2021 31st International Conference on Field-Programmable Logic and Applications (FPL) | 978-1-6654-3759-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/FPL53798.2021.00065

Post-LUT-Mapping Implementation of General Logic on Carry Chains Via a MIG-Based Circuit Representation

Jin Hee Kim
University of Toronto
Toronto, Canada
kimjin14@ece.utoronto.ca

Jason Anderson
University of Toronto
Toronto, Canada
janders@ece.utoronto.ca

***Abstract*—Carry chains on FPGAs have traditionally been only used for fast binary arithmetic operations. In this paper, we propose using the carry chain to implement general logic as a means of reducing the critical path delay and raising performance. To achieve this, we use a Majority-Inverter Graph (MIG) to represent the application during technology mapping, since carry functionality directly maps to the majority logic function. This aligns the subject graph of technology mapping with the capabilities of the carry chain. We first map an application to LUTs, then determine a chain of critical LUTs containing paths of majority "gates" that we deem beneficial for mapping onto the carry chain. We place such paths onto the carry chains, with the remaining logic in LUTs. In an experimental study using a suite of benchmarks, we observe that the proposed approach yields a post-place-and-route critical path delay that is superior to using delay-optimized mapping, yet without the significant area penalty. With carry-chain optimizations, area-delay product is improved by 9% vs. baseline LUT mappings.**

I. Introduction

Field-programmable gate array (FPGA) architecture has evolved to include hardened blocks that perform the key operations deemed important enough to commit silicon area to raise performance. One hardened block is the carry-chain routing to speed up arithmetic circuits that are often implemented on FPGAs. These have traditionally been used for arithmetic operations only, recognized by the synthesis tool when the design contains the + or - operators in the HDL. When no arithmetic operations are present, the adders and dedicated carry chain routing normally remain unused. The carry-chain routing is hardwired and provides fast connections for the carry signal in arithmetic operations, such as ripple-carry addition. In this paper, we automatically infer the usage of carry chains in general applications to speed up circuits without area cost.

Majority-Inverter Graphs (MIGs) have recently been shown to be a promising subject graph versus And-Inverter Graphs (AIGs) for logic optimization [1], [2]. The majority logic function, $f_{MAJ} = ab + bc + ac$, accepts three inputs and evaluates as true if at least two of the three inputs are true. This function is equivalent to the carry function in a full adder. Therefore, given a MIG representation of a circuit, it becomes much easier to select majority nodes that could benefit from being mapped to carry chains. In essence, this is because the subject graph used in logic synthesis is closely aligned with the carry hardware already present in FPGA logic blocks. In this work, we explore how a MIG representation can be used to select paths to map onto the carry chain to improve performance. If the entire circuit is represented using MIG nodes, we can put any path onto the chain; that is, we are not restricted to solely the arithmetic operations.

We consider using the carry chain to improve performance, and compare with respect to other optimization techniques during logic synthesis and technology mapping. We first perform standard technology mapping to look-up-tables (LUTs). Then, we select a chain of critical LUTs that cover a chain of MIG nodes that can be "hoisted" onto, i.e. mapped onto, the carry chain. We use two approaches to determine if carry-chain mapping is possible: a *structural* approach involving a rapid exploration of the MIG covered by a LUT, and a *formal* approach using quantified Boolean formula (QBF). We estimate if the carry-chain mapping will provide a delay improvement, and if so, we perform the mapping. Once LUT and carry-chain mapping is completed, we place and route using VPR [3]. We target a Xilinx-style architecture, which allows the carry-output signal to propagate to the carry-input of the next logic element, as well as exit through general routing.

Using the carry chains allows us to increase the performance of the circuit, while avoiding a large area cost, as compared to delay-optimized technology mapping. We observe that using an area-optimized technology mapper with carry chain mapping, we can achieve delay *superior* to mapping with a delay-optimized mapper with no area increase over the area-optimized mapper. We also apply carry-chain mapping to circuits mapped with a delay-optimized mapper and show a small performance improvement.

The main contributions of this work are:

- MIG-based post-LUT mapping for carry chains via structural MIG analysis and restructuring, and QBF.
- An experimental study to quantify the performance increase and area consequences. We show that carry optimizations, applied to an area-based mapping, provide superior performance to a baseline delay-based mapping, yet require 7% less area. In addition, the proposed carry optimizations improve area-delay product by 9% vs. baseline mappings.

II. Related Work

Previous work by Frederick et. al [4] presented ChainMap, an extension to FlowMap [5] to use the dedicated carry chain routing for chains of LUTs. However, this technique cannot be applied to more recent FPGAs, as they do not contain such a routing path within the architecture.

Preusser et al. [6] considered using the carry-chain architecture for general logic by automatically detecting LUTs whose

978-1-6654-3759-2/21/$31.00 ©2021 IEEE
DOI 10.1109/FPL53798.2021.00065

Authorized licensed use limited to: National Tsing Hua Univ.. Downloaded on May 07,2023 at 15:41:34 UTC from IEEE Xplore. Restrictions apply.

functions can take advantage of carry-chain structures. Their work was not applied post-mapping, but rather, they altered the cut-selection phase of technology mapping to choose cuts with desirable properties for carry-chain remapping. Their work was able to reduce LUT depth by ∼20%, at a significant area penalty. The authors did not place/route the circuits, so the performance consequences are unclear.

Recently, Chu et al. [7] used MIG-based synthesis for improving circuit performance. The authors selected a path in an MIG to map onto the carry chain by estimating the potential improvement prior to LUT mapping. Once MIG nodes are selected for the carry chain, the rest of the circuit is mapped in ABC as an AIG. The final mapping showed an average delay improvement of 8% with 10% increase in the number of LUTs and 6% reduction in channel width on small benchmark circuits having ∼500 LUTs or fewer.

In contrast to [7] and [6], our work is applied post technology mapping to small and large circuits, and provides performance improvements post-routing, with little to no area penalties.

III. Background

A. Majority-Inverter Graph and Mockturtle

A Majority-Inverter Graph (MIG) is a promising circuit representation for logic synthesis. An MIG is composed of 3-input majority nodes with potentially inverted edges between the nodes. The 3-input majority function, $f_{MAJ} = ab+ac+bc$, can also compute 2-input `AND` and 2-input `OR` by tying one of the inputs to 0 or 1, respectively. As such, any AIG can be represented by an MIG. AIGs are a commonly used subject graph for logic optimization using the ABC logic synthesis framework [8]. An MIG subject graph permits a wider variety of Boolean optimization techniques, leading to a post-optimization graph that has fewer logic levels [9]. There have been various logic synthesis techniques proposed for MIGs and this is an active research area [1], [10], [11].

Mockturtle [12] is an open-source library for logic synthesis and technology mapping that is designed to support various subject graphs, such as AIGs, MIGs, or XOR-Majority Graphs (XMGs) [13]. It allows researchers to apply the same synthesis and mapping algorithms to these graphs. For circuits represented as an MIG, there are several logic optimization passes available. We use a combination of algebraic depth-rewriting [14] for logic-level optimization and Boolean cut-rewriting and resubstitution [13], [15] for reducing the number of nodes. The details of these optimizations be found in the respective papers.

Mockturtle includes a LUT mapper for FPGAs, which uses a priority-cuts-based mapper as the underlying algorithm [16]. The LUT mapper is similar to the `&mf` LUT mapping implementation in ABC [8]. First, cut enumeration is performed where N priority cuts are selected for each node and saved. In our work, we change the cost of priority cuts to optimize delay as the primary cost since it results in better delay/area product for baseline LUT mapping. The Mockturtle mapper uses area flow as the primary cost and delay as the secondary cost. The LUT mapper selects cuts that optimize area flow for a few iterations and does a final mapping to optimize the estimated local area. We refer to the existing LUT mapper in Mockturtle as the *area-optimized* LUT mapper. We modified and created another version of the mapper to use delay as the primary cost for selecting the best cut. We refer to this as *delay-optimized* LUT mapper.

B. Quantified Boolean Formula (QBF)

QBF is a generalization of Boolean satisfiability where variables may have existential (∃) or universal quantifiers (∀). For example, a QBF such as: $\exists x, y\ \forall z : f(x, y, z)$ is the decision problem that asks: *Does there exist specific Boolean values for x and y, such that for all Boolean values of z, f(x,y,z) evaluates to true?* QBF has previously been applied in FPGA technology mapping to non-carry-based structures [17].

C. FPGA Carry-Chain Architecture

Logic blocks in modern FPGAs are comprised of multiple fracturable logic elements (FLEs) as shown in Fig. 1. FLEs contain fracturable LUTs – LUTs whose constituent sub-LUTs can be used independently (but possibly requiring some input-signal sharing), carry circuitry, and optional flip-flops (FFs) on the outputs. These are connected through intra- and inter-cluster routing, which we refer to as general routing. There is dedicated hard-wired routing between the FLEs within a logic block for fast carry computation for arithmetic operations; and between vertically adjacent logic blocks, dedicated routing exists as well. Such hard-wired connections are referred to as carry-chain routing.

In Xilinx FPGAs, the logic block consists of 2 slices [18]. Each slice contains four fracturable 6-input LUTs and has hardened circuitry for fast carry propagation. When in arithmetic mode, the 6-input LUT, configured as two 5-input sub-LUTs, are used to realize the propagate (p) and generate (g) signals for the full-adder circuit. The carry signal has dedicated routing to propagate from one FLE to another. The carry-out and sum-out bits are both connected to an output multiplexer, allowing the signals to exit the slice into general interconnect.

IV. Target Architecture

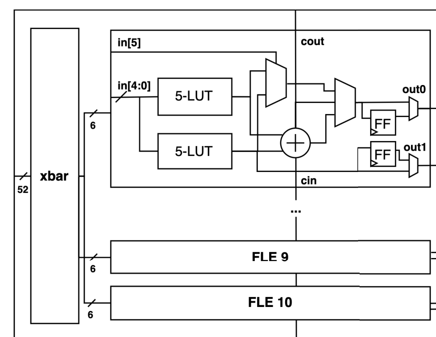

Fig. 1: Target Architecture: 10 Xilinx-Style FLEs within a Logic Block.

Authorized licensed use limited to: National Tsing Hua Univ.. Downloaded on May 07,2023 at 15:41:34 UTC from IEEE Xplore. Restrictions apply.

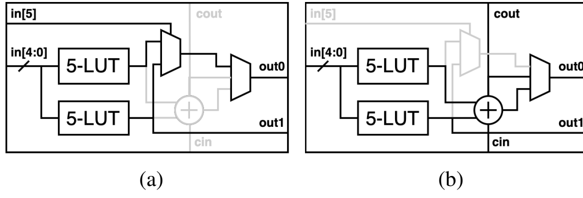

Fig. 2: FLE configuration in VPR a) 6-LUT mode; b) arithmetic mode.

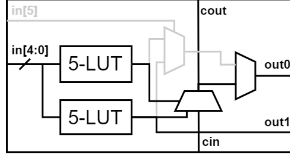

Fig. 3: Xilinx-style FLE configured in arithmetic mode.

We target an architecture with a logic block that contains 10 Xilinx-style FLEs and a 50% depopulated input crossbar as shown in Fig. 1. Each FLE contains a 6-input fracturable LUT. Fig. 2(a) shows a portion of an FLE configured in 6-input LUT mode and Fig. 2(b) shows the FLE configured in arithmetic mode. When in arithmetic mode, the 5-input LUTs drive the full adder, whose carry-out is connected to the adder in its adjacent FLE through dedicated routing.

Observe that our FLE (Fig. 1) contains a full adder, while Xilinx commercial FPGAs contain a 2-to-1 multiplexer in that position, shown in Fig. 3, where the select input of the multiplexer is driven by one of the LUT outputs, and the two data inputs of the multiplexer are driven by the other LUT output and cin, respectively. We are using a full adder for simplicity, as the place and route tool, VPR [3], already has modelling support for it. However, it is important to realize that both architectures can be made *equivalent* by Boolean algebra manipulation of the functions implemented within the dual-output LUT. Specifically, in the full-adder case, once we select a majority node to map to the adder and the cuts to be placed in the 5-LUTs, we assign the 3 inputs to the majority node (x_1, x_2, x_3) to the 3 inputs of the full adder (f_{LUT_a}, f_{LUT_b}, c_{in}) and set the 5-LUT functionality to the selected cuts of the children. In the multiplexer case, the two 5-LUTs will determine the propagate (multiplexer select input) and generate (multiplexer data input) functions and use that to determine whether c_{in} should propagate or not. The resulting carry out function is the same in both cases.

Mapping to an architecture with full adder, the LUT-mask and carry-out function would be:

$$f_{LUT_a} = f_{cut_{x_1}}, f_{LUT_b} = f_{cut_{x_2}},$$
$$c_{out} = f_{LUT_a} \cdot f_{LUT_b} + f_{LUT_b} \cdot c_{in} + f_{LUT_a} \cdot c_{in}$$

or for the multiplexer, the LUT-mask and carry-out function would be:

$$f_{LUT_a} = f_{cut_{x_1}} + f_{cut_{x_2}}, f_{LUT_b} = f_{cut_{x_1}} \cdot f_{cut_{x_2}},$$
$$c_{out} = f_{LUT_a} \cdot c_{in} + \overline{f_{LUT_a}} \cdot f_{LUT_b}$$

Therefore, one can map a MIG node to the carry element (full adder or multiplexer) and determine the LUT mask according to the architecture being targetted.

V. Technology Mapping

We now describe our extension to the LUT mapper to perform post-LUT-mapping carry-chain mapping. We first perform 6-LUT mapping then we select chains of LUTs on the critical path that can be connected together using the carry-chain interconnect. The path of MIG nodes from the selected LUT chain is placed on the carry chain and inverters are removed to match the underlying hardware (inverter removal preserves logic functionality, described below). Lastly, we write the LUT and carry mapping to BLIF for circuit equivalence checking, and place and route.

Fig. 4 illustrates the desired mapping. The left side of the figure shows a chain of 3 LUTs, where $sigA$ and $sigB$ represent the timing-critical signals between the LUTs. The right side shows the mapping, where sigA and sigB lie on the fast carry-interconnect, rather than being routed through the FPGA's general interconnect.

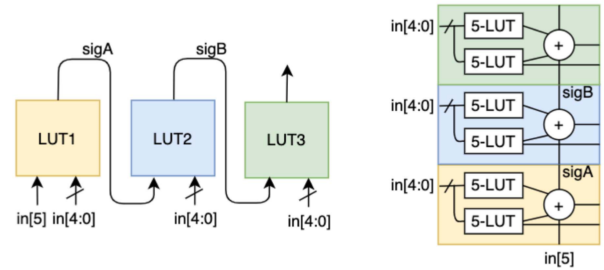

Fig. 4: Mapping a chain of three critical LUTs to the carry chain.

A. Carry Chain Selection

Starting from a critical primary output (PO), we recursively look at the critical driving LUT and create a chain of LUTs until we reach a LUT that cannot be mapped or a LUT that only has primary input (PI) as its inputs. We refer to the LUTs in critical path as *critical LUTs* and the inputs to a critical LUT driven by other critical LUTs as *critical inputs*. We determine whether the critical LUT and the associated critical input on the carry chain can be remapped to use the carry chain. Note that a LUT on the critical path may have multiple critical inputs. In this case, we aim to check whether one of the signals on such inputs can be moved to the carry chain. We map as many chains of critical LUTs as we can starting with those with the least number of critical inputs.

Given a critical 6-LUT having inputs in[5:0] implementing a logic function f and assuming, without loss of generality, that in[5] is the critical input, our aim is to test whether f or its complement can be remapped into the structure shown in Fig. 5. The destination structure comprises two 5 LUTs with shared inputs, and critical LUT input in[5] attached directly to the majority gate, MAJ. The complemented form of f is

Authorized licensed use limited to: National Tsing Hua Univ.. Downloaded on May 07,2023 at 15:41:34 UTC from IEEE Xplore. Restrictions apply.

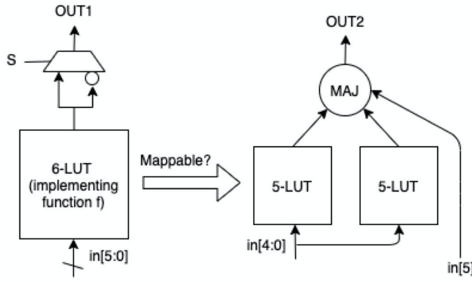

Fig. 5: Mappability test.

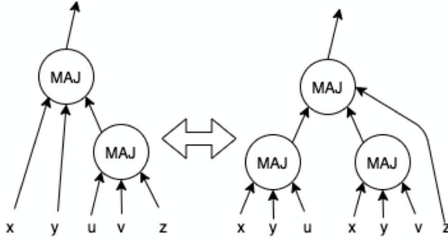

Fig. 6: Distributive rule in MIG algebra [1].

acceptable owing to inversion properties of majority, discussed below. The mappability test is conducted in two ways: 1) a fast structural check using properties of majority; 2) formally using QBF.

The structural check leverages the distributive rule in MIG algebra [1], $MAJ(x, y, MAJ(u, v, z)) = MAJ(MAJ(x, y, u), MAJ(x, y, v), z)$, illustrated in Fig. 6. Observe on the left that z is input to a lower-level majority gate. The rule allows z to be elevated as input to the higher-level majority gate, shown on the right. Consider a MIG subgraph covered by a critical LUT, by iterative application of the distributive rule it is always possible to elevate the critical input to the top-most majority gate in the LUT (the gate driving the LUT output signal). This aligns with the mappability structure shown on the right of Fig. 5. If, however, inputs x, y, u, or v also depend on z, then there is a reconvergent path from z within the LUT's internal MIG subgraph, and it may not be possible to eliminate z from also being required as an input to the LUT. Therefore, the rapid structural check is as follows: Given a critical LUT and an associated critical input to that LUT, check whether there is a reconvergent path from the input to the LUT output in the MIG subgraph covered by the LUT. If no such reconvergent path exists, then the mappability test passes.

If the structural check fails, we formulate a QBF decision problem. Referring to Fig. 5, S is the select input to the multiplexer, $OUT1$ is the true or complemented form of function f, $OUT2$ is the output of the majority gate on the right, and let m_{63-0} be the 64 SRAM configuration cells contained within the two 5-LUTs on the right of the figure. The QBF is as follows:

$$\exists S, m_{63-0}\ \forall in[5:0]\ \exists OUT1, OUT2 : \overline{OUT1 \oplus OUT2} \quad (1)$$

Informally, we are testing whether *there exists* settings for S and m_{63-0} *for all* input combinations in[5:0] *there exists* $OUT1$ and $OUT2$ such that $OUT1$ equals $OUT2$. We solve this using the RAReQS QBF solver [19]. In the experimental study, we show that the fast structural check is sufficient to identify the majority of mappable cases.

Although there is no area cost as shown above, there is a delay cost associated with using the adder. The delay of using a 5-LUT and adder is 1.26× more than using a 6-LUT alone (c.f. Section VI). However, by using the dedicated carry chain, we avoid the delay of using the general routing. For a scenario where a LUT has more than 1 critical input and has multiple fanouts, we would reduce the delay of the path that is placed on the carry chain (saves routing delay) but increase the delay to the other LUTs that are not on the carry chain (owing to the extra adder delay). Critical LUT chains must be at least of length 2 for remapping onto the carry chain. It is not beneficial for performance to remap a single critical LUT to use the carry chain – benefits only arise if some of its neighboring critical LUTs are also remapped.

B. Inverter Removal and BLIF Generation

Once all paths to be placed on the carry chain have been decided, we perform inverter removal on these paths. Carry-chain interconnect on FPGAs does not have optional inverters so it is necessary to remove any inverters in the MIG node path that will be placed on the carry chain. We use the method described in [7] by pushing the bubbles from the MIG path to the input LUTs. The inverter removal is possible because an inverted majority function is equal to a non-inverted majority function with all of its inputs inverted, i.e., $\overline{ab + ac + bc} = \overline{a}\ \overline{b} + \overline{a}\ \overline{c} + \overline{b}\ \overline{c}$.

Lastly, we set the LUT configuration for all of the 6-LUTs that were remapped to use the FLE in arithmetic mode. Each 5-LUT in front of the adders gets connected to the dependent inputs from the original 6-LUT. Then, we set the LUT mask for each 5-LUT. This is then used to generate BLIF for combinational equivalency checking (CEC) and place and route. We used ABC's CEC tool to compare the post-LUT and carry mapping to the original LUT-only mapping to verify equivalency.

The BLIF generated for CEC can be used for place and route, but we cannot guarantee that our FLE will be packed as expected during packing. To ensure a proper packing of LUTs and carry, we generate BLIF that groups (pre-packs) the two 5-LUTs and adder together using VPR's user-specified block method [3], discussed in the next section. This ensures that the 5-LUTs and the adder stay together during place and route. We also limit the length of the carry chains to at most 50, which would be placed across 5 logic blocks. This is because the carry chains cannot be broken up during place and route, due to a limitation of the place and route tool.

Authorized licensed use limited to: National Tsing Hua Univ.. Downloaded on May 07,2023 at 15:41:34 UTC from IEEE Xplore. Restrictions apply.

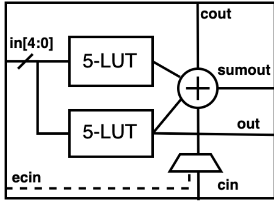

Fig. 7: User-specified block `lut_adder`.

From	To	Delay
in[*]	out	0.235
in[*]	cout	0.329
in[*]	sumout	0.329
cin	cout	0.010
cin	sumout	0.010
ecin	cout	0.020
ecin	sumout	0.020

TABLE I: `lut_adder` delays in ns.

VI. Architecture Modelling

We modified one of the existing architecture descriptions distributed with VPR [3] to reflect our target logic block architecture. A user-specified block, called *lut_adder*, is employed to pack the two 5-LUTs and the adder combination in a chain. These blocks are packed together by using the chain-pack pattern [20] during BLIF generation. The block *lut_adder*, shown in Fig. 7, has 6 inputs, 5 for the LUTs and 1 for the adder carry-in signal. It has 3 outputs, 1 carry-out signal (*cout*), 1 sum-out signal (*sumout*), and 1 output connected to one of the 5-LUTs (*out*). There are delay paths from all inputs to *cout* and the *sumout* output, and there is a delay path from the 5 LUT inputs to the *out* signal. We specify the delay from any LUT input to *out* as 0.235ns and to *cout* as 0.329ns. Referencing previous work on carry chains [21], we estimated the delay of an adder from its input to carry output to be 40% less than the delay through a 5-LUT. Since we route the carry out signal to out0 from Fig. 2(b) in arithmetic mode, we use the same delay to *sumout* as *cout*. The delays of the paths within the user-specified block are summarized in Table I. All of the other delays, such as 6-LUT delay of 0.261ns, remain "as is" from the available architecture in VPR.

In Xilinx-style FPGAs, each FLE has an additional "bypass" input that allows a signal to get onto the carry chain at any point along the chain. To mimic this behaviour, we add an additional input to our *lut_adder* called *ecin*. This input is only connected to the routing for the very first FLE of the logic block. We use this to connect the start of a carry chain only in the case when we do not have any extra LUT inputs to accommodate the carry-in signal, i.e. all 5 inputs are being used for the FLE at the very start of the LUT chain.

VII. Experimental Study

In this section, we present the post-place-and-route results for LUT and carry-mapped circuits compared to baseline LUT-mapped circuits to evaluate how carry mapping can be used as a delay improvement strategy.

A. Experimental Setup

We use the EPFL benchmark suite [22] to evaluate our carry mapping work. We apply logic optimization to reduce the number of nodes. We generated MIG benchmarks using a logic synthesis and optimization framework Cirkit [2], which uses Mockturtle as its back-end library. We used commands `mighty; cut_rewrite; resubstitution; cut_rewrite; resubsitution;` to first optimize for delay then reduce area at the cost of delay using cut rewrite and resubtitution.

We use Mockturtle 0.1 [12] and VPR 7.0 [3] to map and place-and-route the circuits. We set our mapper to map to 6 LUTs (using 8 stored priority cuts for each internal MIG node). We first route all of the baseline circuits to find the minimum channel width for each circuit. Then, we increase that by 25% for each circuit, and route each circuit with that specific fixed channel width, reflecting a medium-congestion routing scenario. We place and route all circuits with 3 different seeds and report the average.

The columns labelled "6-LUT" in Table II summarizes the post-mapping area, depth, and post-place-and-route delay of the baseline implementation. The baseline was generated using delay-optimized priority cuts enumeration followed by area- and delay-optimized technology mapping. We observed this approach led to minimal area increase with considerably lower LUT depth than using area-optimized cut enumeration.

B. Carry Mapping Results

The columns labelled "6-LUT and Carry" in Table II show the results for the proposed LUT and carry mapping. On the left side of the table, we compare carry chain mapping applied to area-optimized LUT mapping. Comparing to the baseline results, we observe the change in LUT count is nearly flat, on average. In terms of LUT depth, there is a $\sim$20% reduction in depth when the carry-chain mapping optimization is applied. Note that the depth numbers in the table reflect *equivalent* LUT depth, where in the carry-mapping case, depth has been scaled according to the delay numbers presented in Section IV. We use the equivalent LUT depth to guide decision making about carry-chain mapping; specifically, we commit to carry re-mapping when the equivalent depth is improved by at least 10% over the baseline. Lowering this threshold, e.g. to 5%, caused the post-place-and-route critical-path delay of the `log2` benchmark to become worse than the baseline. Use of the carry chain creates a placement scenario where certain LUTs and their associated carry logic must be kept together in a strict pattern during placement. We suspect this may lead to a negative impact on routing delays in some cases.

Post-place-and-route critical-path delay shows a reduction of 9% vs. the baseline. In the best cases, the circuits `adder`, `arbiter`, and `square` circuits show improvements of 30%, 41%, and 35%, respectively. The delay reductions come from both logic and routing delay. The reduction in routing delay is expected when LUTs placed on the carry chain do not fanout to multiple LUTs.

We also compare the area and delay of *area-optimized* LUT and carry mapping to 6-LUT mapping using an alternative baseline *delay-optimized* mapper (third group of columns in the table). The delay-optimized mapper produces solutions with a geomean of 764 LUTs; a geomean LUT depth of 11.3; and, geomean post-routing critical path delay of 6.89ns. Comparing to the results in second group of columns of Table II, we observe that we can achieve circuits with slightly higher performance (6.75ns vs. 6.89ns) and considerably better area cost (geomean

Authorized licensed use limited to: National Tsing Hua Univ.. Downloaded on May 07,2023 at 15:41:34 UTC from IEEE Xplore. Restrictions apply.

	Area-opt						Delay-opt					
	6-LUT			6-LUT + carry			6-LUT			6-LUT + carry		
Benchmark	# of LUTs	LUT Depth	Total Delay	# of LUTs	LUT Depth	Total Delay	# of LUTs	LUT Depth	Total Delay	# of LUTs	LUT Depth	Total Delay
adder	335	17	7.29	335	8	5.08	423	10	5.43	423	7	4.62
arbiter	2583	10	5.82	2583	4	3.43	2979	8	5.61	2979	4	3.97
bar	512	4	3.26	512	4	3.26	512	4	3.26	512	4	3.26
cavlc	131	7	2.93	132	4	2.66	132	5	2.33	132	5	2.33
ctrl	29	2	1.30	29	2	1.30	29	2	1.17	29	2	1.17
dec	273	2	1.95	273	2	1.95	273	2	1.95	273	2	1.95
i2c	354	5	2.72	354	5	2.72	359	5	2.59	359	5	2.59
int2float	53	5	2.31	53	3	2.00	54	5	2.33	54	3	2.09
log2	9012	106	52.41	9012	106	52.41	9621	85	50.46	9621	85	50.46
max	2267	20	12.16	2267	20	12.16	2940	18	12.32	2940	16	12.87
mem_ctrl	12266	35	20.40	12277	31	19.92	13069	28	17.44	13069	28	17.44
multiplier	5941	54	26.11	5942	37	23.79	6487	37	23.66	6487	37	23.66
priority	288	42	19.10	288	42	19.10	365	41	22.88	365	41	22.88
router	76	7	3.09	76	7	3.09	77	7	3.36	77	7	3.36
sin	1688	45	25.45	1688	45	25.45	1850	38	24.00	1850	38	24.00
square	3792	35	15.13	3792	17	9.80	3934	21	10.08	3934	15	10.35
voter	1550	17	13.63	1550	17	13.63	1590	14	11.59	1590	14	11.59
geomean	707.5	13.4	7.45	707.8	10.6	6.75	763.6	11.3	6.89	763.6	10.0	6.67
ratio	1	1	1	1.00	0.79	0.91	1	1	1	1.00	0.88	0.97

TABLE II: Num of LUTs, LUT depth, and critical delay (in ns) of 6-LUT mapping and 6-LUT and carry mapping using area- and delay-optimized LUT mappers.

Scenario	Area-delay product (LUT-ns)	Area-delay product normalized to baseline	
		area-opt	delay-opt
Area-opt	5271	1	1.001
Area-opt + carry	4778	0.906	0.908
Delay-opt	5261	0.998	1
Delay-opt + carry	5093	0.966	0.968

TABLE III: Geomean area-delay product results.

of 708 LUTs vs. 764). That is, the carry-chain mapping applied to an area-optimized LUT mapping achieves better performance at considerably less area vs. a delay-optimized LUT mapping.

We applied carry optimization to the delay-optimized LUT mapping and the results are shown on the right side of Table II. With no change in the number of LUTs, we achieve a 12% reduction in levels and a 3.2% reduction in post-routing delay. Below, we discuss why area-optimized LUT mapping solutions offer more potential for carry-chain optimizations. Table III gives geomean area-delay product results for the four flows considered. The area-optimized mapping followed by carry-chain optimization provides 9% improvement over both the area and delay-based mapping baselines.

We performed an analysis to determine why some circuits show significant improvement when mapped to use carry chain. Table IV shows for the area (left) and delay-based mappings (right): the number of critical LUTs that pass the structural check, the QBF check, and the total number of critical LUTs. The last two rows give geomean results and show the ratios of LUTs vs. the total number that are critical.

From the last row of the table, we observe that in the area-based mappings, 81% of critical LUTs pass the structural check; 86% pass the QBF check. This implies that the majority of critical LUTs have at least critical input signal that can be elevated onto the carry chain, and that the rapid structural check is sufficient to determine this. In the delay-based case, fewer critical LUTs can use the carry chain, and there are more critical LUTs overall, accounting for the reduced performance benefits seen in this scenario.

While a preponderance of critical LUTs can be remapped to use the carry chain, we observed that many of such LUTs have *multiple* critical inputs. Moving one critical input signal to use the carry chain is not helpful to the other critical inputs, thereby limiting the performance improvement potential.

	Area-opt			Delay-opt		
	Structural check	QBF check	Crit LUTs	Structural check	QBF check	Crit LUTs
adder	18	18	21	15	15	26
arbiter	1536	1536	1536	1792	1792	1792
bar	512	512	512	512	512	512
cavlc	8	8	8	6	6	6
ctrl	7	8	8	7	8	8
dec	273	273	273	273	273	273
i2c	61	62	62	35	35	35
int2float	6	7	7	6	7	7
log2	123	168	203	267	335	396
max	129	135	135	29	33	36
mem_ctrl	163	198	198	394	427	427
multiplier	57	57	67	393	393	1106
priority	123	123	123	192	193	193
router	18	20	25	12	13	17
sin	167	187	308	389	424	611
square	42	42	46	35	38	50
voter	176	187	596	333	333	1355
geomean	71	76	88	83	88	112
ratio	0.81	0.86	1.00	0.74	0.78	1.00

TABLE IV: Number of mappable critical LUTs using structural check and QBF check.

Authorized licensed use limited to: National Tsing Hua Univ.. Downloaded on May 07,2023 at 15:41:34 UTC from IEEE Xplore. Restrictions apply.

VIII. Conclusion and Future Works

FPGAs include fast dedicated carry routing for speeding up traditional arithmetic operations. However, such circuitry is generally unused in circuits that do not explicitly specify the operations through + or - in the input HDL. We automatically select critical paths to map to carry chains as a post-LUT mapping pass to improve performance. We achieve an area-delay improvement of 9% vs. baseline mappings. The carry-chain mapping approach, when applied to an area-optimized LUT mapping, achieves delay superior to a depth-optimized LUT mapping, while consuming considerably less area. For further work, MIG transformations at the pre-mapping stage may offer further performance improvements.

References

[1] L. Amaru, P.-E. Gaillardon, and G. De Micheli, "Majority-inverter graph: A new paradigm for logic optimization," *IEEE TCAD*, vol. 35, no. 5, pp. 806–819, 2016.

[2] M. Soeken *et al.*, "Optimizing majority-inverter graphs with functional hashing," in *DATE*, 2016, pp. 1030–1035.

[3] J. Luu *et al.*, "VTR 7.0: Next generation architecture and CAD system for FPGAs," *ACM TRETS*, vol. 7, no. 2, pp. 1–30, 2014.

[4] M. T. Frederick and A. K. Somani, "Beyond the arithmetic constraint: depth-optimal mapping of logic chains in LUT-based FPGAs," in *ACM/SIGDA FPGA*, 2008, pp. 37–46.

[5] J. Cong and Y. Ding, "FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," *IEEE TCAD*, vol. 13, no. 1, pp. 1–12, 1994.

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

[7] Z. Chu *et al.*, "Improving Circuit Mapping Performance Through MIG-based Synthesis for Carry Chains," in *ACM GVLSI*, 2017, pp. 131–136.

[8] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *Proc. CAV*. Springer, 2010, pp. 24–40.

[9] L. Amarú, P. Gaillardon, and G. De Micheli, "Majority-Inverter Graph: A novel data-structure and algorithms for efficient logic optimization," in *DAC*, 2014.

[10] W. J. Haaswijk *et al.*, "LUT mapping and optimization for majority-inverter graphs," in *IWLS*, 2016.

[11] L. Amarù *et al.*, "Majority Logic Synthesis," in *ICCAD*, 2018.

[12] M. Soeken *et al.*, "The EPFL logic synthesis libraries," *arXiv preprint arXiv:1805.05121*, 2018.

[13] W. Haaswijk *et al.*, "A novel basis for logic rewriting," in *IEEE ASPDAC*, 2017, pp. 151–156.

[14] L. Amarú *et al.*, "Majority-inverter graph for FPGA synthesis," in *SASIMI*, 2015, pp. 165–170.

[15] H. Riener *et al.*, "Scalable generic logic synthesis: One approach to rule them all," in *DAC*, 2019, pp. 1–6.

[16] A. Mishchenko *et al.*, "Combinational and sequential mapping with priority cuts," in *IEEE/ACM ICCAD*, 2007, pp. 354–361.

[17] A. C. Ling, D. P. Singh, and S. D. Brown, "FPGA PLB evaluation using quantified boolean satisfiability," in *FPL*, 2005, pp. 19–24.

[18] "7 series FPGAs CLB user guide," https://www.xilinx.com/support/documentation /user_guides/ug474_7Series_CLB.pdf, version: 1.8.

[19] M. Janota, W. Klieber, J. Marques-Silva, and E. Clarke, "Solving qbf with counterexample guided refinement," in *SAT*, Berlin, Heidelberg, 2012, p. 114–128.

[20] "Verilog-to-Routing: FPGA Architecture Description," docs.verilogtorouting.org.

[21] J. Luu *et al.*, "On hard adders and carry chains in FPGAs," in *IEEE FCCM*, 2014, pp. 52–59.

[22] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The EPFL combinational benchmark suite," in *IWLS*, 2015.

Authorized licensed use limited to: National Tsing Hua Univ.. Downloaded on May 07,2023 at 15:41:34 UTC from IEEE Xplore. Restrictions apply.