RISC-V® Summit

Reverse Engineering of Rocket-Chip

December 8-10 | Virtual Event

Usman Institute of Technology
عثمان انسٹیٹیوٹ آف ٹیکنالوجی

MERL-UIT
PAKISTAN

Brought to you by
informa tech

riscvsummit.com #RISCVSUMMIT

RISC-V® Summit

December 8-10 | Virtual Event

MERL-UIT PAKISTAN

Reverse Engineering of Rocket-Chip

Dr. Roomi Naqvi, MERL Researchers & Students

Director
MERL-PAKISTAN

#RISCVSUMMIT

Outline

- **Motivation and Goals**
- **Overview of Rocket Chip Generator[1]**
- **Deconstructing the Software Architecture – Rocket Chip Generator**
- **Putting it all Together - Micro-Architecture and Software Specification Document (MASS)**
- **Aghaaz (آغاز) Chip SoC framework – A new beginning**
  - Configuring the Aghaaz SoC
- **Summary and Future Directions**

1. Asanović, Krste/Avizienis, Rimas/Bachrach, Jonathan et al. The Rocket Chip Generator. (2016)

Motivation and Goals

- Develop a Robust SoC generator framework for developing customized SoCs
- Demonstrate our indigenous methodology used to reverse engineer Rocket-Chip for the generation of a custom System on a Chip (SoC).
- An overview of the *Micro-Architecture and Software Specification (MASS)* document developed through our methodology
- Used the ***MASS*** document together with Rocket-Chip generator to generate **Aghaaz (آغاز) SoC**

RISC-V® Summit

Overview of Rocket-Chip SoC Generator

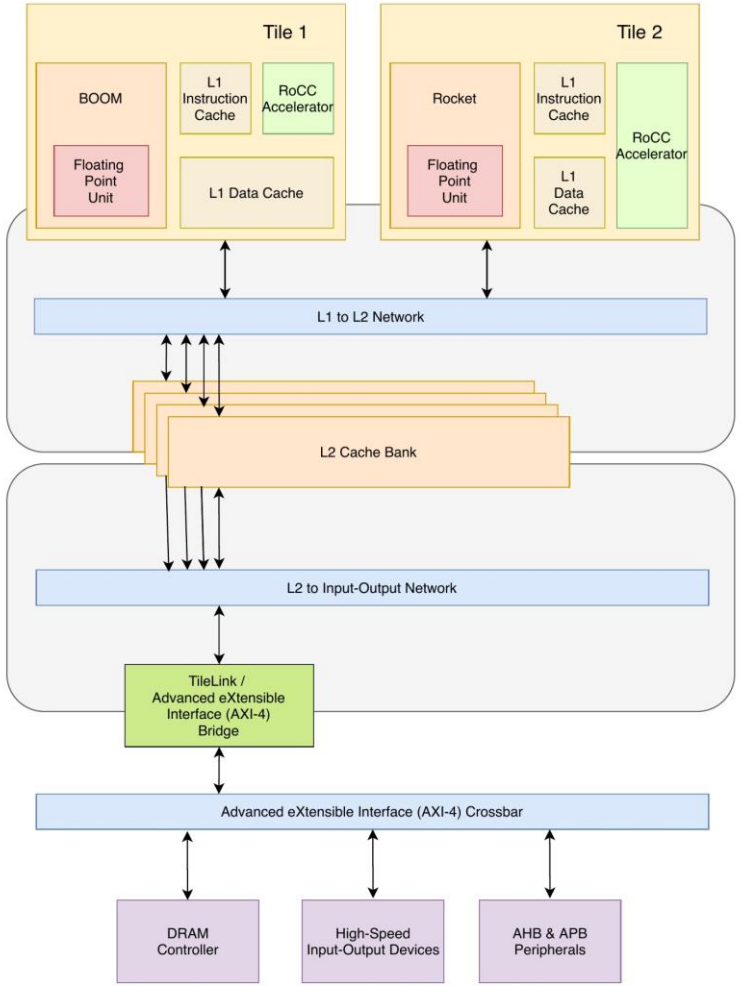
Brought to you by informa tech

riscvsummit.com #RISCVSUMMIT

Rocket-Chip Generator Overview

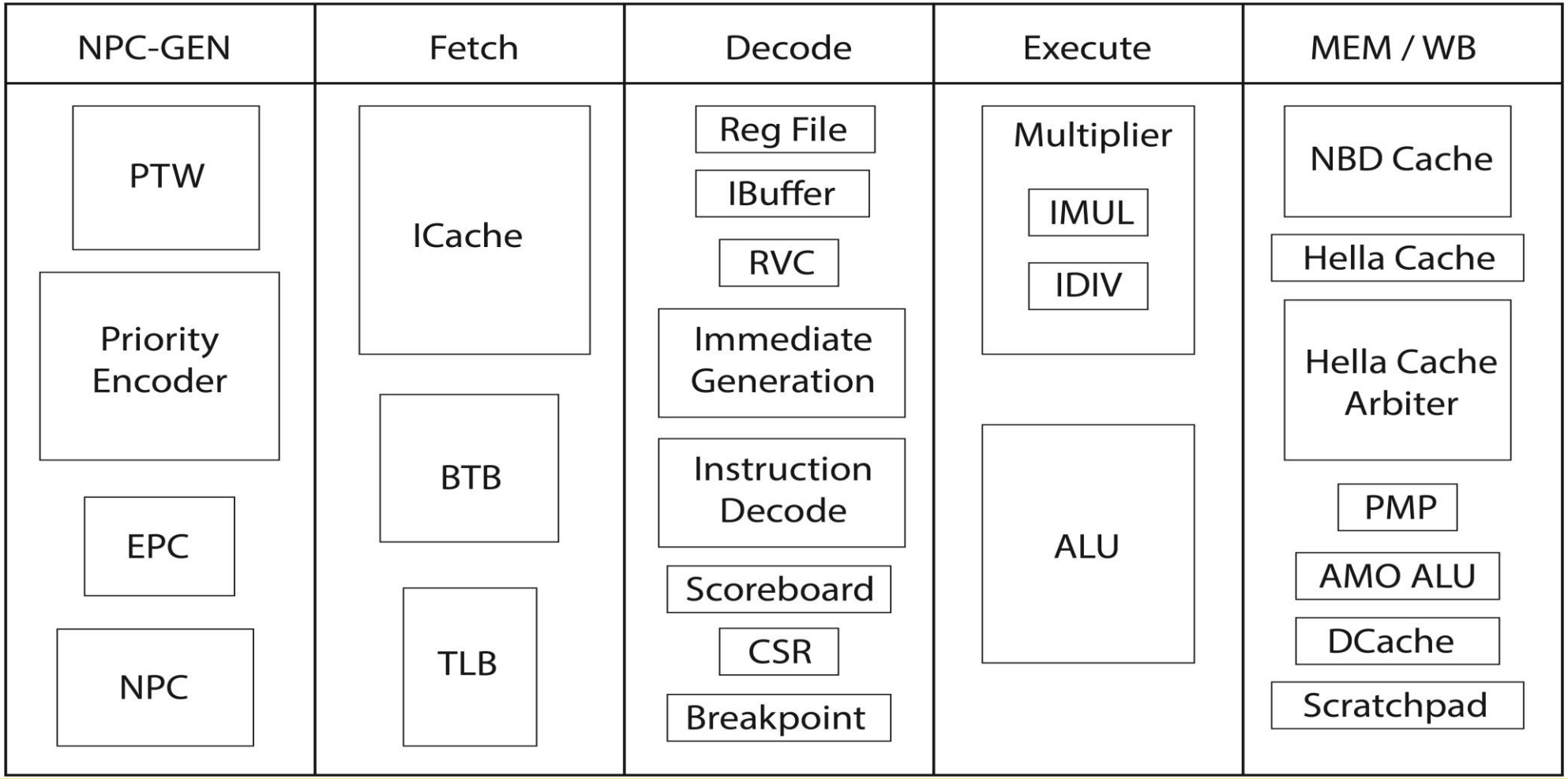
- Collection of SoC building blocks
  - Parameterized
  - Standard Interfaces
- Chisel: open source hardware construction language embedded in scala
- FIRRTL is used to emit Verilog from the Chisel Model
- Composes RISC-V ISA based platforms

Cores	in-order scalar , out-of-order
Core Tiles	FPU, L1 Caches, Custom Accelerators, FPU
Tile Link	On chip fabric , Coherent Caches
Peripheral	Limited Support



Rocket-Core Micro-Architecture

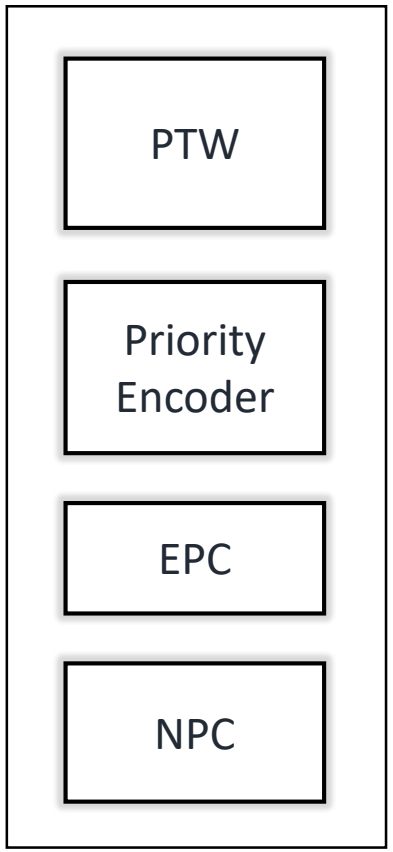
- Rocket Core : 5-stage in-order scalar core generator



Rocket-Core Modules Description

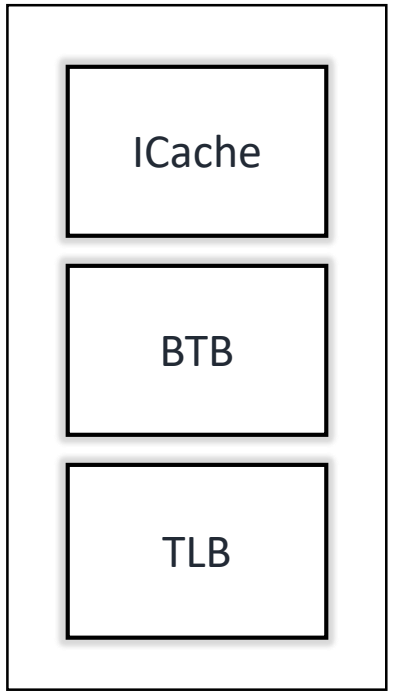
NPC-Gen Modules Description

Module	Description
PTW	Page table is used to access the virtual memory. `nL2TLBEntries` parameter is used by Rocket-Chip PTW for controlling TLB entries.
Priority Encoder	Priority Encoder take all of the data inputs one at a time and converts them into an equivalent binary code at its output. Rocket-Chip Priority Encoder takes pins as inputs from BTB and EPC (from all stages) to select the value of NPC.
EPC	The Exception Program Counter (EPC) is used to store the address of the instruction that was executing when the exception was generated. The size of the EPC register depends upon the XLen which can be ~~of~~ either [31:0] or [63:0].
NPC	New Program Counter (NPC) is used to increment +4 in current PC and store it in a register. The size of the NPC register depends upon the XLen which can be either [31:0] or [63:0].



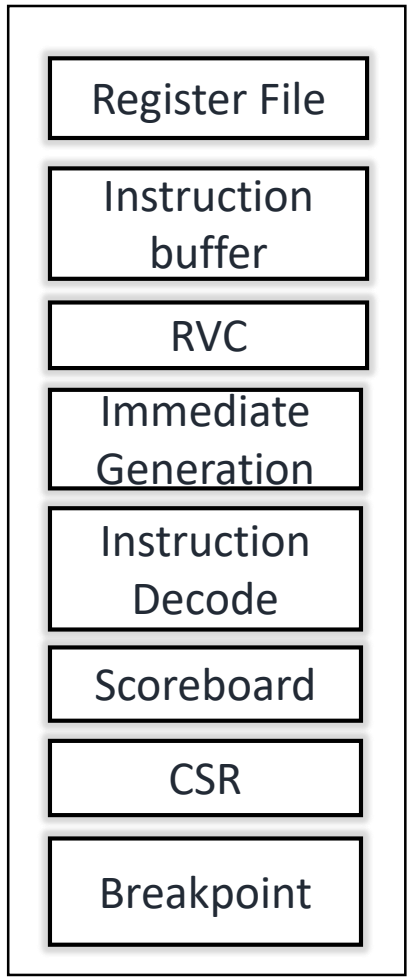
Fetch Modules Description

Module	Description
ICache	The Instruction Cache acts as a buffer memory between external memory and the core processor. In Rocket-Chip size of the ICache sets, ways and block can be configured from the Config.scala file in subsystem directory of Scala files. Size of ICache is calculated by: *nWays x nSets x blockBytes*
BTB	Branch Target Buffer (BTB) is a cache-like component in processors that is used for branch prediction. The main concept of the BTB is to store the program counter (PC) of a branch instruction, and the PC of the target of the branch. The bit size of branch prediction is [1:0].
TLB	Translation Lookaside Buffer (TLB) is a memory cache which is used to keep track of recently used transactions and reduce the time taken to access a user memory location's. It is connected with Rocket-Chip PTW and work as cache for PTW. TLB entries is controlled by PTW of Rocket-Chip.



Decode Modules Description

Module	Description
Register File	Registers are temporary storage locations that hold data and addresses.The width of the rocket-Chip file register is fixed by the Xlen parameter either [31:0] or [63:0].
Instruction buffer	Ibuffer is a temporary register where the opcode of the currently fetched instruction from instruction memory is stored. The RV32 or RV64 instruction opcode is [6:0] bits.
RVC	This Module holds RISC-V Compressed (C-Extension) Instructions. To use the RISC-V Compressed instructions, `useCompressed` [Bool], which is present in Rocket-Chip config file, should be true.
Immediate Generation	Immediate generation module generates immediate for instructions of RISC-V. This module is located in RocketCore.scala and returns immediate bits depending upon the size of ISA [31:0] or [63:0].
Instruction Decode	This module decodes instructions for RISC-V which come from Instruction Memory.
Scoreboard	Scoreboards are designed to control the flow of data between registers and multiple arithmetic units in the presence of conflicts caused by hardware resource limitations (structural hazards) and by dependencies between instructions (data hazards).Scoreboard module lies in RocketCore.scala.
CSR	Control and Status Register (CSR) module contains control registers to enable and handle different interrupts and exceptions.
Breakpoint	A breakpoint is a location in executable code at which the operating system stops execution and breaks into the debugger. This allows you to analyze the target and issue debugger commands.

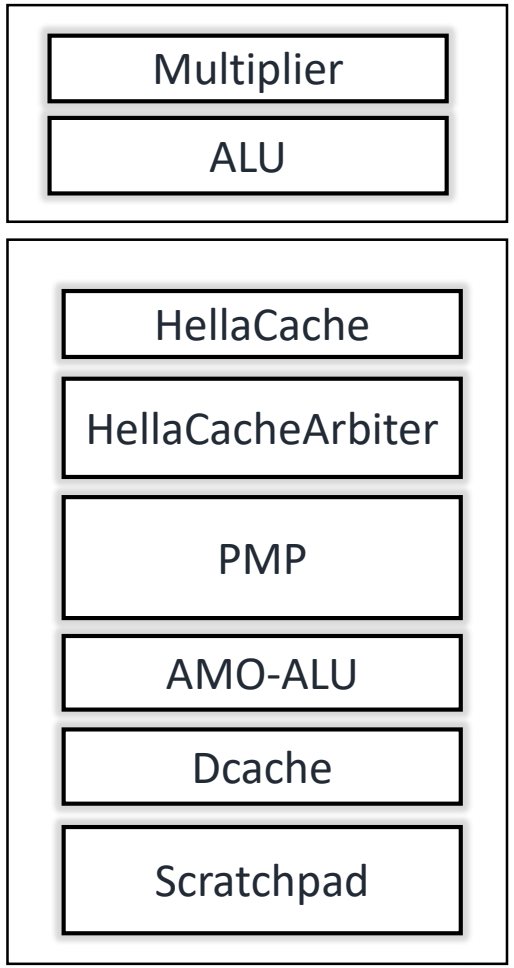


Execute Modules Description

Module	Description
Multiplier	This module is present inside the Rocket-Core execute stage and is used to perform M-Type Instructions. Multiplier operation is parameterizable and depends upon the XLen width (Width of the operands of Multiplier). XLen either [31:0] or [63:0].
ALU	An arithmetic logic unit (ALU) used to perform arithmetic and logic operations in Rocket-Core. ALU operands is parameterizable depends upon the XLen width either [31:0] or [63:0].

Memory/ Write-Back Modules Description

Module	Description
HellaCache	HellaCache is the acronym of Level 1 Data cache in Rocket-Chip. L1 Dcache directly built into the microprocessor, which is used for storing the microprocessor's recently accessed information, thus it is also called the primary cache. The size of the cache depends upon the width of the ISA.
HellaCache Arbiter	This module is for accessing L1 Dcache (HellaCache), by means of requests generated from other modules for read and write operations.
PMP	Physical Memory Protection (PMP) is a part of the RISC-V Privileged Architecture Specification which describes the interface for a standard RISC-V memory protection unit. The PMP defines a finite number of PMP regions which can be individually configured to enforce access permissions to a range of addresses in memory
AMO-ALU	This module is used to perform the A Extension RISCV operations. To use the Rocket Core Atomic ALU module, `useAtmoics` [Bool],which is in the rocket-Chip config file, should be true.
Dcache	This Module is used to hold the data for temporary period of time.Size of the Dcache sets, ways and block can be configured from the Config.scala file in subsystem directory of Scala files. Size of DCache is calculated by*: nWays x nSets x blockBytes*.
Scratchpad	(SPRAM) is a high-speed internal memory directly connected to the CPU core and used for temporary storage to hold very small items of data for rapid retrieval. To add the scratchpad in our SoC we need to call the scratchpad class in our config file which lies in system directory of Scala.



RISC-V® Summit

De-constructing Software Architecture Rocket-Chip

Brought to you by informa tech

riscvsummit.com #RISCVSUMMIT

Code Complexity

- The complexity of code of Rocket-Chip is high, because it contains 372 files in 73 directories, after the submodules the numbers increase to 151,613 files in 8,625 directories and after running the build script to generate an SoC the numbers grow to 185,049 files in 10,780 directories[1]

- The Rocket-Chip implementation code includes many variable names and hundreds of lines of code without any meaningful comments

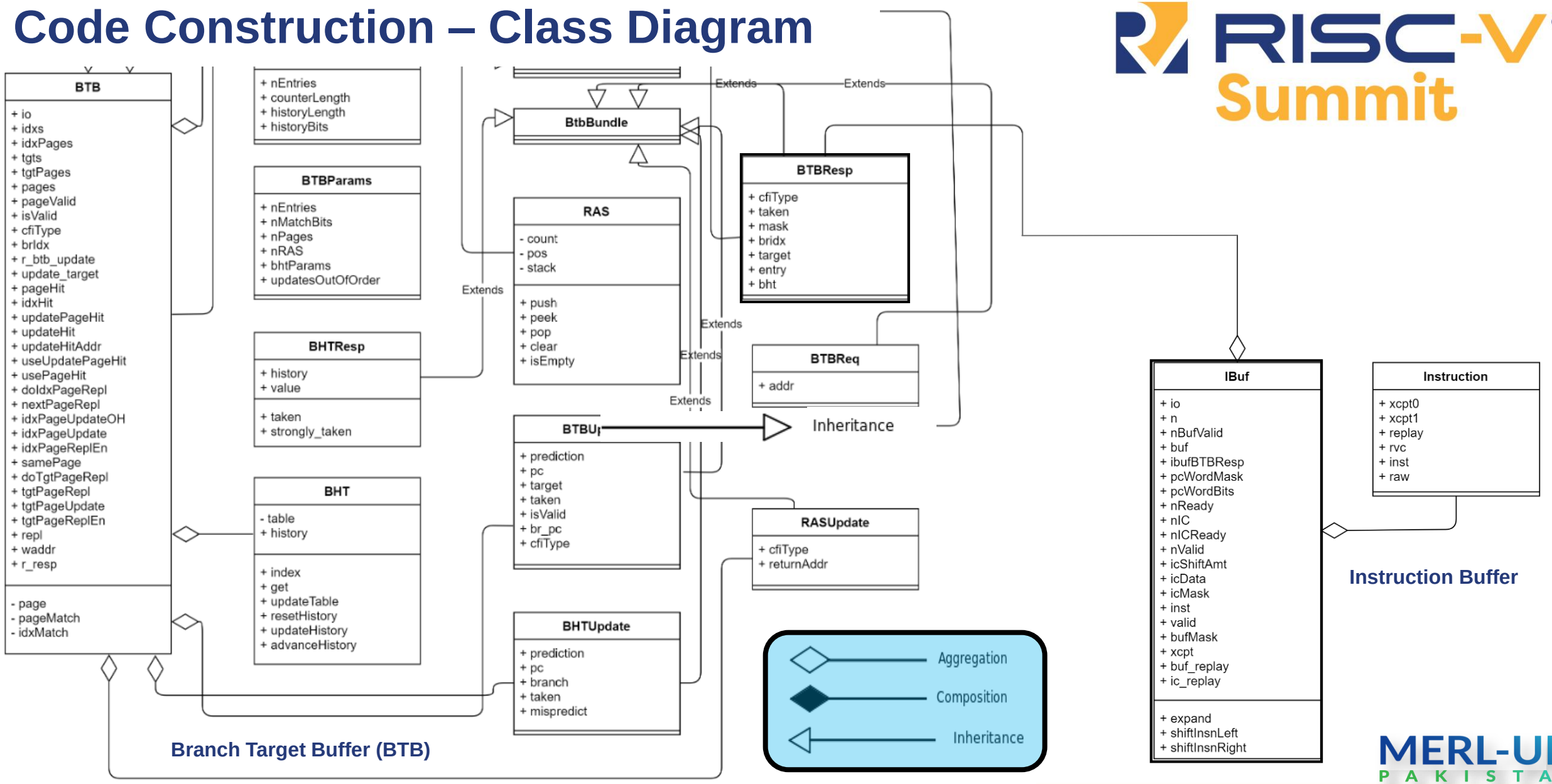
- This lack of documentation aggravates the ability to understand and results in difficulty in modifying Rocket-Core or its component for customized requirements

[1] *Durrum, J., Bryant, A. and Porter, J., 2019. Reverse Engineering RISC-V Generator-Based Designs for Trust and Assurance. AFRL/RYDT WPAFB United States.*

Our Approach

- Object Oriented Programming (OOP) and high-level Functional Programming concepts helps in understanding the code of Rocket-Chip
- Code complexity is reduced by means of Flowcharts, UML Class Diagrams and Block Diagrams of each modules of Rocket-Chip
- Assigned a team of undergraduate CS students to go through the code line-by line
- Develop a detailed document fusing the Software Architecture of the implementation with the Micro-architecture of Rocket-Chip

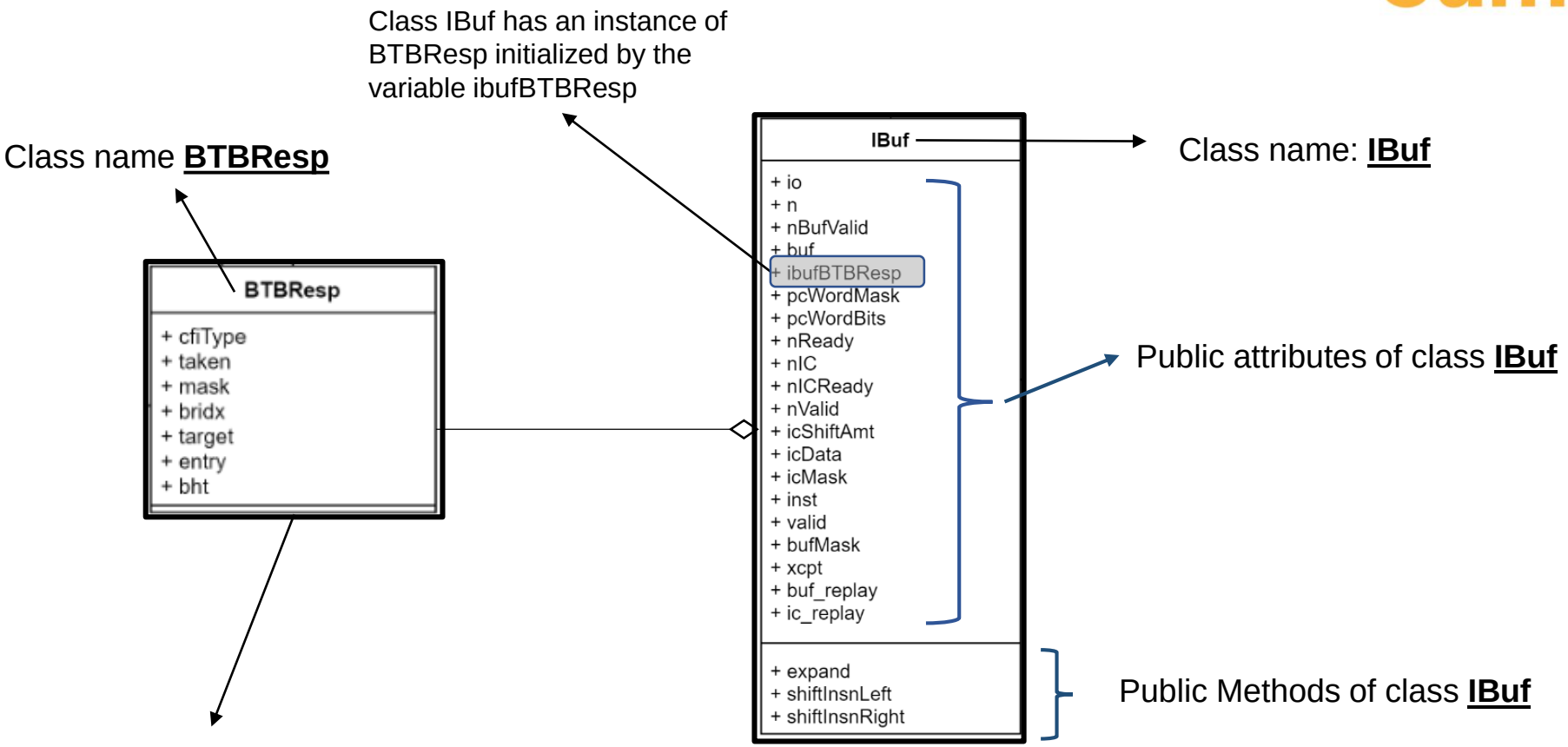
Code Construction – Class Diagram



Branch Target Buffer (BTB)

Instruction Buffer

Class Diagram Details



Class **BTBResp** is inherited from its parent class (**BtbBundle**), therefore it uses the methods of its parent class.

Object Oriented Code of Rocket-Chip

```
abstract trait DecodeConstants extends HasCoreParameters
{
  val table: Array[(BitPat, List[BitPat])]
}
```

Abstraction

Abstraction is achieved by **abstract** keyword, creating an abstract trait of **DecodeConstants**.

```
class RegFile(n: Int, w: Int, zero: Boolean = false) {
  private def access(addr: UInt) = rf(~addr(log2Up(n)-1,0))
  private var canRead = true
}
```

Encapsulation

Private access modifier is used to encapsulate/hide method '**access**' from other extended classes.

```
class PMP(implicit p: Parameters) extends PMPReg {..
```

Inheritance

Keyword extend is used to extend/inherit the **PMP** class with parent class **PMPReg**

```
class DCache(staticIdForMetadataUseOnly: Int, val crossing: ClockCrossingType)(implicit p: Parameters)
extends HellaCache(staticIdForMetadataUseOnly)(p) {
  override def getOMSRAMs(): Seq[OMSRAM] = Seq(module.dcacheImpl.omSRAM)
}
```

Polymorphism

Keyword **override** is used to override method **getOMSRAMs**, which is already defined in its parent class.

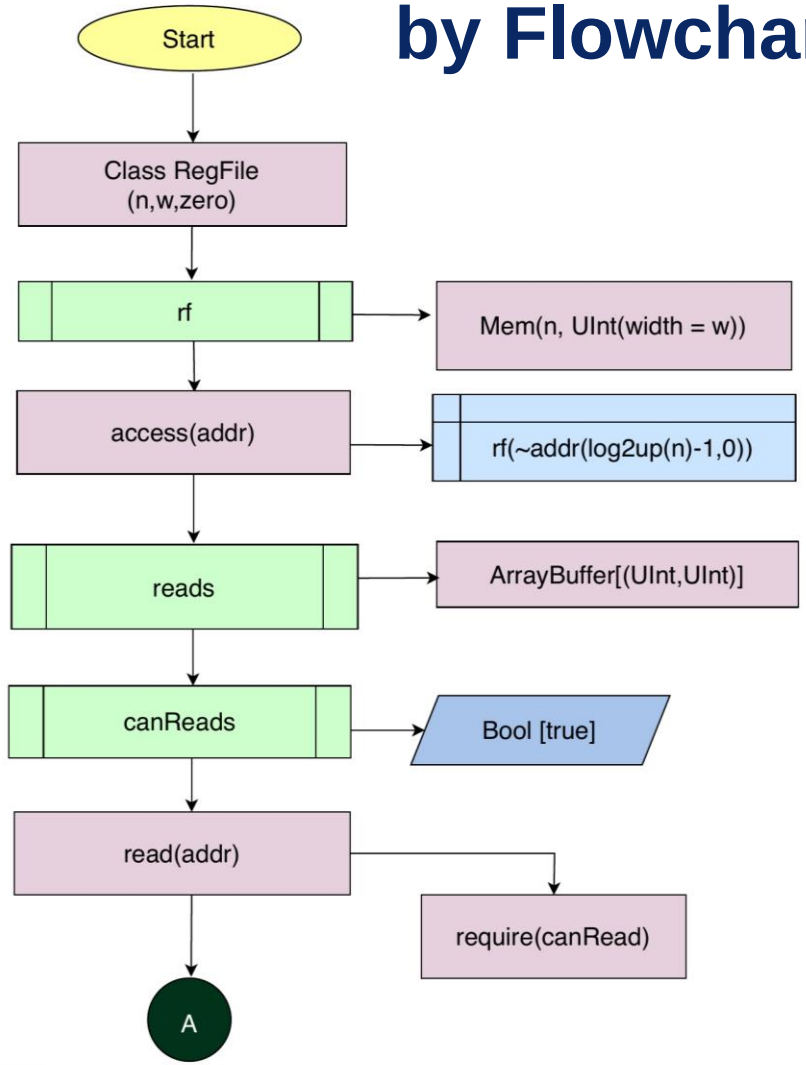
RISC-V® Summit

Putting it all Together
(MASS: Hardware and Software)

Brought to you by informa tech

riscvsummit.com #RISCVSUMMIT

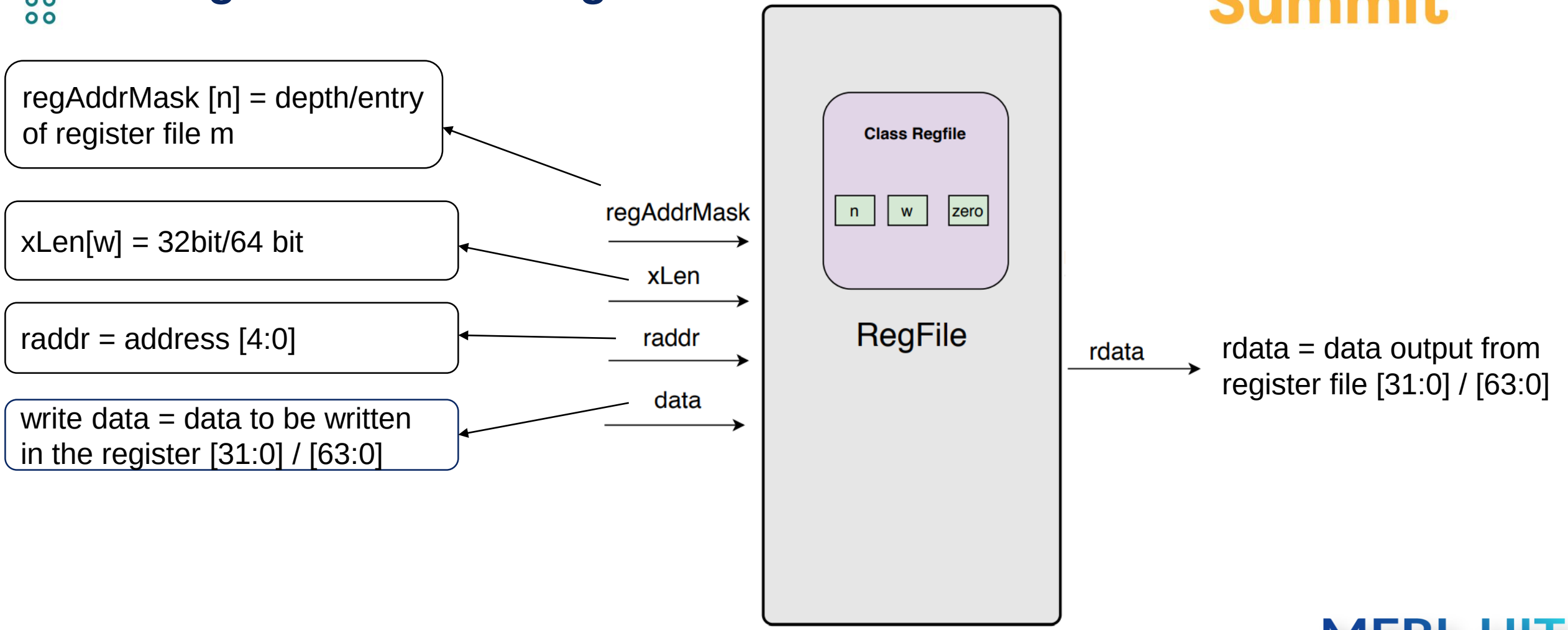
Code Explanation and Back-tracing by Flowcharts




```
class RegFile(n: Int, w: Int, zero: Boolean = false) {
  val rf = Mem(n, UInt(width = w))
  private def access(addr: UInt) = rf(~addr(log2Up(n)-1,0))
  private val reads = ArrayBuffer[(UInt,UInt)]()
  private var canRead = true
  def read(addr: UInt) = {
    require(canRead)
```

- Class **RegFile**: initiated with three parameters n, w and zero
- **Mem** function creates memory/register file
- Private method **access** is defined for accessing the register file data by taking the address as its input parameter
- Private variable **reads**
- Private variable **canRead** *true* for read state, *false* for write state
- "**read**" method : read data from register file , addr is parameter

RegFile – Block Diagram View



Deep Dive into ALU Code View

Object ALU is defined

ALU operation bits are defined in method SZ_ALU_FN

ALU operations are defined

Importing all properties of Chisel

```
package freechips.rocketchip.rocket

import Chisel._
import freechips.rocketchip.config.Parameters

import freechips.rocketchip.tile.CoreModule
```

Class Parameters is imported from directory config of rocket-chip

Class CoreModule is imported from directory tile of rocket-chip

```
  val SZ_ALU_FN = 4
  def FN_X    = BitPat("b????")

  def FN_ADD  = UInt(0)
  def FN_SL   = UInt(1)
  def FN_SEQ  = UInt(2)
  def FN_SNE  = UInt(3)
  def FN_XOR  = UInt(4)
  def FN_SR   = UInt(5)
  def FN_OR   = UInt(6)
  def FN_AND  = UInt(7)
  def FN_SUB  = UInt(10)
  def FN_SRA  = UInt(11)
  def FN_SLT  = UInt(12)
  def FN_SGE  = UInt(13)
  def FN_SLTU = UInt(14)
  def FN_SGEU = UInt(15)

  def FN_DIV  = FN_XOR
  def FN_DIVU = FN_SR
  def FN_REM  = FN_OR

  def FN_REMU = FN_AND

  def FN_MUL    = FN_ADD
  def FN_MULH   = FN_SL
  def FN_MULHSU = FN_SEQ
  def FN_MULHU  = FN_SNE

  def isMulFN(fn: UInt, cmp: UInt) = fn(1,0) === cmp(1,0)
  def isSub(cmd: UInt) = cmd(3)
  def isCmp(cmd: UInt) = cmd >= FN_SLT
  def cmpUnsigned(cmd: UInt) = cmd(1)
  def cmpInverted(cmd: UInt) = cmd(0)
  def cmpEq(cmd: UInt) = !cmd(3)
}
```

```
class ALU(implicit p: Parameters) extends CoreModule()(p)
{
  val io = new Bundle {
    val dw = Bits(INPUT, SZ_DW)
    val fn = Bits(INPUT, SZ_ALU_FN)
    val in2 = UInt(INPUT, xLen)
    val in1 = UInt(INPUT, xLen)
    val out = UInt(OUTPUT, xLen)
    val adder_out = UInt(OUTPUT, xLen)
    val cmp_out = Bool(OUTPUT)
  }
```

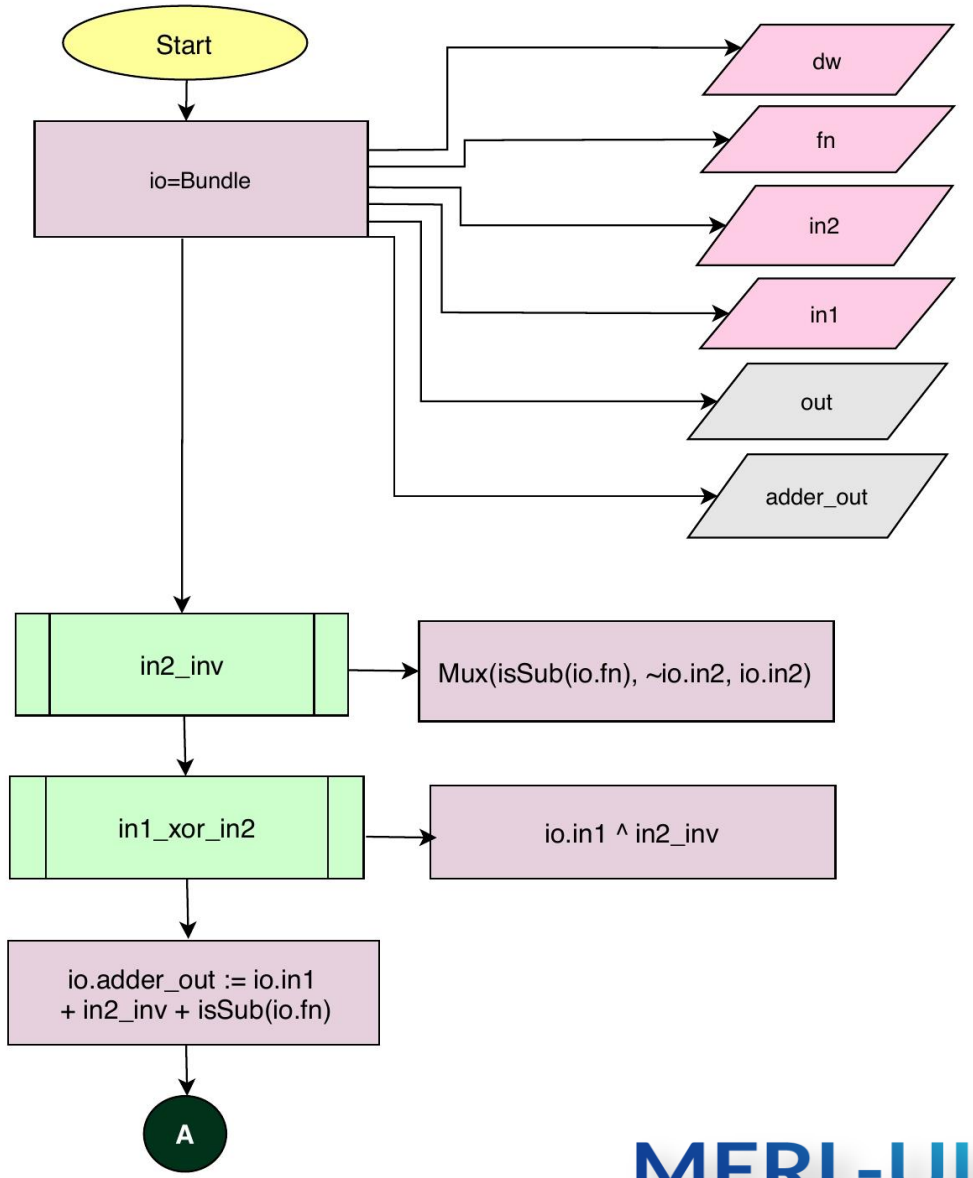
Input/Output of class ALU are defined

```
// ADD, SUB
  val in2_inv = Mux(isSub(io.fn), ~io.in2, io.in2)
  val in1_xor_in2 = io.in1 ^ in2_inv
  io.adder_out := io.in1 + in2_inv + isSub(io.fn)
```

MERL-UIT PAKISTAN

ALU Code Explanation

```
// ADD, SUB
 val in2_inv = Mux(isSub(io.fn), ~io.in2, io.in2)
 val in1_xor_in2 = io.in1 ^ in2_inv
 io.adder_out := io.in1 + in2_inv + isSub(io.fn)
```



Rocket-Core ALU: Code block performs addition/subtraction

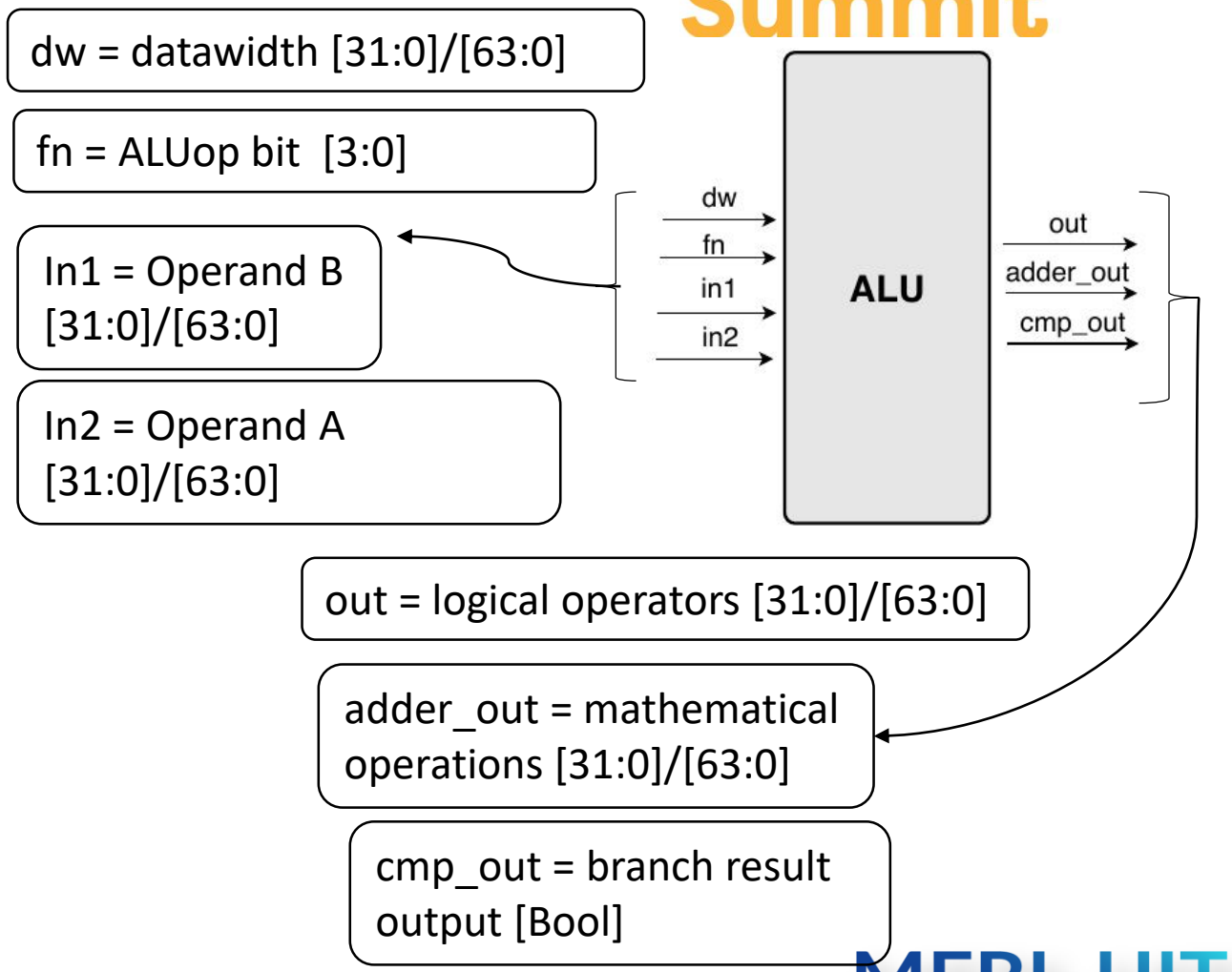
- **adder out**: Result of the Addition/ Substraction
- **in2_inv :** initialized with Mux function which has parameters, **isSub** function with parameter io.fn as selector, Not of io.in2 as input1 of Mux, and io.in2 as input2 of Mux
- **in1_xor_in2:** initialized with io.in1 XOR'ed (^) with in2_inv.
- **io.adder_out:** wired to io.in1 + in2_inv + isSub of io.fn

MERL-UIT PAKISTAN

ALU: Class Diagram to Block Diagram

```
class ALU(implicit p: Parameters) extends CoreModule()(p)
{
  val io = new Bundle {
    val dw = Bits(INPUT, SZ_DW)
    val fn = Bits(INPUT, SZ_ALU_FN)
    val in2 = UInt(INPUT, xLen)
    val in1 = UInt(INPUT, xLen)
    val out = UInt(OUTPUT, xLen)
    val adder_out = UInt(OUTPUT, xLen)
    val cmp_out = Bool(OUTPUT)
  }
```

ALU I/O	Description
dw	Data width is an input of ALU which can be 32bit or 64 bit
fn	fn (ALU operation Bits) as input, width is predefined by 4
in2	Second input, here xLen can be changed to 32 or 64bits
in1	First input, here xLen can be changed to 32 or 64bits
out	ALU output for logical operators, generated by the ALU
adder_out	ALU output for mathematical operators, generated by the ALU
cmp_out	ALU output indicating branch taken or not taken



How did we do it?

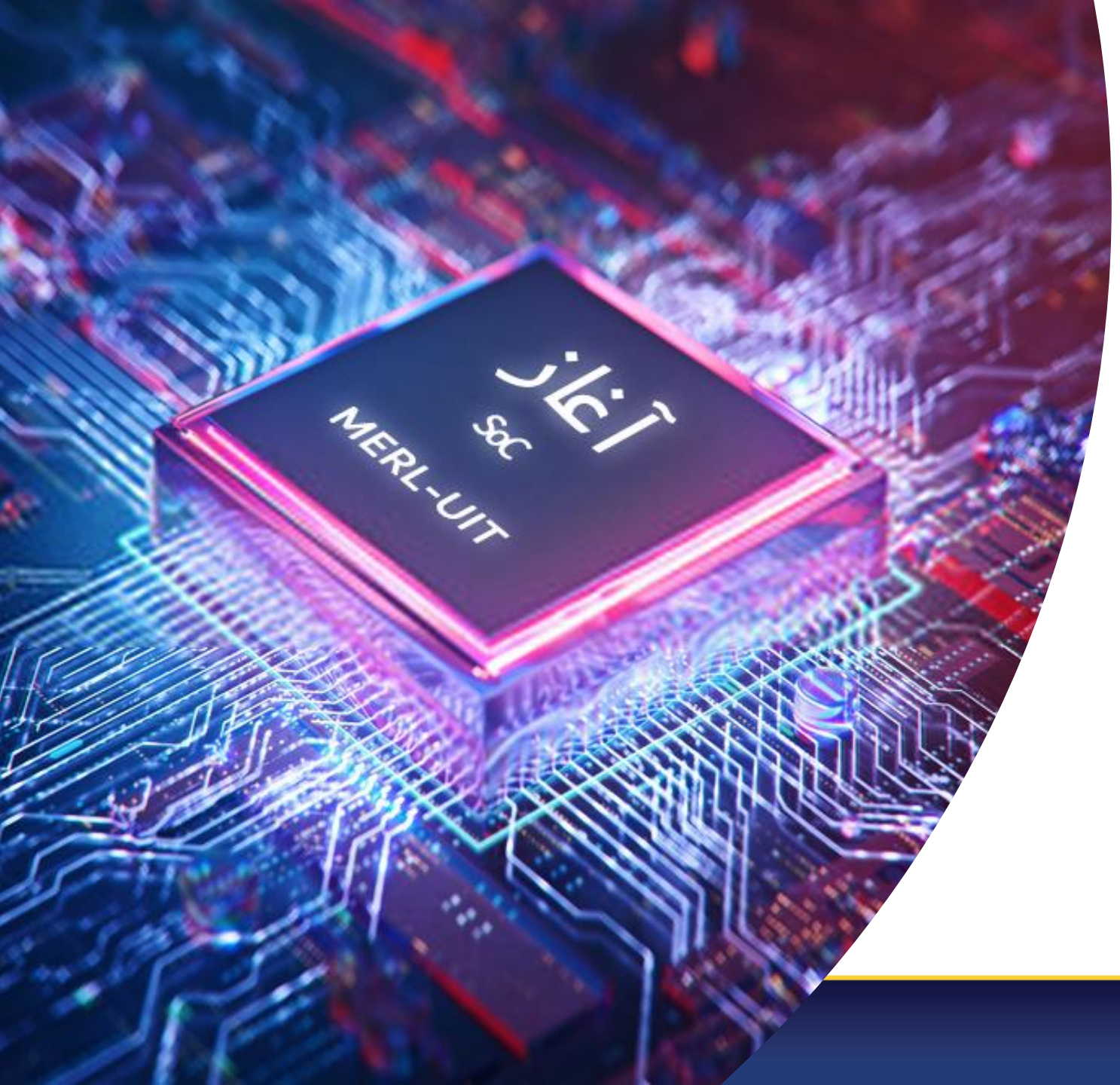
Using Reverse Engineered MASS Document, which includes in depth knowledge of all Rocket-Chip modules.

Rocket-Core MASS Document

- This MASS Document is invaluable for those who have fundamental knowledge of Scala, Chisel and SoC hardware

- This MASS document offers in-depth knowledge of Rocket-Core building blocks and their chisel implementation

- Using Block Diagrams, Class Diagrams, Flowcharts, and describing all the keywords found in the code, description of modules is achieved



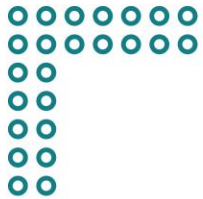
RISC-V® Summit

Aghaaz (آغاز) SoC The Beginning

MERL-UIT PAKISTAN

Specifications of Aghaaz (آغاز) SoC

- RV-32 (32-bit)
- I, M & C extensions
- 64-KB Instruction and Data Cache
- CLINT – Core Local Interrupt Controller
- PLIC – Peripheral Local Interrupt Controller
- Privileged(Machine)



Configuring the Aghaaz SoC and Core

- Create a Aghaaz Core Class with requisite configuration
- Add the Aghaaz Core class in the System configuration

Configuring Aghaaz Core in Rocket-Chip

Path to Default Configuration Scala File

Class named AghazCore, contains the configurations for the core level functionality.
This class is created in Rocket-chip/src/main/scala/subsystem/Config.scala

```
class AghazCore(n: Int, overrideIdOffset: Option[Int] = None) extends Config((site, here, up) => {
  case RocketTilesKey => {
    val prev = up(RocketTilesKey, site)
    val idOffset = overrideIdOffset.getOrElse(prev.size)
    val big = RocketTileParams(
      core   = RocketCoreParams(mulDiv = Some(MulDivParams(
        mulUnroll = 8,
        mulEarlyOut = true,
        divEarlyOut = true)),
        useAtomics= false,
        fpu = None,
        ),
      dcache = Some(DCacheParams(
        rowBits = site(SystemBusKey).beatBits,
        nMSHRs = 0,
        nWays = 16,
        nSets=64,
        blockBytes = site(CacheBlockBytes))),
      icache = Some(ICacheParams(
        rowBits = site(SystemBusKey).beatBits,
        nWays = 4,
        nSets=256,
        blockBytes = site(CacheBlockBytes))))
    List.tabulate(n)(i => big.copy(hartId = i + idOffset)) ++ prev
  }
})
```

Rocket-Core parameters

Data Cache parameters

Instruction Cache parameters

Aghaaz Core Params – Default Configuration

- In the RocketCoreParams, useCompressed is true, means C extension is enabled by default

useCompressed	true	useCompressed is for C-extension, it is true by default, means our Core will have C-extension enabled implicitly.

- Extension I is the base extension and is supported by default as well.

Configuring RV32M Core

RocketCoreParams has the parameters for configuring our Core.

We added MulDiv(Multiplier) with MulDivParms, having default values, in the RocketCoreParams, in order to enable M extension in our Core.

```
core   = RocketCoreParams(mulDiv = Some(MulDivParams(
  mulUnroll = 8,
  mulEarlyOut = true,
  divEarlyOut = true)),
  useAtomics= false,
  fpu = None,
  ),
```

Used to estimate of no of cycles per multiplication

mulEarlyOut will make the multiplier's latency data-dependent, i.e., it will take fewer cycles to multiply small numbers than large ones.

We updated fpu (Floating-Point unit) to None, to disable F and D extensions in our core.

We updated the value of useAtomics to false, to disable A extension.

divEarlyOut is analogous to mulEarlyOut (smaller numbers divide more quickly than larger ones).

MERL-UIT PAKISTAN

Configuring the *icache* and *dcache* in Aghaaz Core

```
dcache = Some(DCacheParams(
  rowBits = site(SystemBusKey).beatBits,
  nMSHRs = 0,
  nWays = 16,
  nSets=64,
  blockBytes = site(CacheBlockBytes))),
icache = Some(ICacheParams(
  rowBits = site(SystemBusKey).beatBits,
  nWays = 4,
  nSets=256,
  blockBytes = site(CacheBlockBytes))))
```

DCacheParams : sets the Data Cache parameters

Parameters	Value	Description
nMSHRs	0	There are no MSHRs (Miss Status Handling Registers)
nWays	16	There are 16 Ways of Cache in DCache
nSets	64	There are 64 Sets of Cache in DCache
blockBytes	64	There are 64 blockBytes in DCache

Size of DCache = *nWays x nSets x blockBytes : 16 x 64 x 64 = 64KB)*

ICacheParams : sets the Instruction Cache parameters

Parameters	Value	Description
nWays	4	There are 4 Ways of Cache in ICache
nSets	256	There are 256 Sets of Cache in ICache
blockBytes	64	There are 64 blockBytes in ICache

Size of ICache =*:4 x 256 x 64 =64KB*

RISC-V® Summit

Blocks of Core

Aghaaz Core provides extensions for I, M and C with 32bit compatibility

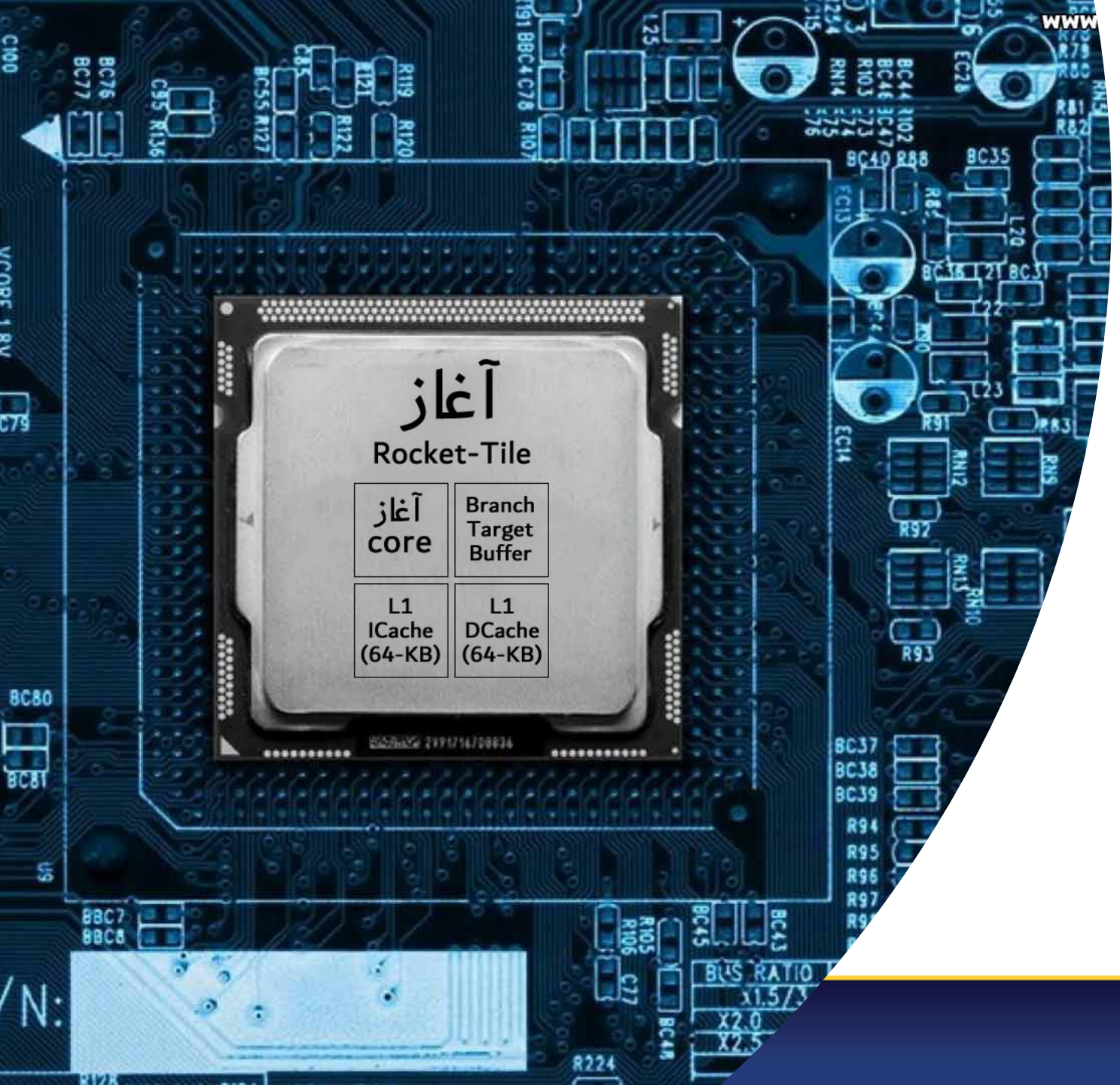
MERL-UIT PAKISTAN

The List of Parameters of RocketCore, their description, and the **Core Module** that is controlled by the parameter are as follows:

Parameter	Description	Module
useVM	Use Virtual Memory you can enable or disable	• ICache • HellaCache
useUser	Use user mode Known as U-mode	RocketCore
useSupervisor	Use Supervision mode Known as S-mode	RocketCore
useDebug	Use debug is true by default means using debug mode	RocketCore
useAtomics	Use atomics is for A extension	RocketCore
useAtomicsOnlyForIO	For Usage of A Extension in Input and Outputs only	RocketCore
useCompressed	It is for C extension.	RocketCore
useRVE	Use E base ISA	CSR
useSCIE	Support custom instructions.	RocletCore
nLocalInterrupts	No. of local interrupts	CSR
nBreakpoints	No. of breakpoints. It is 1 by default	CSR
useBPWatch	Use breakpoint watch. It is false by default	Breakpoint
mcontextwidth	Machine context width	• Breakpoint • CSR
scontextwidth	Supervisor context width	• Breakpoint • CSR
nPMPs	It is a integer value which is 8 by default	• PTW • CSR
nPerfCounters	No. of perf counter 0 by default	CSR

How did we know, which parameter is doing what?

The MASS Document describes the full set of parameters, their description and modules that are managed by that parameter



RISC-V® Summit

Blocks of Tile

Rocket-Tile of Aghaaz SoC contains Aghaaz Core, Branch Target Buffer, L1 Icache & DCache of 64-KB.

MERL-UIT PAKISTAN

Initializing Aghaaz Core to Default Configuration

Path to Default Configuration Scala File

Default Configuration is created in Rocket-chip/src/main/scala/system/Config.scala

```
class oldDefaultConfig extends Config(new AghazCore(1) ++ new WithCoherentBusTopology ++ newBaseConfig)
```

- *oldDefaultConfig* is inherited by the class Config.
- Class Config (Parent Class) includes the key configurations, from which all configurations must be inherited.
- *AghaazCore* object is generated as a parameter with ’1’ indicating that there is only one(1) core in our *AghaazCore*.
- *AghaazCore* object is further added to the *WithCoherentBusTopology* object, which includes configurations for the Bus level.
- These objects are further added by the *newBaseConfig* object, which contains the base configurations.

Customizing AghazCore to 32-bit Configuration

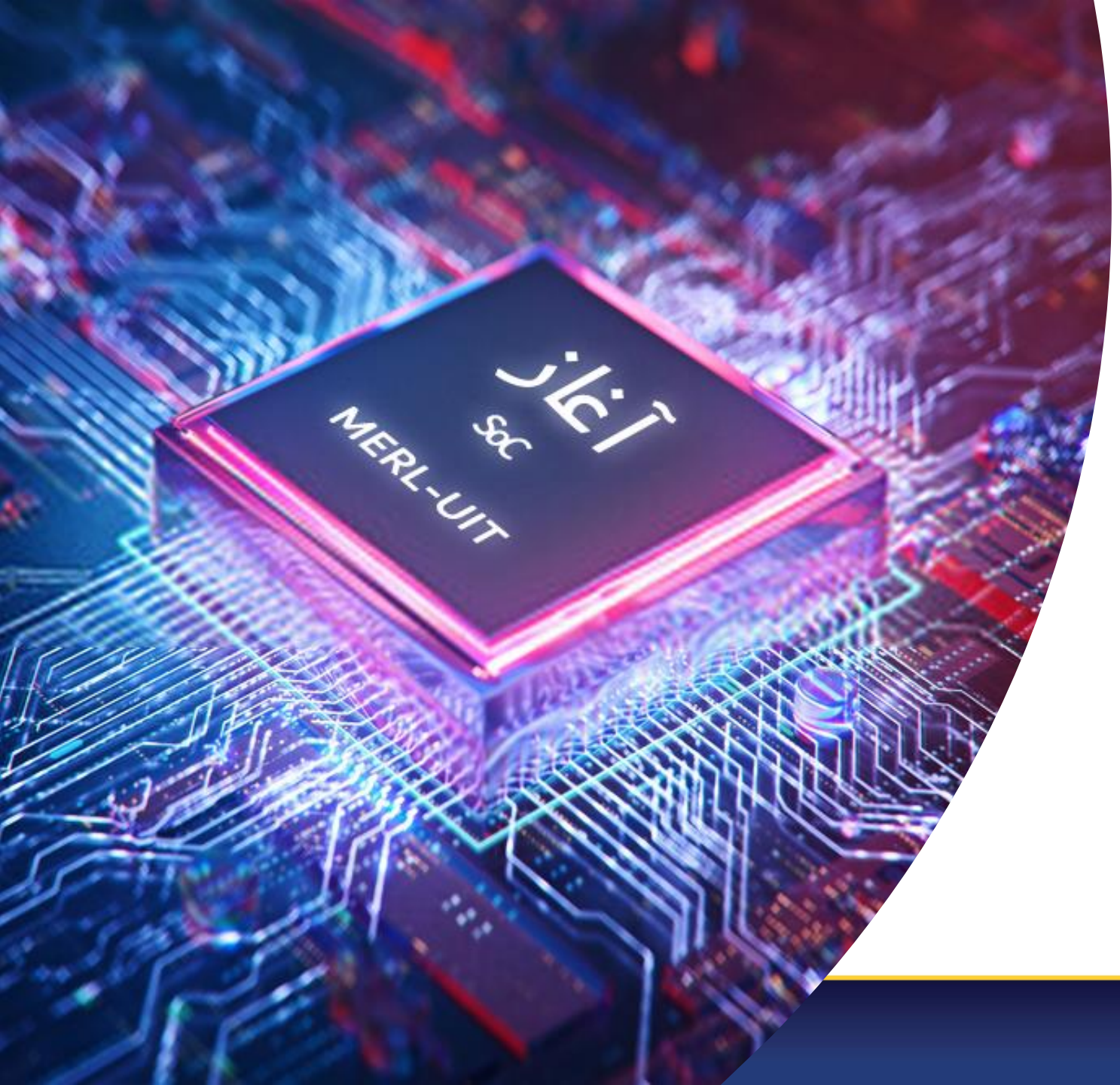
```
class WithRV32 extends Config((site, here, up) => {
  case XLen => 32
  case RocketTilesKey => up(RocketTilesKey, site) map { r =>
    r.copy(core = r.core.copy(
      fpu = None,
      mulDiv = Some(MulDivParams(mulUnroll = 8))))
  }
})
```

```
class DefaultConfig extends Config(new WithRV32 ++ new oldDefaultConfig)
```

RISC-V® Summit

Blocks of SoC

MERL-UIT PAKISTAN



RISC-V® Summit

At this point our Aghaaz SoC is successfully configured and ready to be built

MERL-UIT PAKISTAN

Summary and Future Direction

- Presented an overview of the methodology used to Reverse engineer the Rocket Chip
- Used the prescribed methodology to develop a Micro-Architecture and Software Specification Document (MASS) with more than 400 pages and growing
- Used the MASS document to configure the Aghaaz SoC
- Continue the Reverse Engineering Effort to cover diplomacy
- Enhance the capabilities of Rocket Chip SoC framework to include I/O peripherals and the ability to built a full SoC ready for **CHISEL-to-GDS**
- Work on improving the software architecture of the Rocket Chip

RISC-V® Summit

Acknowledgements

This Research was made possible with the counselling, guidance and support of Dr.Roomi Naqvi, Dr Ali Ahmed Ansari, and Farhan Ahmed Karim. Without their encouragement and guidance, this could not be possible.

This presentation was created under the Lead of Uzair Khan, along with the team of collaborators/ interns under him that gave their level best under the supervision of such experienced mentors.

Team Members
Waleed Waseem
Shahzaib Kashif
Talha Ahmed
Mohsin Raza
Muhammad Shahzaib
Syeda Fizza Jaffery
Almas Ibrahim

MERL-UIT PAKISTAN

MERL-UIT Rocket-Chip Team

RISC-V® Summit

December 8-10 | Virtual Event

MERL-UIT PAKISTAN

Thank you for joining us.

Contribute to the RISC-V conversation on social!

#RISCVSUMMIT @risc_v

Brought to you by informa tech

riscvsummit.com #RISCVSUMMIT