CerVANTES

Co-VAlidatioN Tool for Embedded Systems

Autor: José Cabrita

Orientadores: José Costa e Paulo Flores

INESC-ID, grupo ALGOS

Índice

1 Objectivo

Este projecto tem como objectivo a implementação de um algoritmo de cancelamento de eco num sistema que permita uma implementação mista, com software e hardware, e proceder à descrição desse sistema em SystemC. Para a implementação deste sistema foi seleccionada uma placa ATLYS da Digilent que contêm uma FPGA Spartan-6 e os dispositivos necessários para a recepção e processamento de dados áudio.

2 Introdução

Num sistema com recepção e transmissão de áudio, o eco ocorre quando o sinal áudio recebido é reflectido no ambiente e retransmitido em conjunto com o sinal de áudio transmitido. Esta ocorrência origina uma emissão do sinal de áudio transmitido somado com réplicas do sinal áudio recebido atrasadas e atenuadas o que pode degradar a qualidade do sinal áudio transmitido.

O objectivo de um cancelador de eco é eliminar as réplicas do sinal de áudio recebido sem alterar o sinal a transmitir. Para que o cancelador se adapte ao ambiente onde o sistema está inserido e a qualquer alteração do mesmo será utilizado um sistema com um filtro adaptativo representado na figura 1. A função do filtro adaptativo é gerar, a partir do sinal de áudio recebido, réplicas do sinal para que, quando é subtraído ao sinal a ser transmitido, anule as réplicas geradas pelo ambiente. Para isso é necessário um período de treino para que o filtro se adapte ao eco gerado pelo ambiente, durante este período de treino não deverá ser enviado outro sinal além das réplicas geradas pelo ambiente.

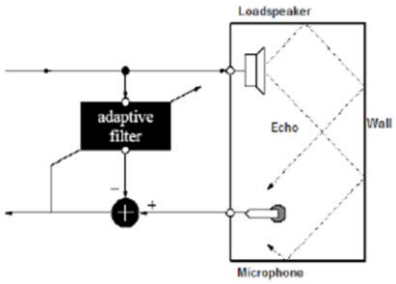

Figura 1 – Esquema do cancelador de eco.

3 Algoritmo do cancelador de eco

O algoritmo utilizado para implementar o cancelador de eco é baseado no algoritmo Least Mean Squares (LMS) como o apresentado em [1]. Este algoritmo permite a adaptação do comportamento do filtro utilizando amostras do resultado desejado. O sistema descrito na figura 1 tem dois modos de funcionamento, o primeiro modo ocorre quando temos um sinal de áudio a transmitir, neste caso será transmitido o sinal de áudio somado com as réplicas do sinal de áudio recebido e o filtro actua na eliminação dessas réplicas, não havendo alteração dos coeficientes do filtro. No segundo modo não existe sinal de áudio a transmitir, ou seja, o sinal transmitido será apenas as réplicas do sinal de áudio recebido. Neste caso o sinal resultante após a actuação do filtro para eliminar as réplicas será um sinal de erro que representa a adaptação do comportamento do filtro ao ambiente, este sinal de erro é então utilizado para corrigir os coeficientes do filtro de modo a garantir uma melhor adaptação de maneira a garantir que o sinal de erro tem um valor médio o mais baixo possível.

No modo de adaptação do filtro são necessários 3 passos, cada vez que é recebida uma amostra do sinal de áudio, para aplicar o algoritmo. O primeiro passo é calcular a saída do filtro, o segundo passo é comparar o resultado na saída do filtro com a amostra do sinal a transmitir (neste caso apenas as réplicas, geradas pelo ambiente, do sinal de áudio recebido) obtendo assim o sinal de erro, no terceiro passo este sinal é utilizado na correcção dos coeficientes. Como demonstrado em [1] o algoritmo a aplicar é o seguinte:

Passo 1:

Cálculo da saída do filtro

$$y(n) = \sum_{i=0}^{N-1} w(i)x(n-i)$$

Onde y(n) representa a saída do filtro, N representa o numero de coeficientes do filtro, w representa o vector de coeficientes do filtro e x(n) representa o vector de amostras do sinal de áudio recebido.

Passo 2:

Cálculo do sinal de erro

$$e(n) = d(n) - y(n)$$

Onde e(n) representa o sinal de erro, y(n) a saída do filtro e d(n) o sinal desejado, neste caso o sinal com as réplicas do sinal de áudio recebido gerado pelo ambiente.

Passo 3:

Actualização dos coeficientes, este passo é efectuado utilizando uma das 2 equações seguintes para cada coeficiente. A equação (1) utiliza uma multiplicação do valor da amostra, correspondente ao coeficiente, pelo valor do erro calculado, para evitar esta multiplicação é

efectuada uma alteração ao algoritmo onde é substituída a multiplicação por uma troca de sinal do valor do erro de acordo com o sinal da amostra correspondente ao coeficiente.

$$w(n) = w(n) + 2\,\mu\,e(n)x(n) \qquad (1)$$

$$w(n) = w(n) + 2\,\mu\,e(n)\,sinal\big(x(n)\big) \qquad (2)$$

Nas equações, w(n) representa um coeficiente, e(n) o sinal de erro, x(n) a amostra correspondente ao coeficiente e μ representa um valor de acerto do algoritmo que depende do número de coeficientes utilizados no filtro e dos valores das amostras do sinal de áudio recebido. Neste algoritmo o valor de μ é estimado utilizando valores máximos de amostras de entrada e o número de coeficientes desejado para o filtro. Para efeitos de utilização em *hardware*, o valor de μ é calculado com divisões por 2, pois em hardware esta operação pode ser efectuada com um deslocamento de uma posição dos bits de uma variável para a direita.

Para verificação de resultados foi desenvolvida uma aplicação, em linguagem de programação C, que replica o funcionamento do cancelador de eco, nesta aplicação, e para efeitos de teste do sistema, o eco é gerado digitalmente considerando que o seu valor seria correspondente a um atraso e uma atenuação do sinal de áudio recebido. Assim a saída do filtro está sempre em adaptação e o objectivo é adaptar-se ao eco gerado.

Os resultados apresentados na figura 2 foram obtidos com a correcção de coeficientes (1) e com os seguintes parâmetros, filtro FIR com 50 coeficientes, eco com 25 amostras de atraso e 0.5 de atenuação e o valor de acerto do algoritmo é 0.0625. O valor absoluto médio do erro calculado é de 150.23 para este conjunto de amostras, ou seja, comparando com o valor absoluto das amostras (32768) o erro é 0.45%.

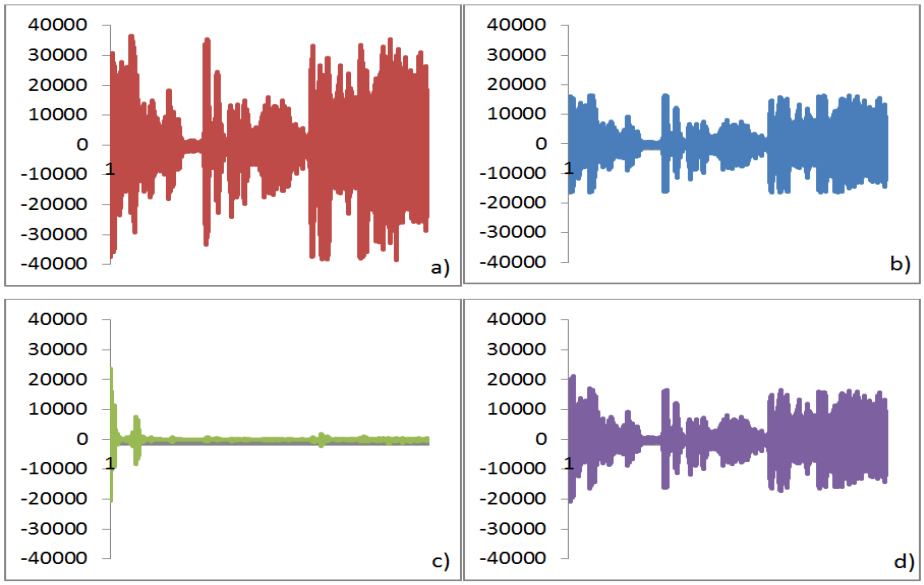

Figura 2 – a) gráfico do som original, b) gráfico do eco gerado com 25 amostras de atraso e 0.5 de atenuação, c) gráfico do sinal de erro calculado pela subtracção da saída do filtro ao eco gerado, d) gráfico da saída do filtro

Os resultados da figura 3 foram obtidos com a correcção de coeficientes (2) e com os seguintes parâmetros, filtro FIR com 50 coeficientes, eco com 25 amostras de atraso e 0.5 de atenuação e o valor de acerto do algoritmo é 0.03125.O valor de acerto foi reduzido, pois o valor 0.0625 é um passo de correcção demasiado grande para que a saída do filtro faça a convergência para o sinal de áudio. Neste caso o valor do erro médio absoluto é de 175.96 o que corresponde a 0.54% do valor absoluto das amostras.

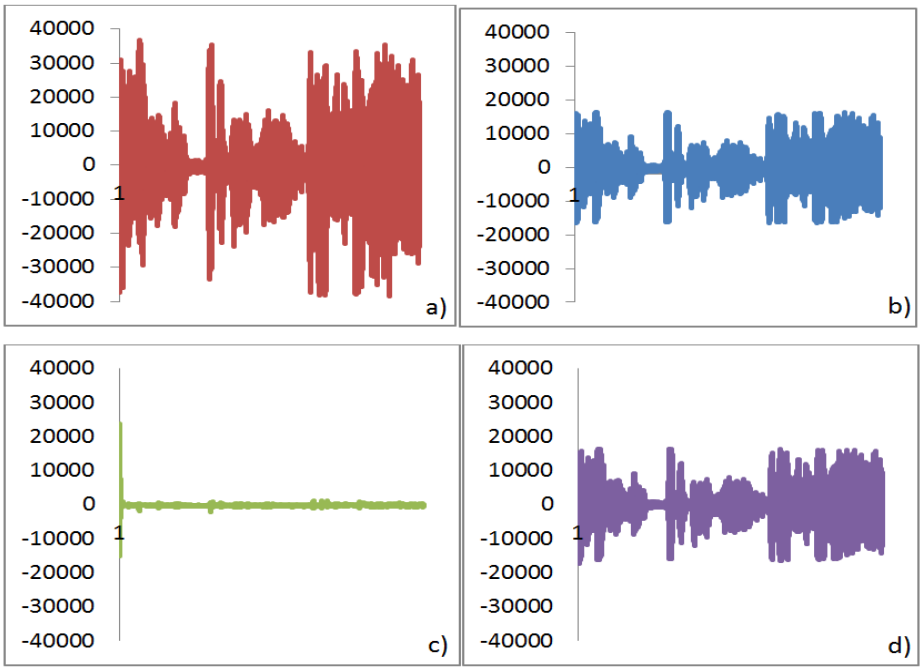

Figura 3 – a) gráfico do som original, b) gráfico do eco gerado com 25 amostras de atraso e 0.5 de atenuação, c) gráfico do sinal de erro calculado pela subtracção da saída do filtro ao eco gerado, d) gráfico da saída do filtro

Podemos verificar que não existem diferenças significativas entre as correcções apresentadas em (1) e (2) em termos de erro de correcção, no entanto os dois métodos divergem na velocidade de conversão e no valor de acerto utilizado na correcção dos coeficientes. O erro relativo absoluto varia positivamente com o aumento do número de coeficientes, ou seja, aumentar o número de coeficientes faz com que seja cada vez mais difícil obter uma réplica perfeita do sinal de eco, no entanto permite o cancelamento do eco com atrasos de amostras cada vez maiores. É possível obter melhores resultados no cancelamento do eco se houver uma maior precisão no valor de ajustamento do algoritmo, no entanto neste caso está a ser utilizado o deslocamento de bits para obtenção do valor de ajustamento o que pode originar alguma perda de precisão na obtenção do cancelamento do eco e por consequência um aumento do erro absoluto relativo.

4 Sistema utilizado na FPGA.

O sistema proposto irá ser desenvolvido numa placa de desenvolvimento ATLYS da Digilent. Esta placa de desenvolvimento é um circuito digital completo baseado numa FPGA Spartan-6 LX45 da Xilinx. A FPGA possui ligação aos vários periféricos instalados na placa ATLYS, incluindo periféricos de, ligação Ethernet Gbit, HDMI Vídeo, memória DDR2 de 128 MByte, USB e ainda de áudio (controlador AC-97 com line-in, line-out, mic e headphone).

Estas características possibilitam a implementação de várias aplicações na placa ATLYS, incluindo aplicações que utilizem processadores embebidos. A Xilinx disponibiliza implementações utilizando o processador embebido MicroBlaze para esta placa.

A FPGA Spartan-6 contêm 6822 slices cada um com quatro LUT (Look Up Tables) de 6 entradas e oito flip-flops, 2.1 Mbits em blocos de memória RAM, quatro controladores de relógio, 6 PLL (Phase Locked Loop), 58 slices para processamento digital de sinais e ainda a possibilidade de obter velocidades de relógio até 500MHz.

A arquitectura do sistema utilizado foi baseada num exemplo disponibilizado pelo fabricante da FPGA Digilent®. O exemplo fornecido utiliza as capacidades de processamento áudio da placa ATLYS e possibilita a gravação de áudio recebido pelas entradas LINE IN e MIC para a memória da placa e também a leitura desses dados com transmissão na saída LINE OUT. Este exemplo foi desenvolvido para a aplicação EDK da Xilinx onde é facilitado o desenvolvimento de *hardware* para FPGA com processador embebido. Neste caso é utilizado o processador Microblaze numa arquitectura descrita de uma forma simplificada na figura 4.

O EDK utiliza 2 aplicações, o SDK (Software Development Kit) e o XPS (Xilinx Platform Studio). O XPS é utilizado para definir o *hardware* que compõe o processador que vai ser utilizado e outro *hardware* a ser incluído na FPGA. Esta aplicação visa simplificar o desenho do *hardware* do processador e gestão de ligações do processador com outro *hardware* desenvolvido. O resultado desta aplicação é um bit file com a descrição do *hardware* para a FPGA.

O SDK é um ambiente de desenvolvimento de *software* para o processador criado no XPS. Para criar um *software* para o processador é necessária a informação do *hardware* fornecida pelo XPS e também o bit file criado para a FPGA.

O *hardware* utilizado no exemplo fornecido pela Digilent está representado num esquema simplificado na figura 4.

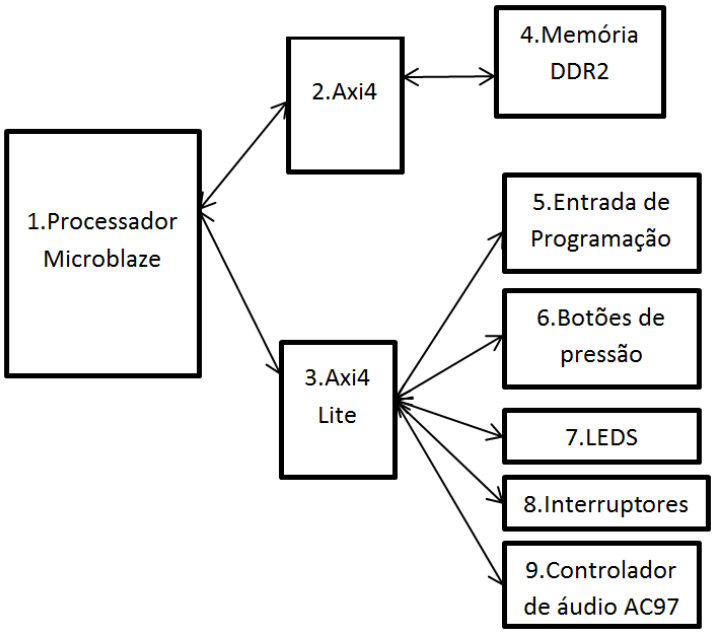

Figura 4 – Diagrama da arquitectura utilizada no exemplo da Digilent.

Os elementos 2 e 3 do diagrama representam controladores de periféricos, estes controladores fazem a gestão da comunicação entre os periféricos e o processador e gerem também transferências de dados para o processador. O elemento 3 representa um tipo de controlador que apenas permite a transferência de dados de e para os registos utilizados pelo processador Microblaze, ao passo que o elemento 3 é um controlador específico para controlar transferências de dados para memória externa da FPGA, neste caso a memoria DDR2 representada pelo elemento 4.

O elemento 5 representa a ligação USB utilizada para carregar o *software* que vai correr no processador Microblaze. Esta entrada é também utilizada, quando é seleccionada a opção de *debug* do *software* a correr no processador, para transmitir dados para o SDK.

Os elementos 6, 7 e 8 representam o *hardware* de controlo dos periféricos da placa, neste caso os botões de pressão, os leds e os interruptores.

Finalmente o elemento 9 representa o *hardware* criado pela Digilent para controlar os dados áudio recebidos e enviados pela placa ATLYS. Este *hardware* recebe dados fornecidos pelo ADC (Conversor Analógico Digital) da placa e verifica a validade e disponibilidade destes dados de acordo com a frequência de amostragem utilizada. Para enviar dados este *hardware* também verifica disponibilidade do DAC (Conversor Digital Analógico) da placa ATLYS, de acordo com a frequência de amostragem, e efectua o envia dos dados.

4.1 Implementação do algoritmo do cancelador de eco

Para implementar o algoritmo do cancelador de eco é utilizada a mesma arquitectura do exemplo fornecido pela Digilent e o esquema de implementação do algoritmo será igual ao utilizado na aplicação desenvolvida em linguagem C. Os dados áudio são passados de um PC para a FPGA através da entrada LINE IN, a partir destes dados é gerado, em *software*, o eco do sinal com atraso nas amostras e atenuação de amplitude. Numa primeira aproximação todo o algoritmo é implementado em *software* no processador Microblaze.

O processador Microblaze funciona com um relógio interno da FPGA com 12.288 MGHz (valor predefinido para a FPGA na placa ATLYS), para definir a frequência de amostragem do sinal áudio é utilizado um divisor de relógio para obter uma frequência de 8000 Hz, neste caso o divisor tem o valor de 1536, ou seja, é recebida uma amostra de áudio por cada 1536 ciclos de relógio do processador.

Para ser possível calcular a resposta do filtro, é necessário ter em memória o número de amostras áudio (16 bits) anteriores correspondente pelo menos ao número de coeficientes do filtro FIR. Para que seja possível guardar dados continuamente em memória é definido um banco de dados rotativo na memória DDR2 da placa ATLYS. Este banco de dados tem um tamanho predefinido, neste caso foi utilizado um tamanho de 8000 amostras, ou seja, 8000 posições de memória de 16 bits cada. Cada vez que o processador recebe uma amostra, a posição de armazenamento do banco de dados é incrementada de uma posição, quando chega à posição 8000 a amostra seguinte é armazenada na posição 0.

Por cada iteração do algoritmo é necessário obter uma amostra do eco gerado (sinal d(n) do algoritmo apresentado na secção 3), em vez de criar um banco de dados para armazenar os valores do eco gerado, este é obtido conforme a necessidade a partir do banco de dados das amostras do áudio recebido.

Quando é recebida uma nova amostra de áudio, essa amostra é armazenada na posição correspondente do banco de dados, em seguida é calculada a resposta do filtro utilizando as amostras armazenadas no banco de dados e um vector onde são guardados os valores dos coeficientes do filtro (cada coeficiente tem 16 bits). Em seguida é utilizada a amostra do eco para calcular o sinal de erro o que corresponde ao passo 2 do algoritmo. Com o sinal de erro é então possível efectuar o passo 3 para corrigir os coeficientes de acordo com o método seleccionado (1) ou (2).

O teste deste algoritmo deverá obter um sinal com amostras de valores muito próximos de zero como mostrado nos exemplos anteriores do algoritmo. Este sinal de erro é enviado para a saída da placa LINE OUT onde é reproduzido em colunas de som.

Para manter a frequência de amostragem de 8KHz é necessário que o processamento, dos passos do algoritmo, não necessite de mais do que 1536 ciclos de relógio do processador, se este

limite for ultrapassado ocorrerá a perca de amostras do sinal de áudio recebido, o que na realidade corresponde a diminuir a frequência real de amostragem do sinal de áudio. Após verificar o funcionamento do algoritmo verificamos que é possível obter até 280 coeficientes mantendo todo o processamento dentro do limite de 1536 ciclos de relógio.

Na figura 5 estão apresentados os resultados obtidos na placa ATLYS com processamento no processador Microblaze da FPGA com os seguintes parâmetros, filtro com 280 coeficientes, eco com 200 amostras de atraso e 0.5 de atenuação, um valor de acerto do algoritmo de 0.015625 e o método (1).

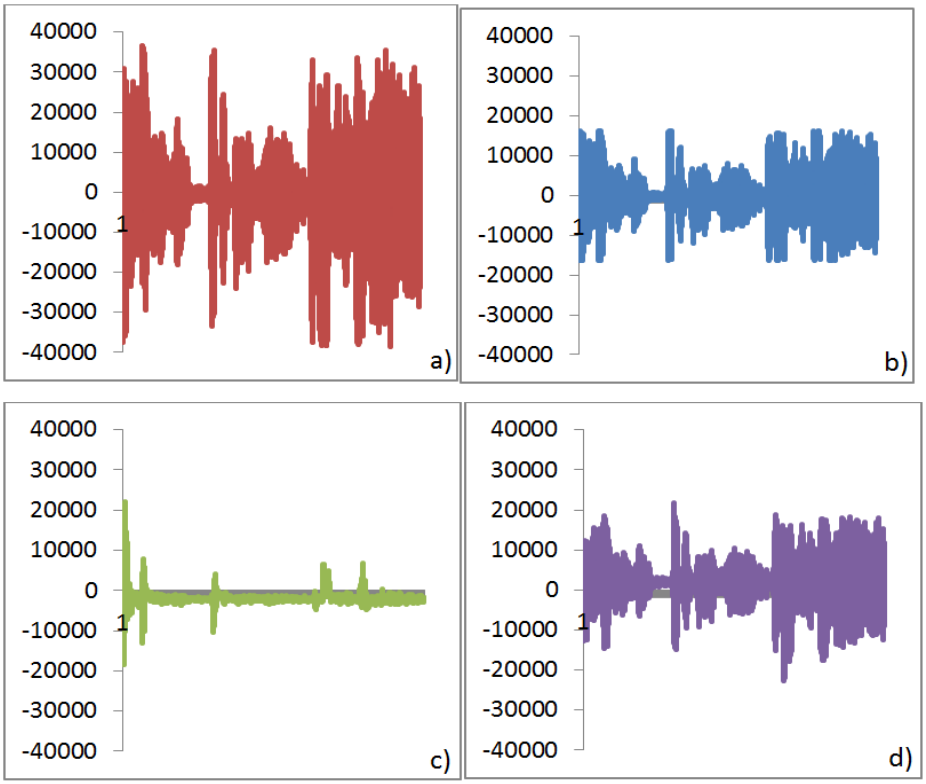

Figura 5 – a) gráfico do som original, b) gráfico do eco gerado com 200 amostras de atraso e 0.5 de atenuação, c) gráfico do sinal de erro calculado pela subtracção da saída do filtro ao eco gerado, d) gráfico da saída do filtro

No gráfico c) é possível verificar o efeito da falta de precisão no valor do acerto do algoritmo, neste caso o valor do erro absoluto é de 2071.01 o que corresponde a 6.32%. Apesar de este valor aumentar relativamente aos resultados com 50 coeficientes, podemos considerar que está ainda dentro de valores aceitáveis para aplicações de cancelamento de áudio, pois, em média, apenas 6.32% de cada amostra do eco não é cancelada e geralmente o eco tem amplitudes muito baixas quando comparadas com a amplitude do sinal original. Na figura 6 temos os resultados com o método (2) com 280 coeficientes, 200 amostras de atraso, 0.5 de atenuação e um valor de acerto de 0.00390625. Também aqui é possível ver os efeitos da falta de precisão do valor do acerto do algoritmo onde o valor do erro absoluto é de 4.46%.

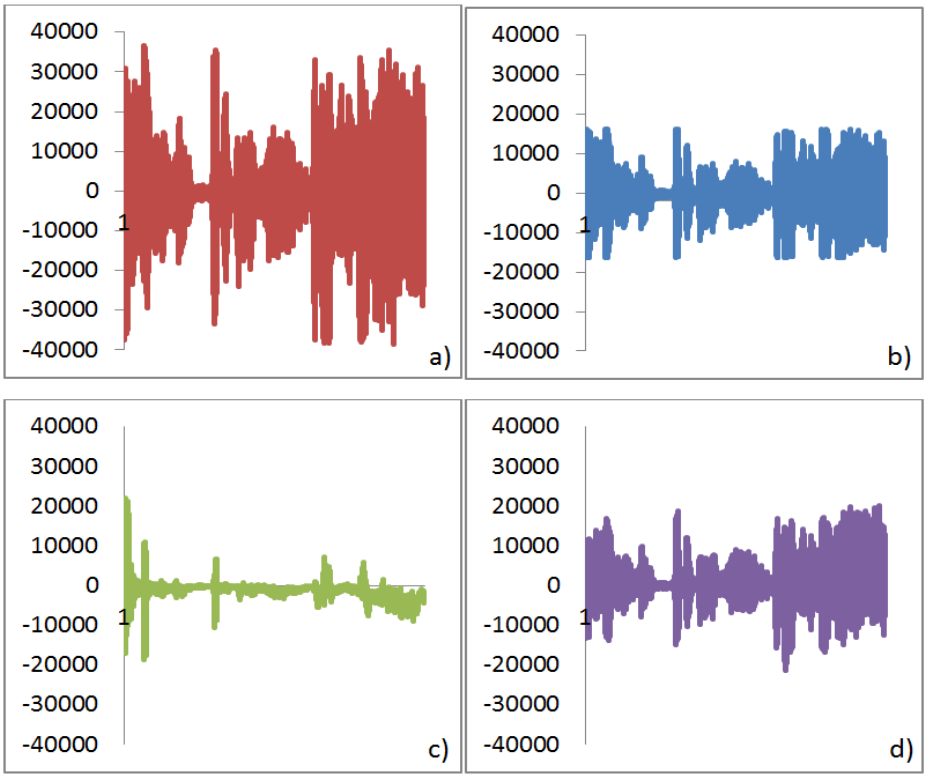

Figura 6 – a) gráfico do som original, b) gráfico do eco gerado com 200 amostras de atraso e 0.5 de atenuação, c) gráfico do sinal de erro calculado pela subtracção da saída do filtro ao eco gerado, d) gráfico da saída do filtro

Para a implementação do algoritmo com parte em *software* e parte em *hardware* foi acrescentado *hardware* dedicado para calcular a resposta do filtro, assim o esquema da figura 4 passa a ser o representado na figura 7. A única alteração efectuada foi acrescentar o *hardware* e o controlador para o cálculo do filtro representado na figura 7 pelo elemento 10.

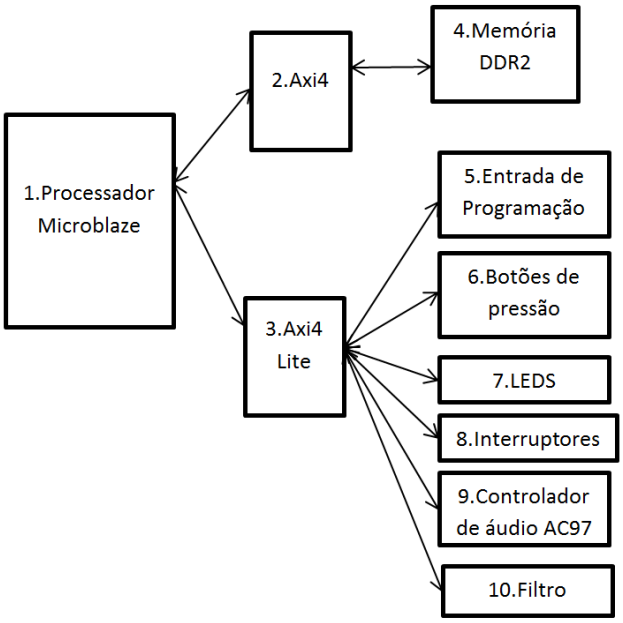

Figura 7 – Diagrama da arquitectura utilizada no com implementação do algoritmo e hardware de cálculo da saída do filtro.

O elemento 10 é composto por dois multiplicadores e dois somadores e um acumulador, este elemento é capaz de efectuar 2 multiplicações num só ciclo de relógio. O relógio utilizado para este *hardware* é o mesmo utilizado para o processador com 12.288 MGHz. Na figura 8 temos uma representação do *hardware* implementado no elemento 10. Os dados são carregados para os registos 1 a 4 para serem multiplicados, no ciclo seguinte é efectuada a soma dos resultados dos multiplicadores e no ciclo seguinte é efectuada a soma do acumulador com a soma dos resultados dos multiplicadores, no último ciclo o valor resultante é guardado no acumulador. Estes passos são controlados por uma máquina de estados com 3 estados também implementada no elemento 10.

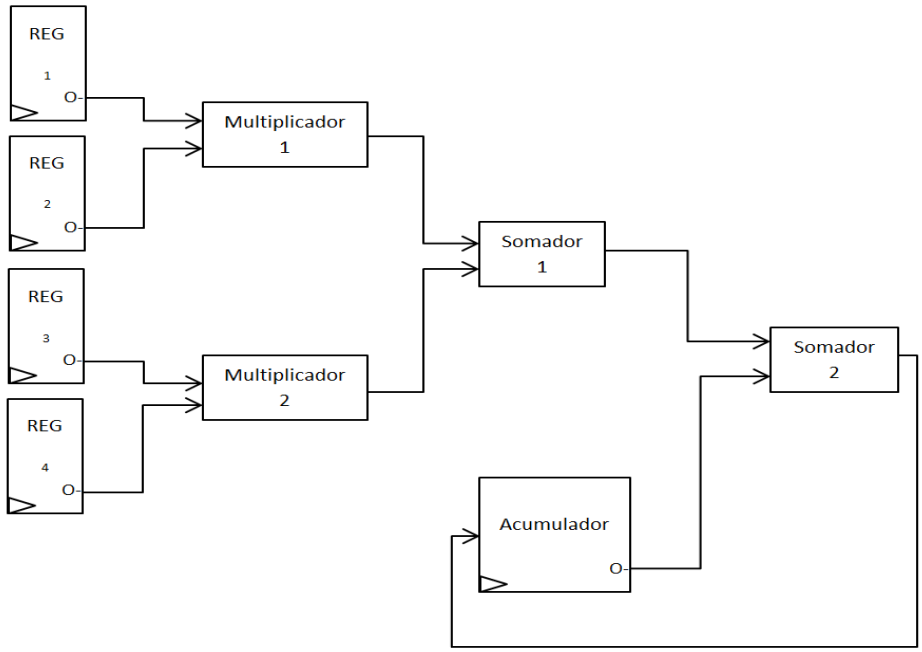

Figura 8 – Datapath implementado no elemento 10.

Este elemento é ligado ao processador através do controlador representado pelo elemento 3 na figura 7, este controlador apenas permite acesso aos registos do processador e apenas consegue efectuar uma transferência de 32 bits (dados de um registo) de cada vez, ou seja, os dados enviados para os registos 1 a 4 do elemento 10 não são enviados simultaneamente. A transferência de 32 bits, dos registos do processador para o elemento 10, demora 17 ciclos de relógio do processador, este tempo de transferência vai influenciar muito a performance do cálculo da resposta do filtro utilizando o *hardware* dedicado. Assim, mantendo os 1536 ciclos de relógio de limite (correspondente a manter uma frequência de amostragem de 8KHz) apenas é possível obter uma implementação com um máximo de 75 coeficientes no filtro FIR. Os resultados obtidos utilizando o *hardware* estão representados na figura 9 com os seguintes parâmetros, 75 coeficientes, 50 amostras de atraso com 0.5 de atenuação, um valor de acerto 0.0625 e com o método (1). O erro médio absoluto obtido foi de 1.35%.

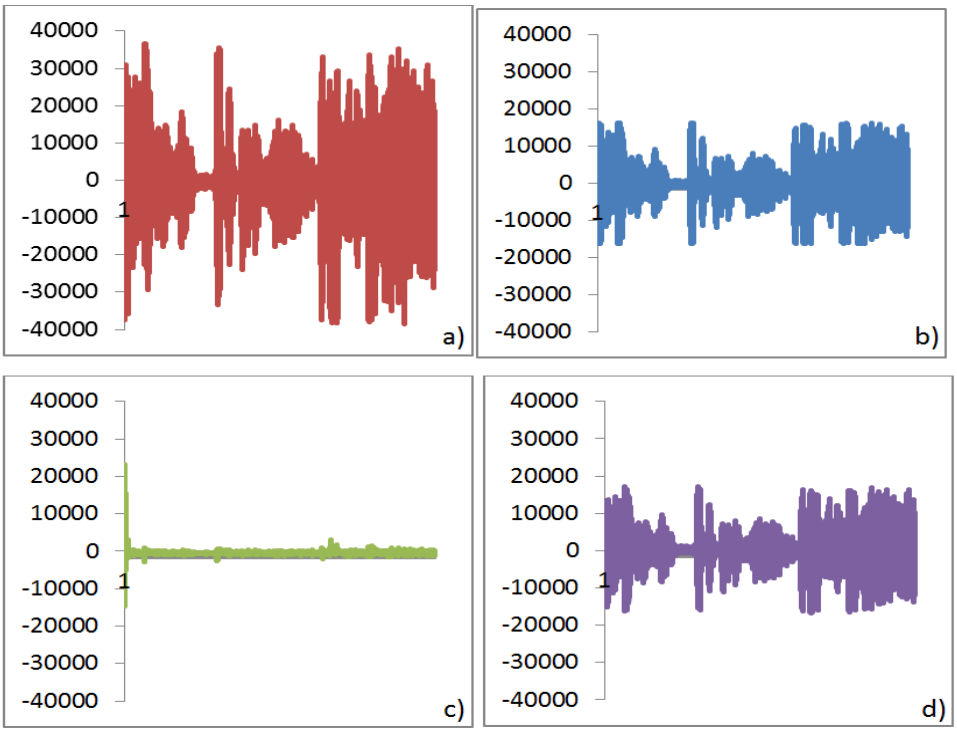

Figura 9 – a) gráfico do som original, b) gráfico do eco gerado com 50 amostras de atraso e 0.5 de atenuação, c) gráfico do sinal de erro calculado pela subtracção da saída do filtro ao eco gerado, d) gráfico da saída do filtro

Na figura 10 temos os resultados obtidos utilizando a implementação mista com o método (2), 75 coeficientes, eco com 50 amostras de atraso e 0.5 de atenuação e um valor de acerto 0.01565. O erro médio absoluto é de 0.96% para este caso.

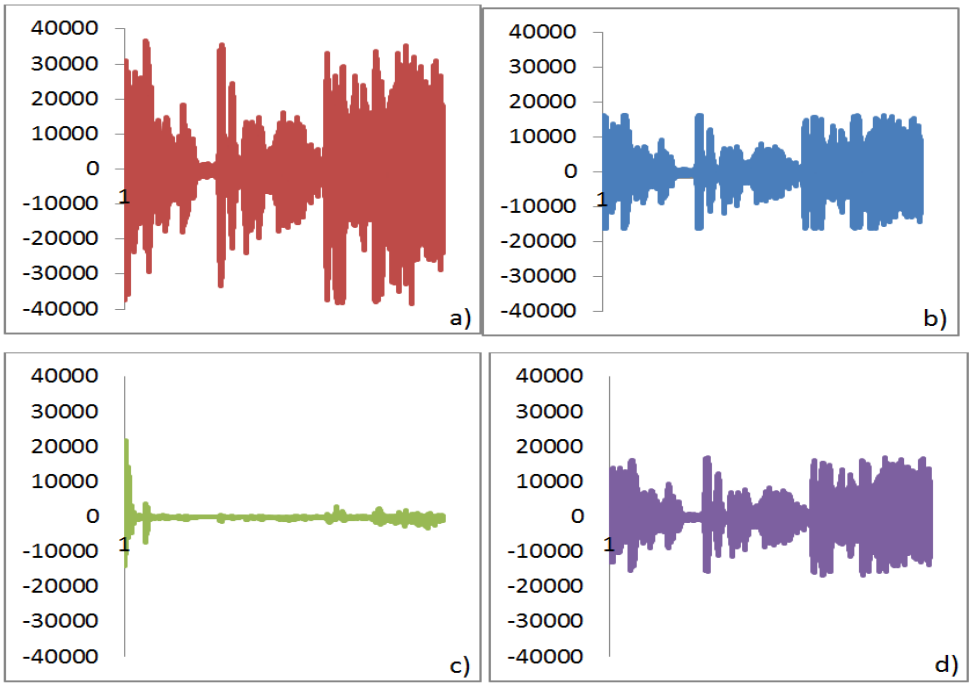

Figura 10 – a) gráfico do som original, b) gráfico do eco gerado com 50 amostras de atraso e 0.5 de atenuação, c) gráfico do sinal de erro calculado pela subtracção da saída do filtro ao eco gerado, d) gráfico da saída do filtro

Apesar dos efeitos verificados pela falta de precisão no valor do acerto do algoritmo e pelo efeito dos tempos de transferências de dados, o sistema implementado está de acordo com o objectivo da bolsa. Temos portanto um sistema implementado com *hardware* e *software* e com a possibilidade de alternar entre os dois métodos, pois com a implementação mista o utilizador pode seleccionar se utiliza só *software* ou também *hardware* através de opções no código implementado no processador Microblaze.

5 SystemC

SystemC é uma linguagem de desenvolvimento de sistemas que tem evoluído como resposta a uma necessidade de melhorar a produtividade no desenho de sistemas electrónicos. Hoje em dia a maior parte dos sistemas contêm *hardware* dedicado em conjunção com *software* e geralmente são os dois desenvolvidos em simultâneo devido a restrições temporais. Para evitar falhas de funcionamento quer da parte do *software* quer da parte do *hardware* é necessário efectuar testes extensivos das aplicações. O SystemC oferece a possibilidade de desenhar componentes de *hardware* e *software* em conjunto e com interacções como no produto final utilizando um nível alto de abstracção.

Para descrever um dado sistema em SystemC é necessário dividir o projecto em módulos. Cada um destes módulos irá efectuar uma dada função no projecto, a função utilizada depende também do nível de abstracção pretendido, se for desejada uma simulação com comportamento de baixo nível é necessário desenhar os módulos de maneira que descrevam o projecto a esse nível. Uma das vantagens do SystemC é permitir a simulação do comportamento do sistema com o mínimo de informação disponível e, à medida que se obtém informação do comportamento do sistema é possível adiciona-la à simulação do sistema.

Um módulo em SystemC é usualmente descrito com um conjunto de sinais que recebe de outros módulos ou sinais de simulação (por exemplo o relógio geral), um conjunto de sinais internos, um construtor que define a interacção do módulo com os sinais do projecto e inicialização dos sinais internos e, finalmente, a descrição da sua função.

Para a descrição do projecto desenvolvido na bolsa foram utilizadas algumas regras de definição dos módulos, pois, devido à possibilidade de descrever o sistema em vários níveis de abstracção, existem várias possibilidades de descrever um módulo com uma certa função.

A descrição de cada módulo deverá iniciar-se com a descrição dos sinais de entrada e saída do módulo, é necessário indicar o tipo de sinal (input, output, clock, etc…), o tamanho em número de bits e o nome a dar a cada sinal internamente no módulo. Em seguida é necessário descrever os sinais internos do modulo, nome da variável, tamanho em número de bits e se se trata de uma variável com ou sem sinal.

O construtor do módulo define o tipo de simulação utilizada pelo SystemC nesse módulo. Neste projecto são utilizados construtores com a possibilidade de terem dados de configuração, ou seja, um módulo pode ser descrito com uma certa função mas com parâmetros variáveis, por exemplo, uma memória pode ter tamanho variável e o tamanho é definido como parâmetro do construtor.

Cada módulo tem um conjunto de sinais que interagem directamente com a função do módulo, ou seja, a função do módulo é executada se existirem mudanças nestes sinais, esta lista de sinais representa a lista de sensibilidade do módulo que é definida no construtor. As listas de sensibilidade utilizadas neste projecto são do tipo SC_THREAD, pois este tipo de listas permite que o SystemC simule o comportamento do módulo de uma forma continua e não apenas quando existem alterações na lista de sensibilidade.

Uma das vantagens das lista de sensibilidade SC_THREAD é permitirem a utilização da função wait(). Esta função permite parar a execução da função do módulo durante um certo número de ciclos de relógio, assim é possível adicionar tempos de processamento numa fase posterior do projecto. Por exemplo, um dado cálculo demora um certo número de ciclos de relógio de um processador a ser efectuados, inicialmente podemos efectuar uma simulação onde se considera que o cálculo demora apenas um ciclo de relógio e mais tarde, através da função wait() adicionar a informação de tempo de processamento real.

Finalmente é feita a descrição da função do módulo. Esta função pode ser descrita como uma função de C ou C++ de acordo com as necessidades de abstracção do projecto. É nesta função que pode ser utilizada a função wait().

Na figura 11 está apresentado um modelo geral de um módulo utilizado para descrever o projecto em SystemC.

```
SC_MODULE (module_name) {

      // input ports
      sc_in_clk example_clock;
      sc_in<bool> example_bool;
      sc_in<sc_int<size>> example_size;

      // output ports
      sc_out<bool> example_bool;
      sc_out<sc_int<size>> example_size;

      //local variables
      sc_int<size> example;
      unsigned int example2;

      // CONSTRUCTOR
      SC_HAS_PROCESS(module_name);
      module_name (sc_module_name instname, unsigned int
example_parameter)
      : sc_module(instname)
      , example2 (example_parameter)

      // sensitivity list and initializations
      {
            SC_THREAD(module_function);
            sensitive << clock.pos();
            sensitive << example_bool.pos();
      }

      // module function
      void module_function() {
            while(1){

                  // module function description

                  wait();
            }
       }
};
```

Figura 11 – Modelo utilizado para descrição de módulos em SystemC.

5.1 Descrição do cancelador de eco em SystemC

Para descrever o cancelador de eco em SystemC foram criados 9 tipos de módulos para executar as diferentes funções necessárias para simular o comportamento do projecto. Em seguida segue uma descrição dos módulos desenvolvidos e as suas ligações no sistema implementado em SystemC estão representadas na figura 12.

1) Módulo de recebimento de amostras áudio. Este módulo simula o recebimento de amostras áudio efectuando a abertura e leitura de um ficheiro que contém as amostras de áudio.
2) Módulo de sincronização da leitura das amostras. Este módulo utiliza um contador para simular a frequência de 8KHz utilizada na FPGA. A simulação vai utilizar um relógio geral de 12.288 MGHz, assim este módulo utiliza um contador de 1536 ciclos de relógio para indicar que a amostra de áudio esta pronta.
3) Módulo de controlo. Este módulo está desenhado para simular o comportamento do processador para indicar a um dado modulo para executar a sua função, ou seja, efectua toda a organização do projecto.
4) Módulo de memória. Este módulo é utilizado para simular acessos à memória. Foi desenvolvido para se comportar como uma memória de duplo porto para escrita e leitura no mesmo ciclo.
5) Módulo de selecção de acesso a memória. Este módulo efectua a selecção de quais módulos conseguem ter acesso à memória e é controlado pelo módulo de controlo.
6) Módulo de cálculo de resposta do filtro FIR. Este módulo efectua o cálculo da resposta do filtro acedendo às posições de memória de dados e coeficientes necessários e guardando o valor num acumulador.
7) Módulo de cálculo do eco. Este módulo acede à memória de amostras de áudio para seleccionar a amostra correcta para o eco configurado.
8) Módulo de cálculo do sinal de erro. Este módulo utiliza o valor calculado no módulo de cálculo de resposta do filtro FIR e o valor obtido no módulo de cálculo do eco para obter o sinal de erro.
9) Módulo de correcção de coeficientes. Este módulo utiliza o valor calculado pelo módulo de cálculo do sinal de erro para corrigir os coeficientes em memória utilizando os métodos já descritos em (1) e (2).

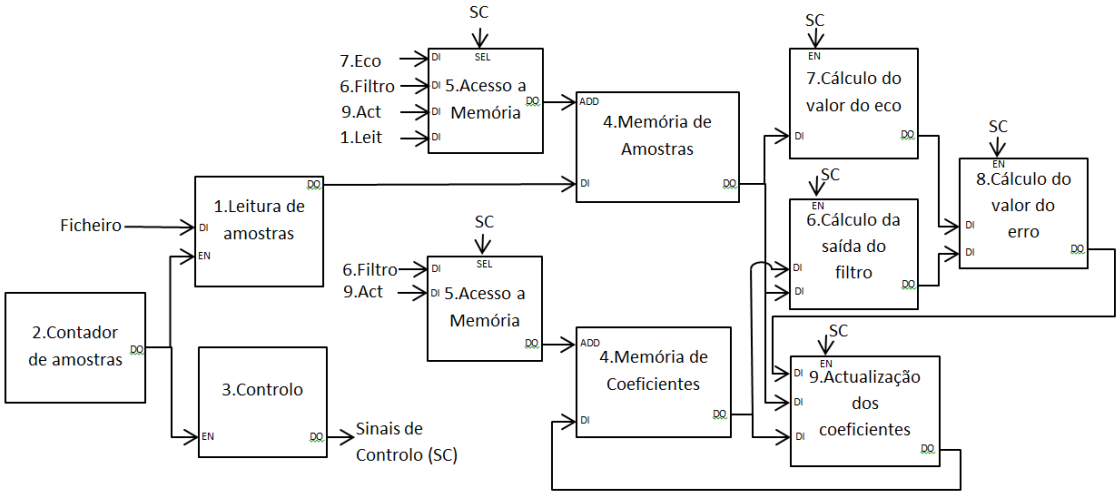

Figura 12 – Arquitectura do sistema descrito em SystemC.

Para simular, em SystemC, o funcionamento do sistema descrito na figura 12 foram criadas três versões do sistema. A primeira é uma versão que simula a implementação do sistema utilizando apenas *software*. Os resultados desta simulação deverão ser semelhantes aos obtidos na implementação do cancelador de eco utilizando apenas *software* no processador MicroBLaze.

A segunda versão corresponde a uma implementação utilizando apenas *hardware* dedicado. Esta versão não foi implementada na FPGA.

A terceira versão corresponde à implementação utilizada na FPGA com *hardware* dedicado para efectuar o cálculo da resposta do filtro FIR e o restante algoritmo do cancelador de eco implementado em *software* a ser executado no processador MicroBlaze.

Para obter uma simulação em SystemC, que obtenha resultados semelhantes ao funcionamento do software implementado na FPGA, utilizou-se a função *wait()* para indicar que cálculos são efectuados utilizando um ciclo de relógio do processador. A função *wait(n)* é utilizada para indicar que o cálculo é efectuado em n ciclos de relógio do processador, assim o simulador deverá esperar n ciclos de relógio. Caso não seja utilizada esta função o simulador considera que as operações são executadas em paralelo no mesmo ciclo de relógio.

Na figura 13 temos um troço de código utilizado no cálculo da saída do filtro onde se verifica a utilização da função *wait()*. Aqui é considerado que as operações de cálculo do endereço de memória, multiplicação do coeficiente do filtro pela a amostra correspondente e soma do resultado da multiplicação com o valor do acumulador demoram, cada uma, um ciclo de relógio do processador a executar sendo assim inserida a função *wait()* após cada uma destas funções.

```
for(count=0;count<ecoef;count++){
    if(count>base.read()){
        addsam.write(buffer+base.read()-count+1);
    }else{
        addsam.write(base.read()-count);
    }
    addcoef.write(count);
    wait();
    temp = ((amostra.read()*coef.read())>>15);
    wait();
    acumulador = acumulador + temp;
    wait();
}
```

Figura 13 – Exemplo de utilização da função *wait()* em SystemC.

A descrição dos bancos de memória utilizados tem um funcionamento simplificado relativamente ao utilizado pelo processador MicroBlaze implementado na FPGA, no entanto a utilização da função *wait()* permite que os bancos de memória implementados em SystemC tenham um comportamento (em termos de ciclos de relógio) semelhante ao comportamento real das memórias implementadas no sistema na FPGA.

5.1.1 Resultados de simulação em SystemC da implementação só com *software*

Na figura 14 está apresentado um resultado de simulação da versão do sistema utilizando apenas *software* com 280 coeficientes, 200 amostras de atraso e 0.5 de atenuação no eco, um valor de acerto do algoritmo de 0.015625 e utilizando o método (1).

Os resultados da figura são obtidos passando a informação dos sinais simulados para um ficheiro do tipo *virtual CD* (*.vcd*) de maneira a ser possível visualizar a informação em forma de onda ao longo da simulação.

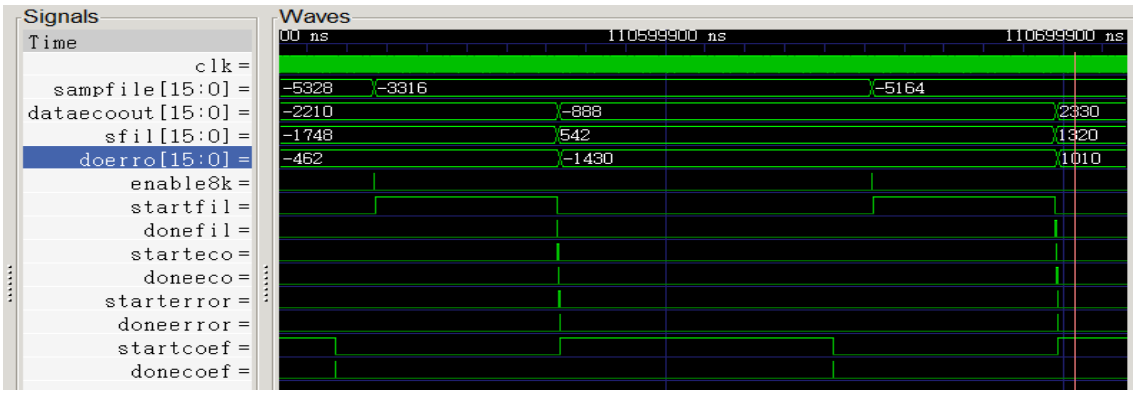

Figura 14 – Resultado de simulação (software) do cancelador de eco utilizando SystemC.

Na figura 14 a onda com o nome *enable8k* indica quando está uma nova amostra disponível possibilitando assim uma fácil visualização do tempo de processamento disponível entre amostras.

O processamento do cálculo de filtro é dado pelo sinal *startfil* e termina com *donefil*, de forma semelhante temos a obtenção do eco com *starteco* e *doneeco*, cálculo do erro com *starterror* e *doneerror* e finalmente a actualização dos coeficientes com *startcoef* e *donecoef*. Assim temos que cada iteração do algoritmo do cancelador de eco inicia-se com o recebimento de uma nova amostra em *enable8k* e termina com a actualização dos coeficientes em *donecoef*.

Com os resultados apresentados podemos verificar que com 280 coeficientes o processamento de cada iteração do algoritmo do cancelador de eco termina muito perto do limite do tempo de processamento para cada amostra. Utilizando a totalidade do tempo de processamento entre amostras seria possível implementar o algoritmo com 290 coeficientes, este resultado é semelhante ao obtido na implementação do algoritmo implementado no MicroBlaze onde o limite de coeficientes implementados se situava nos 280 coeficientes. As diferenças entre a simulação utilizada e o sistema implementado estão relacionadas com a simplificação do acesso à memória para que a simulação em SystemC não seja demasiado complexa.

Os resultados utilizando o método (2) são iguais aos obtidos no método (1) em termos de ciclos de relógio de simulação, pois é considerado que a operação de troca de sinal é efectuada num ciclo de relógio do processador tal como a multiplicação.

5.1.2 Resultados de simulação em SystemC da implementação mista

Na implementação mista é considerado que o cálculo da saída do filtro é efectuado utilizando *hardware* dedicado e os restantes cálculos efectuados em *software*. Esta simulação visa obter resultados semelhantes à implementação na FPGA também com *hardware* dedicado para calcular a saída do filtro. Assim foi inserido o tempo de transferência de dados para o hardware descrito na secção 4.1 (17 ciclos de relógio para cada transferência de 32 bits) utilizando também a função *wait(n)* com n igual a 17. Este valor de 17 ciclos de relógio foi medido utilizando contadores implementados no *hardware* de FPGA.

Os resultados obtidos para 75 coeficientes com 50 amostras de atraso 0.5 de atenuação no eco, um valor de acerto do algoritmo de 0.0625 e utilizando o método (1) estão apresentados na figura 15.

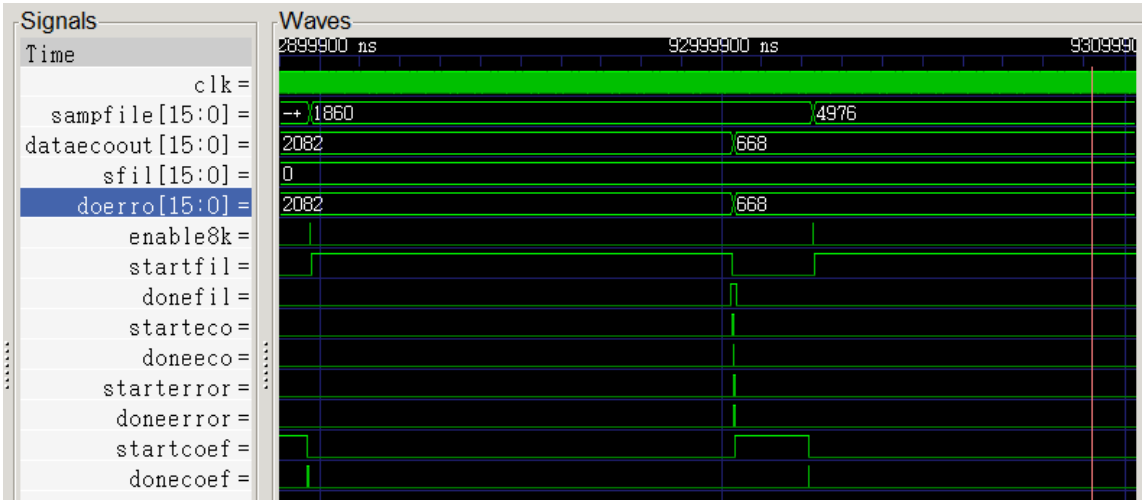

Figura 15 – Resultado de simulação (mista) do cancelador de eco utilizando SystemC.

É possível verificar que, utilizando o atraso de 17 ciclos de relógio na transferência de dados para o *hardware*, o tempo de simulação para cada iteração do algoritmo fica no limite do tempo de processamento entre amostras confirmando assim que a simulação em SystemC tem um comportamento semelhante com o obtido na implementação na FPGA onde foi possível implementar apenas 75 coeficientes com a implementação mista.

Utilizando o método (2) são obtidos resultados semelhantes pois a operação de troca de sinal é efectuada no mesmo tempo, em ciclos de relógio do processador, que a multiplicação.

5.1.3 Resultados de simulação em SystemC da implementação só com hardware

Nesta implementação é considerado que todo o sistema do cancelador de eco é implementado utilizando *hardware* dedicado e que o controlo é efectuado por uma máquina de estados também implementada no *hardware* dedicado. Esta simulação visa obter resultados de um sistema implementado sem a utilização do MicroBlaze, assim é considerado que as operações efectuadas demoram um ciclo de relógio e que existe a possibilidade de paralelização de algumas das operações.

Os resultados para esta implementação estão apresentados na figura 16, com 750 coeficientes, 300 amostras de atraso e 0.5 de atenuação do eco e um factor de correcção do algoritmo de 0.001953125. Os resultados para o método da multiplicação e para o método da alteração de sinal são semelhantes em termos de ciclos de relógio pois a implementação efectuada utiliza o mesmo número de ciclos de relógio nos dois casos.

Figura 16 – Resultado de simulação (hardware) do cancelador de eco utilizando SystemC.

Devido à implementação utilizada para o factor de correcção do algoritmo, o factor de correcção necessário nesta implementação apresenta pouca precisão, pelo que os resultados de cancelamento do eco não são precisos o suficiente para efectuar um cancelamento aceitável do eco. Para utilizar devidamente uma implementação deste género seria necessário alterar o método de obtenção do factor de correcção.

6 Conclusões

Numa primeira fase foi implementado o algoritmo de cancelador de eco em *software* utilizando linguagem de programação C. O objectivo desta implementação é poder obter resultados para poder comparar com as implementações na FPGA e em SystemC a fim de poder detectar eventuais erros.

Numa segunda fase foi efectuada a implementação do algoritmo do cancelador de eco na FPGA Spartan-6 na placa de desenvolvimento ATLYS utilizando uma implementação só de *software* a ser executado no processador MicroBlaze implementado com ferramentas da Xilinx com descrições utilizando a linguagem VHDL. Nesta fase foi também implementado um cancelador de eco onde o cálculo da saída do filtro é calculado utilizando *hardware* dedicado e o restante do algoritmo do cancelador de eco é executado no processador MicroBlaze.

Finalmente os sistemas implementados foram descritos na linguagem SystemC para efectuar uma simulação comportamental do sistema e comparar resultados da simulação com os obtidos nos sistemas implementados.

A implementação do algoritmo do cancelador de eco apresenta alguns resultados pouco eficientes, pois a implementação seleccionado é uma das mais simples. Esta implementação foi seleccionada para que seja mais fácil comparar os resultados da simulação de SystemC com os resultados do sistema real implementado.

As implementações utilizando a placa de desenvolvimento ATLYS com a FPGA Spartan-6 mostraram os resultados esperados apesar de a implementação com *hardware* e *software* apresentar uma performance pouco desejável devido aos tempos de transferência de dados para o hardware. Seria possível resolver esta questão utilizando outros elementos disponibilizados pela Xilinx, mas aumentando a complexidade do sistema implementado.

A simulação utilizando a implementação em SystemC obteve resultados muito semelhantes aos obtidos pelos sistemas reais, apesar de ter sido efectuada uma simplificação do sistema de acesso à memória de dados utilizada na implementação na FPGA.

7 Referências

[1] R. Khalil, “Adaptative Filter Application in Echo Cancellation System”

[2] D. Black, J. Donovan, “SystemC: From The Ground Up”, 2004, Kluwer Academic Publishers

[3] J. Bhasker, “A SystemC Primer”, Second Edition, 2004, Star Galaxy Publishing