***Datasheet* - Processador de 10 instruções**

1. **Arquitetura**

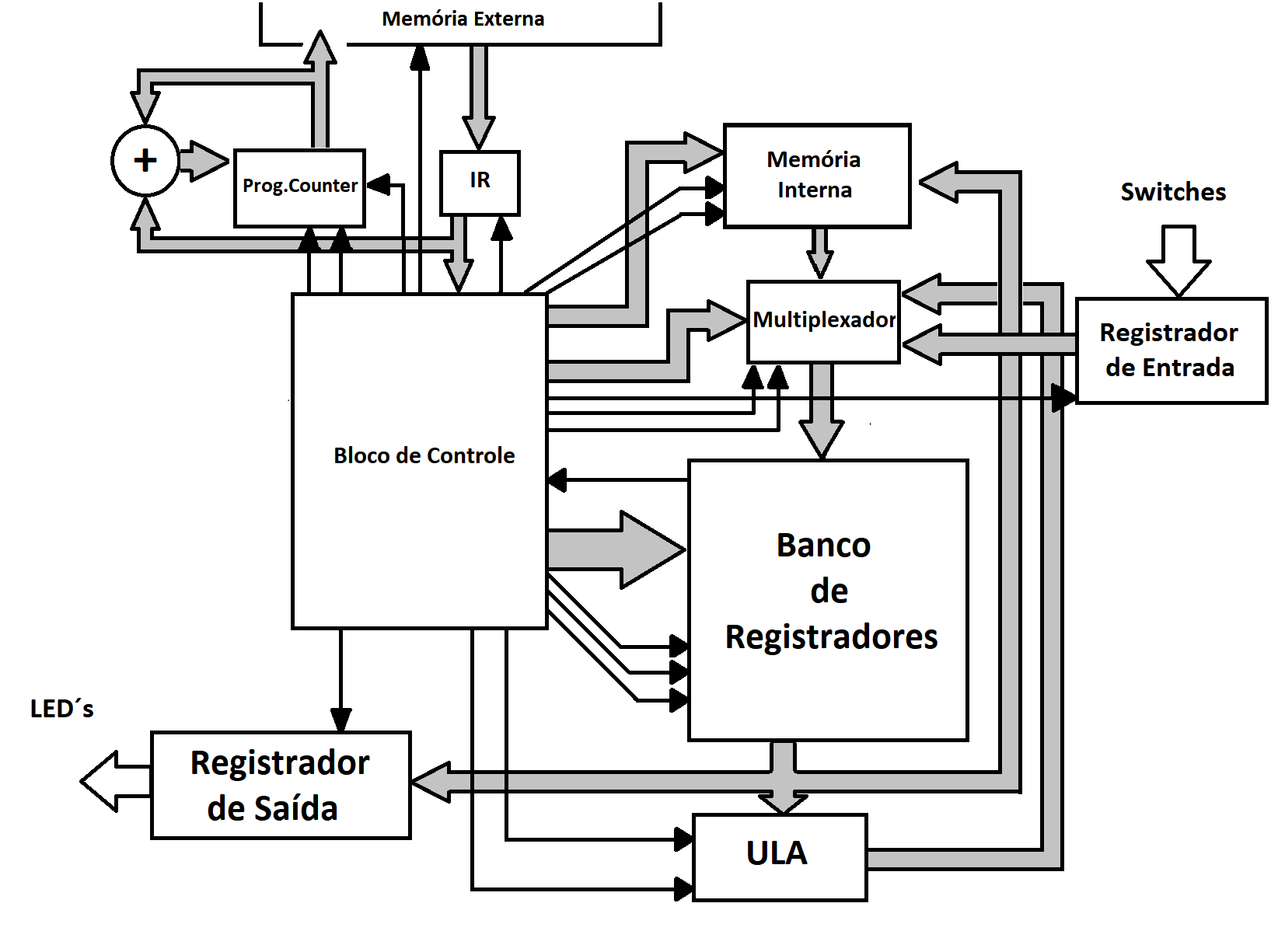
Esse processador é baseado em uma arquitetura de 16 bits e possui dois blocos principais, o bloco de controle e o bloco operacional. No bloco operacional possui um banco de 16 registradores interno com 16 bits cada e uma lógica aritmética (ULA) capaz de realizar 5 operações:

**Tabela 1** – Operações da ULA e respectivas seleções

|  |  |
| --- | --- |
| **ALU Select** | **Operação** |
| 000 | A |
| 001 | A + B |
| 010 | A - B |
| 011 | A << B |
| 100 | A >> B |

No bloco de controle, é feita a lógica sequencial para decodificação de cada instrução, bem como o endereçamento para cada uma delas. Esse controle é feito por meio de o de uma máquina de estados, a partir da qual definirá o tipo de instrução e quantos ciclos de clock a instrução irá percorrer até ser executada. Esse microprocessador conta ainda com uma memória de 512k que pode armazenar dados de saída do banco de registradores.

Para interfaces I/O, existem 2 registradores dedicados com instruções próprias, os quais podem comunicar com 16 entradas digitais e 16 saídas digitais. Esses registradores comunicam com o processador por meio da saída do Banco de registradores e pelo multiplexador anterior aos registradores, ilustrados na figura 1.



**Figura 1** – Ilustração da arquitetura do processador.

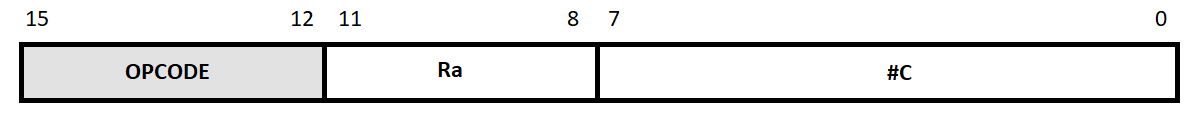
**2. Instruction Set**

O processador em questão possui arquitetura mapeada em 16 bits. Como descrito na seção anterior, essa arquitetura pode ser simplificada em 3 partes principais, o bloco operacional, a unidade de controle e a interface I/O. A unidade de controle é o bloco responsável pela leitura e interpretação das instruções, enviando sinais responsáveis pelo controle do bloco operacional. De maneira simplificada, a unidade de controle diz quais são os blocos do processador necessários e quais dados serão manipulados para realizar a instrução solicitada. Dessa forma, após a leitura da memória externa ao processador, cada instrução será lida e os dados contidos nela serão desmembrados para realizar uma operação. O conjunto de instruções é formado por 10 instruções básicas, as quais são identificadas pela tabela 1.

**Tabela 2** – Conjunto de Instruções do Processador TOP.

|  |  |  |
| --- | --- | --- |
| **Instrução** | **Significado** | **Opcode** |
| **MOV**Ra,d | R[a] <= D[d] | 0000 |
| **MOV**d,Ra | D[d] <= R[a] | 0001 |
| **ADD** Ra, Rb, Rc | R[a] <=R[b] + R[c] | 0010 |
| **SUB**Ra,Rb,Rc | R[a] <= R[b] - R[c] | 0100 |
| **SR**Ra,Rb,Rc | R[a] <=R[b]>>R[c] | 0111 |
| **SL**Ra,Rb,Rc | R[a] <=R[b]<<R[c] | 0110 |
| **OUT** Ra | Ro<= R[a] | 1001 |
| **IN** Ra | R[a] <= Ri | 1000 |
| **MOV** #c,Ra | R[a] = #C | 0011 |
| **JUMPZ** R3, #c | PC <= PC + #C **if(R3 == 0)** | 0101 |

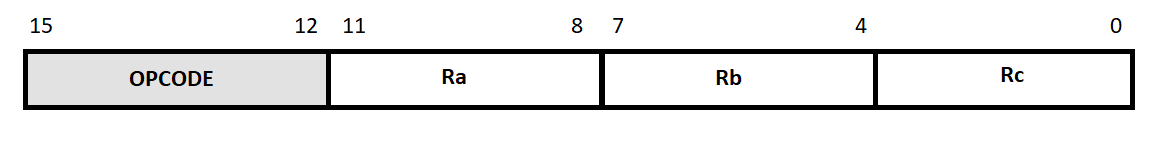
Cada instrução é mapeada de acordo com o número endereços ou constantes numéricas envolvidas na operação em questão, os quais podem variar de 1 a 3 endereços diferentes, exceção feita aos quatro bits iniciais, reservados para o código da instrução (Opcode). Para as operações envolvendo apenas um endereço e uma constante (0011 e 0101), oito bits são reservados para a constante, como ilustrado pela figura 1.



**Figura 1** – Instruções com apenas um endereço e uma constante. “#C” é a constante e “Ra” é o endereço do registrador.

Já a divisão dos bits para as instruções envolvendo 2 endereços , um referente a um registrador e outro referente à memória, é semelhante ao mostrado na figura 1, com a mudança nos oito bits menos significativos, os quais serão destinados ao endereço da memória.

Para as operações que envolvem o endereçamento de três registradores, a divisão de bits difere das duas configurações anteriores, visto que agora devemos alocar um endereço a mais. A figura 2 ilustra a configuração dos bits de instrução para esse caso.

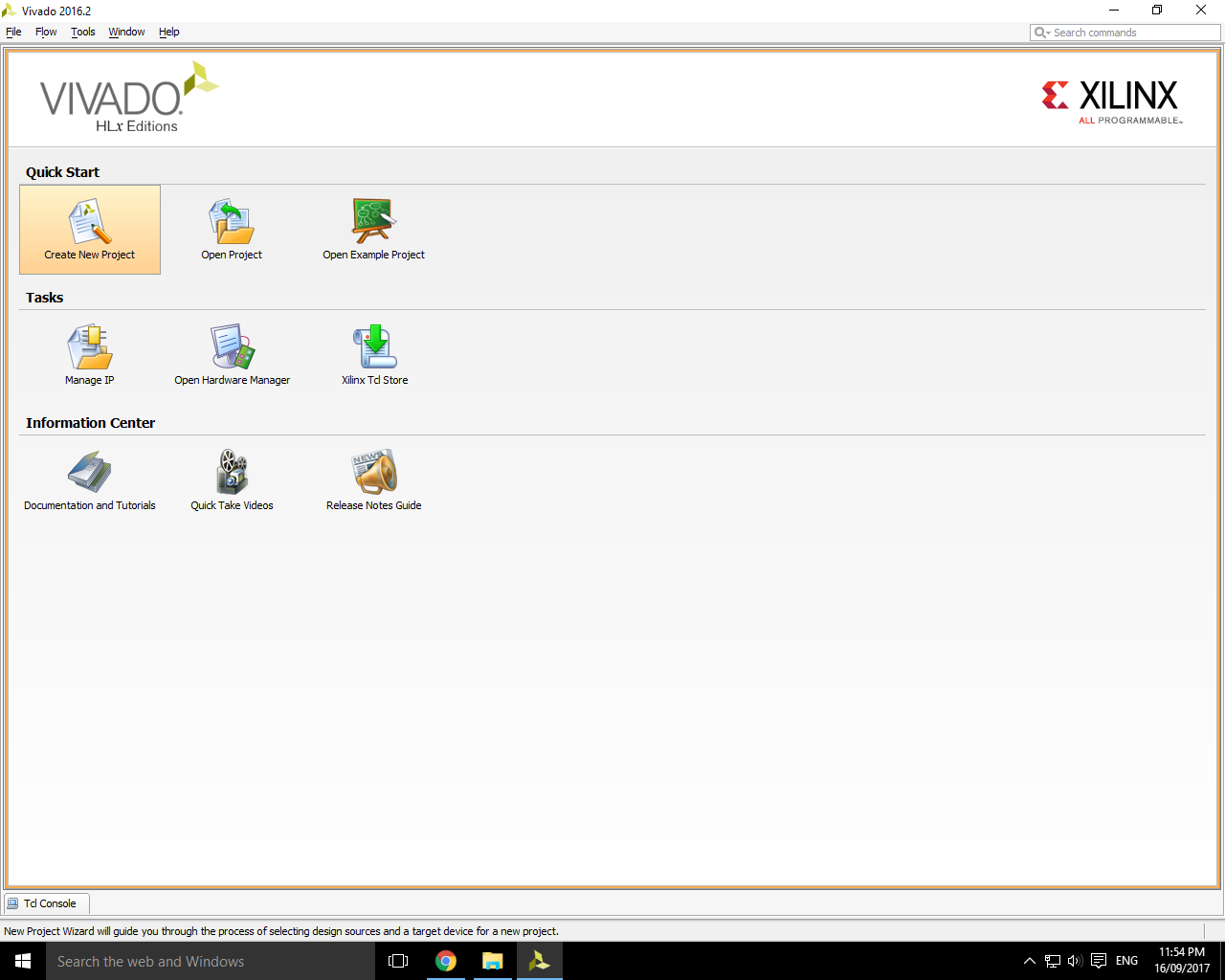


**Figura 2** – Configuração de bits para instruções com três endereçamentos. “Ra”, “Rb”,”Rc” são endereços de registradores.

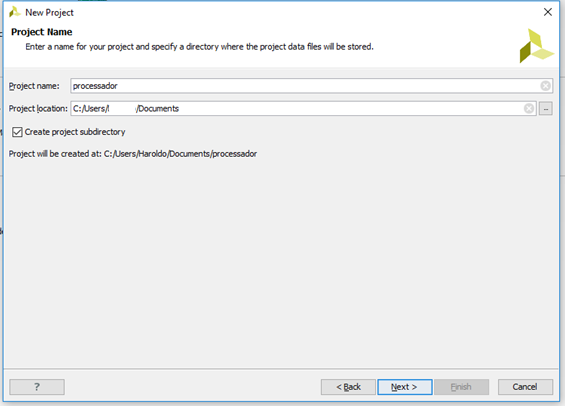
Vale ressaltar que as operações que envolvem endereços de registradores referem-se somente ao banco de registradores presente no bloco operacional e as que envolvem endereços de memória também referem-se à memória presente no bloco operacional.

**Instanciando o processador:**

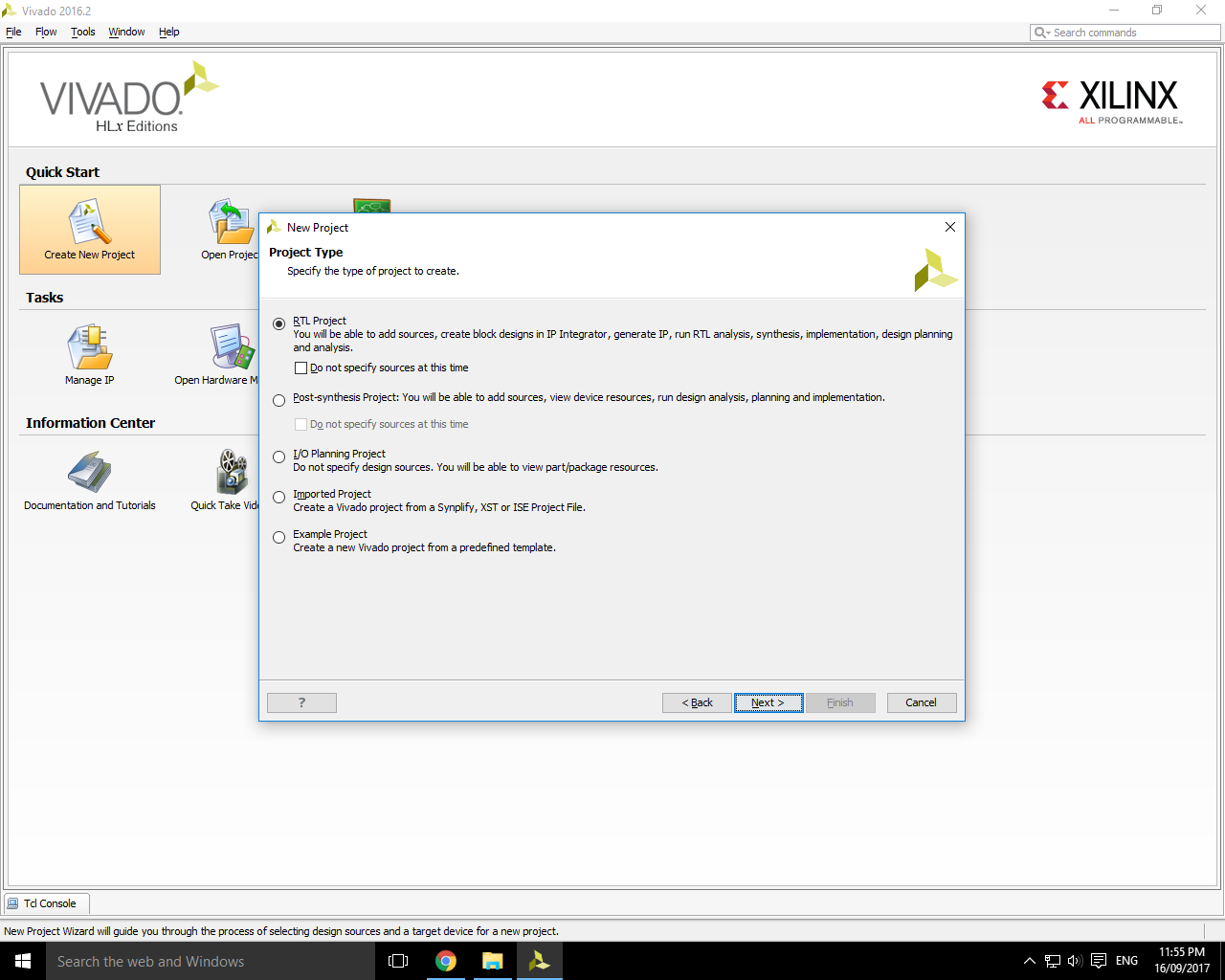
1 – Na tela inicial do Vivado clique em *Create New Project*



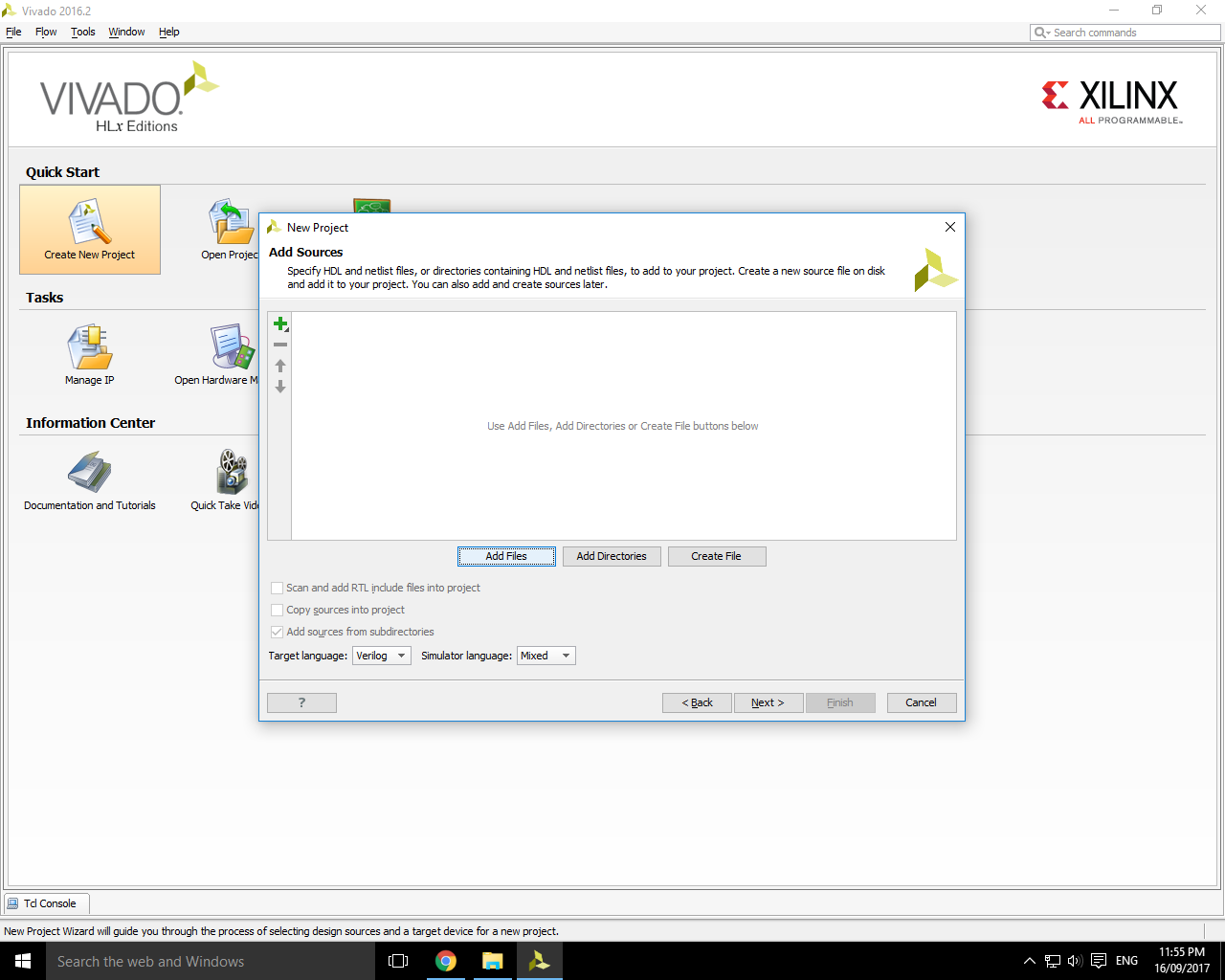
2 – Nomeie o projeto e clique em *Next*



3 – Selecione *RTL Project* e deixe a caixa de seleção *Do notspecifysourcesatthis time* desmarcada, após isso clique em *Next*

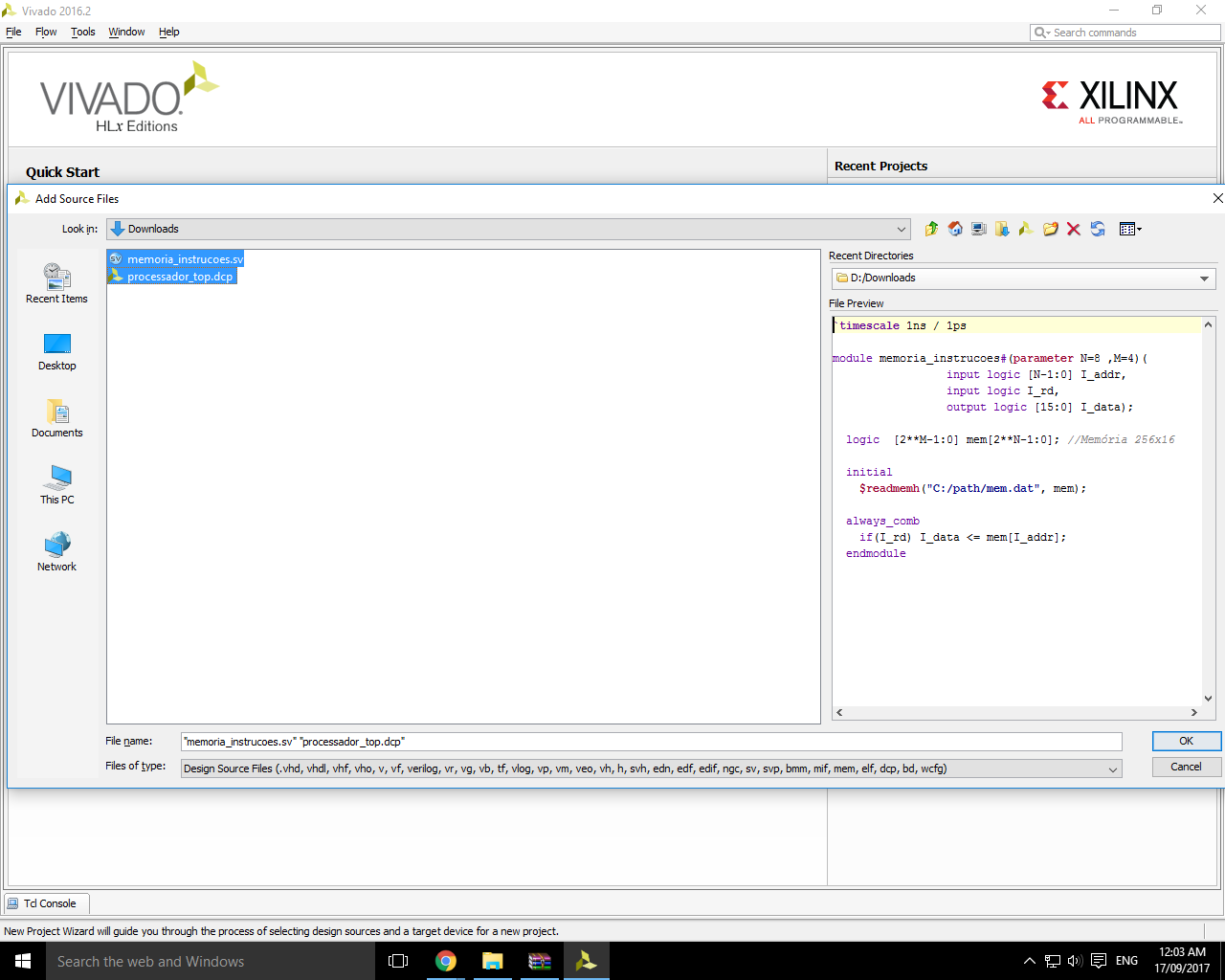


4 – Na janela *AddSources* clique em *Add Files*

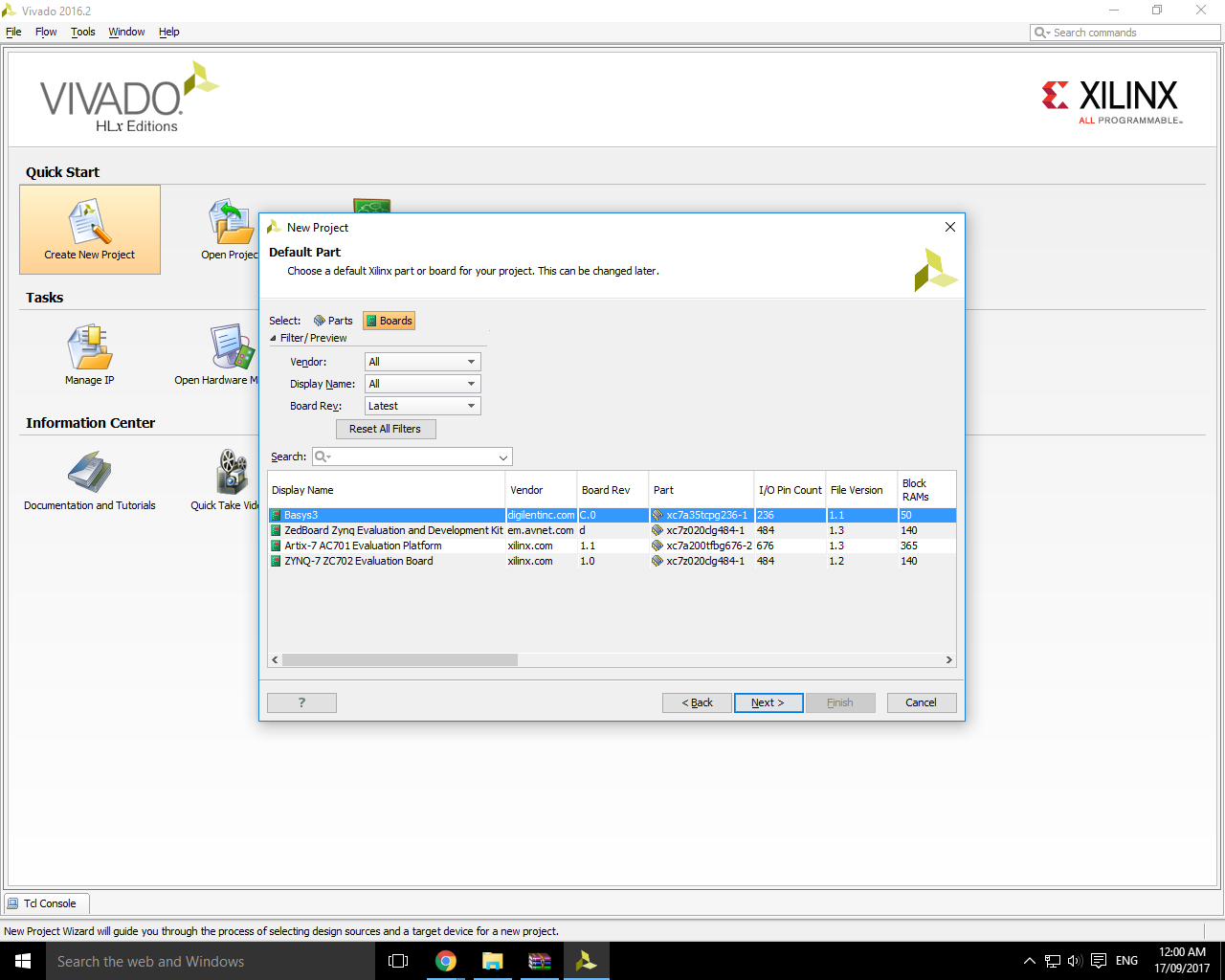


5 – Na janela aberta selecione os arquivos disponibilizados “*memoria\_instrucoes.sv*” e “*processador\_top.dcp*”, clique em *Ok* e depois em *Next.* Clique em *Next* também nas janelas *Addexisting IP (optional)* e *Addconstraints (optional)*

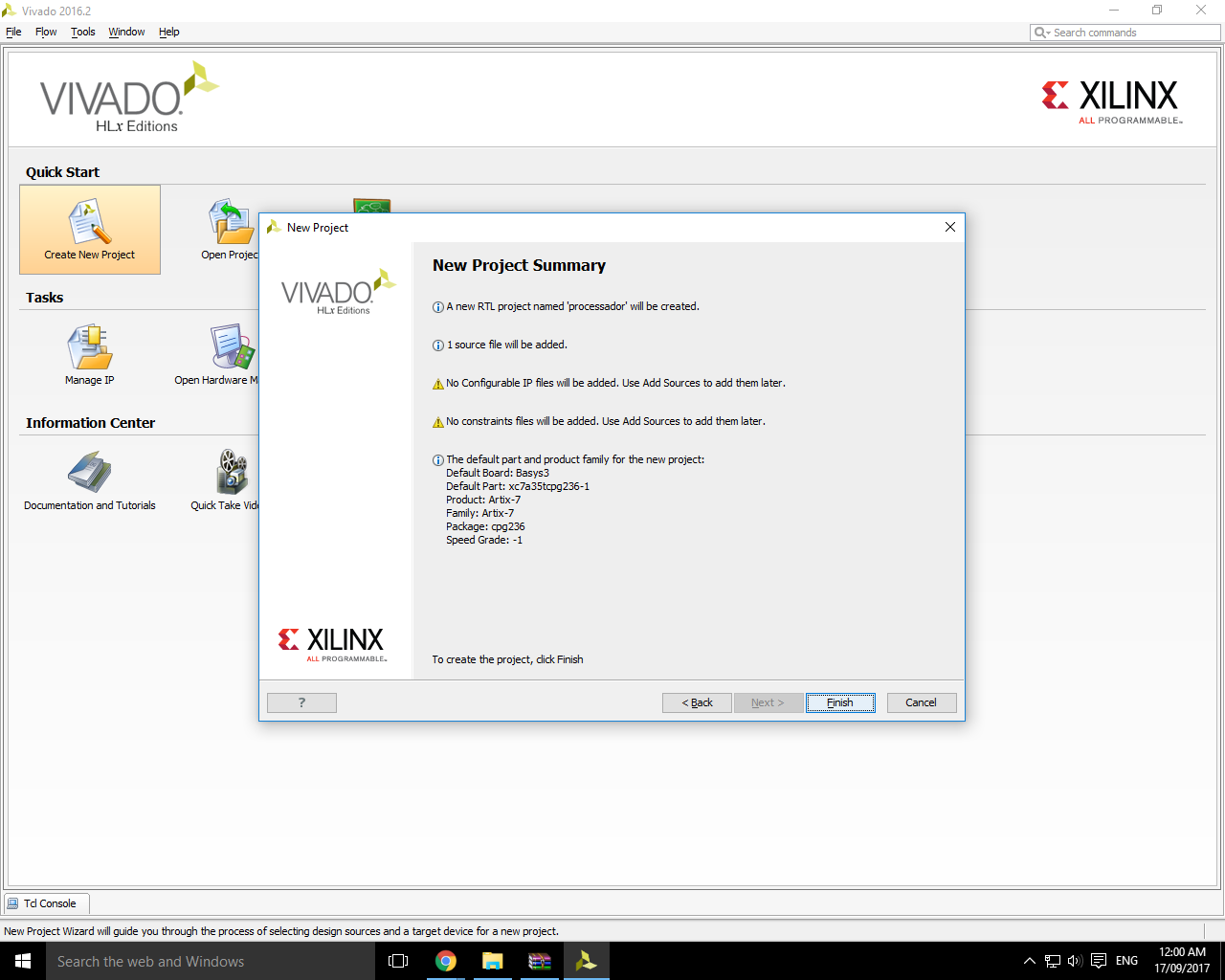
Após a criação do projeto o caminho do arquivo “*mem.dat*” da memória de instruções deve ser alterado



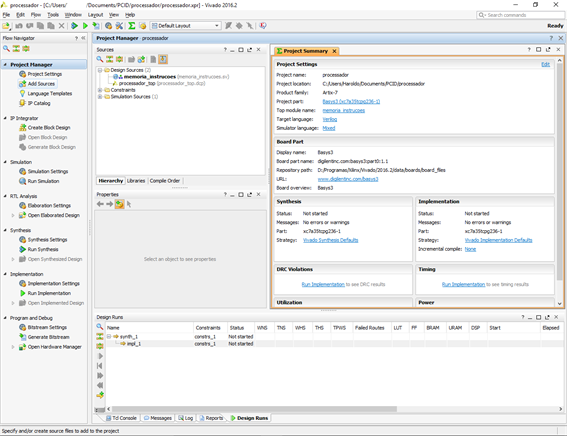
6 – Na janela *Default Part* selecione *Boards*e a seguir *Basys3* clicando em *Next* na sequência(Caso não tenha a placa listada adicione seus arquivos seguindo os passos encontrados em <https://reference.digilentinc.com/reference/software/vivado/board-files>)



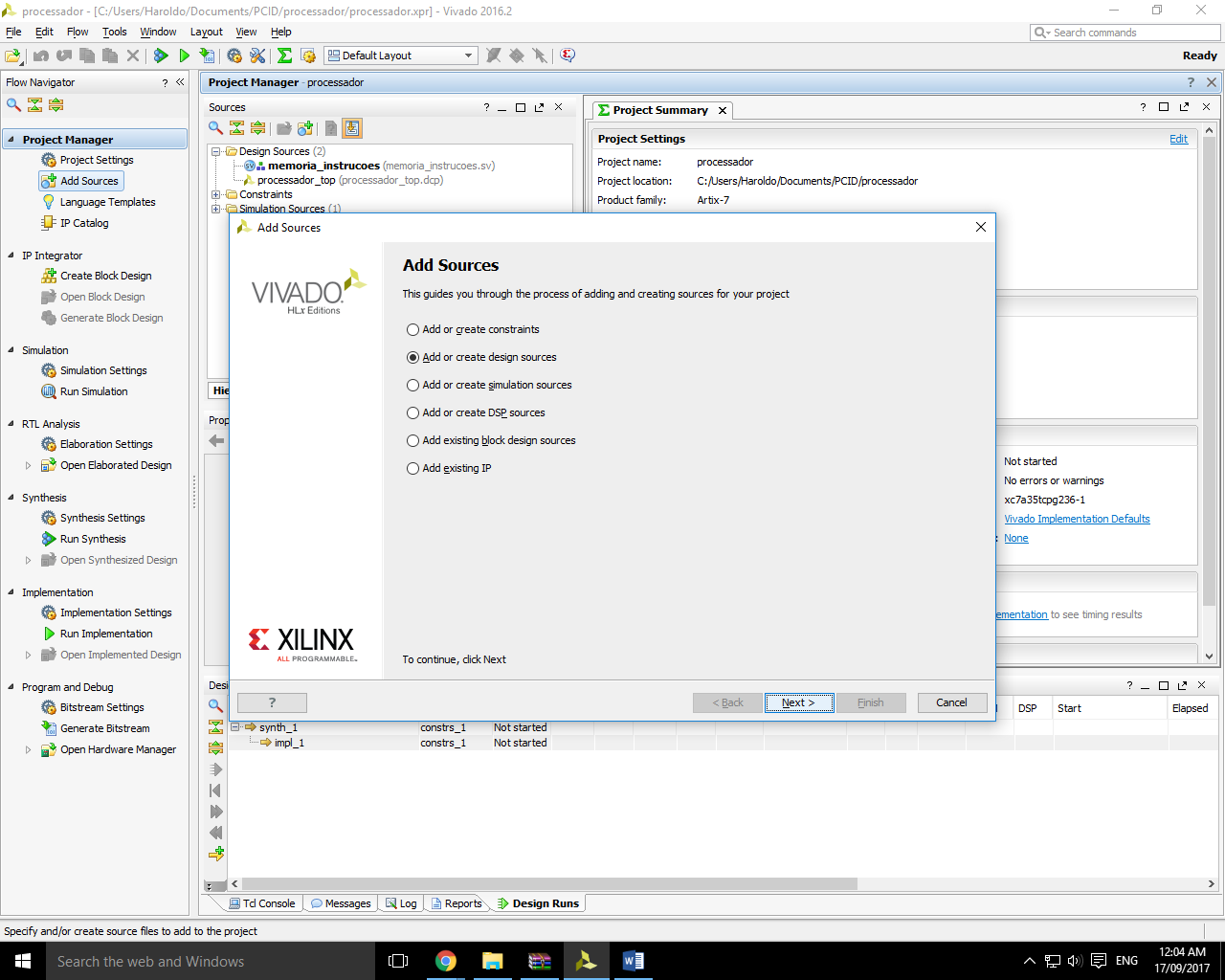
7 – Clique em *Finish*, o projeto então estará criado



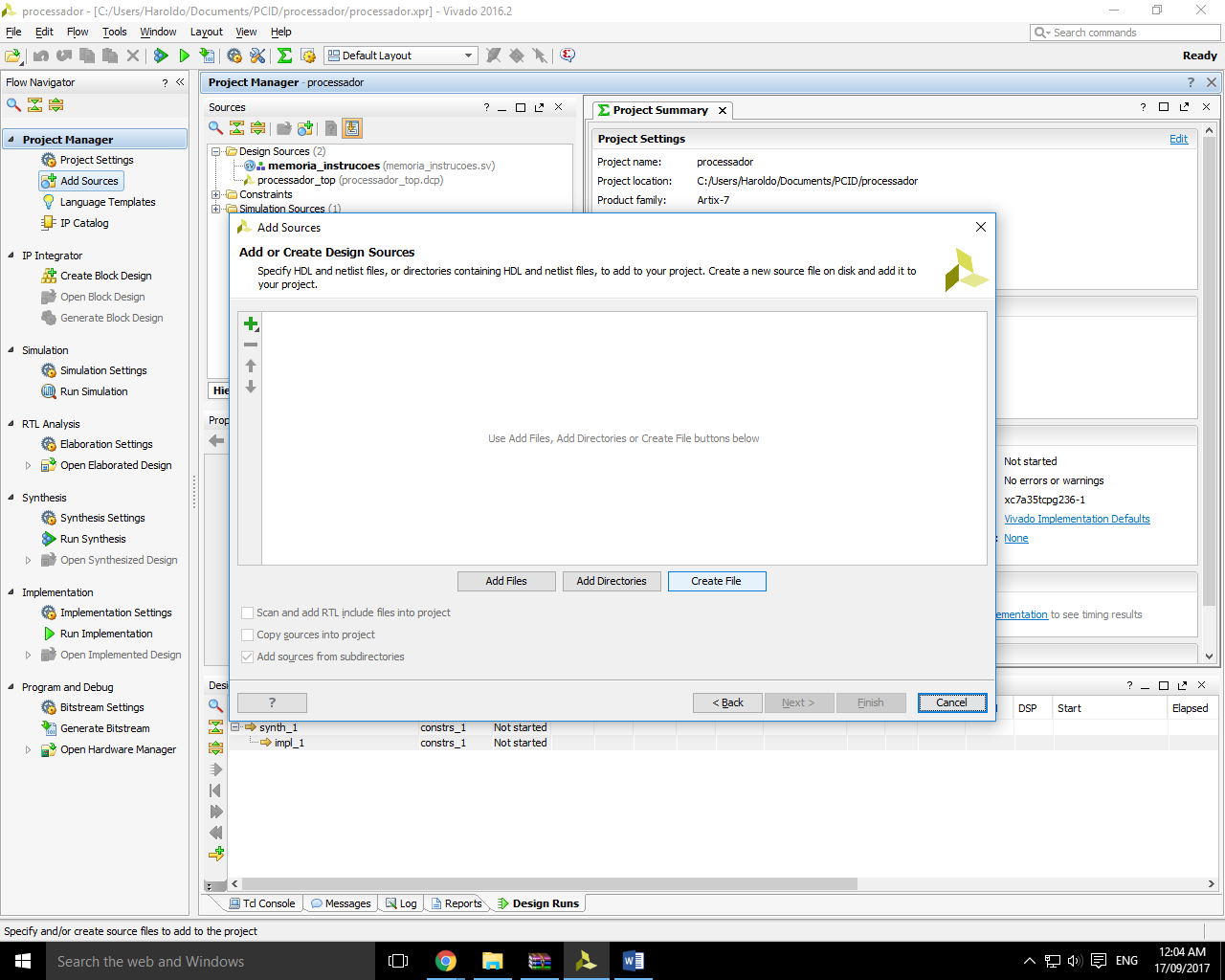
8 – Para criar o arquivo de *top\_level*, na área *Project Manager* clique no botão *AddSouces*



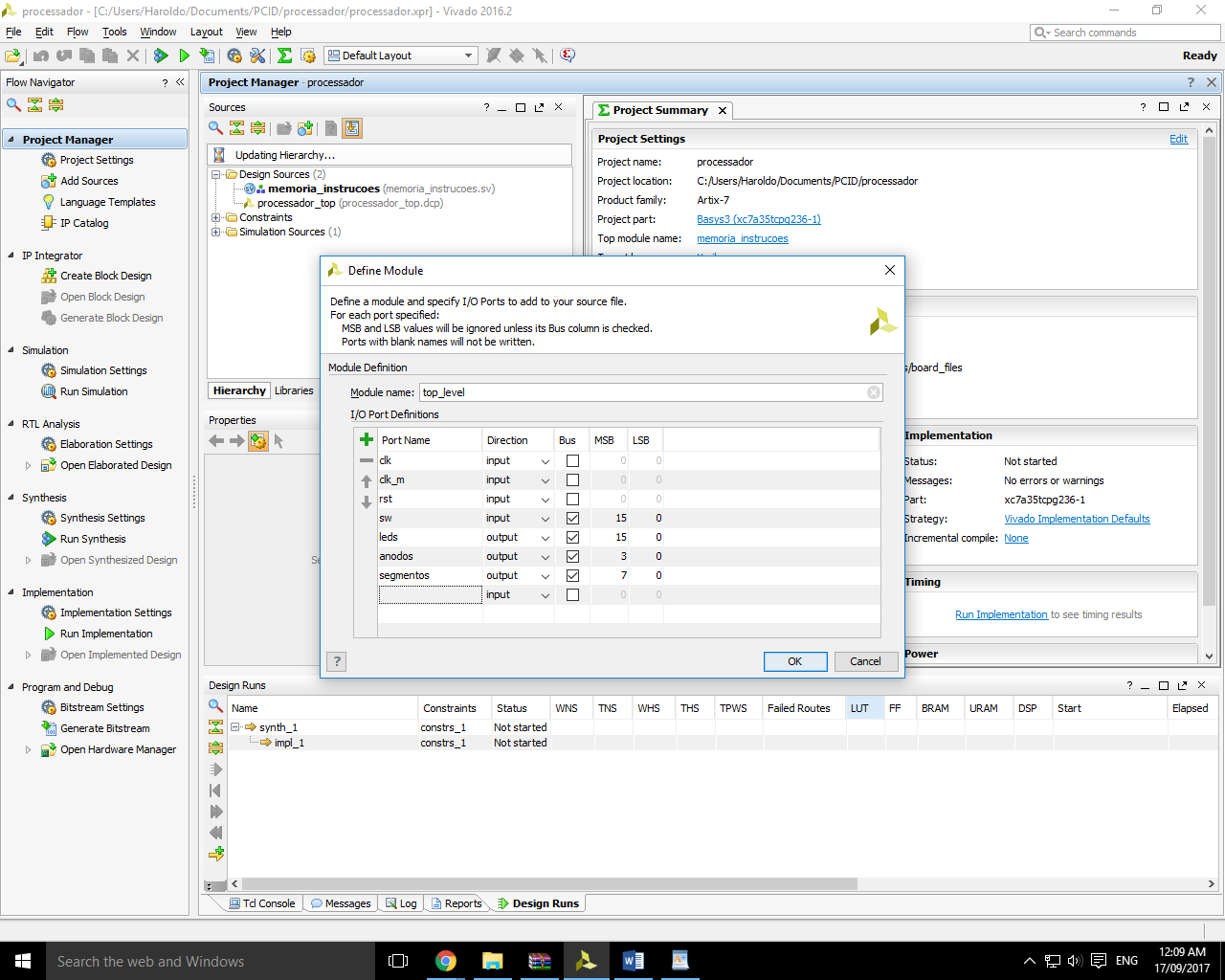
9 – Na janela*Add Sources* selecione*Add or create design souces* e em*Next*



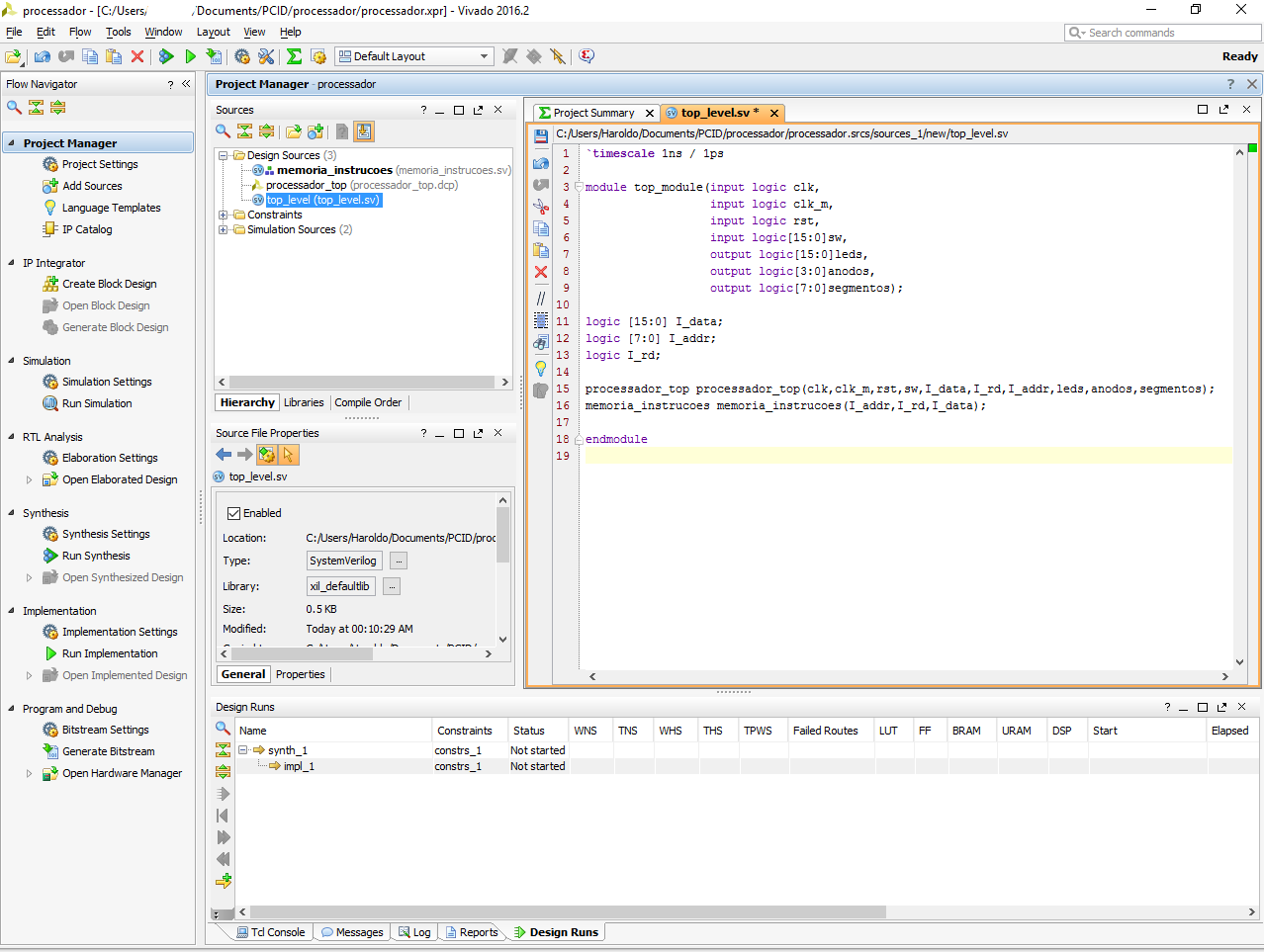
10 – Clique em *Create File* e nomeie o arquivo de *top\_level*



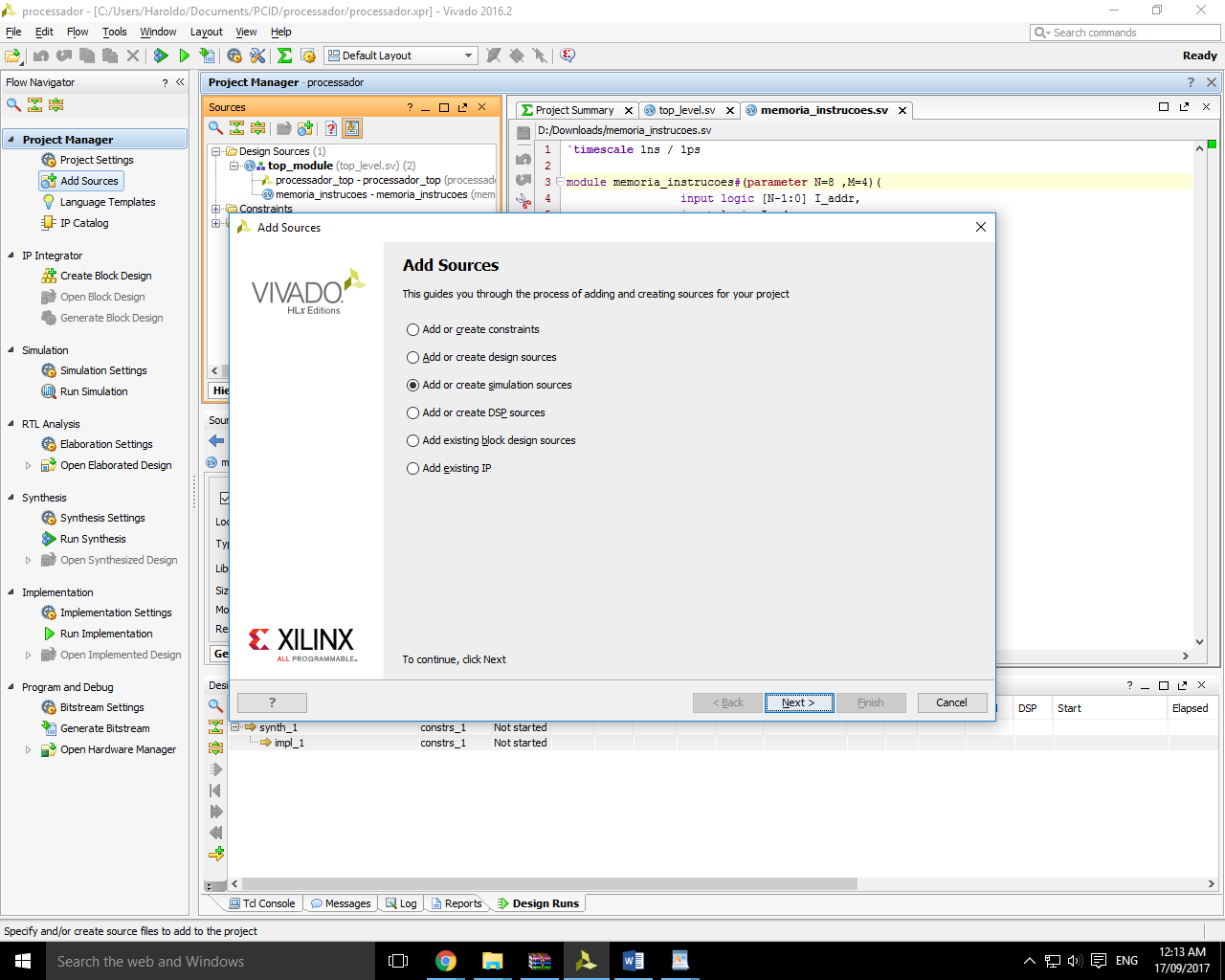
11 – Crie os seguintes sinais de *I/O* e em *Ok*



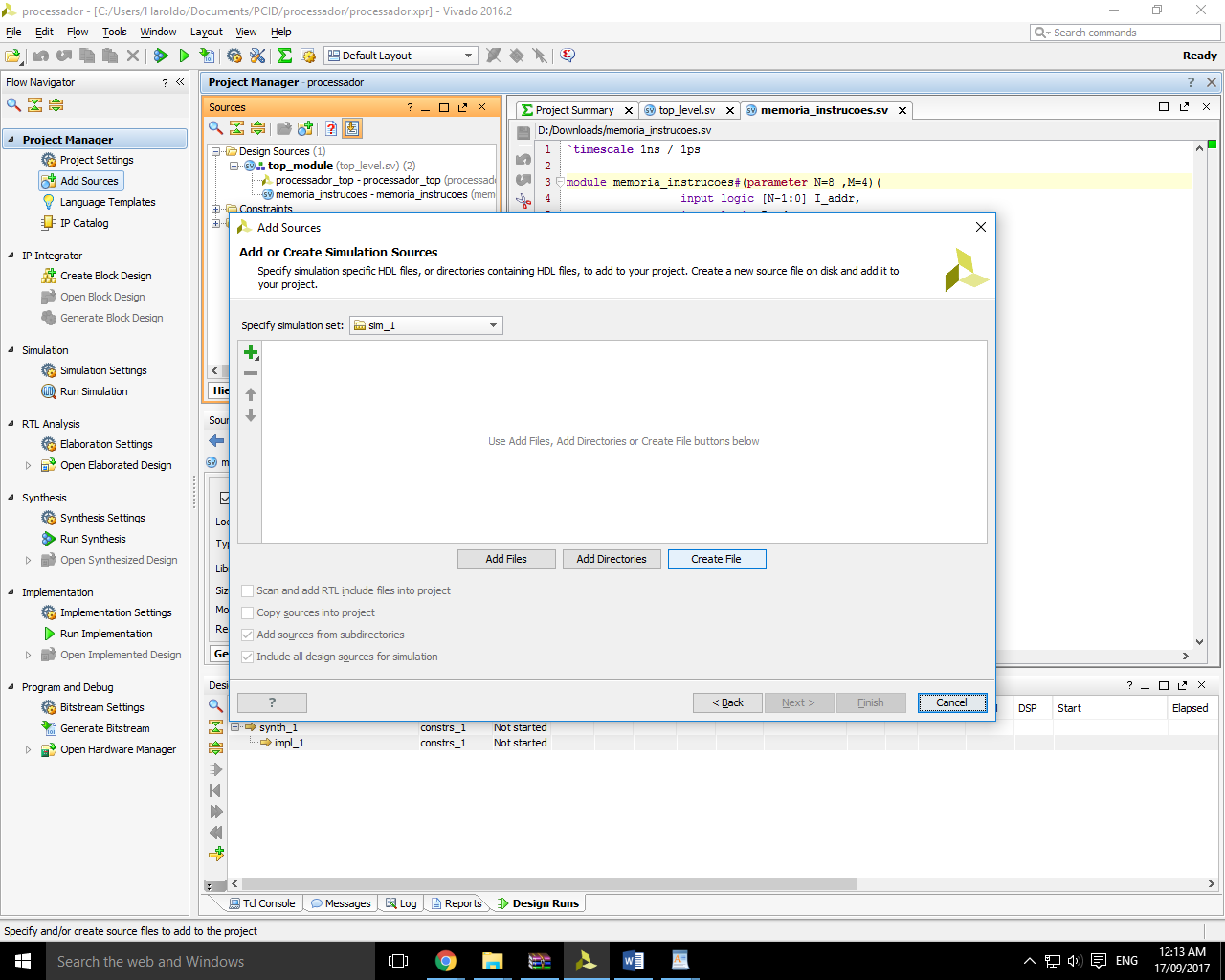
12 – Crie o *top\_level*, instanciando a memória de instruções e o processador que deve ser instanciado como *processador\_top(clk,clk\_m,rst,sw,I\_data,I\_rd,I\_addr,leds,anodos,segmentos)*



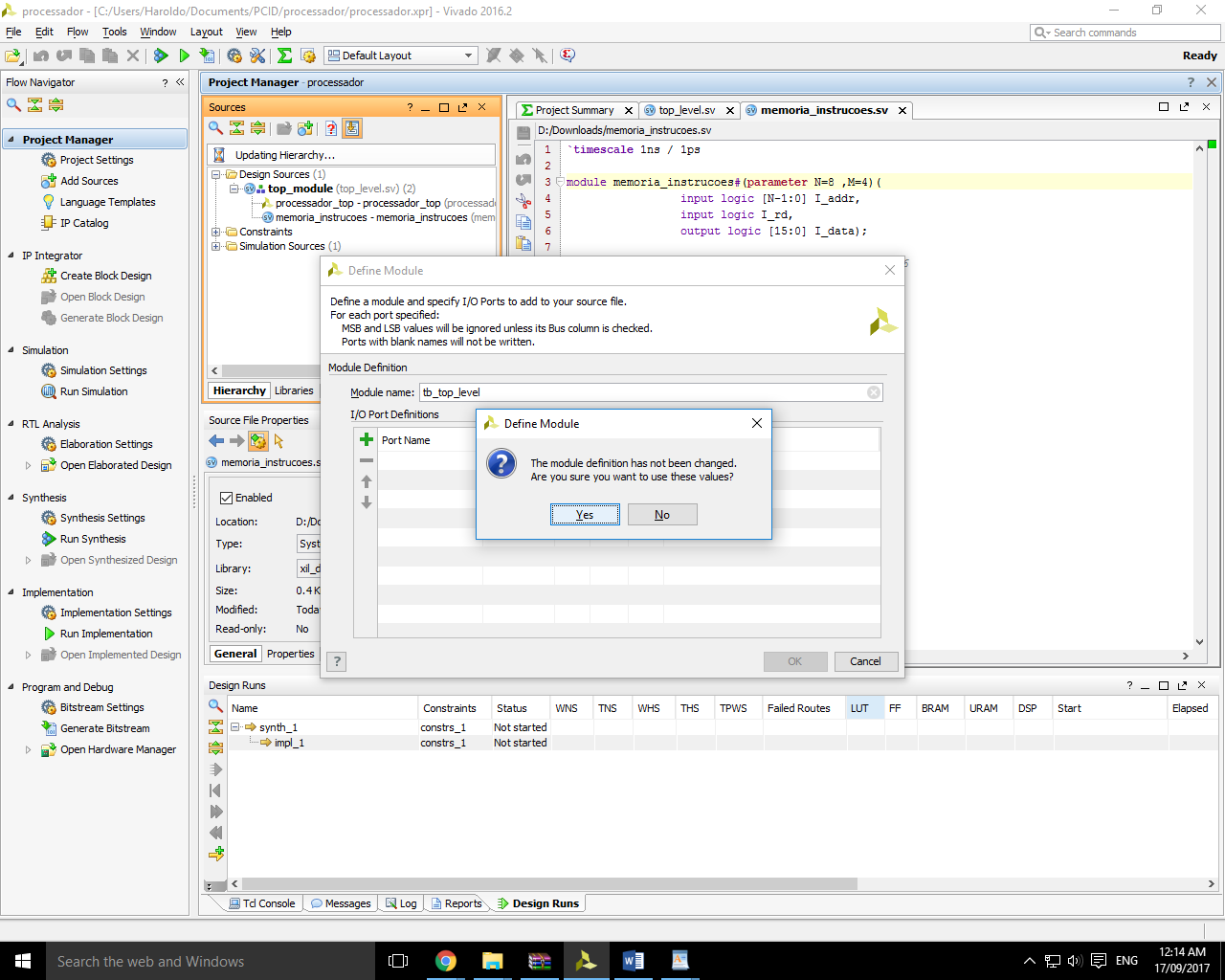
13 – Para criar o arquivo de simulação, na área *Project Manager* clique no botão *AddSouces*e selecione *Addorcreatesimulationsources*



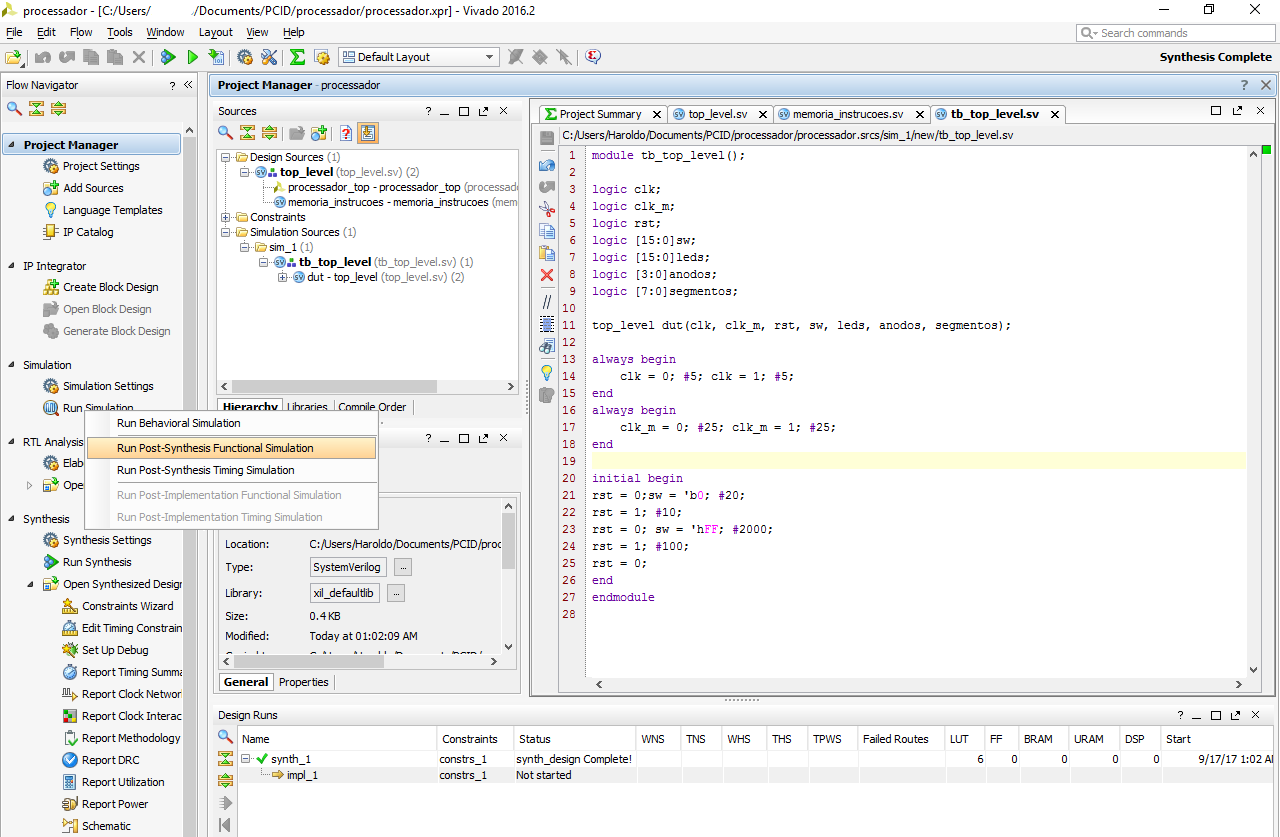
14 – Clique em *Create file*



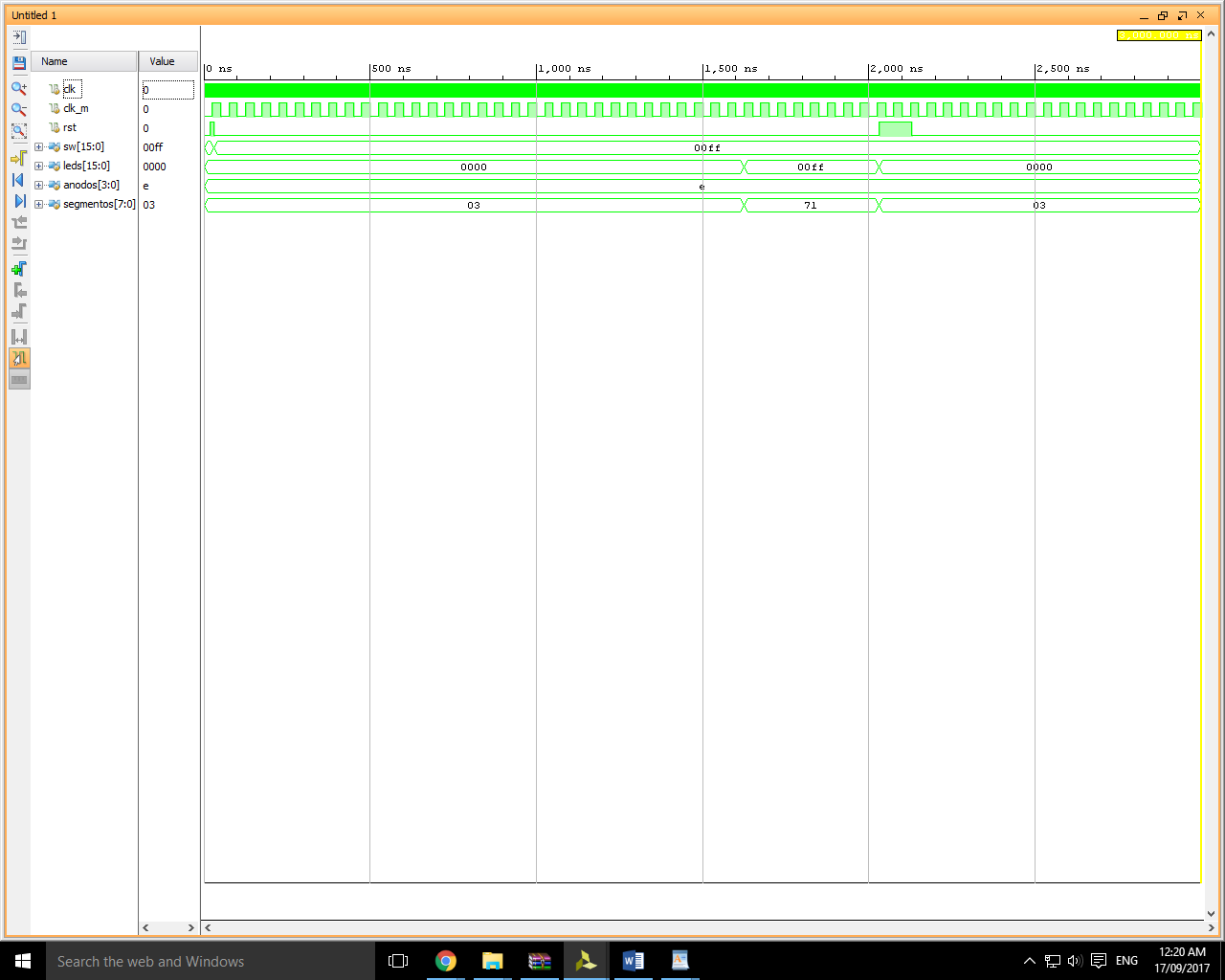
15 – Nomeie o arquivo de simulação, clique em *Next* e deixe sem especificação de *I/O*, clicando em *Yes* na janela que abrir



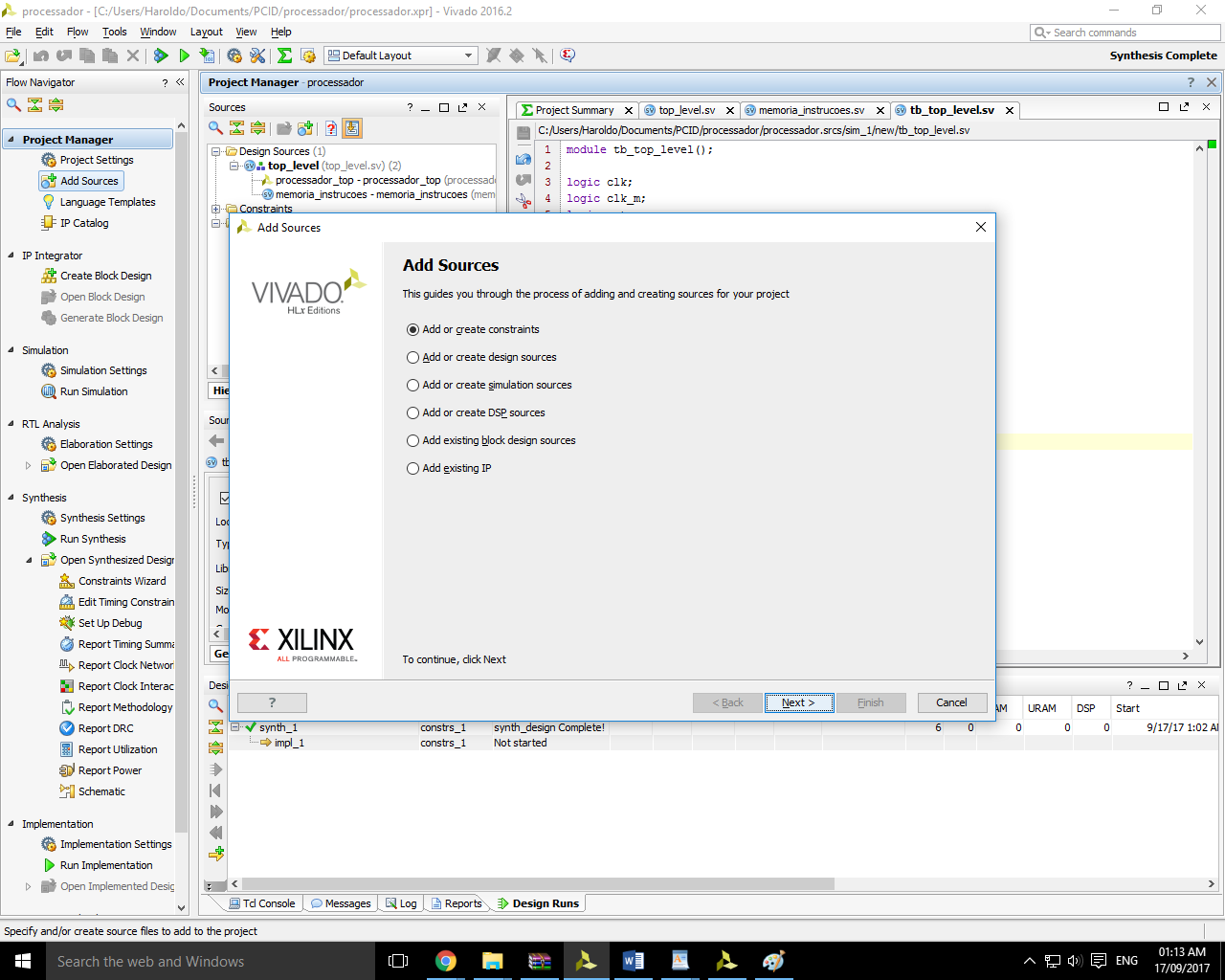
16 – Após a adição dos estímulos de simulação, na área *Synthesis* clique em *RunSynthesis*. Após a execução da síntese clique em *RunSimulation*na área *Simulation*, e então em *Run Post-Synthesis*FunctionalSimulation



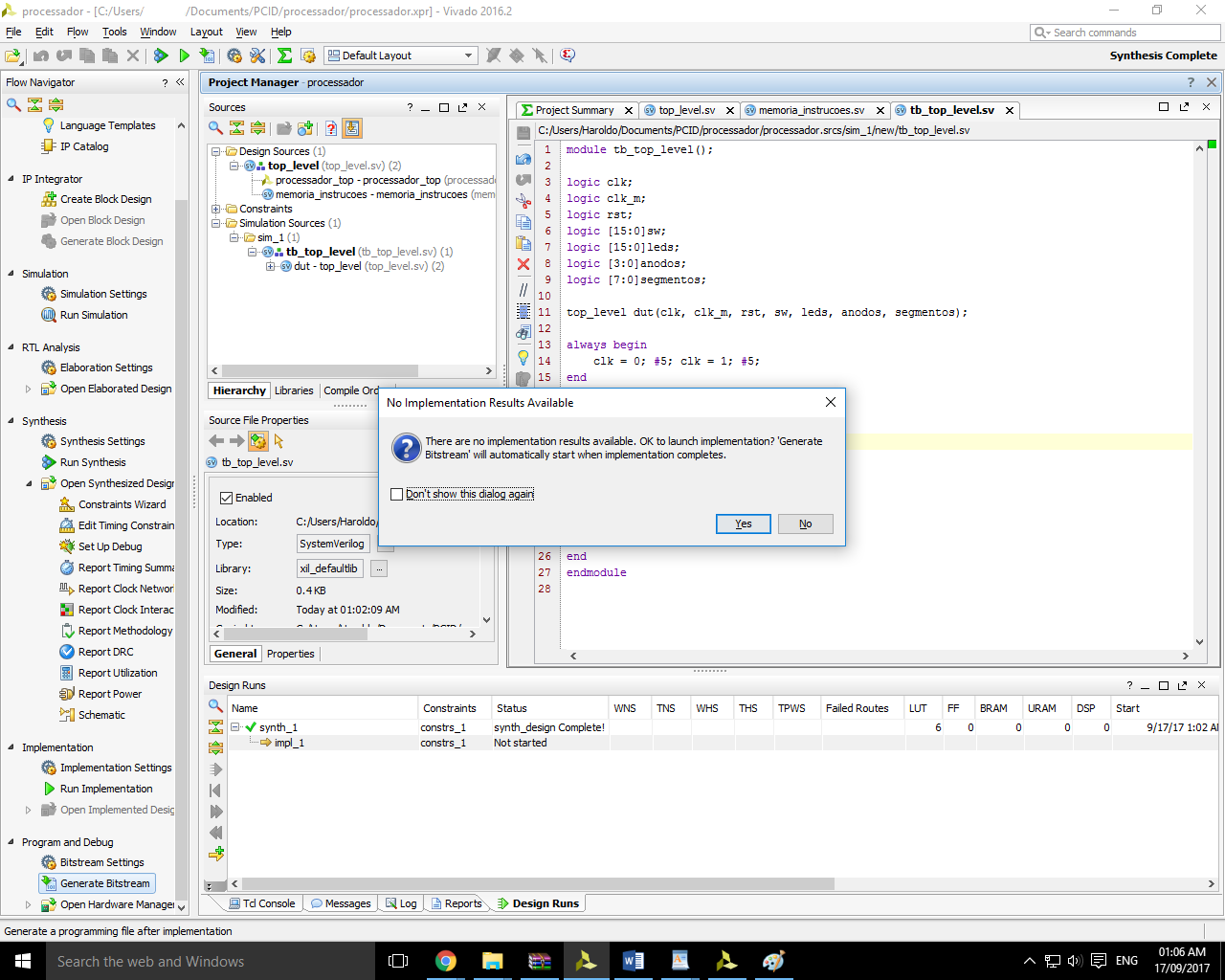
17 – A seguinte janela abrirá ao término da simulação, validando-se o correto funcionamento do circuito



18 – Crie ou adicione o arquivo .*xdc*com o mapeamento das portas *I/O* clicando em *AddSources*na área *Project Manager*, selecionando *Addorcreateconstraints*



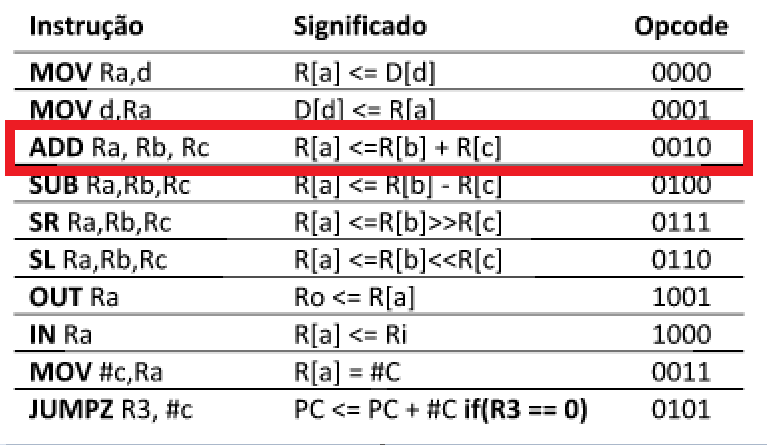
19 – Para carregar o circuito na placa clique em *GenerateBitstream*, na área *Programand Debug*, clicando em *Ok*, caso a seguinte janela apareça, para se realizar a implementação



**Escrevendo o código**

Para escrever o código basta escolher a operação e os registradores e converter o código binário gerado para hexadecimal.

**Exemplo:**



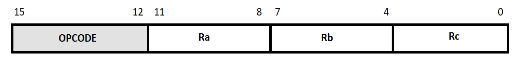
Operação: soma , valor binário 0010,valor hexadecimal 2

Registrador A: valor binário 0111, valor hexadecimal 7

Registrador B: valor binário 1111, valor hexadecimal F

Registrador C: valor binário 0000, valor hexadecimal 0

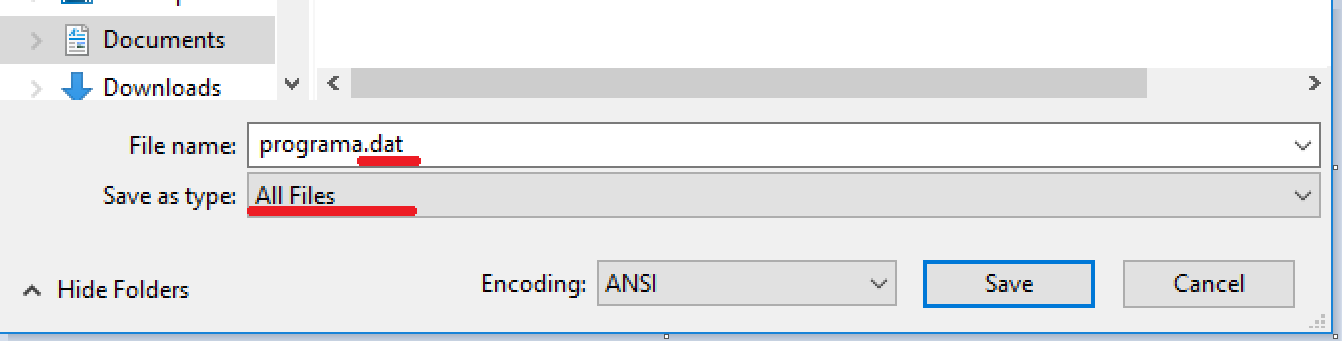
Neste caso basta colocar os valores na seguinte ordem:



0010011111110000 =>27F0

Então a linha de código 27F0 irá somar o valor dos registradores 15 e 0 e guardar no registrador 7.

Escrever o código no bloco de notas e salvar no formato .dat:



O arquivo .dat deve ser salvo no diretório definido no arquivo “*memoria\_instrucoes.sv*”.

**Notas:**

* Cada instrução precisa de três ciclos de clock , o que corresponde a pressionar o pushbutton(reservado para o clock) trêsvezes.