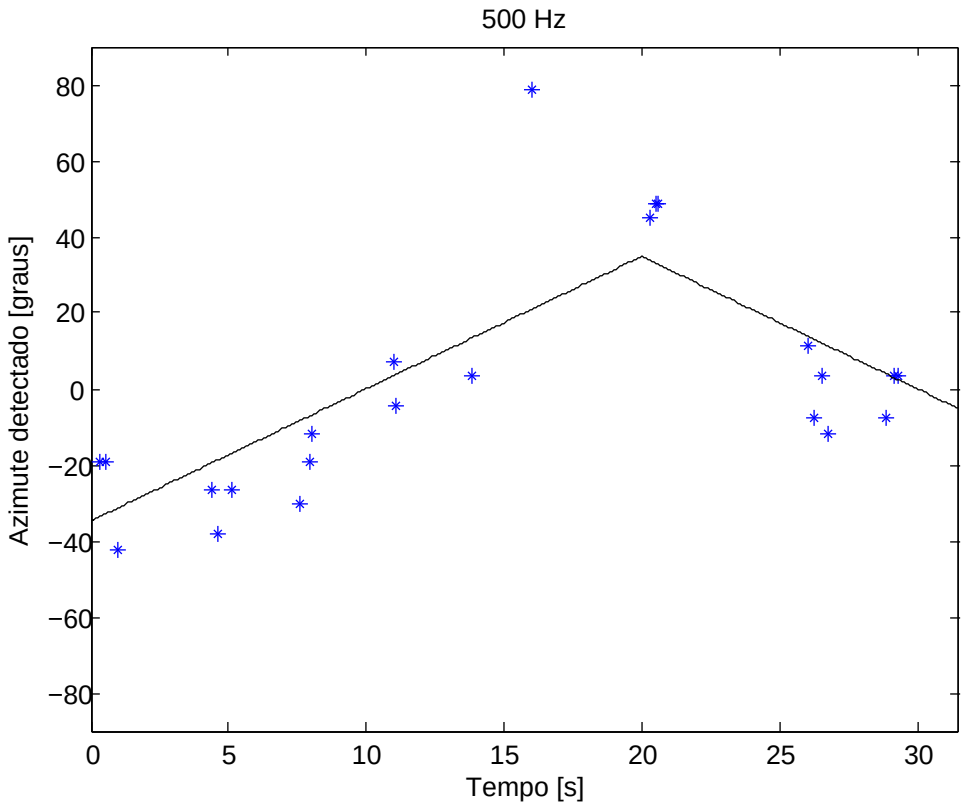

Figura 4.16: Azimutes detectados correspondentes ao máximo do histograma ITD, por cada bloco processado ao longo do tempo do ficheiro, para os limiares $c_0 = 0.998$ e energia= 0.001. É convertido o máximo do histograma ITD para azimute através da formula 4.1. O pontos azuis é o resultado da implementação e a linha preta os valores reais da posição. Com estes valores dos limiares verifica-se que a precisão aumenta e é visível a melhoria no padrão de localização. Esta visualização é a que mais se aproxima dos valores reais, descrevendo o seguinte padrão: o orador fala a cada 3 segundos, começando no lado esquerdo (valores negativos), seguindo para o centro, continuando para a esquerda(valores positivos) e voltando finalmente ao centro.

A figura 4.17 demonstra o padrão da probabilidade dos ILDs em função da frequência para um bloco de processamento. Para determinar os ILDs com frequências superiores

a 500Hz só são considerados os dados provenientes da coerência interaural. É utilizado o limiar $c_0 = 0.6$ para uma frequência de 500Hz com aumentos de 0.022 do limiar c_0 ao longo das frequências, ou seja, atingindo um valor de $c_0 = 0.952$ para a frequência de 5120Hz, pois a filtragem gammatone diminui de amplitude em função da frequência, resultando em ICs mais próximos de um e energias mais próximas de zero. Assim a energia não é considerada em frequências superiores a 500Hz. O intervalo de frequências de 500Hz a 5120Hz foi escolhido devido à proximidade com a frequência da fala humana (60Hz a 7000Hz).

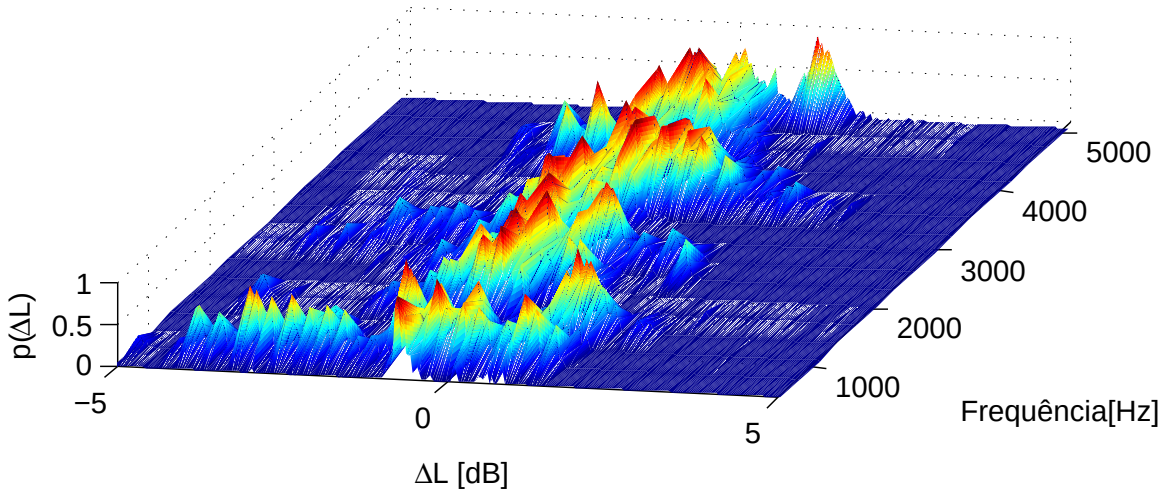

Figura 4.17: Resultado dos ILDs para as 16 bandas de frequências de 500Hz a 5120Hz, para o quinto bloco do ficheiro de 31,9 segundos de áudio. É possível verificar que os ILDs se concentram em torno do centro.

Na figura 4.18 apresentam-se os resultados obtidos para os ILDs com um intervalo de frequências de 500Hz a 5120Hz, para os 31,9 segundos de aúdio.

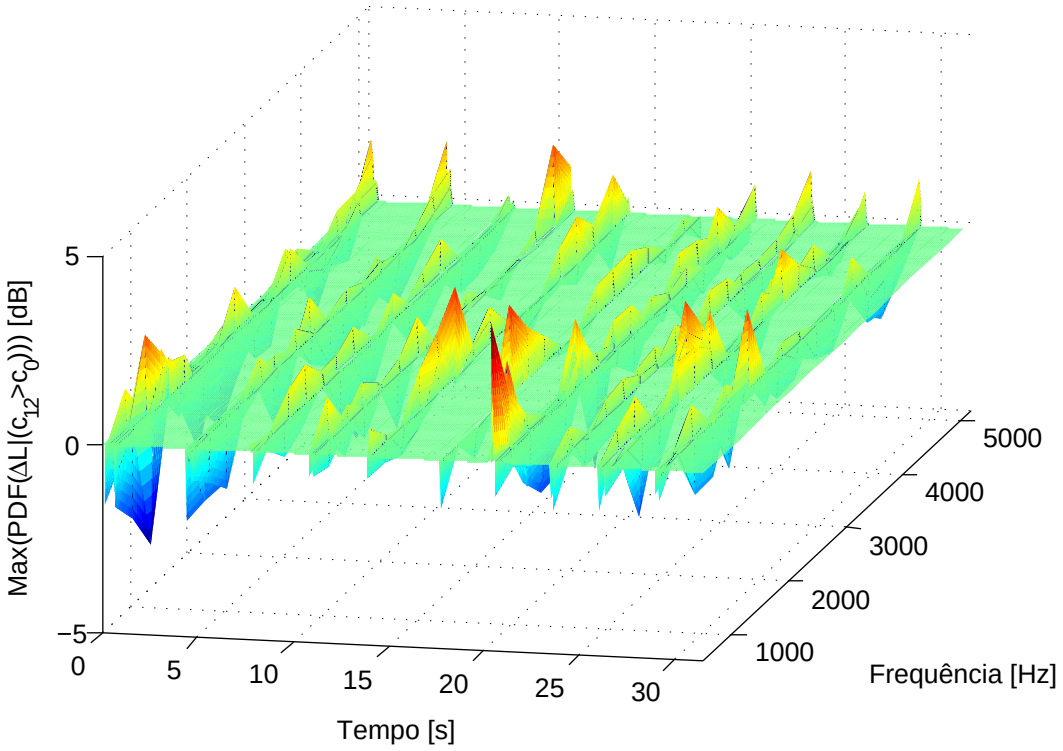

Figura 4.18: Gráfico do máximo de cada um dos histogramas de 16 freuencias ILD em função do tempo decorrido. O resultado dos 16 ILDs com bandas de frequências, de 500Hz a 5120Hz, para os 31,9 presentes no ficheiro de aúdio estéreo. Neste caso os ILDs também possuem valores próximos do centro, revelando assim resultados coerentes em relação à figura 4.17

5

Conclusão e trabalho futuro

Nesta dissertação apresentou-se duas implementações híbridas – com uma FPGA (Field Programming Gate Array) ou CPU, e GPU (Graphic Processing Unit) – de um algoritmo que detecta sinais interaurais para localização espacial de fontes sonoras em cenários acústicos naturais.

Conseguimos assim obter como resultado dados de saída próximos da implementação em Matlab extraindo ainda uma aproximação dos azimutes da posição real orador, através do ITD. Foi considerado o limiar de coerência interaural (c_0) e o limiar de energia na secção dos resultados e discussão para obter o azimute mais aproximado da posição real do orador em determinado instante temporal, fazendo-o de duas formas: mantendo constante a energia, e variando o c_0, e variando a energia e mantendo constante o c_0, resultando num valor para o limiar da coerência interaural $c_0 = 0.998$ e num limiar de energia de 0.001.

A maior dificuldade foi, sem dúvida, conseguir um sistema que possuísse um nível de precisão e de robustez satisfatório, e que atingisse as metas temporais delineadas. Porém, optimizando o código fonte da implementação este problema foi ultrapassado.

É disponibilizado no apêndice A um conjunto de instruções e explicações relativas à implementação Ethernet no dispositivo FPGA e no apêndice B para as comunicações entre o processador embutido Nios II e o hardware da FPGA. No apêndice C encontra-se um guia de utilizador para investigadores que desejem usufruir desta implementação e possam obter ajuda no uso do sistema. Poderão ser efectuadas optimizações ou modificações facilitando assim a integração da implementação em sistemas de maior dimensão.

5. CONCLUSÃO E TRABALHO FUTURO

Em referência às soluções práticas apresentadas de paralelização do modelo de localização. Uma das implementações foi realizada recorrendo a uma FPGA e a uma GPU utilizando a linguagem CUDA, onde se obtiveram resultados mais lentos do que o esperado, não cumprindo os requisitos para atingir tempo real. Na outra implementação recorreu-se a uma CPU e a uma GPU utilizando a linguagem CUDA que se revelou significativamente melhor que a implementação FPGA-GPU, com um resultado em tempo-real para um número de amostras por bloco de processamento superior a 2048. No caso da implementação utilizando uma FPGA e uma GPU propõe-se, como trabalho futuro, a implementação numa FPGA com melhores características que a utilizada neste presente trabalho, tendo uma interface de comunicação de elevado desempenho, como o PCI-Express, e ainda a implementação de acesso directo à memória (DMA) reduzindo assim o peso das comunicações e escritas/leituras no tempo de computação total.

No futuro imediato, propõe-se a ligação do módulo de estimação binaural implementado neste trabalho ao sistema apresentado na figura 1.2, de forma a poder obter um sistema robusto de alto débito de localização espacial binaural em tempo-real. Propõe-se ainda no médio prazo a implementação do modelo desenvolvido por (21), sendo possível localizar e separar fontes sonoras através de modelos Gaussianos da função de densidade de probabilidade fornecidos na solução prática desenvolvida ao longo deste trabalho.

Como trabalho futuro propõe-se o uso de uma GPU NVIDIA com capacidade de computação CUDA superior a 2.0, tendo a implementação de ser alterada para que se obtenha o máximo desempenho através da ocupação de todos os recursos existentes na GPU, já que o limite de *threads* em cada bloco é de 1024, em vez dos 512 *threads* por cada bloco da GPU usada neste trabalho.

Por fim este software será, num futuro próximo, usado de forma integrada em baixo nível num sistema de percepção multissensorial activa para robôs sociais, recorrendo à plataforma IMPEP.

Bibliografia

[1] Altera. Dm9000 driver. *Disponivel em : http://www.alterawiki.com/wiki/EtherNet*, 2011. 29

[2] Altera. Floating-point megafunctions. *Disponivel em : http://www.altera.com/literature/ug/ug_altfp_mfug.pdf*, 2011. 17, 32

[3] Altera. Microc/os-ii real-time operating system. *Disponivel em : http://www.altera.com/literature/hb/nios2/n2sw_nii52008.pdf*, 2011. 29, 32

[4] Altera. Nios ii processor. *Disponivel em : http://www.altera.com/literature/lit-nio2.jsp*, 2011. 17, 29

[5] Altera. Quartus ii development software literature. *Disponivel em : http://www.altera.com/literature/lit-qts.jsp Altera Literature*, 2011. 17

[6] Altera. Simple socket server design example. *Disponivel em : http://www.altera.com/support/examples/nios2/exm-hello_world.html*, 2011. 29

[7] Altera. Sopc builder literature. *Disponivel em : http://www.altera.com/literature/lit-sop.jsp*, 2011. 17

[8] Altera. Stratix iv fpga: High density, high performance and low power. *Disponivel em : http://www.altera.com/products/devices/stratix-fpgas/stratix-iv/stxiv-index.jsp Altera*, 2011. 42

[9] J. Ferreira C. Pinho e J. Dias. Implementation and calibration of a bayesian binaural system for 3d localisation. *In 2008 IEEE International Conference on Robotics and Biomimetics (ROBIO 2008), Bangkok, Tailand*, 2009. 4

[10] B.R. Glasberg e B. C. J. Moore. Derivation of auditory filter shapes from notched-noise data. *Hearing Research*, 1990. 11

[11] R. F. Lyon e C. Mead. Trans . acoustics, speech, and signal processing. *An analog electronic cochlea*, 1988. 10

[12] L. R. Bernstein e C. Trahiotis. The normalized correlation: Accounting for binaural detection across center frequency. *J. Acoust. Soc. Am.*, 1996. 12

[13] C. Faller e J. Merimaa. Source localization in complex listening situations: Selection of binaural cues based on interaural coherence. *The Journal of the Acoustical Society of America*, 2004. 3, 8, 9, 10, 27, 33, 88

[14] A. Handzel e P. Krishnaprasad. Biometric sound-source localization. *In formation Fusion*, 2004. 5

[15] E. Grassi e S. Shamma. A biologically inspired, learning, sound localization algorithm. *Proc. Conference on Information Sciences and Systems*, 2001. 5

[16] Eclipse Foundation. Eclipse framework. *Disponivel em : `http://www.eclipse.org/`*, 2011. 19

[17] T. Hartnstein, R. e Kaiserslautern. Designing embedded processors. netherlands: Springer. *Basics of reconfigurable computing. In: HENKEL, J.; PARAMESWARAN, S.*, 2007. 29

[18] J. Ferreira J. Lobo e J. Dias. Bayesian real-time perception algorithms on gpu — real-time implementation of bayesian models for multimodal perception using cuda. *Real-Time Image Processing*, 2010. 6

[19] Jenkin Kapralos and Milios. Audio-visual localization of multiple speakers in a video teleconferencing setting. *Centre for Vision Research*, 2003. 1

[20] Schnupp King and Doubell. The shape of ears to come: dynamic coding of auditory space. *TRENDS in Cognitive Sciences Vol.5 No.6*, 2001. 1

[21] Michael I. Mandel. Binaural model-based source separation and localization. *Colombia university*, 2010. 58

[22] U. Meyer. Digital signal processing with field programmable gate arrays. *Alemanha: Springer-Verlag*, 2001. 16

[23] B. C. J. Moore. An introduction to the psychology of hearing, fifth edition. *London: Academic Press*, 2004. 11

[24] NVIDIA. Cuda cufft library. *Disponivel em : http://developer.download.nvidia.com/compute/cuda/3_2/toolkit/docs/CUFFT_Library.pdf*, 2010. 33

[25] NVIDIA. Cuda gpus. *Disponivel em : http://developer.nvidia.com/cuda-gpus*, 2011. 22

[26] NVIDIA. Developer zone. *Disponivel em : http://developer.nvidia.com/cuda-downloads*, 2011. 22

[27] NVIDIA. Gpu gems 3 - parallel prefix sum. *Disponivel em : http://http.developer.nvidia.com/GPUGems3/gpugems3_ch39.html*, 2011. 50

[28] NVIDIA. Nvidia cuda compute unified device architecture. *Disponivel em : http://developer.download.nvidia.com/compute/cuda/1_1/NVIDIA_CUDA_Programming_Guide_1.1.pdf*, 2011. 20, 21

[29] B. Sennoner e K. Vehmanen P. Davis, D. Olofson. Jack audio connection kit. *Disponivel em : http://jackaudio.org/ Server and Client*, 2006. 39

[30] J. O. Pickles. An introduction to the physiology of hearing, third edition. *London: Academic Press*, 2008. 11

[31] Victor Podlozhnyuk. Histogram calculation in cuda. *Disponivel em : http://developer.download.nvidia.com/compute/cuda/1_1/Website/projects/histogram256/doc/histogram.pdf Nvidia*, 2011. 38

[32] Meddis R. Simulation of mechanical to neural transduction in the auditory receptor. *The Journal of the Acoustical Society of America*, 1986. 12

[33] D. Zotkin E. Grassi e N. Gumerov. R. Duraiswami, L. Zhiyun. Plane-wave decomposition analysis for spherical microphone arrays. *Proc. to Audio and Acoustics*, 2005. 5

[34] M. J. Hewitt e T. M. Shackleton R. Meddis. Implementation details of a computation model of the inner hair-cell auditory-nerve synapse. *The Journal of the Acoustical Society of America*, 1990. 11

[35] Holdsworth J. P. Rice R. Patterson, I. Nimmo-Smith. An effcient auditory filterbank based on the gammatone function. *Cambridge*, 1987. 9

[36] S Santarelli Shinn-Cunningham, BG and N Kopco. Tori of confusion: Binaural localization cues for sources within reach of a listener. *Journal of Acoustic Soc Am*, 2000. 2

[37] Thrust. A cuda library of parallel algorithms. *Disponivel em : `http://code.google.com/p/thrust/`*, 2011. 36, 38

Apêndice A

Tutorial Ethernet FPGA - Controlador DM9000a

Segue-se um guia para implementar Ethernet na FPGA com o controlador DM9000a, são evidenciados todos os passos em Quartus, SOPC, e NIOS II Eclipse. Para utilizar este guia é necessário obter o ficheiro comprimido que contém todos os ficheiros indispensáveis para a implementação.

Quando concluído o guia, é necessário implementar o cliente remoto, não incluído neste anexo devido à facilidade de implementação. Quando a ligação é estabelecida entre FPGA e *Host*, é possível acender os leds vermelhos a partir do cliente, sendo também possível terminar o servidor(FPGA).

A.1 Quartus

A.1.1 Criar novo projecto

- Criar um projecto em que o caminho das pastas não contenha espaços.

A.1.2 Correr SOPC Builder

- Menu *Tools – SOPC builder.*

A.2 SOPC

- Dar o nome ao projecto SOPC: *NiosSystem.*

- Selecionar *VHDL*.

A.2.1 Criar um processador

- No lado esquerdo ir a *processor*, duplo clique em *Nios II Processor*.

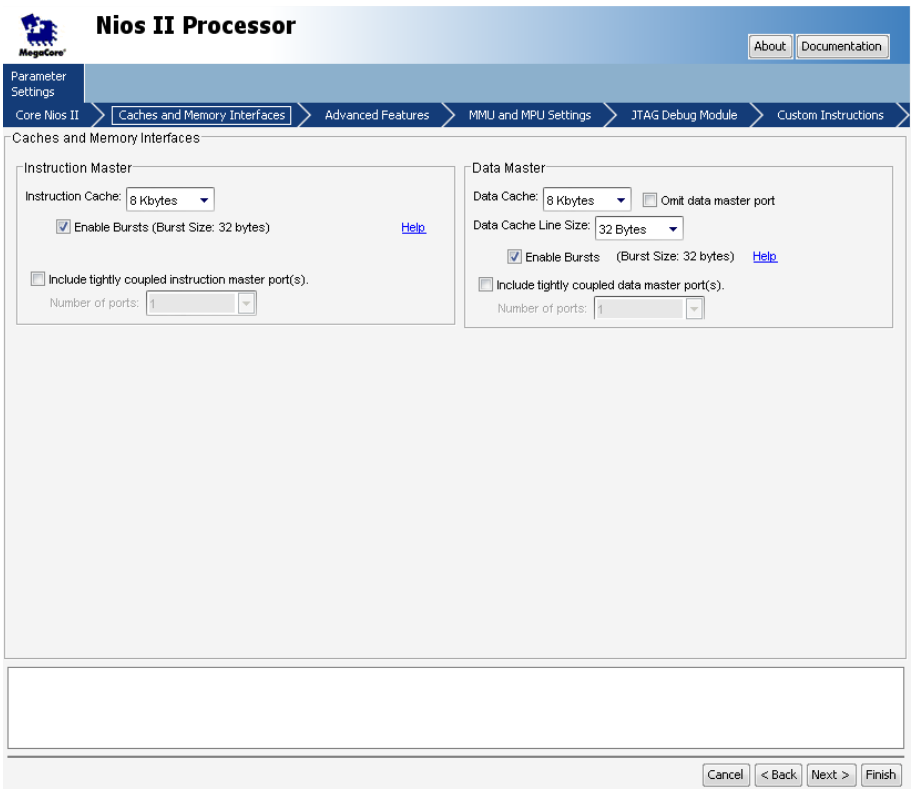

Figura A.1

Passos a seguir na figura A.1.

- Na tabulação *Caches and Memory Interface* alterar os seguintes campos:
- *Instruction Cache*: 8Kbytes.

- Marcar *Enable Bursts*.
- *Data cache*:8Kbytes.
- *Data cache line size*:32Kbytes
- Marcar *Enable Bursts* e clique *Finish*.
- Alterar nome do componente para *cpu*.

A.2.2 Adicionar memória SDRAM

Menu *Memories and Memory controller* – SDRAM – Duplo clique em *SDRAM Controller*:

Passos a seguir na figura A.2.

- Selecionar *Custom* no menu *Presets*.
- Alterar *address with* para 16bits.
- Clique *Finish*.
- Alterar nome do componente para *sdram*.

A.2.3 Conectar memória ao CPU

- Clique duas vezes em CPU.

Passos a seguir na figura A.3.

- Tabulação *Core Nios II* – Em *Reset Vector* e *Exeption Vector*: Selecionar *Sdram*.
- Clique *Finish*.

A.2.4 Adicionar memória Flash

Menu *Memories and Memory controller – Flash – Flash Memory Int* (CFI).
Passos a seguir na figura A.4.

- *Address*: 22bits.
- *Data With*: 8bits.

Figura A.2

Na tabulação *Timing*:

Passos a seguir na figura A.5.

- *Setup*: 40.
- *Wait*: 160.

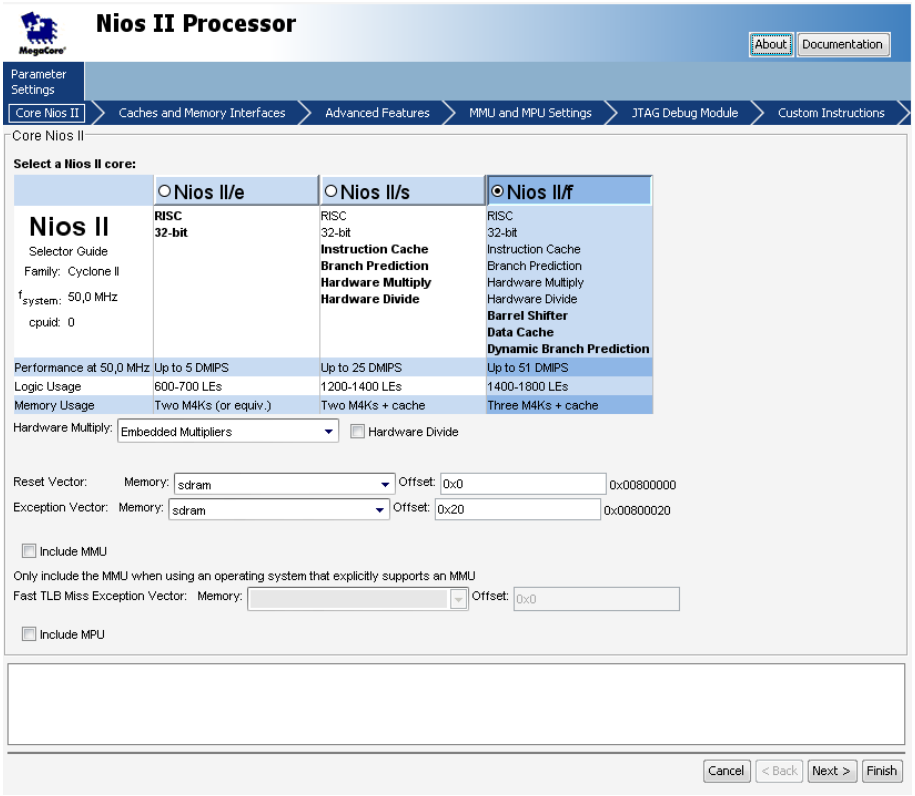

Figura A.3

- *Unit*: ns e clique *Finish*.

No SOPC alterar nome do componente para *ext_flash*.

A.2.5 Adicionar Tristate Bridge

Menu *Bridges and Adapters* – *Memory Mapped* – Duplo clique em *Avalon-MM Tristate Bridge*.

- Clique *Finish*.
- Ligar *Tristate master* ao *Tristate Slave* no *Flash*.
- Alterar nome do componente para *tri_state_bridge*.

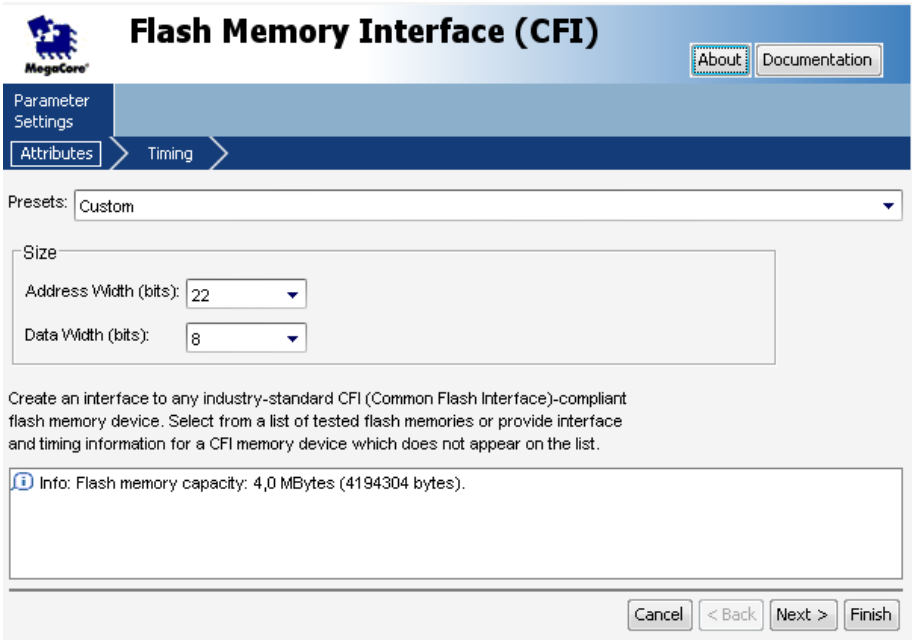

Figura A.4

A.2.6 Adicionar LEDS (Função predefinida da altrera)

Menu *Peripherals* – *Microcontroller Peripherals* – Duplo clique em PIO (*Parallel I/O*).

Passos a seguir na figura A.6.

- Alterar campo *With* para 8 bits.
- Clique *Finish*.
- Alterar nome do componente para *led_pio*.

A.2.7 Adicionar Interface JTAG UART

Menu Interface Protocols – Serial – duplo clique em JTAG UART.

- Clique *Finish*.
- Alterar nome do componente para *jtag_uart*.

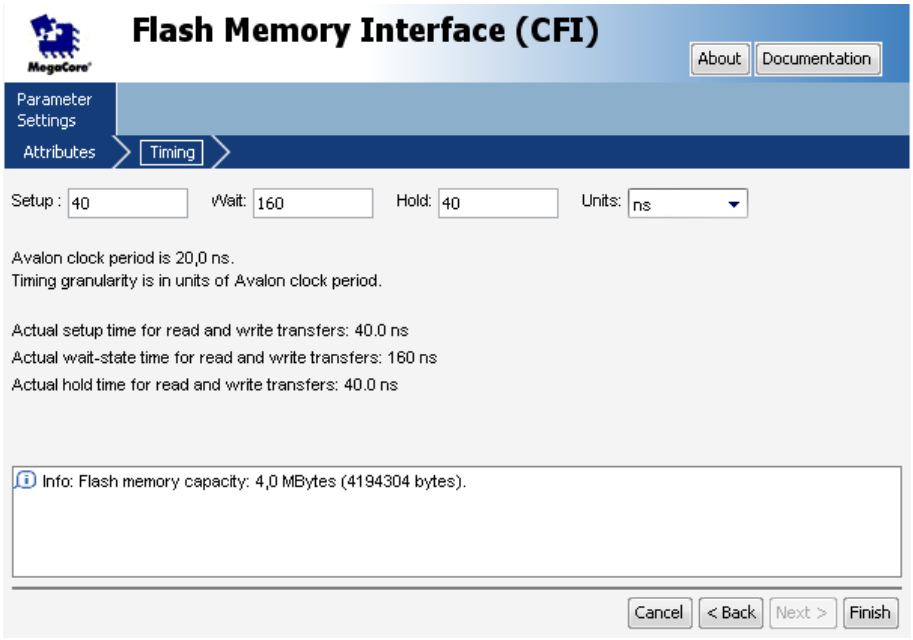

Figura A.5

A.2.8 Adicionar relógio de sistema

Menu *Peripherals* – *Microcontroller Peripherals* – Duplo clique em *Interval Timer*. Passos a seguir na figura A.7.

- Em *Hardware options* – *Presents* – Alterar para *Full-featured*.
- Clique *Finish*.
- Alterar nome do componente para *sys_clk_timer*.

A.2.9 Adicionar relógio de Alta Resolução

Menu *Peripherals* – *Microcontroller Peripherals* – Duplo clique em *Interval Timer*.

- Em *Timeout Period* – Alterar para *10us*.
- *Counter size*: 32
- Em *Hardware options* – *Presents* – Alterar para *Full-featured*.

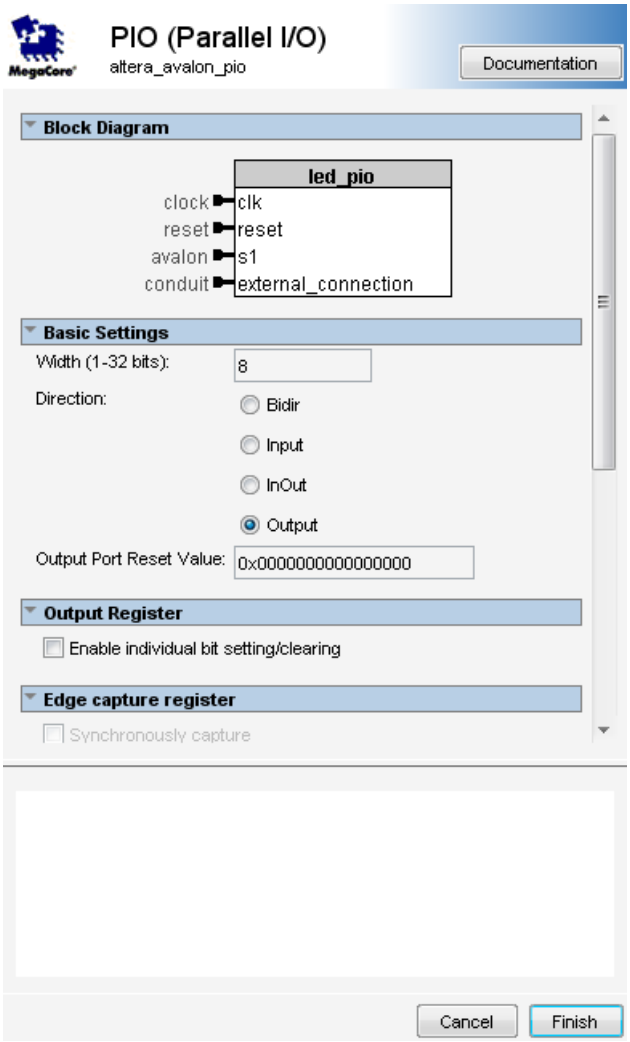

Figura A.6

- Clique *Finish*.
- Alterar nome do componente para *high_res_timer*.

A.2.10 Adicionar controlador de rede

- Copiar todos os ficheiros de *Ethernet FPGA.rar* para a pasta do projecto.
- Duplo clique em *New Component....*
- Menu *File* – *Open* – Escolher o ficheiro *DM9000a.tcl* – *Finish*.

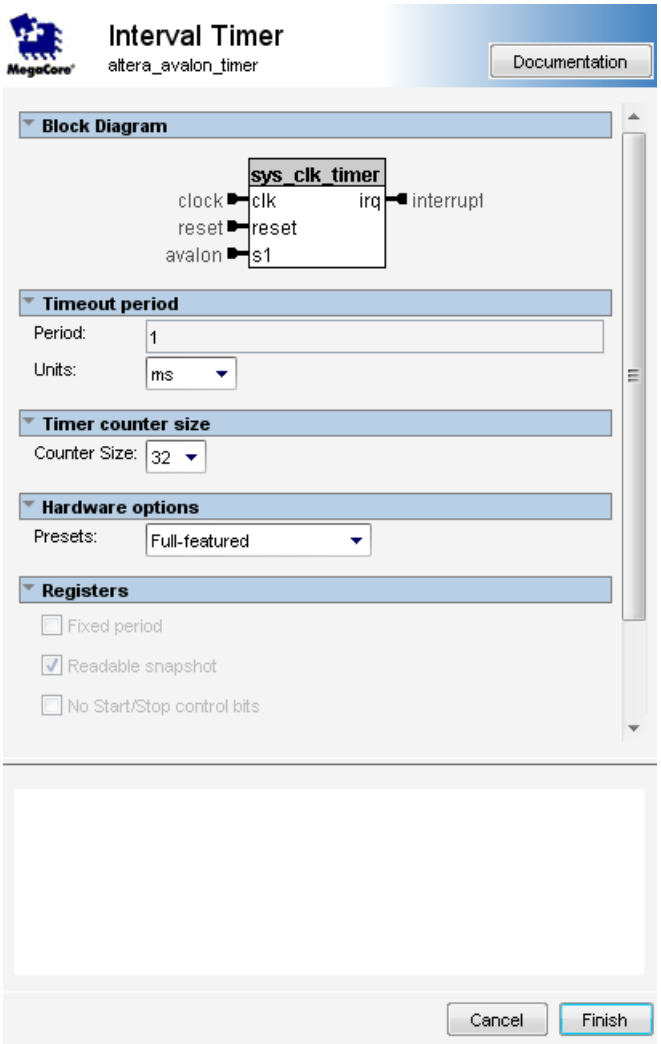

Figura A.7

- Abra o menu *Ethernet*.
- Alterar nome do componente para *dm9000a_inst*.

Ir menu *System* – clique em *Auto Assign Base address* (agora os erros em baixo deverão desaparecer, ficando apenas um *warning* do controlador de rede, devendo este ser ignorado).

- Guardar como *NiosSystem*.
- Clique em *Generate*.

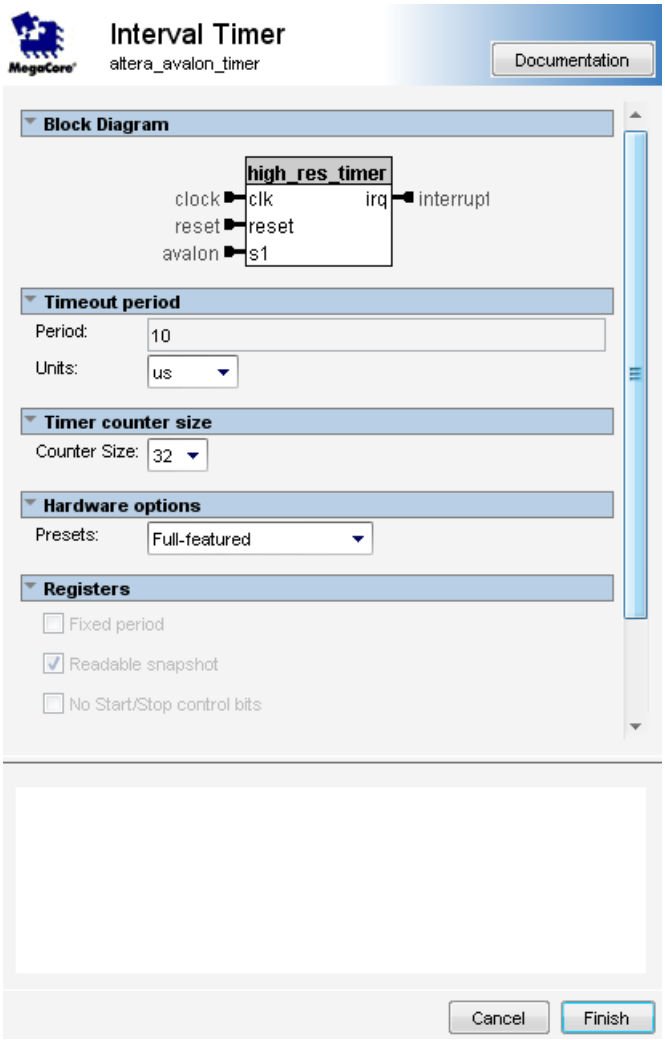

Figura A.8

- Terminar *SOPC builder*.

A.3 Quartus

A.3.1 Adicionar ficheiros ao projecto

- *NiosIIDesign.bdf*.
- *clkgen.v*.
- *bus_to_2pin.vhd*.

- *always_high.vhd.*
- *Reset_delay.v.*
- Definir *NiosIIDesign.bdf* como *Top-Level Entity* – Clique no lado direito em cima do ficheiro *NiosIIDesign.bdf.*

A.3.2 Atrasar fase do relógio 3 nanosegundos

- Criar uma *megafuction*– Menu *Tools – MegaWizard Plug-in Manager.*
- Pasta *I/O – ALT_PLL* - com o nome *altpll0.vhd.*

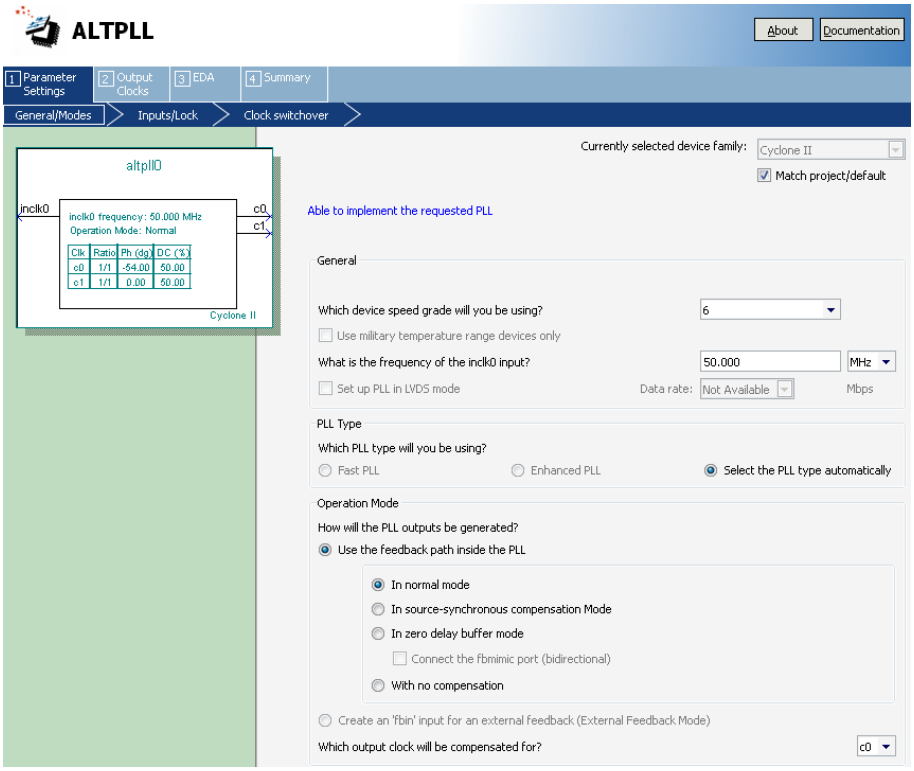

Figura A.9

Passos a seguir na figura A.9.

- Em *General/Modes* colocar *Device speed grade: 6* e *inclk0: 50Mhz.*

- Deixar os outros campos como estão.

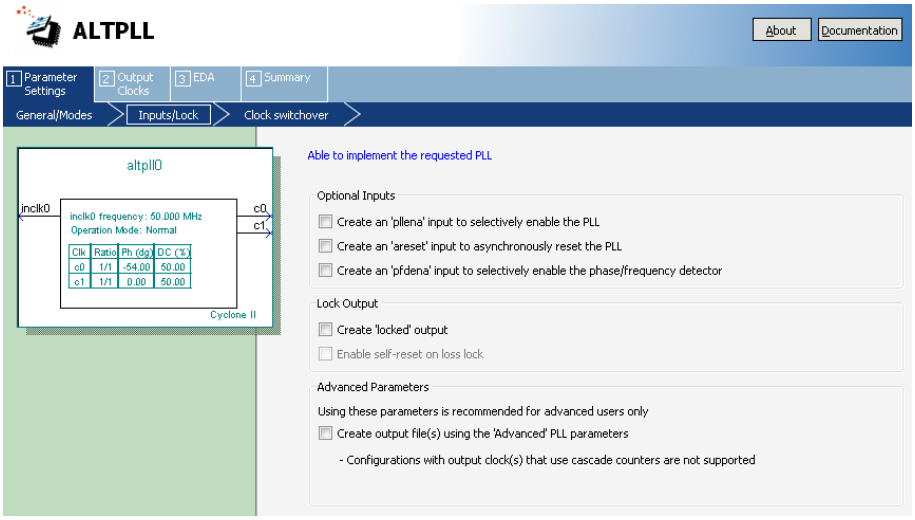

Figura A.10

Passos a seguir na figura A.10.

- No menu *Inputs/Lock* – Desmarcar campos *areset* e *locked*.

Passos a seguir na figura A.11.

- No menu *Output Clocks* em *Clock phase Shift* colocar *-3* e trocar *deg* para *ns*.

Passos a seguir na figura A.12.

- No menu *clk c1* marcar *Use this clock*.
- Clique *Finish*.

Para finalizar a programação do Quartus.

- Importar *pins* do ficheiro fornecido: *Assignments – Import Assignments - DE2_pin_assignments*.
- Compilar projecto.
- Programar FPGA (Aconselhado a programar a FPGA antes de iniciar o Eclipse).

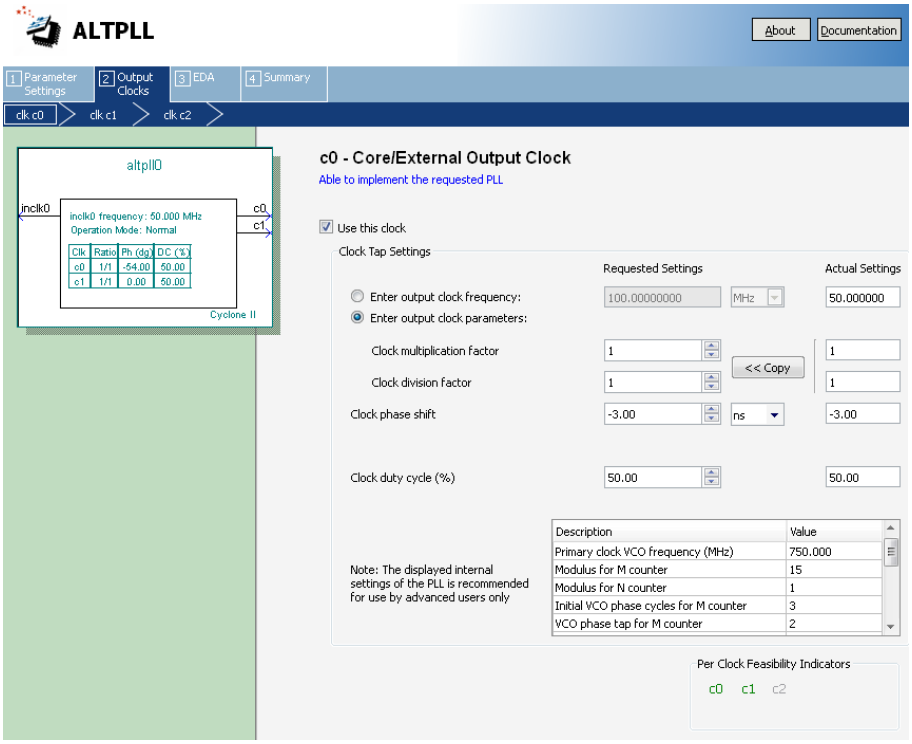

Figura A.11

A.4 Implementação Nios II Eclipse

- Abrir Nios II Eclipse.

A.4.1 Importar Workspace

Menu *File* – *Switch Workspace* – Escolher a pasta do projecto.

- Se as pastas pertencentes ao projectos não derem para abrir A.13.

- Apagar as pastas originadas no *Project Explorer* (apenas no projecto e não fisicamente no HDD).
- Clique no lado direito do rato na área do *Project Explorer*.
- Clique em *Import* A.14.

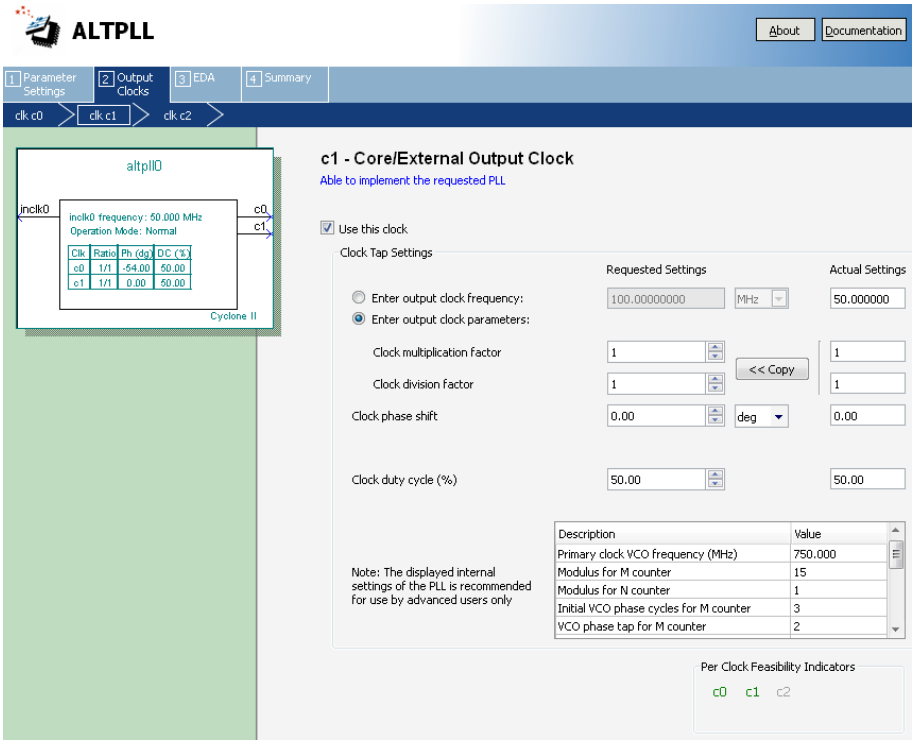

Figura A.12

- Escolher *General – Existing Projects into Workspace* – Escolher a pasta do projecto.

A.4.2 Gerar BSP

(de cada vez que o SOPC é gerado é necessário efectuar este passo)

- Clique lado direito do rato em "*NiosSystem_bsp*" no *Project Explorer*.
- Menu *Nios II*.
- *Generate BSP*.

A.4.3 Limpar dados do projecto e compilar

(pode haver conflitos entre edições de software):

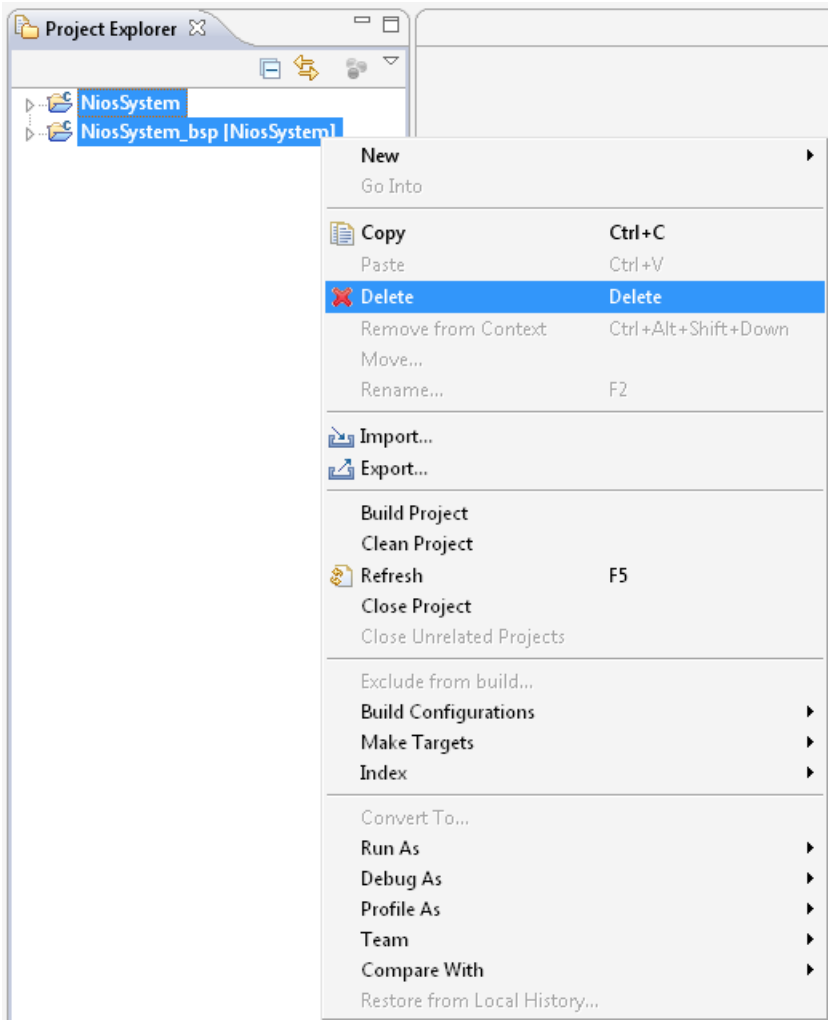

Figura A.13

- Menu *Project*.
- *Clean....*
- *Clean all projects.*
- Clique em *Start a build immediately* e *Build the entire Workspace*.

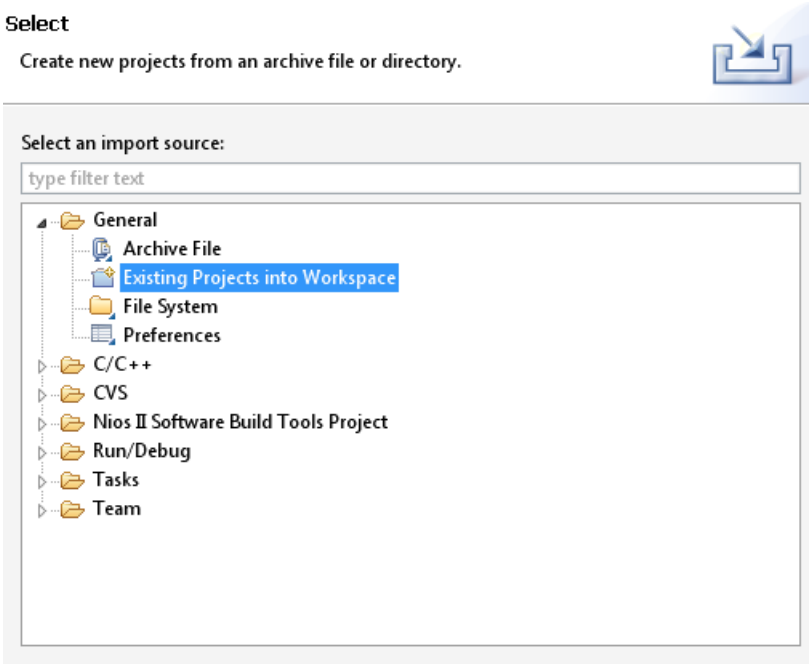

Figura A.14

A.4.4 Programar o processador Nios II

- Menu *Run*.
- *Run Configurations* A.15.
- Duplo Clique em *Nios II Hardware*.
- Em *Project Name* escolher o projecto
- Na tabulação *Target Connection* clique em *Ignore mismatched system ID* e em *Ignore mismatched system timestamp*.
- Clique *Run*.

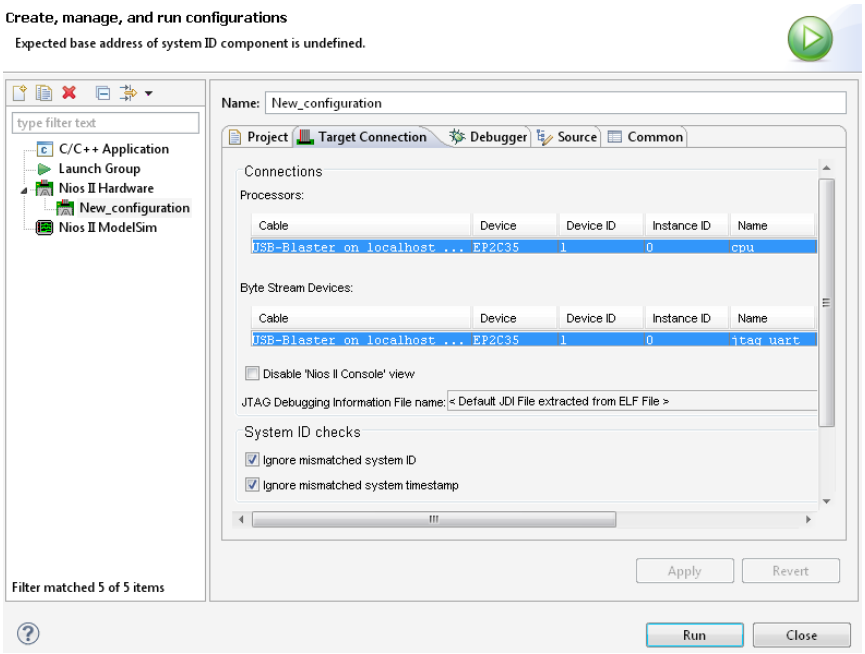
Create, manage, and run configurations
Expected base address of system ID component is undefined.
type filter text
C/C++ Application
Launch Group
Nios II Hardware
New_configuration
Nios II ModelSim
Filter matched 5 of 5 items
Name: New_configuration
Project
Target Connection
Debugger
Source
Common
Connections
Processors:
Cable
Device
Device ID
Instance ID
Name
USB-Blaster on localhost ... EP2C35 1 0 cpu
Byte Stream Devices:
Cable
Device
Device ID
Instance ID
Name
USB-Blaster on localhost ... EP2C35 1 0 jtag_uart
Disable 'Nios II Console' view
JTAG Debugging Information File name:
< Default JDI File extracted from ELF File >
System ID checks
Ignore mismatched system ID
Ignore mismatched system timestamp
Apply
Revert
Run
Close

Figura A.15

Apêndice B

Tutorial Integrar código no SOPC builder

Neste apêndice é abordado o método de conecção entre software NIOS II e hardware. Incluído também um exemplo que liga os leds vermelhos sequencialmente da esqerda para a direita e da direita para esquerda, através de software, mais especificamente um ciclo. Este apêndice é dependente do apêndice anterior para bom funcionamento.

B.1 SOPC builder

Segue-se um exemplo para acender os 18 leds vermelhos em software.

B.1.1 Abrir SOPC builder

- Duplo clique em *New component....*
- Na tabulação *HDL Files*, clique em *Add...* e selecione o ficheiro *User_Logic_Leds.v*.
- Para adicionar uma ligação externa ao CPU - Na tabulação *Signals* - coluna *Interface*, clique em *New conduit...* pertencente ao *acendeled*.
- Em *Signal type* selecione *export*.
- Clique em *Finish* e guarde o componente.

B.1.2 Adicionar Componente de Leds

- Duplo clique em *User_Logic_Leds*.
- No menu *System* – clique em *Auto Assign Base address*.

B.1.3 Gerar código

- Clique em *Generate*.
- Termine o *SOPC builder*.

B.2 Quartus

B.2.1 Substituir ficheiro *NiosIIfDesign*.

Passos para eliminar *NiosIIfDesign* do projecto.

- Eliminar *NiosIIfDesign* na pasta do projecto e alterar nome do ficheiro. *NiosIIf-DesignLed* para *NiosIIfDesign*.
- Adicionar *NiosIIfDesign* ao projecto no Quartus.
- Programar FPGA.

B.3 Nios II Eclipse

B.3.1 Gerar BSP

- Clique lado direito do rato em *NiosSystem_bsp* no *Project Explorer*.
- Menu *Nios II*.
- *Generate BSP*.

B.3.2 Integrar código para acender LEDS

- Abrir ficheiro *iniche_init.c*.
- Na função *main* adicionar o código B.1.
- Guardar alterações.

- Programar o processador *Nios II*.
- Menu *Run*.
- Duplo clique em *Nios II Hardware*.
- Escolher o projecto.
- Na tabulação *Target Connection* clique em *Ignore mismatched system ID* e em *Ignore mismatched system timestamp*.
- Clique *Run*.

```
/*Liga Leds sequencialmente da esquerda para a direita, e direita para
esquerda*/

#include "io.h"

alt u32 cont;
alt_u32 flag;
cont=0;
flag=0;

while(1){
      IOWR(USER_LOGIC_LEDS_0_BASE, 0x0 ,cont);
      if(flag==0){
            cont++;
      }
      if(cont==17){
            flag=1;
      }
      if(flag==1){
            cont--;
      }
      if(cont==0){
            flag=0;
      }

      usleep(40000);
}
```

Figura B.1

Apêndice C

Guia de utilização do módulo de cálculo de ITD e 16 ILDs

C.1 Descrição do programa

Primeiramente é perguntado ao utilizador qual o tamanho do bloco em amostras, quais os valores dos limiares c_0 e de energia, o ITD e ILD máximos e a frequência de cada uma das 16 bandas. Em seguida o programa carrega as amostras correspondentes ao microfone esquerdo e direito. Como saída obtém-se um histograma do ITD, 16 histogramas dos ILDs e os correspondentes máximos dos histogramas, resultando em um ITD e 16 ILDs. Finalmente é gravado em ficheiro no disco rígido do computador de forma a ser futuramente tratado por outra aplicação. Este guia é referente à implementação CPU-GPU.

C.2 Estrutura geral

O programa está dividido em duas partes diferentes, cada uma com a sua tarefa específica. Está escrito em C/C++ para ser usado em conjunto com biblioteca de leitura das amostras dos microfones JACK e a API CUDA.

C.2.1 CUDA

Todos os cálculos necessários para executar o algoritmo são consumados na GPU para que se obtenha o ITD e os 16 ILDs finais.

C.3 Hardware necessário

- GPU capaz de processar CUDA com pelo menos *compute capability* > 1.1

C.4 Software necessário

C.4.1 Sistema Operativo

O programa foi desenvolvido no sistema operativo Ubuntu 10.04 32 bits, ou seja, a versão pré-compilada é para ser usada neste ambiente. É possível compilar para uma versão de 64 bits, todavia o ficheiro "CMakeLists.txt" tem de ser alterado para incluir os caminhos das bibliotecas CUDA necessárias para a sua compilação.

C.4.2 Software específico CUDA

- É necessário ter instalado o pacote de desenvolvimento CUDA com uma versão superior à 3.0. Na versão pré-compilada é usada a versão 3.2.

- Biblioteca CUDA Thrust, que se encontra por defeito instalada nas versões CUDA superiores à 4.0.

C.4.3 Software adicional

- CMake ¿ 2.6 - é necessário para efectuar a compilação caso o utilizador pretenda efectuar mudanças ao *software* ou não possua o sistema operativo Ubuntu 10.4 32 bits.

C.5 Guia de utilização do executável

- Se o utilizador não estiver a executar o sistema operativo Ubuntu 10.4 32 bits, o ficheiro "CMakeLists.txt" tem de ser alterado de forma a incluir os caminhos correctos das bibliotecas CUDA.

Executar os seguintes comandos para compilar:

- "cmake";
- "make".

Para correr o programa basta executar o comando "./BinauralCues".

C.6 Guia de integração em software de terceiros

Para que seja possível integrar o cálculo do ITD e dos ILDs em GPU, deve ter-se em conta os seguintes ficheiros para determinadas amostras de entrada: "histogram256.cu", e "BinauralCues.cu".

- `InitCuda(int np,`
  `float *Critical_Frequency,`
  `float maxitd,`

A função `InitCuda` aloca as variáveis e inicializa os filtros em CPU e GPU, em que `np` é o número de amostras por bloco, `Critical_Frequency` as 16 bandas de frequência, `maxitd` o ITD máximo.

- `ShutdownCUDA();`

  A função `ShutdownCUDA` liberta as variáveis alocadas inicialmente em CPU e GPU.
  A função `BinauralCues` recebe como parâmetros:

- `BinauralCues(int np,`
  `float *Micro_in1,`
  `float *Micro_in2,`
  `float *ild,`
  `float *itd,`
  `float ThresholdIC,`
  `float ThresholdPOWER,`
  `float maxitd,`
  `float maxild,`
  `int *hist_itd,`
  `int *hist_ild1,`
  `int *hist_ild2,`
  `int *hist_ild3,`
  `int *hist_ild4,`
  `int *hist_ild5,`

```
int *hist_ild6,
int *hist_ild7,
int *hist_ild8,
int *hist_ild9,
int *hist_ild10,
int *hist_ild11,
int *hist_ild12,
int *hist_ild13,
int *hist_ild14,
int *hist_ild15,
int *hist_ild16);
```

Em que o `maxild` o ILD máximo, o `Micro_in1` corresponde à entrada do microfone direito, o `Micro_in2` é a entrada do microfone esquerdo, o `ild` é um vector de 16 máximos dos 16 histogramas ILD, o `itd` é o máximo do histograma ITD, `ThresholdIC` é o limiar c_0, `ThresholdPOWER` é o limiar de energia, `hist_itd` é o histograma ITD de resolução 256 e os 16 histogramas ILD correspondem ao `*hist_ild1` até `*hist_ild16`.

C.7 Guia de acréscimo de extensões

Para que o utilizador possa acrescentar novas características ao programa ou manipular as já existentes, na implementação proposta existem um núcleo de funções que têm de ser cumpridas para que seja seguida a linha do modelo proposto por Faller e Merimaa (13).

- `Gammatone(int np, float *Micro_in1, float *Micro_in2,`
  `float *in1Gamma, float *in2Gamma,`
  `float *gain, float *A0, float *A2,`
  `float *B0, float *B1, float *B2,`
  `float *A11, float *A12, float *A13, float *A14);`

Nesta função é efectuada a filtragem gammatone para as 16 bandas de frequência, onde o `np` é numero de amostras a processar, o `Micro_in1` corresponde à entrada

do microfone direito, o `Micro_in2` é a entrada do microfone esquerdo, `in1Gamma` e `in2Gamma` é saída para as 16 bandas de frequências do microfone esquerdo e direito, do `gain` até ao `A14` são os coeficientes do filtro para 16 bandas pré-calculadas na função `InitCUDA`, sendo vectores de 16 elementos.

- `Neural_Cp(cu_in1,cu_in2,in1_2,fft_batch,np);`

- `Neural_Cp(cu_in1,cu_in2,in1_2,np);`

A função `Neural_Cp` converte as saídas do banco de filtros gammatone em variáveis complexas (`fft_batch`) replicando o seus valores para outra variável (`in1_2`) de forma a ser usado na compressão em envelope.

- ```
  cufftExecC2C(plan32FFT, (cufftComplex *)fft_batch,
  (cufftComplex *)fft_batch, CUFFT_FORWARD);
  ```

Seguidamente é realizada a FFT de 32 bandas (16 bandas do microfone direito e 16 bandas do microfone esquerdo) em paralelo através da biblioteca CUFFT.

- ```
  Neural_ComplexPointwiseMulAndScale(fft_batch,
  np, 0.373f / np);
  ```

Quando realizada a FFT é feita a multiplicação ponto a ponto no domínio da frequência e normalizado o resultado.

- ```
  cufftExecC2C(plan32FFT, (cufftComplex *)fft_batch,
  (cufftComplex *)fft_batch, CUFFT_INVERSE);
  ```

Após a multiplicação ponto a ponto é realizada a FFT inversa, obtendo-se a transformada de Hilbert.
  ```

- `Neural_ABS(fft_batch,np);`

Sendo posteriormente adquirida a magnitude.

- `Neural_CompressEnvelope(fft_batch, in1_2,np);`

Depois de concluída a extracção da magnitude é realizada a compressão em envelope com a realimentação do sinal proveniente do banco de filtros gammatone `in1_2`.

- `Neural_RectifyEnvelope(fft_batch,np);`

O envelope é rectificado através da função `Neural_RectifyEnvelope`.

- `Neural_PowerOfTwo(fft_batch,np);`

À saída do envelope rectificado é aplicado o aumento de energia quadrático.

- ```
  cufftExecC2C(plan32FFT, (cufftComplex *)fft_batch,
  (cufftComplex *)fft_batch, CUFFT_FORWARD);
  ```

À saída do aumento de energia quadrático é realizada a conversão para o domínio da frequência.

- ```
  Neural_ComplexPointwiseMulAndScale(fft_batch,
  d_filter_kernel2, np, 0.373f / np);
  ```

É feita a multiplicação ponto a ponto com os coeficientes do filtro passa-baixo.

- ```
  cufftExecC2C(plan32FFT, (cufftComplex *)fft_batch,
  (cufftComplex *)fft_batch, CUFFT_INVERSE);
  ```
  ```

- `Neural_ABSFinal(cu_in1,cu_in2,fft_batch,np);`

Para terminar a tradução neuronal é aplicada a FFT inversa e extraído o valor absoluto do seu resultado.

- `Processor_IC1(cu_num, cu_den1, cu_den2,`
  `maxitds, alpha3, N2, np);`

- `Processor_ILD1(cu_lev1_1, cu_lev2_1,`
  `cu_in1, cu_in2,`
  `maxitds, alpha2, N2, np);`

Realizada a preparação para a soma de prefixos, é efectuada a soma de prefixos através da função `inclusive_scan_by_key` da biblioteca *Thrust*.

- `Processor_IC2(cu_numRes, cu_den1Res,`
  `cu_den2Res,cu_ic,cu_itd,index_ild, maxitds, sfreq, N2);`

Após a soma de prefixos é realizada a extracção do ITD e dos 16 ILDs.

- `Processor_CueSelection(cu_pow,cu_itd,cu_ic,`
  `ThresholdIC,ThresholdPOWER,N2);`

- `Processor_CueSelection(cu_pow,cu_ild,`
  `cu_ic,ThresholdIC,N2);`

Depois de adquiridos os vectores ITD e ILD é realizada a selecção de amostras através dos limiares c_0 e do limiar de energia.

- `Processor_float_to_int_ITD(cu_itd,`
  `cu_itdINT,maxitd,N2);`

- `Processor_float_to_int_ILD(cu_ild,`
  `cu_ild1INT,`
  `cu_ild2INT,`
  `cu_ild3INT,`
  `cu_ild4INT,`
  `cu_ild5INT,`
  `cu_ild6INT,`
  `cu_ild7INT,`
  `cu_ild8INT,`
  `cu_ild9INT,`
  `cu_ild10INT,`
  `cu_ild11INT,`
  `cu_ild12INT,`
  `cu_ild13INT,`
  `cu_ild14INT,`
  `cu_ild15INT,`
  `cu_ild16INT,`
  `maxild,N2);`

Após a aplicação dos limiares são convertidas a variáveis de `float` para `int` para realizar os histogramas.

Depois é efectuado o cálculo dos histogramas do ITD e dos 16 ILDs, com recurso à biblioteca *histogram*.

Finalmente são calculados os máximos dos histogramas através a biblioteca *Thrust*, converte-se um valor de ITD e os 16 valores ILDs para float. O ITD e os 16 ILDs e os seus histogramas são copiados da memória da GPU para a memória RAM.