Forward and Backward Guarding in Early Output Logic

Charlie Brej and Doug Edwards
School of Computer Science, The University of Manchester,Oxford Road, Manchester, M13 9PL, UK
Email: {cbrej,dedwards}@cs.man.ac.uk

***Abstract*—Quasi Delay Insensitive asynchronous logic is a very robust system allowing safe implementations while requiring minimal timing assumptions. Unfortunately the design methodologies using this system have always yielded very slow designs. Early output logic is a method which aims to improve the performance of QDI circuits without decreasing their robustness.**

In order to force QDI restrictions on early output circuits a form of guarding is necessary. This paper presents a new form of guarding which allows partial stage completion allowing desynchronisation of inputs. This is shown to be highly advantageous in cases where the previous style performed poorly. Because the two styles can be mixed, the designs no longer suffer from very poor performance of some QDI constructions.

I. Introduction

Unlike synchronous circuits which heavily rely on timing assumptions, some forms of asynchronous circuits can reduce the timing assumptions to an easily met set, even on a very inconsistent manufacturing process [1]. The removal of timing assumptions can also increase the speed of the cir- cuit as no timing margins have to obeyed. Unfortunately this comes at a price in area, power consumption and speed beyond what is gained by the removal of timing margins.

A. QDI logic

Asynchronous circuits are grouped in classes representing their robustness. The Delay Insensitive[1] (DI) class is the most robust class but its restrictions make it impossible to build computing circuits (circuits with data dependant functionality). The class with the smallest compromise in robustness is called Quasi Delay Insensitive [3](QDI). QDI allows isochronic forks[4] which in turn allow the generation of computing circuits. These circuits are still heavily restricted in their operation and methods such as dual-rail encoding must be used to communicate data in a delay insensitive manner.

Dual-rail encoding uses two wires to represent a single binary value. A transition on each wire (labelled zero and one) signifies a transmission of a data value. Each transmission is responded to with a transition on an acknowledge wire back to the data source. These transmissions of data, separated with an acknowledge transition, can be thought of as tokens. To make computing components easier to implement, both the data and the acknowledge signals can be returned to zero between transmissions. This protocol is often referred to as four-phase or return-to-zero.

B. DIMS logic

A common method of generating QDI asynchronous circuits is Delay Insensitive Minterm Synthesis [5] (DIMS). DIMS gates and latches use the four-phase (return to zero/null) dual-rail protocol. DIMS gates, like one shown in figure 1, rely on representing the input state space in a set of ‘one hot’ minterms (generated by a set of C-elements[5]) and allow each output wire to gather its set of minterms using OR gates. One, and only one, C-element in the gate will activate once all inputs become valid. The output will then become valid and will remain so until all inputs have transitionned back to null. The output is only generated once all inputs are valid and released only once all inputs have returned to null. This strong sequencing ensures that an acknowledge will only arrive to gates which have presented data and is only released once all the inputs have accepted the acknowledge (and released their data).

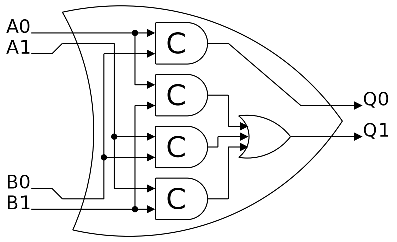

Fig. 1. DIMS Gate

C. Early output logic

The DIMS gate is both large and slow due to the restriction that no output is generated before all inputs have arrived (or released before all inputs have been released). This forces the gate to use large and slow C-elements and stops it from outputting data known to be correct with just a subset of inputs present.

Early output gates [6] do not possess this strong sequencing property and instead only perform the logical part of the operation. An example gate, shown in figure 2, is constructed from an AND gate and an OR gate. This construction allows the generation of outputs once a sufficient number of inputs has arrived. In the case of the early output OR gate pictured, the Q1 output is activated when either A1 or B1 is active. These early output states allow the gate to perform at average rather than worst case timing.

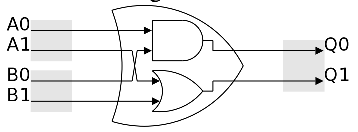

Fig. 2. Early Output Gate

D. Hazards

Early output logic is faster and smaller than DIMS logic but this is achieved only by removing the strong sequencing, of the gates, used to ensure all inputs have contributed to the operation. Early output gates cannot be used as a direct replacement for DIMS gates as acknowledge signals can propagate to the input latches before they have presented data. To ensure all input latches are ready to be acknowledged, early output logic relies on ”guarding logic”[7][8][9].

Guarding logic is used in early output designs to ensure the correct sequencing of the request and acknowledge signals on input latches. The three variations of guarding logic presented in this paper rely on a system of ‘validity’. Validity represents the permission to propagate the acknowledge signal. Latches generate their validity once they have data which can be acknowledged. This can be done by using the OR of their data output pair. In most latch designs (such as the half latch in figure 3), this signal is already available and no additional logic needs to be added. The latch has to wait for the validity signal to arrive on its input before acknowledging. This is done with a C-element combining the latch's internal acknowledge with the validity to generate a guarded validity which is safe to send back to the input latches.

The simplest guarding style collects the validity signals, from all gate inputs, in a C-element and generates the gate's validity. This will protect against latches receiving their acknowledge before asserting data, but the resultant circuit will not be QDI.

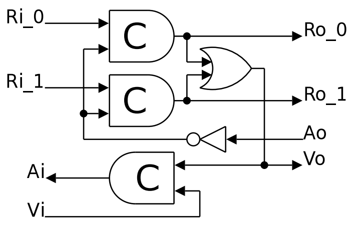

Fig. 3. Early Output Latch

II. Forward Guarding

In order to create fully QDI early output circuits, the method of guarding used must ensure all data signal pairs have returned to zero (null) before the next data cycle can begin. To ensure a full reset has been achieved, each data wire pair must be fully cycled to a valid state and then back to null.

One method of cycling all wire pairs is to connect the validity of each wire pair to a validity gathering tree[8]. This ensures that only when all inputs have arrived and all wire pairs have become valid will the validity reach the output of the stage and be acknowledged.

A. Circuit composition

Figure 4 shows the layout of a early output gate with "Forward Guarding". The gate data output validity is gathered with the validities of the gate inputs in a C-element to generate the gate validity. The validity of each gate signifies the validity of all stage inputs and data wire pairs which contributed (directly and indirectly) to the gate.

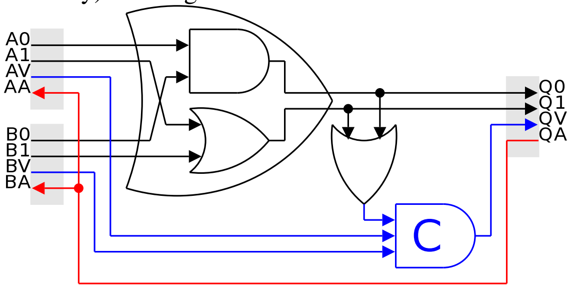

Fig. 4. Forward Guarding

III. Backward Guarding

Backward guarding is a complementary technique to the forward guarding style. Rather than ensure the cycling of data wires on the validity propagation, the guarding is done on the acknowledge propagation. Validity can be generated locally and an acknowledge signal can be propagated to some parts of the circuit while other parts are waiting to become valid.

A. Circuit composition

Figure 5 shows the layout of an early output gate with backward guarding. The validity of the gate is taken from the validity of the data output. These validities are then gathered in the next gate the output feeds into, along with the acknowledge the next gate receives, to generate an acknowledge to propagate back to the input gates. The validity received by the gates on their inputs could be generated locally by placing OR gates across each input pair. Generating the validity on the output of a gate rather than on the inputs reduces the number of components and allows composition with forward guarded gates. As gate outputs often fork, the generation of the data pair validity in one place allows the gate-count used to be reduced. The composability of backward and forward guarded gates can be assured by virtue of their protocol sequencing. As both protocols have the same sequencing and rely on the same set of assumptions, it is possible to mix the two systems in a single stage. The local validity generation allows the fast propagation of the validity while delaying the synchronisation with the data generated further towards the inputs of the stage to be delayed. As the acknowledge can progress through gates which have both inputs valid, often a set of inputs which have already contributed to all their destinations can be acknowledged before other inputs arrive.

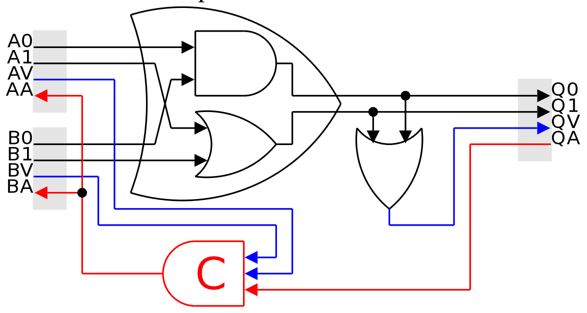

Fig. 5. Backward Guarding

IV. Analysis

To examine the performance implications of the styles a number of measurements were taken. The frequency of early outputs was measured across a range of commonly used circuits, and the performance of two circuits was observed when implemented with the two different guarding styles.

A. Early output cases

To show the suitability and effectiveness of early output logic on a variety of circuits, the ability to generate early outputs with a varying number of valid inputs was studied. The test was conducted on 10 commonly used logic designs, mostly taken from a synchronous microprocessor core. The circuits were: a seven segment encoder (segment A), an ALU slice, an 8 input AND gate, bit 8 of an adder, a MIPS branch unit, an 8 bit compare-if-equal unit, an 8:1 multiplexer, a MIPS processor memory shift unit, bit 4 of an 8 bit shifter and an 8 input XOR gate. All designs were taken from synchronous implementations.

The designs were then fed into the "early tool"[10] in order to determine their early output abilities. The early tool applies all possible combinations of input states (zero, one or null) and analyses the circuits ability to generate results with varying numbers of valid inputs.

B. Balanced stage

In order to demonstrate the different behaviours of forward and backward guarding, two early output circuits were constructed. Each circuit was then guarded with each of the two QDI guarding methods and the resultant systems were observed in a test bench wrapper.

The simulations were performed at gate level. Various components were given integer gate delay numbers. AND/OR gates have the delay of one while C-elements have a delay of two. Inversion delays were ignored or combined into gates. Only 2 and 3 input gates and C-elements were allowed in designs to match the cells available in current tech libraries.

The balanced stage circuit is a 16 input XOR gate constructed from a tree of two input XOR gates. The output of the gate is passed to the 16 latches driving the inputs of the XOR gate. The test is designed to determine the behaviour of the guarding on stages which have: a balanced computational distance for each input, no early output cases and simultaneous input arrival times. The XOR gate has a logic distance of 8 gates on all inputs, no early output states, and all inputs are connected to the same source (thus have simultaneous arrival times).

C. Adder

Complementary to the balanced stage is the adder example circuit. The adder is a 64 bit ripple carry adder. With a computational distance varying from 3 to 129 gates and containing many early output cases with bit level pipelined independent inputs, the circuit poses a different challenge to the guarding logic. This setup simulates the effect on an adder when performing operations on very long integers.

V. Results

A. Early output cases

Early output generation can be demonstrated by observing the behaviour of circuits under all possible input combinations. However this can be seen as an unfair test as the input arrival order and data is random while in the actual implementation the data is rarely random. Some benchmarks presented would show an improvement while others would show a drop in the frequency of early outputs when connected to realistic input sources.

Figure 6 shows the presence of early output states dependant on the percentage of valid inputs present. Most input circuits have very similar inputs present to output probability patterns. There are three circuits which do not follow the general trend. The XOR circuit can not generate an output without the presence of all inputs and so does not move from zero until valid inputs have reached 100%. The two better than average performing circuits are the AND gate and the compare-if-equal. The AND gate requires just one input to be zero in order to determine the result. This increases the probability of generating an output to 50%, as the probability of not being able to generate an output halves with each successive input arrival. Structures composed from large AND or OR gates give very high output probability with low numbers of valid inputs. The compare-if-equal is composed with XOR gates across input pairs and a large OR gate. The large OR gate gives many early output states in the design but not as many as the AND gate due to the XOR gate's negative effect.

The other designs have a very similar curve on the graph despite their quite varied logical functions and logical depths. Generally these designs will generate an output with, on average, 75.6% of inputs valid.

B. Balanced stage

The balanced stage benchmark is designed to observe the effect of the guarding logic choice on circuits which have few or rarely occurring early output states. While both guarding styles perform the same task and have the same protocol, the difference in sequencing of operations gives variations in the performance.

The design was placed in a testbench which generated inputs and consumed the results. The circuit was then simulated for 1,000,000 gate delays (120ms) and the number of results generated was counted. In the balanced stage example, the performance of the forward guarded circuit was 15,625 operations in 1,000,000 gate delays. The backward guarded circuit performed 22% slower at just 12,195 operations in 1,000,000 gate delays. To determine the reason for the difference in performance of different guarding styles, the sequencing of the systems must be examined.

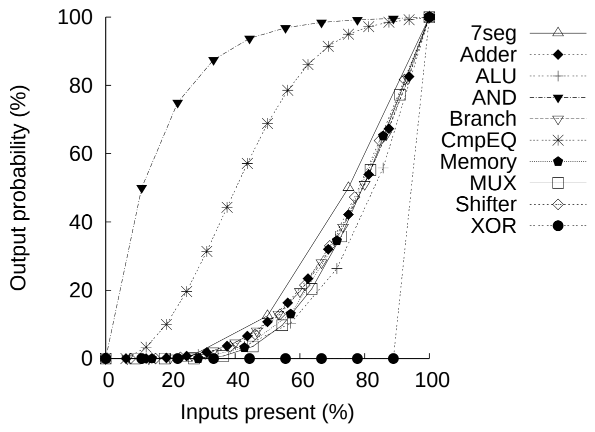

Fig. 6. Early Output Cases

Forward guarding processes the presence of inputs and the validity of the circuit as soon as the inputs arrive. This validity checking is done in parallel with data computation. Backward guarding, on the other hand, only starts to test for circuit and input validity once the result has been generated. Because of the parallelism implemented in the forward guarding method the circuit can be validity checked faster and thus an increase in performance is observed.

C. Adder benchmark

As stated in section 5.1, individual bits of the adder have many early output states. The adder benchmark ensures the presence of early output states by keeping both inputs at zero, allowing the generation of carry out without the presence of carry in. The ripple style carry system ensures a variation in the bit result generation times. Such a system is often thought of as having skewed wave-front pipelining as the lower bits are generated ahead of the higher end bits. This benchmark measures the ability of guarding logic to take full advantage of early output cases and varied logic depth along with its ability to deal with late unnecessary inputs.

Again the test circuit was simulated for 1,000,000 gate delays and the number of cycles completed was recorded. Forward guarding logic completed 1,270 additions while the backward guarded circuit performed 17,280. This demonstrates a case where the use of backward guarding logic gives over 13 times the performance of forward guarding logic. In the worst case operation (zero minus one) the backward guarded version only managed 1,825 operations while the forward guarded version was unaffected and again completed 1,270 operations.

To present the reasons for the vast difference between the circuits the behaviour of the guarding styles is explained.

1) Forward guarding: The cycle time of the forward guarded circuit is around 788 gate delays. As mentioned before, the longest path through the adder is 129 gates. In the forward validity gathering phase, the critical path is from this farthest input to the carry out latch. For each gate along the path, the signal must pass through a C-element (equalling two gate delays), totalling in 258 gate delays. In the acknowledge phase, the signal must reach the furthest input while passing through a C-element for each fork in the critical path (65 forks/C-elements or 130 gate delays). Once the acknowledge reaches the furthest away input, the data can then be released and the validity starts its dropping cycle passing through the same elements as it did on the rising phase and consuming an equal amount of time.

As in this example the worst case path is much longer than the average, the validity cycle time being extended to worst case has a highly negative impact on the performance.

2) Backward Guarding: Backward safe guarding described in section section 3 allows a subset of inputs to move to back into the set phase once all dual-rail wire pairs they effect have become valid and returned back to the null state. The other input subset with members which have not become valid is halted until all inputs are presented and subsequently reset. This action can be repeated allowing the result of the stage to become several stages ahead of some inputs. Unfortunately at each cycle, the halted set of inputs grows, eventually absorbing the whole stage. This partial completion effect has the behaviour of collecting "anti-tokens" [9] to absorb the unnecessary inputs. When a stage can generate an output with some inputs missing, in anti-token designs, the inputs which have yet to present their data, may receive an anti-token. The anti-token will then consume one piece of data when it arrives. This allows the stage to continue operating without waiting for the remaining input to arrive, yet not desynchronise itself from that input stream. Although in this arrangement the anti-tokens are unable to progress through the input latch to the previous stage, multiple anti-tokens can be collected in a single stage.

Anti-token generation and stacking can be demonstrated in an example circuit shown in figure 7. The circuit has been abstracted so that one wire represents a full bundle of three wires (Data0, Data1, Valid) and the other wire represents the acknowledge. In the first state all inputs are present with the exception of input A. This late and unnecessary input cannot be acknowledged until its token has arrived. This in turn stops the progress of the acknowledge signal from reaching either it or any inputs which are combined (directly or indirectly) with the path of inactivity due to the non-presence of the input. In this case the only other input effected is input B while the other inputs are acknowledged (fig. 8).

The release of all other inputs in this case also drops the output and the acknowledge signal is released and shortened down to the single gate which is waiting for one of its inputs to be released signalling the acknowledge is being propagated. Although input C becomes reset it cannot become valid as a gate it feeds is propagating the acknowledge (fig. 9).

All other inputs can now become valid. Again if the set of inputs is sufficient to generate a result this stage can complete and generate another acknowledge pulse. Each acknowledge pulse is effectively an anti-token.

Each anti-token waits for the presence of valid data on all of its inputs before acknowledging them and does not release the acknowledge signal until all inputs have returned to zero (releasing their validity indicates they have accepted the acknowledge). This allows the anti-tokens to stack and not merge. Unfortunately most stages are rarely able to keep more than one anti-token. Stages with arrangements of interleaved AND and OR gates and a large variation in logical distances are very well suited to the backward guarding style.

The carry ripple adder in the benchmark circuit has that exact arrangement which allows an acknowledge to be sent as soon as a result is ready (which is much sooner than the worst case). The variation in logical distances allows the closest inputs to de-assert their data and allows the stage bit outputs to release their acknowledge sooner. This then allows a new computation to start while the acknowledge wave progresses down to the further away inputs. The inputs furthest away will be acknowledged once the signal reaches them. The wave can be broken off from the initiating acknowledging latches and progress down towards the remaining inputs while the top end of the adder starts processing the next wave of data.

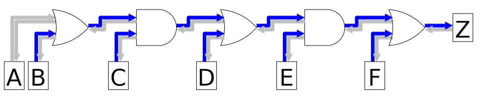

Fig. 7. Backward Guarding Circuit:Initial Inputs

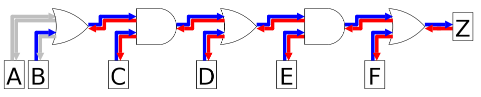

Fig. 8. Backward Guarding Circuit:Acknowledge Arrival

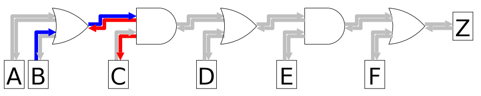

Fig. 9. Backward Guarding Circuit:Input Withdrawal

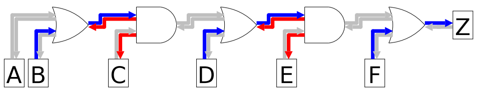

Fig. 10. Backward Guarding Circuit:Anti-Token Stacking

VI. Conclusion

This paper has presented some results and diagnosis of early output logic circuits. The number of early output states with varying number of valid inputs of many common circuits has been shown along with the reasons for their differences in their behaviours.

The early output circuits need a form of guarding to ensure correct operation. The two types of guarding presented (forward and backward) have complementing performances when applied to circuits with or without variations in logic depth and input arrival time. The use of the better suited guarding system has been shown to have a large effect on the performance of the system. This has been demonstrated in the adder example where a 13 times throughput improvement can be achieved with the use of backward guarding, and to a smaller extent in the balanced stage example where the use of backward guarding gave a performance decrease of 22%.

As common circuits are usually a combination of always needed early-arriving inputs and rarely needed late-arriving inputs, no single strategy is perfect. Backward and forward guarding can be mixed (not just in separate stages but also in a single stage). The resultant stage can take advantage of parallelising the validity checking of 'on-time' inputs with the logical operation, while still being capable of completing the operation before the arrival of the late inputs. The late arriving unnecessary inputs need only to handshake with a gate much closer to them, rather than communicating with the output latch.

References

[1] T. M. Mak, "Is CMOS more reliable with scaling?", IEEE Int. On-Line Testing Workshop, July 2002.

[2] J. A. Brzozowski and J. C. Ebergen, "On the delay-sensitivity of gate networks", IEEE Transactions on Computers, 1349-1360, November 1992.

[3] A. J. Martin. The limitations to delay-insensitivity in asynchronous circuits. In William J. Dally, editor, Advanced Research in VLSI, pages 263-278. MIT Press, 1990.

[4] K. v. Berkel, "Beware the isochronic fork", Integration, the VLSI journal, vol. 13, pp. 103128, June 1992.

[5] D. E. Muller and W. S. Bartky, "A theory of asynchronous circuits," Proceedings of an International Symposium on the Theory of Switching, Cambridge, MA: Harvard Univ. Press, pp. 204-243, 1959.

[6] C.F. Brej, "An automatic synchronous to asynchronous circuit convertor", 11th UK Asynchronous Forum, 2001.

[7] Charles L. Seitz, "System Timing," in introduction to VLSI Systems, Carver Mead & Lynn Conway, eds., Addison Wesley, Reading, MA, 1980, pp. 242-252

[8] A. Kondratyev and K. Lwin, "Design of asynchronous circuits using synchronous CAD tools". Proc. DAC'02, New Orleans, USA, 2002,

[9] C.F. Brej,"Early Output Logic using Anti-Tokens", Twelfth International Workshop on Logic and Synthesis (IWLS 2003), May 2003.

[10] "Early resynthesis tool", http://www.cs.man.ac.uk/ brejc8/early/