Asynchronous FPGA Architectures

Rob Payne

Abstract

Field Programmable Gate Arrays (FPGAs) are of increasing importance as processor support devices, and as computational devices in their only right. Current synchronous FPGA architectures create problems for the implementation of asynchronous circuits, due to their creation of hazards, re-ordering of signals and lack of arbitration. This paper examines how the first generation of asynchronous FPGA architectures (MONTAGE, PGA-STC and STACC) tackle these problems.

1 Introduction

In recent years, Field Programmable Gate Arrays (FPGAs) have become the dominant form of programmable logic. In comparison to previous programmable logics such as PLAs, FPGAs can implement far larger logic functions. Rather than being used merely as 'glue logic', FPGAs have sufficient logic resources to implement complete systems and sub-systems. Another difference of many FPGAs from previous programmable logics is the use of SRAM, instead of fuses, for the configuration memory. SRAM-based FPGAs can act as 'soft-hardware'; a configuration file can be loaded into a SRAM-based FPGA's configuration memory and then run in a similar way to software. This reconfigurability is being used in a new class of FPGA-based system, which are often called *transformable systems* [8].

Building asynchronous circuits using FPGAs has several potential benefits over a synchronous implementation. In addition to the advantages normally attributed to asynchronous circuits such as robustness, low power consumption and average-case performance, the mapping problems of partitioning, routing and placement for FPGAs are simplified by not having to meet a global clock constraint. In particular, asynchronous circuits are well matched to the needs of transformable systems. Current applications are constrained by fixed mappings to the FPGA, since, to change the routing and placement requires the recalculation of the global clock period. Asynchronous circuits allow routing and placement to be changed between self-timed modules with ease, so that the full benefits of reconfigurability within transformable systems can be exploited.

The rest of the paper is structured as follows: Section 2 considers the initial work in the field that concentrated on implementation using current synchronously-oriented FPGA architectures. The problems encountered with hazards, signal re-ordering and arbitration on current FPGAs, led to the proposal of FPGA architectures designed for the implementation of asynchronous circuits. These architectures: MONTAGE, PGA-STC and STACC are introduced in Section 3. In following sections, the different ways in which the architectures deal with the problems of hazards (Section 4), signal ordering and delay (Section 5) and arbitration (Section 6) are examined. Finally, Section 7 summaries the approaches of the different architectures, and looks at future directions for asynchronous FPGAs.

2 Asynchronous Circuits on Current FPGAs

Much of the early research on implementing asynchronous circuits using FPGAs concentrated on implementation using commercially available synchronously-oriented FPGAs. Several researchers built Micropipeline [15] libraries [11, 3, 5, 10], using a variety of FPGA architectures. Brunvand used such a library of Micropipeline elements to build a self-timed processor [2]. Shaw and Milne [14] implemented asynchronous circuits on a FPGA-based computing machine called the SPACE machine. The advantage of using current FPGAs is that the chips are readily available standard parts, and can implement synchronous systems as well. However, whilst these initial works showed that asynchronous circuits could be implemented using current FPGAs, they also highlighted the limitations of current FPGAs for the implementation of asynchronous circuits. These limitations are listed below:

Hazards: In delay-insensitive circuits, and the control-path of bundled-data circuits, signals are being continuously sampled so must be free from hazards. Current synchronous FPGAs are not designed to produce hazard-free signals.

Ordering and Delaying Signals: To work correctly, asynchronous circuits rely on the ordering of signals within themselves. In delay-insensitive circuits, this manifests itself as a need for *isochronic forks*; in bundled-data systems, correct operation requires that the request signal is asserted after the data signals (the *bundling constraint*). Current FPGA routing architectures can easily re-order signals and make such ordering and delay constraints difficult to meet.

Arbitration: Arbitration is a common function within asynchronous circuits. Current FPGA architectures provide no support for building the special circuitry needed in arbiters for providing clean output signals from the possible meta-stable state that such circuits can enter.

3 Asynchronous FPGA Architectures

At present, three published designs exist for asynchronous FPGA architectures. These are listed below:

MONTAGE [7] designed at the University of Washington, was the first asynchronous FPGA, though it includes a clock signal for implementing synchronous circuits as well. It is based on the TRIPTYCH [6] architecture that was also developed at Washington. MONTAGE extends TRIPTYCH by adding special arbitration cells, and modifying the function-unit.

PGA-STC [9] developed at U.C.Davis is targeted at the implementation of two-phase bundled-data systems such as Micropipelines [15]. The architecture is loosely based on that of the Xilinx XC4000 series [16], with modifications to the function-unit, and the addition of arbitration cells and programmable delay elements.

STACC [12] (Self-Timed Array of Configurable Cells) is an architecture developed by the author at the University of Edinburgh. It is a dedicated architecture for the implementation of four-phase bundled-data systems. The STACC architecture is based on that of fine-grain FPGA architectures (such as the Algotronix CAL [1]) where the global clock is replaced by an array of timing-cells that generate local register control signals.

The remainder of the paper describes how the architectures overcome the problems of hazards, signal re-ordering and arbitration.

4 Function-Unit Design

The need for hazard-free signals in asynchronous circuits is reflected in the function-units design of both the MONTAGE and PGA-STC architectures. Current FPGAs use a wide variety of function-units, but both PGA-STC and MONTAGE use look up tables (LUTs), since LUT-based implementations are free from hazards on single input changes. Several commercial FPGAs manufacturers also use LUT-based function-units, such as Xilinx [16].

Figure 1 shows the function-unit for MONTAGE; inputs `A`, `B` and `C` are used to select a value from configuration memory. Though free from hazards on single input changes, a LUT based function-unit may still create output hazards on multiple input changes. For the example configuration, the transition of `ABC` from 010 $\rightarrow$ 100 can cause a momentary 1 to occur on the output of the LUT. Both MONTAGE and PGA-STC leave multiple input changes as a problem for mapping tools to avoid.

Another feature of delay-insensitive circuits is the use of feedback to create state-holding elements. Problems may occur if the next state has not been established before the next input change. Hence both PGA-STC and MONTAGE include fast feedback paths within the function-unit, to allow new states to be established quickly. In the MONTAGE function-unit, any of the inputs to the LUT may be replaced with a feedback signal from the output (in the example configuration of Figure 1, the `C` input is fed back). The MONTAGE function-unit also includes a D-latch, principally for use in synchronous designs. However, the MONTAGE designers also utilise the D-latch for the initialisation of state-holding functions. After initialisation, the D-latch is bypassed.

The PGA-STC function-unit (Figure 2) has a similar structure to the MONTAGE function-unit. The principal difference is the inclusion of the Programmable Delay Element (PDE). The PDE is included since PGA-STC is targeted at implementing bundled-data systems, where request signals have to be delayed to match the delay in the data-path. The PDE is considered in more detail in Section 5. The output multiplexor chooses between the LUT output and constant zero and one inputs to allow the initialisation of state-holding elements.

The STACC architecture has a very different structure to MONTAGE and PGA-STC. STACC is designed specifically for implementing bundled-data systems; bundled-data systems have a very clear split between the data-path and the control-path. The data-path is similar to that found in synchronous systems, in that it may also contain hazards. However, the control-path needs to be hazard-free since it uses delay-insensitive style circuits. The approach taken in the STACC architecture is to implement the data-path and the control-path in two different kinds of cell that reflect their different uses.

Figure 3 shows the basic STACC architecture. The timing-cells provide register control to a region of standard FPGA logic cells in the data-array. Each timing-cell is connected to its neighbours by two wires, one in each direction. These are used to initiate request/acknowledge handshakes with neighbouring timing cells. Configuration data determines whether neighbouring regions communicate, and the direction of the data flow between them. Communication may be conditional so values from the data array can control the flow of data. The arbitration function is also performed by the timing-cell.

In bundled-data systems, such as Micropipelines [15], the C-Muller gate is the basic synchronisation element. The STACC timing-cell is based on the idea of a *reconfigurable C-Muller gate*. Figure 4 shows the implementation of a reconfigurable C-Muller gate. Each input to the C-Muller gate can be configured to come, either from a handshaking input, or from the inverted output of the C-Muller gate. When the C-Muller gate's input comes from the handshaking input, it synchronises on that signal. When the inverted output is selected, the input becomes a *don't care* input, since the inverted output of a C-Muller gate is always the next value that the C-Muller gate is waiting to synchronise on. Hence, the reconfigurable C-Muller gate allows an arbitrary synchronisation pattern to be defined between the handshaking inputs. The full STACC timing-cell [12] allows branching and merging within the control-path, so elements such as *Select* and *Merge* gates [15] can be implemented.

5 Ordering Signals and Delay Elements

The architectures adopt a variety of different approaches to ordering and delaying signals. The MONTAGE architecture does not have dedicated delay elements; instead, it relies upon a tight regular routing structure. Figure 5(left) shows how isochronic forks are created by placing each branch of a fork on similar routing paths. Also, since MONTAGE has integrated routing and function blocks, asymmetric forks can easily be generated by routing the signal for the longer fork from the destination cell of the short fork. This is illustrated in Figure 5(right).

In bundled-data systems, to meet the bundling constraint, the request signal must be delayed with respect to the data signals. Furthermore, for performance, the delay of the request signal must be as close as possible to that of the data-path. Providing accurate delays without a timing signal such as a clock is a difficult task. Without a clock, the only practical way of doing this is to utilise gate delays. The MONTAGE architecture does not have special delay elements, so for bundled-data systems, it has to use routing and function-units to build a chain of buffer elements with the appropriate delay.

In contrast, both STACC and PGA-STC are targeted specifically at bundled-data systems, so they include dedicated delay elements. In STACC, the delay-element is integrated as part of the timing-cell. In PGA-STC, the delay element is included as part of the function-unit. In both architectures, a programmable delay is produced by take taps off a chain of inverters. Additionally, the PGA-STC architecture includes a fine-delay generator. The fine-delay generator is required since delays with the PGA-STC architecture are matched to each function-unit, so a delay-chain that is too coarse will introduce a large error in the delay over a series of function-units.

To produce a delay finer than one gate delay, the PGA-STC design utilises a novel structure called a ring-coupled oscillator (Figure 6). It consists of a set of inverter ring oscillators (the horizontal connections). The inverter ring oscillators are coupled to the oscillator below using special two-input inverter elements (the vertical connections); each two-input inverter element consists of two inverters driving the same output. The coupling of the inverter rings causes the oscillation of one inverter ring to be a delayed copy of the oscillation in the ring above. By connecting the bottom oscillator to the top oscillator (effectively forming a ring of ring oscillators), the phase shift around the whole loop is forced to be two inverter delays. So, the phase difference between neighbouring oscillators is two inverter delays divided by the number of inverter ring oscillators. With the addition of special control logic, delays of a fraction of a buffer delay can be generated.

There are two major problems with the coupled ring oscillator in PGA-STC. First, the oscillator is a big power drain, annulling the low power advantages of an asynchronous design. Second, the ring coupled oscillator takes up a large amount of silicon area that could be used for extra function-units. To overcome these problems, a possible adaptation of the PGA-STC architecture would be to only have programmable delay elements in some of the function blocks, or to only use the coarse delay chain.

An alternative delaying method that is being considered for the STACC architecture uses Current Sensing Completion Detection (CSCD) [4]. CSCD utilises the fact that the power dissipation and current flow of a CMOS circuit are close to zero when all of the circuit nodes have been charged. By placing current detectors on the power rails, this can be used to detect the completion of the logic function. The main problem with CSCD is the need for analogue circuitry in the current sensors which continually draw current, and hence increase static power consumption. However, the technique is promising due to its ability to generate data-dependent delays. For example, if the same data is presented to the logic function in succession, then all the circuit nodes are already pre-charged to the correct value, so the function completes immediately. Another benefit is that no configuration bits are required to determine the delay and that the designer does not need to analyse the circuit for delays. An interesting possibility of CSCD is that it could be used to detect meta-stable states which occur during arbitration, without the use of special arbitration cells.

6 Arbitration

Arbitration is a common problem within asynchronous systems. Current synchronous FPGAs do not have the special circuitry required to resolve the meta-stable state that can occur in arbiters, without there being a small chance of failure. Hence, all the current asynchronous architectures include special arbitration elements based on a mutual exclusion element. The mutual-exclusion element ensures that given two request signals `R1` and `R2`, that only one is granted by the signals `G1` or `G2` at the same time.

The architectures differ in the logic that surrounds the mutual exclusion element. Since PGA-STC is targeted at the implementation of two-phase circuits, it adds XOR gates to the mutual exclusion element that allow it to be configured as a two-phase arbiter (Figure 7). The grant signals from the Mutual Exclusion element are used to enable the D-latches, so that the request signals can only pass once they have been granted. The acknowledge signals, `A1` and `A2`, are used to acknowledge when the resource has been used, so that the grant signal can be dis-enabled.

Both STACC and MONTAGE base their arbitration function units on an enabled arbiter circuit. In MONTAGE, the arbitration function unit may also be configured as a synchroniser and as a basic mutual-exclusion element. In STACC (Figure 8), the arbitration function is created using product terms formed from the incoming handshaking signals. The product terms are created by programmable AND (pAND) gates. The request signal is used to enable arbitration, and once arbitration has occurred, an acknowledge is generated which then feeds into the delay element. Hence, the timing-cell is delayed to account for the time taken for meta-stability resolution. A probe signal is fed into the data array, to allow the result of arbitration to be used within the data-path. An alternative implementation of arbitration in STACC [12] uses Q-flops [13] to sample the value of the handshaking signals.

The architectures integrate the arbitration cells to the rest of the architecture in different ways. In both MONTAGE and PGA-STC, arbitration cells are spread throughout the architecture, in place of standard function-units. An issue in both of these architectures is the ratio of data cells to arbitration cells. In the STACC architecture, arbitration is included as part of the timing cell. In STACC, the ratio of data cells to arbitration cells is part of a larger issue concerning the appropriate ratio of data cells to timing cells.

7 Conclusions

This paper has described the three current asynchronous FPGA architectures. The paper has focussed on three main issues: hazard avoidance within the function-units, the ordering and delay of signals and arbitration. The different approaches adopted in each architecture are, to a large extent, associated with the different design aims of each architecture.

MONTAGE was designed as a very general purpose architecture. Two global clock signals are included to allow the implementation of synchronous as well as asynchronous circuits. The basic modification to its synchronous cousin, TRIPTYCH, has been the inclusion of special arbitration cells and the optimisation of the function-unit for the implementation of hazard-free asynchronous elements. The lack of dedicated delay-elements makes MONTAGE more suited towards delay-insensitive asynchronous circuits.

In contrast, both PGA-STC and STACC are targeted specifically at bundled-data systems. Both include programmable delay elements as well as special arbitration cells. However, the two architectures take different approaches to how they integrate the asynchronous elements. PGA-STC is similar to the MONTAGE approach, in that the special cells are connected up using the same routing resource. This gives flexibility in how the elements are inter-connected, so a variety of asynchronous protocols can be implemented, though, the choice of basic elements in PGA-STC is biased towards a two-phase bundled-data implementation.

STACC is the most specialised of the architectures, using a dedicated four-phase bundled-data protocol. The architecture replaces the clock of a synchronous FPGA architecture with an array of timing-cells that provide local timing control. All of the special asynchronous elements are integrated within the timing-array, which uses separate routing resources to implement the control-path of a bundled-data system. Although specialisation limits the style of circuit that can be implemented, it allows the data-cells and timing-cells to be optimised for their particular task.

The future development of asynchronous FPGA architectures are likely to follow trends seen in the development of synchronous FPGAs Future asynchronous FPGA designs, are likely to have increased routing resources, with more emphasis on non-local routing. Function-units are likely to be more complex, or to be grouped together into hierarchies. The new asynchronous features are likely to focus on issues that are not addressed by the current architectures, such as I/O cells and asynchronous configuration interfaces.

In conclusion, the first generation of asynchronous FPGA designs have adopted a diverse range of solutions to the problems encountered with synchronous FPGA architectures. The variety of approaches shown in the new architectures are, in part, a reflection of the range of asynchronous protocols that the architectures were designed for, and in part a reflection of the diversity that already occurs in synchronous FPGA architectures.

Acknowledgements

I would like to thank Gordon Brebner and Iain Lindsay for their advice and guidance.

References

[1] Algotronix Ltd., The King's Buildings, TTC , Edinburgh EH9 3JL. *CAL1024 Datasheet*, 1991.

[2] E. Brunvand. Using FPGAs to Prototype a Self-Timed Computer. In *Workshop on Field Programmable Logic and Applications*, pages 192–198, 1992.

[3] E. Brunvand. Using FPGAs to Implement Self-Timed Systems. *Journal of VLSI Signal Processing*, 6(2):173–190, August 1993.

[4] M. E. Dean, D. L. Dill, and M. Horowitz. Self-Timed Logic Using Current-Sensing Completion Detection (CSCD). In *Proc. International Conf. Computer Design (ICCD)*, pages 187–191. IEEE Computer Society Press, October 1991.

[5] M. Gamble, B. Rahardjo, and R. D. McLeod. Reconfigurable FPGA Micropipelines. Technical report, U. of Manitoba, 1994.

[6] S. Hauck, G. Borriello, and C. Ebeling. TRIPTYCH: An FPGA Architecture with Integrated Logic & Routing. In T. Knight and J. Savage, editors, *Advanced Research in VLSI & Parallel Systems - Proceedings of the 1992 Brown/MIT Conference*, pages 26–43. The MIT Press, Cambridge, Massachusetts, 1992.

[7] S. Hauck, S. Burns, G. Borriello, and C. Ebeling. A FPGA for Implementing Asynchronous Circuits. *IEEE Design and Test of Computers*, 11(3):60–69, 1994.

[8] B. L. Hutchings and M.J. Wirthlin. Implementation Approaches for Reconfigurable Logic Applications. In *5th International Workshop on Field Programmable Logic and Applications*, LNCS 975, pages 419–428, 1995.

[9] K. Maheswaran. Implementing Self-Timed Circuits in Field Programmable Gate Arrays. Master's thesis, U.C.Davis, 1995.

[10] K. Maheswaran and V. Akella. Hazard-free Implementation of the Self-Timed Cell set for the Xilinx 4000 Series FPGA. Technical report, U.C.Davis, 1994.

[11] J. Oldfield and C. Kappler. Implementing Self-timed Systems: Comparision of Configurable Logic Arrays with Full Custom Circuits. In *FPGAs: International Workshop on Field Programmable Logic and Applications*, chapter 6.3. Abingdon EE&CS Books, 1991.

[12] R. E. Payne. Self-Timed FPGA Systems. In *5th International Workshop on Field Programmable Logic and Applications*, LNCS 975, pages 21–35, 1995.

[13] F. U. Rosenberger, C. E. Molnar, T. J. Chaney, and T. Fang. Q-Modules: Internally Clocked Delay-Insensitive Modules. *IEEE Transactions on Computers*, C-37(9):1005–1018, September 1988.

[14] P. Shaw and G. Milne. A Highly Parallel FPGA-Based Machine and its Formal Verification. Technical Report HDV-28-93, U. of Strathclyde, 1993.

[15] I. E. Sutherland. Micropipelines. *Communications of the ACM*, 32(6):720–38, 1989.

[16] Xilinx Inc., San Jose, California. *The Programmable Logic Data Book*, 1994.

List of Figures

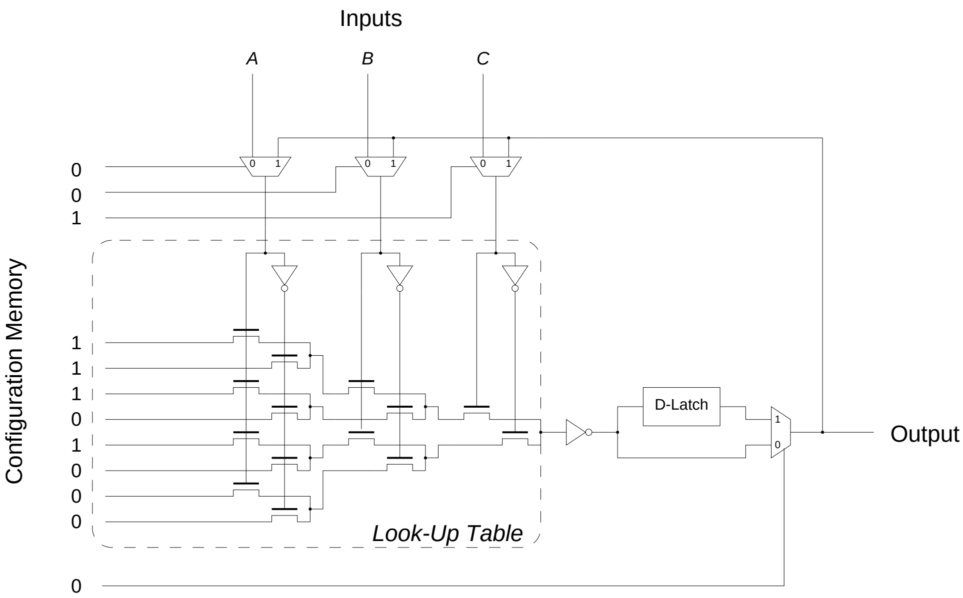

Figure 1: The MONTAGE Function-Unit (configured as a C-Muller Gate)

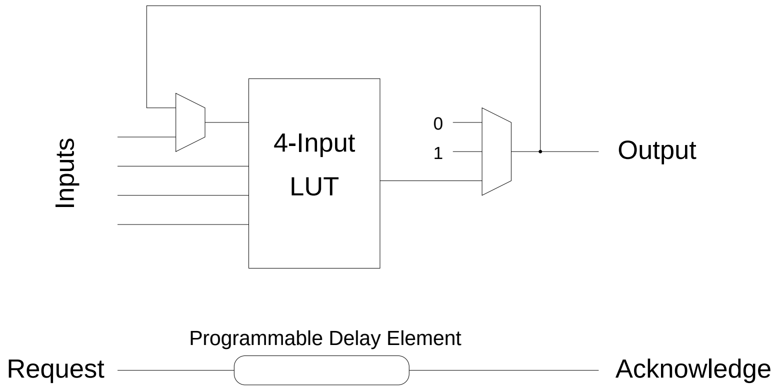

Figure 2: The PGA-STC Function-Block

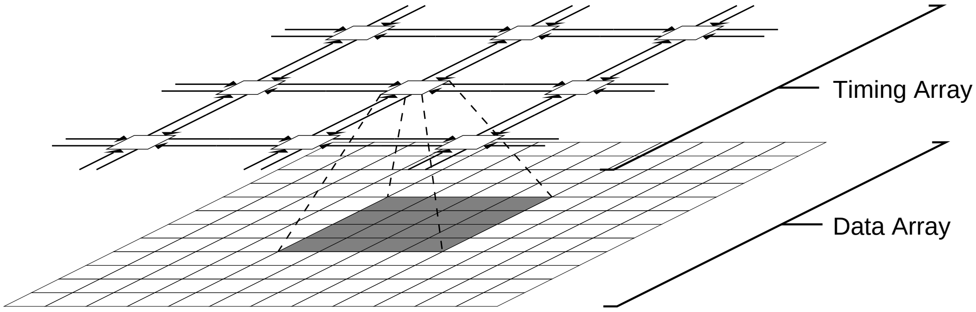

Figure 3: Basic structure of the STACC Architecture

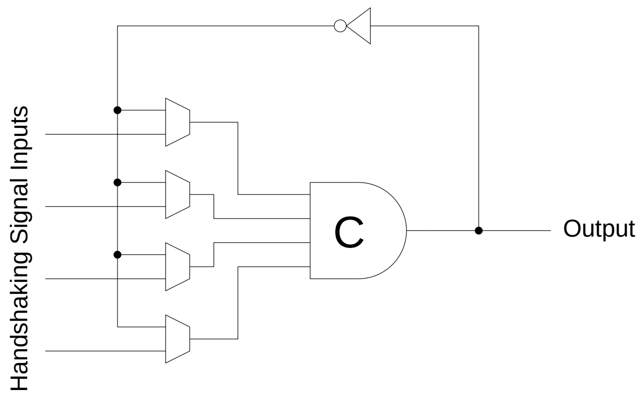

Figure 4: A Four Input Reconfigurable C-Muller gate

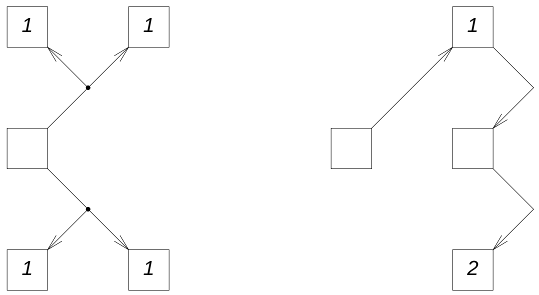

Figure 5: Isochronic (left) and Asymmetric Forks (right) in MONTAGE

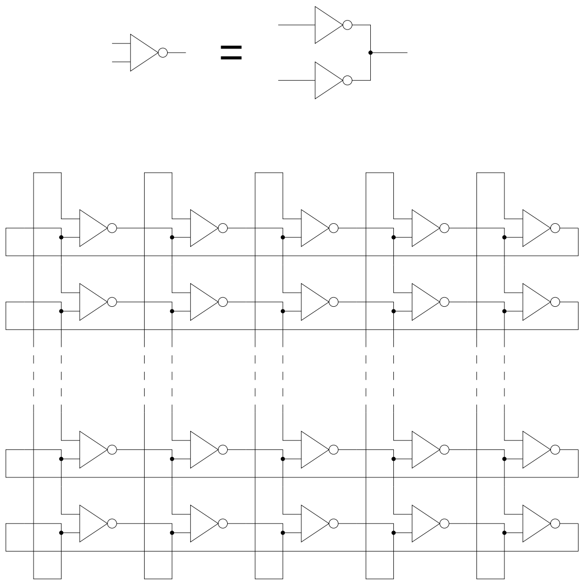

Figure 6: PGA-STC Ring Coupled Oscillator

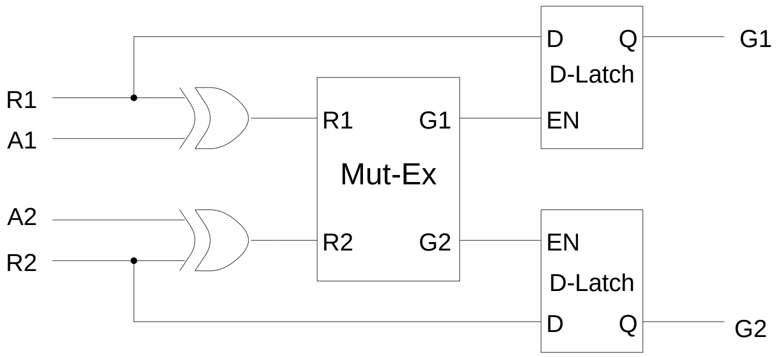

Figure 7: PGA-STC Arbitration Function

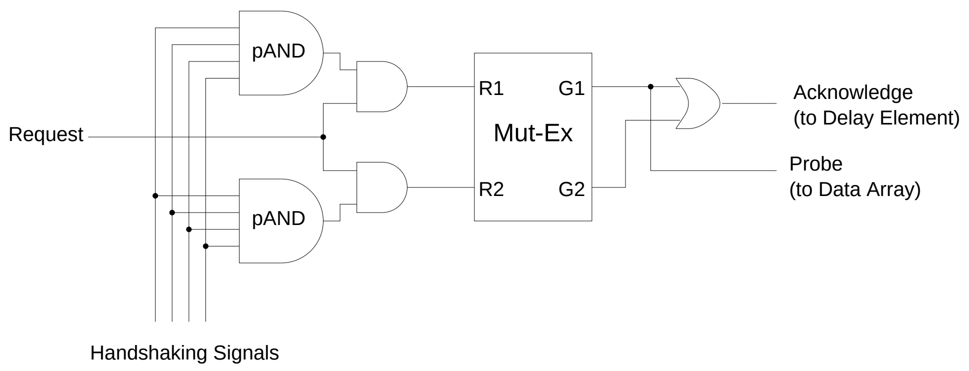

Figure 8: STACC Arbitration Function