Revisiting Sorting Network based Completion Detection for 4 Phase Delay Insensitive Codes

Florian Huemer, Markus Schütz, Andreas Steininger
TU Wien, Institute of Computer Engineering
Treitlstrasse 3, 1040 Vienna, Austria
{florian.huemer@student, markus.schuetz@student, steininger@ecs}.tuwien.ac.at

***Abstract*—Completion detectors (CD) are key components in delay insensitive asynchronous circuit design. Their task is to check whether received data is complete and valid and to inform subsequent logic of this condition. Hence, it is very important to implement CDs in a resource-efficient way. One way to achieve this goal is to make use of binary sorting networks (SN). This work analyses and extends this approach. We show which constraints in form of timing assumptions are necessary if existing SN based solutions are used. Furthermore, modifications to the existing solutions are proposed to obtain quasi delay insensitive (QDI) circuits with minimum overheads. In particular, this work elaborates generic design templates for CDs for 4-phase m-of-n, incomplete m-of-n, Berger and Zero-Sum codes.**

***Keywords*-asynchronous circuits; completion detection; sorting networks; QDI**

I. Introduction

Modern CMOS ASIC technologies considerably suffer from significant parameter variations that originate in process inaccuracies, as well as strong dependencies on supply voltage and temperature (called PVT variations). Timing paradigms that rely on a rigid time grid, like the synchronous approach, need to make worst case assumptions on these parameters and hence end up with a painful gap between guaranteed worst case specifications and actual average case performance that can, unfortunately, not be leveraged. This brings self-timed circuits, often also called "asynchronous logic" into the focus. Here the temporal control is based on explicit handshaking between sender and receiver of a data item, which ultimately establishes a closed-loop control that can adapt the speed of operation to the actual conditions, even dynamically. Among the available techniques, the delay insensitive paradigm is the most flexible one with respect to PVT variations. It is based on a specific delay insensitive encoding of the data which allows the judgment of their validity at the receiver. For this task, special "completion detection" circuits are employed. Once they identify, based on an analysis of the received code word, validity of the data word, they trigger the capturing of that data item at the receiver. In a next step the receiver confirms the reception to the sender by virtue of a dedicated acknowledge signal ACK, which closes the control loop.

Obviously, the completion detectors (CDs) are core elements within the delay-insensitive paradigm, and their implementation can require a significant share of the overall area. Furthermore, it is non-trivial to sustain the delay-insensitive design style for their implementation as well. It is the key contribution of this paper to thoroughly investigate the use of sorting networks (SNs) as a means for efficient implementation of CDs. While this has been proposed in other works, the novelty here is to elaborate an implementation that achieves full (quasi) delay-insensitivity in the most efficient way, i.e. without being overdesigned. As a result our approach provides a very generic template for implementing CDs for delay insensitive systems (we restrict our discussion to the important class of 4-phase protocols) that can be immediately used by an ASIC designer.

After giving background and related work for completion detection and timing models in Section II and SNs in Section III, we will present our solutions for the different codes in practical use: Section IV will be devoted to m-of-n codes, Section V to the special class of incomplete m-of-n codes, and finally Section VI to Berger and Zero-sum codes. Section VII will compare the proposed approach to another generic CD architecture. Finally, Section VIII will conclude the paper.

II. Background and Related Work

A. Completion Detection

Delay insensitive (DI) return-to-zero (RZ) or four-phase protocols always separate two successive valid code words (data phase) by a zero or null phase, which does not carry any information. This spacer is usually encoded by logical zeros on all rails. Verhoeff [1] shows that a DI code has to be unordered. This means it is not allowed for any valid code word to be contained in any other valid code word. In this case "contained" denotes the situation, where the positions of the ones in a code word are a subset of the positions of the ones in another code word. Examples for unordered codes are m-of-n, Berger and Zero-Sum [2] codes.

As an essential part of any asynchronous DI system (e.g. a pipeline) the CD is responsible to detect valid data and null phases. To ensure correct operation of the RZ handshaking protocol the CD must satisfy some basic properties [3] [4]. The CD is attached to the n input data rails of a DI data

path C and generates the signal $done$, indicating whether the information carried by these rails contains a valid data word or the spacer. In the beginning the system is in its null phase, i.e. all (n) input rails of C as well as the $done$ output are zero. Now the environment applies a code word to the DI channel C. Note that, as a consequence of the DI property, the transitions may arrive at the inputs of the CD in any order. However, once a rail made a transition it is not allowed to change its state again until the next null phase. In other words, no rail may ever glitch. As soon as the CD detects a valid code word it asserts the $done$ signal, which eventually causes the environment to reset the input to the null phase. The signal $done$ must retain asserted until the CD detects the spacer. After the de-assertion of the $done$ signal the whole process starts over again.

While completion detection for simple 1-of-n codes (like dual rail) can be implemented very efficiently by an n-input OR gate, more complex codes like m-of-n or Berger require more elaborate circuits. The DIMS [5] design style allows the construction of CDs for arbitrary DI codes in a generic way. For this purpose DIMS uses an array of C gates[1] to exclusively map every possible (valid) input code word to a dedicated signal (one-hot code). In a second stage a wide OR gate joins the outputs of these C gates to produce the $done$ output of the CD. Obviously, circuits constructed with this technique can easily grow very large and inefficient. To overcome this problem, Piestrak [3] proposed an approach to CD utilizing binary SNs. His findings serve as the base for the circuits investigated in this work and will be discussed in further detail in the following sections.

B. Timing Model

Asynchronous circuits can be classified with regard to the imposed timing and delay assumptions. The class of delay insensitive circuits uses the weakest timing assumptions. The only restriction on gate and wire delays is that they have to be positive and finite. However, as shown by Martin [6], the class of circuits which can be constructed with this timing model is very small. This is because the only gates that can be used in DI circuits are inverters and C gates.

To overcome these limitations, isochronic forks [7] are introduced, leading to the class of quasi delay insensitive (QDI) circuits. With this extension to the DI timing model, arbitrary circuits can be constructed, which leads to a wide adoption of the QDI model in practical applications. Figure 1 shows a model of a wire fork. In DI circuits there is effectively no restriction on the delays Δ_1, Δ_2 and Δ_3 (just bounded and positive). However, in QDI circuit the isochronic fork property requires that $\Delta_2 = \Delta_3$. A detailed discussion of isochronic forks is given in [6] and [8].

[1]The Muller C-element, further called C gate for brevity, is a very fundamental basic gate in asynchronous logic. Its function is to output the logic level seen at its inputs when these match, and to retain the last valid output state otherwise.

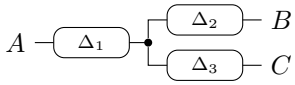


Figure 1. Wire fork model

An important problem that can arise in QDI circuits are the so called orphan transitions [9]. Every (QDI) function block consists of a number of gates which process the bit pattern presented at its inputs and in this way generate the result pattern which is in turn presented at the outputs. At this point it is important to recall that the timing is controlled by a closed control loop: As soon as the input has been properly processed and the output result been received by the successor stage(s), the sender is notified (by the ACK signal) that it can apply the next data item. In this scheme, the availability of the result at the function block output implies that all internal processing has been completed.

This implication becomes problematic, if there are still activities (transitions) going on in the function block *although* the result is already valid. Such transitions that do not have an effect on the output are called orphan transitions. They may occur if there is a gate inside the circuit that switches on a particular input pattern, but whose output is masked or ignored for the given input pattern. In the presence of orphan transitions it is not possible to decide just by observing the outputs, whether the circuit has completed its operation and is ready to proceed with new input data or if orphan transitions still have to be awaited. Considering that in a QDI design the gate producing the orphan transition may have arbitrary delay, it becomes clear that this may cause timing problems, as the orphan transition may interfere with the subsequent data input. Consequently such transitions can lead to a malfunction of QDI circuits and must be considered during the design process – also, and most notably, for the CD implementation.

III. Binary Sorting Networks

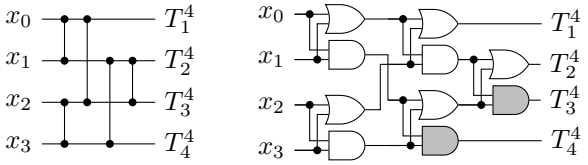
Generally speaking, sorting networks (SN) are used to sort numbers. Their basic building block is the comparator cell, such as the one shown in Figure 2a. A comparator cell has two input (a, b) and two output lines, which hold the maximum and minimum of the inputs, respectively. A binary sorting network can only distinguish two numbers (zero and one). It converts a binary input vector to a sorted binary output vector (e.g. 10101 $\mapsto$ 11100), effectively counting the number of bits set in the input vector. Figure 2b shows the corresponding comparator used in binary SNs. Note that this is only a representation of the logical function, and an efficient (1-stage) CMOS implementation based on NAND and NOR is straightforward to find.

Figure 3 shows how these basic comparator cells are used to form such networks. Figure 3a shows the common representation of a SN with 4 input lines, while Figure 3b shows the gate-level implementation of the corresponding

(a) comparator (b) binary comparator

Figure 2. Comparator cells

binary SN. We use the same notation as in [3]. A binary SN T^n has n inputs $(x_0, ..., x_{n-1})$ and n outputs $(T_1^n, ..., T_n^n)$. The output T_m^n is set if at least m inputs are set to one. For the case of the binary T^4 this means that all the input vectors 1100, 1010, 1001, 0110, 0101, 0011 yield the same result on the output, namely 1100.



(a) T^4 (b) binary T^4

Figure 3. T^4 sorting networks

A detailed discussion of SNs in general is beyond the scope of this work. We refer to Knuth [10] who provides a good introduction to the field and shows efficient implementations of SNs with up to 16 inputs. The size of a SN is measured by the number of comparators used. The problem of designing an optimal sorting network for an arbitrary number of inputs is still open. However, it has been shown that the size of a SN with n inputs (denoted by $S(n)$) is lower bounded by $O(n \log(n))$. The best known algorithm for constructing SNs [11] needs $O(n \log(n)^2)$ comparators.

Table I shows the implementation costs of the best known SNs for a particular n [10]. $T(n)$ denotes the depth of the SN, i.e. the maximum number of comparators (gates) an input signal has to pass as it travels through the network. Since this number directly corresponds to the delay of the SN, we have selected those networks with a minimal depth in favor of networks with a minimal number of comparators.

Table I
SN IMPLEMENTATION COSTS (MINIMAL DEPTH)

n	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$S(n)$	3	5	9	12	16	19	25	31	35	40	47	52	57	61
$T(n)$	3	3	5	5	6	6	7	7	8	8	9	9	9	9

IV. M-OF-N COMPLETION DETECTION

Note that the information produced by a binary SN is exactly what is needed to perform a membership test for an m-of-n code. However, a binary SN (such as the one shown in Figure 3b) is not, by itself, a QDI circuit and thus not yet a complete and functional CD. A "bare" SN does not exhibit hysteresis behavior in the reset phase, which would lead to a violation of the 4-phase protocol. Even if every AND gate in the circuit would be replaced by a C gate, as suggested in [3], there still remains the problem that not all transitions inside the SN are observable (at the output) for every input vector. Hence the circuit contains orphan transitions that can lead to a violation of the RZ protocol. For the special case of 2-of-n codes, [4] shows, that a slight extension to the SN, in form of a C gate at the outputs T_1^n and T_2^n, is sufficient to make the SN QDI and use it as CD for such codes.

We claim that this also holds in the general case, where m ones should be detected on n input rails. First of all, we want to recall that we are talking about *QDI* circuits. That means that it is necessary that all forks inside the network are isochronic. This constraint is essential because the arguments presented here are not valid otherwise.

In order to use a binary SN T^n as CD for an m-of-n code, it must be possible to detect the following two conditions.

$$\bigwedge_{1 \le x \le m} T_x^n = 1 \tag{1}$$

$$\bigvee_{1 \le x \le m} T_x^n = 0 \tag{2}$$

Equation 1 specifies the condition that must be fulfilled before the $done$ output of the CD can be switched to one (data phase). For the reset phase (i.e. the de-assertion of the $done$ signal) Equation 2 must be satisfied. This behavior can be implemented by an m-input C gate.

When the conditions described above are satisfied and the $done$ output of the CD changes its logical state there must not be any orphan transitions left inside the SN (as outlined in Section II-B). How can the absence of orphan transitions be guaranteed?

Assume a comparator cell with both inputs set to zero. If a one is applied to one of the inputs, the output of the OR gate will eventually switch to one. A second one at the other input triggers the AND gate, switching the second output to one as well. So a very fundamental property of the comparator cell, that can be observed here, is that a rising edge on one of the inputs always leads to a rising edge on one of its outputs. Note that only *one* output will switch, while the other will *not* switch for sure. Now we can leverage the isochronic fork constraint: When looking at the forks in Figure 2b, we can infer that once one of the gates has switched its output, the other one must have received its inputs as well, so no transitions can be on the way at that point. And since we know that the other gate will not switch anyway, it cannot have any operation in progress, no matter how long its delay might be.

For the reset phase the behavior of the comparator is complementary. Every falling edge on one of the inputs eventually leads to a falling edge on one of the outputs, and again, once an output fires we know, due to the isochronic fork constraint, that there a no other transitions on the way.

So as a first conclusion we can state for our comparator cell that once we see an output switch in reaction to an input transition we can safely conclude the circuit is stable.

Note that the non-switching gate would be a candidate for an orphan transition in the general case, and only by leveraging our knowledge on the mutual exclusive switching behavior of the gates in combination with the isochronic fork assumption we have been able to rule out this threat (on the cell level).

The structure of SNs ensures that every input to the circuit as well as every output of a gate inside the circuit is connected to exactly one comparator, unless it is a primary output. Knowing that a comparator always responds to an input transition at its output, this implies that every transition applied to any of the primary inputs of the circuit, eventually leads to a transition on one of the primary outputs – this is exactly what we need for a delay insensitive behavior.

With this knowledge we can prove that an m-input C gate at the outputs T_1 to T_m of a T^n SN is sufficient for use as CD of an m-of-n code. Assume that an output condition (as in Equation 1 or 2) is fulfilled and that there are orphan transitions left inside the circuit. This assumption immediately leads to a contradiction, because if there were transitions in the circuit, they would eventually be visible at the output. However, by the initial assumption the output condition is already fulfilled, which means that all transitions must have passed through the whole network completely. Therefore, the output conditions could not already be fulfilled, if there are pending transitions inside the circuit.

Another problem that needs to be addressed, is whether or not SNs can produce glitches at their outputs. For a glitch to ever occur, we must see contradicting transitions (falling and rising) at one of the basic gates, or glitches at the input already. Due to the RZ protocol, the inputs will always see (non-glitching) all rising or all falling (or no) transitions, depending on the phase (data or null). In response, the comparator outputs will also produce all rising or all falling transitions (if any). So this wave of unidirectional transitions propagates throughout the whole SN without any comparator ever seeing contradictory transitions in one phase (note that the phases are well mutually separated by handshaking during which the circuit is stable, as we do not permit orphan transitions). This reasoning also holds for NAND and NOR based implementation, with the waves simply changing polarity from stage to stage.

Figure 4 shows a CD for the 2-of-4 code, using a T^4 SN. Since for a legal code word no more than two rails ever switch to one, the outputs T_3^4 and T_4^4 are not needed and can therefore be removed form the circuit. Hence, the gates that drive these outputs can also be removed (marked gray in Figure 3b).

Another optimization that can be applied to SNs used in m-of-n CDs, results from the following observation: If there is a comparator cell where, for every valid input code word, both inputs eventually switch to one then this cell can be removed from the network.

If the C gate at the output of the SN in Figure 4 is replaced

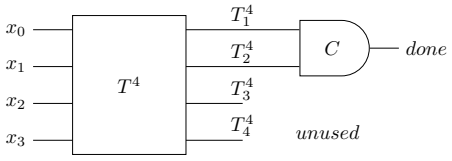


Figure 4. 2-of-4 sorting network based completion detector

by multiple cascaded C gates, as shown in Figure 5, then a QDI mutli-output threshold circuit (QMOTC) is obtained. Note that, since [3] uses the term MOTC to denote a "bare" SN, we refer to our modified version as QMOTC. The QMOTC behaves similar to a SN based CD. The outputs d_1 to d_n are set according to the number n of input rails (x_0 to x_{n-1}) set to one. To reset the outputs, all input rails must be set to zero. The C gates ensure that if the output d_n switches, the outputs d_m where $m < n$ must have already switched to the same value. We will use the QMOTC to construct and improve CDs presented in the following sections.

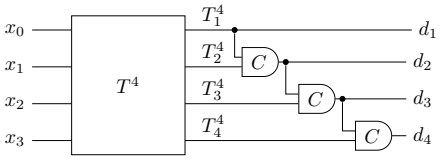


Figure 5. QDI multi-output threshold circuit

V. Incomplete m-of-n Completion Detection

The concatenation of DI codes is also a DI code, with a size given by the product of the individual codes' sizes. For instance, the concatenation of two 2-of-4 codes, with individual sizes of 6, results in a code capable of representing 36 symbols. This allows to custom-tailor codes with a given size without having to resort to the next larger "regular" m-of-n code and pay its higher CD complexity. By permitting the movement of ones between the code groups, i.e. between the individual sub codes, the size of the resulting code can be increased and hence further adapted. Using this technique, the total number m of ones is not increased, which means that the resulting code is still unordered and therefore DI. Note however that, if arbitrary movements of ones would be allowed, we obtain an ordinary m-of-n code, which nullifies the complexity reduction we strived for. So in the construction of incomplete m-of-n codes (denoted by m-of-n*), as proposed by Bainbridge et. al. [12], one carefully restricts the exchange of ones to find an optimum balance between code size and CD complexity.

As further discussed in [12], such codes can be constructed by selecting one code group to be the control group, which is allowed to "donate" and "adopt" ones to and from other code groups (the body groups). Table II shows this approach for the example of the 4-of-8* code, which has a size of 68 and is hence able to encode 6-bit data words. Compared to the complete 4-of-8 code the 4-of-8* code does not comprise the code words 11110000 and 00001111.

Table II
GROUPS OF THE 4-OF-8* CODE

control	body	symbols
3-of-4	1-of-4	4x4
2-of-4	2-of-4	6x6
1-of-4	3-of-4	4x4
		68

Figure 6 shows our proposed SN based CD for the 4-of-8* code. To "count" the number of ones in each code group, QMOTCs are employed. Every valid combination of control and body is then detected by a dedicated C gate. C_{13}, for example, detects the case where there is one rail set in the control group and three rails in the body group. The final result is obtained by an OR gate combining the outputs of the C gates. The overall implementation costs for

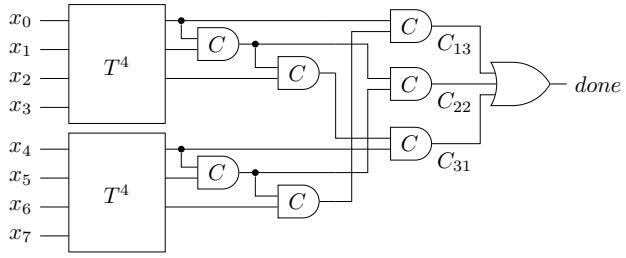


Figure 6. 4-of-8* completion detector

this CD sum up to 172 transistors, while a SN based CD for the complete 4-of-8 code requires 208 transistors. This equals a 17% saving for a 3% reduction in code size (that is unused anyway when encoding 6-bit data). This approach is basically applicable to every incomplete m-of-n code.

VI. BERGER/ZERO-SUM COMPLETION DETECTION

Berger and Zero-Sum [2] codes are both systematic and unordered. Systematic codes consist of a data and a check part. The data part d contains the binary (unencoded) representation of the data, while the synchronization part $c = sync(d)$ is used to make the code unordered (hence DI). Berger codes use the binary representation of the number of zeros in the data part as check bits (synchronization part). Zero-Sum codes additionally associate a weight to every (bit) position in the data part. The synchronization part is then calculated by the sum over the weights of the positions which are zero in the data part.

Figure 7 shows the basic structure of the Berger code CD proposed by Piestrak [3]. The data part of the input vector is denoted by x, while the synchronization part is denoted by c. The data part x is fed into a threshold circuit T^m, which effectively outputs its Hamming weight (i.e. the number of rails set to one). The Unate Products Generator (UPG) is connected to the k rails of the synchronization part c and produces m output signals w_1 to w_m. The output w_i is generated by a conjunction over those rails of c, which must be one if c carries the binary representation of i (assuming c_0 is the LSB). For example, w_5 is generated by a C gate with inputs c_0 and c_2.

The basic idea behind the circuit is that for a certain number j of rails set in the data part there must be a corresponding w_{m-j} generated from the synchronization part. The output network, uses $m-1$ 2-input C gates to detect these conditions. The outputs of these C gates are joined by an OR gate, which generates the $done$ output of the CD. For the cases, where the data part consists of all zeros or ones it is sufficient to check solely the signals w_m and T_m^m, respectively. Hence, these signals are connected directly to the output OR gate. Note, that it can never happen, that more than one of the C gates in the output network (and the signals w_m and T_m^m) are active at the same time. To apply

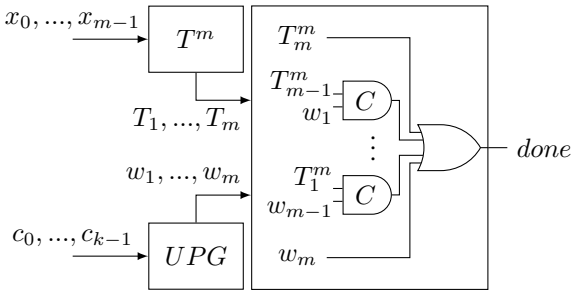
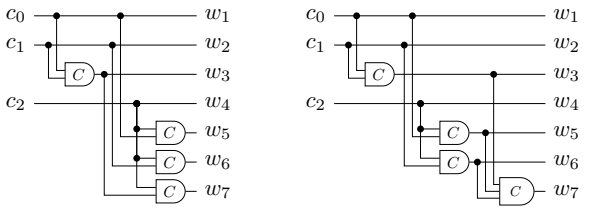


Figure 7. Completion detector for Berger codes [3]

the described CD technique to Zero-Sum codes, the different weights of the data rails (x) must be taken into account. A data rail with weight r must therefore be connected to exactly r inputs of the SN. The size (number of inputs) of the SN is hence given by the sum over the weights of all data rails.

Although the presented approach allows the efficient implementation of CDs for Berger codes, the resulting circuits need slight modifications to fulfill the QDI requirements. If a "bare" SN is used to implement the threshold circuit (i.e. without the cascaded C gates at the outputs), as originally proposed by [3], problems with orphan transitions arise. Consider a CD for a Berger code with 4 data and 3 check bits that uses the T^4 SN (Figure 3b). Now a code word is applied that has two rails set to one in the data part (e.g. $x = 0110$). Hence the synchronization part c must be 010. Eventually ones will emerge on the intermediate signals w_2 and T_2^4, activating the corresponding C gate in the output network, which in turn causes the $done$ output to switch to one. Note that, since two rails of x are set, T_1^4 also switches to one, which represents an orphan transition in this case. In the worst case, the environment of the CD proceeds with the handshaking protocol and a late transition on T_1^4 leads to a malfunction in the following data phase.

There are basically two approaches to overcome this potential problem. Firstly, additional timing constraints can be introduced, which ensure that the described scenario can never happen. The other option, which we are proposing



(a) UPG (b) modified UPG

Figure 8. Unate Product Generators

here, is to use the QMOTC discussed in Section IV to make T_2^4 depend on T_1^4 and thus effectively make the circuit QDI again. However, on the downside, the additional C gates significantly increase the overall delay of the circuit.

A similar problem arises with the UPG. Figure 8a shows the proposed UPG for a Berger code with three check bits. If the code word with the synchronization part $c = 111$ arrives at a CD using this UPG, we have orphan transitions at the gates driving signals w_3, w_5 and w_6. Figure 8b shows a modified version of the UPG which solves this problem. Here the signal w_7 is derived from the signals w_3, w_5 and w_6, which introduces an appropriate causality between the transitions. Note, that this holds for both the set and reset phase of the protocol. The modified UPG does only require a few more transistors. With respect to the overall delay of the CD, the added level of logic (2 vs. 3 gates) should not be a problem either, since the threshold circuit has a greater depth, anyway. However, like before, one can choose between this approach and the introduction of timing constraints.

VII. Comparison with DIMS

Although a detailed comparison can be made case by case only, we will try to give some general aspects here.

A fundamental disadvantage of DIMS seems to be the need for a separate C gate per valid code word. For an m-of-n code this results in $\binom{n}{m}$ C gates with m inputs each – a huge effort and bad scalability. So area-wise the SN approach with its $O(n \log(n)^2)$ growth is clearly superior.

With respect to speed the parallel arrangement of DIMS appears to outperform the more serial SN approach. However, the large number of C gates in DIMS implies equally large forks for the inputs, which causes high capacitive loading and complex routing (which, in turn, makes fulfilling "isochrony" difficult). In addition multi-input C gates and OR gates ultimately require a cascaded (i.e. multistage) implementation. So overall the timing of those complex DIMS circuits tends to become cumbersome. In contrast, as Table I shows, the SN solution proves to come along with a moderate depth, even for 16 bit.

VIII. Conclusion

We have presented implementation templates for CDs in 4-phase QDI designs. These templates are very area efficient as they are based on the use of SNs. Moreover, while providing comparable speed, their scalability towards higher data width is much better than that of related approaches, like DIMS. Our careful analysis of potential orphan transitions has allowed us to achieve full QDI property with minimum efforts, which is another improvement over existing solutions. We have elaborated generic solutions for m-of-n codes, incomplete m-of-n codes as well as Berger and Zero-sum codes, thus covering all relevant DI codes. As a result we can provide a portfolio of reliable and efficient templates that can be used by an ASIC designer out of the box and just scaled to the needs of the given application, thus simplifying a critical part in the design of self-timed systems.

Our future work will be directed towards also covering 2-phase protocols. Furthermore we will also look into improvements of the codes as such, especially with respect to adding fault tolerance to their DI property.

References

[1] T. Verhoeff, "Delay-insensitive codes - an overview," pp. 1–8, 1988.

[2] M. Agyekum and S. Nowick, "Error-correcting unordered codes and hardware support for robust asynchronous global communication," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 1, pp. 75–88, Jan 2012.

[3] S. Piestrak, "Membership test logic for delay-insensitive codes," in *Fourth International Symposium on Advanced Research in Asynchronous Circuits and Systems*, 1998, pp. 194–204.

[4] M. Cannizzaro, W. Jiang, and S. Nowick, "Practical completion detection for 2-of-n delay-insensitive codes," in *2010 IEEE International Conference on Computer Design (ICCD)*, 2010, pp. 151–158.

[5] J. Sparso and J. Staunstrup, "Design and performance analysis of delay insensitive multi-ring structures," in *26th Hawaii International Conference on System Sciences*, vol. i, Jan 1993, pp. 349–358.

[6] A. J. Martin, "The limitations to delay-insensitivity in asynchronous circuits," in *Proceedings of the Sixth MIT Conference on Advanced Research in VLSI*, ser. AUSCRYPT '90. Cambridge, MA, USA: MIT Press, 1990, pp. 263–278.

[7] K. van Berkel, "Beware the isochronic fork," *Integr. VLSI J.*, vol. 13, no. 2, pp. 103–128, June 1992. [Online]. Available: http://dx.doi.org/10.1016/0167-9260(92)90001-F

[8] R. Manohar and Y. Moses, "Analyzing isochronic forks with potential causality," in *Asynchronous Circuits and Systems, 2015. ASYNC '15. 21st IEEE Symposium on*, May 2015, pp. 217–226.

[9] A. Kondratyev, L. Neukom, O. Roig, A. Taubin, and K. Fant, "Checking delay-insensitivity: 104 gates and beyond," in *Eighth International Symposium on Asynchronous Circuits and Systems*, April 2002, pp. 149–157.

[10] D. E. Knuth, *The Art of Computer Programming, Volume 3: (2Nd Edition) Sorting and Searching*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1998.

[11] K. E. Batcher, "Sorting networks and their applications," in *American Federation of Information Processing Societies: AFIPS Conference Proceedings: 1968 Spring Joint Computer Conference, USA*, May 1968, pp. 307–314.

[12] W. Bainbridge, W. B. Toms, D. Edwards, and S. Furber, "Delay-insensitive, point-to-point interconnect using m-of-n codes," in *Ninth International Symposium on Asynchronous Circuits and Systems*, 2003, pp. 132–140.