International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-9 Issue-1S3, December 2019

An Energy Efficient Register File Architecture for VLIW Streaming Processors on FPGAs

Pranav S. Vaidya, Avinash Yadav, Linknath Surya, John J. Lee

Abstract: ***The design of a register file with large scalability, high bandwidth, and energy efficiency is the major issue in the execution of streaming Very Long Instruction Word (VLIW) processors on Field Programmable Gate Arrays (FPGA's). This problem arises due to the fact that accessing multi-ported register files that can use optimized on-chip memory resources as well as enabling the maximum sharing of register operands are difficult provided that FPGA's on-chip memory resources only support up to two ports. To handle this issue, an Inverted Distributed Register File (IDRF) architecture is proposed in this article. This new IDRF is compared with the existing Central Register File (CRF) and the Distributed Register File (DRF) architectures on parameters such as kernel performance, circuit area, access delay, dynamic power, and energy. Experimental results show that IDRF matches the kernel performance with the CRF architecture but 10.4% improvement in kernel performance as compared to DRF architecture. Similar experimental results related to the circuit area, dynamic power, and energy are discussed in this article.***

Keywords: Inverted distributed register file architecture, VLIW streaming multiprocessor, FPGA, multi-ported memory.

I. INTRODUCTION

The high-end real-time streaming applications need 10-100 Giga Operations Per Second (GOPS) [1],[2] by taking advantage of both Instruction and Data Level Parallelism (ILP and DLP). The ILP and DLP present in these applications are often utilized by the implementation of soft SIMD-VLIW processor on FPGAs [3]. To increase ILP, *P* functional units can be composed in a Very Long Instruction Word (VLIW) pipeline to execute P instructions in parallel. Furthermore, to facilitate DLP, Single Instruction Multiple Data (SIMD) functional units can be used such that each functional unit processes *M* words in parallel.

Design of local register files that scale well along the SIMD axis (i.e., with the increase in *M*) and the VLIW axis (i.e., with the increase in *P*) is the main scalable challenge in the implementation of this SIMD-VLIW processor on FPGAs. The previous approaches of Centralized Register File (CRF) and Distributed Register File (DRF) tend to scale well along the SIMD axis by utilizing techniques such as banking [4] but scale poorly along the VLIW axis. Nonetheless, as multi-ported memories required for CRF cannot be mapped efficiently to optimized FPGA memory resources such as BlockRAMs, CRFs scale poorly in area and access delay. Moreover, as DRFs require propagation and staging of multiple copies of operands, they scale poorly in instruction count, dynamic power, and energy.

Due to these reasons, generic soft processors [5] implemented on FPGAs are constrained to a few functional units. To outclass these limitations, this article delineates the architecture of the Inverted Distributed Register File (IDRF) that combines the advantages of the CRF and DRF and alleviates their respective limitations.

Manuscript published on 30 December 2019.
* Correspondence Author (s)
Pranav S. Vaidya*, Department of Electrical & Computer Engineering, Indiana University Purdue University, Indianapolis, Indiana, USA. Email: paranav.vaidya@gmail.com
Avinash Yadav, Department of Electrical & Computer Engineering, Indiana University Purdue University, Indianapolis, Indiana, USA. Email: aviyadav@iu.edu
Linknnath Surya, Department of Electrical & Computer Engineering, Indiana University Purdue University, Indianapolis, Indiana, USA. Email: lbalasu@iu.edu
John J. Lee, Department of Electrical & Computer Engineering, Indiana University Purdue University, Indianapolis, Indiana, USA. Email: johnlee@iu.edu

II. PRIOR WORK

The traditional use of Centralized Register File (CRF) is to implement multi-ported register files in processors. Fig. 1 shows a CRF with *n* registers and *P* functional units in which the input side of each register is directly connected to a *P*-to-one multiplexer of the write port interconnect that is responsible for the selection of one functional unit to the register. For reading these *n* registers, an *n*-to-one multiplexer is used at the input of each functional unit in the read port interconnect.

The CRF allows maximal sharing of the operands between functional units such that any functional unit can write (or read) to (or from) any register in the CRF. Moreover, as the read and write port interconnects are implicit (i.e., compiler-skeptic), the algorithm used for scheduling and register allocation for CRF-based VLIW processors are less complex.

Nevertheless, the implementation of CRF on FPGAs do not scale well with a number of functional units. As BlockRAMs only support two ports, FPGA CAD tools synthesize the CRF to slower, reconfigurable slice resources. This increases area significantly and limits the maximum operating frequency of CRF.

The implementation of a CRF on FPGA is possible using BlockRAMs by time-multiplexing the two BlockRAM ports [6]. In a time-multiplexed CRF, the registers inside the CRF are clocked at multiples of external pipeline frequency such that multiple read and writes can occur in a single clock of the pipeline.

However, this approach is also not scalable with a number of functional units as the maximum external clock frequency generated by FPGA's Programmable Clock Managers (PCMs) is about 375-400MHz [3]. Furthermore, a time-multiplexed CRF requires temporary registers in addition to the implicit read (and write) port interconnects to hold addresses (or data) for pending (or completed) accesses [6].

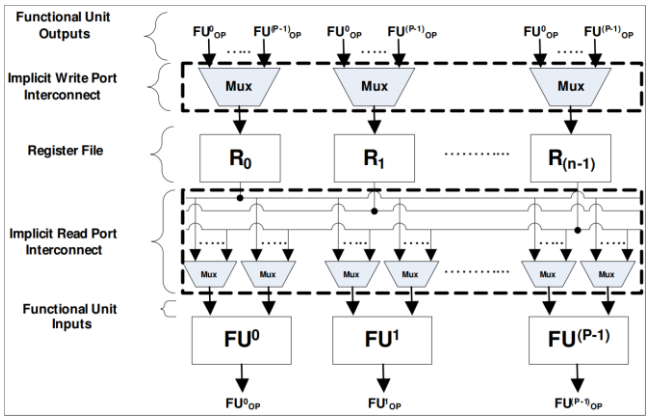


Fig. 1.CRF with n registers and *P* functional units.

On the contrary, the Distributed Register File (DRF) [7] architecture removes implicit interconnects and uses an explicit interconnect (i.e., managed by the compiler), thereby overcoming the problem of scalability. Moreover, with DRF, the complexity of multiplexing and decoding the read (or write) port interconnects is reduced by using a VLIW compiler that can generate static routing information as part of the instruction scheduling. The architecture of DRF is shown in Fig. 2. The outputs of the *P* functional units are controlling the explicit write port interconnects through the signals. Due to the access of each functional unit input to its own dedicated register, DRF does not require any read interconnects, unlike CRF.

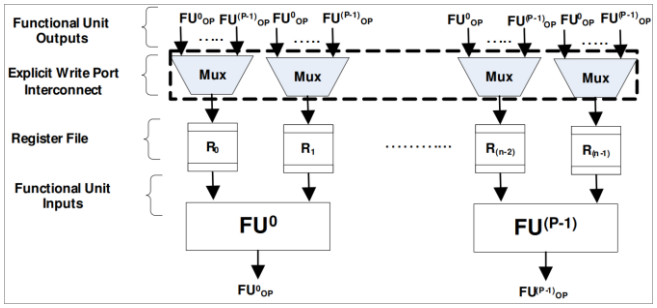


Fig. 2.DRF with 2*P* registers and *P* functional units.

However, DRF has the problem of multiple operend copies. The operand sharing between functional units results in the staging of multiple copies in DRF. For example, let us consider the code segment in Fig. 3 and its equivalent kernel schedule in Fig. 4 for a simple VLIW pipeline composed of the two ALUs (namely A0 and A1) and one Load unit (Load). The instructions are executed by each functional unit along with the write port interconnect (shown in Fig. 4 in a simplified manner as three wires with connections in the lower portion of each cycle (C_k) to emphasize the operand flow (shown with dark lines)). As instructions 2 and 3, as shown in Fig. 4, are operating on the same operand *a* and hence multiple copies of *a* are stored in register files of A0 and A1 in the DRF. Furthermore, in the presence of a structural hazard on the single write port of each destination register file, shared copies have to be propagated further by staging them through multiple register files across multiple cycles [7]. Hence, DRF accrues increased signal activity, register pressure, and kernel length due to copy operations and structural hazards. Consequently, streaming applications may experience the increased cost of dynamic energy with the DRF.

```
(1)   a = load ( ... )
(2)   b = x + a
(3)   c = y + a
```

Fig. 3.Example of code segment.

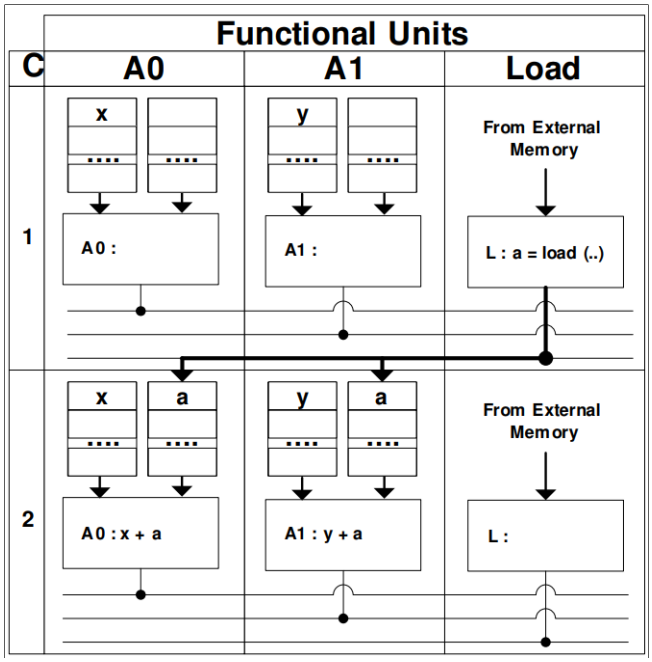


Fig. 4.DRF schedule and operand flow for Fig. 3.

III. PROPOSED APPROCH

Aforementioned limitations of CRF and DRF can be overcome by using an Inverted Distributed Register File (IDRF) proposed here as shown in Fig. 5. The IDRF reverse-modifies the idea introduced in the scheme of DRF by removal of explicit write-port interconnects and addition of explicit read-port interconnects that connect *P* dedicated register files implemented as BlockRAMs to *P* functional units. The control over 2P-to-one MUX of the read port interconnect at the input of each functional unit and each register file is done through "Set Read Instructions (SRI)" encoded in the VLIW instruction bundle by the compiler during code-scheduling.

Furthermore, each SRI consists of Read-Set micro-op and Mux-Select micro-op that are represented as RS_i^k and MS_i^k, respectively. Read-Set micro-op triggers a read from the register file of the k^{th} functional unit during cycle i whereas Mux-Select micro-op selects the operands from the Read-Set and routes them to the inputs of the designated functional unit. For instance, the IDRF instruction schedule corresponding to the code segment of Fig. 3 is shown in Fig. 6. The related set read instruction for cycle 2 causes required operands to be emitted from the respective register files at ordinal locations in *RS*. The *MS* instruction then selects and routes operands from *RS* to respective functional unit inputs. In this way, IDRF not only avoids multiple copies of the same operands but also uses the on-chip memory resources (BlockRAMs) more efficiently.

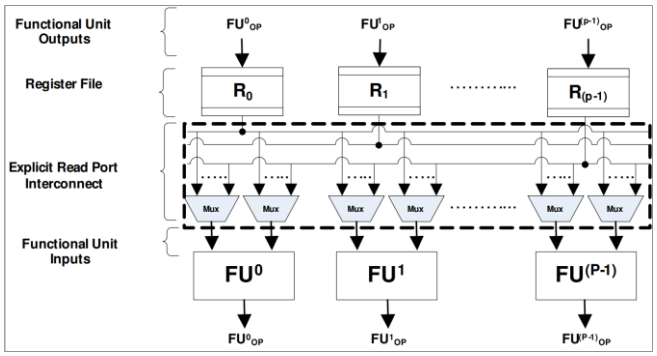


Fig. 5. IDRF with *P* register file and functional units.

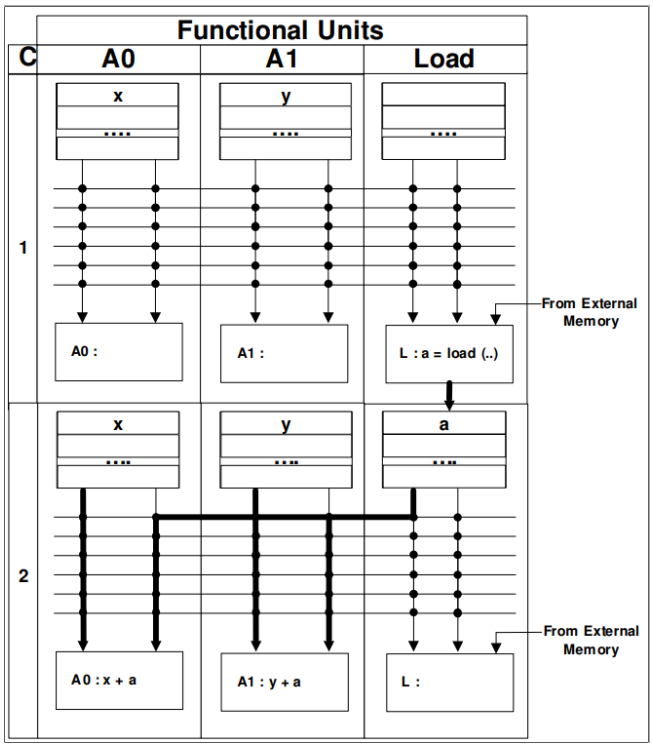
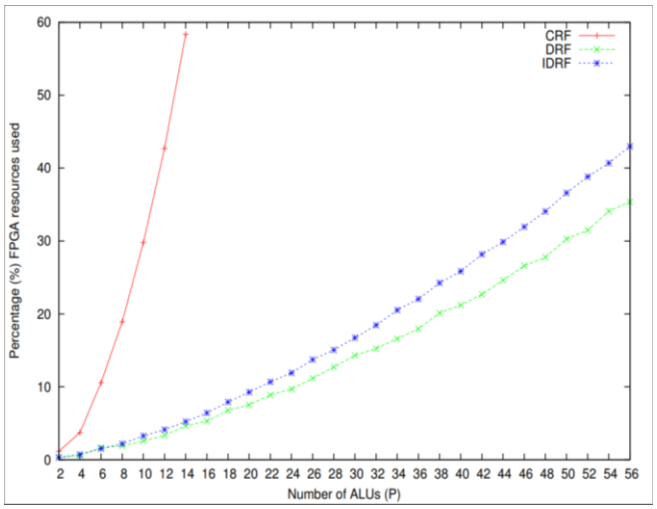


Fig. 6.IDRF schedule and operand flow for Fig. 3.

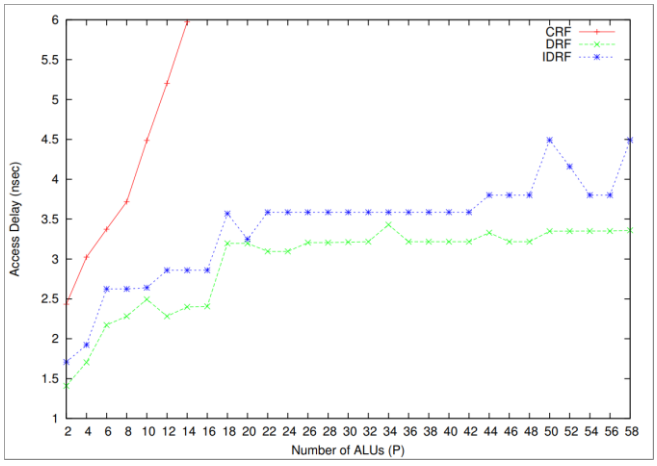
criteria namely, percentage FPGA resources consumed (or area), access delay, kernel performance (in terms of instruction count), dynamic power, and energy.

A. Area and Access Delay

Evaluation Techniques: The Verilog RTL code generators were created to automate in building various configurations of CRF, DRF, and IDRF as a function of the number of functional units (*P*). To ease the comparative analysis of area and access delay, the total number of register allocations in CRF, DRF, and IDRF was kept equal for fair comparison. These configurations were then synthesized for a Xilinx XC5VLX330 Virtex-5 FPGA [3] with 207360 slices and 288 9-Kbit BlockRAMs using Xilinx ISE 13.4 CAD tools with speed optimization goal at normal optimization effort. Fig. 7 shows the percentage area and access delay (in nano-sec) for CRF, DRF, and IDRF as a function of *P*. We addressed area as the geometric mean of the percentage slice resources and percentage BlockRAMs for our analysis.



(a) Area (% FPGA resources used)



(b) Access delay plot (in nanoseconds)

Fig. 7.Area and access delay plots.

IV. EVALUATION OF IDRF

In this section, we first describe evaluation techniques that were used to evaluate the proposed IDRF. Next, we compare the performance of IDRF with CRF and DRF using five

DRF vs CRF/IDRF: Fig. 7 shows that the area utilized by CRF grows more rapidly with *P* than that of either DRF or IDRF. In fact, it was not possible to fit a CRF for *P* >12 in the considerably large Xilinx LX330 FPGA. This is because CRF is synthesized entirely using slice resources whereas register files in both DRF and IDRF synthesize to faster, optimized BlockRAMs. As a result, DRF and IDRF respectively show on average 87.9% and 86.5% area improvement over CRF. Furthermore, Fig. 7(b) shows that as compared to CRF, the improved area utilization leads to an average of 46.6% and 37.0% improvement in access delay in DRF and IDRF, respectively.

DRF vs IDRF: Fig. 7 shows that although both DRF and IDRF are significantly better in area and access delay than those of CRF, nonetheless, on average IDRF occupies 18.52% more area and has 15.33% more access delay than DRF. This is because the larger 2*P*-to-one muxes in the read port interconnect of IDRF occupy more slice resources than the smaller *P*-to-one muxes in the write port interconnect of the DRF.

B. Kernel Performance, Power

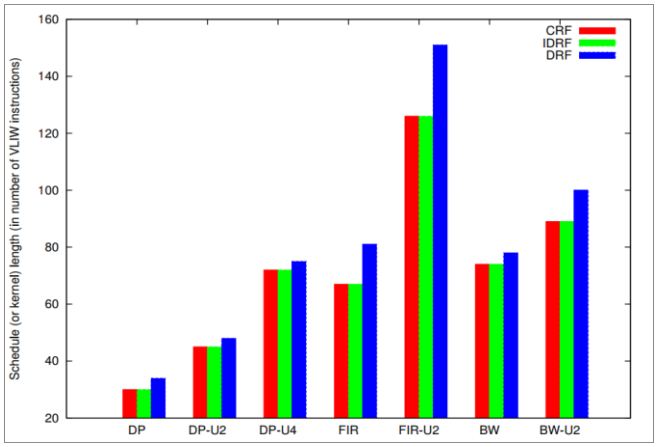
Evaluation Techniques: The VLIW compiler [8] is exploited to generate the instruction schedules and read (or write) activities for a SIMD-VLIW processor with 9 functional units targeting CRF, DRF, and IDRF in order to evaluate the kernel performance, power, and energy of the representative applications as described in Table I. The kernel performance is expressed in terms of clocks per instruction on average, and frequency was kept similar for all evaluated applications and register files. Xilinx Xpower analyzer was used to estimate the energy generated by the CRF, DRF, and IDRF register files clocked at 100MHz.

Table- I: Representative applications analyzed

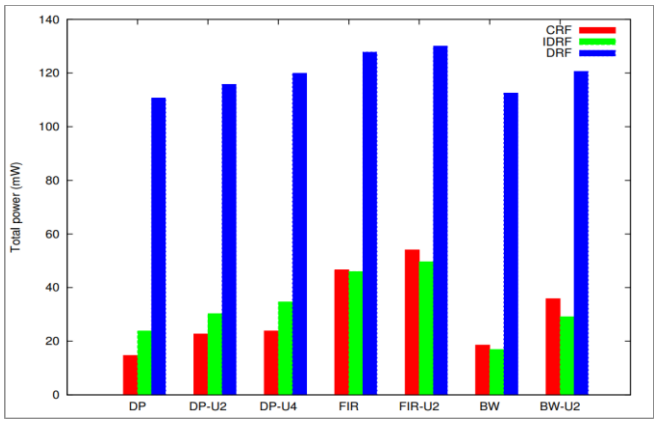
Application	Description
DP	A dot-product of two streams of matrices A (size=8 × 4) and B (size=4 × 8) containing 32-bit floating point numbers.
DP-U2	Dot-product with inner loop unrolled twice.
DP-U4	Dot-product with inner loop unrolled four times.
FIR	A 64-tap 32-bit floating point Finite Impulse Response filter similar to [8].
FIR-U2	A 64-tap 32-bit floating point FIR unrolled twice.
BW	A block warp algorithm used for point-sample rendering [7].
BW-U2	A block warp algorithm unrolled twice.

DRF vs CRF/IDRF*:* Unlike DRF, both CRF and IDRF permit arbitrary sharing of operands without making copy and do not encounter structural hazards due to the disagreement on the shared write ports. Therefore, the CRF and IDRF kernel counts are equal in size and 10.4% smaller than that of DRF (Fig. 8a). The decreased signal activity in CRF and IDRF produce 76.5% and 75.0% reduction in dynamic power and 79.0% and 76.7% reduction in dynamic energy, respectively (Fig. 8b and 8c).

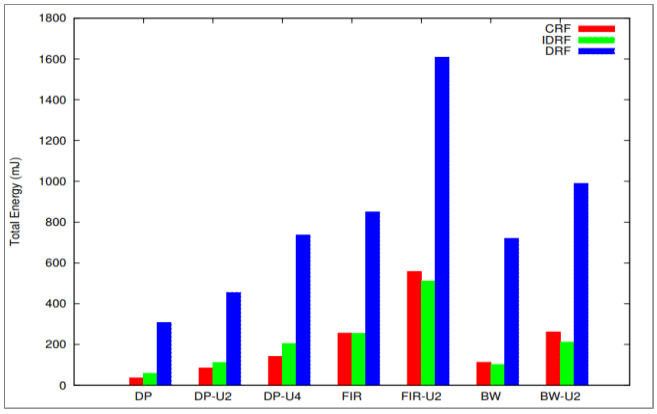
DRF vs IDRF*:* Even though CRF and IDRF perform gradually better with the increase of the number of ALUs than DRF in kernel performance, in terms of dynamic power and energy consumption, on average IDRF consumes 10.9% more dynamic energy and power than CRF. The increment in energy consumption in IDRF is due to the operations performed on BlockRAMs which are otherwise unused and hence clock-gated off in the FPGA implementation of CRF.



(a) Kernel performance *(instruction count)*



(b) Dynamic power *(in mW)*



(c) Dynamic energy (*in mJ)*

Fig. 8.Kernel performance, dynamic power and dynamic energy for the applications described in Table 1.

REFERENCES

1. J. H. Ahn, W. J. Dally, B. Khailany, U. J. Kapasi, and A. Das, Evaluating the Imagine stream architecture, Proc. of ISCA (2004) 14-25.
2. P. Vaidya, J. Lee, F. Bowen, Y. Du, C. Nadungodage, and Y. Xia, Symbiote: A Reconfigurable Logic Assisted Data Stream Management System (RLADSMS), Proc. of SIGMOD, (2010) 1147-1150.
3. Virtex-5 Multi-Platform FPGA, http://www.xilinx.com/products/siliconsolutions/fpgas/virtex/virtex5/index.htm.
4. Virtex-5 Multi-Platform FPGA, http://www.xilinx.com/products/siliconsolutions/fpgas/virtex/virtex5/index.htm.
5. C. LaForest and S. Gregory, Efficient multi-ported memories for FPGAs, in Proc. of IEEE FPGA (2010) 41-50.
6. J. Zalamea, J. Llosa, E. Ayguad, and M. Valero, Software and hardware techniques to optimize register file utilization in VLIW architectures, International Journal on Parallel Programming (2004) 447-474.
7. J. Owens et al., Media processing applications on the Imagine stream processor, Proc. of IEEE ICCD (2002) 295-302.
8. F. Gerneth, FIR Filter Algorithm Implementation Using Intel SSE Instructions, Intel Whitepaper, http://download.intel.com/design/intarch/papers/323411.pdf.

AUTHORS PROFILE

Pranav S. Vaidya graduated in 2015 with a PhD in Electrical and Computer Engineering from Purdue University. During his PhD research, he was actively involved in creating custom coprocessors using FPGAs for Data Stream and Database Management systems. His research interests also include simulation and validation of high-performance hybrid computing systems. He is currently employed with NVIDIA Corp, where he is involved in the development of next-generation, high-performance, CPU and GPU hardware architectures and software required thereof to leverage these high-performance computing systems.

Avinash Yadav is a graduate student at Indiana University Purdue University, Indianapolis. He received his undergrad degree in Electronics and Communication Engineering from India. His area of interest is the Digital VLSI Circuit Design.

Linknath Surya, Graduate Student, Department of Electrical and Computer Engineering, Indiana University Purdue University, Indianapolis.

John J. Lee, Associate Professor, Department of ECE, IUPUI, USA. He received his Ph.D. degree in Electrical and Computer Engineering from the Georgia Institute of Technology. http://www.ece.iupui.edu/~johnlee/