Restructuring of Arithmetic Circuits with Biconditional Binary Decision Diagrams

Luca Amarú[1], Alexios Balatsoukas-Stimming[2], Pierre-Emmanuel Gaillardon[1], Andreas Burg[2], Giovanni De Micheli[1]
Integrated Systems Laboratory (LSI), EPFL, Switzerland[1]
Telecommunication Circuits Laboratory (TCL), EPFL, Switzerland[2]

***Abstract*— *Biconditional Binary Decision Diagrams* (BBDDs) are a novel class of canonical binary decision diagrams where the branching condition, and its associated logic expansion, is *biconditional* on two variables. In this demonstration, we use an efficient BBDD manipulation package as front-end to a commercial synthesis tool to restructure arithmetic operations in critical components of telecommunication circuits. We show that our apprach meets tight timing constraints otherwise beyond the capabilities of traditional synthesis methods.**

I. Demo Description

Biconditional Binary Decision Diagrams (BBDDs) are a promising class of canonical binary decision diagrams recently introduced in [1]. While original BDDs are based on the single-variable Shannon's expansion, BBDDs employ a two-variable *biconditional* expansion, i.e., the branching condition at each decision node is dependent on two variables per time. Such feature improves the expressive power of the binary decision diagram, enabling a compact representation for arithmetic circuits [1]. In [2], an efficient software package for BBDDs has been developed and made available online [3], thus opening the opportunity to bench the potential of the new representation form in real-life designs.

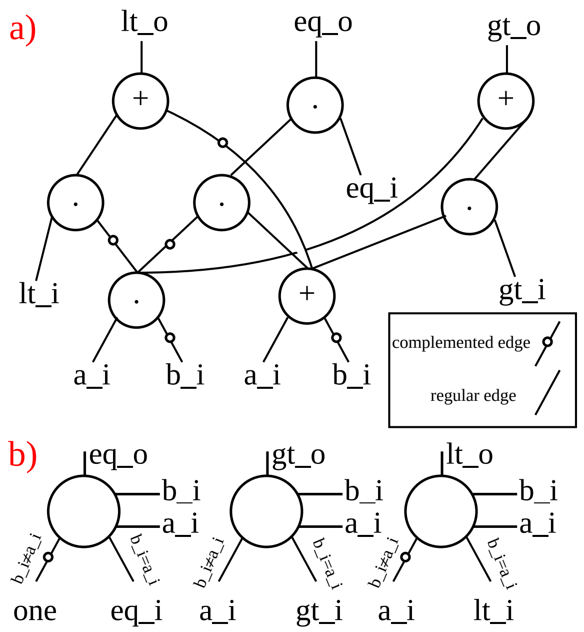

Fig. 1. Representations for the bit_comparator circuit in [4] (inverters are bubbles in edges). a) original circuit b) BBDD re-writing.

In this demonstration, we employ the BBDD manipulation package [3] as front-end to a commercial synthesis tool to restructure arithmetic operations in digital designs. We focus on telecommunication circuits as they are often dominated by arithmetic functions which implementation is key to reach a desired performance. In particular, we consider the *Iterative Decoder for Product Code* from Open Cores available online at [4]. We process all combinational circuits present in such design and we keep the restructured circuits if their size (number of logic nodes) and depth (number of logic levels) are decreased. Fig. 1(a) depicts the *bit_comparator* circuit before processing and Fig. 1(b) illustrates the equivalent circuit after BBDD-restructuring. An advantage in both size and depth is reported. Table I shows the remaining results. BBDD-restructuring is favorable for all circuits except *ext_val* that instead is more compact in its original version. We synthesized the whole design in two versions: (i) the original Iterative Product Decoder and (ii) the Iterative Product Decoder with BBDD-restructuring kept only where advantageous. Using a 90 nm CMOS target technology under a clock constraint of 2.25 ns (444 MHz clock frequency), the original design synthesis generates a negative slack of 0.26 ns while the BBDD-restructured design meets the constraint with 0.00 ns slack. The area for the original design is 17567.87 μm^2 while for the BBDD-restructured design is 21963.76 μm^2. The combinatorial verification for the same design is also successfully carried out with BBDDs in about 3 minutes.

TABLE I
Experimental results for BBDD Restructuring.

Case Study: *Iterative Product Decoder*

Logic Circuits	I/O	BBDD Restr.		Original		Gain
		Size	Depth	Size	Depth	
adder08_bit	16/9	16	8	78	19	✓
bit_comparator	5/3	3	1	8	3	✓
comp_7bits	14/3	21	7	58	14	✓
fulladder	3/2	2	1	9	4	✓
ext_val	16/8	674	16	173	29	✗
twos_c_8bit	8/8	20	8	29	8	✓

References

[1] L. Amaru, P.-E. Gaillardon, G. De Micheli, *Biconditional BDD: A Novel Canonical Representation Form Targeting the Synthesis of XOR-rich Circuits*, Proc. DATE 2013.
[2] L. Amaru, P.-E. Gaillardon, G. De Micheli, *An Efficient Manipulation Package for Biconditional Binary Decision Diagrams*, Proc. DATE 2014.
[3] *BBDD package available at*: *http://lsi.epfl.ch/page-102566-en.html*.
[4] *An iterative decoder for Product Code – from Open Cores*: *http://opencores.org/project,product_code_iterative_decoder*.