Unusual/Unexpected

Katerina Borodina-Petrovic
Olena Nesterenko
Ben Lichtman
How to Set Your FPGA on Fire
Eight Ways to put your FPGA on Fire –
A Systematic Study of Heat Generators

Markus Happe, Hendrik Hangmann, Andreas Agne, Christian Plessl
Computer Engineering Group
University of Paderborn, Germany
Email: markus.happe@upb.de, hhangman@mail.upb.de, {agne, christian.plessl}@upb.de

Due to the continuously shrinking device structures and densities of FPGAs, thermal aspects have become
for many research projects over the last years. Most
system temperature simulations are not able to their
temperature simulation may consume a prohibitively
amount of computation time.

Another approach is the implementation of the system
Using novel techniques, the researchers heated an FPGA up to 195°C.
An increase of 135°C

In 12 minutes

Using only 21% of the FPGA’s processing power.

Katerina Borodina
But Why?
Is TESTING 
SUPER HOT FPGAS

<THIS UGLY
SON OF A Block RAM

How? ...Just Watch
The Free Video >
Previous researchers:

- Used a pipeline of look-up tables (LUTs)
- Used 100% of the LUT slices
- Achieved only 55°C
  - pretty much an average summer day
How to do better?

We can generate heat by toggling as many signals and/or storage elements as possible. Some of the options available:

1. A pipeline of LUTs
2. LUT oscillator
3. Shift register lookup table pipelines
4. Flip-flop (FF) pipeline
5. LUT-FF pipeline
LUT Pipeline

- Connected 6 LUTs
- Toggled on and off
- 14 pipelines in total
- 100MHz clock signal

Fig. 1. LUT-based pipeline
LUT Pipeline Results

- "Achieved" 3°C increase in 700 seconds
- Sad
SRL Pipeline Results

- 4°C - 14°C increase in 700s
- 300 MHz had the best results
- They tried over 300 MHz but the SRLs did not “toggle their signals reliably anymore”
Flip-flop Pipeline

- Cascade flip-flops to build a shift register similar to the SRL used before
- 14 pipelines
Flip-flop Pipeline Results

- 5°C - 22°C in 700 seconds
- Almost reaching 100°C - starting to get somewhere
LUT-FF Pipeline Results

- Past 100°C! Awesome!
- Combining LUTs and FFs gives better results than using them individually!
LUT Oscillator

- An odd number of inverters are connected to each other
- The signal becomes unstable
LUT Oscillator

![LUT Oscillator Diagram](image)

- temperature (°C)
- time (sec.)

Katerina Borodina
LUT Oscillator

As can be seen in Figure 8(b) this heater already heats up the FPGA to 195°C according to the built-in thermal diode. We restricted our experiments to 1,000 ring oscillators in order to prevent the destruction of the FPGA. To be able to compare the diverse LUT and FF-based heaters with each other, we have used the same amount of slices for each LUT and FF-based heater.
An Evolved Circuit,
Intrinsic in Silicon,
Entwined with Physics
【The experiment】

Discriminate between 2 square waves using an FPGA

1 kHz

Output 5V

10 kHz

Output 0V
Hey that sounds easy

Try doing it without a clock

Cell Input $\rightarrow$ < 5 ns $\rightarrow$ Cell Output

(Period of a 10 kHz wave is 0.1ms)

+ Using only a 10x10 corner of the chip
【The solution】

Evolution

(A conventional generational Genetic Algorithm)

1. Randomly generate 50 circuits
2. Evaluate performance of each circuit
3. Keep top performing circuit unchanged
4. Derive 49 new circuits (more details soon)

Repeat until performance is satisfactory
How can we make sure the 49 new circuits we are deriving are better than those in the previous generation?

Two parents chosen using linear rank-based selection:

1. Circuit A (weight = 3)
2. Circuit C (weight = 2)
3. Circuit B (weight = 1)

Weight is inversely proportional to rank
How can we make sure the 49 new circuits we are deriving are better than those in the previous generation?

Two parents combined using:

1. **Crossover**
   Probability = 0.7

2. **Mutation (Per-bit)**
   Probability set such that 2.7 mutations expected per new circuit
The circuits are always tried out ‘for real’ rather than in simulation.
【 Fitness Evaluation 】

For each circuit tested:
- Circuit input was 10 bursts of 500ms square waves
- 5 x 1 kHz waves and 5 x 10 kHz waves
- Order of waves randomized

\[
\text{fitness} = \frac{1}{10} \left| \left( k_1 \sum_{t \in S_1} i_t \right) - \left( k_2 \sum_{t \in S_{10}} i_t \right) \right|
\]

where \( k_1 = \frac{1}{30730.746} \quad \text{and} \quad k_2 = \frac{1}{30527.973} \)

Maximise the difference between average output voltage @ 1 kHz input and average output voltage @ 10 kHz input
## Results

<table>
<thead>
<tr>
<th>1kHz</th>
<th>10kHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>1kV</td>
<td><img src="image1" alt="Voltage Waveform 1kHz" /> <img src="image2" alt="Voltage Waveform 10kHz" /></td>
</tr>
<tr>
<td>0</td>
<td><img src="image3" alt="Voltage Waveform 1kHz" /> <img src="image4" alt="Voltage Waveform 10kHz" /></td>
</tr>
<tr>
<td>220V</td>
<td><img src="image5" alt="Voltage Waveform 1kHz" /> <img src="image6" alt="Voltage Waveform 10kHz" /></td>
</tr>
<tr>
<td>320V</td>
<td><img src="image7" alt="Voltage Waveform 1kHz" /> <img src="image8" alt="Voltage Waveform 10kHz" /></td>
</tr>
<tr>
<td>650V</td>
<td><img src="image9" alt="Voltage Waveform 1kHz" /> <img src="image10" alt="Voltage Waveform 10kHz" /></td>
</tr>
<tr>
<td>1kV</td>
<td><img src="image11" alt="Voltage Waveform 1kHz" /> <img src="image12" alt="Voltage Waveform 10kHz" /></td>
</tr>
</tbody>
</table>

Olena Nesterenko
The final circuit

Functional parts
1. It took **2-3 weeks** to run the experiment

2. This is a paper from **1996**

3. Final result is **specific to hardware**

4. A possible use case: **deep space probes**?
Riddle Me This.
2004
Xilinx gets reports of FPGAs changing configuration over time
Wants to study how its FPGAs are affected by single-bit errors
Turns out that every ~230 hours there is a single bit flip on CRAM on some devices
Bit flips were proportional to number of solder balls places over CRAM
WTF?
The answer? Nukes.

- Yes, nukes
- Because after the first nuke was detonated it spread uranium products everywhere, including Pb
- Put radioactive Pb 210 alongside stable Pb 206
- Now all newly smelted lead may occasionally emit alpha particles which flip bits
- Impossible to avoid - it's in the air and soil - everywhere

Ben
Lead is radioactive. So what?

- Olden days:
  - Bond wires attached to silicon die, bond wires attached to metal legs, the chip encased in plastic, solder attached to legs
- Newden days:
  - Silicon die metallised, solder applied directly to the chip
- Solder is made of Pb
- Radioactive Pb gets into the solder and zaps yo bits
- Now what?

Ben
We needed a source of “low alpha” Pb for soldering critical error sensitive components.

Therefore needed lead smelted pre-WWII.

Good source of this? Old ships:
- Used to use lead ballast in the bottom of ships to weigh the bottom down and stop them flipping over.
- Lots of old ships used lead cladding to stop water getting in and to protect them.
- The older the lead the more of the radioactive stuff would have decayed.

Was actually done for a physics experiment.

Ben
Error correction

● Advantages:
  ○ Don’t have to harvest ships (completely unsustainable)
  ○ Even pirate solder doesn’t protect FPGAs from cosmic rays
  ○ Better for space applications

● Disadvantages:
  ○ Not as cool as pirate ships
  ○ Uses more space
  ○ Less computationally efficient
Easiest way: Use a vote

- Use best of 3 to correct 1 bit errors
- Or use best of 4 to correct 1 bit and detect 2 bit errors
- Called “Triple module redundancy”
- Saturn V rocket used this a lot
  - 7 stage data pipeline
  - Each stage was duplicated 3 times
  - “Vote” taken between each pipeline stage to avoid error
Could you do this in design project B?

- Some FPGAs and synth tools come with redundancy features
- Or just do it manually

Downside: must trust your “voting” components

SPACE AGE TECHNOLOGY AT YOUR FINGERTIPS

What about if it’s just for data?

Ben
Parity

- What is a parity bit?
- What is its purpose?
- Can we do better?
- CRC?
Hamming code

- How most error correction is done in practice
- Good for signals, not really good for static circuitry
- More efficient than triple module redundancy
- The more data you send the fewer redundant bits you need to send
- Most common implementation is [7, 4] hamming code
  - 4 data bits -> 7 coded bits
  - Can correct 1 bit errors and detect 2 bit errors
- Easily implemented with bitwise matrices - suitable for FPGAs
How it works

- Parity bits - convention: make the message even
- Make sure each pattern of parity errors corresponds to only one error possibility
- Balance space efficiency with computational complexity
Questions? Comments?