

UNIVERSITY OF WESTMINSTER

**WestminsterResearch**<http://www.wmin.ac.uk/westminsterresearch>**A system for calculating the greatest common denominator implemented using *asynchrobatic* logic.****David J. Willingham  
Izzet Kale**

School of Electronics and Computer Science

Copyright © [2008] IEEE. Reprinted from Ellervee, Peeter, Jervan, Gert and Nielsen, Ivan Ring, (eds.) 26th Norchip Conference, Tallinn, Estonia, 17 - 18 November 2008. Formal proceedings. IEEE, pp. 194-197. ISBN 9781424424924.

This material is posted here with permission of the IEEE. Such permission of the IEEE does not in any way imply IEEE endorsement of any of the University of Westminster's products or services. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE. By choosing to view this document, you agree to all provisions of the copyright laws protecting it.

---

The WestminsterResearch online digital archive at the University of Westminster aims to make the research output of the University available to a wider audience. Copyright and Moral Rights remain with the authors and/or copyright owners. Users are permitted to download and/or print one copy for non-commercial private study or research. Further distribution and any use of material from within this archive for profit-making enterprises or for commercial gain is strictly forbidden.

---

Whilst further distribution of specific materials from within this archive is forbidden, you may freely distribute the URL of the University of Westminster Eprints (<http://www.wmin.ac.uk/westminsterresearch>).

In case of abuse or copyright appearing without permission e-mail [wattsn@wmin.ac.uk](mailto:wattsn@wmin.ac.uk).

# A system for calculating the Greatest Common Denominator implemented using *Asynchrobatic* Logic

David J. Willingham and İzzet Kale  
Applied DSP and VLSI Research Group,  
University of Westminster,  
London, Great Britain.

Email: [D.Willingham@wmin.ac.uk](mailto:D.Willingham@wmin.ac.uk) [kalei@wmin.ac.uk](mailto:kalei@wmin.ac.uk)

**Abstract**—An *Asynchrobatic* system that uses Euclid’s Algorithm to calculate the greatest common denominator of two numbers is presented. This algorithm is a simple system that contains both repetition and decision, and therefore demonstrates that *Asynchrobatic* logic can be used to implement arbitrarily complex computational systems. Under typical conditions on a  $0.35\mu\text{m}$  process, a 16-bit implementation can perform a 24-cycle test vector in  $2.067\mu\text{s}$  with a power consumption of  $3.257\text{nW}$ .

**Index Terms**—*Asynchrobatic* logic, adiabatic logic, charge-recovery logic, low power circuit techniques.

## I. INTRODUCTION

*Asynchrobatic* logic [1] is a low-power design methodology that uses asynchronous control to drive an adiabatic or quasi-adiabatic data-path. These authors’ previous works have shown that it is possible to implement both simple and more complex data-path structures using this design methodology [1][2]. This work provides a brief explanation of the *Asynchrobatic* logic style, an explanation of Euclid’s method, and then provides details of the implementation and testing of the complete system. It shows that more complex control structures can be implemented using *Asynchrobatic* Logic.

## II. ASYNCHROBATIC LOGIC

As noted in the introduction, *Asynchrobatic* logic is a low-power methodology that combines an asynchronous controller with an adiabatic or quasi-adiabatic data-path to produce processing structures that can operate both asynchronously and adiabatically. There are several components to any *Asynchrobatic* logic system. The asynchronous control logic takes components of four-phase bundled-data asynchronous systems, and combines these with self-timed Stepwise Charging (SWC) circuits. The SWC circuits use tank capacitors to perform their charge recycling.

The asynchronous controller is complemented by an adiabatic data-path. This data-path is pipelined, and each

pipeline stage requires a SWC controller to generate its local power-clock signal.

### A. Asynchronous Logic

The asynchronous systems described herein use Muller C-Elements [3] as their basic building block. As well as simple sequential pipeline stages, the other components used from four-phase bundled-data asynchronous are the logic and control portions of the MUX and DEMUX elements. A more complex system could also use arbitrators to control access to shared resources, although it is not necessary in this example. The various control functions use four-phase bundled data.

### B. Adiabatic Logic

Adiabatic logic is also known as “Charge Recovery” or “Clock-powered” logic. Adiabatic logic uses less energy than standard logic because charge is used more than once. The various charge-recycling schemes obviously have power-overheads, but if these can be amortised against savings elsewhere in the system, there will still be power benefits. In this case, it operates using a four-phase local power-clock, which complements the four-phase signalling used in the asynchronous portion of the circuit. In this example, the adiabatic logic family chosen to implement the data-path logic was Positive Feedback Adiabatic Logic (PFAL) [4].

#### 1) Positive Feedback Adiabatic Logic (PFAL)

PFAL is a dual-rail logic family that, like many adiabatic logic families, is loosely based upon a modification to Differential Cascode Voltage Switch Logic (DCVSL) [5]. It uses a pair of evaluation paths, constructed from nFETs, connected between the power-clock and the complementary outputs; these are combined with a pair of cross-coupled inverters, driven by a power-clock. A simple PFAL buffer is shown in Fig. 1. PFAL was chosen, as it is one of the simplest adiabatic logic families and in appropriate applications, it has the potential to operate in a fully reversible fashion [6]. The link with DCVSL is very useful,

as there are efficient design methodologies for DCVSL circuits [7][8], that can be used to design PFAL gates.

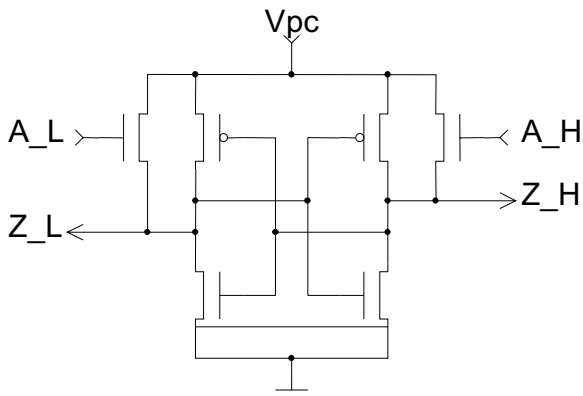


Figure 1. A PFAL Buffer [4]

### III. EUCLID'S ALGORITHM

There are two basic structures required by any programmable system in order to execute arbitrary algorithms: repetition and decision. A very simple algorithm, which can be implemented using a single repetition and a single decision, is that suggested by Euclid for calculating the Greatest Common Denominator (GCD) [9]. This functions by repeatedly subtracting the smaller of the two numbers from the larger. It can be expressed as shown in the following segment of pseudo-code, although the actual implementation does vary slightly from this.

```

while (a != b) do
  if (a > b) then
    a = a - b
  else
    b = b - a
  end if
end while

```

In this work, the asynchronous structure presented by Sparsø and Furber [10] has been modified to operate using the *Asynchronous* methodology. The main differences between the design previously presented and the one presented herein are as follows. Firstly, both of the inequalities are calculated in a merged block, the two results being propagated as necessary. Secondly, the selection and creation of the minuend and subtrahend is performed using a plurality of XOR gates to generate the ones' complement, and a MUX operator feeds this into the bypass buffers. The minuend being the larger of the two inputs is consumed by the subtractor. Finally, the outputs are re-ordered, with the subtrahend always being assigned to variable "A", and the difference always being assigned to variable "B".

### IV. DESIGN

There are two major and complex components to this GCD calculating circuit, the asynchronous controller circuitry and a subtractor. In addition to these blocks, there are also some more simple blocks, including a comparator, and the interfacing between the asynchronous control domain and the adiabatic data-path domain where multiplexers are used to select data inputs.

This design was implemented using a sixteen-bit wide data-path, with each of the inputs interpreted as an unsigned positive integer. The subtraction was performed using standard two's complement arithmetic. The schematic for the design is shown in Fig. 2.

#### A. Subtractor

The subtractor design used was a radix-four parallel-prefix design. It is based upon the adder detailed in these authors' previous work [2]. The adder detailed in that paper has been converted into a selectable subtractor or reverse subtractor by the inclusion of an input pre-processing stage. This consists of XOR gates that selectively complement one of the inputs.

#### B. Asynchronous control

When constructing loops in asynchronous designs, it is necessary to create an initialisation token. This initial asynchronous token is pre-programmed using the global reset signal to force one of the SWC controllers into a charged state, concurrently with forcing the single-bit data-path cell into a state representing the completion condition of the loop (equality). This causes the input MUXes to be set to request a new pair of inputs. In Fig. 2., this token is marked "T0".

#### C. Comparator

The inputs are processed by a comparator. This produces one output to signal equality of the two inputs, which is used to control the while loop, and another output to indicate which of the two inputs is the greater. This controls the programmable subtractor. Like the subtractor, the comparator used radix-four operations to reduce the number of pipeline stages. Equality is derived by comparing each bit with the corresponding bit of the other input and returning true only if all bits are equal (a bus of XNOR functions followed by an AND tree). The comparison is performed by reusing the equality bits and checking if the first input is greater than the second. This is only true if the first input is high and the second input is low. These generate the comparison result using a radix-four method that returns the comparison result for the most-significant bit-pair whose bits are not equal, and reuse the multi-input AND-OR gates from the adder design.

## V. VERILOG MODELLING

The initial design work was carried out using the Hardware Description Language (HDL), Verilog to model the circuits. The design was then translated into SPICE netlists. These were then verified against the Verilog. This also demonstrates that it would in principle be possible to directly translate an *Asynchrobatic* system designed using Verilog into a SPICE netlist, and also alludes to the potential of being able to synthesis more complex systems using HDLs. The behavioural descriptions of the data-path can be for either single rail or dual rail operation, and for dual rail, the alternative of switch-level models could be used. The behavioural description of the data-path is achieved by evaluating on the “posegde” of the power-clock and clearing it on the “negedge” of the power-clock.

## VI. TESTING

Two simple tests are ideal to demonstrate the performance of the GCD algorithm on binary systems will be described. The short test that will complete quickly and the longer test that exercises more logic paths, but still completes within a reasonable simulation time.

The short test used three prime numbers: 2, 3 and P. To ensure that the GCD engine has sufficient bit-width to process the inputs, the prime P is required to meet the constraint shown in (1).

$$3 \times P \leq 2^W - 1 \quad (1)$$

The initial inputs to the system are  $2 \times P$  and  $3 \times P$ . This causes the GCD function to generate the result P after two cycles. The constants used for a sixteen-bit example are shown in (2).

$$W=16, P=21841, 2 \times P=43682 \text{ \& } 3 \times P=65523 \quad (2)$$

The longer test used the Fibonacci numbers  $F_{(n)}$  and  $F_{(n-1)}$ . To ensure that the GCD engine has sufficient bit-width to process the inputs, the  $n^{\text{th}}$  Fibonacci number  $F_{(n)}$  is required to meet the constraint shown in (3).

$$F_{(n)} \leq 2^W - 1 \quad (3)$$

If one was to observe the internal nodes of the circuit just prior to the output MUX, then one would be able to see the entire Fibonacci series from  $F_{(n)}$  to 1 as the engine performed its iterative calculations. The constants used for a sixteen-bit example are shown in (4).

$$W=16, F_{(n)}=46368, F_{(n-1)}=28657 \text{ \& } n=24 \quad (4)$$

The choice of test vectors is important for such simulations, because this simple implementation uses repeated subtraction. This means that a poorly chosen set of test inputs could lead to the circuit functioning as a down counter. This would not constitute an efficient test methodology!

## VII. RESULTS

Results are provided for front-end simulation only, with SPICE netlists that do not include any parasitic capacitance. These were implemented and simulated using an Alcatel (AMIS) 0.35 $\mu\text{m}$  process. These tests were performed using the Eldo MACH fast-SPICE simulator. The SPICE results are only presented for the longer Fibonacci-based test, as this exercises the design more. The results for delay and power obtained when varying process, but keeping the voltage and temperature the same are shown in Table I. The delay was measured from the input request handshake rising to the output handshake rising, and to avoid inaccuracies due to startup, these measurements were taken on the second simulation cycle.

Process; 3.3V; 25°C	Delay ( $\mu\text{s}$ )	Controller Power (nW)	Data-path Power (nW)	Total Power (nW)
ff	1.022	2.627	0.8034	3.430
tt	2.067	2.577	0.6801	3.257
ss	5.205	2.353	0.6252	2.978

TABLE I. POWER AND PERFORMANCE OF *ASYNCHROBATIC* GCD WHEN VARYING PROCESS CONDITIONS

The results show that in the slower process corners, the logic runs slower, and that at slower speeds, overall, the system uses less power. These are both obvious results. It can be seen that the power consumption of the controller blocks dominates data-path power consumption by a factor of about three, this demonstrates the importance of using radix-four arithmetic blocks. It again highlights that *Asynchrobatic* logic is best suited to high data-width applications. Finally, it suggests that some further work is needed to reduce overheads associated with one-bit wide signals on some of the paths. The suggested direction of this is to consider using asynchronous DCVSL circuits in single-bit control signals.

## VIII. CONCLUSIONS:

It has previously been shown that both simple and complex combinational functions can be implemented using *Asynchrobatic* logic. This work has shown that arbitrarily complex systems can be implemented using *Asynchrobatic* logic. Consequently, it shows that *Asynchrobatic* logic can be applied to real-world problems and algorithms, and that it could be applied to practical applications. Furthermore, the use of Verilog to model the design shows that there is potential for design automation. It also demonstrates that the design of these systems is not overly complex.

## ACKNOWLEDGMENT

This research was partially funded by a Quintin Hogg Research Scholarship from the University of Westminster.

## REFERENCES

- [1] Willingham D.J. & Kale I., "Asynchronous, quasi-Adiabatic (*Asynchrobatic*) Logic for Low-Power very Wide Data Width Applications", Proc. ISCAS 2004.
- [2] Willingham D.J. & Kale I., "An *Asynchrobatic*, radix-four, carry look-ahead adder", Proc. PRIME 2008, pp 105-108.
- [3] Muller D.E. & Bartky W.S., "A theory of asynchronous circuits" Proc. Int. Symp. Theory of Switching, pp. 204-243, 1959.
- [4] Vetuli A., Pascoli S.D. & Reyneri L.M., "Positive Feedback in Adiabatic Logic", Elec. Lett., 32(20):1867-1869, 26 Sept. 1996.
- [5] Heller L., Griffin W., Davis J. & Thoma, N., "Cascode voltage switch logic: A differential CMOS logic family", ISSCC Dig. Tech. Papers, 1984, pp. 16-17.
- [6] Willingham D.J. & Kale I., "Using Positive Feedback Adiabatic Logic to implement Reversible Toffoli Gates", submitted to Norchip'08.
- [7] Chu K.M. & Pulfrey D.L., "Design Procedures for Differential Cascode Voltage Switch Logic Circuits", IEEE J-SSC, 21(6):1082-1087, 1986.
- [8] Karoubalis T., Alexiou G.Ph. and Kanopoulos N., "Optimal synthesis of differential cascode voltage switch (DCVS) logic circuits using ordered binary decision diagrams (OBDDs)", Proc Euro-DAC 1995, pp. 282-287.
- [9] Euclide, "Elements", Book VII, John Daye, London, 1570.
- [10] Sparsø J. & Furber S.B., "Principles of Asynchronous Circuit Design: A Systems Perspective", Kluwer Academic, 2002, ISBN 0-7923-7613-7.

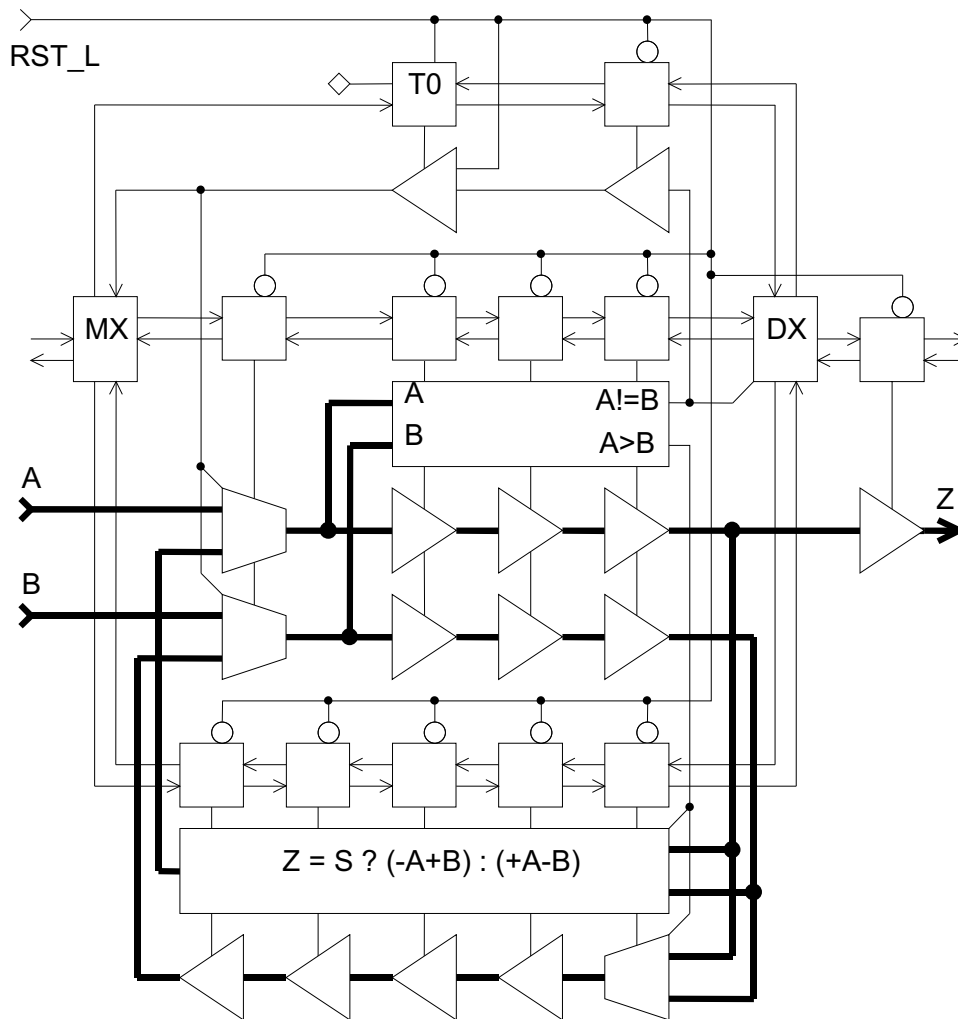


Figure 2. Asynchrobatic GCD