IOP Publishing Superconductor Science and Technology
Supercond. Sci. Technol. 33 (2020) 054006 (16pp) https://doi.org/10.1088/1361-6668/ab7ec3

A semi-custom design methodology and environment for implementing superconductor adiabatic quantum-flux-parametron microprocessors

Christopher L Ayala[1], Ro Saito[2], Tomoyuki Tanaka[2], Olivia Chen[1], Naoki Takeuchi[1], Yuxing He[1] and Nobuyuki Yoshikawa[1,2]

[1] Institute of Advanced Sciences, Yokohama National University, 79-5 Tokiwadai, Hodogaya, Yokohama 240-8501, Japan
[2] Department of Electrical and Computer Engineering, Yokohama National University, 79-5 Tokiwadai, Hodogaya, Yokohama 240-8501, Japan

E-mail: ayala-christopher-pz@ynu.ac.jp

Received 28 November 2019, revised 27 February 2020
Accepted for publication 11 March 2020
Published 30 March 2020

Abstract

We present a comprehensive overview of a design methodology and environment that we developed to enable the implementation of microprocessors and other complex logic circuits using the adiabatic quantum-flux-parametron (AQFP) superconductor logic family. The design environment is catered for both the AIST 10 kA cm^{-2} Nb high-speed standard process as well as the AIST 2.5 kA cm^{-2} Nb standard process (STP2). We detail each aspect of the design flow, highlighting improvements in cell design, and new developments in circuit retiming to reduce the number of synchronizing buffers in the circuit datapath. With retiming, we expect a 14–37% reduction in the overall Josephson junction (JJ) count for some benchmarks. Finally, we show the successful experimental demonstration of an arithmetic logic unit and data shifter for an AQFP microprocessor using the established methodology and environment. The demonstrated circuits show full functionality and wide excitation current margins of nearly ±30%, which corresponds well with simulation results.

Keywords: AQFP logic, microprocessor, semi-custom design, superconductor electronics

(Some figures may appear in colour only in the online journal)

1. Introduction

Superconductor electronics have the potential to provide a high-performance, energy-efficient computing platform to power the present era of 'internet of things', big data, and social media [1–3]. Several energy-efficient superconductor logic families exist to provide such a platform, including energy-efficient RSFQ logic [4], energy-efficient SFQ logic [5], reciprocal quantum logic (RQL) [6], LR-biased RSFQ logic [7], and low voltage RSFQ (LV-RSFQ) logic [8].

Adiabatic quantum-flux-parametron (AQFP) logic is also an extremely energy-efficient superconductor logic family. AQFP logic can operate with bit energies of around $24k_BT$ using a four-phase 5 GHz clock in 10 kA cm^{-2} unshunted Nb/AlO$_x$/Nb Josephson junction (JJ) technology that is available today [9]. When compared to other energy-efficient superconductor logic families [10], adiabatic logic families such as AQFP logic have a major advantage in terms of switching energy. But as with any new technology, one must consider additional overhead when it comes to building systems beyond simple logic gates. A design study on AQFP logic was conducted in [3]. It synthesized a set of benchmark circuits and performed a data-dependent energy analysis while taking into account the additional overhead of buffering data to ensure phase-to-phase propagation and synchronization of signals. The benchmark circuit of a 32 bit adder with a

magnitude comparator and parity check (named C7552) had an energy efficiency of 40 aJ/op in AQFP logic. A simpler RQL-based 32 bit sparse-tree parallel-prefix adder that underwent logic and architecture optimization had an energy efficiency of 97 aJ/op. The RQL adder also lacked the magnitude comparator and parity check components that the AQFP C7552 benchmark had, and yet the AQFP circuit still used less than half the amount of energy per operation of the RQL variant.

But in order to physically realize the circuits described in [2, 3], let alone the systems in [1], a proper design environment including methodologies and electronic design automation tools are necessary. Additionally, it is much more than just simply using tools and methods already available for semiconductor or superconductor technologies. There is a need to re-evaluate current design methodologies and perhaps even develop new ones to enable the implementation of complex circuits such as a microprocessor.

In this paper, we report a methodology and design environment that adopts a semi-custom design approach where logic circuits are built-up using a standard library of AQFP logic cells. The logic cells themselves are designed in a modular way so that they are easy to optimize for wide margins and ultra-low switching energy. Starting with a Cadence Virtuoso environment, we augment it with open-source software as well as scripts and tools developed within our group. The environment is capable of supporting completely manual semi-custom design as well as semi-automated logic synthesis with new post-synthesis and retiming steps to optimize the circuit. The flow is backed by simulation and verification at the analog and digital levels. The design methodology and environment has been successfully used to design the execution units of an AQFP microprocessor. The experimental results of the designed circuits show that they are fully operating with wide excitation margins, indicating the capability of our tools and approaches to deliver working circuits. While the fabrication process used in this paper is limited to a four-layer 10 kA cm^{-2} and 2.5 kA cm^{-2} Nb/AlO$_x$/Nb process resulting in relatively large logic cell designs and sparse circuit densities, the methodologies describe here serve as a baseline to build upon when using more advanced processes such as the eight-layer MIT LL SFQ5ee process [11]. For example, a more compact AQFP cell library can be made using the SFQ5ee process as shown in [12], but the general approach towards building circuits and systems would still remain the same.

The rest of the paper is organized as follows: section 2 describes how the AQFP logic cells are built and how they are used to create logic circuits; section 3 provides a comprehensive overview of the design environment and design flow that we have established for AQFP logic where we review prior works and report new developments; section 4 describes the demonstration and measurement of prototype circuits designed using the established flow; and section 5 concludes the paper.

2. Semi-custom design methodology

Semi-custom design refers to the reuse of pre-designed sub-circuits to create a larger circuit in a hierarchical fashion, as opposed to full-custom design where all elements of the final circuit are made from scratch at the device-level to achieve maximum performance and minimum area at the cost of expensive labor and time [13, 14]. One of the most common ways to achieve a semi-custom design flow is to establish a standard cell library, which is a collection of sub-circuits that perform Boolean logic functions. Standard cells tend to have a fixed height and are designed in such a way so that cells can abut together in a single row to gradually form the power network of the larger circuit. The cells are also built such that the expected functionality of a standalone cell does not change when connected to other cells, or such that its interactive behavior is easy to predict and model.

One of the most successful efforts in developing a semi-custom design methodology for superconductive electronics is the CONNECT cell library for SFQ logic [15]. Since then, the CONNECT cell library now accommodates the AIST 10 kA cm^{-2} Nb nine-layer advanced process (ADP2) [16, 17]. Drawing upon the success of the CONNECT cell library, we developed a similar cell library for AQFP logic using the AIST 2.5 kA cm^{-2} Nb four-layer standard process (STP2) [18] and the AIST 10 kA cm^{-2} Nb four-layer high-speed standard process (HSTP) [19]. With the established AQFP cell libraries, we are able to arrange logic cells into logical rows that form the power-clock network. A complete logic circuit is composed of multiple rows of logic connected together.

2.1. Cell library

The cell design for AQFP logic is centered on a common schematic topology. Figure 1(a) shows the core schematic that is shared among the three active sub-cells used to create Boolean logic cells, namely the buffer (BFR), inverter (INV), and constant (CONST) with some slight differences between each sub-cell. The function of each sub-cell is as follows:

- *BFR:* produces the data input as the output as-is
- *INV:* produces the inverted data input as the output
- *CONST:* always produces a logical '0' or '1' as the output

The optimal parameters for this topology as listed in [19] were found by identifying which pair of the normalized loop inductance ($\beta_L = 2\pi L_1 I_c/\Phi_0$) and the normalized output inductance ($\beta_q = 2\pi L_q I_c/\Phi_0$) gives the best trade-off in bias margins and bit energy using practical device dimensions [20]. By focusing on optimizing this topology, all the sub-cells obtained a reasonable baseline parameter set after which further optimization can easily be done on an individual basis due to the small number of sub-cells. Simulation and refinement are possible through the use of analog simulators such as JSIM [21], and 3D inductance extractors such as InductEx [22].

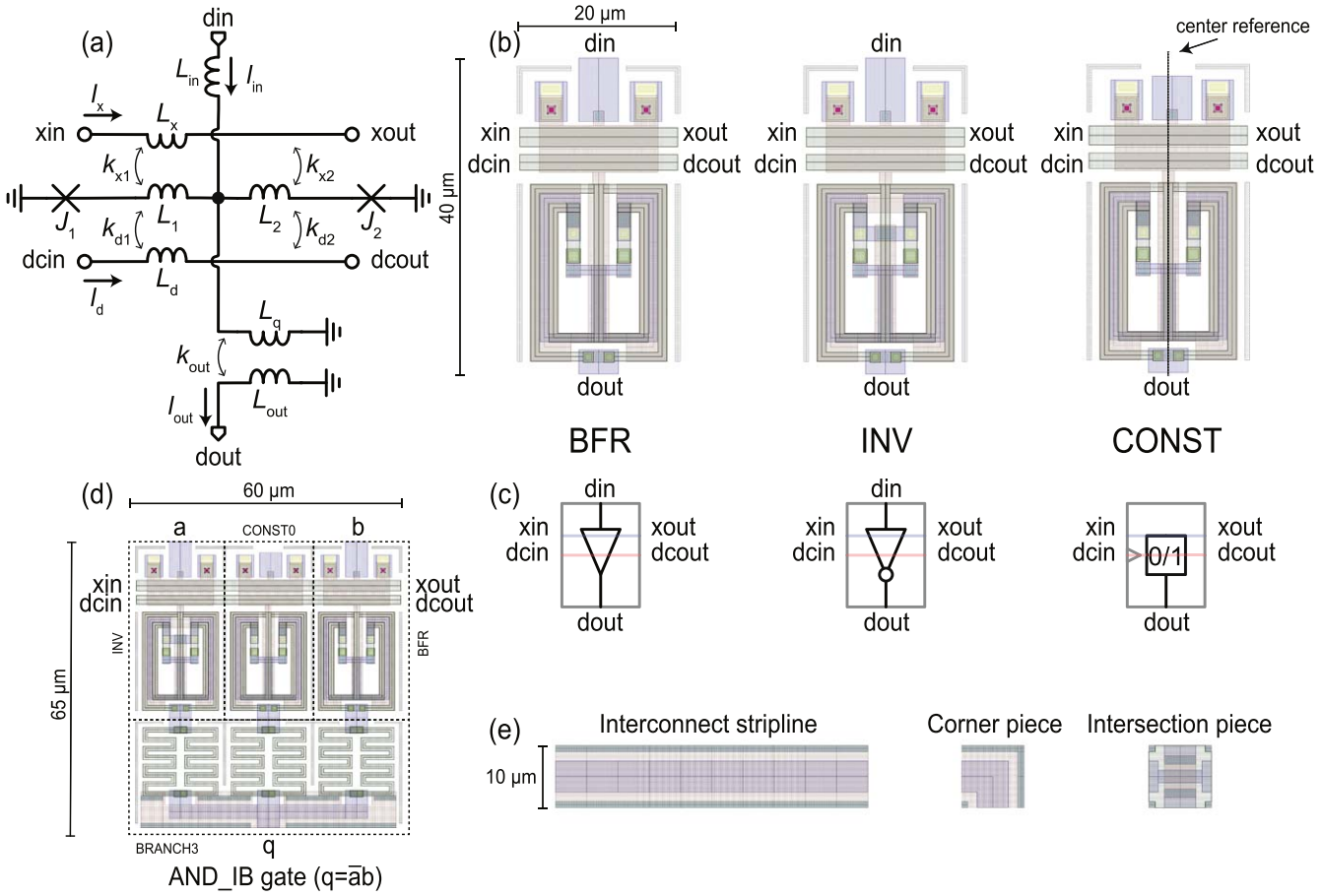


Figure 1. Key components of the cell library. (a) Schematic of the common topology of the AQFP sub-cells. (b) Layout of the sub-cells, namely the buffer (BFR), inverter (INV), and constant (CONST). (c) Symbol view of the sub-cells. (d) Layout of a logic cell composed of an INV, CONST0, and BFR, which produces the function $q = \bar{a}b$. (e) Interconnect stripline pieces to connect the logic cells together.

Figure 1(b) shows the latest layout of the AQFP library using HSTP. The width of the sub-cells reduced from 25 to 20 μm when compared to the original HSTP sub-cells in [19] by re-orienting the JJs. The height of 40 μm remained the same. The general shape of the layout in terms of excitation lines (xin/xout, dcin/dcout) and I/O pins (din/dout) are at standard locations to enable the modular design of Boolean logic cells. If we assume the BFR cell is the standard layout, one can notice there are some minor differences when compared to INV and CONST. In the case of INV, its output transformer has a reversed spiral to create an inverted mutual coupling factor (k_{out}) so that it can produce an inverted output signal. For CONST, the dc-SQUID portion of the cell has no input and as indicated by the center reference in figure 1(b), the CONST layout was intentionally designed with some asymmetry so that it will always switch to a constant logic state when it is clocked. The CONST layout as shown will always produce a logic '0' when the dc-offset flows in the positive direction (left-to-right) across the sub-cell (CONST0). If we mirror the layout along the y-axis and apply a dc-offset in the same direction, the CONST sub-cell will always produce a logic '1' (CONST1).

Figure 1(c) shows the symbolic view of the different sub-cells. Similar to the CONNECT cell library, the symbols are scaled to the physical layout so that the designer can simultaneously compose the schematic/netlist as well as the physical place and route by hand.

The sub-cells are then abutted together in different combinations along with a BRANCH3 sub-cell to create Boolean logic cells. The BRANCH3 sub-cell is a passive sub-circuit that merges (sums) the outputs of any combination of BFR, INV and CONST. The BRANCH3 can also be reversed to create a signal splitter (fan-out) [18]. Figure 1(d) shows how an INV, CONST0, and BFR form together to create an AND_IB (AND with INV and BFR) cell which performs the logic function of $q = \bar{a}b$. The Boolean AND function is created by having a CONST0 as one of the sub-cells connected to BRANCH3, and by the same principle, the Boolean OR function is created by having a CONST1 as one of the sub-cells connected to BRANCH3. The other two sub-cells that would connect to BRANCH3 can be any combination of BFR or INV. For example, a NAND/NOR function can be made with two INV sub-cells, and one CONST1/CONST0 respectively.

Like the CONNECT cell library, the interconnect for AQFP logic is composed of tiles that the designer stitches together to form the cell-to-cell interconnect. Figure 1(e) shows the different interconnect pieces available to the designer. The interconnect are striplines with ground shields above and below the signal layer. The standard interconnect pitch is 10 μm and the interconnect tiles can be abutted together.

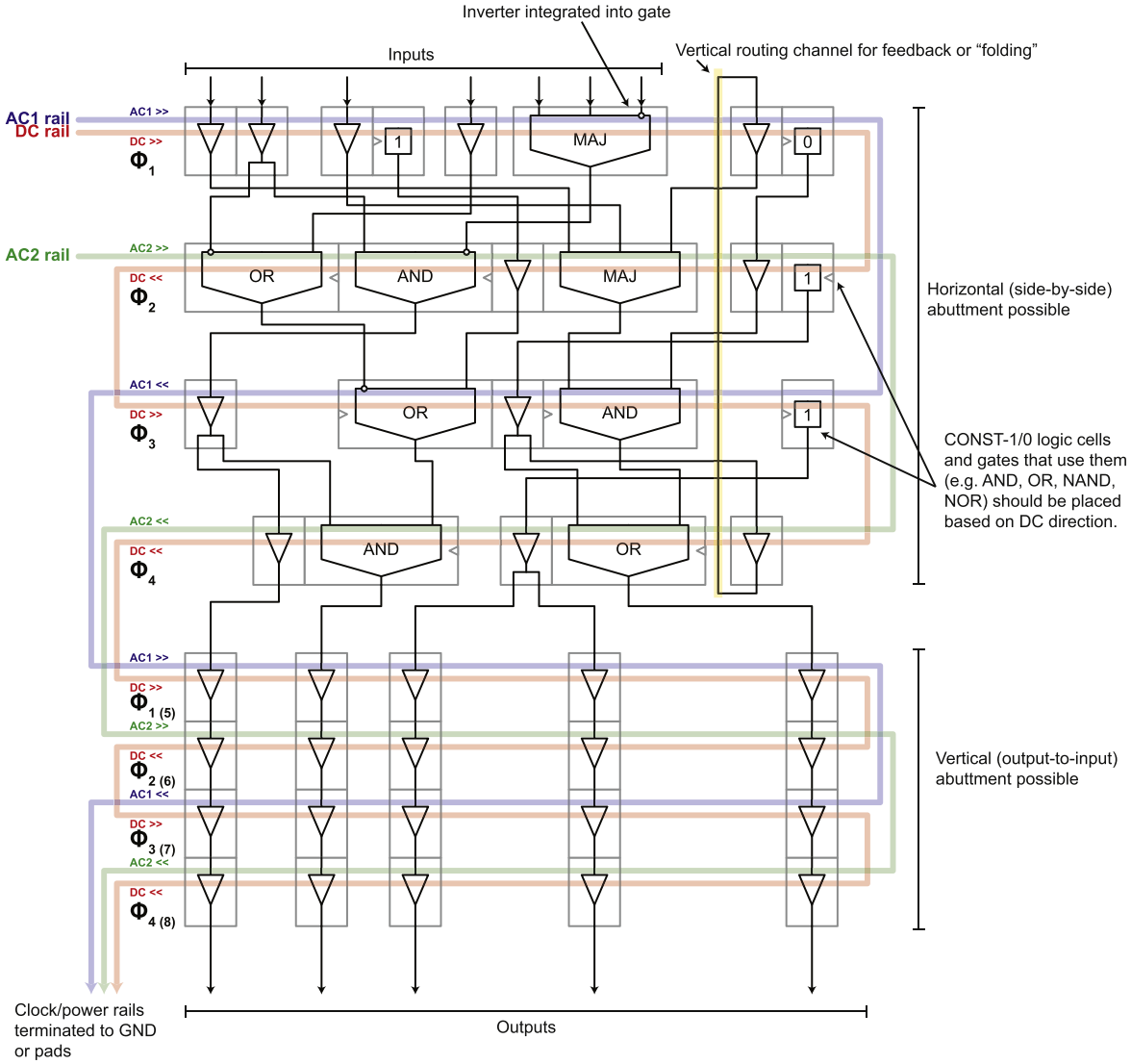


Figure 2. An arbitrary example of AQFP logic circuit design using symbols scaled to the physical layout. The four-phase clock network is composed of 2 ac sinusoidal signals (AC1 rail in blue and AC2 rail in green) with a relative phase shift of 90°, and a dc-offset provided by the DC rail.

2.2. Logic circuit design

Figure 2 shows an example of an AQFP logic circuit using symbols that are scaled to the actual physical dimensions. Logic cells are arranged in logic rows where cells that are to be clocked by the same phase co-exist in the same row. Cells can be abutted together side-by-side or bottom-to-top as depicted in the illustration. The four-phase clock is produced by two ac lines (AC1 rail and AC2 rail) and a dc offset (DC rail). The AC1 rail meanders through the odd rows, whereas the AC2 rail meanders through the even rows. The DC rail meanders through all logic rows. With this arrangement, one can note the relative directions of the AC rail and DC rail for each row. They determine which of the four excitation phases that the entire row is being clocked by [19]. Furthermore, it is important to note the direction of the DC rail in each row as it determines how to orient logic cells built with the CONST sub-cell such as AND, OR, NAND, NOR, and standalone CONST1/0. The symbols for these logic cells are denoted with an arrow indicating which orientation the DC rail should flow. Designers and cell placement tools need to be aware of this, otherwise a CONST1 cell would behave as a CONST0 and vice versa when placed incorrectly.

Because data must flow from one clock phase to the next in a synchronous manner, it is natural to arrange the logic rows in an orderly fashion such that a given row i is clocked by phase $i \bmod 4$ where $i \in \mathbb{N}_0$, with the row after that clocked by phase $(i+1) \bmod 4$, and the next clocked by $(i+2) \bmod 4$, and so on. This generally indicates that the data signals are locally connected to adjacent rows as shown in figure 2. However, in order to accommodate feedback loops the designer should allow space within logic rows as a feed through or pass through vertical channel for the data interconnect to route back to other logic rows instead of just the adjacent row. This also creates the opportunity for circuit 'folding' in which cells clocked by row x can be moved to row y as long as $(x \bmod 4) \equiv (y \bmod 4)$, where $x, y \in \mathbb{N}_0$.

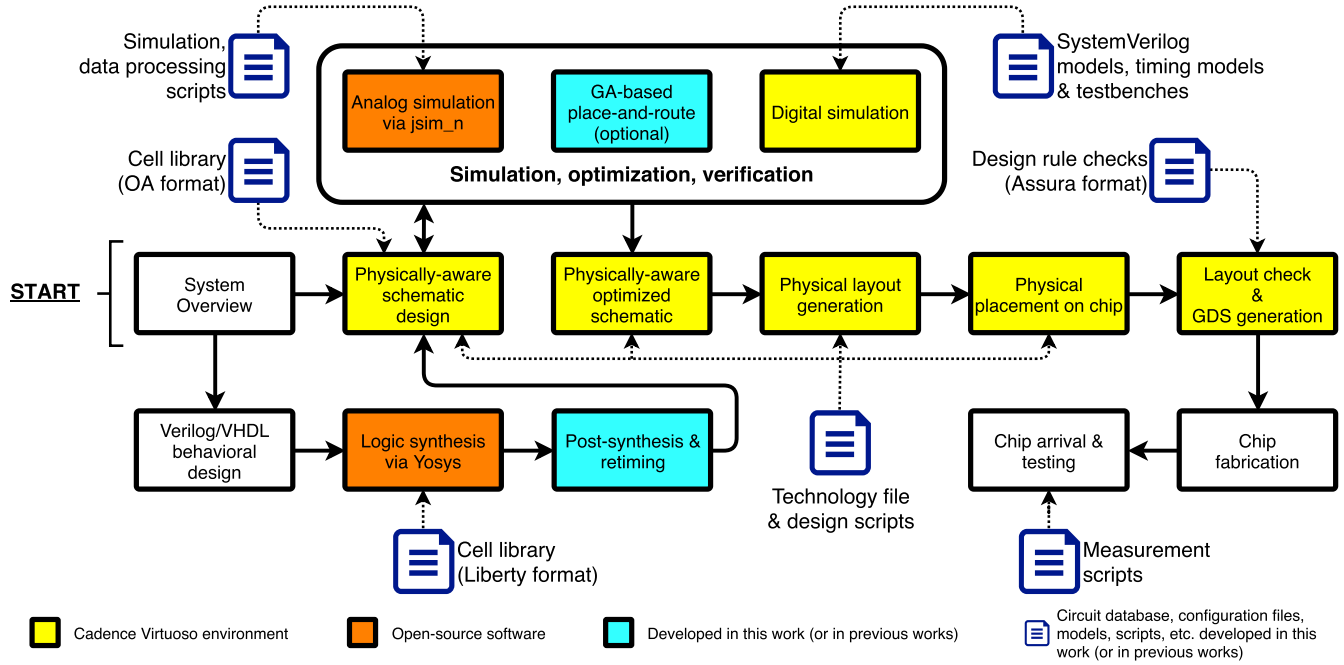


Figure 3. Semi-custom design flow and environment. The yellow boxes indicate tools that belong to the commercial Cadence Virtuoso design environment, the orange boxes indicate tools that are open-source, and the light-blue boxes indicate tools that have been developed by this group. Design files, configuration files, models, and scripts were also developed by this group.

This can allow one to move cells on crowded rows to more sparsely populated rows as long as timing constraints are met.

3. Design environment

The design environment is based on the Cadence Virtuoso full-custom integrated circuit (IC) layout suite. However, we augment this environment with various open-source tools as well as software and scripts developed in our group. Figure 3 shows the overall diagram of the design flow and environment. A semi-custom flow would usually start with the system overview or specification followed by the diagraming of the circuit at the schematic-level using the cell library symbols scaled to the physical layout as mentioned in section 2.1. Because of this, we term this step as 'physically-aware' schematic design as the designer is already considering how to place and route the cells using the schematic symbols. The schematic of course forms a proper netlist which can be simulated in an analog simulator. The designer also has the option to run a place and route tool on their 'physically-aware' schematic in which the genetic algorithm (GA) performs the cell placement and the left-edge channel routing algorithm performs the interconnect routing [23, 24]. Cells also have built-in SystemVerilog digital models to perform digital simulation on the schematic [25]. Once the schematic has been optimized and verified through analog/digital simulation, a script is ran to convert the schematic into the physical layout. The circuit layout is then placed on a pad frame where the circuit is wired up to the pads. A design rule check (DRC) is performed to ensure the final layout has no errors using Assura. In general, the generated circuit itself should have no DRC errors as it follows a correct-by-construction philosophy. The cells that make up the generated circuit have already passed many DRC runs during their development, including the careful consideration of how the cells are structured to abut and connect with each other. This is also why we do not have a layout-versus-schematic check because the construction philosophy implies that the schematic is already equivalent to the layout. Nonetheless, we still do a final DRC sign-off as a sanity check before we generate the GDS file for submission to the foundry.

Another aspect of this design environment is the ability to code high-level behavioral Verilog/VHDL circuit descriptions and run it through an open-source synthesis suite called Yosys [26]. Through the use of a custom-made Liberty file for the AQFP cell library, Yosys is able to generate a partial netlist which we process using our own scripts and tools for post-synthesis so that the netlist becomes a valid AQFP circuit [27]. We also introduce a new step after post-synthesis called retiming to reduce the number of buffers needed to maintain data synchronization. After these steps, the netlist can then be imported into the Cadence Virtuoso environment where the designer can continue with the design flow described earlier.

In the next sections, we describe in more detail the different tools that make up the environment.

3.1. Cadence Virtuoso backend

As stated previously, the design environment is centered on Cadence Virtuoso. Being a full-custom IC suite, we started with Cadence Virtuoso as a means to develop the AQFP cell library. Building the cell library itself is effectively a full-custom design task. Cadence is already well-supported by AIST and other Japanese laboratories working with the superconductor foundry [16, 17] and we are able to get started

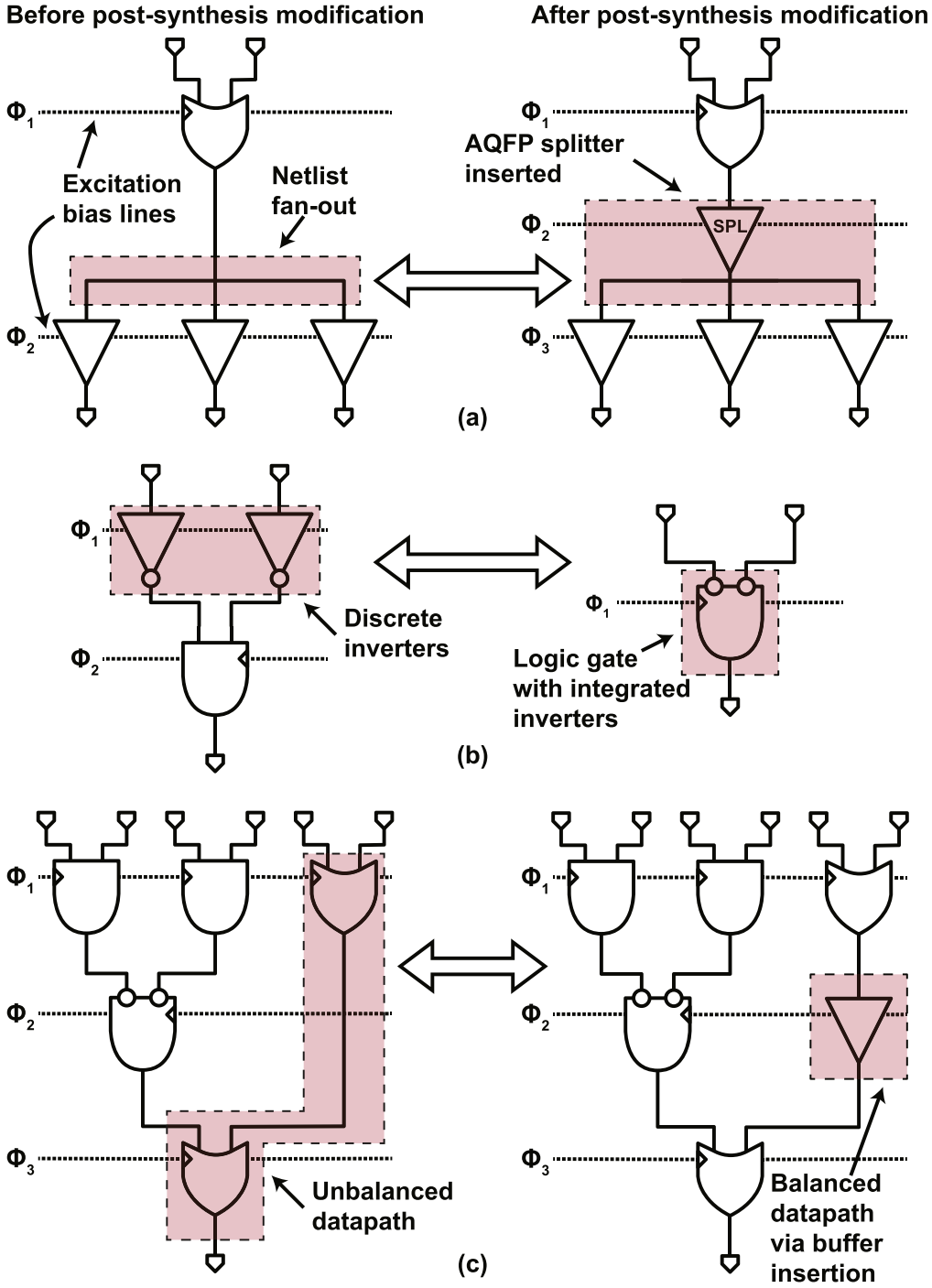


Figure 4. Post-synthesis steps for AQFP logic. (a) Replacing the passive netlist fan-out with an active AQFP splitter element. (b) Collapsing discrete inverter cells with the proceeding logic cell. (c) Rectifying unbalanced datapaths by inserting the appropriate number of buffers to maintain data synchronization from one clock phase to the next.

with the already provided technology files, device primitives, and DRC decks already in place.

The suite also provides numerous ways to import and export your circuit data including the standard GDS files and OpenAccess format. Combined with scripts already available from [15], we have a basic means to simulate our full-custom schematics (e.g. JJ-level schematics) with JSIM. Furthermore, our cell layouts can be easily exported to GDS for inductance and mutual coupling extraction using InductEx.

On the semi-custom end, we heavily use the digital simulator NC Verilog, as a part of the Cadence Xcelium suite for digital simulation. The Assura DRC tool is applicable to both the full-custom cell design and semi-custom logic circuit design.

3.2. Logic synthesis and retiming optimization

3.2.1. Previous work. Our initial work on a logic synthesis methodology for AQFP logic is described in [27]. We used

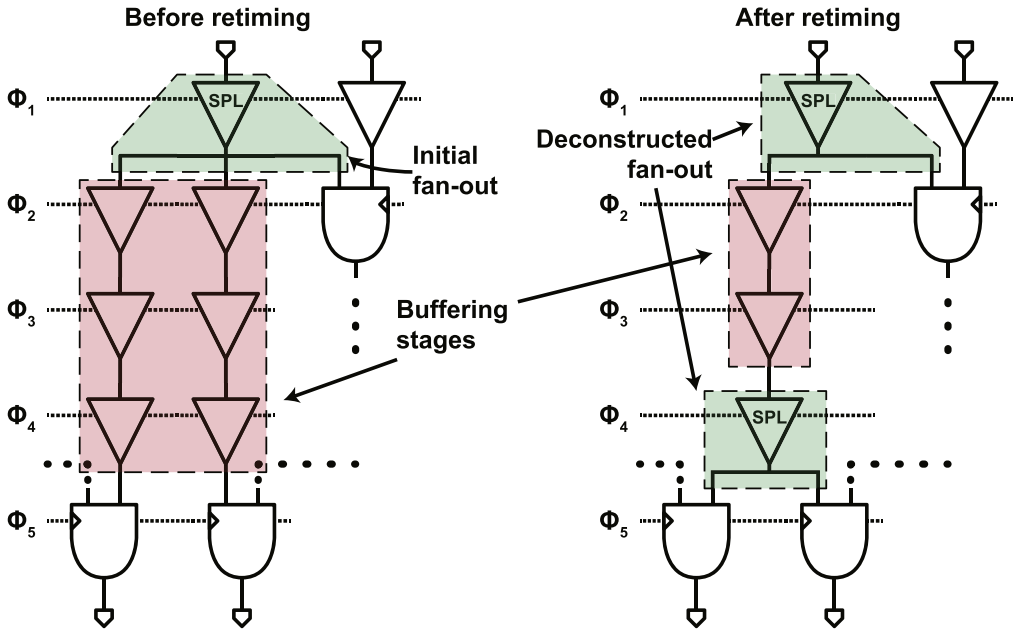


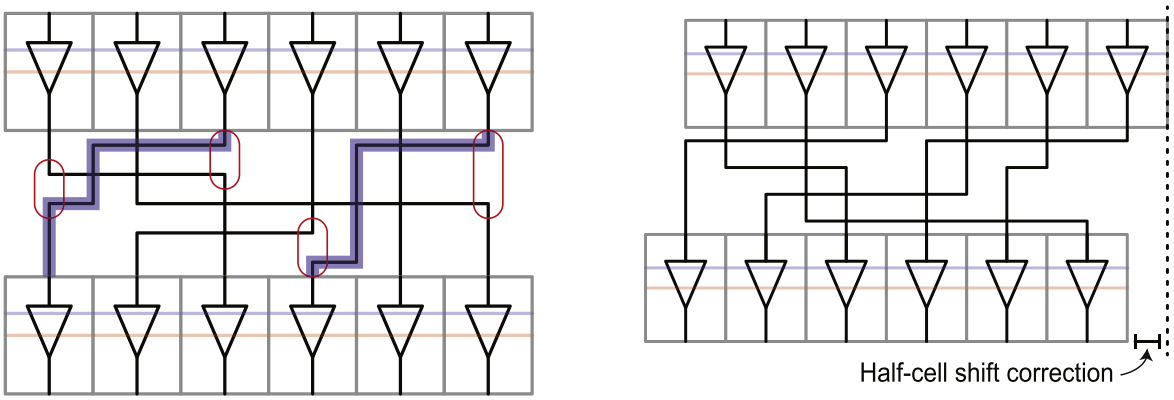
Figure 5. Retiming of AQFP logic circuits. Fan-out elements can be deconstructed and placed near destination logic cells to reduce needless buffering of data signals that were split early in the dataflow.

Table 1. Circuit optimization results before and after retiming has been applied to several benchmark circuits.

Circuit benchmark	Total JJs[a]	Total cells[a]	Buffer count[a]	Buffers removed	JJs removed	Cell reduction	JJ reduction
4 bit adder	326	115	71	12	24	0.14	0.07
8 bit	1088	436	338	80	160	0.19	0.15
16 bit	3790	1671	1458	378	756	0.21	0.20
32 bit	14 128	6602	6177	1704	3408	0.22	0.24
4 bit multiplier	1202	436	292	59	118	0.17	0.10
8 bit	6504	2406	1620	443	886	0.21	0.14
16 bit	30 962	11 564	7886	2263	4526	0.22	0.15
32 bit	130 102	48 903	33 543	9250	18 500	0.22	0.14
4-to-16 line decoder	288	83	26	4	8	0.13	0.03
5-to-32	660	206	90	23	46	0.20	0.07
6-to-64	1888	693	457	84	168	0.16	0.09
7-to-128	5468	2228	1757	329	658	0.16	0.12
8-to-256	18 570	8268	7325	1343	2686	0.15	0.14
9-to-512	65 670	30 795	28 913	5396	10 792	0.16	0.16
4 bit shifter-rotator	584	179	92	11	22	0.11	0.04
8 bit	2276	760	459	128	256	0.22	0.11
16 bit	8684	3266	2424	464	928	0.16	0.11
32 bit	33 358	14 051	12 003	1989	3978	0.14	0.12
64 bit	105 878	46 551	41 635	15 449	30 898	0.27	0.29
128 bit	444 802	207 204	195 531	82 518	165 036	0.30	0.37
					Minimum reduction:	0.11	0.03
					Average reduction:	0.19	0.14
					Maximum reduction:	0.30	0.37

[a] Total JJs is the total number of Josephson junctions, total cells is the total number of logic cells including buffer cells, and buffer count is the total number of buffers. These totals are all before retiming has been applied.

the Yosys synthesis suite [26] to generate a CMOS-like structural Verilog netlist. Yosys is able to map combinational circuits to AQFP logic cells through a Liberty file that describes our cell library and the different logical functions. The resulting netlist resembles that of a conventional CMOS circuit and is not quite ready to be simulated in our AQFP design environment. Figure 4 shows the different post-synthesis modifications that must be applied before the netlist



(a) Vertically aligned rows (b) Rows with half-cell shift correction

Figure 6. Comparison of vertically aligned placement versus placement with a half-cell shift correction. (a) Due to vertically aligned rows, there are column conflicts as shown in the red circles. Dogleg routing (highlighted in purple) is needed to overcome the column conflicts, but also creates additional routing complexity. (b) By introducing a half-cell shift between adjacent rows, the logic cell pins no longer align and cause column conflicts simplifying the implementation of channel routing.

appears more like a correct AQFP logic circuit. First, the passive fan-outs in the netlist must be replaced with active AQFP splitters (figure 4(a)). Second, an optimization step can be applied in which standalone inverters are collapsed into the proceeding logic cell (figure 4(b)). This is because AQFP logic cells can create inverted data in-place for any logic cell, there is no need to have a discrete inverter in an AQFP logic circuit netlist. And lastly, all cells must receive their input from the previous clock phase and transmit their output to a cell that is clocked on the next clock phase. It is not possible to transmit data across more than one phase; data must always be properly synchronized from clock phase to clock phase. Figure 4(c) shows an example of a Yosys generated netlist where one cell transmits data to a cell that is clocked two phases ahead of it, which is perfectly fine in a conventional CMOS netlist but not for AQFP logic. This unbalanced path must be re-balanced by inserting a buffer (or as many buffers needed) so that data is safely transmitted from one phase to the next. We automated splitter insertion and merging discrete inverters into logic gates through a Python script developed in [27], but inserting buffers to ensure data synchronization was done manually.

3.2.2. Automating synchronizing buffer insertion. To automate buffer insertion for proper data synchronization, we first developed a structural netlist interpreter and analyzer which determines the clock phase for each cell in the netlist. Listing 1 shows the pseudo code of the phase analysis. If cells and signal lines of the circuit are considered as nodes and edges respectively, the circuit can be considered as a directed graph. Each node in the netlist operates as an event sender and an event receiver. An 'event' is a packet of information containing a counter that corresponds to the phase number. This event is transferred through the edge that connects two nodes. The phase of a node is calculated as `MAX(node.receivedPhase)+1` or in other words, after events have been received from all inputs to the node, use the largest event information received (the phase) and increment it. Once the phase is fixed, the node can prepare its own event packet with its current phase information and transmit it to all output ports (all output edges).

Listing 1. Phase analysis pseudo code

```
1 //Event listener
2 Function(phase):
3   node.receivedPhase.Add(phase);
4
5 //State checker:
6 if node.inputList.Length==node.receivedPhase.
  Length:
7   node.phase=Max(node.receivedPhase) + 1;
8   for output_port in node.outputList:
9     Send_Event(output_port, node.phase);
```

For example, top-level circuit input pins are fixed to phase 0 at first. Then this 'phase 0' event is transferred from an edge of the input pin to another cell. Assume a two-input one-output AND cell just received a single 'phase 2' event. At this time, this AND cell cannot send an event out because its phase is not yet fixed. It still needs to wait for an event to arrive at its other input. Next, a 'phase 4' event arrives at the other input of the AND cell. Now it can calculate its phase by taking the maximum of its two received events and incrementing it: $\mathrm{MAX}(2,\ 4) + 1 = 5$. The AND cell can now send its 'phase 5' event to all of its outputs. Through this process, the phase information is propagated in parallel and eventually each node will determine its own phase. At present, this approach does not automatically handle feedback loops in the netlist.

To determine where to insert buffers, we calculate the phase difference $n = B - A$ where B and A are the phases of two nodes connected by an edge. If $n \geqslant 2$ then we need to insert a buffer chain between A and B of length $n - 1$.

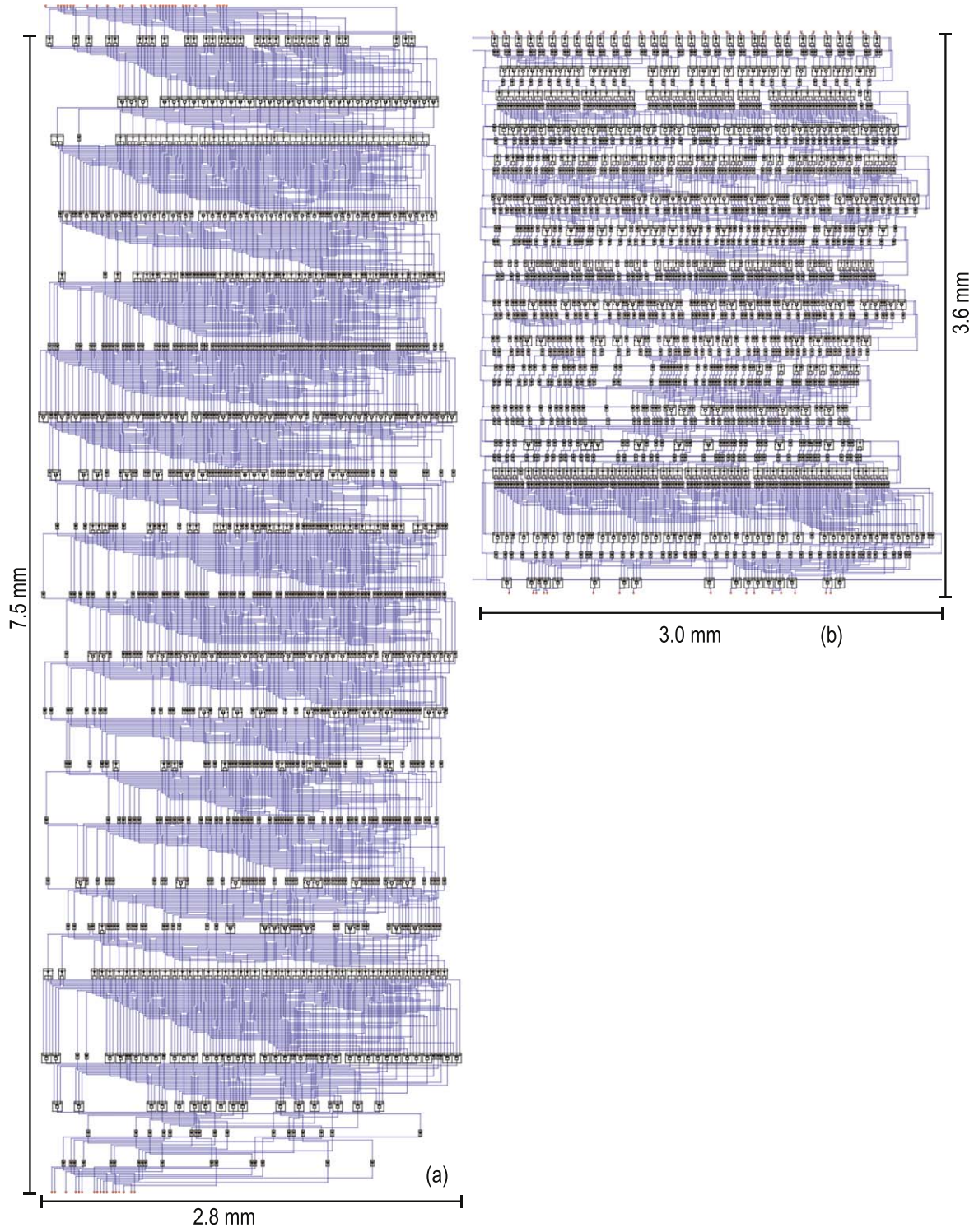


Figure 7. GA-based place and route result of a 16 bit Kogge–Stone adder before implemented refinements (a) and after (b). The dark rectangles are logic cells placed in rows whereas the blue lines are the interconnect.

3.2.3. Retiming for buffer reduction. Through the development of the logic synthesis methodology, it became clear that the largest part of the composition of the synthesized circuits are just buffers to maintain data synchronization. Table 1 shows a list of various benchmark circuits including parallel adders, parallel multipliers, line decoders, and shifter-rotators of different sizes. The table also shows the total cell count as well as the buffer count. It is clear that for all these benchmark circuits, buffers take up a large portion of the design. One way to reduce the number of synchronizing buffers is through retiming [28].

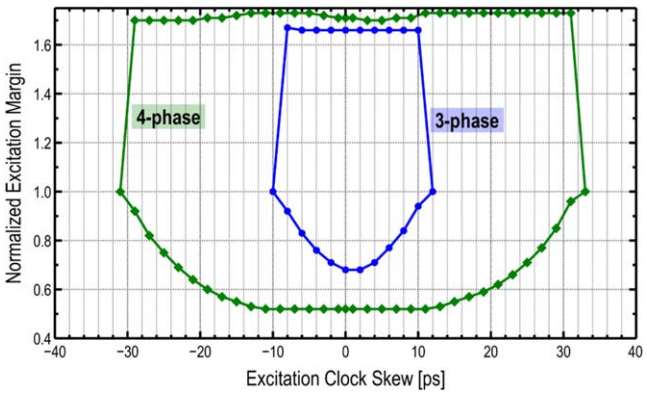


Figure 8. Extracted timing window at 5 GHz using JSIM_n for four-phase clocking and for three-phase clocking. The larger timing window in four-phase clocking is a primary driver towards creating the design methodology around four-phase clocking.

Retiming involves manipulating data taps (or splitters) and moving buffers from the output side of a splitter to the input side.

The difference in excitation phase d between a splitter (at phase n) and a logic cell A (at phase a) is represented as

$$d = n - a. \tag{1}$$

When $d \geqslant 2$, the splitter can be relocated just before the logic cell. Its new excitation phase n' can be calculated as

$$n' = n + d - 1. \tag{2}$$

The number of reduced buffers r is represented as

$$r = (f - 1) \times (d - 1), \tag{3}$$

where f is the number of fan-outs of the splitter.

If another logic cell connected to one of outputs of the splitter shows $d < 2$, we can recursively process the remaining outputs of the splitter that still satisfy $d \geqslant 2$ resulting in deconstructing a large fan-out into two or more smaller fan-outs as shown in figure 5.

We tested our retiming approach on a collection of benchmark circuits synthesized by Yosys and then process them with our post-synthesis interpreter to handle AQFP-related optimizations. The benchmarks were written in behavioral Verilog with a generic size parameter to easily change the size of the circuit. Table 1 shows the results of retiming. The approach appears to be more effective when the circuit size grows. We note an average reduction of 14% in JJs with 37% being the best reduction. We expect improved reduction through the introduction of larger fan-out cells to enable more ways to deconstruct fan-outs. For now, we only considered retiming with a maximum fan-out of 3.

3.3. Place and route

3.3.1. Previous work. The place and route tools in this environment were developed in [23, 24]. The logic cell placer uses the GA [29]. The algorithm generates a population of random placement results and evaluates each result with a fitness function. Because AQFP logic has a soft cell-to-cell interconnect length constraint of 1 mm [18], we designed our fitness function as shown in equation (4):

$$\text{fitness} = \sum_{i=1}^{\text{Num. of nets}} \frac{1}{(\text{length}_i < 1\text{ mm})\,?\,(\text{length}_i)\text{: }(\text{length}_i \times c)}, \tag{4}$$

where c is a weighted penalty multiplier. In short, the more nets whose length is greater than 1 mm exist, the lower the fitness of the placement result. The best placement results are kept and undergo a crossover process where the best results are systematically combined to generate new placement results (children). The fitness of these new placement results are then evaluated and the process repeats for the next generation. This continues until a terminating condition is met such as reaching a certain fitness level, reaching a certain fitness stagnation level after a number of generations, or reaching the maximum number of generations. Furthermore, mutation results are also introduced randomly to prevent the algorithm from falling into a local minimum.

The left-edge channel routing algorithm is used to route the interconnect between placed cells [30]. Conventionally, if we do not assume feedback loops and circuit folding, the logical rows simply propagate data to the next adjacent row just as we showed in figure 2. This makes the left-edge algorithm very suitable for completing the row-to-row routing of AQFP logic circuits. For sufficiently large circuits, it is impossible to completely eliminate all interconnects longer than the constraint of 1 mm. For routing channels (the routing region between adjacent logic rows) where there exists at least one net with a length longer than this constraint, we insert a buffer to serve as a signal repeater [23]. To maintain data synchronization, buffers must be applied to all nets in this routing channel.

3.3.2. Improvement of GA and routing. As we tested our place and route tool on larger circuits, the results became unreasonably large in area. Our placer applied GA to the entire circuit as a whole and when the circuit becomes large, it becomes more difficult to converge to a sufficient solution. One refinement we applied is we conducted the GA process not on the entire circuit but on smaller sections of the circuit. More specifically, we applied GA to adjacent logic rows where

the driving logic row cells (the row that is producing the data signals) have a fixed position, and the receiving logic row cells (the row that receives data signals from the driving logic row) are optimized via GA. Once a sufficient solution has been found, the position of the receiving cells is fixed and that row of cells now becomes the driving logic row, whereas the subsequent logic row becomes the new receiving logic row. In other words, optimization is now done on a row-by-row basis and not by applying GA globally on the entire circuit.

In addition, we exploited a critical property that was key in simplifying the routing problem which is with the way the logic cells and the interconnects were designed relative to each other. Assuming logic cells are tightly packed in a row and the output pins are vertically aligned with the input pins of the row below, we would have column conflicts that would make channel routing more difficult. Figure 6(a) illustrates this exact scenario. The red circles denote where column conflicts are occurring and the highlighted purple routes show that the channel router must be able to produce 'dogleg' routes in which a horizontal segment of wire must be broken up into two horizontal segments existing on different tracks. However, if we introduce a half-cell shift in which we shift the entire row by half the width of a BFR cell ($20\ \mu\text{m}/2 = 10\ \mu\text{m}$) as shown in figure 6(b), we can avoid all those column conflicts and use a very simple implementation of channel routing without 'dogleg' routes. This is possible because the data I/O pins that exist on the top and bottom of the cells have a minimum pitch of $20\ \mu$m even when cells are abutting next to each other as depicted in figure 6. The minimum pitch of the interconnect is $10\ \mu$m so it is capable of using the space between pins as a branching point (vertical wire segment) without conflicting with another I/O pin. Of course this works if the cells are placed on multiples of $20\ \mu$m along the x-axis but the placer has a heuristic adjustment step to try to fit the cells in this manner after GA finishes.

Figure 7 shows an example of a place and route result of a 16 bit Kogge–Stone adder before we implemented the above refinements and afterwards. Figure 7(a) is the place and route result after the GA placer (before refinements) detected stagnation of the fitness level. Figure 7(b) is the result using the refined GA. Approximately 50% improvement in area was obtained.

3.4. Analog simulation

The basic framework to support analog simulation from the schematics designed in Cadence Virtuoso has already been prepared in [15]. However, additional scripts were needed to do modify the generated JSIM netlist to quickly change data patterns, frequency, and eventually more complex scripts to evaluate the parameter margins of cells and to characterize the timing window of cells [31]. We are currently using the noise-enabled flavor of JSIM (JSIM_n) [32] which also provides more precision in the output data.

More recently, the IARPA SuperTools project has called upon the need to develop more modern analog simulators for superconductor electronics. One of those efforts is JoSIM [33].

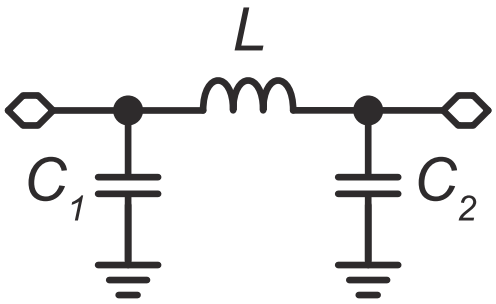


Figure 9. First order modeling of excitation microstripline delay for a length of 5 μm where $L = 1.55$ pH, $C_1 = C_2 = 0.31$ fF.

JoSIM is a drop-in replacement for JSIM, so it is a good fit for our design environment. A major weakness of JSIM is its inability to handle large circuits, often crashing when attempting to simulate a logic circuit consisting of a few thousand JJs. JoSIM aims to rectify this issue in the long term. In the meantime, we are still validating JoSIM with our JSIM results.

3.5. Digital simulation

3.5.1. Previous work. Our first efforts in developing the digital cell-level models for AQFP logic started in [25]. Its features include the following:

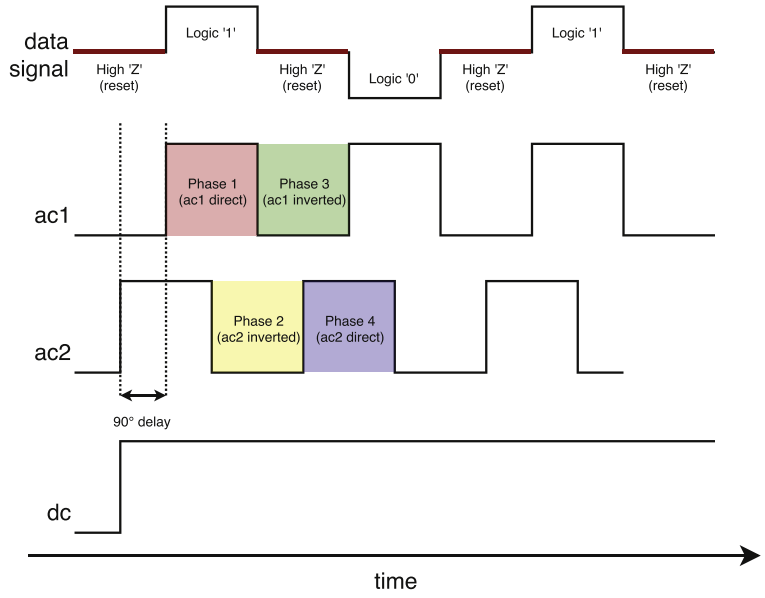
- Using high 'Z' in the data output to faithfully model the reset/null state of the AQFP
- Built-in setup and hold time checks
- Modeling of clock skew

We used a tri-level encoding of the AQFP data signal with logic '0', logic '1', and high 'Z'. High 'Z' corresponds to the AQFP when it is reset after an output is produced. This is particularly important for adding setup (minimum time separation for input to be steady before the clock arrival) and hold time (minimum time input must remain steady after the clock arrival) checks into the model. By sweeping the clock skew applied to the AQFP, we can obtain a timing window as shown in figure 8 and extract the timing checks to be built into the digital models of the gates.

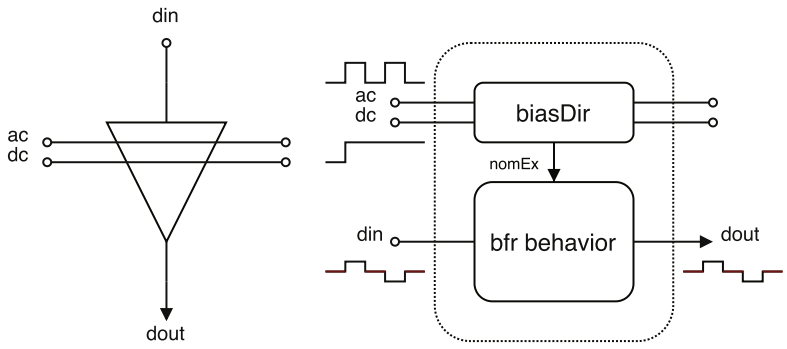
To model clock skew, we performed a first order approximation of the delay of a 5 μm microstrip line as seen in figure 9 with $L = 1.55$ pH and $C_1 = C_2 = 0.31$ fF. Equation (5) shows that the approximate delay of a 5 μm microstrip line is about 0.031 ps. This can be equivalently expressed as a delay per millimeter or 6.20 ps mm^{-1}. This delay is provided to the logic and wiring cells as a transport delay in the digital models

$$t_{5\mu\text{m}} = \sqrt{LC} = \sqrt{L(C_1 + C_2)} \approx 0.031\ \text{ps}. \tag{5}$$

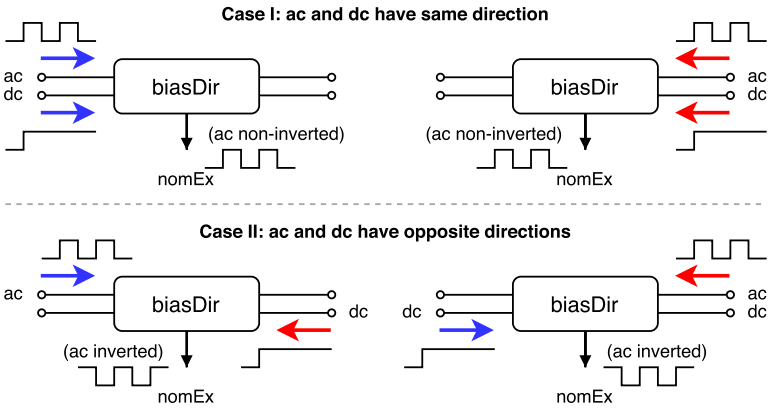
3.5.2. Supporting four-phase clocking. Our AQFP cell library initially used three-phase clocking early in its development. Figure 8 shows that four-phase clocking has an even larger timing window when compared to three-phase clocking. Thus, a four-phase cell library based on using 2 ac excitation currents and 1 dc offset was developed [19]. This introduced new challenges in



(a) Digital representation of data, ac and dc signals



(b) BFR symbol and corresponding digital model



(c) Two cases to normalize the excitation clock

Figure 10. Modeling of the four-phase clock in digital simulation. (a) The testbench provides the appropriate digital representation of the data, ac, and dc signals as depicted. (b) Digital interfacing symbol and its internal building-blocks composed of a `biasDir` module and the logic behavior of the cell. (c) `biasDir` creates a normalized excitation clock based on the relative directions of the ac clock and dc offset provided to the cell.

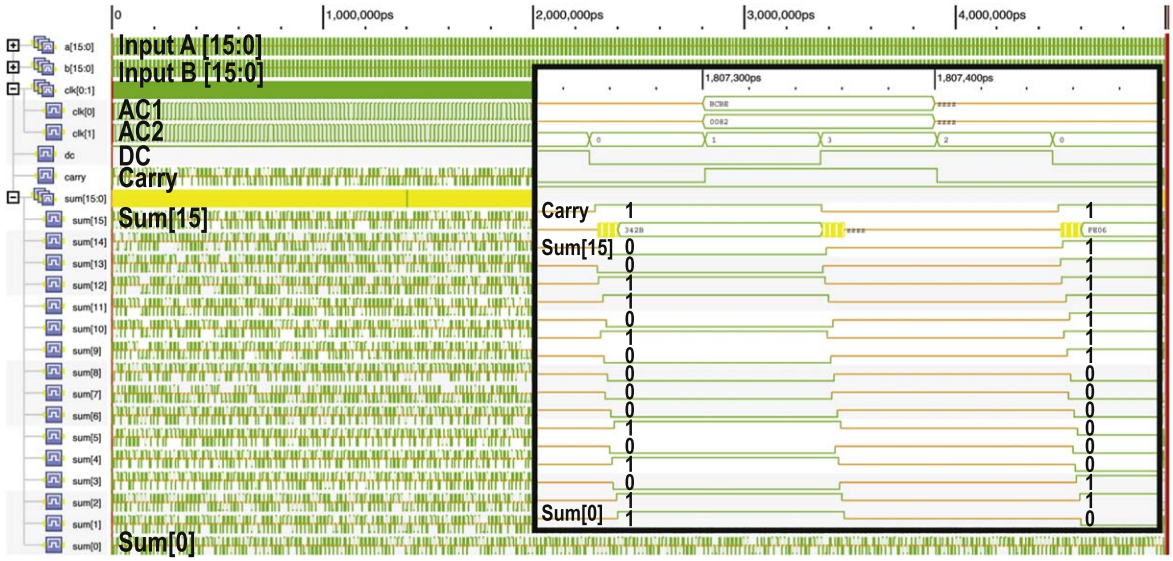


Figure 11. Digital simulation of a 16 bit Kogge–Stone adder at 5 GHz based on the modeled four-phase clocking approach using 2 ac signals and 1 dc signal. Note the time skewed output produced from the first order modeling of the microstripline delay of the excitation clock line in the inset.

modeling the clock for four-phase AQFP because each cell in the digital simulation must understand which clock phase it belongs to. Previously, this was simply connecting the cell to one of three ac excitation lines in three-phase clocking.

With four-phase clocking, we need to extract the excitation phase of a cell through both the ac signal and dc signal it is connected to. Figure 10(a) shows how signals are represented in digital simulation including the ac excitation signal, the dc offset, and the data signals transmitted from one cell to the next. Specifically, the testbench provides a digitized version of AC1, AC2, and the dc offset. The arrival direction of these signals at each cell during simulation is exactly what determines the clock phase that each cell is assigned to. Figure 10(b) illustrates the symbolic view of the BFR from perspective of a user. Internally, that BFR is composed of a module called `biasDir`. This module observes from which direction the ac and dc signals arrive into the cell. Figure 10(c) shows each arrival scenario and how it manipulates the incoming ac signal to provide a normalized excitation clock to the behavioral model of the cell so that it operates on the correct excitation phase. There are two cases: (1) if ac and dc arrive in the same direction, the normalized ac signal applied to the model will remain the same as the original ac signal. (2) If ac and dc arrive in opposite directions, the normalized ac signal applied to the model will be the inverted version of the original ac signal. With the two ac signals (AC1 and AC2 with a phase difference of 90°) combined with the two cases described above, each of the four-phases can be properly assigned to all connected cells. The timing checks were also updated to take into account the larger timing window of four-phase clocking. Figure 11 shows the digital simulation of a 16 bit Kogge–Stone adder using the newly established four-phase digital models. Because of the physically large width of the circuit, one can observe the notable time skew of the output bits that our digital models support.

4. Demonstration of prototype chips

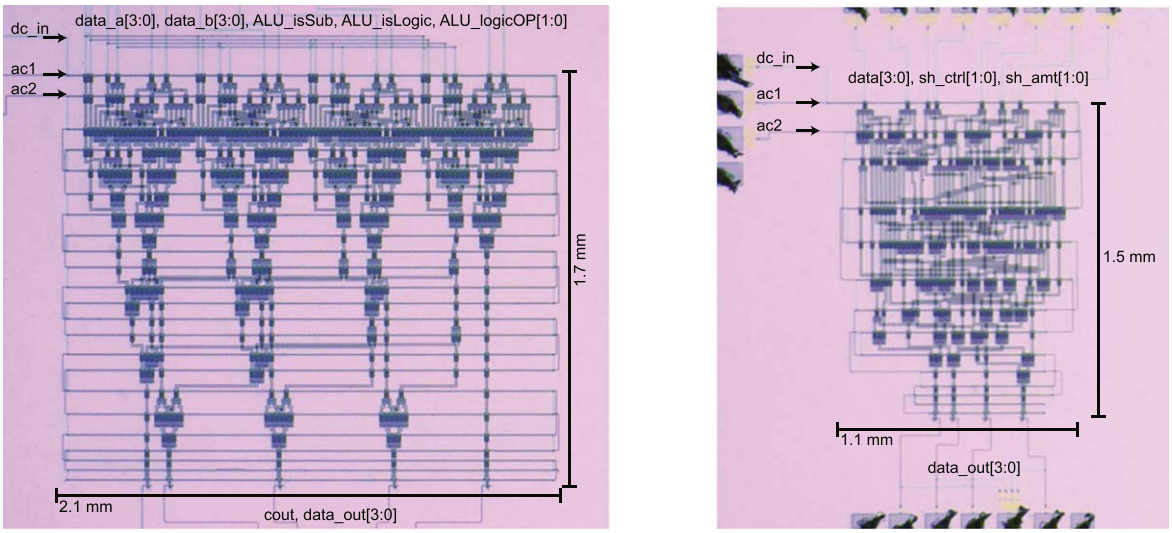
4.1. Previous work

Throughout the development of this design methodology and environment, we have already designed and experimentally verified various AQFP logic circuits including parallel adders, register files, and shifter-rotators [24, 34–36]. Each of them focused on different aspects of the design environment and served as test vehicles for debugging and refining our methodologies to the state that it is now. More recently, a high-speed demonstration of an 8 bit Kogge–Stone adder designed and verified using this environment has been reported [9].

4.2. Microprocessor execution units

Towards the development of a microprocessor, we used this environment to design the execution units of the microprocessor. These include a 4 bit arithmetic logic unit (ALU) and 4 bit data shifter. Figure 12(a) shows the micrograph of the ALU. It performs 6 logical functions, namely `xor`, `xnor`, `and`, `or`, `and_notb` ($a\bar{b}$), and `or_notb` ($a + \bar{b}$). It also performs integer addition and subtraction. The structure uses the Kogge–Stone parallel prefix tree [37] and adopts majority-based logic [34]. The circuit has an area of 2.1 × 1.7 mm, a latency of 20 phases (5 cycles), and is composed of 1058 JJs. Because of the structural nature of the design, it served as an excellent candidate for manual design but with assistance from various interactive design scripts to help accelerate the manual design flow and verification.

The 4 bit data shifter performs 4 types of shifter operations, namely `sll` (shift left logical), `slr` (shift right logical), `sla` (shift left arithmetic, equivalent to shift left logical), and `sra` (shift right arithmetic). The shift amount can vary from 0 to 3. This is also one of the first designs completely done starting from a Verilog behavioral description to layout using the tools available in our design environment, and ultimately



(a) 4-bit ALU chip (b) 4-bit data shifter chip

Figure 12. Micrograph of the prototype execution units for an AQFP microprocessor under development using the established design methodology and environment.

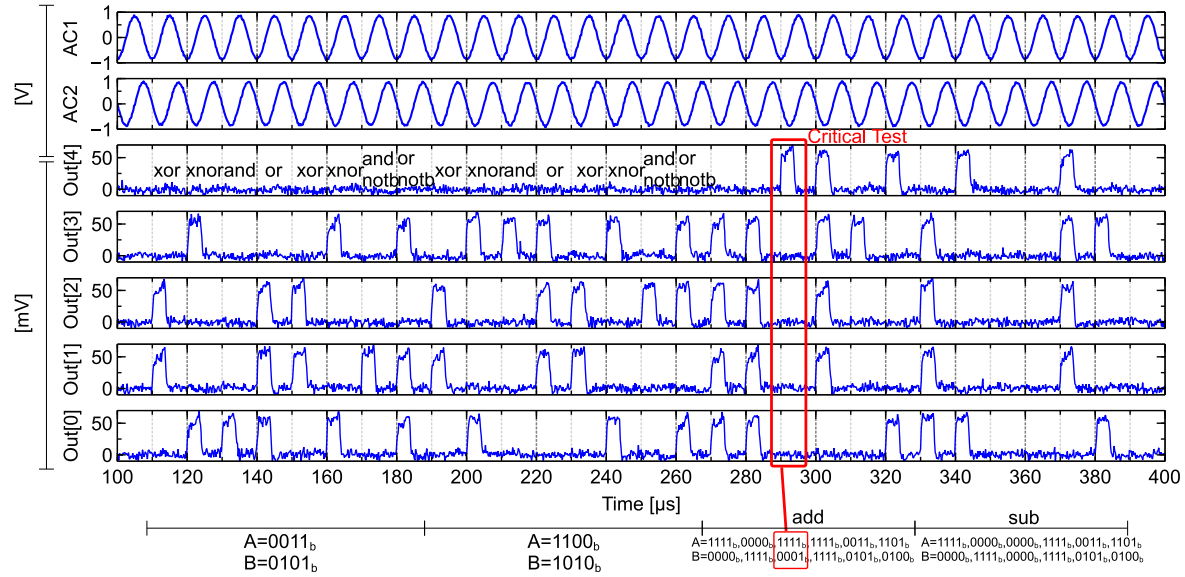


Figure 13. Measurement output waveform of the 4 bit ALU. The first set of test patterns iterate through all the logic operations for a fixed data pattern where the most significant bit (index 3) is the left-most binary digit. The two 4 bit operands collectively cover the two-input truth table for each logic operation. The test is repeated again with the data pattern inverted. Then a series of add and subtract operations are tested including a critical carry test where the carry signal propagates from the least-significant bit to the carry-out (out[4]). All patterns are correct.

showing full operation in experimentation. The circuit has an area of 1.1 × 1.5 mm, a latency of 10 phases (2.5 cycles), and is composed of 498 JJs.

Both circuits were fabricated using the HSTP 10 kA cm^{-2} Nb/AlO$_x$/Nb process. The chips were tested using an immersion probe lowered into a liquid helium Dewar to achieve a 4.2 K cryogenic environment. We surrounded the chip carrier of the probe with two-layers of Mu-metal shielding. A function generator created 2 ac sinusoidal sources (AC1 and AC2) with a relative phase difference of 90°. A dc offset is coupled to both excitation lines on-chip to create the four-phase excitation clock. The excitation clock lines and dc offset are 50 Ω-terminated at room temperature. We applied a nominal amplitude of 900 μA to AC1 and to AC2. The nominal dc offset is 1.2 mA. We used a data pattern generator to create a +10 μA logic '1' and a −10 μA logic '0'. An on-chip dc-SQUID stack [9] with additional room temperature amplification provides output read-out. It uses return-to-zero encoding such that logic '1' values produce a positive current pulse that resets back to zero with a duty cycle proportional to the excitation cycle time. Logic '0' values produce no activity on the output. Figure 13 shows correct operation over all ALU operations for various data patterns at 100 kHz and figure 14 shows correct operation over all shifter operations and shift amounts for various data patterns at 100 kHz.

Furthermore, we measured 2 chips per wafer across a total of 4 wafers for each circuit. Table 2 summarizes the

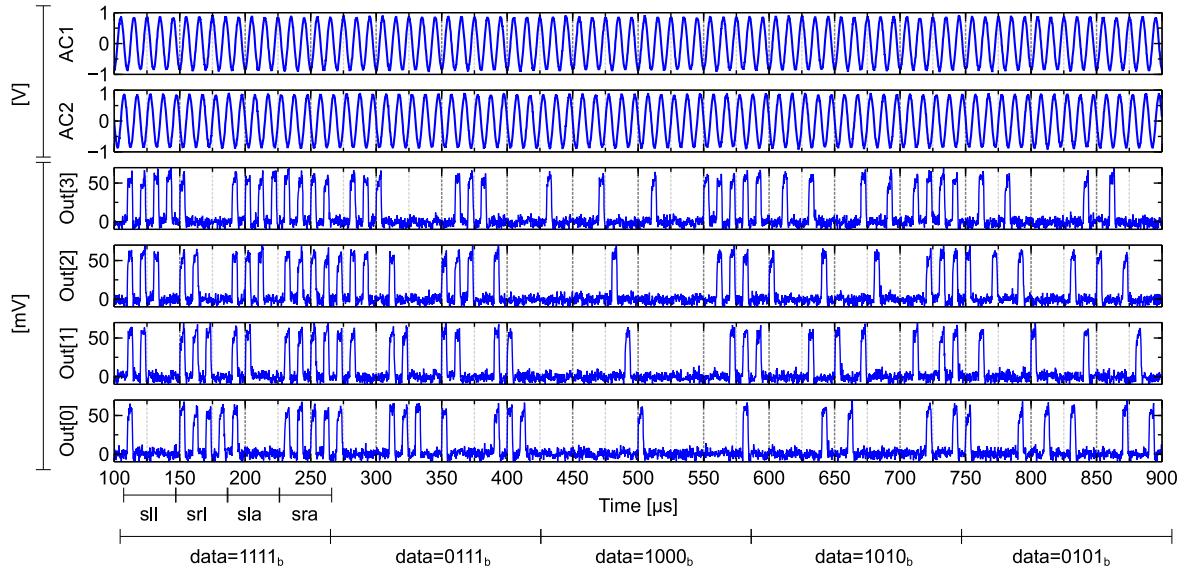
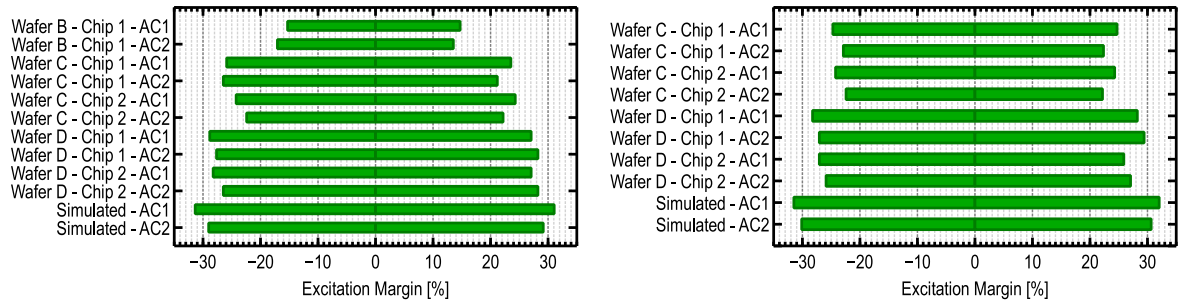


Figure 14. Measurement output waveform of the 4 bit data shifter. Five different data patterns are tested in this example with the most significant bit (index 3) being the left-most binary digit. For each data pattern, all four shifter operations have been demonstrated. For each shifter operation, a shift amount of 0, 1, 2, and 3 have been applied at each cycle. All patterns are correct.



(a) 4-bit ALU (b) 4-bit data shifter

Figure 15. Excitation margins of all fully functional chips for the ALU (a) and the data shifter (b) measured at 100 kHz. The normalized AC amplitude is 900 μA.

Table 2. Summary of measurement results of all ALU and shifter chips designed for the 10 kA cm^{-2} process.

Wafer ID	Measured J_c (kA cm^{-2})	ALU chip 1	ALU chip 2	Shifter chip 1	Shifter chip 2
Wafer A	7.156	X	X	X	U
Wafer B	8.075	F	U	X	X
Wafer C	8.680	F	F	F	F
Wafer D	10.387	F	F	F	F
Working chips:		5/8		4/8	

'X' denotes non-working chips (no output or oscillating output).
'U' denotes partially working chips (unstable output).
'F' denotes fully functioning chips.

measurement results. The measured critical current J_c for Wafers A and B is about 20–30% smaller than the expected value of 10 kA cm^{-2}. This likely had a negative effect on the yield of the circuits on those wafers. For Wafers C and D, the measured J_c was within 15% of the expected value. All circuits from those wafers fully operated. Across all wafers, we successfully demonstrated 5 out of 8 ALU chips, and 4 out of 8 data shifter chips. Lastly, we measured the excitation current (AC1 and AC2) amplitude margins as shown in figure 15 using only the fully functional chips with the midpoint set to 900 μA. The simulated margins are also shown. The best margins were found on Wafer D which were measured to be nearly $\pm$30%. These excitation margins correspond well with the simulation results.

5. Conclusion

We have developed a design methodology and environment for AQFP logic circuits. The environment is capable of supporting completely manual semi-custom design, and semi-automated logic synthesis with post-synthesis and retiming. A place and route tool provides a more automated means to complete the circuit design with reasonable quality-of-results. It has strong verification capabilities for large digital circuits, and has already produced several working designs as demonstrated in experiment including a 4 bit ALU and data

shifter for a microprocessor design with wide operating margins of the excitation currents.

The development direction was to first assist in accelerating the manual design of our circuits with some capabilities for logic synthesis and place and route. With the successful demonstration of several circuits using this flow, it is now clear that future development should be towards building a completely standalone top-down design flow where a designer can simply write the Verilog description of the circuit and provide a configuration file that specifies which tools and/or optimization steps that the top-down design flow should use. This would be a major step towards being independent of expensive commercial tools and is a direction we would like to pursue in the future. Lastly, our environment needs to be further enhanced to support the automation of feedback loop designs and high-performance clocking techniques.

Acknowledgments

This work was supported by the Grant-in-Aid for Scientific Research (S) No. 26 220 904 and the Grant-in-Aid for Early-Career Scientists No. 18K13801 from the Japan Society for the Promotion of Science (JSPS). This work was also supported by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Cadence Design Systems, Inc. The circuits were fabricated in the Clean Room for Analog-digital superconductiVITY (CRAVITY) of the National Institute of Advanced Industrial Science and Technology (AIST) using the standard process (STP2), and the high-speed standard process (HSTP).

ORCID iDs

Christopher L Ayala https://orcid.org/0000-0002-3510-9626
Ro Saito https://orcid.org/0000-0002-8581-3574
Tomoyuki Tanaka https://orcid.org/0000-0002-0020-8712
Olivia Chen https://orcid.org/0000-0002-2208-0262
Naoki Takeuchi https://orcid.org/0000-0003-0396-5222
Yuxing He https://orcid.org/0000-0001-5628-8080
Nobuyuki Yoshikawa https://orcid.org/0000-0001-6191-6715

References

[1] Holmes D S, Ripple A L and Manheimer M A 2013 *IEEE Trans. Appl. Supercond.* **23** 1701610
[2] Dorojevets M, Chen Z, Ayala C L and Kasperek A K 2015 *IEEE Trans. Appl. Supercond.* **25** 1–8
[3] Chen O, Cai R, Wang Y, Ke F, Yamae T, Saito R, Takeuchi N and Yoshikawa N 2019 *Sci. Rep.* **9** 10514
[4] Kirichenko D E, Sarwana S and Kirichenko A F 2011 *IEEE Trans. Appl. Supercond.* **21** 776–9
[5] Mukhanov O A 2011 *IEEE Trans. Appl. Supercond.* **21** 760–9
[6] Herr Q P, Herr A Y, Oberg O T and Ioannidis A G 2011 *J. Appl. Phys.* **109** 103903
[7] Yoshikawa N and Kato Y 1999 *Supercond. Sci. Technol.* **12** 918–20
[8] Tanaka M, Ito M, Kitayama A, Kouketsu T and Fujimaki A 2012 *Japan. J. Appl. Phys.* **51** 053102
[9] Takeuchi N, Yamae T, Ayala C L, Suzuki H and Yoshikawa N 2019 *Appl. Phys. Lett.* **114** 042602
[10] Takeuchi N, Yamanashi Y and Yoshikawa N 2014 *Supercond. Sci. Technol.* **28** 015003
[11] Tolpygo S K, Bolkhovsky V, Weir T J, Wynn A, Oates D E, Johnson L M and Gouker M A 2016 *IEEE Trans. Appl. Supercond.* **26** 1–10
[12] He Y, Ayala C L, Takeuchi N, Yamae T, Hironaka Y, Sahu A, Gupta V, Talalaevskii A, Gupta D and Yoshikawa N 2020 *Supercond. Sci. Technol.* **33** 035010
[13] Mead C and Conway L 1979 *Introduction to VLSI Systems* (Boston, MA: Addison-Wesley Longman Publishing Co., Inc.)
[14] Neil H E and Weste D M H 2010 *CMOS VLSI Design: A Circuits and Systems Perspective* (Reading, MA: Addison-Wesley)
[15] Yorozu S, Kameda Y, Terai H, Fujimaki A, Yamada T and Tahara S 2002 *Physica* C **378–381** 1471–4
[16] Hidaka M, Nagasawa S, Hinode K and Satoh T 2013 *IEEE Trans. Appl. Supercond.* **23** 1100906
[17] Nagasawa S, Hinode K, Satoh T, Hidaka M, Akaike H, Fujimaki A, Yoshikawa N, Takagi K and Takagi N 2014 *IEICE Trans. Electron.* **E97.C** 132–40
[18] Takeuchi N, Yamanashi Y and Yoshikawa N 2015 *J. Appl. Phys.* **117** 173912
[19] Takeuchi N, Nagasawa S, China F, Ando T, Hidaka M, Yamanashi Y and Yoshikawa N 2017 *Supercond. Sci. Technol.* **30** 035002
[20] Takeuchi N, Ehara K, Inoue K, Yamanashi Y and Yoshikawa N 2013 *IEEE Trans. Appl. Supercond.* **23** 1700304
[21] Fang E S and Van Duzer T 1989 A josephson integrated circuit simulator (JSIM) for superconductive electronics application *Extended Abstracts of Int. Superconductivity Electronics Conf.* pp 407–10
[22] Fourie C J, Wetzstein O, Ortlepp T and Kunert J 2011 *Supercond. Sci. Technol.* **24** 125015
[23] Murai Y, Ayala C L, Takeuchi N, Yamanashi Y and Yoshikawa N 2017 *IEEE Trans. Appl. Supercond.* **27** 1–9
[24] Tanaka T, Ayala C L, Chen O, Saito R and Yoshikawa N 2019 *IEEE Trans. Appl. Supercond.* **29** 1–6
[25] Xu Q, Ayala C L, Takeuchi N, Yamanashi Y and Yoshikawa N 2016 *IEEE Trans. on Appl. Supercond.* **26** 1–5
[26] Wolf C Yosys Open SYnthesis Suite (http://clifford.at/yosys/)
[27] Xu Q, Ayala C L, Takeuchi N, Murai Y, Yamanashi Y and Yoshikawa N 2017 *IEEE Trans. Appl. Supercond.* **27** 1–5
[28] Xun L, Papaefthymiou M C and Friedman E G 2002 *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **21** 184–203
[29] Goldberg D E and Holland J H 1988 *Mach. Learn.* **3** 95–9
[30] Yoshimura T and Kuh E S 1982 *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **1** 25–35
[31] Ayala C L, Takeuchi N, Xu Q, Narama T, Yamanashi Y, Ortlepp T and Yoshikawa N 2015 *IEICE Tech. Rep.* **115** 7–12
[32] JSIM and JSIM_n (http://0.sun.ac.za/ix/?q=tools_jsim)
[33] Delport J A, Jackman K, Roux l P and Fourie C J 2019 *IEEE Trans. Appl. Supercond.* **29** 1–5
[34] Ayala C L, Takeuchi N, Yamanashi Y, Ortlepp T and Yoshikawa N 2017 *IEEE Trans. Appl. Supercond.* **27** 1–7
[35] Tsuji N, Ayala C L, Takeuchi N, Ortlepp T, Yamanashi Y and Yoshikawa N 2017 *IEEE Trans. Appl. Supercond.* **27** 1–4
[36] Chen O, Saito R, Tanaka T, Ayala C L, Takeuchi N and Yoshikawa N 2019 *IEEE Trans. Appl. Supercond.* **29** 1–5
[37] Kogge P M and Stone H S 1973 *IEEE Trans. Comput.* **C-22** 786–93