1. INTRODUCTION

A Random Access Memory (RAM) is a computer memory that can be read and changed in any order, typically used to store working data and machine code. A RAM device allows data items to be read or written in almost the same amount of time irrespective of the physical location of data inside the memory. RAM contains multiplexing and demultiplexing circuitry, to connect the data lines to the addressed storage for reading or writing the entry.

2. PROJECTS

2.1. INSTRUCTION CACHE

2.1.1 Instruction INPUTS/OUTPUTS AMBA4 AXI-Lite Bus

2.1.1.1. Signals of the Read and Write Address channels

Write Port	Read Port	Size	Direction	Description
`AWID`	`ARID`	`AXI_ID_WIDTH`	Output	Address ID, to identify multiple streams
`AWADDR`	`ARADDR`	`AXI_ADDR_WIDTH`	Output	Address of the first beat of the burst
`AWLEN`	`ARLEN`	8	Output	Number of beats inside the burst
`AWSIZE`	`ARSIZE`	3	Output	Size of each beat
`AWBURST`	`ARBURST`	2	Output	Type of the burst
`AWLOCK`	`ARLOCK`	1	Output	Lock type, to provide atomic operations
`AWCACHE`	`ARCACHE`	4	Output	Memory type, progress through the system
`AWPROT`	`ARPROT`	3	Output	Protection type
`AWQOS`	`ARQOS`	4	Output	Quality of Service of the transaction
`AWREGION`	`ARREGION`	4	Output	Region identifier, physical to logical
`AWUSER`	`ARUSER`	`AXI_USER_WIDTH`	Output	User-defined data
`AWVALID`	`ARVALID`	1	Output	xVALID handshake signal
`AWREADY`	`ARREADY`	1	Input	xREADY handshake signal

2.1.1.2. Signals of the Read and Write Data channels

Write Port	Read Port	Size	Direction	Description
`WID`	`RID`	`AXI_ID_WIDTH`	Output	Data ID, to identify multiple streams
`WDATA`	`RDATA`	`AXI_DATA_WIDTH`	Output	Read/Write data
`--`	`RRESP`	2	Output	Read response, current RDATA status
`WSTRB`	`--`	`AXI_STRB_WIDTH`	Output	Byte strobe, WDATA signal
`WLAST`	`RLAST`	1	Output	Last beat identifier
`WUSER`	`RUSER`	`AXI_USER_WIDTH`	Output	User-defined data

Write Port	Read Port	Size	Direction	Description
WVALID	RVALID	1	Output	xVALID handshake signal
WREADY	RREADY	1	Input	xREADY handshake signal

2.1.1.3. Signals of the Write Response channel

Write Port	Size	Direction	Description
BID	AXI_ID_WIDTH	Input	Write response ID, to identify multiple streams
BRESP	2	Input	Write response, to specify the burst status
BUSER	AXI_USER_WIDTH	Input	User-defined data
BVALID	1	Input	xVALID handshake signal
BREADY	1	Output	xREADY handshake signal

2.1.2. Instruction INPUTS/OUTPUTS AMBA3 AHB-Lite Bus

Port	Size	Direction	Description
HRESETn	1	Input	Asynchronous Active Low Reset
HCLK	1	Input	System Clock Input
IHSEL	1	Output	Instruction Bus Select
IHADDR	PLEN	Output	Instruction Address Bus
IHRDATA	XLEN	Input	Instruction Read Data Bus
IHWDATA	XLEN	Output	Instruction Write Data Bus
IHWRITE	1	Output	Instruction Write Select
IHSIZE	3	Output	Instruction Transfer Size
IHBURST	3	Output	Instruction Transfer Burst Size
IHPROT	4	Output	Instruction Transfer Protection Level
IHTRANS	2	Output	Instruction Transfer Type
IHMASTLOCK	1	Output	Instruction Transfer Master Lock
IHREADY	1	Input	Instruction Slave Ready Indicator
IHRESP	1	Input	Instruction Transfer Response

2.1.3. Instruction INPUTS/OUTPUTS Wishbone Bus

Port	Size	Direction	Description
rst	1	Input	Synchronous Active High Reset
clk	1	Input	System Clock Input
iadr	AW	Input	Instruction Address Bus

Port	Size	Direction	Description
idati	DW	Input	Instruction Input Bus
idato	DW	Output	Instruction Output Bus
isel	DW/8	Input	Byte Select Signals
iwe	1	Input	Write Enable Input
istb	1	Input	Strobe Signal/Core Select Input
icyc	1	Input	Valid Bus Cycle Input
iack	1	Output	Bus Cycle Acknowledge Output
ierr	1	Output	Bus Cycle Error Output
iint	1	Output	Interrupt Signal Output

2.2. DATA CACHE

2.2.1. Data INPUTS/OUTPUTS AMBA4 AXI-Lite Bus

2.2.1.1. Signals of the Read and Write Address channels

Write Port	Read Port	Size	Direction	Description
AWID	ARID	AXI_ID_WIDTH	Output	Address ID, to identify multiple streams
AWADDR	ARADDR	AXI_ADDR_WIDTH	Output	Address of the first beat of the burst
AWLEN	ARLEN	8	Output	Number of beats inside the burst
AWSIZE	ARSIZE	3	Output	Size of each beat
AWBURST	ARBURST	2	Output	Type of the burst
AWLOCK	ARLOCK	1	Output	Lock type, to provide atomic operations
AWCACHE	ARCACHE	4	Output	Memory type, progress through the system
AWPROT	ARPROT	3	Output	Protection type
AWQOS	ARQOS	4	Output	Quality of Service of the transaction
AWREGION	ARREGION	4	Output	Region identifier, physical to logical
AWUSER	ARUSER	AXI_USER_WIDTH	Output	User-defined data
AWVALID	ARVALID	1	Output	xVALID handshake signal
AWREADY	ARREADY	1	Input	xREADY handshake signal

2.2.1.2. Signals of the Read and Write Data channels

Write Port	Read Port	Size	Direction	Description
WID	RID	AXI_ID_WIDTH	Output	Data ID, to identify multiple streams
WDATA	RDATA	AXI_DATA_WIDTH	Output	Read/Write data
--	RRESP	2	Output	Read response, current RDATA status
WSTRB	--	AXI_STRB_WIDTH	Output	Byte strobe, WDATA signal
WLAST	RLAST	1	Output	Last beat identifier
WUSER	RUSER	AXI_USER_WIDTH	Output	User-defined data
WVALID	RVALID	1	Output	xVALID handshake signal

Write Port	Read Port	Size	Direction	Description
WREADY	RREADY	1	Input	xREADY handshake signal

2.2.1.3. Signals of the Write Response channel

Write Port	Size	Direction	Description
BID	AXI_ID_WIDTH	Input	Write response ID, to identify multiple streams
BRESP	2	Input	Write response, to specify the burst status
BUSER	AXI_USER_WIDTH	Input	User-defined data
BVALID	1	Input	xVALID handshake signal
BREADY	1	Output	xREADY handshake signal

2.2.2. Data INPUTS/OUTPUTS AMBA3 AHB-Lite Bus

Port	Size	Direction	Description
HRESETn	1	Input	Asynchronous Active Low Reset
HCLK	1	Input	System Clock Input
DHSEL	1	Output	Data Bus Select
DHADDR	PLEN	Output	Data Address Bus
DHRDATA	XLEN	Input	Data Read Data Bus
DHWDATA	XLEN	Output	Data Write Data Bus
DHWRITE	1	Output	Data Write Select
DHSIZE	3	Output	Data Transfer Size
DHBURST	3	Output	Data Transfer Burst Size
DHPROT	4	Output	Data Transfer Protection Level
DHTRANS	2	Output	Data Transfer Type
DHMASTLOCK	1	Output	Data Transfer Master Lock
DHREADY	1	Input	Data Slave Ready Indicator
DHRESP	1	Input	Data Transfer Response

2.2.3. Data INPUTS/OUTPUTS Wishbone Bus

Port	Size	Direction	Description
rst	1	Input	Synchronous Active High Reset
clk	1	Input	System Clock Input
dadr	AW	Input	Data Address Bus
ddati	DW	Input	Data Input Bus

Port	Size	Direction	Description
`ddato`	`DW`	Output	Data Output Bus
`dsel`	`DW/8`	Input	Byte Select Signals
`dwe`	1	Input	Write Enable Input
`dstb`	1	Input	Strobe Signal/Core Select Input
`dcyc`	1	Input	Valid Bus Cycle Input
`dack`	1	Output	Bus Cycle Acknowledge Output
`derr`	1	Output	Bus Cycle Error Output
`dint`	1	Output	Interrupt Signal Output

3. WORKFLOW

3.1. FRONT-END OPEN SOURCE TOOLS

3.1.1. Verilator

SystemVerilog System Description Language Simulator

A System Description Language Simulator (translator) is a computer program that translates computer code written in a Programming Language (the source language) into a Hardware Design Language (the target language). The compiler is primarily used for programs that translate source code from a high-level programming language to a low-level language to create an executable program.

type:

```
git clone http://git.veripool.org/git/verilator

cd verilator
autoconf
./configure
make
sudo make install
```

```
cd sim/verilog/regression/wb/vtor
source SIMULATE-IT
```

```
cd sim/verilog/regression/ahb3/vtor
source SIMULATE-IT
```

3.1.2. Icarus Verilog

Verilog Hardware Description Language Simulator

A Hardware Description Language Simulator uses mathematical models to replicate the behavior of an actual hardware device. Simulation software allows for modeling of circuit operation and is an invaluable analysis tool. Simulating a circuit's behavior before actually building it can greatly improve design efficiency by making faulty designs known as such, and providing insight into the behavior of electronics circuit designs.

type:

```
git clone https://github.com/steveicarus/iverilog
```

```
cd iverilog
sh autoconf.sh
./configure
make
sudo make install
```

```
cd sim/verilog/regression/wb/iverilog
source SIMULATE-IT
```

```
cd sim/verilog/regression/ahb3/iverilog
source SIMULATE-IT
```

3.1.3. GHDL

VHDL Hardware Description Language Simulator

A Hardware Description Language Simulator uses mathematical models to replicate the behavior of an actual hardware device. Simulation software allows for modeling of circuit operation and is an invaluable analysis tool. Simulating a circuit's behavior before actually building it can greatly improve design efficiency by making faulty designs known as such, and providing insight into the behavior of electronics circuit designs.

type:

```
git clone https://github.com/ghdl/ghdl
```

```
cd ghdl
./configure --prefix=/usr/local
make
sudo make install
```

```
cd sim/vhdl/regression/wb/ghdl
source SIMULATE-IT
```

```
cd sim/vhdl/regression/ahb3/ghdl
source SIMULATE-IT
```

3.1.4. Yosys-ABC

Verilog Hardware Description Language Synthesizer

A Hardware Description Language Synthesizer turns a RTL implementation into a Logical Gate Level implementation. Logical design is a step in the standard design cycle in which the functional design of an electronic circuit is converted into the representation which captures logic operations, arithmetic operations, control flow, etc. In EDA parts of the logical design is automated using synthesis tools based on the behavioral description of the circuit.

Hardware Description Language Optimizer

A Hardware Description Language Optimizer finds an equivalent representation of the specified logic circuit under specified constraints (minimum area, pre-specified delay). This tool combines scalable logic optimization based on And-Inverter Graphs (AIGs), optimal-delay DAG-based technology mapping for look-up tables and standard cells, and innovative algorithms for sequential synthesis and verification.

type:

```
git clone https://github.com/YosysHQ/yosys

cd yosys
make
sudo make install
```

```
cd synthesis/yosys
source SYNTHESIZE-IT
```

3.2. BACK-END OPEN SOURCE TOOLS

Library type:

```
sudo apt update
sudo apt upgrade

sudo apt install bison cmake flex freeglut3-dev libcairo2-dev libgsl-dev \
libncurses-dev libx11-dev m4 python-tk python3-tk swig tcl tcl-dev tk-dev tcsh
```

```
mkdir qflow
cd qflow
```

3.2.1. Qflow

Back-End Workflow

type:

```
git clone https://github.com/RTimothyEdwards/qflow

cd qflow
./configure
make
sudo make install
```

3.2.2. Magic

Floor-Planner

A Floor-Planner of an Integrated Circuit (IC) is a schematic representation of tentative placement of its major functional blocks. In modern electronic design process floor-plans are created during the floor-planning design stage, an early stage in the hierarchical approach to Integrated Circuit design. Depending on the design methodology being followed, the actual definition of a floor-plan may differ.

Standard Cell Checker

A Standard Cell Checker is a geometric constraint imposed on Printed Circuit Board (PCB) and Integrated Circuit (IC) designers to ensure their designs function properly, reliably, and can be produced with acceptable yield. Design Rules for production are developed by hardware engineers based on the capability of their processes to realize design intent. Design Rule Checking (DRC) is used to ensure that designers do not violate design rules.

Standard Cell Editor

A Standard Cell Editor allows to print a set of standard cells. The standard cell methodology is an abstraction, whereby a low-level VLSI layout is encapsulated into a logical representation. A standard cell is a group of transistor and interconnect structures that provides a boolean logic function (AND, OR, XOR, XNOR, inverters) or a storage function (Flip-Flop or Latch).

type:

```
git clone https://github.com/RTimothyEdwards/magic

cd magic
./configure
make
sudo make install
```

3.2.3. Graywolf

Standard Cell Placer

A Standard Cell Placer takes a given synthesized circuit netlist together with a technology library and produces a valid placement layout. The layout is optimized according to the aforementioned objectives and ready for cell resizing and buffering, a step essential for timing and signal integrity satisfaction. Physical design flow are iterated a number of times until design closure is achieved.

type:

```
git clone https://github.com/rubund/graywolf
```

```
cd graywolf
mkdir build
cd build
cmake ..
make
sudo make install
```

3.2.4. OpenSTA

Standard Cell Timing-Analizer

A Standard Cell Timing-Analizer is a simulation method of computing the expected timing of a digital circuit without requiring a simulation of the full circuit. High-performance integrated circuits have traditionally been characterized by the clock frequency at which they operate. Measuring the ability of a circuit to operate at the specified speed requires an ability to measure, during the design process, its delay at numerous steps.

type:

```
git clone https://github.com/The-OpenROAD-Project/OpenSTA
```

```
cd OpenSTA
mkdir build
cd build
cmake ..
make
sudo make install
```

3.2.5. Qrouter

Standard Cell Router

A Standard Cell Router takes pre-existing polygons consisting of pins on cells, and pre-existing wiring called pre-routes. Each of these polygons are associated with a net. The primary task of the router is to create geometries such that *all terminals assigned to the same net are connected, no terminals assigned to different nets are connected, and all design rules are obeyed.*

type:

```
git clone https://github.com/RTimothyEdwards/qrouter

cd qrouter
./configure
make
sudo make install
```

3.2.6. Irsim

Standard Cell Simulator

A Standard Cell Simulator treats transistors as ideal switches. Extracted capacitance and lumped resistance values are used to make the switch a little bit more realistic than the ideal, using the RC time constants to predict the relative timing of events. This simulator represents a circuit in terms of its exact transistor structure but describes the electrical behavior in a highly idealized way.

type:

```
git clone https://github.com/RTimothyEdwards/irsim

cd irsim
./configure
make
sudo make install
```

3.2.7. Netgen

Standard Cell Verifier

A Standard Cell Verifier compares netlists, a process known as LVS (Layout vs. Schematic). This step ensures that the geometry that has been laid out matches the expected circuit. The greatest need for LVS is in large analog or mixed-signal circuits that cannot be simulated in reasonable time. LVS can be done faster than simulation, and provides feedback that makes it easier to find errors.

type:

```
git clone https://github.com/RTimothyEdwards/netgen

cd netgen
./configure
```

```
make
sudo make install
```

```
cd synthesis/qflow
source FLOW-IT
```

3.3. FOR WINDOWS USERS!

1. Settings → Apps → Apps & features → Related settings, Programs and Features → Turn Windows features on or off → Windows Subsystem for Linux
2. Microsoft Store → INSTALL UBUNTU

Library type:

```
sudo apt update
sudo apt upgrade

sudo apt install bison cmake flex freeglut3-dev libcairo2-dev libgsl-dev \
libncurses-dev libx11-dev m4 python-tk python3-tk swig tcl tcl-dev tk-dev tcsh
```

3.3.1. Front-End

type:

```
sudo apt install verilator
sudo apt install iverilog
sudo apt install ghdl

cd /mnt/c/../sim/verilog/regression/wb/iverilog
source SIMULATE-IT
```

```
sudo apt install yosys

cd /mnt/c/../synthesis/yosys
source SYNTHESIZE-IT
```

3.3.2. Back-End

type:

```
mkdir qflow
cd qflow

git clone https://github.com/RTimothyEdwards/magic
git clone https://github.com/rubund/graywolf
```

```
git clone https://github.com/The-OpenROAD-Project/OpenSTA
git clone https://github.com/RTimothyEdwards/qrouter
git clone https://github.com/RTimothyEdwards/irsim
git clone https://github.com/RTimothyEdwards/netgen
git clone https://github.com/RTimothyEdwards/qflow
```

```
cd /mnt/c/../synthesis/qflow
source FLOW-IT
```

4. CONCLUSION