PU-FinTech WIKI

Definition

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).

An Automation Financial Method (AFM) is the technology and innovation that aims to compete with Traditional Financial Methods in the delivery of financial services. It is an emerging industry that uses technology to improve activities in finance. AFM is the new applications, processes, products, or business models in the financial services industry, composed of complementary financial services and provided as an end-to-end process via the Internet.

FRONT-END Open Source Tools

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
./configure
make
sh autoconf.sh
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

```
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

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

sudo apt install gcc-msp430
```

FRONT-END

type:

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

sudo apt install yosys
```

BACK-END

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
```