Time Multiplexing via Circuit Folding

Po-Chun Chien[†] and Jie-Hong R. Jiang[†‡]
[†]*Graduate Institute of Electronics Engineering,* [‡]*Department of Electrical Engineering*
National Taiwan University
Taipei, Taiwan
{r07943091, jhjiang}@ntu.edu.tw

***Abstract*—Time multiplexing is an important technique to overcome the bandwidth bottleneck of limited input-output pins in FPGAs. Most prior work tackles the problem from a physical design standpoint to minimize the number of cut nets or Time Division Multiplexing (TDM) ratio through circuit partitioning or routing. In this work, we formulate a new orthogonal approach at the logic level to achieve time multiplexing through structural and functional circuit folding. The new formulation provides a smooth trade-off between bandwidth and throughput. Experiments show the effectiveness of the structural method and improved optimality of the functional method on look-up-table and flip-flop usage.**

***Index Terms*—circuit folding, pin-count reduction, time-frame folding, time multiplexing**

I. Introduction

Multi-FPGA boards are commonly used for system emulation [1] and prototyping. As the logic capacity, i.e., the number of look-up-tables (LUTs), of an FPGA increases with new technology nodes, the growth in I/O pin count remains relatively slow. This unbalance growth rate makes the number of available I/O pins for each FPGA relatively small compared to the number of required inter-chip signals, which leads to a significant underutilization of logic resources [2].

To overcome the bottleneck of limited inter-chip I/O bandwidth, time division multiplexing (TDM) [3] was proposed, where physical pins and wires are multiplexed among multiple signals, increasing the effective number of available logic pins. Under this scheme, the system requires two separate clocks, a system clock, on which the FPGAs operate, and a faster I/O clock, on which the inter-chip signals are propagated. The ratio of the system clock to the I/O clock is called the TDM ratio r. Essentially, r times the I/O bandwidth of signals can be transmitted during a system clock. Figure 1 illustrates an example of I/O transmission between two FPGAs with TDM ratio 4. The TDM technique dramatically increases the capability of multi-FPGA systems. However, it reduces the system throughput as the system clock is operating at a lower frequency. Most of the related work, e.g., [4], viewed this problem from a physical design standpoint and tried to minimize the number of cut nets, which corresponds to the number of inter-chip signals, passing through each FPGA. Another line of research, e.g., [5], [6], considers scheduling and temporal partitioning for time-multiplexed FPGAs. They partitioned a combinational circuit into several pipeline stages for time multiplexing. However, the approach cannot control the pin-count reduction as it is determined by the circuit structure. In [7], [8], the problem of pin assignment during pin multiplexing, which is the mapping between logic inputs and outputs to the physical pins, was investigated. The pin-count reduction issue was not addressed.

In this work, we formulate a new orthogonal approach to achieve time multiplexing at the logic level. The proposed structural and functional methods can directly reduce the number of input pins of a logic circuit as desired by folding the computation of the circuit. The resulting circuit will satisfy the input pin count constraint at the cost of additional flip-flops storing required information and additional control circuitry for intended computation. This new approach does not require dynamic reconfiguration of the FPGA, unlike [5], [6]. Neither does it require an additional I/O clock as TDM, the I/O transmission can work in synchronization with the system clock.

This work was supported in part by the Ministry of Science and Technology of Taiwan under grants 106-2923-E-002-002-MY3, 108-2221-E-002-144-MY3, 108-2218-E-002-073.

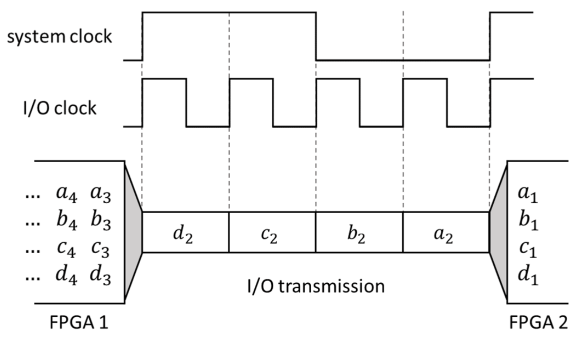


Fig. 1: TDM I/O transmission with ratio 4.

In the literature, the term "folding" is used elsewhere. In [9], a folding transformation technique was proposed to schedule and bind a data-flow graph onto a hardware architecture, where folding refers to the process of executing multiple algorithmic operations in a hardware unit. In [10], a folding technique was proposed to identify structurally identical subcircuits to share gate implementation using dual-edge-triggered flip-flops for time multiplexing. Their primary objective was to minimize the circuit area after technology mapping, while ours is to reduce the input pin count.

The main results of this work include: 1) formulating a new time multiplexing scheme, 2) proposing structural and functional circuit folding methods, that convert a combinational circuit into a sequential one with equivalent input-output behavior modulo time-frame expansion, 3) conducting experimental evaluation on the proposed methods, and demonstrating their effectiveness in input pin reduction to alleviate the I/O pin bottleneck of FPGAs.

The rest of this paper is organized as follows. After Section II introduces the essential preliminaries, the problem of time multiplexing is then formulated in Section III. Our algorithmic solutions are presented in Sections IV and V. Section VI evaluates the experimental results, and finally Section VII concludes this paper.

II. Preliminaries

For our notation, sets are denoted by capital letters, e.g. S; the elements in a set are denoted by minuscule letters, e.g. $x \in S$; and the cardinality of a set S is denoted as $|S|$.

A combinational circuit $\mathcal{C}_C$ is a directed acyclic graph with vertices V and edges $E \subseteq V \times V$. Two subsets $I, O \subset V$ are identified as the *primary inputs* (PIs) and *outputs* (POs), respectively. For $(u, v) \in E$, we call u is a *fanin* of v, and v is a *fanout* of u. Each vertex $v \in V$ is associated with a Boolean variable and with a Boolean function expressed in terms of its fanin variables. The *support set* of v is the set of PIs that can reach v through a path consisting of edges in E.

A sequential circuit $\mathcal{C}_S$ is a combinational circuit augmented with state-holding elements (flip-flops), each of which takes an output of the combinational circuit as its input and produces an output to an input of the combinational circuit. The behavior of $\mathcal{C}_S$ can be described by a finite state machine (FSM) $(I, O, S, s_1, \Delta, \Omega)$, where I is the set of input symbols, O is the set of output symbols, $S \neq \emptyset$ is a finite set of states, $s_1 \in S$ is the initial state, $\Delta : S \times I \to S$ is the state transition function, $\Omega : S \times I \to O$ is the output function. An FSM is completely specified, if for every state $s \in S$ under every input, s's output and next state are defined; otherwise, it is incompletely specified.

In *time-frame expansion*, a sequential circuit is duplicated and cascaded, where the inputs and outputs of the flip-flops from the consecutive time-frames are connected together, resulting in an unrolled, iterative combinational circuit. In the sequel, the timestamp of the input/output of an iterative circuit is denoted by superscript letters. E.g., for a sequential circuit with input x and output y, the timestamped input and output of the iterative circuit is denoted as $x^1, \ldots, x^t$ and $y^1, \ldots, y^t$, respectively. On the other hand, *time-frame folding* (TFF) [11] aims at transforming a k-iterative combinational circuit into a sequential circuit. The resulting sequential circuit has the input-output behavior consistent with the original combinational circuit within the first bounded k time-frames. Essentially, time-frame folding is a reverse operation of time-frame expansion. Figure 2 illustrates the relation of time-frame expansion and folding operations, where the circuit is expanded and folded by 2 time-frames.

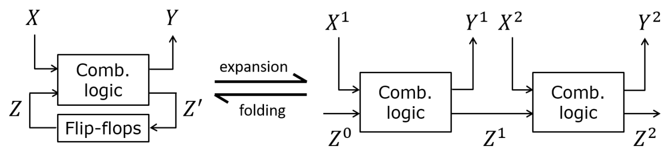


Fig. 2: Time-frame expansion vs. folding.

III. Problem Formulation

The problem of *circuit folding for time multiplexing* can be stated as follows.

Problem Statement (Circuit Folding for Time Multiplexing)**.** *Given a folding number T and a combinational circuit $\mathcal{C}_C$ with inputs $U = \{u_1, \ldots, u_n\}$ and outputs $W = \{w_1, \ldots, w_{n'}\}$, we are asked to fold $\mathcal{C}_C$ into a sequential circuit $\mathcal{C}_S$ with inputs $X = \{x_1, \ldots, x_m\}$ and outputs $Y = \{y_1, \ldots, y_{m'}\}$, where $m = \lceil n/T \rceil$ and $m' \leq n'$, such that unfolding (expanding) $\mathcal{C}_S$ by T time-frames yields a combinational circuit $\mathcal{C}'_C$ with inputs $(X^1, \ldots, X^T)$ and outputs $(Y^1, \ldots, Y^T)$ that is functionally equivalent to $\mathcal{C}_C$ under some proper association of their inputs and outputs. That is, $\mathcal{C}_S$ achieves time multiplexing by taking T clock cycles, each taking m partial inputs, to execute the computation of $\mathcal{C}_C$.*

In the sequel, we assume without loss of generality that n is divisible by T as one can always add dummy inputs (with no fanouts) to $\mathcal{C}_C$ to satisfy the divisibility.

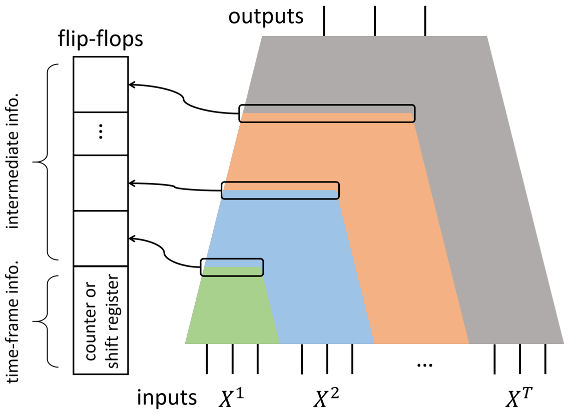


Fig. 3: Illustration of structural circuit folding.

We present two methods, structural circuit folding and functional circuit folding, for time multiplexing as follows.

IV. Structural Circuit Folding

To find the sequential circuit $\mathcal{C}_S$ of the circuit folding problem of Section III, let the inputs U of the given combinational circuit $\mathcal{C}_C$ be divided into T groups: $X^1 = \{u_1, \ldots, u_m\}$, $\ldots$, $X^T = \{u_{(T-1)\times m}, \ldots, u_n\}$. We then traverse the logic gates of $\mathcal{C}_C$ in a topological order by T iterations. At iteration t, for $t = 1, \ldots, T$, a topological traversal is initiated at the inputs X^t. A gate will be visited if and only if all of its fanins have been visited. On a visit to a gate in $\mathcal{C}_C$, a corresponding gate will be duplicated in $\mathcal{C}_S$. If a primary output of $\mathcal{C}_C$ is visited, then it will be scheduled to output Y^t at time-frame t in $\mathcal{C}_S$. At the end of each iteration, the gates in the frontier of the traversal is collected, each of which has a newly introduced flip-flop in $\mathcal{C}_S$ to store its value. After T iterations, all the gates in $\mathcal{C}_C$ have been visited. Moreover, additional flip-flops are introduced to track the time-frame information, either with a $\lceil \log_2(T) \rceil$-bit counter using binary encoding or a T-bit shift register using one-hot encoding. The corresponding control logic is then added to select the correct output at each time-frame. Finally, we can obtain a sequential circuit $\mathcal{C}_S$ with inputs $X = \{x_1, \ldots, x_m\}$. The number of outputs of $\mathcal{C}_S$ is determined by the maximum number of outputs being scheduled in a time-frame among the T time-frames. Figure 3 illustrates the iterative-layering procedure of structural circuit folding. Different colors in the figure indicate the gate traversal at different time-frames. The frontier of each traversal is circled by a rounded rectangle and their signals are stored in the flip-flops, serving as the pseudo inputs to the circuit traversed in the next iteration.

Example 1. *To illustrate the procedure of structural circuit folding, we take the 3-bit adder in Figure 4 as an example. The adder has inputs $U = A \cup B$ and outputs $W = \{s_0, s_1, s_2, c_{out}\}$, where $A = \{a_0, a_1, a_2\}$ and $B = \{b_0, b_1, b_2\}$ are the 2 input 3-bit numbers, with a_i and b_i being the i^{th} bits of A and B, respectively, s_i the i^{th} summation bit, and c_{out} the carry-out bit. The inputs are grouped as $X^1 = \{a_0, b_0\}$, $\ldots$, $X^3 = \{a_2, b_2\}$. The gates in Figure 4 marked in green, blue, and orange correspond to the gates visited at the first, second, and third iteration, respectively. A total of 5 flip-flops are introduced, 2 for storing the intermediate information of g_2 and g_8, which are essentially the carry bits of the first two iterations, and 3 for storing the time-frame information as a shift register. The number of outputs of $\mathcal{C}_S$ is determined by $|Y^3| = 2$. The outputs are*

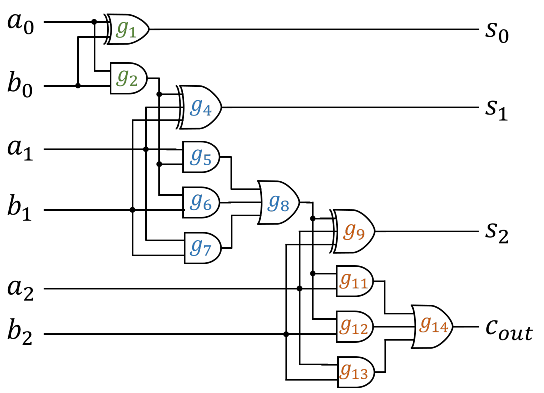


Fig. 4: Example of 3-bit adder (`adder3`) circuit under folding.

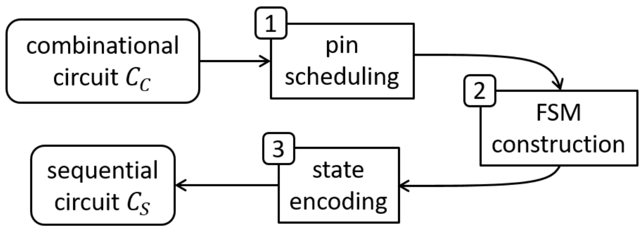


Fig. 5: Computation flow of functional circuit folding.

scheduled as follows: $Y^1 = \{s_0, null\}$, $Y^2 = \{s_1, null\}$, *and* $Y^3 = \{s_2, c_{out}\}$, *where null denotes a dummy output. With the control logic being added for selecting the correct output at each time-frame,* $\mathcal{C}_S$ *can be synthesized to a circuit with 2 inputs, 2 outputs, 5 flip-flops, and 23 AIG nodes (or 8 6-input LUTs) [12].*

Although the structural circuit folding method is efficient and scalable to large circuits, the constructed sequential circuit $\mathcal{C}_S$ can be sub-optimal. Taking `adder3` of Figure 4 for example, we know that ultimately $\mathcal{C}_S$ can be implemented with an 1-bit carry-save adder, consisting of only 1 input, 2 outputs, 1 flip-flop, and 7 AIG nodes (for a full adder implementation). It motivates the functional circuit folding approach as we present next.

V. Functional Circuit Folding

We exploit the recent time-frame folding (TFF) technique [11] to the time multiplexing problem. Note that the original TFF cannot be applied directly because it assumes the given combinational circuit under folding is in an iterative form. However, time multiplexing must work for general combinational circuits not necessarily iterative ones. Below we detail the functional circuit folding method.

As shown in Figure 5, the functional circuit folding algorithm consists of three main computation steps: 1) pin scheduling, 2) FSM construction via time-frame folding, and 3) state encoding, to be presented in the following subsections.

A. Pin Scheduling and Iterative Circuit Conversion

Given a folding number T and a combinational circuit $\mathcal{C}_C$ with inputs $U = \{u_1, \dots, u_n\}$ and outputs $W = \{w_1, \dots, w_{n'}\}$, the pin scheduling procedure permutes the inputs and outputs (and possibly adds dummy inputs and outputs) to convert $\mathcal{C}_C$ into a *virtual* T-iterative combinational circuit $\mathcal{C}'_C$ with inputs $X^1, \dots, X^T$ for $X^t = \{x_1^t, \dots, x_m^t\}$ and outputs $Y^1, \dots, Y^T$ for $Y^t = \{y_1^t, \dots, y_{m'}^t\}$, where $m = \lceil n/T \rceil$ and $m' \leq n'$. The circuit after scheduling must satisfy the property that every primary output $w_i \in W$ is scheduled at some iteration t while its input supports are scheduled in iterations $t' \leq t$.

Algorithm 1 shows a heuristic scheduling procedure of outputs $W = \{w_1, \dots, w_{n'}\}$ with respect to a folding number T. In line 1, the number m of inputs in one circuit iteration is calculated. In line 2, the set of outputs W is sorted according to their support sizes in an ascending order. In line 3, the sets $U_{sup}, Y^1, \dots, Y^T$ are initialized to be empty. In lines 4-8, the loop goes over each output w_i to determine its iteration. In line 5, the support set of w_i is added to U_{sup}. In line 6, the earliest available iteration t for w_i is calculated. In line 7, w_i is assigned to Y^t. Finally, the output schedule is returned in line 9. Note that to make the number of outputs scheduled at each iteration identical, null (dummy) outputs are inserted to $Y^1, \dots, Y^T$.

```
Algorithm 1 OutputSchedule
Input: $\mathcal{C}_C$ with inputs $U = \{u_1, \dots, u_n\}$ and outputs $W = \{w_1, \dots, w_{n'}\}$, folding number $T$
Output: output schedule $Y^1, \dots, Y^T$
1: $m := n/T$;
2: $SortAscend(W)$;
3: $U_{sup}, Y^1, \dots, Y^T := \emptyset$;
4: foreach $w_i$ in $W$ do
5:     $U_{sup} := U_{sup} \cup Support(w_i)$;
6:     $t := \lceil |U_{sup}|/m \rceil$;
7:     $Y^t := Y^t \cup \{w_i\}$;
8: end for
9: return $(Y^1, \dots, Y^T)$;
```

With the outputs being scheduled, the inputs $W = \{w_1, \dots, w_{n'}\}$ can be scheduled accordingly as outlined in Algorithm 2. Let X_{que} be a queue to store the ordered inputs. In line 1, X_{que} is initialized as an empty queue. In lines 2-6, the loop iterates through each scheduled outputs Y^t to fill in the queue. In line 3, the supports X_{sup} of Y^t that have not yet been scheduled during the previous iterations are collected in queue X_{sup}. In line 4, an optional optimization step is performed to reorder X_{sup}. Since the FSM construction algorithm in the later step relies on BDD-based operations, a smaller BDD size of $\mathcal{C}'_C$ would help to reduce the execution time. Therefore, BDD variable reordering with symmetric sifting [13] technique is applied to X_{sup} to minimize the BDD size of outputs Y^t of $\mathcal{C}_C$. In line 5, X_{sup} is pushed into the queue X_{que}. In line 7, X_{que} is evenly divided into T groups $X^1, \dots, X^T$, which are finally returned in line 8.

```
Algorithm 2 InputSchedule
Input: $\mathcal{C}_C$ with inputs $U = \{u_1, \dots, u_n\}$ and outputs $W = \{w_1, \dots, w_{n'}\}$, folding number $T$, and output schedule $Y^1, \dots, Y^T$
Output: input schedule $X^1, \dots, X^T$
1: $X_{que} := \emptyset$;
2: for $t = 1, \dots, T$ do
3:     $X_{sup} := Support(Y^t) \setminus X_{que}$;
4:     $X_{reord} := BddSymSift(\mathcal{C}_C, t, X_{sup})$;
5:     $X_{que} := Append(X_{que}, X_{reord})$;
6: end for
7: $(X^1, \dots, X^T) := Split(X_{que}, T)$;
8: return $(X^1, \dots, X^T)$;
```

Example 2. *Consider the* `adder3` *example. After pin scheduling, we have outputs $Y^1 = \{s_0, null\}$, $Y^2 = \{s_1, null\}$, $Y^3 = \{s_2, c_{out}\}$, and inputs $X^1 = \{a_0, b_0\}$, $X^2 = \{a_1, b_1\}$, $X^3 = \{a_2, b_2\}$. Note that the null (dummy) outputs are inserted to make the number of outputs scheduled at each iteration identical.*

B. FSM Construction via Time-Frame Folding

Given an T-iterative combinational circuit $\mathcal{C}'_C$ with inputs $X^1, \ldots, X^T$ for $X^t = \{x^t_1, \ldots, x^t_m\}$ and outputs $Y^1, \ldots, Y^T$ for $Y^t = \{y^t_1, \ldots, y^t_{m'}\}$, the time-frame folding (TFF) algorithm [11] can be applied to construct an FSM with inputs $X = \{x_1, \ldots, x_m\}$ and outputs $Y = \{y_1, \ldots, y_{m'}\}$, which has the same input-output behavior as $\mathcal{C}'_C$ within the T bounded time-frames. The notion of states at time-frame t is induced by the output functions $Y^{t+1}, \ldots, Y^T$, each of which induces an equivalence relation and effectively forms a partition on the Boolean space of $X^1 \cup \ldots \cup X^t$. With the technique of hyper-function encoding [14] and BDD-based functional decomposition, we are able to determine the refinement of all such partitions. Each cell in the refined partition corresponds to a state at time-frame t. The transitions between states can then be constructed according to the identified state information.

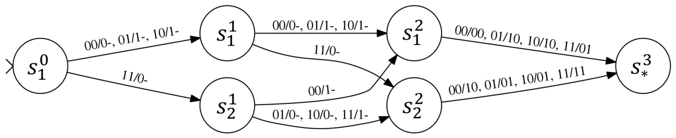
Some minor modifications to the TFF algorithm are needed as we discuss below. In [11], the iterative circuit being folded or transformed is fully-specified, that is, there are no null output functions. Because null functions do not provide any additional information for state partitioning, they can simply be discarded from $Y^{t+1}, \ldots, Y^T$ or be treated as constant functions during the encoding stage of state identification. Similarly, when determining the output response of a state at time-frame t, if there is a null output scheduled at that time-frame, then its corresponding slot should remain unspecified. In the sequel, $S^t = \{s^t_1, \ldots, s^t_k\}$ is used to denote the set of states identified at time-frame t.

Even though by BDD-based functional decomposition we can guarantee that $|S^t|$ is minimum, the FSM constructed may not necessarily be state minimized, since the equivalent states in different time-frames are not yet considered. In the derived FSM, there is a unique initial state s^0_1 and don't-care destination state s^T_* inserted by the time-frame folding algorithm, along with some null (dummy) outputs at several states. As the FSM is incompletely specified, the flexibility leaves room for state minimization. In our implementation, we adopt the SAT-based exact minimization algorithm `MeMin` [15] for FSM simplification.

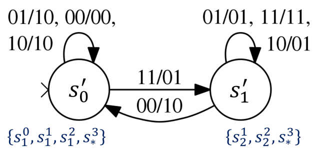
Example 3. *The state diagram in Figure 6a, where the mark ">" indicates the initial state, is obtained by folding the* `adder3` *circuit by 3 time-frames with the functional circuit folding algorithm. It can be further minimized to that in Figure 6b. The number of states reduces from 6 (including the don't-care state s^3_*) to 2. In Figure 6b, each state is annotated with its compatible states in Figure 6a. We can observe that the minimized FSM is essentially a carry-save adder, where s'_0 and s'_1 corresponds to the state with carry-bit of value 0 and 1, respectively.*

C. State Encoding

To convert an FSM into a sequential circuit $\mathcal{C}_S$, a state-encoding step has to be performed. Let S be the state set of the FSM. In our implementation, we apply two different encoding methods: 1) natural binary encoding, which uses $\lceil \log_2 |S| \rceil$ bits, and 2) one-hot encoding, which uses $|S|$ bits, each of which represents a state in S.



(a) FSM before state minimization.



(b) FSM after state minimization.

Fig. 6: FSM by functional circuit folding of `adder3`.

TABLE I: Benchmark statistics.

circuit	#PI	#PO	#gate	#LUT
64-adder	128	65	507	96
128-adder	256	129	844	244
apex2	38	3	1448	581
arbiter*	256	1	361	102
b14_C	276	299	3890	1152
b15_C	484	519	6801	1966
b17_C*	380	3	1634	381
b20_C	521	512	8173	2221
b21_C	521	512	8250	2311
b22_C	766	757	12355	3375
C7552	207	108	1485	340
des	256	245	3087	717
e64	65	65	244	114
g216	216	216	3982	648
g625	625	625	10625	2498
g1296	1296	1296	31447	5184
hyp	256	128	213158	45142
i2	201	1	208	63
i3	132	6	126	38
i4	192	6	186	42
i6	138	67	444	67
i7	199	67	558	67
i10	257	224	1586	507
max	512	130	2776	812
memctrl	1204	1231	15908	5207
toolarge	38	3	2642	1111
voter	1001	1	12400	1667

VI. Experimental Results

The proposed structural and functional methods were implemented in C++ language within the `ABC` system [12], which utilized `CUDD` [16] as the underlying BDD package. Moreover, an open source package `MeMin` [15] was used for state minimization. The two methods were evaluated on 27 combinational circuits selected or converted from several sets of benchmarks, including ITC, MCNC(LGSynth), LEKO/LEKU, Adder, and EPFL benchmarks. The information of these circuits are shown in Table I, where columns 2-5 list the numbers of primary inputs, primary outputs, AIG nodes, and 6-input LUTs, respectively, of the circuits after optimization. The circuits marked with "*" are simplified from the original circuits by extracting some primary outputs and keeping only the structural input support of those outputs. All the experiments were conducted on a Linux server with Intel(R) Core(TM) i7-8700 3.20GHz CPU and 32GB RAM.

We first evaluate the effectiveness of the structural method for time multiplexing by imposing the I/O pin count limitation to 200, according to some commercial FPGA specification. Table II shows the results on folding 17 benchmark circuits with more than 200 pins, where column 2 lists the number of time-frames each circuit

TABLE II: Results of structural circuit folding.

circuit	#frm	#in	#out	#FF	#gate	#LUT	overhead
128-adder	2	128	65	2	641	195	-20.08%
b14_C	2	138	262	453	5213	1540	33.68%
b15_C	3	162	274	561	8928	2473	25.79%
b20_C	3	174	424	734	10654	2964	33.45%
b21_C	3	174	424	726	10497	2952	27.74%
b22_C	4	192	661	1266	16536	4587	35.91%
C7552	2	104	96	117	1828	447	31.47%
des	2	128	245	185	3617	868	21.06%
g1296	7	186	1296	4289	36873	9688	86.88%
g216	2	108	216	167	3483	820	26.54%
g625	4	157	625	1607	15043	4330	73.34%
hyp	2	128	128	256	145628	29805	-33.98%
i2	2	101	1	10	161	47	-22.95%
i10	2	129	180	224	2365	740	45.96%
max	3	171	130	395	3912	1003	23.52%
memctrl	7	172	772	3294	27465	8317	59.73%
voter	6	167	1	166	11446	1921	15.24%

should be folded, and columns 3-8 list the information of folded sequential circuit, including the number of inputs, outputs, flip-flops, AIG nodes, 6-input LUTs, and the LUT overhead incurred comparing to the original combinational circuit, respectively. The experimental results indicate the ability of the structural method on meeting the I/O pin constraint[1] with an average of 34.84% LUT overhead, despite the fact that there are cases, `128-adder`, `hyp`, `i2`, with LUT savings. Notice that the LUT increase could not be a serious problem as the LUT resources are not as critical as the I/O pin bottleneck in FPGAs. As all the experiments were done in less than a second, the results demonstrate the scalability of the structural method.

A simple alternative to fold a circuit by T time-frames can be done by temporarily storing inputs of the first $T-1$ time-frames into flip-flops and defer computing all outputs at the last time-frame. When applied to the same 17 benchmark circuits in Table II, the additional control circuitry to store the input signals of this simple method incurred an average 46.59% LUT overhead, which is 11.75% higher than the proposed structural method. The number of flip-flops required for this simple method is smaller than the structural method in 14 out of the 17 cases, as it is linearly proportionate to the number of primary inputs of the original combinational circuit. However, for cases such as `voter`, the number of flip-flops under the simple folding is 841 whereas that under the structural folding is 166, which is significantly smaller than the former. The number of output pins after this simple folding remains the same as the number of primary outputs of the original combinational circuit, since all the outputs are scheduled to compute at the last time-frame. In contrast, the structural method can achieve output pin reduction on 9 out of the 17 cases. In comparison, the structural method is better than the simple method when taking the number of LUTs, flip-flops and output pins into consideration, as the resources of output pins and LUTs are more critical than those of flip-flops in FPGAs.

To study the potential of latency reduction by circuit folding, we perform case analysis on circuit `i10`, with 257 PIs and 224 POs. The analysis is based on the following assumptions: 1) Assume the maximum I/O transmission rate is 200 bits per I/O clock cycle. 2) Assume TDM ratio $r = 1$, i.e., the system clock cycle equals the I/O clock cycle, for the circuit without folding and the circuit with folding. 3) Assume the combinational logic of both circuits without and with folding can be computed in one I/O clock cycle. With circuit folding, `i10` would be folded by two time-frames into a sequential circuit with 129 inputs and 180 outputs as shown in Table II, with 44 outputs scheduled in the first time-frame and 180 scheduled at the second time-frame. The overall execution requires three system (also I/O) clock cycles: the first cycle transmits 129 inputs, second cycle 129 inputs and 44 outputs, and third cycle 180 outputs. In contrast, without circuit folding, the execution of `i10` requires a total of four I/O clock cycles: the first cycle transmits 200 inputs, second cycle 57 inputs, third cycle 200 outputs, and fourth cycle 24 outputs. Effectively, circuit folding may achieve 25% I/O clock cycle reduction. In fact, TDM aims at increasing the effective I/O pins of FPGA by slowing down the system clock to increase I/O transmissions during a system clock period, while our circuit folding can directly decrease the required number of pins of a logic circuit. The TDM and circuit folding methods are orthogonal, and can be combined to alleviate the FPGA I/O bottleneck issue.

To compare the performance of the structural and functional methods, we conducted experiments on 11 benchmarks, each being folded by 4, 8 and 16 time-frames. A timeout limit of 300 seconds was imposed on pin scheduling and functional circuit folding combined, and the same limit was imposed on `MeMin` for state minimization. Table III shows the 33 results, where columns 2-3 list the folding number and the number of inputs of the folded sequential circuits, respectively, and columns 4-16 list the folded circuit information of the two methods, including the number of outputs, AIG nodes, LUTs, and flip-flops. The results of the functional method are annotated in column 15 with the applied configurations: whether to enable input reordering (r/nr), whether to minimize FSM states (m/nm), and the two encoding options (nat/1hot). Column 9 lists the numbers of states before and after minimization (separated by "/"), columns 13-14 list the reduction on the numbers of LUTs and flip-flops, respectively, of the functional method over the structural method, and column 16 lists the CPU time in seconds of the functional method on each benchmark. An entry "-" in the table indicates that the value cannot be obtained within the timeout limit. The structural method took less than a second for all the experiments, while the functional method generated results for 29 of the 33 instances within the timeout limit. On the other hand, the functional method achieved an average of 44.93% and 64.93% reductions on LUT and flip-flop usage, respectively, over the structural method in the 29 cases.

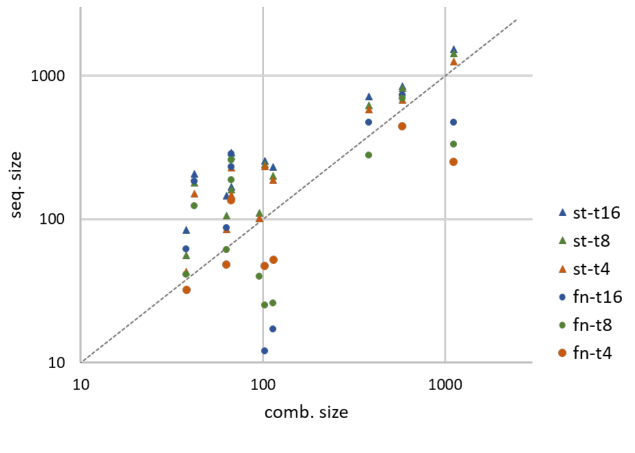


Fig. 7: Circuit size comparison.

In addition, we compared the sizes of the original combinational circuits to their folded sequential circuits under the two methods in terms of the number of LUTs. The results are plotted in Figure 7, where the triangular and circular points correspond to the results

[1]Note that the number of output pins can be larger than 200. In that case, multiple clock cycles can be taken to produce the outputs.

TABLE III: Comparison between structural and functional methods.

name	#frm	#in	structural method				functional method								
			#out	#gate	#LUT	#FF	#out	#state	#gate	#LUT	#FF	#LUT red.	#FF red.	config	runtime
64-adder	16	8	5	279	101	31	5	32/2	32	7	1	93.07%	96.77%	nr/m/nat	0.28
	8	16	9	281	110	15	9	16/-	150	40	4	63.64%	73.33%	nr/nm/nat	9.29
	4	32	17	296	101	7	-	-	-	-	-	-	-	-	-
apex2	16	3	3	2592	843	498	1	474/-	1764	734	474	12.93%	4.82%	r/nm/1hot	0.38
	8	5	3	2446	819	433	2	327/-	1767	696	327	15.02%	24.48%	r/nm/1hot	0.13
	4	10	3	1944	676	238	3	127/-	1177	444	127	34.32%	46.64%	r/nm/1hot	0.12
arbiter*	16	16	1	925	255	227	1	47/4	53	12	2	95.29%	99.12%	r/m/nat	0.57
	8	32	1	894	241	203	1	23/4	104	25	2	89.63%	99.01%	r/m/nat	0.53
	4	64	1	832	233	176	1	11/4	165	47	2	79.83%	98.86%	r/m/nat	0.51
b17_C*	16	24	2	2491	715	424	1	233/-	1149	472	232	33.99%	45.28%	r/nm/1hot	42.89
	8	48	2	2278	621	328	1	86/-	746	279	86	55.07%	73.78%	r/nm/1hot	85.98
	4	95	2	2028	583	235	-	-	-	-	-	-	-	-	-
e64	16	5	53	592	231	148	5	29/14	74	17	4	92.64%	97.30%	r/m/nat	0.12
	8	9	44	534	200	113	9	16/9	108	26	4	87.00%	96.46%	r/nm/nat	0.08
	4	17	59	477	188	91	17	8/-	162	52	3	72.34%	96.70%	r/nm/nat	4.68
i2	16	13	1	499	146	126	1	54/-	207	87	6	40.41%	95.24%	r/nm/nat	0.25
	8	26	1	395	106	84	1	25/-	152	61	25	42.45%	70.24%	r/nm/1hot	0.18
	4	51	1	306	85	50	1	14/-	130	48	14	43.53%	72.00%	r/nm/1hot	0.22
i3	16	9	2	239	84	69	1	40/-	145	62	35	26.19%	49.28%	r/nm/1hot	0.09
	8	17	2	170	56	40	1	22/-	103	41	20	26.79%	50.00%	r/nm/1hot	0.12
	4	33	2	136	43	20	2	10/-	89	32	9	25.58%	55.00%	r/nm/1hot	29.05
i4	16	12	4	689	208	181	1	83/-	458	184	82	11.54%	54.70%	r/nm/1hot	3.85
	8	24	4	620	179	150	1	38/-	295	124	37	30.73%	75.33%	r/nm/1hot	5.48
	4	48	4	524	151	114	-	-	-	-	-	-	-	-	-
i6	16	9	9	543	168	90	5	95/-	519	231	95	-37.50%	-5.56%	nr/nm/1hot	0.12
	8	18	18	591	161	79	9	46/-	557	188	46	-16.77%	41.77%	nr/nm/1hot	0.19
	4	35	25	628	147	61	17	22/-	487	135	22	8.16%	63.93%	nr/nm/1hot	109.29
i7	16	13	13	983	291	207	5	163/147	540	286	146	1.72%	29.47%	r/m/1hot	8.56
	8	25	24	1000	276	191	9	79/-	683	259	79	6.16%	58.64%	r/nm/1hot	0.17
	4	50	39	900	228	131	-	-	-	-	-	-	-	-	-
toolarge	16	3	3	4617	1531	867	1	305/-	1057	470	305	69.30%	64.82%	r/nm/1hot	0.14
	8	5	3	4352	1430	760	2	187/-	805	331	187	76.85%	75.39%	r/nm/1hot	0.11
	4	10	3	3621	1255	439	3	92/-	673	250	92	80.08%	79.04%	r/nm/1hot	0.10

of the structural and functional methods, respectively, and the blue, green, and orange points correspond to results folded by 16, 8 and 4 time-frames, respectively. The data points to the right of the gray dotted line are the cases where the folded circuits are smaller than their combinational counterparts. It is interesting to note that 16 of the 29 results obtained by the functional method achieved circuit size reduction, while all of the results from the structural method incurred LUT overhead. The overhead of the structural method is understandable because circuit folding introduces additional control logic and flip-flop boundaries to the original circuit that restricts combinational synthesis.

VII. Conclusions

We have formulated a circuit folding approach to time multiplexing on FPGAs. The structural and functional methods, orthogonal to prior time multiplexing methods, have been proposed and implemented to demonstrate their potentials to alleviate the I/O-pin bottleneck of FPGAs. Experiments suggested the scalability of the former and the optimization power of the latter. It remains future work to combine structural and functional methods to achieve both scalability and optimality.

References

[1] A. Myaing and V. Dinavahi, "FPGA-based real-time emulation of power electronic systems with detailed representation of device characteristics," *IEEE Transactions on Industrial Electronics (TIE)*, vol. 58, pp. 358 – 368, 2011.

[2] W. N. Hung and R. Sun, "Challenges in large FPGA-based logic emulation systems," in *Proceedings of International Symposium on Physical Design (ISPD)*, pp. 26–33, 2018.

[3] J. Babb, R. Tessier, and A. Agarwal, "Virtual wires: Overcoming pin limitations in FPGA-based logic emulators," in *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 142–151, 1993.

[4] S.-C. Chen, R. Sun, and Y.-W. Chang, "Simultaneous partitioning and signals grouping for time-division multiplexing in 2.5D FPGA-based systems," in *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, 2018.

[5] S. Trimberger, "Scheduling designs into a time-multiplexed FPGA," in *Proceedings of International Symposium on Field Programmable Gate Arrays (FPGA)*, pp. 153–160, 1998.

[6] H. Liu and D. F. Wong, "Network flow based circuit partitioning for time-multiplexed FPGAs," in *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, pp. 497–504, 1998.

[7] S. Hauck and G. Borriello, "Pin assignment for multi-FPGA systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 16, no. 9, pp. 956–964, 1997.

[8] S. Liu, F. Lau, and B. Carrion Schafer, "Investigation and optimization of pin multiplexing in high-level synthesis," in *Proceedings of Great Lakes Symposium on VLSI (GLSVLSI)*, pp. 427–430, 2018.

[9] K. Parhi, C.-Y. Wang, and A. Brown, "Synthesis of control circuits in folded pipelined DSP architectures," *IEEE Journal of Solid-State Circuits (JSSC)*, vol. 27, pp. 29 – 43, 02 1992.

[10] I. Han and Y. Shin, "Folded circuit synthesis: Min-area logic synthesis using dual-edge-triggered flip-flops," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 23, pp. 1–21, 08 2018.

[11] P.-C. Chien and J.-H. Jiang, "Time-frame folding: Back to the sequentiality," in *Proceedings of International Conference of Computer-Aided Design (ICCAD)*, 2019.

[12] R. Brayton and A. Mishchenko, "ABC: An Academic Industrial-Strength Verification Tool," in *Proceedings of International Conference on Computer Aided Verification (CAV)*, pp. 24–40, 2010.

[13] S. Panda, F. Somenzi, and B. F. Plessier, "Symmetry detection and dynamic variable ordering of decision diagrams," in *Proceedings of International Conference of Computer-Aided Design (ICCAD)*, pp. 628–631, 1994.

[14] J.-H. R. Jiang, J.-Y. Jou, and J.-D. Huang, "Compatible class encoding in hyper-function decomposition for FPGA synthesis," in *Proceedings of Design Automation Conference (DAC)*, pp. 712–717, 1998.

[15] A. Abel and J. Reineke, "MEMIN: SAT-based exact minimization of incompletely specified Mealy machines," in *Proceedings of International Conference of Computer-Aided Design (ICCAD)*, pp. 94–101, 2015.

[16] F. Somenzi, "CUDD: CU decision diagram package (release 2.4.1)," *University of Colorado at Boulder*, 2005.