**Nanoprocessor Design**

Project Report

#Group 44

* A.S. Jayathunga - 200265T
* M.R.A.A.K. Gunasinghe - 200196G
* K.A.Anshan Lahiru Kavinda - 200300A
* K.K.A.J.S. Kumarasinghe - 200323V
* W.M.T.B. Weerasekara - 200698X

[**Assigned Lab Task**](#_zg21iu1ekiun) **5**

[**A Brief Description about our implementation of Nanoprocessor**](#_i1dgu8mll52i) **6**

[**Assembly program and its machine code representation of a program to calculate the sum of numbers from 1 to 3.**](#_29g5chdztsut) **9**

[**Top Level Design Of Nanoprocessor**](#_ocy3qttn7j6f) **10**

[**Program Counter**](#_uzkrz2bw9tmt) **11**

[Schematic](#_13fu5orz5ygr) 11

[VHDL File](#_aaz7n7y0guzu) 12

[Timing Diagram](#_5qxv7aj7zmai) 14

[**3-bit Ripple Carry Adder (Incremental)**](#_6adjpaw0q2xm) **15**

[Schematic](#_tq3t2ftycuj) 15

[VHDL File](#_ekfkqmqv6o0) 16

[Timing Diagram](#_46f1pyf49kmu) 19

[**Program Rom**](#_cgx5r05lve06) **20**

[Schematic](#_3whidkfoam8r) 20

[VHDL File](#_qbbp0x9t0kum) 21

[Timing Diagram](#_yz7kpzcw0map) 23

[**Instruction Decoder**](#_60ycoxm4qnpq) **24**

[Block Diagram](#_9dza4uw73mzb) 24

[Schematic](#_k3t5vvqstx20) 25

[VHDL File](#_sl1t8fgx6mei) 26

[Timing Diagram](#_eytw8s8beon0) 28

[**Arithmetic Unit**](#_wzc504724bvz) **29**

[Block Diagram](#_gprxhoy7qfpq) 29

[Schematic](#_slx0re1e9x7e) 30

[VHDL File](#_euoxe6ja1emh) 31

[Timing Diagram](#_ptgnwao6osjd) 34

[**8 Way 4-bit Multiplexer**](#_qypgj9xsoe7z) **35**

[Schematic](#_nlql134sq7r9) 35

[VHDL File](#_2dch88q6egda) 35

[Timing Diagram](#_dffzhooifg18) 39

[**2 way 3-bit Multiplexer**](#_ia28ziobb1jk) **40**

[Schematic](#_hloy71qgxhrw) 40

[VHDL File](#_j46v3c154izu) 40

[Timing Diagram](#_angv0x4o1xjb) 43

[**2 way 4-bit Multiplexer**](#_rshx2ds2c6jk) **44**

[Schematic](#_mgvhrkgsgbv8) 44

[VHDL File](#_jdzjfj88u5f8) 44

[Timing Diagram](#_5ttx0e4fmn6n) 47

[**Register Bank**](#_8o6j5dj2j19p) **48**

[Schematic](#_ilknl1bil25i) 48

[VHDL File](#_btqn2k3afagm) 49

[Timing Diagram](#_bx6ij3wydoba) 53

[**Nanoprocessor**](#_1q5vtarp1oue) **54**

[Schematic](#_wqhiq7mt4g59) 54

[VHDL File](#_9i5ei2elc83h) 54

[Timing Diagram](#_h5e587tpaseh) 63

[Timing Diagram with Internal Signals (Used For Debugging)](#_ny2ye8bqol0c) 64

[**Conclusion from the Lab**](#_8rtv4vcwoqur) **64**

[**Contribution of members to the project**](#_y4mjvb6ncak4) **67**

[A.S. JAYATHUNGA - 200265T](#_8t8p5qaiacln) 67

[M.R.A.A.K. Gunasinghe - 200196G](#_nvxbi4de3jnh) 67

[K.A.Anshan Lahiru Kavinda - 200300A](#_twnclg7fghdo) 67

[K.K.A.J.S. Kumarasinghe - 200323V](#_wbii159s6zsj) 67

[W.M.T.B.Weerasekara - 200698X](#_b1zkcsdqboll) 67

# 

# Assigned Lab Task

Design a 4-bit processor capable of executing four basic instructions and having memory for eight such instructions.

Available Instructions are as follows,

1. Update a register value with the addition of itself and a value of another register. (ADD)
2. Change a value of a register to its two’s complementary negative. (NEG)
3. Update a register value with a 4-bit immediate value. (MOVI)
4. Jump to a specified instruction if the value of a particular register equals zero. (JZR)

After completing the design of the processor, verify its functionality by running a simple assembly program to calculate the sum of numbers from 1 to 3. Assembly program must be converted to the relevant machine code of our processor and stored in the memory we designed for our processor. Number of instructions of our assembly program cannot exceed eight since our processor only has the capability of accommodating eight instructions at once. Final result of the program must be displayed on a seven segment display integrated to our processor. Processor should also have two output flags to indicate the current state of the output value of the Arithmetic Unit. One of the flags indicates whether the value is zero and other flag indicates if an overflow condition occurs. These two flags must be connected to two LEDs.

# 

# A Brief Description about our implementation of Nanoprocessor

**RESET : Center push button**

**Register\_7\_output : LED (2 downto 0)**

**zero\_flag: LED 14**

**overflow(Carry)\_flag: LED 15**

PS : R0 Register in Register Bank is Read Only with the value zero.

Instruction Set

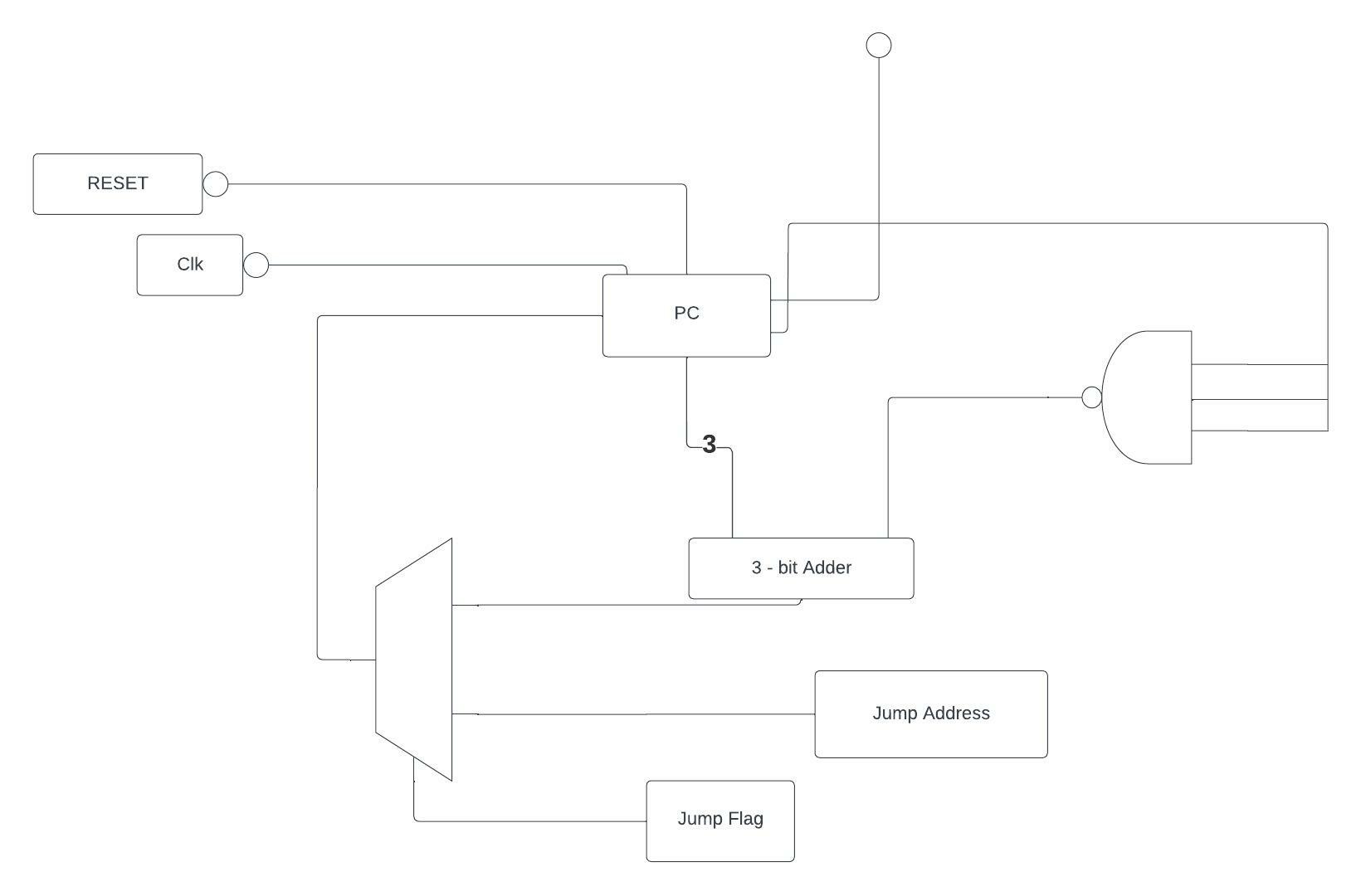
| **Instruction** | **Operation** | **Format (12-bit Instruction)** |
| --- | --- | --- |
| MOVI R,d | R ← d | 10RRR000dddd |
| ADD Ra,Rb | Ra ← Ra+Rb | 00RaRaRaRbRbRb0000 |
| NEG R | R ← -R | 01RRR0000000 |
| JZR R,d | If R==0 : PC ← d  Else : PC ← PC+1 | 11RRR0000ddd |
| NOP | Do Nothing | 000000000000 |

PS: Here all zeros (000000000000) can be considered as NOP because, even though it is originally an ADD instruction, here addition is performed on the read only register with value zero. Hence it does not manipulate any register value and basically does not make any effect on the processor.

**Program counter and 3-bit adder Modification**

Program counter value increments by 1 after each clock cycle until it reaches the last value of the Program ROM ( 7 in this implementation) and stops.

This is achieved by designing the 3-bit incremental adder in such a way that it changes its incremental value from 1 to 0 once its input value gets a specified number (7 in this implementation).



MUX Designs

* 8 way 4-bit MUX was designed using four 8 to 1 MUXs.
* 2 way 3-bit MUX was designed using three 2 to 1 MUXs.
* 2 way 4-bit MUX was designed using one 2 to 1 MUX and one 2 way 3-bit MUX.

# Assembly program and its machine code representation of a program to calculate the sum of numbers from 1 to 3.

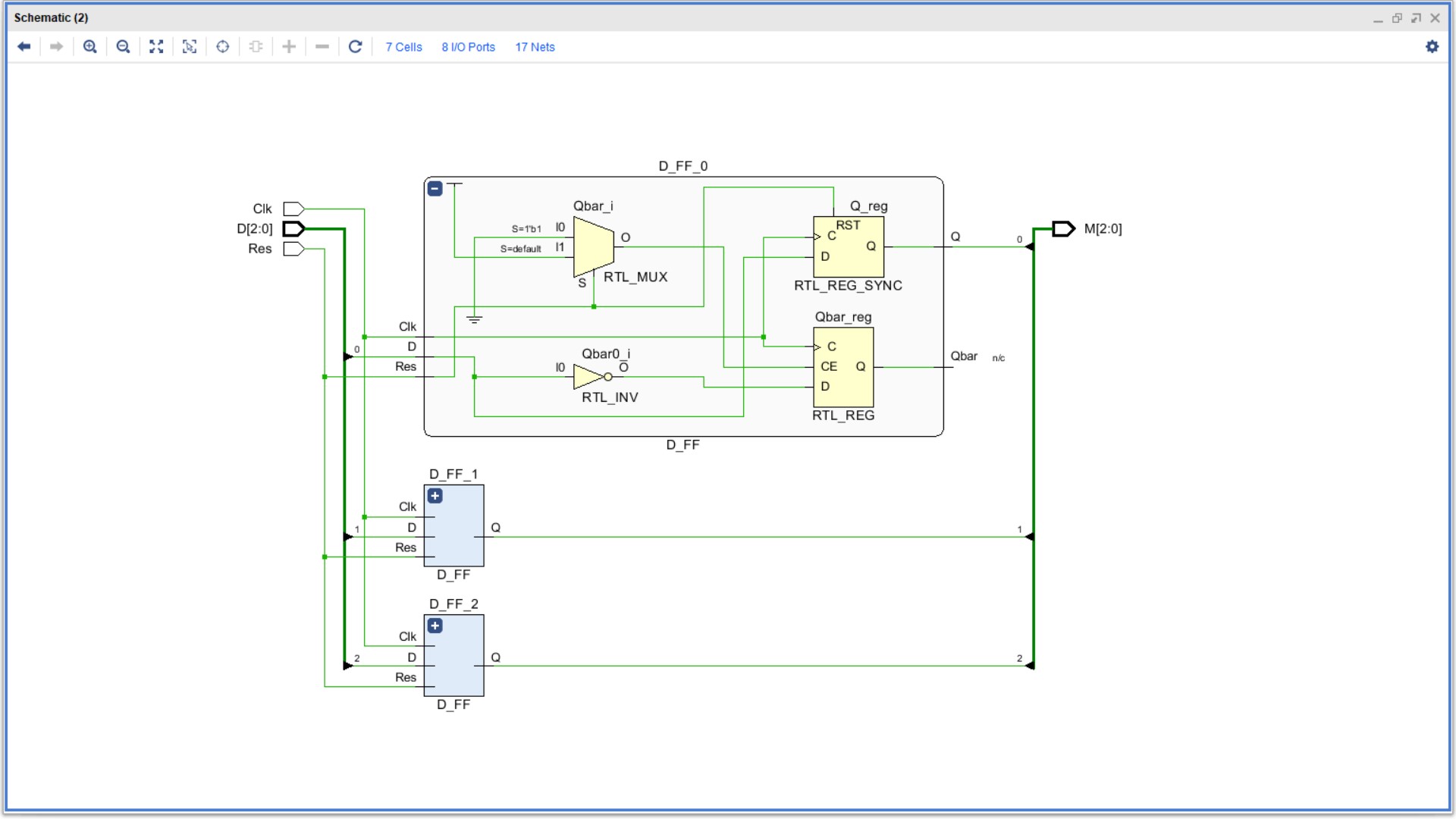
| Assembly Code | Machine Code | Operation | Registers Contents |
| --- | --- | --- | --- |
| MOVI R1,1 | 100010000001 | R1 <- 1 | R7=0,R1=1 |
| MOVI R2,1 | 100100000001 | R2 <- 1 | R7=0,R1=1,R2=1 |
| ADD R7,R2 | 001110100000 | R7 <- R7 + R2 | R7=1,R1=1,R2=1 |
| ADD R2,R1 | 000100010000 | R2 <- R2 + R1 | R7=1,R1=1,R2=2 |
| ADD R7,R2 | 001110100000 | R7 <- R7 + R2 | R7=3,R1=1,R2=2 |
| ADD R2,R1 | 000100010000 | R2 <- R2 + R1 | R7=3,R1=1,R2=3 |
| ADD R7,R2 | 001110100000 | R7 <- R7 + R2 | R7=6,R1=1,R2=3 |
| NOP | 000000000000 | R0 <- R0 + R0 | R7=6,R0=0 |

# 

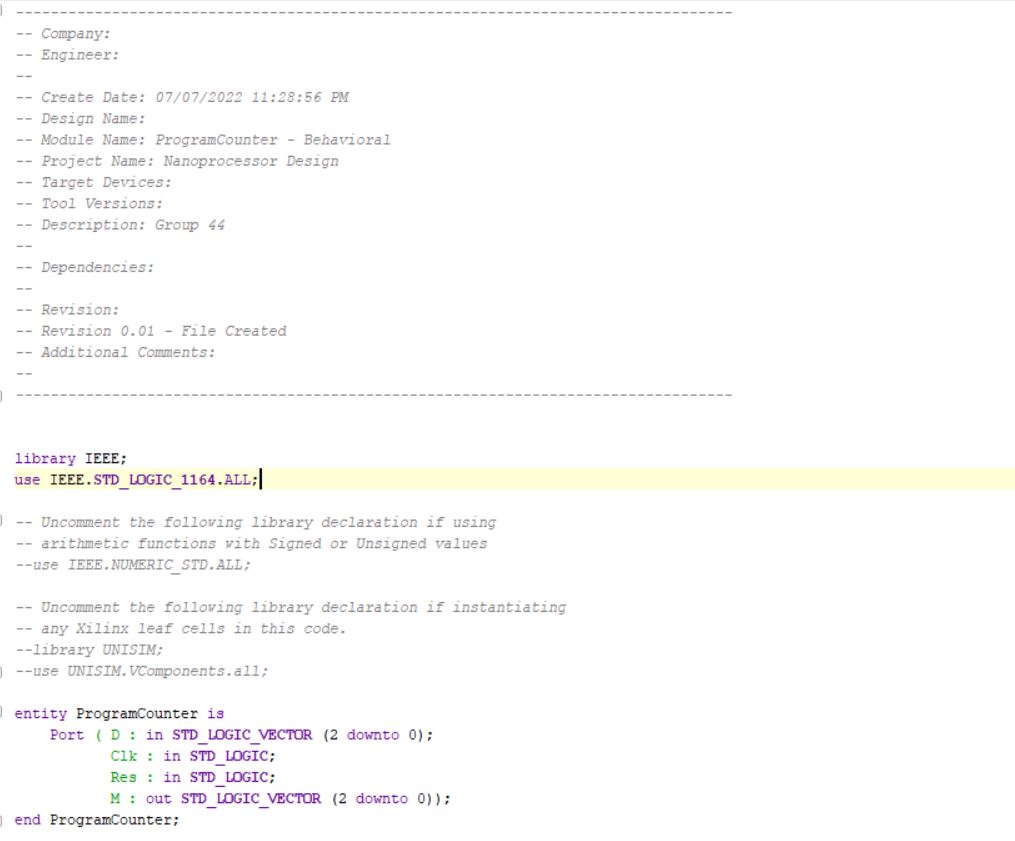
# Top Level Design Of Nanoprocessor

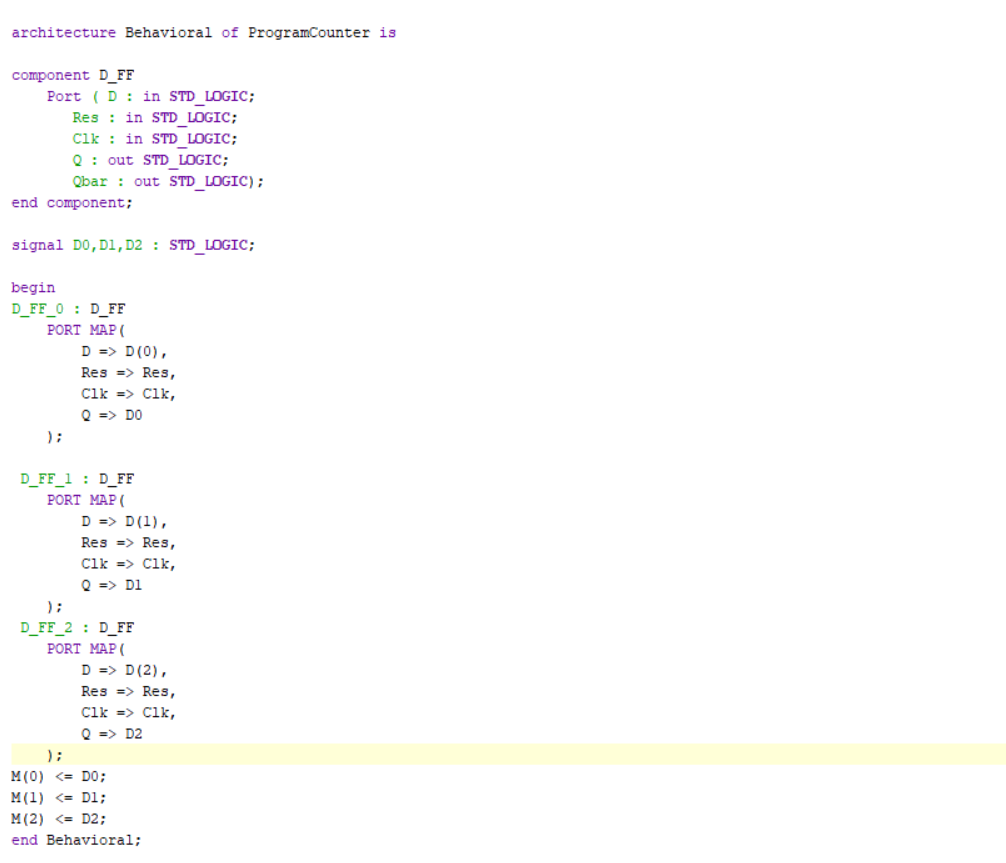
# Program Counter

## Schematic

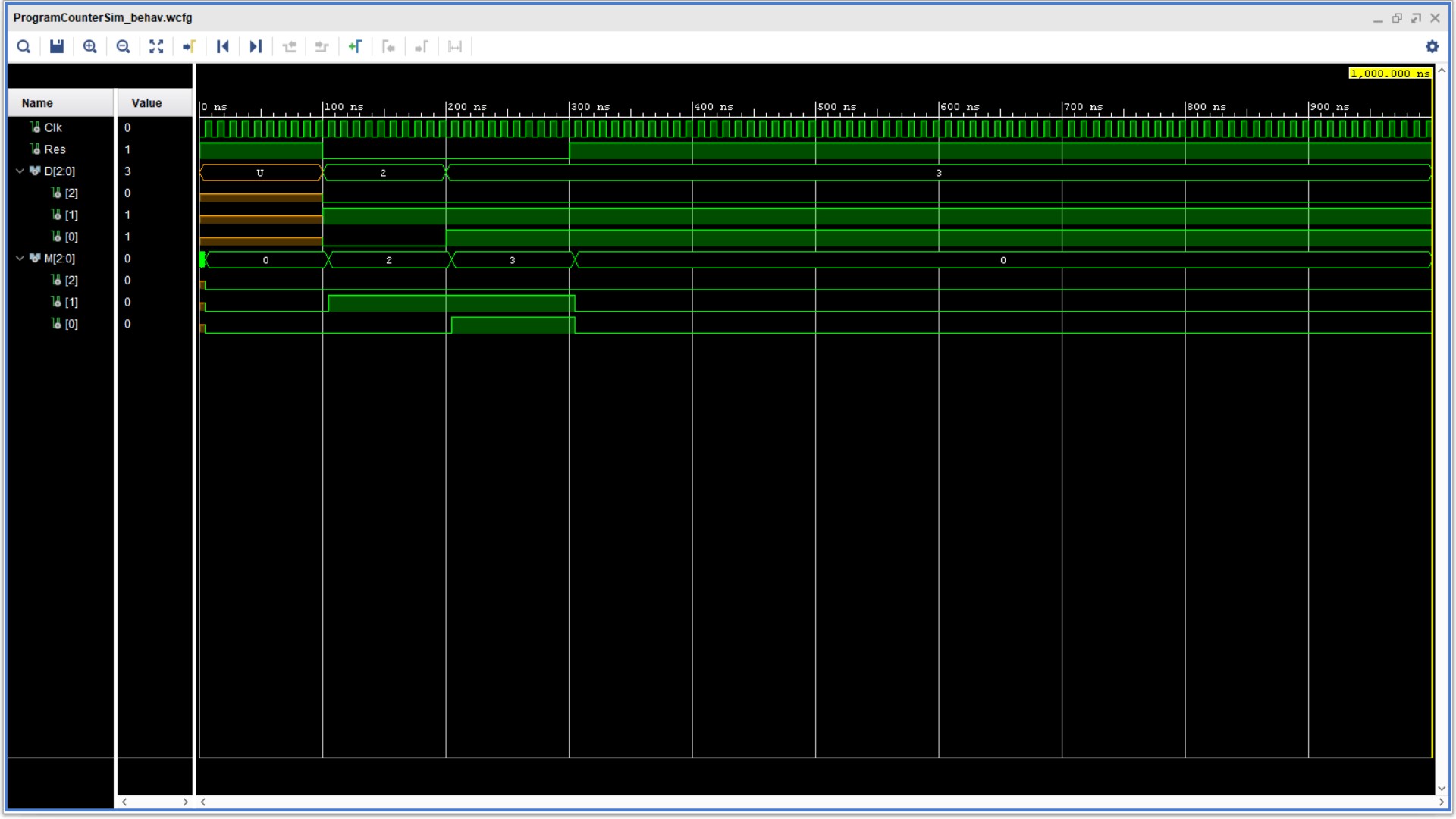


## VHDL File





## Timing Diagram

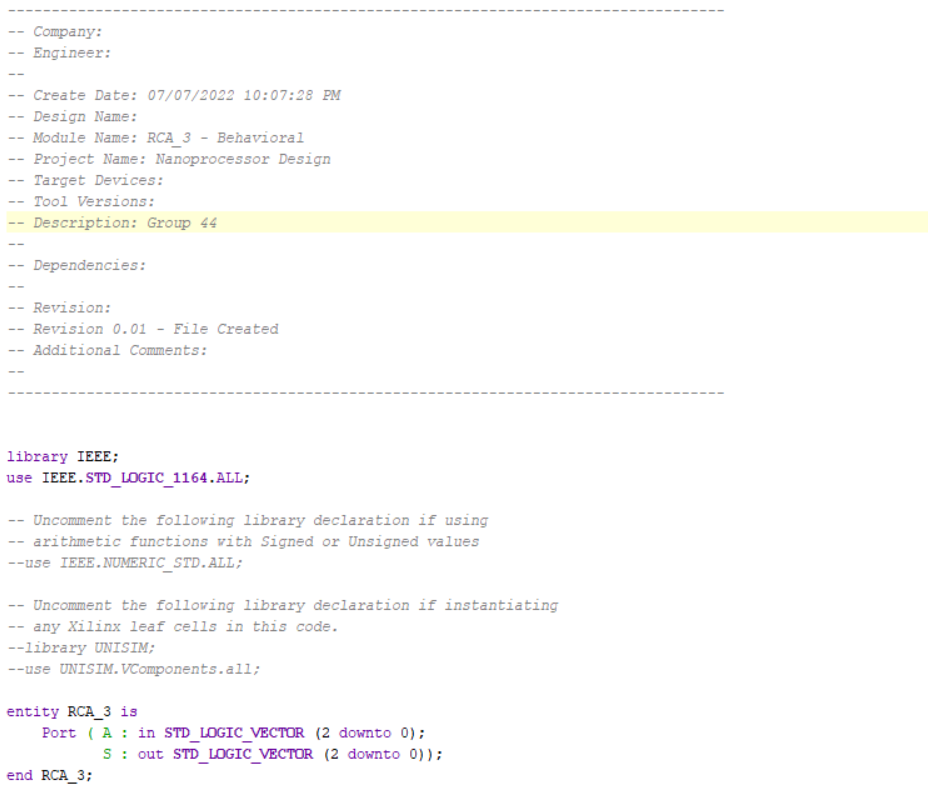


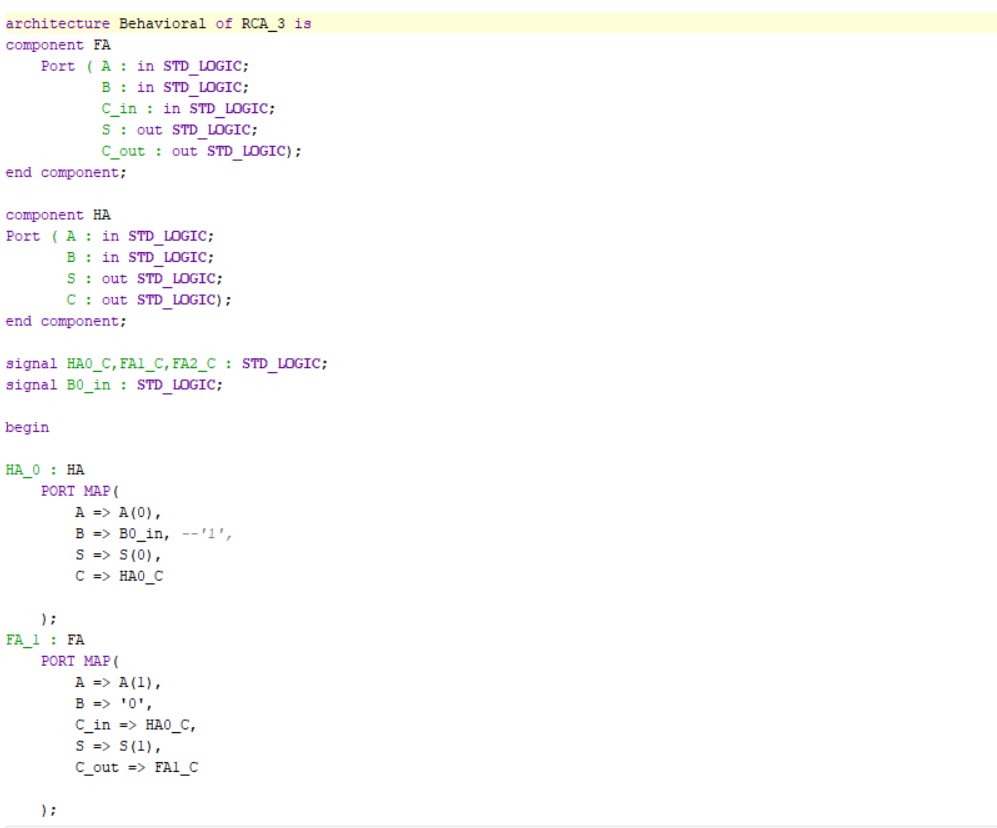
# 3-bit Ripple Carry Adder (Incremental)

## Schematic



## VHDL File





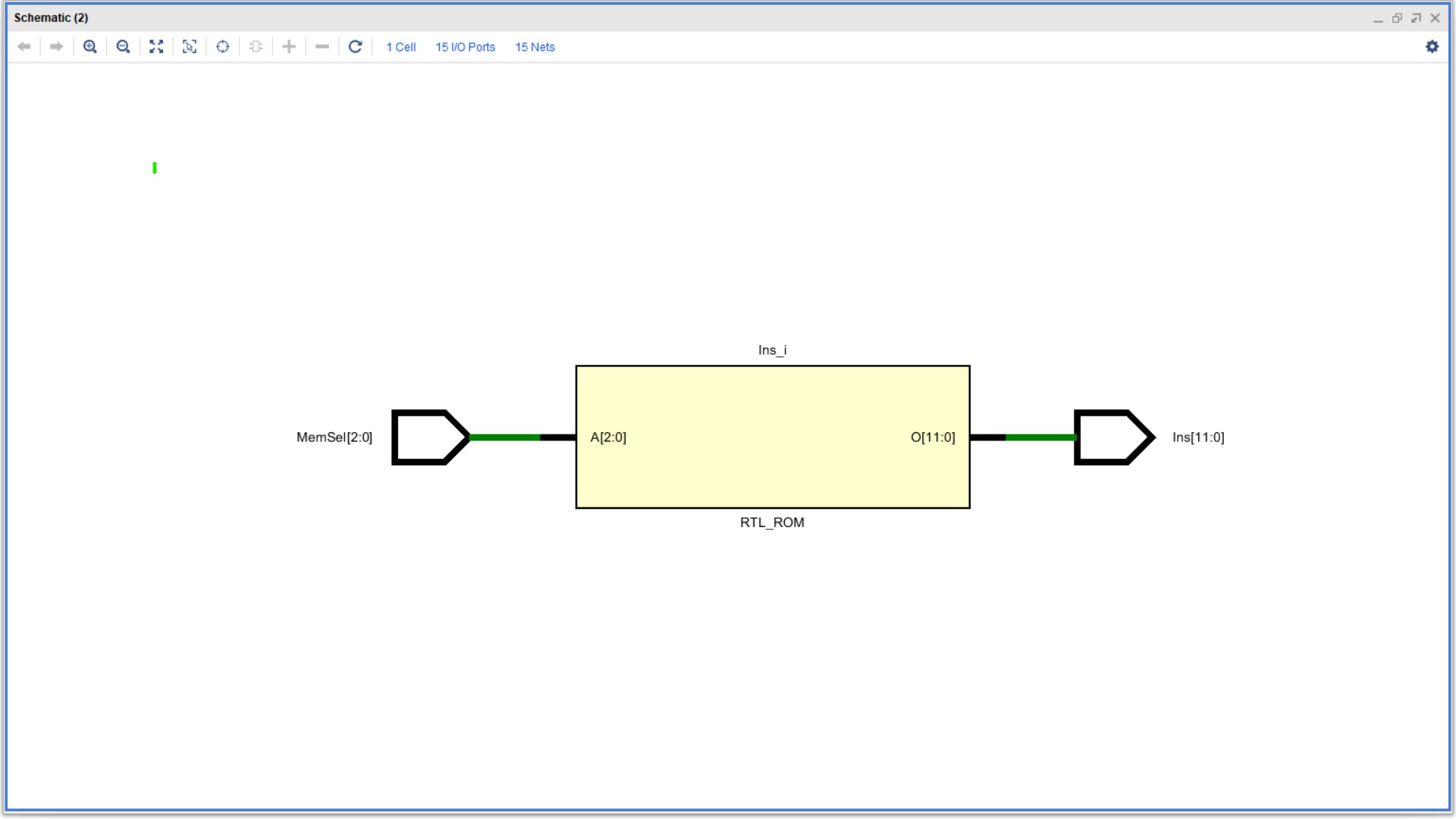


## Timing Diagram

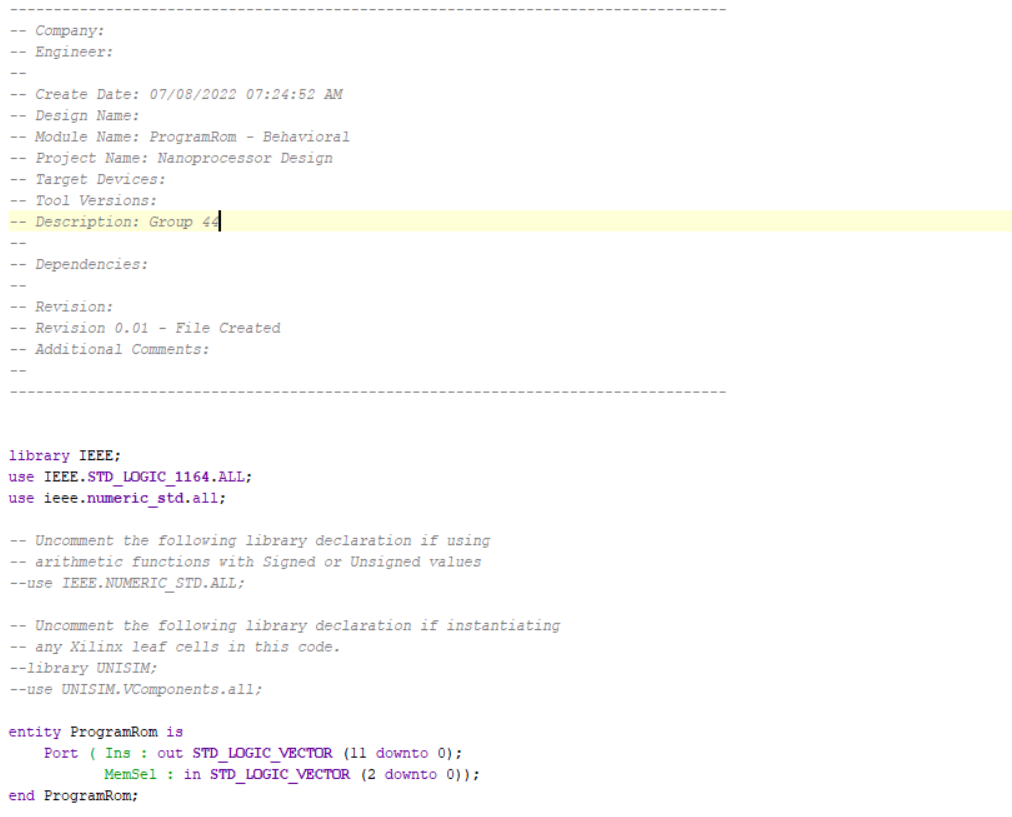
## 

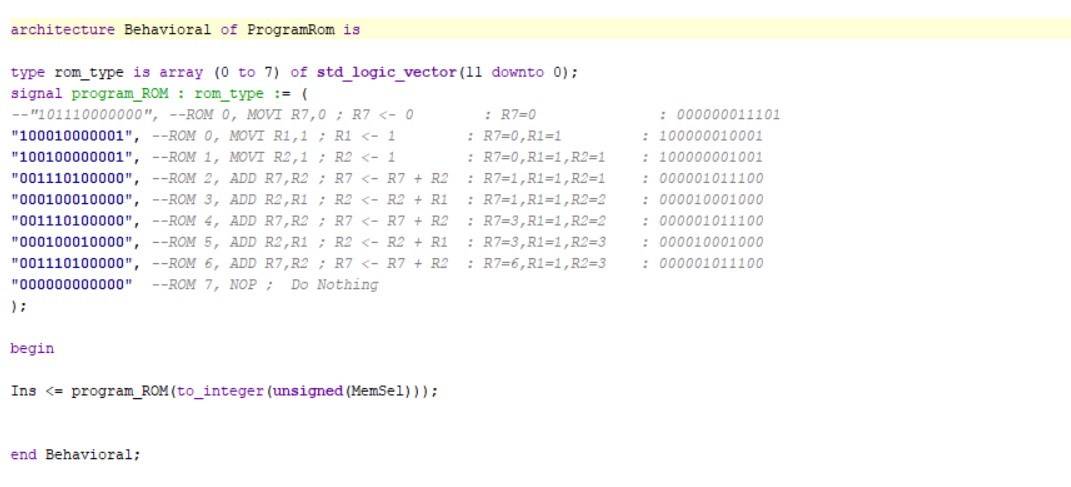
# Program Rom

## Schematic

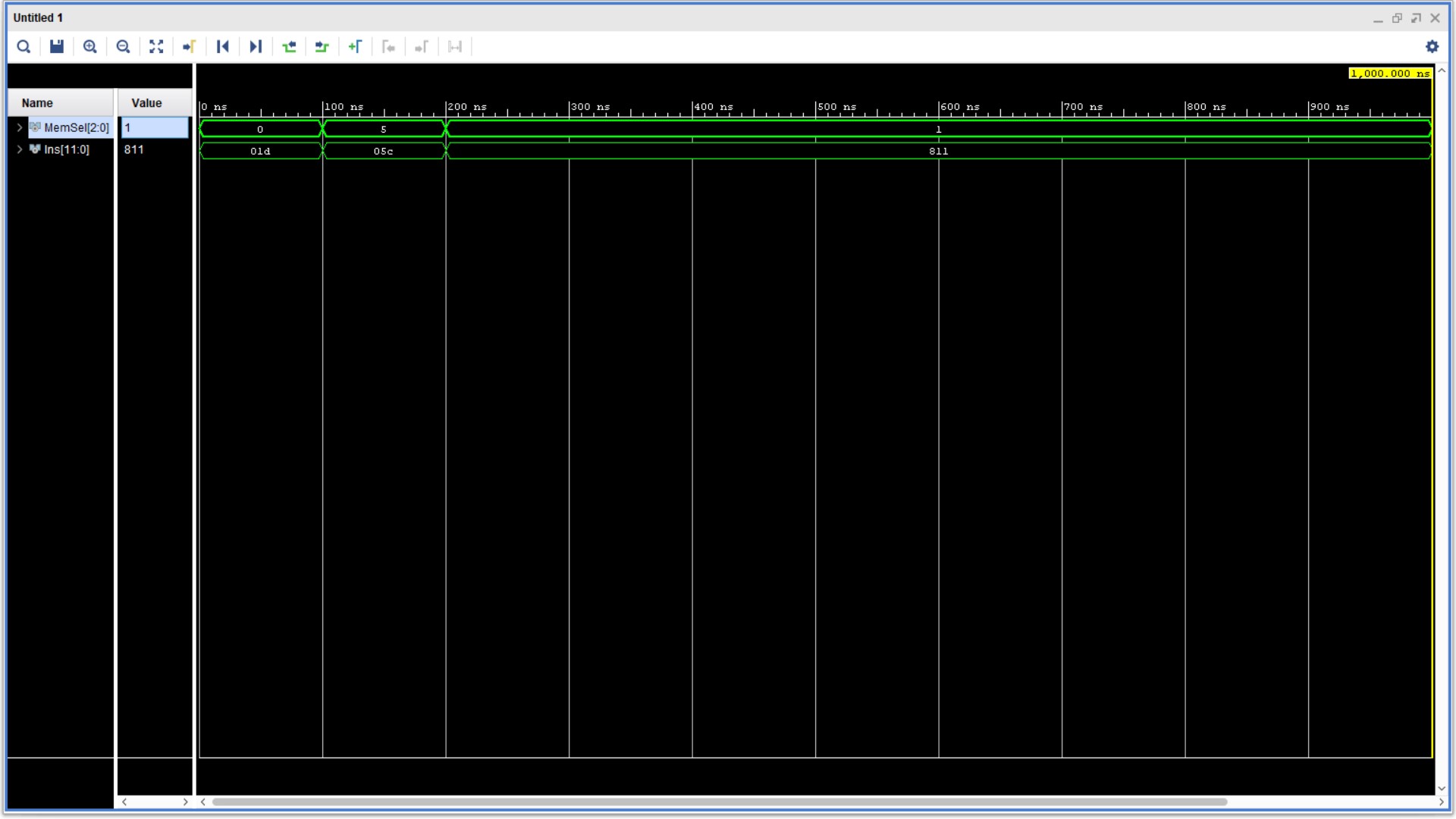


## VHDL File



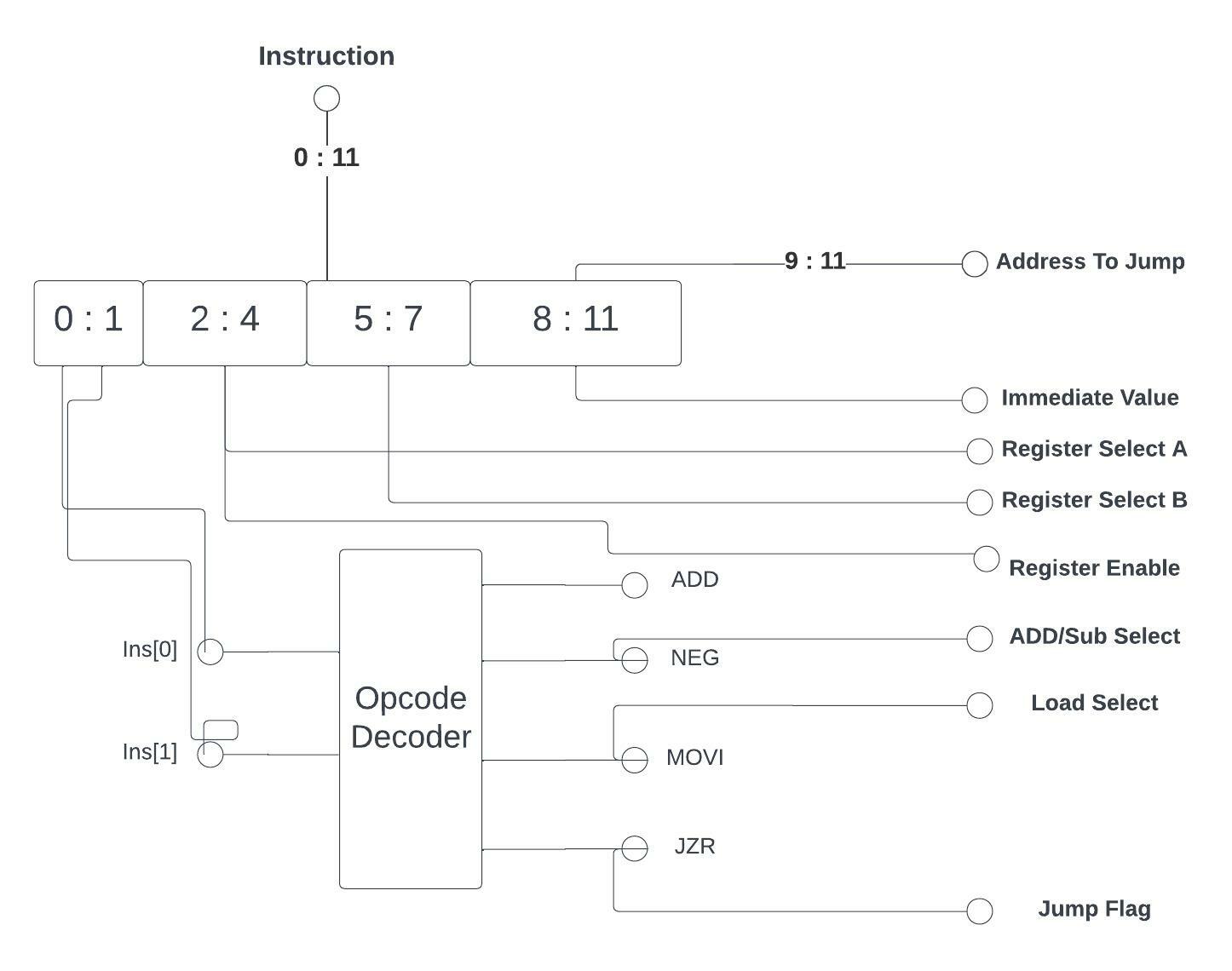


## Timing Diagram



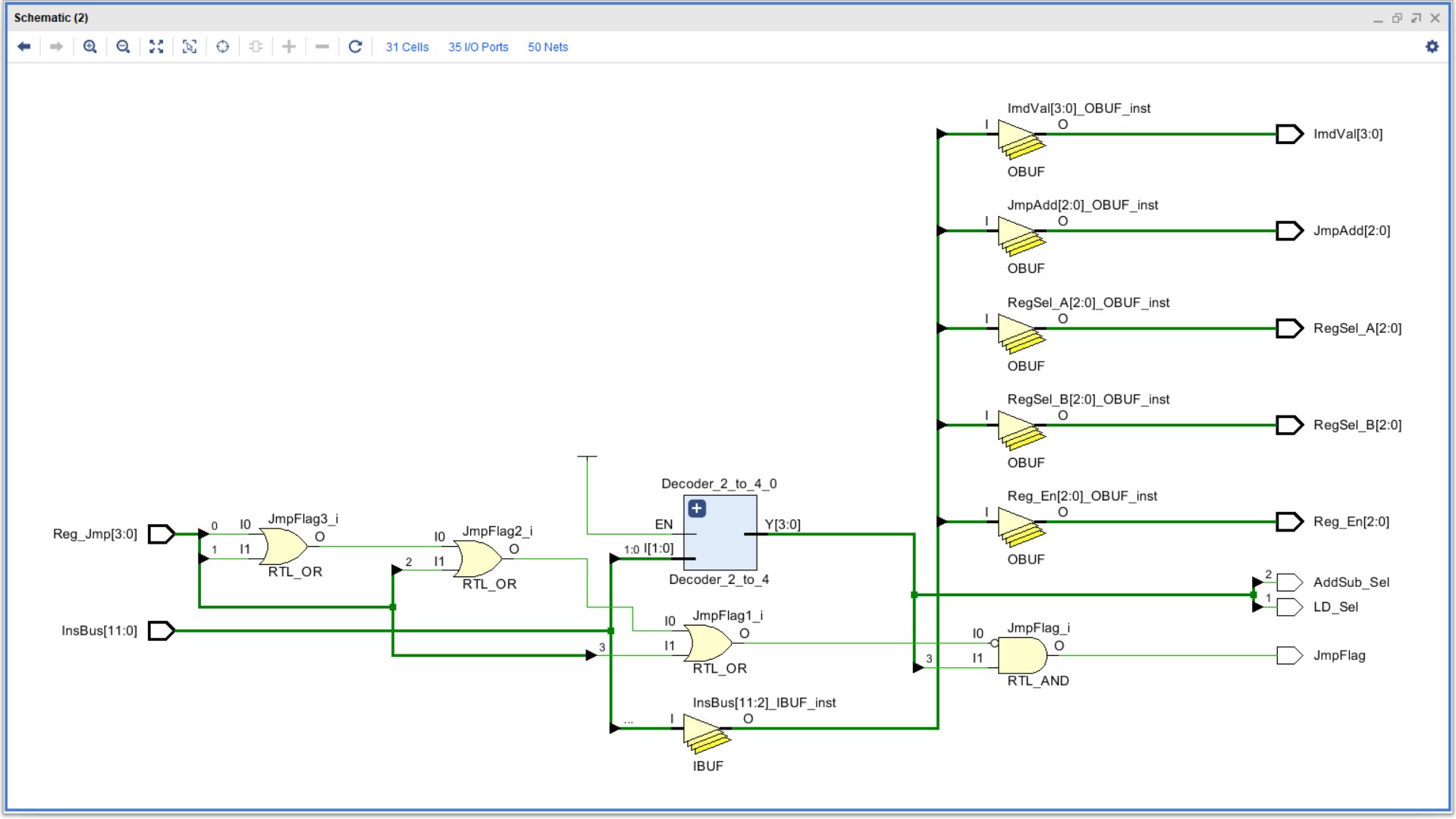
# Instruction Decoder

## Block Diagram

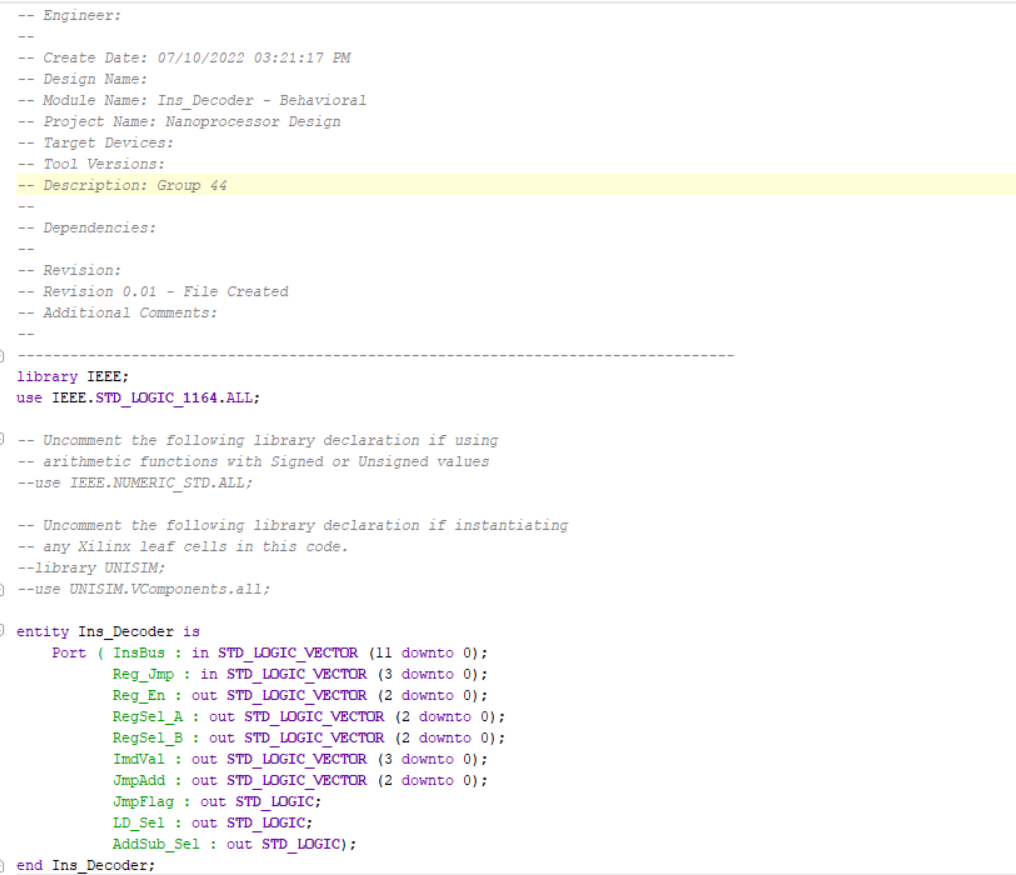


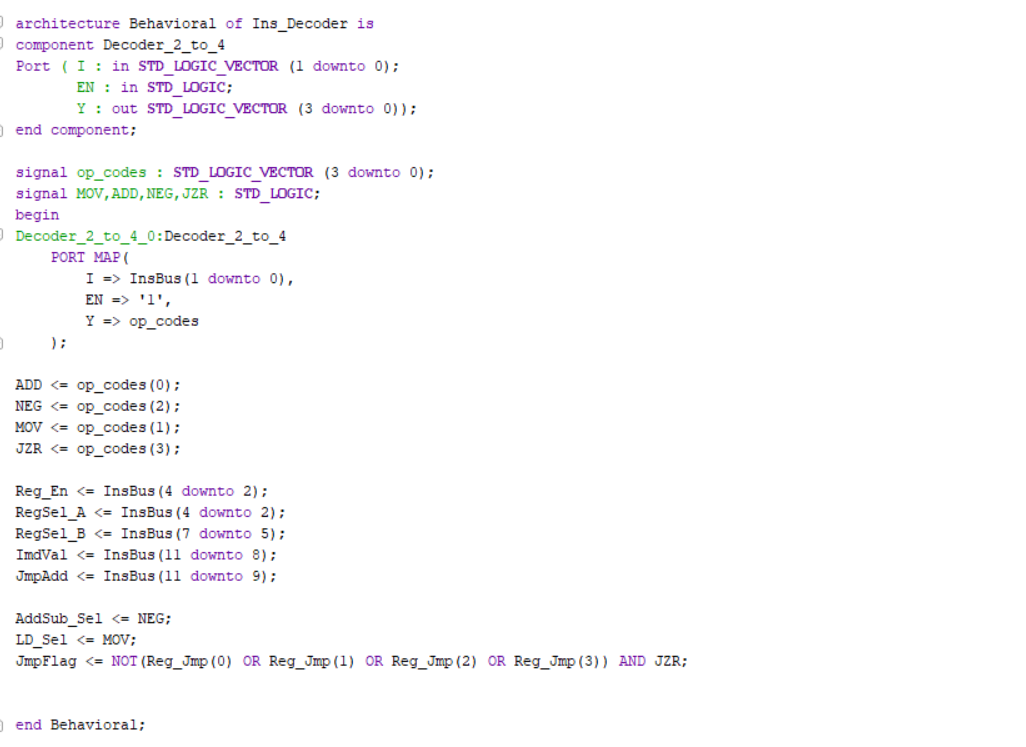
# 

## Schematic

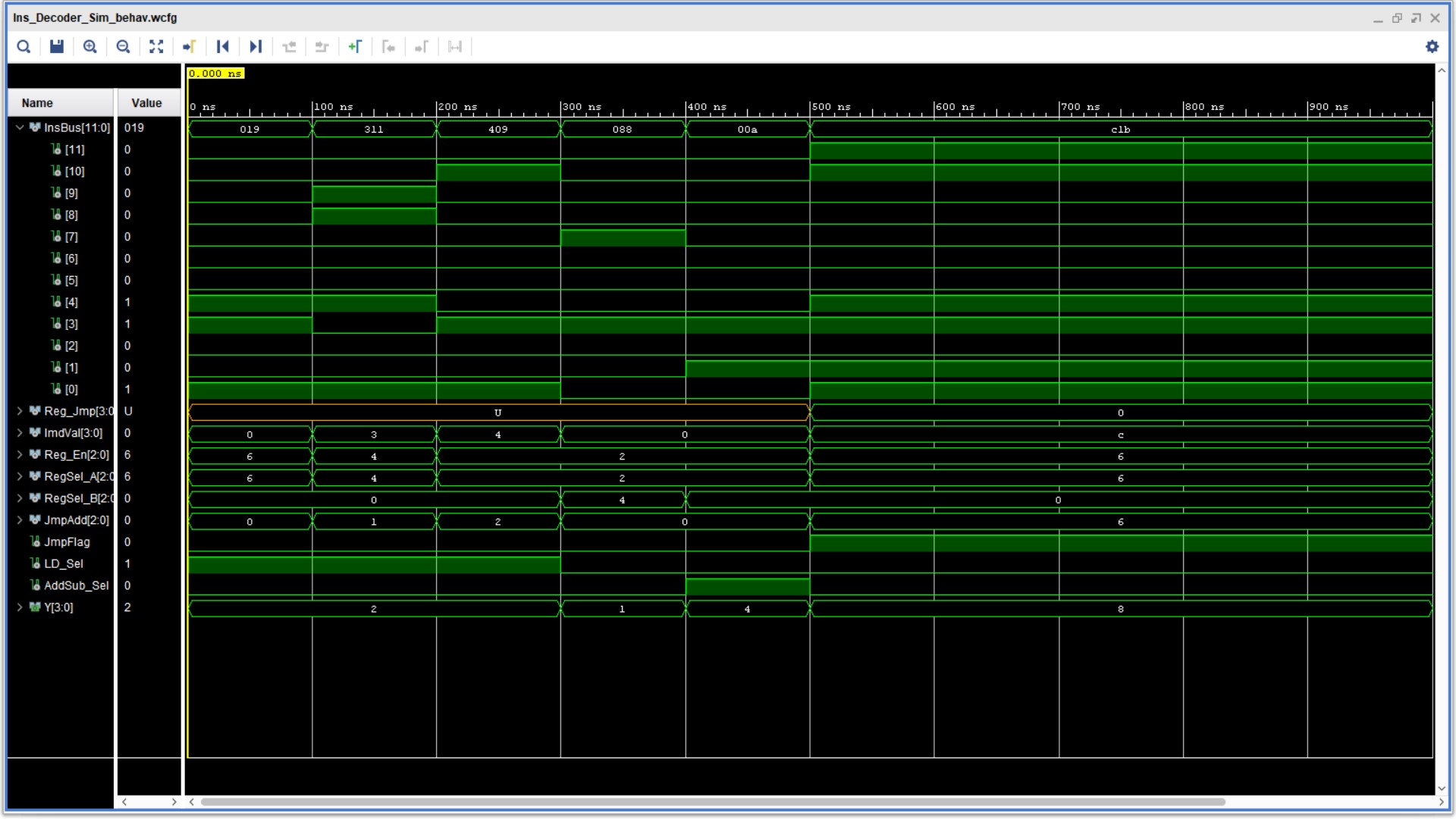


## VHDL File





## Timing Diagram

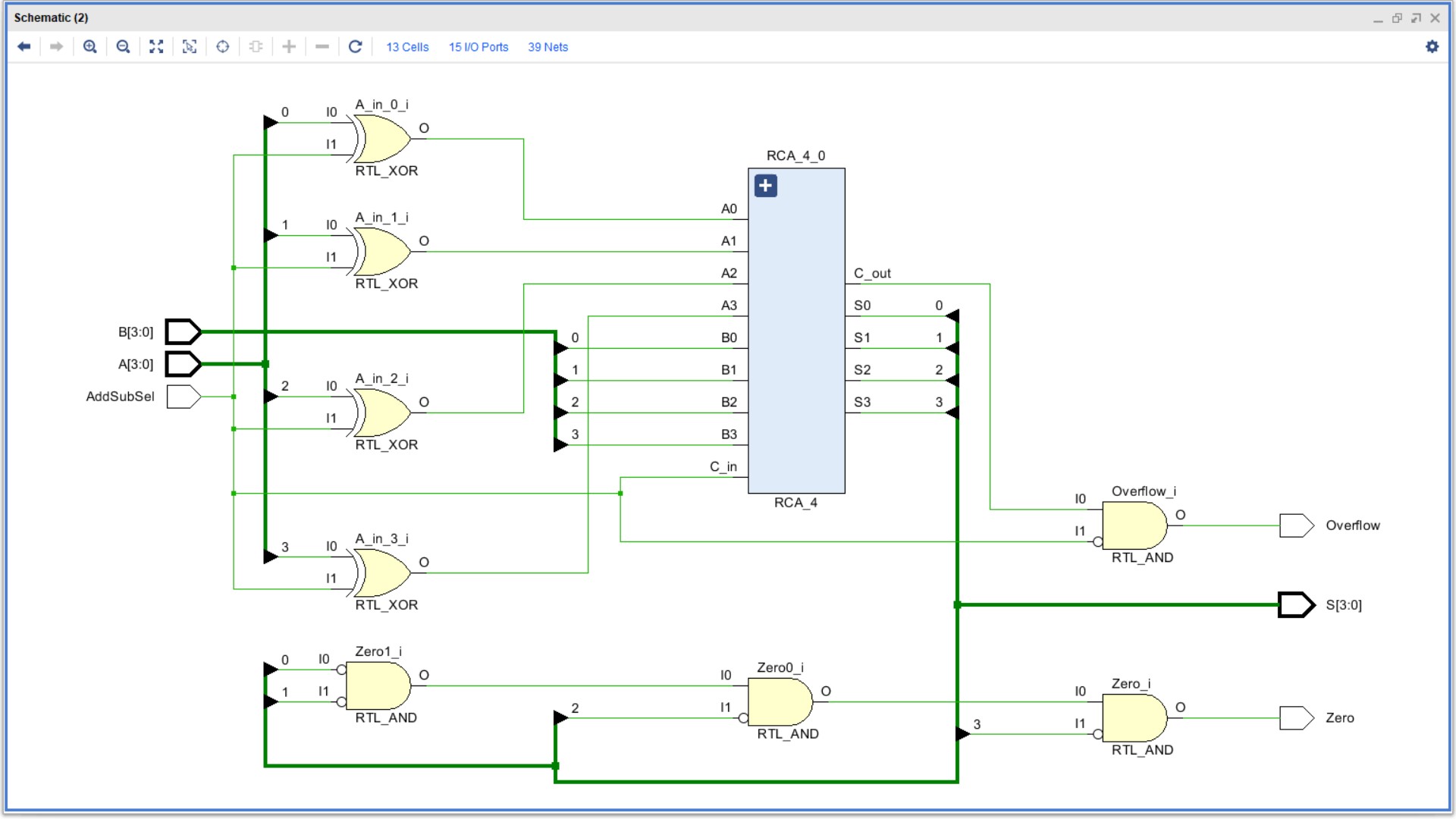


# Arithmetic Unit

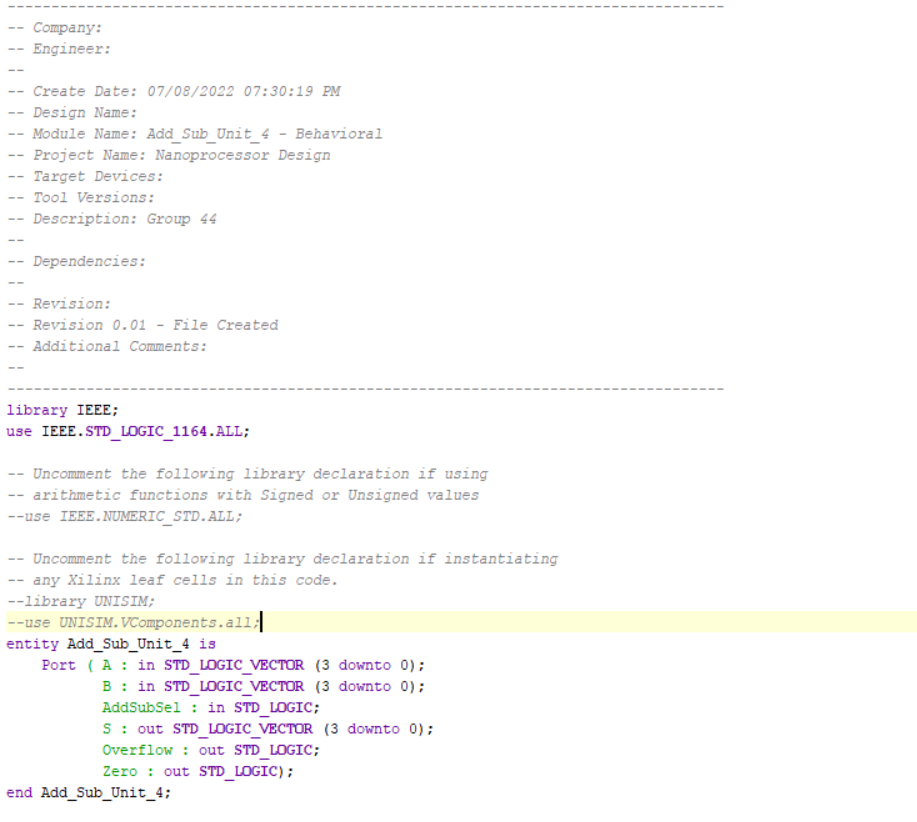
## Block Diagram

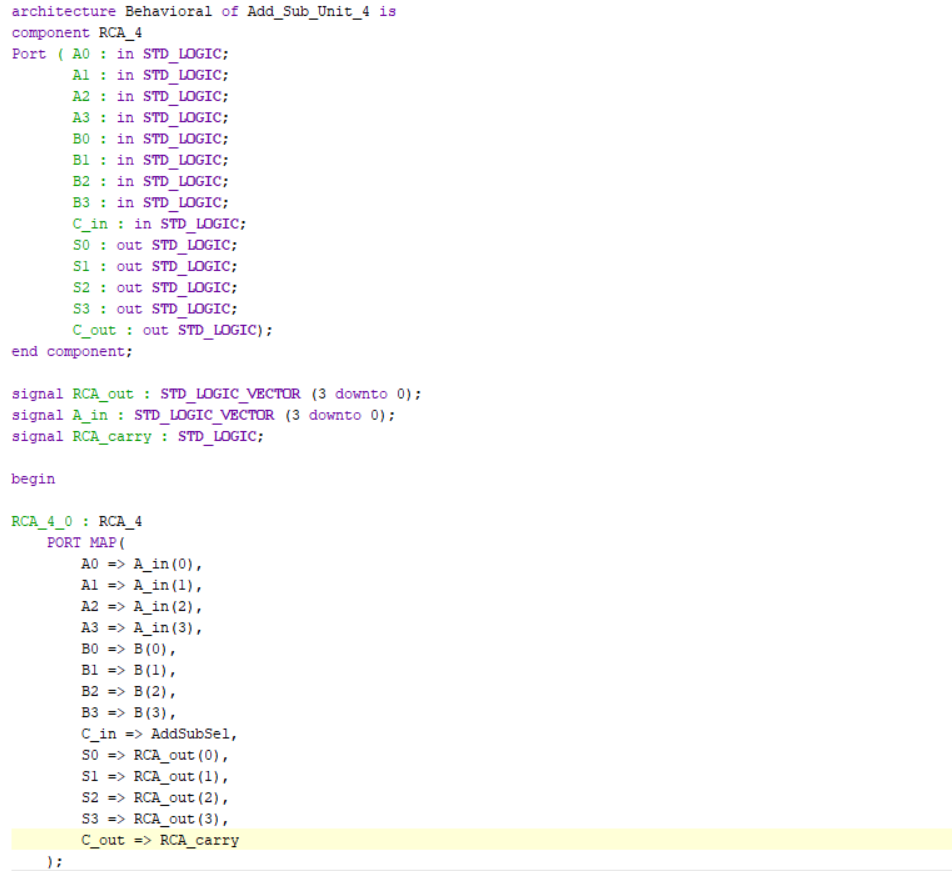


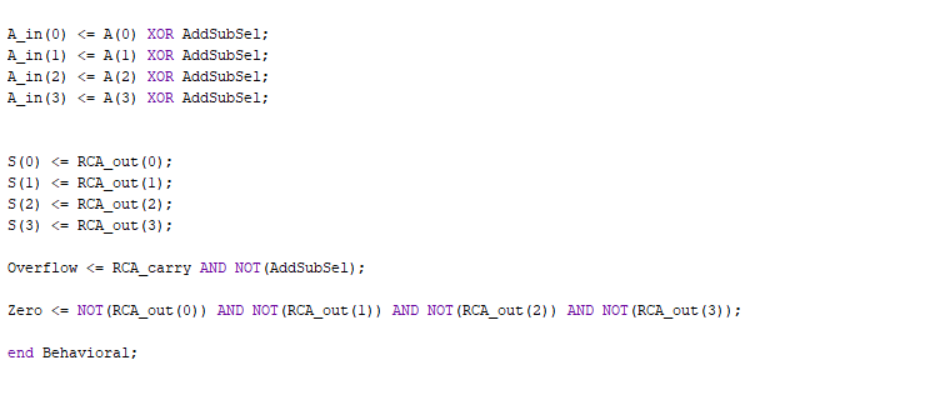
## Schematic



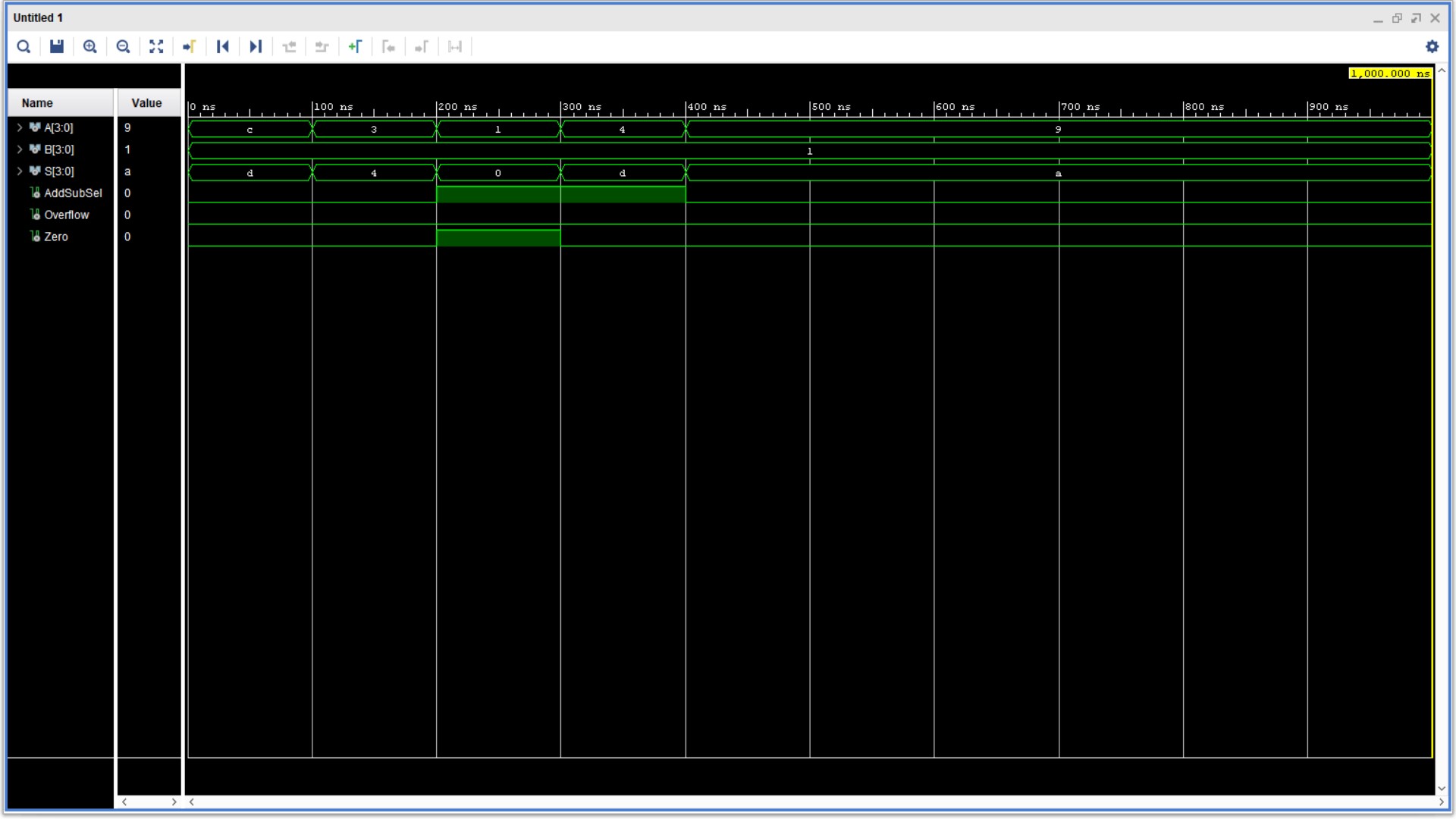
## VHDL File





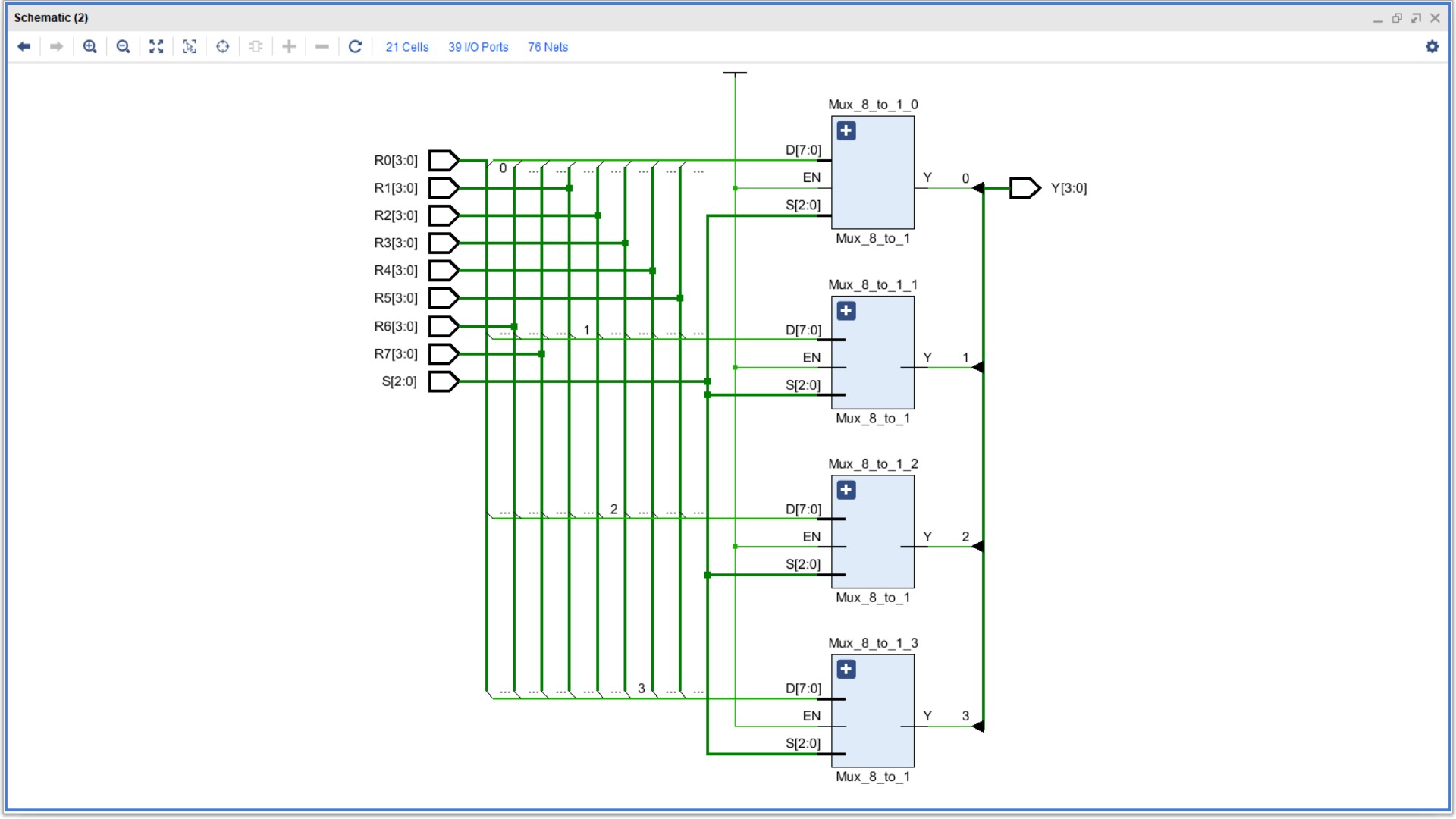


## Timing Diagram

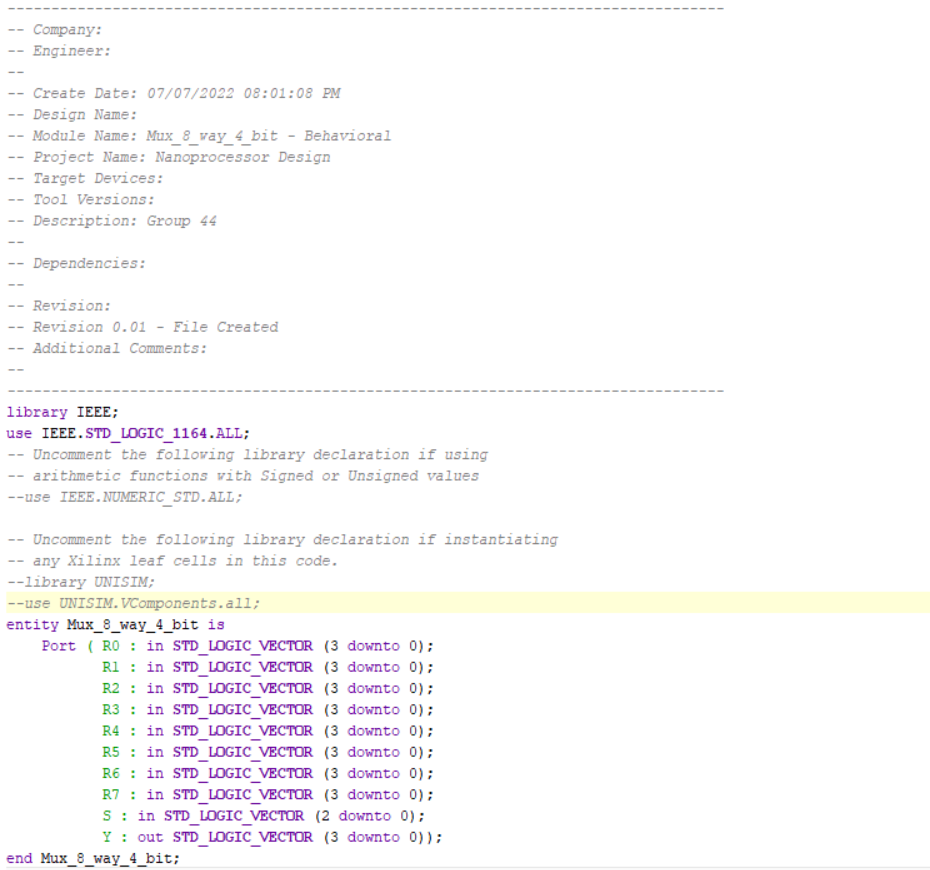


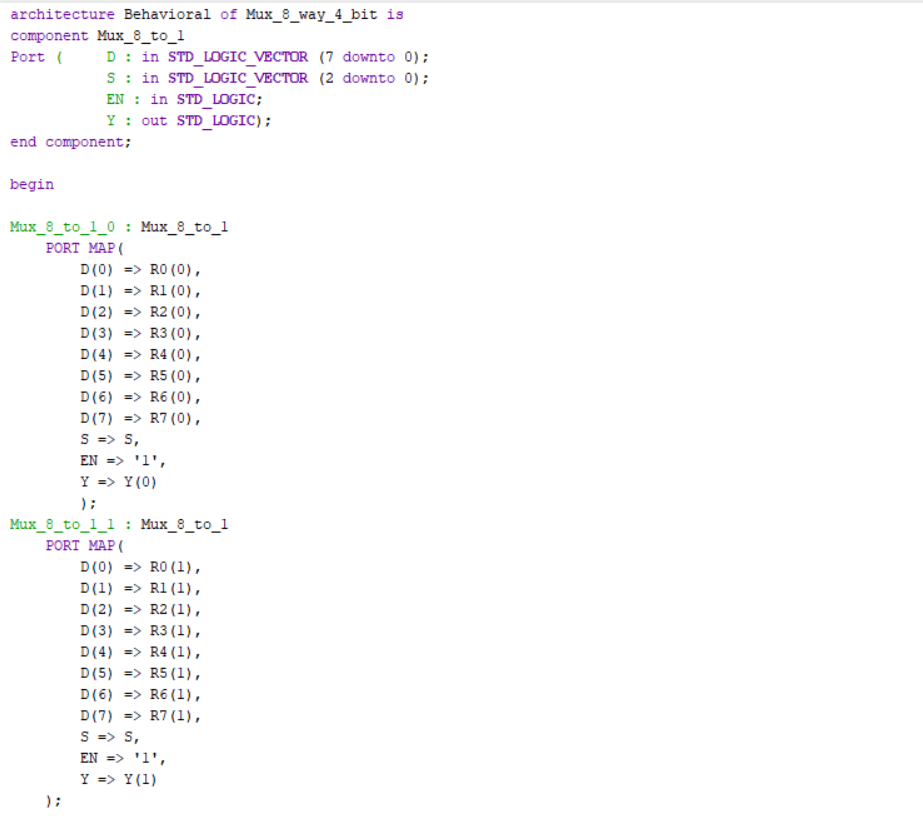
# 8 Way 4-bit Multiplexer

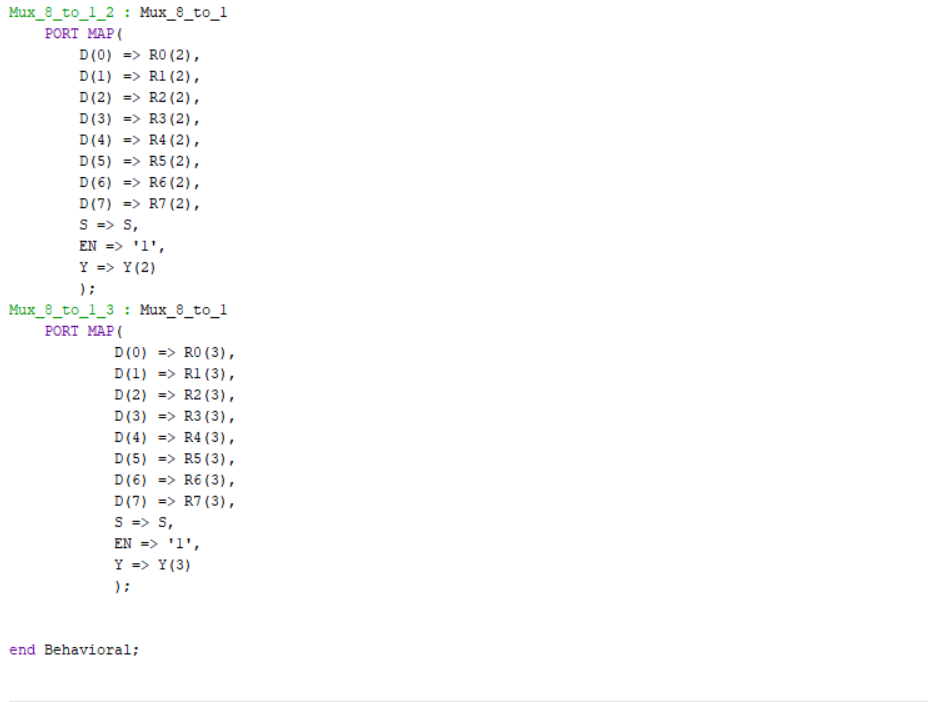
## Schematic



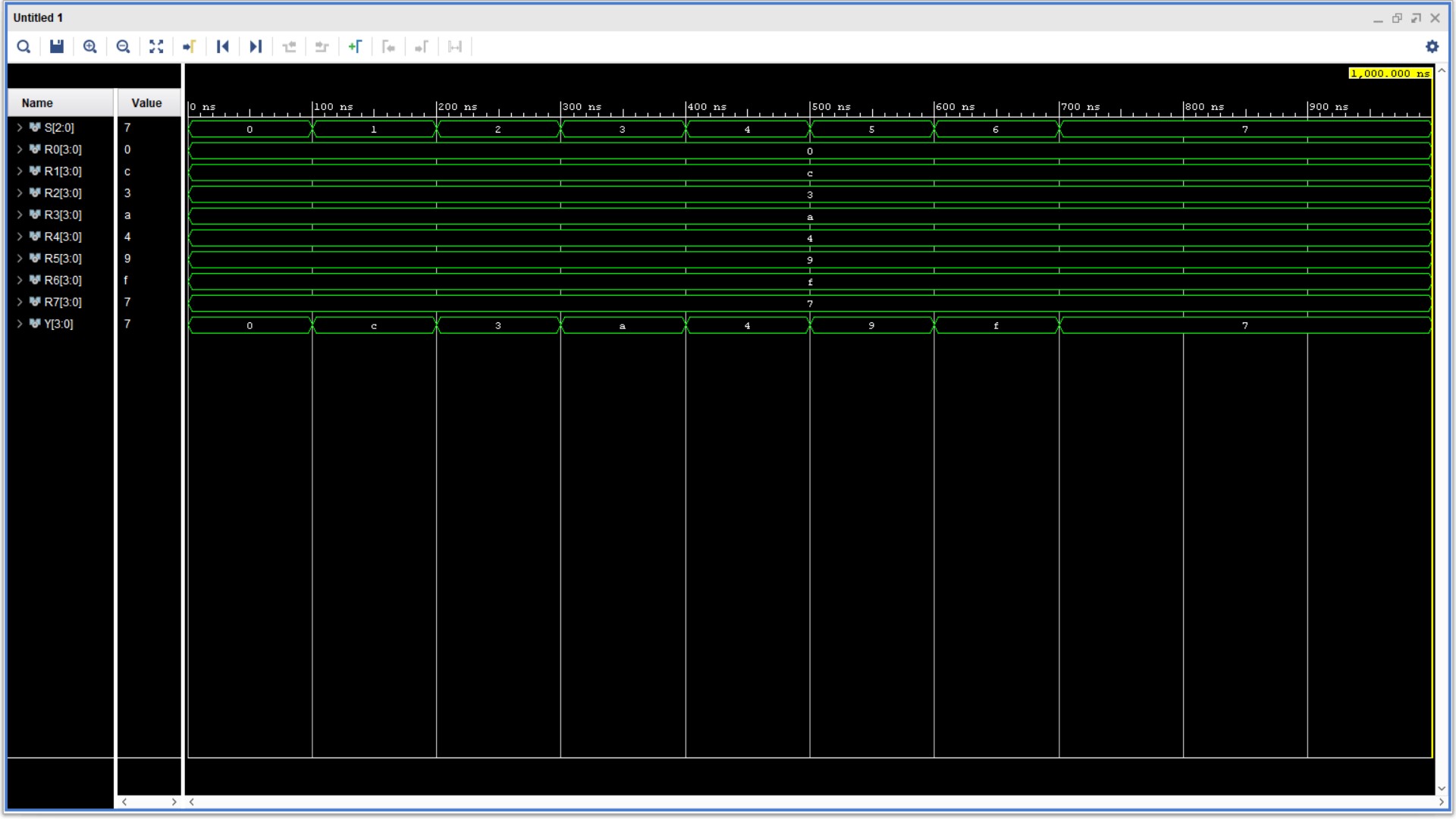
## VHDL File





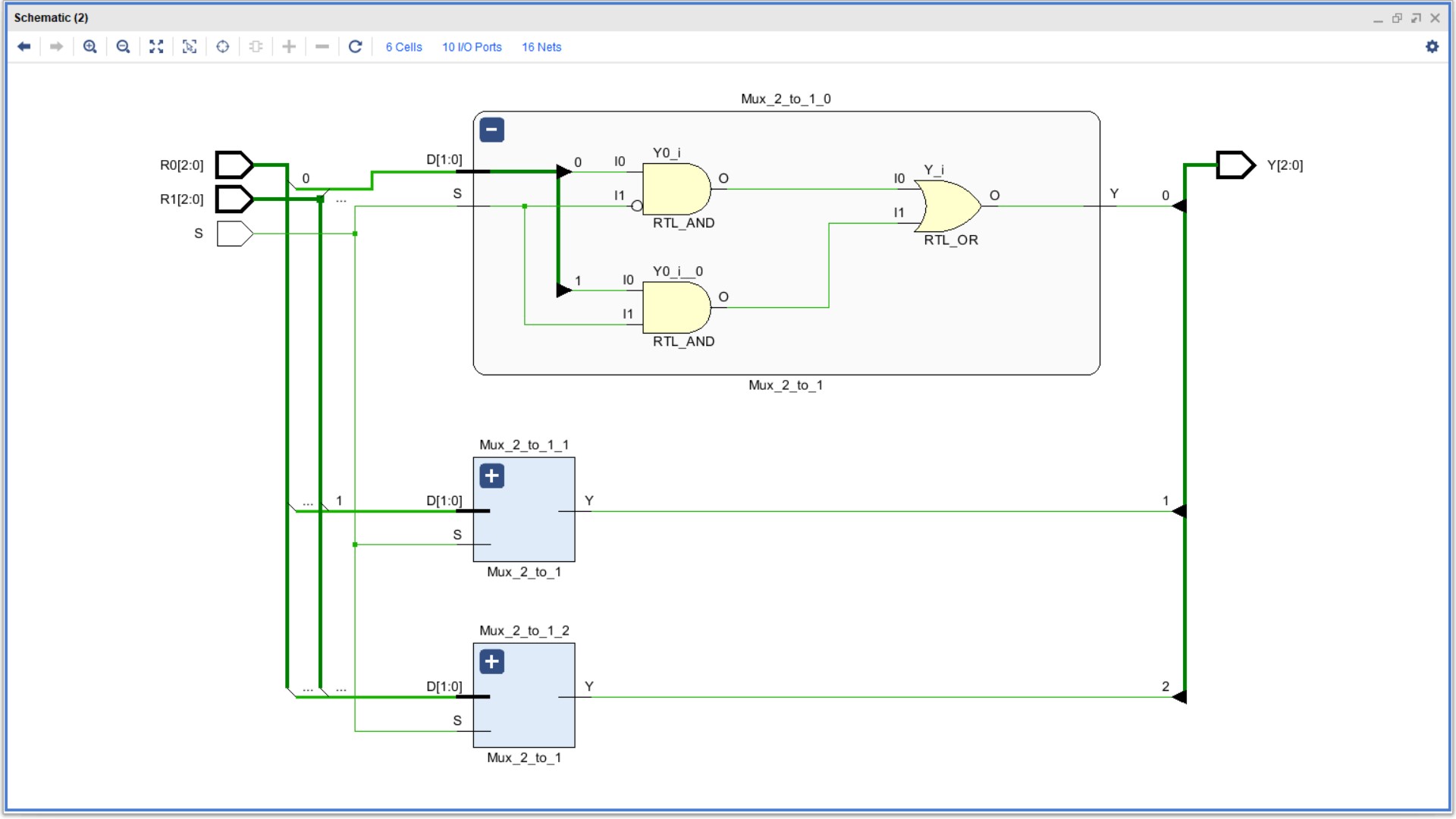


## Timing Diagram



# 2 way 3-bit Multiplexer

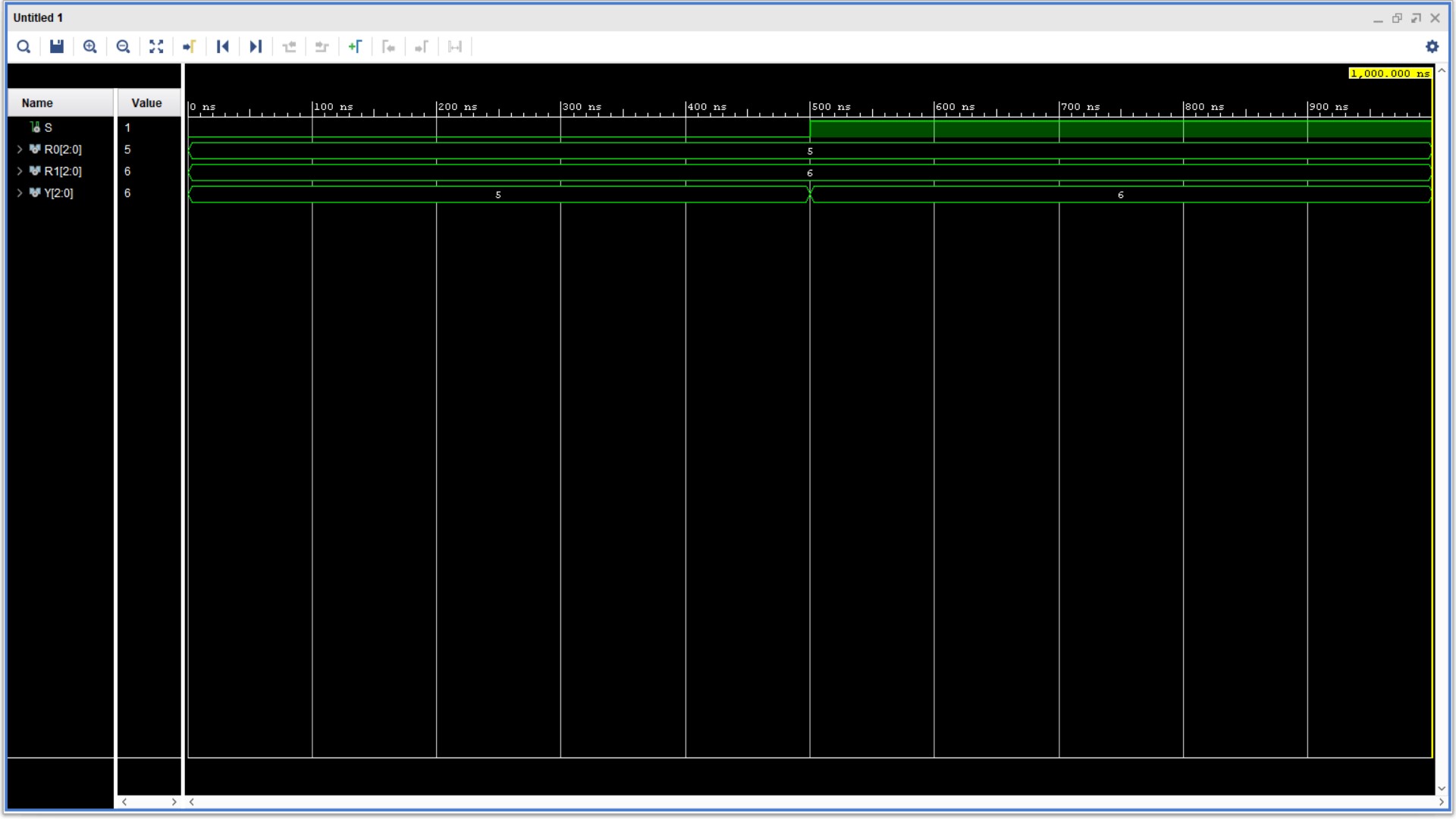
## Schematic



## VHDL File

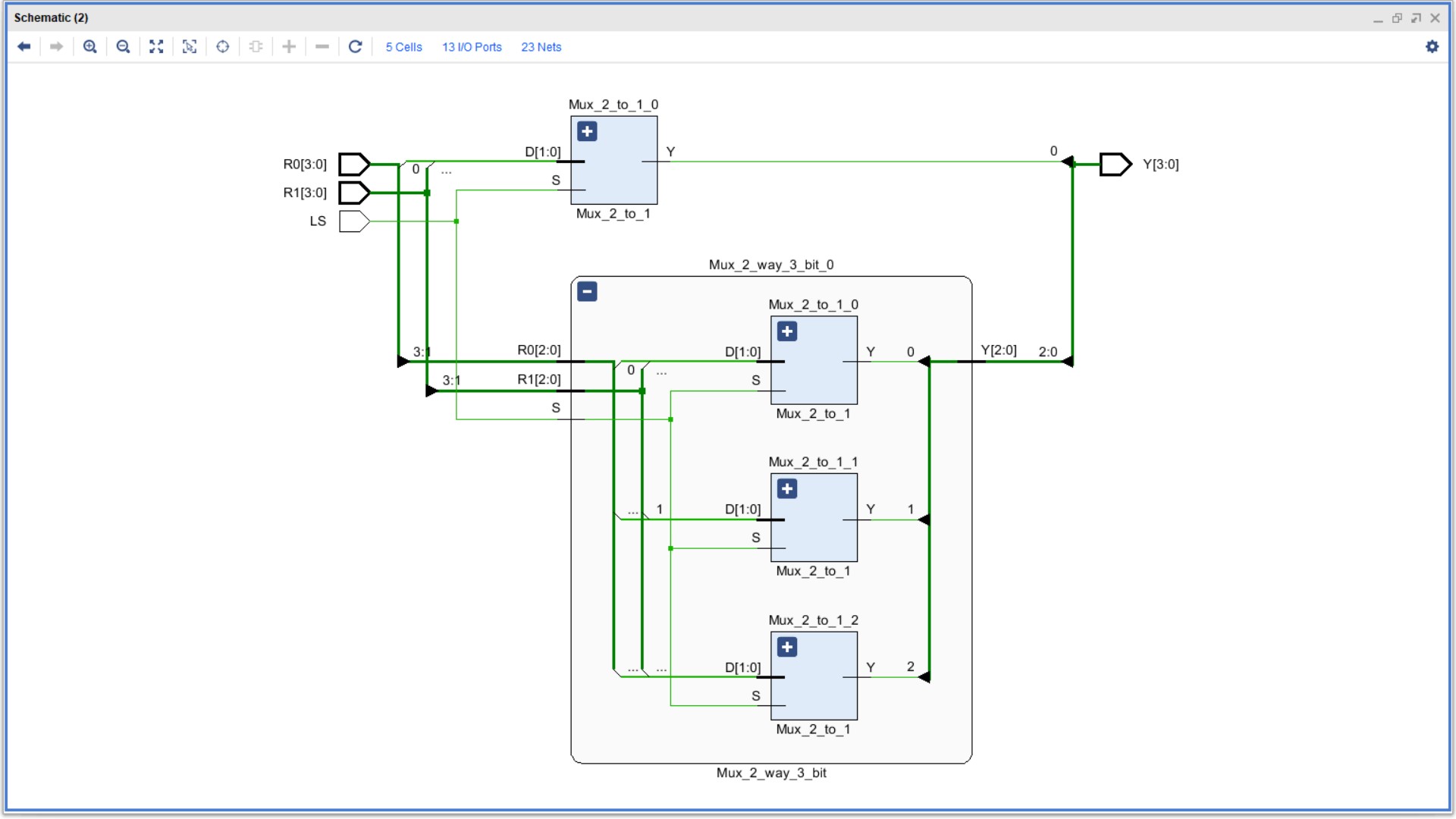
## 

## Timing Diagram

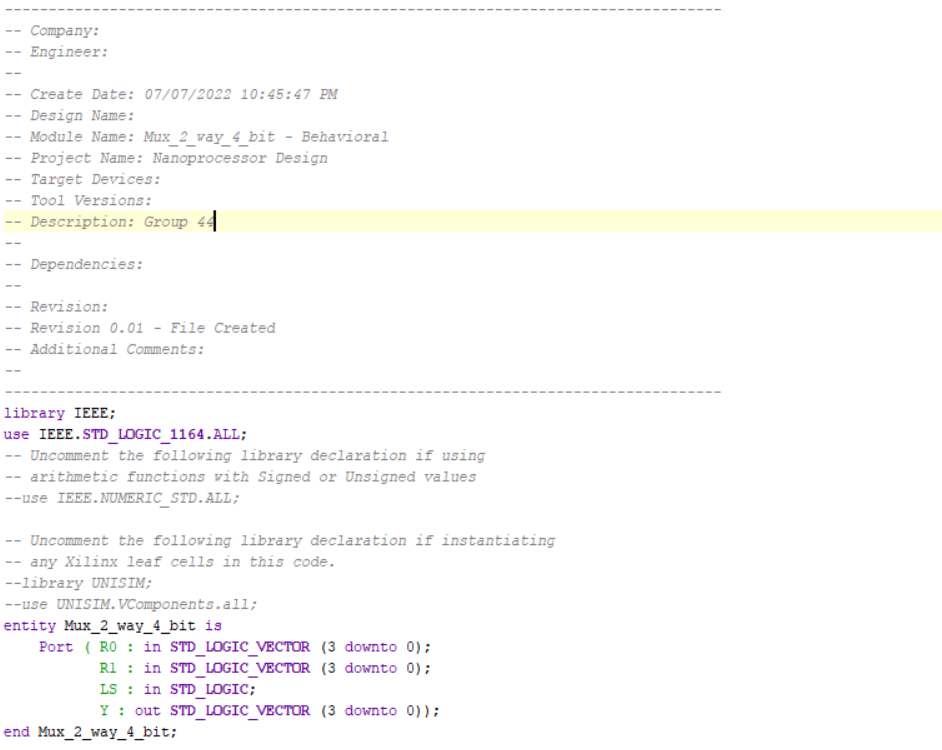


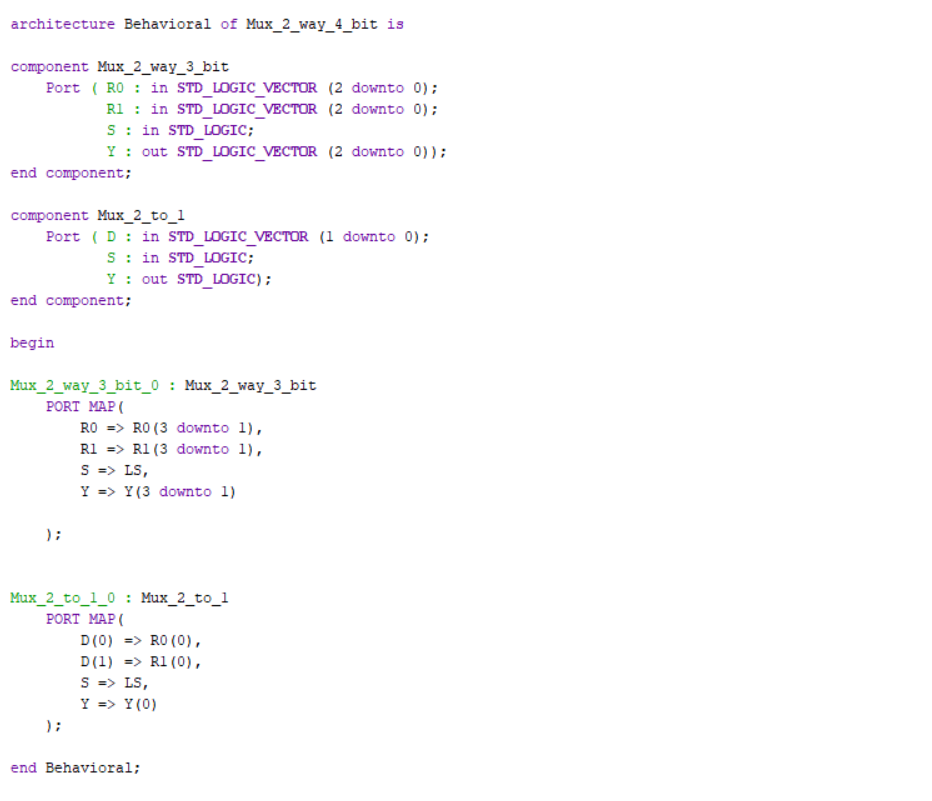
# 2 way 4-bit Multiplexer

## Schematic

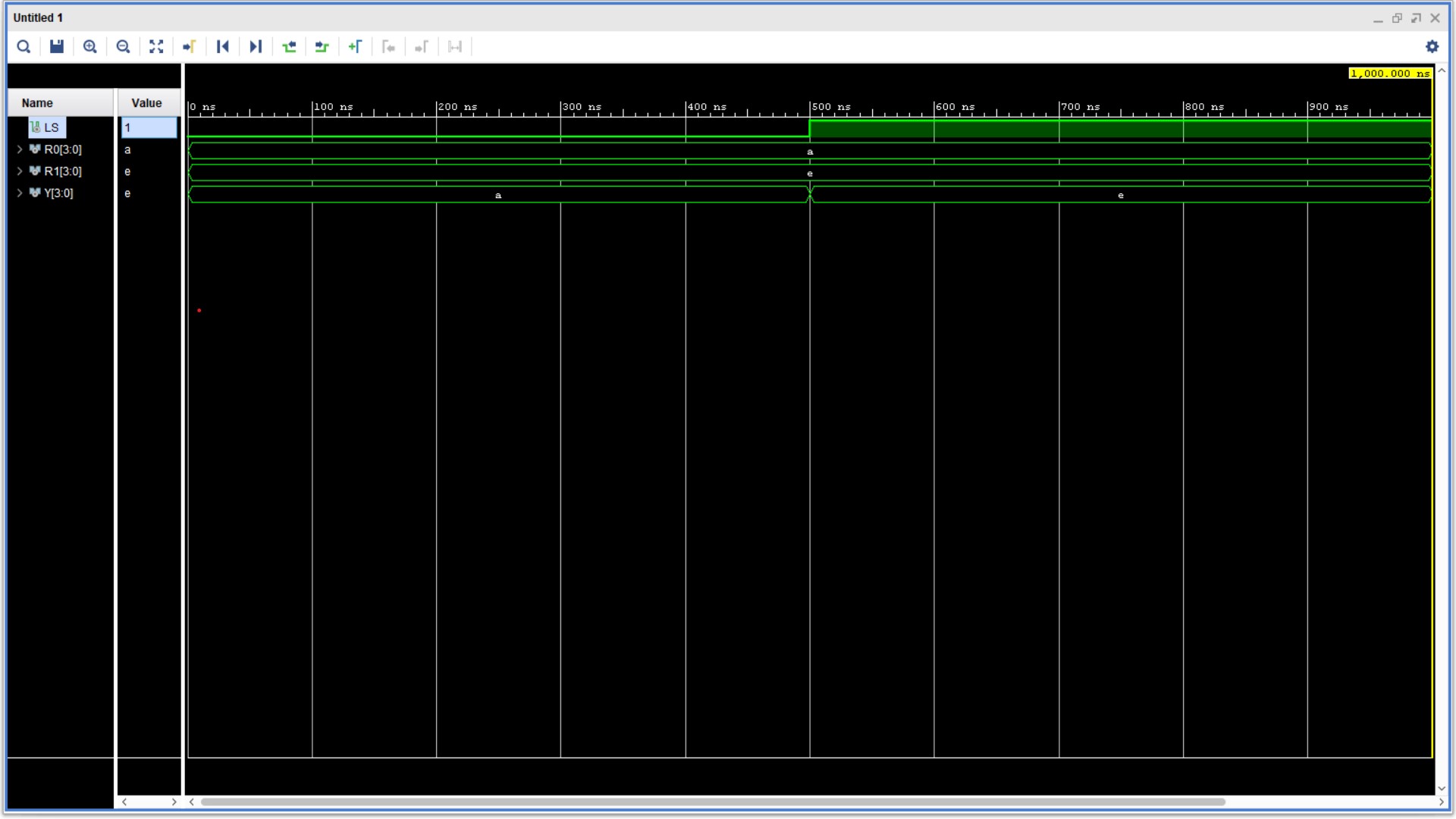


## VHDL File



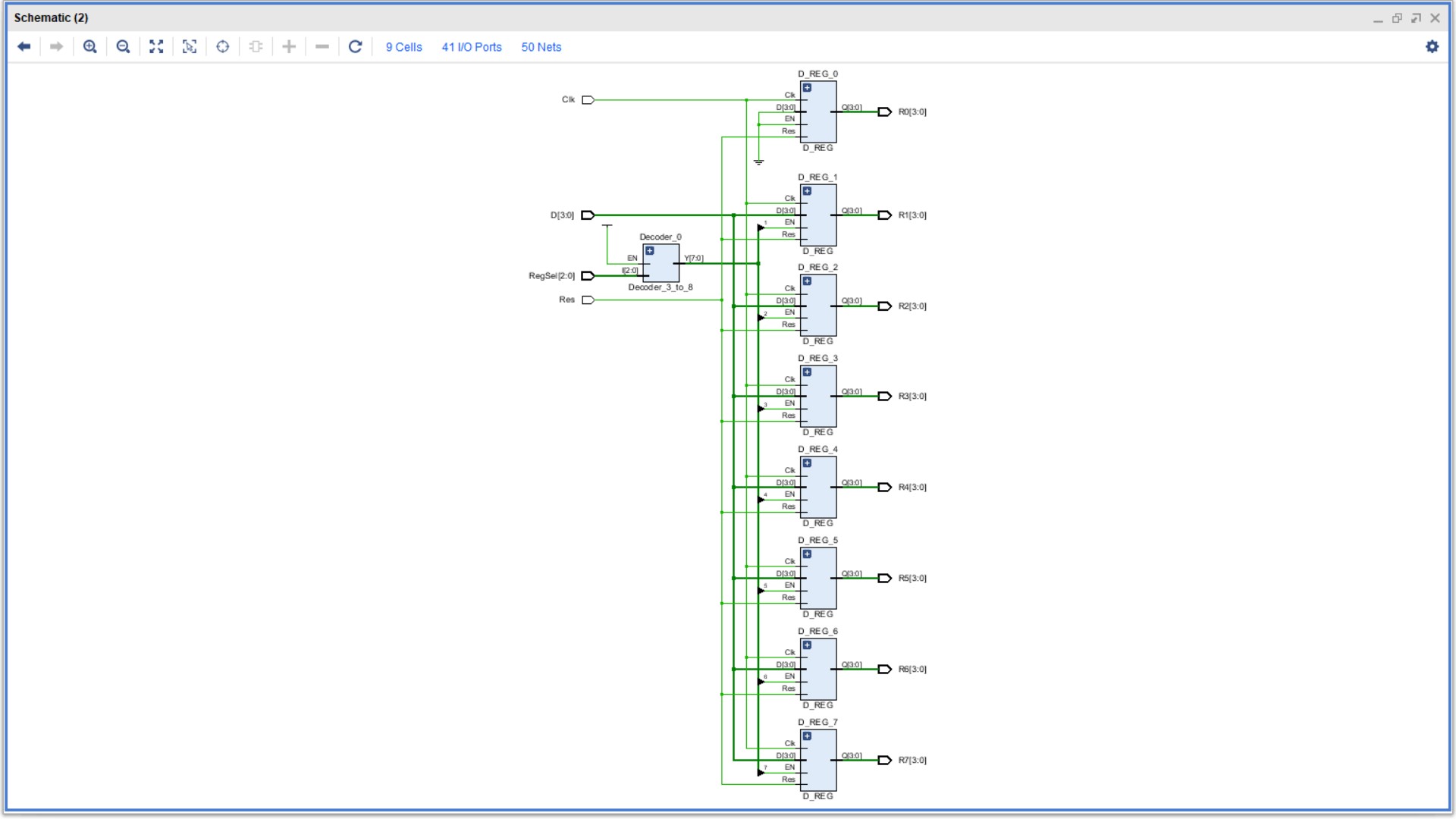


## Timing Diagram

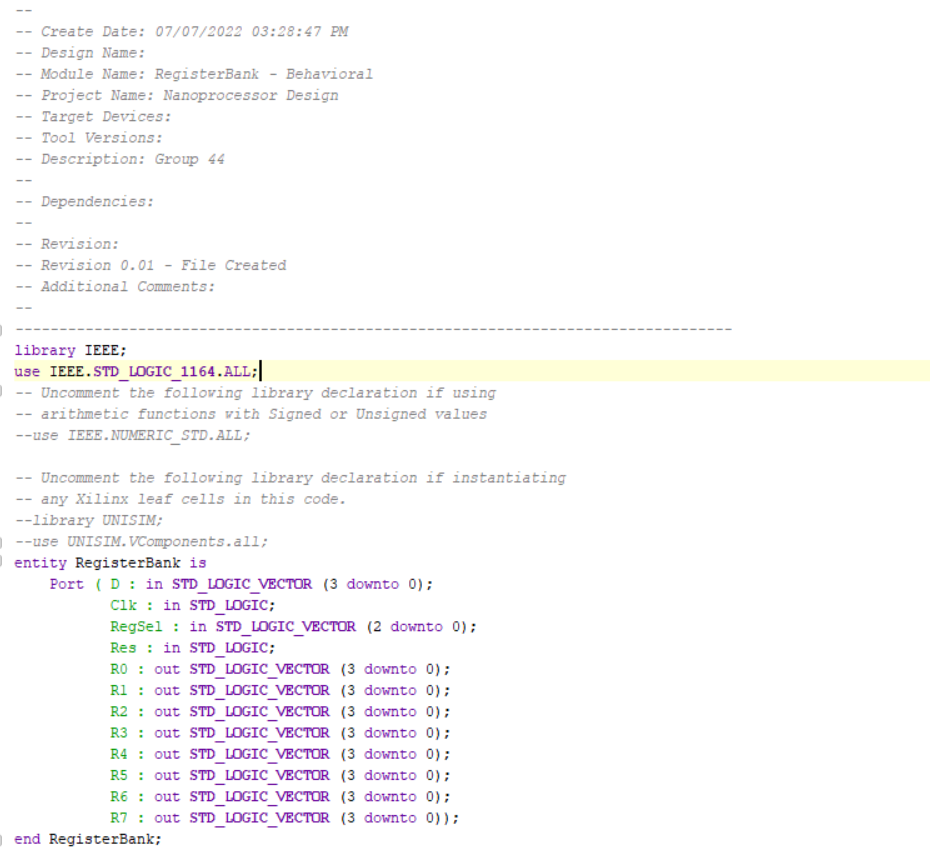


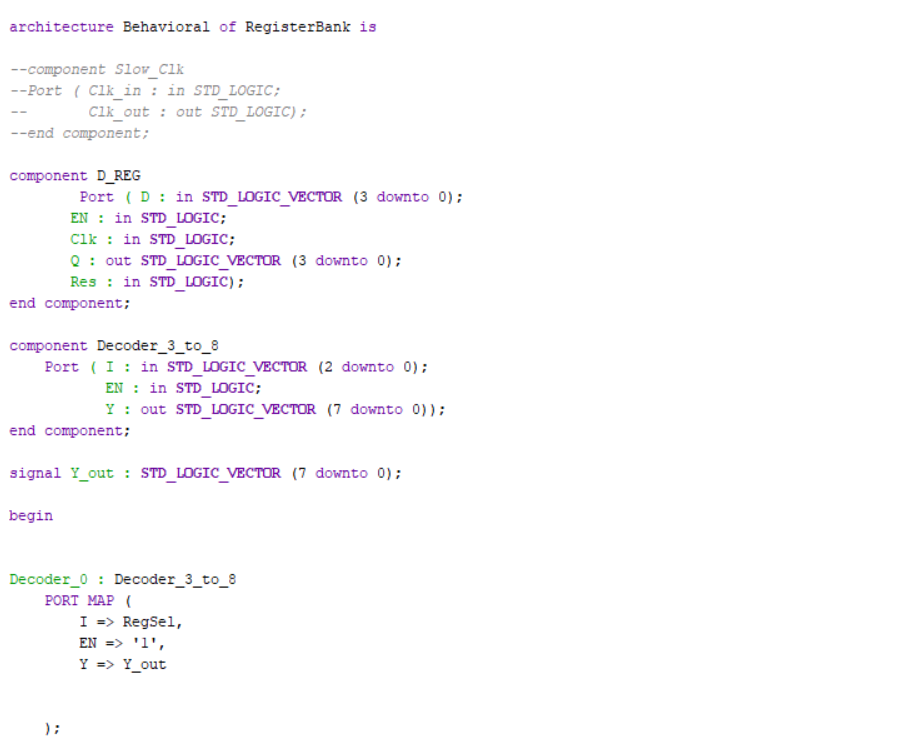
# Register Bank

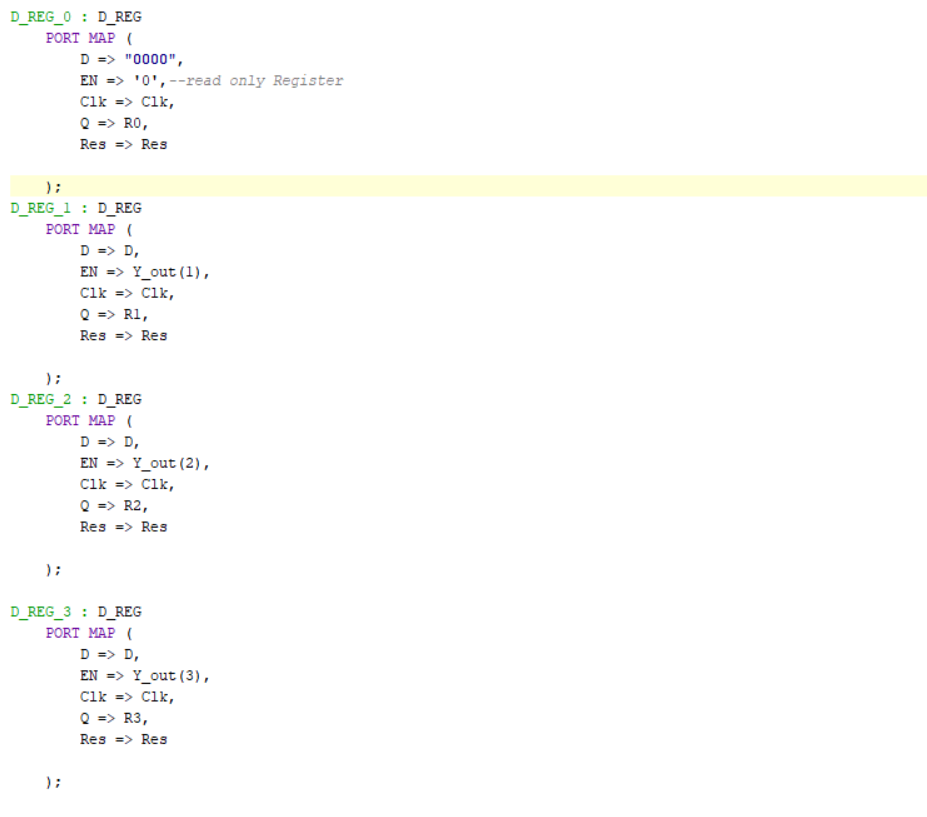
## Schematic

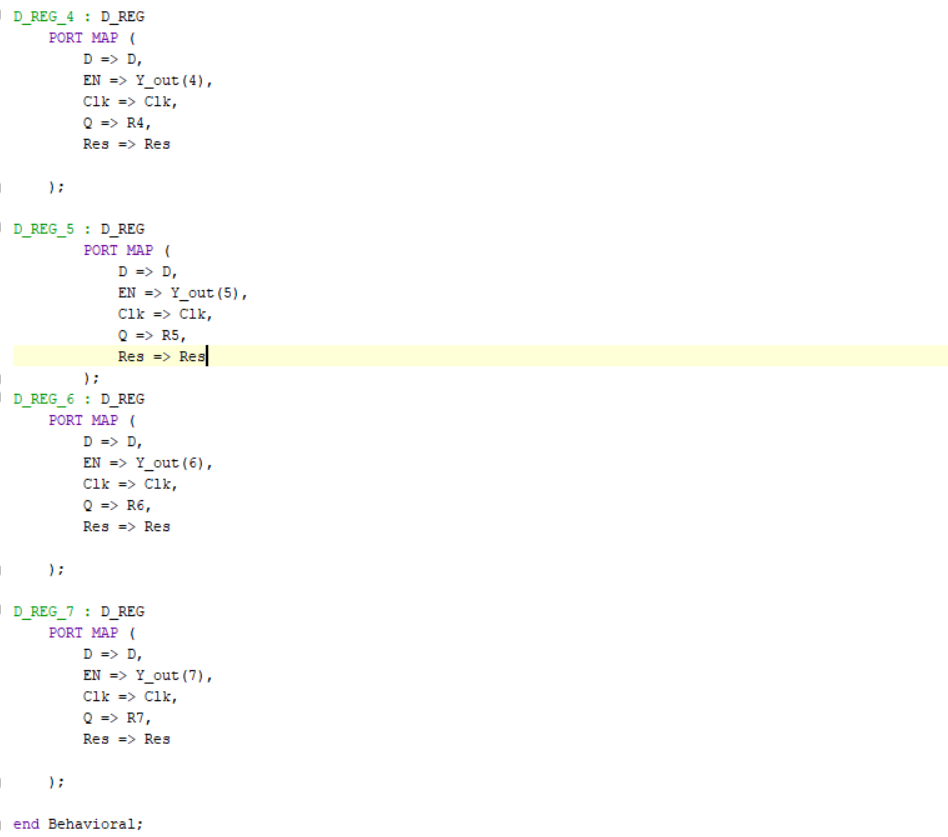


## VHDL File







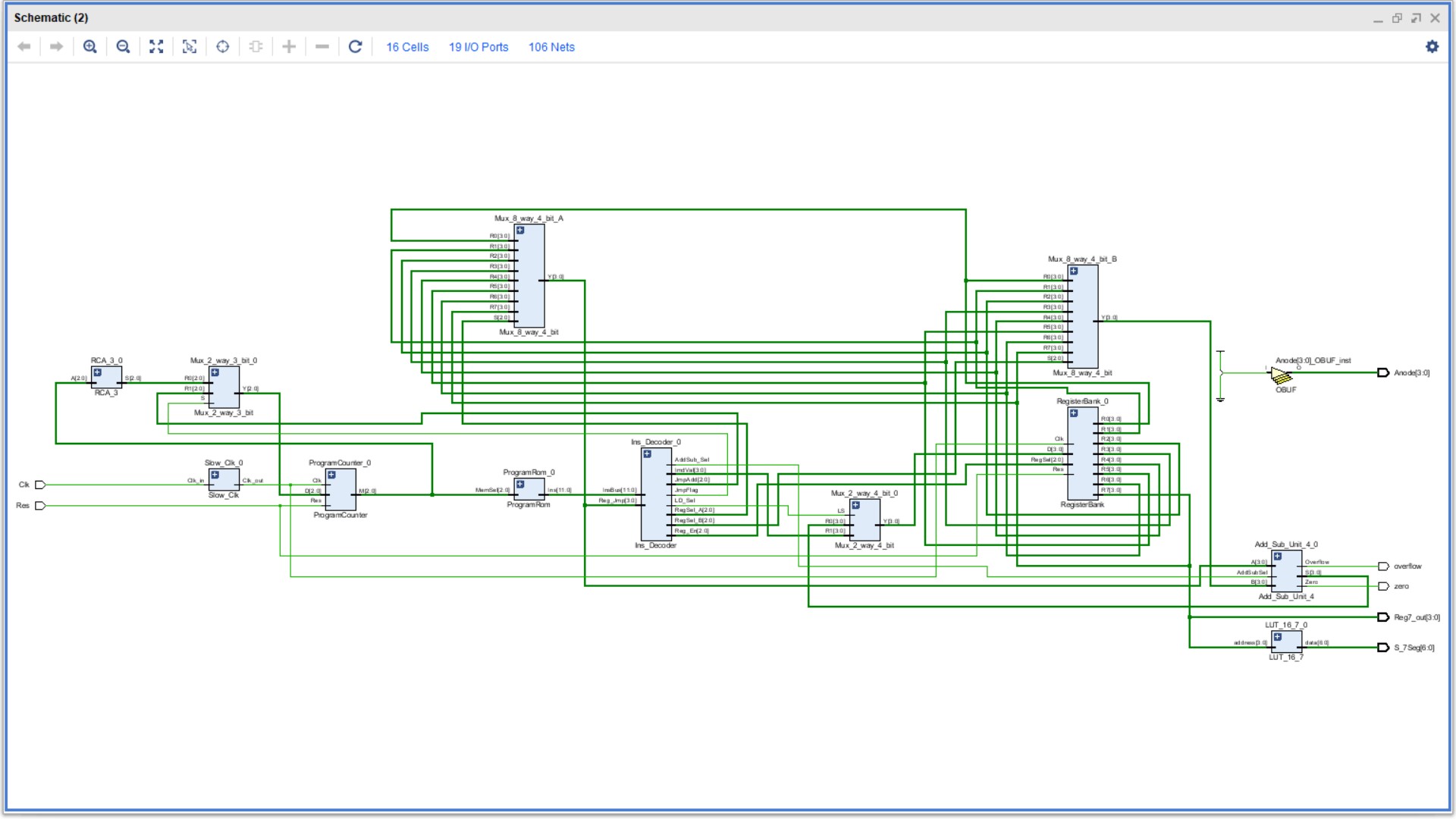


## Timing Diagram

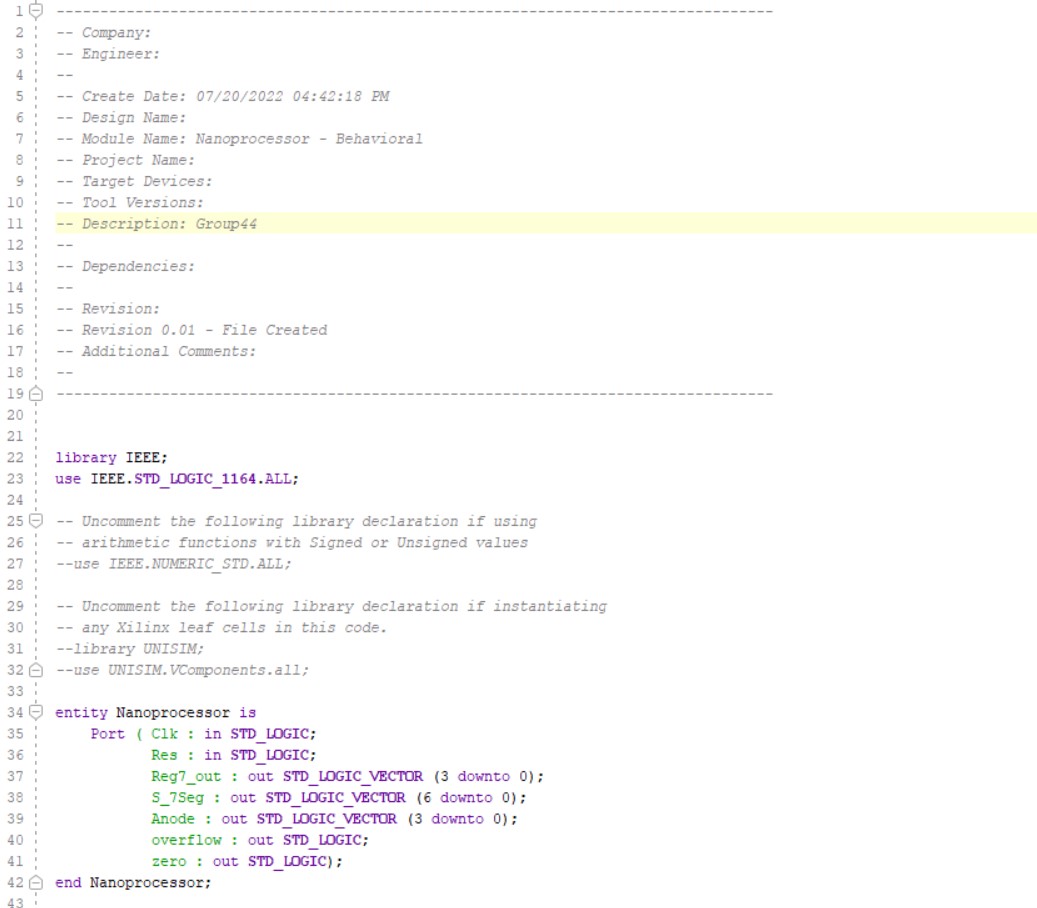
## 

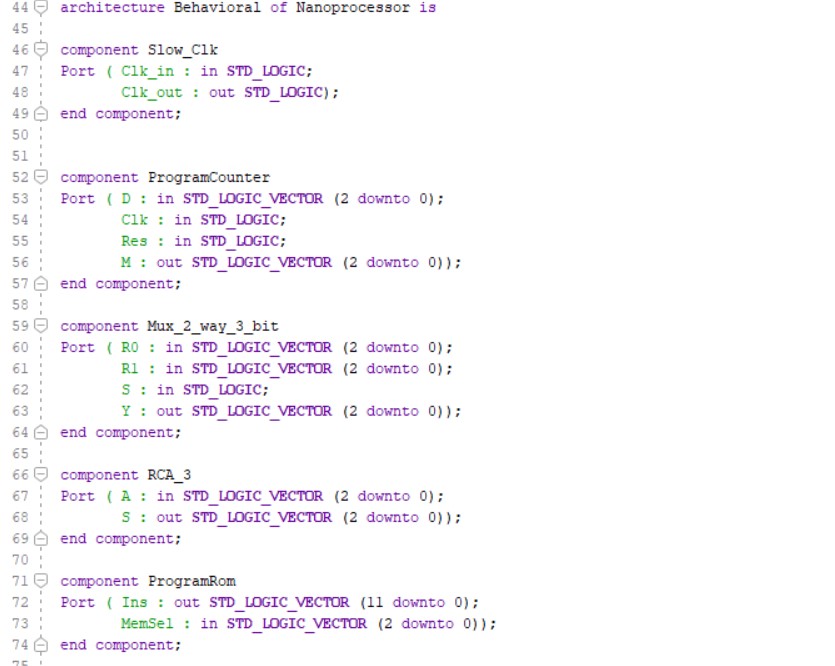
# Nanoprocessor

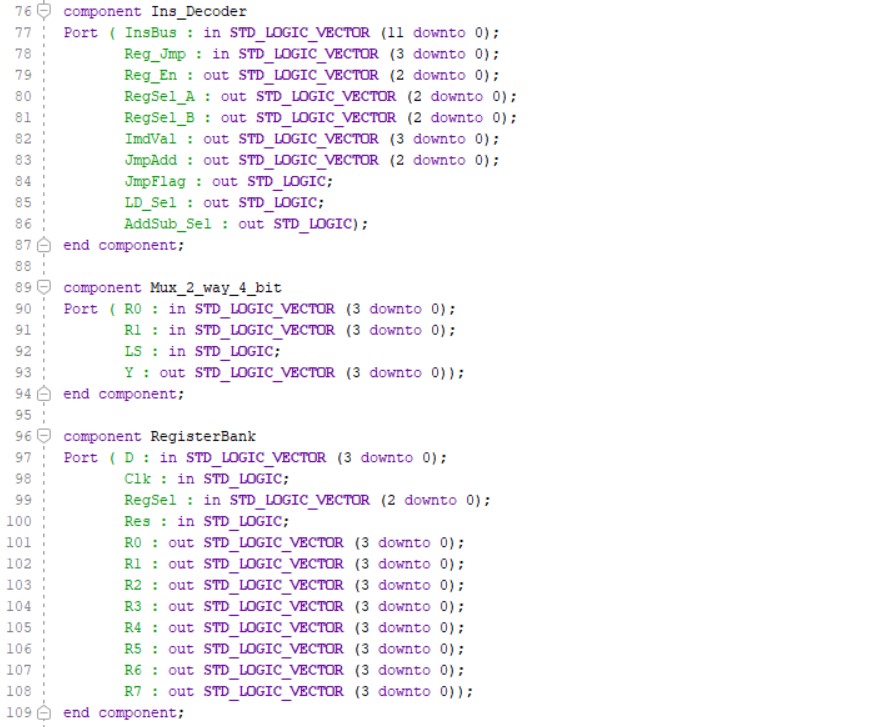
## Schematic

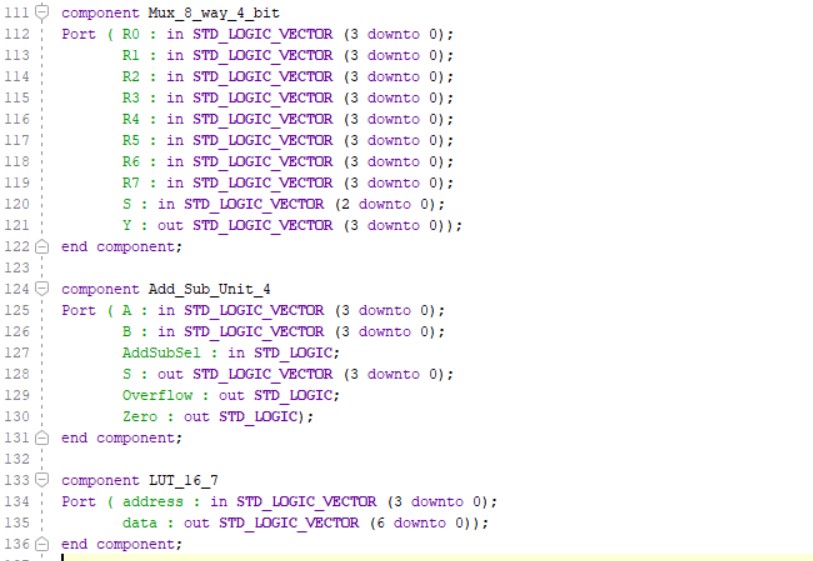


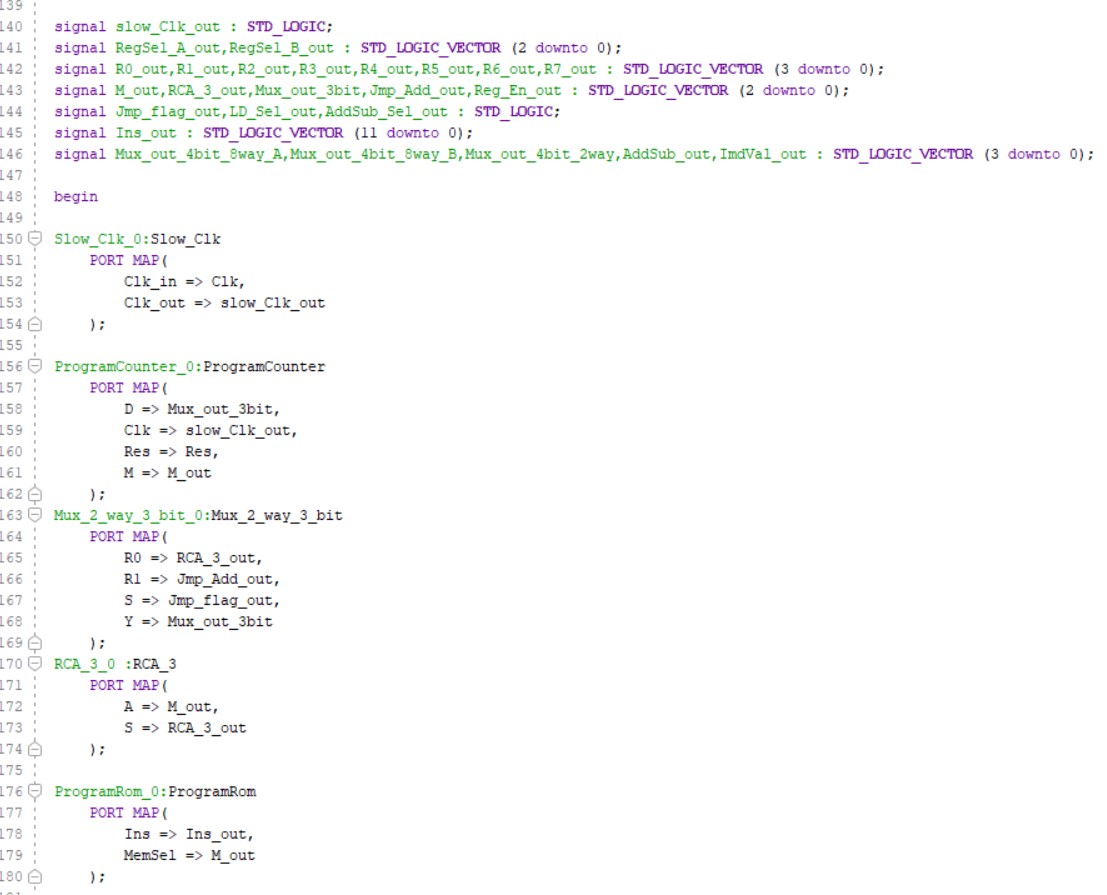
## VHDL File

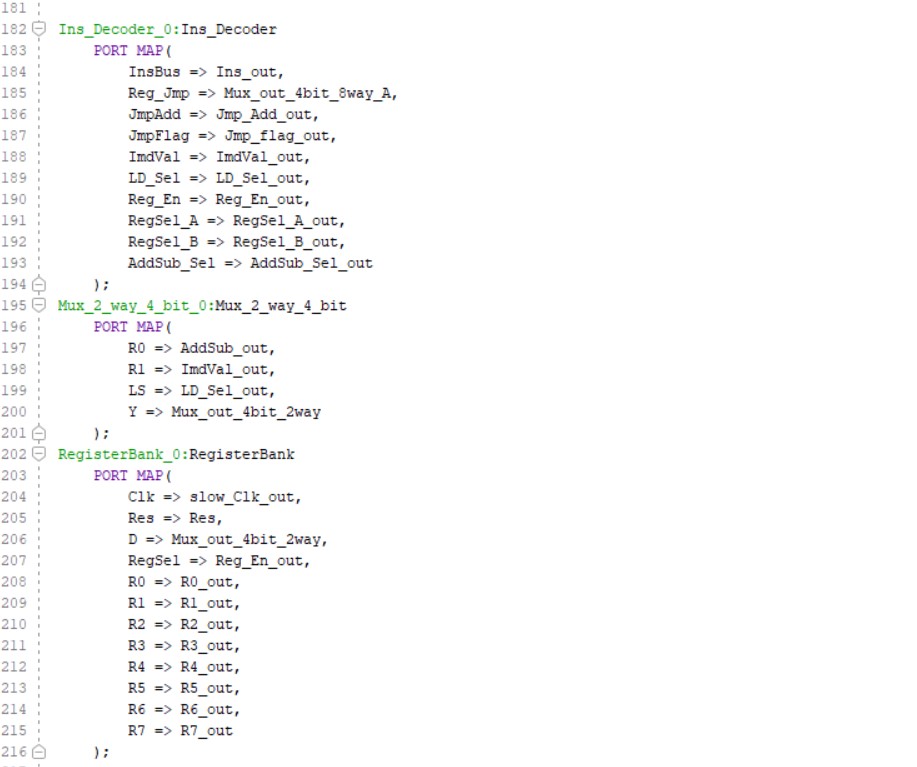


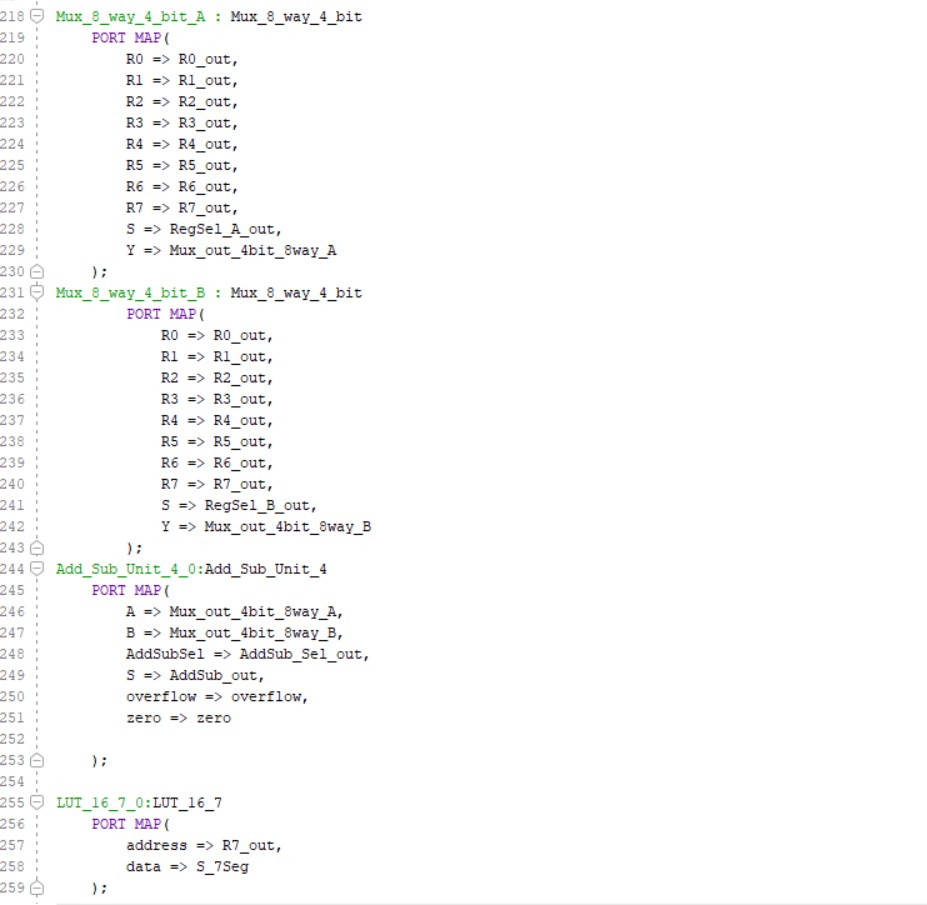






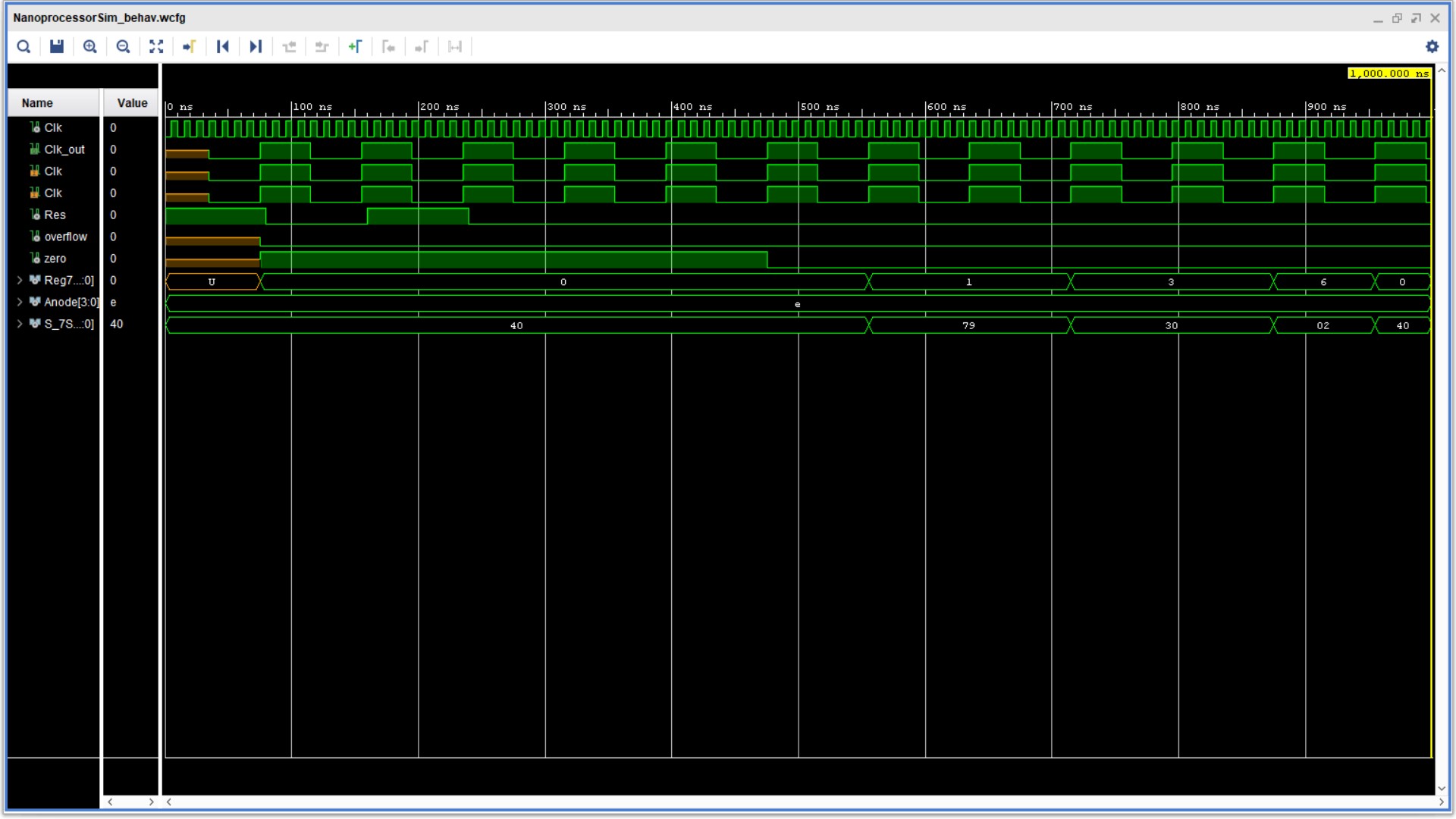




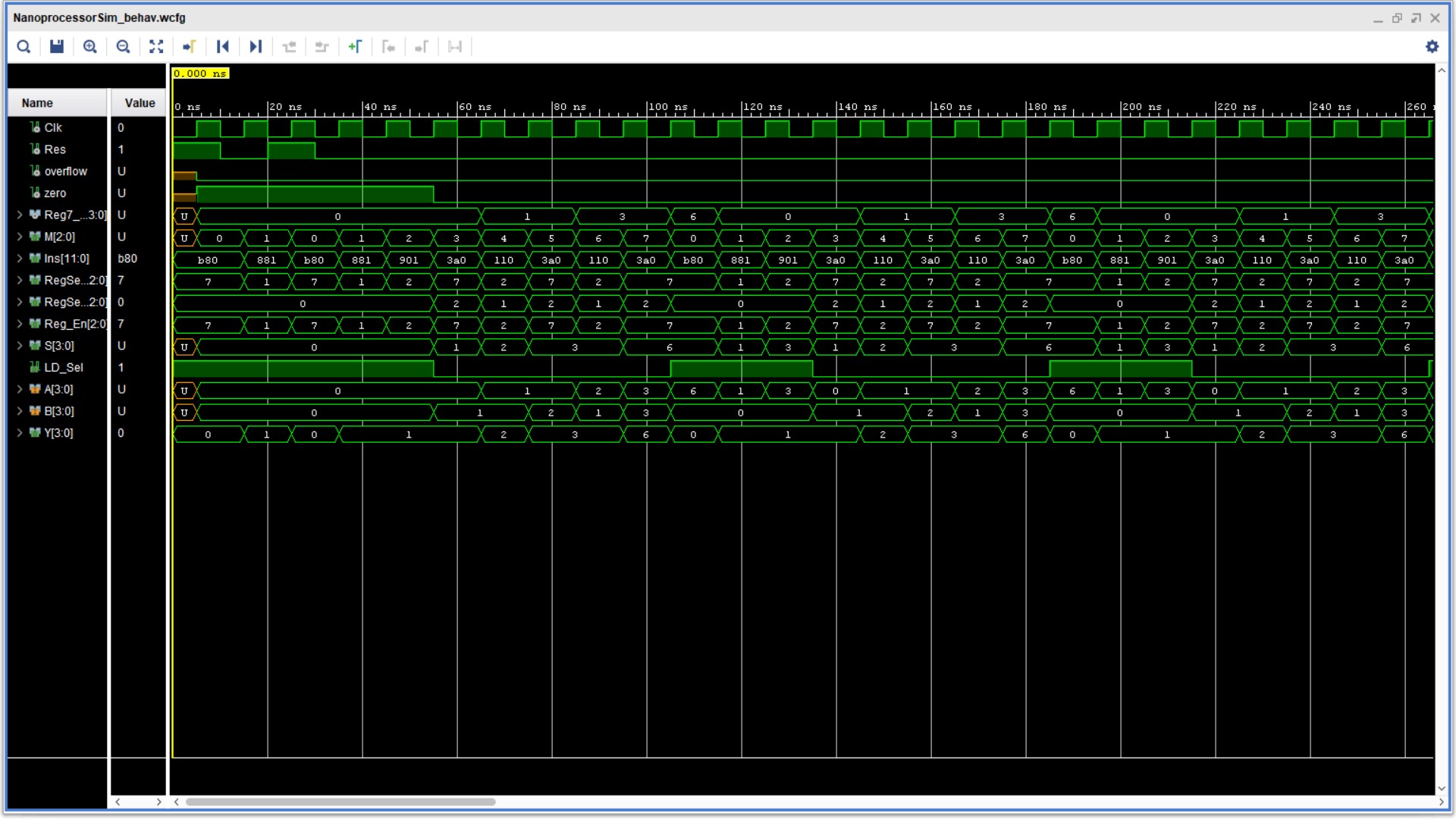




## Timing Diagram



## Timing Diagram with Internal Signals (Used For Debugging)



# Conclusion from the Lab

* More Instructions can be added by increasing the size of an instruction.
* Program Rom can be expanded to store larger programs with more instructions.
* “Jump if not zero” Instruction must be added in order to write assembly programs with conditional loops.
* This is a 4-bit architecture nanoprocessor; modern computers come with 64-bit architecture.
* We can write programs in assembly language directly to memory if we use an assembler to convert assembly language to relevant machine code of our processor architecture.
* Efficiency of the processor is determined by the maximum clock frequency the processor can support without any component of the machine being damaged.

1. Slice Logic

--------------

+-------------------------+------+-------+-----------+-------+

| Site Type | Used | Fixed | Available | Util% |

+-------------------------+------+-------+-----------+-------+

| Slice LUTs\* | 36 | 0 | 20800 | 0.17 |

| LUT as Logic | 36 | 0 | 20800 | 0.17 |

| LUT as Memory | 0 | 0 | 9600 | 0.00 |

| Slice Registers | 49 | 0 | 41600 | 0.12 |

| Register as Flip Flop | 49 | 0 | 41600 | 0.12 |

| Register as Latch | 0 | 0 | 41600 | 0.00 |

| F7 Muxes | 0 | 0 | 16300 | 0.00 |

| F8 Muxes | 0 | 0 | 8150 | 0.00 |

+-------------------------+------+-------+-----------+-------+

\* 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.

7. Primitives

-------------

+----------+------+---------------------+

| Ref Name | Used | Functional Category |

+----------+------+---------------------+

| FDRE | 49 | Flop & Latch |

| OBUF | 17 | IO |

| LUT4 | 17 | LUT |

| LUT5 | 12 | LUT |

| LUT6 | 9 | LUT |

| CARRY4 | 8 | CarryLogic |

| LUT3 | 4 | LUT |

| IBUF | 2 | IO |

| LUT1 | 1 | LUT |

| BUFG | 1 | Clock |

+----------+------+---------------------+

# Contribution of members to the project

## A.S. JAYATHUNGA - 200265T

* Design and simulation of Instruction Decoder.

## M.R.A.A.K. Gunasinghe - 200196G

* Design and Simulation of 3-bit Ripple Carry Adder.

## K.A.Anshan Lahiru Kavinda - 200300A

* Design and Simulation of Register Bank.

## K.K.A.J.S. Kumarasinghe - 200323V

* Design and Simulation of 8 way 4 bit Multiplexer.

## W.M.T.B.Weerasekara - 200698X

* Design and Simulation of Add/Sub unit.

The rest of the components' designing, integration of components, testing, and debugging were done collaboratively with the participation of all the members of the group.