Obtivemos então o vetor codificado em 8 bits, a partir da aplicação do PCM no sinal modulante. Para dar prosseguimento ao objetivo de recuperar o sinal analógico original será necessário a representação das amostras obtidas com PWM através da aproximação da amostragem ideal.

Através da comparação do valor absoluto de cada amostra com uma onda dente de serra, a geração do PWM é feita. O funcionamento da saída se dá da seguinte forma: enquanto a onda dente de serra for menor que o valor da amostra a saída terá nível lógico alto e quando não for menor a saída terá nível lógico baixo, produzindo assim uma modulação em largura de pulso.

A geração do PWM na FPGA será realizada comparando a amostra e o valor em um contador incrementado por um clock, levando em conta que o contador deverá resetar para zero quando ultrapassar seu valor máximo, significando que se deve atualizar a amostra utilizando a próxima em seguida.

Para definir o valor da frequência do clock que servirá para incrementar o contador se usará da seguinte relação desta variável com a frequência de amostragem do sinal, que por sua vez é igual a 800 Hz.

$$fs = \frac{fclk}{cmax + 1}$$

Onde $cmax$ é o valor máximo do contador, cujo valor deve respeitar a seguinte inequação: $cmax > 2^{N+1}$, em que N é o número de bits de quantização utilizado. Esta inequação existe para garantir que o sinal PWM gerado não possua nunca um duty cicle superior a 50% da frequência de amostragem. Como neste trabalho utilizou-se 8 bits de quantização, tem-se que: $cmax > 2^{8+1} \Rightarrow cmax > 2^{9} \Rightarrow cmax >$ 512. A fim de obter um valor da frequência do clock redondo, escolheu-se um valor de $cmax$ igual a 999 que satisfaz a inequação. Desta forma, tem-se que:

$$fs = \frac{fclk}{cmax + 1} \Rightarrow fclk = fs(cmax + 1) = 8000(999 + 1) \Rightarrow fclk = 800kHz$$

Após a definição dos parâmetros, é possível iniciar o desenvolvimento da lógica em Verilog para replicar o circuito desejado para gerar o PWM. Para evitar falhas e comprovar o funcionamento do módulo geral, foi decidido programar um módulo para cada “componente” do circuito completo, ou seja, programou-se um módulo separado para o contador, a amostragem e o comparador. Por fim, juntou-se ambos para o módulo geral do PWM.

Primeiramente, foi criado o módulo das amostras, dado por:

```
module amostras(
    input clk,
    input [13:0] A,
    output reg [11:0] amostra
);

reg [7:0] mem [0:799];
initial begin
    $readmemb("C:/Users/linco/Desktop/PWM/C.txt",mem);
end
always @(A) begin
    amostra = mem[A];
end

endmodule
```

Onde a ideia é criar um vetor 800x8 que seja capaz de armazenar os valores, em que cada uma dessas posições serão preenchidas com as amostras codificadas do sinal modulante decorrente do processo de PCM através de um arquivo .txt, onde também o valor de saída deste módulo será controlado pelo valor de entrada “A”, onde a saída será dada pelo valor da posição “A” da memória. Por exemplo, se o valor de “A” é 1, a saída amostra será o valor da memória na posição 1.

Já o módulo do contador é dado por:

```
module contador(
    input clk,
    output reg [11:0] cont,
    output reg [13:0] A
);
 initial cont = 12'd0;
 initial A = 14'd0;

always @(posedge clk) begin
    cont <= cont + 12'd1;
    if(cont >= 12'd999)begin
        cont <= 12'd0;
        A <= A + 14'd1;
        if(A == 14'd799) A <= 14'd0;
    end
end
 endmodule
```

Como sabemos, o contador é incrementado com o clock, de 1 em 1 e possui valor máximo de 999 e assim que ele passa esse valor ele volta a valer 0 e recomeça todo o processo. Além disso, o contador também deve servir para indicar qual posição da memória, isto é, qual amostra, deve ser comparada para gerar o PWM. Ou seja, quando o contador reinicia deve-se atualizar a amostra utilizando a próxima. Isso é feito incrementando 1 na variável de controle da posição da amostra toda vez que o valor do contador é extrapolado. Como há no total 800

amostras, esse é processo é realizado até chegar neste valor, que então faz com que o valor da posição da amostra volte a 0, reiniciando assim o ciclo.

Assim temos o módulo de comparação, que é dado por:

```
module comparador(
    input [11:0] amostra,
    input [11:0] cont,
    output reg pwm
    );

always @(*) begin
    if(amostra > cont) pwm = 1;
    else pwm = 0;
end

endmodule
```

Cuja função do módulo é comparar os valores de entrada referentes à amostra atual e do contador e, se o valor absoluto da amostra é maior do que o do contador, a saída PWM é igual a 1(nível lógico alto). Caso contrário, a saída PWM é 0 (nível lógico baixo).

Escolheu-se o nível lógico igual 1 por ser o padrão. Desta forma, quando o sinal for recuperado ele terá o seu valor entre 0 e 1 ainda que a forma de onda seja mantida.

Na verdade, o seu valor máximo terá aproximadamente o valor de $\frac{valor\ máximo\ da\ amostra}{valor\ máximo\ do\ contador} * 1 = \frac{255}{999} = 0{,}255$, já que a amostra com o valor de 255 só terá o valor lógico 1 no período do PWM durante uma fração de 0,255 do período deste. Ou seja, para se chegar ao valor correto na reconstrução do sinal será necessário aplicar um ganho juntamente com o filtro passa baixa, o que será mais explicado na próxima etapa do trabalho.

Para realizar a interconexão desses três módulos no módulo principal, temos um módulo dado por:

```
module PWM (
    input clk,
    output saida
);

 (*keep=1*) wire [11:0] cont;
 (*keep=1*) wire [11:0] amostra;
 (*keep=1*) wire [13:0] A;
 contador C(clk, cont, A);
 amostras Amos(clk, A, amostra);
 comparador comp(amostra, cont, saida);

 endmodule
```

Em que a expressão (*keep=1*) é feita apenas para melhor visualização dessas variáveis na simulação. A conexão do módulo é realizada da seguinte forma: a saída do módulo contador “A” vai para a entrada do módulo amostras para indicar qual posição da memória e, consequentemente, qual amostra deverá ser comparada. Então essa amostra e a saída “cont” do módulo contador vão para a entrada do módulo comparador onde são comparadas com intuito de gerar o sinal PWM.

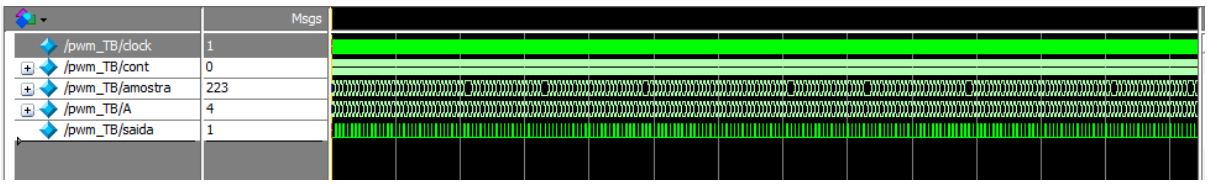
O código de simulação é mostrado abaixo, onde se implementa um clock com período de 1250 ns, ou seja, com frequência de 800 kHz:

```
1   `timescale 1ns/10ps
2
3   module PWM_TB;
4   reg clock;
5   reg [11:0] cont;
6   reg [11:0] amostra;
7   reg [13:0] A;
8   wire saida;
9
10  PWM DUT(
11      .clk(clock),
12      .saida(saida)
13  );
14
15  initial begin
16      clock = 0;
17  end
18
19  always begin
20      #625 clock = ~clock;
21  end
22
23  initial begin
24      $init_signal_spy("/PWM_TB/DUT/cont","cont",1);
25      $init_signal_spy("/PWM_TB/DUT/amostra","amostra",1);
26      $init_signal_spy("/PWM_TB/DUT/A","A",1);
27  end
28
29  initial begin
30      #1005000000 $stop;
31  end
32
33
34  endmodule
```

Também é possível ver que o valor definido para a frequência do clock é exatamente o calculado anteriormente:

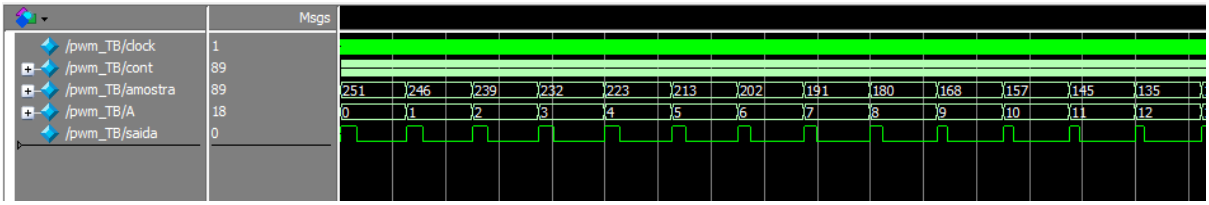
	Clock Name	Type	Period	Frequency	Rise	Fall
1	clk	Virtual	1250.000	0.8 MHz	0.000	625.000

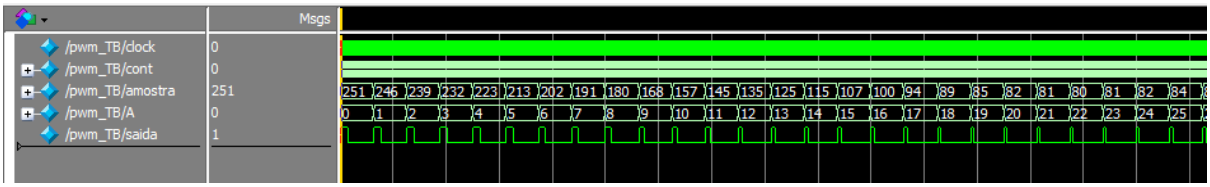
O resultado da simulação é mostrado a seguir:



Sabendo que, com a visão original não seria possível realizar a análise, dado que o vetor de saída PWM tem um tamanho relativamente grande, será necessário dar um zoom para poder fazer uma análise mais qualificada.

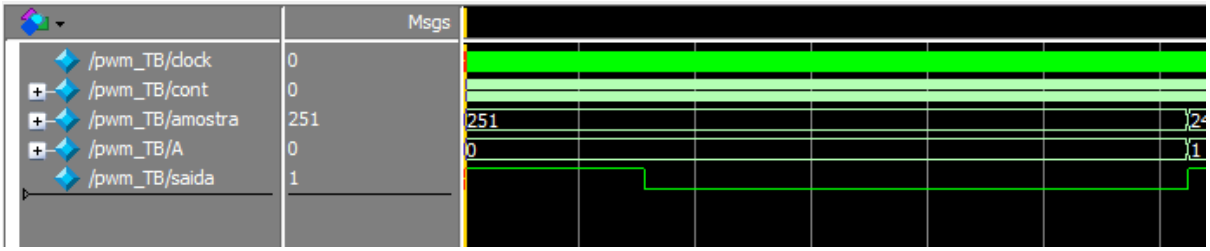
Assim, após o uso do zoom para visualizar as primeiras 12 amostras do sinal modulante codificado e, em seguida as 25 primeiras, percebe-se que:

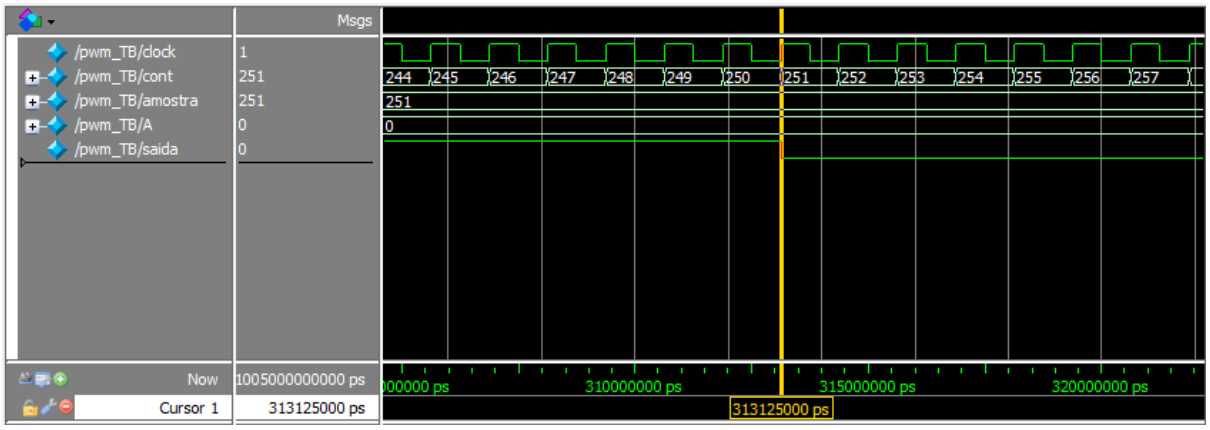




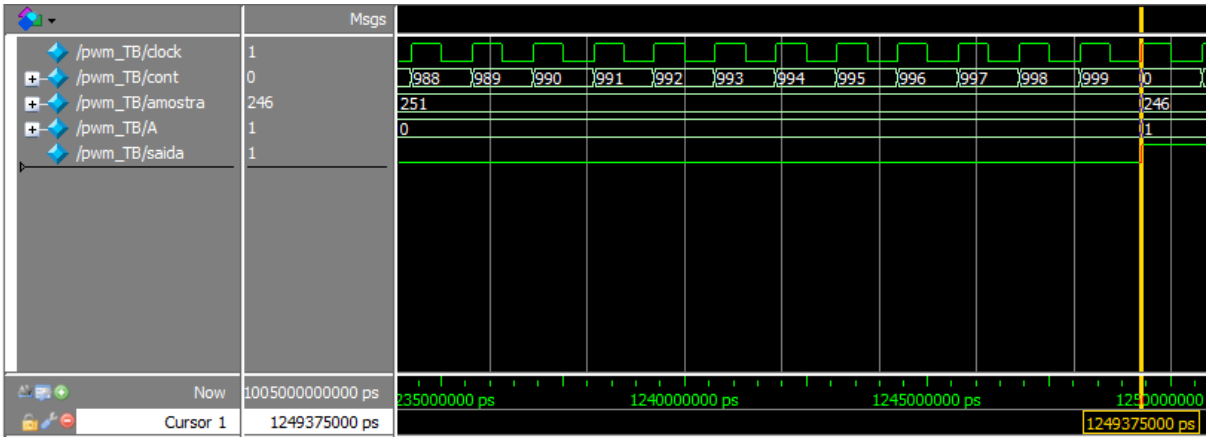
É possível observar que, conforme o valor absoluto da amostra vai diminuindo, o período em que o sinal PWM fica em estado alto também diminui, assim comprovando o que se esperava. Podemos verificar a validade do funcionamento deste módulo analisando o comportamento da onda de PWM em uma única amostra, através da comparação entre a proporção do tempo que a saída fica em nível lógico alto e o tempo total do período da onda PWM para essa amostra em específico e a proporção do valor absoluto da amostra com o valor máximo do contador igual a 999. Em teoria, eles deveriam ser iguais ou ao menos muito próximos.

Escolhendo a primeira amostra para fazer esse teste, tem-se que:





Ou seja, o sinal PWM muda de nível lógico em 0,313125 ms. Por fim, o contador extrapola no seguinte estante:



Ou seja, em 1,249375 ms, o valor ideal seria em 1,25 ms correspondente ao período de amostragem. No entanto, como se utilizou da simulação em gate level, levou-se em conta

desta forma os atrasos inerentes aos componentes utilizados para realizar a simulação além também da própria resolução utilizada.

De qualquer forma, as proporções ficam da seguinte forma:

$$\frac{valor\ da\ amostra}{valor\ máximo\ do\ contador} = \frac{251}{999} = 0{,}2512$$

$$\frac{tempo\ em\ nível\ lógico\ alto}{período\ da\ onda\ para\ a\ amostra} = \frac{0{,}313125\ ms}{1{,}249375\ ms} = 0{,}2506$$

Ou seja, houve uma diferença entre as proporções de apenas 0.0006, um valor relativamente bem baixo, comprovando assim a eficácia do módulo proposto.

Após a realização da implementação no Quartus, será implementada a mesma ideia no no Scilab.

Onde os mesmos valores das variáveis definidas anteriormente serão empregados, sem a aplicação de um valor “físico” em relação às operações realizadas para chegar ao resultado.

Inicialmente, o modo pelo qual o vetor das amostras do sinal modulante codificado não o permite fazer operações com cada uma delas em específico, dado que cada bit no total de 8 para cada amostra ocupa uma posição na matriz em específico. Para resolver esse problema faremos a conversão de cada uma das amostras binárias em números decimais, de forma que cada amostra ocupe apenas uma posição no vetor.

Isso é realizado da seguinte forma:

```
--> for i = 1:800
  >  Bin(i) = c(i,1)*(2^7) + c(i,2)*(2^6) + c(i,3)*(2^5) + c(i,4)*(2^4) + c(i,5)*(2^3)+c(i,6)*(2^2)+c(i,7)*(2^1)+c(i,8)*(2^0);
  > end
```

$(cmax + 1) * Número\ de\ amostras\ = 1000 * 800 = 800000$amostras,

dado que cada amostra terá uma comparação com um valor do contador que vai de 0 até 999, constrói-se um vetor de zeros chamado de PWM para armazenar cada um dos 800000 valores que ainda serão calculados.

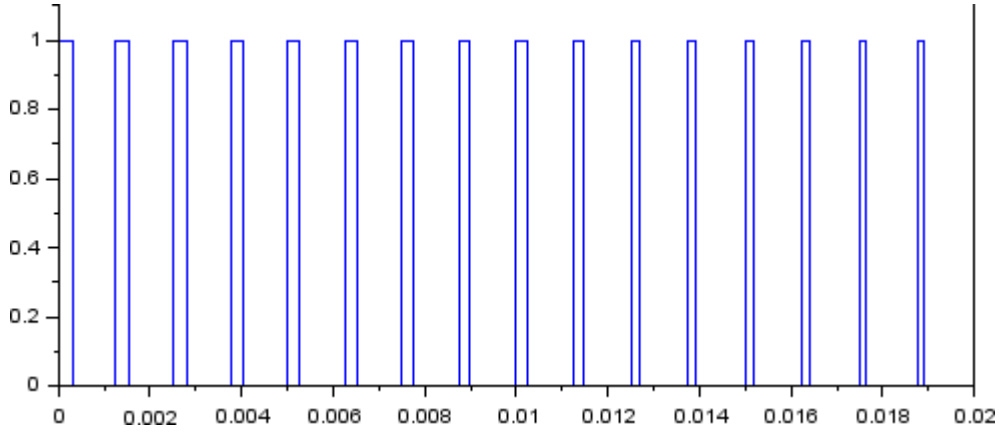
$$PWM = zeros\,(1{,}800000);$$

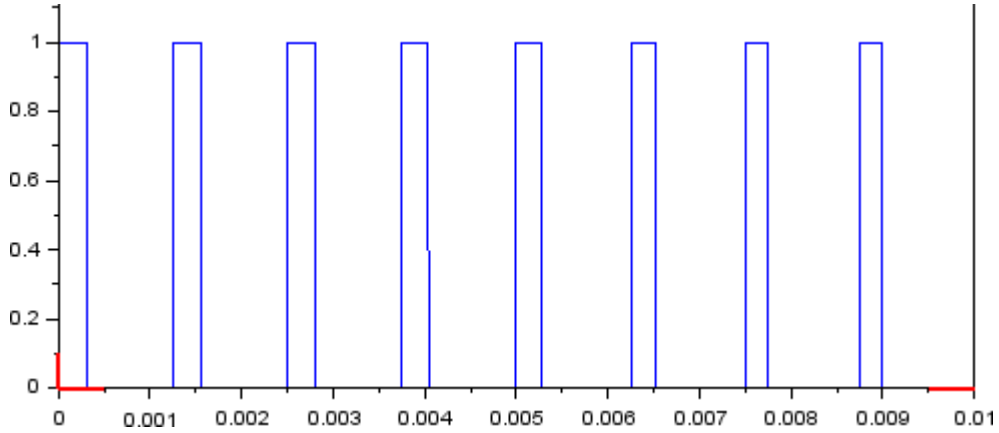
Agora aplica-se o algoritmo para fazer a comparação entre o valor das amostras e o contador. Desta forma será usado um “for” de 1 a 800 para varrer cada uma das amostras e outro “for” de 1 a 1000 para varrer todos os valores possíveis do contador.

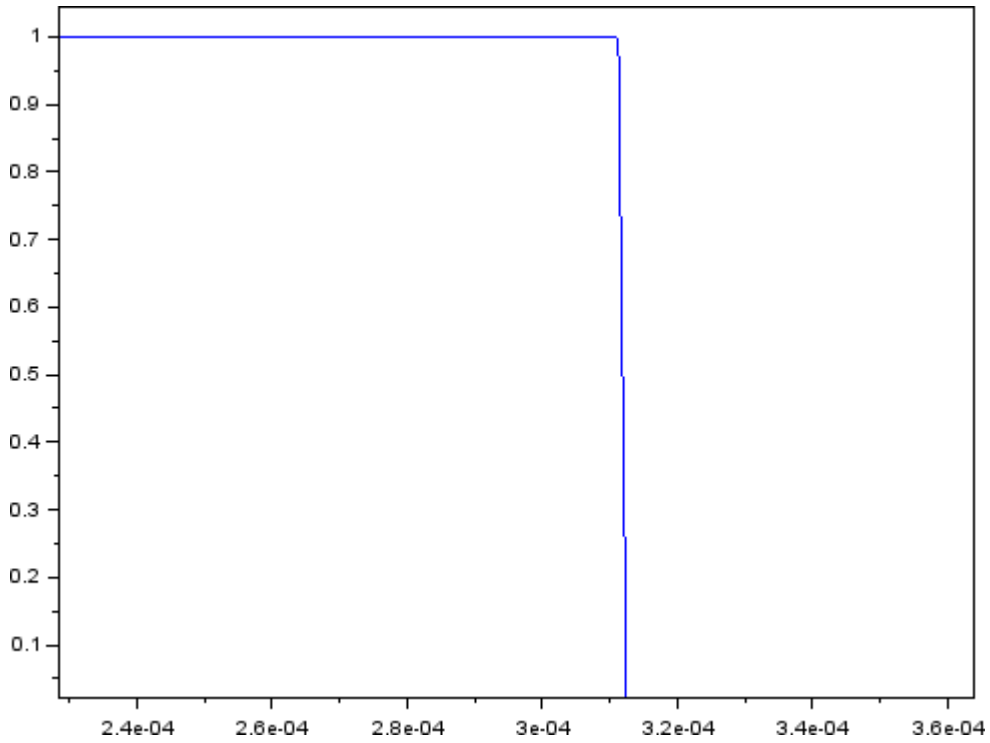
Além disso, é feito a comparação entre o valor do contador e o valor da amostra no índice indicado pelo for através de um “if”. Falta apenas a concatenação de cada operação de comparação com os valores abstraídos do PWM, pois, quando extrapolado o valor do contador passa-se para a seguinte posição do vetor de amostras e assim começa um novo ciclo de obtenção dos valores do PWM. Para resolver isso, usa-se de uma variável auxiliar a fim de corrigir a posição do PWM, isto é, conforme o índice da posição do vetor de amostras aumenta em 1 o índice de posição do vetor de PWM deverá ser incrementado em 1000, justamente o valor de extrapolação do contador. Este algoritmo é apresentado a seguir:

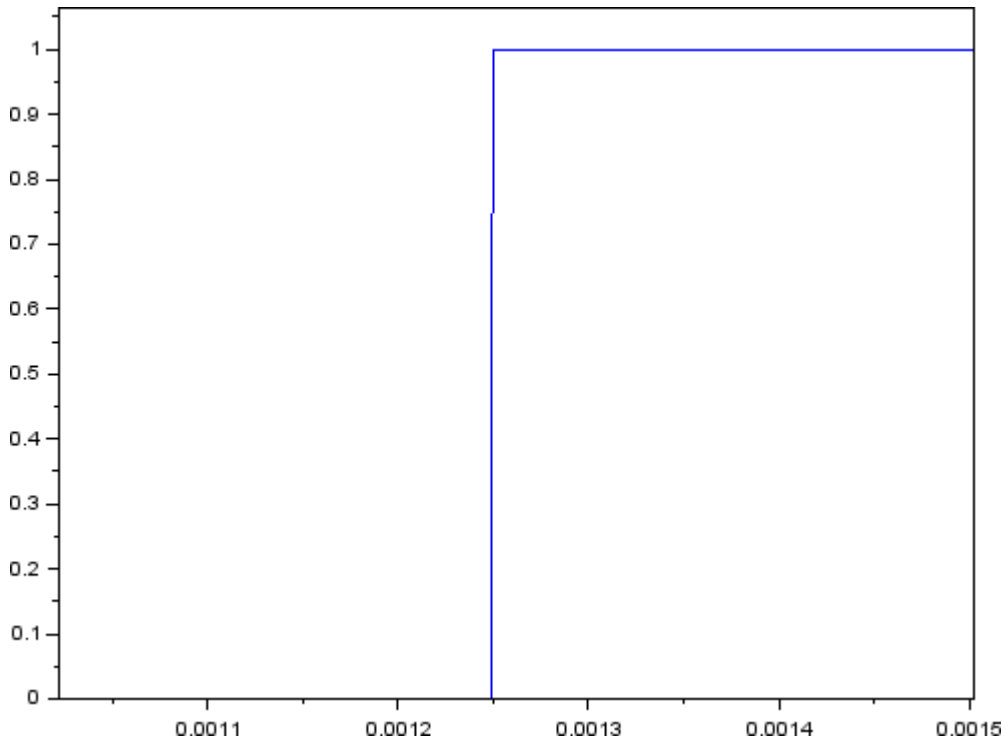
```
--> for i=1:800
  > for cont = 1:1000
  > j = i-1;
  > aux = cont + 1000*j;
  > if(cont < Bin(i))
  > PWM(aux)=1;
  > else
  > PWM(aux)=0;
  > end
  > end
  > end
```

```
fclk = 1000*800;

tclk = 1/fclk;

t = 0:tclk:1-tclk;

plot(t,PWM)
```


1
0.8
0.6
0.4
0.2
0
0
0.002
0.004
0.006
0.008
0.01
0.012
0.014
0.016
0.018
0.02


1
0.8
0.6
0.4
0.2
0
0
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008
0.009
0.01


1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
2.4e-04
2.6e-04
2.8e-04
3e-04
3.2e-04
3.4e-04
3.6e-04


1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.0011
0.0012
0.0013
0.0014
0.0015

Dessa forma, as proporções ficam da seguinte forma:

$$\frac{valor\ da\ amostra}{valor\ máximo\ do\ contador} = \frac{251}{999} = 0{,}2512$$

$$\frac{tempo\ em\ nível\ lógico\ alto}{período\ da\ onda\ para\ a\ amostra} = \frac{0{,}0003125}{0{,}00125} = 0{,}25$$

Concluímos que, houve uma leve diferença entre as proporções de 0.0012, um valor relativamente baixo, comprovando também assim a eficácia do algoritmo implementado. Além disso, pode-se notar também a correspondência entre o PWM implementado com o Quartus com o implementado no Scilab, com ambos dando resultados praticamente idênticos.