Improving Circuit Mapping Performance Through MIG-based Synthesis for Carry Chains

Zhufei Chu[1,2] Xifan Tang[2] Mathias Soeken[2] Ana Petkovska[2] Grace Zgheib[2]
Luca Amarù[3] Yinshui Xia[1] Paolo Ienne[2] Giovanni De Micheli[2] Pierre-Emmanuel Gaillardon[4]
[1]Faculty of Electrical Engineering & Computer Science, Ningbo University, Ningbo, China
[2]École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland
[3]Synopsys Inc., Sunnyvale CA, USA
[4]Electrical and Computer Engineering Department, University of Utah, Salt Lake City, UT, USA
Email: chuzhufei@nbu.edu.cn

ABSTRACT

Hard-wired carry chains in FPGAs are designed to improve efficiency of important arithmetic primitives. Although they are proven to be effective for arithmetic-rich functions, there are very few studies on the optimization opportunities of carry chains for general logic that is poor in arithmetic operations. Recently, *Majority-Inverter Graphs* (MIGs) were proposed for efficient Boolean logic optimization. MIGs open an opportunity for efficient mapping of critical paths onto hard carry chains, as the carry logic of a full adder is naturally a *majority* (MAJ) gate. In this paper, we propose an MIG-based synthesis method to exploit hard adders in FPGAs for the mapping of general logic. The proposed heuristic algorithm selects MAJ nodes to be mapped on the carry chains and the associated LUTs; then, the efficiency of carry chain mapping is examined theoretically for efficient LUT utilization. The experimental results show that, compared to traditional design flow *Verilog-to-Routing* (VTR 7.0), the proposed approach can improve delay by up to 25% with an average of 8%, while the channel width is reduced by up to 20% with an average of 6%.

Keywords

FPGA; majority-inverter graph; carry chain; technology mapping

1. INTRODUCTION

Many arithmetic intensive applications can be efficiently designed by FPGAs and display chains of additions mapped to *Logic Blocks* (LBs). When hard adder chains can be exploited, thus avoiding routing multiplexers, performance can improve up to 15% for arithmetic benchmarks [1]. Typically, the hard adders are technology mapped only when additions/subtractions are specified in *High-level Design Language* (HDL) descriptions. Although they have been shown to be very effective to implement arithmetic functions, these hard blocks have less flexibility compared to soft blocks due to their specificity. To further exploit the efficiency of hard adder chains, we concentrate on the problem of how to improve performance through mapping of general logic when the logic circuit contains a limited number of full adders.

Majority-Inverter Graph (MIG) [2, 3] is a new paradigm for logic optimization that receives renewed research attention due to the emergence of new computing devices as well as the increasing complexity of logic circuits. Regarding the carry chains, it is interesting that the carry function of a full adder is exactly the *majority* (MAJ) operator. This feature brings promising optimization possibilities in FPGA synthesis: a logic network represented as MIG corresponds directly to the physical implementation of the carry chains, i.e., chains of majority operations, even if no specific adder chains are found. This is a complex task when using other logic representations such as *And-Inverter Graphs* (AIGs) [4] or *Binary Decision Diagrams* (BDDs) [5]. In contrast, the MIG-based logic network gives us a great opportunity to improve mapping performance by taking advantage of carry chains.

In this paper, an MIG-based synthesis method for carry chains is proposed with the goal to improve mapping performance of general logic. Instead of exploiting arithmetic functions, we aim at automatic realization of logic that can be tightly implemented by both *Look-Up Tables* (LUTs) and specific carry chains for general logic functions. We present a heuristic algorithm that selects MAJ nodes from a given MIG to be mapped on carry chains and the associated LUTs. The efficiency of carry-chain mapping is examined theoretically for efficient LUT utilization. The experimental results show that the proposed approach improves delay by up to 25% with an average of 8%, while the channel width is reduced by up to 20% with an average of 6%.

This paper is organized as follows: Section 2 reviews the necessary background on FPGA architecture and MIG-based logic optimization. Section 3 presents a brief introduction of the state-of-the-art work. Afterwards, in Section 4, the proposed MIG-based synthesis for carry chains is described in detail. In Section 5, the experimental results are presented and discussed. Finally, Section 6 concludes this paper.

2. BACKGROUND

In this section, we review the necessary background about modern FPGA architectures and MIG logic optimizations.

2.1 Modern FPGA Architecture

Modern FPGA architectures consist of an array of tiles, as shown in Fig. 1, each of which contains a LB, *Connection Boxes* (CBs), *Switch Blocks* (SBs), and routing tracks [6]. CBs are used to connect the heterogenous blocks to routing tracks, while SBs interconnect the routing tracks. The functionalities of FPGAs are realized by the heterogenous logic

GLSVLSI '17, May 10–12, 2017, Banff, AB, Canada.

DOI: http://dx.doi.org/10.1145/3060403.3060432

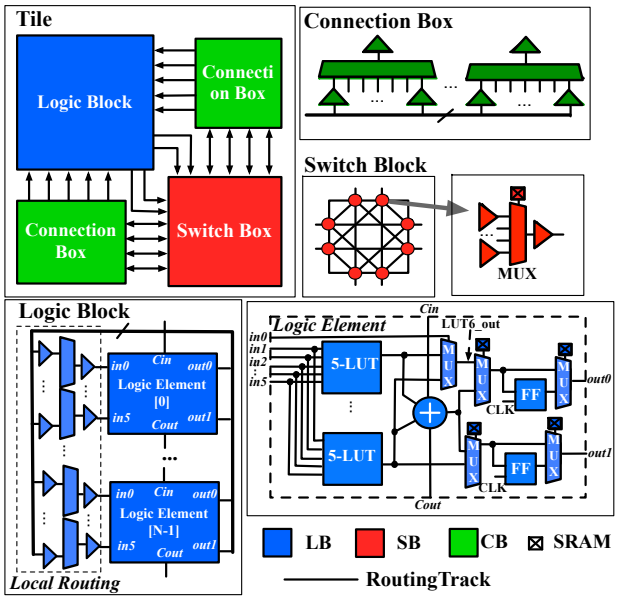

Figure 1: FPGA tile architecture and the considered LB.

resources that consist of LBs, memory banks, and *Digital Signal Processing* (DSP) units. LBs comprise a number of *Logic Elements* (LEs) and a local routing architecture, which provides intra-block interconnections. Basically, there are at least a fracturable LUT [7], a *Flip-flop* (FF), and an output selector in every LE. To boost the functionality of logic resources, FPGA vendors embed additional elements, such as hard carry chains and shift registers, within the LBs. In this paper, we consider the LB resembling commercial FPGA architectures [8, 9]. As depicted in Fig. 1, a LB contains two 5-input LUTs, two FFs and a carry logic. The two LUTs, referred to as carry-chain LUTs in the rest of paper, can be either grouped to implement a 6-input LUT or realize two distinct 5-input LUTs with shared inputs. The output of carry logic can be connected to both fast carry output C_{out} and normal outputs of LEs, *out*0 and *out*1, which allow for early exits of carry signals. When only one or two inputs of carry logic are used (can happen in MIG), the 5-input LUTs can be efficiently utilized to map additional logic.

2.2 Majority-Inverter Graph

MIGs [2, 3] can efficiently represent Boolean functions due to the expressive power of the majority operator. Indeed, by means of fixing one input as constant binary value 0 or 1, the majority operator can be configured to behave as a traditional conjunction (AND) or disjunction (OR) operator, respectively. Consequently, traditional *And-Or-Inverter Graphs* (AOIGs) are a special case of MIGs. MIGs can be easily derived from AOIGs as shown in Fig. 2, in which the lines with bubbles indicate complemented edges require additional inverters.

MIG-based representations are extremely competitive at logic rewriting. The axiomatic system for the MIG Boolean algebra, referred to as Ω, is defined by five primitive transformation rules: commutativity ($\Omega.C$), majority ($\Omega.M$), associativity ($\Omega.A$), distributivity ($\Omega.D$), and inverter propagation ($\Omega.I$). The majority operation of three variables x, y, and z is denoted as $\langle xyz \rangle$ in Ω. A strong property of MIGs and their algebraic framework is reachability. It has been proven that, by using a sequence of transformations drawn from the primitive five rules, it is possible to traverse the entire MIG

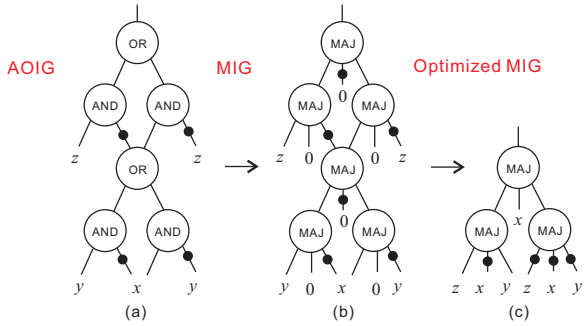

Figure 2: Example of transposing an optimal (a) AOIG representation into (b) an initial MIG representation and (c) an optimal MIG representation after using MIG algebraic transformation.

representation space [2]. This result is of paramount interest to logic synthesis because it guarantees that the best MIG under a given target metric can always be reached. However, deriving a sequence of Ω transformations is an intractable problem. As for traditional logic optimization, heuristic techniques can obtain fast solutions with reasonable quality [10]. From the perspective of logic optimization, it is practical to improve the FPGA performance by reducing the maximum number of logic levels (depth) with affordable area penalty, which leads to a better delay of the physical implementation of the circuit [14]. Thanks to the flexible MIG algebraic framework, it is possible to obtain a better MIG, both in depth and area, as shown in Fig. 2(c).

$$\Omega = \begin{cases} \textbf{Commutativity} - \Omega.C \\ \langle xyz \rangle = \langle yxz \rangle = \langle zyx \rangle \\ \textbf{Majority} - \Omega.M \\ \begin{cases} \langle xyz \rangle = x = y & if\ x = y \\ \langle xyz \rangle = z & if\ x = \bar{y} \end{cases} \\ \textbf{Associativity} - \Omega.A \\ \langle xu\langle yuz \rangle \rangle = \langle zu\langle yux \rangle \rangle \\ \textbf{Distributivity} - \Omega.D \\ \langle xy\langle uvz \rangle \rangle = \langle \langle xyu \rangle \langle xyv \rangle z \rangle \\ \textbf{Inverter Propagation} - \Omega.I \\ \overline{\langle xyz \rangle} = \langle \bar{x}\bar{y}\bar{z} \rangle \end{cases}$$

3. RELATED WORK

Carry chains are widely studied since they were introduced by commercial FPGAs. Most of the existing papers focus on the carry-chain architectures and how to implement best the arithmetic operators. Most of the widely-used tools rely on users explicitly writing additions/subtractions using HDL primitives or using a predefined adder macro to recognize carry-chain logic. Frederick and Somanni [11] proposed a mapping algorithm that exploits carry chains for general circuits. The results proved that carry chains have the potential to improve the delay of general-purpose circuits. However, the technique proposed is only applicable to carry-select chains, which are no longer present in modern FPGAs. To target arbitrary carry-chain structures, Preußer and Spallek [12] proposed an approach to the automatic mapping that uses carry chains for general logic implementation. The algorithm searches the cuts that cover LUTs and the carry-chain node in the same logic block, which impose much tighter constraints than in our approach. The benefit of this approach is that it naturally utilizes the carry-chain LUTs, while the tight constraints may decrease the mapping possibilities. Petkovska et al. [13] propose to map logic

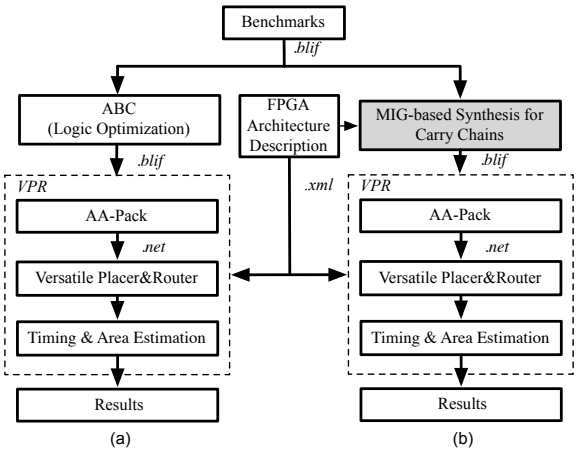

Figure 3: Comparison of (a) the state-of-the-art academic design flow and (b) the proposed design flow.

on the carry chains for circuits expressed using a gate-level descriptions. Moreover, the carry-chain LUTs are specified using a pre-mapping selection algorithm. The results show that they can identify more full adders than with the existing design flow, and that the area is improved by 6% on average, with no delay penalty. However, their method is effective only for circuits rich with adders.

In terms of MIG-based FPGA synthesis, Amarù et al. [14] proposed MIG optimization techniques targeting high-speed FPGA implementations, which aggressively reduce the logic depth and enable denser LUT mapping on FPGAs. They also gave insight on how to exploit the FPGA carry-chain logic during the logic optimization. However, their method targetted LUTs operating in normal mode instead of arithmetic mode, thus it misses out on some opportunities to improve the performance using carry chains.

4. MIG-BASED SYNTHESIS FOR CARRY CHAINS

In this section, we describe the proposed MIG-based synthesis for carry chains. First, we introduce proposed flow, before describing the details of the proposed algorithm.

4.1 Proposed Design Flow

We use the state-of-the-art FPGA design flow VTR 7.0 [15], shown in Fig. 3(a), as our baseline flow. The `BLIF` format circuit is first synthesized and technology mapped by ABC, the output of which serves as an input to VPR to pack, place, and route it for a given FPGA architecture description.

Instead of using ABC, which is based on AIGs, for logic synthesis and technology mapping, the proposed flow uses an MIG-based logic synthesis, as shown in Fig. 3(b). The algorithm consists of four main steps: critical path evaluation, inverter propagation, carry-chain candidate selection, and carry-chain LUT mapping. The extracted carry-chain logic are marked as *don't-touch* nodes and implemented on the FPGA's hard carry chains, while the rest is mapped onto LUTs as usual.

4.2 Critical Path Evaluation

Given an MIG, we enumerate all critical paths from primary inputs (PIs) to primary outputs (POs). For each critical path, a cost function is used to evaluate the carry-chain implementation benefits, including the gains of both nodes and edges after carry-chain extraction. The nodes gain N_{nodes}

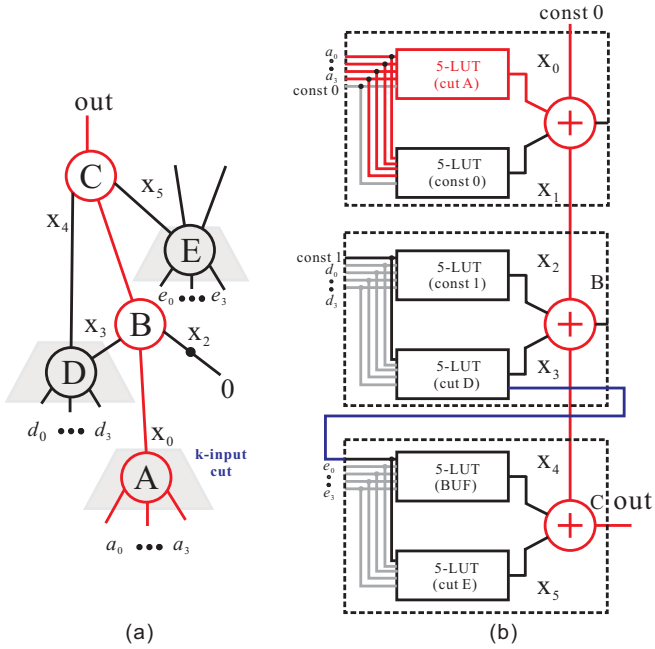

Figure 4: (a) A critical path in an MIG and (b) its LE carry-chain implementation.

consists of three parts: 1) the number of nodes in critical path, n_{path}; 2) the number of nodes that will be implemented by carry-chain LUTs, n_{fill}; 3) the number of nodes in 2) that will be duplicated due to its fanout to a node that does not belong to 1) and 2), n_{dup}. The nodes gain N_{nodes} is then defined as

$$N_{\text{nodes}} = n_{\text{path}} + n_{\text{fill}} - n_{\text{dup}} \tag{1}$$

The nodes in 1) and 2) are marked as blackbox nodes. The edges gain N_{edges} considers the number of internal edges to connect all blackbox nodes, n_{inter}, and the external nets that connect blackbox nodes to non-blackbox nodes, n_{exter}. Hence, edges gain N_{edges} is then defined as

$$N_{\text{edges}} = n_{\text{inter}} - n_{\text{exter}} \tag{2}$$

The cost function to evaluate a path is defined as

$$f = \alpha N_{\text{nodes}} + \beta N_{\text{edges}} \tag{3}$$

where α and β are weight factors used to provide similar weights to the two terms of the equation for combinational optimization:

$$\alpha + \beta = 1.0 \tag{4}$$

$$\alpha N^{\text{avg}}_{\text{nodes}} = \beta N^{\text{avg}}_{\text{edges}} \tag{5}$$

where $N^{\text{avg}}_{\text{nodes}}$ and $N^{\text{avg}}_{\text{edges}}$ are the average nodes and edges gain among all critical paths. The path with highest cost is selected as candidate for carry-chain implementation.

Figure 4(a) provides an example in which the nodes $\{A, B, C\}$ are on the candidate path, $n_{\text{path}} = 3$; supposing the nodes $\{D, E\}$ will be implemented by carry-chain LUTs, $n_{\text{fill}} = 2$; the blackbox nodes are then $\{A, B, C, D, E\}$, because node E has another fanout to a non-blackbox node, $n_{\text{dup}} = 1$. Therefore, $N_{\text{nodes}} = 3 + 2 - 1 = 4$. For edges gain, each MAJ node has 3 input edges and the number of blackbox nodes is 5, thus $n_{\text{inter}} = 15$. Node E has 2 external nets that connect non-blackbox nodes, thus $n_{\text{exter}} = 2$, and $N_{\text{edges}} = 15 - 2 = 13$. The carry-chain implementation of Fig. 4(a) is shown in Fig. 4(b), in which MAJ nodes $\{A, B, C\}$ on the critical path

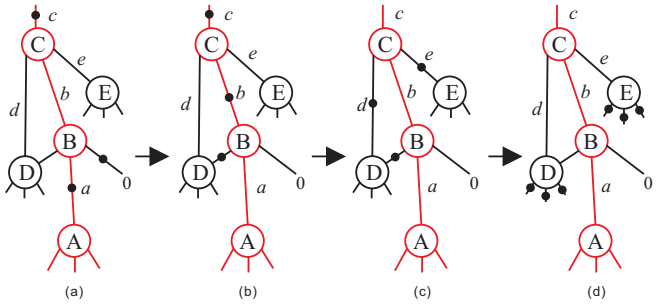

Figure 5: Inverter propagation in the carry-chain edges.

are mapped to carry chains, while nodes $\{D, E\}$ are mapped to carry-chain LUTs. The first full adder of the carry chain is used as a half adder because the first `Cin` must be connected to a constant 0 in order for A to be propagated from the LUTs to the `cout`. The details of how to determine the nodes to be implemented by carry-chain LUTs are described in Section 4.5.

4.3 Inverter Propagation

For most commercial products, the carry-chain architecture does not allow inverter operation along the carry-in/carry-out paths. Although inverters can be realized by the LUTs, it will bring in an additional delay and area overhead. To optimize the inverters used in carry chains, we used Boolean algebra inverter propagation ($\Omega.I$) rule. An approach by Testa et al. [17] shows that ($\Omega.I$) rule is efficient in complemented edges minimization for emerging nanotechnologies.

The $\Omega.I$ rule can be applied to all edges along the path recursively in a bottom-up or top-down manner. For example, Fig. 5(a) shows the original graph with inverters. The critical path $\{A, B, C\}$ is highlighted in red. In a bottom-up manner, we first find the output edge a of node A that is complemented, then we apply $\Omega.I$ to change complement states of all inputs/outputs of node B, as shown in Fig. 5(b). Next, after edges b and c are processed, as shown in Fig. 5(c), there are no inverters along the critical path. If the number of complemented edges in carry chains is even, then the complemented edges compensate each other and the final result is a carry chain with no inverters. Otherwise, at most one inverter exists at a primary input or output.

To make sure minimum number of inverters are used to feed the carry chains, the $\Omega.I$ can be also applied to carry-chain inputs by just one level. Figure 5 (c) shows an example. The output edges d and e, which are inputs to node D and E, respectively, are complemented. We apply $\Omega.I$ on node D and E to minimize inverter numbers in carry-chain inputs. Finally, the MIG after critical path inverter propagation is shown in Fig. 5(d).

4.4 Selection of Carry-Chain Candidates

Note that the carry-chain pattern (adder/MAJ + 5-input LUTs) is a better choice for technology mapping than a 6-input LUT only when the associated 5-input LUTs are efficiently utilized. After extracting a chain of MAJ nodes from a logic network, it is possible that two inputs of some MAJ nodes are primary inputs. Consequently, to accommodate such MAJ nodes, the two 5-input LUTs have to be configured as buffers, which we define as *loose carry-chain pattern*. In addition, there could be more than one loose carry-chain pattern in a chain. Figure 6 illustrates an example of two cascaded loose carry-chain patterns, where the number of

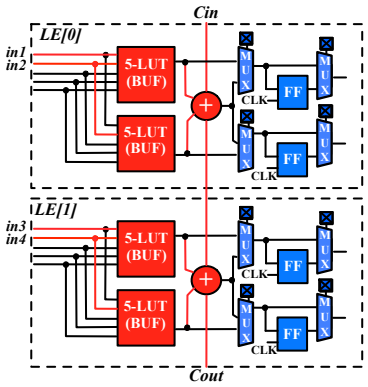

Figure 6: Two cascaded loose carry-chain patterns.

used inputs of the four 5-input LUTs is only four. Note that the two LEs in Fig. 6 have only five inputs (including `Cin`) in total. If only the carry-chain output `Cout` is used (the normal outputs `out0` and `out1` of LEs are not used), the two LEs could be alternatively implemented with a single 6-input LUT. Compared to a single 6-input LUT, the two LEs are less efficient in both area and delay. The pin-to-pin delays of two loose carry-chain patterns are

`Cin` to `Cout`:	$2 \cdot t_{\text{adder}}$
`in1` to `Cout`:	$t_{\text{lr}} + t_{\text{lut5}} + 2 \cdot t_{\text{adder}}$
`in2` to `Cout`:	$t_{\text{lr}} + t_{\text{lut5}} + 2 \cdot t_{\text{adder}}$
`in3` to `Cout`:	$t_{\text{lr}} + t_{\text{lut5}} + t_{\text{adder}}$
`in4` to `Cout`:	$t_{\text{lr}} + t_{\text{lut5}} + t_{\text{adder}}$
Average:	$t_{\text{lr}} + 0.8 \cdot t_{\text{lut5}} + 1.6 \cdot t_{\text{adder}}$

where t_{adder}, t_{lr} and t_{lut5} represent the delays of an adder, local routing and a 5-input LUT, respectively. The average pin-to-pin delay of a LE operating in the mode of a single 6-input LUT is $t_{\text{lr}} + t_{\text{lut6}} + t_{\text{mux}}$, where t_{lut6} and t_{mux} denotes the delay of a 6-input LUT and an output selector (a $2:1$ multiplexer). When the pin-to-pin delay of two loose carry-chain patterns is larger than that of single 6-input LUT mode, two loose carry-chain patterns should be replaced by a 6-input LUT in terms of performance. Note this is only established on the fact that only `cout` is used, while other regular outputs are not used. Similar conclusion can be drawn for LE in fracturable 5-LUT mode.

Therefore, we restore some loose carry-chain patterns, which have less than four or five used inputs in total, to be processed by standard LUT technology mapping. Given a critical path extracted from an MIG (Section 4.2), the carry-chain nodes are selected based on the above described loose carry-chain patterns checking. Therefore, the original carry chain is decomposed into several parts. However, as the carry chains start with a half adder and exit through the `sumout`, small pieces of carry chains are also too expensive to be implemented. Considering this issue, we evaluate the loose pattern and carry-chain length simultaneously. For example, if two sub-carry-chains are separated by only one MIG node, we will connect them together to construct a rational chain implementation. From our experimental results, we found that MIG nodes that have relatively low logic-levels are prone to be removed from the carry chains. This is mainly because these low-level nodes generally are fed by primary inputs, which are loose pattern mapping.

4.5 Carry-Chain LUT Mapping

The MIG network will be partitioned into two parts after the critical path is extracted. As shown in Fig. 7, the first

Table 1: Experimental Results Compared to Traditional Design Flow

Benchmark	Traditional Design Flow						Improvement by the Proposed Flow (%)						
	#LUT	D	RA	TA	CW	WL	#LUT	D	RA	TA	CW	WL	#Adder
pm1	17	1.52	4.06	15.1	26	109	23.53	-1.24	2.68	0.00	7.69	7.34	3
f51m	23	2.30	3.36	22.6	24	95	13.04	-24.44	-18.68	-33.33	-16.67	-2.11	3
o64	35	2.40	21.5	30.2	34	808	11.43	-5.32	0.00	0.00	0.00	-15.84	7
C432	52	7.12	8.83	45.3	34	362	36.54	-7.79	16.69	16.67	5.88	9.39	11
i5	69	3.06	47.5	52.8	42	1294	11.59	-3.85	20.55	0.00	14.29	-1.78	5
m4	88	2.21	8.83	67.9	34	312	6.82	-1.87	-3.70	0.00	-11.76	-11.22	5
cavlc	122	2.79	13.9	98.1	30	395	14.75	-0.99	0.00	8.04	0.00	21.01	13
too_large	173	4.35	25.4	136	40	1327	3.47	-6.50	-15.04	0.00	-15.00	-4.67	5
dalu	234	5.20	31.9	181	50	1742	11.54	-9.11	8.08	12.71	-24.00	20.44	22
t481	290	4.85	35.2	219	40	2209	-1.72	-0.38	-16.58	-0.06	-20.00	-6.38	18
i8	307	3.42	57.2	234	48	3347	0.98	-8.31	-1.40	0.00	-4.17	5.71	8
in1	387	3.58	39.7	294	34	2640	3.88	-2.89	0.00	2.72	0.00	-8.71	10
C6288	433	11.18	46.6	332	40	3120	5.31	-25.60	-1.85	2.41	-5.00	-3.88	21
prom2	527	3.38	50.9	400	34	2553	-0.95	-11.85	-3.94	-2.00	-11.76	0.00	3
Avg.							10.01	-7.87	-0.94	0.51	-5.75	0.66	

#LUT: number of LUTs, D: delay (ns), RA: routing area, TA: total area (area unit: 10^4 number of minimum transistor size), CW: channel width, WL: wirelength, #Adder: number of adders

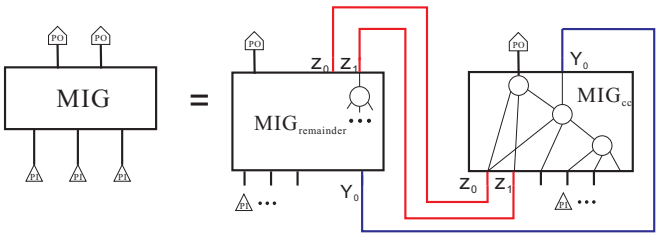

Figure 7: MIG network splitting.

part MIG_{cc} contains the carry-chain nodes and external edges, while the other part $\text{MIG}_{\text{remainder}}$ is the remainder MIG network without carry chains. For the inputs/outputs of carry-chain nodes, additional primary outputs/inputs are created to ensure correct circuit functionality.

Considering the utilization of carry-chain LUTs, we employ the CutMap algorithm [18] to find a pair of cuts, named cut_a and cut_b, in $\text{MIG}_{\text{remainder}}$ that will be implemented with the two carry-chain LUTs connected to an adder. Since there can be several options for each carry-chain LUTs, for each pair of LUTs, the two cuts with the highest cumulative gain are selected [13]. The gain of a cut is computed as the number of nodes covered by the cut that do not need to be duplicated. A node has to be duplicated if at least one of its fanout nodes is not covered by the cut.

To guarantee that the cuts are legally mapped to the LUTs, the maximum number of independent inputs among the two cuts should be less than or equal to K. Hence, for each cut pair, we make sure the cuts can be mapped by verifying that

$$nPI(cut_a) + nPI(cut_b) - nShared(cut_a, cut_b) \leq K \quad (6)$$

where K is the allowed maximum number of independent inputs, $nPI(cut_x)$ indicates the number of inputs of cut_x and $nShared(cut_a, cut_b)$ presents the shared number of inputs of cut_a and cut_b. In this paper, we consider $K = 5$ because the LB shown in Fig. 1 has at most five independent inputs.

For each input pair of carry-chain nodes in MIG_{cc}, we explore the corresponding primary outputs in $\text{MIG}_{\text{remainder}}$ to find two cuts. In Fig. 7, the Z_0 node in $\text{MIG}_{\text{reminder}}$ drives two nodes in MIG_{cc} and therefore has a fanout of 2. Note that the LB in Fig. 1 supports at most two extra outputs among the two fracturable LUTs plus one carry output. To find the best cut size for all pairs, a heuristic algorithm is used to sort carry-chain inputs from bottom-up. The inputs of the pairs are marked as visited when they have extra output. Hence, if a input node is already visited, it means the other node in the pair can have a cut size of $K - 1$ because the visited node just occupy one independent input. An example can be found in Fig. 4(b) where the LUT to implement 'cut D' has an extra output.

5. EXPERIMENTAL RESULTS

In this section, we first introduce the experimental methodology used for the experiments. Then, the results of these experiments on several MCNC benchmarks are presented.

5.1 Methodology

The design flow used for our experiment is shown in Fig. 3. Instead of using ABC for logic optimization in the baseline flow, we implemented our MIG-based approach in C++ as a command called '`fpga`' on top of the CirKit framework.[1]. The benchmarks considered from MCNC are general combinational circuits that have limited number of full adders. The arbitrary circuit description is transformed into MIG using '`xmglut`' command in CitKit [19]. We use the proposed approach on top of FPGA physical design stage, as a preoptimization step, which produces an alternative but functional equivalent `BLIF` description to the flow. The '`cec`' command is used to check the equivalence. The script of ABC used for technology mapping is "`resyn; resyn2; if -K 6`" for all soft logics from $\text{MIG}_{\text{remainder}}$ that is mapped into 6-input LUTs.

In this paper, we consider the FPGA architecture shown in Fig. 1, where each LB consists of $I = 10$ LEs, and $N = 50$ inputs. A length-4 uni-directional routing architecture, featured by $F_s = 3$, $F_{c,in} = 0.15$ and $F_{c,out} = 0.10$ is employed. We add 30% slack to the minimum routing channel width to simulate the low stress routing.

5.2 Area and Delay

The results of evaluating the proposed approach are shown in Table 1. The left part indicates results for the traditional design flow, while the right part presents the improvement percentage by the proposed approach. The number of adders

[1]github.com/msoeken/cirkit

used for carry-chain implementation is also reported in the last column.

In total, the proposed flow reduces the delay by 8% on average. The delay is composed of the total amount of logic and net delay. The stacked bar plot of delay distribution is shown in Fig. 8. Generally, the net delay is reduced in most of the considered benchmarks. This is because we use hard-wired carry chains for delay-efficient connections. However, one can note that the carry-chain implementation does not guarantee the net delay reduction. As the cost of routing to the hard carry chain is high, the net delay for the routing purpose can be considerable. The logic delay is determined by the technology mapping, as presented in Section 4.5, several extra primary outputs should be created to feed the carry chains. Basically, the technology mapper use more LUTs to implement these extra connections. This is why the proposed flow consumes 10% more LUTs than the traditional approach. Without efficient carry-chain LUT utilization, the increased logic delay caused by additional LUTs is expected to compensate the decreased net delay. Consequently, it indicates that the impact of the hard adder logic on delay improvement of an application is highly dependent on how the rest of the application is implemented by the soft logic.

The channel width is reduced by 6%, while the routing and total used area is practically the same with only few percentages points of difference, on average. Finally, the benefits of carry-chain implementations, by absorbing more logic into carry-chain LUTs, result in channel width improvement. After loose pattern removal that are too expensive for carry-chain implementation, the number of used adders range from 3 to 22. The benchmark with the largest delay reduction is `C6288`, which used 21 adders to achieve a 25.6% performance improvement.

5.3 Discussion

The performance improvement for general logic using hard-wired FPGA carry chains is encouraging. However, as also discussed in [13], the VPR packer may fail to associate the carry-chain LUTs with their respective adders and ends up wasting the carry-chain LUTs as buffers while placing their targeted functions in other logic blocks. This leads to increased logic delay, number of logic blocks, and used area. Moreover, the current VPR tool does not allow the permutation of positions of carry chains during physical synthesis. These tight constraints limit the possibility of better physical results.

Moreover, the critical path in terms of logic levels, might not be a critical path in the circuit after packing, placement, and routing. The carry chains we extracted currently are based on only one critical path at logic level. Further improvement can be achieved by extracting more critical paths by iteratively extracting the carry chains, or refining the cost function to embody the physical significance.

6. CONCLUSIONS

Hard-wired carry chains are highly efficient for arithmetic primitives implementation. Although existing research demonstrated very effective results for arithmetic-rich applications, due to the inherent inflexible design, general applications can merely take advantage of it. The MIG-based logic synthesis opens opportunities for FPGA carry chains for general purpose circuits. Instead of identifying adder primitives in either high-level description or gate-level netlist, we find carry chains effortlessly by using MIGs since the carry signal of an adder is realized by a majority gate. The proposed synthesis method for carry chains first evaluates all critical paths to determine a path candidate for carry-chain implementation. Then, the nodes along the path are further analyzed to identify and remove inefficient carry-chain mapped blocks, called loose patterns. As a result, long critical paths are decomposed into several parts for efficient carry-chain implementation. By evaluating MCNC benchmarks, which have a limited number of adders by construction, the proposed method can improve performance by 8% and channel width by 6% on average, respectively.

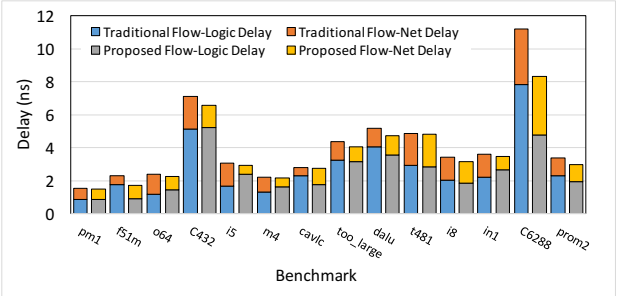

Figure 8: Comparisons of logic and net delay.

Acknowledgments. This research was partly supported by H2020-ERC-2014-ADG 669354 CyberCare, by NSF China 61501268, by Zhejiang Provincial NSF LQ15F040001, and by Ningbo City NSF 2015A610112.

7. REFERENCES

[1] J. Luu, C. McCullough, S. Wang, S. Huda, B. Yan, C. Chiasson, K. B Kent, J. Anderson, J. Rose, and V. Betz. On hard adders and carry chains in FPGAs. In *FCCM*, pages 52–59. 2014.

[2] L. Amarù, P.-E. Gaillardon, and G. De Micheli. Majority-inverter graph: A novel data-structure and algorithms for efficient logic optimization. In *DAC*, pages 1–6. 2014.

[3] L. Amarù, P.-E. Gaillardon, and G. De Micheli. Majority-inverter graph: A new paradigm for logic optimization. *IEEE Trans on CAD*, 35(5):806–819, 2016.

[4] R. Brayton and A. Mishchenko. ABC: An academic industrial-strength verification tool. In *CAV*, pages 24–40. 2010.

[5] R. E Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans on Computers*, 100(8):677–691, 1986.

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

[7] M. Hutton *et al.* Improving FPGA performance and area using an adaptive logic module. In *FPL*, pages 135–144, 2004.

[8] Altera. Stratix 10 advance information brief. *Altera Corporation*, July 2015.

[9] Xilinx. Virtex-7 user guide DS180 (v1.17). *Xilinx*, May 2015.

[10] G. De Micheli. *Synthesis and optimization of digital circuits*. McGraw-Hill Higher Education, 1994.

[11] M. T. Frederick and A. K. Somani. Beyond the arithmetic constraint: depth-optimal mapping of logic chains in LUT-based FPGAs. In *FPGA*, pages 37–46. 2008.

[12] T. B. Preußer and R. G. Spallek. Enhancing FPGA device capabilities by the automatic logic mapping to additive carry chains. In *FPL*, pages 318–325. 2010.

[13] A. Petkovska, G. Zgheib, D. Novo, M. Owaida, A. Mishchenko, and P. Ienne. Improved carry chain mapping for the VTR flow. In *ICFPT*, pages 80–87, 2015.

[14] L. Amarù, A. Petkovska, P.-E. Gaillardon, D. Novo, P. Ienne, and G. De Micheli. Majority-inverter graph for FPGA synthesis. In *SASIMI*, pages 165–170, 2015.

[15] J. Rose, J. Luu, C. W. Yu, O. Densmore, J. Goeders, A. Somerville, K. B Kent, P. Jamieson, and J. Anderson. The VTR project: Architecture and CAD for FPGAs from Verilog to routing. In *FPGA*, pages 77–86, 2012.

[16] Berkeley Logic Synthesis and Verification Group ABC: A system for sequential synthesis and verification, http://www.eecs.berkeley.edu/~alanmi/abc/.

[17] E. Testa, M. Soeken, O. Zografos, L. Amarù, P. Raghavan, R. Lauwereins, P.-E. Gaillardon, and G. De Micheli. Inversion optimization in majority-inverter graphs. In *NANOARCH*, 2016.

[18] J. Cong and Y.-Y. Hwang. Simultaneous depth and area minimization in LUT-based FPGA mapping In *FPGA*, 1995.

[19] W. Haaswijk, M. Soeken, L. Amarù, P.-E. Gaillardon, and G. De Micheli. A novel basis for logic rewriting. In *ASPDAC*, 2017.