A Catalog-based AIG-Rewriting Approach to the Design of Approximate Components

Mario Barbareschi, Salvatore Barone, Nicola Mazzocca, and Alberto Moriconi

name.surname@unina.it

University of Naples Federico II

Outline

Approximate computing

Problems:

- Increasing volume of information to be processed
- Physical limits for ICs manufacturing process

Idea:

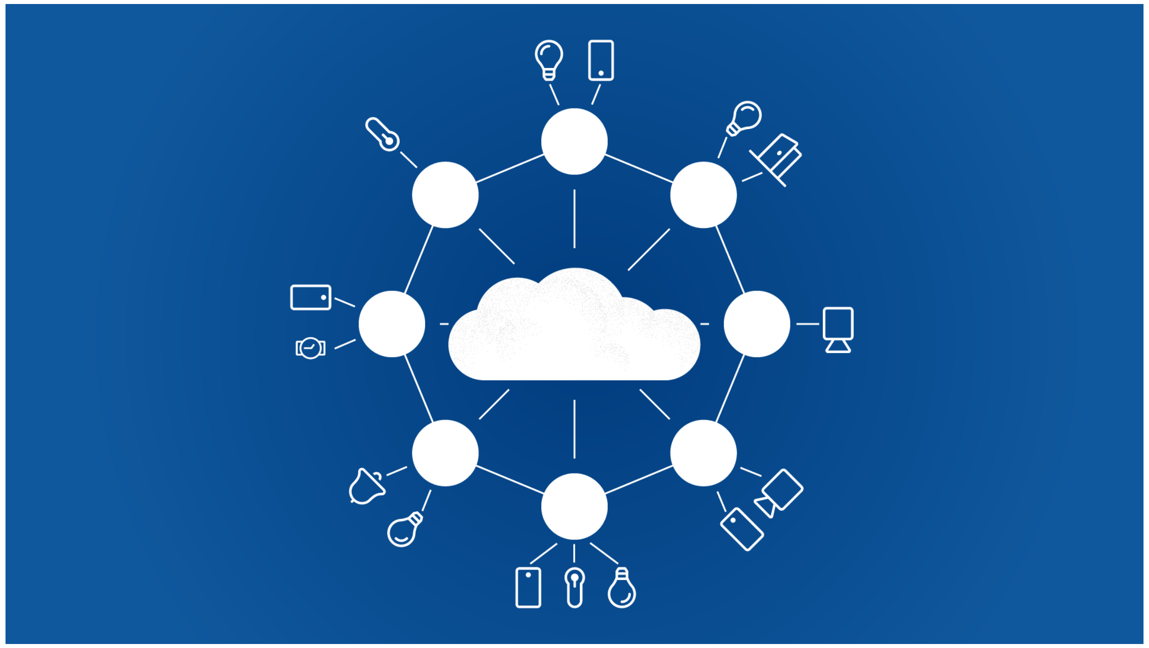
- Relax correctness contraints
- Exploit inherent error resiliency of specific problems

The well-known Baboon reference photo at different approximation levels

How:

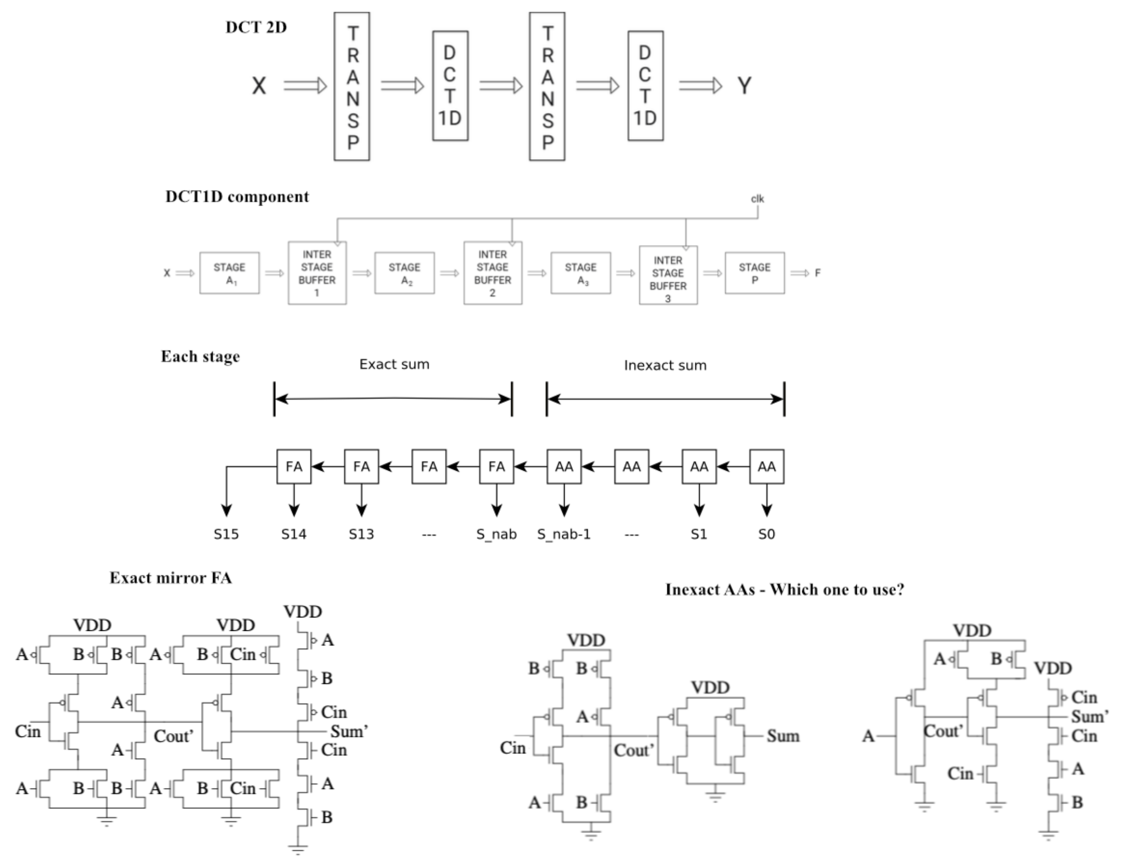
- A plethora of different techniques at the software and hardware level

Approximate circuits and hardware accelerators



- Can provide benefits in terms of energy, performance and area
- Interesting applications to hardware accelerators in embedded real-time systems
  - Applications to AI accelerators in the edge for resource-constrained devices
  - Enable satisfying otherwise unobtainable time constraints

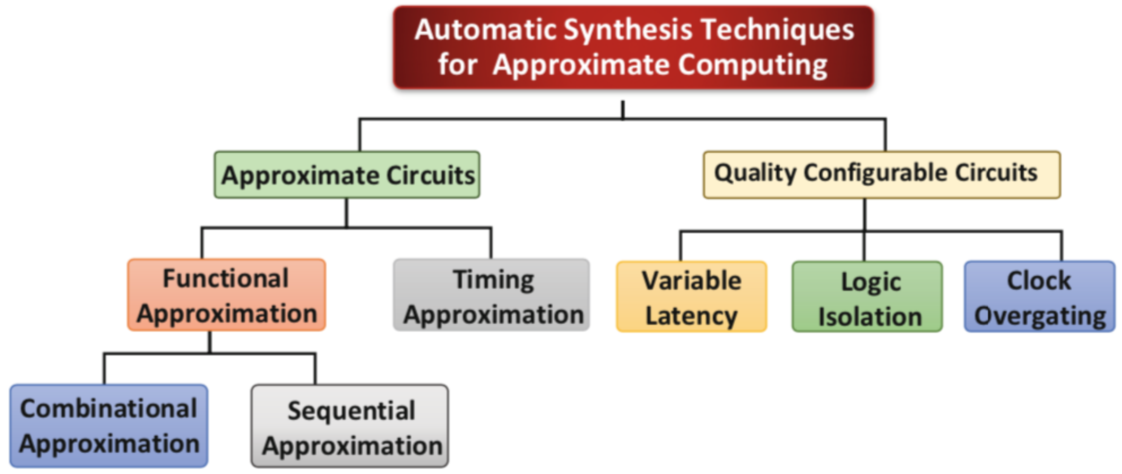
Approximate circuits and hardware accelerators



- Pioneering efforts have focused on finely hand-tuned circuits, especially arithmetic
- Adoption could benefit from systematic and automatic methodologies, starting from HDLs and desired constraints

Related works

- Many possible techniques!



- We are interested in functional approximation, the modification of the logic implemented by the circuit; some techniques used at this level are:
  - Modifications of classical coverage methods (Shin et al.)
  - Fault injection (Wu et al.)
  - Fusion of nodes with similar functionality
  - ...
- We will focus on combinational circuits described in HDL

Research aims

We aim to provide a methodology and a tool that is:

- Automatic

Research aims

We aim to provide a methodology and a tool that is:

- Automatic
- Generic, can be applied to all kinds of circuits

Research aims

We aim to provide a methodology and a tool that is:

- Automatic
- Generic, can be applied to all kinds of circuits
- Independent from the error metric (they can be selected from a library or provided by the user, if needed)

Research aims

We aim to provide a methodology and a tool that is:

- Automatic
- Generic, can be applied to all kinds of circuits
- Independent from the error metric (they can be selected from a library or provided by the user, if needed)
- Assists in exploring the design space, providing not a single solution but an estimation of good design trade-offs between accuracy and performance gain

Research aims

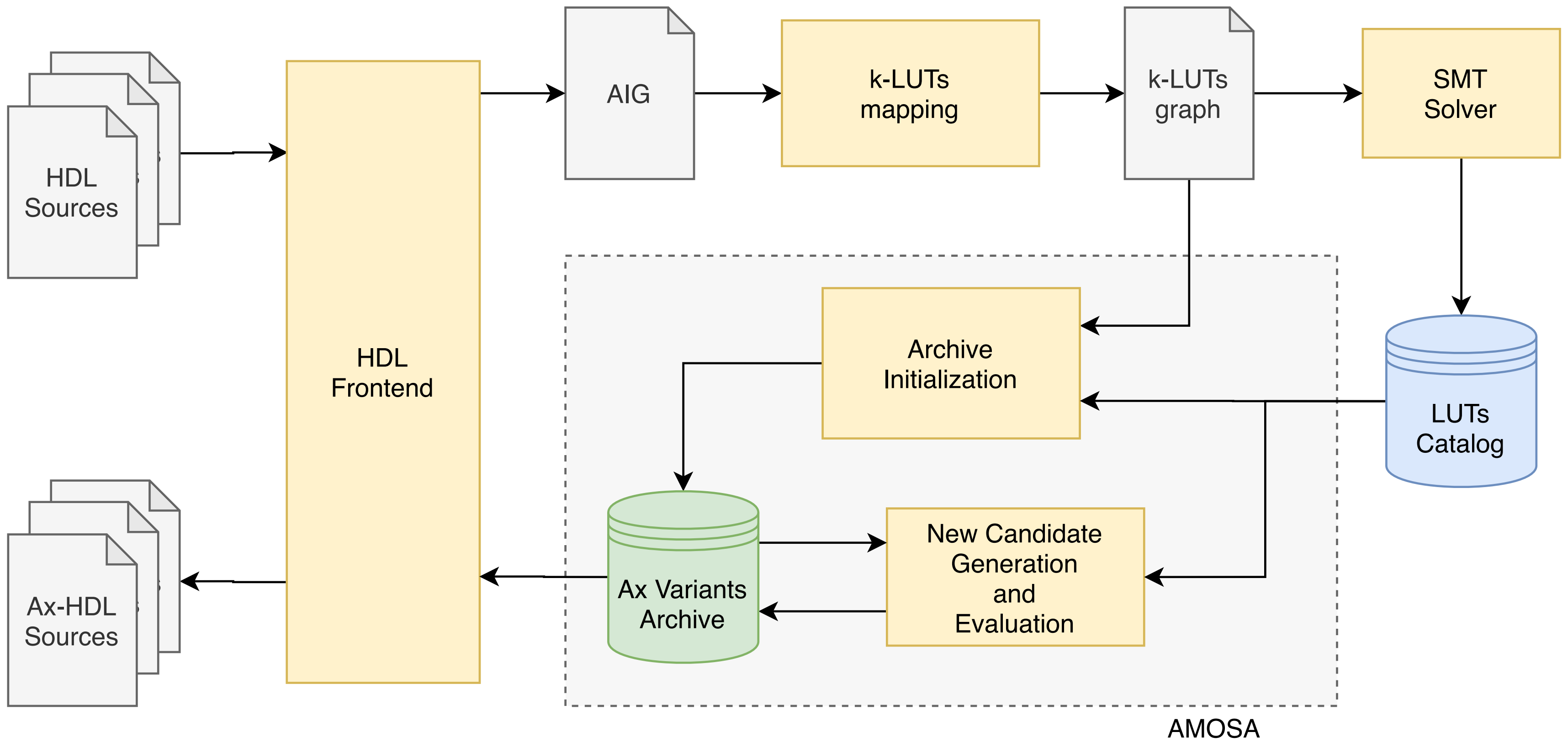
We aim to provide a methodology and a tool that is:

- Automatic
- Generic, can be applied to all kinds of circuits
- Independent from the error metric (they can be selected from a library or provided by the user, if needed)
- Assists in exploring the design space, providing not a single solution but an estimation of good design trade-offs between accuracy and performance gain
- Exploits functional approximation in a way that doesn't cut off big parts of the design space

Our flow in brief

Basically, our approximation approach consists in:

1. Breaking the original circuit in k-LUTs
2. Synthesize them to AIGs at different degrees of approximation, building a catalogue
3. Solve the multi-objective combinatorial optimization problem of choosing the best entries from the catalogue



The pyALS tool

`https://github.com/SalvatoreBarone/pyALS`

It's distributed under GPL-3.0 and leverages some great open source software and libraries:

- Implemented as a plug-in for the Yosys Open Synthesis Suite
- Can be integrated with GHDL for VHDL parsing and synthesis

You can use the included Docker image to get started quickly!

M. Barbareschi, S. Barone, N. Mazzocca and A. Moriconi, "A Catalog-based AIG-Rewriting Approach to the Design of Approximate Components," in IEEE Transactions on Emerging Topics in Computing, doi: 10.1109/TETC.2022.3170502.

Exact synthesis

Our approach is based on the concept of exact synthesis. At its core, it's a pretty straightforward idea: find a combinational circuit that

- Implements a given specifications
- Is optimal w.r.t. some cost criteria, usually the number of nodes and/or the circuit depth

Ideally, an exact synthesis algorithm:

- Takes as input a circuit specification, e.g. a truth table
- Gives as output the smallest (or the fastest, or...) circuit that implements such specification

Satisfiability Modulo Theories

Modern SMT solvers enable us to:

- Write mathematical constraints over a set of variables
- Find a variable assignment that satisfies such constraints, if it exists

They are not a panacea, however: they tend to scale pretty poorly; so we need to find a circuit representation that:

- Is easy to manipulate mathematically, in order to express constraints
- Does not get too big when circuit size increases

We choose AND-inverter graphs for our methodology because they scale well with circuit size and well supported. Notably, ABC is an academic, state of the art software system, developed at Berkeley, that uses AIGs for synthesis, mapping and verification.

A word on simulated annealing...

Our optimization heuristic is based on simulated annealing, a well known search algorithm based on the physical metaphor of the annealing process of solids:

- We try to minimize an energy function associated to our circuit variants
  - It actually is a vector valued energy, because it keeps track of both error rate and number of gates

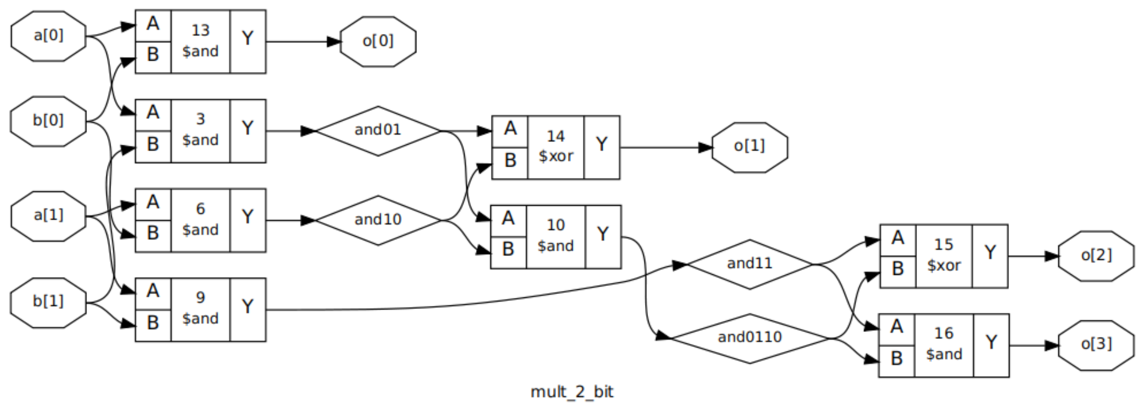
We choose the AMOSA algorithms because it's a multi-objective version that progressively builds an archive of solution that approximate the Pareto front.

Bandyopadhyay, Sanghamitra, et al. "A simulated annealing-based multiobjective optimization algorithm: AMOSA." IEEE transactions on evolutionary computation 12.3 (2008)

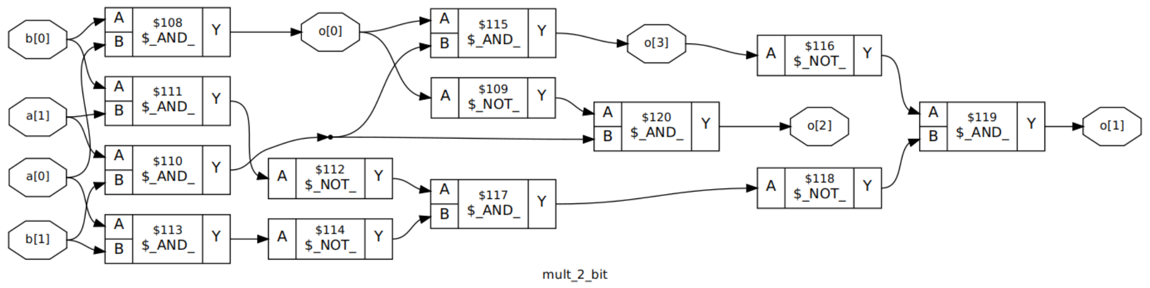
An example circuit

mult_2_bit.vhd

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity mult_2_bit is
  port(
    a : in  std_logic_vector (1 downto 0);
    b : in  std_logic_vector (1 downto 0);
    o : out std_logic_vector (3 downto 0)
  );
end entity mult_2_bit;

architecture dataflow of mult_2_bit is
  signal and01   : std_logic;
  signal and10   : std_logic;
  signal and11   : std_logic;
  signal and0110 : std_logic;
begin
  and01   <= a(0)  and b(1);
  and10   <= a(1)  and b(0);
  and11   <= a(1)  and b(1);
  and0110 <= and01 and and10;

  o(0)  <= a(0)  and b(0);
  o(1)  <= and01 xor and10;
  o(2)  <= and11 xor and0110;
  o(3)  <= and11 and and0110;
end architecture dataflow;
```



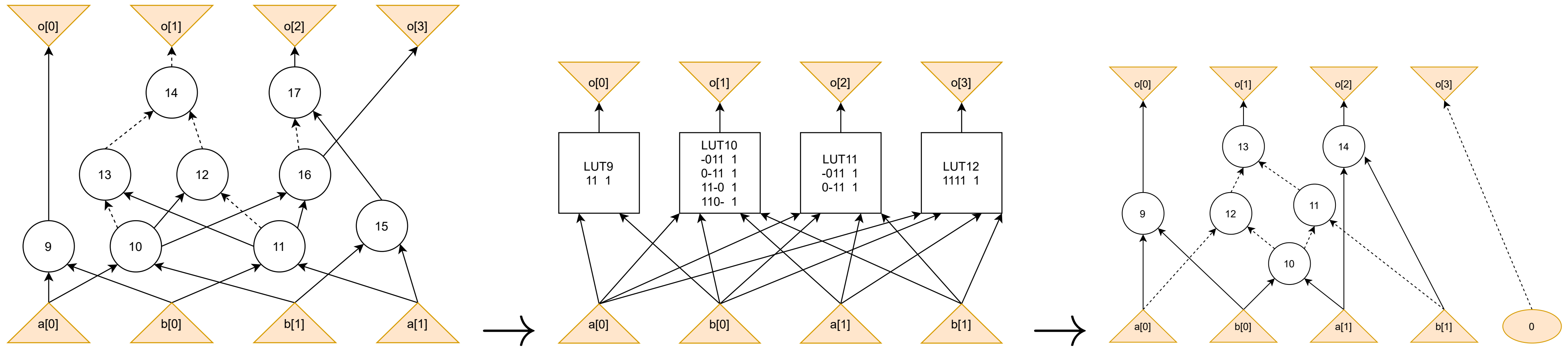
The 2 bit multiplier



Its AIG representation

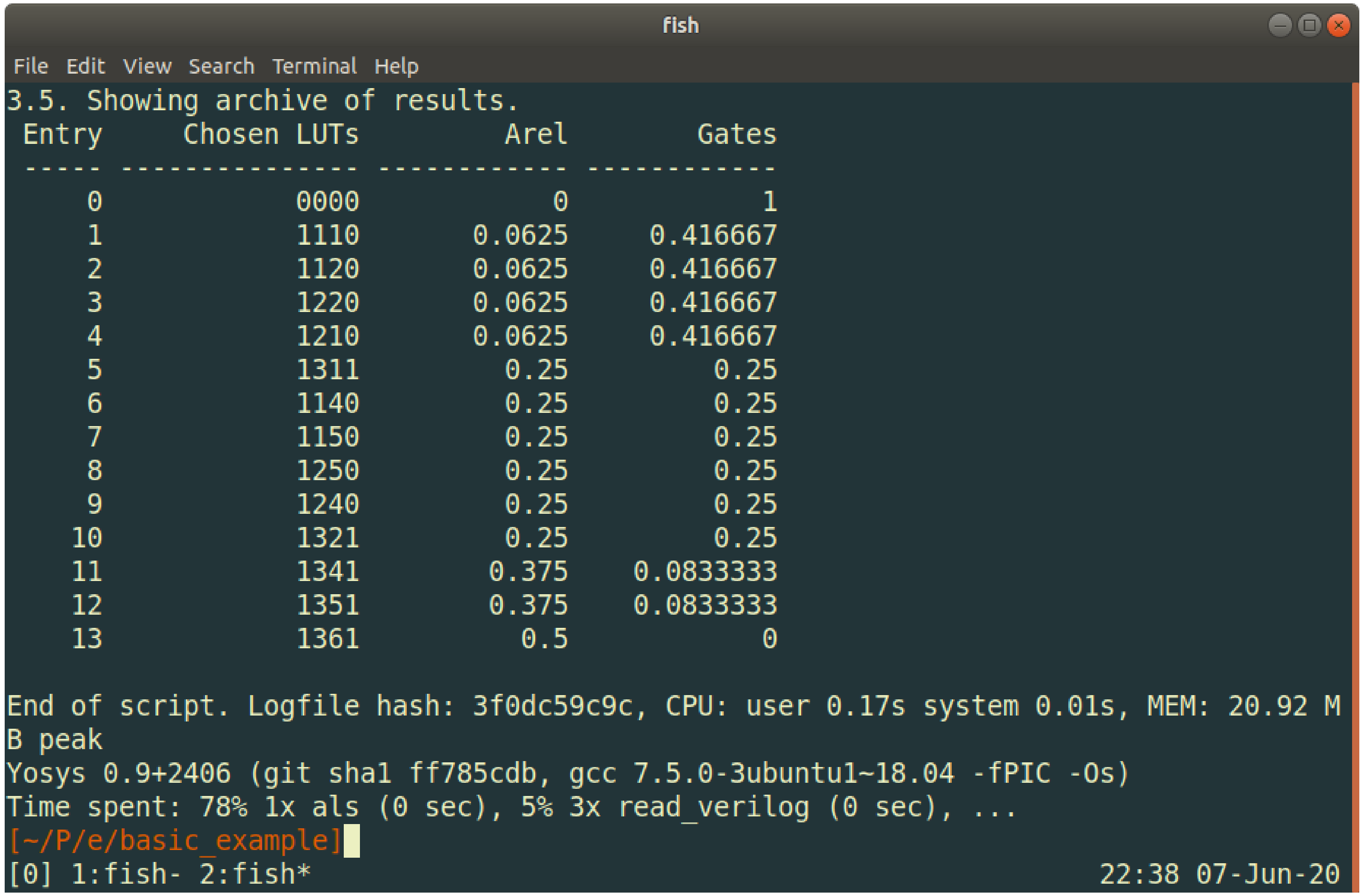
Getting it all together

For the two bit multiplier of our example, a possible optimization step could be this:



Getting it all together

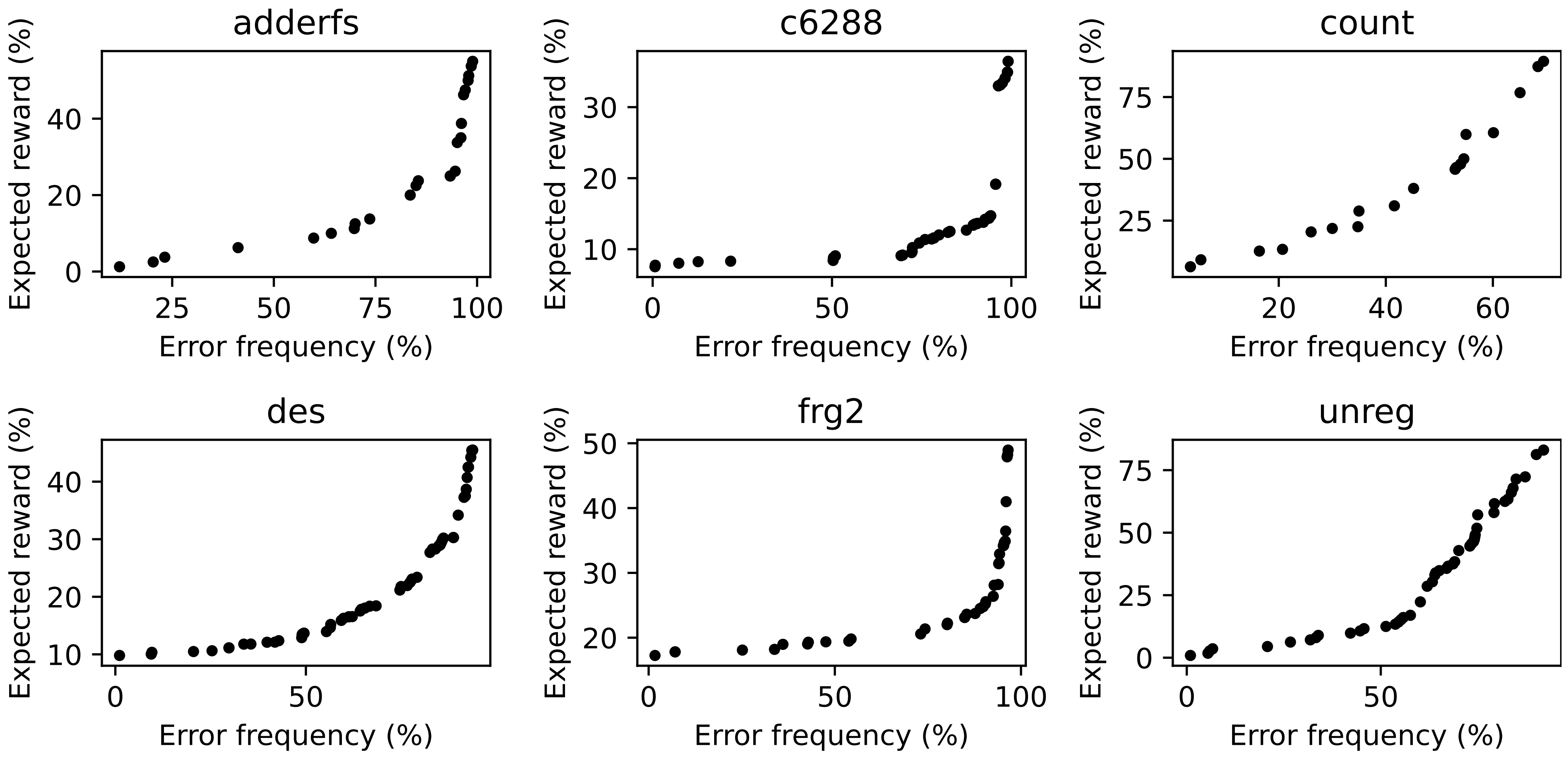
And a possible tool output could be this:



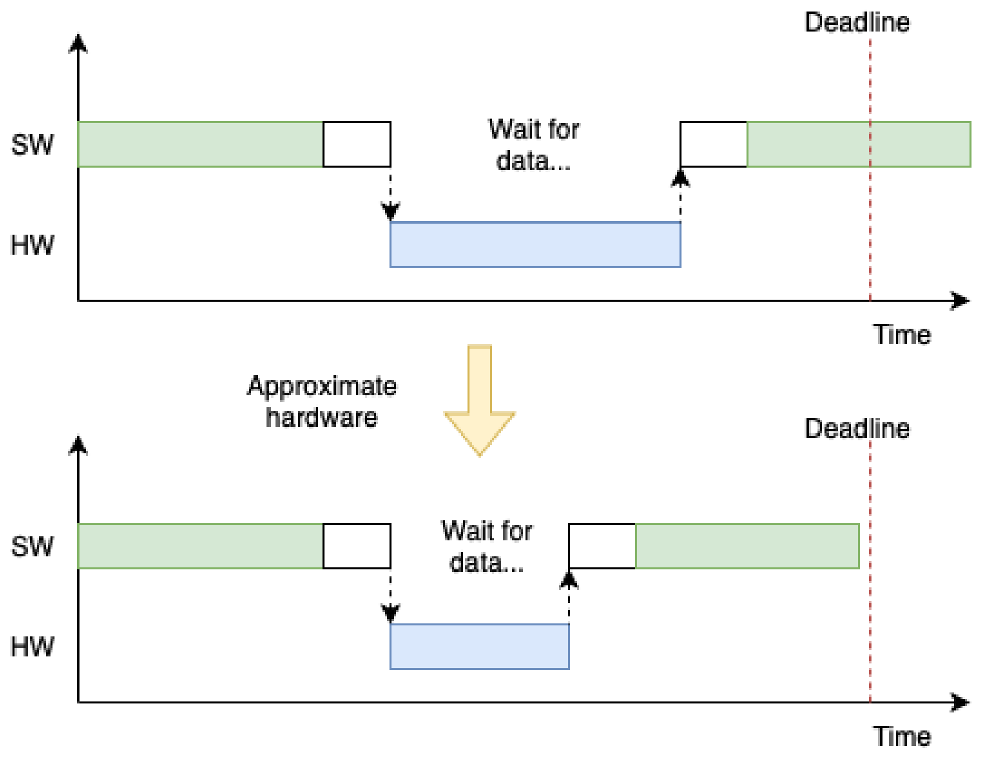
```
3.5. Showing archive of results.
 Entry      Chosen LUTs         Arel        Gates
 ----- --------------- ------------ ------------
     0            0000            0            1
     1            1110       0.0625     0.416667
     2            1120       0.0625     0.416667
     3            1220       0.0625     0.416667
     4            1210       0.0625     0.416667
     5            1311         0.25         0.25
     6            1140         0.25         0.25
     7            1150         0.25         0.25
     8            1250         0.25         0.25
     9            1240         0.25         0.25
    10            1321         0.25         0.25
    11            1341        0.375    0.0833333
    12            1351        0.375    0.0833333
    13            1361          0.5            0

End of script. Logfile hash: 3f0dc59c9c, CPU: user 0.17s system 0.01s, MEM: 20.92 M
B peak
Yosys 0.9+2406 (git sha1 ff785cdb, gcc 7.5.0-3ubuntu1~18.04 -fPIC -Os)
Time spent: 78% 1x als (0 sec), 5% 3x read_verilog (0 sec), ...
[~/P/e/basic_example]
[0] 1:fish- 2:fish*                                                   22:38 07-Jun-20
```

Benchmark - LGSynth91

We tested our methodology and tool on the well-known LGSynth91 combinational circuit synthesis benchmark collection, that contains both arithmetic and generic logic circuits, using an error frequency metric:



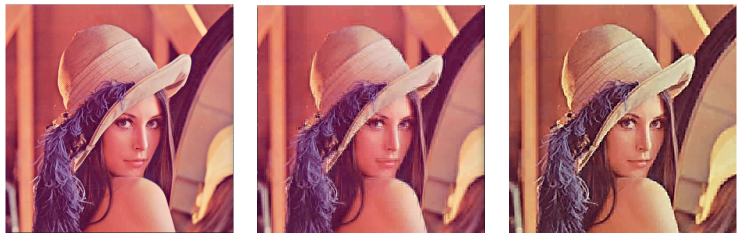
Case study - Hardware accelerators and real-time systems



- Typical scenario in embedded real-time applications that require some tasks to be hardware-accelerated
  - Edge AI
  - Image compression
  - Image processing and filtering
- Optimize w.r.t. execution time of an hardware task
- Our tool can be used in conjunction with WCET evaluation tools
- Relaxing correctness constraints allows otherwise unobtainable schedules of hardware tasks!

Case study - JPEG compression

	Exact Circuit			Minimum Error Configuration						Minimum Area Configuration					
Circuit	Gates	FPGA LUTs	Power (mM)	Error	Expected Reward (%)	FPGA LUTs	Area Savings (%)	Power (mW)	Power Savings (%)	Error	Expected Reward (%)	FPGA LUTs	Area Savings (%)	Power (mW)	Power Savings (%)
C6288	2406	754	0.47	0	1.00	750	0.53	0.467	1.06	4.26E+09	37.00	484	35.81	0.28	41.31
f51m	43	12	0.10	1	30.00	8	33.33	0.09	5.05	7.50E+01	65.00	5	58.33	0.08	24.24
z4ml	20	6	0.08	1	8.00	6	0.00	0.08	0.00	7.00E+00	42.00	1	83.33	0.06	23.17
8x8 bits Dadda multiplier	383	141	0.21	0	19.00	114	19.15	0.17	17.39	2.38E+02	43.00	39	72.34	0.10	51.69
8x8 bits array multiplier	420	163	0.20	1	16.00	155	4.91	0.19	3.57	2.52E+02	45.00	48	70.55	0.10	46.94
8x8 bits Wallace multiplier	398	149	0.20	0	12.00	132	11.41	0.17	13.93	2.50E+02	33.00	69	53.69	0.14	32.34
16 bits carry select adder	138	44	0.18	0	2.00	44	0.00	0.17	5.68	4.10E+04	19.00	42	4.55	0.16	6.82
16 bits carry-skip adder	128	39	0.17	0	4.00	39	0.00	0.17	1.16	1.78E+04	22.00	19	51.28	0.14	20.93
16 bits Han-Carlson adder	120	38	0.17	1	1.00	38	0.00	0.16	5.23	6.17E+04	36.00	23	39.47	0.13	22.67
8 bits carry-lookahead adder	54	24	0.61	1	9.00	16	33.33	0.11	81.64	3.43E+02	60.00	4	83.33	0.08	87.21
8 bits ripple carry adder	59	19	0.12	1	4.00	13	31.58	0.12	0.00	1.45E+02	30.00	13	31.58	0.11	6.96
8 bits Han-Carlson adder	53	18	0.12	1	3.00	13	27.78	0.12	0.00	3.77E+02	61.00	6	66.67	0.03	26.09


(a) Visual test with Lena

(b) Visual test with Baboon

Visual test. Floating-point DCC on the left; in the center, the multiplier-less BAS08 algorithm (Bouguezel et al.) is used. On the right, the BAS08 algorithm is further approximated using approximate adders, obtaining a 25% LUT saving (from 566 to 422 LUTs on Zynq 7000 SoC) and similar reductions in execution time.

Conclusions

- Methodology based on exact synthesis of selected k-cuts
- Allows tackling a number of relevant problems in hardware acceleration for real-time hardware tasks such as image processing and AI in the Edge
- An open source implementation is available
- Area savings up to 60% at minimum error for selected circuits and up to 25% in a realistic case study

Thank you!

Thank you for your attention!