https://www.pulserain.com
858-877-3485 858-408-9550

Doc# TRM-0963-00001, Rev 1.0.0

PulseRain FRV2000 RISC-V Microcontroller

Technical Reference Manual

Aug, 2020

This page is intentionally left blank.

Table of Contents

References

1. The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 20191213, Editors Andrew Waterman and Krste Asanovi´c, RISC-V Foundation, December 2019.
2. The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Document Version 20190608-Priv-MSU-Ratified, Editors Andrew Waterman and Krste Asanovi´c, RISC-V Foundation, June 2019
3. RISC-V External Debug Support, Version 0.13.2, Editors: Tim Newsome, Megan Wachs, SiFive, Inc., Fri Mar 22, 2019
4. David Patterson and Andrew Waterman, The RISC-V Reader: An Open Architecture Atlas
5. Andrew Waterman: Design of the RISC-V Instruction Set Architecture
6. Building Embedded Systems – Programmable Hardware, Changyi Gu, APress Media, July 2016 http://www.apress.com/us/book/9781484219188
7. Computer Organization and Design - The Hardware / Software Interface (3rd edition), David A. Patterson and John L. Hennessy, Morgan Kaufmann Publication, 2005
8. Wishbone B4, WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores, OpenCores Organization, 2010
9. C++ preprocessor __VAR_ARGS__ number of arguments, https://stackoverflow.com/questions/2124339/c-preprocessor-va-args-number-of-arguments
10. C Pre-Processor Magic, by Jonathan Heathcote, http://jhnet.co.uk/articles/cpp_magic
11. Future Technology Devices International Ltd FT2232H Dual High Speed USB to Multipurpose UART / FIFO IC Datasheet, Version 2.6
12. ADXL345 3-Axis Digital Accelerometer, Analog Devices, Inc. 2015

Acronyms and Abbreviations

Acronyms / Abbreviations	Definition
ABI	Application Binary Interface
ADC	Analog to Digital Converter
API	Application Program Interface
BCD	Binary-Coded Decimal
CISC	Complex Instruction Set Computer
CODEC	Coder-Decoder
DPTR	Data Pointer
FARM	FPGA + Arduino + RISC-V + Make
FPGA	Field Programmable Gate Array
I2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
IO	Input and Output
IRQ	Interrupt Request Line
ISA	Instruction Set Architecture
ISR	Interrupt Service Routine
JTAG	Joint Test Action Group
LSB	Least Significant Bit
MCU	Microcontroller Unit
MSB	Most Significant Bit
NOP	No Operation
OCD	On-chip Debugger
OS	Operating System
PC	Personal Computer or Program Counter
PSW	Program Status Word
PWM	Pulse Width Modulation
RISC	Reduced Instruction Set Computer
RISC-V	Berkeley Fifth Generation RISC Instruction Set
SFR	Special Function Register
SPI	Serial Peripheral Interface
SRAM	Static Random-Access Memory
UART	Universal Asynchronous Receiver-Transmitter
Wi-Fi	Wireless Fidelity

1 Introduction

1.1 RISC-V Soft MCU for FPGA

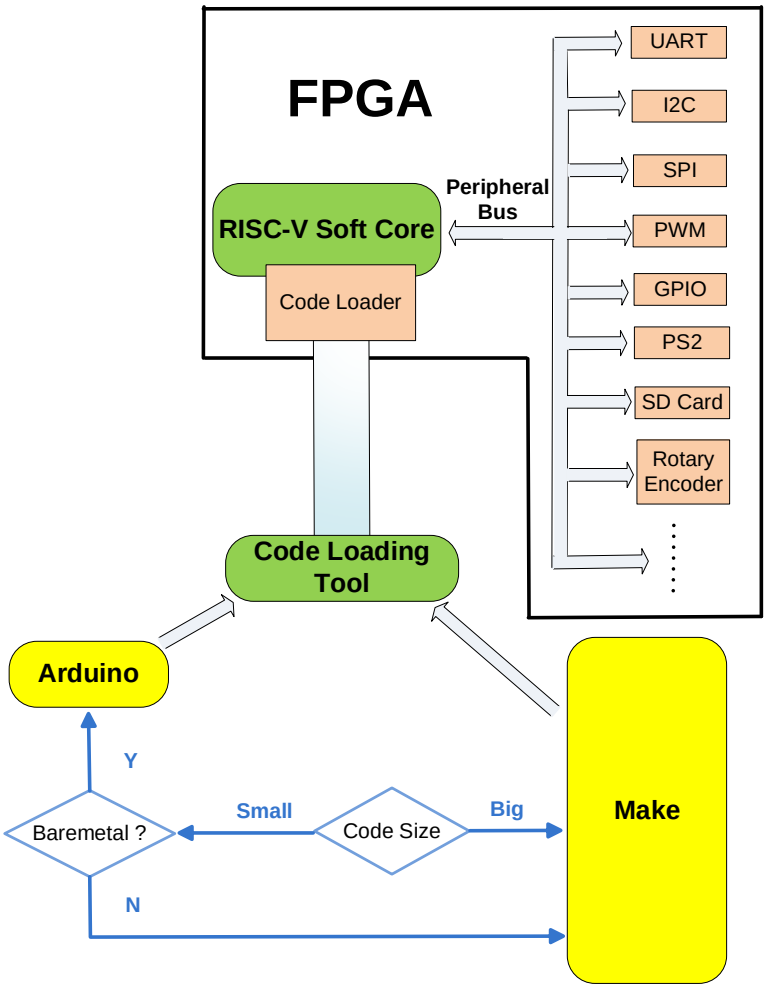

Figure 1-1 FARM (FPGA + Arduino + RISC-V + Make)

Putting MCU core into FPGA is becoming a universal practice these days. However, most MCU soft cores available today are tied to specific FPGA vendors, with close source implementation. And there will be oodles of hoops for engineers to jump through if they decide to migrate from one proprietary core to another.

However, the rise of RISC-V has brought new possibilities to the fore. RISC-V is an open instruction set that allows designers to produce new processors without the constraint of patents, license fee or royalties. Leveraging the power of RISC-V, PulseRain Technology has come up with the **award winning** FRV2000 Serial RISC-V soft core for more portable FPGA solutions. And in this regard, PulseRain Technology is proud of its contribution to the open source community.

1.2 FARM (FPGA + Arduino + RISC-V + Make)

To push the envelope, PulseRain Technology has also proposed a new design approach for quick development, called FARM (FARM + Arduino + RISC-V + Make). As shown in Figure 1-1, with the FARM design approach, FPGA becomes the center piece of the whole system. The FPGA contains a RISC-V Soft core CPU and a Hardware Based Code Loader. And it also contains most of the peripheral implementations. For those peripherals that cannot be implemented in FPGA (such as sensors), the FPGA can still implement most of their control functions or data read/write.

And there are mainly two ways to write software for the RISC-V core in Figure 1-1. For bare-metal design, the software developers can write and upload code directly through Arduino IDE, and fully utilize the abundance of peripheral libraries afforded by Arduino. On the other hand, if the system grows into a large scale one, the design can use Make tools for better code organizing and architecture. And the code upload can be achieved through the tool provided by PulseRain Technology.

1.3 Design Flow

Under the FARM design flow, the hardware engineer (PCB designer), the logic engineer (RTL designer) and the software engineer can work together in the way shown in Figure 1-2.

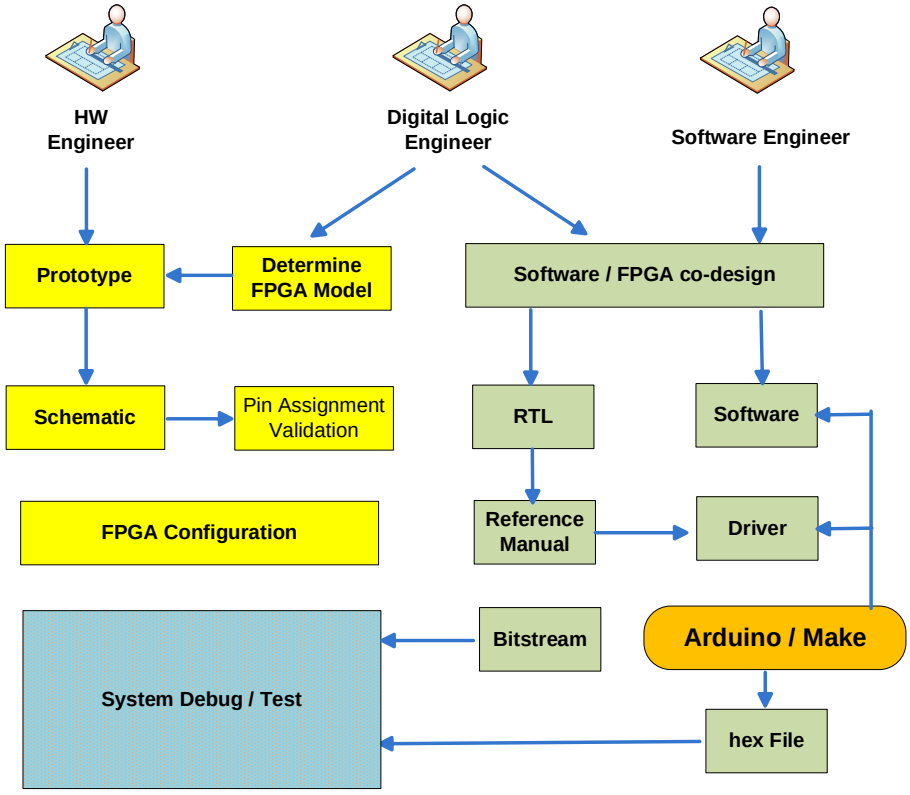

Figure 1-2 FARM Design Flow

1.4 GitHub Repositories

And all the code can be found on PulseRain Technology's public repositories: http://git.pulserain.com

In particular, the following repositories will be the focus of this document:

- https://github.com/PulseRain/PulseRain_RISCV_MCU
  Repository for PulseRain Technology's RISC-V MCU.

  There are multiple branches in this repository. And each branch corresponds to a specific hardware platform. For each variation of MCU, it usually contains two submodules:
  1. RISC-V Soft MCU core, such as FRV2100
  2. Hardware Based Bootloader
  3. Peripherals (Usually from PulseRain RTL Lib)

- https://github.com/PulseRain/Hardware_Bootloader
  Repository for Hardware Based Bootloader (code loader), as illustrated in Figure 1-1.

- https://github.com/PulseRain/FRV2100
  Repository for PulseRain FRV2100 RISC-V soft core. (code name: Reindeer)

- https://github.com/PulseRain/PulseRain_rtl_lib
  Repository for PulseRain Peripherals, such as UART, I2C etc.

- https://github.com/PulseRain/Arduino_RISCV_IDE
  Repository for Arduino RISC-V Board Support Package

- https://github.com/PulseRain/Reindeer_MachXO3D
  Repository for Lattice MachXO3D Breakout Board.
  The PulseRain FRV2100 RISC-V core has been ported to this platform.

2 Hardware

2.1 Platform/MCU/Processor Core Architecture

The beauty of soft core MCU is that the MCU can be easily customized to fit various hardware platforms. Generally speaking, the processor core inside the MCU will have a limited choice from the available processor cores, while the peripherals have to match whatever is available on the hardware platform.

In this regard, the PulseRain RISC-V MCU will have a 3-layer architecture (Figure 2-1), as

Platform -> MCU -> Processor Core

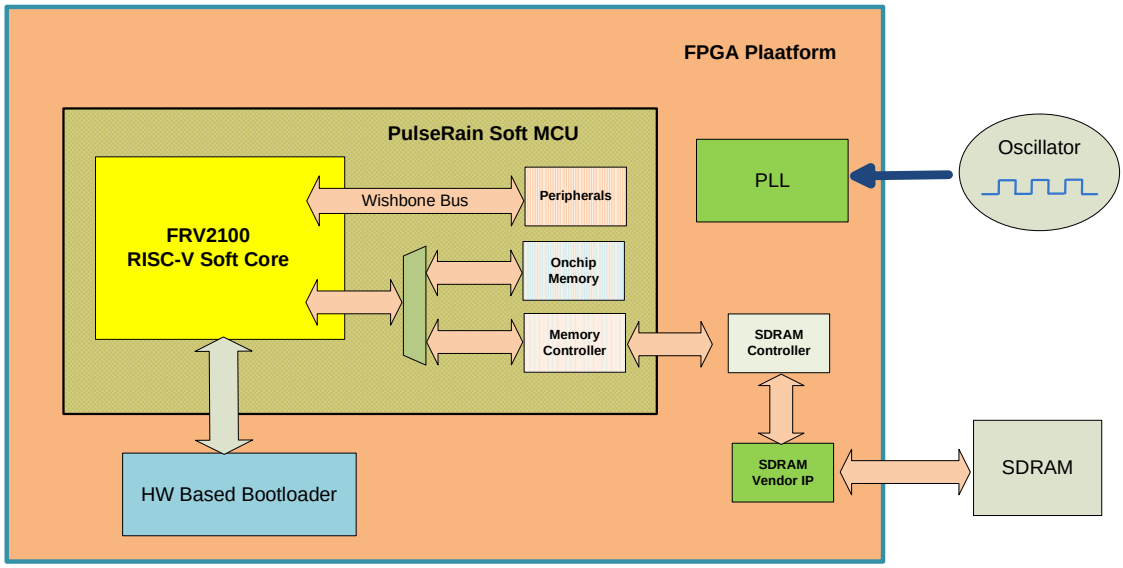

Figure 2-1 Platform – MCU – Processor Core

Each layer will be mapped to a separate code repository.

- Under this architecture, the platform will store all the hardware specific code and IPs, such as the top-level wrapper, PLL etc. It will also have the project and constraints to generate bitstreams for the specific FPGA. And the MCU will be pulled in as submodules at the platform level. Optionally, the platform could also pull in Hardware Based Bootloader (code loader) as another submodule.

- And the MCU will basically pull in two submodules:
  1. peripherals, most likely it will be included in PulseRain_rtl_lib
  2. PulseRain RISC-V soft core processor

  The peripherals are also setup/configured at this level.

- At processor core level, it can be PulseRain FRV2100 (Reindeer) or FRV2000 (Rattlesnake) at this point.

2.1.1 Platform Folder Structure

A typical folder structure for Platform is illustrated in Figure 2-2. Please note the folders that are marked as red color are external links (submodules) to other repositories.

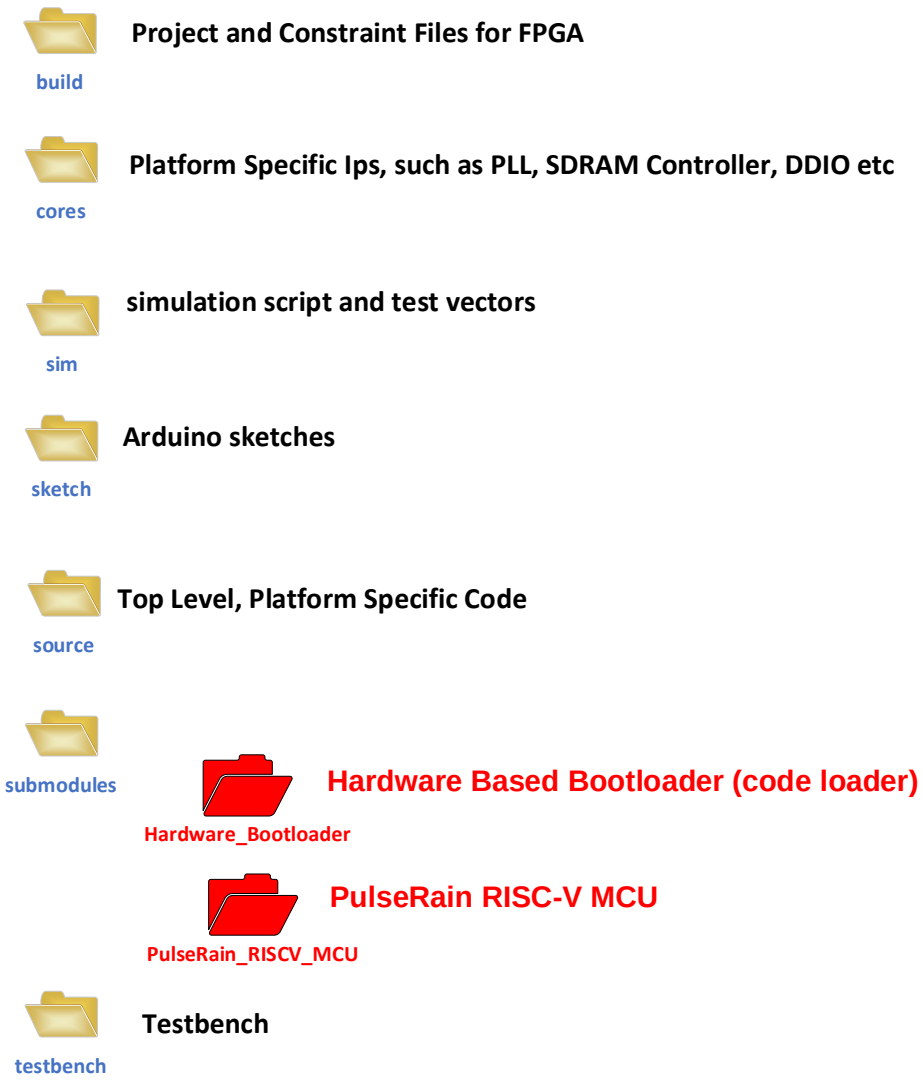

Figure 2-2 Platform Folder Structure

2.1.2 MCU Folder Structure

A typical folder structure for PulseRain RISC-V MCU is illustrated in Figure 2-3. It usually contains two submodules (marked as red color in Figure 2-3): peripherals and processor core.

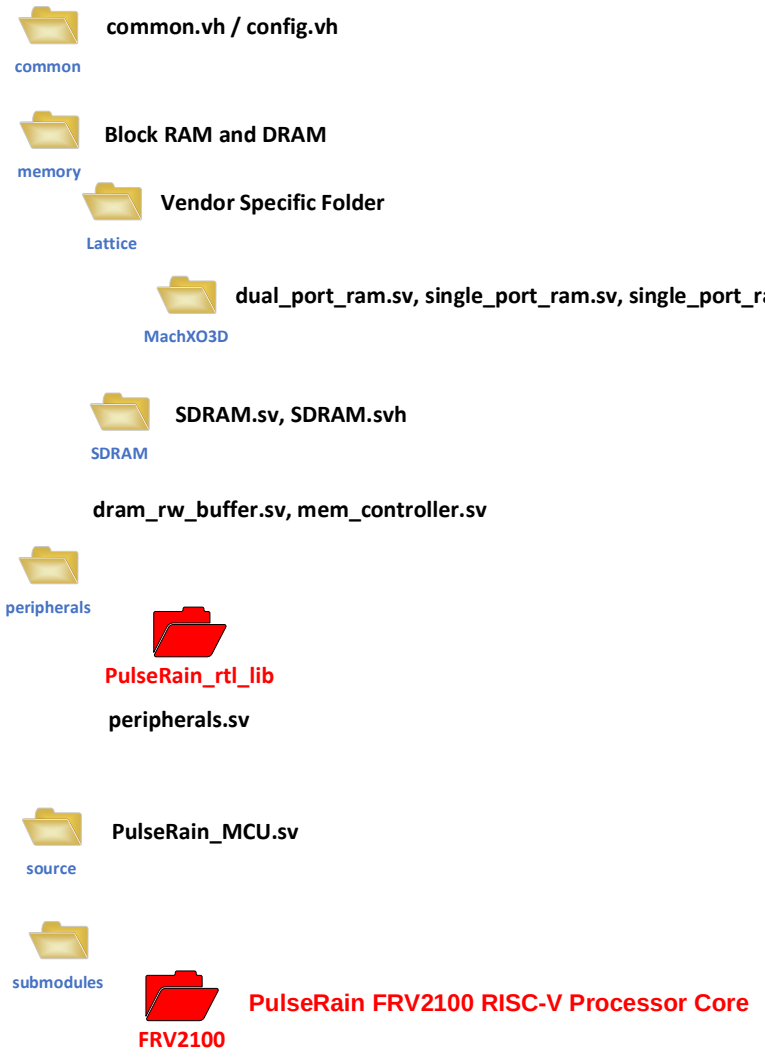

Figure 2-3 PulseRain RISC-V MCU Folder Structure

And to port PulseRain RISC-V MCU to various FPGA devices, the following files need to be customized:

1. config.vh in the **common** folder
   The config.vh defines the following parameters for a specific MCU variation. It should be customized to match the resources available to the FPGA:

Parameter	Definition
SRAM_SIZE_IN_BYTES	Size for FPGA block RAM in number of bytes
DRAM_SIZE_IN_BYTES	Size of DRAM (in bytes) external to FPGA
MCU_MAIN_CLK_RATE	The clock rate of the MCU, in number of Hz
…	Memory Mapped Address for various peripherals.

Table 2-1 Parameters to Customize the MCU

2. dual_port_ram.sv, single_port_ram.sv, single_port_ram_8bit.sv
   Those files are usually stored in a Vendor/Device specific sub-folder in the memory directory. Those files are used for implement dual port block-ram or single port block-ram. And in this regard, each FPGA vendor has its own ways.

   Generally, most FPGA vendors will provide a code template to infer those memories through the synthesis tools. And they would also provide a primitive or parameterized component to achieve the same. And the files listed above should be customized to realize those block ram per FPGA device.

3. peripherals.sv
   For each variation of the MCU, it usually has its own flavor of peripheral set. And the addresses of those peripherals are defined in config.vh. But those peripherals are put together and instantiated in peripherals.sv. And the implementation of the peripherals can usually be found in PulseRain_rtl_lib.

4. processor core
   The processor core is pulled into the MCU as a submodule, as illustrated in Figure 2-3. At this point, the choice for process core can be either FRV2100 (Reindeer) or FRV2200 (Rattlesnake), as shown in Table 2-3.

2.1.3 Processor Core Folder Structure

A typical folder structure for PulseRain RISC-V Soft Core Processor is illustrated in Figure 2-4. The details of PulseRain RISC-V Soft Core Processor will be explored in a later part of this document.

And the file names listed in Figure 2-4 can be categorized in Table 2-2

File Name	Description
PulseRain_Reindeer_core.sv	Top Level wrapper for the processor core
Reindeer_CSR.sv	Control and Status Register
Reindeer_reg_file.sv	32 bit General Purpose Registers
Reindeer_machine_timer.sv	machine timer defined in Ref [1]
Reindeer_memory.sv	memory wrapper for both SRAM and DRAM, including memory mapped registers
Reindeer_mm_reg.sv	memory mapped register
Reindeer_fetch_instruction.sv Reindeer_execution_unit.sv Reindeer_instruction_decode.sv Reindeer_data_access.sv Reindeer_controller.sv	2 x 2 pipeline, and the pipeline controller

Table 2-2 Files for PulseRain RISC-V Soft Core Processor

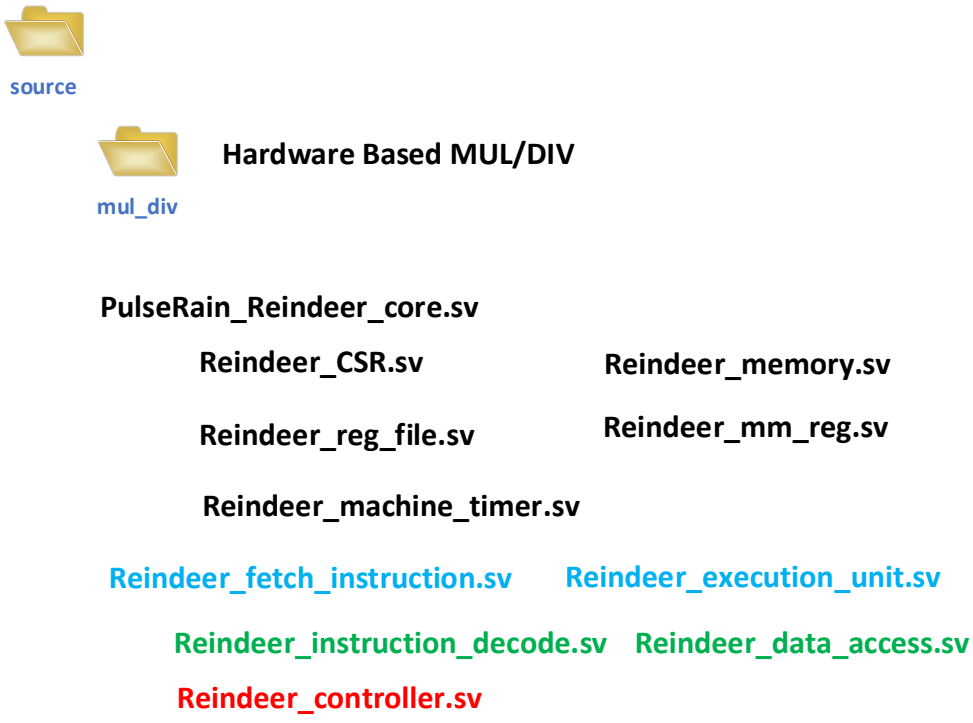

Figure 2-4 Folder Structure for PulseRain RISC-V Soft Core Processor

2.2 Hardware Based Bootloader

Traditionally, bootloaders are implemented in software (Figure 2-5), where the bootloader is the first piece of software to run. And most Arduino official boards, such as Arduino UNO Rev 3, take this approach.

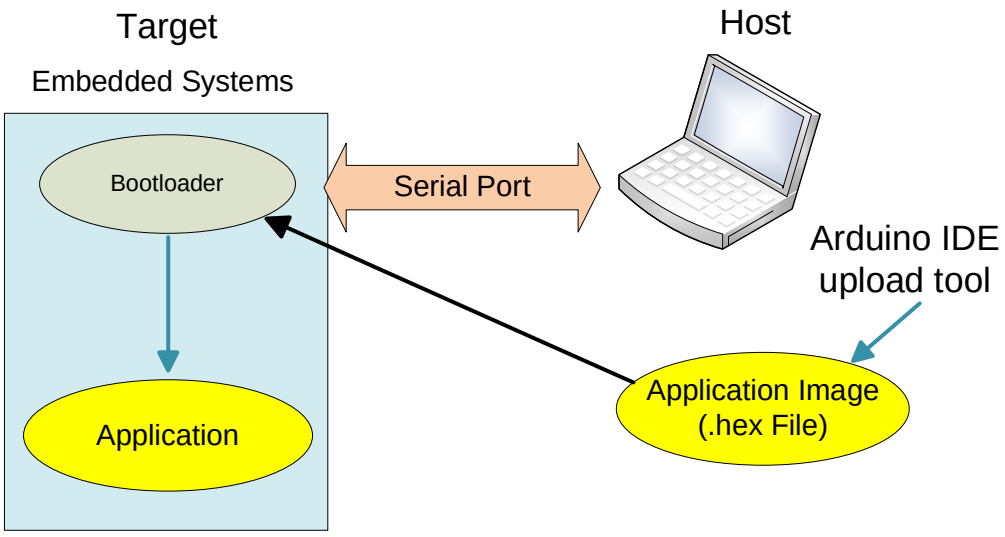

Figure 2-5 Traditional Software Based Bootloader

However, the drawback of this approach is that the designer has to find a way to put the bootloader into the non-volatile memory. Due to the wide variety of FPGA based platforms, sometimes it is just infeasible, if not possible, to do so.

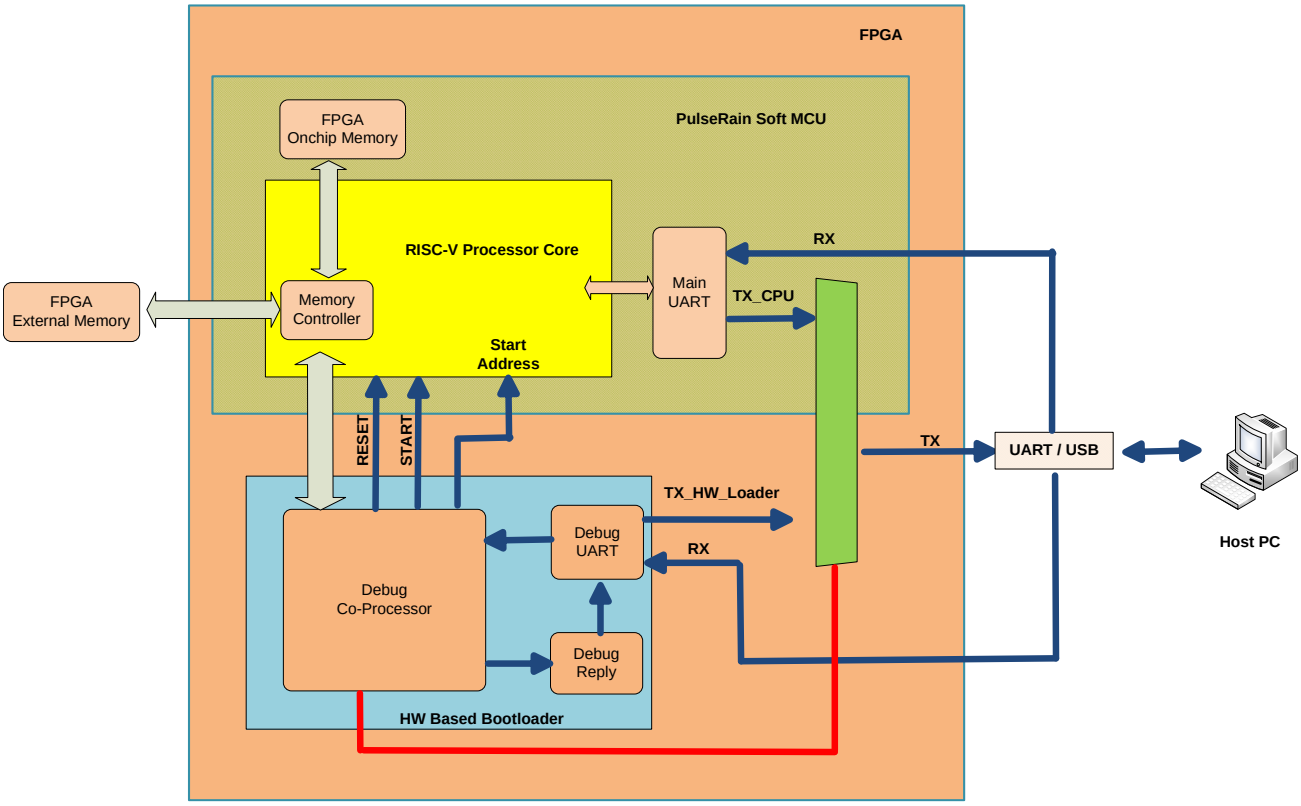

Figure 2-6 Hardware Based Bootloader

And the hardware based (RTL based) solution takes the problem from a different vantage point, as illustrated in Figure 2-6.

With this approach, the bootloader and the processor core share the same UART, and it will also coordinate with the memory controller to load code into the FPGA block-ram (on-chip) or DRAM (off-chip). Unlike the traditional software bootloader, the hardware-based bootloader does not need non-volatile memory to store code, and it can also be used to reset/restart the processor core, and to provide the initial address for the Programmer Counter inside the processor core. It is more reliable, flexible and portable than the traditional approach.

Accordingly, the Arduino IDE will run a python script on the host side, as a tool to upload binary images (Figure 2-17). This python script can be executed independent of Arduino IDE. For .elf files, this python script will invoke GNU Tool Chain to parse the sections inside .elf file, and pass them to the hardware-based bootloader.

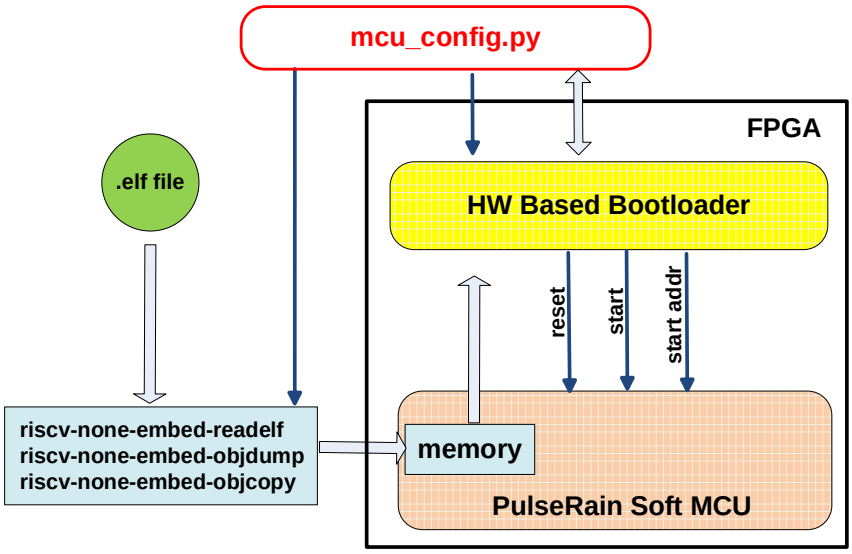

Figure 2-7 Host Tool to Upload Image

2.3 RISC-V Soft Core Processor

2.3.1 Award Winning PulseRain RISC-V Processor Core

PulseRain Technology's FRV2000 Serial Soft-Core Processor has won industrial awards on more than one occasions. In 2018, the PulseRain Reindeer (FRV2100) processor was the 3rd place winner in RISC-V Foundation's Soft CPU Contest. In 2019, the PulseRain Rattlesnake (FRV2200) managed to win the 1st place in the same contest. The FRV2100 supports RV32I[M] instruction set, while the FRV2200 supports 16-bit compressed instruction extension additionally. And optionally, the FRV2200 can be hardened for IoT security. Table 2-3 has the details of those two processors.

- 2018 RISC-V Soft CPU Contest:
  PulseRain Reindeer 3rd Place
- 2019 RISC-V Soft CPU Contest:
  PulseRain Rattlesnake ***1st Place***

Model Number	Code Name	Instruction Set	Security Hardened
FRV2100	Reindeer	RISC-V RV32I (Optional HW-based MUL/DIV)	No
FRV2200	Rattlesnake	RISC-V RV32IMC (HW-based MUL/DIV, 16 bit Compressed Instruction Extension)	Optional

Table 2-3 PulseRain RISC-V Soft Core Processor

2.3.2 Address Space

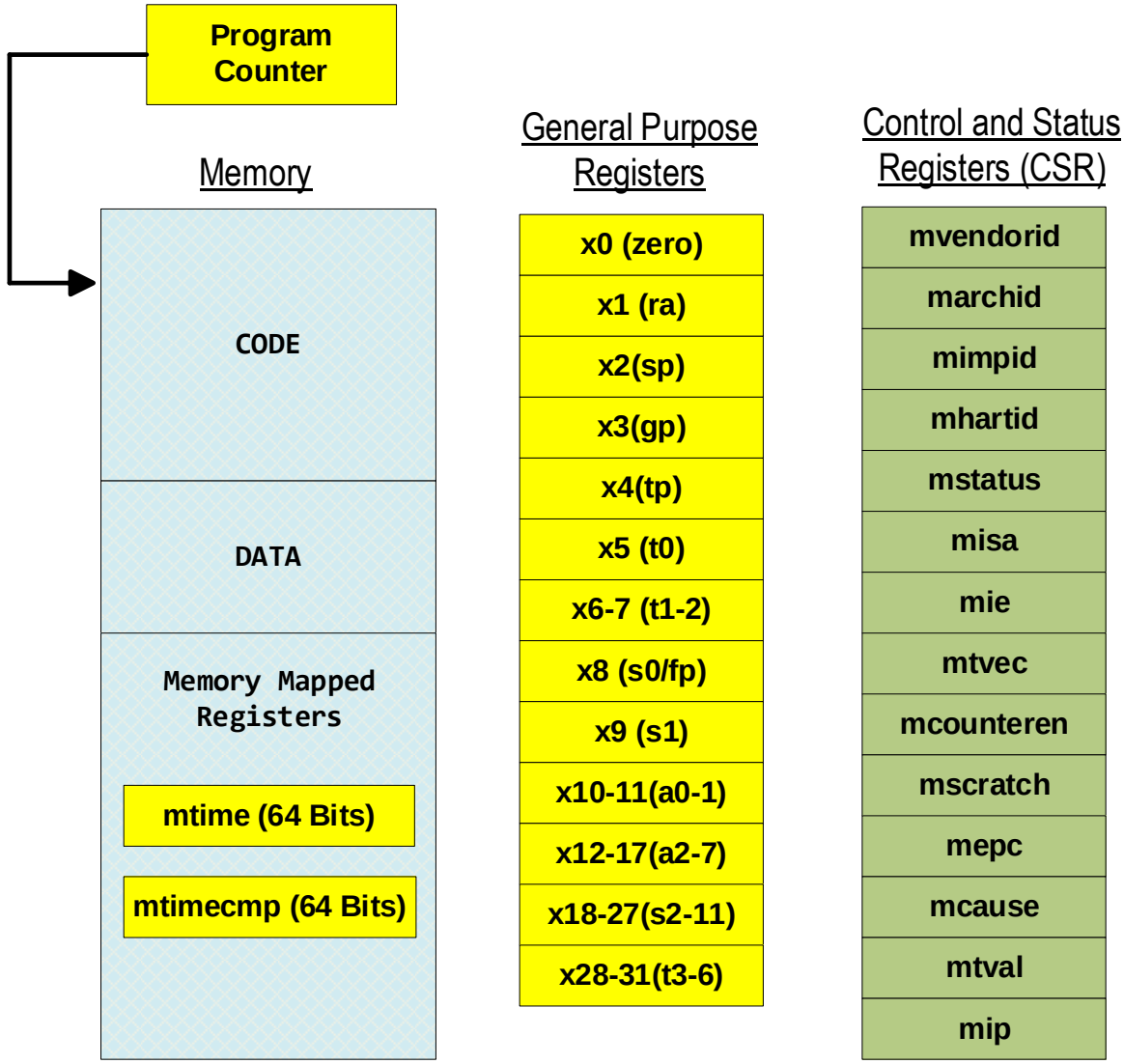

Figure 2-8 RISC-V Address Spaces

Per RISC-V Instruction Set Manual (Ref [1][2]), PulseRain FRV2000 processor has 3 separate address spaces, as shown in Figure 2-8. The 3 address spaces are:

1. Address Space for Memory
   The memory space can be allocated to code, memory or can be used for memory mapped registers.

   Physically, the code and data in FRV2000 share the same storage. In other words, the FRV2000 has a Von Neumann architecture. And the processor records the current instruction address through a Program Counter (PC).

As for the memory mapped registers, they are mainly used for peripheral control and status. In fact, the mtime (machine mode timer) and mtimecmp (machine mode timer compare) are also mapped into the memory space. And those registers will be discussed in a latter part of this document.

2. Address Space for General Purpose Registers
   Per RISC-V RV32I instruction set (Ref [1]), PulseRain FRV2000 has 32 general purpose registers. Each register is 32-bit wide.

3. Address Space for Control and Status Registers (CSR)
   Per RISC-V Instruction Set Manual Volume II (Ref [2]), PulseRain FRV2000 supports a handful of CSRs, as illustrated in Figure 2-8. The details of those CSRs will be discussed in a latter part of this document.

2.3.3 General Purpose Registers and Calling Convention

As mentioned in previous sections, FRV2000 supports 32 general purpose registers. They are marked as x0-x31. Among them, x0 (register zero) is a read-only register, and it always returns zero.

One of the design goals for RISC-V instruction set is to provide support for high level programming languages, such as C or C++, and to keep compatibility at ABI level. Thus, RISC-V has also standardized the calling convention. For assembly language, the name of the 32 registers are given new names to reflect their functions in the calling convention, as shown in Table 2-4.

Register Name	Assembly Name	Description	Value Unchanged after return from the function call
x0	*zero*	*Read Only, always zero*	*Undefined*
x1	*ra*	*Return Address*	*No*
x2	*sp*	*Stack Pointer*	*Yes*
x3	*gp*	*Global Pointer*	*Undefined*
x4	*tp*	*Thread Pointer*	*Undefined*
x5	*t0*	*temp register, or be used as* ***alternate link register*** *(discussed in a later part of this document)*	*No*
x6~x7	*t1~t2*	*Temp Registers*	*No*

Register Name	Assembly Name	Description	Value Unchanged after return from the function call
x8	*s0/fp*	*Saved by Callee Function, or be used as stack frame pointer*	*Yes*
x9	*s1*	*Saved by Callee Function*	*Yes*
x10~x11	*a0~a1*	*Function Parameters or Function Return Value*	*No*
x12~x17	*a2~a7*	*Function Parameters*	*No*
x18~x27	*s2~s11*	*Saved by Called Function*	*Yes*
x28~x31	*t3~t6*	*Temp Register*	*No*

Table 2-4 RISC-V Calling Convention

2.3.4 Privilege Level and Machine Mode

The RISC-V Instruction Set Manual Volume II (Ref [2]) has defined 3 working mode, as Machine Mode Supervisor Mode and User Mode. And each mode is mapped to a particular privilege level, as illustrated in Figure 2-9.

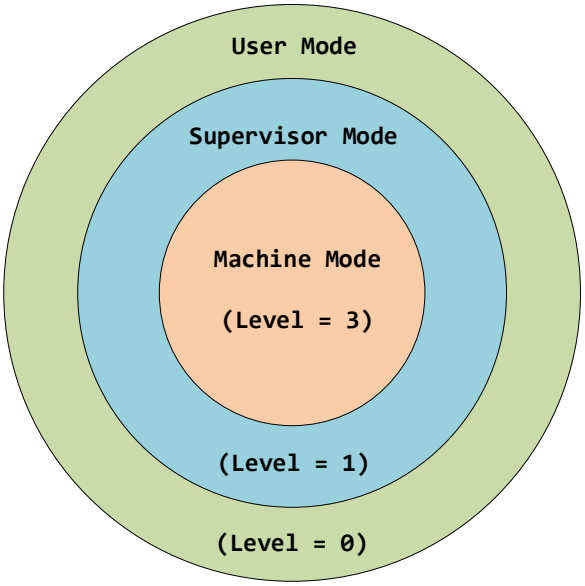

Figure 2-9 RISC-V Privilege Levels

And the Machine Mode is mandatory in Ref [2], while the Supervisor Mode and User Mode are optional. For PulseRain FRV2000 Processor, only the Machine Mode is supported.

2.3.5 Control and Status Register (CSR)

As mentioned early, the RISC-V Instruction Set Manual Volume II (Ref [2]) has defined a separate address space for Control and Status Registers (CSR).

And 12 bits of address are used for CSR indexing. Among the 12 bits, [11 : 10] are used for read/write permission. If they are both high (2'b11), it means the correspondent CSR is a read-only register. Otherwise, the CSR can be written to.

Address [9 : 8] correspond to the privilege level associated with this CSR. As PulseRain FRV2000 processor only supports Level 3 (Machine Mode), the rest of this document will only discuss those CSRs listed in Table 2-5.

Address	Read / Write Permission	Privilege Level	Register Name	Description
0xF11	*Read Only*	*3*	*mvendorid*	*Vendor ID*
0xF12	*Read Only*	*3*	*marchid*	*Architecture ID*
0xF13	*Read Only*	*3*	*mimpid*	*Implementation ID*
0xF14	*Read Only*	*3*	*mhartid*	*HART ID*
0x300	*R/W*	*3*	*mstatus*	*Machine Status*
0x301	*R/W*	*3*	*misa*	*ISA and Extensions Supported*
0x304	*R/W*	*3*	*mie*	*Interrupt Enable*
0x305	*R/W*	*3*	*mtvec*	*Base address for Exception Vectors*
0x340	*R/W*	*3*	*mscratch*	*Scratch Register*
0x341	*R/W*	*3*	*mepc*	*The PC value when exception happens*
0x342	*R/W*	*3*	*mcause*	*The cause of the Exception*
0x343	*R/W*	*3*	*mtval*	*Machine Trap Value Register*
0x344	*R/W*	*3*	*mip*	*Interrupt Pending*
0xB00	*R/W*	*3*	*mcycle*	*Number of Clock Cycles (lower 32 bits)*
0xB02	*R/W*	*3*	*minstret*	*Number of Instruction Retired (lower 32 bits)*

Address	Read / Write Permission	Privilege Level	Register Name	Description
0xB80	*R/W*	*3*	*mcycleh*	*Number of Clock Cycles (higher 32 bits)*
0xB82	*R/W*	*3*	*minstreth*	*Number of Instruction Retired (higher 32 bits)*

Table 2-5 CSR Registers Supported by PulseRain FRV2000 Processor

2.3.5.1 mvendorid (Vendor ID)

In order to distinguish between vendors, this CSR is defined by RISC-V ISA to save the vendor ID. In fact, it is a 32-bit read-only value derived from JEDEC manufacturer ID. For all the RISC-V processors designed by PulseRain Technology, this register will always be set as 32'h0000055E.

2.3.5.2 marchid (Architecture ID)

For PulseRain FRV2000 Processor Core, this CSR is read-only with the value as 32'h80200007.

2.3.5.3 mimpid (Implementation ID)

This is a 32-bit read-only CSR, defined in Table 2-6.

Bit Index	Comments
[31 : 24]	*always 8'h0A*
[23 : 8]	*Determined at MCU level, Platform Specific*
[7 : 0]	*For FRV2100, it is 8'h64.* *For FRV2200, it is 8'hC2*

Table 2-6 Bit Definition for mimpid

2.3.5.4 mstatus

This CSR is responsible for recording the processor status. Its full definition can be found in Ref [2]. However, for PulseRain FRV2000 Processors, only those bits shown in Table 2-7 are implemented.

Bit Index	Comments
[7]	*MPIE (Machine Previous Interrupt Eanble). When Interrupt or Exception happens. this bit will record the current MIE value. When return through the* ***mret*** *instruction, this value will be copied back to MIE.*

Bit Index	Comments
[3]	*MIE (Machine Interrupt Enable). When this bit is set to zero, all the external interrupt and timer interrupt will be disabled.*

Table 2-7 Bit Definition for mstatus

2.3.5.5 mscratch

This CSR can be used as 32-bit general purpose storage.

2.3.5.6 CSRs related to Interrupt / Exception

The following CSRs are for Interrupt and Exception handling, and they will be discussed with more details in a later part of this document.

- mtvec (Machine Trap Vector Base-Address Register)
- mip (Machine Interrupt Register, Pending Interrupt)
- mie (Machine Interrupt Register, Interrupt Enable)
- mcause (Machine Cause Register)
- mepc (Machine Exception Program Counter)
- mtval (Machine Trap Value Register)

2.3.5.7 Counters

As a way to monitor the hardware performance, RISC-V ISA has defined a few counters to keep track of the time span starting from reset. Those counters are:

- mcycle and mcycleh
  RISC-V ISA has defined a 64-bit cycle register to keep track of the number of clock cycles passed since reset. The lower 32 bits and the higher 32 bits of this register are stored in CSR mcycle and CSR mcycleh respectively. As PulseRain FRV2000 is a 32-bit processor, the 64-bit value can not be read out atomically. Thus, the software has to follow certain rules in order to get this 64-bit value correctly. For PulseRain FRV2000 processor, the software can achieve this through one of the following two ways:
  1. read the mcycle first, immediately followed by a read on mcycleh. Make sure there are no interrupts between those two reads. The processor will automatically save the higher 32 bits of cycle register into mcycleh when mcycle is being read.

  2. Another way to this can be found in Ref [2]. As shown below, the software can use a loop to read between mcycleh and mcycle, and exit the loop if the two mcycleh read return the same value.

```
again:
    rdcycleh x3
    rdcycle  x2
    rdcycleh x4
    bne x3, x4, again
```

- minstret and minstreth
  RISC-V ISA has also defined another 64-bit register to keep track of the number of instructions retired since reset. Its lower 32-bit and higher 32-bit are stored in CSR minstret and CSR minstreth respectively.

  Like CSR mcycle and CSR mcycleh, this 64-bit register can not be read out atomically in a single read. And the same software approach mentioned previously for mcycle/mcycleh can also be used here to read minstret/minstreth.

2.3.6 Timer

To prepare for RTC (Real Time Clock) implementation, the RISC-V ISA has defined two 64-bit registers for this purpose. They are called mtime and mtimecmp. And different from CSR, those two registers are memory-mapped into the memory space, as illustrated in Figure 2-8.

- mtime
  On PulseRain FRV2000 processor, mtime is a 64-bit timer counter runs at 1MHz resolution. And its lower 32-bit part and higher 32-bit part are mapped to the addresses shown in Table 2-8. Like other 64-bit read, the software has to make sure it can get the value correctly through multiple reads. And the solution is already mentioned in Section 2.3.5.7.

- mtimecmp
  As its name suggests, the purpose of mtimecmp is to compare its value against that of mtime's. When the mtime is no less than mtimecmp, a timer interrupt will be generated. And its lower 32-bit part and higher 32-bit part are mapped to the addresses shown in Table 2-8.

  As mtimecmp is a 64-bit register, it takes at least two write instructions to update its entire value. And partial update of mtimecmp might cause spurious timer interrupt. And in this regard, there are mainly two ways to deal with it:

  1. Disable timer interrupt before writing to mtimecmp. And the timer interrupt can be re-enabled after the writing is completed.

  2. Use the code sequence suggested in Ref [2], as shown below:

```
li t0, -1            # Write 0xFFFFFFFF into t0
sw t0, mtimecmp      # Set mtimecmp's lower 32 bits as 0xFFFFFFFF
sw a1, mtimecmp + 4  # Set mtimecmp's higher 32 bits
sw a0, mtimecmp      # Set mtimecmp's lower 32 bits
```

  Please note that the assembly code above has to be executed exactly in order. Compiler optimization, the intervention of ISR etc. may affect the correctness of the above code.

Address	Name	Description
0x20000000	MTIME_LOW	lower 32 bits of mtime
0x20000004	MTIME_HIGH	higher 32 bits of mtime
0x20000008	MTIMECMP_LOW	lower 32 bits of mtimecmp
0x2000000C	MTIMECMP_HIGH	higher 32 bits of mtimecmp

Table 2-8 Memory Mapped Address for mtime and mtimecmp

2.3.7 Trap (Exception and Interrupt)

Exception and interrupt are the mechanism for processors to handle asynchronous events. The difference between them is that:

1. The exception is caused by the software execution itself. The usual cases are:
   - Accessing a non-exist CSR
   - Unaligned access to memory
   - Break Point or System Call from Operating Systems

   However, please note that divided-by-zero will not cause exception on RISC-V processors.

2. The interrupt is caused by events independent of software execution. On PulseRain FRV2000 processor, the interrupt can only come from two sources, as shown in :
   a) Timer Interrupt
   b) External Interrupt, mainly caused by peripherals

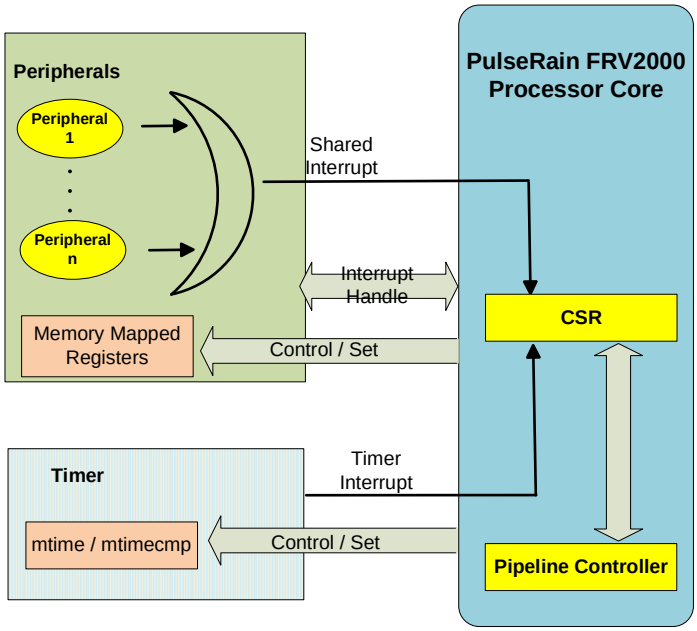

Figure 2-10 Interrupt for PulseRain FRV2000 Processor

2.3.7.1 Trap Trigger and Handle

As the exception and interrupt are very similar, they are handled in the same way on PulseRain FRV2000 RISC-V Processor. And they can both be called trap. When trap happens, the processor will go through the following procedures:

1. If the trap is an interrupt, the following two CSRs will be checked to see if the interrupt is masked
   a) The MIE bit in mstatus (Table 2-7).
   b) The correspondent bits in mie (Machine Interrupt Register). And the bit definition for mie can be found in Table 2-9.

Bit Index	Comments
7	*Machine Mode Timer Interrupt Enable(MTIE)*
11	*Machine Mode External Interrupt Enable (MEIE)*
[31 : 12]	*For future expansion*

Table 2-9 Bit Definition for MIE

2. Identify the cause of the trap.

   When trap happens, the processor needs to put the cause of the trap into CSR mcause. The MSB (bit 31) of mcause will identify whether this trap is interrupt (high) or exception (low). The resting bits of mcause are used as Exception Code. At this point, only the lower 4 bits of them are being used. If the trap is an interrupt, its correspondent Exception Code (The interrupt source) is shown in Table 2-10.

Exception Code	Interrupt Source
7	*Machine Mode Timer Interrupt*
11	*Machine Mode External Interrupt*

Table 2-10 Interrupt Source

   And if the trap is an exception, the correspondent exception category is shown Table 2-11.

Exception Code	Exception Category
0	*Unaligned Instruction Memory Access*
1	*Instruction Fetch Failure*

Exception Code	Exception Category
2	*Illegal Instruction*
3	*Break Point*
4	*Unaligned Data Memory Load Access*
5	*Memory Load Failure*
6	*Unaligned Data Memory Store Access*
7	*Memory Store Failure*
11	*Environment Call for Machine Mode*

Table 2-11 Exception Category

3. Identify the instruction address of the trap
   For the machine mode, RISC-V ISA has defined a CSR called mepc (Machine Exception Program Counter). For exception, the mepc will have the address of the instruction that causes the exception. For interrupt, the value in the mepc will be used by mret instruction to return from ISR.

4. Provide additional parameters for the trap
   To help handle the trap, RISC-V ISA has defined a CSR called mtval (Machine Trap Value Register). to provide additional parameters for the trap. When the exception is caused by memory access, the load/store address will be saved in this CSR.

5. Load the PC with trap handler
   For machine mode, RISC-V ISA has defined a CSR called mtvec (Machine Trap Vector Base-Address Register). This CSR will be used to determine the address of the trap handler. The lower two bits of this CSR are used for trap mode, while the higher 30 bits are used as base address. And the trap mode is defined in Table 2-12.

Trap Mode	Description
0	*Direct Mode. In this mode, PC = mtevc*

Trap Mode	Description
1	*Vector Mode. In this Mode* *PC = BASE + 4 * Exception_Code*。 *Exception Code comes from mcause, which can be found in* Table 2-10 *and* Table 2-11
2，3	*Reserved*

Table 2-12 Trap Mode

6. Setup CSR mip for interrupts
   For interrupts, the correspondent bits in CSR mip should be set to indicate the pending status. And these bits are defined in Table 2-13:

Bit Index	Comments
7	*Machine Mode Timer Interrupt Pending (MTIP)*
11	*Machine Mode External Interrupt Pending (MEIP)*

Table 2-13 Bit Definition for MIP

2.3.7.2 Trap Return

As mentioned early, in machine mode, when the trap handler is completed, it needs to use the mret instruction to return. And the Program Counter will be loaded with the value in CSR mepc.

2.3.7.3 WFI

As a way to provide support for Operating System scheduling, RISC-V Instruction Manual Volume II (Ref [2]) has defined an instruction for interrupt waiting, called WFI (Wait for Interrupt). When the processor encounters WFI instruction, the pipeline will enter stall state, until an interrupt happens.

2.3.7.4 ECALL and EBREAK

To provide support for Operating System and Debugger, RISC-V ISA has also defined ECALL (Environment Call) and EBREAK (Breakpoint) instructions. Both of those instructions will cause exception, and their correspondent exception code can be found in Table 2-11.

One thing to be noted is that when the processor encounters ECALL/EBREAK, it will fill mepc with the address of current instruction, instead of the one for the next instruction.

2.3.8 Internal Structure

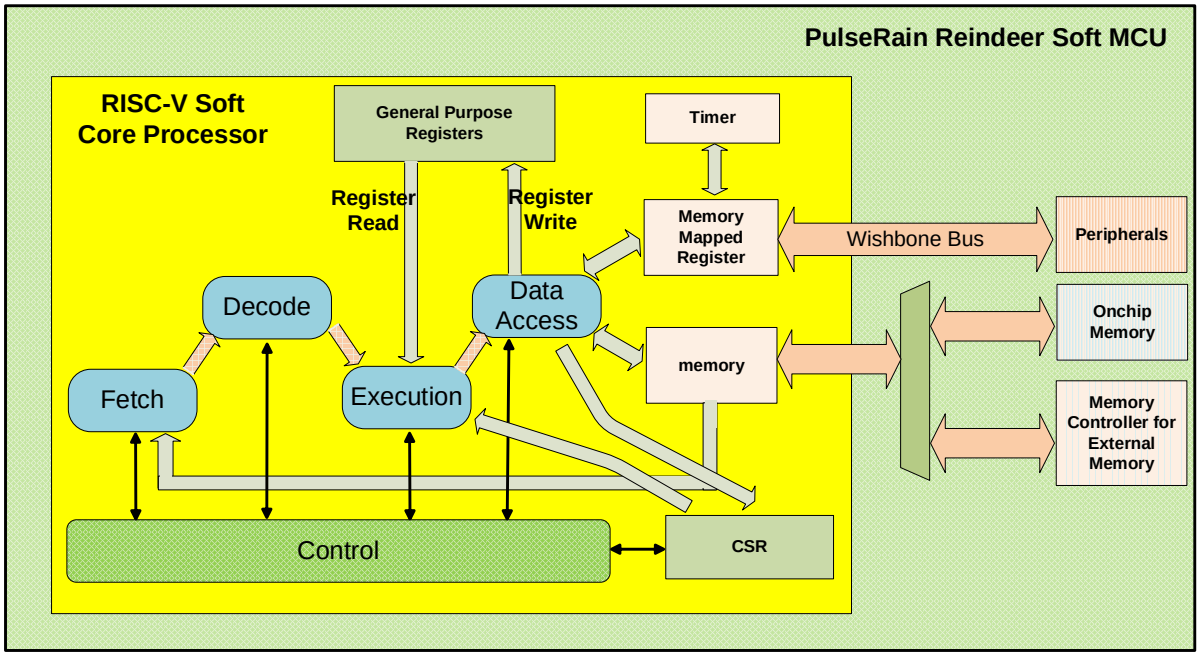

Figure 2-11 The Internal Structure of PulseRain FRV2000 Soft Core Processor

The internal structure for PulseRain FRV2000 Processor can be found in Figure 2-11 (the yellow block). It is mainly comprised of the following modules:

- General Purpose Registers
  Per RISC-V specification (Ref [1]), the Pulserain FRV2000 processor supports 32 of 32-bit general purpose registers, as mentioned in Section 2.3.3.

- 64-bit timer, as mentioned in Section 2.3.6

- Memory Mapped Registers for Peripherals
  The peripherals will be discussed in a latter part of this documents.

- CSR, as mentioned in Section 2.3.5.

- Pipeline and Control

And the RTL implementation of the above modules will be discussed next.

2.3.8.1 General Purpose Registers

As mentioned in Section 2.3.2, PulseRain FRV2000 processor has 32 general purpose registers. Each register is 32-bit wide. For FPGA implementation, the traditional approach will take 32*32=1024 registers, plus two 32:1 mux for the two source registers (rs1 and rs2 in RISC-V ISA).

To be resource efficient, PulseRain FRV2000 processor takes a new approach by using two single-port block RAM to realize the registers, as shown Figure 2-12.

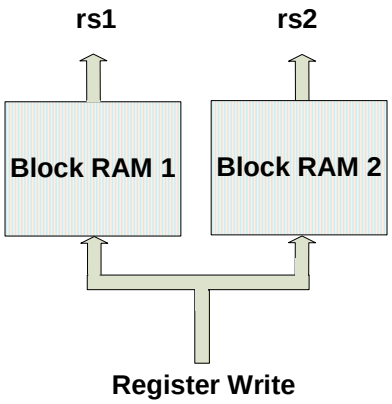

Figure 2-12 General Purpose Register Implemented in Block RAM

As mentioned in Section 2.1.2, the Block RAM is platform dependent, and each FPGA vendor has its own way to implement the Block RAM. When porting the processor to different FPGA platforms, the designers need to provide replacement for those ram files mentioned in 2.1.2.

2.3.8.2 CSR

All the CSRs listed in Table 2-5 have been implemented in PulseRain FRV2000 Processor. As reading CSRs might cause side effect, the read address and the read enable will only be active after instruction decoding, and there in no speculative read for CSR.

2.3.8.3 System Timer

As illustrated in Figure 2-11, peripherals are implemented outside of processor core, at MCU level. Theoretically, system timer is also a peripheral. However, given the fact that the timer is officially defined in the RISC-V ISA, and is mandatory to all processor implementation, PulseRain FRV2000 processor has given the timer a special treatment by pulling the timer into the processor core.

And the default resolution for the system timer is 1 microsecond (1 MHz base frequency), as defined by the MTIME_CYCLE_PERIOD macro in common.vh.

2.3.8.4 Pipeline

PulseRain FRV2000 is a soft-core processor of Von Neumann architecture. It features a 2 x 2 pipeline, striving to make a balance between speed and area. And it offers a flexible/portable choice for soft-core processors across all FPGA platforms.

The pipeline of FRV2000 is composed of 4 stages:

- Instruction Fetch (IF)
- Instruction Decode (ID)
- Instruction Execution (IE)
- Register Write Back and Memory Access (MEM)

However, unlike traditional pipelines, FRV2000's pipeline stages are mapped to a 2 x 2 layout, as illustrated Figure 2-13.

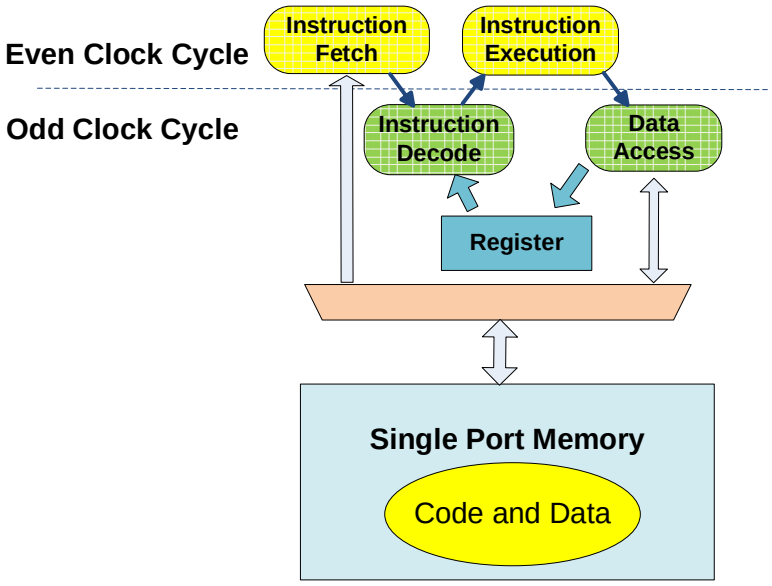

Figure 2-13 Two by Two Pipeline

In the 2 x 2 layout, each stage is active every other clock cycle. For the even cycle, only If and IE stages are active, while for the odd cycle, only ID and MEM stages are active. In this way, the Instruction Fetch and Memory Access always happen on different clock cycles, thus to avoid the structural hazard caused by the single port memory.

Thanks to using single port memory to store both code and data, PulseRain FRV2000 soft-core processor is quite portable and flexible across all FPGA platforms. And it can also achieve very high clock rate on low speed grade FPGAs.

2.3.8.5 Pipeline Stage - Instruction Fetch

The base instruction format (RV32I) for RISC-V can be found in Figure 2-14. As shown in Figure 2-14, the source register addesses are always in bits [19 : 15] and [24 : 20]. As reading general registers will not cause any side effect, the instruction fetch will speculatively read the general registers without instruction decoding.

And for C Extension (16-bit Compressed Instructions), the Instruction Fetch unit will translate them into RV32I instructions.

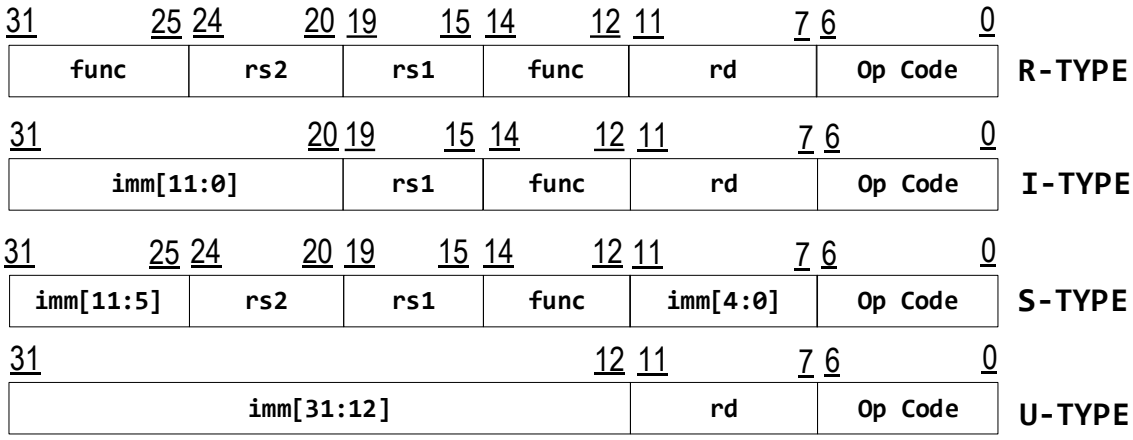

Figure 2-14 Instruction Format for RV32I

2.3.8.6 Pipeline Stage - Instruction Decode

As shown in Figure 2-14, the Operation Code is always in bits [6 : 0], with bits [1 : 0] for instruction length. As all the Compressed Instructions are translated into RV32I instructions in the Instruction Fetch stage, the Instruction Decode Stage will only look at bits [6: 2] for decoding.

2.3.8.7 Pipeline Stage - Instruction Execution

Instruction Execution Stage has to do the following jobs:

- ALU (Arithmetic Logic Unit)

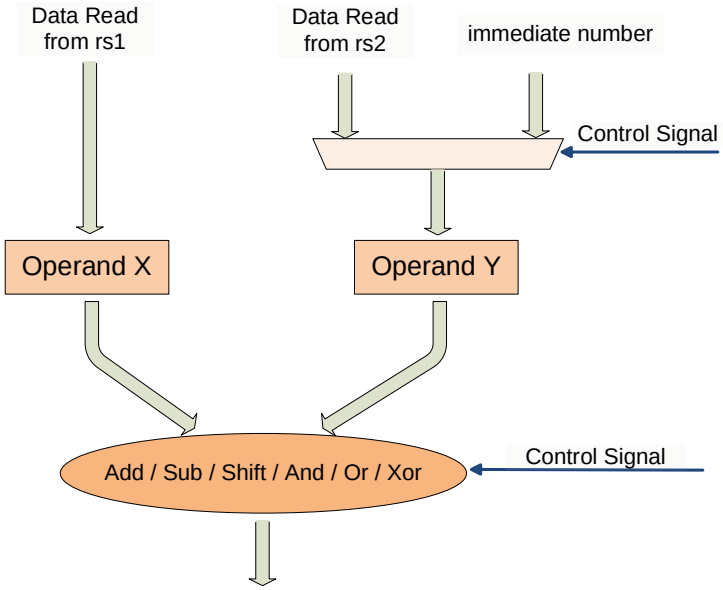

Figure 2-15 ALU

As shown in Figure 2-15, the ALU will support “Add”, “Subtract”, “Shift”, “And”, “Or”, “Xor” operations. for the two operands (X and Y), the operand X always comes from the general purpose registers, while the operand Y can come from general purpose register or from the immediate number embedded in the instructions. The selection of data and the arithmetic/logic operations are all administrated by the pipeline controller.

- MUL / DIV (M Extension)
  For PulseRain FRV2000 Processors, the M extension can be optionally supported by setting the ENABLE_HW_MUL_DIV macro config.vh

- Unconditional Jump (JAL / JALR)
- LUI / AUIPC (Load Upper Immediate / Add Upper Immediate to PC)

All the above instructions will update the target register (rd). The details are listed below in Table 2-14.

Instruction	Value To Be Written to the target register (rd)
ALU	*The output of ALU*
MUL/DIV	*The result of MUL/DIV*
JAL / JALR	*PC + 4 (32 bit instruction) or PC + 2 (16 bit instruction)*
LUI	*Bit [31 : 12] left shift by 12 bits to generate 32 bit value*
AUIPC	*Bit[31 : 12] left shift by 12 bits, then add to PC*

Table 2-14 The value to be written to rd

In addition, the instruction decoder also needs to deal with the following instructions:

- CSR
  In FRV2000, the instruction decoder has a dedicated bus to CSR for data update.

- Conditional Branch
- ECALL / EBREAK
- LOAD / STORE

2.3.8.8 Pipeline Stage – Data Access

At this stage, the general registers will be updated. And the memory access generated by LOAD / STORE will also happen in this stage. Thanks to the 2 x 2 pipeline layout, the Data Access and Instruction Fetch will not collide with each other for memory access.

2.3.8.9 Pipeline Control

The main job of pipeline control is to handle the branch instruction and exception/interrupt, as illustrated in Figure 2-16. For the 2 x 2 pipeline layout, FETCH_EXE state and DECODE_DATA state will correspond to the even clock cycle and single clock cycle in Figure 2-13. And branch instructions will bring the state from FETCH_EXE to INIT, and reload the pipeline.

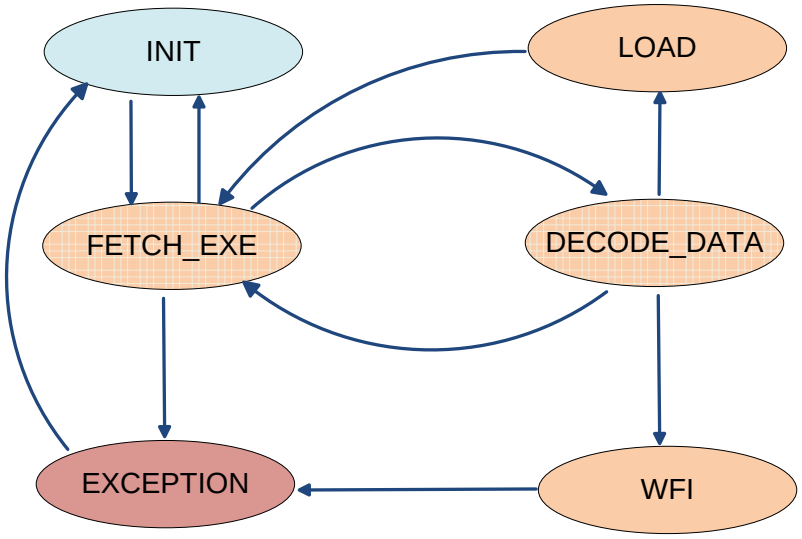

Figure 2-16 State Diagram for Pipeline Control

And for exception and interrupt, the EXCEPTION state will identify the cause of the trap, and set up the exception code (Table 2-10 and Table 2-11) accordingly. The return address (mepc) will also be determined in this state.

2.3.9 Address Assignment

For the memory address space of PulseRain FRV2000 processor, the addresses are assigned as following:

Address	Description
0x00000000 – 0x1FFFFFFF	Reserved
0x20000000 – 0x3FFFFFFF	Memory Mapped Register
0x40000000 – 0x7FFFFFFF	Reserved
0x80000000 – 0xFFFFFFFF	Code and Data

Table 2-15 Address Assignment for PulseRain FRV2000 Processor

2.3.10 Instruction Supported

2.3.10.1 Base Instruction

PulseRain FRV2000 Processors support the RISC-V RV32I base instruction set, as listed in Table 2-16, Table 2-17, Table 2-18, Table 2-19, Table 2-20 and Table 2-21.

Mnemonics	Description	FRV2100	FRV2200
ADDI rd, rs1, imm	rd = rs1 + imm	Y	Y
SLTI rd, rs1, imm	rd = (rs1 < imm) ? 1 : 0 (signed compare)	Y	Y
SLTIU rd, rs1, imm	rd = (rs1 < imm) ? 1 : 0 (unsigned compare)	Y	Y
SEQZ rd, rs	pseudo instruction for SLTI rd, rs1, 1	Y	Y
ANDI rd, rs1, imm	rd = rs1 & imm	Y	Y
ORI rd, rs1, imm	rd = rs1 | imm	Y	Y
XORI rd, rs1, imm	rd = rs1 ^ imm	Y	Y
SLLI rd, rs1, imm	rd = rs1 << imm[4:0]	Y	Y
SRLI rd, rs1, imm	rd = rs1 >> imm[4:0]	Y	Y
SRAI rd, rs1, imm	rd = rs1 >>> imm[4:0]	Y	Y
LUI rd, imm[31 : 12]	rd = {imm[31 : 12], 12'h000}	Y	Y
AUIPC rd, imm[31 : 12]	rd = PC + {imm[31 : 12], 12'h000}	Y	Y
NOP	pseudo instruction for ADDI 0, 0, 0	Y	Y

Table 2-16 Integer Register – Immediate Instructions

Mnemonics	Description	FRV2100	FRV2200
ADD rd, rs1, rs2	rd = rs1 + rs2	Y	Y
SUB rd, rs1, rs2	rd = rs1 - rs2	Y	Y
SLT rd, rs1, rs2	rd = (rs1 < rs2) ? 1 : 0 (signed compare)	Y	Y
SLTU rd, rs1, rs2	rd = (rs1 < rs2) ? 1 : 0 (unsigned compare)	Y	Y
AND rd, rs1, rs2	rd = rs1 & rs2	Y	Y
OR rd, rs1, rs2	rd = rs1 | rs2	Y	Y
XOR rd, rs1, rs2	rd = rs1 ^ rs2	Y	Y
SLL rd, rs1, rs2	rd = rs1 << rs2[4:0]	Y	Y
SRL rd, rs1, rs2	rd = rs1 >> rs2[4:0]	Y	Y
SRA rd, rs1, rs2	rd = rs1 >>> rs2[4:0]	Y	Y

Table 2-17 Integer Register – Register Instructions

Mnemonics	Description	FRV2100	FRV2200
JAL rd, imm[20 : 1]	rd = PC + 4; PC = PC + {11{imm[20]}, imm[20 : 1], 1'b0}	Y	Y
JALR rd, rs1, imm[11 : 0]	rd = PC + 4; PC = (rs1 + {20{imm[11]}, imm[11 : 0]}) & 0xFFFFFFFE	Y	Y

Table 2-18 Unconditional Jump Instructions

Mnemonics	Description	FRV2100	FRV2200
BEQ rs1, rs2, imm[12 : 1]	PC = (rs1 == rs2) ? PC + {19{imm[12]}, imm[12:1], 1'b0} : PC + 4	Y	Y
BNQ rs1, rs2, imm[12 : 1]	PC = (rs1 != rs2) ? PC + {19{imm[12]}, imm[12:1], 1'b0} : PC + 4	Y	Y

BLT rs1, rs2, imm[12 : 1]	PC = (rs1 < rs2) ? // Signed Compare
	PC + {19{imm[12]}, imm[12:1], 1'b0} :
	PC + 4
		Y	Y

BGE rs1, rs2, imm[12 : 1]	PC = (rs1 >= rs2) ? // Signed Compare
	PC + {19{imm[12]}, imm[12:1], 1'b0} :
	PC + 4
		Y	Y

BLTU rs1, rs2, imm[12 : 1]	PC = (rs1 < rs2) ? // Signed Compare
	PC + {19{imm[12]}, imm[12:1], 1'b0} :
	PC + 4
		Y	Y

BGEU rs1, rs2, imm[12 : 1]	PC = (rs1 >= rs2) ? // Signed Compare
	PC + {19{imm[12]}, imm[12:1], 1'b0} :
	PC + 4
		Y	Y

Table 2-19 Conditional Branch Instructions

Mnemonics	Description	FRV2100	FRV2200
LB rd, rs1, imm[11 : 0]	rd [7 : 0] = (int8_t)*(rs1 + {20{imm[11]}, imm[11 : 0]}) rd [31 : 8] = sign extension	Y	Y
LBU rd, rs1, imm[11 : 0]	rd [7 : 0] = (int8_t)*(rs1 + {20{imm[11]}, imm[11 : 0]}) rd [31 : 8] = 0	Y	Y
LH rd, rs1, imm[11 : 0]	rd [15 : 0] = (int16_t)*(rs1 + {20{imm[11]}, imm[11 : 0]}) rd [31 : 16] = sign extension	Y	Y
LHU rd, rs1, imm[11 : 0]	rd [15 : 0] = (int16_t)*(rs1 + {20{imm[11]}, imm[11 : 0]}) rd [31 : 16] = 0	Y	Y
LW rd, rs1, imm[11 : 0]	rd [31 : 0] = (int32_t)*(rs1 + {20{imm[11]}, imm[11 : 0]})	Y	Y

Table 2-20 Load Instructions

Mnemonics	Description	FRV2100	FRV2200
SB rs1, rs2, imm[11 : 0]	(int8_t)*(rs1 + {20{imm[11]}, imm[11 : 0]}) = rs2 & 0xFF	Y	Y
SH rs1, rs2, imm[11 : 0]	(int16_t)*(rs1 + {20{imm[11]}, imm[11 : 0]}) = rs2 & 0xFF	Y	Y
SW rs1, rs2, imm[11 : 0]	(int32_t)*(rs1 + {20{imm[11]}, imm[11 : 0]}) = rs2	Y	Y

Table 2-21 Store Instructions

Mnemonics	Description	FRV2100	FRV2200
FENCE / FENCE.I	Flush the pipeline	Y	Y

Table 2-22 FENCE/FENCE.I Instructions

2.3.10.2 CSR Instruction

For CSR instructions, the CSR register address can be found in bit [31 : 20]. And RISC-V has defined 6 CSR access instructions, which can be found in Table 2-23.

Mnemonics	Description	FRV2100	FRV2200
CSRRW rd, rs1, CSR	(CSR)→(rd)； (rs1)→(CSR)	Y	Y
CSRRS rd, rs1, CSR	(CSR)→(rd)； (rs1) | (CSR) →(CSR)	Y	Y
CSRRC rd, rs1, CSR	(CSR)→(rd)； (rs1) | (~(CSR)) →(CSR)	Y	Y
CSRRWI rd, CSR, imm	(CSR)→(rd)； immediate →(CSR)	Y	Y
CSRRSI rd, CSR, imm	(CSR)→(rd)； (CSR) | immediate →(CSR)	Y	Y
CSRRCI rd, CSR, imm	(CSR)→(rd)； (~(CSR)) & immediate →(CSR)	Y	Y

Table 2-23 CSR Instructions

2.3.10.3 ECALL / EBREAK Instruction

Mnemonics	Description	FRV2100	FRV2200
ECALL	Please see Section 2.3.7.4	Y	Y
EBREAK	Please see Section 2.3.7.4	Y	Y

Table 2-24 ECALL/EBREAK Instructions

2.3.10.4 M Extension

Mnemonics	Description	FRV2100	FRV2200
MUL rd, rs1, rs2	Signed Multiplication, keep lower 32 bits to rd	Optional	Optional
MULH rd, rs1, rs2	Signed Multiplication, keep higher 32 bits to rd	Optional	Optional
MULHSU rd, rs1, rs2	Signed multiplies unsigned number, keep higher 32 bits to rd	Optional	Optional
MULHU rd, rs1, rs2	Unsigned multiplication, keep higher 32 bits to rd	Optional	Optional
DIV rd, rs1, rs2	Signed division, save quotient to rd	Optional	Optional
DIVU rd, rs1, rs2	Unsigned division, save quotient to rd	Optional	Optional
REM rd, rs1, rs2	Signed division, save remainder to rd	Optional	Optional
REMU rd, rs1, rs2	Unsigned division, save remainder to rd		

Table 2-25 MUL / DIV

Please note that those instructions in Table 2-25 will never trigger exceptions. And the following are two special cases that could happen during the division:

1. Divided by Zero
   In this case, the quotient is 32'hFFFFFFFF, and the remainder will be the same as rs2.

2. Overflow during signed division
   This could happen when rs1 = 32'80000000 and rs2 = 32'hFFFFFFFF. In this case, the quotient will be 32'h80000000, and the remainder will be zero.

2.3.10.5 C Extension

The C extension (16-bit Compressed Instructions) is not an independent instruction set. Each instruction in the C extension can be mapped to a 32-bit instruction from Section 2.3.10.1. For PulseRain FRV2200, its instruction fetch stage will translate the 16-bit instructions into their 32-bit counterparts, and keep the other pipeline stages consistent and unchanged for 32-bit processing only.

And to compress the instructions into 16-bit format, the C extension has introduced 3-bit register index rd', rs1' and rs2' that can only be mapped to register x8-x15.

The instructions from C extension are listed in Table 2-26, Table 2-27, Table 2-28, Table 2-29, Table 2-30, Table 2-31, Table 2-32 and Table 2-33.

Mnemonics	32-Bit Counterpart	FRV2100	FRV2200
C.LWSP	lw rd, offset[7 : 2](x2)	N	Y
C.LW	lw rd', offset[6 : 2](rs1')	N	Y

Table 2-26 16-bit Load

Mnemonics	32-Bit Counterpart	FRV2100	FRV2200
C.SWSP	sw rs2, offset[7 : 2](x2)	N	Y
C.SW	sw rs2', offset[6 : 2](rs1')	N	Y

Table 2-27 16-bit Store

Mnemonics	32-Bit Counterpart	FRV2100	FRV2200
C.J	jal x0, offset[11:1]	N	Y
C.JAL	jal x1, offset[11:1]	N	Y
C.JR	jalr x0, rs1, 0	N	Y
C.JALR	jalr x1, rs1, 0	N	Y

Table 2-28 16-bit Unconditional Jump

Mnemonics	32-Bit Counterpart	FRV2100	FRV2200
C.BEQZ	beq rs1', x0, offset[8:1]	N	Y
C.BNEZ	bne rs1', x0, offset[8:1]	N	Y

Table 2-29 16-bit Conditional Branch

Mnemonics	32-Bit Counterpart	FRV2100	FRV2200
C.LI	addi rd, x0, imm[5:0] (rd ≠ 0)	N	Y
C.LUI	lui rd, nzuimm[17:12] (rd ≠ 0，rd ≠ 2)	N	Y

Table 2-30 16-bit Constant-Generation

Mnemonics	32-Bit Counterpart	FRV2100	FRV2200
C.ADDI	addi rd, rd, nzimm[5:0] (rd ≠ 0)	N	Y
C.ADDI4SPN	addi rd', x2, nzuimm[9:2]	N	Y
C.SLLI	slli rd, rd, shamt[4:0] (shamt[4:0] ≠ 0)	N	Y
C.SRLI	srli rd', rd', shamt[4:0] (shamt[4:0] ≠ 0)	N	Y
C.SRAI	srai rd', rd', shamt[4:0] (shamt[4:0] ≠ 0)	N	Y
C.ANDI	andi rd', rd', imm[5:0]	N	Y

Table 2-31 16-bit Integer Register-Immediate Instruction

Mnemonics	32-Bit Counterpart	FRV2100	FRV2200
C.MV	add rd, x0, rs2	N	Y
C.ADD	add rd, rd, rs2	N	Y
C.AND	and rd', rd', rs2'	N	Y
C.OR	or rd', rd', rs2'	N	Y
C.XOR	xor rd', rd', rs2'	N	Y
C.SUB	sub rd', rd', rs2'	N	Y

Table 2-32 16-bit Integer Register-Register Instruction

Mnemonics	32-Bit Counterpart	FRV2100	FRV2200
C.NOP	addi x0, x0, 0	N	Y
C.EBREAK	ebreak	N	Y

Table 2-33 16-bit Miscellaneous

2.3.11 Prologue / Epilogue and Alternate Link Register

During the early discussion of calling convention in Section 2.3.3, it was mentioned that the register x5 can be used as a temp register or as an alternate link register (Table 2-4). And the usage of alternate link register is explained below:

The main purpose of introducing RISC-V C extension is to improve code density. While at the beginning of each function call, there should be some code to push the current register values into the stack. And at call return, there should also be some other code to pop the stack and restore the register values. Those code

are called prologue and epilogue respectively. As the prologue and epilogue appear frequently, it is natural for designers to look for ways to reduce their overhead.

In this regard, ARM has introduced special instructions called Load-Multiple and Store-Multiple to transfer multiple words between registers and memory in a single instruction cycle. According to Ref [5], the Load-Multiple and Store-Multiple can improved the code density of Linux Kernel by 8%.

However, to keep the C extension as a simplified subset of RV32I, RISC-V has taken a different approach, borrowing pages from IBM S/390's playbook, with something similar to millicode routine.

As the only job for prologue and epilogue is to move data between register and stack, this job can be outsourced to shared code routine. In other words, the prologue and epilogue can also do function calls to the shared code routine to complete this job.

However, this function call for prologue and epilogue is a special one, as it does not need prologue and epilogue to do its job. It cannot use x1(ra) for return address either. And that's where the x5 (t0 / alternate link register) comes in. According to Ref [5], the millicode routine can improve the code density of Linux Kernel by 7.5%, without resorting to special instructions like Load-Multiple or Store-Multiple.

2.4 Peripherals

2.4.1 Wishbone FASM Interface

As illustrated in Figure 2-1, all the peripheral registers for PulseRain FRV2000 are mapped into the memory mapped address space, and are accessed through Wishbone FASM interface (Ref [8]). All peripherals share the same Wishbone bus, and the processor core is the only master on the bus.

For master read, the signals are as following on the master side:

Signal Name	Description
RD_STB_O	Read Strobe, always high for FRV2000
RD_ADR_O	Read Address, 32-bit shared signal
RD_DAT_I	32-bit read data. The data from each peripheral are mux-ed into RD_DAT_I. And the RD_DAT_I should be valid on the next cycle after RD_ADDR_O is provided. If long read latency is expected, software should insert NOP to compensate for the read delay.
RD_ACK_I	Read Acknowledge from the slave. For FRV2000, this signal it not used and can be tied to 1

Table 2-34 Wishbone FASM Read Signal

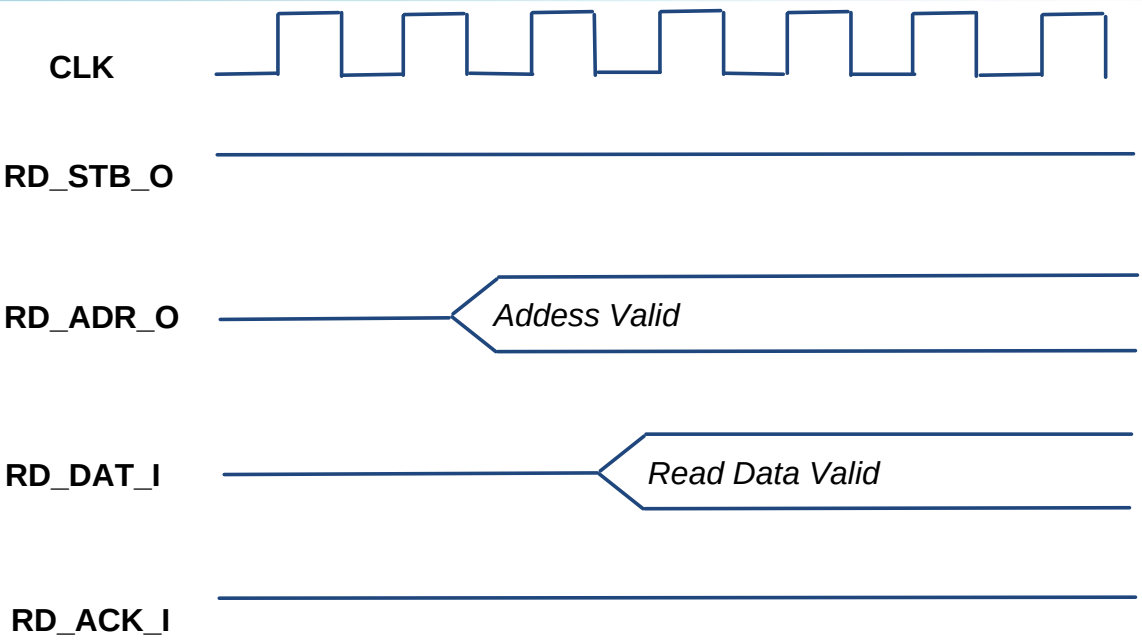

Figure 2-17 Timing Diagram for Master Read

For master write, the signals are as following on the master side:

Signal Name	Description
WR_STB_O	Write Strobe, always high for FRV2000
WR_WE_O	Write Enable
WR_ADR_O	Write Address, 32-bit shared signal
WR_DAT_O	32-bit write data, available at the same time as WR_ADR_O

Table 2-35 Wishbone FASM Write Signal

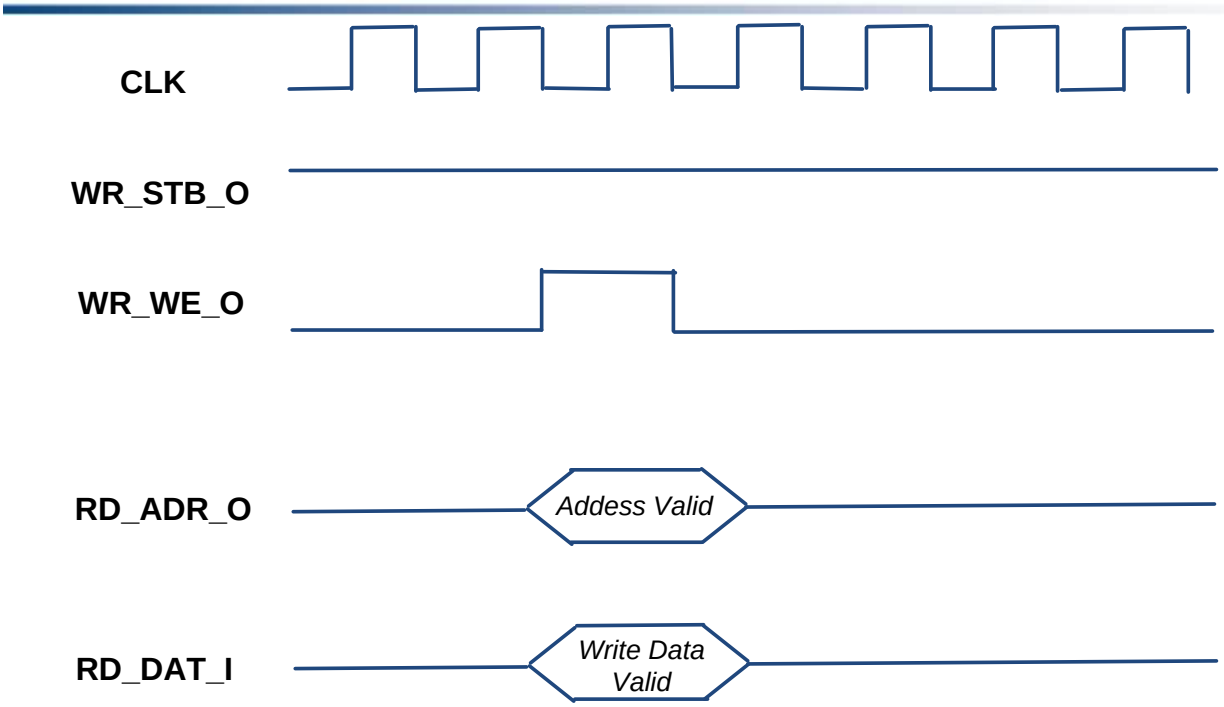

Figure 2-18 Timing Diagram for Master Write

2.4.2 Peripheral Address Mapping

As mentioned early, all the peripherals have their registers exposed to the processor core as memory mapped registers, and these memory mapped registers are accessed through Wishbone FASM interface. Each variant of PulseRain MCU has its own set the peripherals that match what's available on the hardware platform. And the following is just one example of the most commonly used configuration, with addresses listed in Table 2-36:

Address	Register Name	Description
0x20000000	MTIME_LOW	lower 32 bits of mtime (Section 2.3.6)
0x20000004	MTIME_HIGH	higher 32 bits of mtime (Section 2.3.6)
0x20000008	MTIMECMP_LOW	lower 32 bits of mtimecmp (Section 2.3.6)
0x2000000C	MTIMECMP_HIGH	higher 32 bits of mtimecmp (Section 2.3.6)
0x20000010	UART_TX	TX control / status register for UART
0x20000014	UART_RX	RX control / status register for UART
0x20000018	GPIO	control / status register for general purpose In / Out
0x2000001C	INT_SOURCE	status register for external interrupt
0x20000020	INT_ENABLE	control register for external interrupt

Table 2-36 Peripheral Address Mapping

2.4.3 UART

A typical PulseRain RISC-V MCU will carry a UART with a fixed baud rate of 115200 bps. Its transmitter can be controlled and monitored through the register UART_TX (Table 2-36). And its receiver can be controlled and monitored through the register UART_RX (Table 2-36).

The bit definition of UART_TX can be found in Table 2-37:

Bits Index	Name	R/W	Default	Description
7 : 0	TX Data	Write Only	0	The byte to be transmitted. Always zero when being read.
30 : 8	Reserved	N/A	0	Always zero
31	tx_active	Read Only	0	When this bit is high, the transmitter is busy

Table 2-37 UART_TX Bit Definition

And the bit definition of UART_RX can be found in Table 2-38:

Bits Index	Name	R/W	Default	Description
7 : 0	RX Data	Read Only	0	The byte received.
28 : 8	Reserved	N/A	0	Always zero
29	uart_rx_fifo_not_empty	Read Only	0	This bit will be set to high if there is data in the receiving FIFO.
30	uart_rx_fifo_full	Read Only	0	This bit will be set to high if the receiving FIFO is full.
31	Reserved	N/A	0	Always zero

Table 2-38 UART_RX Bit Definition

2.4.4 GPIO

A typical PulseRain RISC-V MCU will support 32 Input Pins and 32 Output Pins as GPIO. And they are controlled and monitored through the peripheral register GPIO (Table 2-36).

When the GPIO register is being written to, the correspondent output pin will change accordingly. When the GPIO register is being read, the logic level for the correspondent input pin will be extracted.

2.4.5 External Interrupt

As shown in Figure 2-10, all external interrupts will be wired-or together as one shared interrupt to the PulseRain FRV2000 processor core. The external interrupts can come from the peripherals, or come from the INTx pins on the MCU port list. All interrupts are level triggered.

2.4.5.1 Interrupt Status

The external interrupt status can be found out from the register INT_SOURCE (Table 2-36). The bit definition of this register varies with the MCU configuration. A typical example can be found in Table 2-39.

Bits Index	Name	R/W	Default	Description
0	Reserved	N/A	0	Always zero
1	uart_rx_fifo_not_empty	Read Only	0	Interrupt when UART RX FIFO is not empty
29 : 2	Reserved	N/A	0	Always zero
31 : 30	INT1 and INT0	Read Only	0	Status of INTx pin on the port list

Table 2-39 INT_SOURCE Bit Definition

2.4.5.2 Interrupt Enable / Disable

The interrupt sources can be enabled/disabled individually by a register called INT_ENABLE (Table 2-36). Its bit definition will match what is defined in INT_SOURCE. A logic high in the correspondent bit will enable the interrupt, and

2.5 RTL Simulation

2.5.1.1 Compatibility Test

The PulseRain RISC-V MCU can be simulated at hardware platform level, which includes PLL and other IPs. To run RTL simulation, please see the detailed procedures below:

1. Checkout the master platform
   The master platform is a mocked hardware platform for the PulseRain RISC-V MCU. Take the FRV2100 for example:

   ```
   $ git clone https://github.com/PulseRain/Reindeer_Master_Platform.git
   $ cd Reindeer_Master_Platform
   $ git submodule update --init –recursive
   ```

2. Simulate with Modelsim
   Launch Modelsim, enter the sim subfolder, and run the do files
   ModelSim > cd `Reindeer_Master_Platform/sim/modelsim`
   ModelSim > do `build_lib.do`
   ModelSim > do `build_soc.do`
   ModelSim > do `run_compliance.do`

   The run_compliance.do script will try to match the test vectors derived from the RISC-V official compatibility test.

2.5.1.2 Simulate elf file

Elf file can also be simulated. However, the elf file has to be converted and pre-loaded into DRAM (make a DRAM sim file). For this purpose, a python script called “dram_dat_gen.py” has been provided in the sim/modelsim directory. And the python script can be run like the following:

```
python dram_dat_gen.py filename_of_elf_file.elf > sdram_ISSI_SDRAM_test_component.dat
```

After that, the simulation can be started by the running the do file:

```
ModelSim > do run_sim.do
ModelSim > run 100us
```

3 Software

PulseRain RISC-V MCU can provide a software interface that is compatible with the Arduino. In other words, through Arduino IDE, users can write sketches in Arduino Language and program the MCU through UART.

3.1 Compiler

PulseRain Technology will provide a board package to be used by the Arduino IDE. And this board package will include the GNU RISC-V Embedded GCC (https://xpack.github.io/riscv-none-embed-gcc/), which will produce bard-metal images to be uploaded by the Arduino IDE.

Those images may or may not contain the code to initialize the stack pointer. To be safe, the hardware will usually program the x2 register with the biggest physical address available when the MCU starts to run.

3.2 Arduino Language

With the board package provided, PulseRain RISC-V MCU is able to present a software programming interface that is compatible with the Arduino Language.

3.2.1 setup() and loop()

For processors, there are generally two ways to do the control flow:

1. Use an embedded OS, with tasks being handled by threads or processes.
2. Use an endless super-loop, with tasks being handled sequentially, and the asynchronous events are handled in ISR.

The second option is more straightforward and does not require complicated scheduler, at the expense of resource efficiency. And small to medium scale projects often take this approach. And this is also the approach that Arduino takes.

With the Arduino IDE, the user will provide a sketch that must contain the following two functions:

- setup()
- loop()

The setup() will be called only once, while the loop() will be invoked repetitively. In fact, behind the scene, the Arduino will also wrapper those two functions in the way shown in Figure 3-1.

```
#include "Arduino.h"

void main()
{
        setup();

        while(1) {
            loop();
        }
}
```

Figure 3-1 Wrap setup() and loop() in main() Function

3.2.2 Data Type

Compatible with the Arduino Language, the following data types are defined in Arduino.h:

```
typedef unsigned long     uint32_t;
typedef long              int32_t;

typedef unsigned short    uint16_t;
typedef short             int16_t;

typedef unsigned char     uint8_t;
typedef signed char       int8_t;

typedef uint8_t           byte;
typedef uint16_t          word;
```

List 3-1 Data Type for FP51-1T

Please note that String object is currently not supported by FP51-1T's software library.

3.2.3 APIs

Compatible with Arduino Language, the following APIs are supported by FP51-1T's software library:

3.2.3.1 Digital IO

- *void digitalWrite (uint8_t pin, uint8_t value)*

  Parameters:

  pin: IO pin index (0 ~ 31).

  value: HIGH or LOW (Behind the scene, HIGH / LOW are defined as the following:

  #define HIGH 1

  #define LOW 0

  Return Value: None

  Remarks: Use this function to set the logic value of the IO pin

- *uint8_t digitalRead (uint8_t pin)*

  Parameters:

  pin: IO pin index (0 ~ 31).

  Return Value: The current logic value of the IO pin (LOW (0) or HIGH (1))

  Remarks: Use this function to get the current logic value of the IO pin

3.2.3.2 Time

- *void delay (uint32_t delay_in_ms)*

  Parameters:

  delay_in_ms: delay value in millisecond

  Return Value: None

  Remarks: Use this function to delay in the granularity of millisecond.

- *void delayMicroseconds (uint32_t delay_in_us)*

  Parameters:

  delay_in_us: delay value in microsecond

  Return Value: None

  Remarks: Use this function to delay in the granularity of microsecond.

- *uint32_t millis ()*

Parameters: None

Return Value: Number of milliseconds passed since reset.

Remarks: Use this function to get the number of milliseconds passed since reset

- *uint32_t micros ()*

  Parameters: None

  Return Value: Number of microseconds passed since reset.

  Remarks: Use this function to get the number of microseconds passed since reset

3.2.3.3 Interrupt

- *void interrupts()*

  Parameters: None

  Return Value: None

  Remarks: Use this function to enable interrupt globally

- *void noInterrupts ()*

  Parameters: None

  Return Value: None

  Remarks: Use this function to disable interrupt globally

3.2.3.4 ISR Handler

- *void attachIsrHandler(uint8_t index, void (*isr_handler_pointer)(), uint8_t mode)*

Parameters:

index: IRQ index, the valid values are platform dependent

isr_handler_pointer: function pointer to the ISR handler

mode: only RISING is supported at this point

Return Value: None

Remarks: Use this function to attach ISR to the correspondent IRQ index. To detach the ISR, just set the isr_handler_pointer to NULL. This function is on par with Arduino Language's *attachInterrupt()* and *detachInterrupt()* functions.

3.2.3.5 Serial

Like the Arduino Language, PulseRain RISC-V MCU also supports Serial port as console in/out, and it supports the following functions:

- *Serial.begin()*
- *Serial.available()*
- *Serial.print()*
- *Serial.println()*
- *Serial.read()*
- *Serial.readBytes()*
- *Serial.write()*

3.3 Arduino IDE

The Arduino IDE supports 3rd party package for additional boards. The procedures to add PulseRain RISC-V MCU board support to Arduino IDE are as following:

3.3.1 Install Arduino IDE

PulseRain RISC-V MCU can be developed and programmed through Arduino IDE. On Windows 10, the Arduino IDE can be installed as an App directly from Windows App store. Otherwise, the Windows installer for Arduino IDE can be found at Arduino Website:

https://www.arduino.cc/en/Main/Software

3.3.2 Setup Arduino IDE

After the Arduino IDE is installed, launch it and click the menu File / Preferences, as shown in Figure 3-2.

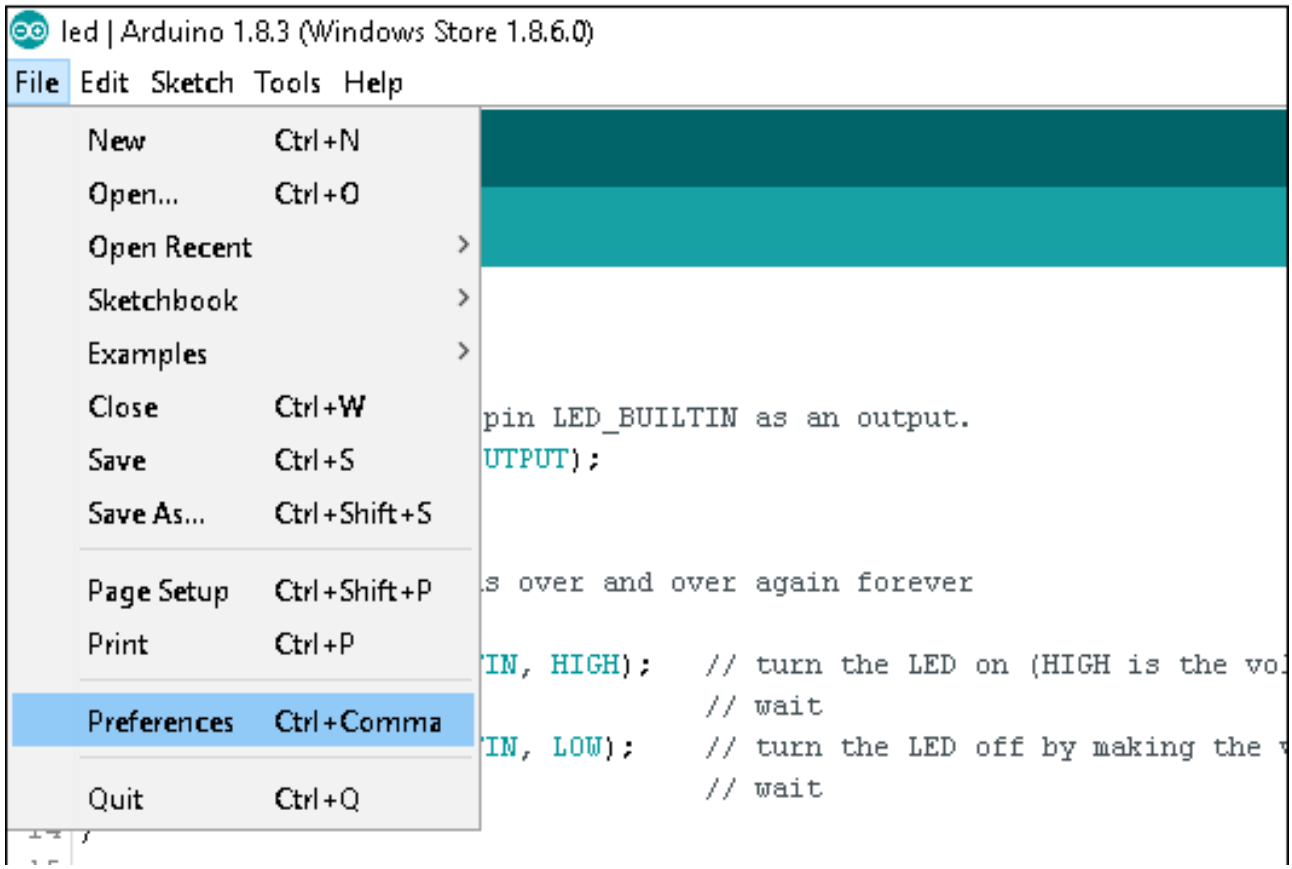

Figure 3-2 Arduino IDE, Preferences Menu

1) The File / Preferences menu will bring out a dialog like the one shown in Figure 3-3. Please set the "Additional Boards Managers URL" to
https://raw.githubusercontent.com/PulseRain/Arduino_RISCV_IDE/master/package_pulserain.com_index.json

(If this input box is not empty, use semicolon to separate multiple URLs.) And click OK to close the dialogue.

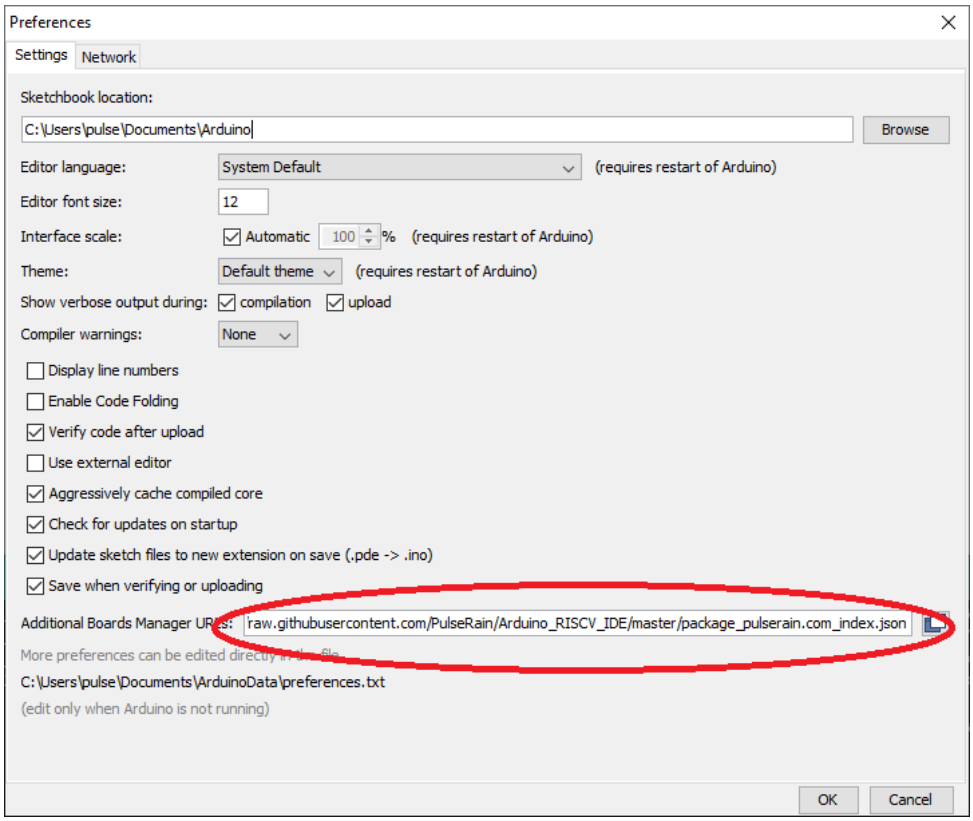

Figure 3-3 Arduino IDE, Preferences Dialogue

2) Now click the menu Tools / Boards / Boards Manager, as shown in Figure 3-4.

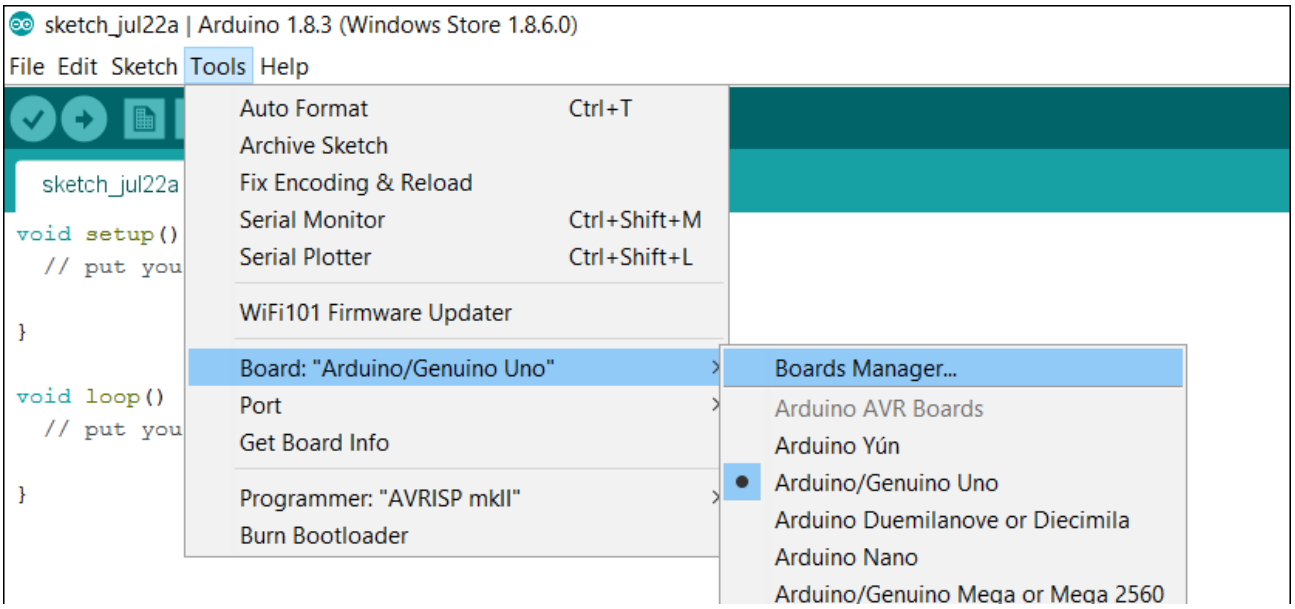

Figure 3-4 Arduino IDE, Boards Manager Menu

3) The "Boards Manager" will bring out a dialogue like the one shown in Figure 3-5. Type in "Reindeer" in the search box to find the board support package for the MachXO3D Breakout Board. Click "Install" to download and install the board support package.

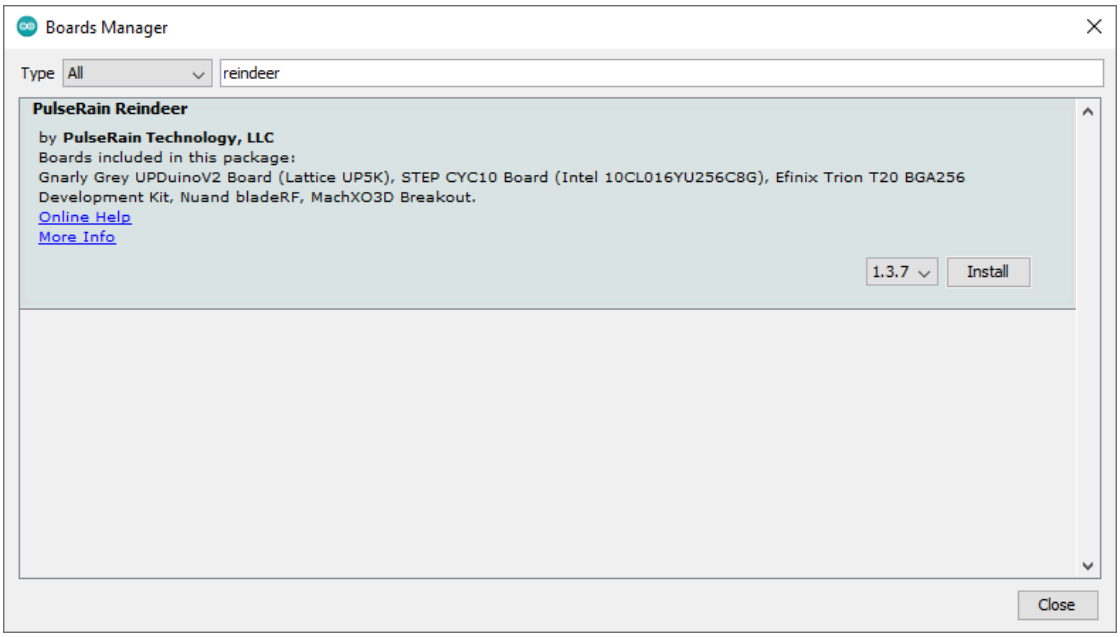

Figure 3-5 Arduino IDE, Boards Support Package for Lattice MachXO3D Breakout Board

3.3.3 Write Sketches

Just like Arduino, you need to select the correct COM port and Board Name before you can start writing sketches.

To select the COM port in Arduino IDE, click the menu Tool / Port, as shown in Figure 3-6. If you have trouble determining which COM port corresponds to the Breakout Board, you can always open the device manager to check, as illustrated inFigure 3-7.

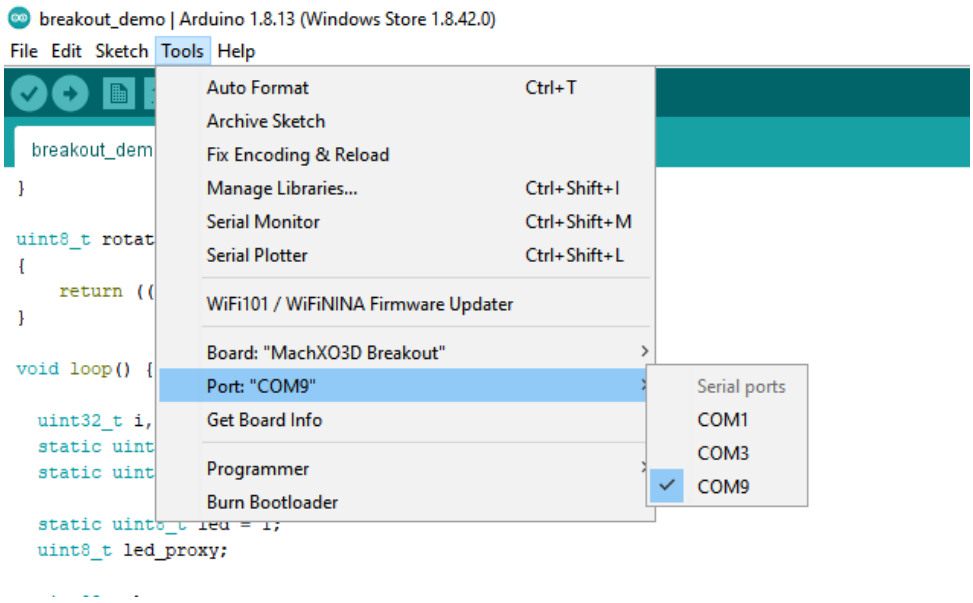

Figure 3-6 Arduino IDE, Select COM port

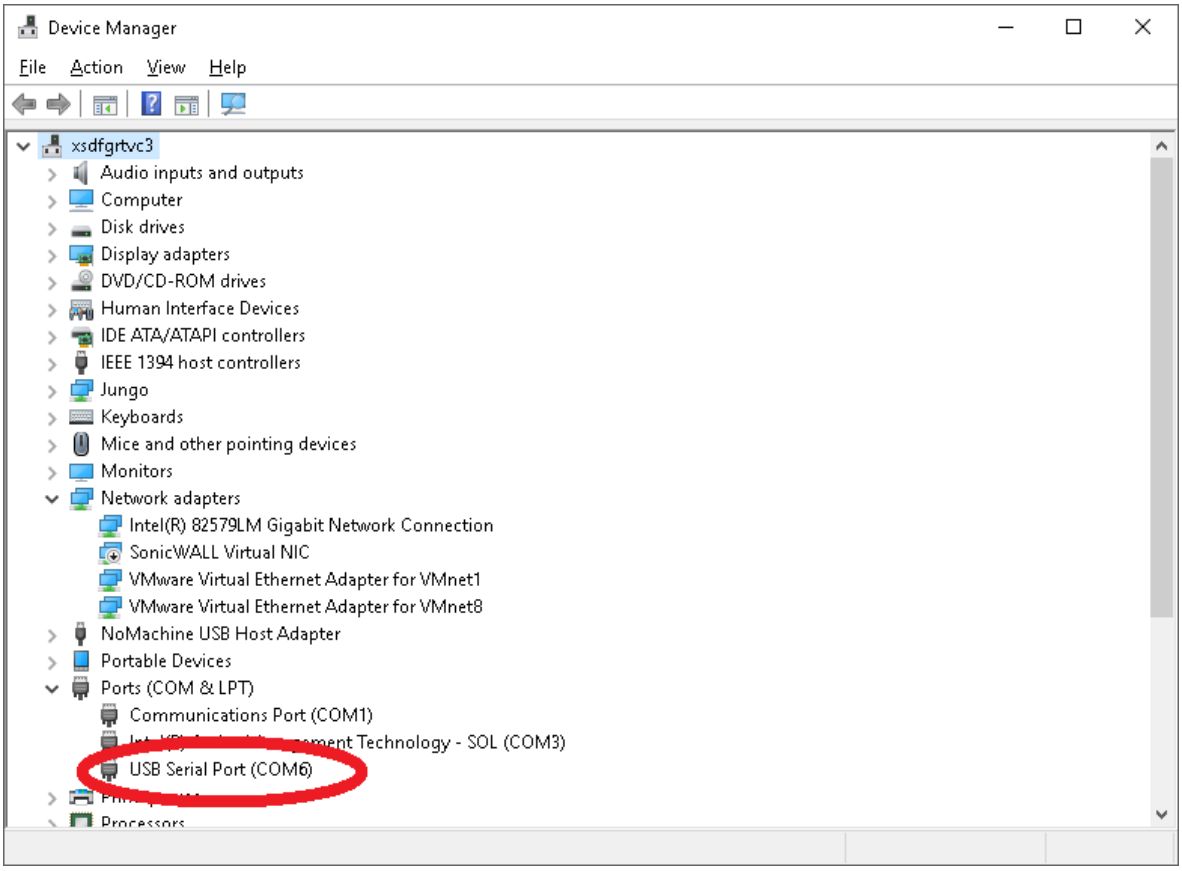

Figure 3-7 USB Serial Port in Windows Device Manager

Take the MachXO3D Breakout board for example. To select the board name as "MachXO3D Breakout", click the menu Tool / Board. If the previous steps were done right, you should see the name "MachXO3D Breakout" in PulseRain RISC-V (Reindeer), as illustrated in Figure 3-8.

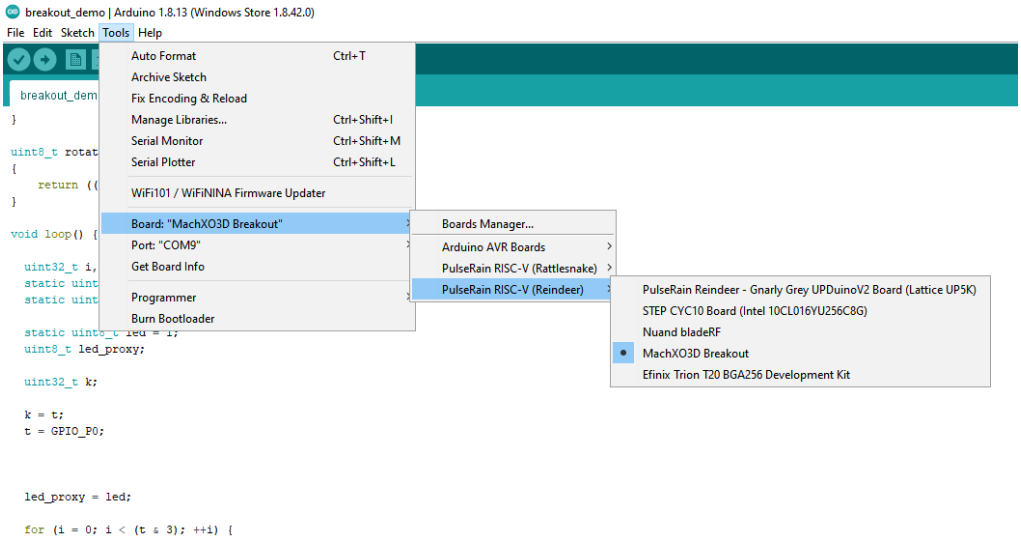

Figure 3-8 Arduino IDE, Select MachXO3D Breakout

Now you can start to write some sketches, you can open the demo Sketch provided in the GitHub repository Reindeer_MachXO3D/sketch/breakout_demo.

After opening the demo Sketch in Arduino IDE, you can type in "ctrl-U" (or menu Sketch/Upload) to compile and upload the sketch to the Breakout Board. And it is also recommended to turn on the option of "Show Verbose Output" in Preferences Dialogue (Menu File / Preferences), as illustrated in Figure 3-9. In this way, the path of the .hex file can be located through the verbose output.

Also, you can type in “Ctrl-Shift-M” (or menu Tools / Serial Monitor), as illustrated in Figure 3-10, to see the output of the COM port (Please set baud rate to 115, 200)

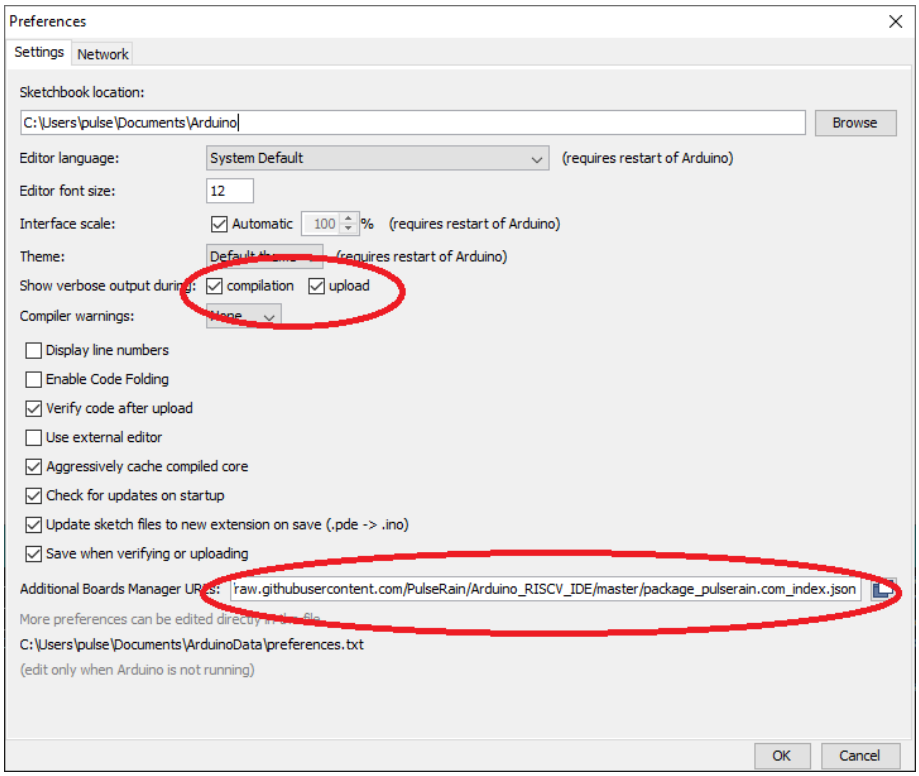

Figure 3-9 Turn on Verbose Option

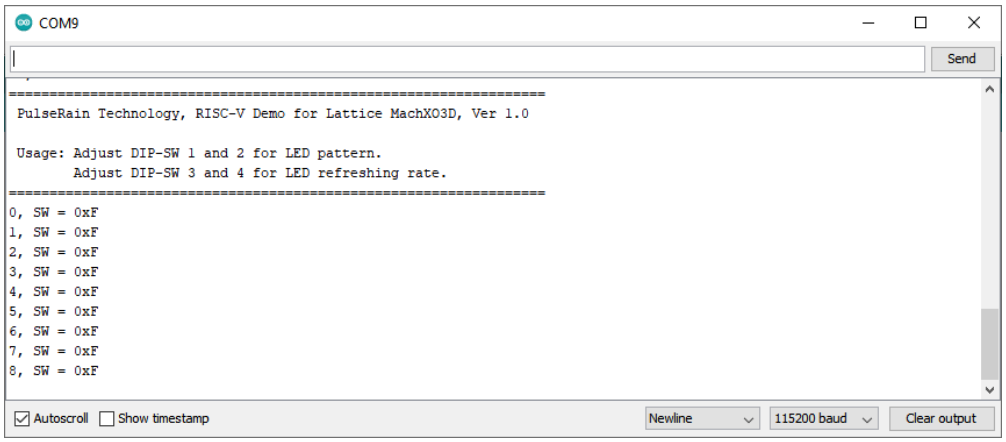

Figure 3-10 Arduino IDE, COM Port output