PU-OR1K WIKI

A Processing Unit (PU) is an electronic system within a computer that carries out instructions of a program by performing the basic arithmetic, logic, controlling, and I/O operations specified by instructions. Instruction-level parallelism is a measure of how many instructions in a computer can be executed simultaneously. The PU is contained on a single Metal Oxide Semiconductor (MOS) Integrated Circuit (IC).

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.

FRONT-END Open Source Tools

Library

```
sudo apt install libelf-dev
```

Verilator

SystemVerilog System Description Language Simulator

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

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

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

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

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

Icarus Verilog

Verilog Hardware Description Language Simulator

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

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

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

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

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

GHDL

VHDL Hardware Description Language Simulator

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

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

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

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

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

Yosys-ABC

Verilog Hardware Description Language Synthesizer

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

Hardware Description Language Optimizer

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

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

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

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

BACK-END Open Source Tools

Library

type:

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

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

```
mkdir qflow
cd qflow
```

Magic

Floor-Planner

A Floor-Planner of an Integrated Circuit (IC) is a schematic representation of tentative placement of its major functional blocks. In modern electronic design process floor-plans are created during the floor-planning design stage, an early

stage in the hierarchical approach to Integrated Circuit design. Depending on the design methodology being followed, the actual definition of a floor-plan may differ.

Standard Cell Checker

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

Standard Cell Editor

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

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

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

Graywolf

Standard Cell Placer

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

```
git clone https://github.com/rubund/graywolf
cd graywolf
mkdir build
cd build
cmake ..
make
sudo make install
```

OpenSTA

Standard Cell Timing-Analizer

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

```
git clone https://github.com/The-OpenROAD-Project/OpenSTA
cd OpenSTA
mkdir build
cd build
cmake ..
make
sudo make install
```

Qrouter

Standard Cell Router

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

```
git clone https://github.com/RTimothyEdwards/qrouter
cd qrouter
./configure
make
sudo make install
```

Irsim

Standard Cell Simulator

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

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

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

Netgen

Standard Cell Verifier

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

```
git clone https://github.com/RTimothyEdwards/netgen
cd netgen
./configure
make
sudo make install
```

Qflow

Back-End Workflow

```
git clone https://github.com/RTimothyEdwards/qflow
cd qflow
./configure
make
sudo make install
```

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

for WINDOWS users!

open Microsoft Store and install Ubuntu

FRONT-END

type:

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

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

```
sudo apt install yosys
```

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

BACK-END

Library

type:

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

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

type:

```
mkdir qflow
cd qflow
```

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

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

Basic parameters

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

Caching parameters

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

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

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

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
OPTION_FTOI_ROUNDING	Rounding behavior for `lf.ftoi.s`	CPP
FEATURE_BRANCH_PREDICTOR	Branch predictor implementation	SIMPLE

Note: *C/C++ double to integer conversion assumes truncation (rounding `toward zero`). The default (`CPP`) value of OPTION_FTOI_ROUNDING forces `toward zero` rounding mode exclusively for `lf.ftoi.s` instruction regardless of `rounding mode` bits of FPCSR. While with `IEEE` value `lf.ftoi.s` performs conversion in according with `rounding mode` bits of FPCSR. And these bits are `nearest-even` rounding mode by default. All other floating point instructions always perform rounding in according with `rounding mode` bits of FPCSR.*

Exception handling options

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

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

Parameter	Description	Default
FEATURE_CARRY_FLAG	Enable checking and setting the carry flag	`ENABLED`

Instruction enabling options

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