

Processing Unit verified with UVM

QueenField

1. INTRODUCTION

1.1. OPEN SOURCE PHILOSOPHY

For Windows Users!

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

type:

```
sudo apt update  
sudo apt upgrade
```

1.1.1. Open Source Hardware

1.1.1.1. MSP430 Processing Unit

1.1.1.2. OpenRISC Processing Unit

1.1.1.3. RISC-V Processing Unit

1.2. Open Source Software

1.2.2.1. MSP430 GNU Compiler Collection

1.2.2.2. OpenRISC GNU Compiler Collection

1.2.2.3. RISC-V GNU Compiler Collection

1.2. RISC-V ISA

1.2.1. ISA Bases

1.2.1.1. RISC-V 32

1.2.1.2. RISC-V 64

1.2.1.3. RISC-V 128

1.2.2. ISA Extensions

1.2.2.1. Base Integer Instruction Set

RV32I : Base Integer Instruction Set (32 bit)

RV32I	31:25	24:20	19:15	14:12	11:7	6:0
LUI RD, IMM	IIIIII	IIII	IIII	III	RD4:0	0110111
AUPIC RD, IMM	IIIIII	IIII	IIII	III	RD4:0	0010111
JAL RD, IMM	IIIIII	IIII	IIII	III	RD4:0	1101111
JALR RD,RS1,IMM	IIIIII	IIII	RS14:0	000	RD4:0	1101111
BEQ RS1,RS2,IMM	IIIIII	RS24:0	RS14:0	000	IIII	1100011
BNE RS1,RS2,IMM	IIIIII	RS24:0	RS14:0	001	IIII	1100011
BLT RS1,RS2,IMM	IIIIII	RS24:0	RS14:0	100	IIII	1100011
BGE RS1,RS2,IMM	IIIIII	RS24:0	RS14:0	101	IIII	1100011
BLTU RS1,RS2,IMM	IIIIII	RS24:0	RS14:0	110	IIII	1100011
BGEU RS1,RS2,IMM	IIIIII	RS24:0	RS14:0	111	IIII	1100011
LB RD, RS1	IIIIII	IIII	RS14:0	000	RD4:0	0000011
LH RD, RS1	IIIIII	IIII	RS14:0	001	RD4:0	0000011
LW RD, RS1	IIIIII	IIII	RS14:0	010	RD4:0	0000011
LBU RD, RS1	IIIIII	IIII	RS14:0	100	RD4:0	0000011
LHU RD, RS1	IIIIII	IIII	RS14:0	101	RD4:0	0000011
SB RS2,RS1	IIIIII	RS24:0	RS14:0	000	IIII	0100011
SH RS2,RS1	IIIIII	RS24:0	RS14:0	001	IIII	0100011
SW RS2,RS1	IIIIII	RS24:0	RS14:0	010	IIII	0100011
ADDI RD,RS1,IMM	IIIIII	IIII	RS14:0	000	RD4:0	0010011
SLTI RD,RS1,IMM	IIIIII	IIII	RS14:0	010	RD4:0	0010011
SLTIU RD,RS1,IMM	IIIIII	IIII	RS14:0	011	RD4:0	0010011
XORI RD,RS1,IMM	IIIIII	IIII	RS14:0	100	RD4:0	0010011
ORI RD,RS1,IMM	IIIIII	IIII	RS14:0	110	RD4:0	0010011
ANDI RD,RS1,IMM	IIIIII	IIII	RS14:0	111	RD4:0	0010011
SLLI RD,RS1,IMM	0000000	IIII	RS14:0	001	RD4:0	0010011
SRLI RD,RS1,IMM	0000000	IIII	RS14:0	101	RD4:0	0010011
SRAI RD,RS1,IMM	0100000	IIII	RS14:0	101	RD4:0	0010011
ADD RD,RS1,RS2	0000000	RS24:0	RS14:0	000	RD4:0	0110011
SUB RD,RS1,RS2	0100000	RS24:0	RS14:0	000	RD4:0	0110011
SLL RD,RS1,RS2	0000000	RS24:0	RS14:0	001	RD4:0	0110011
SLT RD,RS1,RS2	0000000	RS24:0	RS14:0	010	RD4:0	0110011
SLTU RD,RS1,RS2	0000000	RS24:0	RS14:0	011	RD4:0	0110011
XOR RD,RS1,RS2	0000000	RS24:0	RS14:0	100	RD4:0	0110011
SRL RD,RS1,RS2	0000000	RS24:0	RS14:0	101	RD4:0	0110011
SRA RD,RS1,RS2	0100000	RS24:0	RS14:0	101	RD4:0	0110011
OR RD,RS1,RS2	0000000	RS24:0	RS14:0	110	RD4:0	0110011
AND RD,RS1,RS2	0000000	RS24:0	RS14:0	111	RD4:0	0110011
FENCE PRED,SUCC	0000PPP	PSSSS	00000	000	00000	0001111
FENCE.I	0000P00	00000	00000	001	00000	0001111

RV64I : Base Integer Instruction Set (64 bit)

RV64I	31:25	24:20	19:15	14:12	11:7	6:0
LWU RD, RS1	IIIIII	IIII	RS14:0	110	RD4:0	0000011
LD RD, RS1	IIIIII	IIII	RS14:0	011	RD4:0	0000011
SD RD, RS1,RS2	IIIIII	RS24:0	RS14:0	011	IIII	0000011
SLLI RD, RS1,IMM	0000000	IIII	RS14:0	001	RD4:0	0010011
SRLI RD, RS1,IMM	0000000	IIII	RS14:0	001	RD4:0	0010011
SRAI RD, RS1,IMM	0100000	IIII	RS14:0	001	RD4:0	0010011
ADDIW RD, RS1	IIIIII	IIII	RS14:0	000	RD4:0	0011011
SLLIW RD, RS1	0000000	IIII	RS14:0	001	RD4:0	0011011

RV64I	31:25	24:20	19:15	14:12	11:7	6:0
SRLIW RD, RS1	0000000	IIII	RS14:0	101	RD4:0	0011011
SRAIW RD, RS1	0100000	IIII	RS14:0	101	RD4:0	0011011
ADDW RD, RS1,RS2	0000000	RS24:0	RS14:0	000	RD4:0	0111011
SUBW RD, RS1,RS2	0100000	RS24:0	RS14:0	000	RD4:0	0111011
SLIW RD, RS1,RS2	0000000	RS24:0	RS14:0	001	RD4:0	0111011
SRLW RD, RS1,RS2	0000000	RS24:0	RS14:0	101	RD4:0	0111011
SRAW RD, RS1,RS2	0100000	RS24:0	RS14:0	101	RD4:0	0111011

1.2.2.2. Standard Extension for Integer Multiply and Divide

RV32M : Standard Extension for Integer Multiply and Divide (32 bit)

RV32M	31:25	24:20	19:15	14:12	11:7	6:0
MUL RD,RS1,RS2	0000001	RS24:0	RS14:0	000	RD4:0	0110011
MULH RD,RS1,RS2	0000001	RS24:0	RS14:0	001	RD4:0	0110011
MULHSU RD,RS1,RS2	0000001	RS24:0	RS14:0	010	RD4:0	0110011
MULHU RD,RS1,RS2	0000001	RS24:0	RS14:0	011	RD4:0	0110011
DIV RD,RS1,RS2	0000001	RS24:0	RS14:0	100	RD4:0	0110011
DIVU RD,RS1,RS2	0000001	RS24:0	RS14:0	101	RD4:0	0110011
REM RD,RS1,RS2	0000001	RS24:0	RS14:0	110	RD4:0	0110011
REMU RD,RS1,RS2	0000001	RS24:0	RS14:0	111	RD4:0	0110011

Standard Extension for Integer Multiply and Divide (64 bit)

RV64M	31:25	24:20	19:15	14:12	11:7	6:0
MULW RD,RS1,RS2	0000001	RS24:0	RS14:0	000	RD4:0	0111011
DIVW RD,RS1,RS2	0000001	RS24:0	RS14:0	100	RD4:0	0111011
DIVUW RD,RS1,RS2	0000001	RS24:0	RS14:0	101	RD4:0	0111011
REMW RD,RS1,RS2	0000001	RS24:0	RS14:0	110	RD4:0	0111011
REMUW RD,RS1,RS2	0000001	RS24:0	RS14:0	111	RD4:0	0111011

1.2.2.3. Standard Extension for Atomic Instructions

RV32A : Standard Extension for Atomic Instructions (32 bit)

RV32A	31:25	24:20	19:15	14:12	11:7	6:0
LR.W AQRL,RD,RS1	00010AQRL	00000	RS14:0	010	RD4:0	0101111
SC.W AQRL,RD,RS2,RS1	00011AQRL	RS24:0	RS14:0	010	RD4:0	0101111
AMOSWAP.W AQRL,RD,RS2,RS1	00001AQRL	RS24:0	RS14:0	010	RD4:0	0101111
AMOSADD.W AQRL,RD,RS2,RS1	00000AQRL	RS24:0	RS14:0	010	RD4:0	0101111
AMOSXOR.W AQRL,RD,RS2,RS1	00100AQRL	RS24:0	RS14:0	010	RD4:0	0101111
AMOOR.W AQRL,RD,RS2,RS1	01000AQRL	RS24:0	RS14:0	010	RD4:0	0101111
AMOAMD.W AQRL,RD,RS2,RS1	01100AQRL	RS24:0	RS14:0	010	RD4:0	0101111
AMOMIN.W AQRL,RD,RS2,RS1	10000AQRL	RS24:0	RS14:0	010	RD4:0	0101111
AMOMAX.W AQRL,RD,RS2,RS1	10100AQRL	RS24:0	RS14:0	010	RD4:0	0101111
AMOMINU.W AQRL,RD,RS2,RS1	11000AQRL	RS24:0	RS14:0	010	RD4:0	0101111
AMOMAXU.W AQRL,RD,RS2,RS1	11100AQRL	RS24:0	RS14:0	010	RD4:0	0101111

RV64A : Standard Extension for Atomic Instructions (64 bit)

RV64A	31:25	24:20	19:15	14:12	11:7	6:0
LR.D AQRL,RD,RS1	00010AQRL	00000	RS14:0	011	RD4:0	0101111
SC.D AQRL,RD,RS2,RS1	00011AQRL	RS24:0	RS14:0	011	RD4:0	0101111
AMOSWAP.D AQRL,RD,RS2,RS1	00001AQRL	RS24:0	RS14:0	011	RD4:0	0101111
AMOSADD.D AQRL,RD,RS2,RS1	00000AQRL	RS24:0	RS14:0	011	RD4:0	0101111
AMOSXOR.D AQRL,RD,RS2,RS1	00100AQRL	RS24:0	RS14:0	011	RD4:0	0101111
AMOOR.D AQRL,RD,RS2,RS1	01000AQRL	RS24:0	RS14:0	011	RD4:0	0101111
AMOAMD.D AQRL,RD,RS2,RS1	01100AQRL	RS24:0	RS14:0	011	RD4:0	0101111
AMOMIN.D AQRL,RD,RS2,RS1	10000AQRL	RS24:0	RS14:0	011	RD4:0	0101111
AMOMAX.D AQRL,RD,RS2,RS1	10100AQRL	RS24:0	RS14:0	011	RD4:0	0101111
AMOMINU.D AQRL,RD,RS2,RS1	11000AQRL	RS24:0	RS14:0	011	RD4:0	0101111
AMOMAXU.D AQRL,RD,RS2,RS1	11100AQRL	RS24:0	RS14:0	011	RD4:0	0101111

1.2.2.4. Standard Extension for Single-Precision Floating-Point**RV32F : Standard Extension for Single-Precision Floating-Point (32 bit)**

RV32F	31:25	24:20	19:15	14:12	11:7	6:0
FLW FRD,RS1	IIIIII	IIII	FRS1	010	FRD	0000111
FSW FRS2,RS1	IIIIII	FRS2	FRS1	010	IIII	0100111
FMADD.S RM,FRD,FRS1,FRS2,FRS3	FRS3_00	FRS2	FRS1	RM	FRD	1000011
FMSUB.S RM,FRD,FRS1,FRS2,FRS3	FRS3_00	FRS2	FRS1	RM	FRD	1000111
FNMSUB.S RM,FRD,FRS1,FRS2,FRS3	FRS3_00	FRS2	FRS1	RM	FRD	1001011
FNMADD.S RM,FRD,FRS1,FRS2,FRS3	FRS3_00	FRS2	FRS1	RM	FRD	1001111
FADD.S RM,FRD,FRS1,FRS2,FRS3	0000000	FRS2	FRS1	RM	FRD	1010011
FSUB.S RM,FRD,FRS1,FRS2,FRS3	0000100	FRS2	FRS1	RM	FRD	1010011
FMUL.S RM,FRD,FRS1,FRS2,FRS3	0001000	FRS2	FRS1	RM	FRD	1010011
FDIV.S RM,FRD,FRS1,FRS2,FRS3	0001100	FRS2	FRS1	RM	FRD	1010011
FSGNJ.S FRD,FRS1,FRS2	0010000	FRS2	FRS1	000	FRD	1010011
FSGNJS FRD,FRS1,FRS2	0010000	FRS2	FRS1	001	FRD	1010011
FSGNJXS FRD,FRS1,FRS2	0010000	FRS2	FRS1	010	FRD	1010011
FMIN.S FRD,FRS1,FRS2	0010100	FRS2	FRS1	000	FRD	1010011
FMAX.S FRD,FRS1,FRS2	0010100	FRS2	FRS1	001	FRD	1010011
FSQRT.S FRD,FRS1,FRS2	0101100	00000	FRS1	RM	FRD	1010011
FLE.S FRD,FRS1,FRS2	1010000	FRS2	FRS1	000	FRD	1010011
FLT.S FRD,FRS1,FRS2	1010000	FRS2	FRS1	001	FRD	1010011
FEQ.S FRD,FRS1,FRS2	1010000	FRS2	FRS1	010	FRD	1010011
FCVT.WS RM,RD,FRS1	1100000	00000	FRS1	RM	FRD	1010011
FCVT.WUS RM,RD,FRS1	1100000	00010	FRS1	RM	FRD	1010011
FCVT.SW RM,RD,FRS1	1101000	00000	FRS1	RM	FRD	1010011
FCVT.SWU RM,RD,FRS1	1101000	00010	FRS1	RM	FRD	1010011
FMV.XS RD,FRS1	1110000	00000	FRS1	000	RD	1010011
FCLASS.S RD,FRS1	1110000	00000	FRS1	001	RD	1010011
FMV.SX RD,FRS1	1111000	00000	RS1	000	FRD	1010011

RV64F : Standard Extension for Single-Precision Floating-Point (64 bit)

RV64F	31:25	24:20	19:15	14:12	11:7	6:0
FCVT.L.S RM,RD,FRS1	1100000	00010	FRS1	RM	FRD	1010011
FCVT.LU.S RM,RD,FRS1	1100000	00011	FRS1	RM	FRD	1010011
FCVT.S.L RM,RD,FRS1	1101000	00010	FRS1	RM	FRD	1010011
FCVT.S.LU RM,RD,FRS1	1101000	00011	FRS1	RM	FRD	1010011

1.2.2.5. Standard Extension for Double-Precision Floating-Point

RV32D : Standard Extension for Double-Precision Floating-Point (32 bit)

RV32F	31:25	24:20	19:15	14:12	11:7	6:0
FLW FRD,RS1	IIIIII	IIII	FRS1	011	FRD	0000111
FSW FRS2,RS1	IIIIII	FRS2	FRS1	011	IIII	0100111
FMADD.D RM,FRD,FRS1,FRS2,FRS3	FRS3_01	FRS2	FRS1	RM	FRD	1000011
FMSUB.D RM,FRD,FRS1,FRS2,FRS3	FRS3_01	FRS2	FRS1	RM	FRD	1000111
FNMSUB.D RM,FRD,FRS1,FRS2,FRS3	FRS3_01	FRS2	FRS1	RM	FRD	1001011
FNMADD.D RM,FRD,FRS1,FRS2,FRS3	FRS3_01	FRS2	FRS1	RM	FRD	1001111
FADD.D RM,FRD,FRS1,FRS2,FRS3	0000001	FRS2	FRS1	RM	FRD	1010011
FSUB.D RM,FRD,FRS1,FRS2,FRS3	0000101	FRS2	FRS1	RM	FRD	1010011
FMUL.D RM,FRD,FRS1,FRS2,FRS3	0001001	FRS2	FRS1	RM	FRD	1010011
FDIV.D RM,FRD,FRS1,FRS2,FRS3	0001101	FRS2	FRS1	RM	FRD	1010011
FSGNJ.D FRD,FRS1,FRS2	0010001	FRS2	FRS1	000	FRD	1010011
FSGNJN.D FRD,FRS1,FRS2	0010001	FRS2	FRS1	001	FRD	1010011
FSGNJX.D FRD,FRS1,FRS2	0010001	FRS2	FRS1	010	FRD	1010011
FMIN.D FRD,FRS1,FRS2	0010101	FRS2	FRS1	000	FRD	1010011
FMAX.D FRD,FRS1,FRS2	0010101	FRS2	FRS1	001	FRD	1010011
FSQRT.D FRD,FRS1,FRS2	0101101	00000	FRS1	RM	FRD	1010011
FLE.D FRD,FRS1,FRS2	1010001	FRS2	FRS1	000	FRD	1010011
FLT.D FRD,FRS1,FRS2	1010001	FRS2	FRS1	001	FRD	1010011
FEQ.D FRD,FRS1,FRS2	1010001	FRS2	FRS1	010	FRD	1010011
FCVT.W.D RM,RD,FRS1	1100001	00000	FRS1	RM	FRD	1010011
FCVT.WU.D RM,RD,FRS1	1100001	00010	FRS1	RM	FRD	1010011
FCVT.D.W RM,RD,FRS1	1101001	00000	FRS1	RM	FRD	1010011
FCVT.D.WU RM,RD,FRS1	1101001	00010	FRS1	RM	FRD	1010011
FCLASS.D RD,FRS1	1110001	00000	FRS1	001	RD	1010011

RV64D : Standard Extension for Double-Precision Floating-Point (64 bit)

RV64D	31:25	24:20	19:15	14:12	11:7	6:0
FCVT.L.D RM,RD,FRS1	1100001	00010	FRS1	RM	FRD	1010011
FCVT.LU.D RM,RD,FRS1	1100001	00011	FRS1	RM	FRD	1010011
FCVT.D.L RM,RD,FRS1	1101001	00010	FRS1	RM	FRD	1010011
FCVT.D.LU RM,RD,FRS1	1101001	00011	FRS1	RM	FRD	1010011
FMV.X.D RD,FRS1	1110001	00000	FRS1	000	RD	1010011
FMV.D.X RD,FRS1	1111001	00000	RS1	000	FRD	1010011

1.2.3. ISA Modes

1.2.3.1. RISC-V User

1.2.3.2. RISC-V Supervisor

1.2.3.3. RISC-V Hypervisor

1.2.3.4. RISC-V Machine

1.3. OpenRISC ISA

1.4. MSP430 ISA

2. PROJECTS

2.1. CORE-RISCV

2.1.1. Definition

2.1.2. RISC Pipeline

In computer science, instruction pipelining is a technique for implementing instruction-level parallelism within a PU. Pipelining attempts to keep every part of the processor busy with some instruction by dividing incoming instructions into a series of sequential steps performed by different PUs with different parts of instructions processed in parallel. It allows faster PU throughput than would otherwise be possible at a given clock rate.

Typical	Modified	Module
FETCH	FETCH	riscv_if
...	PRE-DECODE	riscv_id
DECODE	DECODE	riscv_id
EXECUTE	EXECUTE	riscv_execution
MEMORY	MEMORY	riscv_memory
WRITE-BACK	WRITE-BACK	riscv_wb

- IF – Instruction Fetch Unit : Send out the PC and fetch the instruction from memory into the Instruction Register (IR); increment the PC to address the next sequential instruction. The IR is used to hold the next instruction that will be needed on subsequent clock cycles; likewise the register NPC is used to hold the next sequential PC.
- ID – Instruction Decode Unit : Decode the instruction and access the register file to read the registers. This unit gets instruction from IF, and extracts opcode and operand from that instruction. It also retrieves register values if requested by the operation.
- EX – Execution Unit : The ALU operates on the operands prepared in prior cycle, performing one functions depending on instruction type.
- MEM – Memory Access Unit : Instructions active in this unit are loads, stores and branches.
- WB – WriteBack Unit : Write the result into the register file, whether it comes from the memory system or from the ALU.

2.1.3. CORE-RISCV Organization

The CORE-RISCV is based on the Harvard architecture, which is a computer architecture with separate storage and signal pathways for instructions and data. The implementation is heavily modular, with each particular functional block of the design being contained within its own HDL module or modules. The RISCV implementation was developed in order to provide a better platform for processor component development than previous implementations.

Core	Module description
riscv_core	Core
...riscv_if	Instruction Fetch
...riscv_id	Instruction Decoder
...riscv_execution	Execution Unit
....riscv_alu	Arithmetic & Logical Unit
....riscv_lsu	Load Store Unit
....riscv_bu	Branch Unit
....riscv_mul	Multiplier Unit
....riscv_div	Division Unit
...riscv_memory	Memory Unit
...riscv_wb	Data Memory Access (Write Back)
...riscv_state	State Unit
...riscv_rf	Register File
...riscv_bp	Correlating Branch Prediction Unit
....riscv_ram_1rw	RAM 1RW1
.....riscv_ram_1rw_generic	RAM 1RW1 Generic
...riscv_du	Debug Unit

In a Harvard architecture, there is no need to make the two memories share characteristics. In particular, the word width, timing, implementation technology, and memory address structure can differ. In some systems, instructions for pre-programmed tasks can be stored in read-only memory while data memory generally requires read-write memory. In some systems, there is much more instruction memory than data memory so instruction addresses are wider than data addresses.

2.1.4. Parameters

Parameter	Type	Default	Description
JEDEC_BANK	Integer	0x0A	JEDEC Bank
JEDEC_MANUFACTURER_ID	Integer	0x6E	JEDEC Manufacturer ID
XLEN	Integer	64	Data Path Width
PLEN	Integer	64	Physical Memory Address Size
PMP_CNT	Integer	16	Physical Memory Protection Entries
PMA_CNT	Integer	16	Physical Memory Attribute Entries
HAS_USER	Integer	1	User Mode Enable
HAS_SUPER	Integer	1	Supervisor Mode Enable
HAS_HYPER	Integer	1	Hypervisor Mode Enable
HAS_RVM	Integer	1	“M” Extension Enable
HAS_RVA	Integer	1	“A” Extension Enable
HAS_RVC	Integer	1	“C” Extension Enable
HAS_BPU	Integer	1	Branch Prediction Unit Control Enable
IS_RV32E	Integer	0	Base Integer Instruction Set Enable
MULT_LATENCY	Integer	1	Hardware Multiplier Latency
ICACHE_SIZE	Integer	16	Instruction Cache size
ICACHE_BLOCK_SIZE	Integer	64	Instruction Cache block length
ICACHE_WAYS	Integer	2	Instruction Cache associativity
ICACHE_REPLACE_ALG	Integer	0	Instruction Cache replacement
DCACHE_SIZE	Integer	16	Data Cache size
DCACHE_BLOCK_SIZE	Integer	64	Data Cache block length
DCACHE_WAYS	Integer	2	Data Cache associativity
DCACHE_REPLACE_ALG	Integer	0	Data Cache replacement algorithm
HARTID	Integer	0	Hart Identifier

Parameter	Type	Default	Description
PC_INIT	Address	'h200	Program Counter Initialisation Vector
MNMIVEC_DEFAULT	Address	PC_INIT-'h004	Machine Mode Non-Maskable
MTVEC_DEFAULT	Address	PC_INIT-'h040	Machine Mode Interrupt Address
HTVEC_DEFAULT	Address	PC_INIT-'h080	Hypervisor Mode Interrupt Address
STVEC_DEFAULT	Address	PC_INIT-'h0C0	Supervisor Mode Interrupt Address
UTVEC_DEFAULT	Address	PC_INIT-'h100	User Mode Interrupt Address
BP_LOCAL_BITS	Integer	10	Number of local predictor bits
BP_GLOBAL_BITS	Integer	2	Number of global predictor bits
BREAKPOINTS	Integer	3	Number of hardware breakpoints
TECHNOLOGY	String	GENERIC	Target Silicon Technology

2.1.5. Instruction Inputs/Outputs Bus

Port	Size	Direction	Description
ins_stb	1	Input	Strobe
ins_stb_ack	1	Output	Strobe acknowledge
ins_d_ack	1	Output	Data acknowledge
ins_adri	PLEN	Input	Start address
ins_adro	PLEN	Output	Response address
ins_size	3	Input	Syze
ins_type	3	Input	Type
ins_prot	3	Input	Protection
ins_lock	1	Input	Locked access
ins_d	XLEN	Input	Write data
ins_q	XLEN	Output	Read data
ins_ack	1	Output	Acknowledge
ins_err	1	Output	Error

2.1.6. Data Inputs/Outputs Bus

Port	Size	Direction	Description
dat_stb	1	Input	Strobe
dat_stb_ack	1	Output	Strobe acknowledge
dat_d_ack	1	Output	Data acknowledge
dat_adri	PLEN	Input	Start address
dat_adro	PLEN	Output	Response address
dat_size	3	Input	Syze
dat_type	3	Input	Type
dat_prot	3	Input	Protection
dat_lock	1	Input	Locked access
dat_d	XLEN	Input	Write data
dat_q	XLEN	Output	Read data
dat_ack	1	Output	Acknowledge
dat_err	1	Output	Error

2.2. PU-RISCV

2.2.1. Definition

The RISC-V implementation has a 32/64/128 bit Microarchitecture, 6 stages data pipeline and an Instruction Set Architecture based on Reduced Instruction Set Computer. Compatible with AMBA and Wishbone Buses. For Researching and Developing.

Processing Unit	Module description
riscv_pu	Processing Unit
...riscv_core	Core
...riscv_imem_ctrl	Instruction Memory Access Block
...riscv_biu - imem	Bus Interface Unit (Instruction)
...riscv_dmem_ctrl	Data Memory Access Block
...riscv_biu - dmem	Bus Interface Unit (Data)

A PU cache is a hardware cache used by the PU to reduce the average cost (time or energy) to access instruction/data from the main memory. A cache is a smaller, faster memory, closer to a core, which stores copies of the data from frequently used main memory locations. Most CPUs have different independent caches, including instruction and data caches.

2.2.2. Instruction Cache

2.2.2.1. Instruction Organization

Instruction Memory	Module description
riscv_imem_ctrl	Instruction Memory Access Block
...riscv_membuf	Memory Access Buffer
....riscv_ram_queue	Fall-through Queue
...riscv_memmisaligned	Misalignment Check
...riscv_mmu	Memory Management Unit
...riscv_pmachk	Physical Memory Attributes Checker
...riscv_pmpchk	Physical Memory Protection Checker
...riscv_icache_core	Instruction Cache (Write Back)
....riscv_ram_1rw	RAM 1RW
.....riscv_ram_1rw_generic	RAM 1RW Generic
...riscv_dext	Data External Access Logic
...riscv_ram_queue	Fall-through Queue
...riscv_mux	Bus-Interface-Unit Mux
riscv_biu	Bus Interface Unit

2.2.2.2. Instruction INPUTS/OUTPUTS AMBA4 AXI-Lite Bus

2.2.2.2.1. Signals of the Read and Write Address channels

Write Port	Read Port	Size	Direction	Description
AWID	ARID	AXI_ID_WIDTH	Output	Address ID, to identify multiple streams
AWADDR	ARADDR	AXI_ADDR_WIDTH	Output	Address of the first beat of the burst
AWLEN	ARLEN	8	Output	Number of beats inside the burst
AWSIZE	ARSIZE	3	Output	Size of each beat
AWBURST	ARBURST	2	Output	Type of the burst
AWLOCK	ARLOCK	1	Output	Lock type, to provide atomic operations

Write Port	Read Port	Size	Direction	Description
AWCACHE	ARCACHE	4	Output	Memory type, progress through the system
AWPROT	ARPROT	3	Output	Protection type
AWQOS	ARQOS	4	Output	Quality of Service of the transaction
AWREGION	ARREGION	4	Output	Region identifier, physical to logical
AWUSER	ARUSER	AXI_USER_WIDTH	Output	User-defined data
AWVALID	ARVALID	1	Output	xVALID handshake signal
AWREADY	ARREADY	1	Input	xREADY handshake signal

2.2.2.2.2. Signals of the Read and Write Data channels

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

2.2.2.2.3. Signals of the Write Response channel

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

2.2.2.3. Instruction INPUTS/OUTPUTS AMBA3 AHB-Lite Bus

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

2.2.2.4. Instruction INPUTS/OUTPUTS Wishbone Bus

Port	Size	Direction	Description
rst	1	Input	Synchronous Active High Reset
clk	1	Input	System Clock Input
iadr	AW	Input	Instruction Address Bus
idati	DW	Input	Instruction Input Bus
idato	DW	Output	Instruction Output Bus
isel	DW/8	Input	Byte Select Signals
iwe	1	Input	Write Enable Input
istb	1	Input	Strobe Signal/Core Select Input
icyc	1	Input	Valid Bus Cycle Input
iack	1	Output	Bus Cycle Acknowledge Output
ierr	1	Output	Bus Cycle Error Output
iint	1	Output	Interrupt Signal Output

2.2.3. Data Cache

2.2.3.1. Data Organization

Data Memory	Module description
riscv_dmem_ctrl	Data Memory Access Block
...riscv_membuf	Memory Access Buffer
....riscv_ram_queue	Fall-through Queue
...riscv_mmemmisaligned	Misalignment Check
...riscv_mmu	Memory Management Unit
...riscv_pmachk	Physical Memory Attributes Checker
...riscv_pmpchk	Physical Memory Protection Checker
...riscv_dcache_core	Data Cache (Write Back)
....riscv_ram_1rw	RAM 1RW
.....riscv_ram_1rw_generic	RAM 1RW Generic
...riscv_dext	Data External Access Logic
...riscv_mux	Bus-Interface-Unit Mux
riscv_biu	Bus Interface Unit

2.2.3.2. Data INPUTS/OUTPUTS AMBA4 AXI-Lite Bus

2.2.3.2.1. Signals of the Read and Write Address channels

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

Write Port	Read Port	Size	Direction	Description
AWUSER	ARUSER	AXI_USER_WIDTH	Output	User-defined data
AWVALID	ARVALID	1	Output	xVALID handshake signal
AWREADY	ARREADY	1	Input	xREADY handshake signal

2.2.3.2.2. Signals of the Read and Write Data channels

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

2.2.3.2.3. Signals of the Write Response channel

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

2.2.3.3. Data INPUTS/OUTPUTS AMBA3 AHB-Lite Bus

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

2.2.3.4. Data INPUTS/OUTPUTS Wishbone Bus

Port	Size	Direction	Description
rst	1	Input	Synchronous Active High Reset
clk	1	Input	System Clock Input
dadr	AW	Input	Data Address Bus
ddati	DW	Input	Data Input Bus
ddato	DW	Output	Data Output Bus
dsel	DW/8	Input	Byte Select Signals
dwe	1	Input	Write Enable Input
dstb	1	Input	Strobe Signal/Core Select Input
dcyc	1	Input	Valid Bus Cycle Input
dack	1	Output	Bus Cycle Acknowledge Output
derr	1	Output	Bus Cycle Error Output
dint	1	Output	Interrupt Signal Output

2.3. CORE-OR1K

2.3.1. Definition

2.3.2. RISC Pipeline

In computer science, instruction pipelining is a technique for implementing instruction-level parallelism within a PU. Pipelining attempts to keep every part of the processor busy with some instruction by dividing incoming instructions into a series of sequential steps performed by different PUs with different parts of instructions processed in parallel. It allows faster PU throughput than would otherwise be possible at a given clock rate.

Typical	Modified	Module
FETCH	FETCH	or1k_cache_lru or1k_fetch_cappuccino or1k_icache or1k_immu
DECODE	DECODE	or1k_decode
EXECUTE & CONTROL	EXECUTE & WRITE-BACK	or1k_execute_alu or1k_execute_ctrl_cappuccino or1k_rf_cappuccino or1k_wb_mux_cappuccino
MEMORY	MEMORY	or1k_dcache or1k_dmmu or1k_lsu_cappuccino
WRITE-BACK	CONTROL	or1k_store_buffer or1k_cfgrs or1k_ctrl_cappuccino or1k_pcu or1k_pic or1k_ticktimer

- IF – Instruction Fetch Unit : Send out the PC and fetch the instruction from memory into the Instruction Register (IR); increment the PC to address the next sequential instruction. The IR is used to hold the next instruction that will be needed on subsequent clock cycles; likewise the register NPC is used to hold the next sequential PC.
- ID – Instruction Decode Unit : Decode the instruction and access the register file to read the registers. This unit gets instruction from IF, and extracts opcode and operand from that instruction. It also

retrieves register values if requested by the operation.

- EX – Execution Unit : The ALU operates on the operands prepared in prior cycle, performing one functions depending on instruction type.
- MEM – Memory Access Unit : Instructions active in this unit are loads, stores and branches.
- WB – WriteBack Unit : Write the result into the register file, whether it comes from the memory system or from the ALU.

2.3.3. CORE-OR1K Organization

The CORE-OR1K is based on the Harvard architecture, which is a computer architecture with separate storage and signal pathways for instructions and data. The implementation is heavily modular, with each particular functional block of the design being contained within its own HDL module or modules. The OR1K implementation was developed in order to provide a better platform for processor component development than previous implementations.

Core	Module description
<code>or1k_core</code>	Top-level, instantiating bus interfaces, data cache and CPU
<code>...or1k_dcache</code>	Data cache implementation
<code>...or1k_bus_if_xx</code>	Bus interface, depending on desired bus standard
<code>...or1k_cpu</code>	Pipeline implementation wrapper
<code>....or1k_cpu_xx</code>	Pipeline implementation, depending on configuration
<code>.....or1k_fetch_xx</code>	Pipeline-implementation-dependent fetch stage
<code>.....or1k_decode</code>	Generic decode stage
<code>.....or1k_execute_alu</code>	Generic ALU for execute stage
<code>.....or1k_lsu_xx</code>	Pipeline-implementation-dependent load/store unit
<code>.....or1k_wb_mux_xx</code>	Pipeline-implementation-dependent writeback stage mux
<code>.....or1k_rf_xx</code>	Pipeline-implementation-dependent register file
<code>.....or1k_ctrl_xx</code>	Pipeline-implementation-dependent control stage

In a Harvard architecture, there is no need to make the two memories share characteristics. In particular, the word width, timing, implementation technology, and memory address structure can differ. In some systems, instructions for pre-programmed tasks can be stored in read-only memory while data memory generally requires read-write memory. In some systems, there is much more instruction memory than data memory so instruction addresses are wider than data addresses.

2.3.4. Parameters

2.3.4.1. Basic Parameters

Parameter	Description	Default	Values
<code>OPTION_OPERAND_WIDTH</code>	CPU data and address widths	32	32, 64
<code>OPTION_CPU0</code>	CPU pipeline core	CAPPUCCINO	CAPPUCCINO
<code>OPTION_RESET_PC</code>	Program Counter upon reset	0x100	N

2.3.4.2. Caching Parameters

Parameter	Description	Default	Values
<code>FEATURE_DATACACHE</code>	Enable memory access data caching	NONE	ENABLED
<code>OPTION_DCACHE_BLOCK_WIDTH</code>	Address width of a cache block	5	n
<code>OPTION_DCACHE_SET_WIDTH</code>	Set address width	9	n

Parameter	Description	Default	Values
OPTION_DCACHE_WAYS	Number of blocks per set	2	n
OPTION_DCACHE_LIMIT_WIDTH	Maximum address width	32	n
OPTION_DCACHE_SNOOP	Bus snooping for cache coherency	NONE	ENABLED
FEATURE_INSTRUCTIONCACHE	Memory access instruction caching	NONE	ENABLED
OPTION_ICACHE_BLOCK_WIDTH	Address width of a cache block	5	n
OPTION_ICACHE_SET_WIDTH	Set address width	9	n
OPTION_ICACHE_WAYS	Number of blocks per set	2	n
OPTION_ICACHE_LIMIT_WIDTH	Maximum address width	32	n

2.3.4.3. Memory Management Unit (MMU) Parameters

Parameter	Description	Default	Values
FEATURE_DMMU	Enable the data bus MMU	NONE	ENABLED
FEATURE_DMMU_HW_TLB_RELOAD	Enable hardware TLB reload	NONE	ENABLED
OPTION_DMMU_SET_WIDTH	Set address width	6	n
OPTION_DMMU_WAYS	Number of ways per set	1	n
FEATURE_IMMU	Enable the instruction bus MMU	NONE	ENABLED
FEATURE_IMMU_HW_TLB_RELOAD	Enable hardware TLB reload	NONE	ENABLED
OPTION_IMMU_SET_WIDTH	Set address width	6	n
OPTION_IMMU_WAYS	Number of ways per set	1	n

2.3.4.4. System Bus Parameters

Parameter	Description	Default
FEATURE_STORE_BUFFER	Load store unit store buffer	ENABLED
OPTION_STORE_BUFFER_DEPTH_WIDTH	Load store unit store buffer depth	8
BUS_IF_TYPE	Bus interface type	WISHBONE32
IBUS_WB_TYPE	Instruction bus interface	B3_READ_BURSTING
DBUS_WB_TYPE	Data bus interface type option	CLASSIC

2.3.4.5. Hardware Unit Configuration Parameters

Parameter	Description	Default
FEATURE_TRACEPORT_EXEC	Traceport hardware interface	NONE
FEATURE_DEBUGUNIT	Hardware breakpoints and debug unit	NONE
FEATURE_PERFCOUNTERS	Performance counters unit	NONE
OPTION_PERFCOUNTERS_NUM	Performance counters to generate	0
FEATURE_TIMER	Internal OpenRISC timer	ENABLED
FEATURE_PIC	Internal OpenRISC PIC	ENABLED
OPTION_PIC_TRIGGER	PIC trigger mode	LEVEL
OPTION_PIC_NMI_WIDTH	Non maskable interrupts width	0
OPTION_RF_CLEAR_ON_INIT	clearing all registers on initialization	0
OPTION_RF_NUM_SHADOW_GPR	Number of shadow register files	0
OPTION_RF_ADDR_WIDTH	Address width of the register file	5
OPTION_RF_WORDS	Number of registers in the register file	32
FEATURE_FASTCONTEXTS	Fast context switching of register sets	NONE
FEATURE_MULTICORE	coreid and numcores SPR registers	NONE
FEATURE_FPU	FPU, for cappuccino pipeline only	NONE

Parameter	Description	Default
OPTION_FTOI_ROUNDING	Rounding behavior for <code>lf.ftoi.s</code>	CPP
FEATURE_BRANCH_PREDICTOR	Branch predictor implementation	SIMPLE

2.3.4.6. Exception Handling Options

Parameter	Description	Default
FEATURE_DSX	Enable setting the <code>SR[DSX]</code> flag	ENABLED
FEATURE_RANGE	Enable checking and raising range exceptions	ENABLED
FEATURE_OVERFLOW	Enable checking and raising overflow exceptions	ENABLED

2.3.4.7. ALU Configuration Options

Parameter	Description	Default
FEATURE_MULTIPLIER	Specify the multiplier implementation	THREESTAGE
FEATURE_DIVIDER	Specify the divider implementation	SERIAL
OPTION_SHIFTER	Specify the shifter implementation	BARREL
FEATURE_CARRY_FLAG	Enable checking and setting the carry flag	ENABLED

2.3.4.8. Instruction Enabling Options

Parameter	Description	Default
FEATURE_MAC	<code>1.mac*</code> multiply accumulate instructions	NONE
FEATURE_SYSCALL	<code>1.sys</code> OS syscall instruction	ENABLED
FEATURE_TRAP	<code>1.trap</code> instruction	ENABLED
FEATURE_ADDC	<code>1.addc</code> add with <code>carry</code> flag instruction	ENABLED
FEATURE_SRA	<code>1.sra</code> shift right arithmetic instruction	ENABLED
FEATURE_ROR	<code>1.ror*</code> rotate right instructions	NONE
FEATURE_EXT	<code>1.ext*</code> sign extend instructions	NONE
FEATURE_CMOV	<code>1.cmov</code> conditional move instruction	ENABLED
FEATURE_FFL1	<code>1.f[f1]</code> 1 find first/last set bit instructions	ENABLED
FEATURE_ATOMIC	<code>1.lwa</code> and <code>1.swa</code> atomic instructions	ENABLED
FEATURE_CUST1	<code>1.cust*</code> custom instruction	NONE
FEATURE_CUST2	<code>1.cust*</code> custom instruction	NONE
FEATURE_CUST3	<code>1.cust*</code> custom instruction	NONE
FEATURE_CUST4	<code>1.cust*</code> custom instruction	NONE
FEATURE_CUST5	<code>1.cust*</code> custom instruction	NONE
FEATURE_CUST6	<code>1.cust*</code> custom instruction	NONE
FEATURE_CUST7	<code>1.cust*</code> custom instruction	NONE
FEATURE_CUST8	<code>1.cust*</code> custom instruction	NONE

2.3.5. Instruction Inputs/Outputs Bus

2.3.6. Data Inputs/Outputs Bus

2.4. PU-OR1K

2.4.1. Definition

The OpenRISC implementation has a 32/64 bit Microarchitecture, 5 stages data pipeline and an Instruction Set Architecture based on Reduced Instruction Set Computer. Compatible with Wishbone Bus. Only For Researching.

A PU cache is a hardware cache used by the PU to reduce the average cost (time or energy) to access instruction/data from the main memory. A cache is a smaller, faster memory, closer to a core, which stores copies of the data from frequently used main memory locations. Most CPUs have different independent caches, including instruction and data caches.

2.4.2. Instruction Cache

2.4.2.1. Instruction Organization

Instruction Memory	Module description
riscv_imem_ctrl	Instruction Memory Access Block
...riscv_membuf	Memory Access Buffer
....riscv_ram_queue	Fall-through Queue
...riscv_memmisaligned	Misalignment Check
...riscv_mmu	Memory Management Unit
...riscv_pmachk	Physical Memory Attributes Checker
...riscv_pmpchk	Physical Memory Protection Checker
...riscv_icache_core	Instruction Cache (Write Back)
....riscv_ram_1rw	RAM 1RW
.....riscv_ram_1rw_generic	RAM 1RW Generic
...riscv_dext	Data External Access Logic
...riscv_ram_queue	Fall-through Queue
...riscv_mux	Bus-Interface-Unit Mux
riscv_biu	Bus Interface Unit

2.4.2.2. Instruction INPUTS/OUTPUTS AMBA4 AXI-Lite Bus

2.4.2.2.1. Signals of the Read and Write Address channels

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

Write Port	Read Port	Size	Direction	Description
AWREADY	ARREADY	1	Input	xREADY handshake signal

2.4.2.2.2. Signals of the Read and Write Data channels

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

2.4.2.2.3. Signals of the Write Response channel

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

2.4.2.3. Instruction INPUTS/OUTPUTS AMBA3 AHB-Lite Bus

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

2.4.2.4. Instruction INPUTS/OUTPUTS Wishbone Bus

Port	Size	Direction	Description
rst	1	Input	Synchronous Active High Reset

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

2.4.3. Data Cache

2.4.3.1. Data Organization

Data Memory	Module description
riscv_dmem_ctrl	Data Memory Access Block
...riscv_membuf	Memory Access Buffer
....riscv_ram_queue	Fall-through Queue
...riscv_memmisaligned	Misalignment Check
...riscv_mmu	Memory Management Unit
...riscv_pmachk	Physical Memory Attributes Checker
...riscv_pmpchk	Physical Memory Protection Checker
...riscv_dcache_core	Data Cache (Write Back)
....riscv_ram_1rw	RAM 1RW
.....riscv_ram_1rw_generic	RAM 1RW Generic
...riscv_dext	Data External Access Logic
...riscv_mux	Bus-Interface-Unit Mux
riscv_biu	Bus Interface Unit

2.4.3.2. Data INPUTS/OUTPUTS AMBA4 AXI-Lite Bus

2.4.3.2.1. Signals of the Read and Write Address channels

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

2.4.3.2.2. Signals of the Read and Write Data channels

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

2.4.3.2.3. Signals of the Write Response channel

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

2.4.3.3. Data INPUTS/OUTPUTS AMBA3 AHB-Lite Bus

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

2.4.3.4. Data INPUTS/OUTPUTS Wishbone Bus

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

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

2.5. CORE-MSP430

2.5.1. Definition

2.5.2. RISC Pipeline

2.5.3. CORE-MSP430 Organization

2.5.4. Parameters

2.5.4.1. Basic System Configuration

Description	Parameter	Type	Default
Program Memory Size	PMEM_SIZE	integer	16384
Data Memory Size	DMEM_SIZE	integer	4096
Include/Exclude Hardware Multiplier	MULTIPLYING	bit	1
Include/Exclude Serial Debug interface	DBG_ON	bit	1

2.5.4.2. Advanced System Configuration (for experienced users)

Description	Parameter	Type	Default
Peripheral Memory Space	PER_SIZE	integer	512
Custom user version number	USER_VERSION	bit vector	0
Watchdog timer	WATCHDOG	bit	1
Non-Maskable-Interrupt support	NMI_EN	bit	1
Number of available IRQs	IRQ_16	bit	1
Number of available IRQs	IRQ_32	bit	0
Number of available IRQs	IRQ_64	bit	0
Input synchronizers	SYNC_NMI	bit	1
Input synchronizers	SYNC_CPU_EN	bit	0
Input synchronizers	SYNC_DBG_EN	bit	0
Debugger definition	DBG_RST_BRK_EN	bit	0

2.5.4.3. Expert System Configuration (experts only)

Description	Parameter	Type	Default
Hardware breakpoint/watchpoint units	DBG_HWBRK	bit vector	1
Select serial debug interface protocol	DBG_UART	bit	0
Select serial debug interface protocol	DBG_I2C	bit	1
I2C broadcast address	DBG_I2C_BROADCASTC	bit	1
Hardware breakpoint RANGE mode	HWBRK_RANGE	bit	1
ASIC version	ASIC	bit	1

2.5.4.4. ASIC System Configuration (experts/professionals only)

Description	Parameter	Type	Default
LOW POWER MODE: SCG	SCG_EN	bit vector	1
FINE GRAINED CLOCK GATING	CLOCK_GATING	bit	1
ASIC CLOCKING	ASIC_CLOCKING	bit	1
LFXT CLOCK DOMAIN	LFXT_DOMAIN	bit	1
MCLK: Clock Mux	MCLK_MUX	bit	1
SMCLK: Clock Mux	SMCLK_MUX	bit	1
WATCHDOG: Clock Mux	WATCHDOG_MUX	bit	1
WATCHDOG: Clock No-Mux	WATCHDOG_NOMUX_ACLK	bit	0
MCLK: Clock divider	MCLK_DIVIDER	bit	1
SMCLK: Clock divider (/1/2/4/8)	SMCLK_DIVIDER	bit	1
ACLK: Clock divider (/1/2/4/8)	ACLK_DIVIDER	bit	1
LOW POWER MODE: CPUOFF	CPUOFF_EN	bit	1
LOW POWER MODE: OSCOFF	OSCOFF_EN	bit	1

2.5.4.5. System Constants (do not edit)

Description	Parameter	Type	Default
Program Memory Size	PMEM_AWIDTH	integer	13
Data Memory Size	DMEM_AWIDTH	integer	11
Peripheral Memory Size	PER_AWIDTH	integer	8
Data Memory Base Addresses	DMEM_BASE	integer	N
Program Memory	PMEM_MSB	integer	N
Data Memory	DMEM_MSB	integer	N
Peripheral Memory	PER_MSB	integer	N
Number of available IRQs	IRQ_NR	integer	16
Instructions type	INST_SOC	integer	0
Instructions type	INST_JMPC	integer	1
Instructions type	INST_TOC	integer	2
Single-operand arithmetic	RRC	integer	0
Single-operand arithmetic	SWPB	integer	1
Single-operand arithmetic	RRA	integer	2
Single-operand arithmetic	SXTC	integer	3
Single-operand arithmetic	PUSH	integer	4
Single-operand arithmetic	CALL	integer	5
Single-operand arithmetic	RETI	integer	6
Single-operand arithmetic	IRQX	integer	7
Conditional jump	JNE	integer	0
Conditional jump	JEQ	integer	1
Conditional jump	JNC	integer	2
Conditional jump	JC	integer	3
Conditional jump	JN	integer	4
Conditional jump	JGE	integer	5
Conditional jump	JL	integer	6
Conditional jump	JMP	integer	7
Two-operand arithmetic	MOV	integer	0
Two-operand arithmetic	ADD	integer	1
Two-operand arithmetic	ADDC	integer	2
Two-operand arithmetic	SUBC	integer	3
Two-operand arithmetic	SUBB	integer	4

Description	Parameter	Type	Default
Two-operand arithmetic	CMP	integer	5
Two-operand arithmetic	DADD	integer	6
Two-operand arithmetic	BITC	integer	7
Two-operand arithmetic	BIC	integer	8
Two-operand arithmetic	BIS	integer	9
Two-operand arithmetic	XORX	integer	10
Two-operand arithmetic	ANDX	integer	11
Addressing modes	DIR	integer	0
Addressing modes	IDX	integer	1
Addressing modes	INDIR	integer	2
Addressing modes	INDIR_I	integer	3
Addressing modes	SYMB	integer	4
Addressing modes	IMM	integer	5
Addressing modes	ABSC	integer	6
Addressing modes	CONST	integer	7
Instruction state machine	I_IRQ_FETCH	bit vector	000
Instruction state machine	I_IRQ_DONE	bit vector	001
Instruction state machine	I_DEC	bit vector	010
Instruction state machine	I_EXT1	bit vector	011
Instruction state machine	I_EXT2	bit vector	100
Instruction state machine	I_IDLE	bit vector	101
Execution state machine	E_SRC_AD	bit vector	X5
Execution state machine	E_SRC_RD	bit vector	X6
Execution state machine	E_SRC_WR	bit vector	X7
Execution state machine	E_DST_AD	bit vector	X8
Execution state machine	E_DST_RD	bit vector	X9
Execution state machine	E_DST_WR	bit vector	XA
Execution state machine	E_EXEC	bit vector	XB
Execution state machine	E_JUMP	bit vector	XC
Execution state machine	E_IDLE	bit vector	XD
Execution state machine	E_IRQ_0	bit vector	X2
Execution state machine	E_IRQ_1	bit vector	X1
Execution state machine	E_IRQ_2	bit vector	X0
Execution state machine	E_IRQ_3	bit vector	X3
Execution state machine	E_IRQ_4	bit vector	X4
ALU control signals	ALU_SRC_INV	integer	0
ALU control signals	ALU_INC	integer	1
ALU control signals	ALU_INC_C	integer	2
ALU control signals	ALU_ADD	integer	3
ALU control signals	ALU_AND	integer	4
ALU control signals	ALU_OR	integer	5
ALU control signals	ALU_XOR	integer	6
ALU control signals	ALU_DADD	integer	7
ALU control signals	ALU_STAT_7	integer	8
ALU control signals	ALU_STAT_F	integer	9
ALU control signals	ALU_SHIFT	integer	10
ALU control signals	EXEC_NO_WR	integer	11
Debug interface	DBG_UART_WR	integer	18
Debug interface	DBG_UART_BW	integer	17
Debug interface CPU_CTL register	HALT	integer	0
Debug interface CPU_CTL register	RUN	integer	1
Debug interface CPU_CTL register	ISTEP	integer	2

Description	Parameter	Type	Default
Debug interface CPU_CTL register	SW_BRK_EN	integer	3
Debug interface CPU_CTL register	FRZ_BRK_EN	integer	4
Debug interface CPU_CTL register	RST_BRK_EN	integer	5
Debug interface CPU_CTL register	CPU_RST	integer	6
Debug interface BRKx_CTL register	BRK_MODE_RD	integer	0
Debug interface BRKx_CTL register	BRK_MODE_WR	integer	1
Debug interface BRKx_CTL register	BRK_EN	integer	2
Debug interface BRKx_CTL register	BRK_I_EN	integer	3
Debug interface BRKx_CTL register	BRK_RANGE	integer	4
Basic clock module: BCSCCTL2 Control Register	SELMX	integer	7
Basic clock module: BCSCCTL2 Control Register	SELS	integer	3
MCLK Clock gate	MCLK_CGATE	bit	1
SMCLK Clock gate	SMCLK_CGATE	bit	1
Debug interface: CPU version	CPU_VERSION	bit vector	010
Debug interface: Software breakpoint opcode	DBG_SWBRK_OP	bit vector	X4343
UART interface auto data synchronization	DBG_UART_AUTO_SYNC	bit	1
Counter width for the debug interface UART	DBG_UART_XFER_CNT_W	integer	16
Debug UART interface data rate	DBG_UART_BAUD	integer	2000000
Debug UART interface data rate	DBG_DCO_FREQ	integer	20000000
Debug UART interface data rate	DBG_UART_CNT	integer	N
Debug UART interface data rate	DBG_UART_CNTB	bit vector	N
Debug interface input synchronizer	SYNC_DBG_UART_RXD	bit	1
MULTIPLIER CONFIGURATION	MPY_16X16	bit	1

2.5.5. Instruction Inputs/Outputs Bus

2.5.6. Data Inputs/Outputs Bus

2.6. PU-MSP430

2.6.1. Definition

The MSP430 implementation has a 16 bit Microarchitecture, 3 stages data pipeline and an Instruction Set Architecture based on Reduced Instruction Set Computer. Compatible with Wishbone Bus. Only For Researching.

A PU cache is a hardware cache used by the PU to reduce the average cost (time or energy) to access instruction/data from the main memory. A cache is a smaller, faster memory, closer to a core, which stores copies of the data from frequently used main memory locations. Most CPUs have different independent caches, including instruction and data caches.

2.6.2. Instruction Cache

2.6.2.1. Instruction Organization

Instruction Memory	Module description
riscv_imem_ctrl	Instruction Memory Access Block
...riscv_membuf	Memory Access Buffer
....riscv_ram_queue	Fall-through Queue
...riscv_memisaligned	Misalignment Check
...riscv_mmu	Memory Management Unit
...riscv_pmachk	Physical Memory Attributes Checker
...riscv_pmpchk	Physical Memory Protection Checker
...riscv_icache_core	Instruction Cache (Write Back)

Instruction Memory	Module description
.....riscv_ram_1rw	RAM 1RW
.....riscv_ram_1rw_generic	RAM 1RW Generic
...riscv_dext	Data External Access Logic
...riscv_ram_queue	Fall-through Queue
...riscv_mux	Bus-Interface-Unit Mux
riscv_biu	Bus Interface Unit

2.6.2.2. Instruction INPUTS/OUTPUTS AMBA4 AXI-Lite Bus

2.6.2.2.1. Signals of the Read and Write Address channels

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

2.6.2.2.2. Signals of the Read and Write Data channels

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

2.6.2.2.3. Signals of the Write Response channel

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

2.6.2.3. Instruction INPUTS/OUTPUTS AMBA3 AHB-Lite Bus

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

2.6.2.4. Instruction INPUTS/OUTPUTS Wishbone Bus

Port	Size	Direction	Description
rst	1	Input	Synchronous Active High Reset
clk	1	Input	System Clock Input
iadr	AW	Input	Instruction Address Bus
idati	DW	Input	Instruction Input Bus
idato	DW	Output	Instruction Output Bus
isel	DW/8	Input	Byte Select Signals
iwe	1	Input	Write Enable Input
istb	1	Input	Strobe Signal/Core Select Input
icyc	1	Input	Valid Bus Cycle Input
iack	1	Output	Bus Cycle Acknowledge Output
ierr	1	Output	Bus Cycle Error Output
iint	1	Output	Interrupt Signal Output

2.6.3. Data Cache

2.6.3.1. Data Organization

Data Memory	Module description
riscv_dmem_ctrl	Data Memory Access Block
...riscv_membuf	Memory Access Buffer
....riscv_ram_queue	Fall-through Queue
...riscv_memmisaligned	Misalignment Check
...riscv_mmu	Memory Management Unit
...riscv_pmachk	Physical Memory Attributes Checker
...riscv_pmpchk	Physical Memory Protection Checker
...riscv_dcache_core	Data Cache (Write Back)
....riscv_ram_1rw	RAM 1RW
.....riscv_ram_1rw_generic	RAM 1RW Generic

Data Memory	Module description
...riscv_dext	Data External Access Logic
...riscv_mux	Bus-Interface-Unit Mux
riscv_biu	Bus Interface Unit

2.6.3.2. Data INPUTS/OUTPUTS AMBA4 AXI-Lite Bus

2.6.3.2.1. Signals of the Read and Write Address channels

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

2.6.3.2.2. Signals of the Read and Write Data channels

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

2.6.3.2.3. Signals of the Write Response channel

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

2.6.3.3. Data INPUTS/OUTPUTS AMBA3 AHB-Lite Bus

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

2.6.3.4. Data INPUTS/OUTPUTS Wishbone Bus

Port	Size	Direction	Description
rst	1	Input	Synchronous Active High Reset
clk	1	Input	System Clock Input
dadr	AW	Input	Data Address Bus
ddati	DW	Input	Data Input Bus
ddato	DW	Output	Data Output Bus
dsel	DW/8	Input	Byte Select Signals
dwe	1	Input	Write Enable Input
dstb	1	Input	Strobe Signal/Core Select Input
dcyc	1	Input	Valid Bus Cycle Input
dack	1	Output	Bus Cycle Acknowledge Output
derr	1	Output	Bus Cycle Error Output
dint	1	Output	Interrupt Signal Output

3. WORKFLOW

3.1. HARDWARE

1. System Level (SystemC/SystemVerilog)

The System Level abstraction of a system only looks at its biggest building blocks like processing units or peripheral devices. At this level the circuit is usually described using traditional programming languages like SystemC or SystemVerilog. Sometimes special software libraries are used that are aimed at simulation circuits on the system level. The IEEE 1685-2009 standard defines the IP-XACT file format that can be used to represent designs on the system level and building blocks that can be used in such system level designs.

2. Behavioral & Register Transfer Level (VHDL/Verilog)

At the Behavioural Level abstraction a language aimed at hardware description such as Verilog or VHDL is used to describe the circuit, but so-called behavioural modeling is used in at least part of the circuit description. In behavioural modeling there must be a language feature that allows for imperative programming to be used to describe data paths and registers. This is the always -block in Verilog and the process -block in VHDL.

A design in Register Transfer Level representation is usually stored using HDLs like Verilog and VHDL. But only a very limited subset of features is used, namely minimalistic always blocks (Verilog) or process blocks (VHDL) that model the register type used and unconditional assignments for the datapath logic. The use of HDLs on this level simplifies simulation as no additional tools are required to simulate a design in Register Transfer Level representation.

3. Logical Gate

At the Logical Gate Level the design is represented by a netlist that uses only cells from a small number of single-bit cells, such as basic logic gates (AND, OR, NOT, XOR, etc.) and registers (usually D-Type Flip-flops). A number of netlist formats exists that can be used on this level such as the Electronic Design Interchange Format (EDIF), but for ease of simulation often a HDL netlist is used. The latter is a HDL file (Verilog or VHDL) that only uses the most basic language constructs for instantiation and connecting of cells.

4. Physical Gate

On the Physical Gate Level only gates are used that are physically available on the target architecture. In some cases this may only be NAND, NOR and NOT gates as well as D-Type registers. In the case of an FPGA-based design the Physical Gate Level representation is a netlist of LUTs with optional output registers, as these are the basic building blocks of FPGA logic cells.

5. Switch Level

A Switch Level representation of a circuit is a netlist utilizing single transistors as cells. Switch Level modeling is possible in Verilog and VHDL, but is seldom used in modern designs, as in modern digital ASIC or FPGA flows the physical gates are considered the atomic build blocks of the logic circuit.

3.1.1. Front-End Open Source Tools

3.1.1.1. Modeling System Level of Hardware

A System Description Language Editor is a computer tool allows to generate software code. A System Description Language is a formal language, which comprises a Programming Language (input), producing a Hardware Description (output). Programming languages are used in computer programming to implement algorithms. The description of a programming language is split into the two components of syntax (form) and semantics (meaning).

SystemVerilog System Description Language Editor

type:

```
git clone --recursive https://github.com/emacs-mirror/emacs
```

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

3.1.1.2. Simulating System Level of Hardware

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

SystemVerilog System Description Language Simulator

type:

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

```

cd verilator
autoconf
./configure
make
sudo make install

cd sim/verilog/tests/wb/verilator
source SIMULATE-IT

cd sim/verilog/tests/ahb3/verilator
source SIMULATE-IT

cd sim/verilog/tests/axi4/verilator
source SIMULATE-IT

```

3.1.1.3. Verifying System Level of Hardware

A UVM standard improves interoperability and reduces the cost of repurchasing and rewriting IP for each new project or Electronic Design Automation tool. It also makes it easier to reuse verification components. The UVM Class Library provides generic utilities, such as component hierarchy, Transaction Library Model or configuration database, which enable the user to create virtually any structure wanted for the testbench.

SystemVerilog System Description Language Verifier

type:

```

git clone --recursive https://github.com/QueenField/UVM

cd sim/verilog/pu/riscv/wb/msim
source SIMULATE-IT

cd sim/verilog/pu/riscv/ahb3/msim
source SIMULATE-IT

cd sim/verilog/pu/riscv/axi4/msim
source SIMULATE-IT

```

3.1.1.4. Describing Register Transfer Level of Hardware

A Hardware Description Language Editor is any editor that allows to generate hardware code. Hardware Description Language is a specialized computer language used to describe the structure and behavior of digital logic circuits. It allows for the synthesis of a HDL into a netlist, which can then be synthesized, placed and routed to produce the set of masks used to create an integrated circuit.

VHDL/Verilog Hardware Description Language Editor

type:

```

git clone --recursive https://github.com/emacs-mirror/emacs

cd emacs
./configure
make
sudo make install

```

3.1.1.5. Simulating Register Transfer Level of Hardware

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

Verilog Hardware Description Language Simulator

type:

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

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

cd sim/verilog/tests/wb/iverilog
source SIMULATE-IT

cd sim/verilog/tests/ahb3/iverilog
source SIMULATE-IT

cd sim/verilog/tests/axi4/iverilog
source SIMULATE-IT
```

VHDL Hardware Description Language Simulator

type:

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

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

cd sim/vhdl/tests/wb/ghdl
source SIMULATE-IT

cd sim/vhdl/tests/ahb3/ghdl
source SIMULATE-IT

cd sim/vhdl/tests/axi4/ghdl
source SIMULATE-IT
```

3.1.1.6. Synthesizing Register Transfer Level of Hardware

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

Verilog Hardware Description Language Synthesizer

type:

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

```
cd yosys
make
sudo make install
```

VHDL Hardware Description Language Synthesizer

type:

```
git clone --recursive https://github.com/ghdl/ghdl-yosys-plugin
cd ghdl-yosys-plugin
make GHDL=/usr/local
sudo yosys-config --exec mkdir -p --datdir/plugins
sudo yosys-config --exec cp "ghdl.so" --datdir/plugins/ghdl.so
```

3.1.1.7. Optimizing Register Transfer Level of Hardware

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

Verilog Hardware Description Language Optimizer

type:

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

cd yosys
make
sudo make install
```

3.1.1.8. Verifying Register Transfer Level of Hardware

A Hardware Description Language Verifier proves or disproves the correctness of intended algorithms underlying a hardware system with respect to a certain formal specification or property, using formal methods of mathematics. Formal verification uses modern techniques (SAT/SMT solvers, BDDs, etc.) to prove correctness by essentially doing an exhaustive search through the entire possible input space (formal proof).

Verilog Hardware Description Language Verifier

type:

```
git clone --recursive https://github.com/YosysHQ/SymbiYosys
```

3.1.2. Back-End Open Source Tools

I. Back-End Workflow Qflow for ASICs

type:

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

type:

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

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

3.1.2.1. Planning Switch Level of Hardware

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

Floor-Planner

type:

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

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

3.1.2.2. Placing Switch Level of Hardware

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

Standard Cell Placer

type:

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

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

3.1.2.3. Timing Switch Level of Hardware

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

Standard Cell Timing-Analyzer

type:

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

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

3.1.2.4. Routing Switch Level of Hardware

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

Standard Cell Router

type:

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

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

3.1.2.5. Simulating Switch Level of Hardware

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

Standard Cell Simulator

type:

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

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

3.1.2.6. Verifying Switch Level of Hardware LVS

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

Standard Cell Verifier

type:

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

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

3.1.2.7. Checking Switch Level of Hardware DRC

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

Standard Cell Checker

type:

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

cd magic
```

```
./configure
make
sudo make install
```

3.1.2.8. Printing Switch Level of Hardware GDS

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

Standard Cell Editor

type:

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

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

II. Back-End Workflow Symbiflow for FPGAs

3.2. SOFTWARE

3.2.1. Compilers

type:

```
sudo apt install autoconf automake autotools-dev curl python3 libmpc-dev \
libmpfr-dev libgmp-dev gawk build-essential bison flex texinfo gperf \
libtool patchutils bc zlib1g-dev libexpat-dev
```

3.2.1.1. RISC-V GNU C/C++

type:

```
git clone --recursive https://github.com/riscv/riscv-gnu-toolchain

cd riscv-gnu-toolchain

./configure --prefix=/opt/riscv-elf-gcc
sudo make clean
sudo make

./configure --prefix=/opt/riscv-elf-gcc
sudo make clean
sudo make linux

./configure --prefix=/opt/riscv-elf-gcc --enable-multilib
sudo make clean
sudo make linux
```

3.2.1.2. RISC-V GNU Go

type:

```
git clone --recursive https://go.googlesource.com/go riscv-go
cd riscv-go/src
./all.bash
cd ../../
sudo mv riscv-go /opt
```

3.2.2. Simulators

type:

```
sudo apt install device-tree-compiler libglib2.0-dev libpixmap-1-dev pkg-config
```

3.2.2.1. Spike (For Hardware Engineers)

Building Proxy Kernel

type:

```
export PATH=/opt/riscv-elf-gcc/bin:${PATH}

git clone --recursive https://github.com/riscv/riscv-pk

cd riscv-pk
mkdir build
cd build
../configure --prefix=/opt/riscv-elf-gcc --host=riscv64-unknown-elf
make
sudo make install
```

Building Spike

type:

```
export PATH=/opt/riscv-elf-gcc/bin:${PATH}

git clone --recursive https://github.com/riscv/riscv-isa-sim

cd riscv-isa-sim
mkdir build
cd build
../configure --prefix=/opt/riscv-elf-gcc
make
sudo make install
```

3.2.2.2. QEMU (For Software Engineers)

type:

```
export PATH=/opt/riscv-elf-gcc/bin:${PATH}

git clone --recursive https://github.com/qemu/qemu

cd qemu
./configure --prefix=/opt/riscv-elf-gcc \
--target-list=riscv64-sofmmu,riscv32-sofmmu,riscv64-linux-user,riscv32-linux-user
make
sudo make install
```

4. CONCLUSION

4.1. HARDWARE

type:

```
cd synthesis/wb/qflow
source SYNTHESIZE-VERILOG-IT

cd synthesis/ahb3/qflow
source SYNTHESIZE-VERILOG-IT

cd synthesis/axi4/qflow
source SYNTHESIZE-VERILOG-IT
```

4.1.1. GSCL 45 nm ASIC

type:

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

cd synthesis/ahb3/qflow
source FLOW-VERILOG-IT

cd synthesis/axi4/qflow
source FLOW-VERILOG-IT
```

4.1.2. Lattice iCE40 FPGA

type:

```
cd synthesis/wb/symbiflow
source FLOW-VERILOG-IT

cd synthesis/ahb3/symbiflow
source FLOW-VERILOG-IT

cd synthesis/axi4/symbiflow
source FLOW-VERILOG-IT
```

4.2. SOFTWARE

4.2.1. RISC-V Tests

type:

```
export PATH=/opt/riscv-elf-gcc/bin:${PATH}

rm -rf tests
rm -rf riscv-tests

mkdir tests
mkdir tests/dump
mkdir tests/hex

git clone --recursive https://github.com/riscv/riscv-tests
cd riscv-tests

autoconf
./configure --prefix=/opt/riscv-elf-gcc/bin
```

```

make

cd isa

source ../../elf2hex.sh

mv *.dump ../../tests/dump
mv *.hex ../../tests/hex

cd ..

make clean

elf2hex.sh:
riscv64-unknown-elf-objcopy -O ihex rv32mi-p-breakpoint rv32mi-p-breakpoint.hex
riscv64-unknown-elf-objcopy -O ihex rv32mi-p-csr rv32mi-p-csr.hex
...
riscv64-unknown-elf-objcopy -O ihex rv64um-v-remw rv64um-v-remw.hex

type:
export PATH=/opt/riscv-elf-gcc/bin:${PATH}

spike rv32mi-p-breakpoint
spike rv32mi-p-csr
...
spike rv64um-v-remw

```

4.2.2. RISC-V Bare Metal

```

type:
rm -rf hello_c.elf
rm -rf hello_c.hex

export PATH=/opt/riscv-elf-gcc/bin:${PATH}

riscv64-unknown-elf-gcc -o hello_c.elf hello_c.c
riscv64-unknown-elf-objcopy -O ihex hello_c.elf hello_c.hex

C Language:
#include <stdio.h>

int main() {
    printf("Hello QueenField!\n");
    return 0;
}

type:
export PATH=/opt/riscv-elf-gcc/bin:${PATH}

spike pk hello_c.elf

type:
rm -rf hello_cpp.elf
rm -rf hello_cpp.hex

```

```

export PATH=/opt/riscv-elf-gcc/bin:${PATH}

riscv64-unknown-elf-g++ -o hello_cpp.elf hello_cpp.cpp
riscv64-unknown-elf-objcopy -O ihex hello_cpp.elf hello_cpp.hex

C++ Language:
#include <iostream>

int main() {
    std::cout << "Hello QueenField!\n";
    return 0;
}

type:
export PATH=/opt/riscv-elf-gcc/bin:${PATH}

spike pk hello_cpp.elf

type:
rm -rf hello_go.elf
rm -rf hello_go.hex

export PATH=/opt/riscv-elf-gcc/bin:${PATH}
export PATH=/opt/riscv-go/bin:${PATH}

GOOS=linux GOARCH=riscv64 go build -o hello_go.elf hello_go.go
riscv64-unknown-elf-objcopy -O ihex hello_go.elf hello_go.hex

Go Language:
package main

import "fmt"
func main() {
    fmt.Println("Hello QueenField!")
}

```

4.2.3. RISC-V Operating System

4.2.3.1. GNU Linux

Running Linux RISC-V 32 bit with Buildroot

```

type:
export PATH=/opt/riscv-elf-gcc/bin:${PATH}

git clone --recursive https://github.com/buildroot/buildroot

cd buildroot
make qemu_riscv32_virt_defconfig
make

qemu-system-riscv32 \
-M virt \
-nographic \

```

```
-bios output/images/fw_jump.elf \
-kernel output/images/Image \
-append "root=/dev/vda ro" \
-drive file=output/images/rootfs.ext2,format=raw,id=hd0 \
-device virtio-blk-device,drive=hd0 \
-netdev user,id=net0 \
-device virtio-net-device,netdev=net0
```

Running Linux RISC-V 64 bit with Buildroot

type:

```
export PATH=/opt/riscv-elf-gcc/bin:${PATH}
```

```
git clone --recursive https://github.com/buildroot/buildroot
```

```
cd buildroot
make qemu_riscv32_virt_defconfig
make
```

```
qemu-system-riscv64 \
-M virt \
-nographic \
-bios output/images/fw_jump.elf \
-kernel output/images/Image \
-append "root=/dev/vda ro" \
-drive file=output/images/rootfs.ext2,format=raw,id=hd0 \
-device virtio-blk-device,drive=hd0 \
-netdev user,id=net0 \
-device virtio-net-device,netdev=net0
```

4.2.3.2. GNU Hurd

4.2.4. RISC-V Distribution

4.2.4.1. GNU Debian

4.2.4.2. GNU Fedora

Running Fedora

type:

```
export PATH=/opt/riscv-elf-gcc/bin:${PATH}
```

```
qemu-system-riscv64 \
-nographic \
-machine virt \
-smp 4 \
-m 2G \
-kernel Fedora-RISCV.elf \
-bios none \
-object rng-random,filename=/dev/urandom,id=rng0 \
-device virtio-rng-device,rng=rng0 \
-device virtio-blk-device,drive=hd0 \
-drive file=Fedora-RISCV.raw,format=raw,id=hd0 \
-device virtio-net-device,netdev=usernet \
```



```
-netdev user,id=usernet,hostfwd=tcp::10000-:22
```