Name: Sreyan Biswas

Digital Electronics Verilog Projects

Project 1: 1 to N Adder

❖ Aim of the project:

i. Design a Verilog code to find the Sum of first N Natural Numbers.
ii. To implement the design on the FPGA resource and note down the resources utilised and time taken.

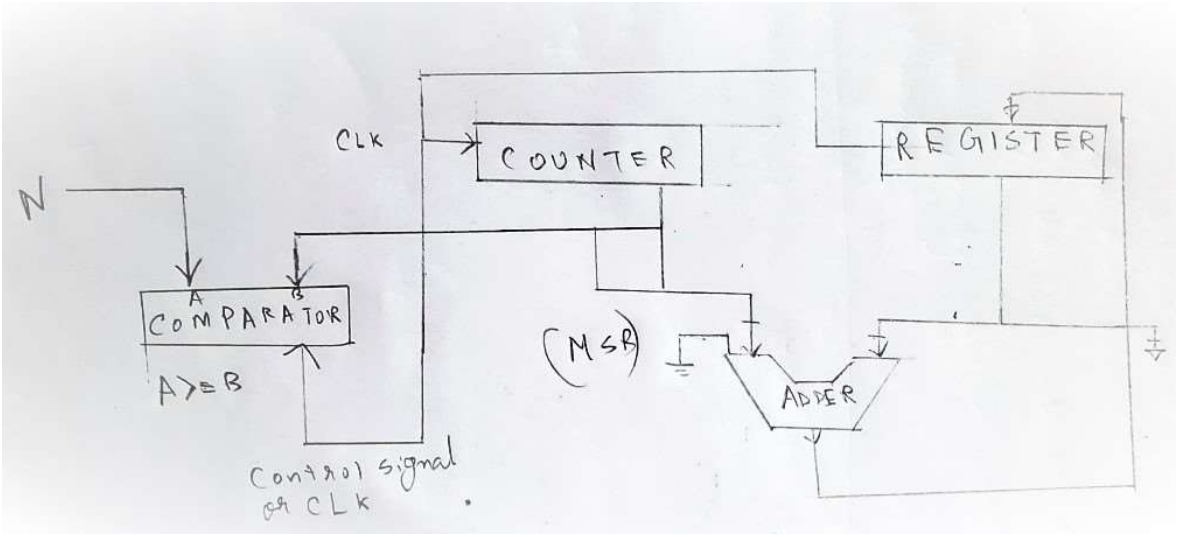
❖ Software used: ISE design suite

❖ Coding Language used: Verilog

❖ Simulation specifications:

- Behavioural design method was not used in this design.
- The design was made using the following MACRO libraries:
  - ADDSUB_MACRO: Used as 15-bit adder
  - COUNTER_TC_MACRO: Used as 8-bit counter
  - EQ_COMPARE_MACRO: Used as 8-bit comparator

Block Diagram:



- We can clearly see in the block diagram that we need an Comparator, Adder, register, counter and comparator.
- We first start the counter from '1' and then the adder adds this counter value to the register, which is the final sum being generated.
- Now the output of the adder is again stored in the register.
- The comparator checks that whether the counter value has reached the given input 'N' value.
- If the comparator output is true then we stop the counter and display the register value as output.
- If the output of the comparator is False, then the counter incremented and we repeat the process of adding the counter value to the register and storing it. This cycle is repeated

❖ Results:

a) **Timing diagram**

Time period of each clock cycle=**2ns**

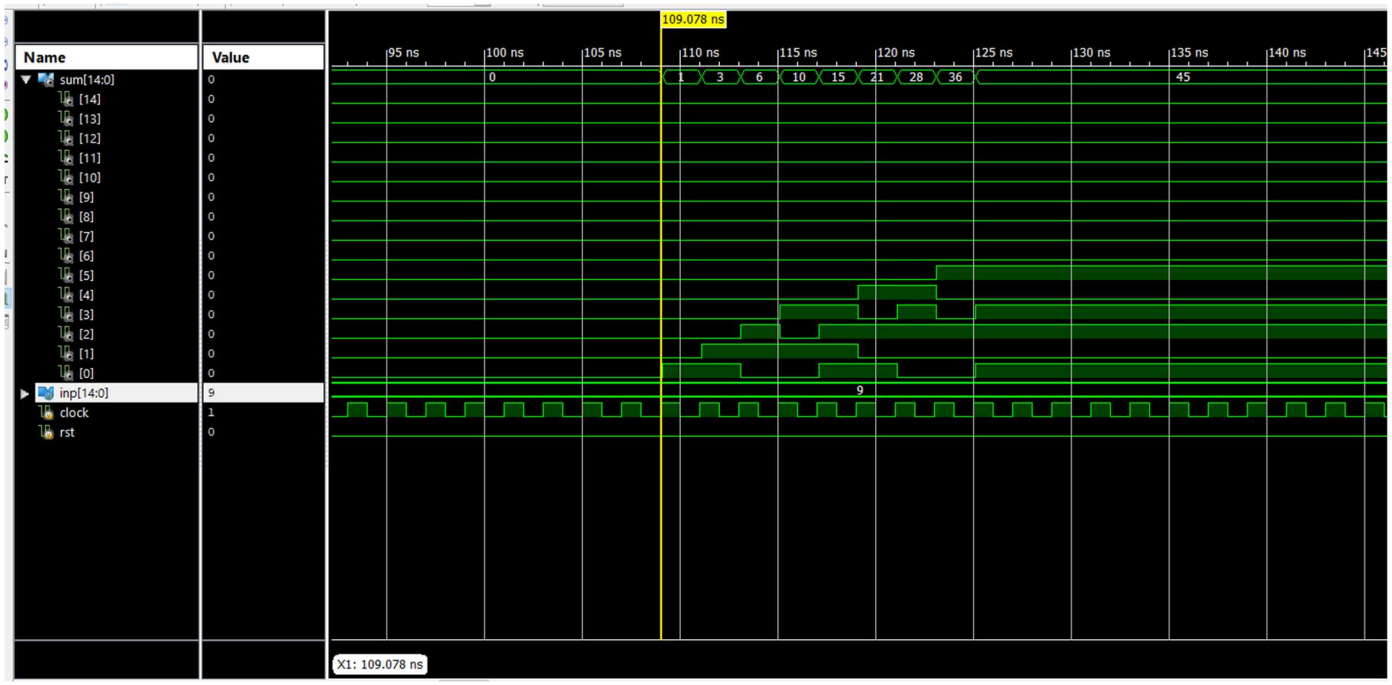
Input:

N=9

Clock period=2ns

Output:

Sum= 45

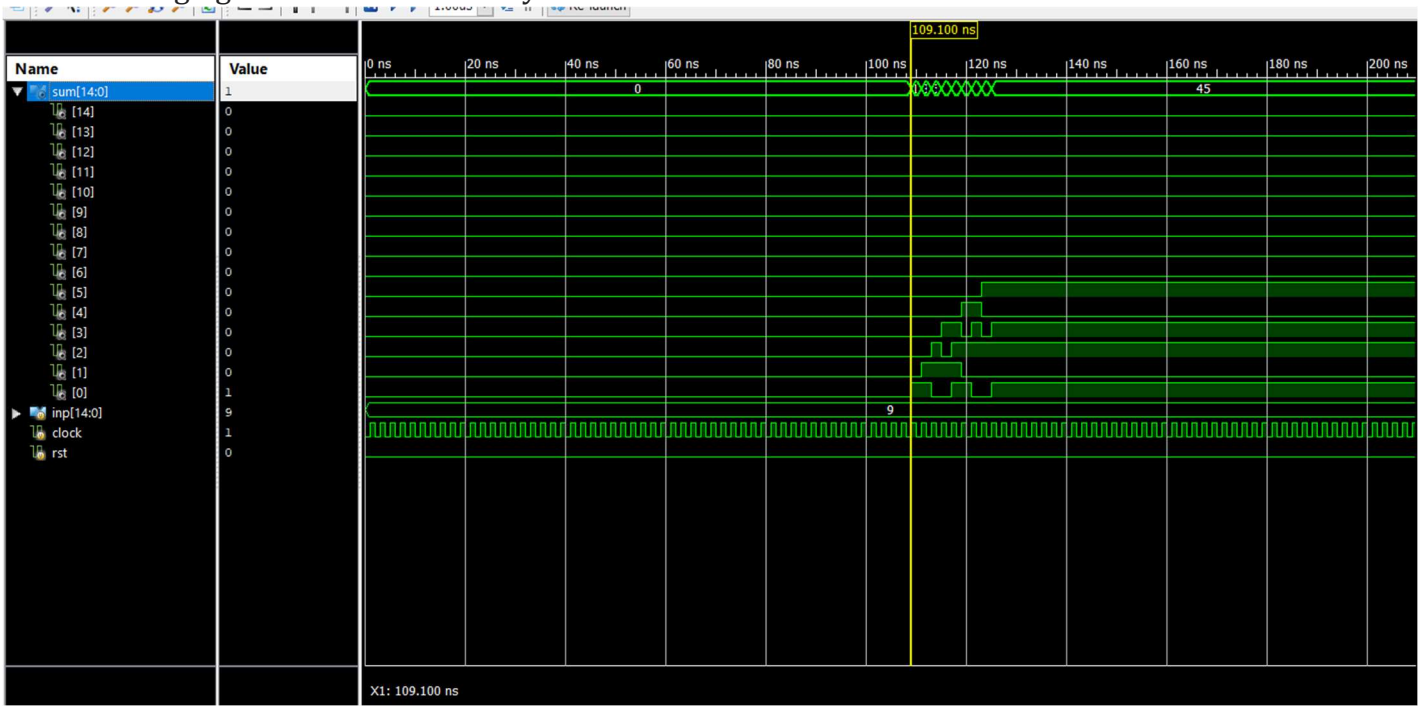


We can clearly see that for each clock cycle the output 'sum' changes as the summation keeps progressing. So for N=9, we need 8 clock cycles to compute the required sum after the output sum has reached '1'.

However to reach the state in which the sum=1 for the first , it took 109ns.

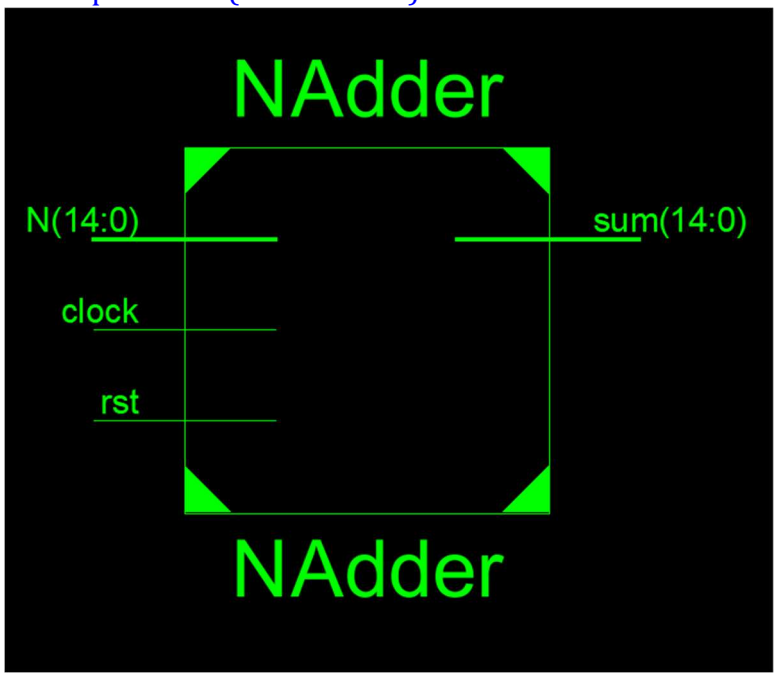
So number of clock cycles required to reach the state sum=1, will be 109/2 = 54 cycles

The following figure shows the clock cycles till the sum='1' is reached for the first time

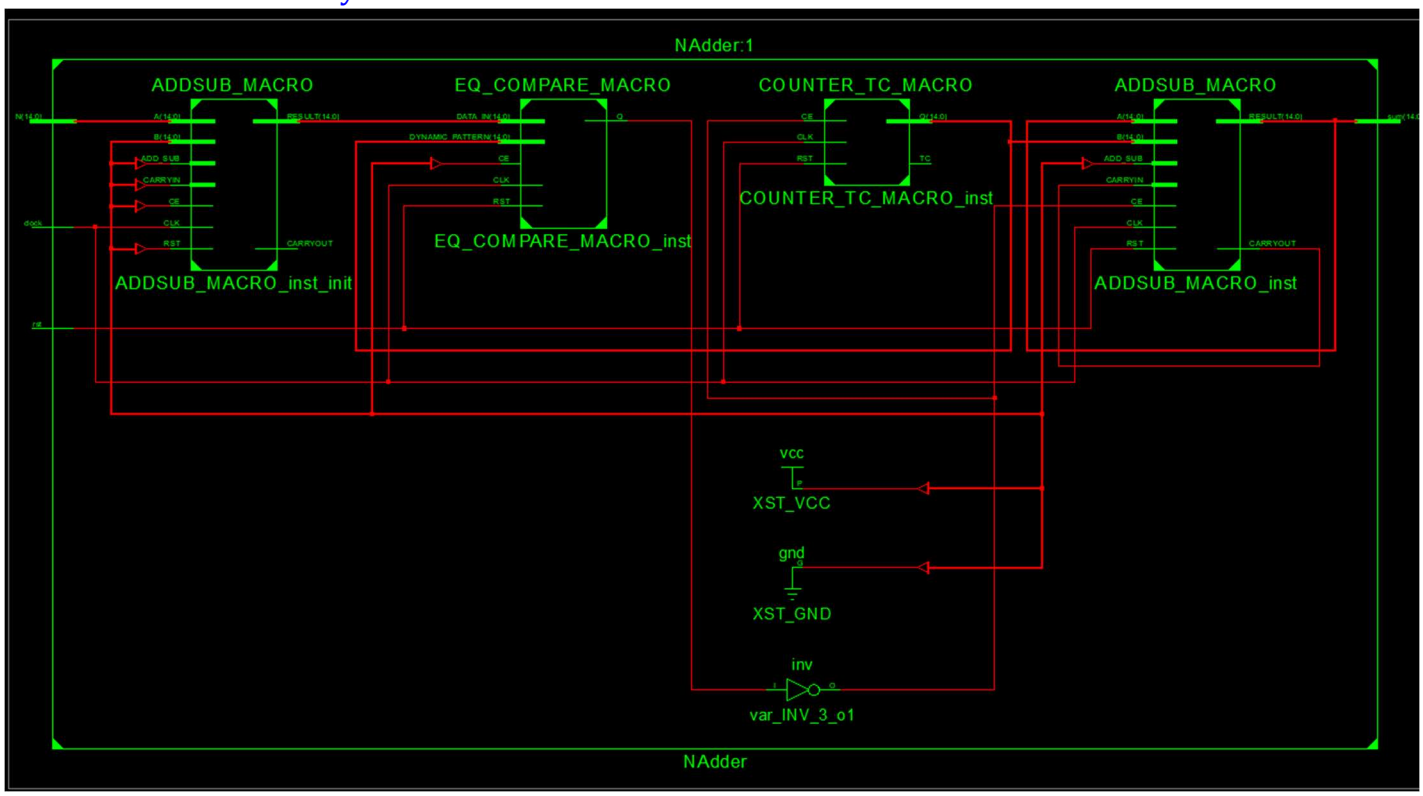


b) **RTL Schematic**

The top module (1 to N adder):



Internal Structure and layout of the different MACRO modules:



We can clearly see the Comparator, Counter, Adder modules in the schematic above

c) **Design Summary**

NAdder Project Status (03/07/2022 - 20:59:52)			
Project File:	p2_15bits_all.xise	**Parser Errors:**	No Errors
Module Name:	NAdder	**Implementation State:**	Placed and Routed
Target Device:	xc7a100t-3csg324	• **Errors:**	No Errors
Product Version:	ISE 14.7	• **Warnings:**	34 Warnings (0 new)
Design Goal:	Balanced	• **Routing Results:**	All Signals Completely Routed

Design Strategy:	Xilinx Default (unlocked)	• **Timing Constraints:**	All Constraints Met
Environment:	System Settings	• **Final Timing Score:**	0 (Timing Report)

Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	0	126,800	0%
Number of Slice LUTs	16	63,400	1%
Number used as logic	16	63,400	1%
Number using O6 output only	16		
Number using O5 output only	0		
Number using O5 and O6	0		
Number used as ROM	0		
Number used as Memory	0	19,000	0%
Number used exclusively as route-thrus	0		
Number of occupied Slices	9	15,850	1%
Number of LUT Flip Flop pairs used	16		
Number with an unused Flip Flop	16	16	100%
Number with an unused LUT	0	16	0%
Number of fully used LUT-FF pairs	0	16	0%
Number of slice register sites lost to control set restrictions	0	126,800	0%
Number of bonded IOBs	32	210	15%
Number of RAMB36E1/FIFO36E1s	0	135	0%
Number of RAMB18E1/FIFO18E1s	0	270	0%
Number of BUFG/BUFGCTRLs	1	32	3%
Number used as BUFGs	1		
Number used as BUFGCTRLs	0		
Number of IDELAYE2/IDELAYE2_FINEDELAYs	0	300	0%
Number of ILOGICE2/ILOGICE3/ISERDESE2s	0	300	0%
Number of ODELAYE2/ODELAYE2_FINEDELAYs	0		
Number of OLOGICE2/OLOGICE3/OSERDESE2s	0	300	0%
Number of PHASER_IN/PHASER_IN_PHYs	0	24	0%
Number of PHASER_OUT/PHASER_OUT_PHYs	0	24	0%
Number of BSCANs	0	4	0%
Number of BUFHCEs	0	96	0%
Number of BUFRs	0	24	0%
Number of CAPTUREs	0	1	0%
Number of DNA_PORTs	0	1	0%
Number of DSP48E1s	4	240	1%
Number of EFUSE_USRs	0	1	0%
Number of FRAME_ECCs	0	1	0%
Number of IBUFDS_GTE2s	0	4	0%
Number of ICAPs	0	2	0%
Number of IDELAYCTRLs	0	6	0%
Number of IN_FIFOs	0	24	0%
Number of MMCME2_ADVs	0	6	0%
Number of OUT_FIFOs	0	24	0%
Number of PCIE_2_1s	0	1	0%

Number of PHASER_REFs	0	6	0%
Number of PHY_CONTROLs	0	6	0%
Number of PLLE2_ADVs	0	6	0%
Number of STARTUPs	0	1	0%
Number of XADCs	0	1	0%
Average Fanout of Non-Clock Nets	2.15		

Performance Summary			[-]
Final Timing Score:	0 (Setup: 0, Hold: 0)	**Pinout Data:**	Pinout Report
Routing Results:	All Signals Completely Routed	**Clock Data:**	Clock Report
Timing Constraints:	All Constraints Met		

d) **Timing constraints**

	Met	Constraint	Check	Worst Case Slack	Best Case Achievable	Timing Errors	Timing Score
1	Yes	Autotimespec constraint for clock net clock_BUFGP	SETUPHOLD	0.476ns	4.923ns	0	00

```
--------------------------------------------------------------------------------
 Release 14.7 Trace  (nt64)
 Copyright (c) 1995-2013 Xilinx, Inc.  All rights reserved.

 D:\ISE_install\14.7\ISE_DS\ISE\bin\nt64\unwrapped\trce.exe -intstyle ise -v 3
 -s 3 -n 3 -fastpaths -xml NAdder.twx NAdder.ncd -o NAdder.twr NAdder.pcf

 Design file:              NAdder.ncd
 Physical constraint file: NAdder.pcf
 Device,package,speed:     xc7a100t,csg324,C,-3 (PRODUCTION 1.10 2013-10-13)
 Report level:             verbose report

 Environment Variable      Effect
 --------------------      ------
 NONE                      No environment variables were set
 --------------------------------------------------------------------------------

 INFO:Timing:2698 - No timing constraints found, doing default enumeration.

 INFO:Timing:3412 - To improve timing, see the Timing Closure User Guide (UG612).
 INFO:Timing:2752 - To get complete path coverage, use the unconstrained paths
    option. All paths that are not constrained will be reported in the
    unconstrained paths section(s) of the report.
 INFO:Timing:3339 - The clock-to-out numbers in this timing report are based on
    a 50 Ohm transmission line loading model.  For the details of this model,
    and for more information on accounting for different loading conditions,
    please see the device datasheet.

 Data Sheet report:
 -----------------
 All values displayed in nanoseconds (ns)

 Setup/Hold to clock clock
 ------------+------------+------------+------------+------------+------------------+---
-----+
             |Max Setup to|  Process   |Max Hold to |  Process   |                  |
Clock  |
 Source      | clk (edge) |   Corner   | clk (edge) |   Corner   |Internal Clock(s) |
Phase  |
 ------------+------------+------------+------------+------------+------------------+---
-----+
```

```
 N<0>        |    0.453(R)|      FAST  |    1.656(R)|      SLOW  |clock_BUFGP       |
0.000|
 N<1>        |    0.401(R)|      FAST  |    1.720(R)|      SLOW  |clock_BUFGP       |
0.000|
 N<2>        |    0.349(R)|      FAST  |    1.791(R)|      SLOW  |clock_BUFGP       |
0.000|
 N<3>        |    0.378(R)|      FAST  |    1.771(R)|      SLOW  |clock_BUFGP       |
0.000|
 N<4>        |    0.325(R)|      FAST  |    1.836(R)|      SLOW  |clock_BUFGP       |
0.000|
 N<5>        |    0.328(R)|      FAST  |    1.824(R)|      SLOW  |clock_BUFGP       |
0.000|
 N<6>        |    0.263(R)|      FAST  |    1.908(R)|      SLOW  |clock_BUFGP       |
0.000|
 N<7>        |    0.348(R)|      FAST  |    1.800(R)|      SLOW  |clock_BUFGP       |
0.000|
 N<8>        |    0.180(R)|      FAST  |    2.012(R)|      SLOW  |clock_BUFGP       |
0.000|
 N<9>        |    0.287(R)|      FAST  |    1.890(R)|      SLOW  |clock_BUFGP       |
0.000|
 N<10>       |    0.256(R)|      FAST  |    1.915(R)|      SLOW  |clock_BUFGP       |
0.000|
 N<11>       |    0.284(R)|      FAST  |    1.845(R)|      SLOW  |clock_BUFGP       |
0.000|
 N<12>       |    0.183(R)|      FAST  |    2.017(R)|      SLOW  |clock_BUFGP       |
0.000|
 N<13>       |    0.175(R)|      FAST  |    2.025(R)|      SLOW  |clock_BUFGP       |
0.000|
 N<14>       |    0.259(R)|      FAST  |    1.914(R)|      SLOW  |clock_BUFGP       |
0.000|
 rst         |    3.354(R)|      SLOW  |    1.931(R)|      SLOW  |clock_BUFGP       |
0.000|
 ------------+------------+------------+------------+------------+------------------+---
-----+

 Clock clock to Pad
 ------------+-----------------+-----------+-----------------+-----------+------------
------+--------+
             |Max (slowest) clk|  Process  |Min (fastest) clk|  Process  |
| Clock  |
 Destination |  (edge) to PAD  |   Corner  |  (edge) to PAD  |   Corner  |Internal
Clock(s) | Phase  |
 ------------+-----------------+-----------+-----------------+-----------+------------
------+--------+
 sum<0>      |         7.971(R)|      SLOW  |         3.391(R)|      FAST  |clock_BUFGP
|   0.000|
 sum<1>      |         7.879(R)|      SLOW  |         3.343(R)|      FAST  |clock_BUFGP
|   0.000|
 sum<2>      |         7.984(R)|      SLOW  |         3.401(R)|      FAST  |clock_BUFGP
|   0.000|
 sum<3>      |         8.090(R)|      SLOW  |         3.445(R)|      FAST  |clock_BUFGP
|   0.000|
 sum<4>      |         8.096(R)|      SLOW  |         3.456(R)|      FAST  |clock_BUFGP
|   0.000|
 sum<5>      |         8.158(R)|      SLOW  |         3.476(R)|      FAST  |clock_BUFGP
|   0.000|
 sum<6>      |         8.041(R)|      SLOW  |         3.420(R)|      FAST  |clock_BUFGP
|   0.000|
 sum<7>      |         8.006(R)|      SLOW  |         3.412(R)|      FAST  |clock_BUFGP
|   0.000|
 sum<8>      |         8.076(R)|      SLOW  |         3.437(R)|      FAST  |clock_BUFGP
|   0.000|
 sum<9>      |         8.034(R)|      SLOW  |         3.434(R)|      FAST  |clock_BUFGP
|   0.000|
 sum<10>     |         8.100(R)|      SLOW  |         3.454(R)|      FAST  |clock_BUFGP
|   0.000|
 sum<11>     |         8.399(R)|      SLOW  |         3.612(R)|      FAST  |clock_BUFGP
|   0.000|
 sum<12>     |         8.295(R)|      SLOW  |         3.568(R)|      FAST  |clock_BUFGP
|   0.000|
 sum<13>     |         8.293(R)|      SLOW  |         3.562(R)|      FAST  |clock_BUFGP
|   0.000|
```

```
 sum<14>      |          8.341(R)|      SLOW  |          3.565(R)|      FAST  |clock_BUFGP
|    0.000|
 -----------+-----------------+------------+-----------------+------------+------------
------+--------+

 Clock to Setup on destination clock clock
 ---------------+---------+---------+---------+---------+
                | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
 Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
 ---------------+---------+---------+---------+---------+
 clock          |    4.923|         |         |         |
 ---------------+---------+---------+---------+---------+


 Analysis completed Mon Mar 07 20:59:50 2022
 --------------------------------------------------------------------------------

 Trace Settings:
 -------------------------
 Trace Settings
 Peak Memory Usage: 5003 MB
```

❖ Verilog Code:

Main Module:

```
`timescale 1 ns/1 ps

module NAdder(
   input [14:0] N,
   //if input is not 14 bits then the MACROS dont work properly because they need all the inputs and outputs to tbe of the same width=15
   //the counter width should also be 15 bits because thats going to the input of the 15bit adder
   input clock, rst,

   output wire[14:0] sum //ouptut is 15bits
    );

    //DEFINING TEMPERORY USABLE VARIABLES
   wire var, carryo; //carry out
   wire[14:0] count, temp;
   //14 bits because the n can be 255(8bit) and n+1 will be 9bits
   //comp_output is the comparator output. comp_output=0 till the desired value is not reached by the output of the counter

   //temp=n+1 always. we need to run the counter n+1 times because the output of the counter starts from '0'
   //so when count=0, sum=0. //when count=1,sum=1.
   //so even if do n=1, then we will get the desired output sum='1', after 2 cycles of the counter



    //MACRO MODULE TO PERFORM ADDITION OF CURRENT NUMBER TO PREV SUM
   ADDSUB_MACRO #(
      .DEVICE("7SERIES"), // Target Device: "VIRTEX5", "VIRTEX6", "SPARTAN6", "7SERIES"
      .LATENCY(1),        // Desired clock cycle latency, 0-2
      .WIDTH(15)          // Input / output bus width, 1-15
   ) ADDSUB_MACRO_inst (
      .CARRYOUT(carryo), // 1-bit carry-out output signal
      .RESULT(sum),     // Add/sub result output, width defined by WIDTH parameter
      .A(sum),              // Input A bus, width defined by WIDTH parameter
      .ADD_SUB(1'b1),   // 1-bit add/sub input, high selects add, low selects subtract
      .B(count),            // Input B bus, width defined by WIDTH parameter
      .CARRYIN(carryo),   // 1-bit carry-in input
      .CE(~var),            // 1-bit clock enable input
      .CLK(clock),          // 1-bit clock input
      .RST(rst)            // 1-bit active high synchronous reset
   );

    //MACRO MODULE TO PERFORM ADDITION OF 1 TO COUNTER AT EVERY POSEDGE OF CLOCK. output is temp=n+1
   ADDSUB_MACRO #(
      .DEVICE("7SERIES"), // Target Device: "VIRTEX5", "VIRTEX6", "SPARTAN6", "7SERIES"
      .LATENCY(1),        // Desired clock cycle latency, 0-2
      .WIDTH(15)          // Input / output bus width, 1-15
   ) ADDSUB_MACRO_inst_init (
      .CARRYOUT(), // 1-bit carry-out output signal
      .RESULT(temp),     // Add/sub result output, width defined by WIDTH parameter
      .A(N),              // Input A bus, width defined by WIDTH parameter
      .ADD_SUB(1'b1),   // 1-bit add/sub input, high selects add, low selects subtract
      .B(15'b1),              // Input B bus, width defined by WIDTH parameter
      .CARRYIN(1'b0),   // 1-bit carry-in input
      .CE(1'b1),            // 1-bit clock enable input
      .CLK(clock),          // 1-bit clock input
      .RST(1'b0)           // 1-bit active high synchronous reset
   );

    //MACRO FOR COUNTER
   COUNTER_TC_MACRO #(
      .COUNT_BY(48'h000000000001), // Count by value
      .DEVICE("7SERIES"), // Target Device: "VIRTEX5", "VIRTEX6", "7SERIES"
      .DIRECTION("UP"), // Counter direction, "UP" or "DOWN"
      .RESET_UPON_TC("FALSE"), // Reset counter upon terminal count, "TRUE" or "FALSE"
      .TC_VALUE(48'h000000000000), // Terminal count value
      .WIDTH_DATA(15) // Counter output bus width, 1-15
   ) COUNTER_TC_MACRO_inst (
      .Q(count), // Counter output bus, width determined by WIDTH_DATA parameter
      .TC(), // 1-bit terminal count output, high = terminal count is reached
      .CLK(clock), // 1-bit positive edge clock input
      .CE(~var), // 1-bit active high clock enable input
```

```
                // we stop the counter as soon as we reach our required output(count)=n+1
       .RST(rst) // 1-bit active high synchronous reset
   );

    //MACRO TO COMPARE TWO NUMBERS, USED TO STOP COUNTER AT USER INPUT
   EQ_COMPARE_MACRO #(
       .DEVICE("7SERIES"),       // Target Device: "VIRTEX5", "VIRTEX6","7SERIES"
       .LATENCY(0),              // Desired clock cycle latency, 0-2
       .MASK(48'hFFFFFFFFFFFF),  // Select bits to be masked, must set SEL_MASK="MASK"
       .SEL_MASK("DYNAMIC_PATTERN"),        // "MASK" = use MASK parameter,
                                 //   "DYNAMIC_PATTERN" = use DYNAMIC_PATTERN input bus
       .SEL_PATTERN("DYNAMIC_PATTERN"), // "STATIC_PATTERN" = use STATIC_PATTERN parameter,
                                        //   "DYNAMIC_PATTERN = use DYNAMIC_PATTERN input bus
       .STATIC_PATTERN(48'hFFFFFFFFFFFF), // Specify static pattern, must set SEL_PATTERN = "STATIC_PATTERN"
       .WIDTH(15)                // Comparator output bus width, 1-15
   ) EQ_COMPARE_MACRO_inst (
       .Q(var),     // 1-bit output indicating a match
       .CE(1'b1),   // 1-bit active high input clock enable
       .CLK(clock), // 1-bit positive edge clock input
       .DATA_IN(temp), // Input Data Bus, width determined by WIDTH parameter
                   // temp =n+1. we wanna run the counter for n+1 times
       .DYNAMIC_PATTERN(count), // Input Dynamic Match/Mask Bus, width determined by WIDTH parameter
       .RST(rst)  // 1-bit input active high reset
   );

endmodule

```

Test bench

```
`timescale 1 ns/1 ps
module main_testbench;

   // DEFINING THE INPUTS
   reg[14:0] inp;
    wire [14:0] sum;
   reg clock, rst;

    //MAIN MODULE
   NAdder A(

    .sum(sum),
    .clock(clock),
   .N(inp),
   .rst(rst)

   );


    //INITIALIZING THE INPUTS TO OUR TESTBENCH
   initial begin
        clock = 1'b0;
        rst = 1'b0;
      inp = 15'd9;
      //inp = 15'b000000000000111;
   end

   always #1
   begin
      clock = ~clock;
   end
   initial begin
   $monitor("t=%2d n=%3d, out=%5d \n",$time,inp,sum);
   end
endmodule
```

Project 2: Processor using structural code

❖ Aim of the project:

Design a Processor to perform the following operations on 4 registers:

i. Addition
ii. Subtraction
iii. Move
iv. Input
v. Output

❖ Software used: ISE design suite

❖ Coding Language used: Verilog

❖ Defined OpCode:

We have 4 registers which are 8-bit each. We denote the registers as follows

Register	Code
A	00
B	01
C	10
D	11

We also have a '**input_data**' port for giving 8-bit data input

OpCode for Instructions:

i) **Additon**

OPCODE: 00

Syntax: ADD <dest> <src1><src2>

Example: Instruction code= 00000110

OPCODE	SRC1	SRC2	DEST
00	00	01	10
ADD	A	B	C

The above code translates to C=A+B

ii) **Subtraction**

OPCODE: 01

Syntax: SUB <dest> <src1><src2>

Example: Instruction code= 01000110

OPCODE	SRC1	SRC2	DEST
01	00	01	10
SUB	A	B	C

The above code translates to C=A-B

iii) **Move**

OPCODE: 11

Syntax: MOV <xx><src><dest>

Example: Instruction code= 11000110

OPCODE	<xx>	SRC	DEST
11	00	01	10
MOV	Don't care	B	C

The above code translates to C=B. We copy the value of the register B into register C

iv) **Input**

OPCODE: 100

Syntax: IN <xxx><dest>

Example: Instruction code= 100 001 10

OPCODE	**<xxx>**	**DEST**
100	001	10
IN	Don't care	C

The above code translates to C=input_data

We store the 8-bit input data which comes through the input port and store it in the register C

v) **Output**

OPCODE: 101

Syntax: OUT <xxx><src>

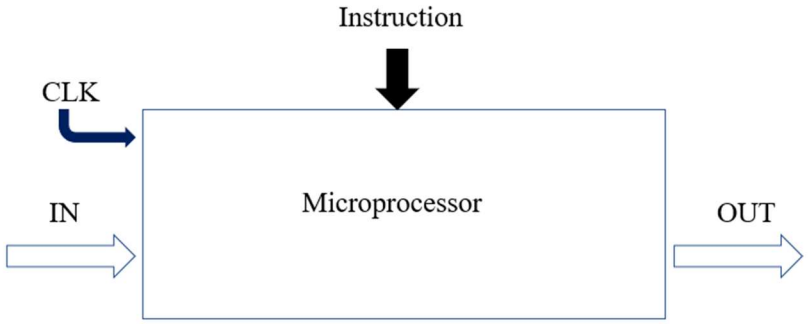
Example: Instruction code= 101 001 10

OPCODE	**<xxx>**	**SRC**
101	001	10
IN	Don't care	C

The above code translates to output=C

We take the data stored in the register C and then route it to the output port

The **Block diagram** of the Microprocessor has been shown below



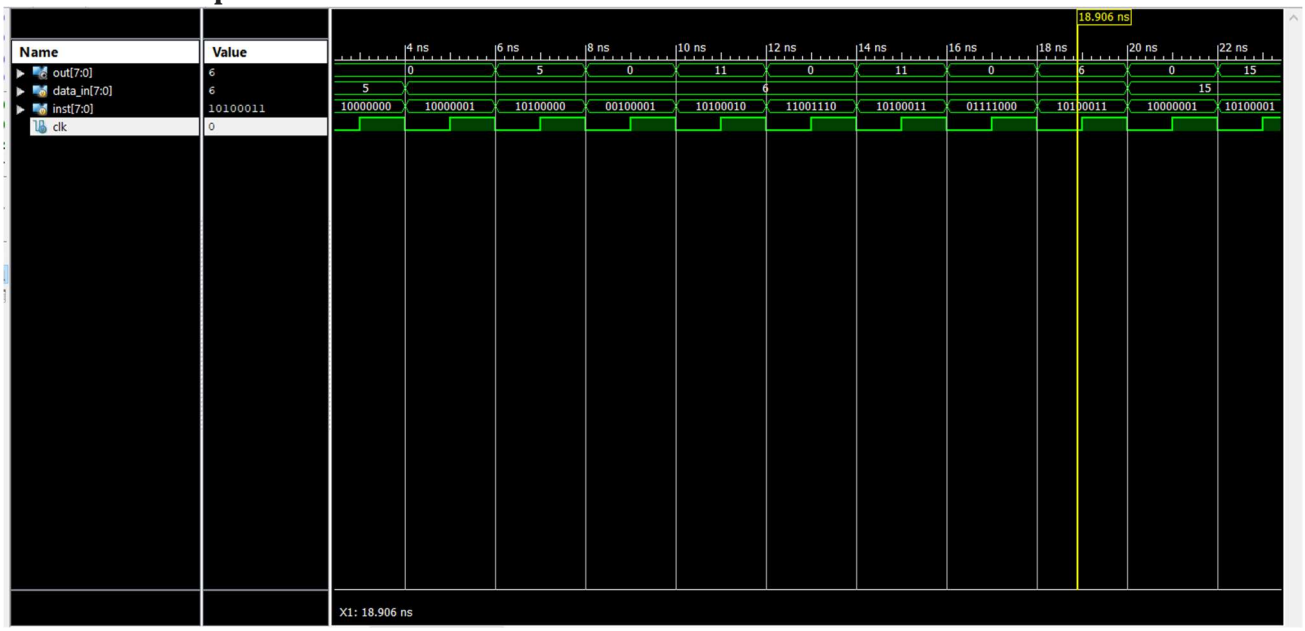
❖ Results:

a) **Console Output**

Complete output of the test bench code:

```
time=0,inst=00000000, out=  0
input 5 to A
time=2,inst=10000000, out=  0
input 6 to B
time=4,inst=10000001, out=  0
output A
time=6,inst=10100000, out=  5
ADD C=A+B
time=8,inst=00100001, out=  0
output C
time=10,inst=10100010, out= 11
move C to D
time=12,inst=11001110, out=  0
output D
time=14,inst=10100011, out= 11
SUB D=C-A
time=16,inst=01111000, out=  0
output D
time=18,inst=10100011, out=  6
input 15 to B
time=20,inst=10000001, out=  0
output B
time=22,inst=10100001, out= 15
output A
time=24,inst=10100000, out=  5
SUB C=B-A
time=26,inst=01100100, out=  0
output C
time=28,inst=10100010, out= 10
```

b) **Waveform output**



The above waveform correspond to the following portion of the testbench code:

```
input 5 to A
time=2,inst=10000000, out=  0
input 6 to B
time=4,inst=10000001, out=  0
output A
time=6,inst=10100000, out=  5
ADD C=A+B
time=8,inst=00100001, out=  0
output C
time=10,inst=10100010, out= 11
move C to D
time=12,inst=11001110, out=  0
output D
time=14,inst=10100011, out= 11
SUB D=C-A
time=16,inst=01111000, out=  0
output D
time=18,inst=10100011, out=  6
input 15 to B
time=20,inst=10000001, out=  0
```

Here we can clearly see that the given instructions produce the expected results in the output.

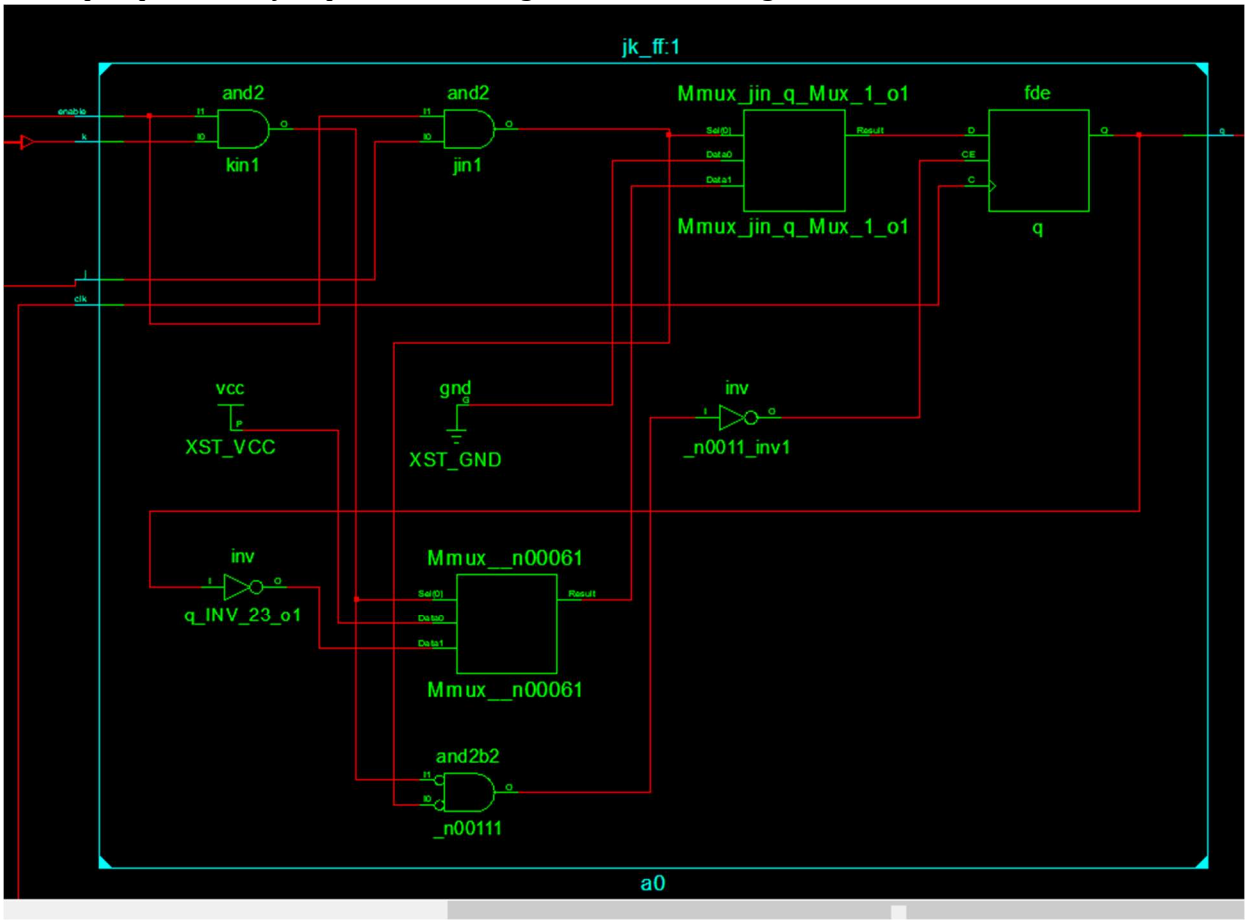
- **Instruction=** 10000000
  We input '5' to the register A and then input '6' to register B.
- **Instruction=** 10000001
  The data is fetched from the register A and the output is given as '5'. This is the expected output
- **Instruction=** 10100000
  Then we add the values of register 'A' and 'B' and then store it in the register 'C'
- **Instruction=** 00100001
  We fetch the data from register 'C' and then get it as the output

❖ Simulation specifications:

- Behavioural design method was not used in this design.
- No use of the keyword 'always' except in the flip flop module
- The entire code has been done in structural method.
- Any '**assign**' keyword can be replaced using some gates with an output and multiple inputs.
- The '**assign**' keyword has also been used to connect the wires
- The JK flip flop has been implemented using a '**reg q**'. And then the code for this particular module has been done in behavioural method
- Code for JK flip flop with an **enable** input to activate memory mode whenever the **enable=0**:

```
193 module jk_ff( input j, input k,input enable, input clk, output q);
194    reg q;
195    wire jin,kin;
196    assign jin= (j&enable);
197    assign kin= (k&enable);
198    always @ (posedge clk )
199       case ({jin,kin})
200          2'b00 :  q <= q;
201          2'b01 :  q <= 0;
202          2'b10 :  q <= 1;
203          2'b11 :  q <= ~q;
204       endcase
205
206 endmodule
```

- The flip flop is actually implemented using MUX in the Verilog Schematic . As it can be seen below



❖ Verilog Code:

Main Module:

```
`timescale 1ns / 1ps
module processor(data_in,out,inst,clk);
    input [7:0]data_in,inst;
    input clk;
    output [7:0]out; //9 bit output for sum but for now lets keep it 8-bit
    //wire [7:0]A,B,C,D;  //cant declare them as reg

    wire enIN,enOUT,enADD,enSUB,enMOV;
    and(enIN,inst[7],~inst[6],~inst[5]); //100
    and(enOUT,inst[7],~inst[6],inst[5]); //101
    and(enADD,~inst[7],~inst[6]); //00
    and(enSUB,~inst[7],inst[6]); //01
    and(enMOV,inst[7],inst[6]); //11

    wire enA,enB,enC,enD;
    wire [7:0] ff_in;
    wire [7:0] output_data;
    outdata
o1(enOUT,inst[1:0],output_data,aq0,aq1,aq2,aq3,aq4,aq5,aq6,aq7,bq0,bq1,bq2,bq3,bq4,bq5,bq6,bq7,cq0,cq1,cq2,cq3,cq4,cq5,cq6,cq7,dq0,dq1,dq2,dq3,dq4,dq5,dq6,dq7);
    assign out=output_data;

    //inst[5:4] // desitnation
    //inst[3:2],inst[1:0] // source 1,2
    wire [7:0] output_sum;
    wire enAsum,enBsum,enCsum,enDsum;
    add_out
add1(enADD,inst[5:4],inst[3:2],inst[1:0],enAsum,enBsum,enCsum,enDsum,output_sum,aq0,aq1,aq2,aq3,aq4,aq5,aq6,aq7,bq0,bq1,bq2,bq3,bq4,bq5,bq6,bq7,cq0,cq1,cq2,cq3,cq4,cq5,cq6
,cq7,dq0,dq1,dq2,dq3,dq4,dq5,dq6,dq7);
    //assign out=output_sum;  // works only if out has not been assigned earlier

    wire enAmov,enBmov,enCmov,enDmov;
    wire [7:0] output_mov;
    mov_out
mov1(enMOV,inst[3:2],inst[1:0],enAmov,enBmov,enCmov,enDmov,output_mov,aq0,aq1,aq2,aq3,aq4,aq5,aq6,aq7,bq0,bq1,bq2,bq3,bq4,bq5,bq6,bq7,cq0,cq1,cq2,cq3,cq4,cq5,cq6,cq7,dq0,dq
1,dq2,dq3,dq4,dq5,dq6,dq7);

    wire [7:0] output_sub;
    wire enAsub,enBsub,enCsub,enDsub;
    sub_out
sub1(enSUB,inst[5:4],inst[3:2],inst[1:0],enAsub,enBsub,enCsub,enDsub,output_sub,aq0,aq1,aq2,aq3,aq4,aq5,aq6,aq7,bq0,bq1,bq2,bq3,bq4,bq5,bq6,bq7,cq0,cq1,cq2,cq3,cq4,cq5,cq6
,cq7,dq0,dq1,dq2,dq3,dq4,dq5,dq6,dq7);

    wire enAin,enBin,enCin,enDin; //enAin=enable A in
    INdata i1(enIN,inst[1:0],enAin,enBin,enCin,enDin);
```

```
    assign ff_in[0]=(enADD&output_sum[0]) | (enIN&data_in[0]) | (enMOV&output_mov[0]) | (enSUB&output_sub[0]);
    assign ff_in[1]=(enADD&output_sum[1]) | (enIN&data_in[1]) | (enMOV&output_mov[1]) | (enSUB&output_sub[1]);
    assign ff_in[2]=(enADD&output_sum[2]) | (enIN&data_in[2]) | (enMOV&output_mov[2]) | (enSUB&output_sub[2]);
    assign ff_in[3]=(enADD&output_sum[3]) | (enIN&data_in[3]) | (enMOV&output_mov[3]) | (enSUB&output_sub[3]);
    assign ff_in[4]=(enADD&output_sum[4]) | (enIN&data_in[4]) | (enMOV&output_mov[4]) | (enSUB&output_sub[4]);
    assign ff_in[5]=(enADD&output_sum[5]) | (enIN&data_in[5]) | (enMOV&output_mov[5]) | (enSUB&output_sub[5]);
    assign ff_in[6]=(enADD&output_sum[6]) | (enIN&data_in[6]) | (enMOV&output_mov[6]) | (enSUB&output_sub[6]);
    assign ff_in[7]=(enADD&output_sum[7]) | (enIN&data_in[7]) | (enMOV&output_mov[7]) | (enSUB&output_sub[7]);

    assign enA= enAsum | enAin | enAmov | enAsub;
    assign enB= enBsum | enBin | enBmov | enBsub;
    assign enC= enCsum | enCin | enCmov | enCsub;
    assign enD= enDsum | enDin | enDmov | enDsub;

    jk_ff a0(ff_in[0],~ff_in[0],enA, clk, aq0);  // j,k,enable, clk, q
    jk_ff a1(ff_in[1],~ff_in[1],enA, clk, aq1);
    jk_ff a2(ff_in[2],~ff_in[2],enA, clk, aq2);
    jk_ff a3(ff_in[3],~ff_in[3],enA, clk, aq3);
    jk_ff a4(ff_in[4],~ff_in[4],enA, clk, aq4);
    jk_ff a5(ff_in[5],~ff_in[5],enA, clk, aq5);
    jk_ff a6(ff_in[6],~ff_in[6],enA, clk, aq6);
    jk_ff a7(ff_in[7],~ff_in[7],enA, clk, aq7);

    jk_ff b0(ff_in[0],~ff_in[0],enB, clk, bq0);  // j,k,enable, clk, q
    jk_ff b1(ff_in[1],~ff_in[1],enB, clk, bq1);
    jk_ff b2(ff_in[2],~ff_in[2],enB, clk, bq2);
    jk_ff b3(ff_in[3],~ff_in[3],enB, clk, bq3);
    jk_ff b4(ff_in[4],~ff_in[4],enB, clk, bq4);
    jk_ff b5(ff_in[5],~ff_in[5],enB, clk, bq5);
    jk_ff b6(ff_in[6],~ff_in[6],enB, clk, bq6);
    jk_ff b7(ff_in[7],~ff_in[7],enB, clk, bq7);

    jk_ff c0(ff_in[0],~ff_in[0],enC, clk, cq0);  // j,k,enable, clk, q
    jk_ff c1(ff_in[1],~ff_in[1],enC, clk, cq1);
    jk_ff c2(ff_in[2],~ff_in[2],enC, clk, cq2);
    jk_ff c3(ff_in[3],~ff_in[3],enC, clk, cq3);
    jk_ff c4(ff_in[4],~ff_in[4],enC, clk, cq4);
    jk_ff c5(ff_in[5],~ff_in[5],enC, clk, cq5);
    jk_ff c6(ff_in[6],~ff_in[6],enC, clk, cq6);
    jk_ff c7(ff_in[7],~ff_in[7],enC, clk, cq7);

    jk_ff d0(ff_in[0],~ff_in[0],enD, clk, dq0);  // j,k,enable, clk, q
    jk_ff d1(ff_in[1],~ff_in[1],enD, clk, dq1);
    jk_ff d2(ff_in[2],~ff_in[2],enD, clk, dq2);
    jk_ff d3(ff_in[3],~ff_in[3],enD, clk, dq3);
    jk_ff d4(ff_in[4],~ff_in[4],enD, clk, dq4);
    jk_ff d5(ff_in[5],~ff_in[5],enD, clk, dq5);
    jk_ff d6(ff_in[6],~ff_in[6],enD, clk, dq6);
    jk_ff d7(ff_in[7],~ff_in[7],enD, clk, dq7);
endmodule


module sub_out(
input enSUB,
input [1:0] destination,
input [1:0] source1,
input [1:0] source2,
output enA,enB,enC,enD,
output [7:0] out,
input aq0,aq1,aq2,aq3,aq4,aq5,aq6,aq7,
input bq0,bq1,bq2,bq3,bq4,bq5,bq6,bq7,
input cq0,cq1,cq2,cq3,cq4,cq5,cq6,cq7,
input dq0,dq1,dq2,dq3,dq4,dq5,dq6,dq7);

    wire [7:0] s1,s2; //fetch data
    outdata o_s2(enSUB,source2,s2,aq0,aq1,aq2,aq3,aq4,aq5,aq6,aq7,bq0,bq1,bq2,bq3,bq4,bq5,bq6,bq7,cq0,cq1,cq2,cq3,cq4,cq5,cq6,cq7,dq0,dq1,dq2,dq3,dq4,dq5,dq6,dq7);
    outdata o_s1(enSUB,source1,s1,aq0,aq1,aq2,aq3,aq4,aq5,aq6,aq7,bq0,bq1,bq2,bq3,bq4,bq5,bq6,bq7,cq0,cq1,cq2,cq3,cq4,cq5,cq6,cq7,dq0,dq1,dq2,dq3,dq4,dq5,dq6,dq7);

    //s1-s2,cin=1
    Adder sub_out(s1,~s2,1'b1,out,cout); //a,b,cin,sum,cout
    /*  wire [7:0] temp;
    Adder sub_out(s1,~s2,1,temp,cout); //a,b,cin,sum,cout
    assign out=temp;
    Adder final_out(~temp,0,1,out,cout2);
    */
    INdata i_sub(enSUB,destination,enA,enB,enC,enD);

endmodule

module mov_out(
input enMOV,
input [1:0] destination,
input [1:0] source,
output enA,enB,enC,enD,
output [7:0] out,
input aq0,aq1,aq2,aq3,aq4,aq5,aq6,aq7,
input bq0,bq1,bq2,bq3,bq4,bq5,bq6,bq7,
input cq0,cq1,cq2,cq3,cq4,cq5,cq6,cq7,
input dq0,dq1,dq2,dq3,dq4,dq5,dq6,dq7);

    outdata o_s1(enMOV,source,out,aq0,aq1,aq2,aq3,aq4,aq5,aq6,aq7,bq0,bq1,bq2,bq3,bq4,bq5,bq6,bq7,cq0,cq1,cq2,cq3,cq4,cq5,cq6,cq7,dq0,dq1,dq2,dq3,dq4,dq5,dq6,dq7);
    INdata i_mov(enMOV,destination,enA,enB,enC,enD);

endmodule

module add_out(
input enADD,
input [1:0] destination,
input [1:0] source1,
input [1:0] source2,
output enA,enB,enC,enD,
output [7:0] out,
input aq0,aq1,aq2,aq3,aq4,aq5,aq6,aq7,
input bq0,bq1,bq2,bq3,bq4,bq5,bq6,bq7,
input cq0,cq1,cq2,cq3,cq4,cq5,cq6,cq7,
input dq0,dq1,dq2,dq3,dq4,dq5,dq6,dq7);

    wire [7:0] s1,s2; //fetch data
    //wire cout;
    outdata o_s2(enADD,source2,s2,aq0,aq1,aq2,aq3,aq4,aq5,aq6,aq7,bq0,bq1,bq2,bq3,bq4,bq5,bq6,bq7,cq0,cq1,cq2,cq3,cq4,cq5,cq6,cq7,dq0,dq1,dq2,dq3,dq4,dq5,dq6,dq7);
    outdata o_s1(enADD,source1,s1,aq0,aq1,aq2,aq3,aq4,aq5,aq6,aq7,bq0,bq1,bq2,bq3,bq4,bq5,bq6,bq7,cq0,cq1,cq2,cq3,cq4,cq5,cq6,cq7,dq0,dq1,dq2,dq3,dq4,dq5,dq6,dq7);
    //COUT IS OPEN
    Adder sum_out(s1,s2,1'b0,out,cout); //a,b,cin,sum,cout
    INdata i_sum(enADD,destination,enA,enB,enC,enD);

endmodule


module outdata(
input enOUT,
input [1:0] sel,
output [7:0] out,
input aq0,aq1,aq2,aq3,aq4,aq5,aq6,aq7,
input bq0,bq1,bq2,bq3,bq4,bq5,bq6,bq7,
input cq0,cq1,cq2,cq3,cq4,cq5,cq6,cq7,
input dq0,dq1,dq2,dq3,dq4,dq5,dq6,dq7);
    //1:4 DeMUX
    wire enA,enB,enC,enD;
    assign enA = (~sel[0]) & (~sel[1]) &enOUT; //can be done using and gate too
    assign enB = (~sel[1]) & sel[0] &enOUT;
    assign enC = sel[1] & (~sel[0]) &enOUT;
```

```
    assign enD = sel[1] & sel[0] &enOUT;

    assign out[0]=(aq0&enA) | (bq0&enB) |(cq0&enC) | (dq0&enD);
    assign out[1]=(aq1&enA) | (bq1&enB) |(cq1&enC) | (dq1&enD);
    assign out[2]=(aq2&enA) | (bq2&enB) |(cq2&enC) | (dq2&enD);
    assign out[3]=(aq3&enA) | (bq3&enB) |(cq3&enC) | (dq3&enD);
    assign out[4]=(aq4&enA) | (bq4&enB) |(cq4&enC) | (dq4&enD);
    assign out[5]=(aq5&enA) | (bq5&enB) |(cq5&enC) | (dq5&enD);
    assign out[6]=(aq6&enA) | (bq6&enB) |(cq6&enC) | (dq6&enD);
    assign out[7]=(aq7&enA) | (bq7&enB) |(cq7&enC) | (dq7&enD);

endmodule

module INdata( input enIN,
                   input [1:0] sel,
                   output enA,enB,enC,enD);

    assign enA = (~sel[0]) & (~sel[1]) &enIN;
    assign enB = (~sel[1]) & sel[0] &enIN;
    assign enC = sel[1] & (~sel[0]) &enIN;
    assign enD = sel[1] & sel[0] &enIN;
endmodule

module jk_ff( input j, input k,input enable, input clk, output q);
   reg q;
    wire jin,kin;
    assign jin= (j&enable);
    assign kin= (k&enable);
   always @ (posedge clk )
        case ({jin,kin})
            2'b00 :  q <= q;
            2'b01 :  q <= 0;
            2'b10 :  q <= 1;
            2'b11 :  q <= ~q;
        endcase

endmodule

/*
module jk_ff(input j, input k,input enable, input clk, output q);

    wire jin,kin;
    assign jin= (j&enable);
    assign kin= (k&enable);
    wire nand1_out; // output from nand1
    wire nand2_out; // output from nand2

    nand(nand1_out, jin,clk,qbar);
    nand(nand2_out, kin,clk,q);
    nand(q,qbar,nand1_out);
    nand(qbar,q,nand2_out);

endmodule
*/

module Adder( a,b,cin,
                  sum,cout);  // dont do input a,b here
    input [7:0]a,b;
    input cin;
    output wire [7:0]sum;
    output cout;
    FullAdder FA1(a[0],b[0],cin,sum[0],cout1);  //FA1 is an instance of Fulladder module
    FullAdder FA2(a[1],b[1],cout1,sum[1],cout2);
    FullAdder FA3(a[2],b[2],cout2,sum[2],cout3);
    FullAdder FA4(a[3],b[3],cout3,sum[3],cout4);
    FullAdder FA5(a[4],b[4],cout4,sum[4],cout5);
    FullAdder FA6(a[5],b[5],cout5,sum[5],cout6);
    FullAdder FA7(a[6],b[6],cout6,sum[6],cout7);
    FullAdder FA8(a[7],b[7],cout7,sum[7],cout);

endmodule

module FullAdder(a,b,cin,sum,cout);
    input a,b,cin;
    output wire sum,cout;
    wire s1,c1,c2,c3;
    xor(s1,a,b);
    xor(sum,s1,cin);
    and(c1,a,b);
    and(c2,b,cin);
    and(c3,a,cin);
    or(cout,c1,c2,c3);
endmodule

```

Test bench:

```
`timescale 1ns / 1ps
module test1;

    // Inputs
    reg [7:0] data_in;
    reg [7:0] inst;
    reg clk;

    // Outputs
    wire [7:0] out;
    always #1 clk = ~clk;

    // Instantiate the Unit Under Test (UUT)
    processor uut (
        .data_in(data_in),
        .out(out),
        .inst(inst),
        .clk(clk)
    );

    initial begin
        // Initialize Inputs
        data_in = 0;
        inst = 0;
        clk = 0;

        #2
        $display("input 5 to A");
        data_in = 8'd5;
        inst = 8'b10000000;
        #2
        data_in = 8'd6;
        $display("input 6 to B");
        inst = 8'b10000001;

        #2
        $display("output A");
        inst = 8'b10100000;
        #2
        $display("ADD C=A+B");  //c=11,a=5,b=6
        inst = 8'b00100001;
          #2
          $display("output C");
          inst = 8'b10100010;

          #2
          $display("move C to D");
          inst = 8'b11001110;
          #2
          $display("output D");
          inst = 8'b10100011;

          #2
          $display("SUB D=C-A"); //11-5=6
          inst = 8'b01111000;

          #2
          $display("output D");
          inst = 8'b10100011;

          #2
          data_in = 8'd15;
          $display("input 15 to B");
          inst = 8'b10000001;

          #2
          $display("output B");
          inst = 8'b10100001;

          #2
          $display("output A");
          inst = 8'b10100000;

          #2
          $display("SUB C=B-A"); //15-5=10
          inst = 8'b01100100;  // we cant do B=B-A bcoz we dont store the value in some temporary register.

          #2
          $display("output C");
          inst = 8'b10100010;
     end
    initial $monitor("time=%g,inst=%b, out=%d",$time,inst,out);
    initial #100 $finish;
endmodule
```