Engineering Science and Technology, an International Journal xxx (2017) xxx–xxx

HOSTED BY

ELSEVIER

Contents lists available at ScienceDirect

Engineering Science and Technology, an International Journal

journal homepage: www.elsevier.com/locate/jestch

Full Length Article

Asynchronous carry select adders

P. Balasubramanian [1]

School of Electrical and Electronic Engineering, Nanyang Technological University, 50 Nanyang Avenue, 639798 Singapore, Singapore

A R T I C L E I N F O

Article history:
Received 3 November 2016
Revised 25 January 2017
Accepted 3 February 2017
Available online xxxx

Keywords:
Asynchronous circuits
Full adder
Carry select adder
Indication
Early output
ASIC
Standard cells
CMOS

A B S T R A C T

This paper discusses the standard cell based designs of asynchronous carry select adders (CSLAs) corresponding to strong-indication, weak-indication, and early output timing regimes realized using a delay-insensitive dual-rail code for data representation and processing, and a 4-phase return-to-zero protocol for handshaking. Many 32-bit asynchronous CSLAs corresponding to a uniform input partition viz. 8-8-8-8 and a non-uniform input partition viz. 8-7-6-4-3-2-2 were considered for implementation and comparison. All the asynchronous CSLAs were physically realized in semi-custom ASIC design style using a 32/28 nm CMOS process technology. The simulation results show that the 32-bit early output asynchronous CSLA based on the uniform input partition (8-8-8-8) enables optimized data path latency, area occupancy, and average power dissipation compared to the rest.

1. Introduction

Carry select adder (CSLA) is a square root time high-speed adder [1], which offers a good compromise between the low area demand of ripple carry adders (RCAs) and the high-speed performance of carry lookahead adders (CLAs) [2,3]. Both ASIC and FPGA implementations of homogeneous CSLAs, and a hybrid architecture involving CSLA and CLA have been considered in the existing literature based on the synchronous design method [4–16]. However, with respect to robust asynchronous design methods employing delay-insensitive code(s) for data representation and processing and a 4-phase return-to-zero protocol for handshaking, to the best of our knowledge, there is no dedicated work available in the literature dealing with asynchronous CSLA excepting a theoretical work [17] that proposed just a mathematical model to predict the timing attributes of asynchronous CSLAs. This article deals with the physical implementation of a variety of 32-bit asynchronous CSLAs based on uniform and non-uniform input partitions which correspond to different timing regimes viz. strong-indication, weak-indication, and early output based on a 32/28 nm CMOS process whilst presenting the mathematical estimates alongside for comparison.

The rest of this article is organized as follows. Section 2 gives background information about robust asynchronous circuit design. Section 3 describes the asynchronous CSLA architecture and distinguishes between CSLAs constructed using uniform (8-8-8-8) and non-uniform (8-7-6-4-3-2-2) input partitions. Section 4 presents the simulation results obtained for various 32-bit asynchronous CSLAs corresponding to different asynchronous design methods and timing regimes viz. strong-indication, weak-indication, and early output. Finally, Section 5 states the conclusions.

2. Asynchronous design preliminaries

An asynchronous logic block comprising an asynchronous digital system is the combinational logic equivalent of a synchronous digital system [18]. Asynchronous logic blocks constructed using delay-insensitive data codes and adhering to the 4-phase (return-to-zero) handshake protocol are robust.

The dual-rail code is the simplest member of the family of delay-insensitive data codes [19], based on which a data wire W is encoded using two data wires W1 and W0 as shown in Fig. 1. W = 1 is represented by W1 = 1 and W0 = 0, and W = 0 is represented by W1 = 0 and W0 = 1. These two conditions represent 'valid data', and the condition of both W1 and W0 assuming 0 is referred to as the 'spacer'. The 4-phase return-to-zero handshaking requires that the application of inputs from the external environment follows the predefined sequence: valid data-spacer-valid data-spacer, and so forth.

E-mail address: balasubramanian@ntu.edu.sg.

Peer review under responsibility of Karabuk University.

[1] The research work was performed when the author was affiliated with the School of Computer Science and Engineering, Nanyang Technological University, 50 Nanyang Avenue, Singapore 639798.

http://dx.doi.org/10.1016/j.jestch.2017.02.003
2215-0986/© 2017 Karabuk University. Publishing services by Elsevier B.V.

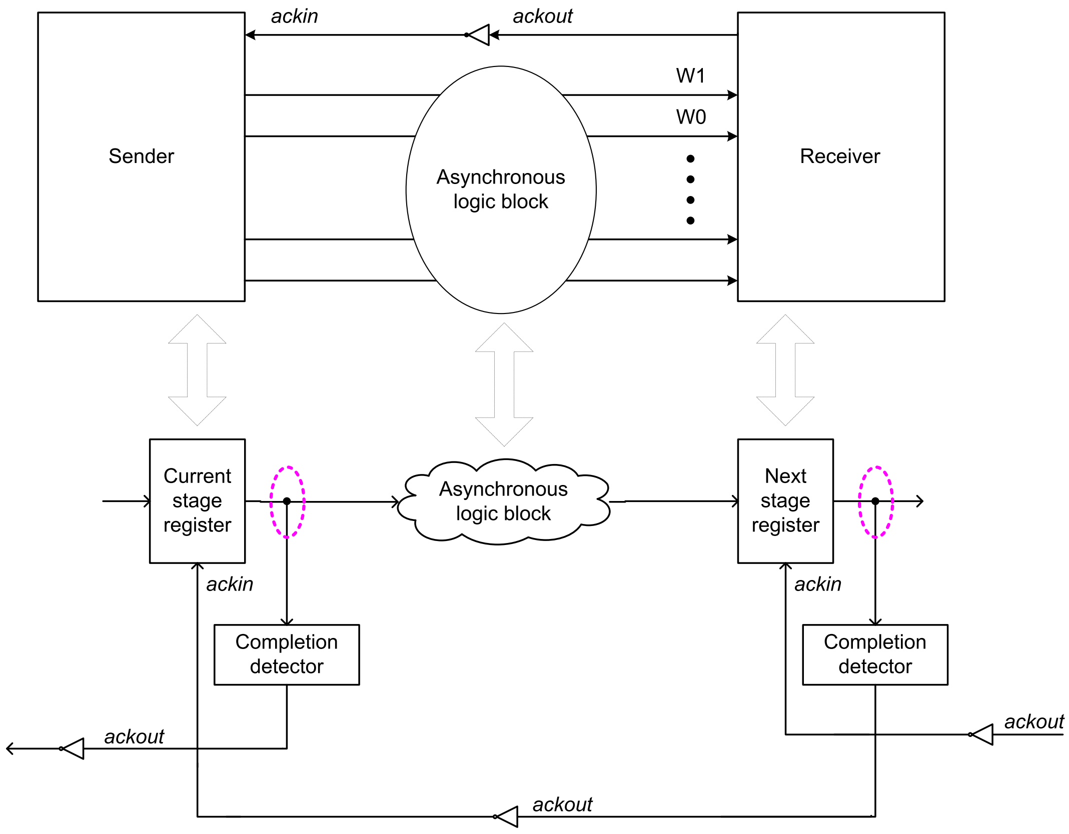

Fig. 1. Block diagram of an asynchronous circuit stage correlated with the sender-receiver analogy for illustration.

The representative block diagram of a typical asynchronous circuit stage is shown in Fig. 1 that is accompanied by the sender-receiver analogy. The valid data and spacer processing operations of an asynchronous circuit stage following the predefined input sequences of valid data-spacer-valid data-spacer and so forth are explained in [18], and the reader is referred to the same for details. In Fig. 1, the junction dots shown enclosed within the pink ovals in dotted lines represent isochronic forks. Isochronicity forms the weakest compromise to delay-insensitivity [20], and an isochronic fork implies that all the nets forking out from the junction tend to experience similar signal transitions occurring concurrently.

Referring to Fig. 1, the 4-phase return-to-zero handshake protocol is explained as follows. The dual-rail data bus that feeds the current stage register (i.e., sender) is initially in the spacer state, and the common acknowledge input viz. *ackin* for the current stage register is binary 1 since the common acknowledge output viz. *ackout* provided by the next stage register (i.e., receiver) is binary 0. The current stage register now transmits a code word which corresponds to valid data. This results in low to high transitions on anyone of the corresponding rails of all the dual-rail bus wires which feed the asynchronous logic block. After the next stage register receives a code word subsequent to completion of data processing in the asynchronous logic block it drives *ackout* to 1, and *ackin* assumes 0. The current stage register waits for *ackin* to become 0 and then resets the data bus i.e., the data bus feeding the asynchronous logic block is driven to the spacer state. After an unbounded but finite and positive amount of time taken for the resetting of the asynchronous logic block and the passage of spacer to the following register stage, the next stage register drives *ackout* (*ackin*) to 0 (1). With this, a single data transaction is said to be completed and the asynchronous circuit is ready to commence the next data transaction.

The timing diagram of the 4-phase asynchronous signaling protocol is shown in Fig. 2 for clarity with the request (*req*) and acknowledge (*ack*) wires explicitly shown to describe the handshaking process. It can be observed that four transitions are required to complete a data transaction based on this signaling protocol and there is an intermediate return-to-zero phase of both the *req* and *ack* wires preceding every transaction since the signaling convention is level-sensitive, i.e., valid data corresponding to logic high viz. binary 1, and the spacer data corresponding to logic low viz. binary 0.

Asynchronous logic blocks are classified as strongly indicating [21,22], weakly indicating [21,23], and early output [24,25] types. Indication in an asynchronous logic block means acknowledging the receipt of primary inputs through the primary outputs whilst involving the intermediate outputs. With respect to an asynchronous circuit stage, the indication mechanism may be local or global; local – if the asynchronous logic block within the asynchronous circuit stage by itself indicates the receipt of all the primary inputs, and global – if the asynchronous circuit stage on the whole indicates the receipt of all the primary inputs along with the asynchronous logic block present in it. It was shown in [26] that local weak-indication is preferable over global weak-indication for asynchronous circuits employing delay-insensitive data encoding and following a 4-phase (return-to-zero) handshake protocol from a combined power-cycle time-area perspective.

The input-output timing correlation of strong-indication, weak-indication, and early output asynchronous logic blocks is illustrated by a representative timing diagram, portrayed as Fig. 3. A strong-indication asynchronous logic block starts to process and produce the required primary outputs only after receiving all the primary inputs whether they are valid data or spacer. A weak-indication asynchronous logic block starts to process and produce