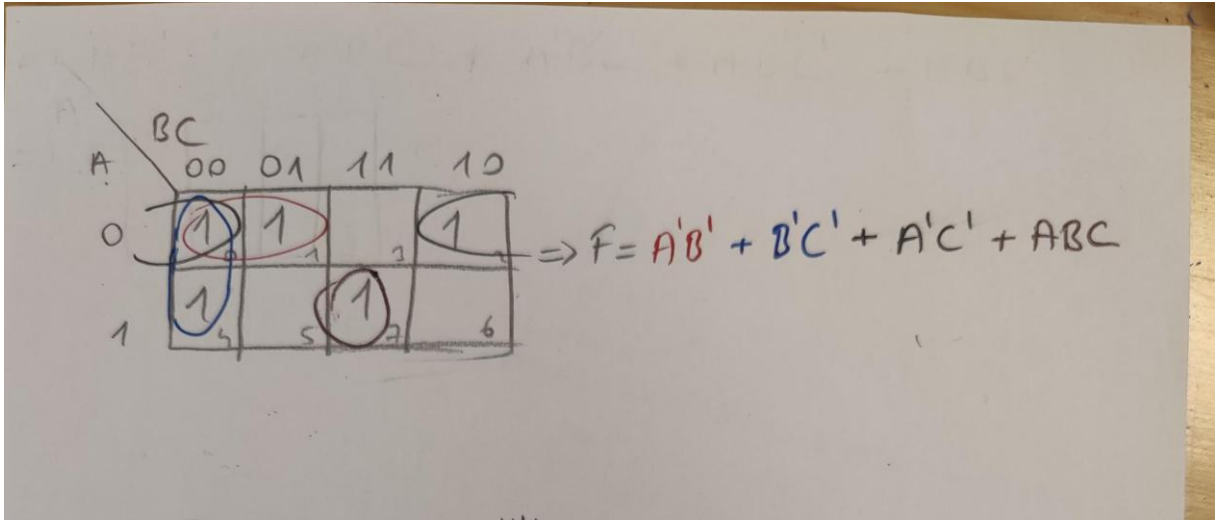


Mantık Kapılarıyla Boolean Fonksiyonları

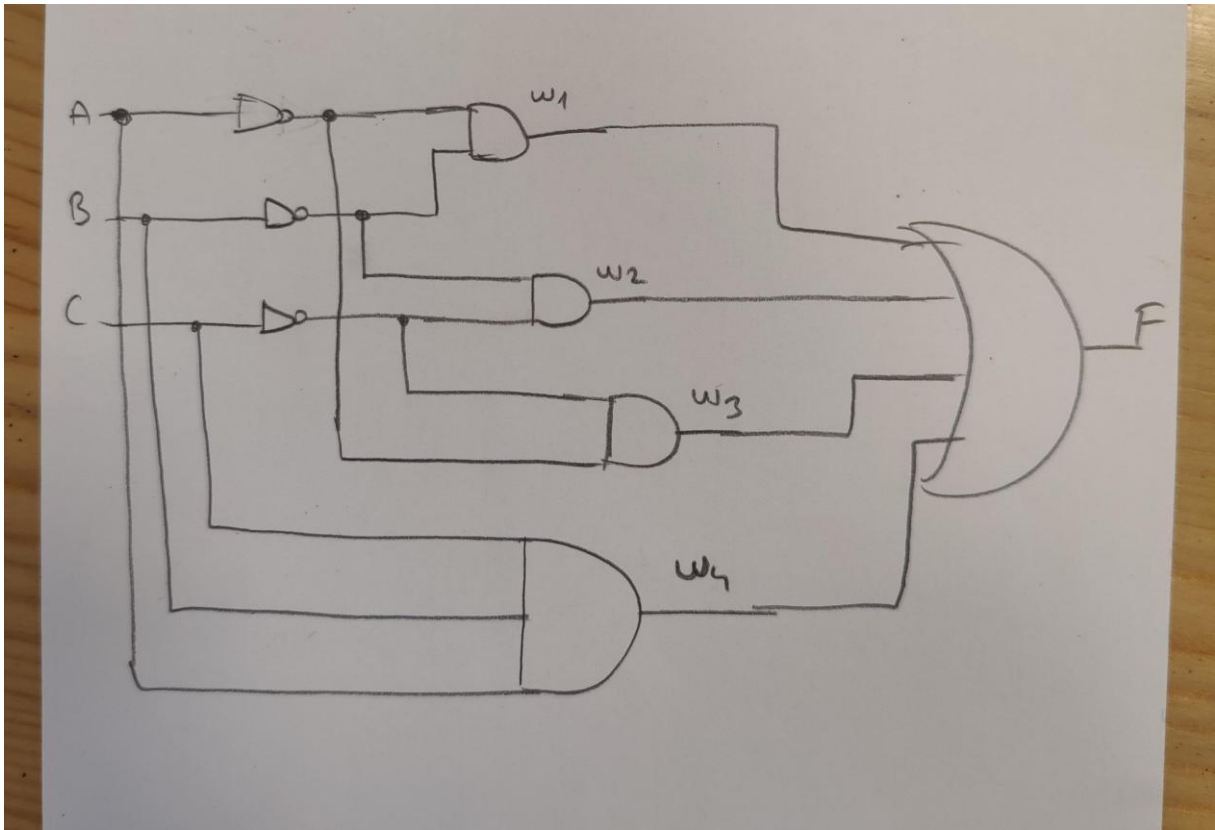
Aşağıda verilen örnekler, Vivado programı üzerinden System Verilog dili yardımıyla logic devreleri tasarlanarak simüle edilecektir.

1. ÖRNEK: $F(A,B,C) = \sum m(0,1,2,4,7)$

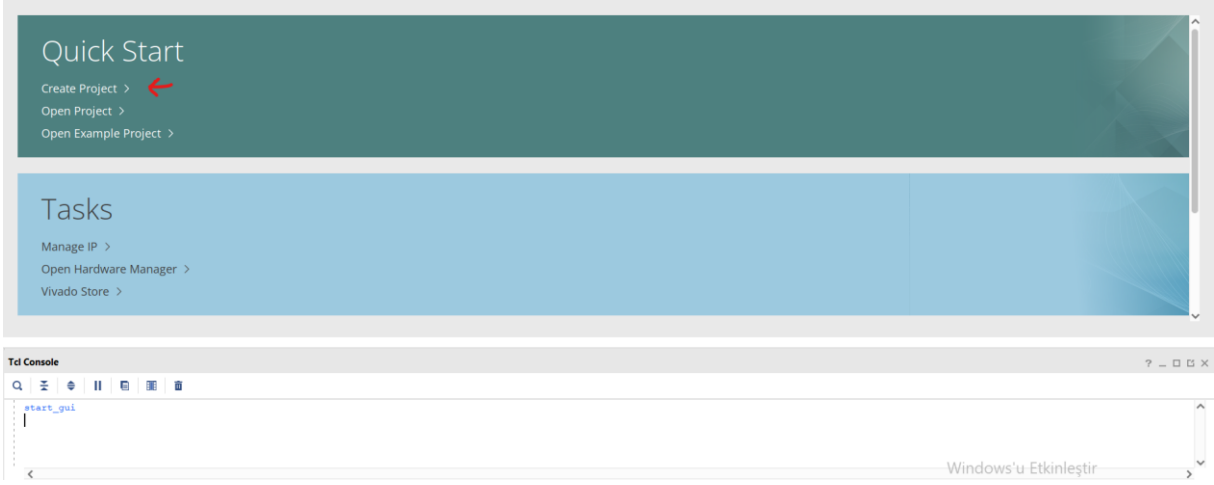
Bu Boolean fonksiyonunu K- Map yardımıyla sadeleştiririm:



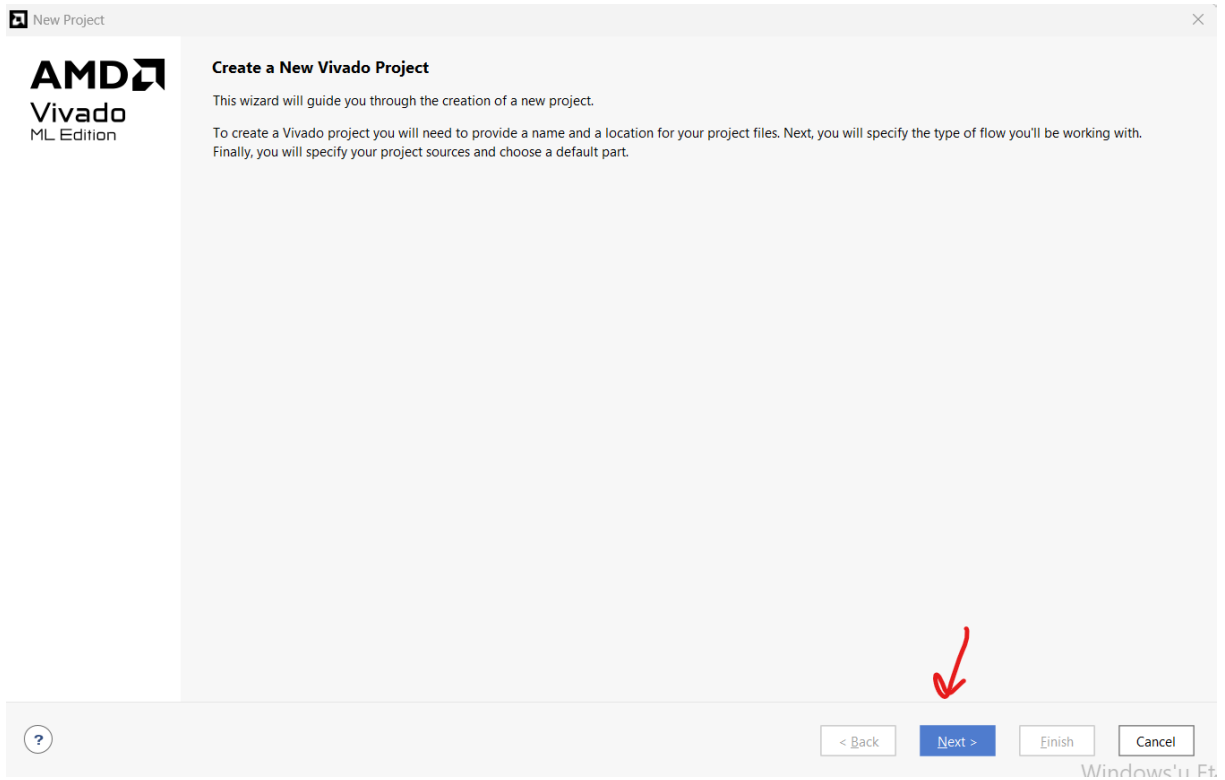
Ardından bulduğumuz fonksiyonu mantık kapıları yardımıyla modelleyelim:



Artık Vivado üzerinden kodumuzu yazmaya başlayabiliriz. Vivado’yu açtıktan sonra bir proje oluşturalım veya önceden oluşturmuş olduğumuz bir projeyi açalım.



Çıkan ekranda “Next”’e tıklayalım.



Açılan ekrandan ilk olarak proje isminizi oluşturalım, ardından dosyalarımızı nereye kaydetmek istiyorsak “...” kısmından klasör seçimi yapalım. (Bu adımdan önce çalışmalarınızı tek bir klasörde muhafaza etmek için bir klasör oluşturabilirsiniz.) İşlemimiz bittikten sonra “Next”’e tıklayalım.

New Project

Project Name
Enter a name for your project and specify a directory where the project data files will be stored.

Project name: lcd_onCalisma_1

Project location: C:/Users/zulal/OneDrive/Desktop/lcd-portfolio

☒ Create project subdirectory

Project will be created at: C:/Users/zulal/OneDrive/Desktop/lcd-portfolio/lcd_onCalisma_1

< Back Next > Finish Cancel

(1) (2)

Bu dönem yapacağımız uygulamalarda sadece “Simulation” ve “RTL Analysis” yapacağımız için “Do not specify sources at this time” kutucuğunu seçiyoruz. Sonrasında tekrar “Next”e tıklıyoruz.

New Project

Project Type
Specify the type of project to create.

☒ **RTL Project**
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.
☒ Do not specify sources at this time
☐ Project is an extensible Vitis platform

☐ **Post-synthesis Project**
You will be able to add sources, view device resources, run design analysis, planning and implementation.
☐ Do not specify sources at this time

☐ **I/O Planning Project**
Do not specify design sources. You will be able to view part/package resources.

☐ **Imported Project**
Create a Vivado project from a Synplify Project File.

☐ **Example Project**
Create a new Vivado project from a predefined template.

< Back Next > Finish Cancel

(1) (2)

Çalışmalarımızda herhangi bir FPGA kartına gömme işlemi yapmayacağımız için açılan ekrandan “Boards” kısmına gelip herhangi bir kartı üstüne tıklayarak seçebiliriz.

New Project

Default Part

Choose a default AMD part or board for your project.

PartsBoards

[Reset All Filters](#)

Category: All

Package: All

Temperature: All

Family: All

Speed: All

Static power: All

Search:

PartI/O Pin CountAvailable IOBsLUT ElementsFlipFlopsBlock RAMsUltra RAMsDSPsBUFGsGb TransceiversGTPE2 Transceiver

?

< Back

Next >

Finish

Cancel

New Project

Default Part

Choose a default AMD part or board for your project.

PartsBoards

To fetch the latest available boards from git repository, click on 'Refresh' button. [Dismiss](#)


[Reset All Filters](#)

Vendor: All

Name: All

Board Rev: Latest

Search:

Display Name	Preview	Status	Vendor	File Version	Part	I/O Pin Count	Board Rev	Available IO
Kria K24C SOM Add Companion Card Connections		Installed	xilinx.com	1.0	Commercial temperature grade K24 SOM	530	Rev_A01	81
Kria K24I SOM Add Companion Card Connections		Installed	xilinx.com	1.0	Industrial temperature grade K24 SOM	530	Rev_A01	81
Kria K26C SOM Add Companion Card Connections		Installed	xilinx.com	1.4	Commercial temperature grade K26 SOM	784	Rev_B01	189

Refresh

?

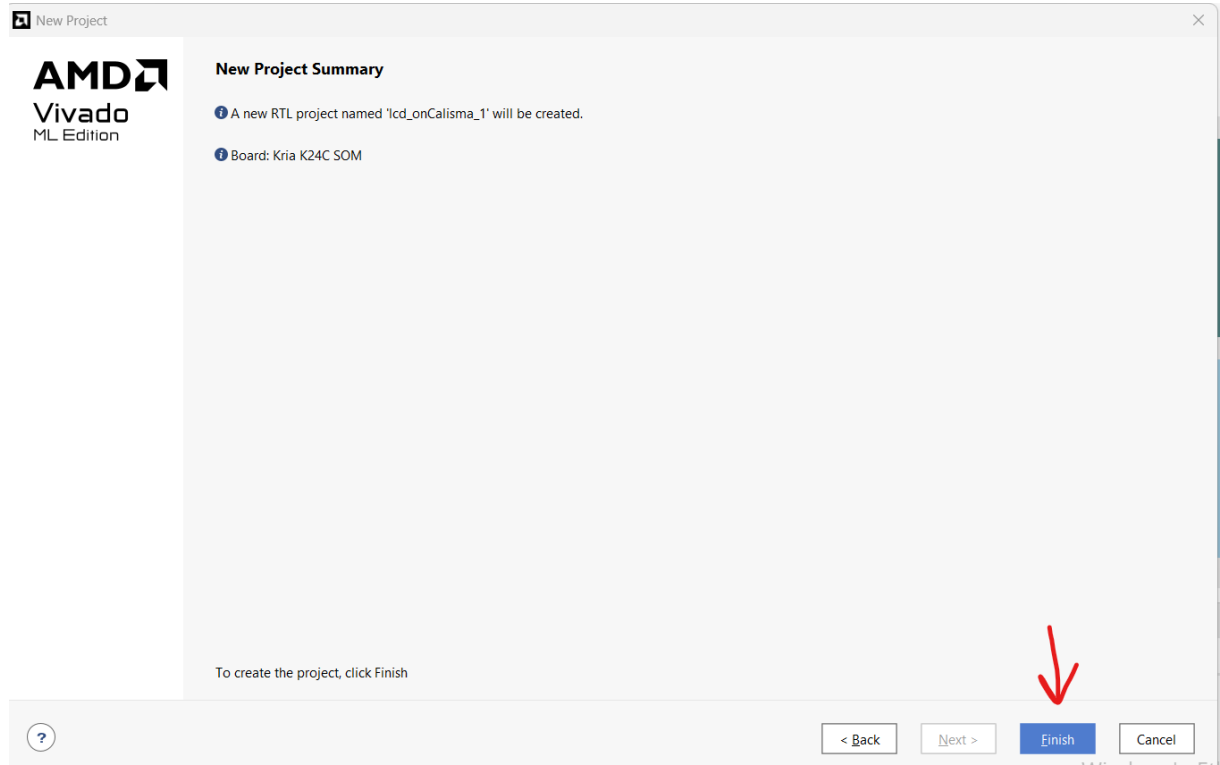
< Back

Next >

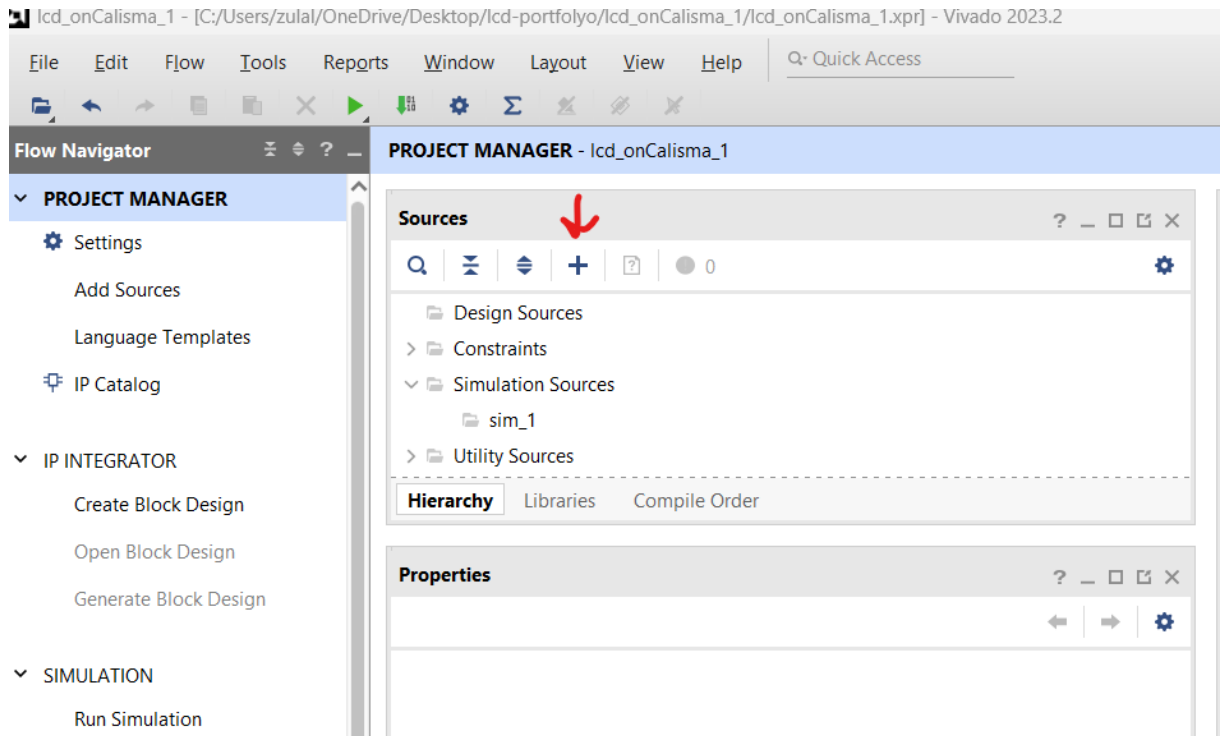
Finish

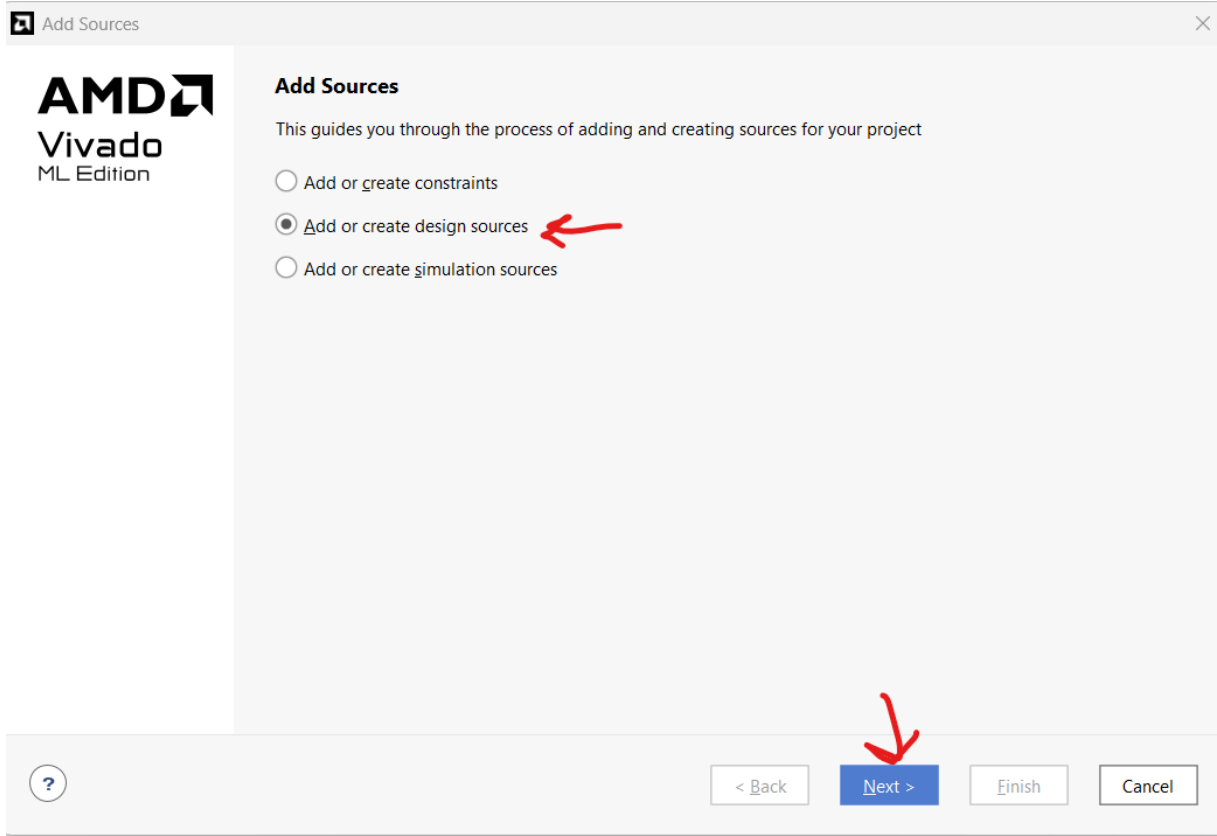
Cancel

En son, “Finish” diyerek projemizi oluşturabiliriz.

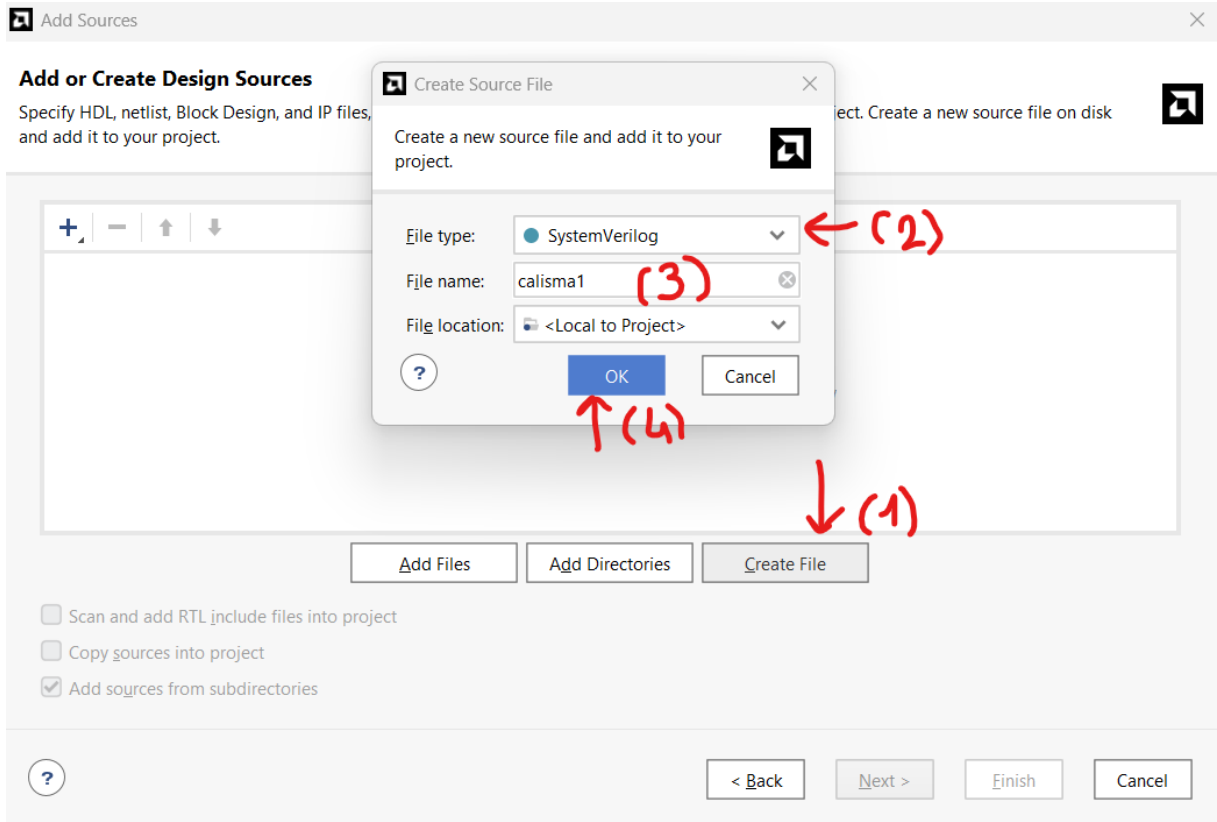


Açılan proje arayüzünün sol üst tarafından “+” simgesine tıklayarak yeni bir design projesi oluşturalım:





“Create File” diyerek dosya türünü “System Verilog” olarak seçelim, dosyamızın ismini oluşturalım (Türkçe karakter kullanmamaya dikkat!) ve “OK” e basalım. “Finish” diyelim.



Karşınıza son adım olarak çıkan bu kısımdan, manuel bir şekilde kod üzerinden girmeden “input, output ya da inout” larınızı oluşturabilirsiniz. Ama daha iyi öğrenmek ve kafa karışıklılığı olmaması adına burayı boş bırakalım ve “OK”e basarak design sources dosyamızı oluşturalım.

Define Module

Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

Module Definition

Module name: calisma1

I/O Port Definitions

Port Name	Direction	Bus	MSB	LSB
input	▼	<input type="checkbox"/>	0	0

?

OK

Cancel

Tekrardan “Design Sources” kısmına gelelim ve oluşan dosyamıza çift tıklayarak dosyamızı açalım:

File Edit Flow Tools Reports Window Layout View Help Q Quick Access

Flow Navigator

PROJECT MANAGER

Settings

Add Sources

Language Templates

IP Catalog

IP INTEGRATOR

Create Block Design

Open Block Design

Generate Block Design

SIMULATION

PROJECT MANAGER - lcd_onCalisma_1

Sources

Design Sources (1)

calisma1 (calisma1.sv) ←

Constraints

Simulation Sources (1)

sim_1 (1)

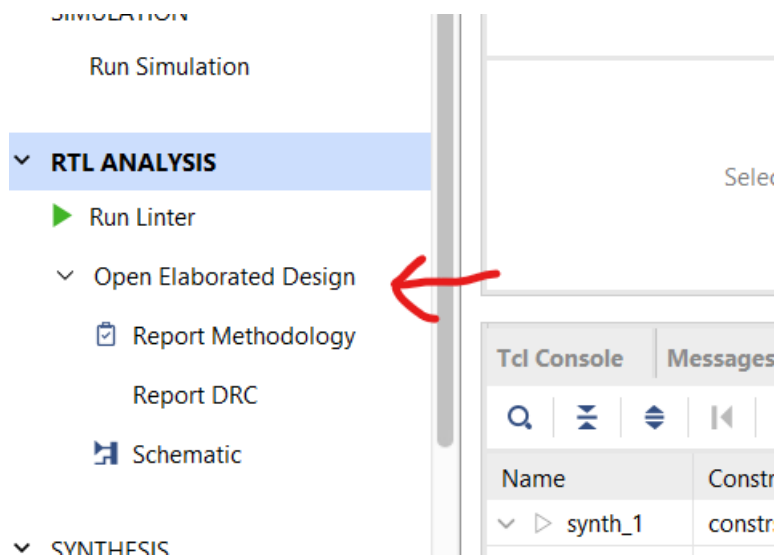
Hierarchy Libraries Compile Order

Properties

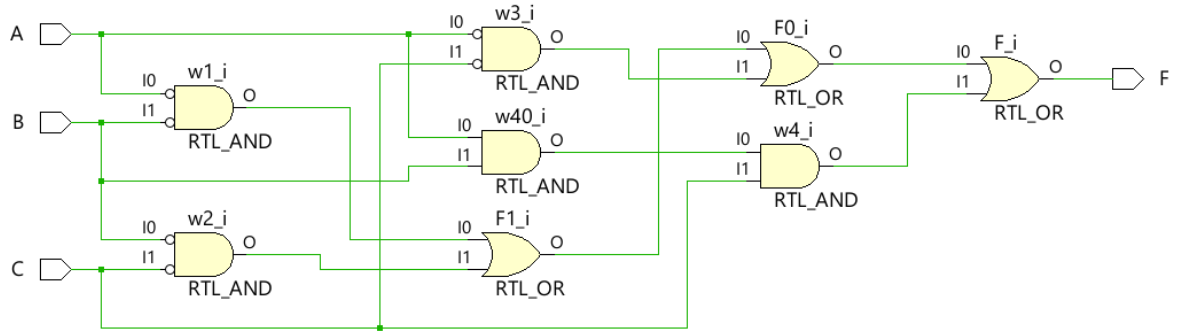
Kodumuzu yazmaya başlayalım:

```
module calisma1(  
    input logic A, //1 bit  
    input logic B, //1 bit  
    input logic C, //1 bit  
    output logic F //1 bit  
);  
  
    // Ara sinyaller  
    logic Anot, Bnot, Cnot, w1, w2, w3, w4;  
  
    not(Anot,A); // önce output(lar), ardından input(lar) yazılır.  
    not(Bnot,B);  
    not(Cnot,C);  
    and(w1,Anot,Bnot); // A'B'  
    and(w2,Bnot,Cnot); // B'C'  
    and(w3,Anot,Cnot); // A'C'  
    and(w4,A,B,C); // ABC  
    or(F,w1,w2,w3,w4); // A'B' + B'C' + A'C' + ABC  
  
    // assign F = ((!A && !B) || (!B && !C) || (!A && !C) || (A && B && C));  
    // A'B' + B'C' + A'C' + ABC  
    // "assign" komutu kullanarak tek satırda da işi halledebiliriz.  
endmodule
```

Ardından Vivado ekranımıza geri gelelim sol taraftan “RTL Analysis” kısmının altındaki “Open Elaborated Design” kısmına tıklayarak açılan ekrandan “OK”e basalım.



Devre tasarımıımız şu şekil gözükecektir:



Artık simülasyon kısmına geçebiliriz. Simülasyon kısmını yazmak için fonksiyonumuzun doğruluk tablosunu çıkartmakla işe başlayalım:

A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Tekrar “Sources” kısmına gelerek “+” simgesinden bu sefer “add or Create Simulation sources” kısmına tıklıyoruz:

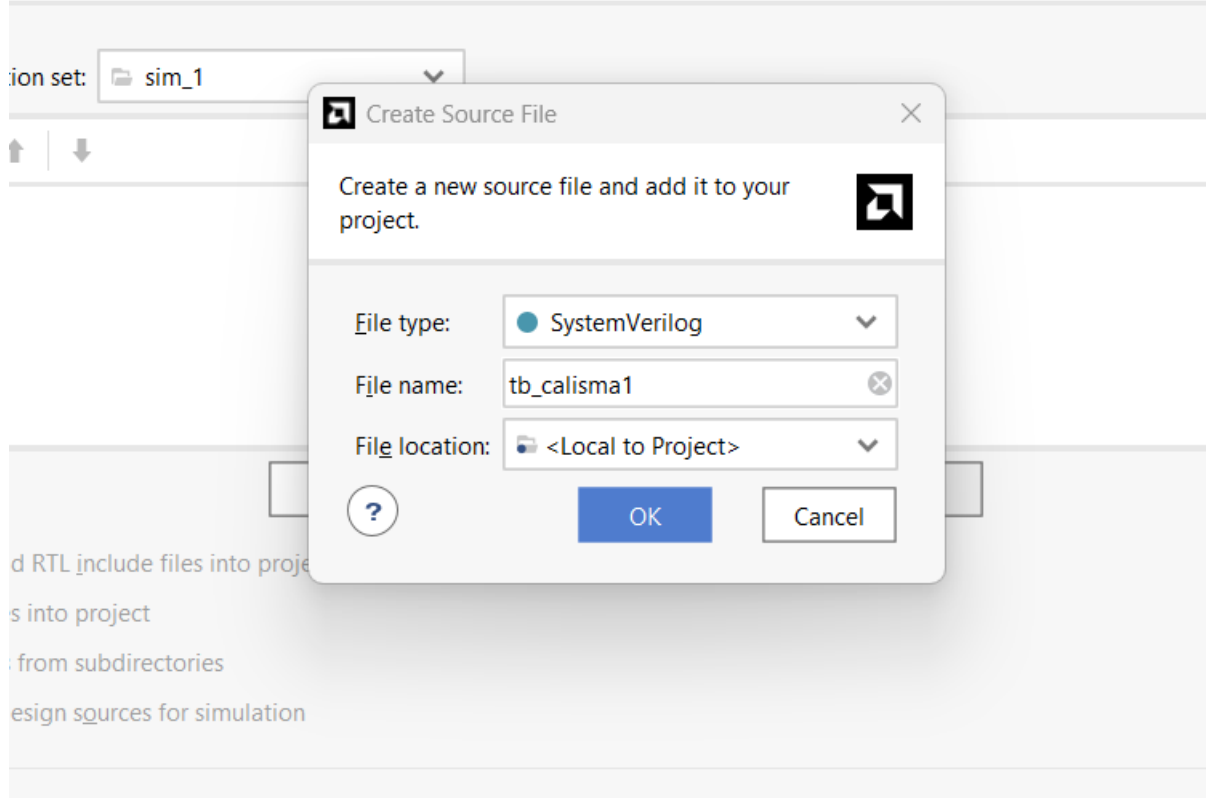
Add Sources

This guides you through the process of adding and creating sources for your project

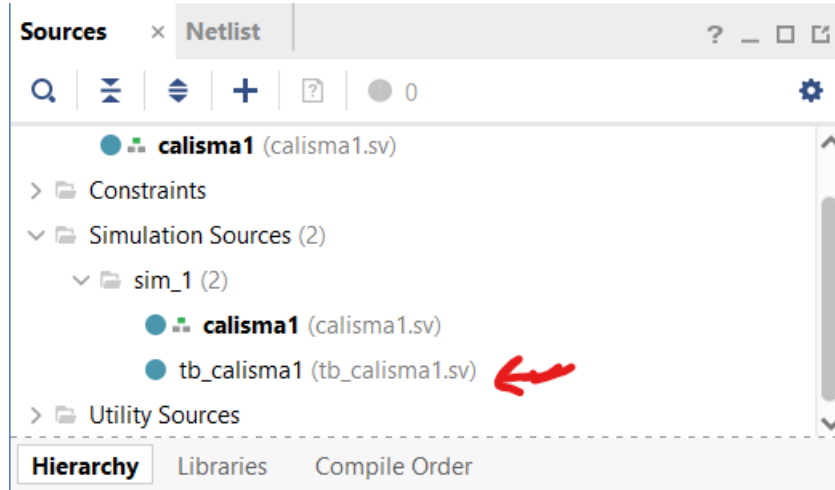
- ☐ Add or create constraints
- ☐ Add or create design sources
- ☒ Add or create simulation sources



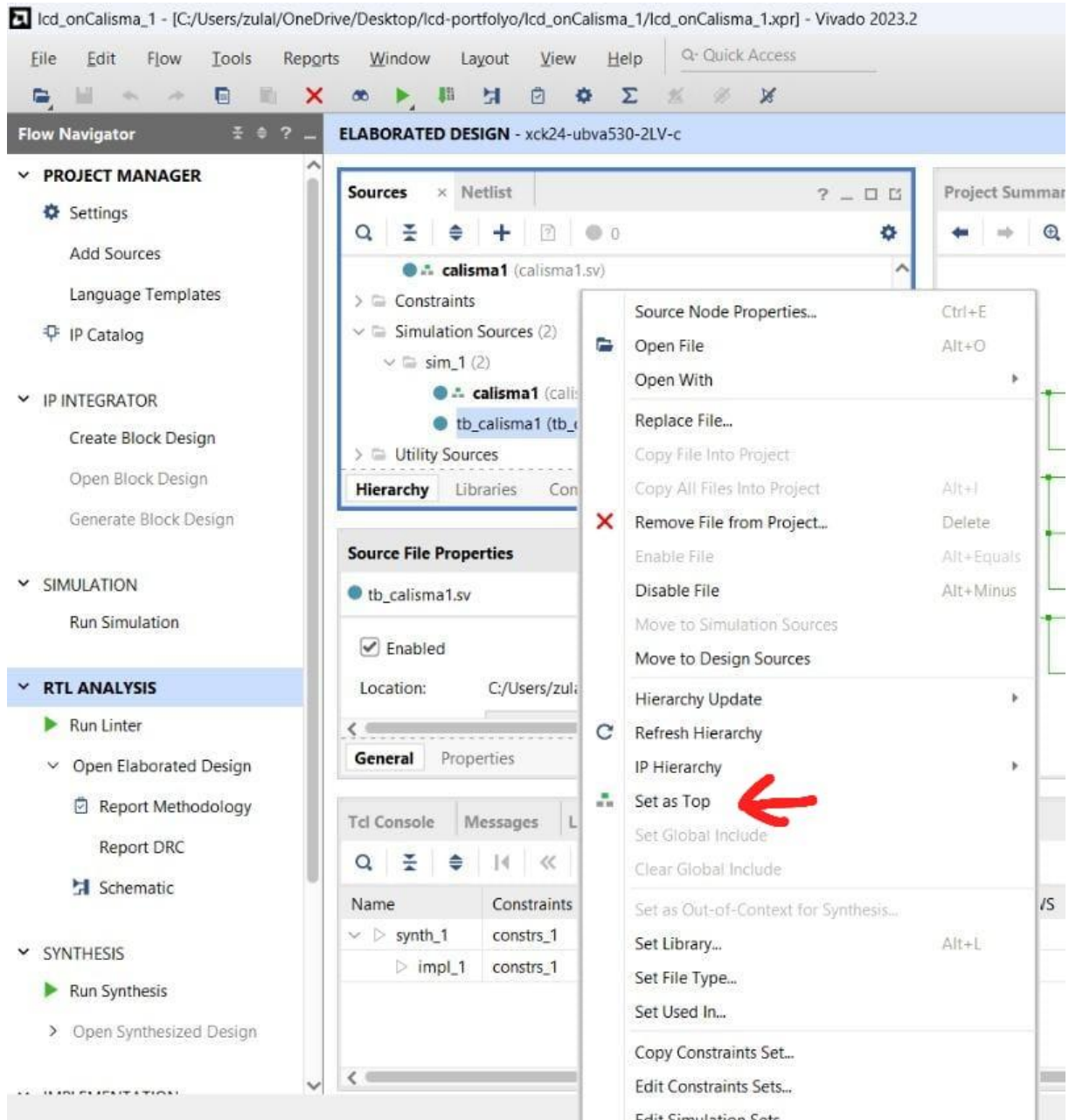
“Create File” diyerek dil olarak System Verilog seçtikten sonra dosya ismimizi önceden oluşturduğumuz design dosya ismimizin önüne “tb_dosyaAdı” olacak şekilde yapıyoruz. (tb = testbench’in kısaltması olarak kullanılır.)



Ardından “Finish”e basarak simülasyon dosyamızı oluşturmuş oluyoruz. “Sources” kısmından simulation bölümünün altındaki testbench dosyamıza tıklayarak dosyamızı açıyoruz.



(NOT : Bu görselde de görüldüğü üzere testbench dosyamız seçili değil. Testbench dosyamızı seçmek için dosyanın adının üzerine sağ tık yaparak “Set as top” şeklinde ayarlamalıyız.



Ayarlamayı yaptıktan sonra kodumuzu yazmaya başlayabiliriz.

```
module tb_calisma1();  
  
    logic A; // reg  
    logic B; // reg  
    logic C; // reg  
    logic F; // wire
```

```

/*
System Verilog'un güzel taraflarından birisi de reg - wire ayırımına
gerek duymamasıdır. Verilog diliyle kodlama yapmamız gerekseydi inputlar
için reg,
    outputlar için wire kullanmamız gerekecekti. Her ne kadar bu tip basit
projelerde
    kafa karışıklığı yaratmasa da işler komplikeleştikçe bu durum karışıklığa
yol açabilir.
*/

calisma1 calisma1_Inst(
    .A(A),
    .B(B),
    .C(C),
    .F(F)
);

// design kodunu testbench kodunda kullanmak için bu modülün çağrılması
gerekir.
// .A(A) -> giriş veya çıkış portlarını üst seviyedeki modüllere bağlamak
için kullanılır.

/*
NOT: Tüm modülleri kullanacaksak (bağlayacaksak),
    calisma1 calisma1_Inst(.*);
şeklinde de yazabiliriz.
(Aynı şekilde sadece System Verilog'a özgü bir özelliktir.
Verilogta hepsini üstteki gibi teker teker bağlamak zorundasınız.)
*/

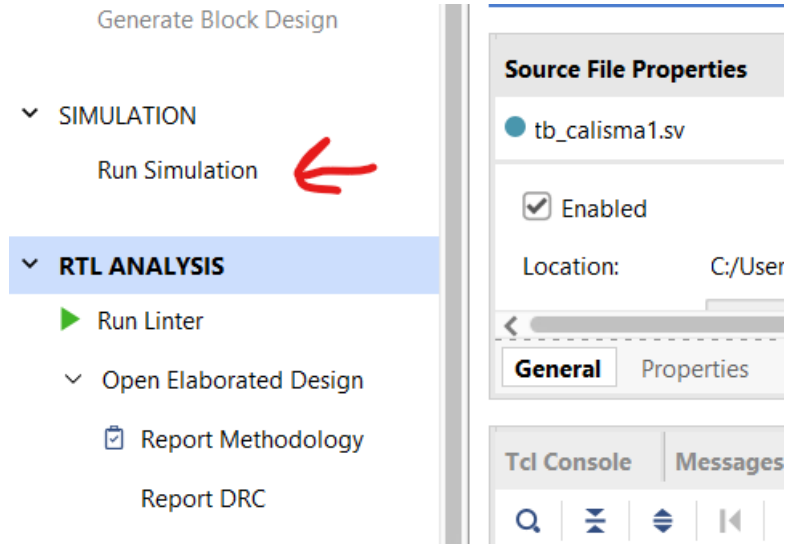
initial begin
    A = 1'b0; B = 1'b0; C = 1'b0; #10;
    A = 1'b0; B = 1'b0; C = 1'b1; #10;
    A = 1'b0; B = 1'b1; C = 1'b0; #10;
    A = 1'b0; B = 1'b1; C = 1'b1; #10;
    A = 1'b1; B = 1'b0; C = 1'b0; #10;
    A = 1'b1; B = 1'b0; C = 1'b1; #10;
    A = 1'b1; B = 1'b1; C = 1'b0; #10;
    A = 1'b1; B = 1'b1; C = 1'b1; #10;

    // 1 -> 1 bit, b -> binary, 0,1 -> değer
    // #10 -> 10ns bekle
    $stop; // simülasyonu durdurur, olası hataları görmek için işlevlidir.
end

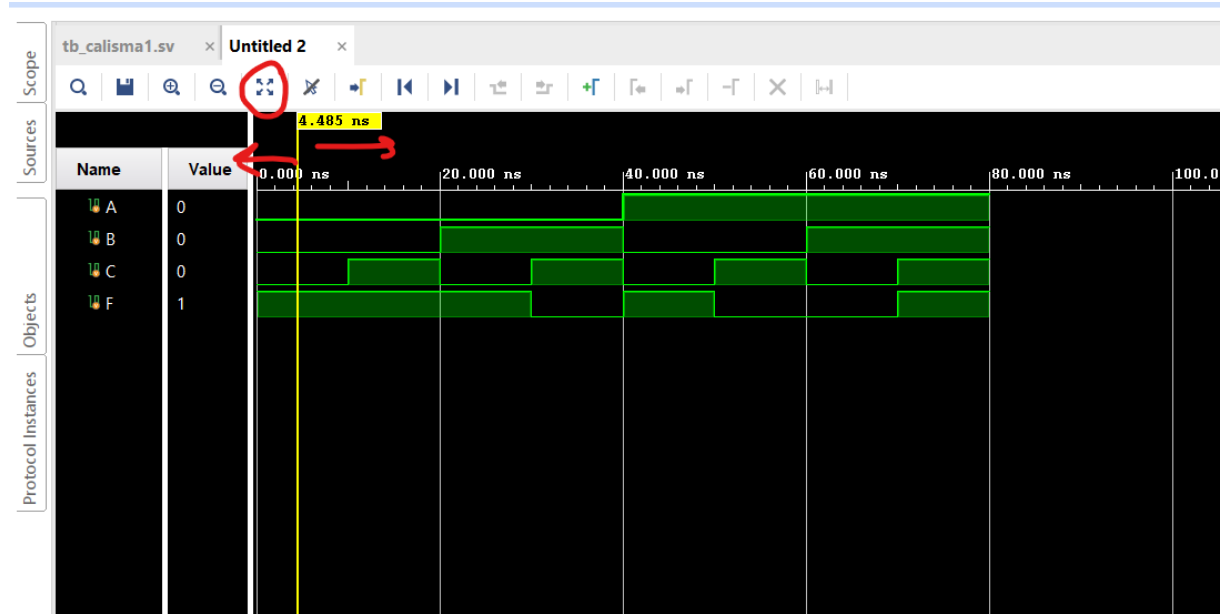
endmodule

```

Kodumuzu yazdıktan sonra tekrar Vivado ekranına gelerek sol taraftan “Simulation – Run Simulation – Run Behavioral Simulation” diyerek simülasyonu açıyoruz.



Çıkan simülasyon ekranından büyüteç yardımıyla yakınlaştırp uzaklaştırabilir, istersek tam ekranda tüm sonuçları da görüntüleyebiliriz. Sarı çubuğu sağ – sol hareketler yardımıyla hareket ettirerek anlık değişimleri daha detaylı gözlemleyebiliriz.



Bu örnekte basit bir Boolean fonksiyonunu Vivado üzerinden System Verilog dili yardımıyla kodlamış bulunmaktayız.