INTEGRATION, the VLSI journal 52 (2016) 34–40

Contents lists available at ScienceDirect

INTEGRATION, the VLSI journal

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

Comments on "Dual-rail asynchronous logic multi-level implementation"

P. Balasubramanian [1]

School of Computer Engineering Nanyang Technological University, 50 Nanyang Avenue, Singapore 639798, Singapore

ARTICLE INFO

Article history:
Received 27 January 2015
Received in revised form
18 August 2015
Accepted 19 August 2015
Available online 29 August 2015

Keywords:
Asynchronous logic
Decomposition
Multi-level implementation
Boolean network
Node

ABSTRACT

In this research communication, we comment on "Dual-rail asynchronous logic multi-level implementation" [Integration, the VLSI Journal 47 (2014) 148–159] by expounding the problematic issues, and provide some clarifications on delay-insensitivity, robust asynchronous logic, multi-level decomposition, and physical implementation.

© 2015 Elsevier B.V. All rights reserved.

1. Introduction

A synthesis method for designing asynchronous logic as a multi-level delay-insensitive Boolean network by employing dual-rail codes and 4-phase handshaking was proposed in [1]. The resulting synthesis solution is said to correspond to a modified weak-indication timing model, which is just an alias for the well known early output logic scheme. It is shown why the synthesis method of [1] is faulty, and describes the many problems inherent therein with respect to delay-insensitive logic decomposition, multi-level synthesis and implementation, monotonic function behavior, and also points out the confusions in the terminologies used in [1]. For the benefit of readers, this paper first discusses essential asynchronous design concepts relevant to the published literature [1], followed by an exposition of the major problematic issues implicit in the synthesis method of [1] through an illustration. Finally, related state-of-the-art multi-level self-timed logic synthesis methods are highlighted for reference.

2. Preliminaries

Asynchronous circuits are primarily classified into two broad types as *bundled-mode* and *input/output mode* [2]. Of these, the input/output mode circuits constitute the robust class since they do not include timing assumptions on when the environment should respond, and hence they are also referred to as *self-timed*[2]*circuits* [2]. On the contrary, bundled-mode circuits inherently incorporate timing assumptions into their normal operation. The following circuit models adhere to the input/output mode:

- Delay-Insensitive (DI)
- Quasi-Delay-Insensitive (QDI)
- Speed-Independent (SI)

The DI model guarantees correct circuit operation irrespective of gate delays and wire delays, i.e. unbounded and arbitrary, but positive and finite gate delays and wire delays are considered. The DI model is the most robust of all the unbounded delay models and DI circuits are guaranteed to be correct-by-construction. It was shown in [3] that C-elements and inverters are the only DI elements and unfortunately, the class of pure DI circuits would be very limited and impractical to realize when engaging only these two logical operators.

DI circuits with isochronic fork assumptions [4] are referred to as QDI circuits. Similar to the DI circuit, the QDI circuit conforms to the unbounded delay model for gates and wires, but with the exclusion of *isochronic forks*. The isochronic fork assumption has been defined in [4] as: "In an isochronic fork, when a transition on one output is acknowledged, and thus completed, the transitions

E-mail address: balasubramanian@ntu.edu.sg
[1] Tel.: +65 6790 6643.

[2] In this paper, the term 'self-timed' implies (quasi) delay-insensitivity. The asynchronous logic described in [1] may be referred by the term 'self-timed logic' in this paper to maintain terminological consistency with other related works.

http://dx.doi.org/10.1016/j.vlsi.2015.08.001
0167-9260/© 2015 Elsevier B.V. All rights reserved.

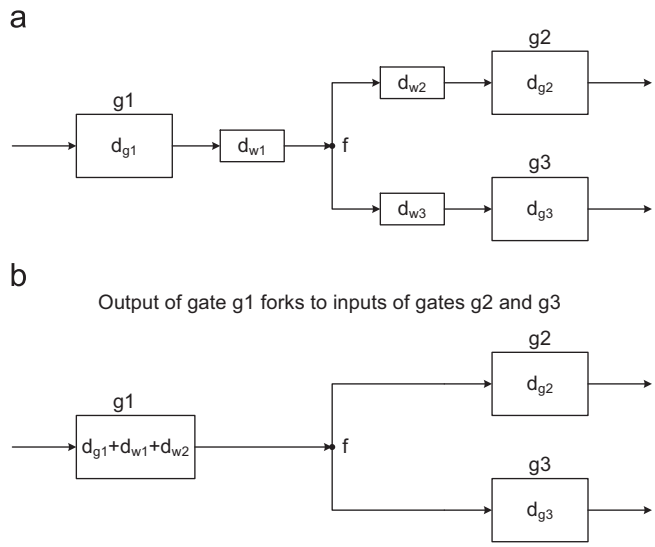

Fig. 1. Illustrating DI, QDI and SI delay models.

on all outputs are acknowledged, and thus completed". Isochronic forks are usually confined to small circuit areas and require careful verification at the layout stage [5]. However, it is not necessary that every fork should be designated as an isochronic fork in a QDI circuit. Martin et al. [6] have showed that the main building blocks of QDI logic including the isochronicity assumption can be successfully implemented even in nanoscale technologies. This is encouraging to note with regards to the feasibility of the QDI style in the nanotechnology era, where stricter design rules and large parametric variations are anticipated [7].

A SI circuit operates correctly regardless of the gate delays; wires are ideally assumed to have zero delay—hence, unbounded gate delays and bounded wire delays are considered. Every fork is treated as an isochronic fork in a SI circuit. Typically, wire delays are accounted for in the delays of logic gates according to the SI model, and as a result wires are assumed to be ideal (i.e., no delay). Technically, QDI and SI circuit implementations look similar in practice [8].

Referring to the circuit fragment in Fig. 1(a), d_{g1}, d_{g2} and d_{g3} signify the propagation delays of gates g1, g2 and g3 respectively, while d_{w1}, d_{w2} and d_{w3} represent the delay values of the corresponding nets w1, w2 and w3. For the DI model, the values of d_{g1}, d_{g2}, d_{g3}, d_{w1}, d_{w2} and d_{w3} can be arbitrary, while in the case of QDI model; d_{w2} is assumed to be equal to d_{w3} with the node 'f' being labelled as an isochronic fork junction. Under the SI model, $d_{w1}=d_{w2}=d_{w3}=0$, but the wire delays are accounted for in the delay of the gate g1, whose output acts as inputs for gates g2 and g3. Hence, the delay of gate g1 can be modelled as $d_{g1}+d_{w1}+d_{w2}$ or $d_{g1}+d_{w1}+d_{w3}$ as shown in Fig. 1(b).

The robustness attribute of asynchronous/self-timed designs usually results from employing a DI code for data representation, communication and processing; and the 4-phase return-to-zero (RTZ) protocol for handshaking. The dual-rail code is the simplest member of the generic family of DI m-of-n codes [9], where m lines are asserted 'high' out of a total of n physical lines to represent a codeword. The size (i.e. number of unique symbols) of a m-of-n code is given by the binomial co-efficient, n choose $m=n!/m!(n-m)!$. Among the family of DI codes [9], the dual-rail (1-of-2) code is widely preferred owing to its simplicity, ease of logic implementation, and convenient mapping with binary data. According to dual-rail data encoding, each data wire d is represented using two encoded wires as $d^0/d(0)$ and $d^1/d(1)$, as shown in Fig. 2. A transition on the d^0 wire indicates that binary 0 has been transmitted, while a transition on the d^1 wire indicates that binary 1 has been transmitted. Since the request signal is embedded

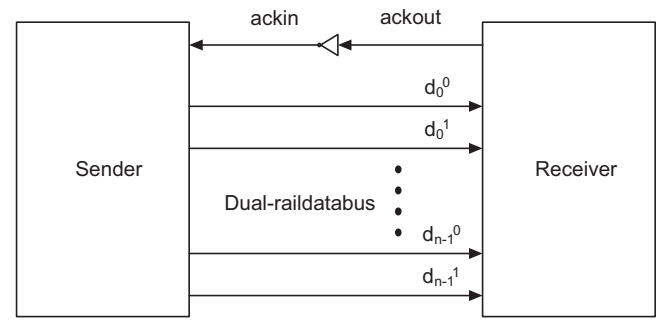

Fig. 2. DI dual-rail data encoding and 4-phase RTZ handshaking.

within the data wires, a transition on either d^0 or d^1 informs the receiver about the validity of the data (i.e., *valid data*). When both d^0 and d^1 assume binary 0 at the same time, it is referred to as the *spacer*. However, d^0 and d^1 are not permitted to simultaneously transition to binary 1, as it is illegal and invalid, since the coding scheme adopted is *unordered* [10], where no codeword should form a subset of another codeword.

With reference to Fig. 2, the 4-phase handshaking protocol is explained as follows[3]:

- The dual-rail data bus is initially in the spacer state. The sender transmits the codeword (i.e., valid data). This results in low to high transitions on the bus wires (i.e. any one of the rails of all the dual-rail signals is asserted as binary 1).
- After the receiver receives the codeword, it drives the 'ackout' ('ackin') wire to binary 1 (binary 0).
- The sender waits for the 'ackin' signal to become 0 and then resets the data bus (i.e. the data bus is driven to the spacer state).
- After an unbounded but a finite, positive amount of time, the receiver drives the 'ackout' ('ackin') wire to binary 0 (binary 1). A data transaction is now said to be complete, and the system is ready to proceed with the next transaction.

The timing diagram for the 4-phase asynchronous signalling protocol is shown in Fig. 3, with the request (*req*) signal, which is actually embedded within the data wires, explicitly shown to describe the handshaking.

Unlike a conventional combinational logic circuit, a self-timed function block is not only expected to produce the desired output (s) for the corresponding input(s), but also should indicate the completion of computation on all its internal nodes [2]. In other words, the outputs of a self-timed function block tend to indicate the arrival of all the inputs and the completion of internal data processing. Self-timed/input-output mode asynchronous circuits can be classified into 3 types as *strongly indicating*, *weakly indicating*, and *early output*.

2.1. Strong-indication

A strong-indication self-timed circuit waits for all the valid/spacer inputs to arrive and then starts to produce the required valid/spacer outputs. The sequencing constraints [11] are stated as:

- All the inputs attain valid/spacer state before any output attains valid/spacer state.
- All the outputs should have attained valid/spacer state before any input attains spacer/valid state respectively.

[3] The explanation remains valid for data representation using any DI data encoding scheme.

Download English Version:

https://daneshyari.com/en/article/542678

Download Persian Version:

https://daneshyari.com/article/542678

Daneshyari.com