A DESIGN SPACE EXPLORATION FOR A FLEXIBLE EFPGA-BASED NEUROMORPHIC PROCESSING CORE

UNIVERSITY OF TWENTE.

B. ELSHINNAWEY(1)
BELALSHINNAWEY@GMAIL.COM

A. YOUSEFZADEH(1)
A.YOUSEFZADEH@UTWENTE.NL

S. SOHAIL(1)
SAMEED.SOHAIL@UTWENTE.NL

(1) UNIVERSITY OF TWENTE

HeiChips 2026

INTRODUCTION

Neuromorphic computing is a rapidly evolving field of research. Earlier research focused on replicating biologically accurate computational dynamics [1], whereas contemporary research adopts a brain-inspired yet practical approach to algorithm design. In addition to algorithm design, neuromorphic systems often require specialized input encoding and pre-processing schemes that are unique to the application, leading to lower performance in inflexible designs. This shift promoted the development of neuromorphic computing designs built with flexibility in mind. This flexibility often led to improved performance metrics despite the added overhead [2]. Instead of adopting a micro-kernel or a general-purpose processor design common in neuromorphic hardware, we explored the use of embedded field-programmable gate arrays (eFPGAs) for flexible neuromorphic computing [3]. We designed an eFPGA fabric using the FABulous eFPGA framework [4], to accelerate neuromorphic workloads, incorporated an event-activated clock network for event-based workloads, designed logic elements that enabled higher-frequency computations, and a hard in-fabric macro inspired by neuromorphic design principles to handle memory and computation. We synthesized the design using the open-source SKY130 process and benchmarked its performance against a comparable ASIC accelerator. Our results demonstrate that the eFPGA achieves comparable ASIC performance in power, latency, and throughput while outperforming similar open-source eFPGA implementations [3].

OBJECTIVES

How can we optimize the eFPGA design for flexibility while maintaining as much performance as possible compared to an ASIC design?

Sub-Objectives:

1. What are the architecture elements for the eFPGA core that allow for flexible computing while approaching the performance of ASIC-based designs?
2. What flexibility aspects can an eFPGA core achieve for a neuromorphic processing core?
3. How do the architectural elements that add flexibility affect the performance metrics (area and power) of the eFPGA core?

METHODOLOGY

In our design exploration, we follow an iterative process, where we start from a generic FPGA fabric with LUT-4 logic cells. We then test different bit-stream configurations from workloads related to neuromorphic processing. The result is used to define the fabric size and content. The process is outlined in Figure 1.

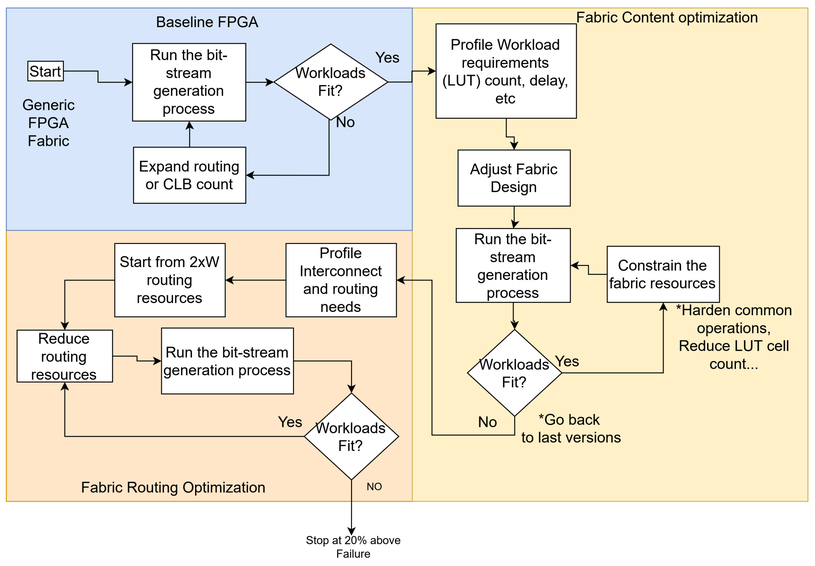

Figure 1: Iterative process to eFPGA core design.

The iterative process derives an eFPGA fabric sized for the target workloads. The fabric content stage identifies acceleration primitives for common operations. For example, DSP blocks in heterogeneous FPGAs implement arithmetic logic in hard logic instead of soft LUT-based logic.

The final stage of the process is optimizing the routing fabric of the FPGA. In our design, we aim to route inputs/outputs and primitives to user logic effectively. The core goal is achieving 100% utilization of all the memory ports and arithmetic primitives in the eFPGA by the user logic while maintaining routability. The end result is a core that can be integrated within a larger many-core system.

DESIGN

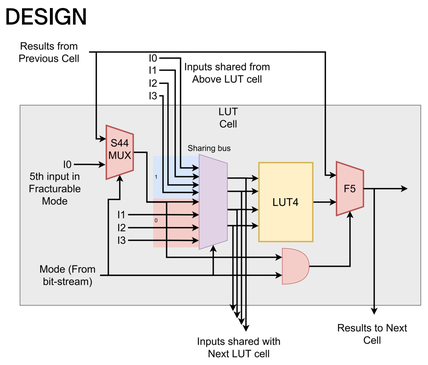

Figure 2-a: LUT-4 cell with shared operands and forwarded result.

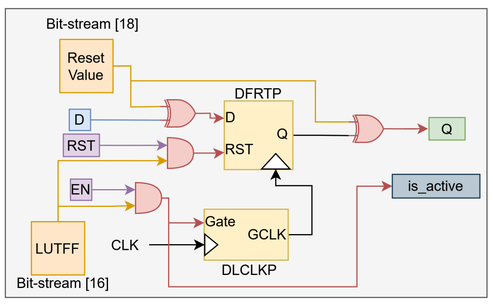

Figure 2-b: Flipflop design with static enable from bitstream, and runtime enable pin.

LUT & Flip-flop (FF) cells:

The basic logic element in our FPGA is shown in Figure 2-a. The figure shows a basic 4-input lookup table, with shared input operands and shared output. The shared operands allow combining the LUT-4 cells to form up to LUT-7 cells. The results are forwarded as a carry-chain mechanism to allow for faster combinatorial logic. Figure 2-b shows the FF used in our design. The FF uses a clock gating signal from both the bitstream and the eFPGA fabric. The is_acitve signal is used to signal if the FF is enabled to other tiles.

Fabric Clock gating:

In order to enable the user to use effective clock gating in the eFPGA, we implemented the design in Figure 3. In the figure, the clock signal propagates north between configurable logic blocks. These blocks require buffering the clock signal to fanout to all the FFs in the design. The design propagates the is_active signals from Figure 2-b south towards the clock source, so the clock signal does not consume energy when the FFs are not in use. This allows the user to implement event-activated logic on the FPGA without using a constant clock signal to save energy.

Co-located Compute and Register file (CRF):

In Figure 4, we show the design of a combined register file and an arithmetic unit. This is to avoid using scattered fabric FFs for storing states. It also increases the density of the in-fabric state memory. In terms of computational performance, the co-located registers provide the arithmetic unit with a static combinaitorial path, regardless of user logic. Figure 4 also shows an external interface that is wired outside the eFPGA fabric to provide a faster path to pre-fill the internal registers from external RAM. The bitstream parameters enable the arithmetic unit inside the CRF to support different fixed-point formats and signed/unsigned modes. The op-code and the internal read path allow the user to perform operations on internal data without fetching, saving energy and cycles.

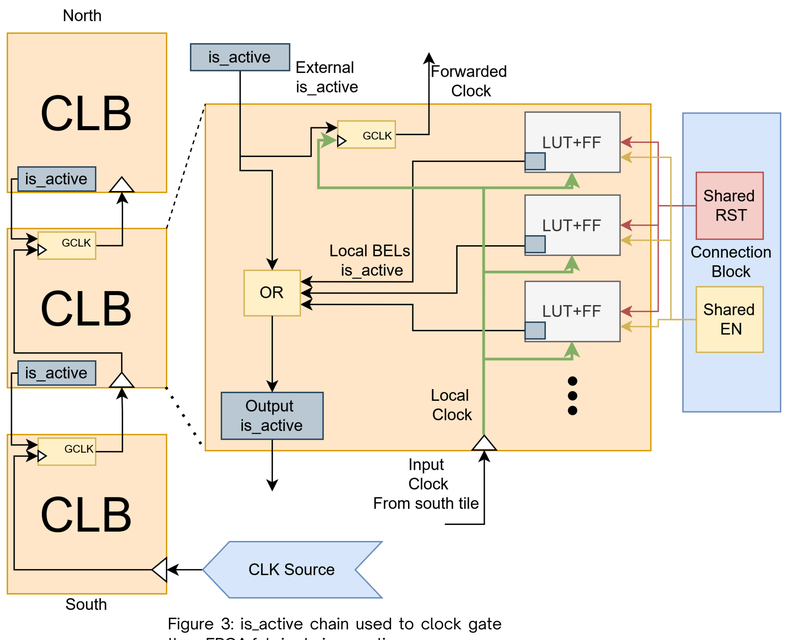

Figure 3: is_active chain used to clock gate the eFPGA fabric during run-time.

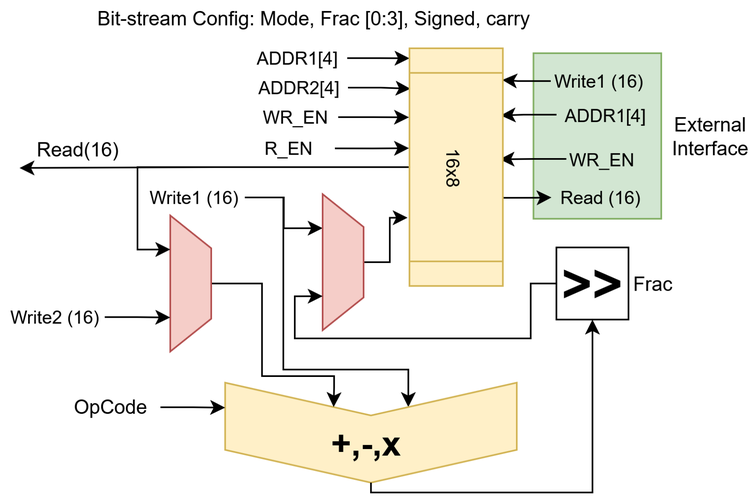

Figure 4: CRF cell design containing a co-located memory and compute cell.

Fabric Design:

Figure 5 contains the fabric block diagram showing 64 CLBs with 8 LUT-4 cells per block. The fabric also contains 4 CRF blocks with 4 arithmetic elements and 32 words, 16-bit each of state memory. The CRF elements in the design are clocked by a separate clock line from the edge of the fabric at 200MHz, while the rest of the fabric is clocked at 50MHz (synthesised on SKY130). The south edge of the fabric contains 4 loop controller (LC) elements that are used to pre-fill data from SRAM to the CRF at 4x the fabric frequency. The router interface is included since the design is meant to integrate in a many-core accelerator. The south IO blocks allow user designs to send packets and pre-load multiple words to the CRF through the LC. The edge of the eFPGA is terminated with tiles that reflect the routing wires back to the fabric. The shared operands from Figure 2a flow from east to west, while the is_active chain from Figure 3 flows north to south. This design allows the user to implement neuromorphic workloads on the FPGA using the CRF blocks as neuron engines, along with the event-activated clock network.

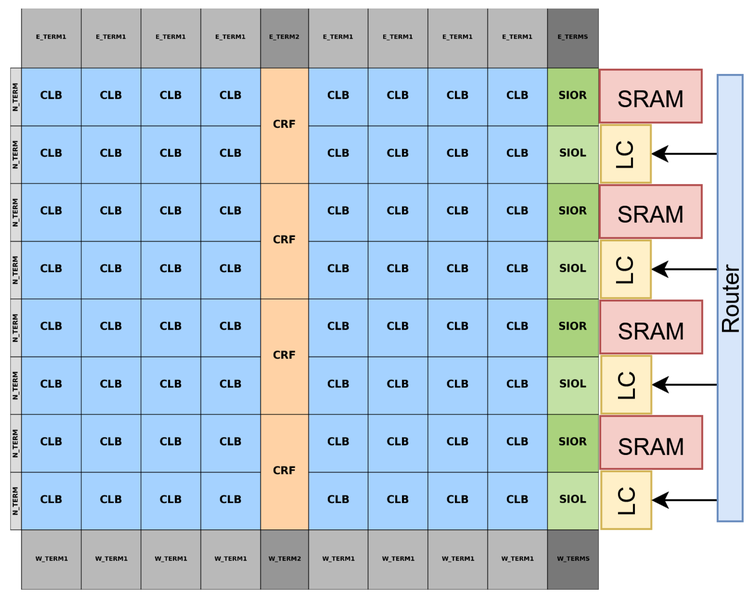

Figure 5: Complete Fabric block diagram with integration blocks.

RESULTS & DISCUSSION

Metric	NCLG (No Gating)	CLG (Gated)	Difference
Physical Stats			
Total Area (μm^2)	22 335.17	22 579.16	+1.09%
Total Cells	2 063	2 088	+25
Total Wires	2 355	2 381	+26
Key Cell Breakdown			
Clock Gates (dlclkp)	0	9	+9
Clock Buffers (clkbuf)	1	0	−1
Flip-Flops	8 (dfxtp)	8 (dfrtp)	0
Mux 2:1 (mux2)	54	38	−16
Mux 4:1 (mux4)	340	340	0
Logic Gates (AND/OR/etc.)	18	43	+25

Table 1: Area comparison between Gated and baseline CLB.

Idle Fabric, Clock gate, vs No gate:

As seen in Table 1, the baseline design consumes more area due to the higher mux count to keep the FFs value stable. In Table 2, the idle gated fabric consumes 95.8% less energy than the baseline, proving the event-driven architecture is more efficient both in area and in power.

Group	No Gating Total Power (W)	With Gating Total Power (W)	Reduction (%)
Sequential	1.95×10^{-3}	3.43×10^{-6}	99.8%
Clock	1.22×10^{-3}	33.8×10^{-6}	97.2%
Combinational	0.100×10^{-3}	0.100×10^{-3}	0.0%
Total System	$\mathbf{3.28 \times 10^{-3}}$	$\mathbf{0.137 \times 10^{-3}}$	**95.8%**

Table 2: Idle Fabric with single FF toggle comparison. Clock gated vs No-gating.

Fracturable LUT-4 cells:

In Figure 6, our design is compared with the baseline LUT-4 cells. The baseline implements a 7-input function using the FPGA routing fabric and two LUT-4 cells. Our design uses the shared results bus to implement the cascaded cell from [5]. The final point to compare is the fracturable LUT-5 cell using the shared operand bus and the F5 mux. The result shows that the S44 cell has moderate timing and power, while the LUT-5 cell is faster and consumes less power.

The fracturable cell, however, does not scale to higher order functions. Figure 7 shows the same test done to compare S44 vs LUT-5 cells arranged to form a LUT-7 cell. Our design provides the user the ability to synthesize their logic using different strategies (area vs time) based on their requirements.

FPGA area, routing vs arithmetic:

Table 3 compares the area of CLBs and CRF blocks in terms of configuration memory, routing, arithmetic logic, and area percentage of FPGA. The CRFs have a lot of density compared to CLBs, with 37% of the area as usable logic. Almost half of the CLB is routing. The configuration memory is mostly used by the routing fabric to configure the routing multiplexers. Overall, the CLB tiles are 91% routing, and they consume 67% of the FPGA area. The results highlight one of the fundamental limits of FPGAs.

	CLB	CRF tile
Routing	17,368 µm² 43.5 %	22,106 µm² 27.4 %
Config memory	12,012 µm² 30.1 %	16,760 µm² 20.7 %
Top level routing	7,446 µm² 18.6 %	11,914 µm² 14.7 %
Arithmetic	3,113 µm² 7.8 %	30,026 µm² 37.2 %
Total cell area	39,939 µm²	80,806 µm²
% of total FPGA fabric	66.8 % (64 × CLB)	5.3 % (4 × CRF)

Table 3: Area comparison for Logic vs Routing

SKY130 event Convolution ASIC vs FPGA

	Baseline†	CLKG+FRAC†	CRF FPGA	ASIC 100 MHz	ASIC 200 MHz
Frequency	25 MHz	50 MHz	50 MHz	100 MHz	200 MHz
Clock gating	No	Yes	Yes	Yes	Yes
Kernel	Fixed	Fixed	Runtime	Runtime	Runtime
Cycles/MAC	—	—	6	4	4
Total power (mW)	3.77	5.82	2.56	1.45	2.58
Throughput (MMAC/s)	—	—	8.33	25.0	50.0
Energy/MAC (pJ)	—	—	307	58	52
Efficiency (GMAC/s/W)	—	—	3.25	17.2	19.4
Relative to CRF FPGA:					
Power	—	2.3× ↑	1×	1.8× ↓	≈1×
Throughput	—	—	1×	3× ↑	6× ↑
Energy/MAC	—	—	1×	5.3× ↓	5.9× ↓

† Fixed kernel—power comparison only.

Table 4: ASIC vs eFPGA Event convolution

The results in Table 4 show a summary of the performance of the FPGA when running the event-based convolution. The ASIC has 6 times the throughput of the eFPGA, and 4 times the maximum frequency. At 50 MHz with a fixed kernel on soft logic, the eFPGA consumes 2.26 times the total power, but takes four times longer to finish the task. This comes with a massive area penalty as seen in Table 3. The overall penalty of 90% in terms of area, and x6 in terms of performance is expected, since eFPGA fabrics are dominated by routing.

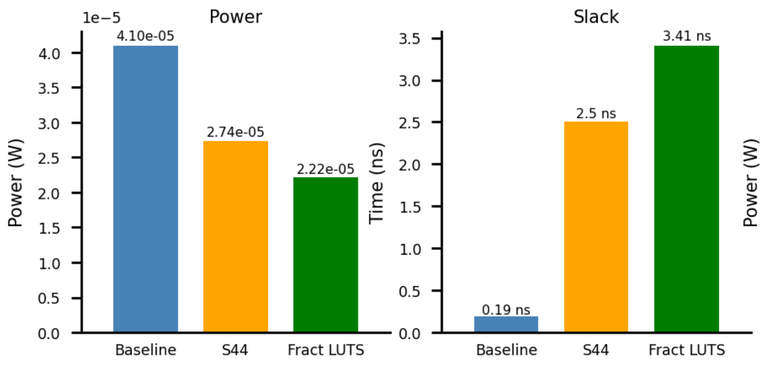

Figure 6: LUT-4 fabric routed vs two cascaded LUT-4 vs Fracturable LUT5. Power and delay, 5-input function.

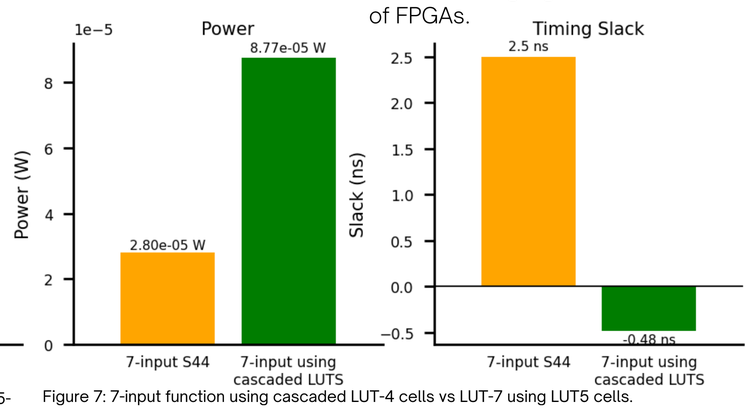

Figure 7: 7-input function using cascaded LUT-4 cells vs LUT-7 using LUT5 cells.

CONCLUSION

In this research, we explored the design of an eFPGA-based neuromorphic core. The design used neuromorphic computing principles such as event-based activations, co-located state and compute, and used multiple SRAM blocks to achieve higher parallelism. The logic cell design and the routing fabric allowed the core to utilize flexible data flows and customizable input pre-processing. This, however, comes at a cost of 90% in terms of area, and a penalty of 6 in terms of throughput.

ACKNOWLEDGEMENTS

The research for this poster was done as part of the SPARK-RV project at the University of Twente. Special thanks to the organizers of ICNCE 2026 and the University of Twente for sponsoring this submission.

REFERENCES