Grupo: **Felipe Kenzo Araki – 2022005633**
Glauber da Silva Moura – 2022000299
Isabelle Francine Guedes Romão – 2021029710
Kauan Barbosa da Silva - 2022010132
Pedro Andrade Gomes - 2022006926

RELATÓRIO PBLE04

Proposta SDR (Software Defined Radio)

A proposta do projeto é criar um rádio definido por software, comumente utilizando-se DDS (Direct Digital Synthesis) via verilog.

Etapa 1: Criação do sinal modulante via LUT (Look Up Table).

Parâmetros definidos: LUT com 2000 amostras, PWM com resolução de 0 - 24999, frequência de operação do PWM de 50MHz, FPB (filtro passa baixa) de primeira ordem feito via analpf() (SciLab) com valor de resistor de 100 ohm e capacitor 320 uF (220 uF paralelo com 100 uF);

Justificativas: As 2000 amostras da LUT foram escolhidas visando dois requerimentos: Valores suficientes para representar com qualidade o sinal modulante e utilização da frequência base da FPGA de 50 MHz, sem divisões de clock ou PLLs. A grosso modo, escolhemos duas mil amostras pois é um valor alto suficiente para representar o sinal modulante;

A contagem de 0 - 24999 (25000) do PWM foi escolhida pois a frequência base de operação da FPGA de 50 MHz, poderia ser utilizada diretamente, sem necessidade de circuitos divisores ou PLLs para realizar o ajuste de clock.

Os valores da LUT foram obtidos a partir de um script no Octave.

Para gerar o sinal modulante utilizando FPGA, foi empregada uma abordagem baseada em PWM (Pulse Width Modulation) e uma *lookup table* (LUT) contendo valores amostrados de um cosseno. Essa tabela é essencialmente um vetor com 2000 valores que representam a função cosseno dentro de um período completo, com valores variando de 1 até -1.

Esses 2000 pontos foram escolhidos como compromisso entre resolução adequada do sinal e limitações de hardware, permitindo manter a frequência original de operação da FPGA, que é de 50 MHz. Como a FPGA não trabalha diretamente com números negativos nem com números decimais, os valores da LUT foram ajustados para um intervalo positivo. Para isso, os valores do cosseno, originalmente no intervalo [-1, 1], foram escalonados para o intervalo [1, 25000], utilizando o valor máximo de 24999 como referência (sendo 25000 o total possível de contagem do PWM).

Diferentemente de uma LUT tradicional de cosseno, a tabela construída contém os valores já ajustados para controle direto do ciclo de trabalho (duty cycle) do PWM, ou seja, cada valor representa diretamente o tempo em que a saída do PWM ficará em nível alto.

Durante a execução, ao iniciar o código na placa e resetar o sistema, o índice da LUT começa em zero e percorre sequencialmente até 1999. A cada ciclo de PWM, o valor correspondente da LUT é lido e enviado como parâmetro para configurar o duty cycle. Por exemplo, se o valor lido for 24508, o sinal PWM permanecerá em nível alto até esse valor de contagem e, após isso, ficará em nível baixo até completar o período de 25000. Ao fim de cada ciclo do PWM, o índice da LUT é incrementado e o próximo valor é utilizado. Quando o índice atinge o fim da tabela, ele é resetado, voltando para o início, gerando assim um sinal modulante contínuo.

Com a geração da forma de onda pronta, o sinal PWM é encaminhado para um filtro passa-baixas na saída da FPGA. O filtro foi montado com um resistor de 100ohms e capacitores de 220µF e 100µF em paralelo (totalizando aproximadamente 320µF, que era o valor originalmente desejado). Este filtro suaviza o sinal PWM, removendo suas componentes de alta frequência e permitindo obter, na saída, o sinal analógico correspondente ao modulante.

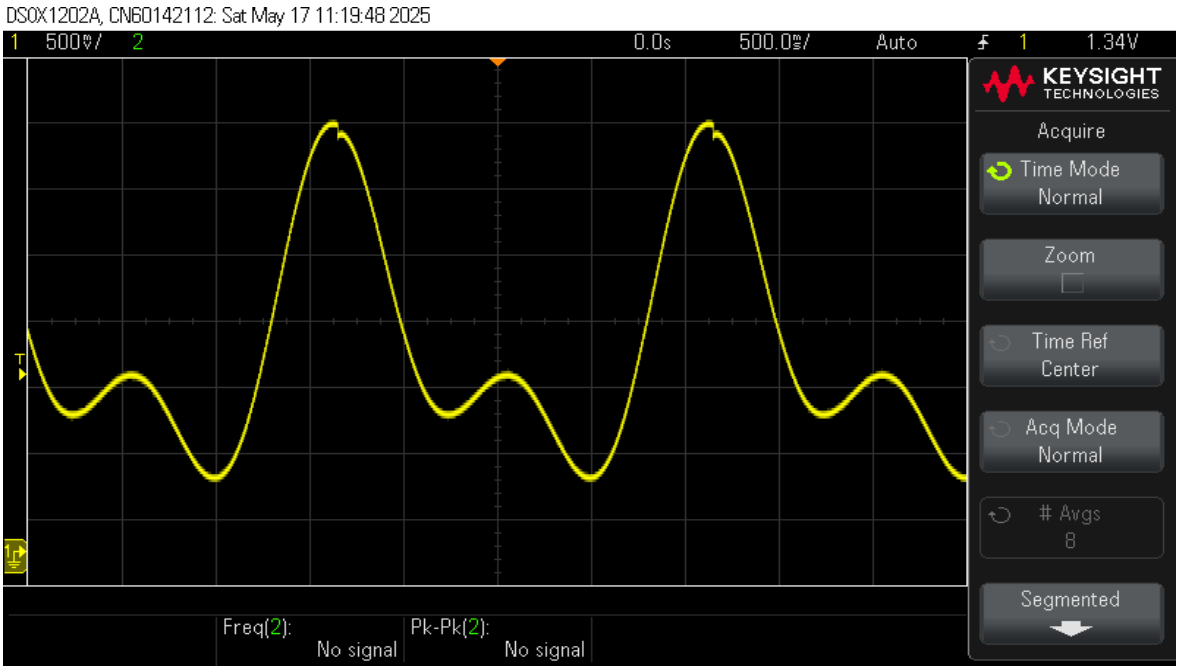


Imagem: *PrintScreen extraído do osciloscópio via USB representando o sinal modulante em sua forma filtrada/analógica;*

O resultado final é um sinal analógico gerado com boa resolução e fidelidade, utilizando recursos da própria placa FPGA com frequência base de 50 MHz e um PWM com período de 25000 ciclos, sincronizado com a LUT de 2000 amostras.

Conversão analógico para digital de *m*(*t*) com o MCP3002 colocando os bits sequencialmente na FPGA

O sistema em questão realiza a aquisição de dados em um fluxo serial de bits, processando-os em amostras de 8 bits, descartando os 2 bits menos significativos de uma amostra original de 10 bits para manter uma resolução adequada. O processo é controlado por sinais de clock em diferentes frequências, garantindo a sincronização e o tempo de amostragem necessário.

No fluxo de amostragem temos:

- Entrada: Um fluxo serial de 10 bits por amostra (proveniente de um conversor analógico-digital, como o MCP3002).
- Saída: Uma amostra de 8 bits, obtida descartando os 2 bits menos significativos para evitar perda excessiva de resolução.

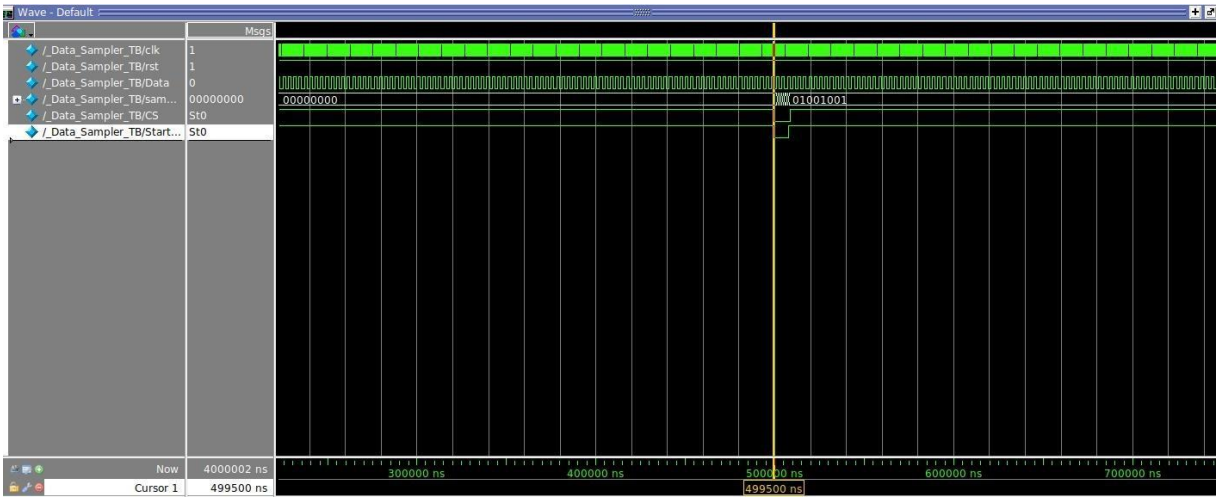
A cada pulso de clock, 1 bit é deslocado para a saída. Após 8 pulsos de clock, uma amostra completa de 8 bits é formada. Como a entrada original tem 10 bits, os últimos 2 bits são ignorados para simplificação do processamento.

Em relação ao tempo, temos:

- A cada 5 ms, uma amostra completa (8 bits) deve ser capturada. Isso resulta em uma taxa de amostragem de 200 Hz (1 amostra a cada 5 ms).
- Clock Base (Principal): 50 MHz (usado para sincronização geral da placa).
- Clock do Conversor (MCP): 50 kHz (frequência de alimentação do MCP3008).
- Clock Interno (Processamento): 200 Hz (utilizado para controle interno da placa, como conversão e processamento de dados).

O chip (MCP3002) é ativado no início do processo de amostragem. Após a captura dos 8 bits necessários, o chip é desligado para economizar energia e evitar leituras desnecessárias. Durante o processo, o sistema opera com 3 clocks distintos:

- Clock Base (50 MHz): Sincronização principal da placa.
- Clock do MCP (50 kHz): Controla a comunicação com o conversor A/D.
- Clock Interno (200 Hz): Gerencia o processamento interno (filtragem, conversão, etc.).



Abaixo temos o código utilizado para a simulação:

```
module _Data_Sampler
(
input clk,            //deve ser alimentado com 50khz
input rst,
input Data,

output reg CS,
output reg StartConv,                  //flag de controle de conversao
output reg [7:0] sampler    //registro de dados
);

reg [3:0] countSamp;        //contador de controle de conversao
reg [7:0] countCtrl;        //contador de controle de frequencia de taxa de amostragem = 200Hz


//always de controle: contador
always@(posedge clk or negedge rst)
        begin
                if(~rst)
        begin
```

```
countCtrl <= 0;
StartConv <= 1;   //reset em nivel alto
                        end else
            begin
//se bate o tempo do periodo de 5ms, reseta o contador e habilita a conversao
                        if(countCtrl == 249)
```

```
            begin countCtrl <= 0;
        StartConv <= 0;

                                                                                                    begin


end else
        //quando a conversoa terminar, 10 ciclos de clock, desabilita a conversao
                        if(countCtrl == 7)
            begin

        StartConv <= 1;
        countCtrl <= countCtrl + 1;
            end
         //incrementa
             else
        countCtrl <= countCtrl + 1;
            end
            end
            end

        //o mcp atualiza na borda de descida
        always@(posedge clk or negedge StartConv or negedge rst)
            begin
                if(~rst)
            begin
        CS <= 1;                                //coloca CS em nivel alto=desabilita conversor
        sampler <= 0;      //zera o registro de controle
        countSamp <= 0;//zera o contador de controle
            end else
            begin
                if(~StartConv)
             begin
                if(countSamp == 0)
             begin
        CS <= 0;


        countSamp <= countSamp + 1; end
            else
            begin
        CS <= 0;
        countSamp  <=  countSamp  +  1;
        sampler <= (sampler << 1) | Data;
            end
            end
            else
        //se StartConv for desabilitado, zera o contador e desabilita CS
            begin
        countSamp <= 0;
        CS <= 1;
        //sampler <= 0;
        //nao preciso zerar o contador, a interacaoo natural desse always
        //com o outro fara isso naturalmente
        //depois disso, sampler contera o valor dos 8 bits mais significativos do
        //MCP3002 referente ao sinal convertido end
            end end
```

```
endmodule
```

O sistema realiza uma amostragem eficiente de dados em um fluxo serial, convertendo amostras de 10 bits em 8 bits a cada 5 ms. A utilização de múltiplos clocks (50 MHz, 50 kHz e 200 Hz) permite um controle preciso do tempo de amostragem e processamento interno, enquanto a desativação do chip após a captura garante economia de energia. A remoção dos 2 bits menos significativos mantém uma resolução adequada para a aplicação proposta.

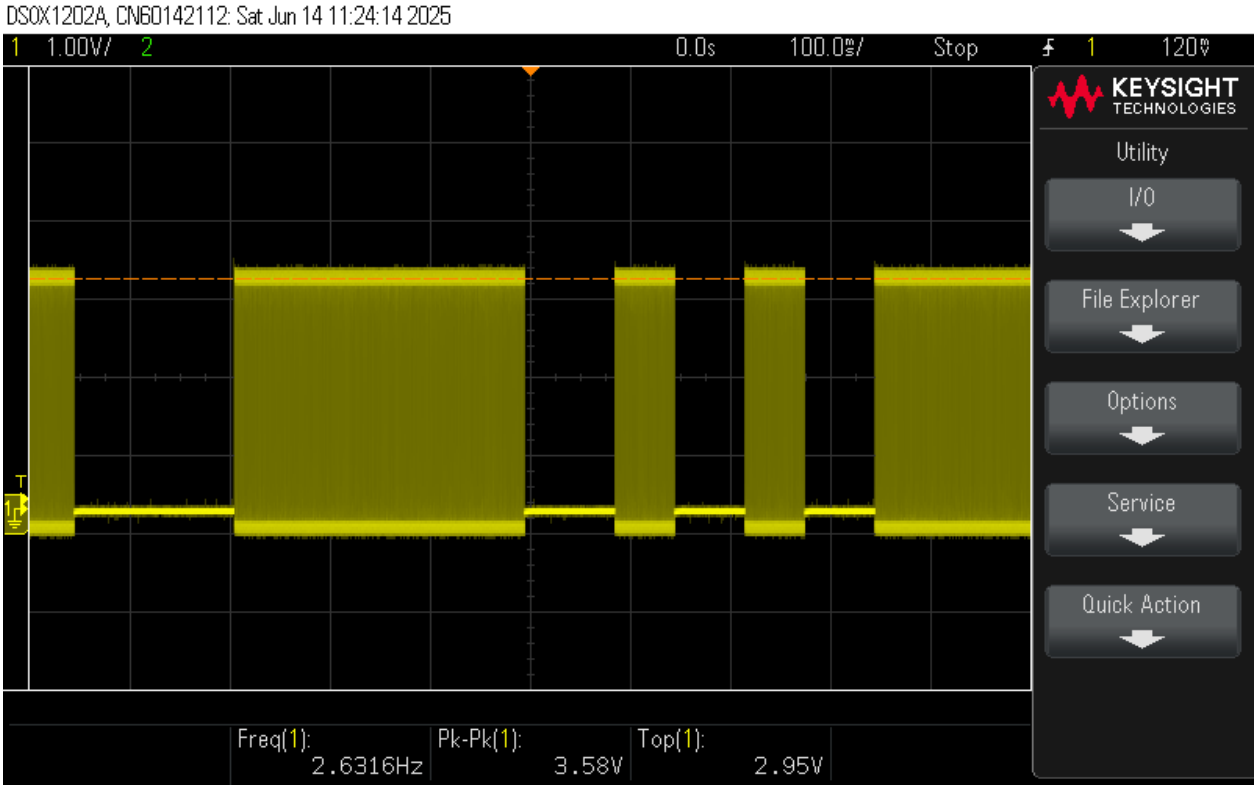
Representação PWM do sinal BASK

Nesta etapa o sistema opera com base em um registro contendo uma amostra completa de 8 bits, processando-a em um ciclo de 5 ms (equivalente a uma frequência de 200 Hz). A saída final é gerada combinando um sinal de clock de 100 kHz com o bit mais significativo do registro de amostras. Ele armazena o valor inteiro da amostra (8 bits), proveniente da etapa anterior de aquisição e é atualizado periodicamente conforme a lógica de controle.

Foi utilizado o código abaixo:

```
module _BASK_PWM
(
	input clk,			//frequencia base de 50meg
	input rst,
	input [7:0] sampler,		//vem direto do shift register do sata sampler
	input Allow,				//sinal de frequencia de 200hz

	output wire BASK
);

	(*keep*) reg [9:0] cnt;		//registro de atualizacao
	reg [9:0] read;				//registro de leitura de fato
	reg [14:0] clk16Gen;		//contador pra gerar o clock de 1600hz
	reg clk16;
	reg [8:0] clk100Gen;		//contador pra gerar o clock de 100khz
	reg clk100;

	and bask_out(BASK, clk100, cnt[9]);

//gera o clock de 16000
	always@(posedge clk or negedge rst)
		begin
			if(~rst)
		begin
			clk16Gen <= 0;
			clk16 <= 0;
				end
			else if(clk16Gen == 31249)
		begin
			clk16 <= ~clk16;
```

```
                    clk16Gen <= 0;
                        end
                    else clk16Gen <= clk16Gen + 1;
                        end
//gera o clock de 100khz

always@(posedge clk or negedge rst)
            begin
                    if(~rst)
            begin
                    clk100Gen <= 0;
                    clk100 <= 0;
                        end
                    else if(clk100Gen == 499)
            begin
                    clk100 <= ~clk100;
                    clk100Gen <= 0;
                        end
                    else clk100Gen <= clk100Gen + 1;
                        end

//Allow é um sinal que majoritariamente fica em nivel alto, ativa em baixo
//atualiza o registro de leiura com o valor da nova conversao
always@(negedge Allow or posedge clk16 or negedge rst)
            begin
                    if(~rst) cnt <= 0;
                    else if(~Allow) if(~clk16) cnt <= {sampler, 2'b0};
                    else cnt <= cnt;
                    else cnt <= (cnt <<
                        1); end
        endmodule
```



O sistema emprega três blocos “always” para controle de temporização e geração de sinais:

1. Clock de 1,6 kHz:
    - Responsável pelo deslocamento (*shift*) do registro interno.
    - Determina a taxa em que os bits são processados.
2. Clock de 100 kHz:
    - Utilizado exclusivamente para a saída PWM.
    - Se o bit atual (MSB do registro CNT) for 1, a saída reproduz este clock.
    - Se o bit for 0, a saída permanece em nível baixo.
3. Atualização do Registro:
    - Sensível a duas bordas:
        - Borda de atualização: Carrega um novo valor no registro.
        - Borda de deslocamento (*shift*): Realiza o deslocamento do registro a 1,6 kHz.

A lógica de sinal de controle e saída temos o sinal “Allow” onde ele é proveniente de um módulo externo, indica quando o sistema está em modo de conversão ou modificação.

- Nível Baixo “0”: Indica que uma conversão está em andamento (registro não deve ser alterado).
- Nível Alto “1”: Indica que o sinal pode ser modificado (novo valor pode ser carregado no registro).

Isso ocorre no registro “CNT”, quando “CLK16” ativa:

- “Allow = 1” → O registro é atualizado com um novo valor.
- “Allow = 0” → O registro é deslocado (shift).

A saída final é produzida por uma porta lógica AND com as seguintes entradas:

- CLK100K (100 kHz): Sinal de clock de alta frequência.
- Bit mais significativo (MSB) de CNT:
    - Se CNT[MSB] = 1 → Saída = CLK100K (pulsos em 100 kHz).
    - Se CNT[MSB] = 0 → Saída = 0 (nível baixo).

O sistema descrito implementa uma modulação BASK utilizando PWM, onde:

- O registro CNT armazena e processa amostras de 8 bits.
- Um clock de 1,6 kHz controla o deslocamento dos bits.
- Um clock de 100 kHz gera a portadora PWM, ativada apenas quando o bit mais

significativo de CNT é 1.

O sinal “Allow” sincroniza a atualização do registro, garantindo que novas amostras só sejam carregadas quando o sistema estiver pronto. Essa abordagem permite uma modulação eficiente, com controle preciso da saída BASK em função dos dados armazenados no registro CNT.

Sinal BASK usando FPF

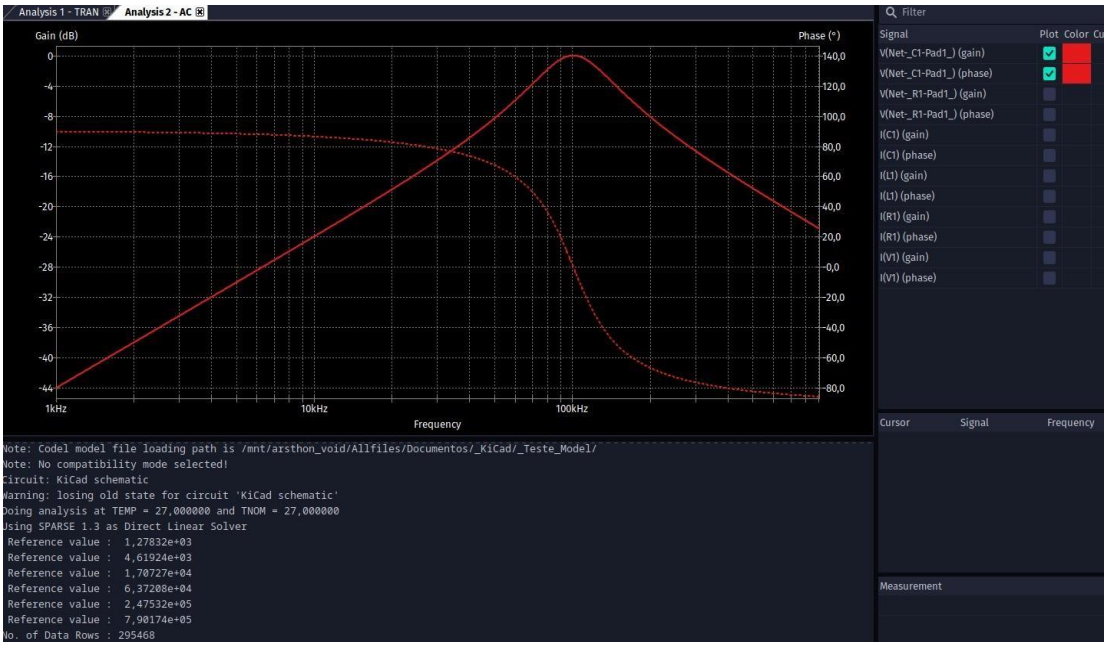
Nesta etapa o circuito foi projetado com os seguintes componentes fixos:

- Indutor (L): 1 mH (valor fixado)
- Resistor (R): 100 Ω (valor inicial)

O capacitor (C) foi calculado para atender às especificações do filtro, resultando em:

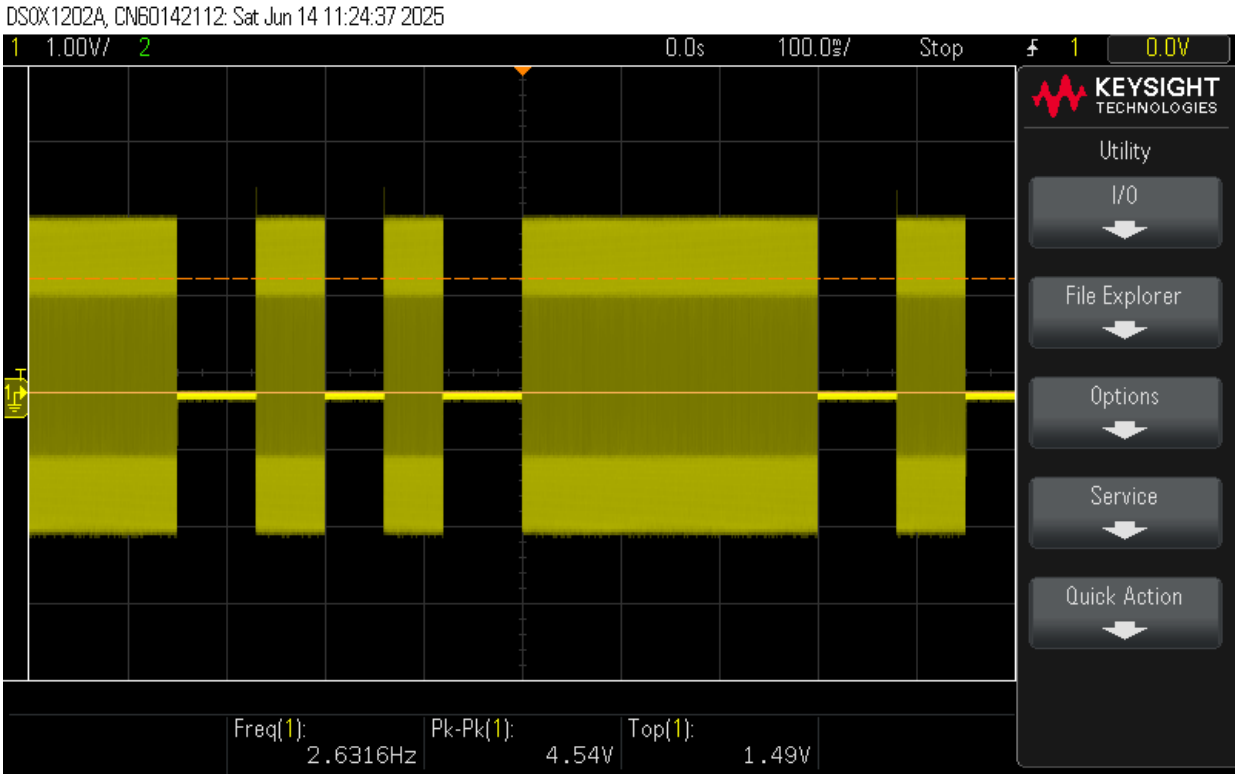
- Capacitor (C): 2,5 nF

Após análise por simulação, verificou-se que o comportamento do filtro poderia ser otimizado com o seguinte ajuste:



O filtro apresenta:

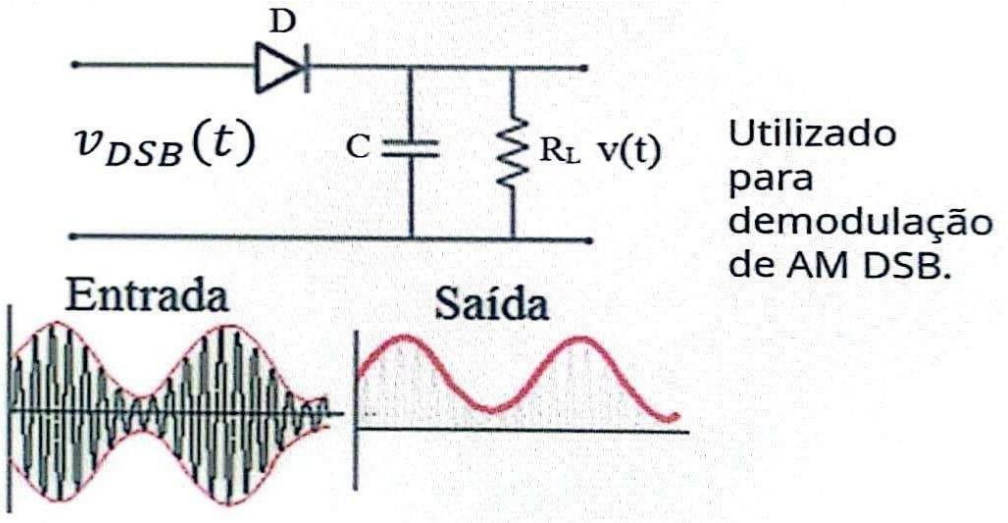
- Pico de ressonância (1p) acentuado na frequência de 100 kHz, conforme projetado.



A modificação do resistor para 1 kΩ melhorou significativamente a resposta do filtro, mantendo a ressonância bem definida em 100 kHz, conforme requerido pelo projeto. O ajuste garantiu um desempenho mais adequado às especificações desejadas.

Circuito detector de envelope para demodular o sinal BASK

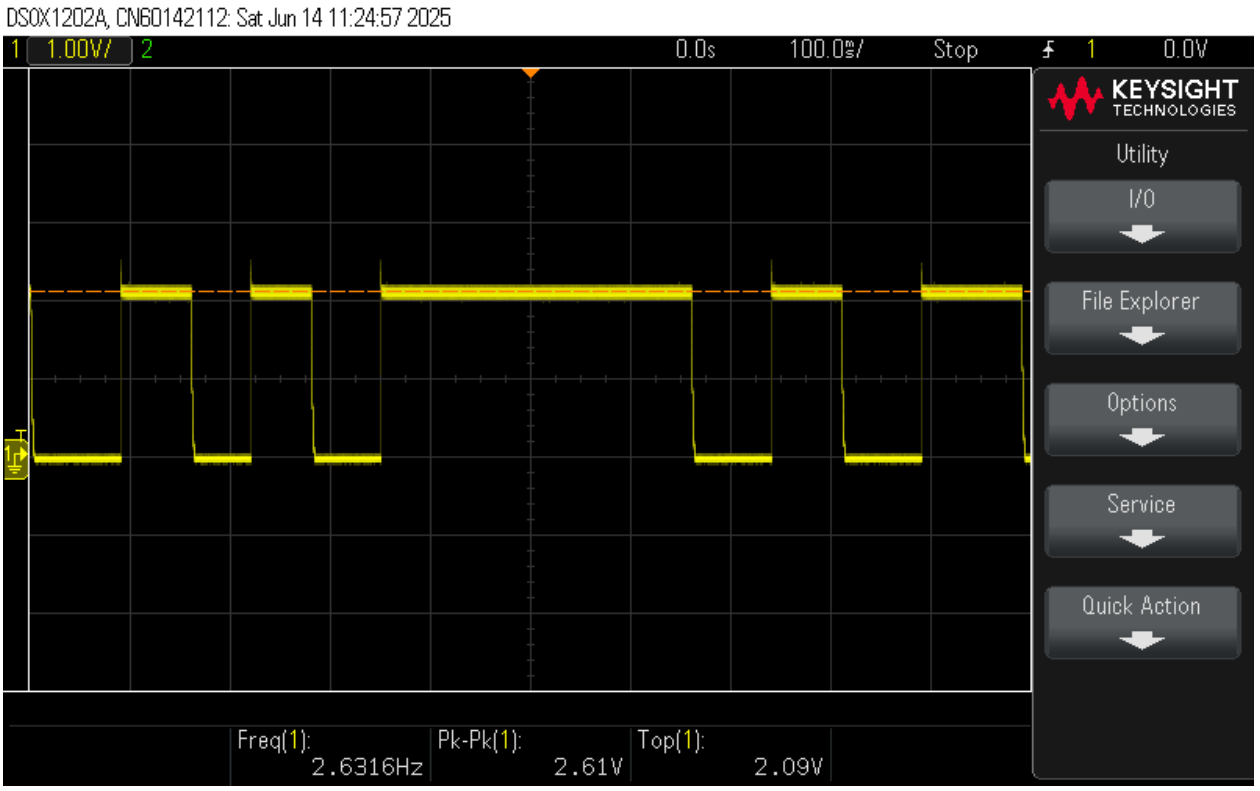
Nesta etapa, foi pedido a implementação de um circuito detector de envelope, devido às suas características adequadas para a faixa de frequência operacional do sistema. Este circuito foi projetado para recuperar o sinal modulado em BASK (Binary Amplitude Shift Keying) de forma eficiente.



O cálculo dos componentes foi fundamentado para que estabelecem a relação entre a constante de tempo do circuito ($\tau = RC$) e a taxa de símbolos do sinal BASK.

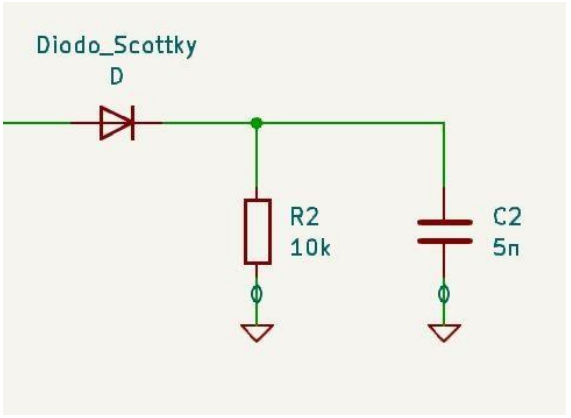
- Resultado da Equação: Obteve-se um valor aproximado de 50 μs para a constante de tempo τ.
- Escolha dos Componentes:
  - Resistor (R): Selecionado arbitrariamente como 10 kΩ (valor comercial comum e adequado para a aplicação).
  - Capacitor (C): Calculado para manter $\tau \approx 50$ μs:
    $C = \tau R = 50 \times 10^{-6} 10 \times 10^{3} = 5$ nF $C = R\tau = 10 \times 10^{3} 50 \times 10^{-6} = 5$ nF
  - Produto RC Resultante: 10 kΩ × 5 nF = 50 μs, atendendo ao requisito teórico.

$$\frac{1}{f_c} \ll R_L C < \frac{2\pi}{W}$$



O circuito tem como função recuperar o sinal BASK extraindo o envelope do sinal modulado, reconstruindo a forma de onda original. E a integração com a FPGA onde a FPGA recebe os bits demodulados em uma de suas entradas e, realiza o processamento digital retransmitindo os dados para a próxima etapa.

Os bits demodulados na saída do detector devem corresponder exatamente à sequência original aplicada na modulação BASK. A correta operação do circuito será confirmada pela comparação entre os bits transmitidos e os bits recuperados.



O detector de envelope projetado com R = 10 kΩ **e** C = 5 nF apresenta constante de tempo adequada (50 μs) para a demodulação eficiente do sinal BASK. Sua integração com a FPGA permite a recuperação fiel da informação digital original, assegurando que a sequência de bits de saída corresponda à entrada do sistema. A escolha dos valores comerciais para os componentes garantiu viabilidade de implementação sem comprometer o desempenho teórico esperado.

Conversão digital para analógico dos bits obtidos do sinal BASK usando PWM e FPB de 1ª ordem

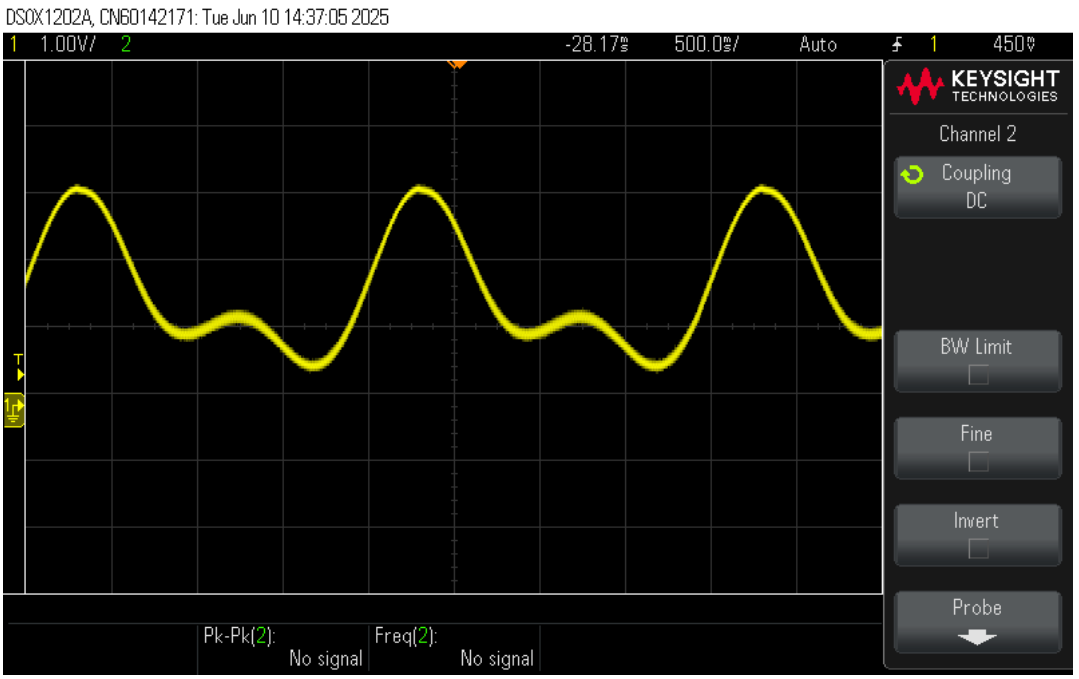
Nesta etapa, realizamos as correções nos módulos responsáveis pela amostragem do sinal. Em seguida, iniciamos uma nova fase do projeto, implementando um módulo gerador de PWM e também o módulo de leitura dos bits provenientes do filtro detector de voltagem.

Estamos trabalhando com um sinal BASK modulado, cuja demodulação é feita por um detector de voltagem. A saída desse filtro são os bits do sinal original. Para isso, desenvolvemos um módulo capaz de ler esses bits e, com base neles, gerar o PWM. A leitura dos bits ocorre na mesma frequência em que são gerados, ou seja, 1,6 kHz, enquanto o PWM opera a 204 kHz.

A cada bit recebido, ele é inserido em um registrador e deslocado até que os 8 bits estejam completos. Ao final dos 8 ciclos de 1,6 kHz, o registrador está totalmente preenchido. O valor armazenado é então enviado ao registrador de duty cycle do PWM, que está perfeitamente sincronizado com o último bit capturado.

Após o carregamento do último bit, o PWM finaliza sua contagem — que vai de 0 a 1019 (totalizando 1020 ciclos, pois os dois bits menos significativos são desconsiderados). Assim, o PWM conclui sua contagem exatamente quando o registrador enche, atualiza o duty cycle, zera o registrador e reinicia o processo de amostragem.

Por fim, o sinal PWM é direcionado ao pino de saída, passando por um filtro passa-baixa. O resultado final do sinal filtrado é apresentado na imagem abaixo.



Modulação 4ASK

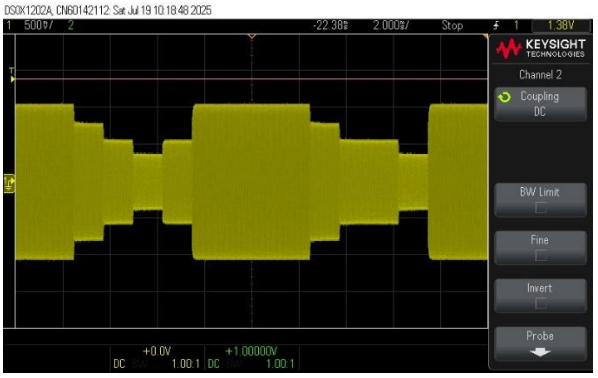
Nesta etapa, utilizamos a modulação BASK já implementada anteriormente e, a partir dela, desenvolvemos o módulo para gerar a modulação 4ASK.

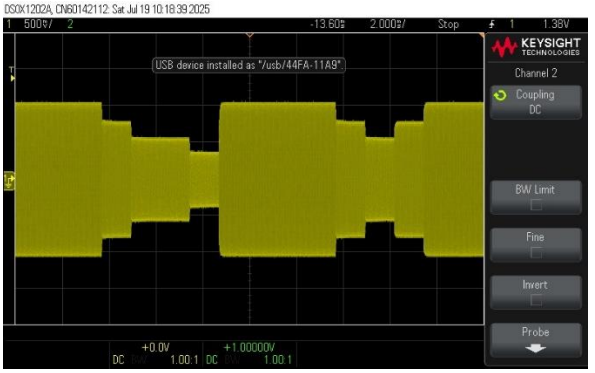
O módulo de BASK funciona de forma simples: ele verifica o bit mais significativo presente no registrador. Se esse bit for 0, a saída é 0 (sem portadora); se for 1, a saída é uma portadora senoidal de 100 kHz — caracterizando a modulação BASK.

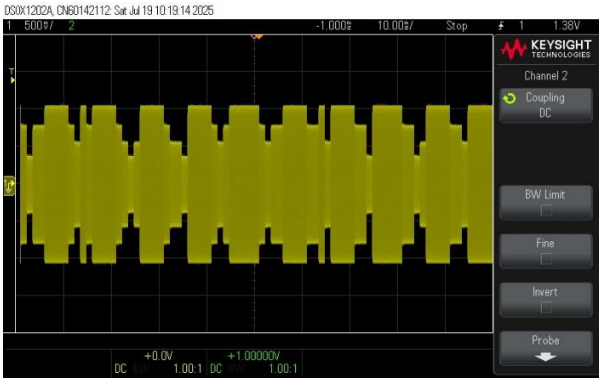
Já no caso da modulação 4ASK, existem quatro níveis possíveis de amplitude, o que exige uma abordagem um pouco diferente. Para isso, foi adicionado um novo bloco always no código, que agora analisa dois bits (os dois mais significativos do registrador), em vez de apenas um. Além disso, o deslocamento (shift) é feito de dois em dois bits.

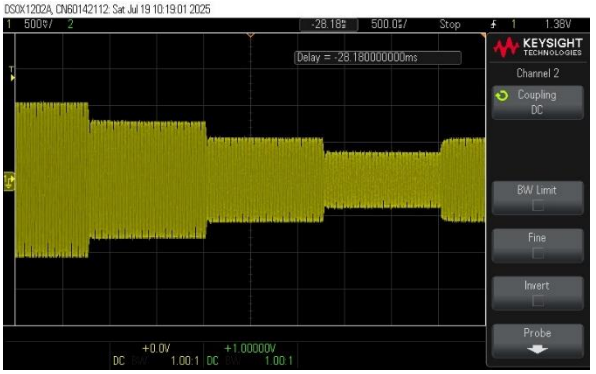
Cada combinação desses dois bits resulta em uma saída com amplitude diferente:
00: menor amplitude;
01: amplitude intermediária;
10: amplitude maior;
11: amplitude máxima.

Dessa forma, o sinal 4ASK apresenta quatro níveis distintos de amplitude, como é possível observar nas imagens abaixo. Importante ressaltar que, mesmo no menor nível, a amplitude não é nula, o que difere da modulação BASK onde a ausência de portadora corresponde ao nível mais baixo.





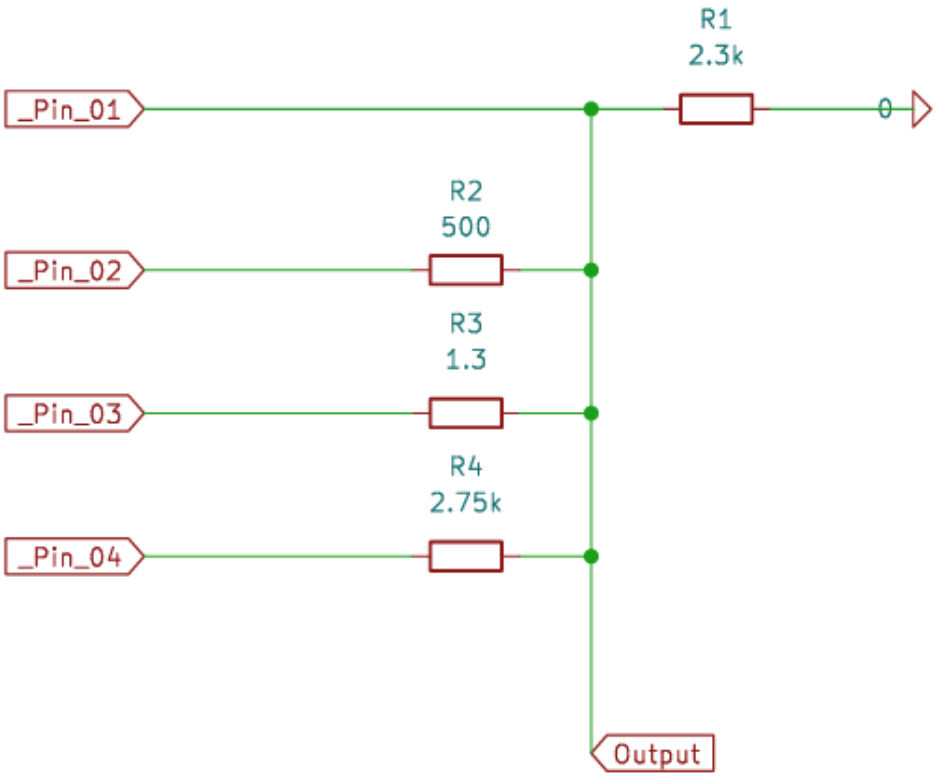




A sugestão inicial era variar o duty cycle da portadora para obter diferentes amplitudes. No entanto, tanto na simulação quanto nos testes práticos, observamos que essa abordagem não funciona como esperado. Assim como em tentativas anteriores, percebemos que a amplitude do sinal permanece praticamente inalterada; o que se altera, na verdade, é o componente DC da onda — o que não atende ao objetivo da modulação 4ASK. Essa abordagem, portanto, está incorreta e foi descartada.

Para implementar a solução correta, utilizamos quatro pinos da FPGA, correspondendo às quatro possíveis combinações de dois bits. Para cada conjunto de bits, um pino específico é ativado como saída. Em cada um desses pinos, foi conectado um divisor de tensão apropriado. É por meio desses divisores de tensão que conseguimos gerar os diferentes níveis de amplitude desejados para o sinal 4ASK.

Os valores dos resistores foram calculados de forma precisa para que as amplitudes resultantes estivessem de acordo com os níveis definidos pela modulação. Com isso, obtivemos os quatro níveis distintos de forma confiável e controlada.



Cada pino da FPGA é responsável por uma saída específica, mas todos geram a mesma frequência de 100 kHz. A diferença está em qual pino está ativo em determinado momento. Cada pino está conectado a um divisor resistivo diferente, o que gera uma amplitude distinta para o sinal, dependendo do par de bits de entrada.

Quando um dos pinos está ativo (emitindo a portadora), os outros três permanecem em alta impedância. Isso é fundamental, pois se os pinos inativos estivessem em nível lógico baixo (0), haveria fluxo de corrente indesejado, o que comprometeria o funcionamento do circuito e distorceria as amplitudes.

Ou seja, a cada instante o sinal de 100 kHz sai por apenas um pino, e os demais ficam em alta impedância, evitando interferências. Todos esses pinos convergem para um mesmo ponto de saída física, após passarem por seus respectivos divisores de tensão.

Dessa forma, conseguimos gerar os quatro níveis de amplitude característicos da modulação 4ASK — não pela variação da portadora, mas sim por meio de hardware controlado, usando divisores de tensão e o controle adequado de impedância nos pinos da FPGA.