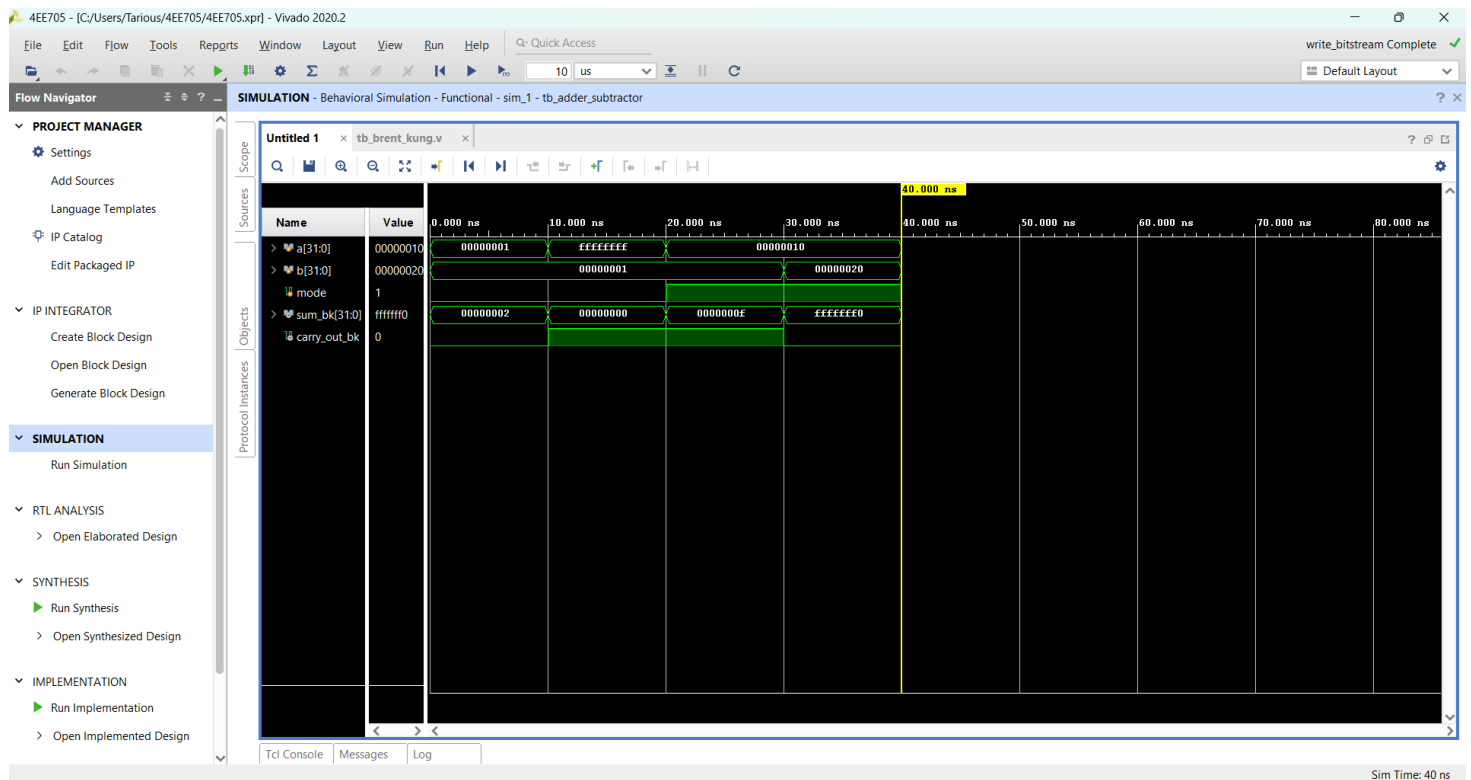


Design a 32-bit Brent-Kung adder. For testing the adder, you will be using BRAM, VIO, ILA and Control unit (which can read the data from BRAM and send it to your ADDER module).

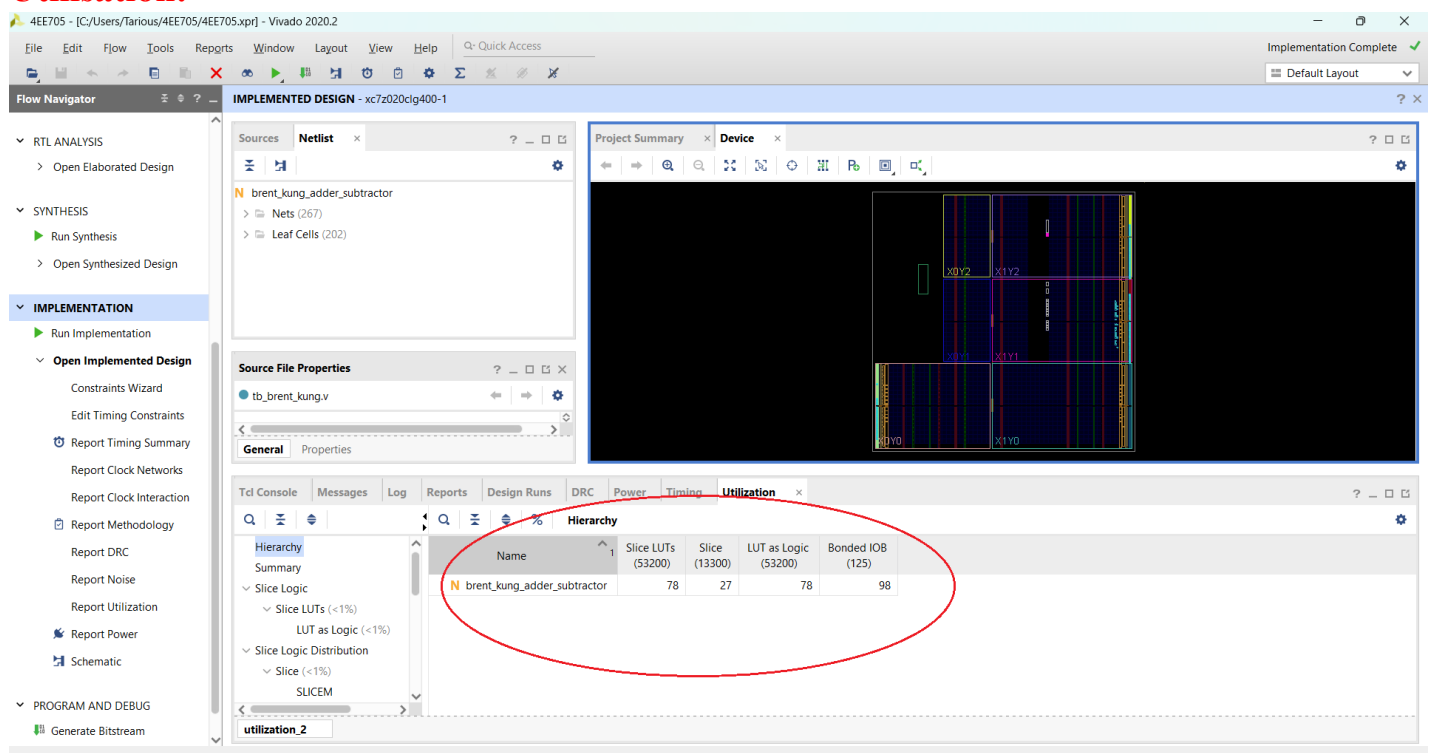
Task-1 - Design a 32-bit Adder/Subtractor and write a testbench and simulate the circuit.

Design Using Brent-Kung Adder

Simulation Waveform Using Brent Kung Adder



Utilisation:



Resource Utilisation:

Utilization - Synth Design - synth_1

C:/Users/Tarious/4EE705/4EE705.runs/synth_1/brent_kung_adder_subtractor_utilization_synth.rpt

Read-only

1

Copyright 1986-2020 Xilinx, Inc. All Rights Reserved.

2

3

Tool Version : Vivado v.2020.2 (win64) Build 3064766 Wed Nov 18 09:12:45 MST 2020

4

Date : Fri Jan 17 18:31:45 2025

5

Host : LAPTOP-M216NI01 running 64-bit major release (build 9200)

6

Command : report_utilization -file brent_kung_adder_subtractor_utilization_synth.rpt -pb brent_kung_adder_subtractor_utilization_synth.pb

7

Design : brent_kung_adder_subtractor

8

Device : 7z020clg400-1

9

Design State : Synthesized

10

11

Utilization Design Information

12

13

Table of Contents

14

15

1. Slice Logic

16

1.1 Summary of Registers by Type

17

2. Memory

18

3. DSP

19

4. IO and GT Specific

20

5. Clocking

21

6. Specific Feature

22

7. Primitives

23

8. Black Boxes

24

9. Instantiated Netlists

25

26

1. Slice Logic

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

Utilization - Synth Design - synth_1

C:/Users/Tarious/4EE705/4EE705.runs/synth_1/brent_kung_adder_subtractor_utilization_synth.rpt

Read-only

45

1.1 Summary of Registers by Type

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

2. Memory

65

66

67

68

69

70

71

72

73

74

* Note: Each Block RAM Tile only has one FIFO logic available and therefore can accommodate only one FIFO36E1 or one FIFO18E1. However, if a FIFO18E1 occupies a Block RAM Tile, that tile can still accomm

75

76

77

3. DSP

78

79

80

81

82

83

84

85

Utilization - Synth Design - synth_1

C:/Users/Tarious/4EE705/4EE705.runs/synth_1/brent_kung_adder_subtractor_utilization_synth.rpt

Read-only

4. IO and GT Specific

Site Type	Used	Fixed	Available	Util%
Bonded IOB	98	0	125	78.40
Bonded IPADs	0	0	2	0.00
Bonded IOPADs	0	0	130	0.00
PHY_CONTROL	0	0	4	0.00
PHASER_REF	0	0	4	0.00
OUT_FIFO	0	0	16	0.00
IN_FIFO	0	0	16	0.00
IDELAYCTRL	0	0	4	0.00
IBUFDS	0	0	121	0.00
PHASER_OUT/PHASER_OUT_PHY	0	0	16	0.00
PHASER_IN/PHASER_IN_PHY	0	0	16	0.00
IDELAYE2/IDELAYE2_FINEDELAY	0	0	200	0.00
ILOGIC	0	0	125	0.00
OLOGIC	0	0	125	0.00

5. Clocking

Site Type	Used	Fixed	Available	Util%
BUFGCTRL	0	0	32	0.00
BUFG	0	0	16	0.00
MMCME2_ADV	0	0	4	0.00
PLL2_ADV	0	0	4	0.00
BUFMCE	0	0	8	0.00
BUFMCE	0	0	72	0.00
BUFR	0	0	16	0.00

6. Specific Feature

Utilization - Synth Design - synth_1

C:/Users/Tarious/4EE705/4EE705.runs/synth_1/brent_kung_adder_subtractor_utilization_synth.rpt

Read-only

6. Specific Feature

Site Type	Used	Fixed	Available	Util%
BSCAN2	0	0	4	0.00
CAPTURE2	0	0	1	0.00
DMA_PORT	0	0	1	0.00
EFUSE_USR	0	0	1	0.00
FRAME_ECCE2	0	0	1	0.00
ICAPE2	0	0	2	0.00
STARTUPE2	0	0	1	0.00
XADC	0	0	1	0.00

7. Primitives

Ref Name	Used	Functional Category
IBUF	65	IO
LUT6	50	LUT
OBUF	33	IO
LUT2	26	LUT
LUT3	14	LUT
LUT4	12	LUT
LUT5	2	LUT

8. Black Boxes

Ref Name	Used
----------	------

9. Instantiated Netlists

Ref Name	Used
----------	------

Utilization - Synth Design - synth_1

C:/Users/Tarious/4EE705/4EE705.runs/synth_1/brent_kung_adder_subtractor_utilization_synth.rpt

Read-only

135	EFUSE_USR	0	0	1	0.00
136	FRAME_FOC2	0	0	1	0.00
137	ICAPE2	0	0	2	0.00
138	STARTUPE2	0	0	1	0.00
139	XADC	0	0	1	0.00

140

141

142

143 7. Primitives

144

145

146	Ref Name	Used	Functional Category
149	IBUF	65	IO
150	LUT6	50	LUT
151	OBUF	33	IO
152	LUT2	26	LUT
153	LUT3	14	LUT
154	LUT4	12	LUT
155	LUT5	2	LUT

156

157

158

159 8. Black Boxes

160

161

162	Ref Name	Used
-----	----------	------

163

164

165

166

167 9. Instantiated Netlists

168

169

170	Ref Name	Used
-----	----------	------

171

172

173

174

175

Verilog Code:

// 32-bit Brent-Kung Adder/Subtractor Implementation

```

module brent_kung_adder_subtractor(
    input [31:0] a,
    input [31:0] b,
    input mode, // mode = 0 for addition, mode = 1 for subtraction
    output [31:0] sum,
    output carry_out
);
    wire [31:0] b_xor_mode;
    wire carry_in;
    assign b_xor_mode = b ^ {32{mode}}; // XOR for 2's complement subtraction
    assign carry_in = mode;
    wire [31:0] p, g, c;
    // Generate propagate (p) and generate (g) signals
    assign p = a ^ b_xor_mode;
    assign g = a & b_xor_mode;

```

// Brent-Kung Prefix Tree for Carry Generation

```
genvar i;  
generate  
  for (i = 0; i < 32; i = i + 1) begin  
    if (i == 0) assign c[i] = g[i] | (p[i] & carry_in);  
    else assign c[i] = g[i] | (p[i] & c[i-1]);  
  end  
endgenerate
```

// Compute Sum

```
assign sum = p ^ {c[30:0], carry_in};  
assign carry_out = c[31];  
endmodule
```

Test Bench Code:**// Testbench for 32-bit Brent-Kung Adder/Subtractor**

```
module tb_adder_subtractor;
```

```
  reg [31:0] a, b;
```

```
  reg mode;
```

```
  wire [31:0] sum_bk;
```

```
  wire carry_out_bk;
```

// Instantiate Module

```
  brent_kung_adder_subtractor uut1 (
```

```
    .a(a),
```

```
    .b(b),
```

```
    .mode(mode),
```

```
    .sum(sum_bk),
```

```
    .carry_out(carry_out_bk)
```

```
  );
```

initial begin

// Test Cases

```
$display("Time\tMode\tA\tB\tSum_BK\tCarry_BK");
```

```
$monitor("%0t\t%b\t%h\t%h\t%h\t%b", $time, mode, a, b, sum_bk, carry_out_bk);
```

```
a = 32'h00000001; b = 32'h00000001; mode = 0; #10; // Addition
```

```
a = 32'hFFFFFFFF; b = 32'h00000001; mode = 0; #10; // Addition with overflow
```

```
a = 32'h00000010; b = 32'h00000001; mode = 1; #10; // Subtraction
```

```
a = 32'h00000010; b = 32'h00000020; mode = 1; #10; // Subtraction with negative result
```

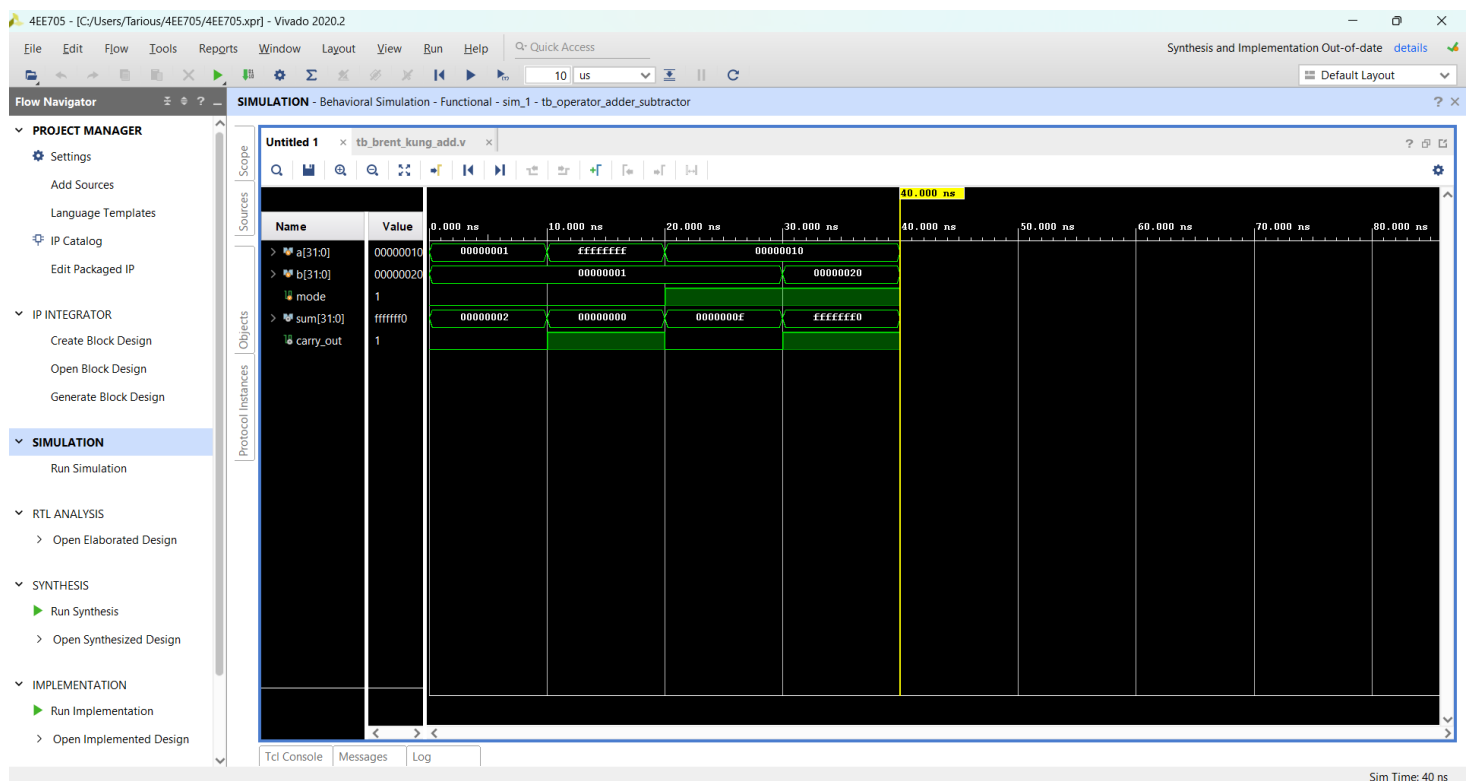
```
$finish;
```

end

endmodule

Using + operator only

Simulation Waveform Using + Operator Only



Utilisation:

The screenshot shows the Vivado 2020.2 interface with the 'Utilization - Synth Design - synth_1' report open. The report is titled 'Hierarchy' and shows the utilization of the 'operator_adder_subtractor' design. The utilization is summarized in the following table:

Name	Slice LUTs (53200)	Slice (13300)	LUT as Logic (53200)	Bonded IOB (125)
operator_adder_subtractor	31	9	31	98

The report also includes a 'Table of Contents' section with the following items:

- 1. Slice Logic
 - 1.1 Summary of Registers by Type
- 2. Memory
- 3. DSP
- 4. IO and GT Specific
- 5. Clocking
- 6. Specific Feature
- 7. Primitives
- 8. Black Boxes
- 9. Instantiated Netlists

Resource Utilisation:

The screenshot shows the 'Utilization - Synth Design - synth_1' report with the 'Table of Contents' section expanded. The report shows the utilization of resources for the 'operator_adder_subtractor' design. The utilization is summarized in the following table:

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	32	0	53200	0.06
LUT as Logic	32	0	53200	0.06
LUT as Memory	0	0	17400	0.00
Slice Registers	0	0	106400	0.00
Register as Flip Flop	0	0	106400	0.00
Register as Latch	0	0	106400	0.00
F7 Muxes	0	0	26600	0.00
F8 Muxes	0	0	13300	0.00

```

Utilization - Synth Design - synth_1
C:/Users/Tarious/4EE705/4EE705.runs/synth_1/operator_adder_subtractor_utilization_synth.rpt

40 | P8 Muxes | 0 | 0 | 13300 | 0.00 |
41 |-----+-----+
42 | * Warning! The Final LUT count, after physical optimizations and full implementation, is typically lower. Run opt_design after synthesis, if not already completed, for a more realistic count.
43 |
44 |
45 | 1.1 Summary of Registers by Type
46 |-----+-----+
47 |
48 | Total | Clock Enable | Synchronous | Asynchronous |
49 |-----+-----+-----+-----+
50 |
51 | 0 | - | - | - |
52 | 0 | - | - | Set |
53 | 0 | - | - | Reset |
54 | 0 | - | Set | - |
55 | 0 | - | Reset | - |
56 | 0 | Yes | - | - |
57 | 0 | Yes | - | Set |
58 | 0 | Yes | - | Reset |
59 | 0 | Yes | Set | - |
60 | 0 | Yes | Reset | - |
61 |-----+-----+-----+-----+
62 |
63 |
64 | 2. Memory
65 |-----+-----+
66 |
67 | Site Type | Used | Fixed | Available | Util% |
68 |-----+-----+-----+-----+-----+
69 |
70 | Block RAM Tile | 0 | 0 | 140 | 0.00 |
71 | RAMB36/FIFO* | 0 | 0 | 140 | 0.00 |
72 | RAMB18 | 0 | 0 | 280 | 0.00 |
73 |-----+-----+-----+-----+
74 | * Note: Each Block RAM Tile only has one FIFO logic available and therefore can accommodate only one FIFO36E1 or one FIFO18E1. However, if a FIFO18E1 occupies a Block RAM Tile, that tile can still accom
75 |
76 |
77 | 3. DSP
78 |-----+-----+
79 |
80 |-----+-----+-----+-----+

```

```

Utilization - Synth Design - synth_1
C:/Users/Tarious/4EE705/4EE705.runs/synth_1/operator_adder_subtractor_utilization_synth.rpt

76 |
77 | 3. DSP
78 |-----+-----+
79 |
80 |-----+-----+-----+-----+
81 | Site Type | Used | Fixed | Available | Util% |
82 |-----+-----+-----+-----+-----+
83 | DSPs | 0 | 0 | 220 | 0.00 |
84 |-----+-----+-----+-----+
85 |
86 |
87 | 4. IO and GT Specific
88 |-----+-----+
89 |
90 |-----+-----+-----+-----+
91 | Site Type | Used | Fixed | Available | Util% |
92 |-----+-----+-----+-----+-----+
93 | Bonded IOB | 98 | 0 | 125 | 78.40 |
94 | Bonded IPADs | 0 | 0 | 2 | 0.00 |
95 | Bonded IOPADs | 0 | 0 | 130 | 0.00 |
96 | PHY_CONTROL | 0 | 0 | 4 | 0.00 |
97 | PHASER_REF | 0 | 0 | 4 | 0.00 |
98 | OUT_FIFO | 0 | 0 | 16 | 0.00 |
99 | IN_FIFO | 0 | 0 | 16 | 0.00 |
100 | IDELAYCTRL | 0 | 0 | 4 | 0.00 |
101 | IBUFDS | 0 | 0 | 121 | 0.00 |
102 | PHASER_OUT/PHASER_OUT_PHY | 0 | 0 | 16 | 0.00 |
103 | PHASER_IN/PHASER_IN_PHY | 0 | 0 | 16 | 0.00 |
104 | IDELAY2/IDELAY2_FINEDELAY | 0 | 0 | 200 | 0.00 |
105 | ILOGIC | 0 | 0 | 125 | 0.00 |
106 | OLOGIC | 0 | 0 | 125 | 0.00 |
107 |-----+-----+-----+-----+
108 |
109 |
110 | 5. Clocking
111 |-----+-----+
112 |
113 |-----+-----+-----+-----+
114 | Site Type | Used | Fixed | Available | Util% |
115 |-----+-----+-----+-----+-----+
116 | BUFGCTRL | 0 | 0 | 32 | 0.00 |

```



```
Utilization - Synth Design - synth_1
C:/Users/Tarious/4EE705/4EE705.runs/synth_1/operator_adder_subtractor_utilization_synth.rpt

133 | CAPTUREE2 | 0 | 0 | 1 | 0.00 |
134 | DNA_PORT | 0 | 0 | 1 | 0.00 |
135 | EFUSE_USR | 0 | 0 | 1 | 0.00 |
136 | FRAME_ECCE2 | 0 | 0 | 1 | 0.00 |
137 | ICAPE2 | 0 | 0 | 2 | 0.00 |
138 | STARTUPE2 | 0 | 0 | 1 | 0.00 |
139 | XADC | 0 | 0 | 1 | 0.00 |
140 | -----+-----+-----+-----+-----+
141 |
142 |
143 | 7. Primitives
144 | -----
145 |
146 | +-----+-----+-----+-----+
147 | | Ref Name | Used | Functional Category |
148 | +-----+-----+-----+-----+
149 | | IBUF | 65 | IO |
150 | | OBUF | 33 | IO |
151 | | LUT3 | 31 | LUT |
152 | | CARRY4 | 9 | CarryLogic |
153 | | LUT1 | 1 | LUT |
154 | +-----+-----+-----+-----+
155 |
156 |
157 | 8. Black Boxes
158 | -----
159 |
160 | +-----+-----+
161 | | Ref Name | Used |
162 | +-----+-----+
163 |
164 |
165 | 9. Instantiated Netlists
166 | -----
167 |
168 | +-----+-----+
169 | | Ref Name | Used |
170 | +-----+-----+
171 |
172 |
173 |
```

Verilog Code:**// 32-bit Adder/Subtractor using '+' Operator**

```
module operator_adder_subtractor(  
    input [31:0] a,  
    input [31:0] b,  
    input mode, // mode = 0 for addition, mode = 1 for subtraction  
    output [31:0] sum,  
    output carry_out  
);  
    assign {carry_out, sum} = mode ? a - b : a + b;  
endmodule
```

Test Bench Code:**// Testbench for 32-bit Adder/Subtractor using '+' Operator**

```
module tb_operator_adder_subtractor;  
    reg [31:0] a, b;  
    reg mode;  
    wire [31:0] sum;  
    wire carry_out;  
  
    // Instantiate Module  
    operator_adder_subtractor uut (  
        .a(a),  
        .b(b),  
        .mode(mode),  
        .sum(sum),  
        .carry_out(carry_out)  
    );  
  
    initial begin
```

// Test Cases

```
$display("Time\tMode\tA\tB\tSum\tCarry");
```

```
$monitor("%0t\t%b\t%h\t%h\t%h\t%b", $time, mode, a, b, sum, carry_out);
```

```
a = 32'h00000001; b = 32'h00000001; mode = 0; #10; // Addition
```

```
a = 32'hFFFFFFFF; b = 32'h00000001; mode = 0; #10; // Addition with overflow
```

```
a = 32'h00000010; b = 32'h00000001; mode = 1; #10; // Subtraction
```

```
a = 32'h00000010; b = 32'h00000020; mode = 1; #10; // Subtraction with negative result
```

```
$finish;
```

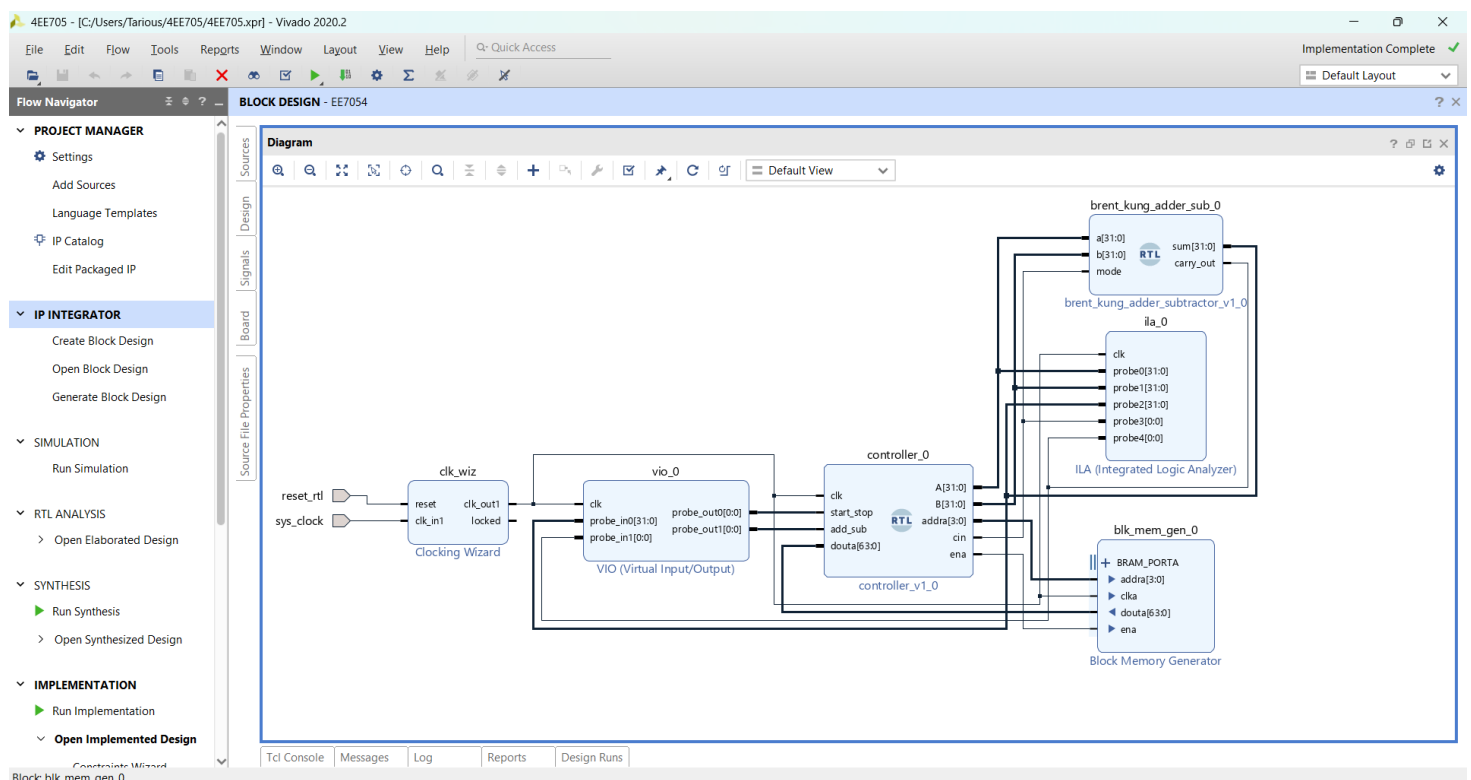
```
end
```

```
endmodule
```

Task 2: Design a controller which takes 2 signals from VIO: (a) A start/stop signal which can continuously read the data from BRAM and send it to the Adder (b) A signal to select addition or subtraction operation to be performed. BRAM will provide the inputs to your Adder as shown in the figure given below.

Create a testbench to test the controller with BRAM.

Block Design



Constraint File Code:

```
#system clk H16 - 125MHz

set_property -dict {PACKAGE_PIN H16 IOSTANDARD LVCMOS33} [get_ports sys_clock]

#Inputs en and reset from slider switches

set_property -dict {PACKAGE_PIN M20 IOSTANDARD LVCMOS33} [get_ports reset_rtl]
```

Verilog Code for Controller:

```
`timescale 1ns / 1ps

module controller (
    input clk,           // Clock signal
    input start_stop,    // Start/Stop signal
    input add_sub,       // Add/Subtract mode (0 = Add, 1 = Subtract)
    input [63:0] douta,  // Data output from BRAM
    output reg [31:0] A,  // Operand A
    output reg [31:0] B,  // Operand B
    output reg [3:0] addra = 4'b0000, // Address to BRAM
    output reg cin,
    output reg ena // Enable signal
        // Carry-in to the adder (add_sub passed here)
);

// Sequential block for managing operations
always @(posedge clk) begin
    if (start_stop) begin
        ena<=1;

        A <= douta[63:32]; // Upper 32 bits of douta
        // Set carry-in based on add_sub (0 for Add, 1 for Subtract)
        if (add_sub)
            begin
                B <= ~douta[31:0];
```

```
        cin <= 1;
    end
    else
        begin
            B <= douta[31:0];
            cin <= 0;
        end
        // Increment address
        addra <= addra + 1;
    end
    else begin
        // If start/stop is low, reset A, B, and cin while holding the address
        A <= 32'd0;
        B <= 32'd0;
        cin <= 1'b0;
        ena<=0;
    end
end
endmodule
```

Test Bench Code for Controller

```
`timescale 1ns / 1ps
module controller_tb;
    // Inputs to the controller
    reg clk;
    reg start_stop;
    reg add_sub;
    reg [63:0] douta;
```

// Outputs from the controller

```
wire [31:0] A;  
wire [31:0] B;  
wire [3:0] addra;  
wire cin;  
wire ena;
```

// Instantiate the controller module

```
controller uut (  
    .clk(clk),  
    .start_stop(start_stop),  
    .add_sub(add_sub),  
    .douta(douta),  
    .A(A),  
    .B(B),  
    .addra(addra),  
    .cin(cin),  
    .ena(ena)  
);
```

// Generate clock signal

```
always begin  
    #5 clk = ~clk; // Clock period is 10ns  
end
```

// Stimulus

```
initial begin
```

// Initialize inputs

```
clk = 0;  
start_stop = 0;  
add_sub = 0;  
douta = 64'h0000000000000000;
```

// Test case 1: Start the operation with Add mode (add_sub = 0)

#10;

douta = 64'h1234567890ABCDEF; **// Test data for douta**

start_stop = 1;

add_sub = 0; **// Add operation**

#20;

// Check values of A, B, and other outputs

\$display("A = %h, B = %h, addra = %h, cin = %b, ena = %b", A, B, addra, cin, ena);

// Test case 2: Start the operation with Subtract mode (add_sub = 1)

#10;

douta = 64'hFEDCBA9876543210; **// Test data for douta**

add_sub = 1; **// Subtract operation**

#20;

// Check values of A, B, and other outputs

\$display("A = %h, B = %h, addra = %h, cin = %b, ena = %b", A, B, addra, cin, ena);

// Test case 3: Stop the operation

#10;

start_stop = 0; **// Stop the operation**

#20;

// Final check

\$display("A = %h, B = %h, addra = %h, cin = %b, ena = %b", A, B, addra, cin, ena);

// End the simulation

\$finish;

end

endmodule