Riscduino

Release 0.1

Dinesh Annayya

Jun 24, 2022

CONTENTS

CHAPTER

ONE

INTRODUCTION

Riscduino is a Open source, 32 bit RISC V based SOC design targetted to pin compatible with arudino platform. This project uses only open source tool set for RTL to GDS implementations. The SOC follows openroad/openlane flow and development environment is compatible with efabless/carvel MPW methodology.

The Github Repo could be found here and database include all the RTL, Verification and Silicon implementation scripts. * Riscdino Single Core database * Riscdino Dual Core database * Riscdino Quad Core database

The documentation contains the following chapters:

- *Description* contains the general information about the Riscduino SoC,
- getting-started contains the general information about how to use the Riscduino SoC,
- tool-versioning contains the tool versions prefered for usage with the current Riscduino SoC,
- *quick start guide* contains a guide on how to get quickly started with using Riscduino SoC without many details,
- riscduino-with-openlane contains information on how to build your user project with OpenLANE inside the Riscduino SoC,
- *MPW Shuttle* contains information about riscduino project in different MPW shuttle
- *Simulation* contains information on how to simulate,
- *Pinout description* describes the pinout of the SoC,
- *RISCV* describes the RISCV configuration,
- qspi describes the SPI configuration,
- uart describes the UART interface,
- usb1.1 describes the USB1.1 host interface,
- memory-mapped-register lists the memory mapped registers by address,
- references contains list of references,
- further-work lists things to be added to the documentation.

1.1 Description

This section provides basic description of the Riscduino SoC.

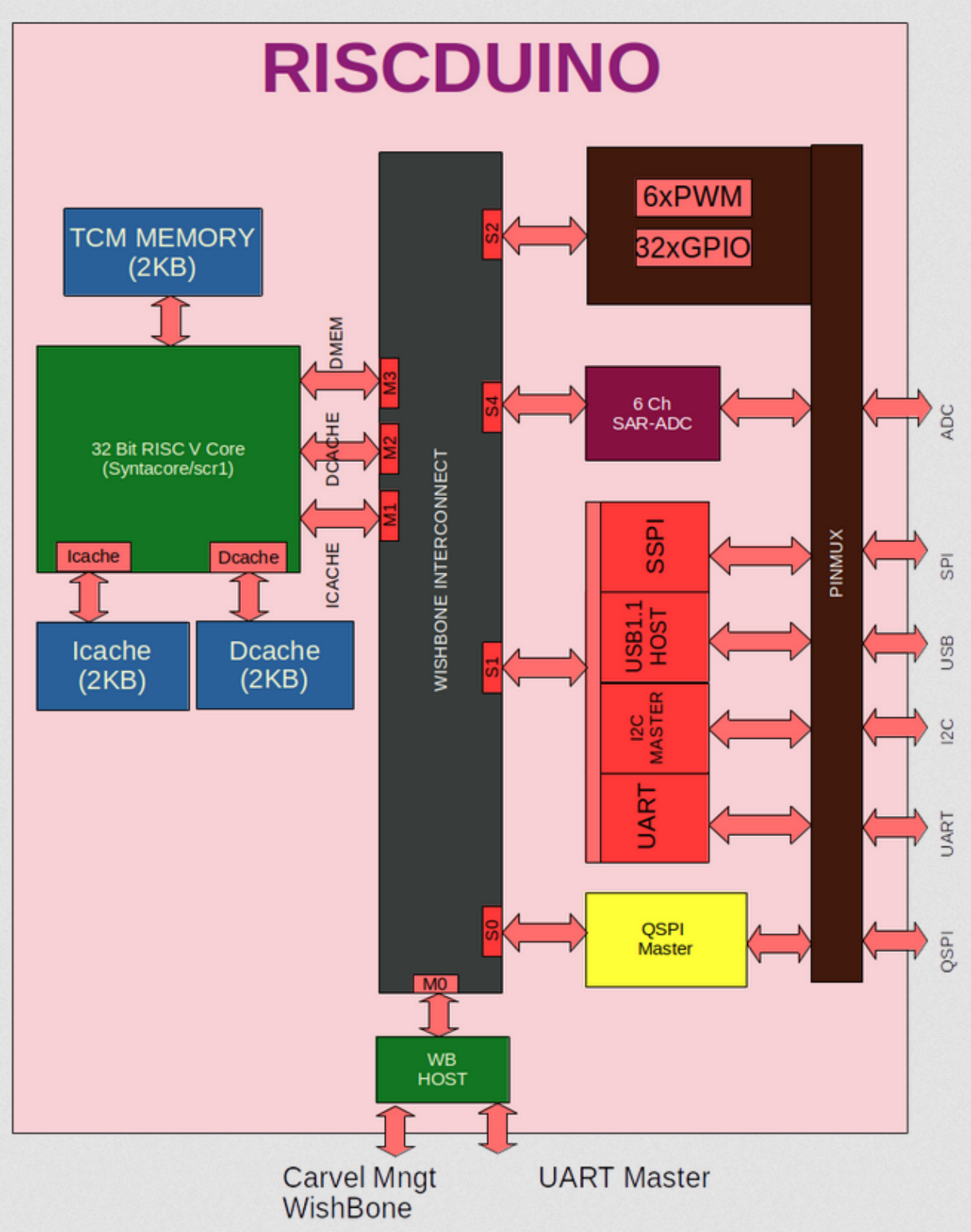
1.1.1 Block Diagram

1.1.2 Key features

- Open sourced under Apache-2.0 License (see LICENSE file) - unrestricted commercial use allowed.
- 32 Bit RISC-V core
- 2KB SRAM for instruction cache
- 2KB SRAM for data cache
- 2KB SRAM for Tightly coupled memory - For Data Memory
- Quad SPI Master
- UART with 16Byte FIFO
- USB 1.1 Host
- I2C Master
- UART Master
- Simple SPI Master
- 6 Channel ADC (in Progress)
- 6 PWM
- 3 Timer (16 Bit), 1us/1ms/1second resolution
- Pin Compatbible to arudino uno
- Wishbone compatible design
- Written in System Verilog
- **Open-source tool set**
  - simulation - iverilog
  - synthesis - yosys
  - backend/sta - openlane tool set
- Verification suite provided.

1.1.3 License

The Riscduino is an open-source design, licensed under the terms of Apache 2.0.


RISCDUINO
TCM MEMORY
(2KB)
32 Bit RISC V Core
(Syntacore/scr1)
Icache
Dcache
Icache
(2KB)
Dcache
(2KB)
DMEM
DCACHE
ICACHE
M3
M2
M1
WISHBONE INTERCONNECT
S2
S4
S1
S0
M0
6xPWM
32xGPIO
6 Ch
SAR-ADC
SSPI
USB1.1
HOST
I2C
MASTER
UART
QSPI
Master
PINMUX
ADC
SPI
USB
I2C
UART
QSPI
WB
HOST
Carvel Mngt
WishBone
UART Master

1.1.4 Repository

The complete chip design may be obtained from the git repository located at GitHub *Riscdino database <https://github.com/dineshannayya/riscduino/>*

1.1.5 Process

The Riscduino chip is tagetted to part of efabless MPW Shuttle and in SkyWater 0.13um CMOS technology, with process specifications and data at GitHub google/skywater-pdk repository.

1.2 quick start guide

This section describes how to simulate and .gds generarion with the Riscduino repository

1.2.1 Prerequisites

- Docker

If you have followed the Getting Started Guide you should have all of these installed. To proceed make sure, that your environment variables are set correctly:

```
git clone https://github.com/efabless/caravel-lite.git
cd caravel-lite
make install_mcw
```

```
export OPENLANE_IMAGE_NAME=riscduino/openlane:<Check the mpw version on project mpw4/
↪mpw5/latest>
export OPENLANE_TAG=<Check the mpw version on project mpw4/mpw5/latest>
export CARAVEL_ROOT=<Set the caravel lite root>
export MCW_ROOT=$CARAVEL_ROOT/mgmt_core_wrapper
```

1.2.2 Building individual design

To build your design go into `openlane` and run make with your design name as a target:

```
cd openlane
make <design>
```

This will run your design throught the OpenLANE workflow and if successfull produce a `.gds` file of your project. The subdirectory `runs/<design>` will be created in your designs folder, which contains the results of the run. The following result files in `runs/<design>/` are important:

- `<design>/runs/<design>/reports/final_summary_report.csv`: Contains the results of the run including violations
- `<design>/runs/<design>/results/magic/<design>.lef`
- `<design>/runs/<design>/results/magic/<design>.gds`

The `.gds` and `.lef` files can also be found in the `gds` and `lef` directories on the top level of the repository.

1.3 MPW Shuttle

This section provides details on Riscduino submission in different MPW shuttle

Table 1: shuttle

MPW Shuttle	Tape-out Date	Project Name	Sub-mission Link	Silicon Status	Project details
MPW-2	November 15, 2021	YiFive	Link	Waiting	Single Riscv core without cache+SDRAM, Not compatible with Arudino pins
MPW-3	November 15, 2021	riscduino	Link	Waiting	Single Riscv core without cache
MPW-4	December 31, 2021	riscduino-R1	Link	Waiting	Single Riscv core with cache + SRAM
MPW-5	March 21, 2022	riscduino-SCore (S2)	Link	Waiting	Single Riscv core with cache + DFFRAM
MPW-5	March 21, 2022	riscduino-Dual Core (D0)	Link	Waiting	Dual Riscv core with cache + SRAM
MPW-5	March 21, 2022	riscduino-Quad Core (Q0)	Link	Waiting	Quad Riscv core with cache + SRAM

1.4 Pinout description

This section describes lists the pinout for the SoC, and provides the description for pins.

1.4.1 Pinout Diagram

1.4.2 Pinout Map

Table 2: Pinout

ATMGA328 Pin No	Functionality	Arudino Pin Name	Carvel Pin Mapping
1	PC6/RESET		mprj_io[0]
2	PD0/RXD	D0	mprj_io[1]
3	PD1/TXD	D1	mprj_io[2]
4	PD2/INT0	D2	mprj_io[3]
5	PD3/INT1/OC2B(PWM0)	D3	mprj_io[4]
6	PD4	D4	mprj_io[5]
7	VCC		
8	GND		
9	PB6/XTAL1/TOSC1		mprj_io[6]
10	PB7/XTAL2/TOSC2		mprj_io[7]
11	PD5/OC0B(PWM1)/T1	D5	mprj_io[8]
12	PD6/OC0A(PWM2)/AIN0	D6	mprj_io[9]/analog_io[2]
13	PD7/A1N1	D7	mprj_io[10]/analog_io[3]
14	PB0/CLKO/ICP1	D8	mprj_io[11]
15	PB1/OC1A(PWM3)	D9	mprj_io[12]

continues on next page

Table 2 – continued from previous page

ATMGA328 Pin No	Functionality	Arudino Pin Name	Carvel Pin Mapping
16	PB2/SS/OC1B(PWM4)	D10	mprj_io[13]
17	PB3/MOSI/OC2A(PWM5)	D11	mprj_io[14]
18	PB4/MISO	D12	mprj_io[15]
19	PB5/SCK	D13	mprj_io[16]
20	AVCC		
21	AREF		analog_io[10]
22	GND		
23	PC0/ADC0	A0	mprj_io[18]/analog_io[11]
24	PC1/ADC1	A1	mprj_io[19]
25	PC2/ADC2	A2	mprj_io[20]
26	PC3/ADC3	A3	mprj_io[21]
27	PC4/ADC4/SDA	A4	mprj_io[22]
28	PC5/ADC5/SCL	A5	mprj_io[23]
Sflash	sflash_sck		mprj_io[24]
Sflash	sflash_ss[0]		mprj_io[25]
Sflash	sflash_ss[1]		mprj_io[26]
Sflash	sflash_ss[2]		mprj_io[27]
Sflash	sflash_ss[3]		mprj_io[28]
Sflash	sflash_io[0]		mprj_io[29]
Sflash	sflash_io[1]		mprj_io[30]
Sflash	sflash_io[2]		mprj_io[31]
Sflash	sflash_io[3]		mprj_io[32]
UARTM	uartm_rxd		mprj_io[34]
UARTM	uartm_txd		mprj_io[35]
USB HOST	usb_dp		mprj_io[36]
USB HOST	usb_dn		mprj_io[37]

1.5 Simulation

This section provides basic description of how to simulate the Riscduino SoC.

Basic Usage command : make verify-<test directory name> SIM=<RTL/GL> DUMP=<ON/OFF>

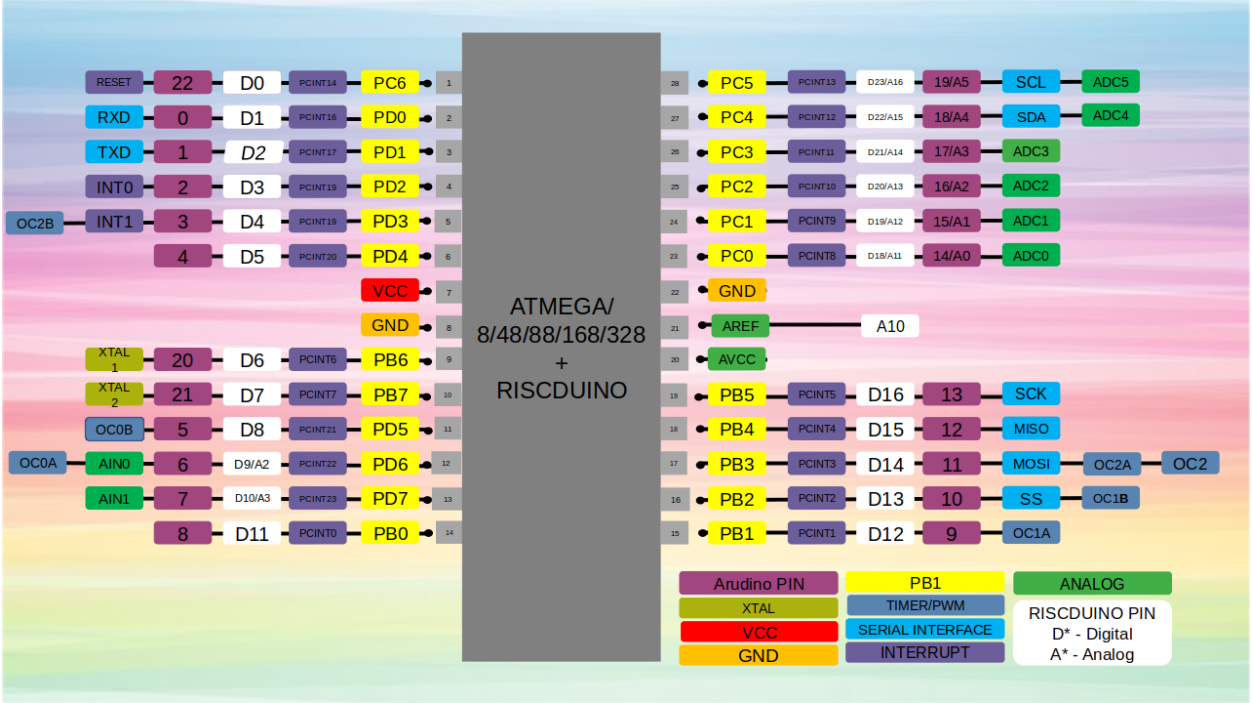
SIM=RTL - For RTL Simulation (Default). SIM=GL - For Gate Level Simulation.

DUMP=ON - For simulation with waveform dump enabled. DUMP=OFF - For simulation with waveform dump disabled (Default)

1.5.1 Dut Test case

Below test-case run faster as it uses only user_project_wrapper and wishbone to control the test case

- make verify-user_basic
- make verify-user_uart
- make verify-user_uart1
- make verify-user_risc_boot
- make verify-user_sspi



- make verify-user_i2cm
- make verify-user_uart_master
- make verify-user_pwm
- make verify-user_gpio
- make verify-user_timer
- make verify-user_qspi
- make verify-user_usb

1.5.2 Core RISC-V Regression

Below test is validate the RISC v compliance

- make verify-riscv_regress

  This test has has following sub tests:
  riscv_isa riscv_compliance isr_sample coremark dhrystone21 hello

1.5.3 Caravel+ user_project_wrapper Test case

Below Test run take more time as it's full chip run with caravel + user_project_wrapper

- make verify-wb_port
- make verify-risc_boot
- make verify-uart_master

If you want to dump, then you can add DUMP=ON swicth

1.5.4 Test Case Details

Below test-case run faster as it uses only user_project_wrapper and wishbone to control the test case

1. user_basic AIM: To Validate the Test the different clock selection option for wishbone , risc core, usb and validate device signature
2. user_uart AIM: Test to validation uart rx to tx loop back. TEST SEQUENCE: 40 random char are sent from TB to DUT and RISCV core does the hardware loop back,i.e UART-RX to UART-TX
3. uart_risc_boot AIM: Test the RISC CORE boot TEST SEQUENCE:
    1. Make sure that core hex file loaded into spi memory
    2. Though wishbone sequence wake-up the core.
    3. core boots from spi
    4. core write 6 Pimux register with some signature
    5. After some delay, testbench validation these signature through wishbone
4. **user_spi**
   AIM: Test SPI DDR,QUAD,DUAL,SINGLE mode. BACKGROUND:

   SINGLE - This is single pin tx/rx transaction and seperate pin for Tx (spi_sdo[0]) and Rx (spi_sdi[0]) pin, data valid for one spi clock DUAL - This uses two pin for tx/rx transaction, Tx: spi_sdo[1:0] & Rx: spi_sdi[1:0], data valid for one spi clock QUAD - This uses four pins for tx/rx transaction, Tx: spi_sdo[3:0] & Rx : spi_sdi[3:0], Data valid for one spi clock DDR - This uses four pins for tx/rx transaction, Tx: spi_sdo[3:0] & Rx : spi_sdi[3:0], Data valid for half spi clock

   TEST SEQUENCE:
   Test SPI SRAM Memory through Indirect Access Test SPI SRAM memory through Direct Access TEST SPI FLASH Memory in DDR Mode TEST SPI FLASH Memory in QUAD Mode TEST SPI FLASH Memory in DUAL Mode TEST SPI FLASH Memory in SINGLE Mode
5. **user_i2cm**
   AIM: Test the I2C Interface TEST SEQUENCE:

   I2C Write and Read access
6. **user_uart_master**
   AIM: Test the uart master port TEST SEQUENCE:

   Configure the uart master baudrate through la_in pins From TB uart master send the Write command to wake the device Write some general purpose register with uart master and read back and validate

7. **riscv_regress**
   test is validate the RISC v compliance and this have below sub tests

   riscv_isa riscv_compliance isr_sample coremark dhrystone21 hello

Below Test run take more time as it's full chip run with caravel + user_project_wrapper 1. wb_port

AIM: Test user_project_wrapper through caravel wishbone interface TEST SEQUENCE:

Boot through caravel riscv core Write some general purpose register user project wrapper and read back and validate

2. **risc_boot**
   AIM: Boot the User RISC core through caravel core TEST SEQUENCE:

   Boot through caravel riscv core Wake-up the user riscv core User risc core write some general porpose register with signtaure Read back through caravel riscv core and validate the signature

3. **uart_master**
   AIM: Test the uart master port from caravel core TEST SEQUENCE:

   Configure the uart master baudrate through la_in pins using caravel core From TB uart master send the Write command to wake the device

1.6 RISCV

Riscduino SOC Integrated 32 Bits RISC V core. Initial version of Single core RISC-V core is picked from Syntacore SCR1 (https://github.com/syntacore/scr1)

1.6.1 core customization

Following Design changes are done on the basic version of syntacore RISC core

- Some of the sv syntex are changed to standard verilog format to make compatibile with opensource tool iverilog & yosys
- Instruction Request are changed from Single word to 4 Word Burst
- Multiplication and Divsion are changed to improve timing
- Additional pipe line stages added to improve the RISC timing closure near to 50Mhz
- 2KB instruction cache
- 2KB data cache
- Additional router are added towards instruction cache
- Additional router are added towards data cache
- Modified AXI/AHB interface to wishbone interface for instruction and data memory interface

1.6.2 Features

- RV32I or RV32E ISA base + optional RVM and RVC standard extensions
- Machine privilege mode only
- 2 to 5 stage pipeline
- 2KB icache
- 2KB dcache
- Optional Integrated Programmable Interrupt Controller with 16 IRQ lines
- Optional RISC-V Debug subsystem with JTAG interface
- Optional on-chip Tightly-Coupled Memory

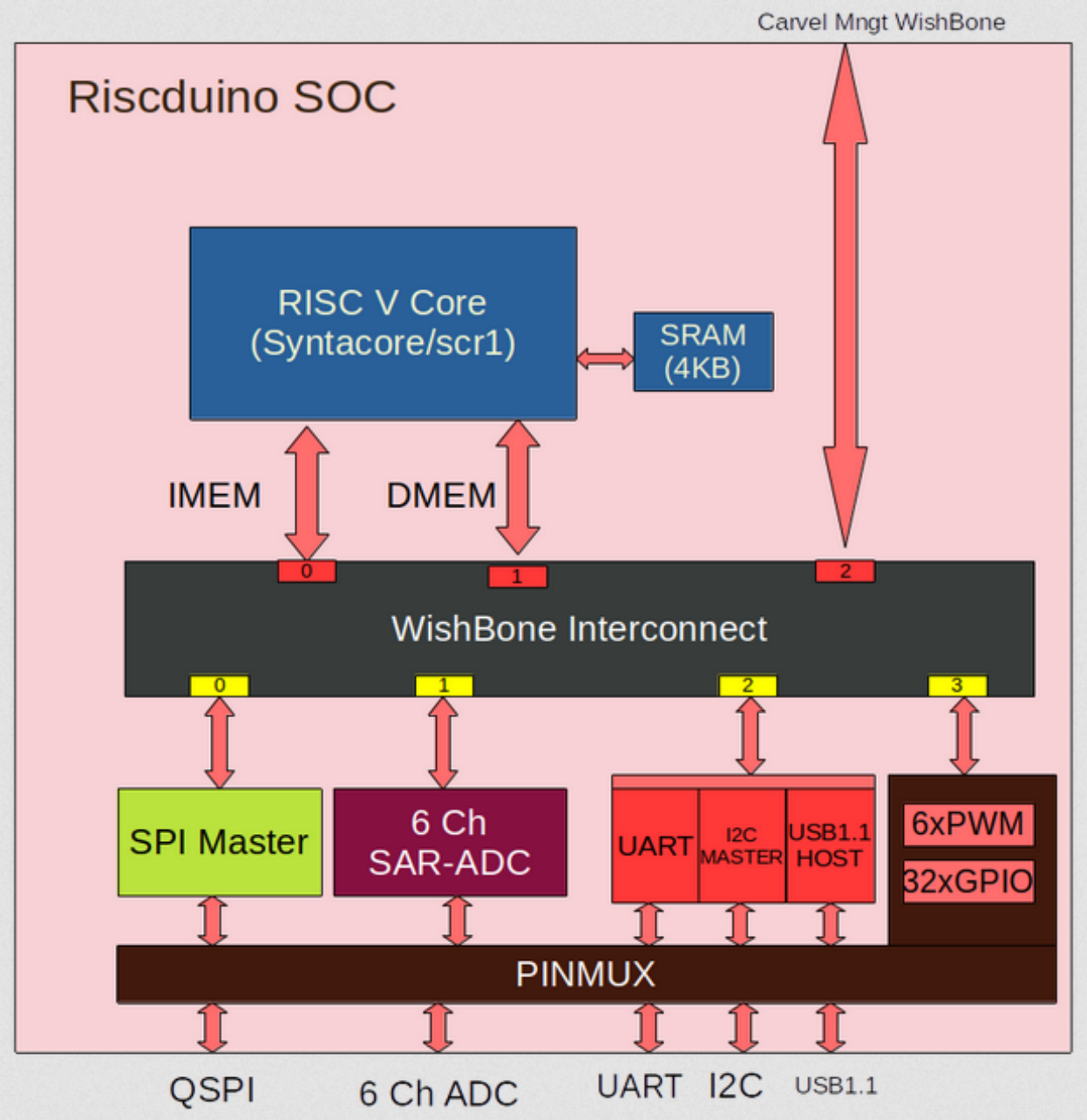
1.6.3 Block Diagram

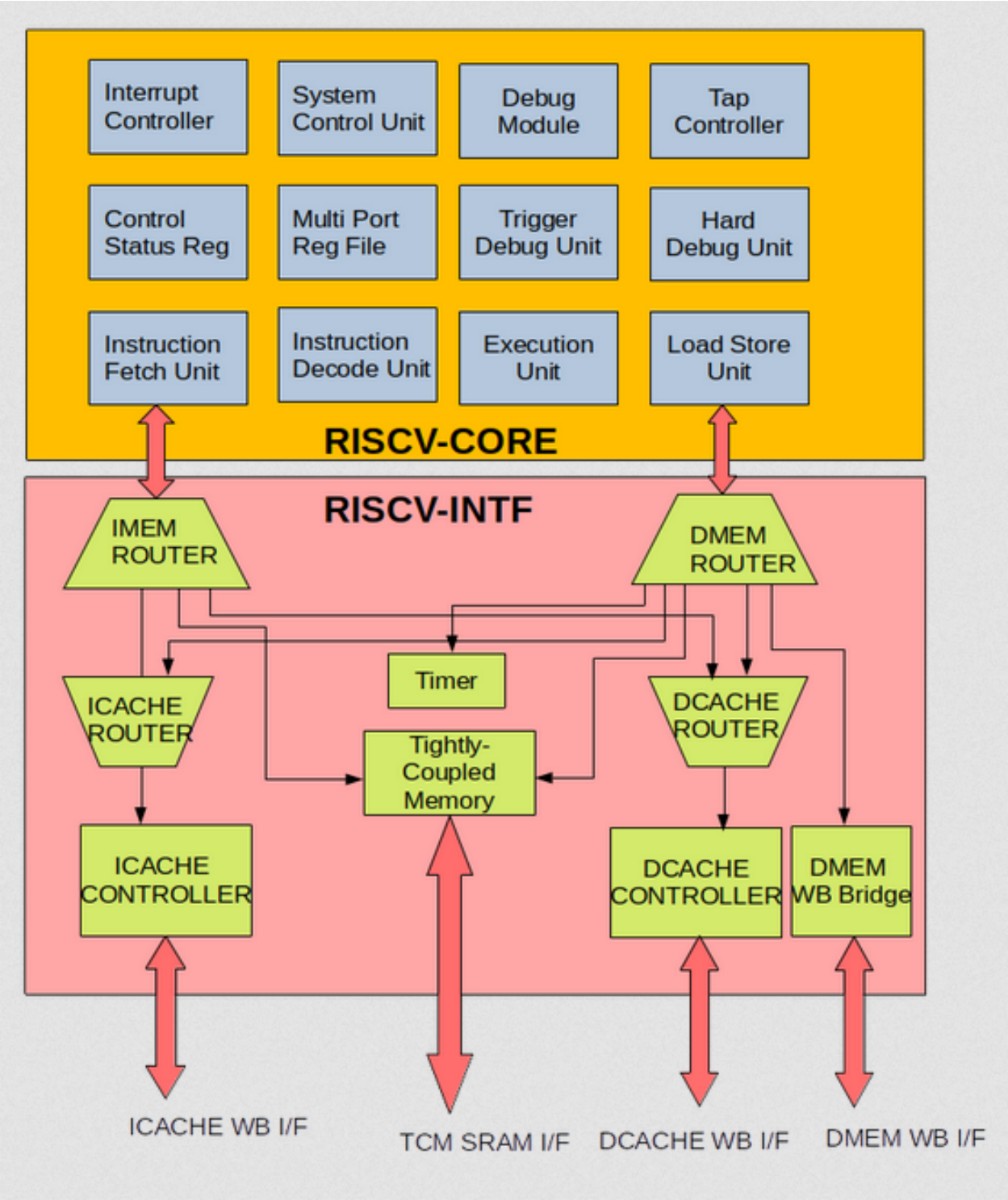
Following RISCV core configuration are been tried in different MPW Shuttle

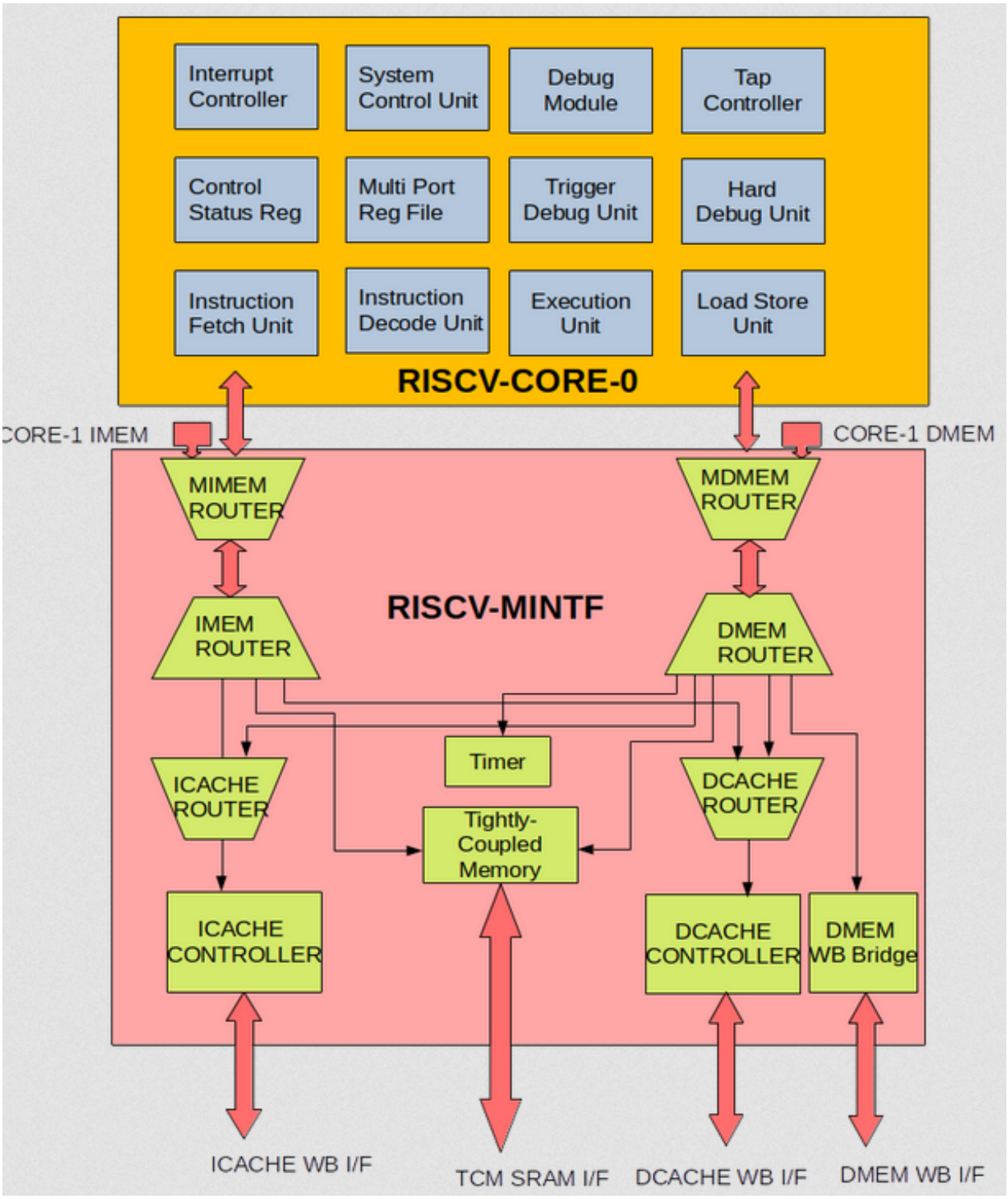
1.6.4 Riscv Single core

1.6.5 Riscv Single core + cache

1.6.6 Riscv Two core + cache


Carvel Mngt WishBone
Riscduino SOC
RISC V Core
(Syntacore/scr1)
SRAM
(4KB)
IMEM
DMEM
0
1
2
WishBone Interconnect
0
1
2
3
SPI Master
6 Ch
SAR-ADC
UART
I2C
MASTER
USB1.1
HOST
6xPWM
32xGPIO
PINMUX
QSPI
6 Ch ADC
UART
I2C
USB1.1


Interrupt Controller
System Control Unit
Debug Module
Tap Controller
Control Status Reg
Multi Port Reg File
Trigger Debug Unit
Hard Debug Unit
Instruction Fetch Unit
Instruction Decode Unit
Execution Unit
Load Store Unit
RISCV-CORE
RISCV-INTF
IMEM ROUTER
DMEM ROUTER
Timer
ICACHE ROUTER
DCACHE ROUTER
Tightly-Coupled Memory
ICACHE CONTROLLER
DCACHE CONTROLLER
DMEM WB Bridge
ICACHE WB I/F
TCM SRAM I/F
DCACHE WB I/F
DMEM WB I/F


Interrupt Controller
System Control Unit
Debug Module
Tap Controller
Control Status Reg
Multi Port Reg File
Trigger Debug Unit
Hard Debug Unit
Instruction Fetch Unit
Instruction Decode Unit
Execution Unit
Load Store Unit
RISCV-CORE-0
CORE-1 IMEM
CORE-1 DMEM
MIMEM ROUTER
MDMEM ROUTER
RISCV-MINTF
IMEM ROUTER
DMEM ROUTER
Timer
ICACHE ROUTER
DCACHE ROUTER
Tightly-Coupled Memory
ICACHE CONTROLLER
DCACHE CONTROLLER
DMEM WB Bridge
ICACHE WB I/F
TCM SRAM I/F
DCACHE WB I/F
DMEM WB I/F