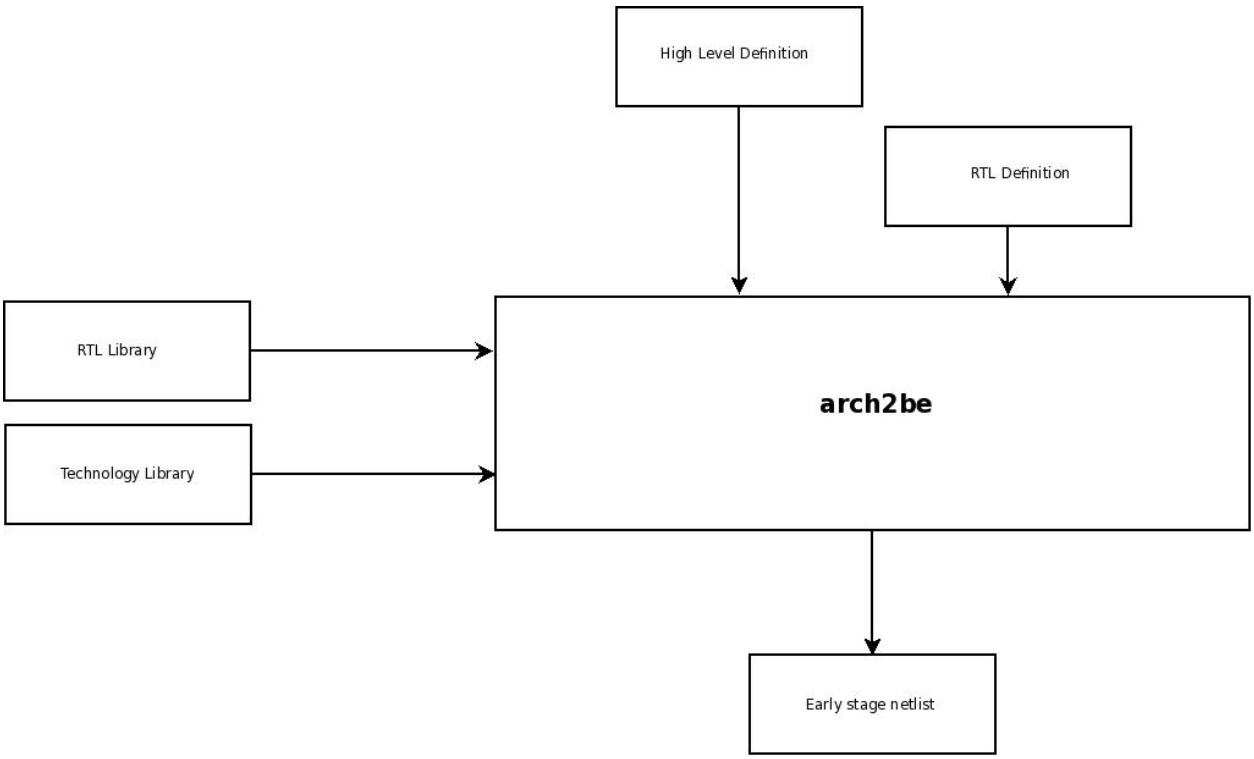
arch2be - Functional Specification Document

Nitij Mangal

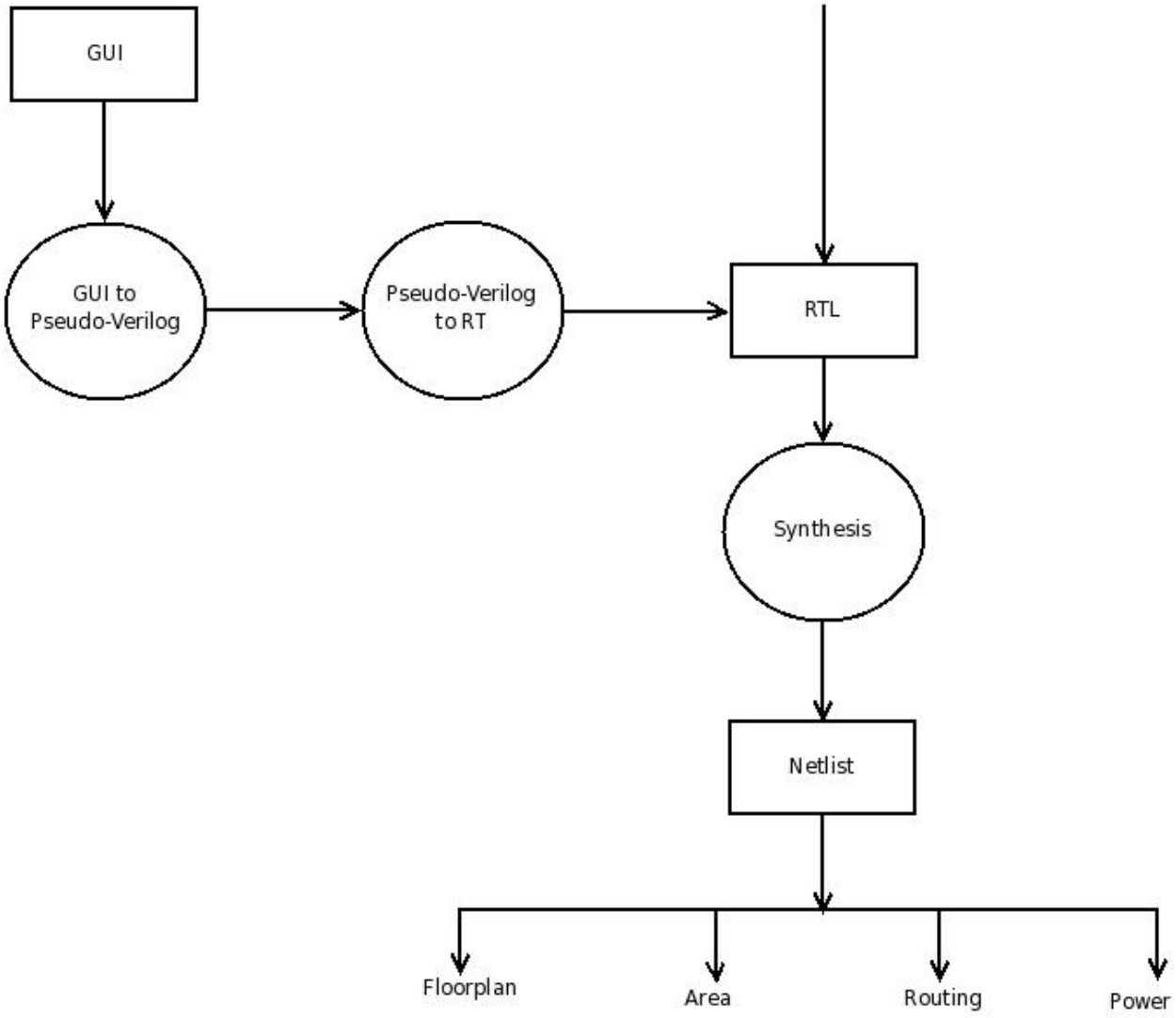
January 2009

1 Overview

arch2be is an EDA tool which enables VLSI designers to start the backend flow for a chip while the project is still in the early stage of architecture definition. It can convert a high-level architectural defintion to an early stage netlist which captures the global details of design - including area, top level routing, and power - which are of immense importance to the backend designers to start the P&R flow. The tool can take input both the high level definition as well as the RTL. As the RTL is developed over time, the high-level definition can be replaced by a more accurate RTL. This provides opportuinity to evolve the output netslist as well benchmark results over the development by seamlessy integrating **arch2be** into the entire design flow.



2 Tool flow



3 Structure of the tool

The tool is based on the concept of abstracting out information required to handle global backend issues - namely area, routing and power. There are 3 important stage of the tool.

3.1 Design Input

The tool provides two ways of specifying a design:

1. Create a new design using GUI
2. Read existing design
   (a) Created through GUI

(b) RTL

(c) Netlist

3.2 Design Abstraction

3.3 Design Output

4 Design Input Interface

arch2be provides a gui interface to the architects and front end designers to allow them to quickly define the high level functionality of their modules. The module based interface allows both top down and bottom up specification. A design is defined as a collection of objects, where each object is an instantiation of one of the following classes from the toolkit:

- Module
- Ports
- Flop Stage
- Datapath
- Combinational Logic
- Ram
- Fifo
- Connectivity

The top level object has to be a *Module. Each* Module object can have instantiations of objects of other classes.

4.1 Classes

The table below explains the toolkit available in detail along with the mandatory and optional parameters for each.

Class	Application	Required Parameters	Optional Parameters
Module	Create or instantiate modules	.name	.source (RTL/Netlist/GUI)
Port		.name .direction .width	
Flop Stage	Create flops or pipe stage State machine flops	.name .in .out .clk .rst	.clkedge (def:pos) .rstedge (def:neg)
Datapath	Create computational blocks	Operand (+/-/*) Width	Underyling structure (e.g.CLA)
CombiLogic	Random logic, state machine	No. of inputs/outputs	
Ram		Name Size (Width, depth)	No. of R/W/RW ports Default to 1R 1W Storage (ram, flop)
Fifo		Name Size (Width, depth)	Synch/Asynch (default: Synch) Storage (flop, ram)
Connectivity	Wires, buses	Name Width	

4.2 Typical GUI Design Input Session

A typical design definition session would consist of the following:

Steps	GUI Interface
Create module	Create a rectangular box to create a new module. Pops up a dialogue box to enter name
Create submodule	Create a rectangular box inside the module Pops up a dialogue box to enter name ⇒ create new module or link to library ⇒ instance of existing module
Create ports	Click and Drag (specify name and width)
Create flop stages	Click and Drag (width defaults to automatic)
Create datapath	Click and Drag (width defaults to automatic) add, multiply subtract; user can specify type e.g. CLA
Create combination logic	Click and Drag (specify complexity and number of stages)
Create connectivity	Use connectors (width defaults to automatic)

User can double click any object to specify or edit parameters.

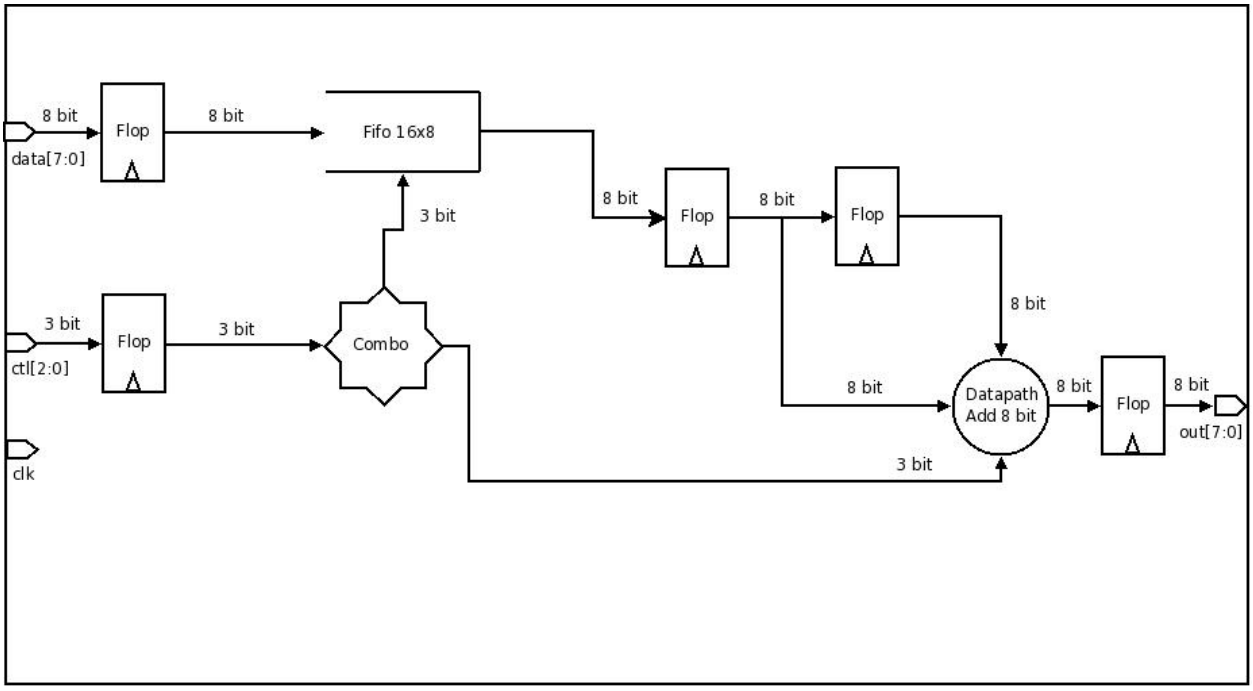
4.3 GUI Output

The output of GUI is stored in text format in a .gui file. Each object has type, parameters and position stored as
ObjectName Class (llx,lly) (urx,ury) [param=value]

The connectivity objects are stored in the different format to preserve the drawing. ***Rajeev to add routing format***

5 GUI to pesudoVerilog

The output of GUI is converted into a pseduoVerilog. The conversion takes place in two stages. The first stage is to convert connectivity objects into verilog style wires. Example:We will use to following example as our first design.



The GUI should output the following psuedo-verilog

```
module modulename;
input data[7:0];
input ctl[2:0];
input clk
output out[7:0];

::flop(.name=flop1 .in=data[7:0] .out=data_d1[7:0] .clk=clk .rst=reset_)
::flop(.name=flop2 .in=ctl[2:0] .out=ctl_d1[2:0] .clk=clk .rst=reset_)
::fifo(.name=fifo1 .iname=fifo1a .file=fifo.v .clk-clk .rst=reset_
.width=32 .depth=8 .in=data_d1[31:0] .out=fifo1_out[31:0]
.rd_en=ctl1 .wr_en=ctl2);
::combo.(name=combo1 .in=ctl_d1[2:0] .out=combo1_out[2:0])
::flop(.name=flop3 .in=fifo1_out[7:0] .out=fifo1_out_d1[7:0] .clk=clk .rst=reset_)
::flop(.name=flop4 .in=fifo1_out_d1[7:0] .out=fifo1_out_d2[7:0] .clk=clk .rst=reset_)
::datapath(.name=dp1 .op=+ .in=''fifo1_out_d1[7:0],fifo1_out_d2[7:0]''
.out=dp1_out[7:0] .clk=clk)
::flop(.name=flop5 .in=dp1_out[7:0] .out=out[7:0] .clk=clk .rst=reset_)

endmodule
```

5.1 Generating Verilog

To generate verilog from the pseudo language use the command p2v on the eQAtor command line. Typical usage is **p2v -p *PseudoVerilogFile* -v *VerilogOutputFile***. Optional arguments are -verbose -debug -norun -help

6 Next Steps

1. Specify GUI toolkit - mandatory and optional args
2. format to store GUI data
3. GUI to pseudoverilog
4. RTL interprettor
5. Synthesis tool
6. Module with known area vs. module with known functionality

7 Enhancement

1. Error checking