# A Field Programmable Transistor Array Featuring Single-Cycle Partial/Full Dynamic Reconfiguration

Jingxiang Tian, Gaurav Rajavendra Reddy, Jiajia Wang, William Swartz Jr., Yiorgos Makris and Carl Sechen
Department of Electrical Engineering, The University of Texas at Dallas, Richardson, TX 75080
Email: {jxt122130, gxr141930, jxw143630, wps100020, gxm112130, cms057000}@utdallas.edu

***Abstract*—We introduce a CMOS computational fabric consisting of carefully arranged regular rows and columns of transistors which can be individually configured and appropriately interconnected in order to implement a target digital circuit. Termed Field Programmable Transistor Array (FPTA), this novel reconfigurable architecture enables several highly-desirable features including (i) simultaneous storage of three configurations along with the ability to dynamically switch between them in a fraction of a single cycle, while retaining the fabric's computational state, (ii) rapid or full modification of a stored configuration in a time proportional to the number of modified configuration bits through the use of hierarchically arranged, high throughput, asynchronously pipelined memory buffers, and (iii) support for libraries containing cells of the same height and variable width, just as in a typical standard cell circuit, thereby simplifying transition from a prototype to a custom IC design. Besides presenting the design details of this fabric in a 130nm technology and demonstrating the aforementioned capabilities, we also briefly discuss the development of a complete CAD flow for programing this fabric and we use numerous benchmark circuits to contrast its area efficiency against a typical FPGA implemented in the same technology node.**

## I. Introduction

We present a novel field programmable device, developed on conventional static CMOS processes, which has significant differences and potential advantages over field programmable gate arrays (FPGAs). Specifically, our design seeks to:

**1) Improve area utilization**: Unlike the basic configurable logic block (CLB) of FPGAs, which employs look-up tables (LUTs) to generate combinational logic functions [1], our FPTA relies on a carefully-arranged, configurable array of transistors, which can be interconnected to implement standard library cells. Thereby, for logic outside the custom cells (e.g., full adders, D flip-flops, multiplexers) that both FPGAs and our FPTA explicitly possess, we surmise that transistor utilization in our FPTA is better than in FPGA LUTs. In other words, an FPGA may allocate an entire LUT to implement even a relatively simple gate, while our FPTA allocates only the precise number of columns of transistors required.

**2) Enable time-sharing between multiple circuits:** Our design supports simultaneous storage of three separate configurations, each with its own computational state. Therefore, we also provide the means to switch, in a fraction of a single cycle, between any of the three stored configurations, or load a new configuration while toggling between the other two.

**3) Support rapid circuit updates:** Instead of being serially loaded, our FPTA configuration is stored in hierarchically arranged, high throughput, asynchronously pipelined memory buffers. This enables not only faster configuration but also rapid dynamic partial reconfiguration wherein only a portion of a circuit is reloaded by addressing specific transistor columns.

In the following, we discuss the FPTA architecture and design features which support each of the above three objectives (Sections II-IV). To demonstrate the aforementioned capabilities, we designed and laid out an FPTA prototype in the IBM 130nm 1.2V process and we developed a CAD tool flow (Section V) to synthesize, place and route designs onto it. Results using various benchmark circuits (Section VI) confirm that, despite the added features of single-cycle switching between multiple designs and rapid partial reconfiguration, our FPTA achieves better area utilization in comparison to a typical FPGA in the same technology node.

## II. Transistor-Level Programming

To present our FPTA's ability to support transistor-level programming, we first describe its overall architecture, including the basic logic cell structure and the routing resources.

### A. FPTA Architecture Overview

The FPTA architecture, which resembles a standard cell circuit, consists of numerous long rows of transistors and can work with cell libraries similar to those used for a typical standard cell-based ASIC, where each cell has the same height and variable width. In the FPTA, the granularity of the width of the cells is one column of transistors. As shown in Fig. 1(a), a column consists of two pMOS transistors above two nMOS transistors. This basic column is replicated repeatedly in the horizontal direction, forming a row. Vertical transistors in Fig. 1(a) can be programmed to be always on, always off, or to receive logic signals. Among the horizontal transistors, while the innermost (blue-colored) ones are strictly used for isolation, the outermost ones not only support isolation but also enable the use of logic functions that require up to three transistor in series. The metal1 (M1) layer is used to interconnect the transistors and various logic gates.

The actual hardware implementation of the row-based FPTA groups every four columns of transistors into so-called 'logic cells'. In Fig. 1(a), a potential logic gate output is illustrated by means of a small green rectangle. Each potential output is optionally connected to a vertical metal2 (M2) track, by means of a programmed switch. In addition, the two pMOS and the two nMOS inputs in a column are directly connected to individual vertical M2 tracks. Each of these tracks is driven by either a programming bit or a logic signal.

In addition to the four columns of transistors, each logic cell comes with a custom D-flip-flop (DFF), a full adder (FA) and a multiplexer (MUX), whose inputs and outputs (I/Os) are optionally connected to logic cell I/Os. The D input (*Signal_in*) of the DFF, which is shown in Fig. 1(b), is either connected to a logic transistor in the first column of the logic

978-3-9815370-8-6/17/$31.00 

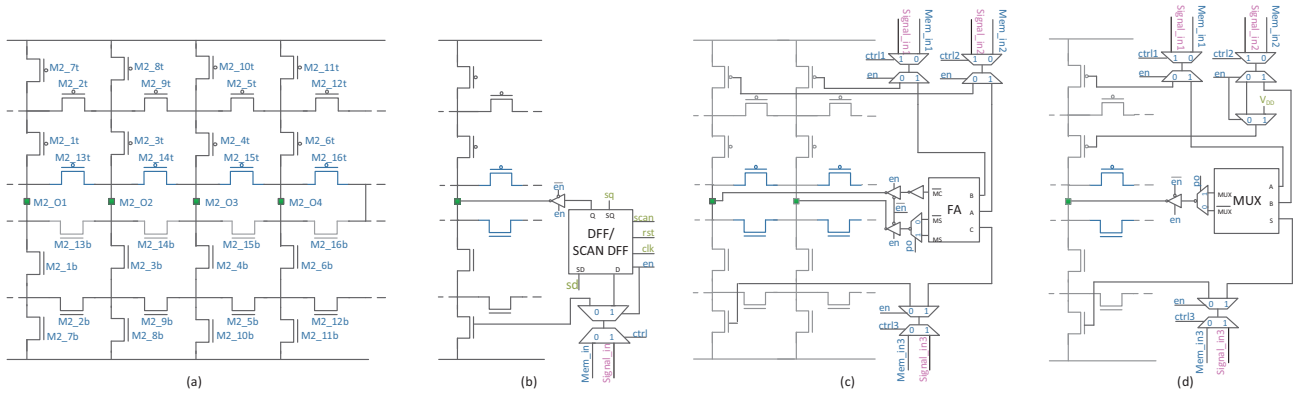


Fig. 1: (a) Logic cell structure, (b) Built-in D flip-flop, (c) Built-in full adder, and (d) Built-in multiplexer

cell (if *ctrl* = 1 and *enable* = 0) or to the D input of the DFF if *enable* = 1 via the de-multiplexer. All DFFs provided by the logic cells are connected in a scan chain. The three inputs of the FA, which is shown in Fig. 1(c), span across the 2nd and 3rd columns of the logic cell. The carry and sum outputs (either inverted or non-inverted) are provided at the outputs of the 2nd and 3rd columns, respectively. The custom MUX, which is shown in Fig. 1(d), only occupies one column. The MUX output is provided in either inverting or non-inverting form at the output of the 4th column.

Figs. 2(a) and (b) depict how two different cells, *i.e.,* a NAND3 and an AOI22 gate, are programmed in a logic cell. All the transistors, except for the ones highlighted in black, are turned off. Among the highlighted transistors, the ones with signal names at their gate terminals receive primary inputs; the rest are turned on to complete the circuit. Transistors highlighted in blue are turned on with programming bits to form the output node of the logic function.

Note that a pull-down (or pull-up) network of three transistors in series is the maximum possible. The motivation for limiting to three transistors in series was based on area efficiency versus power efficiency concerns. In [2], it was shown that a standard cell library limited to two transistors in series for each of the pull-up and pull-down networks was sufficient to generate circuits with the best power efficiency. However, for FPGAs, the vast majority of the delay and power are due to the interconnection networks. Thus, we decided to reduce the number of nets by allowing more complex cells with up to three transistors in series.

### B. Routing Architecture

Each logic cell includes two routing switchboxes, one just above the logic cell and one just below. The two switchboxes are exactly symmetric, so only the upper switchbox, as shown in Fig. 2(c), is described in detail. It is more efficient to supply the programming bits for the top part of the logic cell (pMOS transistors) from above, and for the bottom part of the logic cell (nMOS transistors) from below. Certainly the upper and lower pair of switchboxes belonging to vertically neighboring logic cells can be viewed as a single switchbox.

Metal layer 2 (M2) and metal layer 4 (M4) are the vertical routing resources, while metal layer 3 (M3) and metal layer 5 (M5) are the horizontal routing resources. Metal layer 6 (M6) is primarily used to route the programming bits. In Fig. 2(c), each metal line is labeled with the letter 'M' followed by the layer number, and then an underscore followed by the line or track number. In Fig. 1(a), the transistors in the logic cell are labeled as to how they connect with M2 lines in Fig. 2(c). Each small square (of various colors) shown in Fig. 2(c) is a switch, implemented by an nMOS transistor controlled by a programming bit, with the source and drain connecting the two perpendicular metal lines (on different layers) as shown in Fig. 2(d) (with the programming bit called *ctrl* in this case). Since the pass transistors do not pass a full *Vdd*, a half-keeper is added to each metal segment, as shown in Fig. 2(d).

There are 12 vertical M4 lines that go over the logic cell unit along with switches connecting to the 17 horizontal M3 lines (switches in gray) and to the 9 M5 lines (switches in red). Each of the 9 M3 lines and each of the 9 M5 lines has 4 switches to M4. Each of the remaining 8 M3 lines has 3 switches to M4. For the switchbox above (below) the logic cell, the 16 M2 lines terminate inside the logic cell, either at a pMOS (nMOS) gate input or at an output. In fact, 4 M2 lines connect to the outputs of the logic cell and 12 M2 lines connect to pMOS (nMOS) inputs. Each of the 12 M2 lines, which connect to pMOS or nMOS inputs, has 7 connection choices to M3 lines via switches. Each of the 4 M2 lines, which connect to outputs, has 8 connection choices to M3, 4 in the upper switchbox and 4 in the lower one. M3 tracks 11, 13, 15, and 17 facilitate local connections from an output of a logic cell to inputs of nearby logic cells.

The various metal line segments terminate at the boundary of a logic cell. Vertical M2 metal line segments literally terminate at the top and bottom of the logic cell. However, vertical M4 line segments can be connected to M4 segments of other logic cells above and below, in either direction, using the optional bi-directional repeater, shown in Fig. 2(e).

The M3 and M5 horizontal metal line segments terminate at the left and right boundaries of the logic cell but connect to adjacent M3 and M5 line segments in the neighboring logic cells via nMOS pass transistors. In order to limit the delay on these lines, after each 4 logic cells, the nMOS pass transistor is replaced with a bi-directional repeater.

## III. Programming & Single-Cycle Reconfiguration

We now proceed to describe how the FPTA is programmed and how its local memory structure can support single-cycle switching between multiple configurations.

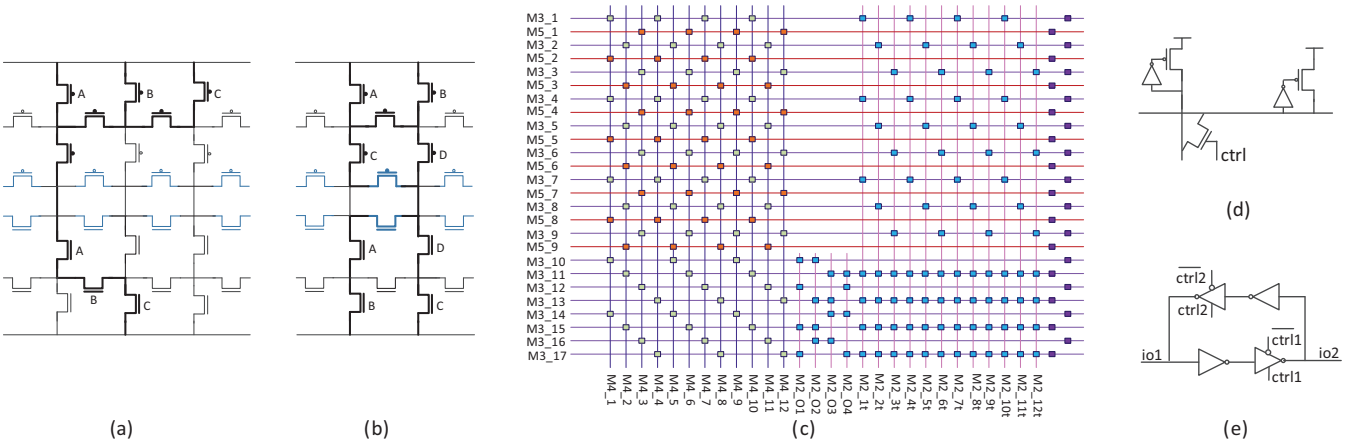


Fig. 2: (a) Configuration of a NAND3, (b) Configuration of an AOI22, (c) Upper switchbox, (d) Pass transistor with half-keepers, and (e) Bi-directional repeater

### A. Programming Path Structure

The dashed box in the right portion of Fig. 3 shows the structure of the functional blocks comprising one complete logic cell, which is called a unit. Besides the previously mentioned switchbox wiring, termed upper switchbox and lower switchbox in Fig. 3, the local memory which stores the programming bits used to configure a unit is also shown. Most of the programming bits are used to configure the switchboxes. To shorten the wiring, the local memory is separated into four parts, surrounding the two switch boxes. The left portion of Fig. 3 shows the detailed connections between different functional blocks. As the unit structure is approximately symmetric, we only expand the upper half.

Using the IBM 130nm 6-2-0 metal stack process (6 thin and 2 thick metal layers), the layout of a single unit is 430um by 72um. Eight units are combined to form the basic replicated block (termed a group) in the FPTA. Fig. 4 shows the block-level schematic of a group, whose size is 430um by 620um.

### B. Local Memory Structure

The local memory structure, which consists of three latches driven by transmission gate switches, is shown in Fig. 5(a). Three latches are used to allow three separate FPTA bit streams to be stored at the same time, enabling a single-cycle switch from one digital design (*e.g.,* state machine) to another.

When writing programming bits in from the 33 bit-lines (BLs) available per unit, one of the global control signals *clka*, *clkb*, or *clkc* selects the proper latch to receive the bit. Subsequently, one of the global copy signals *cpa*, *cpb*, or *cpc* is used to select the stored programming bits which actually configure the FPTA. For each memory cell, only one of *cpa*, *cpb*, or *cpc* is high at any given time, representing the latch that is supplying the current programming bit to the FPTA.

This design enables dynamic reconfiguration in a fraction of a clock cycle, by turning off the copy signal that is currently on (among *cpa*, *cpb*, or *cpc*) and globally turning a different one on. Furthermore, a completely programmed system can be running while the programming bits of a new system are being loaded. For example, while the system configured by *cpa* is running, *clkb* may be on and the programming bits comprising

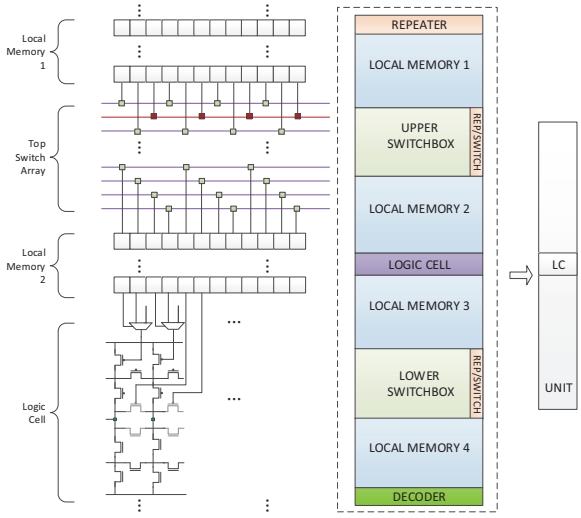


Fig. 3: Unit structure

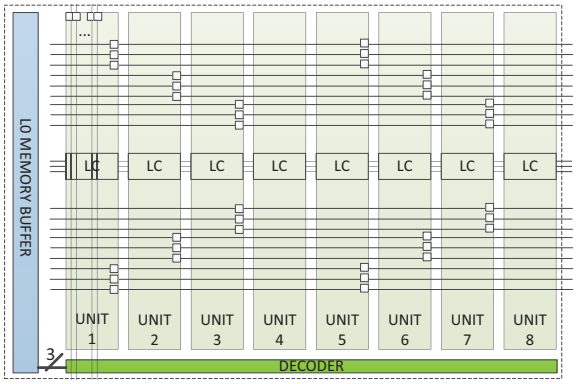


Fig. 4: Group structure with logic cells (LCs) for each unit

an alternative system may be loaded into the second latch of the memory cell. Since *cpb* is off, the current state of the FPTA is not impacted by the loading of an alternate system. After this alternate system is completely loaded using *clkb*, a third system can be subsequently loaded by turning off *clkb* and turning on *clkc*. While *clkc* is being used to load the third system configuration, in a fraction of a clock cycle, the second system can start (or resume) execution by turning on *cpb*.

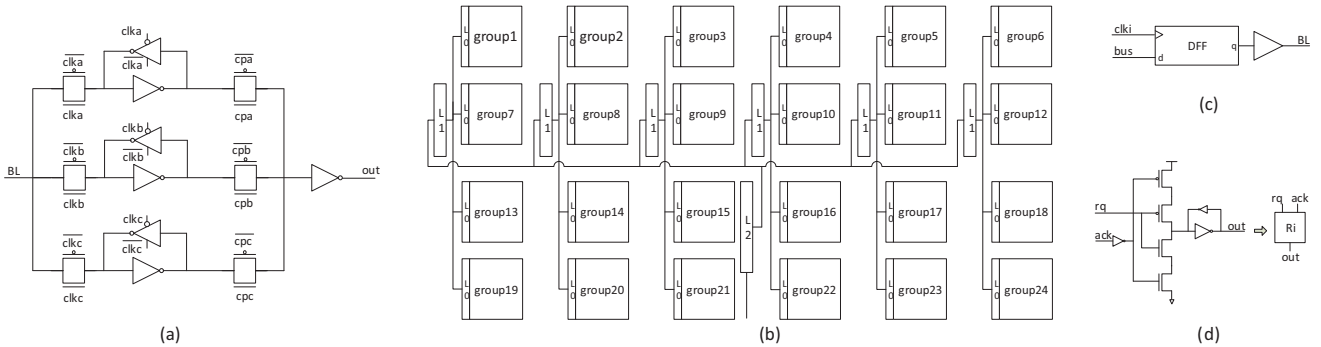


Fig. 5: (a) Local memory structure, (b) Asynchronous memory buffer pipeline, (c) Memory buffer, and (d) Muller C-element

In summary, it is possible to store the programming for three separate systems and switch between them in a fraction of a clock cycle, or toggle between two systems while loading a third one. In order to properly enable toggling between running systems, not unlike swapping jobs in a central processing unit of a computer, separate system or finite state machine flip-flops must be available for each copy signal. Along those lines, with respect to Fig. 1(b) which was simplified, there is not just one flip-flop whose output may optionally appear at the first column of each logic cell, but, in fact, three different flip-flops enabled by, respectively, *cpa*, *cpb*, or *cpc*.

## IV. Rapid Partial Reconfiguration

Lastly, we describe the architecture through which the FPTA configurations are stored and we explain how they can be selectively updated to support rapid partial reconfiguration.

### A. Memory Buffers L2, L1 and L0

The Level 0 Memory (L0) shown on the left side of Fig. 4 is actually a memory buffer used to supply programming bits to the local memory for each of the 8 units comprising the block. Each unit has 16 columns, each of which contains a payload of 33 programming bits. The total number of columns for each group is, therefore, 16 x 8, or 128. This means that 7 address bits must be appended to the 33-bit payload to direct it to the proper column. Three of the address bits select a unit and the remaining four select a column within that unit. Thus, when writing a 33-bit payload to a local memory, the L0 Memory buffer passes a word of 40 bits to the group.

We designed an asynchronous memory buffer pipeline to rapidly program the FPTA. Our prototype has six groups horizontally and four groups vertically, as shown in Fig. 5(b). When writing programming bits, L0 memory buffers are supplied by Level 1 memory buffers (L1). Each L1 memory buffer supplies a word to one of four L0 memory buffers, arranged vertically. The word size for the L1 memory is 42 bits: 40 bits needed by the selected L0 memory buffer plus 2 address bits to select that L0 memory buffer. The six L1 memory buffers are fed by a Level 2 memory buffer (L2), as shown in Fig. 5(b)). Its word size consists of 45 bits: 42 bits needed by an L1 memory buffer plus 3 bits to select the particular L1 memory buffer. An off-chip memory then supplies 45-bit words to the L2 memory buffer. In order to facilitate fast (*i.e.*, pipelined) read-out of the programming bits, the various memory buffers are, in fact, bi-directional; for brevity, this part is not presented here.

Fig. 5(c) shows the structure of the programming bit register. The *clk* triggers the DFF to pass the data from the bus to the BL. The DFFs in the L2, L1, and L0 memory buffers are controlled by distinct *clk* signals. These *clk* signals are derived from an asynchronous pipeline control unit, described next.

### B. Programming Bit Asynchronous Pipeline

We use an asynchronous pipeline to achieve a high programming rate. For example, after an L2 memory buffer receives a 33-bit payload from off-chip, it forwards it (along with the address) to the corresponding L1 memory buffer as soon as the latter is ready to receive it. When the L1 memory buffer receives the address and payload, the L2 memory buffer is freed to accept a new address and payload from off-chip. The rate at which programming bits can be sent from off-chip is extremely high. In fact, detailed circuit simulations show that the programming bit data rate is nominally 9.0 Gbps.

We used a bounded-delay asynchronous pipeline control scheme [3]. Each stage of the asynchronous pipeline consists of a Muller C-element, shown in Fig. 5(d). When a stage Ri receives a request (*req*) from the previous stage Ri-1, if the acknowledge (*ack*) from the next stage Ri+1 is available (active low), then a request is generated to Ri+1 along with an acknowledge to Ri-1.

The asynchronous control scheme for writing programming bits to the local memories is shown in Fig. 6. The signal *clki* (where *i* is the stage index of the memory buffer) is generated by the logical AND of Ri and the inversion of the (bounded) delayed Ri. This is basically a pulse generator that ensures that the various local clocks (*clki*) are non-overlapping in their high portions. The signal *clki* is used to trigger the DFFs in the Li memory buffer, as shown in Fig. 5(c). R2 receives the request to write programming bits (RQ_WR) from off-chip. Since the L2 memory buffer feeds any of six L1 memory buffers in the prototype, acknowledge signals from all six of them are OR-ed to form the acknowledge for R2. Also, since each L1 memory buffer feeds any of four L0 memory buffers, acknowledge signals from all four of them are OR-ed to form the acknowledge for R1. The local memory driven by the L1 memory buffer is controlled by local clock *clk00*. Note that the last stage uses the delayed request as its acknowledge.

The delay elements (Ds), shown in Fig. 6, are set based on careful worst-case simulation of the extracted layout of the

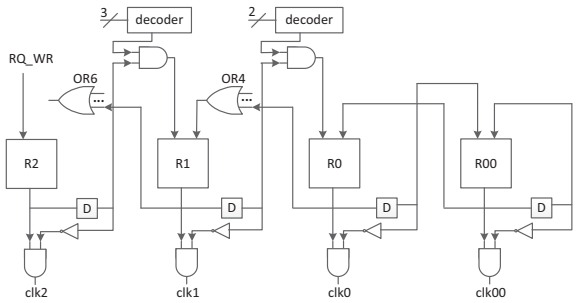


Fig. 6: Asynchronous write-in control scheme

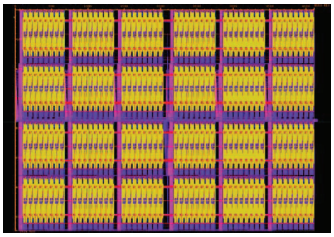

Fig. 7: Layout of prototype comprising a 6X4 array of groups, each group comprising 8 logic cells

prototype. Since this pipeline is used for programming bits and is not a signal datapath, maximum throughput is not required. Therefore, we conservatively double the worst-case simulated delay (including worst-case corner) to set the delay element value with sufficient margin to handle process, voltage, and temperature (PVT) variations.

## V. Cell Library and CAD Tool Flow

Our base library consists of all the possible inverting gates that are feasible with our FPTA and its series limit of three transistors. The 24 base library components are: INV, NAND2, NOR2, AOI12, AOI22, OAI12, OAI22, NAND3, NOR3, AOI31, OAI31, AOI41, OAI41, AOI32, OAI32, AOAI311, OAOI311, AOAI211, OAOI211, AOOAI212, OAAOI212, AOAAI2111, OAOOI2111 and MAJI (which is the inverted mirror carry). In addition, the following custom cells are built into the logic cells: FA, FAI, DFF, MUX and MUXI.

To further increase logic density, following the philosophy of [2], numerous compounds of the 24 base library cells are also provided. Compound cells are created by appending an inverter (or a NAND2 or NOR2) to one input and/or the output of each of the 24 base cells, resulting in a total of 234 compound cells. Compound cells are placed as a unit but are decomposed into their constituent base cells prior to routing.

We also developed the necessary CAD tool-flow for programming the FPTA. Our tool-flow consists of industry-standard commercial tools along with open-source software which has been modified to work with our architecture. Synopsys Design Compiler is used to synthesize a gate-level netlist from the Register-Transfer Level (RTL) description of the design. The cell library, consisting of 24 base cells, 11 built-in cells and 234 compounds was characterized using Synopsys SiliconSmart. Placement is done through TimberWolf [4], which is very effective in row-based placement. For routing, we modified the source code of the open-source tool VPR (Versatile Place and Route) [5], [6] to make it compatible with our architecture. Finally, bit-stream generation is done through a Python script which we developed for this purpose.

TABLE I: Area utilization compared to a commercial FPGA

| Benchmark | Cell Count (Synopsys DC) | FPTA Utilization | Altera Stratix Utilization |
|---|---|---|---|
| B04 | 317 | 1.02% | 2% |
| B05 | 353 | 1.29% | 2% |
| B12 | 539 | 2.02% | 4% |
| SPI | 1240 | 4.27% | 8% |
| B14 | 2123 | 8.69% | 10% |
| Tv80 | 3077 | 11.13% | 19% |
| B15 | 3461 | 12.18% | 22% |
| B20 | 4407 | 17.78% | 20% |
| B21 | 4635 | 19.07% | 20% |
| B22 | 6702 | 26.85% | 29% |
| B17 | 10942 | 37.75% | 68% |
| AES_cipher | 9422 | 38.91% | 47% |
| AES_inv_cipher | 13578 | 52.07% | 72% |
| WB_conmax | 16436 | 70.00% | 148% |
| B18* | 25303 | 87.85% | 140% |

(∗) One instance of B15 is removed from B18 to reduce the size of the benchmark in order to ensure that it fits within the available resources

## VI. Demonstrations

We designed and laid out a prototype for fabrication using the IBM 130nm 1.2V process. It includes a 6 x 4 array of groups, each containing 8 logic cells, for a total of 192 logic cells. The layout is shown in Fig. 7, and its overall size is 4113.41um x 2769.50um.

### A. Area Utilization Efficiency

In order determine the area utilization efficiency of our FPTA, we compared it with a commercial FPGA, Altera Stratix EP1S10, which uses the same 130nm technology and has a core size of 23mm X 23mm [7]. To make a fair comparison, we scaled up our FPTA to the same size, resulting in an array of 51 x 35 groups of 8 logic cells, for a total of 14,280 logic cells. We then implemented various benchmarks from ITC'99 [8] and opencores [9] on both our FPTA and the Altera chip. A comparison of the resource utilization is presented in Table I. Note that the same switchbox sizes were used throughout the scaled up array and the benchmarks were 100% routed.

Despite the additional area overhead due to having three memory cells per programming bit, the density (or utilization) of the FPTA is quite competitive with the Altera chip. We attribute this observation to the fact that for logic outside the custom cells (*e.g.,* full adders, carry units, flip-flops, multiplexers) that both designs possess, the transistor utilization of the logic cells in the FPTA is better than the transistor utilization of the LUTs in the Altera design. Essentially, even a relatively simple logic function might take up an entire LUT, whereas in the FPTA, only the precise number of columns needed to implement the gate are used. Thus, simple gates such as NAND2, NAND3, NOR2, NOR3, and up to three or even four input AOI and OAI gates are comparatively very area-efficient in the FPTA.

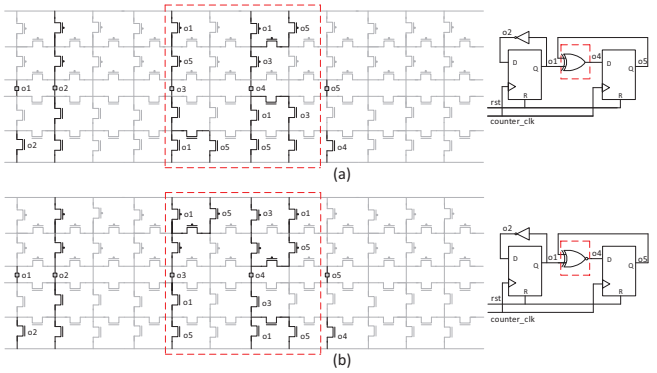


Fig. 8: (a) 2-bit up counter, and (b) 2-bit down counter

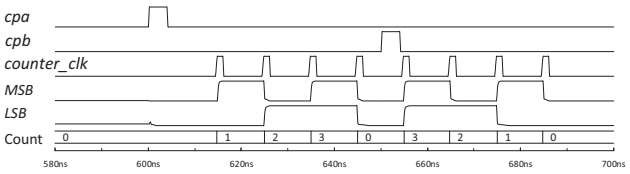


Fig. 9: Within-cycle reconfiguration demonstration

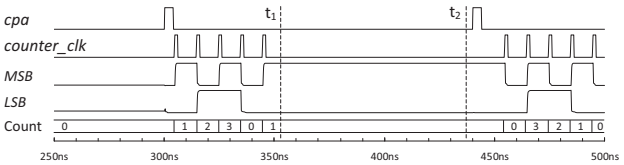


Fig. 10: Partial dynamic reconfiguration demonstration

### B. Single-Cycle Switching Between Configurations

The single-cycle reconfiguration capability of the FPTA is demonstrated using two 2-bit counters, shown in Fig. 8, one counting upwards and the other counting downwards, along with the waveforms illustrated in Fig. 9. Two separate bit-streams are generated for the 'up counter' and 'down counter' and are loaded into the A and B latches of the local memory of the FPTA, respectively. As shown in Fig. 9, the up counter is activated when the *cpa* pulse is provided and the counter starts counting 'up' from 0 to 3 based on the *counter_clk* pulses. As soon as *cpb* arrives, the bit-stream corresponding to the down counter is activated, within a single cycle, and the counter counts 'down' from 3 to 0. The waveforms confirm that the FPTA resources can be time-shared between two different bit-streams with a single-cycle toggle.

### C. Selective Partial Dynamic Reconfiguration

The partial/selective dynamic reconfiguration capability is demonstrated using an example of a 2-bit counter, which is initially configured as an up counter, as shown in Fig. 8(a). By selectively changing only the bits corresponding to the logic cell in the middle, its functionality is changed into a down counter, as shown in Fig. 8(b). This selective reconfiguration mode also allows the retention and transfer of computational state between the initial and the modified bit-stream, as illustrated in the waveforms of Fig. 10. Initially, the bit-stream of an up counter is loaded and the counter starts counting 'up' from 0, soon after receiving the *cpa* pulse. The counter is run through one full counting cycle and is stopped at the count '1' of its second counting cycle (time $t_1$). Between $t_1$ and $t_2$, the portion shown within the dashed red rectangle in Fig. 8 is reconfigured. This converts the up counter into a down counter. At time $t_2$, the down counter starts counting 'down' from the same state (count '1') where the up counter had stopped. Selective reconfiguration eliminates the need to reload the entire bit-stream for a small design change; hence, the time required for reconfiguring the FPTA is proportional to the number of bits changed in the bit-stream.

## VII. Conclusion

We developed a novel field programmable transistor array featuring (i) simultaneous storage of three configurations, along with the ability to dynamically switch between them in a fraction of a single cycle, while retaining the fabric's computational state, (ii) rapid partial or full modification of a stored configuration, in a time proportional to the number of modified configuration bits, through the use of hierarchically arranged, high throughput, asynchronously pipelined memory buffers, and (iii) support for ASIC-compatible libraries containing cells of the same height and variable width, thereby simplifying transition from a prototype to a custom IC. As demonstrated through fully routed implementation of benchmark circuits, this FPTA has equivalent or superior area efficiency over a typical FPGA, despite the triple number of programming bits and the hierarchical programming logic required for supporting multiple configurations and selective partial/full dynamic reconfiguration. We believe that this FPTA opens the possibility of new computational modalities, such as dynamically evolving circuits and field programmable device virtualization, which we are investigating in our ongoing research.

## References

[1] H. Yang, J. Zhang, J. Sun, and L. Yu, "Review of advanced FPGA architectures and technologies," *Journal of Electronics*, vol. 31, no. 5, pp. 371–393, 2014.

[2] M. Rahman, R. Afonso, H. Tennakoon, and C. Sechen, "Power reduction via separate synthesis and physical libraries," in *ACM/IEEE Design Automation Conference*, 2011, pp. 627–632.

[3] G. N. Hoyer, G. Yee, and C. Sechen, "Locally clocked pipelines and dynamic logic," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 10, no. 1, pp. 58–62, 2002.

[4] TIMBERWOLF, "Timberwolf systems inc.," http://www.twolf.com/.

[5] V. Betz and J. Rose, *VPR: a new packing, placement and routing tool for FPGA research*, pp. 213–222, Springer Berlin Heidelberg, 1997.

[6] V. Betz, J. Rose, and A. Marquardt, Eds., *Architecture and CAD for Deep-Submicron FPGAs*, Kluwer Academic Publishers, 1999.

[7] Altera Corporation, ," https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/stx/stratix_handbook.pdf.

[8] F. Corno, M. S. Reorda, and G. Squillero, "RT-level ITC'99 benchmarks and first ATPG results," *IEEE Design Test of Computers*, vol. 17, no. 3, pp. 44–53, Jul 2000.

[9] Opencores, ," http://opencores.org/.