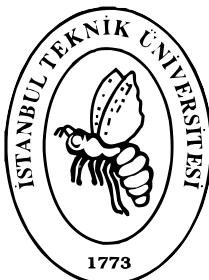


İSTANBUL TEKNİK ÜNİVERSİTESİ

ELEKTRİK - ELEKTRONİK

FAKÜLTESİ



EHB 326E
Introduction to Embedded Systems
2018-2019 Güz
(CRN:12273)

Öğretim Üyesi: Prof. Dr. Müştak Erhan Yalçın
Ders Yardımcısı: Araş. Gör. Emrah Abdioğlu

Final Projesi Raporu

Adı : Ceyhun
Soyadı : YAMANEREN
Öğrenci Nu. : 040140010

İmza

FİNAL PROJESİ RAPORU

1) AMAÇ

Booth algoritması kullanılarak 8-bitlik bir çarpma bloğu tasarlanmıştır. Sistemde bulunan Picoblaze, Block RAM'dan "A" ve "B" sayılarını okuyup, çarpanı bloğuna iletmektedir. Çarpanı bloğu "done" sinyalini "1" yaptığında, blokta oluşan 16-bitlik "result" değeri, Picoblaze tarafından "s0" ve "s1" register'larına kaydedilmektedir.

2) BOOTH ALGORİTMASI

Booth algoritması, iki adet n-bitlik işaretli sayının çarpılması için kullanılan bir çarpma algoritmasıdır. Bu algoritmasının avantajı, çarpan sayıdaki ardışık bit durumları bulunduğuunda (00, 11), diğer çarpma algoritmalarına göre daha az toplama ve çıkarma işlemi yapılmasınadır.

$A \times X$ işlemi için, öncelikle X sayısı üzerinde inceleme yapılması gerekmektedir. n -bitlik X sayısının en önemsiz bitinin sağına bir adet "0" biti eklenerek " $n+1$ " bitlik bir X_{Yeni} sayısı elde edilir. Örneğin 4-bitlik bir 1011 sayısı için, X_{Yeni} sayısı 10110 olacaktır. Daha sonra elde edilen sayının en önemsiz bitinden itibaren, yan yana olan bit çiftlerinin değerlerine göre, aşağıdaki tabloda belirtilen işlemler yapılır.

x_i	x_{i-1}	İşlem	y_i
0	0	sadece kaydır	0
1	1	sadece kaydır	0
1	0	çıkar ve kaydır	1'
0	1	ekle ve kaydır	1

Şekil 1 – Booth Algoritması Çarpan Sayı Bit Çiftlerine Göre İşlemler [1]



Şekil 2 – Booth Algoritması Çarpan Sayı Bit Çiftlerine Göre İşlemler - 2 [1]

Şekil 1'de görülen tablodaki işlemler 10110 sayısını için yapıldığında, en önemsiz iki bit olan "10"dan dolayı, çarpanın tümleyeninin toplanması (yani çıkarılması), "01"den dolayı çarpanın toplanması, "10"dan dolayı çarpanın çıkarılması, "11"den dolayı yalnızca kaydırma işlemi yapılması gerekmektedir. Bu sebeple Şekil 1'deki tabloya bakılarak, işlem sırası sağdan sola olmak üzere, $Y=(1'101')$ sayısı elde edilmiştir.

Şekil 3'teki adımlara göre, A sayısı “1110” seçilirse, daha önce edilmiş Y sayısının en önemsiz bitinden başlanarak adım adım işlemler gerçekleştirilir. İlk adım, çıkarma işlemidir. Bu sebeple A sayısının tümleyeni toplanmıştır. Daha sonra ise işaret bitine göre bir sağa kaydırılmıştır. İşaret bitine göre kaydırma işlemi yapılırken, sayının en önemli bitine göre, sağa kaydırma işlemi yapıldıktan sonra sayıya gelmesi gereken en önemli bit seçilir. Bu sayede sağa kaydırma işleminde işaret biti korunmuş olur. Y sayısının diğer bitlerinde gözüken işlemler de yapıldığında, 8-bitlik sonuç “00001010” olarak bulunmuştur.

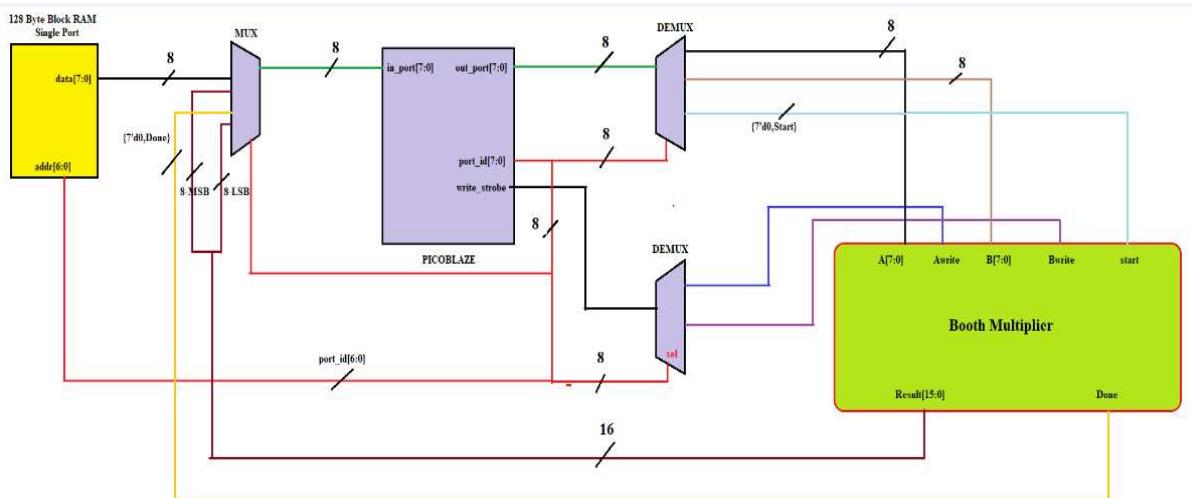
A	1110	-2
X	\times 1011	-5
Y	1'101*	recoded multiplier
-A'yi Topla	0010	(y1=1, A'yi çıkaracağımız yani A'nın 2'ye tümleyenini toplayacağımız)
Kaydır	000010	(Topladıktan sonra işaret bitine göre 1 bit sağa kaydıracağımız)
Kaydır	000010	(y2=0, sadece kaydıracağımız.)
A'yi Topla	+ 1110	(y3=1, A'yi toplayacağımız)
	111010	
Kaydır	1111010	(İşaret biti 1 olduğu için sağa 1 ekleyerek kaydırık.)
-A'yi Topla	+ 0010	
	0001010	
Kaydır	00001010	Sonuç = 10

Şekil 3 – Booth Algoritması İşlem Adımları [1]

3) GENEL TASARIM

Tasarlanacak olan sistemin en genel hali Şekil 4'te verilmiştir. Görüldüğü üzere sistemde, bir adet 128 Byte'lık single port Block RAM, bir adet Picoblaze işlemci, bir adet MUX, iki adet DEMUX ve bir adet Booth çarpanı bloğu bulunmaktadır.

Sistem öncelikle, Block RAM'den iki adet sekiz bitlik sayı okuyup Picoblaze'in register'larına yazmaktadır. Daha sonra bu iki sayı sırasıyla A ve B olarak, Booth çarpanı bloğuna gönderilirmektedir. Sayılar ilgili port_id değerleri ile A ve B girişine yazılırken, write_strobe'un bağlı olduğu DEMUX da, A ve B gönderiminde kullanılan aynı port_id'lerdeki çıkışlarda, A ve B'nin gönderimi sırasında oluşan write_strobe işaretini, Awrite ya da Bwrite girişlerine göndererek, gönderilen A ve B sayılarının start komutu verilene ve çarpma işlemi bitene kadar, çarpanı bloğunda saklanması sağlanır.

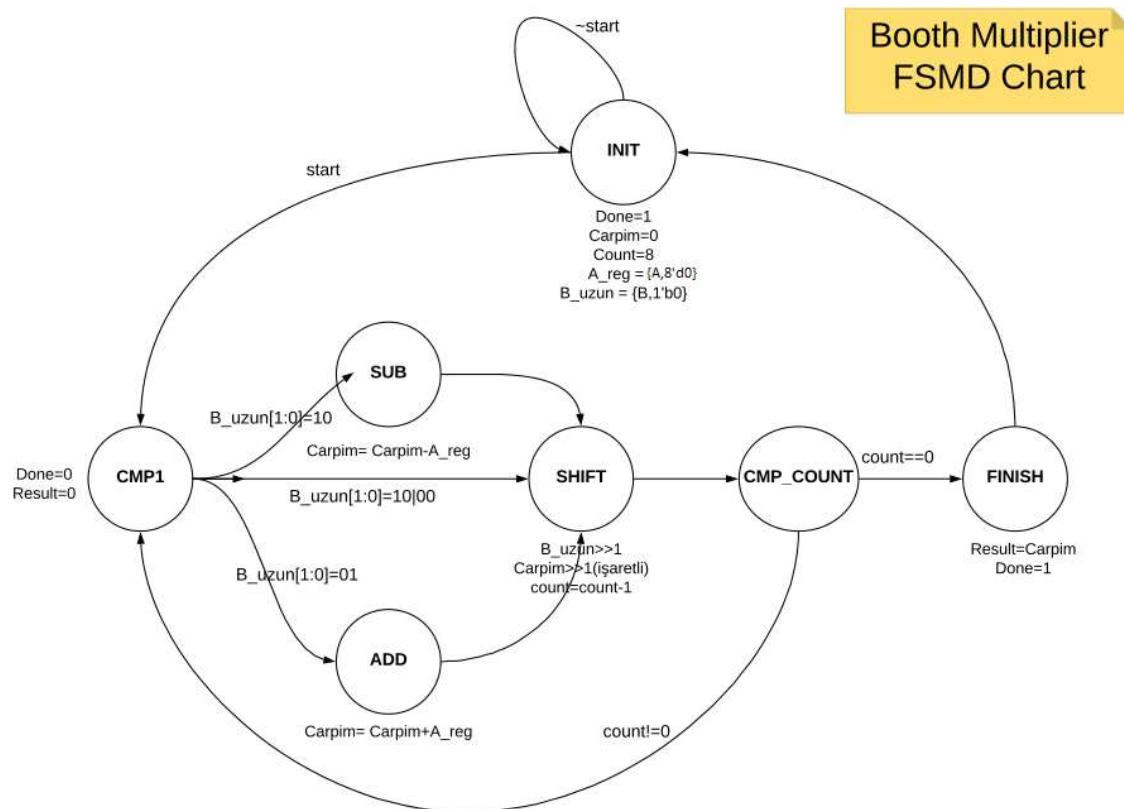


Şekil 4 – Sistemin genel hali

Daha sonra Picoblaze tarafından start="1" verildiğinde, çarpanı bloğunda A ve B sayıları çarpılmaya başlar. Bu sırada Picoblaze, ilgili port_id'den, Done sinyalinin "1" gelmesini yakalamaya çalışır. Çarpanı bloğu, çarpma işlemi bittikten sonra öncelikle Done="1" yapar ve ardından çarpım sonucunu Result çıkışına verir. Picoblaze, Done="1"'i yakaladıktan sonra, ilgili port_id'lerden, Result sonucunun en önemli 8-bitini s0 register'ına, en önemsiz 8-bitini s1 register'ına kaydeder. RAM'de bulunan 128 adet 8-bitlik işaretli sayı, birbirleriyle ardışık olarak çarpılarak Picoblaze içerisindeki program çalışmaya devam eder. RAM'in tüm adresleri okunduktan sonra, RAM'in 0. Adresinden itibaren tekrardan sayılar çarpılmaya başlanır ve bu olay döngü şeklinde devam eder.

4) BOOTH ÇARPICI BLOĞU TASARIMI

Aşağıdaki şekilde, Booth Multiplier bloğunun, FSMD (Finite State Machine with Data) şeması görülmektedir.



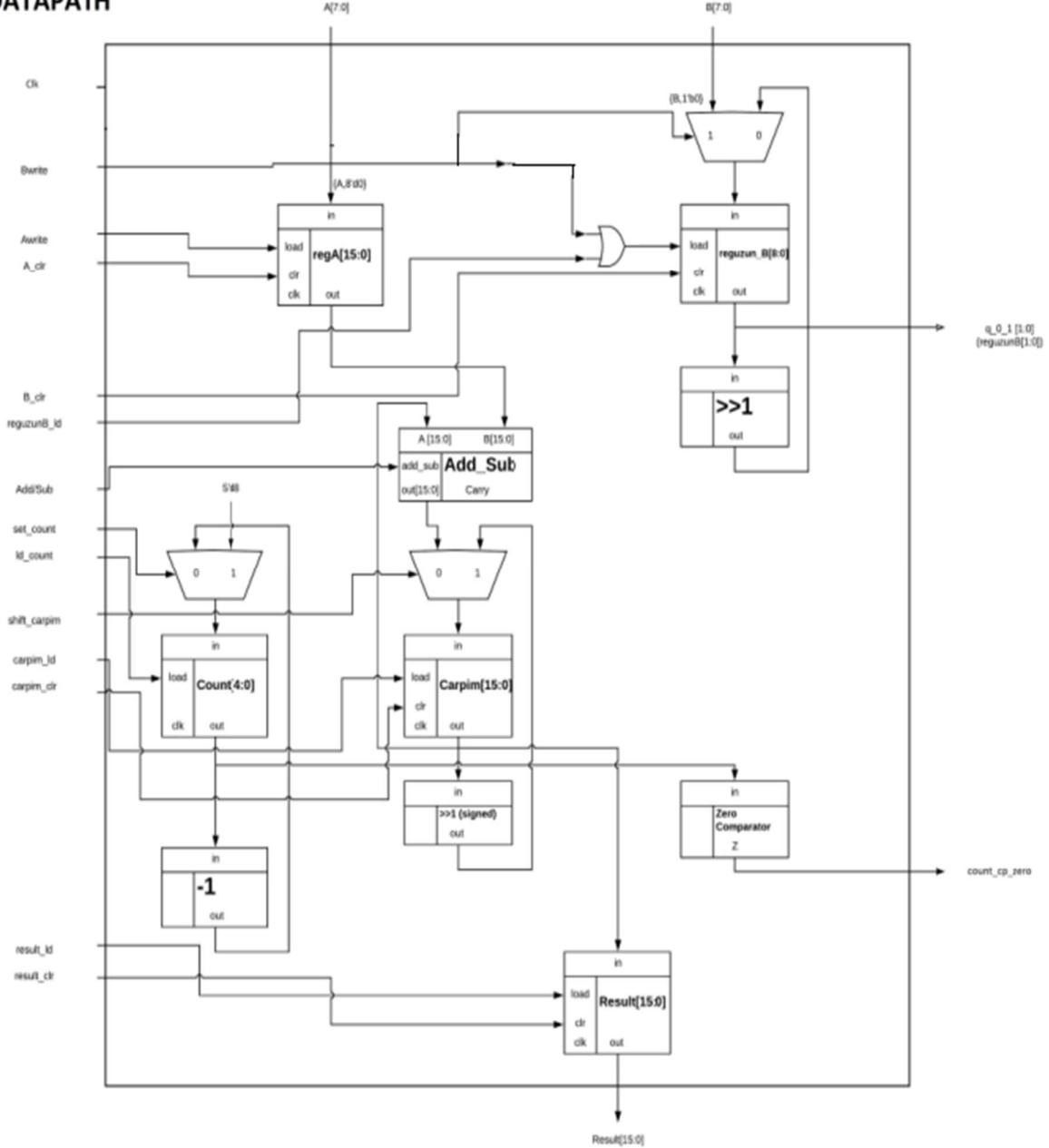
Sekil 5 – Booth çarpanı bloğu, FSMD şeması

Şekil 5'te görüldüğü üzere, durum makinesi ilk başta INIT durumundadır. Bu durumdayken, A_reg isimli register'a, 16-bitlik şekilde $\{A,8'd0\}$ sayısı atanmakta, B_uzun isimli register'a $\{B,1'b0\}$ sayısı atanmaktadır. A_reg isimli register'a bu şekilde bir atama yapılmasının sebebi, A sayısının, çarpım register'ı ile işleme sokulacağı zaman, sayıların en önemli bitlerinden başlayacak şekilde işlem yapılması istenmesidir. B_uzun isimli register'in 9-bit olması, Booth algoritmasından dolayı kaynaklanan, çarpan sayısının en önemsiz bitinin yanına bir adet "0"

bitinin eklenmesinden kaynaklanmaktadır. INIT durumunda count isimli sayıçı register'ına “8” değeri atanarak, çarpma işlemi için gereken adımların, “8” kere yapılması sağlanmıştır. Çarpıcının start girişi “0” olduğu sürece, Booth çarpcı devresi durumunu korumakta, start=“1” olduğu zaman ise devre CMP1 durumuna geçmektedir. CMP1 durumunda, çarpma işlemi başladığı için Done=“0” yapılmakta, çarpma işleminin sonucunun bulunduğu register olan Result, bu durumda sıfırlanmaktadır. Çünkü INIT durumunda, çarpım işlemi bittikten sonra ortaya çıkan Result değeri, yeni bir çarpma işlemi başlatılana kadar, Picoblaze tarafından algılanabilmesi için çıkışta gözükmelidir. CMP1 durumunda, B_uzun değerinin en önemsiz iki bitinin aldığı değerlere bakılarak, Booth algoritması gereği, “10” değeri için SUB durumuna, “01” durumu için ADD durumuna, diğer değerler için direk “SHIFT” durumuna geçiş yapılmaktadır. SUB durumunda A_reg register'ındaki değer, Carpim register'ındaki değerden çıkarılmakta, ADD durumunda A_reg register'ındaki değer, Carpim register'ındaki değerle toplanmakta ve bu işlemlerden sonra SHIFT durumuna geçiş yapılmaktadır. SHIFT durumunda; B_uzun register'ındaki değer, bir kere sağa kaydırılmakta, Carpim register'ındaki değer bir kere işaret biti gözünde bulundurularak sağa kaydırılmakta ve count register'ındaki değer “1” azaltılmaktadır. Daha sonra CMP_COUNT durumuna geçilerek, count register'ı içerisinde tutulan sayının “0”a eşit olup olmadığı karşılaştırılmaktadır. Eğer count değeri, sıfırdan farklı ise tekrar CMP1 durumuna geçilmekte ve çarpma işlemi için gereken adımlar tekrarlanmaktadır. Count değerinin sıfıra eşit olduğu anda ise, çarpma işlemi tamamlandığı için, FINISH durumuna geçilmektedir. FINISH durumunda, Carpim register'ındaki değer Result register'ına aktarılmaktadır. Ayrıca Done çıkışı “1” yapılarak, çarpma işleminin tamamlandığı anlaşılmaktadır. FINISH durumundan sonra ise INIT durumuna dönülerek, yeni çarpma işlemi için devre hazır hale getirilmektedir.

FSMD tasarıımı tamamlandıktan sonra, yukarıda görülen aritmetik ve lojik işlemler için, ayrıca verilerin tutulması için, register blokları ve kombinazonsal işlem bloklarının, bir datapath modülü içerisinde tasarlanması ve bir datapath modülünün oluşturulması gerekmektedir. Bu sebeple Şekil 6'daki datapath modülü tasarımını yapılmıştır. Görüldüğü üzere modül içerisinde, 5 adet farklı bit uzunluklarında register, 2 adet sağa kaydırma bloğu (biri işaretli), bir adet toplama-çıkarma bloğu, bir adet “1” sayısı çıkarma bloğu, 1 adet sıfır karşılaştırıcı bloğu, 3 adet MUX, ve iki adet lojik kapı kullanılmıştır. Datapath modülünde, register'lara girecek veriler için, register'ların kontrolü ve kombinazonsal blokların kontrolü için, controller modülünden gelen, kontrol girişleri, controller'e durum geçişlerinde kullanılmak üzere gönderilen kontrol çıkışları ve çarpılacak sayıların ile çarpım işleminin çıkışa verilmesi için gerekli giriş ve çıkış işaretleri bulunmaktadır.

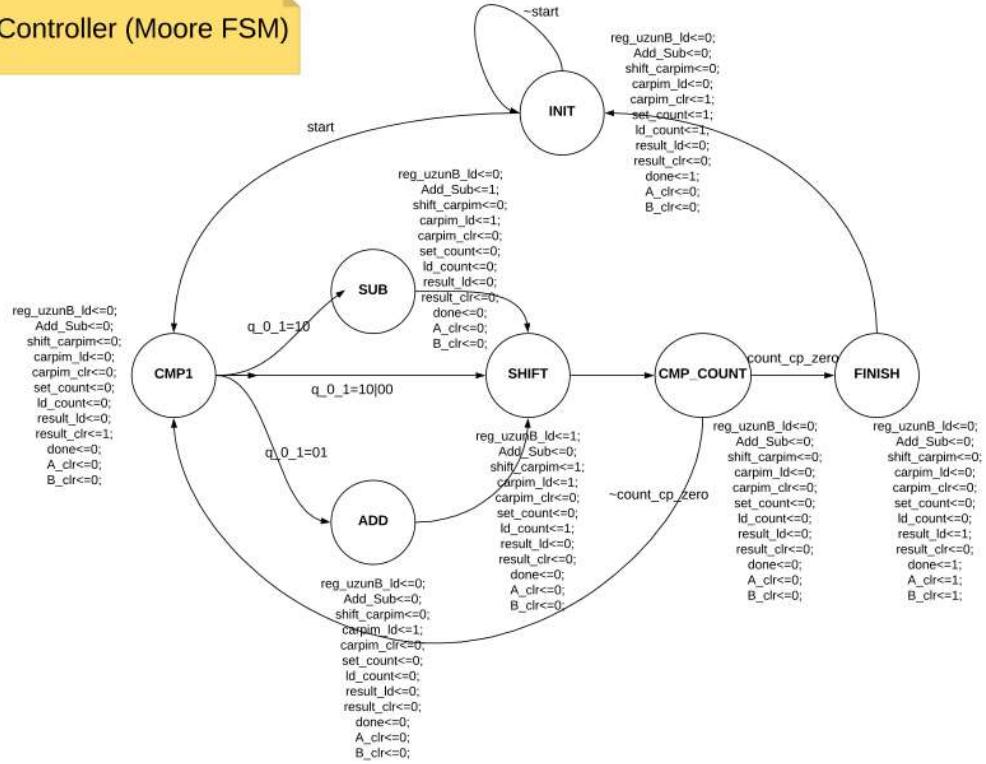
DATAPATH



Şekil 6 – Booth çarpiç bloğu, Datapath modülü

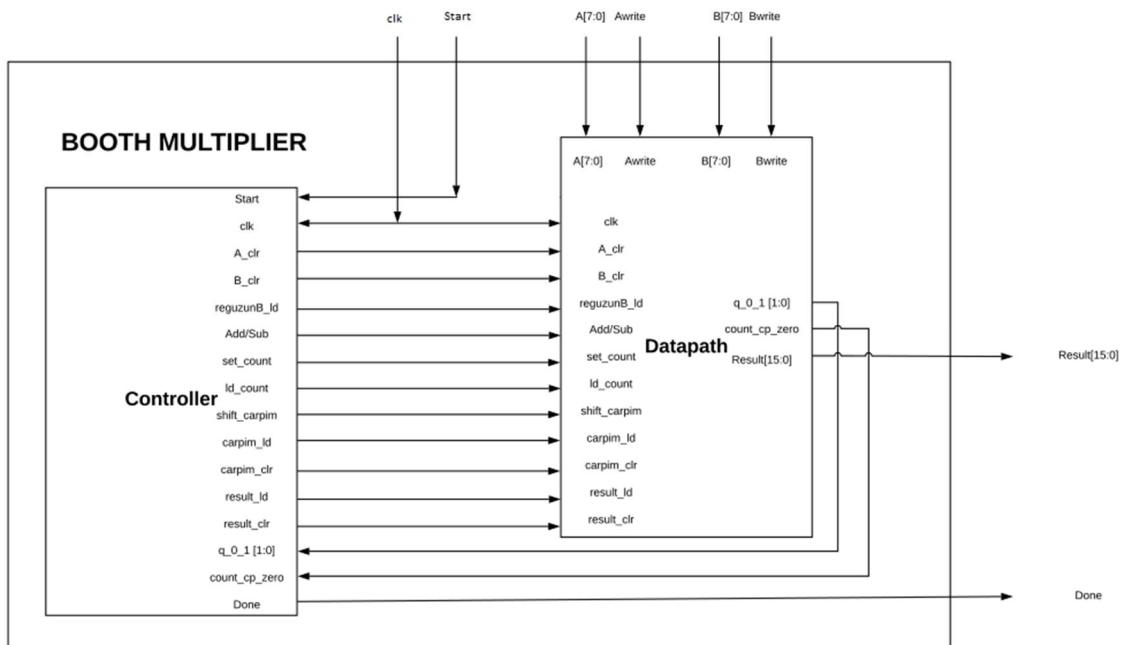
Datapath modülü oluşturulduktan sonra, FSMD şemasında görülen işlemlerin, ilgili durumlarda uygulanması için gerekli kontrol sinyallerini üreten controller modülünün tasarımları yapılmıştır. Controller modülü, FSMD şemasında da görüldüğü üzere, Moore makinesi şeklinde bir FSM yapısında tasarlanmıştır. Aşağıdaki şekilde Controller modülü için tasarlanan Moore makinesinin FSM şeması görülmektedir. Görüldüğü üzere, Şekil 5'teki FSMD şemasında yapılması gereken işlemler, datapath modülünden alınan ve modüle gönderilen kontrol işaretleri ile uygulanmıştır.

Booth Multiplier Controller (Moore FSM)



Şekil 7 – Booth çarpıcı bloğu, Controller modülü, FSM şeması

Tasarlanan Controller ve Datapath modülleri, aşağıdaki şekilde görüldüğü üzere, bir üst modülde gerçekleştirilmiş ve bu modüller birbirlerine bağlanarak Booth Multiplier bloğunun tasarımını tamamlanmıştır.



Şekil 8 – Booth Multiplier Modülü

Controller, Datapath modülleri ve üst modül, akış şemaları ile tasarlardıktan sonra, Xilinx ISE üzerinde açılan projede, Verilog dili ile gerçeklenmiştir. Bloğun bütün alt modüllerinin ve üst modüllerinin Verilog kodları **EK’te** gösterilmiştir.

5) PICOBLAZE İÇİN İLGİLİ PROGRAM TASARIMI

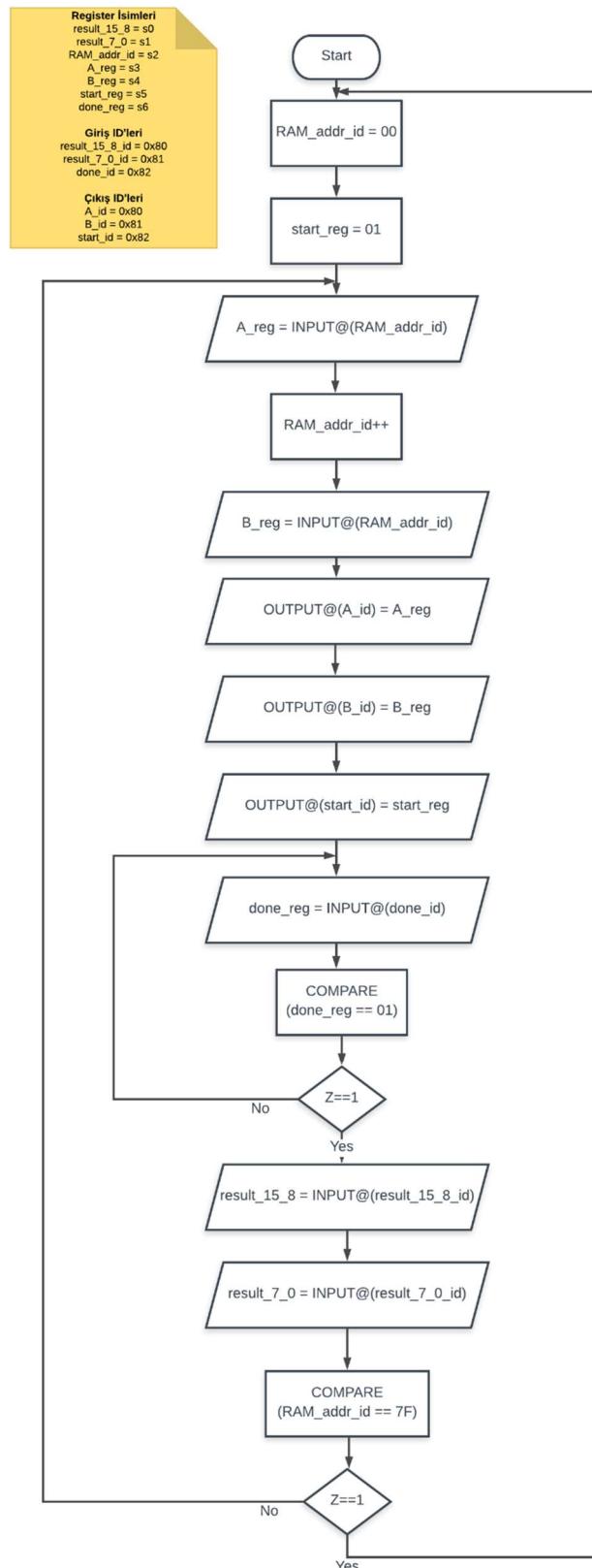
Booth Multiplier modülü Verilog ile tasarlandıktan sonra, Şekil 4’teki sistemde görülen, çarpıcı modülde çarpılacak sayıları Block RAM’den çekip çarpıcı modüle aktaran, çarpma işlemini modül üzerinde başlatan, çarpma modülünden gelen done işaretine göre, çarpım sonucunu kendi register’larına kaydeden Picoblaze işlemcisi için, anlatılan komutların yerine getirilebilmesi amacıyla assembly dilinde (KCPSM3 standartı) bir program yazılmıştır. Tasarlanan programın akış şeması, Şekil 9’da gösterilmiştir.

Program çalışmaya başladığı anda, öncelikle block RAM’den okuma yapması için gerekli olan adresin, port_id üzerinden RAM’e gönderilmesi için ve bu adreslerin program içerisindeki ilgili döngülerde ayarlanabilmesi için, RAM_addr_id isimli bir register kullanılmıştır. Başlangıçta bu register’a “0x00” değeri yüklenerek, RAM’in sıfırıncı adresinden itibaren okuma yapılması amaçlanmıştır. Sonrasında, start_reg isimli bir register’a “0x01” değeri yüklenmiştir. Çünkü Picoblaze’in, çarpıcı bloğu üzerinde çarpma işlemini başlatabilmesi için, çarpıcı bloğun start girişini “1” yapması gerekmektedir. Picoblaze’in çıkışları 8-bit olduğundan, çarpıcı bloğun start girişine giden 8-bitlik işaretin sadece en önemsiz biti, çarpıcı bloğun start girişine bağlanmıştır.

Register’a değer yüklemeleri yapıldıktan sonra, RAM_addr_id register’ının sahip olduğu değerdeki port_id üzerinden, A_reg register’ına giriş alınır. Bu giriş çarpılacak A sayısını, Picoblaze içine yazma islevi için kullanılmıştır. Daha sonrasında RAM_addr_id register’ının değeri “1” artırılarak, register’ın sahip olduğu değerdeki port_id’den B_reg register’ına çarpılacak B sayısı yüklenmiş olur. Çarpılacak iki sayı Picoblaze içerisinde yazıldıktan sonra, çarpıcı bloğunun A ve B girişlerinin olduğu port_id değerlerine A_reg ve B_reg değerleri yazılır. Bu yazma işlemleri sırasında, Picoblaze’in write_strobe çıkışı, çarpıcı bloğun Awrite ve Bwrite girişlerine bağlı olduğu için, çıkışa verilen A ve B sayıları, çarpıcı modül içerisinde doğru zamanda alınarak saklanmış olur. Daha sonra start_reg içerisindeki “0x01” değeri, çarpıcı bloğun start girişinin olduğu port_id değerinde çıkışa verilerek, çarpıcı bloğunda çarpma işlemi başlatılmış olur.

Çarpma işlemi devam ederken, program sürekli bir döngü halinde, done işaretinin geleceği port_id üzerinden, done_reg register’ına bir okuma yapar. Bu portta, girişin en önemsiz biti dışındaki bütün bitler her zaman sıfır, en önemsiz bit ise çarpıcı bloğundan gelen done sinyalidir. Eğer done işaretinin “1” durumu, bu giriş anında yakalanırsa, done_reg register’ının değeri “0x01” değeri ile karşılaştırılır ve eşitlik durumunda, çarpıcı bloğun çıkışındaki 16-bitlik Result değeri, en önemli 8 bitten başlamak üzere, result_15_8(s0) ve result_7_0(s1) register’larına kaydedilir. Böylece RAM’den alınan iki sayının, çarpma bloğuna gönderildikten sonra, çarpım sonucunun Picoblaze içerisindeki s0 ve s1 register’larına kaydedilme işlemi bitmiş olur. Bütün RAM elemanlarının çarpılması istendiğinden, RAM_addr_id register’ının değeri “0x7F” ile kontrol edilir. Bu sayede eğer RAM’deki bütün elemanlar çarpıcıya gönderildi ise, RAM_addr_id register’ı sıfırlanarak,

RAM'deki değerler en baştan çarpıcıya gönderilir. Eğer RAM_addr_id değeri son adres'e ulaşmamışsa, ardışık adreslerdeki değerler çarpılmaya devam edilir.



Şekil 9 – Picoblaze program akış şeması

Şekil 9'da görüldüğü üzere, özellikle giriş portları için 0x80, 0x81, 0x82 port_id değerleri kullanılmıştır. Çünkü, block RAM'in adres girişi 7-bitliktir. Yani adres girişi “0x00” ile “0x7F” arasındaki değerleri alabilmektedir. Picoblaze'in port_id çıkışının en önemsiz 7 biti, RAM'in adres girişine direkt olarak bağlandığında, az önce belirtilen iki değer arasındaki herhangi bir porttan RAM dışında bir değer alınmak istendiğinde, RAM'in o adresindeki değer Picoblaze tarafından okunamamış olur ve RAM'de erişilemeyen adresler ve değerler ortaya çıkar. Bu veri kaybına yol açmaktadır. Bu sebeple RAM'in adres aralığı dışındaki giriş portları, RAM dışındaki okumaların yapılacağı giriş portları olarak seçilmiştir.

Yazılan assembly programı, KCPSM3 assemblr kullanılarak, Picoblaze'in Program Memory modülü olarak, “PROJE.v” Verilog dosyası ile elde edilmiştir. Elde edilen Program Memory modülü, daha önceden oluşturulmuş Xilinx ISE projesine eklenmiştir. Daha sonra ise Picoblaze işlemcisinin Verilog dosyası olan “kcpsm3.v” dosyası da projeye eklenerek, Picoblaze'in tümü ile projeye eklenmesi sağlanmıştır. Yazılan assembly kodu **EK'tedir**.

6) SİSTEMİN EN ÜST MODÜLDE TOPLANARAK GERÇEKLENMESİ

Tasarlanan Booth çarpıcı modülü ve programlanmış Picoblaze işlemcisinin, Şekil 4'te görülen sistemdeki gibi birbirlerine bağlanması ve istenilen port_id'lerden veri alışverişi yapılabilmesi, ayrıca A ve B sayılarının çarpıcı bloğuna düzgün bir şekilde yazılabilmesi için MUX modüllerinin tasarımını, daha önceden oluşturulmuş Xilinx ISE projesi üzerinde gerçekleştirilmiştir. Ayrıca sistemde kullanılacak Block RAM, Core Generator kullanılarak projeye eklenmiştir.

```

1  module DEMUX (
2      input [7:0] in,
3      input [7:0] port_id,
4      output reg [7:0] cikis_A,
5      output reg [7:0] cikis_B,
6      output reg [7:0] cikis_start
7  );
8
9  always@*
10 begin
11     case(port_id)
12         8'h80: begin
13             cikis_A=in;
14             cikis_B=0;
15             cikis_start=0;
16         end
17         8'h81: begin
18             cikis_A=0;
19             cikis_B=in;
20             cikis_start=0;
21         end
22         8'h82: begin
23             cikis_A=0;
24             cikis_B=0;
25             cikis_start=in;
26         end
27         default: begin
28             cikis_A=0;
29             cikis_B=0;
30             cikis_start=0;
31         end
32     endcase
33 endmodule

```

Şekil 10 – Şekil 4'teki sistemdeki DEMUX modülü

Şekil 10'da görülen DEMUX modülü, Picoblaze'deki out_port çıkışının, uygun port_id değerlerine göre dağıtılmmasını sağlayan bir kombinazonsal devre elemanıdır. Görüldüğü üzere eğer DEMUX'un select girişi olan port_id değeri, "0x80", "0x81" ve "0x82" ise, daha önceden Picoblaze assembly kodunda belirtilmiş ve kararlaştırılmış olan portlara uygun çıkışlar verilir. Bu üç port_id değeri dışındaki değerlerde, DEMUX çıkışları her zaman sıfır olarak verilmektedir.

```

35  module DEMUX_strobe (
36    input in,
37    input [7:0] port_id,
38    output reg cikis_A_write,
39    output reg cikis_B_write
40  );
41
42  always@*
43  begin
44    case(port_id)
45      8'h80: begin
46        cikis_A_write=in;
47        cikis_B_write=0;
48      end
49      8'h81: begin
50        cikis_A_write=0;
51        cikis_B_write=in;
52      end
53      default: begin
54        cikis_A_write=0;
55        cikis_B_write=0;
56      end
57    endcase
58  end
59 endmodule
60
61

```

Şekil 11 – Şekil 4'teki sistemdeki DEMUX_strobe modülü

Şekil 11'de görülen DEMUX_strobe modülü, A ve B sayılarının, Picoblaze tarafından çarpıcı modülüne yazdırıldığı zaman, Picoblaze'in out_port'tan çıkış verdiği sırada, işaretin doğru bir şekilde alınabildiğini gösteren ve çıkıştaki işaretin o anda doğru olduğunu anlaşılmasını sağlayan write_strobe işaretinin; çarpıcı bloğundaki Awrite ve Bwrite girişlerine, ilgili port_id değerlerinde gönderilmesini sağlayan, kombinazonsal bir devredir. Bu modül sayesinde, Picoblaze'den, çarpıcı bloğunun A ya da B girişine bir değer yazıldığında, ortaya çıkan write_strobe işaretinden de yararlanılarak, doğru zamanda doğru A ve B değerleri çarpıcı bloğuna yazılmış olur. Böylece Picoblaze'de daha az komut kullanılarak, daha kısa bir sürede çarpıcı bloğuna yazma işlemleri tamamlanmış olur.

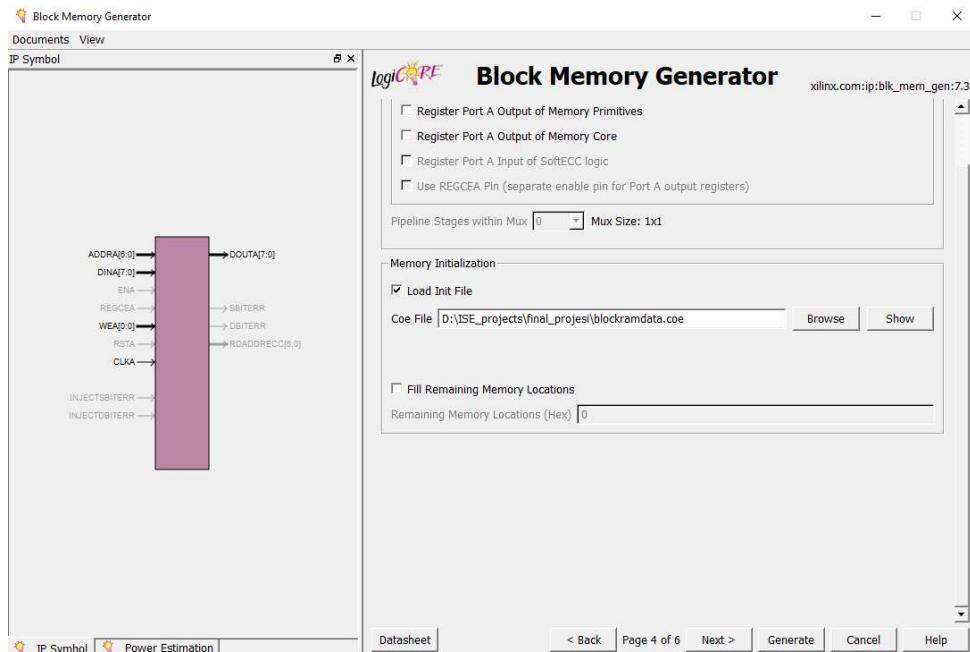
```

63  module MUX (
64    input [7:0] ram_giris,
65    input [7:0] result_msb_giris,
66    input [7:0] result_lsb_giris,
67    input [7:0] done_giris,
68    input [7:0] port_id,
69    output reg [7:0] out
70  );
71
72  always@*
73  begin
74    case(port_id)
75      8'h80: out=result_msb_giris;
76      8'h81: out=result_lsb_giris;
77      8'h82: out=done_giris;
78      default: out=ram_giris;
79    endcase
80  end
81
82 endmodule

```

Şekil 12 – Şekil 4'teki sistemdeki MUX modülü

Şekil 12'de görülen MUX modülü, Picoblaze'deki in_port çıkışına, uygun port_id değerlerinde uygun girişlerin gelmesini sağlayan bir kombinazonsal devre elemanıdır. Görüldüğü üzere eğer MUX'un select girişi olan port_id değeri, "0x80", "0x81" ve "0x82" ise, daha önceden Picoblaze assembly kodunda belirtilmiş ve kararlaştırılmış olan portlardan uygun girişler alınır. Bu üç port_id değeri dışındaki değerlerde, MUX çıkışına yani Picoblaze girişine, RAM'in çıkışı gelmektedir. Böyle bir tasarımın yapılmama sebebi, port_id değerlerinden "0x00" ile "0x7F" arası bütün değerlerde RAM'den okuma yapılmak istenmesi, sadece "0x80", "0x81" ve "0x82" portlarından RAM'in dışında okuma yapılmak istenmesidir. Böylece RAM'deki adres kontrolü için de ayrı bir MUX ya da DEMUX kullanımı gerekmeyez.



Şekil 13 – Şekil 4'teki sistemdeki Block RAM

Şekil 13'teki Block RAM, çarpılacak sayıların bulunduğu bir hafızadır. Block RAM tasarlanırken, 128*8-bitlik, single port, always enabled bir RAM tasarlanmıştır. RAM'de herhangi bir yazma ve okuma önceliği bulunmamaktadır. Başlangıç olarak, çarpılmak istenen rastgele sayılar ile oluşturulmuş "blockramdata.coe" isimli veri dosyası, RAM içerisinde aktarılmıştır.

Aşağıdaki şekilde Şekil 4'teki sistemi gerçekleyen en üst modülün Verilog kodu gösterilmiştir. Görüldüğü üzere, sistemdeki Picoblaze işlemci, program hafıza modülü, çarpıcı modülü, MUX, DEMUX ve DEMUX_strobe modülleri, uygun wire'lar kullanılarak birbirlerine bağlanılmışlardır. Block RAM'in write enable girişine "0" verilerek, içerisindeki verilerde değişim önlenmiştir. Ayrıca adres girişine, işlemcinin port_id çıkışının en önemsiz 7-bitli bağlanmıştır. Picoblaze işlemcisi bağlanırken, interrupt kullanılmayacağı için interrupt girişi toprağa bağlanmıştır. booth_top modülü (çarpıcı modül) start girişine, DEMUX'dan gelen 8-bitlik çıkışın en önemsiz biti bağlanmıştır. MUX'un, çarpıcı modülünden gelecek done işaretini iletmesi ise, ilgili MUX girişine {7'd0,done} bağlanarak, 8-bitlik bir işaret elde edilmiştir.

```

1   `timescale 1ns / 1ps
2
3   module UST_modul(
4     input clk,
5     input reset
6   );
7   wire [9:0] address;
8   wire [17:0] instruction;
9   wire [7:0] port_id;
10  wire write_strobe;
11  wire [7:0] out_port;
12  wire read_strobe;
13  wire [7:0] in_port;
14  wire interrupt_ack;
15  wire [7:0] A;
16  wire A_write;
17  wire [7:0] B;
18  wire B_write;
19  wire [7:0] start;
20  wire [15:0] Result;
21  wire done;
22  wire [7:0] ram_giris;
23
24  blockram_1 blockram (
25    .clka(clk), // input clka
26    .wea(1'b0), // input [0 : 0] wea
27    .addr(port_id[6:0]), // input [6 : 0] address
28    // .dina(dina), // input [7 : 0] dina
29    .douta(ram_giris) // output [7 : 0] douta
30  );
31
32  kcpsm3 picoblaze (
33    .address(address),
34    .instruction(instruction),
35    .port_id(port_id),
36    .write_strobe(write_strobe),
37    .out_port(out_port),
38    .read_strobe(read_strobe),
39    .in_port(in_port),
40    .interrupt(1'b0),
41    .interrupt_ack(interrupt_ack),
42    .reset(reset),
43    .clk(clk)
44  );
45
46  project prog_rom (
47    .address(address),
48    .instruction(instruction),
49    .clk(clk)
50  );
51  booth_top multiplier(
52    .A(A),
53    .Awrite(A_write),
54    .B(B),
55    .Bwrite(B_write),
56    .start(start[0]),
57    .clk(clk),
58    .done(done),
59    .Result(Result)
60  );
61  DEMUX demuxl (
62    .in(out_port),
63    .port_id(port_id),
64    .cikis_A(A),
65    .cikis_B(B),
66    .cikis_start(start)
67  );
68
69  DEMUX_strobe demux_strobe (
70    .in(write_strobe),
71    .port_id(port_id),
72    .cikis_A_write(A_write),
73    .cikis_B_write(B_write)
74  );
75
76  MUX muxl (
77    .ram_giris(ram_giris), //?
78    .result_msb_giris(Result[15:8]),
79    .result_lsb_giris(Result[7:0]),
80    .done_giris({7'd0,done}),
81    .port_id(port_id),
82    .out(in_port)
83  );
84
85 endmodule

```

Şekil 14 – UST modülü (Sistemin en üst modülü)

Tasarlanan UST modülünün, RTL şematiği EK'te gösterilmiştir.

7) SİSTEMİN BENZETİMİ

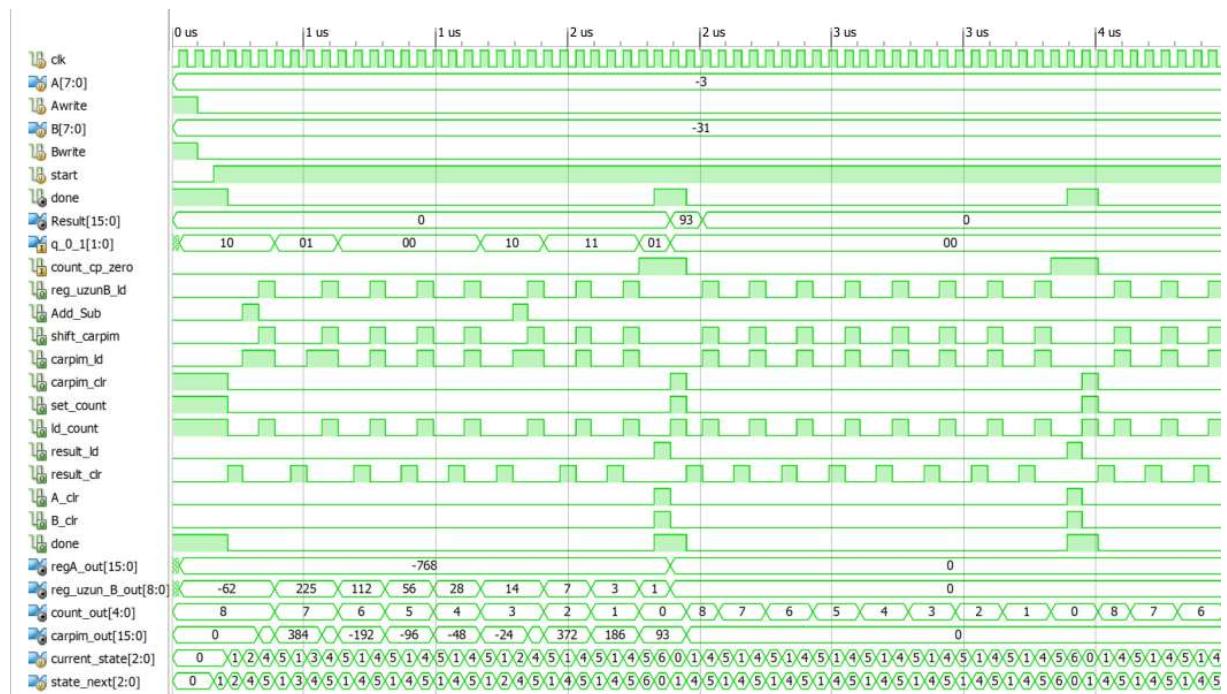
Tasarlanan çarpıcı bloğunun (Booth_top modülü) davranışsal benzetiminin yapılabilmesi için aşağıda görülen test bench kodu yazılmıştır.

```

1  `timescale 1ns / 1ps
2
3  module booth_tb;
4      // Inputs
5      reg [7:0] A;
6      reg Awrite;
7      reg [7:0] B;
8      reg Bwrite;
9      reg start;
10     reg clk;
11     // Outputs
12     wire done;
13     wire [15:0] Result;
14     // Instantiate the Unit Under Test (UUT)
15     booth_top uut (
16         .A(A),
17         .Awrite(Awrite),
18         .B(B),
19         .Bwrite(Bwrite),
20         .start(start),
21         .done(done),
22         .Result(Result),
23         .clk(clk)
24     );
25     initial begin
26         // Initialize Inputs
27         clk=0;
28         A = -8'd3;
29         Awrite = 1;
30         B = -8'd31;
31         Bwrite = 1;
32         start = 0;
33         // Wait 100 ns for global reset to finish
34         #100;
35         // Add stimulus here
36         Awrite = 0;
37         Bwrite = 0;
38         #60;
39         start=1;
40     end
41     always #30 clk=~clk;
42
43 endmodule

```

Şekil 15 – booth_top modülü test bench kodu



Şekil 16 – booth_top modülü benzetim sonucu

Şekil 16'dan görüldüğü üzere, çarpıcı girişine gelen ve girişlere yazılan $A=-3$, $B=-31$ sayıları, başarıyla çarpılarak sonuç 93 olarak bulunmuştur. Ayrıca sonuç çıkışa verilmeden bir clock cycle önce, done işaretini ="1" olmuşdur. Tasarımdan dolayı, sonuç alındıktan ve done işaretini yandıktan sonra, daha önceden tutulmuş A ve B sayıları silindiğinden, yeni sayılar çarpıcı bloğuna yazılmadığı için, start ="1" olduğundan sonraki bütün çarpma işlemlerinde sonuç "0" olarak elde edilmiştir.

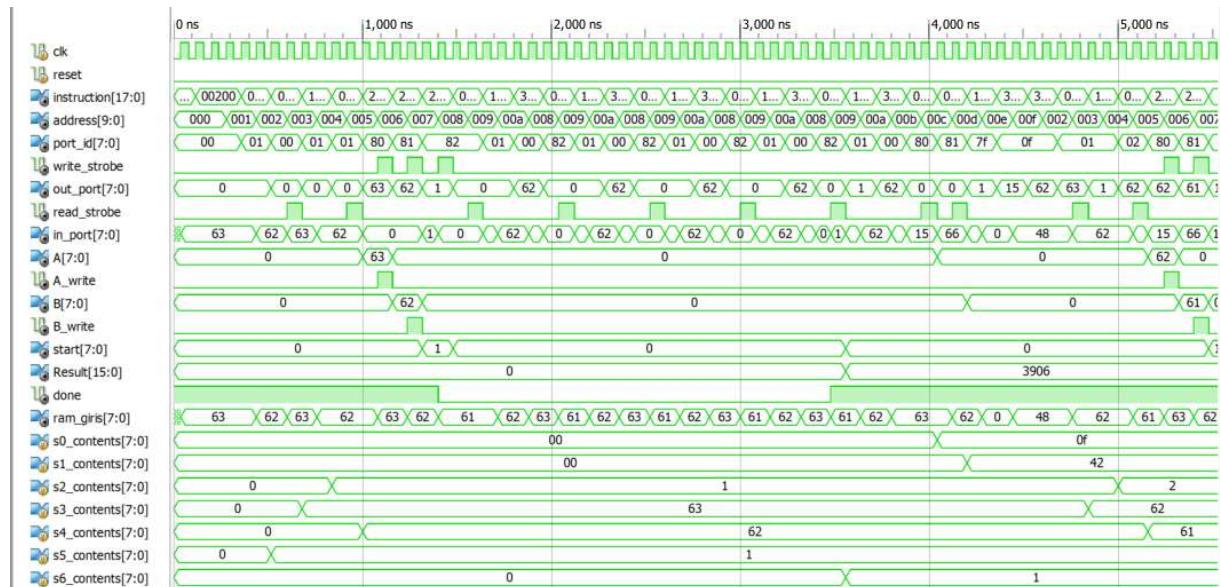
Tasarlanan sistemin (UST modulu) davranışsal benzetiminin yapılabilmesi için aşağıda görülen test bench kodu yazılmıştır.

```

1  `timescale 1ns / 1ps
2
3
4  module UST_tb;
5
6    // Inputs
7    reg clk;
8    reg reset;
9
10   // Instantiate the Unit Under Test (UUT)
11   UST_modul uut (
12     .clk(clk),
13     .reset(reset)
14   );
15
16   initial begin
17     // Initialize Inputs
18     clk = 0;
19     reset = 0;
20
21     // Wait 100 ns for global reset to finish
22     #100;
23
24     // Add stimulus here
25
26   end
27   always #40 clk=~clk;
28 endmodule
29

```

Şekil 17– UST modülü test bench kodu

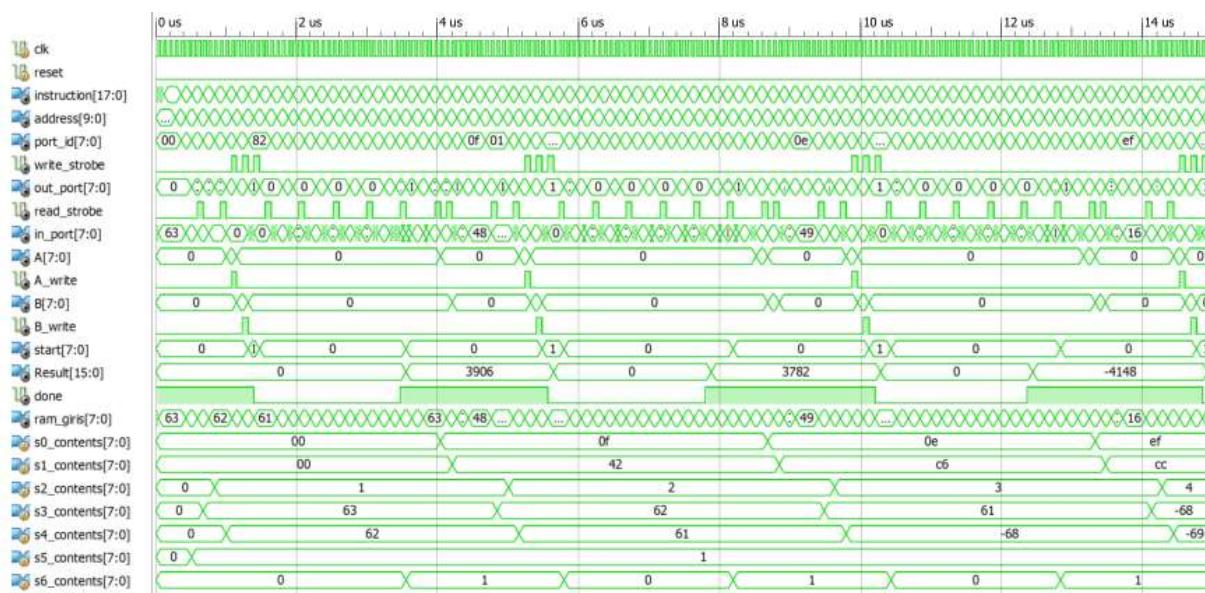


Şekil 18– UST modülü benzetim sonucu -1-

Şekil 18'deki benzetim sonucunda, A, B, out_port, in_port, s2_contents, s3_contents, s4_contents, s5_contents, s6_contents, ram_giris, result sinyalleri "signed decimal" olarak,

start sinyali “unsigned decimal” olarak, port_id, s0_contents, s1_contents, address sinyalleri “hexadecimal” olarak gözükmektedir.

Şekilden görüldüğü üzere, devre çalışmaya başladığında, RAM’ın 0. Adresinden ve 1. Adresinden, 1. Adresinden ve 2. Adresinden,... olmak üzere değerler okunarak, Picoblaze’deki s3 ve s4 register’larına kaydedilmişlerdir. Kaydedilen bu değerler, çarpıcı bloğu üzerindeki A ve B girişlerine verilmiştir, bu sırada oluşan write_strobe işaretlerinden dolayı, A ve B girişlerine verilen değerler çarpıcı bloğuna yazılmıştır. Start sinyali “0x01” yapıldıktan sonra, çarpıcı bloğu çarpma işlemi yapmaya başlamıştır. Bu sırada, read_strobe işaretlerinden anlaşılabileceği üzere, Picoblaze ilgili port_id’den done sinyalinin “1” olduğu durumu yakalamaya çalışmaktadır. Çarpım bloğu, çarpım sonucunu verdiğiinde, Picoblaze yanınan done sinyalini algılar ve çarpıcı bloğu çıkışındaki Result sonucunu, en önemli 8-bit s0 register’ına olmak üzere, s0 ve s1 register’larına yazar. Böylece bir döngüdeki Block RAM’den iki sayının şeşilmesi, çarpım işlemine sokulması ve çarpım sonucunun Picoblaze içerisinde yazılması işlemleri tamamlanmış olur. Sonraki adımlarda ise s2 register’ındaki RAM adres değerine göre, RAM’ın farklı adreslerindeki değerler ile çarpma işlemi yapılmasına devam edilir. RAM’deki son adres olan “0x7F”e gelindiğten sonra, RAM’ın tekrardan “0x00” adresinden itibaren okunmaya başlanması sağlanır.



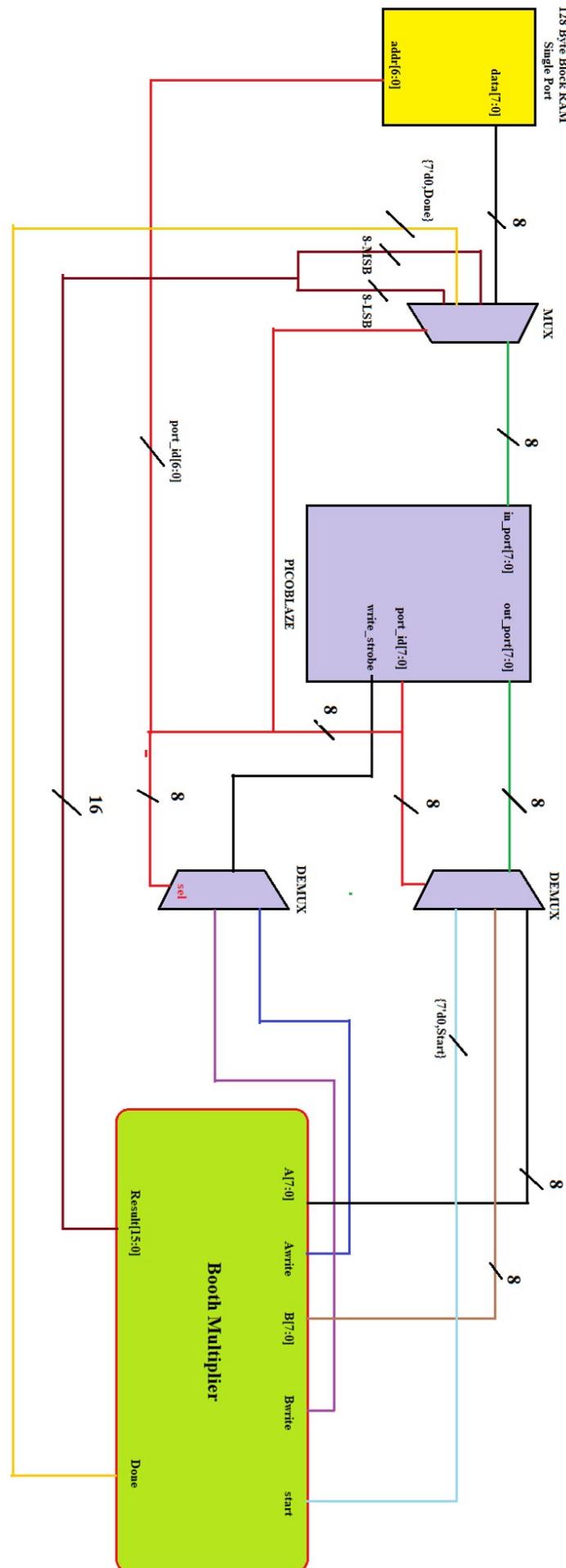
Şekil 19– UST modülü benzetim sonucu -2-

8) REFERANSLAR

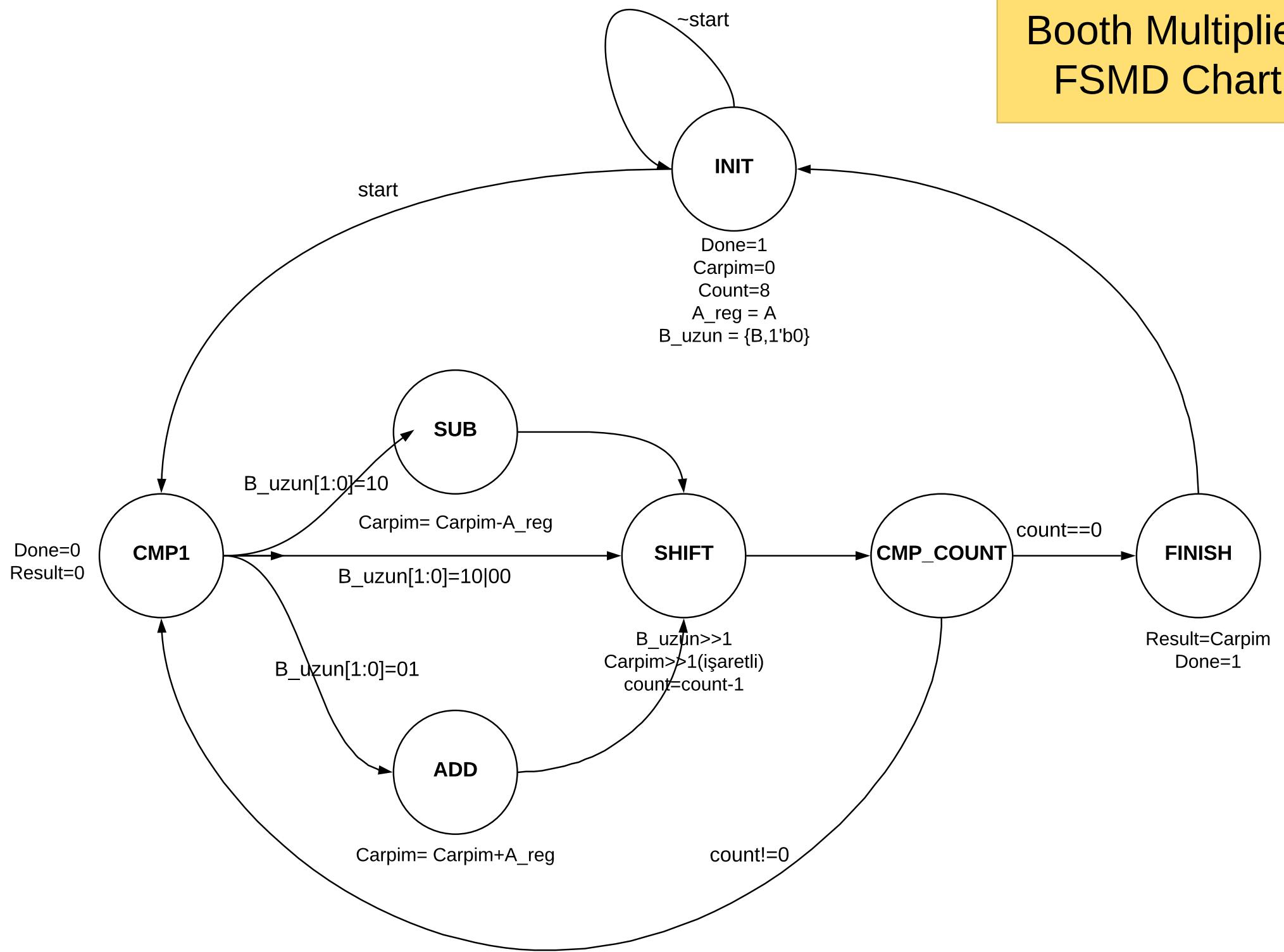
- [1] - <http://talhakum.com/2017-06-02-booth-algorithm/>

EKLER

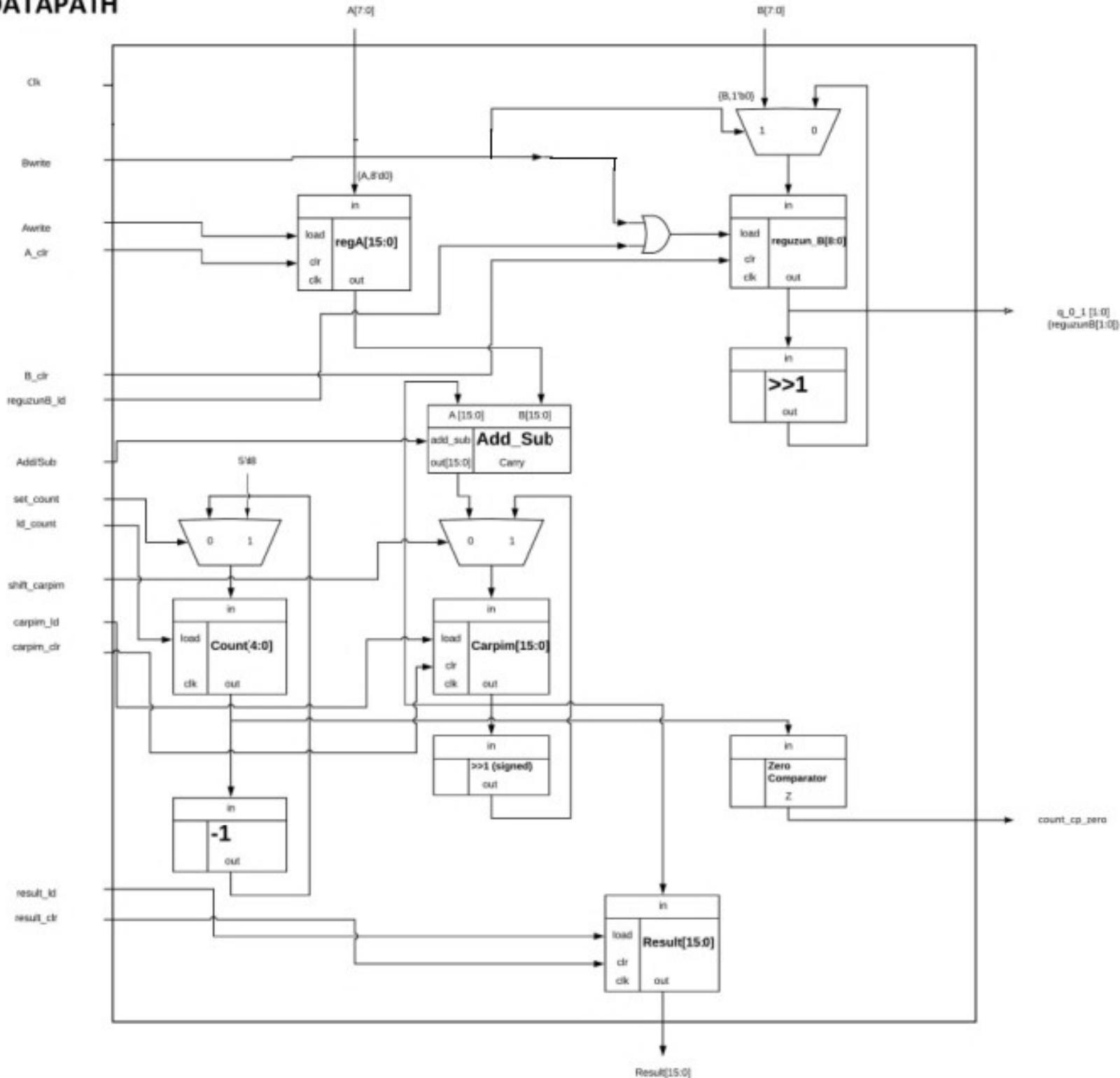
Kodlar, Semalar, Benzetim Sonuçları



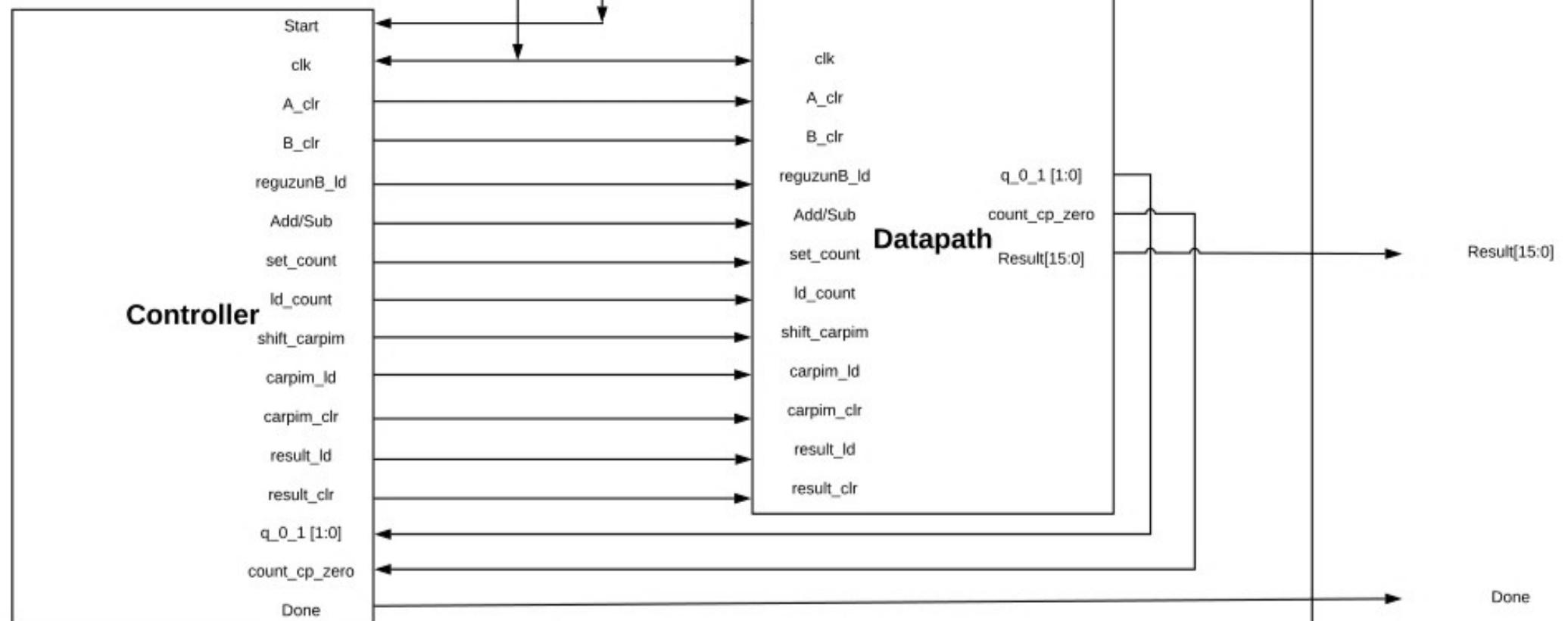
Booth Multiplier FSMD Chart



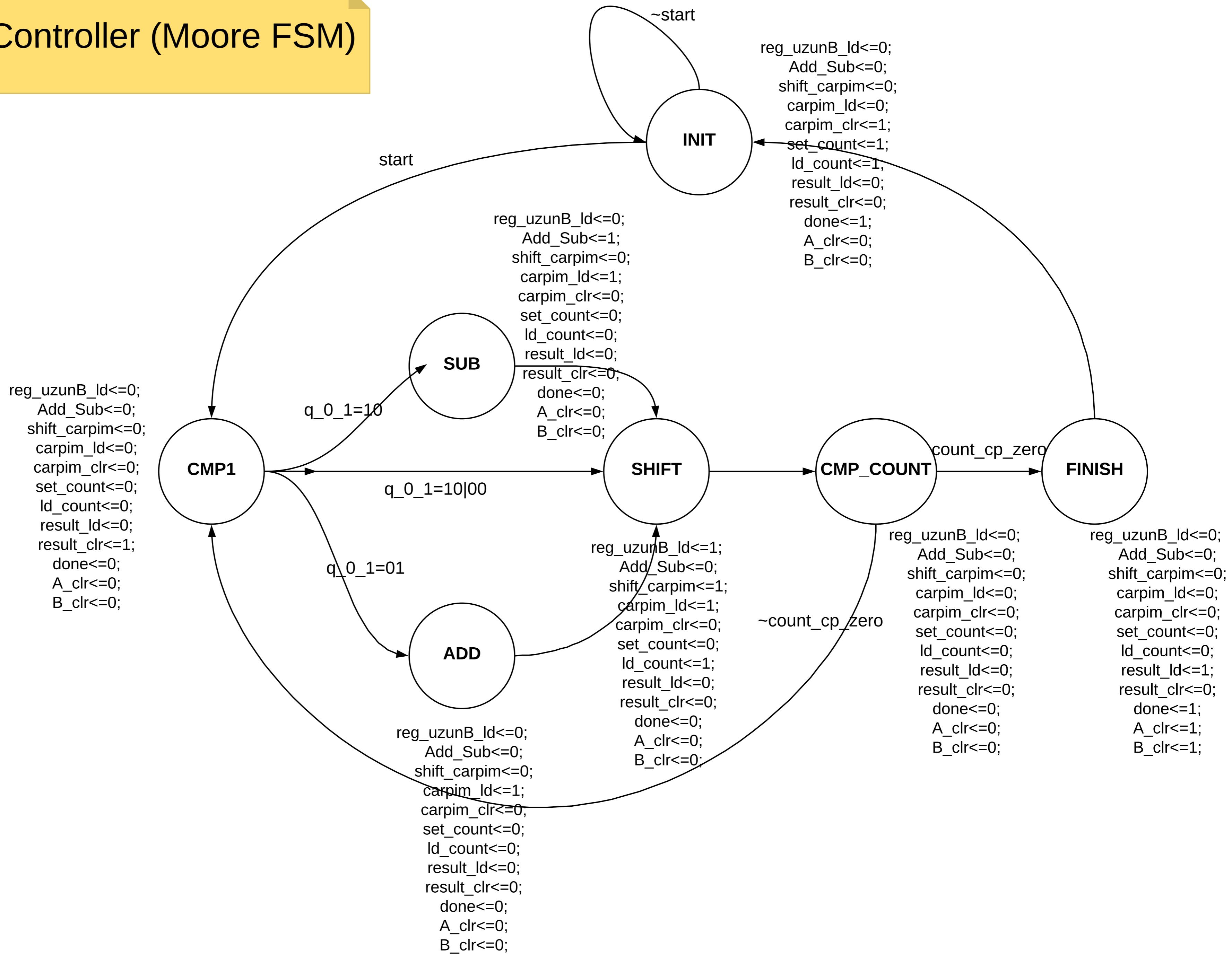
DATAPATH



BOOTH MULTIPLIER



Booth Multiplier Controller (Moore FSM)



```
1 `timescale 1ns / 1ps
2
3 module datapath(
4     input [7:0] A,
5     input [7:0] B,
6     input clk,
7     input Awrite,
8     input Bwrite,
9     input reg_uzunB_ld,
10    input Add_Sub,
11    input shift_carpim,
12    input carpim_ld,
13    input carpim_clr,
14    input set_count,
15    input ld_count,
16    input result_ld,
17    input result_clr,
18    input A_clr,
19    input B_clr,
20    output [1:0] q_0_1,
21    output count_cp_zero,
22    output [15:0] Result
23 );
24
25 wire [15:0] carpim_out, shift_carry_out, mux_16_bit_out, add_sub_out, regA_out;
26 wire [8:0] mux_9_bit_out, shift_9_bit_out, reg_uzun_B_out;
27 wire [4:0] mux_5_bit_out, count_out, sub_1_out;
28 wire reg_uzun_B_ld_wire;
29 wire C;
30
31 assign reg_uzun_B_ld_wire = /*(~start)&*/Bwrite|reg_uzunB_ld;
32 assign q_0_1 = reg_uzun_B_out[1:0];
33
34 reg_A reg_a1 (
35     .clk(clk),
36     .in({A,8'd0}),
37     .load(Awrite),
38     .clr(A_clr),
39     .out(regA_out)
40 );
41
42 reg_uzunB reg_uzunB1 (
43     .clk(clk),
44     .in(mux_9_bit_out),
45     .load(reg_uzun_B_ld_wire),
46     .clr(B_clr),
47     .out(reg_uzun_B_out)
48 );
49
50 carpim carpim1 (
51     .clk(clk),
52     .in(mux_16_bit_out),
53     .load(carpim_ld),
54     .clr(carpim_clr),
55     .out(carpim_out)
56 );
57
```

```

booth_multiplier.v
58     count count1 (
59         .clk(clk),
60         .in(mux_5_bit_out),
61         .load(ld_count),
62         .out(count_out)
63     );
64
65
66     result result1 (
67         .clk(clk),
68         .in(carpim_out),
69         .clr(result_clr),
70         .load(result_ld),
71         .out(Result)
72     );
73
74
75     mux_2_1_16bit mux0 (
76         .sifir(add_sub_out),
77         .bir(shift_carry_out),
78         .sel(shift_carpim),
79         .out(mux_16_bit_out)
80     );
81
82
83     mux_2_1_5bit mux1 (
84         .sifir(sub_1_out),
85         .bir(5'd8),
86         .sel(set_count),
87         .out(mux_5_bit_out)
88     );
89
90     mux_2_1_9bit mux2 (
91         .sifir(shift_9_bit_out),
92         .bir({B,1'b0}),
93         .sel(Bwrite),
94         .out(mux_9_bit_out)
95     );
96
97
98     add_sub_mod add_sub_1 (
99         .A(carpim_out),
100        .B(regA_out),
101        .add_sub(Add_Sub),
102        .C(C),
103        .out(add_sub_out)
104    );
105
106    right_shift shift0 (
107        .in(reg_uzun_B_out),
108        .out(shift_9_bit_out)
109    );
110
111    right_shift_carry shift1 (
112        .in(carpim_out),
113        .out(shift_carry_out)
114    );

```

```
115
116     sub_1 sub2 (
117         .in(count_out),
118         .out(sub_1_out)
119     );
120
121     zero_comparator zero_1 (
122         .in(count_out),
123         .Z(count_cp_zero)
124     );
125
126
127
128
129 endmodule
130
131 // registerlar
132 module reg_A (
133     input clk,
134     input [15:0] in,
135     input load,
136     input clr,
137     output [15:0] out
138 );
139
140     reg [15:0] register_t;
141     initial begin
142         register_t<=16'd0;
143     end
144     always@(posedge clk)
145     begin
146         if(clr)
147             register_t<=16'd0;
148         else if(load)
149             register_t<=in;
150         end
151
152     assign out=register_t;
153 endmodule
154
155
156 module reg_uzunB (
157     input clk,
158     input [8:0] in,
159     input load,
160     input clr,
161     output [8:0] out
162 );
163
164     reg [8:0] register_t;
165     initial begin
166         register_t<=9'd0;
167     end
168     always@(posedge clk)
169     begin
170         if(clr)
```

```
172         register_t<=9'd0;
173     else if(load)
174         register_t<=in;
175     end
176
177     assign out=register_t;
178 endmodule
179
180 module carpim (
181     input clk,
182     input [15:0] in,
183     input load,
184     input clr,
185     output [15:0] out
186
187 );
188
189     reg [15:0] register_t;
190     initial begin
191         register_t<=16'd0;
192     end
193     always@(posedge clk)
194     begin
195         if(clr)
196             register_t<=16'd0;
197         else if(load)
198             register_t<=in;
199     end
200
201     assign out=register_t;
202 endmodule
203
204 module count (
205     input clk,
206     input [4:0] in,
207     input load,
208     output [4:0] out
209
210 );
211
212     reg [4:0] register_t;
213     initial begin
214         register_t<=5'd8;
215     end
216     always@(posedge clk)
217     begin
218         if(load)
219             register_t<=in;
220     end
221
222     assign out=register_t;
223 endmodule
224
225
226 module result (
227     input clk,
228     input [15:0] in,
```

```
229     input clr,
230     input load,
231     output [15:0] out
232
233 );
234
235     reg [15:0] register_t;
236     initial begin
237         register_t<=16'd0;
238     end
239     always@(posedge clk)
240     begin
241         if(clr)
242             register_t<=0;
243         else if(load)
244             register_t<=in;
245     end
246
247     assign out=register_t;
248 endmodule
// işlem blokları
250
251 module right_shift (
252     input [8:0] in,
253     output [8:0] out
254 );
255
256     assign out = {1'b0, in[8:1]};
257
258 endmodule
259
260 module add_sub_mod (
261     input [15:0] A,
262     input [15:0] B,
263     input add_sub,
264     output C,
265     output [15:0] out
266 );
267     // add_sub 0 ise toplama, 1 ise çıkarma
268     assign {C,out} = (~add_sub) ? (A+B) : (A-B);
269 endmodule
270
271 module right_shift_carry (
272     input [15:0] in,
273     output [15:0] out
274 );
275
276     assign out = {in[15], in[15:1]};
277
278 endmodule
279
280 module sub_1 (
281     input [4:0] in,
282     output [4:0] out
283 );
284
285 
```

```

booth_multiplier.v
286     assign out = in-5'd1;
287 endmodule
288
289 module zero_comparator (
290     input [4:0] in,
291     output z
292 // count 0 ise z=1
293 );
294
295     assign z = ~(|in);
296 endmodule
297
298 module mux_2_1_9bit (
299     input [8:0] sifir,
300     input [8:0] bir,
301     input sel,
302     output reg [8:0] out
303 );
304
305     always @*
306     begin
307         case(sel)
308             1'd0: out=sifir;
309             1'd1: out=bir;
310         endcase
311     end
312
313 endmodule
314
315 module mux_2_1_16bit (
316     input [15:0] sifir,
317     input [15:0] bir,
318     input sel,
319     output reg [15:0] out
320 );
321
322     always @*
323     begin
324         case(sel)
325             1'd0: out=sifir;
326             1'd1: out=bir;
327         endcase
328     end
329
330 endmodule
331
332 module mux_2_1_5bit (
333     input [4:0] sifir,
334     input [4:0] bir,
335     input sel,
336     output reg [4:0] out
337 );
338
339     always @*
340     begin
341         case(sel)
342             1'd0: out=sifir;

```

```
343      1'd1: out=bir;
344      endcase
345  end
346
347 endmodule
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
```

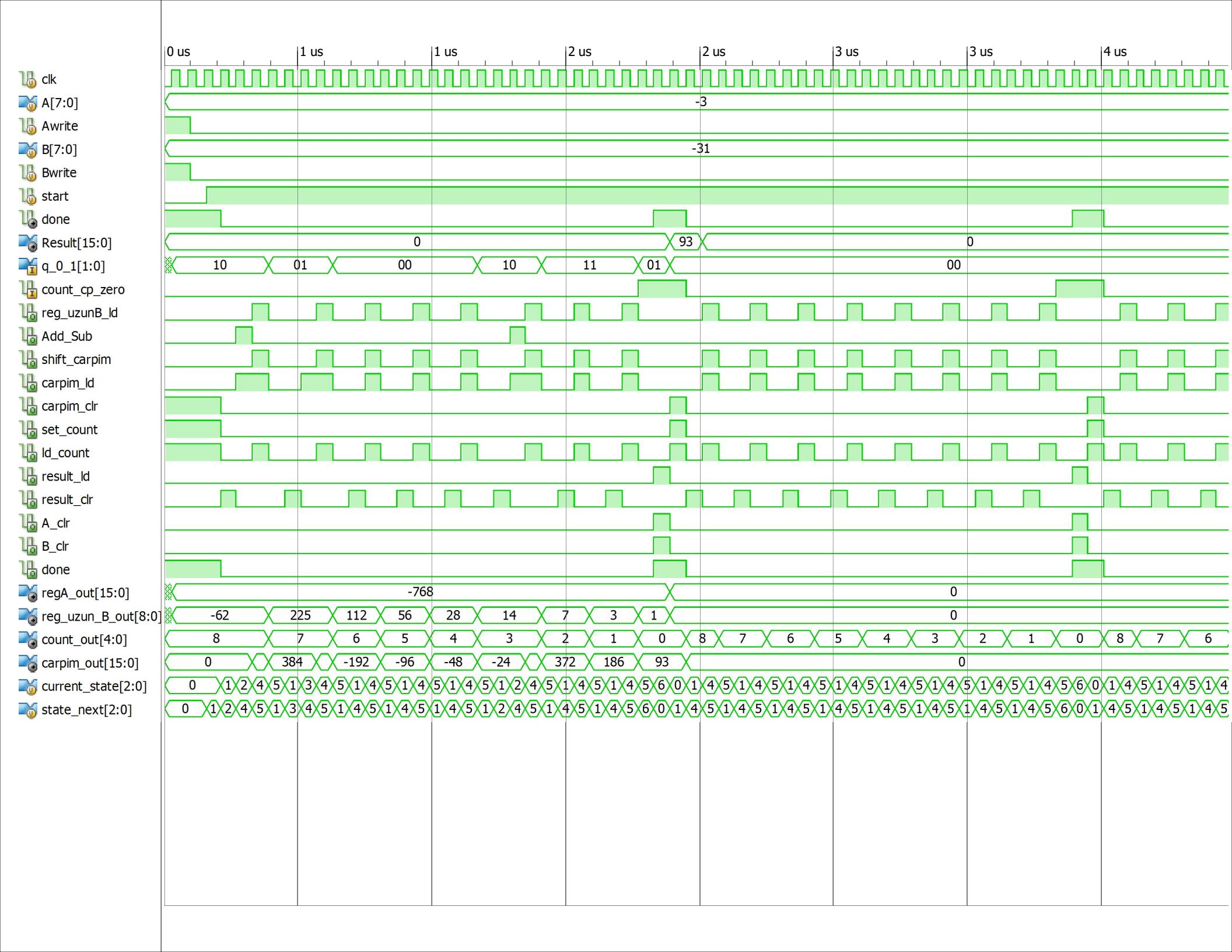
```
1 `timescale 1ns / 1ps
2
3 module controller(
4
5     input start,
6     input clk,
7     input [1:0] q_0_1,
8     input count_cp_zero,
9     output reg reg_uzunB_ld,
10    output reg Add_Sub,
11    output reg shift_carpim,
12    output reg carpim_ld,
13    output reg carpim_clr,
14    output reg set_count,
15    output reg ld_count,
16    output reg result_ld,
17    output reg result_clr,
18    output reg A_clr,
19    output reg B_clr,
20    output reg done
21
22 );
23
24     reg [2:0] current_state, state_next;
25
26     localparam INIT = 3'd0,
27             CMP1 = 3'd1,
28             SUB = 3'd2,
29             ADD = 3'd3,
30             SHIFT = 3'd4,
31             CMP_COUNT = 3'd5,
32             FINISH = 3'd6;
33
34
35     initial
36     begin
37         current_state<=INIT;
38     end
39
40     //state register
41     always@(posedge clk)
42     begin
43         current_state<=state_next;
44     end
45
46     //next_state_logic and output_logic
47     always@*
48     begin
49         state_next<=current_state;
50         case(current_state)
51             INIT : begin
52                 reg_uzunB_ld<=0;
53                 Add_Sub<=0;
54                 shift_carpim<=0;
55                 carpim_ld<=0;
56                 carpim_clr<=1;
57                 set_count<=1;
```

```
58          ld_count<=1;
59          result_ld<=0;
60          result_clr<=0;
61          done<=1;
62          A_clr<=0;
63          B_clr<=0;
64          if(start)
65              state_next<=CMP1;
66          else
67              state_next<=INIT;
68      end
69  CMP1 : begin
70      reg_uzunB_ld<=0;
71      Add_Sub<=0;
72      shift_carpim<=0;
73      carpim_ld<=0;
74      carpim_clr<=0;
75      set_count<=0;
76      ld_count<=0;
77      result_ld<=0;
78      result_clr<=1;
79      done<=0;
80      A_clr<=0;
81      B_clr<=0;
82      case(q_0_1)
83          2'b10: state_next<=SUB;
84          2'b01: state_next<=ADD;
85          2'b11: state_next<=SHIFT;
86          2'b00: state_next<=SHIFT;
87      endcase
88  end
89  SUB : begin
90      reg_uzunB_ld<=0;
91      Add_Sub<=1;
92      shift_carpim<=0;
93      carpim_ld<=1;
94      carpim_clr<=0;
95      set_count<=0;
96      ld_count<=0;
97      result_ld<=0;
98      result_clr<=0;
99      done<=0;
100     A_clr<=0;
101     B_clr<=0;
102     state_next<=SHIFT;
103  end
104 ADD : begin
105     reg_uzunB_ld<=0;
106     Add_Sub<=0;
107     shift_carpim<=0;
108     carpim_ld<=1;
109     carpim_clr<=0;
110     set_count<=0;
111     ld_count<=0;
112     result_ld<=0;
113     result_clr<=0;
114     done<=0;
```

```
115          A_clr<=0;
116          B_clr<=0;
117          state_next<=SHIFT;
118      end
119  SHIFT : begin
120      reg_uzunB_ld<=1;
121      Add_Sub<=0;
122      shift_carpim<=1;
123      carpim_ld<=1;
124      carpim_clr<=0;
125      set_count<=0;
126      ld_count<=1;
127      result_ld<=0;
128      result_clr<=0;
129      done<=0;
130      A_clr<=0;
131      B_clr<=0;
132      state_next<=CMP_COUNT;
133  end
134  CMP_COUNT : begin
135      reg_uzunB_ld<=0;
136      Add_Sub<=0;
137      shift_carpim<=0;
138      carpim_ld<=0;
139      carpim_clr<=0;
140      set_count<=0;
141      ld_count<=0;
142      result_ld<=0;
143      result_clr<=0;
144      done<=0;
145      A_clr<=0;
146      B_clr<=0;
147      if(count_cp_zero)
148          state_next<=FINISH;
149      else
150          state_next<=CMP1;
151  end
152  FINISH : begin
153      reg_uzunB_ld<=0;
154      Add_Sub<=0;
155      shift_carpim<=0;
156      carpim_ld<=0;
157      carpim_clr<=0;
158      set_count<=0;
159      ld_count<=0;
160      result_ld<=1;
161      result_clr<=0;
162      done<=1;
163      A_clr<=1;
164      B_clr<=1;
165      state_next<=INIT;
166  end
167      endcase
168  end
169 endmodule
170
```

```
1 `timescale 1ns / 1ps
2
3 module booth_top(
4     input [7:0] A,
5     input Awrite,
6     input [7:0] B,
7     input Bwrite,
8     input start,
9     input clk,
10    output done,
11    output [15:0] Result
12 );
13
14     wire [1:0] q_0_1;
15     wire count_cp_zero;
16     wire reg_uzunB_ld;
17     wire Add_Sub;
18     wire shift_carpim;
19     wire carpim_ld;
20     wire carpim_clr;
21     wire set_count;
22     wire ld_count;
23     wire result_ld;
24     wire result_clr;
25     wire A_clr;
26     wire B_clr;
27
28 controller cul (
29     .start(start),
30     .clk(clk),
31     .q_0_1(q_0_1),
32     .count_cp_zero(count_cp_zero),
33     .reg_uzunB_ld(reg_uzunB_ld),
34     .Add_Sub(Add_Sub),
35     .shift_carpim(shift_carpim),
36     .carpim_ld(carpim_ld),
37     .carpim_clr(carpim_clr),
38     .set_count(set_count),
39     .ld_count(ld_count),
40     .result_ld(result_ld),
41     .result_clr(result_clr),
42     .done(done),
43     .A_clr(A_clr),
44     .B_clr(B_clr)
45 );
46
47 datapath dat1 (
48     .A(A),
49     .B(B),
50     .clk(clk),
51     .Awrite(Awrite),
52     .Bwrite(Bwrite),
53     .reg_uzunB_ld(reg_uzunB_ld),
54     .Add_Sub(Add_Sub),
55     .shift_carpim(shift_carpim),
56     .carpim_ld(carpim_ld),
57     .carpim_clr(carpim_clr),
```

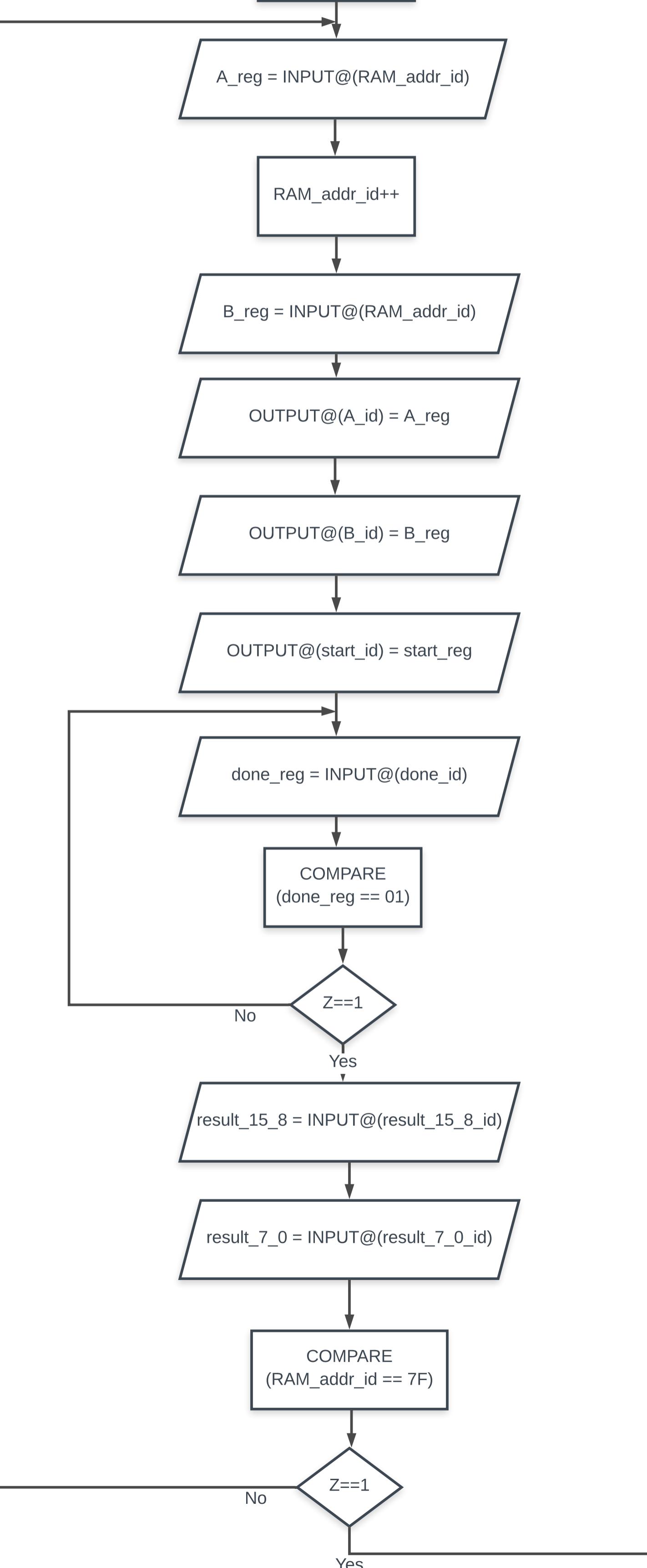
```
58      .set_count(set_count),
59      .ld_count(ld_count),
60      .result_ld(result_ld),
61      .result_clr(result_clr),
62      .q_0_1(q_0_1),
63      .count_cp_zero(count_cp_zero),
64      .Result(Result),
65      .A_clr(A_clr),
66      .B_clr(B_clr)
67  );
68 endmodule
69
```



Register İsimleri
result_15_8 = s0
result_7_0 = s1
RAM_addr_id = s2
A_reg = s3
B_reg = s4
start_reg = s5
done_reg = s6

Giriş ID'leri
result_15_8_id = 0x80
result_7_0_id = 0x81
done_id = 0x82

Çıkış ID'leri
A_id = 0x80
B_id = 0x81
start_id = 0x82



```

1 ;*****
2 ; Ceyhun Yamaneren - 040140010
3 ; Int. to Embedded Systems dersi Final Projesi, Picoblaze KCMP3 standartı
4 ; assembly kodu
5 ; Booth algoritması ile çarpma işlemi projesi
6 ;*****
7 ;
8 ;giriş id'leri
9 CONSTANT result_15_8_id, 80
10 CONSTANT result_7_0_id, 81
11 CONSTANT done_id, 82
12 ;
13 ;cıkış id_leri
14 ;
15 CONSTANT A_id, 80
16 CONSTANT B_id, 81
17 CONSTANT start_id, 82
18 ;
19 ;register isimleri
20 ;
21 NAMEREG s2, RAM_addr_id
22 NAMEREG s0, result_15_8
23 NAMEREG s1, result_7_0
24 NAMEREG s3, A_reg
25 NAMEREG s4, B_reg
26 NAMEREG s5, start_reg
27 NAMEREG s6, done_reg
28 ;
29 ; Program
30 ;
31 main:      LOAD RAM_addr_id, 00
32           LOAD start_reg, 01
33 read_RAM:   INPUT A_reg, (RAM_addr_id)
34           ADD RAM_addr_id, 01
35           INPUT B_reg, (RAM_addr_id)
36           OUTPUT A_reg, A_id
37           OUTPUT B_reg, B_id
38           OUTPUT start_reg, start_id
39 wait_done:  INPUT done_reg, done_id
40           COMPARE done_reg, 01
41           JUMP NZ, wait_done
42 take_resu:  INPUT result_15_8, result_15_8_id
43           INPUT result_7_0, result_7_0_id
44           COMPARE RAM_addr_id, 7F
45           JUMP Z, main
46           JUMP NZ, read_RAM
47
48

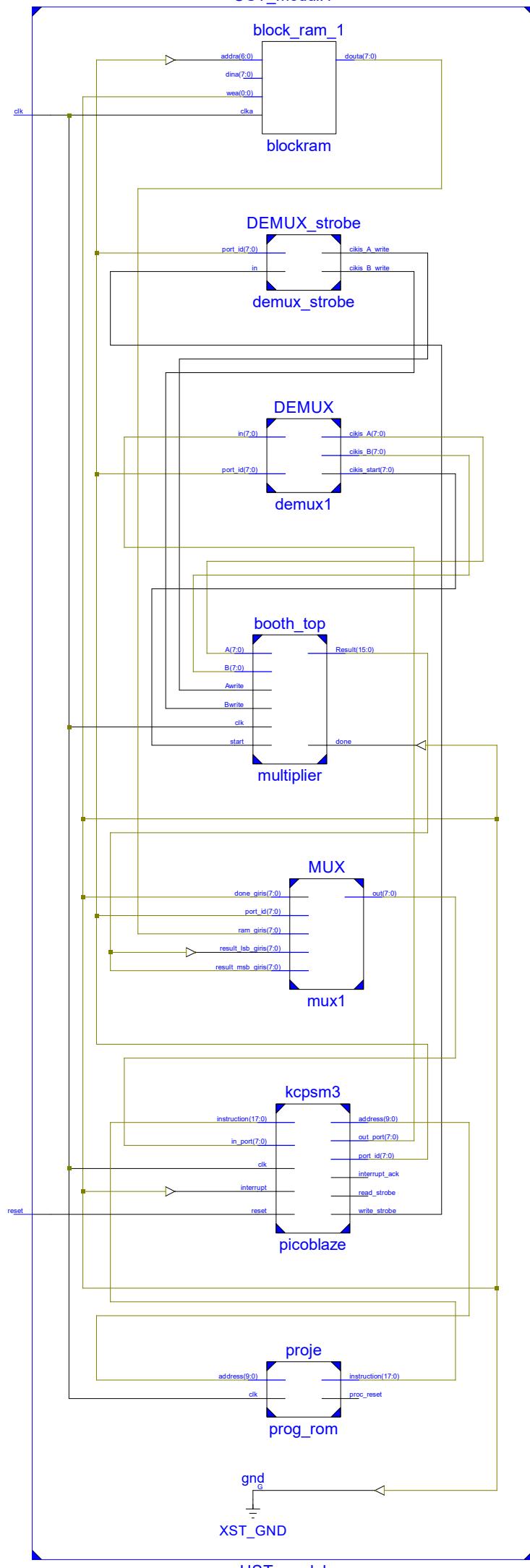
```

```
1 `timescale 1ns / 1ps
2
3 module UST_modul(
4     input clk,
5     input reset
6 );
7 wire [9:0] address;
8 wire [17:0] instruction;
9 wire [7:0] port_id;
10 wire write_strobe;
11 wire [7:0] out_port;
12 wire read_strobe;
13 wire [7:0] in_port;
14 wire interrupt_ack;
15 wire [7:0] A;
16 wire A_write;
17 wire [7:0] B;
18 wire B_write;
19 wire [7:0] start;
20 wire [15:0] Result;
21 wire done;
22 wire [7:0] ram_giris;
23
24 block_ram_1 blockram (
25     .clka(clk), // input clka
26     .wea(1'b0), // input [0 : 0] wea
27     .addrb(port_id[6:0]), // input [6 : 0] addra
28     // .dina(dina), // input [7 : 0] dina
29     .doutb(ram_giris) // output [7 : 0] douta
30 );
31
32 kcsm3 picoblaze (
33     .address(address),
34     .instruction(instruction),
35     .port_id(port_id),
36     .write_strobe(write_strobe),
37     .out_port(out_port),
38     .read_strobe(read_strobe),
39     .in_port(in_port),
40     .interrupt(1'b0),
41     .interrupt_ack(interrupt_ack),
42     .reset(reset),
43     .clk(clk)
44 );
45
46 proje prog_rom (
47     .address(address),
48     .instruction(instruction),
49     .clk(clk)
50 );
51
52 booth_top multiplier(
53     .A(A),
54     .Awrite(A_write),
55     .B(B),
56     .Bwrite(B_write),
57     .start(start[0]),
```

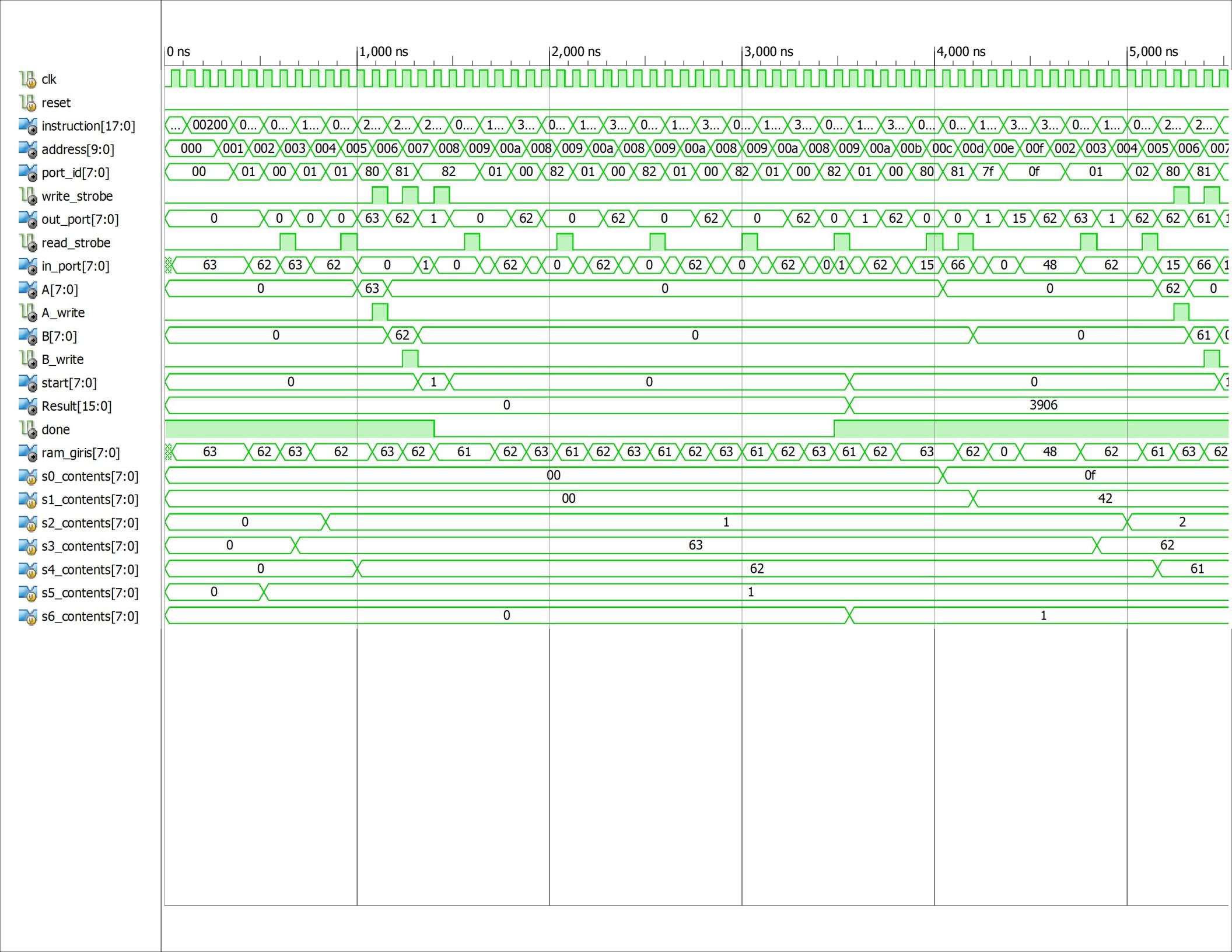
```
58      .clk(clk),
59      .done(done),
60      .Result(Result)
61  );
62
63 DEMUX demux1 (
64     .in(out_port),
65     .port_id(port_id),
66     .cikis_A(A),
67     .cikis_B(B),
68     .cikis_start(start)
69 );
70
71 DEMUX_strobe demux_strobe (
72     .in(write_strobe),
73     .port_id(port_id),
74     .cikis_A_write(A_write),
75     .cikis_B_write(B_write)
76 );
77
78 MUX mux1 (
79     .ram_giris(ram_giris), //?
80     .result_msb_giris(Result[15:8]),
81     .result_lsb_giris(Result[7:0]),
82     .done_giris({7'd0,done}),
83     .port_id(port_id),
84     .out(in_port)
85 );
86
87 endmodule
88
89 module DEMUX (
90     input [7:0] in,
91     input [7:0] port_id,
92     output reg [7:0] cikis_A,
93     output reg [7:0] cikis_B,
94     output reg [7:0] cikis_start
95 );
96     always@*
97     begin
98         case(port_id)
99             8'h80: begin
100                 cikis_A=in;
101                 cikis_B=0;
102                 cikis_start=0;
103             end
104             8'h81: begin
105                 cikis_A=0;
106                 cikis_B=in;
107                 cikis_start=0;
108             end
109             8'h82: begin
110                 cikis_A=0;
111                 cikis_B=0;
112                 cikis_start=in;
113             end
114             default: begin
```

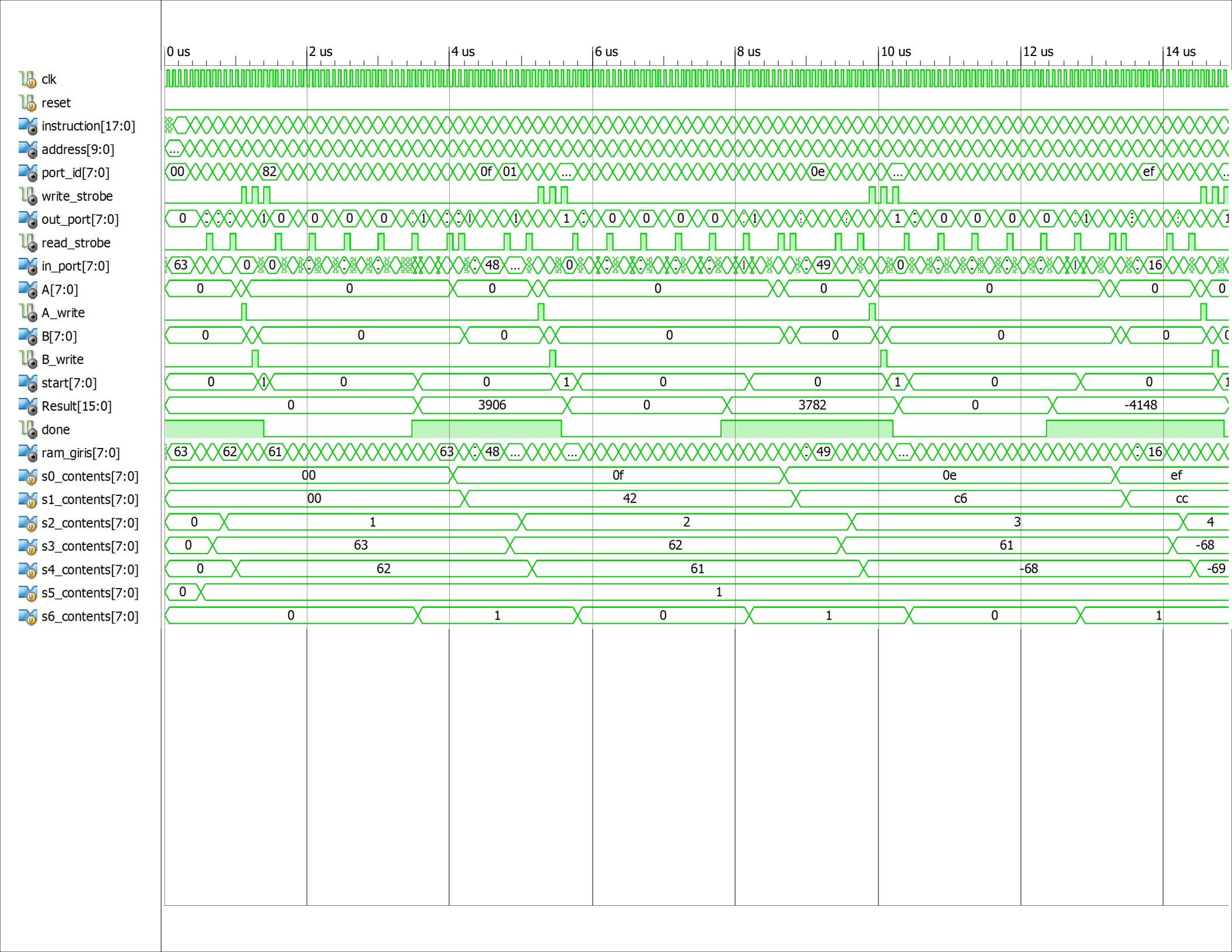
```
115                     cikis_A=0;
116                     cikis_B=0;
117                     cikis_start=0;
118                 end
119             endcase
120         end
121     endmodule
122
123 module DEMUX_strobe (
124     input in,
125     input [7:0] port_id,
126     output reg cikis_A_write,
127     output reg cikis_B_write
128 );
129     always@*
130     begin
131         case(port_id)
132             8'h80: begin
133                 cikis_A_write=in;
134                 cikis_B_write=0;
135             end
136             8'h81: begin
137                 cikis_A_write=0;
138                 cikis_B_write=in;
139             end
140             end
141         default: begin
142             cikis_A_write=0;
143             cikis_B_write=0;
144         end
145         end
146     endcase
147 end
148 endmodule
149
150
151 module MUX (
152     input [7:0] ram_giris,
153     input [7:0] result_msb_giris,
154     input [7:0] result_lsb_giris,
155     input [7:0] done_giris,
156     input [7:0] port_id,
157     output reg [7:0] out
158 );
159     always@*
160     begin
161         case(port_id)
162             8'h80: out=result_msb_giris;
163             8'h81: out=result_lsb_giris;
164             8'h82: out=done_giris;
165             default: out=ram_giris;
166         endcase
167     end
168
169 endmodule
170
```

UST_modul1



UST_modul





```
1  `timescale 1ns / 1ps
2
3
4  module booth_tb;
5
6      // Inputs
7      reg [7:0] A;
8      reg Awrite;
9      reg [7:0] B;
10     reg Bwrite;
11     reg start;
12     reg clk;
13
14     // Outputs
15     wire done;
16     wire [15:0] Result;
17
18     // Instantiate the Unit Under Test (UUT)
19     booth_top uut (
20         .A(A),
21         .Awrite(Awrite),
22         .B(B),
23         .Bwrite(Bwrite),
24         .start(start),
25         .done(done),
26         .Result(Result),
27         .clk(clk)
28     );
29
30     initial begin
31         // Initialize Inputs
32         clk=0;
33         A = -8'd3;
34         Awrite = 1;
35         B = -8'd31;
36         Bwrite = 1;
37         start = 0;
38
39         // Wait 100 ns for global reset to finish
40         #100;
41
42         // Add stimulus here
43         Awrite = 0;
44         Bwrite = 0;
45         #60;
46         start=1;
47     end
48
49     always #30 clk=~clk;
50
51 endmodule
52
53
```

```
1 `timescale 1ns / 1ps
2
3
4 module UST_tb;
5
6     // Inputs
7     reg clk;
8     reg reset;
9
10    // Instantiate the Unit Under Test (UUT)
11    UST_modul uut (
12        .clk(clk),
13        .reset(reset)
14    );
15
16    initial begin
17        // Initialize Inputs
18        clk = 0;
19        reset = 0;
20
21        // Wait 100 ns for global reset to finish
22        #100;
23
24        // Add stimulus here
25
26    end
27    always #40 clk=~clk;
28 endmodule
29
30
```

```
1 ;pico2 blockram data file FBG
2 memory_initialization_radix=2;
3 memory_initialization_vector=
4 00111111,
5 00111110,
6 00111101,
7 10111100,
8 10111011,
9 00111010,
10 00111001,
11 10111000,
12 00110111,
13 00110110,
14 10110101,
15 00110100,
16 00110011,
17 00110010,
18 00110001,
19 00110000,
20 00101111,
21 00101110,
22 00101101,
23 00101100,
24 00101011,
25 00101010,
26 00101001,
27 00101000,
28 00100111,
29 00100110,
30 00100101,
31 00100100,
32 00100011,
33 00100010,
34 00100001,
35 00100000,
36 00011111,
37 00011110,
38 00011101,
39 00011100,
40 00011011,
41 00011010,
42 00011001,
43 00011000,
44 00010111,
45 00010110,
46 00010101,
47 00010100,
48 00010011,
49 00010010,
50 00010001,
51 00010000,
52 00001111,
53 00001110,
54 00001101,
55 00001100,
56 00001011,
57 00001010,
58 00001001,
59 00001000,
60 00000111,
61 00000110,
62 00000101,
63 00000100,
64 00000011,
65 00000010,
66 00000001,
67 00000000,
68 00111111,
69 00111110,
70 00111101,
71 00111100,
72 00111011,
73 00111010,
```

74 00111001,
75 00111000,
76 00110111,
77 00110110,
78 00110101,
79 00110100,
80 00110011,
81 00110010,
82 00110001,
83 00110000,
84 00101111,
85 00101110,
86 00101101,
87 00101100,
88 00101011,
89 00101010,
90 00101001,
91 00101000,
92 00100111,
93 00100110,
94 00100101,
95 00100100,
96 00100011,
97 00100010,
98 00100001,
99 00100000,
100 00011111,
101 00011110,
102 00011101,
103 00011100,
104 00011011,
105 00011010,
106 00011001,
107 00011000,
108 00010111,
109 00010110,
110 00010101,
111 00010100,
112 00010011,
113 00010010,
114 00010001,
115 00010000,
116 00001111,
117 00001110,
118 00001101,
119 00001100,
120 00001011,
121 00001010,
122 00001001,
123 00001000,
124 00000111,
125 00000110,
126 00000101,
127 00000100,
128 00000011,
129 00000010,
130 00000001,
131 00000000;
132