**Introdução da linguagem VHDL**

O VHDL (*Very High Speed Integrated Circuit Hardware Description Language*) é uma das linguagens HDL, talvez a mais importante, utilizadas para descrição de circuitos digitais e sua síntese. Portanto, não é uma linguagem de programação!

A linguagem VHDL foi desenvolvida com a necessidade do Departamento de Defesa dos Estados Unidos (DARPA), para documentar e projetar os circuitos ASIC *(Application-Specific Integrated Circuit)*. Esta linguagem podia ser utilizada também simulação de circuitos digitais. Primeiramente foi concedido à algumas empresas, entre elas: IBM, Texas e Intermetrics; para o desenvolvimento da linguagem e algumas ferramentas para o desenvolvimento de circuitos digitais.

Dessa maneira, foi concedido para o IEEE criar padrões para o VHDL, o primeiro foi o IEEE 1076–1987. Atualmente, o último padrão lançado (quando escrito este texto) foi o IEEE 1076–2019.

Para a descrição de circuitos digitais para FPGA existem outras linguagens, tais como: Verilog, SystemVerilog, SystemC entre outras. Para o projeto de circuitos digitais integrados, a Cadence, por exemplo, utiliza o Verilog em suas ferramentas.

A linguagem VHDL permite instruções concorrentes, ou seja, que ocorrem em paralelo, como ocorre com o circuito descrito abaixo, o qual temos uma porta AND em paralelo com uma porta OR.

**Verilog e SystemVerilog**

Verilog é uma linguagem de descrição de hardware inventada entre os anos de 1983 a 1984. Os inventores são [Prabhu Goel](https://en.wikipedia.org/wiki/Prabhu_Goel), [Phil Moorby](https://en.wikipedia.org/wiki/Phil_Moorby) and Chi-Lai Huang. É uma linguagem que se assemelha à linguagem de programação C. O Verilog também é muito utilizado para o desenvolvimento de ASICs e circuitos digitais em FPGA. O Verilog é principalmente utilizado nas ferramentas de desenvolvimento da Cadence, para ASIC (*Application-specific integrated circuit*). A sua variante, o VerilogA, é utilizado para modelar circuitos analógicos por meio de funções Laplacianas, sendo possível também simular circuitos digitais.

Por outro lado, o SystemVerilog é uma linguagem de descrição de um nível de abstração mais alto em relação ao VHDL e Verilog. Criada por volta dos anos 2000. Principalmente utilizada para a verificação e *testbench* de outras descrições de circuitos em outras linguagens. Já que as linguagens em HDL permitem que sejam utilizados módulos de diferentes linguagens, que podem ser implementadas em um mesmo sistema. O SystemVerilog vem ganhando espaço no mercado cada vez mais.

No entanto, o VHDL ainda é muito utilizado em sistemas muito críticos como o Aero Espacial, devido a sua maturidade e confiabilidade. É uma linguagem altamente “tipada”, e durante a simulação é possível ver com mais detalhes os valores dos sinais. No VHDL é possível em simulação apresentar até 9 tipos diferentes de valores como mostrado na tabela abaixo.

|  |  |
| --- | --- |
| Value | Description |
| ‘U’ | Uninitialised signal which hasn’t been assigned a value yet. |
| ‘X’ | Unknown value as it’s impossible to determine the value. |
| ‘0’ | Logic level 0 |
| ‘1’ | Logic level 1 |
| ‘Z’ | High impedance |
| ‘W’ | Weak signal, it’s not possible to determine the logic level |
| ‘L’ | Signal has a weak pull down meaning it should go to 0 |
| ‘H’ | Weakly pulled down signal that should probably go to 1 |
| ‘-‘ | Don’t care. |

Source: https://fpgatutorial.com/vhdl-types-and-conversions/

**Exemplos de código em VHDL**

A seguir temos alguns exemplos de código em VHDL. O primeiro código representa duas portas lógicas: AND e OR; em paralelo.

**library** IEEE;

**use** IEEE.STD\_LOGIC\_1164.**ALL**;

-- Entity declaration

**entity** Example\_1 **is**

**Port** (

inputA : in STD\_LOGIC;

inputB : in STD\_LOGIC;

outputA : out STD\_LOGIC;

outputB : out STD\_LOGIC;

);

**end** Example\_1;

-- Architecture definition

**architecture** Behavioral **of** Example\_1 **is**

**begin**

outputA <= inputA AND inputB;

outputB <= inputA OR inputB;

**end** Behavioral;

No início do arquivo temos as bibliotecas declaradas para esta descrição. Em seguida é escrita a palavra reservada entidade (“Entity”) e seu nome “Example\_1” (que deve ser o mesmo nome do arquivo deste circuito “Example\_1.vhd”) e nesta entidade temos os *ports* que são as entradas e saídas deste módulo, utilizando a palavra reservada *in* ou *out*. Uma observação sobre a linguagem VHDL é que ela não é *case sensitive*. Portanto a nomenclatura “input\_1” e “INPUT\_1” são considerados os mesmos sinais.

O tipo STD\_LOGIC é utilizada para declarar sinais de somente 1 bit, já o STD\_LOGIC\_VECTOR é utilizada, geralmente para declarar sinais de mais de um bit usando o designador (X **downto** Y) ou (Y **to** X). O ***downto*** é para caracterizar um sinal *little endian* e o ***to*** para *big endian*. O programador também tem a possibilidade também de criar os seus próprios tipos de sinais.

Conforme o código acima declarado, pode-se afirmar que “outputA” e “outputB” ocorrem paralelamente. O circuito equivalente desta descrição, depois de sintetizada será o seguinte:

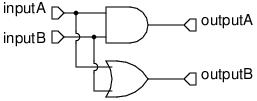


Figura 1: Example\_1.vhd

No entanto, os circuitos reais que estão na FPGA, de maneira bem simplificada, são os seguintes: seguintes:

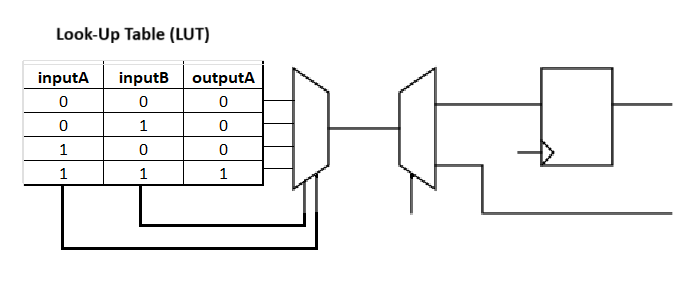


Figura 2: AND gate

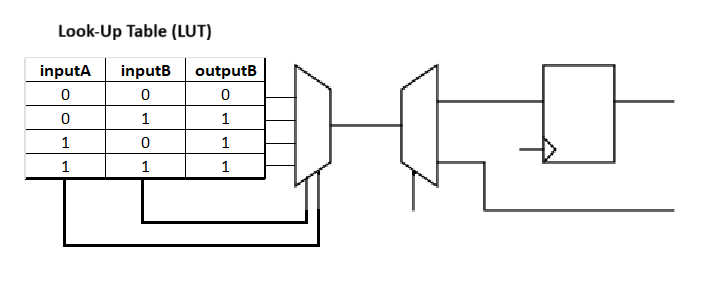


Figura 3: OR gate

Sendo que a *Look-Up table (LUT)* são memórias que foram programadas pelo sistema, que neste caso, utilizamos o *Quartus*. Dependendo da FPGA estas memórias podem ser *SRAM*, *Flash* ou *Anti-Fuse.*

Na maioria dos casos da FPGA Intel se utiliza memórias SRAM. Sendo que esta memória é volátil, portanto, é necessário, caso o projetista queira gravar permanentemente o circuito que se deseja na FPGA; é preciso que se grave numa memória FLASH externa à FPGA para que quando se ligue o sistema, a memória FLASH transfira o conteúdo para as LUTs da FPGA. Este processo pode levar alguns milissegundos para ser completado.

**Circuitos Sequenciais**

Um exemplo de circuitos sequencial, ou seja, um circuito com memória, é por exemplo um registador de 8 bits, a seguir o código:

**library** IEEE;

**use** IEEE.STD\_LOGIC\_1164.**ALL**;

**use** IEEE.NUMERIC\_STD.**ALL**;

**entity** register\_top\_level **is**

**Port** (

        clk     : **in**  STD\_LOGIC;

        reset   : **in**  STD\_LOGIC;

enable : **in** STD\_LOGIC;

        input   : **in**  STD\_LOGIC\_VECTOR(0 **to** 7); *-- 8 bits LITTLE ENDIAN*

        output   : **out** STD\_LOGIC\_VECTOR(0 **to** 7)  *-- 8 bits LITTLE ENDIAN*

    );

**end** register\_top\_level;

**architecture** Behavioral **of** register\_top\_level **is**

**signal** s\_output: std\_logic\_vector(0 **to** 7);

**begin**

processor\_register:  **process**(clk, reset)

**begin**

**if** reset = '1' **then**

            s\_output <= (**others** => '0');

**elsif** rising\_edge(clk) **then** *-- or event'clk and clk='1' then*

**if** enable = ‘1’ **then**

            s\_output<= input;

**end if**;

**end** **if**;

**end** **process**;

output <= s\_output;

**end** Behavioral;

**Exemplo de Média Móvel em VHDL**

Este exemplo de código em VHDL, faz a aquisição de dados pelo porto inputA, que é acumulado pela memória SRAM (“array”) por meio da variável *buffer* Esta contém 4 bytes na memória SRAM. E com a variável de soma (*sum*) é a realizada a adição destes últimos valores de adquirido buffers, e dessa maneira é calculado a média, divisão por quatro, através do deslocamento de bits da variável *sum,* deslocando 2 bits para a direita (22)*,* ou seja, dividindo por quatro.

**library** IEEE;

**use** IEEE.STD\_LOGIC\_1164.**ALL**;

**use** IEEE.NUMERIC\_STD.**ALL**;

**entity** moving\_avarage **is**

**Port** (

        clk     : **in**  STD\_LOGIC;

        rst     : **in**  STD\_LOGIC;

        inputA  : **in**  STD\_LOGIC\_VECTOR(7 **downto** 0); *-- 8 bits*

        avarage : **out** STD\_LOGIC\_VECTOR(7 **downto** 0)  *-- 8 bits*

    );

**end** **entity** moving\_avarage;

**architecture** Behavioral **of** moving\_avarage **is**

**type** bufferA **is** **array**(0 **to** 3) **of** unsigned(7 **downto** 0); *--memory sram  for 4 samples*

**signal** m\_bufferA :bufferA;

**signal** sum   : unsigned(9 **downto** 0); *--  4 \* 255 = 1020*

**begin**

**process**(clk, rst)

**begin**

**if** rst = '1' **then**

            m\_bufferA(0) <= (**others** => '0');

            m\_bufferA(1) <= (**others** => '0');

            m\_bufferA(2) <= (**others** => '0');

            m\_bufferA(3) <= (**others** => '0');

            sum       <= (**others** => '0');

            avarage   <= (**others** => '0');

**elsif** rising\_edge(clk) **then**

*-- shift the buffer*

            m\_bufferA(3) <= m\_bufferA(2);

            m\_bufferA(2) <= m\_bufferA(1);

            m\_bufferA(1) <= m\_bufferA(0);

            m\_bufferA(0) <= unsigned(inputA);

*-- sum the values in the buffer*

            sum <= ("00" & m\_bufferA(0)) + ("00" & m\_bufferA(1)) + ("00" & m\_bufferA(2)) + ("00" & m\_bufferA(3));

*-- avarage calculation*

            avarage <= std\_logic\_vector(sum(9 **downto** 2));   *-- sum >> 2, divide by 4*

**end** **if**;

**end** **process**;

**end** **architecture** Behavioral;

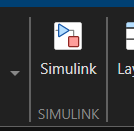
**Exemplo de filtro FIR no Matlab**

**Passo a Passo**

Antes de começar, instale o Matlab **R2025b**.

1. Abra o Matlab.

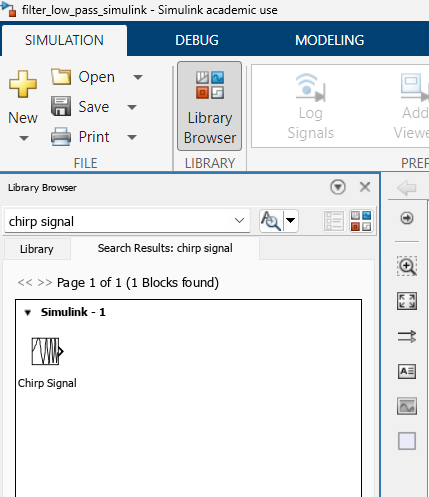
2 – Abra o Simulink.



4- Depois de aberto o ambiente (Simulink). Clique em *Library Browser.*



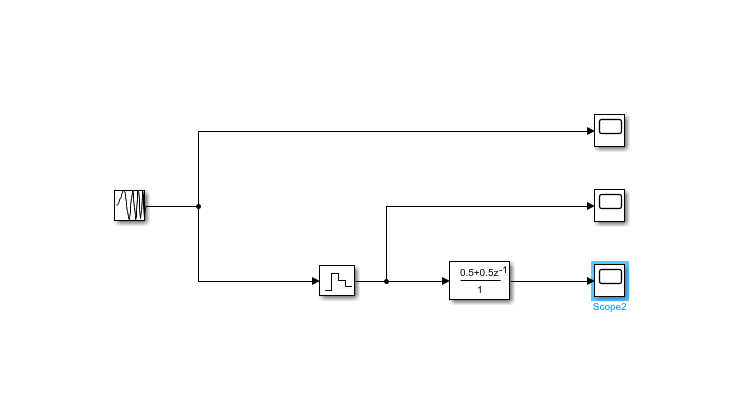
5 – Procure por *Chirp Signal*



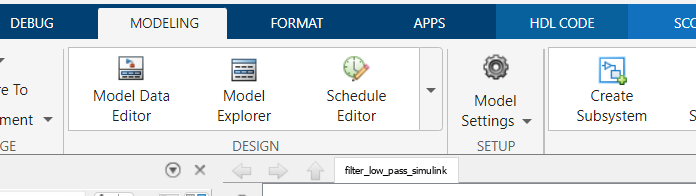
6 – Arreste para área de trabalho do *Simulink.*

7- Procure por ***Zero Order Hold*, *Discrete FIR Filter***e coloque três ***Scope***.

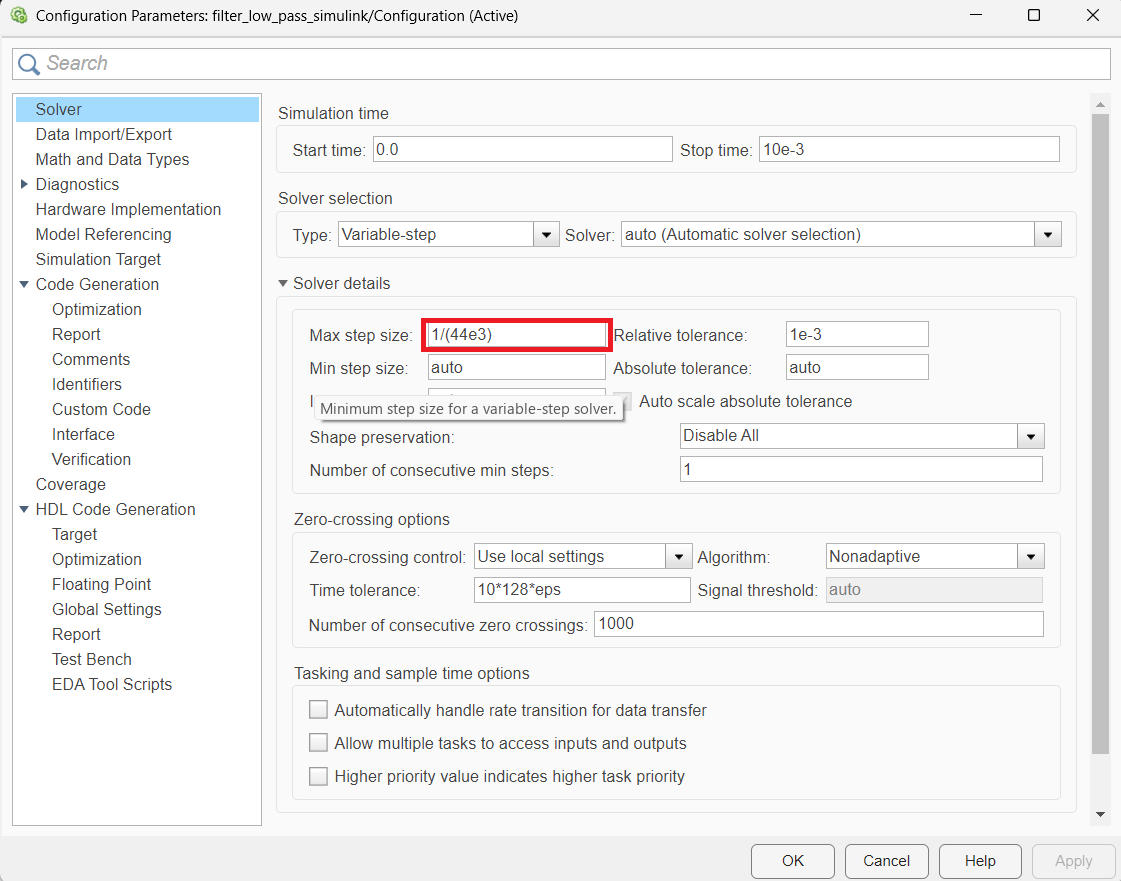
8 – Interligue os componentes conforme figura abaixo



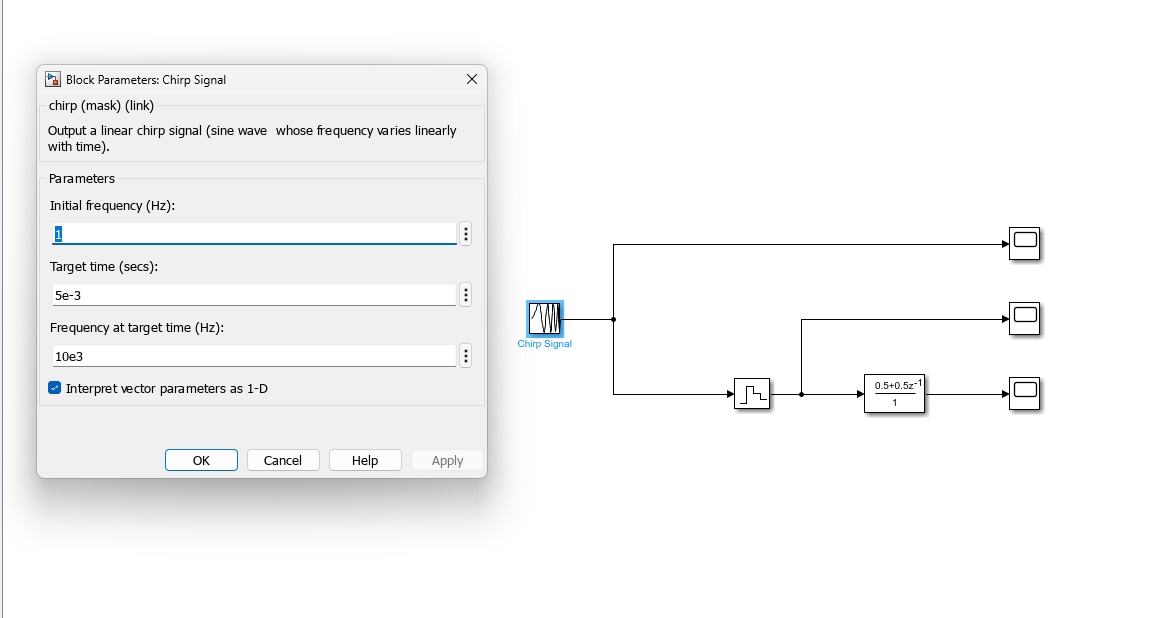
9 – Clique na aba *Modeling,* depois em *Model Settings.*



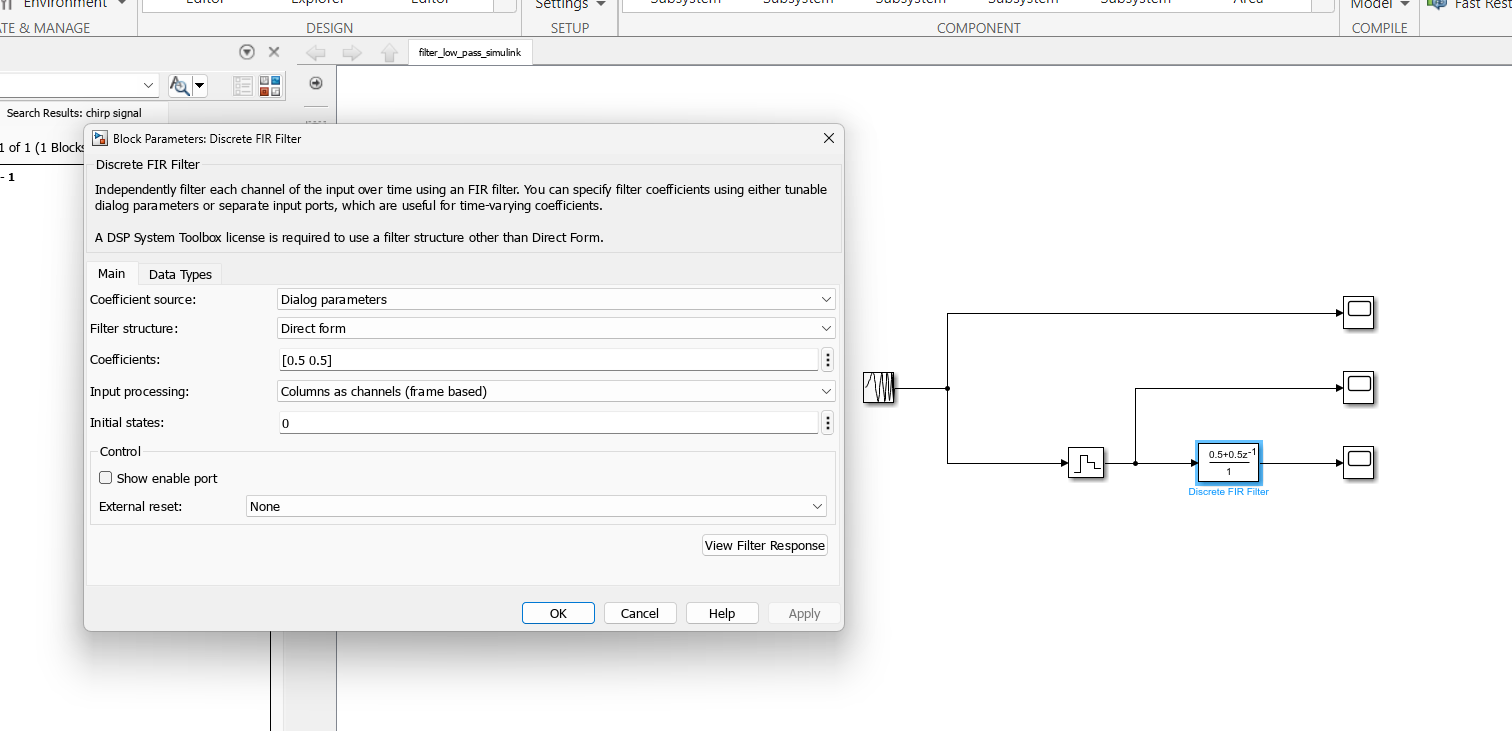
10 – Mude o campo *Max step size*., conforme figura abaixo



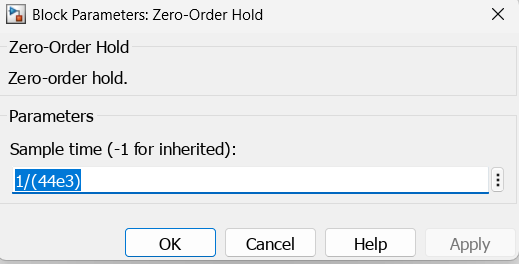
11 – Clique duas vezes no ícone *Chip Signal*. E altere os campos conforme a figura, clique em OK.



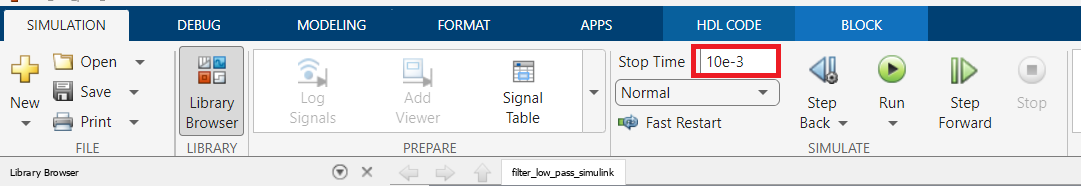
12 – Clique duas vezes em *Discrete Filter FIR.* E altere os campos e clique em OK.

****

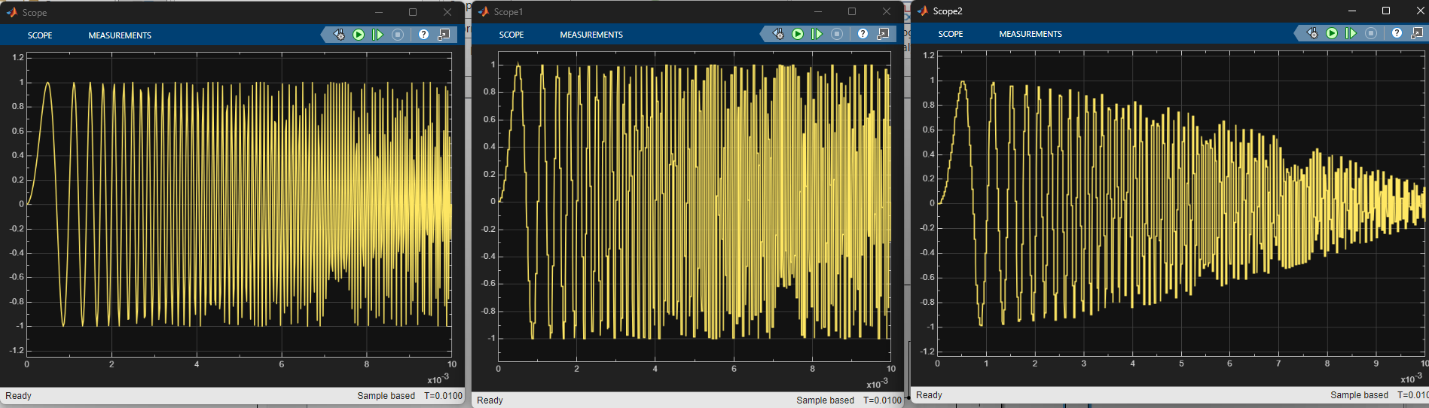
13- Clique em *Zero Order Hold*. E altere o valor do campo *Sample Time.*



15- Clique na aba *SIMULATION*. Mude o *Stop Time* para 10e-3. E clique em *Run.*

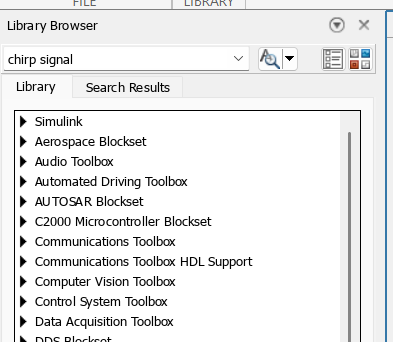


16 – Depois de simulado clique duas vezes nos *Scopes.* Você verá as seguintes curvas.

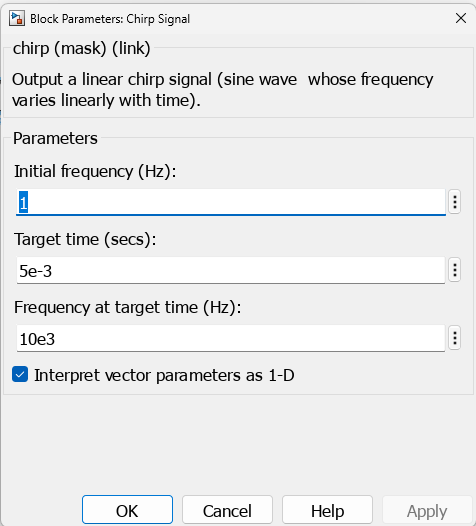


**Passo – a - Passo Projeto Filtro IIR**

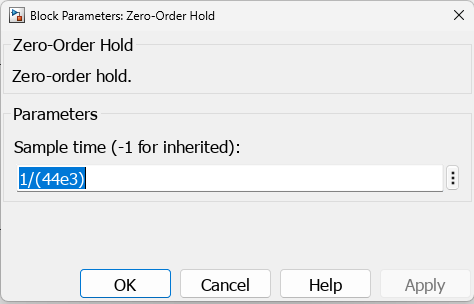
1. Primeiro execute o *Matlab R2025b*.
2. Clique na aba *Home*.
3. Abra o *Simulink*.
4. Crie no novo projeto.
5. Clique na aba *SIMULATION*.
6. Clique no botão *Library Browser*.
7. Na Library Browser, procure por *Chirp Signal*, e adicione no projeto.



1. Clique duas vezes no módulo *Chirp Signal* e configure conforme a figura a seguir.



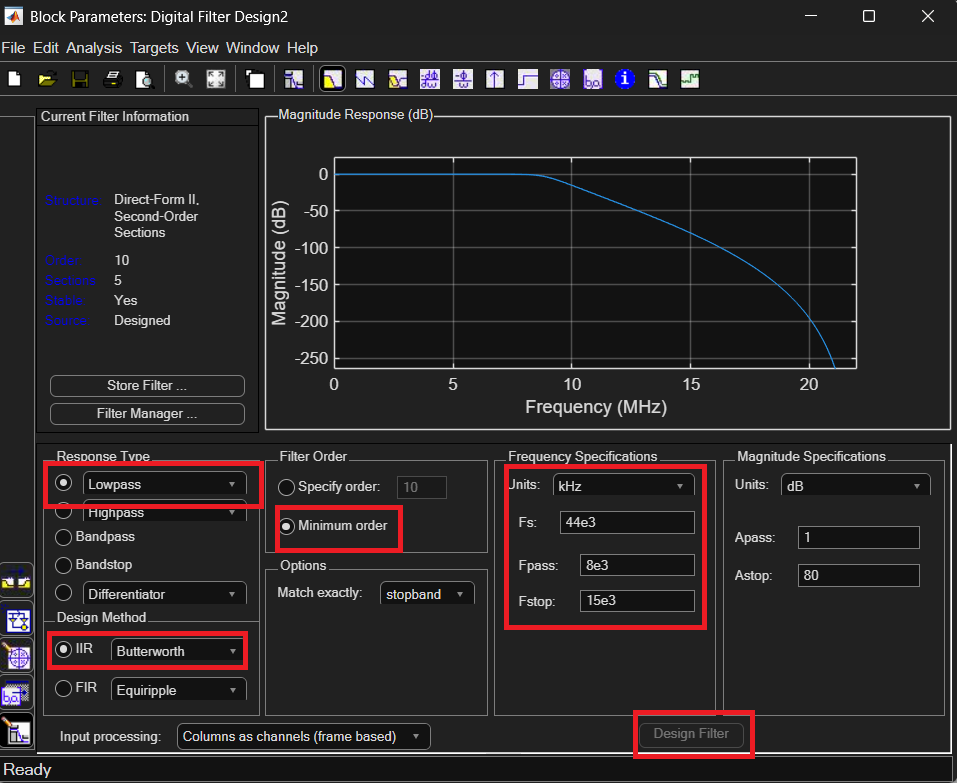
1. Agora procure por *Zero-Order Hold* no Library Browser. Adicione no projeto. E clique duas vezes sobre o ícone do módulo. Configure, conforme a figura a seguir.



1. Procure por ***Filter Designer*** no ***Library Browser***, adicione no projeto. E, coloque também três ***Scopes***.
2. Conecte os módulos conforme figura a seguir.



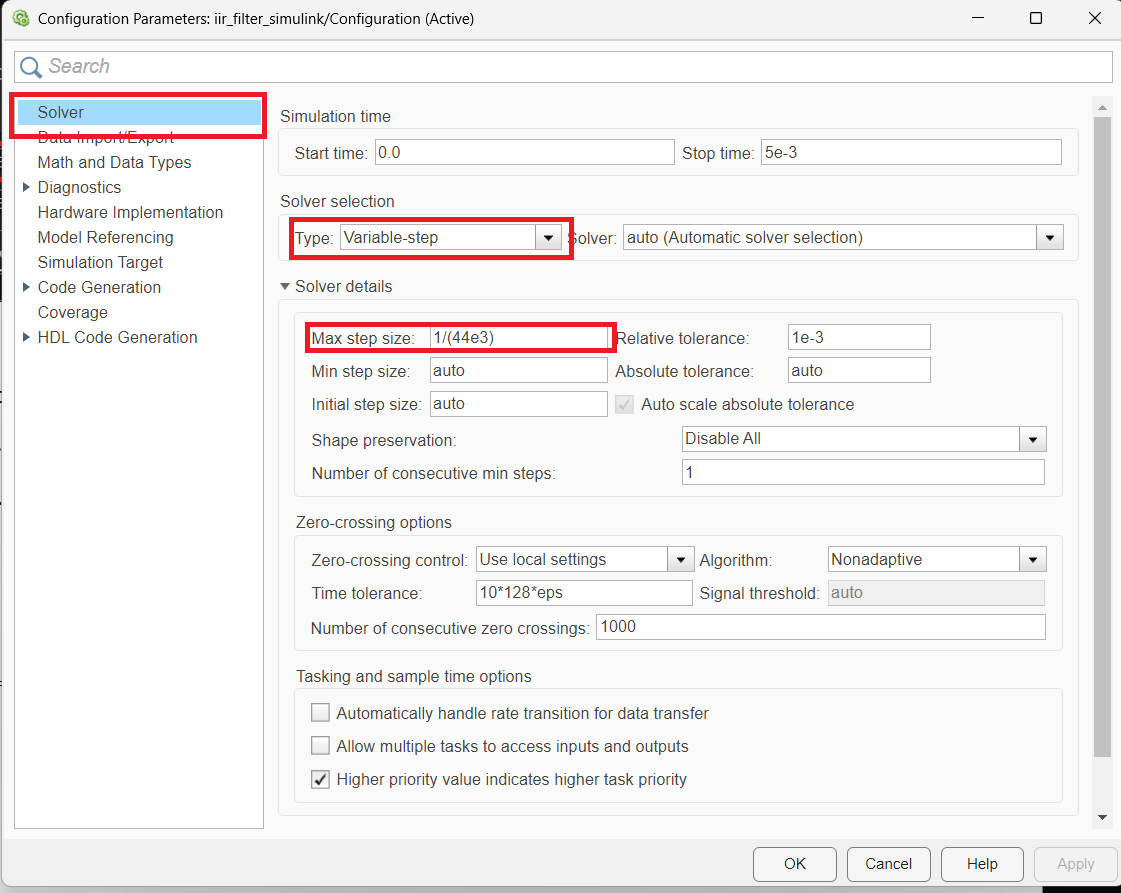
1. Clique duas vezes no Filter Designer. E configure conforme a figura a seguir.



13 – Após configurado clique em no botão ***Design Filter***. E feche a janela.

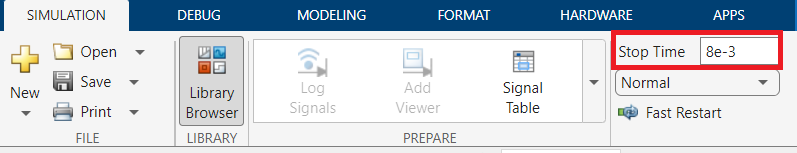
14 – Clique na aba ***MODELING***.

15 – Depois em ***Model Settings****.*

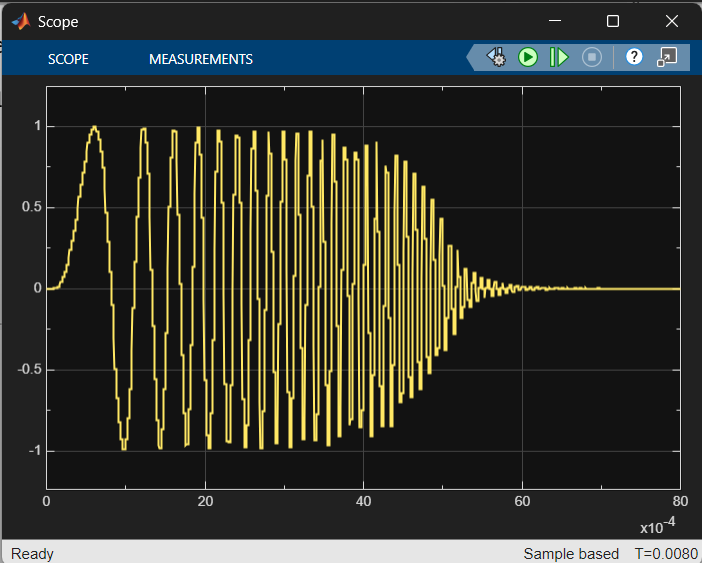


16 – Configure conforme a imagem anterior. Clique em *OK*.

17 - Clique na aba *SIMULATION*. E configure o *Stop Time* para 8 ms.

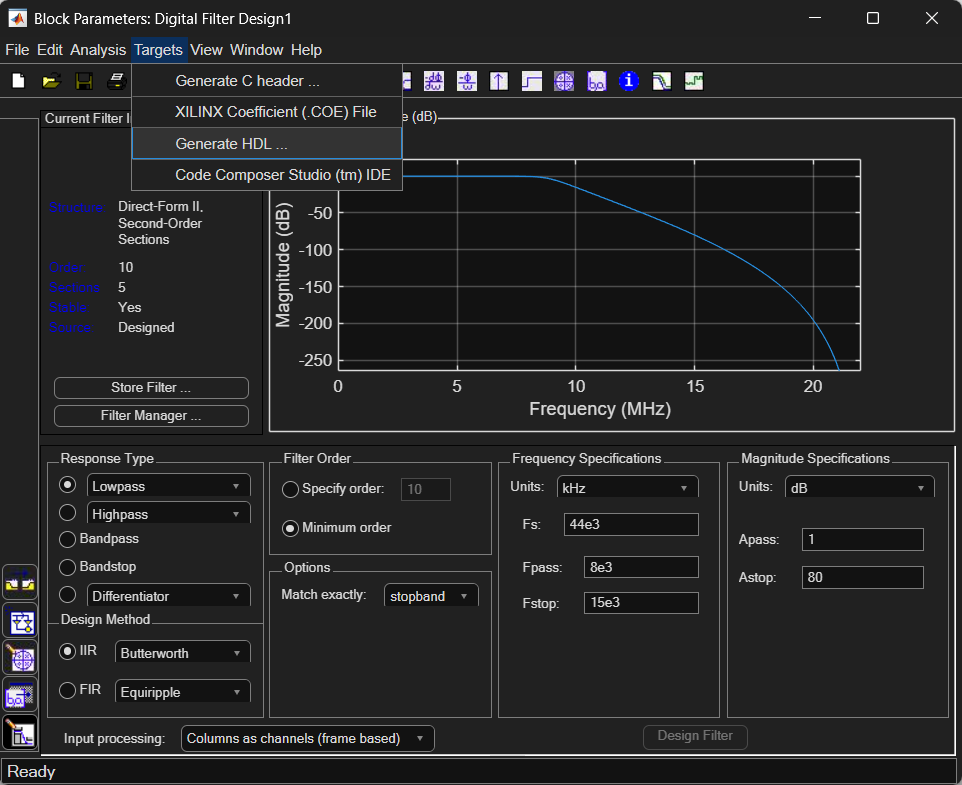


18 – Clique em Run para rodar a simulação. O resultado do Scopes de saída de filtro tem que ser o seguinte.

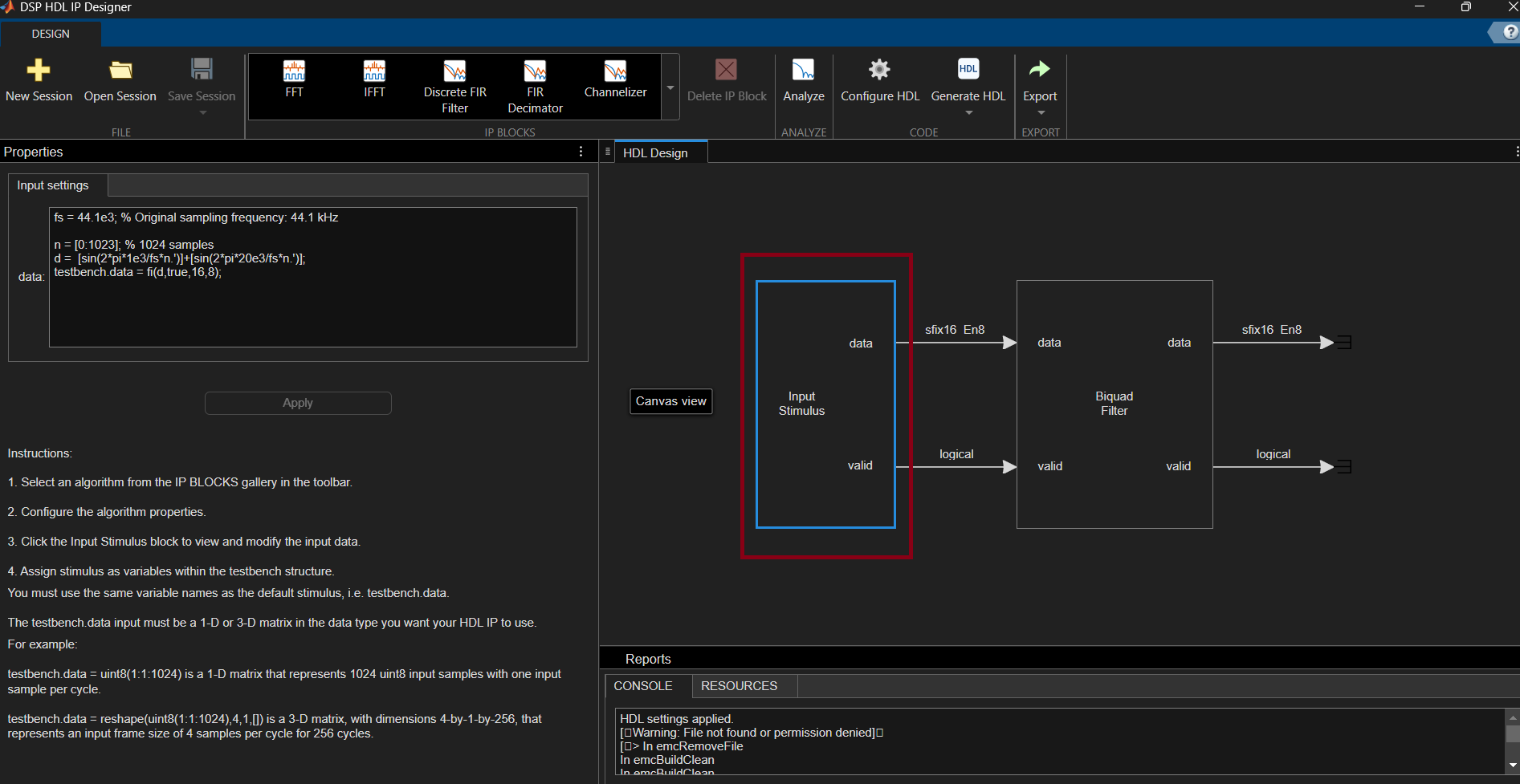


**2ª Parte – Sintetizando o filtro IIR para VHDL.**

1. Clique duas vezes em ***Filter Designer***. Aparecerá a tela abaixo.
2. Clique no menu Targets ->Generate HDL...



4 - Aparecerá a janela abaixo. E click no módulo ***Input Stimulus****.*



6 – No lado esquerdo existe um campo para codificar o testbech no Input settings. Copie e cole o código abaixo.

fs = 44.1e3; % Original sampling frequency: 44.1 kHz

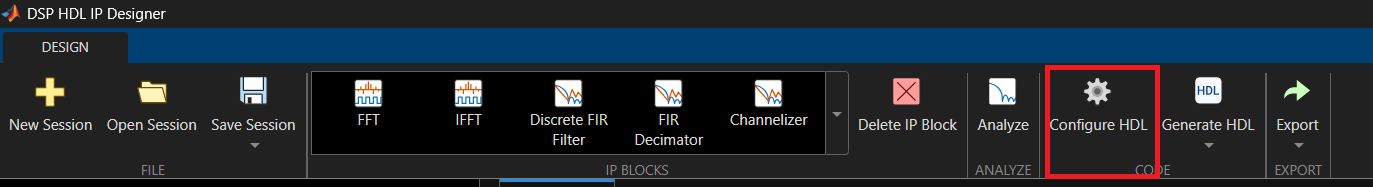
n = [0:1023]; % 1024 samples

d = [sin(2\*pi\*1e3/fs\*n.')]+[sin(2\*pi\*20e3/fs\*n.')];

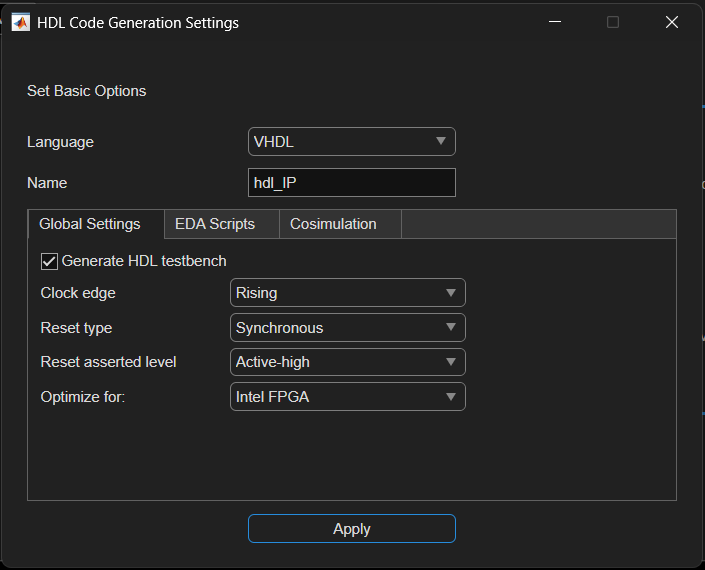
testbench.data = fi(d,true,16,8);

7 – Clique em apply.

5 – Clique em ***Configure HDL***.

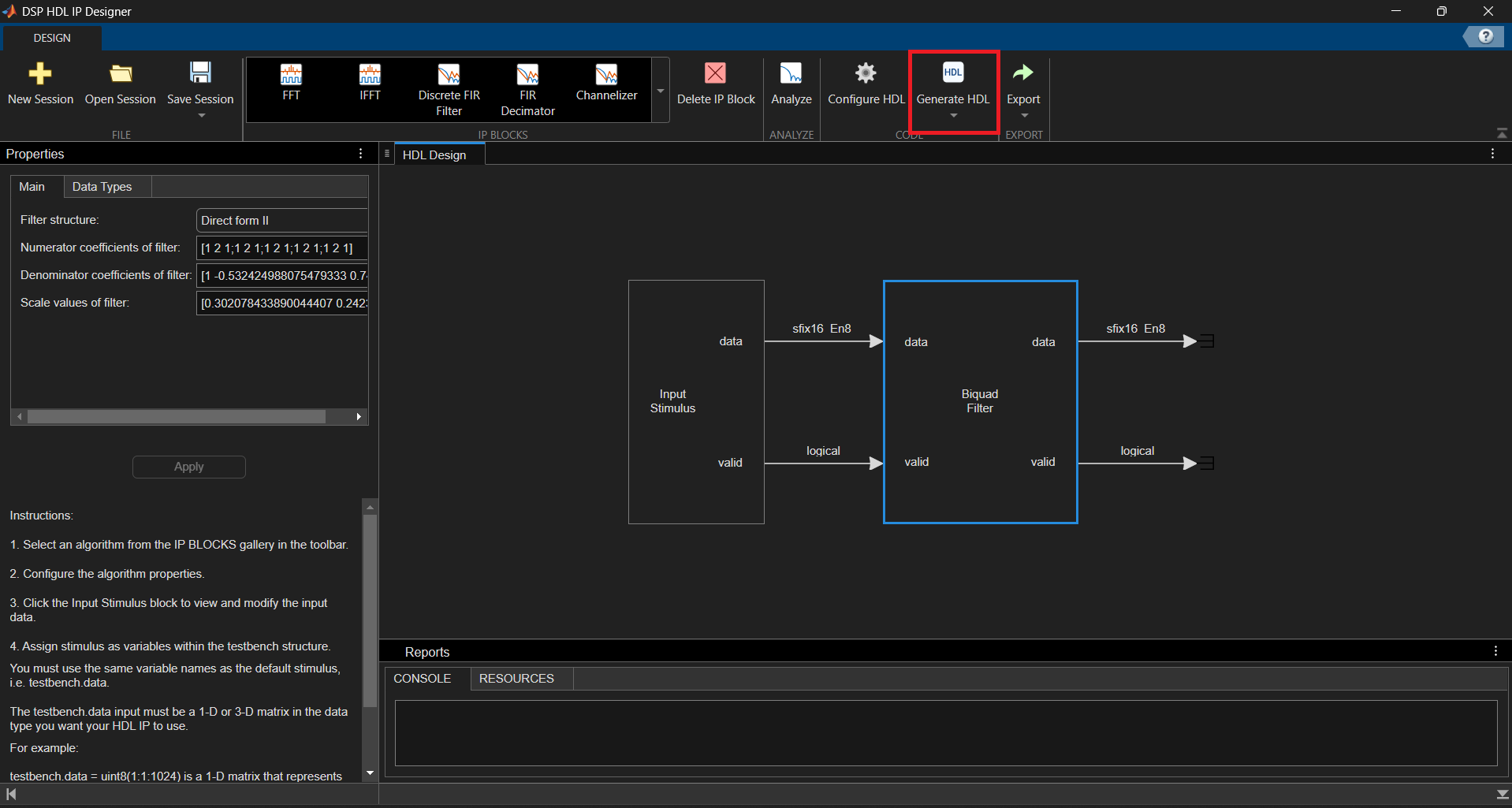


6 – A janela abaixo vai ser mostrada. Escolha a marca da FPGA que quer utilizar. Clique em *Apply*. Feche a janela.



8 – Salve o Projeto em ***Save Session***. Grave o projeto numa pasta na raiz (exemplo: c:\proj\_IIR).

7 – Clique em ***Generate HDL*.**

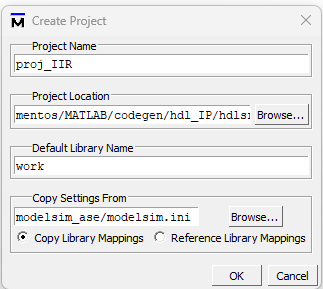


**3ª Parte Simulação no ModelSim.**

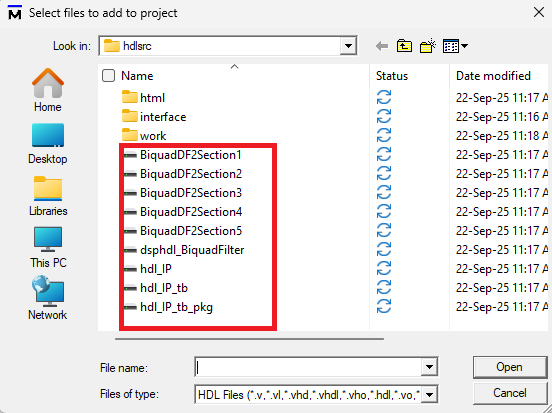
1-Abra o ModelSim.

2-Clique em File->New->Project..

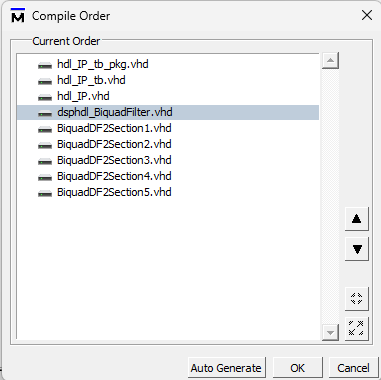
3 - Em ***Project Location***. Clique em ***Browse...*** e escolha a pasta onde estão os códigos VHDL que foram gerados pelo Matlab.



4 – Clique em ***Add Existing File***. E adicione no projeto todos os arquivos VHDL.

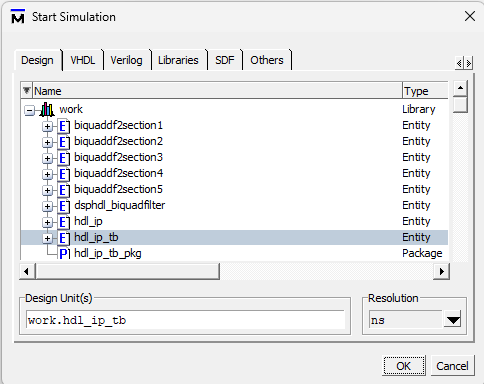


5 -Clique no menu **Compile -> Compile Order...** E coloque os arquivos na ordem que aparece na Figura abaixo. Clique em ***Auto Generate***, depois em *OK*.



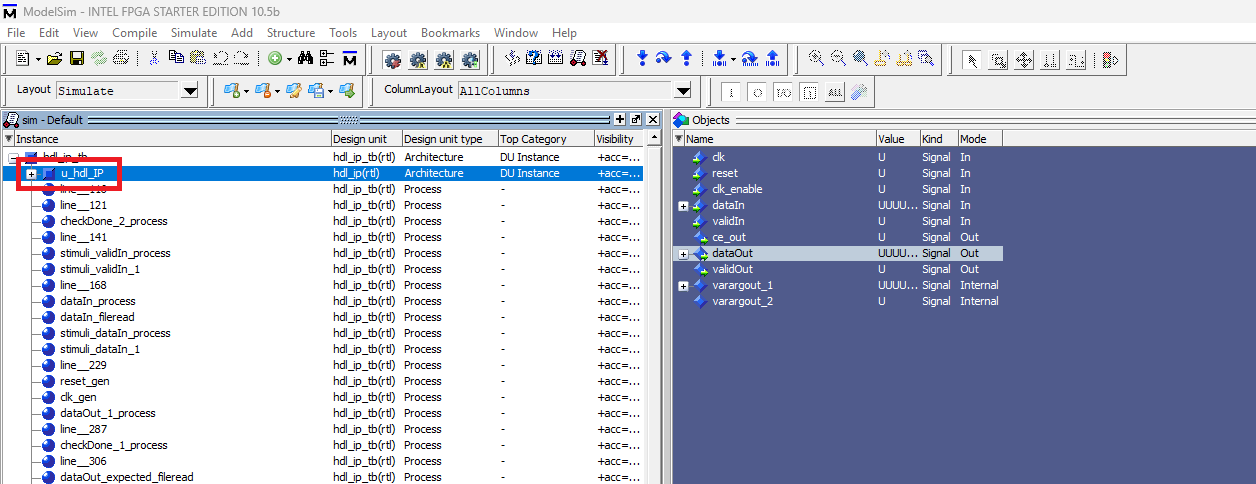
6 – Clique no menu ***Compile->Compile All...***

7 – Clique no menu ***Simulate-> Start Simulation…*** clique na pasta ***work*** e selecione o arquivo ***hdl\_ip\_tb***. Deixe a resolução em ***ns****.*Clique em***ok****.*

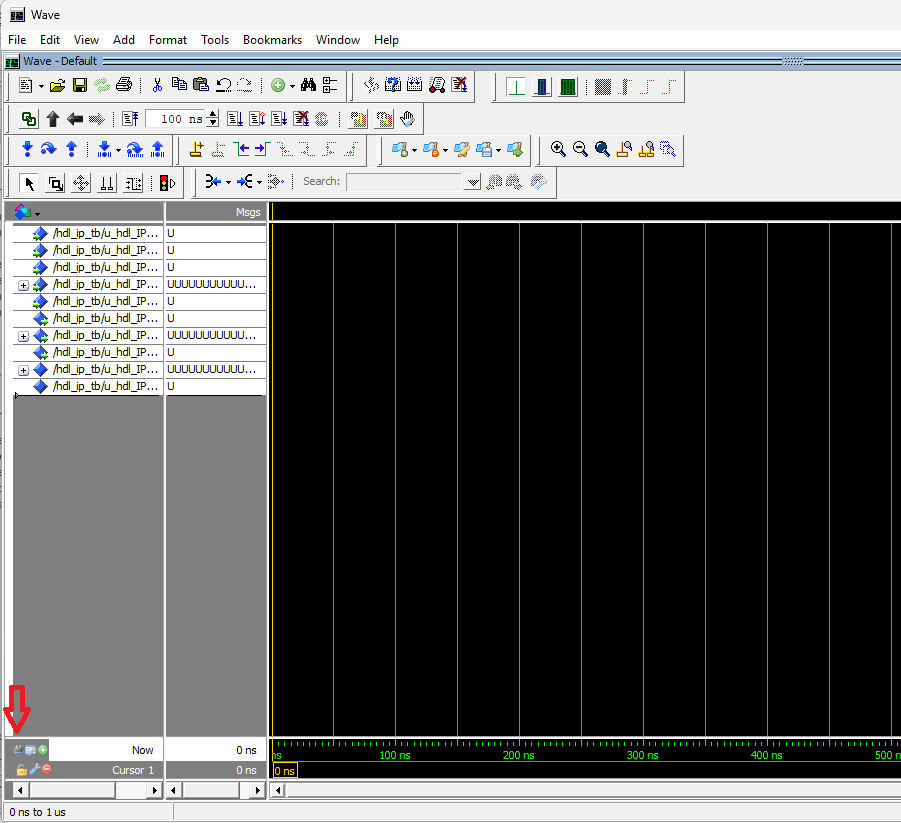


8 -Clique no menu ***View->Wave***.

9 – Selecione a instância conforme a figura.



10 – Selecione na caixa Objects, todos os sinais. E arraste para a janela Wave.

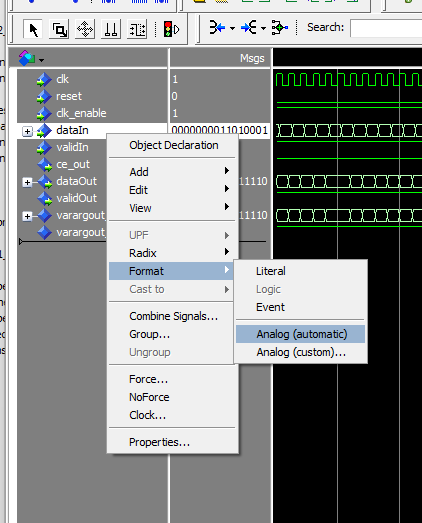


1. – Clique no botão como indicado na figura acima.

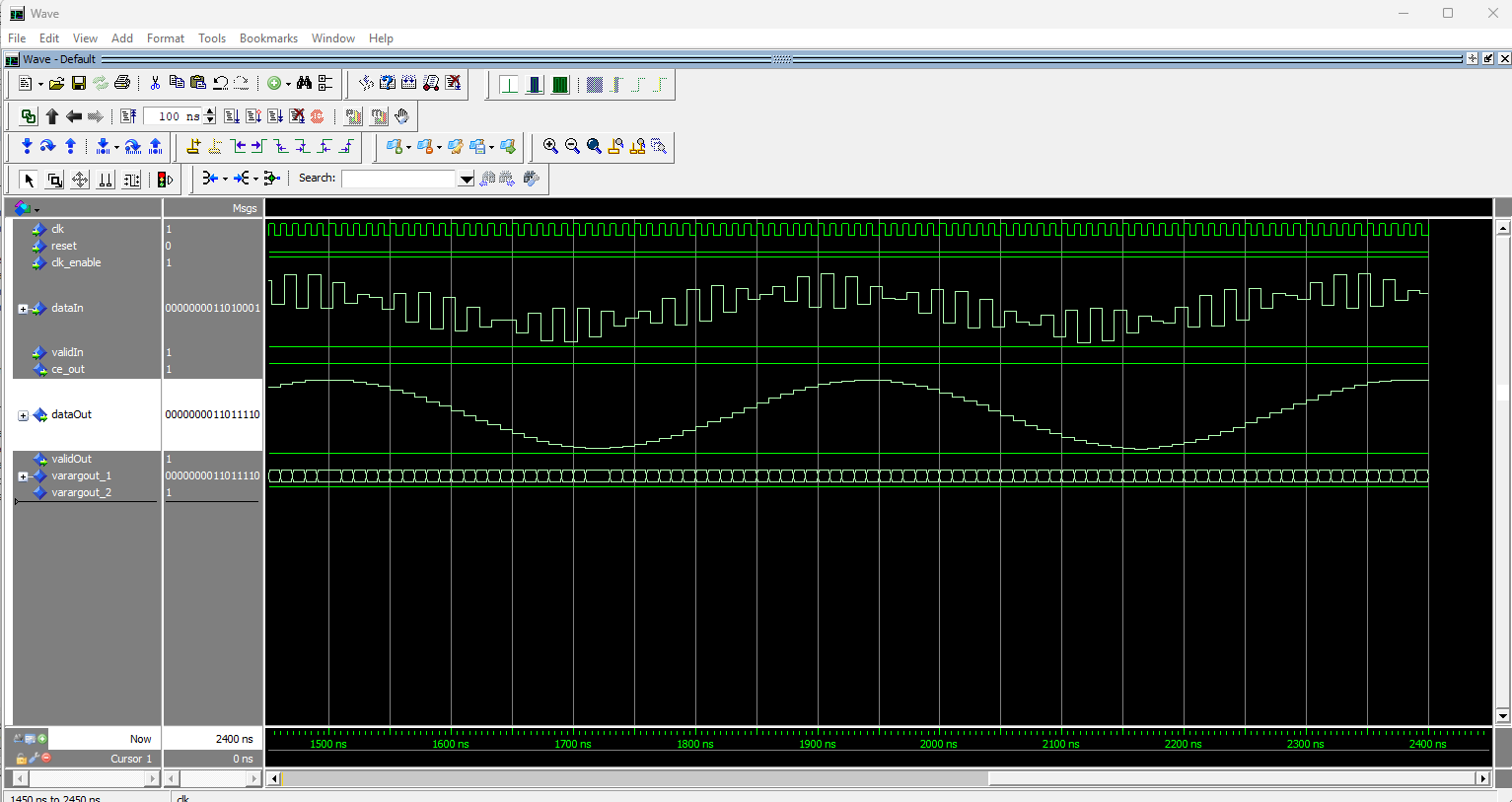
13 -Clique no botão ***Run*** várias vezes para rodar a simulação.



14 – Clique na port de entrada de sinal e selecione *Analog (automatic)*, conforme figura.



15 -O mesmo para o port de saída dataOut. O resultado será o seguinte...



**Linguagem C**

Criada por volta dos anos 1970 por [Dennis Ritchie](https://pt.wikipedia.org/wiki/Dennis_Ritchie) na AT&T Bells Lab. C é uma linguagem utilizada na maioria das arquiteturas de processadores. O C é o “pai das linguagens”, sendo que influenciou outras linguagens de programação, entre elas: C++, Java, C#, Python, JavaScript e PHP. O C também é uma linguagem utilizada para desenvolvimento de sistemas operacionais. Além disso, é amplamente usada para programação de microcontroladores de diversas arquiteturas, em sistemas embarcados - juntamente com *assembly*, neste caso para melhorar o desempenho da aplicação a ser desenvolvida.

Exemplo simples de programa em C:

**#include** *<stdio.h> /\* Pacotes com funções de entrada e saída \*/*

**int** main(void)

{

**printf**("Olá, Mundo\n!");

**return** 0; */\* Retorna 0, pois `main` retorna um `int` \*/*

}

*/\* Nova linha após fechar a chave principal \*/*

**Diretivas em C**

As principais diretivas em C, são:

* #include: Inserir o conteúdo de um arquivo de cabeçalho (.h) no código fonte.
* #define: Definir macros, que são atalhos para constantes ou expressões.
* #undef: Desfaz uma definição feita com #define.
* **Compilação Condicional:** Controlam quais partes do código são incluídas na compilação.
  + #if <expressão>: Inclui o código se a expressão for verdadeira.
  + #elif <expressão>: Uma alternativa ao #if.
  + #else: Inclui o código se a condição anterior for falsa.
  + #endif: Fecha um bloco de compilação condicional.
  + #ifdef <identificador>: Inclui código se um identificador foi definido.
  + #ifndef <identificador>: Inclui código se um identificador não foi definido.

**Tipos de variáveis**

Os principais tipos de variáveis utilizadas nesta linguagem são as seguintes:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tipo** | **Número de bits** | **Formato I/O** | **Início** | **Fim** |
| char | 8 | %c | -128 | 127 |
| unsigned char | 8 | %c | 0 | 255 |
| int | 32 | %d | -2.147.483.648 | 2.147.483.647 |
| unsigned int | 32 | %u | 0 | 4.294.967.295 |
| long int | 32 | %li | -2.147.483.648 | 2.147.483.647 |
| unsigned long int | 32 | %lu | 0 | 4.294.967.295 |
| short int | 16 | %hi | -32.768 | 32.767 |
| unsigned short int | 16 | %hu | 0 | 65.535 |
| float | 32 | %f | 1,17549x1038 | 3,40282x1038 |
| double | 64 | %lf | 2,2207x10308 | 1,79769x10308 |
| long double | 96 |  |  |  |

O nome das variáveis pode ser qualquer palavra desde que não seja uma palavra-chave da linguagem. Sendo a linguagem do tipo *case sensitive*, o nome da variável não pode inicializar com um algarismo (exemplo: 1casa). Também não é possível utilizar os seguintes símbolos:

{ ( +- \* / ; . , ?

**Palavras Reservadas**

As seguintes palavras reservadas da linguagem C são estas:

|  |  |  |  |
| --- | --- | --- | --- |
| **auto** | **double** | **int** | **struct** |
| **break** | **enum** | **register** | **typedef** |
| **char** | **extern** | **return** | **union** |
| **const** | **float** | **short** | **unsigned** |
| **continue** | **for** | **signed** | **void** |
| **default** | **goto** | **sizedof** | **volatile** |
| **do** | **if** | **static** | **while** |

**Operadores aritméticos**

Estes operadores são utilizados em operações:

|  |  |
| --- | --- |
| **Operador** | **Operação** |
| + | soma |
| - | subtração |
| \* | multiplicação |
| / | Divisão inteira |
| % | Resto da divisão |
| ++ | Incremento |
| -- | Decremento |

A precedência dos operadores aritméticos são, primeiramente de multiplicação, divisão e módulo (%). Após estes, são a adição e subtração. Pode-se também utilizar os parênteses para alterar a ordem natural de precedência.

**Operadores relacionais**

Os operadores relacionais são usados principalmente em condições de laço, for, while e em condicionais: if e else if.

|  |  |
| --- | --- |
| **Operador** | **Operação** |
| == | igual |
| != | diferente |
| > | Maior que |
| < | Menor que |
| >= | Maior ou igual |
| <= | Menor ou igual |

**Ponteiros em C**

A definição de ponteiro em C, é que este tipo é uma variável que armazena o ENDEREÇO de uma variável. Exemplo:

**int** idade = 20;

**int** \*valor;

**int**: //é o tipo da variável \*valor.

valor //: é o endereço de memória da variável \*valor.

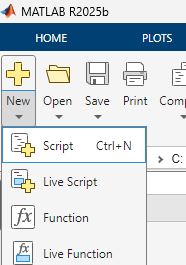
valor = &idade;

*&idade* é uma maneira para declarar o endereço da variável *idade*.

**Passo-a-passo como gerar código C a partir de um Script Matlab**

1. - Abra o Matlab R2025b.

2 – Clique em New-> Script.



1. – Copie este código para o Script da média móvel.

function [data\_out] = moving\_avg(num\_terms, matrix\_in)

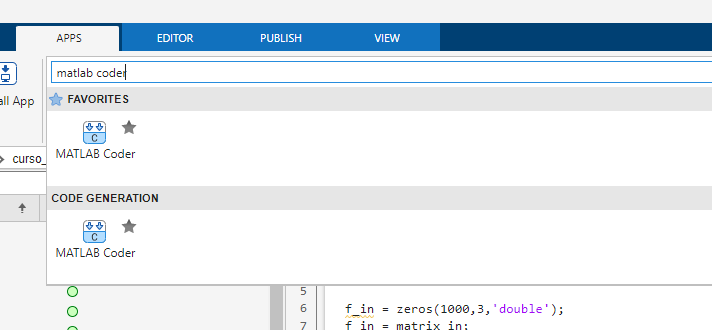
coef=(double((1/num\_terms)).\*ones(1,num\_terms));

data\_out = conv(coef,matrix\_in);

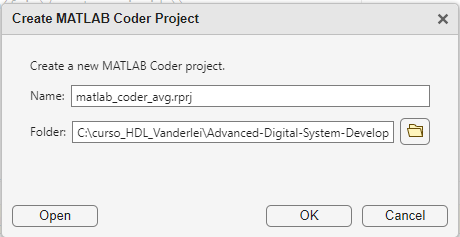
end

4- Salve o script em uma pasta de sua preferência.

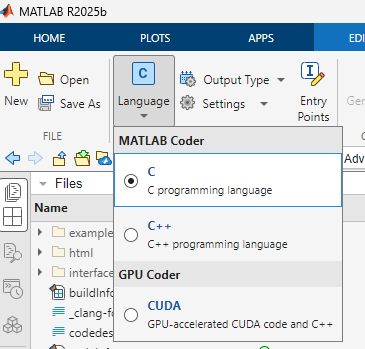
5 -Procure na aba APP o ***Matlab Coder C.***

.

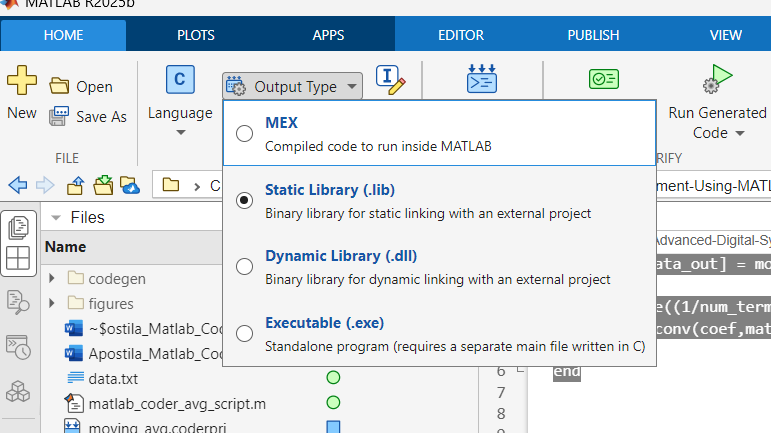
6 -Dê um nome para o projeto. E escolha o local no campo Folder.



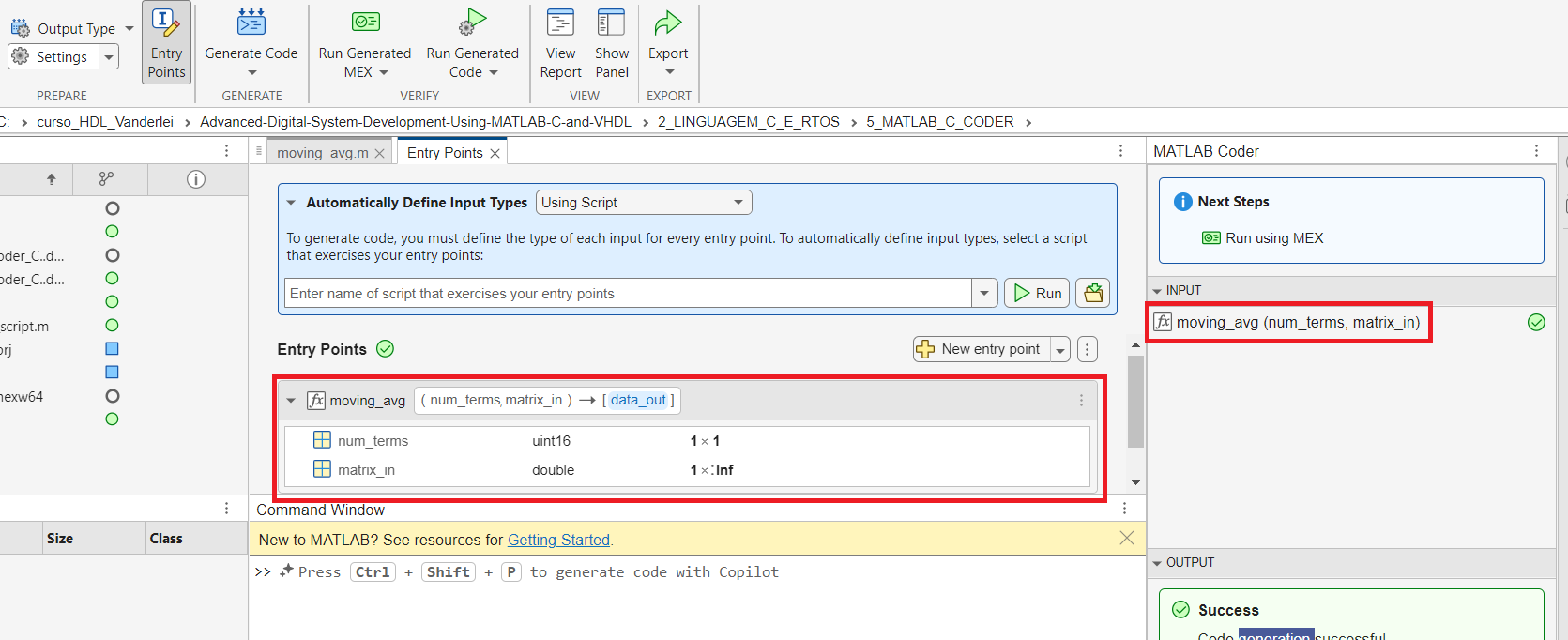
8 - Selecione a linguagem C, conforme figura abaixo.



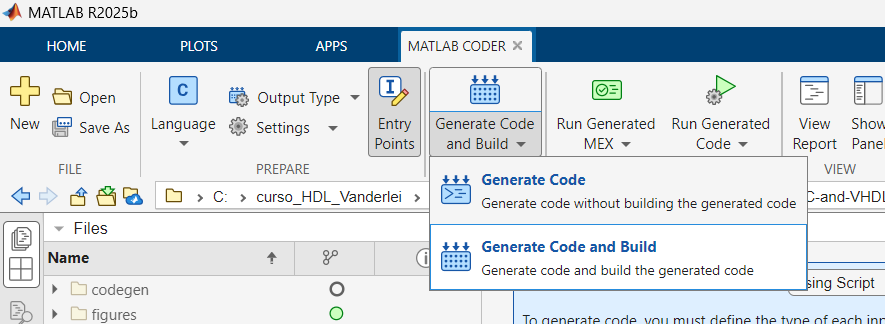
1. – Clique em Output Type->Static Library (.lib)



1. – Clique e configure os tipos de variáveis, conforme indicado na figura abaixo.

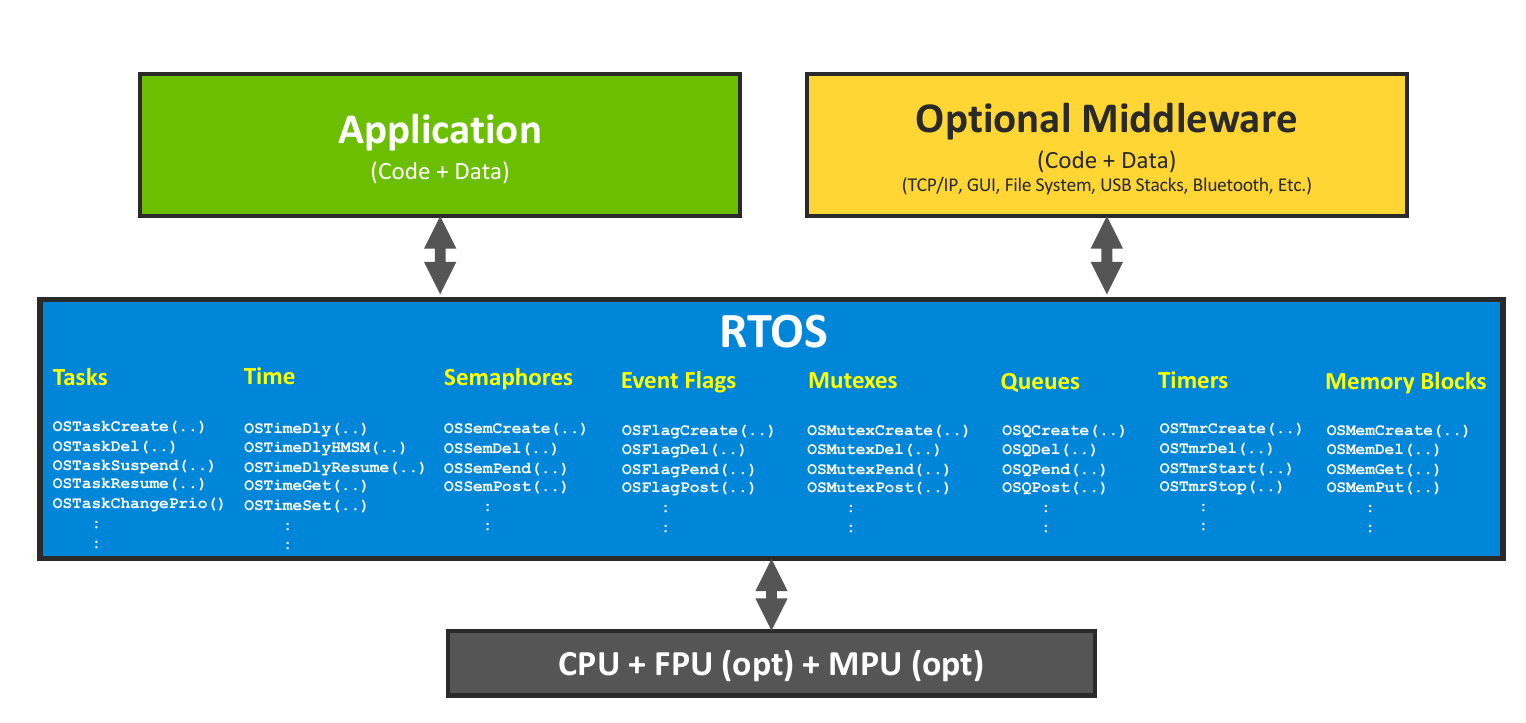


1. – Execute o ***Generate Code and Build.***



***RTOS (Real Time Operational System)***

Os Sistemas Operacionais de Tempo Real (RTOS) vêm sendo utilizados devido ao aumento da grande complexidade dos sistemas embarcados que estão sendo desenvolvidos atualmente. Principalmente do uso intensivo do sistema de vários periféricos, entre eles: USB, Ethernet, Pilha TCP/IP, Bluetooth, GPIOs, WhatchDog e entre outros. Assim, dessa maneira, foi criado o conceito de sistemas operacionais de tempo real. Dentro deles, podem ser criado *threads,* ou mais comumente chamados de *tasks*. Portanto, não há mais neste tipo de sistema o super loop, comumente utilizado em sistemas microcontrolados, mas sim de micro loops infinitos. A figura abaixo é exibida a estrutura de um sistema RTOS.



**Fonte:** [**https://www.silabs.com/documents/public/presentations/introduction-to-real-time-operating-systems.pdf**](https://www.silabs.com/documents/public/presentations/introduction-to-real-time-operating-systems.pdf)

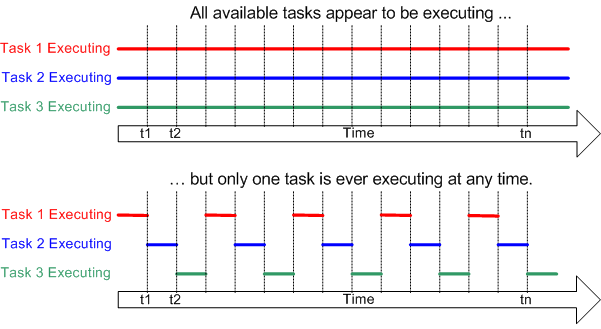
Para que ocorra o chaveamento de tasks, é necessário o compartilhamento destas tarefas e o gerenciamento de recursos utilizando o conceito de *time slice*.

Os RTOS mais utilizados no mercado (quando escrito este documento) são os: FreeRTOS (Amazon), ThreadX (Microsoft) e Zephyer (Linux Foundation). Estas empresas pensando no desenvolvimento de aplicações em nuvem e *Internet of Thinks* (IoT), foram adquirindo sistemas operacionais de outras empresas. E adaptando o para suas aplicações em nuvem, entre elas: AWS (Amazon) e Azure (Microsoft).

O FreeRTOS aqui tratado não é exatamente Free (Free vem de *Freedom*) não é *open source* ou seja, se houver a necessidade de modificação deste sistema, será necessário o compartilhamento deste sistema modificado, à comunidade, ou o pagamento do custo desta modificação. É recomendado o acesso ao site da FreeRTOS e seus respectivos documentos, para que não haja dúvida sobre esses a assuntos sobre licenças impostas no seu respectivo sistema operacional.

O suporte do FreeRTOS não existe *help desk*, portanto quando houver dúvida sobre a utilização do sistema é preciso contatar a comunidade ou documentos fornecidos pelo site do mesmo.

Em se tratando de funcionamento do FreeRTOS, vemos na figura abaixo, retirada do site do sistema RTOS, em que é possível habilitar o modo preemptivo, na qual na visão do usuário não é possível distinguir que há o chaveamento entre *tasks*. A impressão é que tudo está ocorrendo ao mesmo tempo (para o usuário), mas na verdade existe o chaveamento de execução de *tasks* durante o passar do tempo, conforme também de suas prioridades.



**Prática para utilizar o FreeRTOS no Matlab**

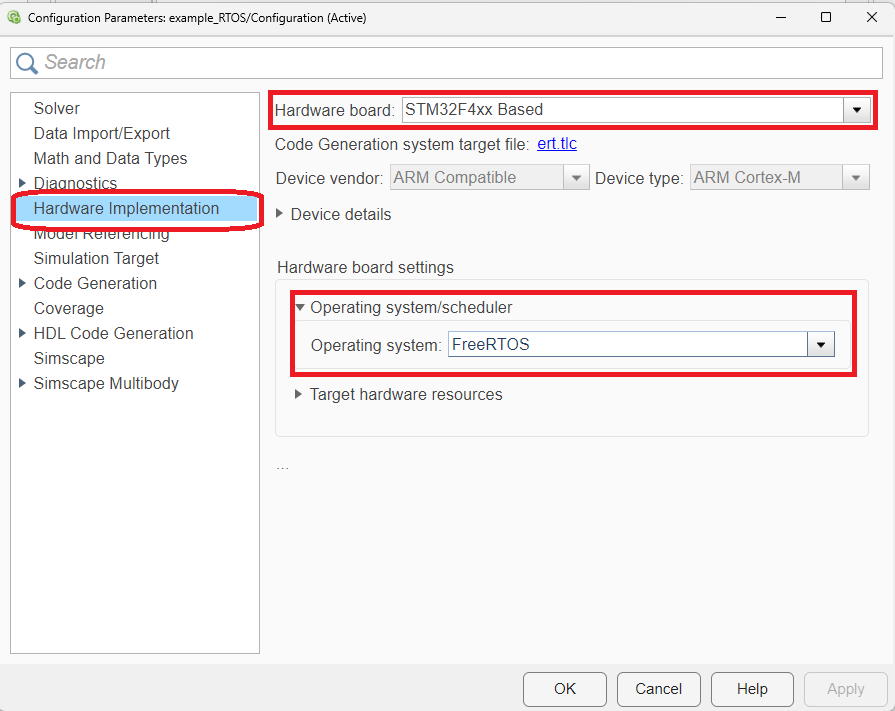
1 -Abra o Matlab **R2025b.**

2 -Abra o Simulink.

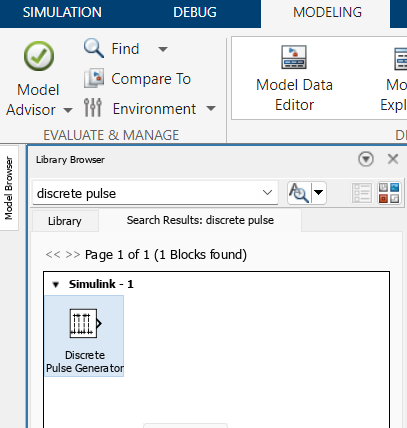
3 – Clique na aba ***MODELING***, depois em ***Model Settings.***



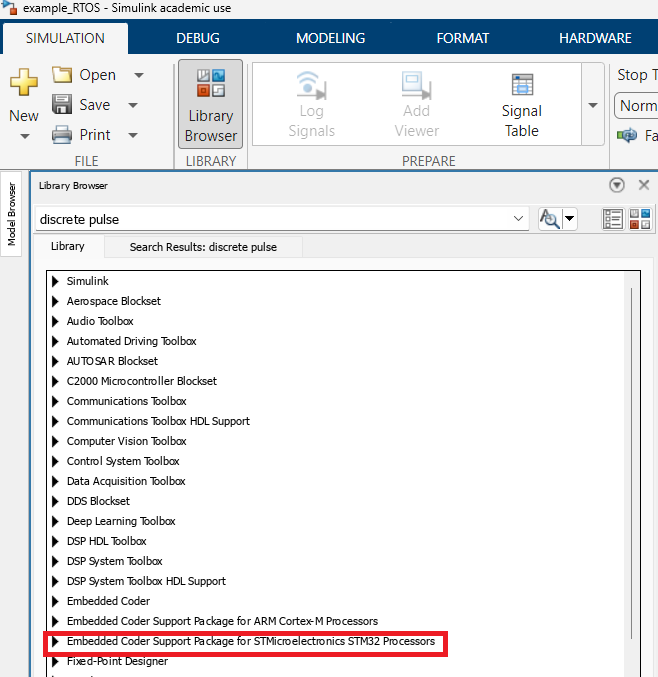
4 -Clique em *Hardware Implementation*. Escolha a placa que quer utilizar e escolha o sistema operacional FreeRTOS. Clique em *OK*.



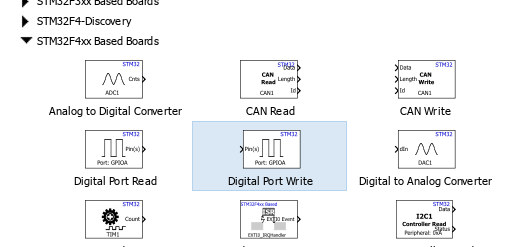
5 – Clique na aba ***SIMULATION****.* Depois clique em ***Library Browser****.* Busque por ***Discrete Pulse Generator.*** Coloque dois na área de trabalho.



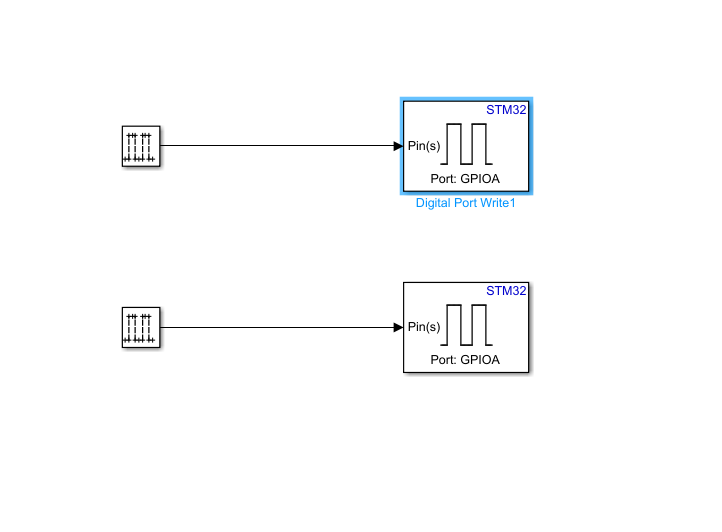
6 – Depois procure no ***Library Browser*** o toolbox ***Embedded Coder Support Package for STMicrocontrollers STM32 Processor.*** Após isso procure pela família da placa correspondente.



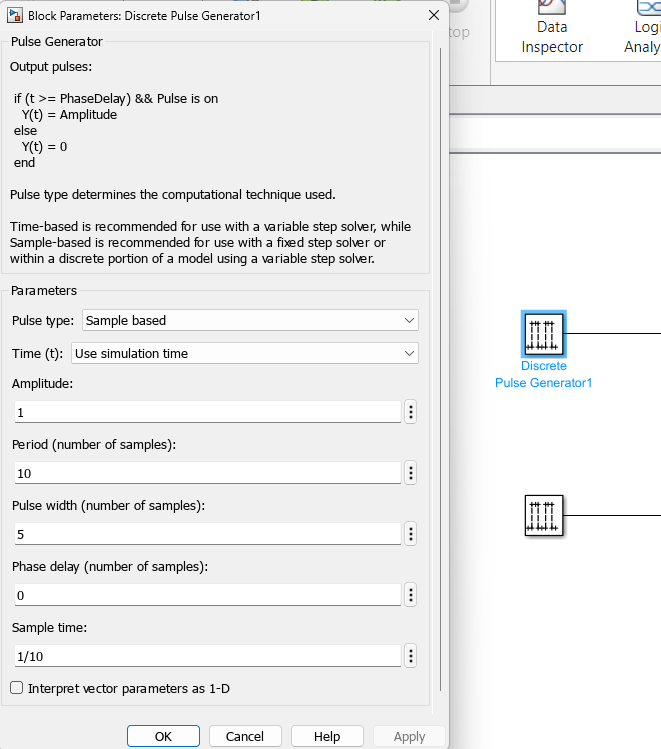
7 – Procure pelo bloco *Digital Port Write.*



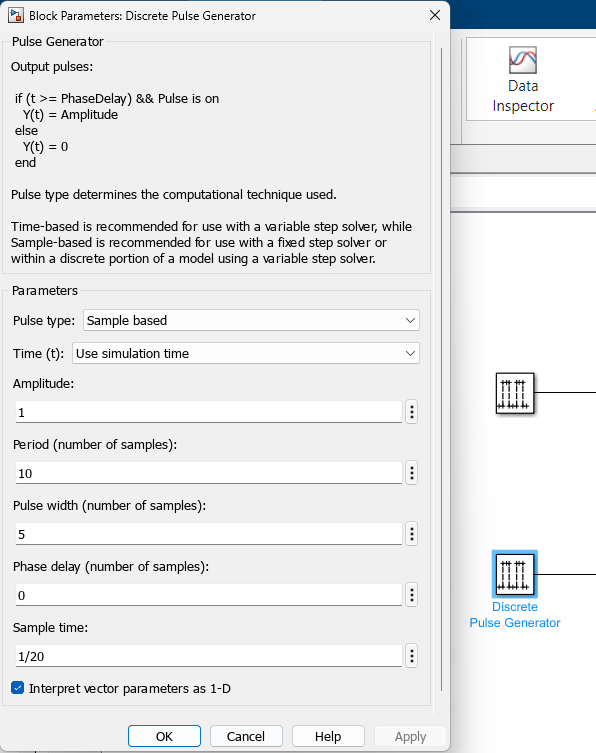
8 - Faça as seguintes ligações conforme a figura abaixo.



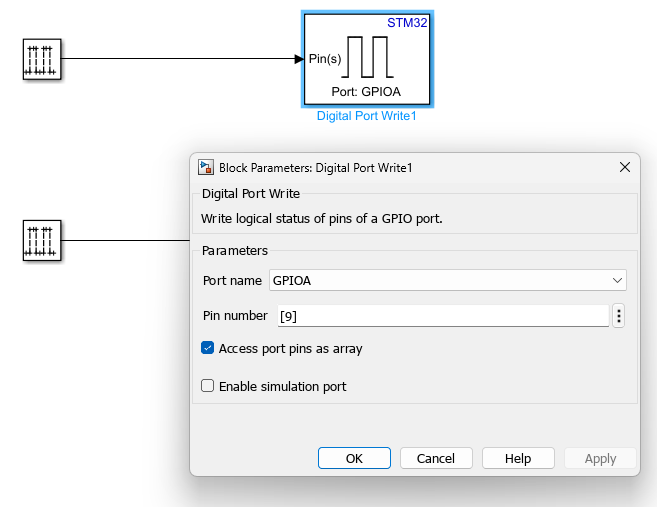
9 – Configure o seguinte o primeiro ***Pulse Generator***.



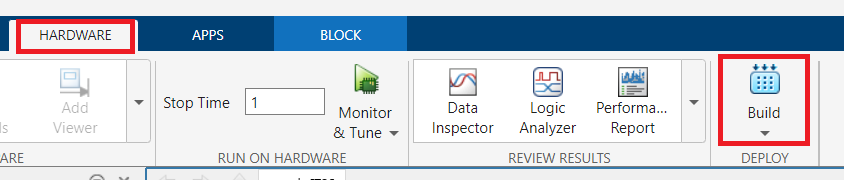
10 - Configure o seguinte o segundo ***Pulse Generator.***



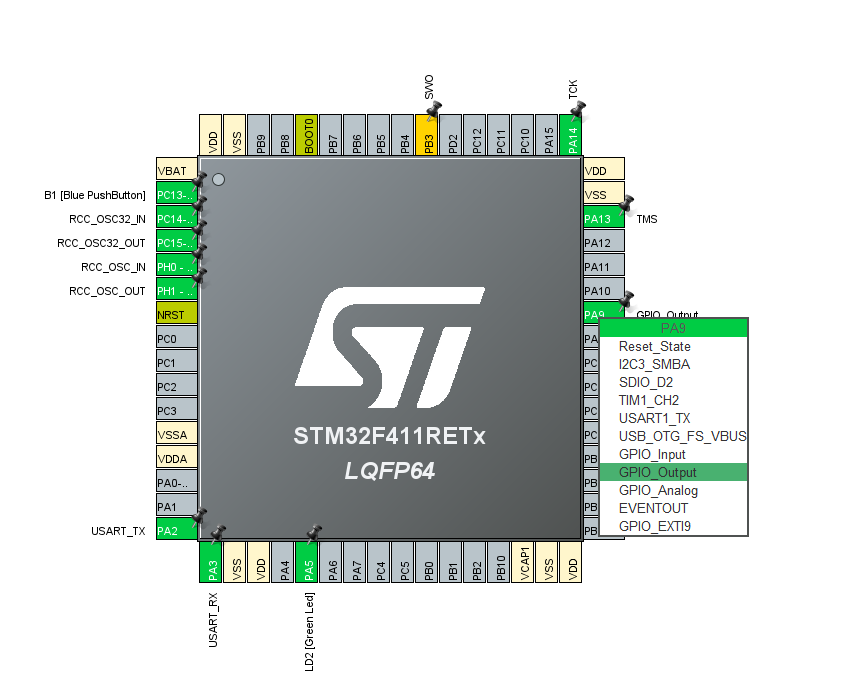
11 – No primeiro Digital Port Write. Escolha o *Port name*. E o número do pino. Neste caso foi utilizado o port do led que tem na placa. Faça o mesmo para o outro *Digital Port Write* e escolha um port e o número do pino desejável para ver o sinal no osciloscópio.



12 – Salve o projeto e clique em Build.

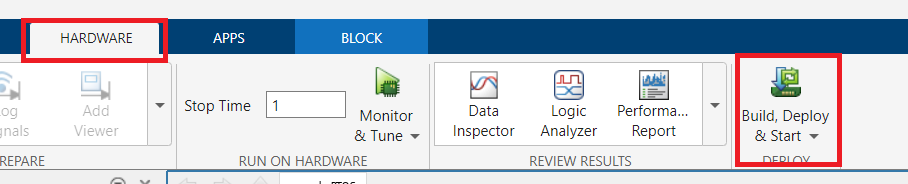


13 - Talvez seja necessário no STMCubeMX, configurar os pinos.



14 – Conecte a placa na porta USB do PC

15 – Clique em ***Build Deploy & Start*** para compilar todo projeto e gravar na placa.

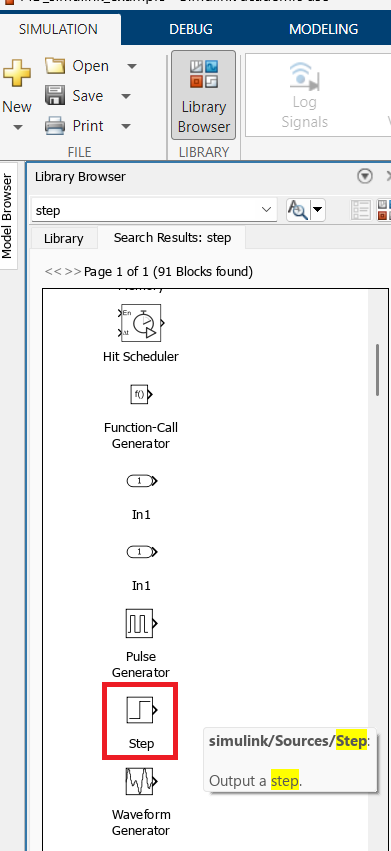


**Passo a Passo PID**

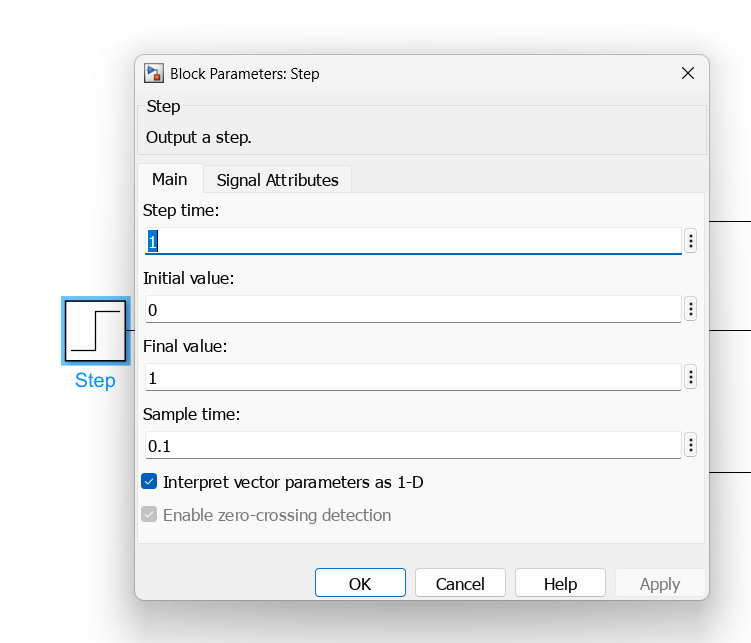
1 – Abra o **Matlab R2025b**.

2 – Abra o **Simulink**.

3 – Procure no **Library Browser o block Step**.

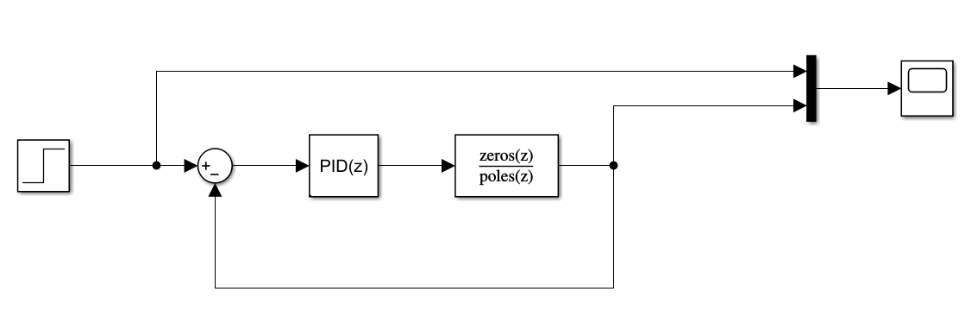


4 – Configure o modulo ***Step*** como figura a seguir.



5 – Procure o modulo ***Sum, Discrete PID Controller, Mux, Scope*** e ***Discrete Zero-Pole*** no *Library Browser.* E coloque na área de trabalho.

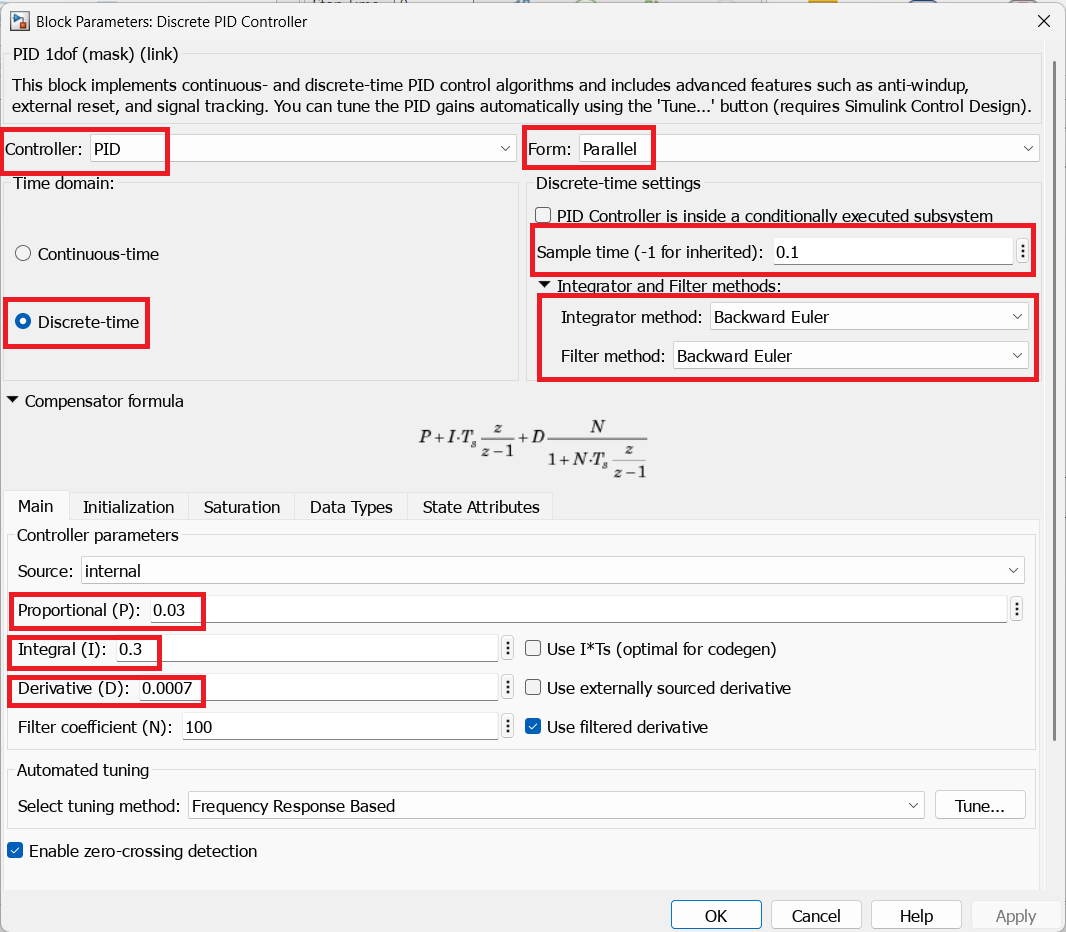
8 – Conecte os módulos como a figura a seguir.



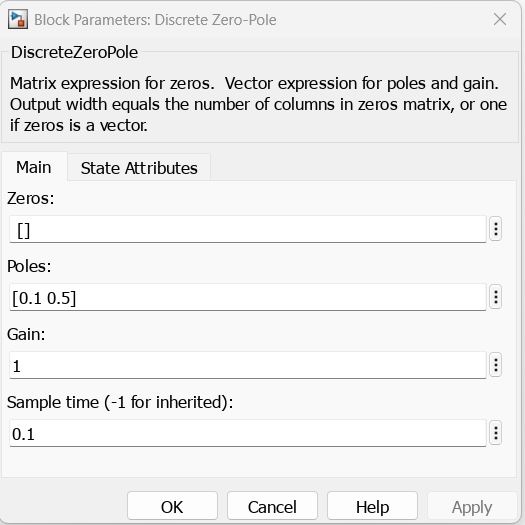
9 – Configure o ***sum***.



10 – Configure o PID conforme figura a seguir.



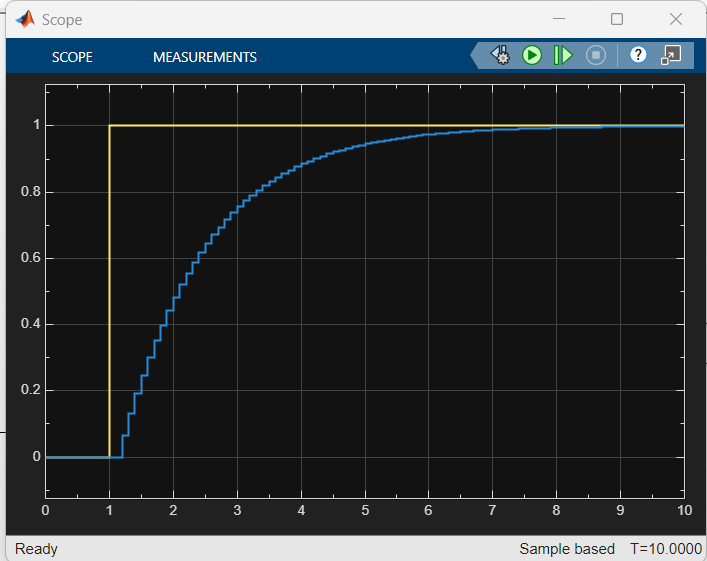
11 – O ***Discrete Zero-Pole***.



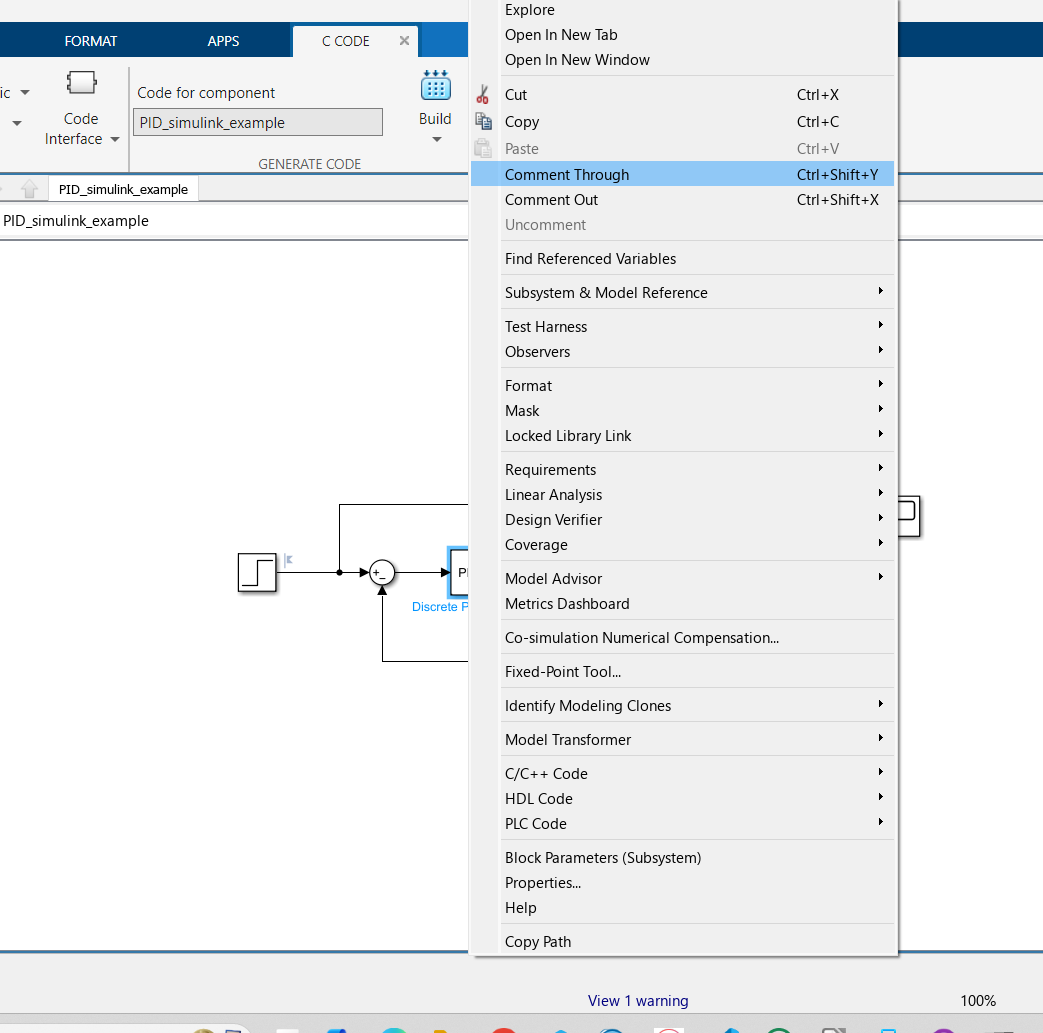
12 – Configure o ***Stop Time*** em 10. Depois clique em *Run.*

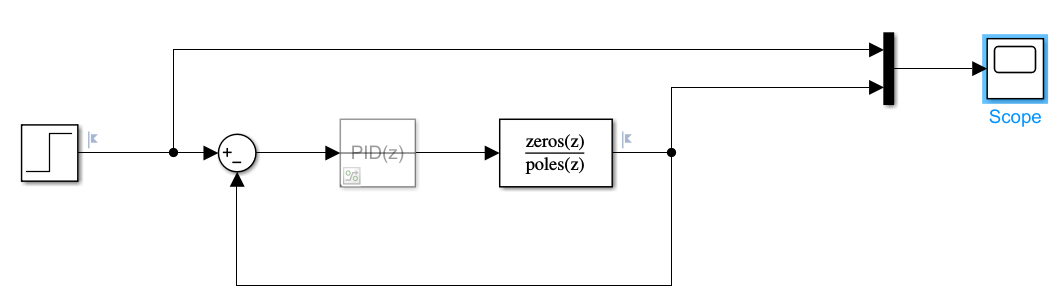
**

13 – Clique duas vezes em ***Scope***.

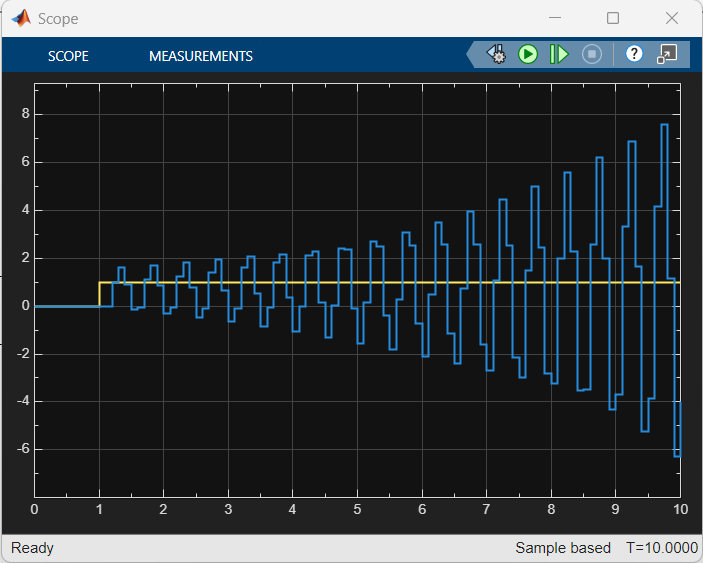


14 – Comente o modulo PID, clicando com o botão direito do mouse.





15 - Simule novamente e clique duas vezes em **Scope**.



**Redes Neurais Artificiais**

O conhecimento segundo Fischler e Firschein (1987) é dado como a seguinte definição:

“Conhecimento se refere à informação armazenada ou a modelos utilizados por uma pessoa ou máquina para interpretar, prever e responder apropriadamente ao mundo exterior.”

Outro autor que expõe suas ideais sobre o conhecimento é Anderson (1988), o qual diz que a “representação de conhecimento no interior de uma rede neural é muito complicado.” No entanto há regras para a representação do conhecimento, uma delas:

“**Regra 1**: Entradas similares de classes normalmente devem produzir representação similares no interior da rede, e, portanto, devem ser classificadas como pertencentes à mesma categoria.”

Existem várias medidas de similaridade, mas a mais comum é o conceito de distância euclidiana, sendo que representa um vetor m – por – 1. Em que os seus elementos são representados por números Reais ():

Com este vetor, pode-se deduzir da distância euclidiana entre um para de vetores m por 1, ; definida por:

Dessa maneira, quanto menor a distância euclidiana, maior a similaridade entre os vetores e . Outra medida de similaridade é o produto escalar dos vetores por .



Quanto maior o produto escalar, mais similar são os dados de entrada com a saída. E justamente esta similaridade é utilizada nos neurônios (*perceptron*) de redes neurais.

As Redes Neurais Artificiais são baseadas no modelo biológico do cérebro humano. O treinamento destas redes neurais através da aplicação de dados e sabendo os resultados na saída desta rede neural. Utilizando algoritmos como o *backpropagation*. Através deste algoritmo, é possível ajustar os parâmetros desta rede, ou seja, os pesos das sinapses?? e ativação dos seus vieses (*bias*).

A unidade primária das redes neurais artificiais é o *perceptron*. Sendo que é alimentado por entradas, representados por xi e estes multiplicados por seus respectivos pesos, representados por wi. Além destes sinais é aplicado o viés (*bias*) (McCulloch, Pitts, 1943). Diferentemente de uma rede neural biológica, que conduz seus sinais quimicamente e eletricamente. As redes neurais artificiais conduzem eletricamente, podendo ser digitais ou analógicas. Sendo que é mais comum as redes artificiais digitais, pois é mais facilmente ajustar os seus pesos e bias, conforme a necessidade e treinamentos destas.

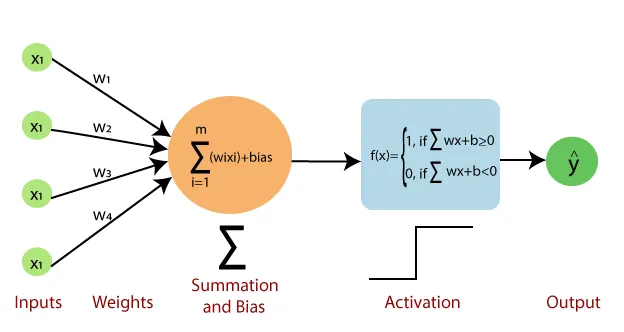


Figura 4: Perceptron.

**Redes Neurais Informadas por física (*PINN - Physics-Informed Neural Networks*)**

Também chamadas de Redes Neurais Treinadas por Teoria (*TTNs*). Têm incorporada aproximadores de funções universais de leis da física, com intuito de resolver equações que não podem ser resolvidos por meio analítico. Principalmente leis da física que governam sistemas dinâmicos.

Estas equações diferenciais são resolvidas por meio métodos numéricos, e por isso são dadas como aproximações do que descritas por equações diferenciais exatas. É onde as redes neurais profundas, convolucionais e recorrente perdem robustez, e falha para garantir convergência. (Maziar, Perdikaris, Karniadakis, 2017).

Indo na direção de resolver estes problemas de modelagem de sistemas físicos e biológicos, onde há um vasto conhecimento que não é utilizado em máquinas de aprendizagem modernas - problemas que também depende da variável tempo são levados em consideração (Maziar, Perdikaris, Karniadakis, 2017), as PINNs vêm em encontro para preencher esta lacuna.

As aplicações segundo os autores Farea et al. (2024) são: Dinâmica de Fluidos, Ciência de Materiais, Sistemas Estruturais, Mecânica Quântica, Geofísica, Sistemas de Energia e Oncologia.

No entanto alguns desafios são apresentados (Farea et al., 2024): estabilidade computacional, tempo de treinamento, ajuste de hiper parâmetros são alguns dos gargalos de treinamentos distribuídos, além de computação paralela.

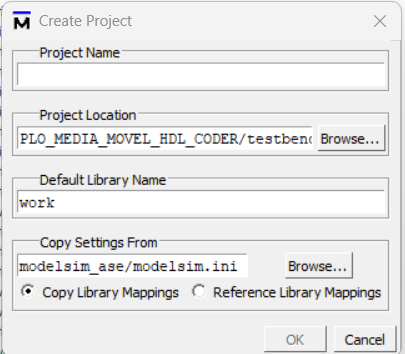
**Apêndice A**

**ModelSim Passo-a-passo**

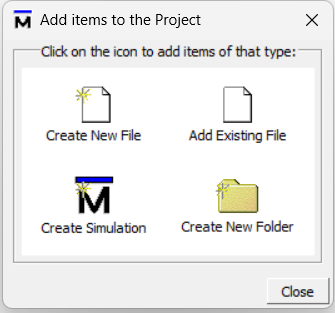
Para a simulação de circuitos no ModelSim, iremos simular um contador de quatro bits, primeiramente é necessário abrir o programa. Será visualizada a janela a seguir e clique na aba ***Projects****.*



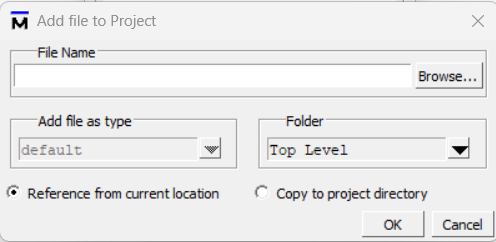
2-Para criar um projeto. Ir em **File->New->Project...**



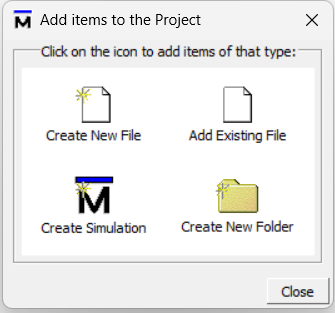
3-Escolha um nome para o projeto, e escolha um ***Project Location*** diferente do default, de preferência uma pasta na raiz (por exemplo, C:\exemplo\_counter). Clique em **OK**.



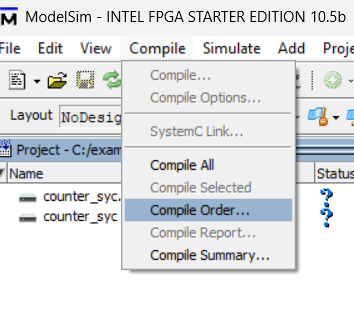
4- Clique no ícone ***Add Existing File***.



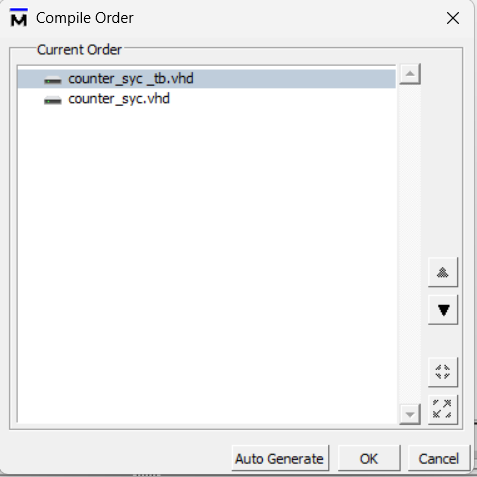
5- Clique em ***Browse***. E escolha os arquivos em VHDL do projeto do contador, já prontos: counter\_syc.vhd e counter\_syc\_tb.vhd.



6- Clique no botão ***Close*** em ***Add items to the Project***.

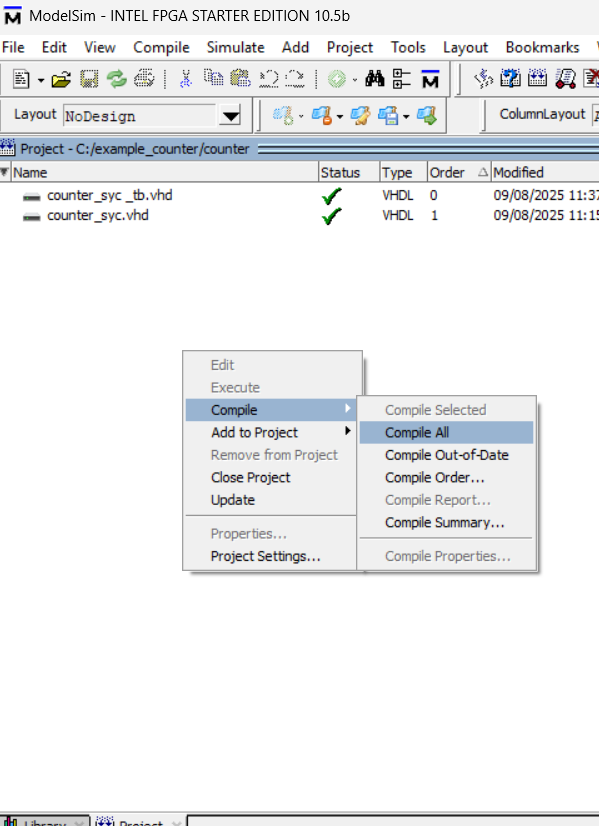


7- Clique no menu ***Compile*** e depois em **Compile Order...**

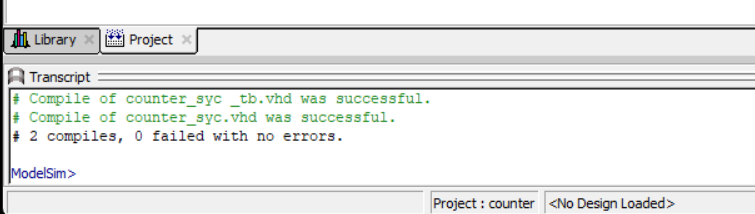


8- Se o arquivo counter\_syc\_tb.vhd estiver no topo, clique em ***Auto Generate*** e depois em ***OK***. Caso contrário clique no arquivo counter\_sys\_tb.vhd e na seta para cima. Depois clique em ***Auto Generate***. Clique em ***OK***.

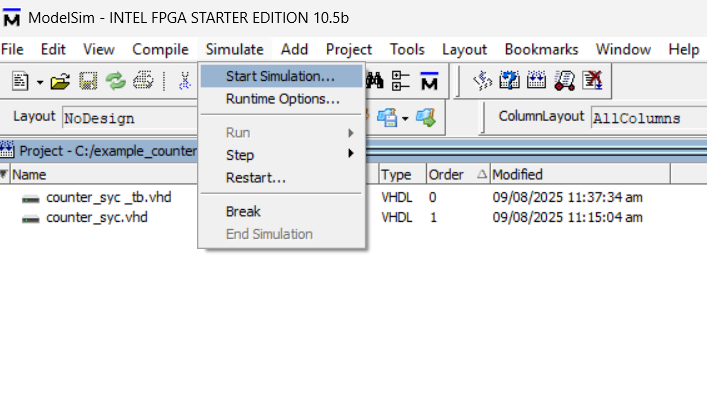
9 - Clique com o botão direito do mouse na área abaixo dos arquivos e ***selecione Compile All.***



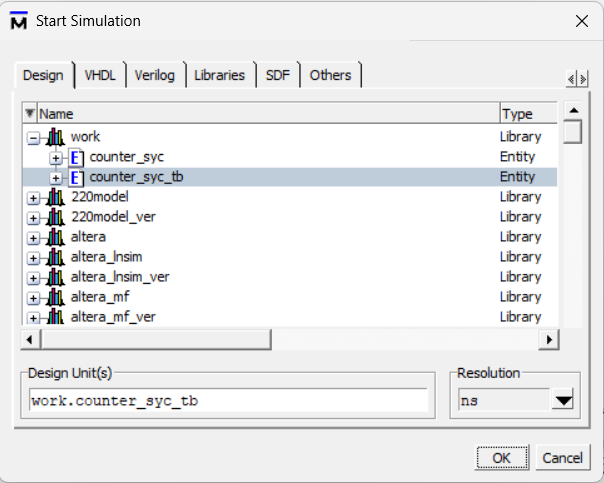
10 – Se a compilação for bem sucedida conforme a figura abaixo:



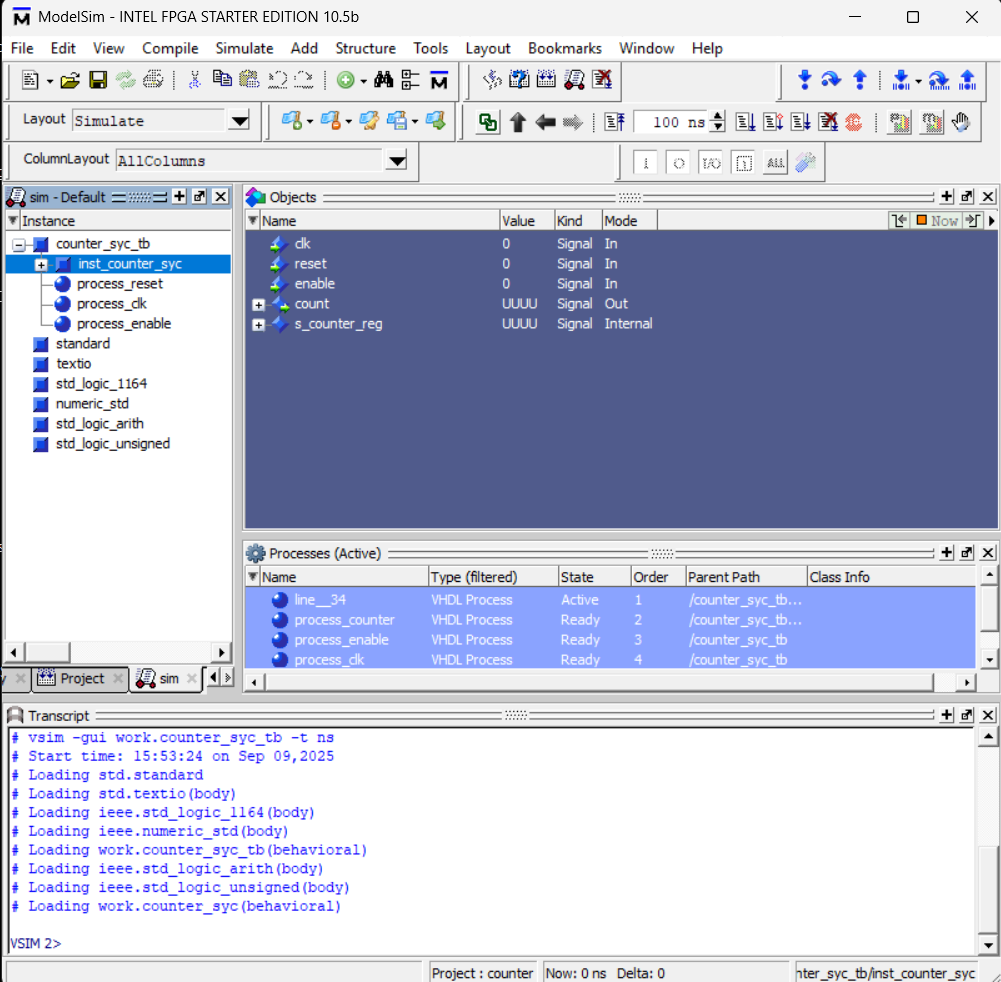
11- Em **Simulate->Start Simulation…**



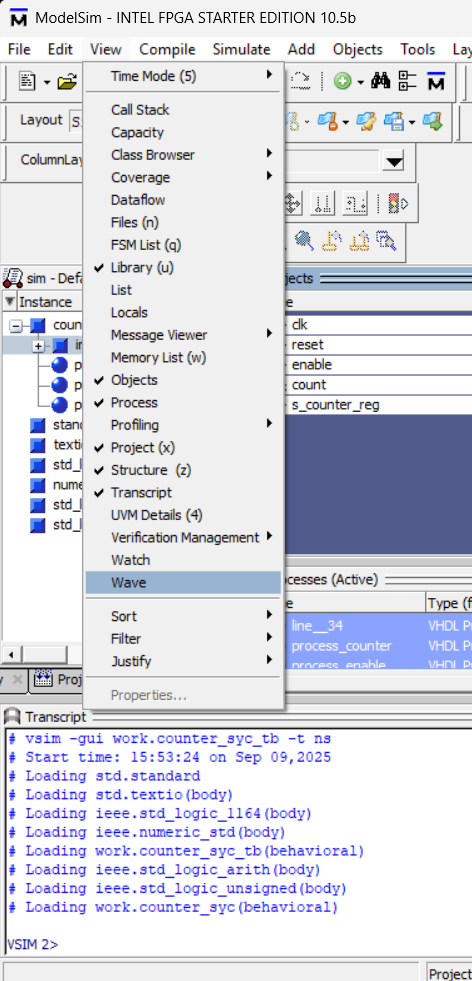
12- Selecione o arquivo counter\_sysc\_tb e resolution “ns”.



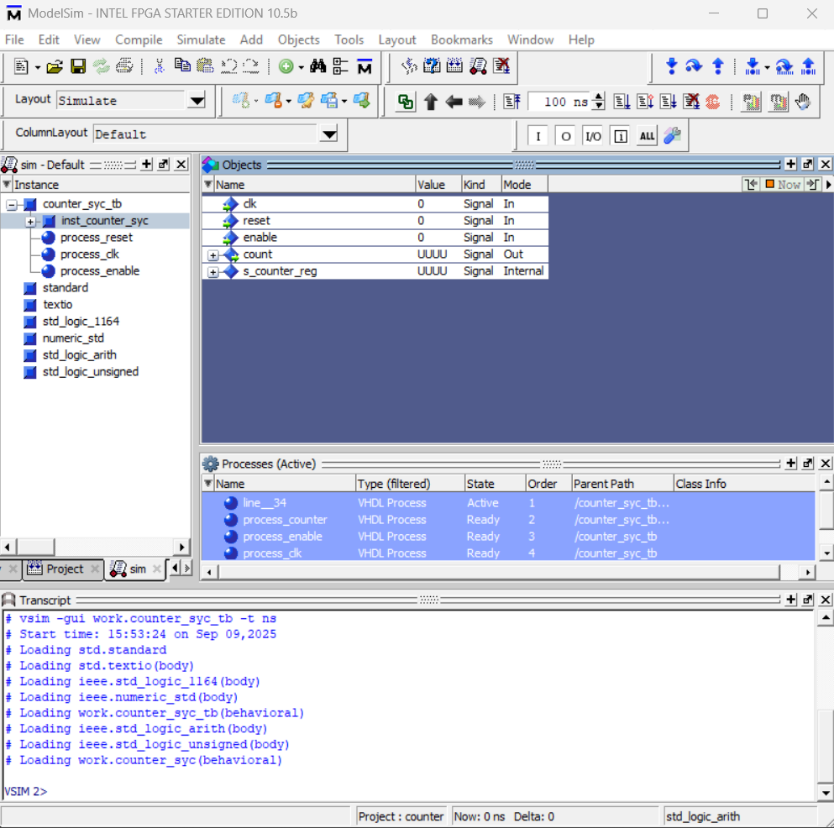
13- Aparecerá a janela abaixo. Na aba “sim-Default”, selecione “inst\_counter\_syc”.



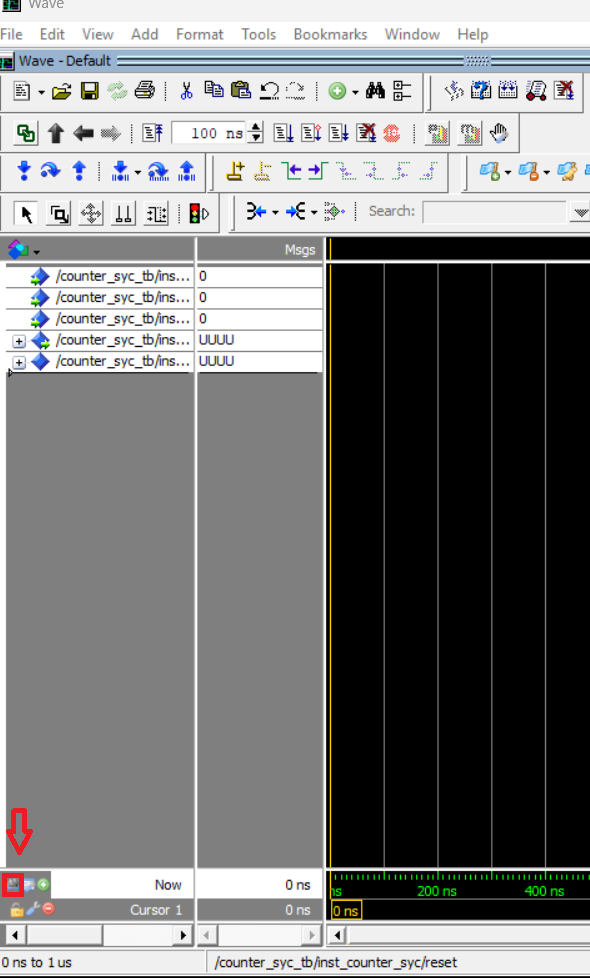
14- Selecione **View->Wave**



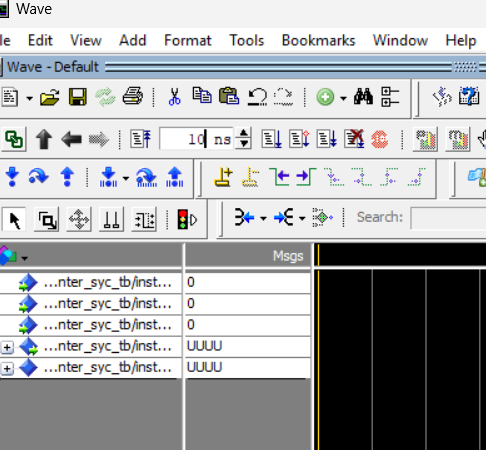
14 – Na aba **Objects** selecione todos os sinais, utilizando a tecla Ctrl.



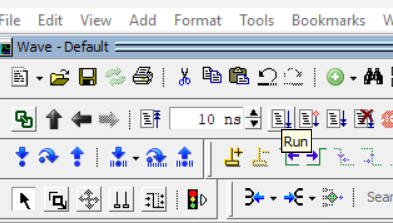
15 – Arraste os sinais para a janela ***Wave***. E clique no ícone destacado em vermelho.



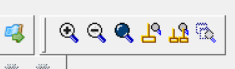
17- Mude a o passo de simulação para 10 ns.

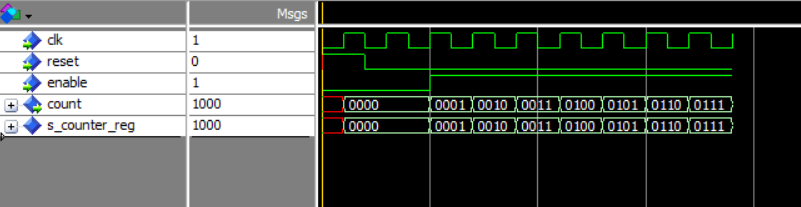


18- Clique em **Run** algumas vezes para ver os resultados da simulação.

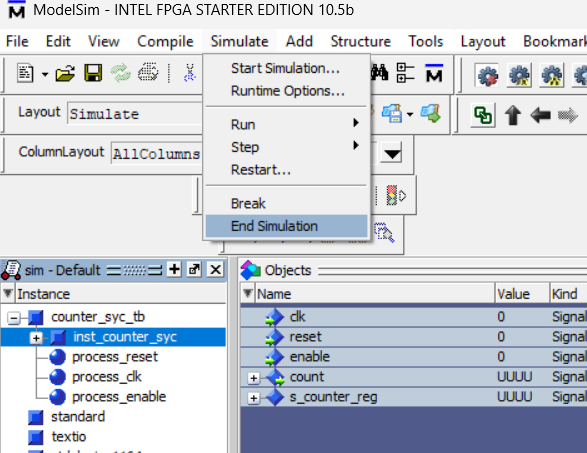


19-Aumente o Zoom para ver com mais detalhes os sinais.





20 - Para terminar a simulação feche a janela ***Wave*** e clique em **Break** e depois **End Simulation**.



**Bibliografia**

Artigos

* Farea et al, Understading Physics-Informed Neural Networks: Thechniques, Applications, Trends, and Challenges.

Livros

Haykin, Simon. Redes Neurais Princípios e prática.