Carry Cut-Back Adder (CCBA) Source Code

This project provides the VHDL and scripting source for implementing and testing the Carry Cut-Back Adder (CCBA) circuit. If you use this code, please cite our article: V. Camus *et al.*, "Design of approximate circuits by fabrication of false timing paths: The carry cut-back adder," in IEEE JETCAS, 2018.

This project contains:

- the CCBA source in VHDL language to implement fully-custom or regular (i.e. with uniformly-sized elements and equally-spaced cuts) CCBA structures
- a VHDL top-level wrapper to set the desired architecture and structure of the CCBA adder to be simulated and synthesized
- a behavioral testbench to characterize the errors for 32-bit CCBA structures
- synthesis scripts for Synopsys Design Compiler and Cadence Genus
- a gate-level timing verification testbench to compare the synthesized netlist against its expected behavior
- simulation scripts for MentorGraphics Modelsim/Questa
- part of the Nangate 45nm cell library, a free predictive library developed by Nangate, provided with its own license in lib/NangateOpenCellLibrary_PDKv1_3_v2010_12

File structure

```
.
├── bench
│   ├── tb_adder32.vhd
│   ├── tb_adder32_gate.vhd
│   └── stimuli.txt
├── rtl
│   ├── ccba_pkg.vhd
│   ├── ccba.vhd
│   ├── ccba_regular.vhd
│   └── wrapper_ccba_regular_adder32.vhd
├── sim
│   └── sim_modelsim.tcl
├── sim_gate
│   └── sim_gate_modelsim.tcl
├── syn
│   ├── syn_cadence.tcl
│   ├── syn_synopsys.tcl
│   └── timing_constraints.tcl
├── lib
│   └── NangateOpenCellLibrary_PDKv1_3_v2010_12
│       ├── liberty
│       │   ├── NangateOpenCellLibrary_typical.lib
│       │   └── NangateOpenCellLibrary_typical.db
│       ├── verilog
│       │   └── NangateOpenCellLibrary.v
│       ├── LICENSE.md
│       └── README.md
├── LICENSE.md
├── README.md
└── README.pdf
```

How to run

Customize the CCBA design architecture and parameters

Optionally, you can customize the CCBA structure. The only file you need to modify is the rtl/wrapper_ccba_regular_adder32.vhd file in the CCBA parameter constant declarations. This will generate a regular CCBA structure, i.e. with uniformly-sized elements and equally-spaced cuts.

The CCBA parameters are:

- **ADDER_ARCH** (string) architecture between "multiplexed" and "input_induced"
- **CUT_NUMBER** (integer) number of cuts
- **CUT_SPACING** (integer) bit spacing between two cuts (for multiple cuts only, ignored otherwise)
- **CUT_1ST_POS** (integer) index of the first cut
- **PROP_WIDTH** (integer) bitwidth of the PROP blocks
- **ADD1_WIDTH** (integer) distance between the cuts and the PROP blocks
- **SPEC_WIDTH** (integer) bitwidth of the SPEC blocks (for multiplexed architecture, ignored otherwise)
- **CUT_TYPE** (char) guess/input-override type, between '0', '1', 'a' and 'b'

For a full explanation of the CCBA working principle, check our associated article: V. Camus *et al.*, "Design of approximate circuits by fabrication of false timing paths: The carry cut-back adder," in IEEE JETCAS, 2018.

Run a behavioral simulation using MentorGraphics Modelsim/Questa

1. Enter the sim directory.
2. Launch the sim_modelsim.tcl simulation script in command-line mode with:

```
vsim -c -do sim_modelsim.tcl
```

   or use the following command to open the GUI and display the waveforms and assertions (reject the default prompting to exit Modelsim at the end of the simulation):

```
vsim -do sim_modelsim.tcl
```

   You might need to adjust the launch commands to fit your EDA software infrastructure (e.g. running `questa` instead of `vsim`).

3. This testbench uses the random stimuli in bench/stimuli.txt to perform 10 thousands random additions. The error count is reported on screen by Modelsim/Questa. A file named results.txt is generated in the sim directory with the listing of the CCBA errors compared to an exact addition, in the form:

```
STIM_NB  EXACT                  APPROX                 ERROR_PATTERN
57       01100001100011000011111  01100001100011000010011  ..................100..
125      00011001111000011000000  00011001111000011000100  ...................1..
190      10111110001111010011101  10111110010001010011101  .........1000.........
221      10001100001111111001000  10001100001111111010000  ..................10..
```

Synthesize the design using Synopsys Design Compiler

1. Enter the syn directory.

2. Optionally modify the syn_synopsys.tcl synthesis script with the desired DELAY constraint in nanoseconds (default is 0.5), or with the desired standard-cell library file DB_FILE (default is the provided free 45nm NangateOpenCellLibrary).

3. Launch the syn_synopsys.tcl synthesis script in command-line mode with:

```
dc_shell -f syn_synopsys.tcl
```

   or open the `design_vision` GUI and run the synthesis script with:

```
source syn_synopsys.tcl
```

   You can then visualize the schematic and report additional information, such as specific timing paths: `report_timing -from a[2] -to s[8]`. You might need to adjust the launch commands to fit your EDA software infrastructure (e.g. running `design_vision -no_gui` instead of `dc_shell`).

4. Check in the generated reports.txt file for:
   - the applied delay constraints from the timing_constraints.tcl script
   - the timing report and slack
   - the area report
   - the power report

   If you have modified the DELAY constraint or customized the CCBA architecture, check the timing report and slack to see if the synthesized design fits the timing constraint.

5. The script exports the Verilog netlist and the SDF information in the files adder32.v and adder32.sdf.

Synthesize the design using Cadence Genus

1. Enter the syn directory.

2. Optionally modify the syn_cadence.tcl synthesis script with the desired DELAY constraint in nanoseconds (default is 0.5), or with the desired standard-cell library file LIB_FILE (default is the provided free 45nm NangateOpenCellLibrary).

3. Launch the syn_cadence.tcl synthesis script in command-line mode with:

```
genus -legacy_ui -f syn_cadence.tcl
```

   or open the GUI with `genus -legacy_ui -gui` and run the synthesis script with:

```
source syn_cadence.tcl
```

   You can then visualize the schematic and report additional information, such as specific timing paths: `report timing -from cin -to s[8]`. You might need to adjust the launch commands to fit your EDA software infrastructure (e.g. running `genus_gui` instead of `genus -gui`).

4. Check in the generated reports.txt file for:
   - the applied delay constraints from the timing_constraints.tcl script
   - the timing report and slack
   - the area report
   - the power report

   If you have modified the DELAY constraint or customized the CCBA architecture, check the timing report and slack to see if the synthesized design fits the timing constraint.

5. The script exports the Verilog netlist and the SDF information in the files adder32.v and adder32.sdf.

Run a gate-level timing simulation using MentorGraphics Modelsim/Questa

1. Enter the sim_gate directory.

2. If you have modified the DELAY constraint for the synthesis, do not forget to adjust it as well in the sim_gate_modelsim.tcl simulation script. The DELAY set in the simulation script overwrites the generic value in the testbench file.

3. Launch the sim_gate_modelsim.tcl simulation script with:

```
vsim -do sim_gate_modelsim.tcl
```

   which will open the GUI and display the waveform (reject the default prompting to exit Modelsim at the end of the simulation), or:

```
vsim -c -do sim_gate_modelsim.tcl
```

   which will run in command-line mode only.

4. This testbench uses the same random stimuli from bench/stimuli.txt to perform 10 thousands random additions. The error count is reported on screen by Modelsim/Questa. A file named results.txt is generated in the sim_gate directory with the listing of inconsistencies between the gate-level timed CCBA netlist and the behavioral wrapper instantiation, in the form:

```
STIM_NB  RTL                      GATE                     ERROR_PATTERN
67       11011011000000010000001  11011010000000010000001  .......0.............
395      10110111001101111111001  10110111001100111111001  ............0........
```

   If the gate-level netlist is correct and fits the timing constraint, there should not be any error. Do not hesitate to modify the DELAY at the beginning of the simulation script to check the level or location of the first timing errors, e.g. 0.48 instead of 0.5 ns.

License

This project is licensed under the BSD 3-Clause Clear License. See the LICENSE.md file from the main directory for rights and limitations.

A human-readable summary of (and not a substitute for) the BSD 3-Clause Clear License is:

- permissions:
  - private use
  - commercial use
  - distribution
  - modification
- conditions:
  - license and copyright notice must be included with the software
- limitations:
  - limitation of liability
  - does NOT grant any rights in the patents of contributors
  - does NOT provide any warranty

The Nangate 45nm Open Cell Library provided by default for CCBA synthesis and gate-level simulation was developed by Nangate and is provided under the Nangate Open Cell Library License, Version 1.0, February 20, 2008 (http://www.nangate.com/). See the LICENSE.md file in the lib/NangateOpenCellLibrary_PDKv1_3_v2010_12 directory for rights and limitations.

A human-readable summary of (and not a substitute for) the Nangate Open Cell Library License, Version 1.0 is:

- permissions:
  - private use
  - distribution
  - modification
- conditions:
  - license and copyright notice must be included with the library
- limitations:
  - limitation of liability
  - NOT suited for commercial purpose
  - measuring or benchmarking the library against another library is prohibited
  - does NOT provide any warranty

Acknowledgments

Thank you to Mattia Cacciotti, Marta Franceschi and Koen Goetschalckx for their help in reviewing and testing this code.