UNIVERSIDADE FEDERAL DE CAMPINA GRANDE

CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

PROGRAMA DE CAPACITAÇÃO EM MICROELETRÔNICA

DAVI LIMA DE MEDEIROS

davi.medeiros@ee.ufcg.edu.br

CAMPINA GRANDE

14/10/2022

SUMÁRIO

1. Identificador de Sequência

Para o identificador de sequência, era preciso identificar a presença dos 4 bits menos significativos da sequência de bits recebidos serialmente. Esses bits recebidos serialmente são referentes à um nome, e essas letras devem ser interpretadas a partir do código ASCII.

"American Standard Code for Information Interchange" é um sistema de representação de letras, algarismos e sinais de pontuação e de controle, através de um sinal codificado em forma de código binário desenvolvido em 1960. Portando, o programa deveria receber um nome serialmente, identificar a primeira letra e mandar para a saída a representação da letra em 4 bits a partir da tabela a seguir.

Bin	Oct	Dec	Hex	Sinal
0100 0000	100	64	40	@
0100 0001	101	65	41	A
0100 0010	102	66	42	B
0100 0011	103	67	43	C
0100 0100	104	68	44	D
0100 0101	105	69	45	E
0100 0110	106	70	46	F
0100 0111	107	71	47	G
0100 1000	110	72	48	H
0100 1001	111	73	49	I
0100 1010	112	74	4A	J
0100 1011	113	75	4B	K
0100 1100	114	76	4C	L
0100 1101	115	77	4D	M
0100 1110	116	78	4E	N
0100 1111	117	79	4F	O
0101 0000	120	80	50	P
0101 0001	121	81	51	Q
0101 0010	122	82	52	R
0101 0011	123	83	53	S
0101 0100	124	84	54	T
0101 0101	125	85	55	U
0101 0110	126	86	56	V
0101 0111	127	87	57	W
0101 1000	130	88	58	X
0101 1001	131	89	59	Y
0101 1010	132	90	5A	Z
0101 1011	133	91	5B	[
0101 1100	134	92	5C	\
0101 1101	135	93	5D	]
0101 1110	136	94	5E	^
0101 1111	137	95	5F	_

Bin	Oct	Dec	Hex	Sinal
0110 0000	140	96	60	`
0110 0001	141	97	61	a
0110 0010	142	98	62	b
0110 0011	143	99	63	c
0110 0100	144	100	64	d
0110 0101	145	101	65	e
0110 0110	146	102	66	f
0110 0111	147	103	67	g
0110 1000	150	104	68	h
0110 1001	151	105	69	i
0110 1010	152	106	6A	j
0110 1011	153	107	6B	k
0110 1100	154	108	6C	l
0110 1101	155	109	6D	m
0110 1110	156	110	6E	n
0110 1111	157	111	6F	o
0111 0000	160	112	70	p
0111 0001	161	113	71	q
0111 0010	162	114	72	r
0111 0011	163	115	73	s
0111 0100	164	116	74	t
0111 0101	165	117	75	u
0111 0110	166	118	76	v
0111 0111	167	119	77	w
0111 1000	170	120	78	x
0111 1001	171	121	79	y
0111 1010	172	122	7A	z
0111 1011	173	123	7B	{
0111 1100	174	124	7C	|
0111 1101	175	125	7D	}
0111 1110	176	126	7E	~

Figura 01 – Tabela ASCII para o alfabeto.

Para a resolução da primeira parte precisei criar, assim como criei em outros desafios, uma chave de validação (normalmente nomeada de *valid*) para determinar se o sistema está valido ou não. A partir do momento que eu indico para o sistema que a validade do sistema é

igual a 1, é que eu permito que o programa receba os valores que eu inseri serialmente. Também utilizei de funções com Strings para facilitar o processo (Sites que utilizei para estudo estão nas referências bibliográficas).

O código simplesmente seleciona a primeira letra dos valores colocados serialmente como String, passa esse valor para minha variável de entrada, e na função apresenta o valor de saída como essa letra em binário, assim como na tabela ASCII. Para visualização completa do código, basta entrar no link: https://www.edaplayground.com/x/s7g9 .

```
Validity of the process: 0 ,Output in binary: 0000
Validity of the process: 1 ,Output in binary: 0011
```

Figura 02- Saída da primeira parte da questão (Entrada serial = carlos).

Para a segunda parte do desafio, como foi dito no texto, *"Modifique o circuito digital para que este identifique a presença de uma sequência de 4 bits quaisquer em um conjunto de bits recebidos serialmente. A sequência a ser identificada será definida pelo usuário através de uma entrada específica de 4 bits (paralela)"* acreditei que deveria criar outro programa, contudo para os próximos desafios fiz todas as etapas em um mesmo programa.

Criei apenas um checador de sequência para identificar os valores de entradas da seguinte forma, ao adicionar um valor serialmente, eu desloco um bit do meu *"check_seq"* para a direita e adiciono o novo valor no lugar. Assim eu consigo determinar os valores que eu coloquei na entrada serial com a sequência que determinei de forma paralela. Assim a cada pulso de clock o sistema testa as entradas com o valor da sequência que eu preciso, até a sequência ser atingida e o valor do processo ir para 1. Para visualização completa do código, basta entrar no link: https://www.edaplayground.com/x/GFRC .

```
entrou
1
FIND
 Sequence identifier: 0, Sequence value: 0011, cheq_sequencia: 1011
 Sequence identifier: 0, Sequence value: 0011, cheq_sequencia: 1011
 Sequence identifier: 1, Sequence value: 1011, cheq_sequencia: 1011
xmsim: *W,RNQUIE: Simulation is complete.
```

Figura 03- Saída da parte 2 do identificador

2. Ordenadores e Mediana

O segundo desafio pedia 2 tarefas, implemente um circuito digital que ordene 8 números de 8 bits sem sinal, e implemente um circuito digital que realiza a filtragem da mediana de 9 entradas. Para este desafio e o log na base 2 pedi ajuda a alguns colegas para identificarmos o melhor método de fazer. Por fim, escolhi por fazer um método combinacional, que fazia as combinações simultaneamente e utilizava um método novo para min que faz a quantidade de dados de entrada ser variável.

Para os ordenadores foi feito assim, pego o maior número, retiro da sequência e coloco no primeiro lugar da lista (assim como na imagem abaixo), pego novamente o maior número e vou fazendo assim com todos os valores restantes. Para valores repetidos fiz da seguinte forma, marco ele, coloco na posição mais à direita e sabendo disso, coloco os próximos valores na próxima posição marcada.

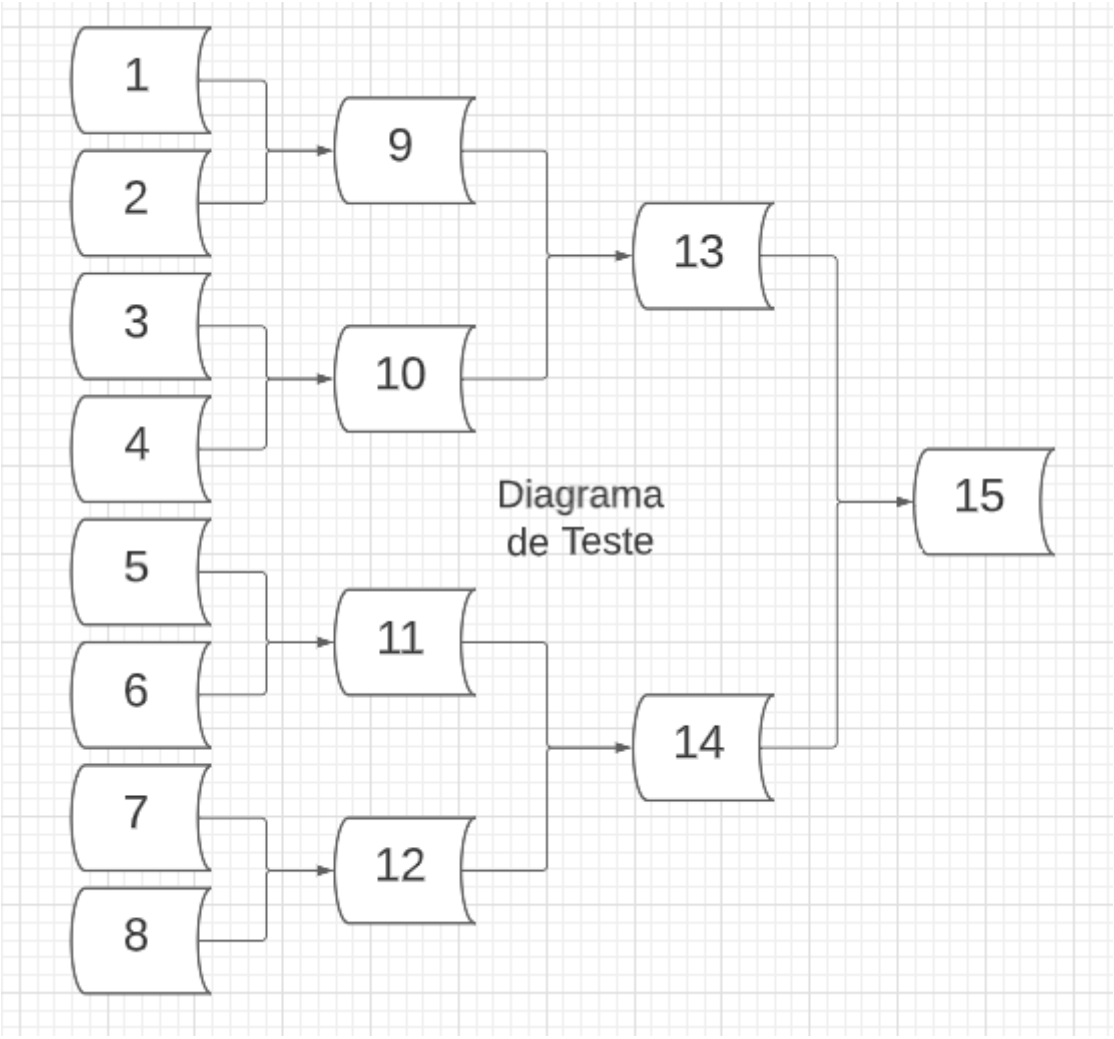


Figura 04- Diagrama de teste para determinar como seria a ideia do ordenador

Utilizando uma matriz de i colunas, e cada coluna i representa um estágio da operação, a coloquei como uma matriz de tamanho Nxlog n, já que cada estágio é dividido por 2, é só fazer operações de log de base 2 da quantidade de números da entrada.

Já para a mediana fiz da seguinte forma, chamei a operação de maior numero, e coloquei como saída a posição do meio já que o numero de entradas é ímpar, como o numero de entradas é variável, inseri a soma e divisão por 2 caso o número de entradas seja par. Fiz dos dois métodos, com números entrando serialmente e com números entrando de forma aleatória pela função pedida. Para visualização completa do código, basta entrar no link: https://www.edaplayground.com/x/WWkE .

```
At first, the numbers in serial:
OK, HIGHEST NUMBER - fe
OK, MEDIAN b4
Putting in the correct order
fe
d7
c8
be
b4
96
82
64
32
-----------------------------------
```

Figura 05- Saída para valores de entrada serial

```
Now, using the unrandom function:
OK, HIGHEST NUMBER - f2
OK, MEDIAN 83
Putting in the correct order
f2
d7
bf
bf
83
78
5c
42
25
-----------------------------------
```

Figura 06- Saída para valores de entrada aleatória

3. Logaritmo na Base 2

Outro desafio interessante, para faze-lo tive que pesquisar muito, achei algumas formas. Uma que encontrei foi utilizando a função *"$log10(k)/$log10(2)"*, já que uma forma de fazer o logaritmo na base 2 é (site nas referências bibliográficas) "log2(x) = log10(x) / log10(2)". Contudo essa função não é sintetizável, por sinal tive que estudar oque caracteriza uma função sintetizável, então essa ideia foi descartada.

O método que melhor resolveu o problema foi esse: *"A Log Base 2, também conhecida como logaritmo binário, é o logaritmo da base 2. O logaritmo binário de x é a potência à qual o número 2 deve ser elevado para obter o valor x. Por exemplo, o logaritmo binário de 1 é 0, o logaritmo binário de 2 é 1 e o logaritmo binário de 4 é 2. Ele é frequentemente usado em ciência da computação e teoria da informação"*.

Partindo dessa ideia, temos o X é igual a quantidade de vezes que podemos dividir o número B por 2 até dá 1. Contudo ainda estava difícil calcular valores que o numero não é uma potencia de 2. Mas foi aí que fui alertado que bastava utilizar os valores fracionados como $2^{1/2}$ e realmente pensa-los como raízes, e assim fazer a manipulação, multiplicando por raiz de 2 e dividindo por 2. Com essa ideia continuei com as divisões por 1/x e multiplicações por raiz de x, até 1/5 pois como a questão pede em ponto fixo e ao perguntar ao professor Gutemberg ele disse que eu poderia escolher quantos bits eu queria (ao qual escolhi 4 casas decimais) o que já nos traz uma precisão boa como pode ver na imagem abaixo.

```
OK, in: 168 -> out: 7.2500
OK, in: 178 -> out: 7.5000
OK,  in: 178 -> out: 7.5000
OK, in:  33 -> out: 5.0000
OK, in: 101 -> out: 6.5000
OK, in:  24 -> out: 4.5000
OK, in: 108 -> out: 6.5625
OK, in:  91 -> out: 6.5000
OK, in:  54 -> out: 5.5625
OK, in: 217 -> out: 7.5625
```

Figura 07- Saída com valor serial = 178 e entradas aleatórias

Como quando foi discutido com o professor, o primeiro método não é sintetizável. Contudo discutindo o desafio, me deram a ideia de usar a função do compilador para testar os meus dados, pesquisei no GIT e encontrei a função em Verilog que se encaixava na situação e apenas a transformei para ponto fixo e comparei com meus valores obtidos. Infelizmente poucos valores batem com a precisão do código não sintetizável, porém mostra que o código está até correto porem impreciso.

```
OK,  in: 178 -> out: 7.5000
ERROR, in:  33 -> out: 5.0000 - using the second method: 5.0625
ERROR, in: 101 -> out: 6.5000 - using the second method: 6.6875
ERROR, in:  24 -> out: 4.5000 - using the second method: 4.5625
ERROR, in: 108 -> out: 6.5625 - using the second method: 6.7500
OK, in:  91 -> out: 6.5000
```

Figura 08- Teste do meu Log 2 com a função do próprio compilador.

Para visualização completa do código, basta entrar no link: https://www.edaplayground.com/x/73VX .

4. Decodificador Huffman

Dessa vez o desafio foi simples, como eu já havia pego o ritmo me preocupei em fazer esse desafio o mais correto possível. E diferente do primeiro desafio, foquei em usar corretamente a minha chave *"valid"*, que nesse código é responsável por avisar ao programa que quando valid é 1, o programa pode pegar os valores da entrada serial e rodar o programa. Antes eu estava errando o momento de adicionar valores, já que para esse código devo adicionar os valores e acionar o valid antes da subida do próximo clock.

"A codificação de Huffman é um código de comprimento variável que utiliza a probabilidade de ocorrência dos símbolos no conjunto de dados para definir seu comprimento (número de bits). Ele é usualmente utilizado para compressão de dados e utiliza uma árvore binária completa no processo de codificação/decodificação", então bastava que eu coloque os valores serialmente na entrada, e eu tenha na saída o meu símbolo.

Símbolo	Palavra-Código
S1	00
S2	01
S3	10
S4	110
S5	111000
S6	111001
S7	111010
S8	1110110
S9	1110111
S10	1111000
S11	1111001
S12	1111010
S13	1111011
S14	1111100
S15	1111101
S16	1111110
S17	11111110
S18	11111111

Figura 09- tabela de Huffman disponibilizada.

Para esse desafio pensei no processo que, ao adicionar os valores serialmente de acordo com a tabela, de acordo com os valores que eu adiciono, o programa vai testando com os casos possíveis e assim determina a saída. Para essa questão tive que aprender a utilizar o *"casez"*, que nunca havia utilizado. *"Alguns bits do padrão de seleção podem ser marcados como não*

importa. A codificação de prioridade é um exemplo em que casez é um bom ajuste". Então simplesmente criei um contador que determinava o tamanho da minha entrada (no máximo 8 bits), fiz o casez determinando os valores que eu queria que fosse a saída e assim como pediu na questão cuidei para que os valores que ocorressem como erro tivessem como saída o símbolo S0.

```
Initializing huffman decoder...
 OK, valid process -- The Symbol is : S12
xmsim: *W,RNQUIE: Simulation is complete.
```

Figura 10- Saída para a entrada = 1111010.

Também me preocupei em fazer a máquina de estados da forma que o código ficasse o mais legível possível e menos acoplado, facilitando manutenções futuras e entendimento por parte de terceiros, assim como li no livro de SystemVerilog disponível na biblioteca fazendo a máquina de estados de forma separada. Para visualização completa do código, basta entrar no link: https://www.edaplayground.com/x/w8Pz .

5. Filtro de Sobel

Nesse desafio foi proposto algo diferente, já nos foi dado o Código de Nelson Campos (pesquisando sobre ele no LinkedIn, vi que ele se formou pela UFCG) e esse código era basicamente a questão proposta. Fazer um filtro de Sobel, uma convolução da imagem original com duas matrizes 3x3, e o resultado dessa matriz, é somado para me dar o resultado que é a imagem aplicada no filtro.

Entrei no link disponibilizado, entendi o código (ao qual irei explicar) e fiz as alterações necessárias para que eu pudesse testa-lo. O código é dividido em 3 partes, a primeira é a parte da manipulação dos valores de pixel. Para a solução das matrizes, Nelson apenas percebeu que poderia ser solucionado a partir do deslizamento dos pixels a nas matrizes, então ele pega o valor da entrada e desliza a cada pulso de clock para a posição desejada.

```
sliding[0][0] <= inputPixel;
sliding[1][0] <= sliding[0][0];
sliding[1][1] <= sliding[0][1];
sliding[1][2] <= sliding[0][2];
sliding[2][0] <= sliding[1][0];
sliding[2][1] <= sliding[1][1];
sliding[2][2] <= sliding[1][2];
```

Figura 11- Deslizamento do Pixel para a posição correta.

A segunda parte é simplesmente onde é feita a manipulação de convolução das matrizes, a geração do Gx e Gy, e consequentemente a criação do G final.

```
always_comb begin
  if (gx1 > gx2) gx <= gx1-gx2;
   else gx <= gx2 - gx1;
  if (gy1 > gy2) gy <= gy1-gy2;
   else gy <= gy2-gy1;
end

logic [WORD_SIZE+2:0] g;

always_comb g <= gy+gx;
```

Figura 12- Manipulação necessária para o filtro de Sobel

Já a terceira parte é o teste, ele chama as funções e aplica um valor de entrada aleatório a entrada. Contudo para que eu consiga testar o programa corretamente, e implementando alguma ideia minha ao desafio, pensei em testar o filtro com a própria imagem. Pesquisando a imagem que o próprio Nelson usa, descobri que Lena é uma foto famosa, usada para teste de vários filtros e em diversos lugares, tentei por meio de alguns conversores muito ruins de “Image to .hex” mas encontrei em um arquivo no MatLab (link do arquivo nas referências bibliográficas) a imagem de Lena em hexadecimal.

Fiz uma função que seleciona os valores da imagem de Lena e manda para a entrada da função criada por Nelson. E assim, a saída se torna os valores da imagem e elas são transformadas em sequência. Contudo a sequencia de leitura é feita na ordem da esquerda para

a direita e de cima para baixo, ordem que eu acredito estar errada. Por isso pensei em uma forma do filtro ler a imagem na ordem correta com os valores entrando da forma ideal (da forma que pensei), não implementei, pois, como a questão não especifica o cálculo para uma imagem, por isso fiquei livre para determinar a ordem inicial como a desejada.

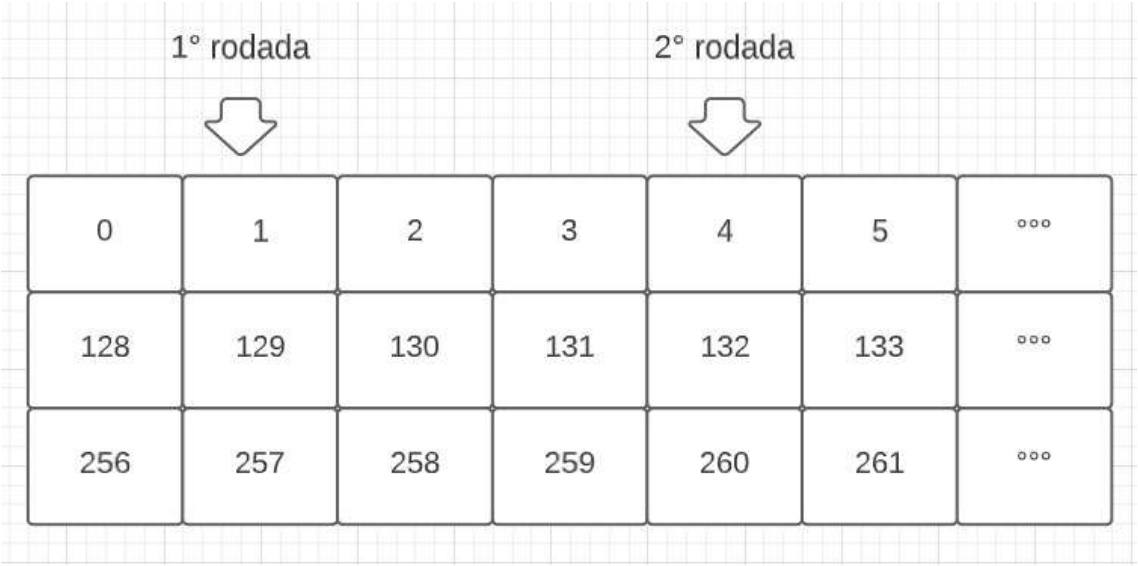


Figura 13- Como os valores estão dispostos e a sequência de leitura.

Primeiramente percebi que os números estariam organizados dessa forma até 16384, e a matriz será de 3x3 a cada rodada de leitura.

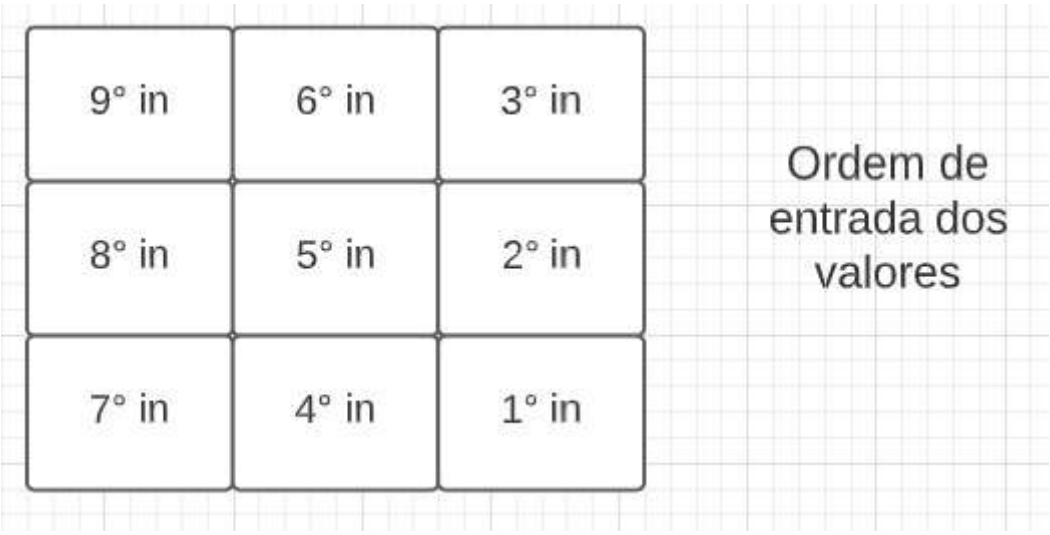


Figura 14- Ordem de entrada dos valores.

Com a organização de quais números vão entrar em cada rodada, percebi que a ordem da entrada para o deslocamento utilizado por Nelson é a da imagem acima.

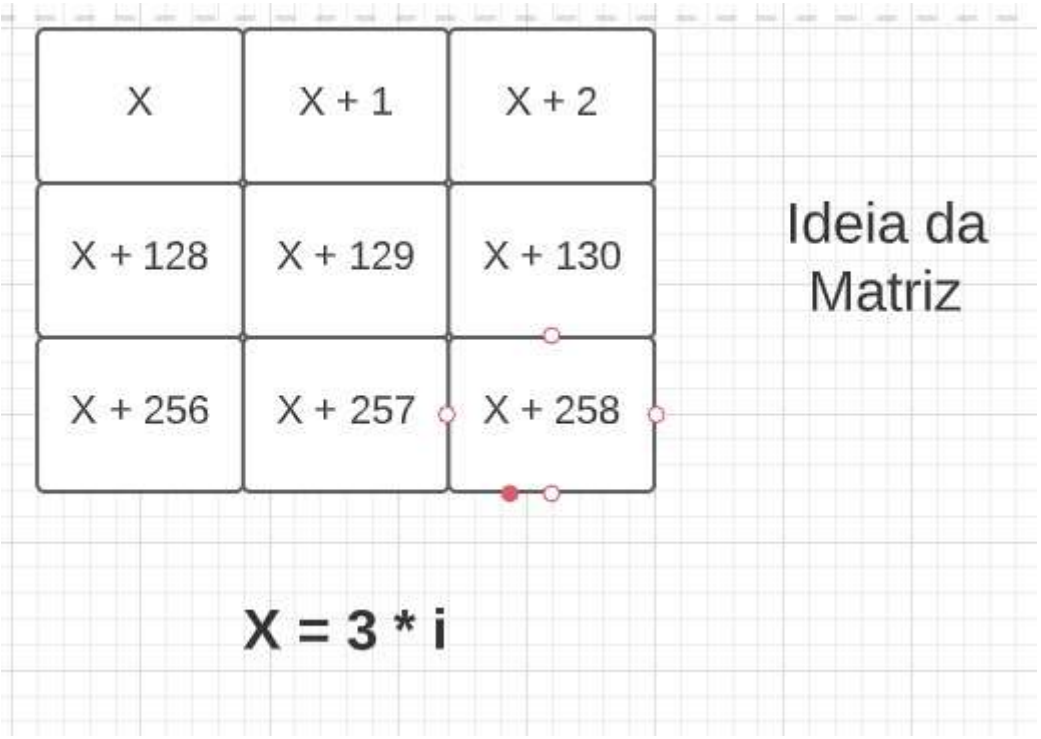


Figura 15- Ideia da matriz para a implementação e varredura dos valores

Por fim, a matriz seria utilizada dessa forma no TesteBench, onde i é a mesma variável utilizada no código, e ela serviria para a varredura de todos os valores. Por fim o filtro roda e tem como saída os valores contidos na figura abaixo. Para visualização completa do código, basta entrar no link: https://www.edaplayground.com/x/Urfw .

Figura 16- Valores da saída para a entrada (Lena.hex).

6. Exemplo de FFT

Conversando com o professor Marcos, ele disse que uma das formas também de avaliar o candidato para o desafio seria como eu poderia encontrar as soluções, então pesquisei no próprio link que nos foi enviado para o desafio do filtro de Sobel e encontrei simplesmente uma FFT pronta. Por isso para a FFT, irei fazer de forma diferente, irei comentar e deduzir o código, como ele foi implementado, e mostrando como eu entendi o processo inteiro. Já que após encontrar um código FFT pronto e determinando como usá-lo, seria difícil eu criar uma FFT a partir de outras ideias e sem inspirar-se demais no primeiro material. (Logo o link que disponibilizarei **NÃO É DE MINHA AUTORIA, É APENAS PARA DEMONSTRAR O QUE FOI PEDIDO E PARA QUE FIQUE CLARO QUE EU ENTENDI O CÓDIGO QUE ENCONTREI**).

Ao conversar com uma professora de Analises na universidade, fui estudar o método do diagrama da borboleta, um método utilizado na computação para a solução de uma FFT. O problema que para o desafio em questão necessitava de 4 entradas para calcular a FFT, e como cada valor é passado a cada ciclo do clock, deveria ser uma TFT a cada 4 ciclos de clock.

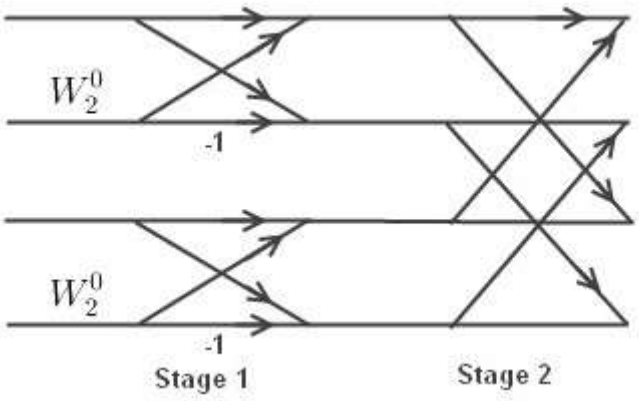


Figura 17- Método do Butterfly diagram.

O programa do Nelson foi dividido em diversas partes, percebi que no código dele temos um padrão para cada estágio, esse padrão foi replicado no módulo onde no 1 estágio é calculado o método da borboleta como na imagem acima, já a segunda metade recebe os dados do método borboleta e calcula a FFT e a iFFT (imagem da FFT e iFFT abaixo).

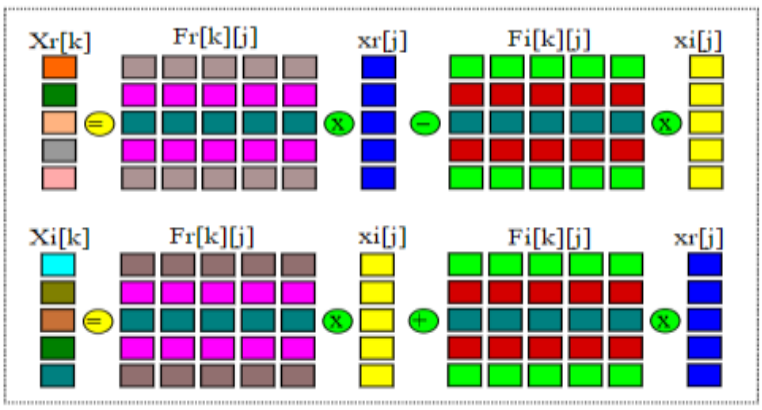


Figura 18- imagem da FFT do site do Nelson Campos

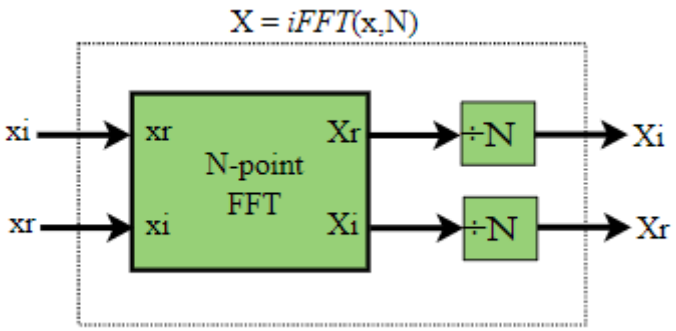


Figura 19- imagem da FFT inversa retirada do site do Nelson Campos

Após o calculo da FFT e da iFFT, foi feita uma função para determinar o tipo de calculo que o usuário quer, se ele quer o calculo da FFT ou da iFFt. Depois no testbench foi feito as chamadas das funções e aplicada nas entradas.

O único problema que encontrei para a melhor solução dessa questão conforme oque foi pedido no desafio. Seria a definição de dois tipos de pacotes, um para entrada que tinha 2 arrays de 8 bits com tamanho 16, que eu utilizaria na geração de estímulos para a entrada do recursive FFT, já a saída seria um array de números reais com tamanho 64.

Mais uma vez gostaria de ressaltar, que para esse único desafio em questão, o código **NÃO É MEU**, e não alterei nenhum valor do mesmo. Apenas fiz uma rápida descrição do código dele para provar que encontrei um código que solucionava a questão do desafio, o compreendi e fiz uma explicação para que esse desafio não passasse em branco. Para visualização completa do código citado, basta entrar no link: https://www.edaplayground.com/x/3mKR , e também, caso queira acessar o site onde o autor publicou o código criado por ele, basta entrar no link: https://sistenix.com/fft.html.

```
Error-[ZONVS] Zero or negative value for size
FFT.sv, 14
FFT, "xo"
  Zero or negative value for size is not allowed.
  Value: 0
  Please fix the size value as a positive number.
```

Figura 20- Saída do código do Nelson Campos

7. Projeto de Uma Central de Um Veículo Elétrico

No e-mail diz que eu posso também colocar informações ou projetos feitos por eu mesmo que eu acredite que possam contribuir para mostrar minhas capacidades e conhecimentos. Logo este tópico será para apresentar o projeto de “implementar uma central de controle virtual para um VEÍCULO ELÉTRICO” que realizei a um tempo atrás como projeto da disciplina de arquitetura de sistemas digitais.

Esse projeto apresenta bastantes utilidades que pude fazer com o microcontrolador Atmega328p, foi programado e configurado em assembly. Utilizei a IDE Microchip Studio, e o projeto continha diversas atividades. Criei, um painel de instrumentos com um VELOCIMETRO, otimizando um display de 7 segmentos. Um TACÔMETRO, LCD, UMA BATERIA, POWER TRAIN, utilização do EEPROM, UM SONAR para funcionar como sensor de ré, FARÓIS INTELIGENTES e por fim um VCO (Oscilador Controlador de Tensão) e um SENSOR DE IMPACTO E AIRBAG (por não terem mais portas disponíveis no chip, as implementei separadamente. Onde esses 2 últimos tem grande relação com microeletrônica analógica, então deixarei para explicar mais detalhadamente na sessão de microeletrônica analógica.

A parte de montagem foi feita no SIMULIDE, onde mostrarei imagens do sistema funcionando, e como foi feito a muito tempo, ainda não implementava os programas a partir de Camel Case ou Pascal Case, nem os comentários ou valores estão em inglês.

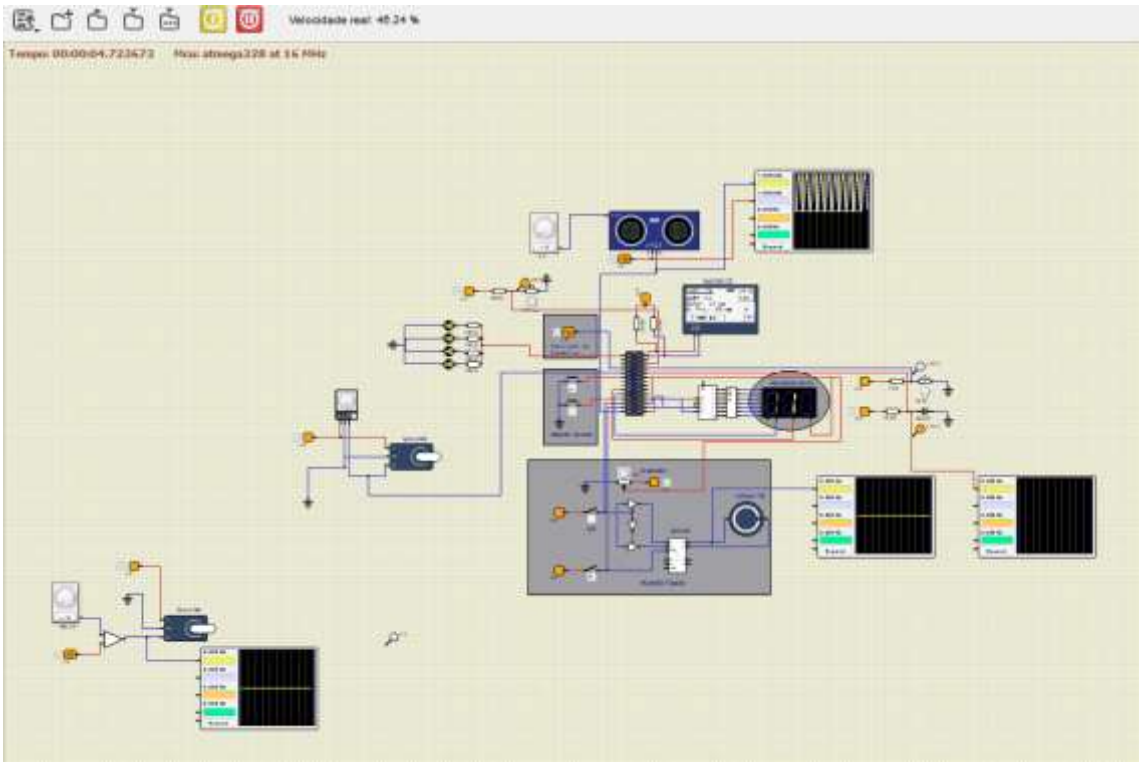


Figura 21- Visão do programa parte 1 em visão completa.

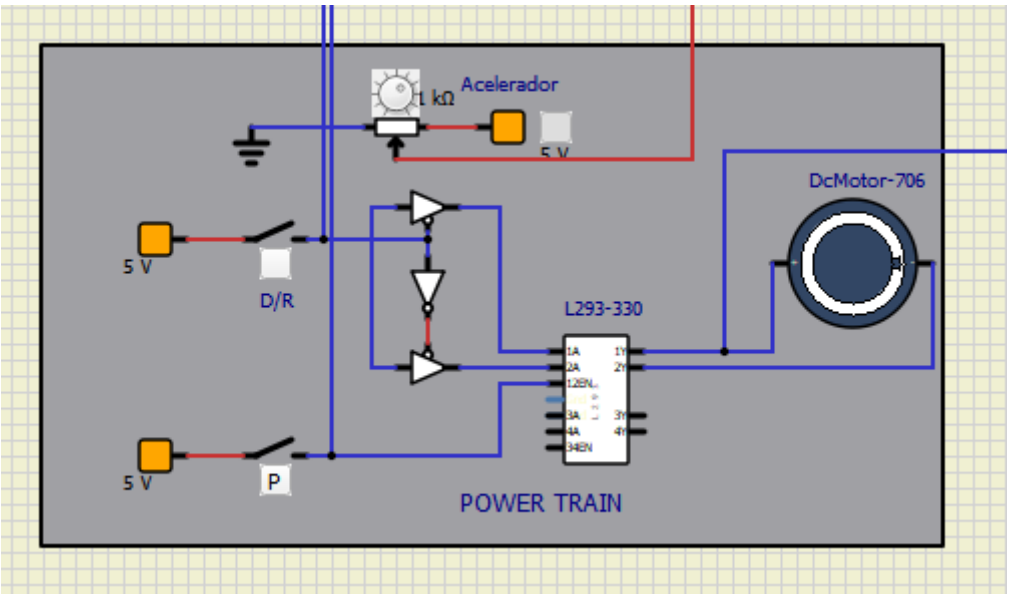


Figura 22- Power Train e MotorDc.

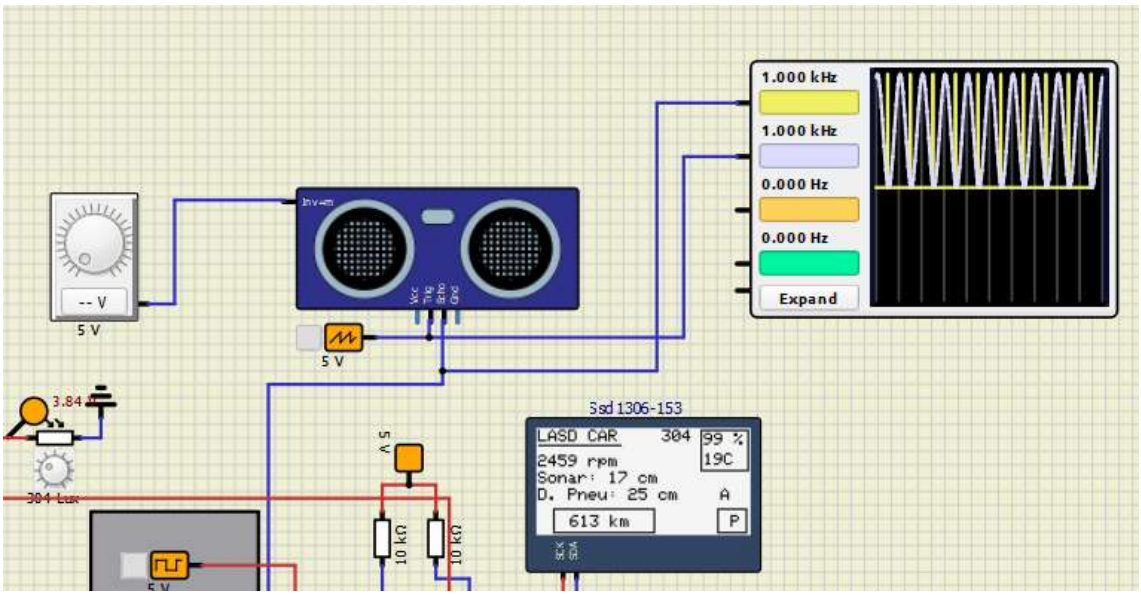


Figura 23- Display com informações e o Sonar em operação.

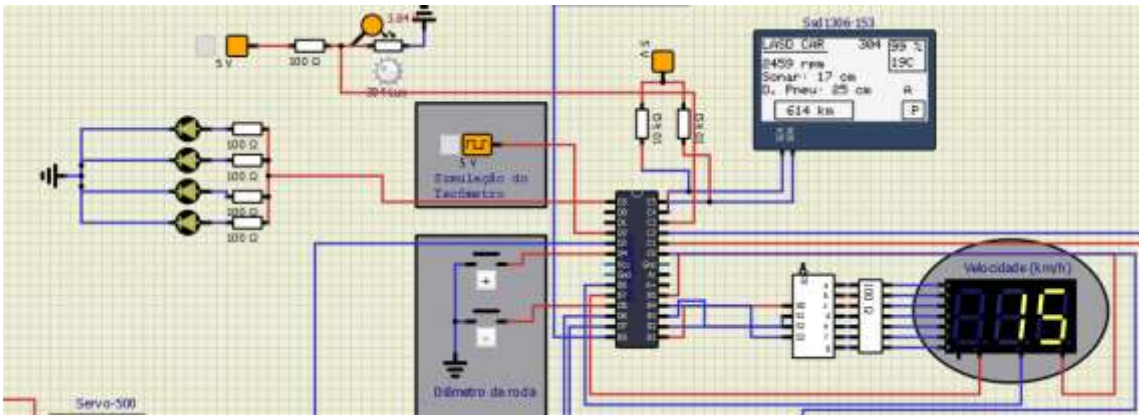


Figura 24- Tacômetro, Diâmetro da roda, faróis inteligentes e display 7 segmentos.

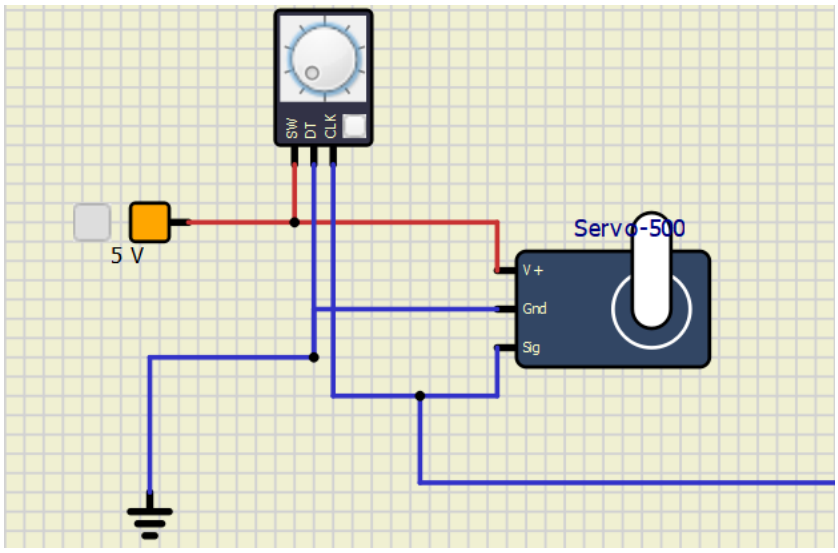
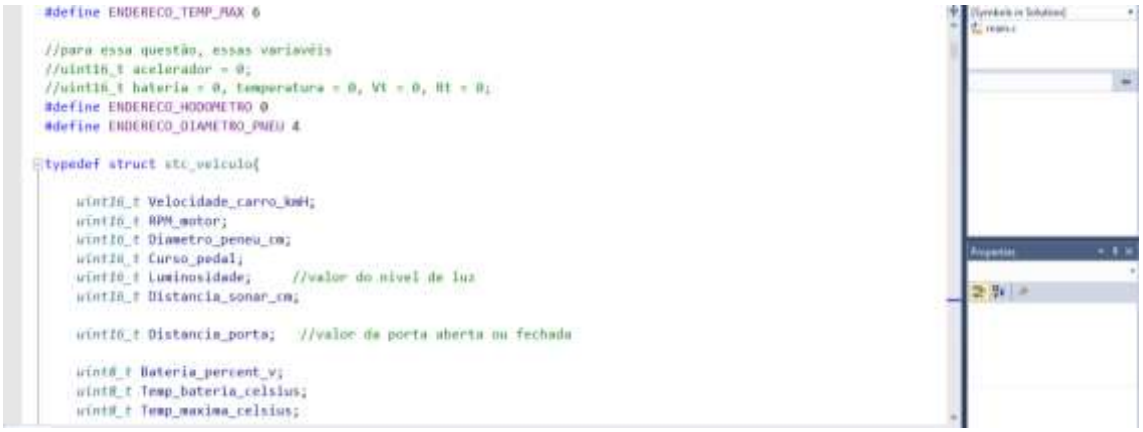


Figura 25- Sensor da porta.

Para a visualização de todo o projeto desta parte basta entrar no meu Github pelo link: https://github.com/superarmax?tab=repositories . O nome da pasta é Repositorio-de-LASD-10.



```
#define ENDERECO_TEMP_MAX 6

//para essa questão, essas variavéis
//uint16_t acelerador = 0;
//uint16_t bateria = 0, temperatura = 0, Vt = 0, Ht = 0;
#define ENDERECO_HODOMETRO 0
#define ENDERECO_DIAMETRO_PNEU 4

typedef struct stc_veiculo{

    uint16_t Velocidade_carro_kmH;
    uint16_t RPM_motor;
    uint16_t Diametro_peneu_cm;
    uint16_t Curso_pedal;
    uint16_t Luminosidade;     //valor do nivel de luz
    uint16_t Distancia_sonar_cm;

    uint16_t Distancia_porta;   //valor da porta aberta ou fechada

    uint8_t Bateria_percent_v;
    uint8_t Temp_bateria_celsius;
    uint8_t Temp_maxima_celsius;
```

Figura 26- Exemplo do código no IDE.

```
void anima_velocidade(uint16_t velocidade_carro, uint8_t *flag_disparo);
void anima_LCD(stc_veiculo veiculo, uint8_t *flag_disparo);
void load_EEPROM(stc_veiculo *veiculo);
void USART_Init(unsigned int ubrr);
void USART_Transmit(unsigned char data);
unsigned char USART_Receive(void);

void leitura_sensores_ADC(stc_veiculo *veiculo, uint8_t *flag_disparo);

ISR(USART_RX_vect)
{
    char recebido;
```

Figura 27- Exemplo 2 do código no IDE.

8. Oscilador Controlador por Tensão (VCO)

Eu deixei essa parte separada pois, essa parte se encaixa muito bem com microeletrônica analógica, essa questão foi para um ponto extra que fiz separadamente pois o projeto inicial não possuía mais portas de entrada no Atmega328p. O circuito do Oscilador Controlado por Tensão foi projetado utilizando amplificadores operacionais, resistores, mosfet e capacitor. Este circuito é composto por três partes: Amplificador Integrador, Schmitt-Trigger e Comparador de Tensão.

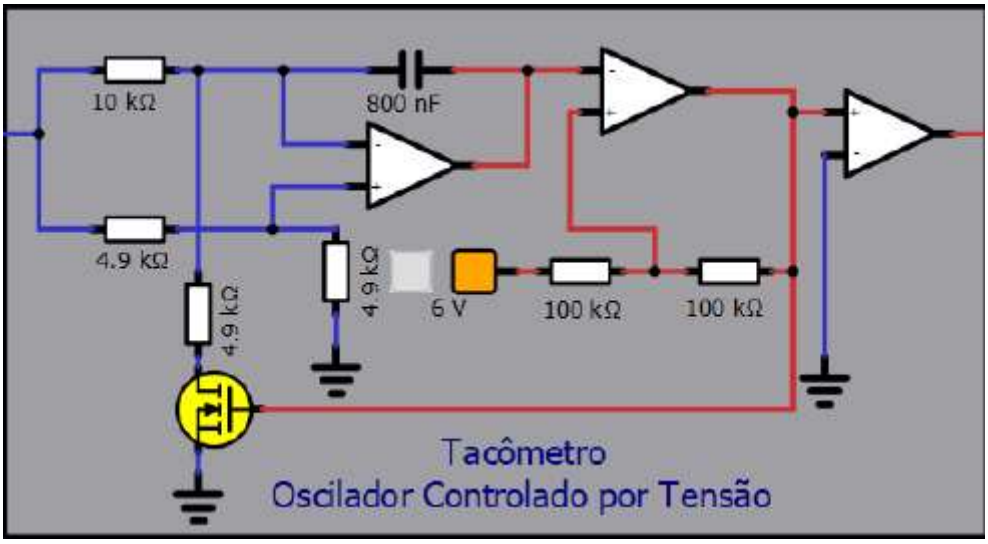


Figura 28-Tacômetro Oscilador Controlado por Tensão.

Os dois resistores do Schmitt-Trigger possuem o mesmo valor de resistência, equivalente a 100k Ohm. Os demais resistores possuem resistências iguais e um pouco menor que o dobro de R1, correspondente a 4,9k Ohm. A tensão aplicada no motor ´e conectada na entrada Vin do VCO. Dessa forma, quando variamos o duty cicle desse sinal, na saída teremos uma variação da frequência, cujo o valor máximo será 40Hz. A tensão de referência V_{ref} é a tensão de alimentação positiva do Schmitt-Trigger, equivalente a 12V. Sabendo disto, foi conectado um Comparador de Tensão com o intuito de diminuir a tens˜ao do Schmitt-Trigger para 5V. A saída do Comparador de Tensão é conectada ao pino PD2 do Atmega. Portanto, quando variamos a resistência do potenciômetro (acelerador), variamos também a velocidade do carro.

9. Sensor de Impacto e Airbag

Na implementação do airbag utilizei um sensor de força com o intuito de simular o impacto sofrido pelo carro durante uma colisão. O sensor utilizado tem uma variação de resistência muito pequena, então para ter o funcionamento adequado, foi necessário a utilização de um amplificador de instrumentação.

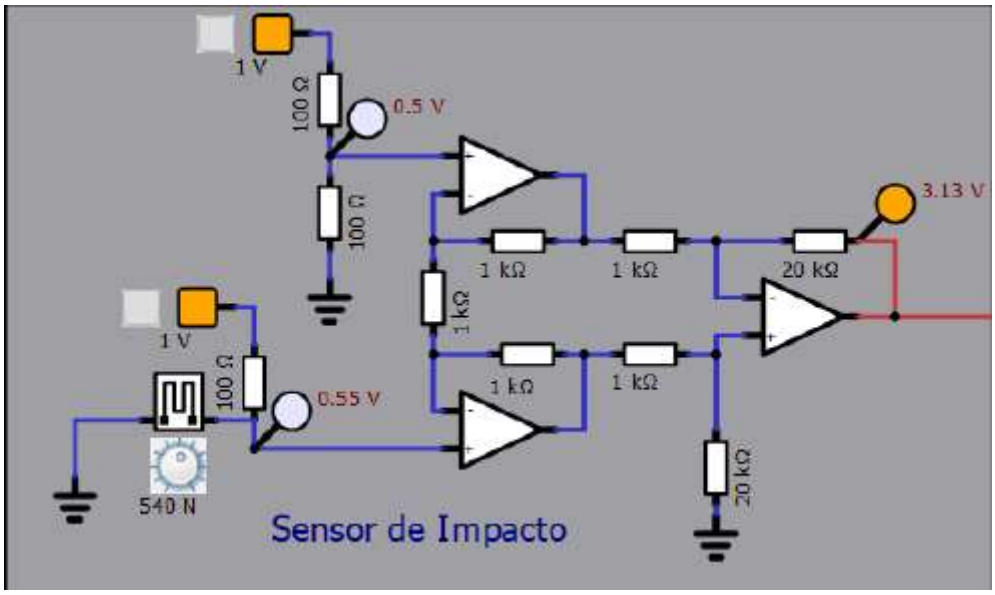


Figura 29- Circuito do Sensor de Impacto.

Ai basicamente o circuito funciona juntamente com o sonar, se a distância aferida fosse menor que 3 metros, a velocidade do carro reduzia muito rápido, ocorrendo uma frenagem brusca. Se houvesse uma frenagem brusca e o sensor de impacto detectasse a força maior ou igual a 30K N, o pino ia para nível lógico alto e alimentava o capacitor que representava o acionamento do airbag.

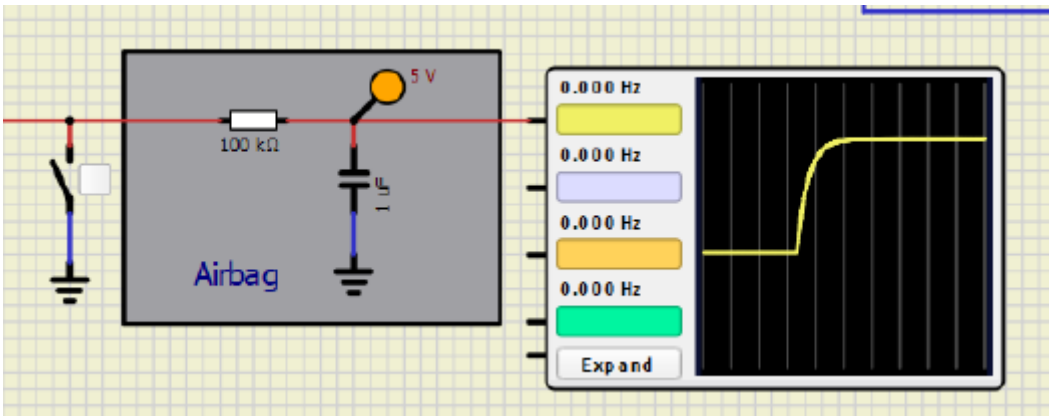


Figura 30- Circuito simulando o Airbag.

10. Filtro Passa-Baixa

Esse é o ultimo desafio que eu fiz, ainda conseguiria fazer mais desafios, contudo por questão de tempo, não pude fazer os outros desafios da parte analógica (contudo olhando por cima acredito conseguir fazer a simulação do circuito de carga e descarga de um circuito RC e a simulação do dispositivo MOSFET).

Para fazer um filtro passa-baixa, um circuito que permite a passagem de baixas frequências sem dificuldades e reduz ou atenua a amplitude das frequências maiores que a frequência de corte. Para essa questão a frequência de corte era 1kHz com um ganho de 40db. Como a questão não especifica se o filtro é de primeira ou segunda ordem, fiquei livre para escolher a que eu achei melhor, logo, primeira ordem.

Para isso, determinei o R1= 10 k Ohms, o R2 = 99k Ohms e o R3 = 1k Ohms respectivamente, e a partir disso determinei o ganho para 100. Convertendo esse 100 para dB é justamente 40 dB.

$$A_v = \frac{99k}{1K} + 1 = 100\,\frac{V}{V}$$

$$A_{vdB} = 20Log100 = 40\ dB$$

Eu não sabia o meu valor de C, mas sabendo a frequência de corte e o valor de R1, eu isolo o capacitor, e substituindo os valores eu descubro o valor da capacitância.

$$C = \frac{1}{2\pi R1 fe}$$

$$C = \frac{1}{2\pi * 10 * 1k * 1k} = 15{,}92\ nF$$

Para o ganho na banda de rejeição, eu não soube nem oque é nem como fazer infelizmente, então para compensar vou fiz a função de transferência desse circuito. Para observação desse circuito basta entrar no link: https://everycircuit.com/circuit/5272692590379008/filtro-passa-baixa-ativo.

$$H(s) = \frac{1+\frac{99}{1}}{s(10*1k*15{,}92n)+1} = \frac{100}{159{,}2*10^{-6}s+1}$$

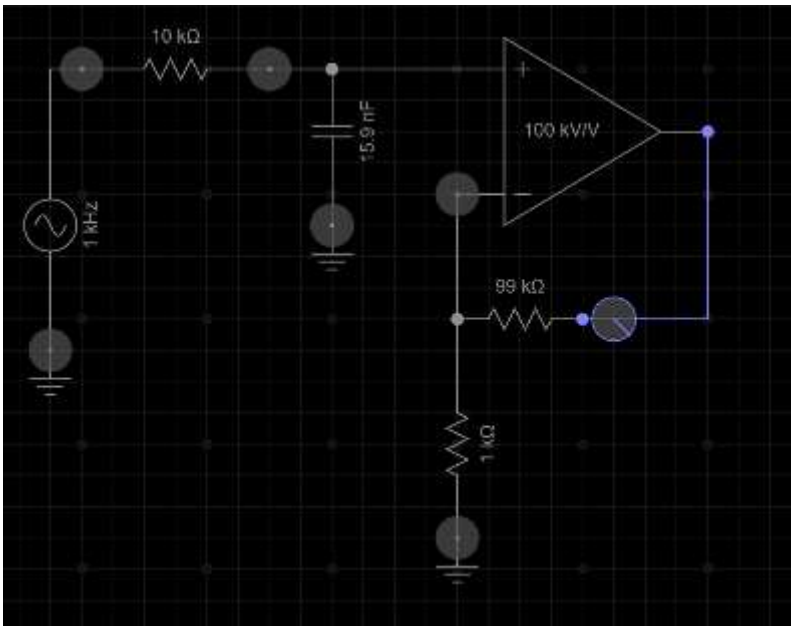


Figura 31- Filtro Passa-baixa montado.

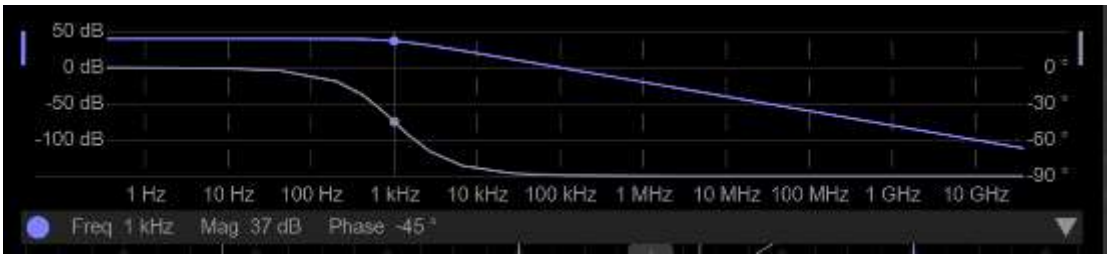


Figura 32- Resposta em frequência.

Referências Bibliográficas

WIKIPÉDIA. Sobel operator. 2013. <https://en.wikipedia.org/wiki/Sobel_operator>. Acessado em 05 de outubro de 2022.

CAMPOS, N. Fixed Point Recursive FFT. Publicado em março de 2018. < https://sistenix.com/fft.html >. Acessado em 02 de outubro de 2022.

CAMPOS, N. 2D Convolution in Hardware. Publicado em dezembro de 2016. < https://sistenix.com/sobel.html >. Acessado em 06 de outubro de 2022.

WIKIPÉDIA. Butterfly diagram. Atualizado em setembro de 2022. < https://en.wikipedia.org/wiki/Butterfly_diagram>. Acessado em 02 de outubro de 2022.

AlwaysLearn.com Website. A DFT and FFT TUTORIAL. Publicado em fevereiro de 2019. < http://www.alwayslearn.com/DFT%20and%20FFT%20Tutorial/DFTandFFT_FFT_Butterfly_4_Input.html>. Acessado em 03 de outubro de 2022.

PALINI, M. Image to hex, hex To image conversion. Publicado em janeiro de 2012. < https://uk.mathworks.com/matlabcentral/fileexchange/34713-imagetohex-hextoimage-cconversion>. Acessado em 08 de outubro de 2022.

MEEUS, W. Case statements in VHDL and (System)Verilog. Publicado em dezembro de 2020. < https://insights.sigasi.com/tech/case-statements-vhdl-verilog/>. Acessado em 29 de setembro de 2022.

CILETTI, M. Advanced Digital Design with the VERILOG HDL. Phi learning Private Limited, v.1, 2014. (Biblioteca UFCG)

ChipVerify. System Verilog Strings. Publicado em 2015. < https://www.chipverify.com/systemverilog/systemverilog-strings>. Acessado em 01 de outubro de 2022.