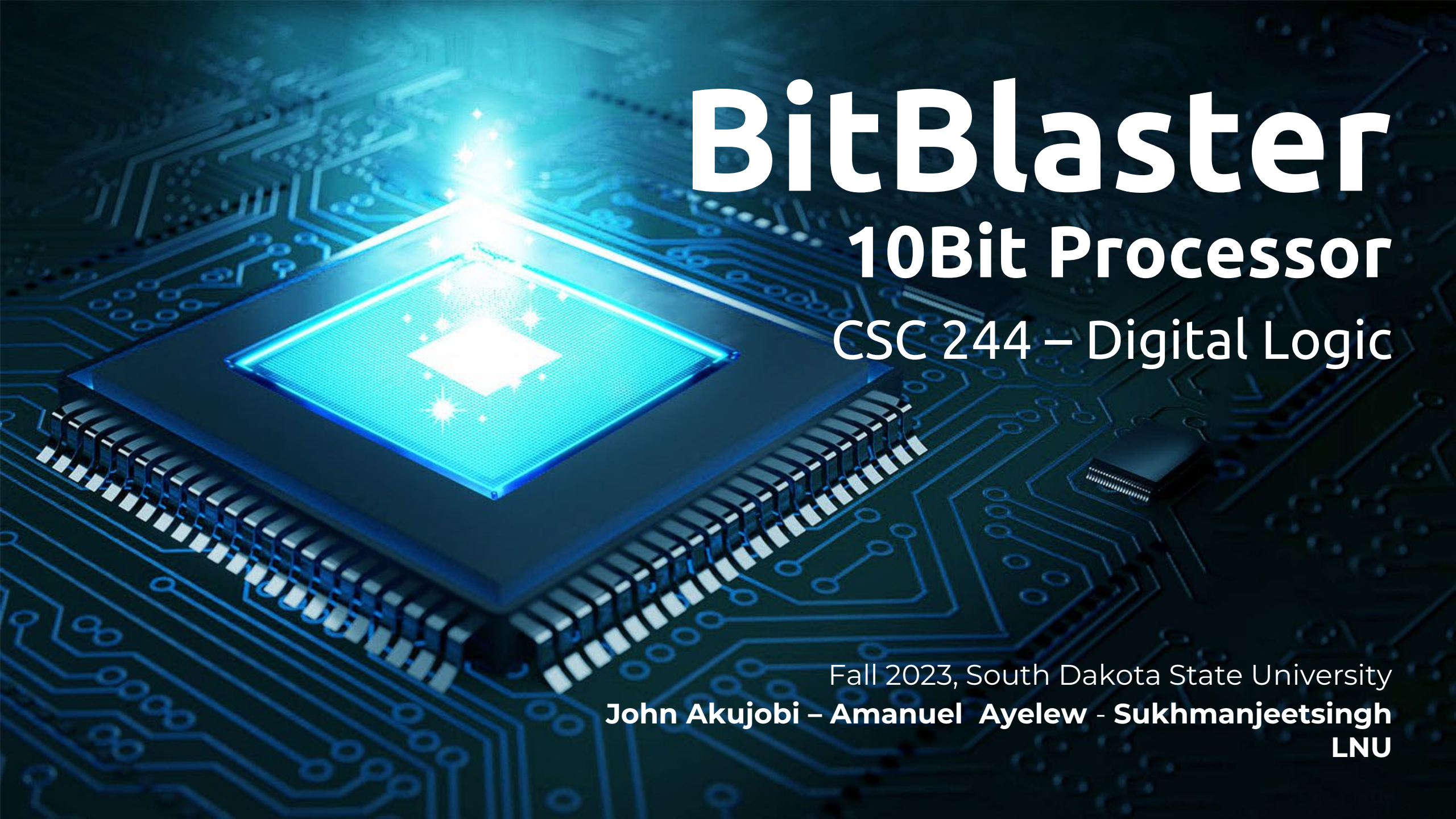


BitBlaster

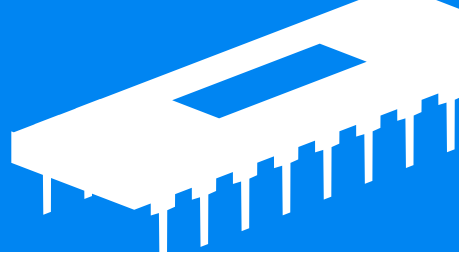
10Bit Processor

CSC 244 – Digital Logic

Fall 2023, South Dakota State University

John Akujobi – Amanuel Ayelew - Sukhmanjeetsingh LNU

System Architecture



Controller

Controls the processor

Counter

Sets up timesteps

Instruction Register

Saves the instruction

Input Logic

Gets & processes instructions from external input

SDRAM

Stores data

Top Level

Connects all modules

Register File

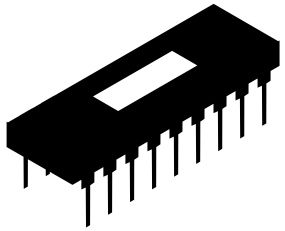
Saves data in 4 registers

Multistage ALU

Processes Arithmetic operations

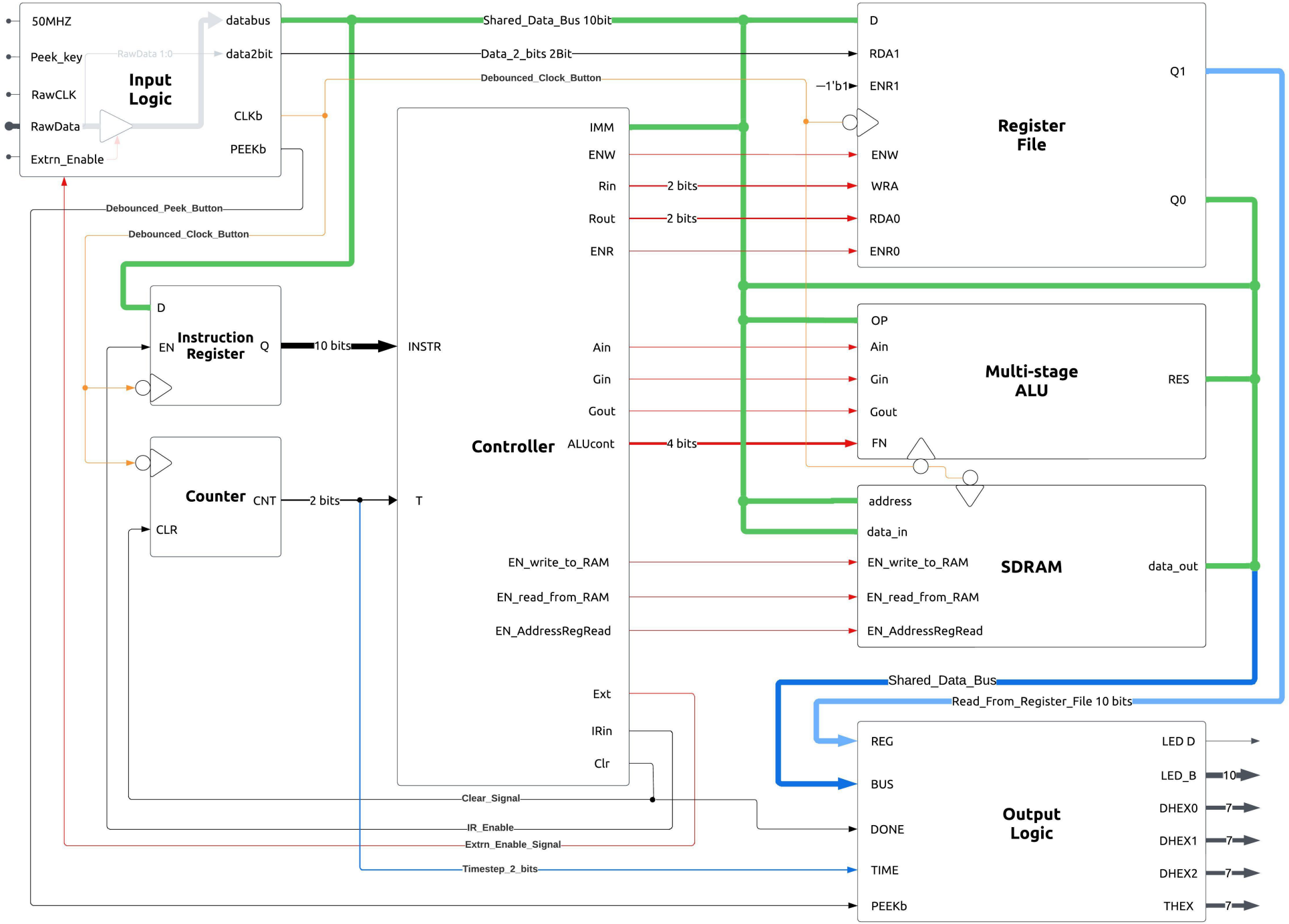
Output Logic

Displays the output



Shared BUS: Transfers data between components

Top Level View



Mnemonic Example

Add (00_XX_YY_0010)

- This instruction add Register Ry to Register Rx and **stores** the result in Register Rx by using the opcode provided, for example:
- Use Register 1 (binary 01) as Rx Register
- And Register 2 (binary 10) as Ry Register
- Encode instruction

Opcode

00	XX	YY	0010

Register Rx

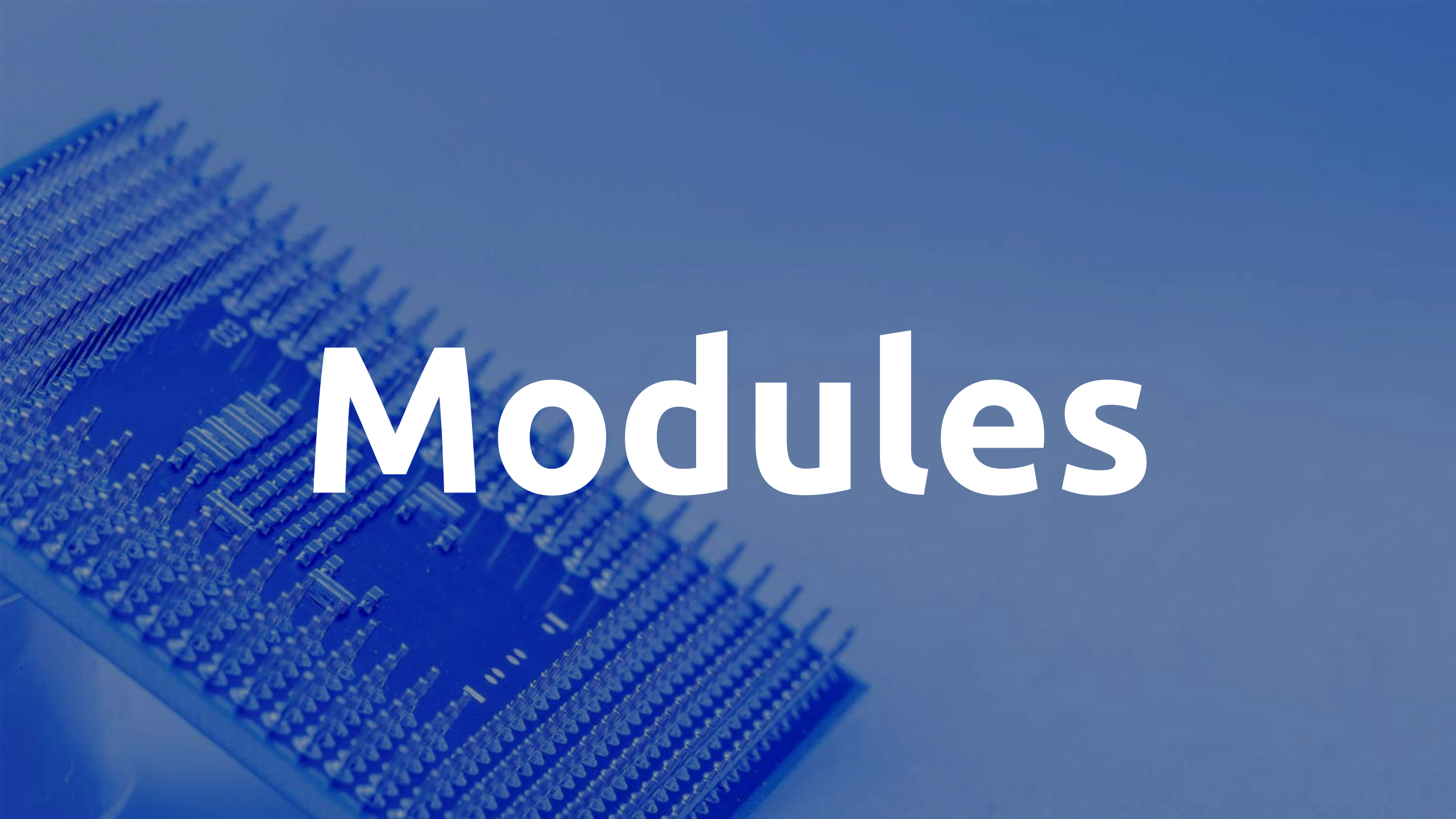
XX → 01

Register Ry

YY → 10

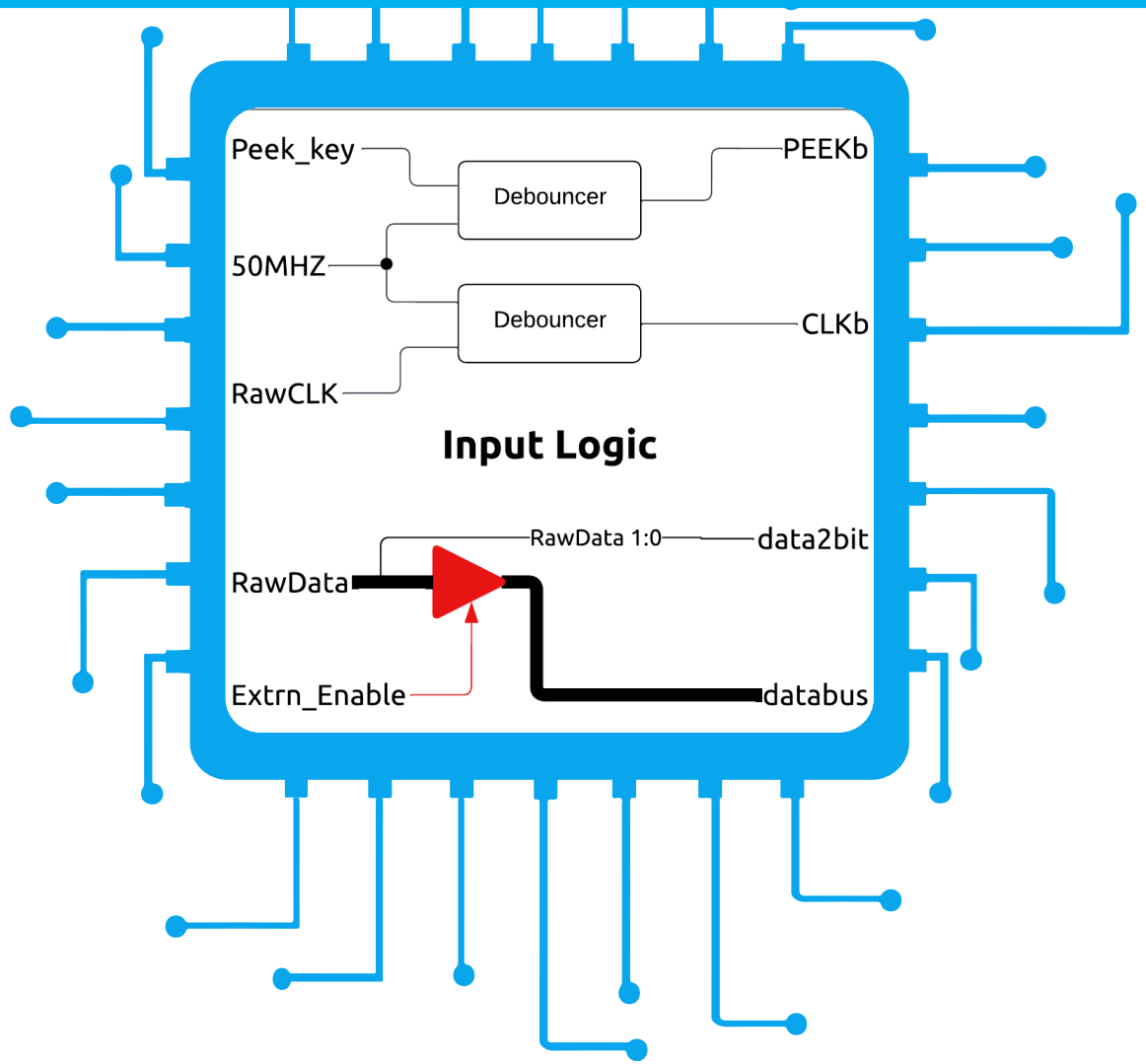
Encoded Instruction

00	01	10	0010


Modules

Input Logic

- Debounces the peek and clock signals
- Adds a tristate buffer to external input
- Receives input from external sources
- Splices data for the peek operation



INPUTS

- **Peek_key:** Lets user view the registers
- **50MHZ:** Internal clock of the DE10 Lite
- **RawCLK:** Clock button
- **RawData:** Data from the switches
- **Extrn_Enable:** Lets data write to bus

OUTPUTS

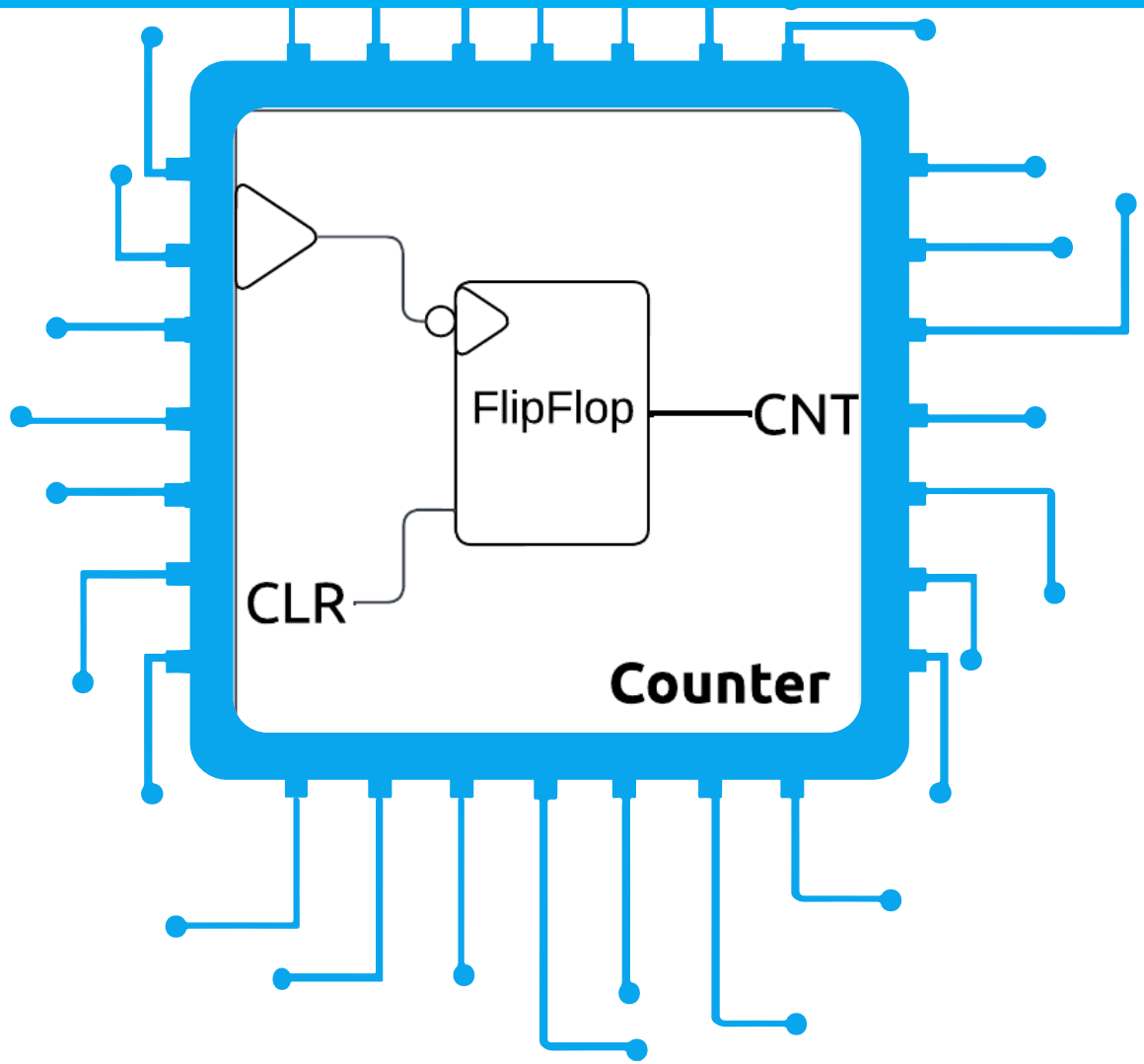
- **PEEKb:** Peek signal after debounced
- **CLKb:** Clean clock signal after debounced
- **data2bit:** To select register to peek
- **databus:** Data to be written onto the bus

COMPONENTS

- **Debouncer:** Cleans noisy signals

Counter

- **Provides Timesteps for Controller**
- **Synchronous clearing**
- **Neg-edge triggered counting**



INPUTS

- **CLR:** clear signal to reset counter
- **CLKb:** debounced clock signal

OUTPUTS

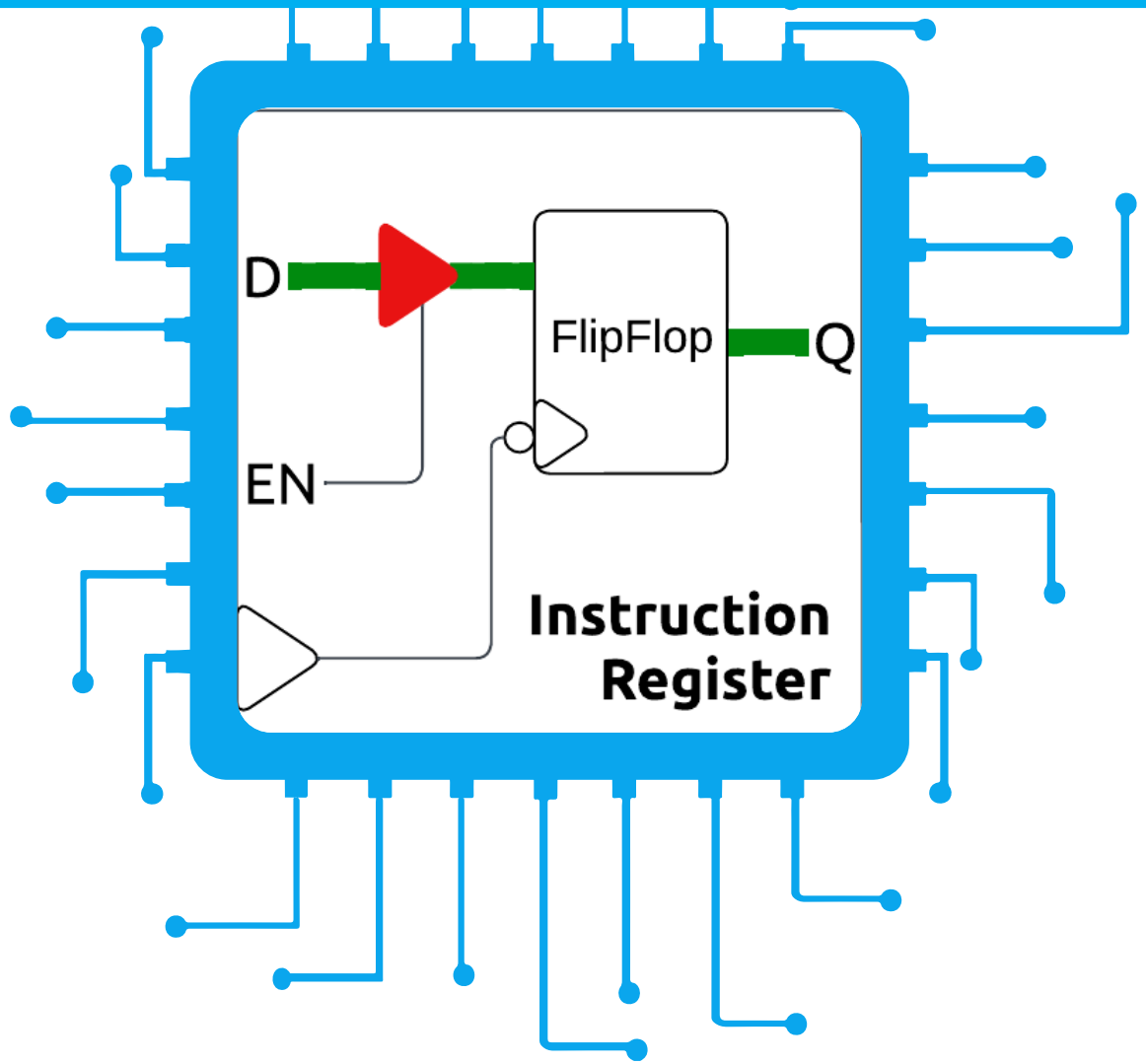
- **CNT:** 2bit counter value of current count
  This is fed into the controller to help it keep track of the timesteps

COMPONENTS

- **Address Register:** Saves memory address
- **1024x10**: 10 bit Memory array

Instruction Register :

Stores the Instruction during operation
Provides the instruction to the controller



INPUTS

- **D:** Data retrieved from the bus
- **CLKb:** Debounced clock (neg triggered)
- **EN:** Enable signal to allow the register to save inputs

OUTPUTS

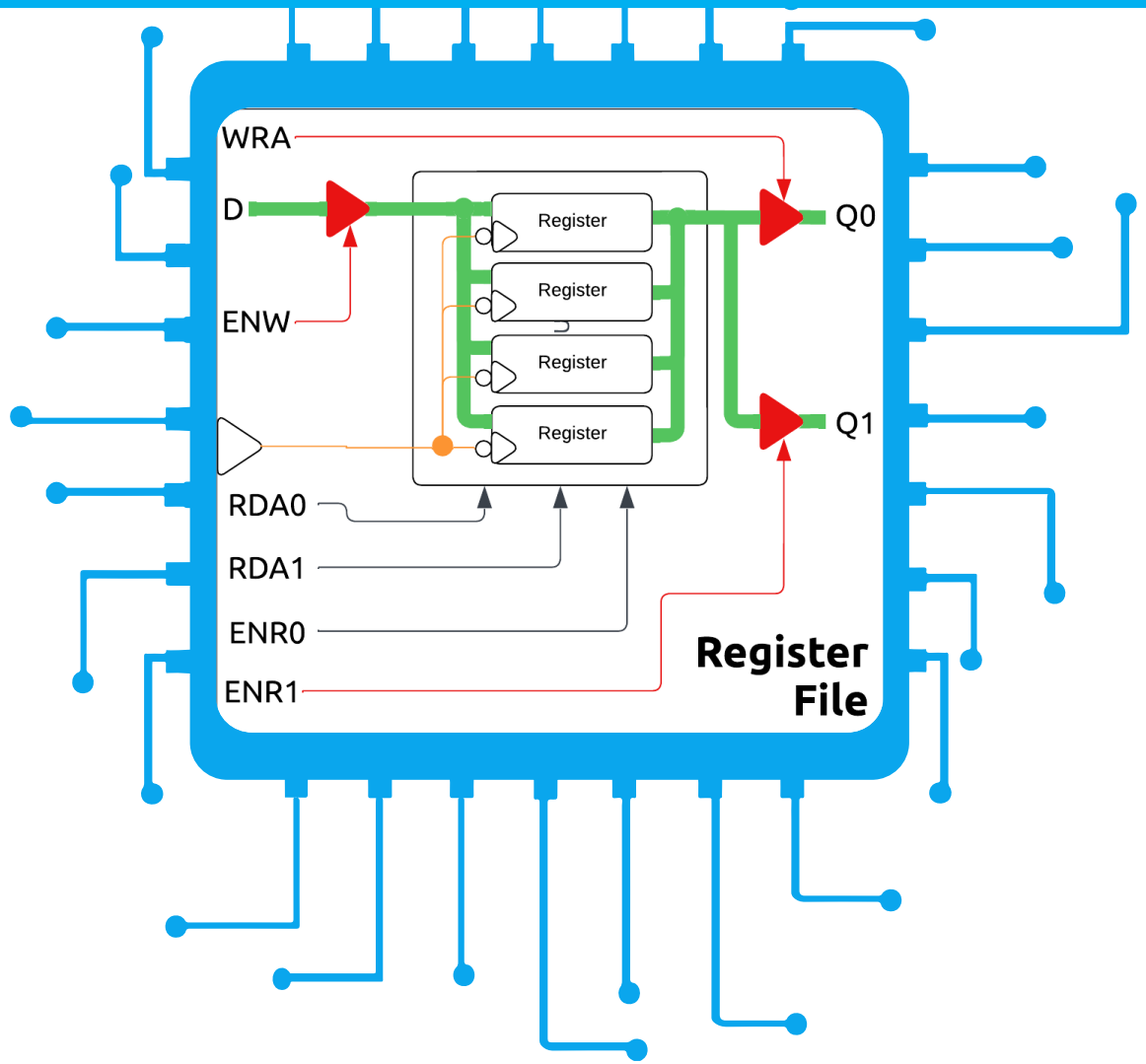
- **Q:** Saved instruction to be sent to the controller

COMPONENTS

- **Register:** Saves the actual data

Register File

- **Data Storage**
- **Reading & Writing**
- **Intermediate Storage**



INPUTS

- **D**: Common 10-bit input data
- **ENW:** Enable writing to registers
- **ENR0:** Enables Q0 output write to bus
- **ENR1:** Enables Q1 output
- **CLKb:** Neg triggered clock for registers
- **WRA:** Selects register to write to
- **RDA0:** Selects register to read from
- **RDA1:** Selects register to peek

OUTPUTS

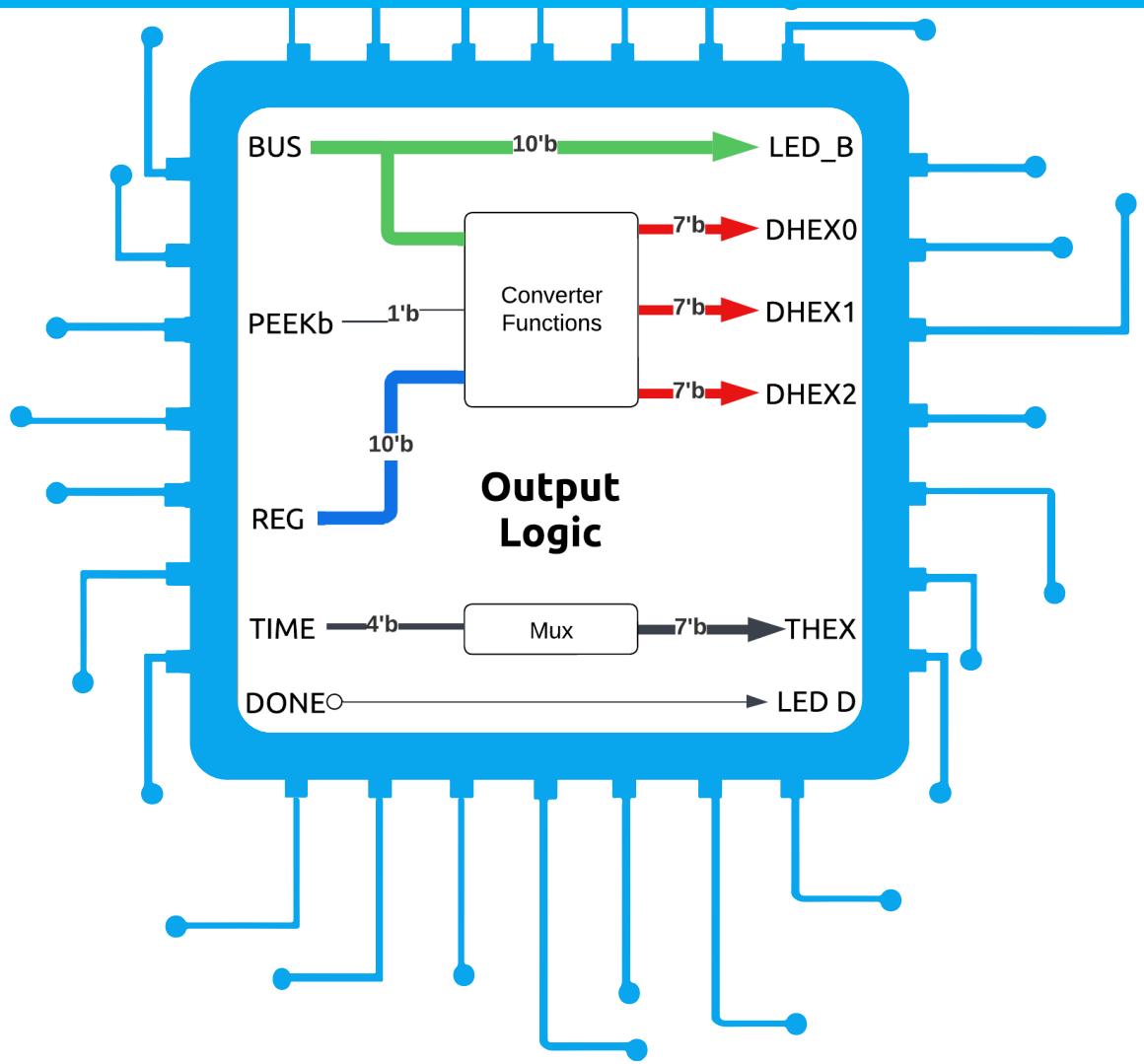
- **Q0:** Retrieved data from the register
- **Q1:** Data from the peeked register

COMPONENTS

- **4 Registers:** Saves 10-bit data

Output Logic

- **Provides a human Interface**
- **Displays current data on the Bus**
- **Lets users peek into register**



INPUTS

- **BUS:** Data from the shared bus
- **PEEKb:** Input from user to peek
- **REG:** Data peeked from register
- **TIME:** Current timestep from counter
- **DONE:** Clear signal from controller

INPUTS

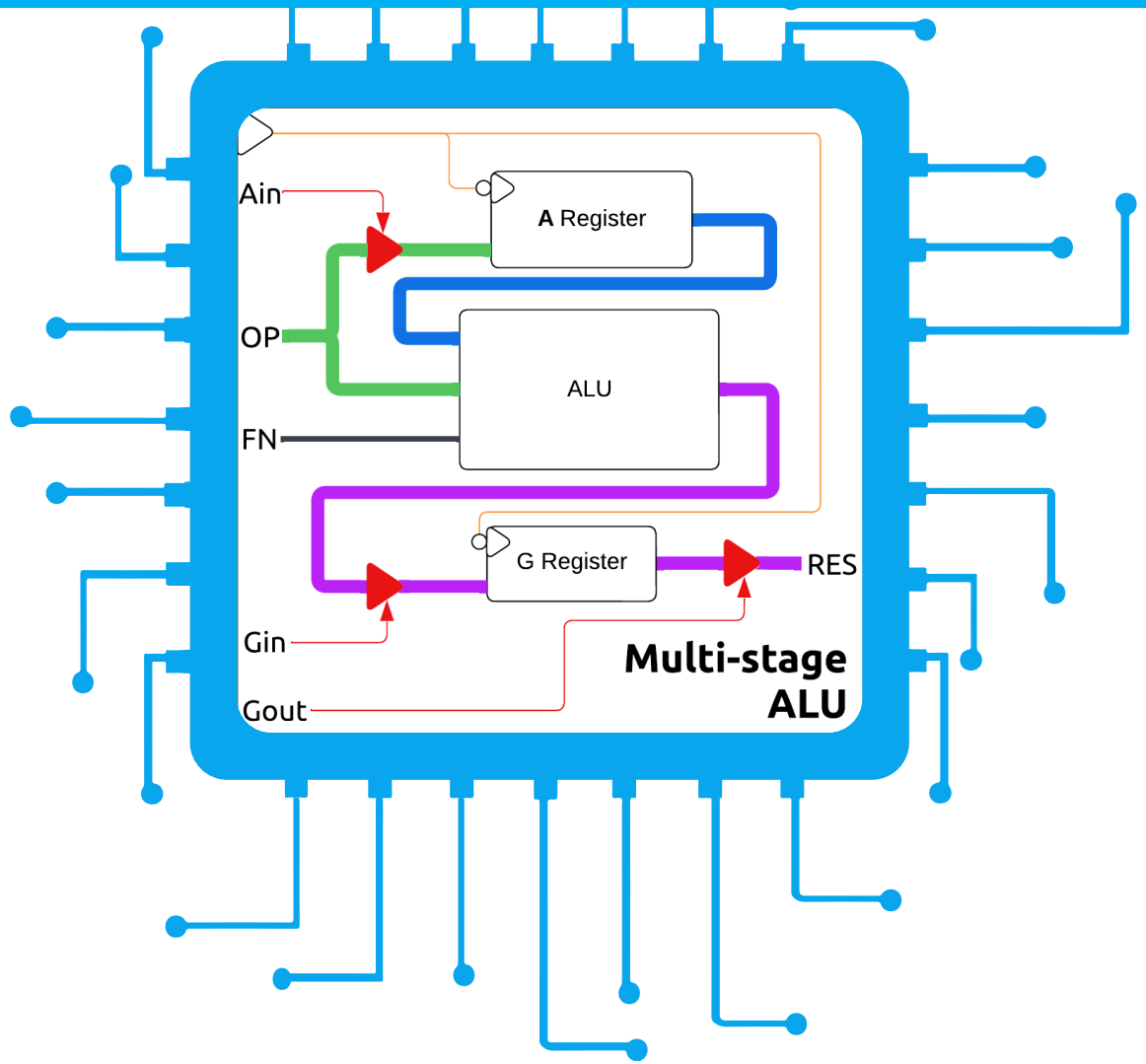
- **LED_B:** Array of LEDs showing data on bus
- **DHEX#:** 7 segment hex displays * 3
  - Shows data on bus or register in peek
- **THEX:** 7 seg display showing current timestep
- **LED D:** shows that the operation is done

COMPONENTS

- **Converter Function:** converts data to hex
- **Mux function**: Converts 4'b time to 7'b display

Multi-stage ALU

- **Executes arithmetic operations**
- **Stores data temporary in G register**



INPUTS

- **CLKb:** Clock for neg triggered registers
- **Ain:** Enable for the A register
- **OP:** Data from the shared bus
- **FN:** Arithmetic instruction from controller
- **Gin:** Enable write to G register
- **Gout:** Enable read from G register

OUTPUTS

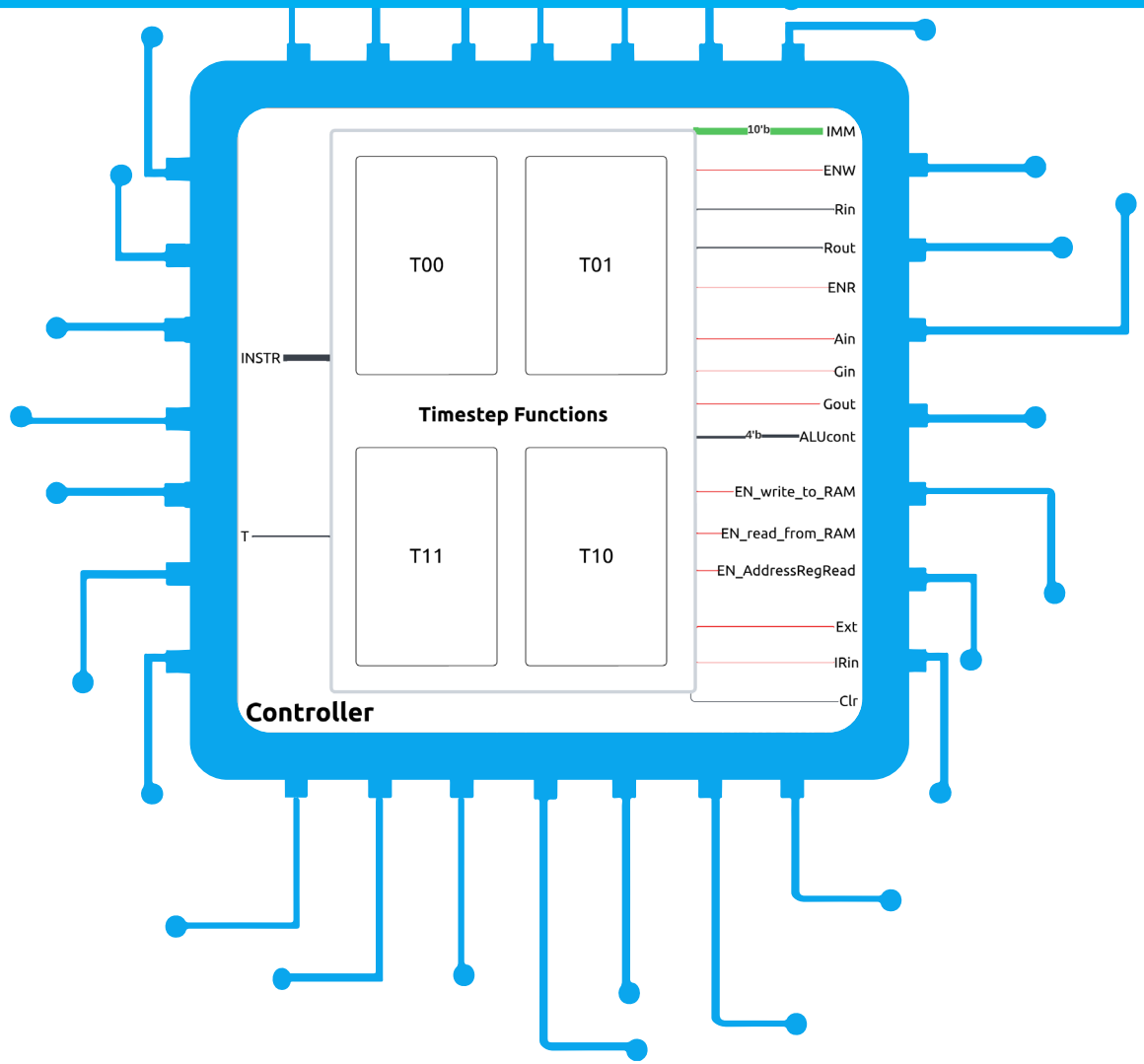
- **RES:** Calculated result

COMPONENTS

- **A Register:** Saves memory address
- **ALU:** Performs calculations
- **G Register:** Saves result from ALU

Controller

- **Controls the other modules in the processor.**



INPUTS

- **INSTR:** Instruction from instruction register
- **T:** Timestep count from counter

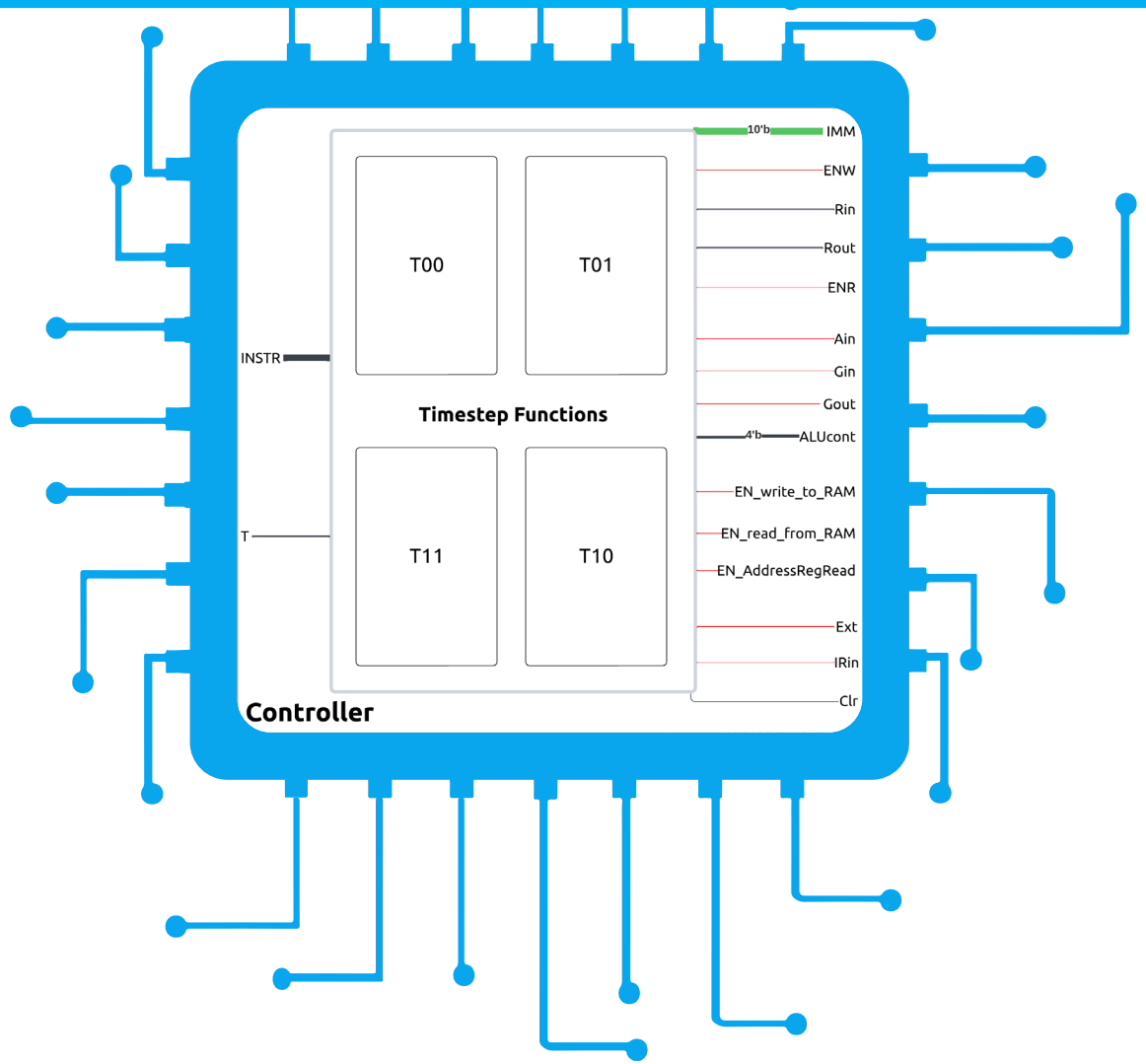
COMPONENTS

- **Timestep Functions:** Enables the output of the controller depending on timestep

OUTPUTS

- **Ext:** Enables the external data input in the input logic
- **IRin:** Enables the instruction register
- **Clr:** Sends a reset signal to the counter, and a done signal to the output logic

Controller OUTPUTS



REGISTER FILE

- **IMM:** Immediate value to put into the bus
- **ENW:** To write data to the register file
- **Rin:** Address for register to be written to
- **Rout:** Address for register to be read from
- **ENR:** To read data from the register file

MULTI-STAGE ALU

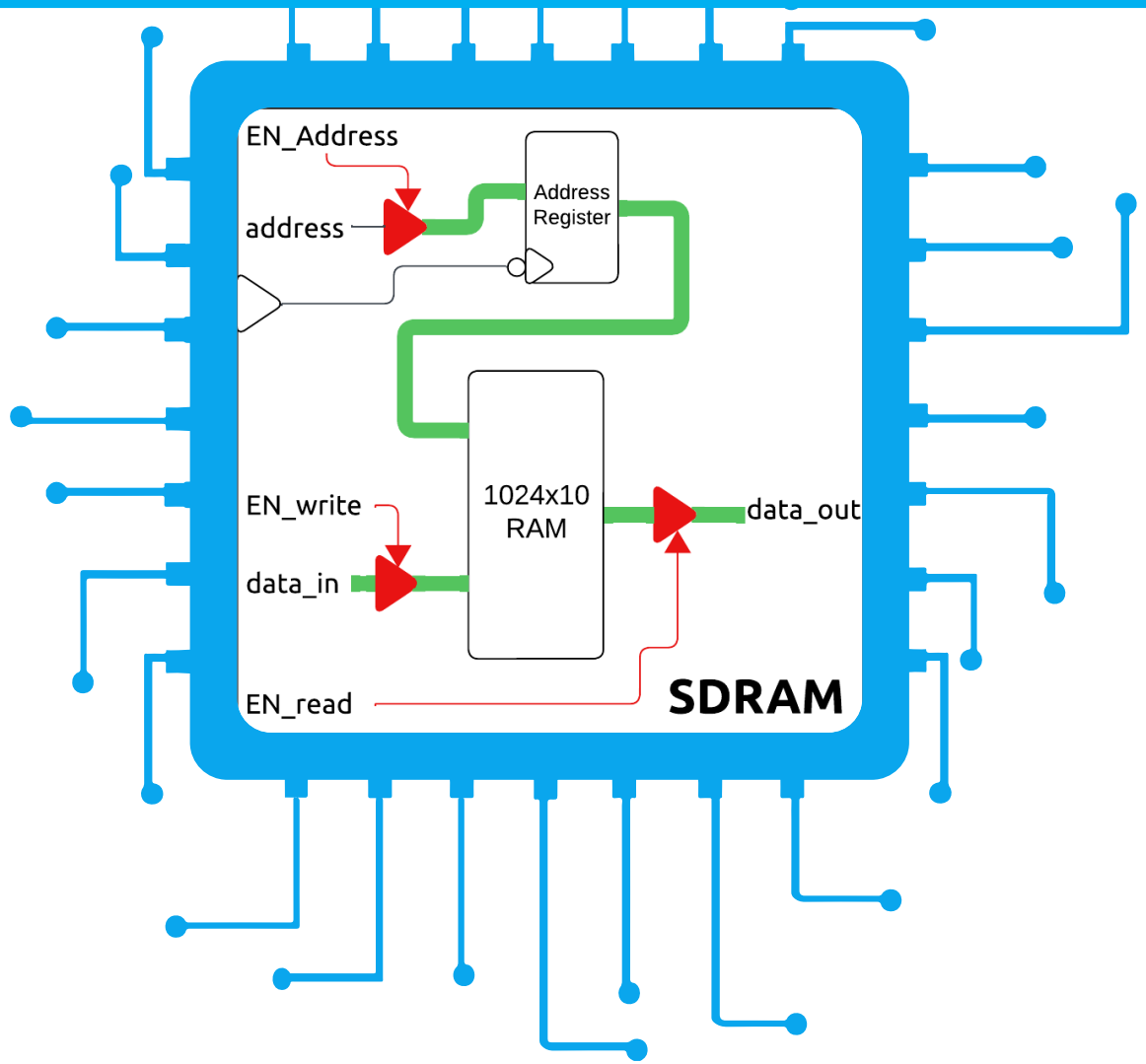
- **Ain:** save data to the “A” register
- **Gin:** allow data write to register “G”
- **Gout:** let “G” register write to the bus
- **ALUcont:** selects ALU operation to perform

SDRAM

- **EN_write_to_RAM:** enable write to ram
- **EN_read_from_RAM:** let ram write to bus
- **EN_AddressRegRead:** Let address register read

SDRAM

- **Stores data from the registers**



INPUTS

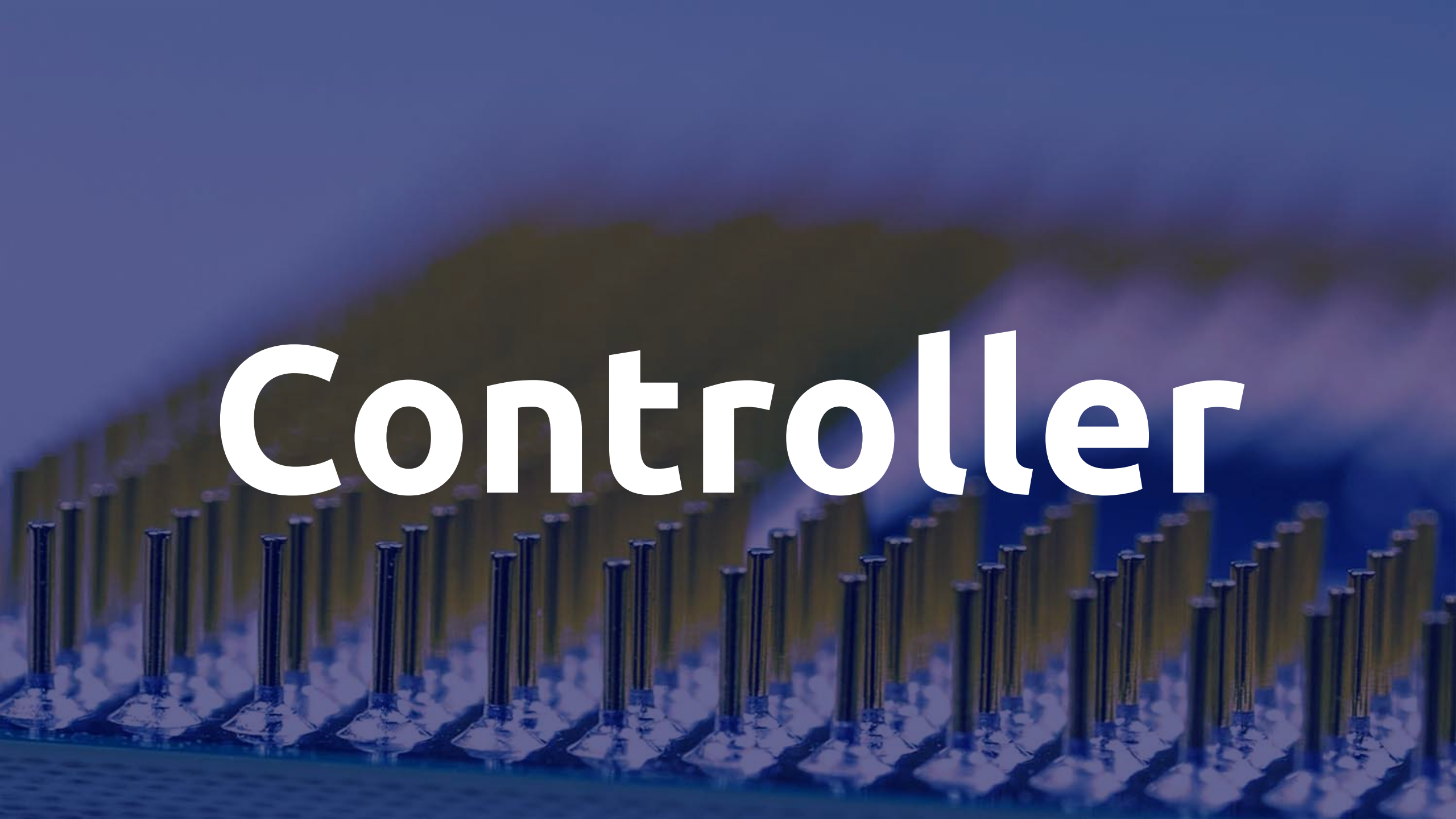
- **address:** Memory address to be used
- **EN_Address:** Lets Address Register read
- **CLKb:** Negative edge triggered
- **EN_write:** Allows write to ram
- **data_in:** Data to be saved into RAM
- **EN_read:** Allows bus to read from RAM

OUTPUTS

- **data_out:** Data from ram to be sent back into the bus

COMPONENTS

- **Address Register:** Saves memory address
- **1024x10**: 10 bit Memory array


Controller

Managed Components

Register File

- Allows data to be written to or read from the register file
- Picks which register to operate

Counter

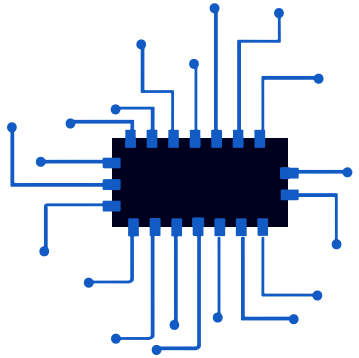
- Resets the counter using the Clr signal

Output Logic

-Provide the DONE input that shows the end of the operation

Multi-stage ALU

- Decides the operation to perform
- Data input into the A and G registers
- Result output from G register



Input Logic

- Controls the extern that allows external data to be written to the bus

SDRAM

- Input into the address register
- Input into the RAM
- Output from the RAM to the bus

Shared Bus

- Writes the immediate value using IMM into the bus.
This happens in the immediate operations

Instruction Register

- Tells it to save the instruction from the bus or not.
This happens in the T00 timestep.

Controller Implementation

Default output disable

All outputs are disabled by default at the start of each timestep

- Ensure only the things we want enabled are doing so.

```
// Initialize all outputs to default values
IMM = 10'bzzzzzzzzzz;   // Default value for IMM
Rin = 2'b0;             // Default value for Rin
Rout = 2'b0;            // Default value for Rout
ENW = 1'b0;             // Default value for ENW
ENR = 1'b0;             // Default value for ENR
Ain = 1'b0;             // Default value for Ain
Gin = 1'b0;             // Default value for Gin
Gout = 1'b0;            // Default value for Gout
ALUcont = 4'bzzzz;      // Default value for ALUcont
Ext = 1'b0;             // Default value for Ext
IRin = 1'b0;            // Default value for IRin
Clr = 1'b0;             // Default value for Clr
```

Nested if statements

```
If (T == 01)
        if (INST == 0101)
                #####
        else if (INST == 0110)
                #####
else if (T == 10)
        if (INST == 0101)
                #####
        else if (INST == 0110)
                #####
else if (T == 10)
        if (INST == 0101)
                #####
        else if (INST == 0110)
                #####
```

How the Bus works

Shared bus

The Bus is a communication pathway that connects the components allowing them to transmit and receive data.

Reading: Any component can receive data from the bus at any time.

Writing: However, only one component is allowed to write at the same time.

If multiple components try to do so, we will get an error.

The controller works to make sure this doesn't happen using **Tri-state buffers**

TRI-STATE BUFFER

Exists in High **(1)**, Low **(0)**,

High impedance (**z**)

•**Role:** Allows multiple devices to connect to the bus without interfering with each other's signals.

•**Function:** When not in use, a device's bus connection is put into a High Impedance state, effectively disconnecting it from the bus.

Keys & Meanings

Key	Meaning and Details
Function	The name of the operation being carried out
enIR	Enable for the instruction register
en	Ry register Ry = INST [5:4]
ENR	Enables the bus to read from the register Enable for Q0
WNR	Enables the bus to write to the register
IIIIII	Values used in immediate operations
UU	Insignificant values, doesn't affect operation and can be anything
enRout	Enable signal for the register file to write to bus
enRin	Enable signal for register file to read from bus

Key	Meaning and Details
INST	10 bit instruction gotten from instruction register
XX	Rx register Rx = INST [7:6] Contains the result at the end of most of the operations
YY	Ry register Ry = INST [5:4]
ENR	Enables the bus to read from the register Enable for Q0
WNR	Enables the bus to write to the register

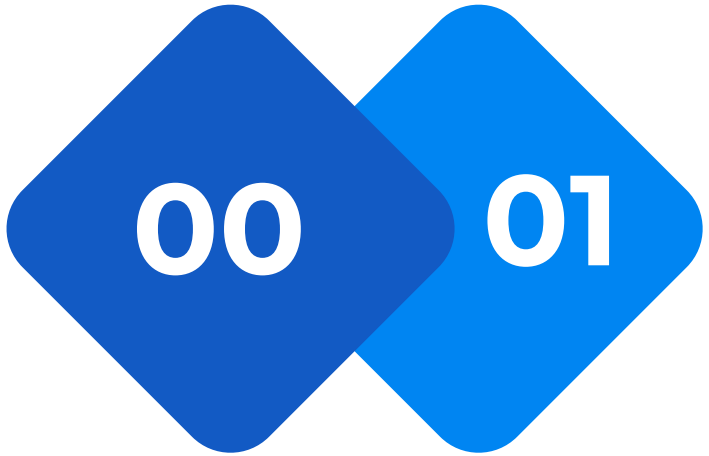
Memory Movement

LOAD, COPY

T00: 0th Timestep

- Enable Extern: get the instruction from the switches into the BUS

- Enable IR: store instruction in Instruction Register



T 01: 1st Timestep

- Enable Extern: writes the data value to the bus

- **Rin = Rx**: Prep the Rx register to read from the bus

- Enable Rin: Lets the register file read data from the bus

- **CLR = 1**: End the operation using the clear signal and reset the counter.

Load

00_XX_UU_0000

Memory Movement

Function	Mnemonic	Opcode	Details	Procedure	T0=00	T1=01
LOAD	ld Rx	00_XX_UU_0000	Load data into Rx from slide switches	Rx ← Data	**Extrn = 1** **enIR = 1**	enRin = 1 Rin = Rx
COPY	cp Rx, Ry	00_XX_YY_0001	Copy value from Ry to Rx	Rx ← [Ry]		Rout = Ry enRout = 1 Rin = Rx Clr = 1

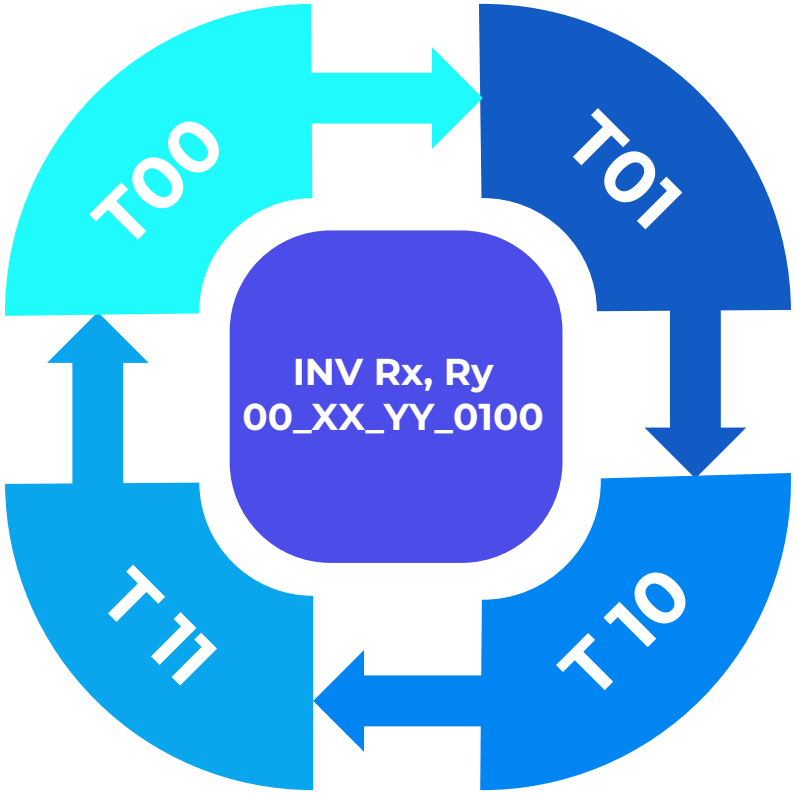
Arithmetic – 1 Operand

INVERT, FLIP

T00: 0th

- Enable enExtern: get the instruction from the switches into the BUS

- Enable IR: store instruction in Instruction Register



T01: 1st

- Rout = Rx: Select the Rx register to write to bus.

- Enable enRout: - Let register file write data from Rx to bus

- Enable Ain: Let the A register save the value from the bus

T 11: 3rd

- Enable Gout: Let the result in G register be written into the bus

- Rin = Rx: Select the Rx register to read from the bus

- Enable enRin: Let the register file read from the Bus into Rx

- Enable Clr: send the done signal.

T 10: 2nd

- **Rout = Ry:** Select Ry to write

- **Enable enRout:** Let the Register file to write from Ry to the Bus

- **ALUcont = INST[3:0]:** Sends the arithmetic operation code to ALU

- **Enable Gin:** Allows G register to store the result

Arithmetic – 1 Operands

Function	Opcode	Details	Procedure	T0=00	T1=01	T2=10	T3=11
INVERT inv Rx, Ry	00_XX_YY_0100	Twos-complement of Ry to Rx	Rx ← −[Ry]	Extrn = 1 enIR = 1	Rout = Rx ENR = 1 Ain = 1	Rout = Ry ENR = 1 ALUcont = INST[3:0] Gin = 1	Gout = 1 Rin = Rx ENW = 1 Clr = 1
FLIP flp Rx, Ry	00_XX_YY_0101	Flip bits of Ry and store in Rx	Rx ← ~[Ry]				

Arithmetic – 2 Operands

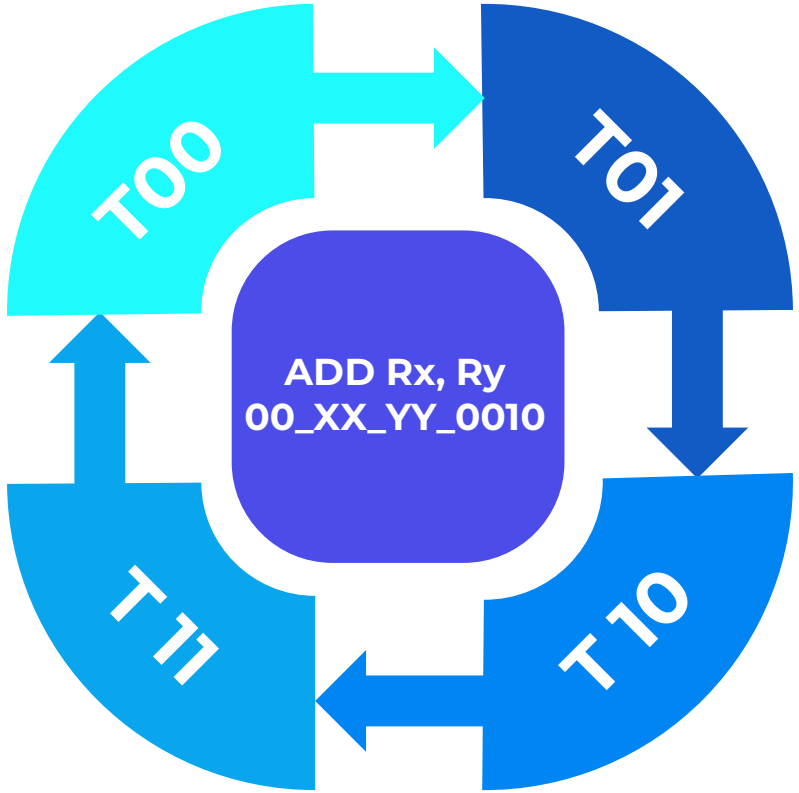
ADD, SUB, XOR, OR, AND, LSL, LSR, ASR

T00: 0th

- **Enable enExtern**: get the instruction from the switches into the BUS
- **Enable IR:** store instruction in Instruction Register

T01: 1st

- **Rout = Rx:** Select the Rx register to write to bus.
- **Enable enRout:** - Let register file write data from Rx to bus
- **Enable Ain:** Let the A register save the value from the bus



T 11: 3rd

- **Enable Gout:** Let the result in G register be written into the bus
- **Rin = Rx:** Select the Rx register to read from the bus
- **Enable enRin:** Let the register file read from the Bus into Rx
- **Enable Clr:** send the done signal.

T 10: 2nd

- **Rout = Ry:** Select Ry to write
- **Enable enRout:** Let the Register file to write from Ry to the Bus
- **ALUcont = INST[3:0]:** Sends the arithmetic operation code to ALU
- **Enable Gin:** Allows G register to store the result

Arithmetic – 2 Operands

Function	Opcode	Details	Procedure	T0=00	T1=01	T2=10	T3=11
ADD add Rx, Ry	00_XX_YY_0010	Add values in Rx and Ry	Rx←[Rx] + [Ry]	Extrn = 1 enIR = 1	Rout = Rx ENR = 1 Ain = 1	Rout = Ry ENR = 1 ALUcont = INST[3:0] Gin = 1	Gout = 1 Rin = Rx ENW = 1 Clr = 1
SUB sub Rx, Ry	00_XX_YY_0011	Subtract Ry from Rx	Rx←[Rx] - [Ry]				
AND and Rx, Ry	00_XX_YY_0110	Bit-wise AND Rx and Ry	Rx←[Rx] & [Ry]				
OR or Rx, Ry	00_XX_YY_0111	Bit-wise OR Rx and Ry	Rx←[Rx] | [Ry]				
XOR xor Rx, Ry	00_XX_YY_1000	Bit-wise XOR Rx and Ry	Rx ← [Rx] ∧ [Ry]				
LSL lsl Rx, Ry	00_XX_YY_1001	Logical shift left Rx by Ry	Rx←[Rx]<<[Ry]				
LSR lsr Rx, Ry	00_XX_YY_1010	Logical shift right Rx by Ry	Rx←[Rx]<<[Ry]				
ASR asr Rx, Ry	00_XX_YY_1011	Arithmetic shift right Rx by Ry	Rx←[Rx]>>>[Ry]				

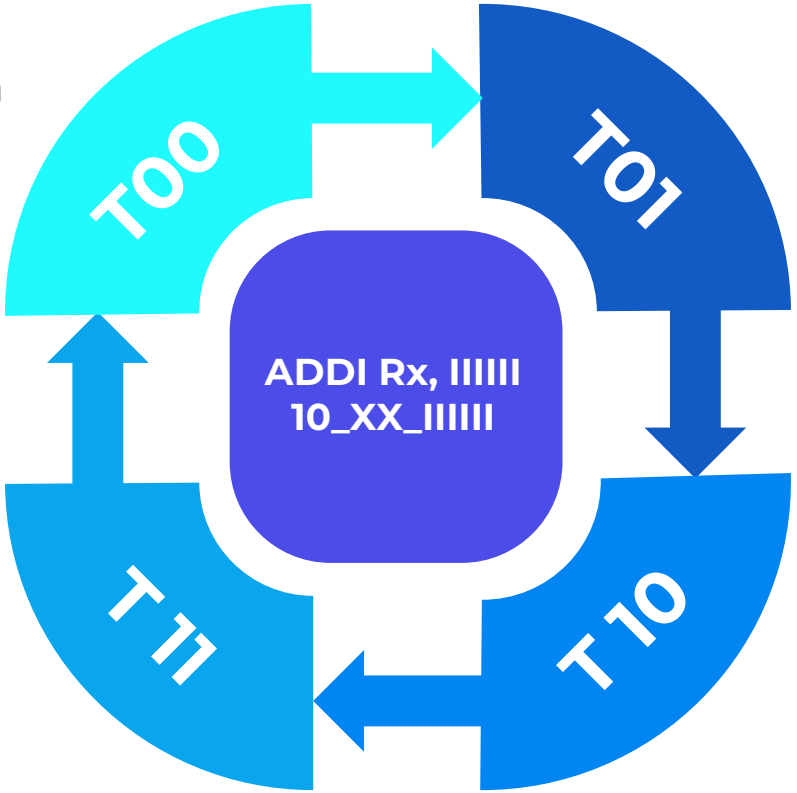
Immediate Instructions

ADDI, SUBI

T00: 0th

- Enable Extern: get the instruction from the switches into the BUS

- Enable IR: store the instruction in Instruction Register



T01: 1st

- Rout = Rx: Select the Rx register to write to bus.

- Enable enRout: - Let register file write data from Rx to bus

- Enable Ain: Let the A register save the value from the bus

T 11: 3rd

- Enable Gout: Let the result in G register be written into the bus

- Rin = Rx: Select the Rx register to read from the bus

- Enable enRin: Let the register file read from the Bus into Rx

- Enable Clr: send the done signal.

T 10: 2nd

- IMM[5:0] = INST[5:0]: write the immediate value into IMM which is connected to the bus

- IMM[9:6] = 4'b0000: Zero out the remaining values of IMM.

- **ALUcont = 4b'0010:** Instruct the ALU to perform addition operation

Immediate Instructions

Function & Mnemonic	Opcode	Details	Procedure	T0=00	T1=01	T2=10	T3=11
ADDI addi Rx, 6'bIIIIII	`10_XX_IIIIII`	Add immediate value (`IIIIII`) to RX	Rx ← [[Rx]+ 10'b0000IIIIII]	Extrn = 1 enIR = 1	Rout = Rx ENR = 1 Ain = 1	IMM[5:0] = INST[5:0] IMM[9:6] = 4'b0000 ALUcont = 4b'0010	Gout = 1 Rin = Rx enRin = 1 Clr = 1
SUBI subi Rx, 6'bIIIIII	`11_XX_IIIIII`	Subtracts immediate value (`IIIIII`) from RX	Rx ← [[Rx] - 10'b0000IIIIII]			IMM[5:0] = INST[5:0] IMM[9:6] = 4'b0000 ALUcont = 4b'0010	

Extra Credit

RAM Instructions

STR, LDR

T01: 1st

- **Rout = Rx:** Select the Rx register to write
- **Enable enRout:** allows register file to write on bus.
- **EN_Addreg:** Let the address register store this data (address) from the bus

T 10: 2nd

- **Rout = Ry:** select Ry to write
- **Enable ENR0:** allows register file to write on bus
- **EN_WriteRAM:** allow data to be written into the RAM
- **Clr = 1:** End the operation by sending a clear signal

T00: 0th

- Enable Extern: get instruction from the switches into the BUS

- Enable IR: store instruction in Instruction Register

RAM Instructions

Function	Mnemonic	Opcode	Details	Procedure	T0=00	T1=01	T2=10
LDR	ldr Rx, *Ry	`00_XX_YY_1100`	Load data stored in RAM (at the 10-bit address stored in Ry) to register Rx	Ram{Rx} ← [Ry]	Extrn = 1 enIR = 1	Rout = Ry ENR = 1 EN_Addreg = 1	enRAMread = 1 Rin = Rx ENW = 1 Clr = 1
STR	Str *Rx, Ry	`00_XX_YY_1101`	Store the data from Ry into RAM at the 10-bit address stored in Rx	Rx ← Ram{Ry}		Rout = Rx ENR = 1 EN_Addreg = 1	enRAMwrite = 1 Rout = Ry ENR = 1 Clr = 1

Mapped Inputs and Outputs

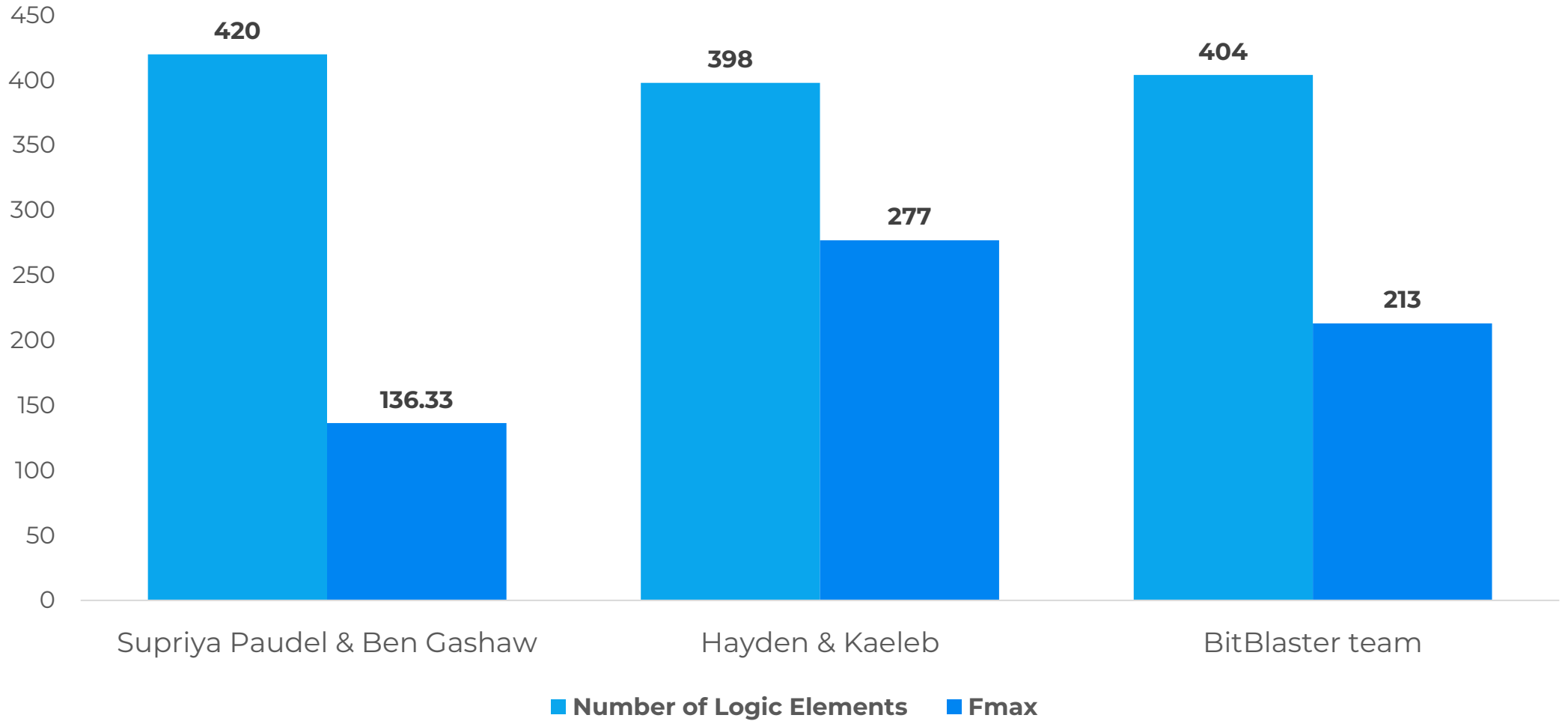
Inputs	Mapped Input
50 MHz	50 MHz clock input for ADC (Bank 3)
Peek_key	KEY 1
RawCLK	KEY0
RawData	SWITCH[9:0]
Extern Enable	

Outputs	Mapped Output
DHEX0	HEX0
DHEX1	HE X1
DHEX2	HEX2
THEX	HEX3
LED_D_DONE	HEX3 - Decimal Point

Instruction Program Table

Function	Opcode	Mnemonic	Instruction
Load	00_XX_UU_0000	ld Rx	Rx ← Data
Copy	00_XX_YY_0001	cp Rx, Ry	Rx ← [Ry]
Addition	00_XX_YY_0010	add Rx, Ry	Rx ← [Rx] + [Ry]
Logical Shift Right	00_XX_YY_1010	lsr Rx, Ry	Rx ← [Rx] >> [Ry]
Addition	10_XX_IIIIII	addi Rx, 6'bIIIIII	Rx ← [Rx] + 10'b0000IIIIII
Store to RAM	00_XX_YY_1101	Str *Rx, Ry	Rx ← Ram{Ry}

Comparison


450
400
350
300
250
200
150
100
50
0
420
136.33
398
277
404
213
Supriya Paudel & Ben Gashaw
Hayden & Kaeleb
BitBlaster team
Number of Logic Elements
Fmax

Tools used

Git GUI for Desktop

Visual Studio Code
IDE

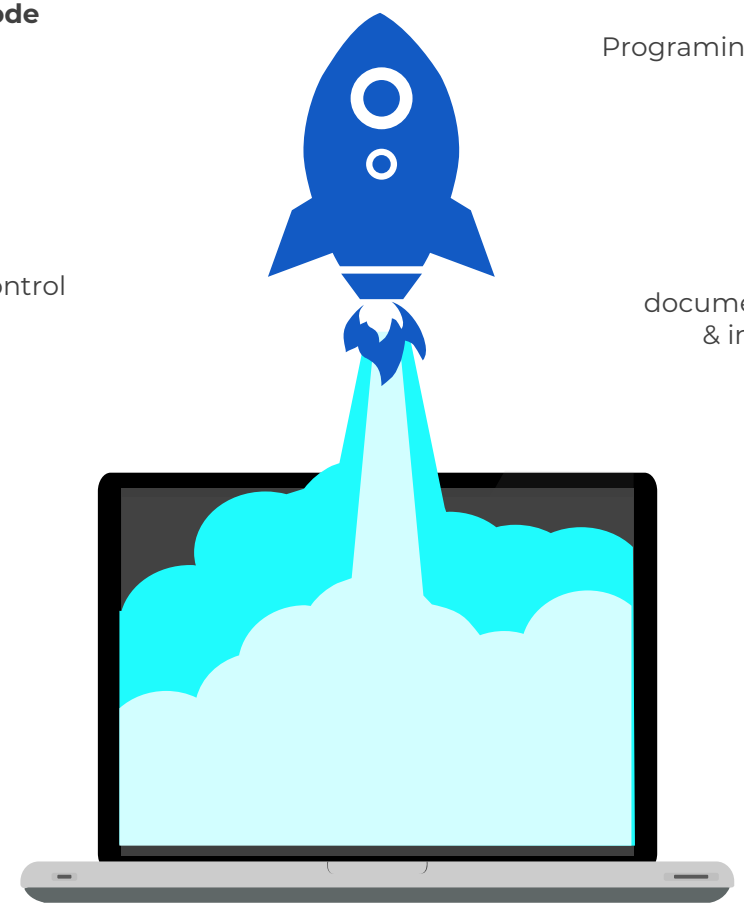
Compiling SV code
Programing DE10

GitHub Desktop
GIT GUI

GitHub
Version control

Notion
Writing documentation & info store

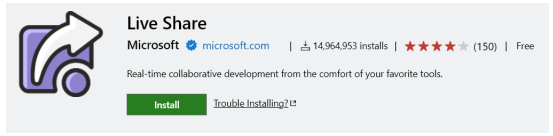
Lucid Chart
Designing diagrams



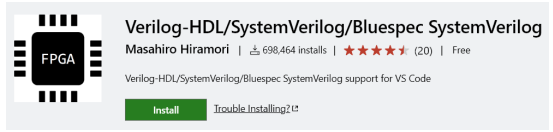
GitLens
Git supercharged

Tracking commits and changes

GitHub Copilot
Questions and debugging



Team collaboration



Syntax highlighting

DONUT 244

De10-lite Oriented Neuron for Understanding Technology - Assistant for Digital Logic CSC 244 at SDSU

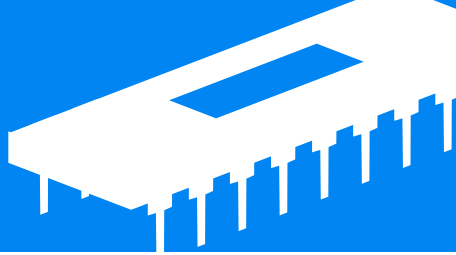
Thank You!

BitBlaster 10bit Processor

Fall 2023, South Dakota State University

John Akujobi - **Amanuel Ayelew** - **Sukhmanjeetsingh LNU**

Challenges and Learning



- **Research Challenge**

The primary difficulty here is the scarcity of information about the BitBlaster 10-bit processor. The lack of recent updates and limited resources can hinder a comprehensive understanding.

- **Tutorial and Learning Challenges**

The absence of readily available tutorials, especially on platforms like YouTube, can make it challenging to grasp the concepts and operational aspects of the BitBlaster 10-bit processor.

- **Code Debugging**

Debugging the code related to the BitBlaster 10-bit processor poses difficulties. Identifying and resolving issues in the code can be time-consuming and may require a deep understanding of the processor's architecture.

-Theoretical Understanding

Grasping the theoretical aspects of the BitBlaster 10-bit processor can be challenging. This includes understanding the underlying principles, architecture, and operational details.

-Top Level Design Challenges

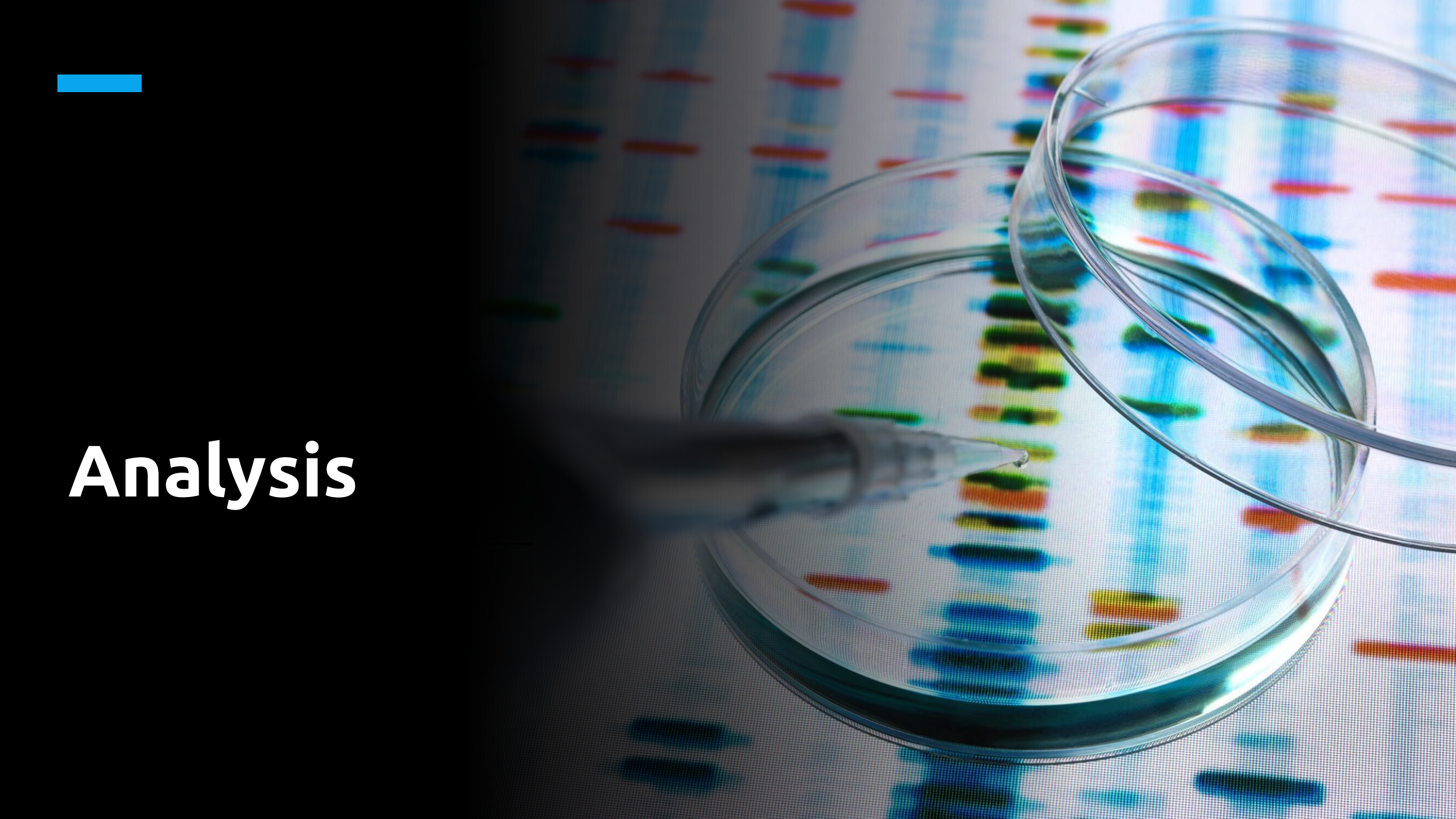
Designing at the top level involves managing complex interactions between different modules and components. This complexity can be overwhelming and may require a holistic understanding of the processor's architecture.

Challenges and Learning

In the project's early stages, understanding interactions between diverse modules, like the controller, ALU, and register file, proved challenging due to the complex architecture. Time was spent unraveling how control signals and data flowed, especially through the crucial bus connecting every module. To overcome this, we used YouTube videos and reference books on digital design. A GitHub project on a 32-bit processor provided valuable insights.

Key Learning Points:
- Developed strong SV coding skills, emphasizing understanding over code quality.
- Clear comprehension of top-level design guided subsequent module coding, particularly in the controller.
- Improved collaborative skills through regular meetings and streamlined communication channels, emphasizing the importance of teamwork in complex projects.

Analysis



Instruction Program

- The copy function enables external input and the instruction register, copies the content from register Ry to the output, sets the input register to the content of register Rx, and clears the operation.
- The Load function involves enabling external input and the instruction register, then loading data into register Rx from slide switches, setting the input register to the content of register Rx, and clearing the operation.

Function	Details	T0=00	T1=01
Add Add Rx,Ry	Add values in Rx and Ry Rx [Rx] +[Ry]	Extrn =1 EnIR = 1	Extrn =0 Rout = Rx EnRout = 1 A
Load Ld Rx	Load data into Rx from slide switches Rx - Data	Enable Extern Enable Instruction	EnRin Rin = Rx

T2 = 10	T3 = 11
Rout =Ry EnRout =1 ALU_Cntr = instruction Gin= 1	EnRout =0 Gin =0 Gout =1 Rin = Rx enRin=1 Clr =1

Conclusion