Unconventional Arithmetic: A System for Computation using Action Potentials

J Edwards[1], S O'Keefe[2], and W D Henderson[1]

[1] Thoughtful Technology, Newcastle, UK jonny@thoughtfultech.co.uk
[2] YCCSA, University of York, York, UK simon.okeefe@york.ac.uk

Abstract. This paper examines a scheme to perform arithmetic and logic computation using time delays inspired by neuronal Action Potentials. The method is reliant on a simple abstraction which utilises very little logical infrastructure, in fact, the only requirements necessary to carry out computation are a binary channel, a clock, and a rudimentary instruction look-up table.
The conclusions are that the method is viable for all forms of arithmetic and logical computation including comparison, however one practical aspect that hinders a full move to a time delay based architecture is the inability to perform random memory access without waiting for the data to recirculate.

1 Introduction

It is not an overstatement to say we are fixated with digital processing. Since Shannon's initial exposition of methods to perform digital operations [1] a Herculean effort has been applied both academically and commercially to construct ever more sophisticated methods for performing digital arithmetic and logical operations. This paper takes a step back from that research, and addresses the problem from a fundamentally different starting point, that of using *time* to represent data rather than the manipulation of transistors which expose themselves as states within a processing unit. The theme of the work is that time is a free resource and is only limited in resolution by the accuracy of the clock one is using, unlike electronics which require matter (atomic states) to be manipulated. We demonstrate surprisingly simple methods to perform all major arithmetic and logical operations using a single general processing unit which can be easily replicated.

The paper is structured as follows: Initially the action potential method is described in detail, this is demonstrated in the context of computation with examples for each of the arithmetic and logic operations. The paper then presents a more complex example which demonstrates the chaining of operations. Following this, we examine practical considerations, most notably the crucial issue of clock synchronisation. The paper concludes with a summary, which revisits the deep-rooted debate comparing analogue (often biological) systems with our predominant implementation of computation via digital methods.

2 Time Delay Processing

Models of neuronal activation are central to the description of function in the brain, they are broadly split between statistical measures of firing rates called **Rate Codes** or codes that are related to the delay between two spikes, often referred to as **Pulse Codes** ([2] chapter 1).

The Action Potential model proposed by [3] is a pulse code. In the Action Potential model the signal is transferred via spikes along a channel, and the information is contained in the time *between* spikes. In this work we describe an abstract model which is inspired by this "delay timing" value representation. The model consists of a processing unit with a binary channel, which has the capacity to carry unit impulses, and a clock with a variable speed, relative to an underlying synchronising, system wide, clock. The processing unit sends operand values as a time delay *between* pulses, so two unit impulses act as the "head" and "tail" of a time based value. The resultant signal is analogue in time. Figure 1 explains this visually, the value `two` is represented by passing an impulse delimited signal across the channel, with the clock scaled to the value 1. This has similarity with Pulse Width Modulation (PWM) [2], but with only the "head" and "tail" impulses.

Importantly, a continuous stream of these values can exist on one processing unit. Using this as our model for processing, all arithmetic and logical operators can be derived.

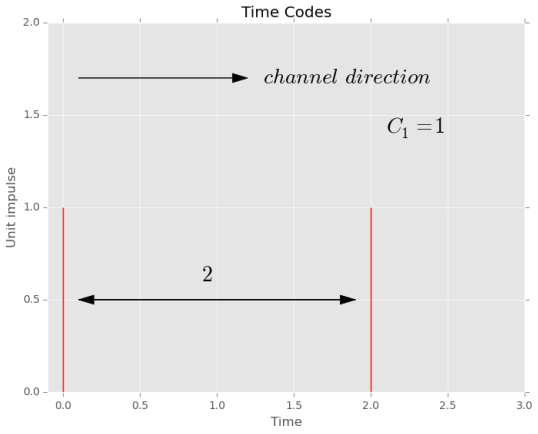

Fig. 1. A Time Delay Unit representing the value 2 as a delay between two impulses.

3 Requirements for a Realisable Computing Architecture

At the lowest level, the major functions of a processing unit are to move data within memory and to perform simple mathematical computation ([4] page 11).

Typically an operation requires a specific hardware part, so for instance addition is performed using the half and full adder circuitry ([4] page 90).

Different architectural approaches have been explored. The simplest is the Minimal Instruction Set Architecture (MISC), similar to the Java Virtual Machine (JVM)[3]. This sets a minimum level for the infrastructure necessary to perform computation, and again reduces down to simple arithmetic and memory manipulation. The following section describes the architecture necessary for all the arithmetic operations including comparison, it demonstrates that these can be built with relative ease on a general simple binary channel and clock architecture. The channel forms a flow from left to right, hence in the figures below the result is calculated at the right hand side.

3.1 Addition

Addition is the simplest operation to perform with the processing unit, simply "forgetting" the "tail" of the first operand and the "head" of the second operand. The signal then becomes the conglomeration of the two values, and hence addition is performed. Figure 2 gives an example of this for the sum $1+1$:

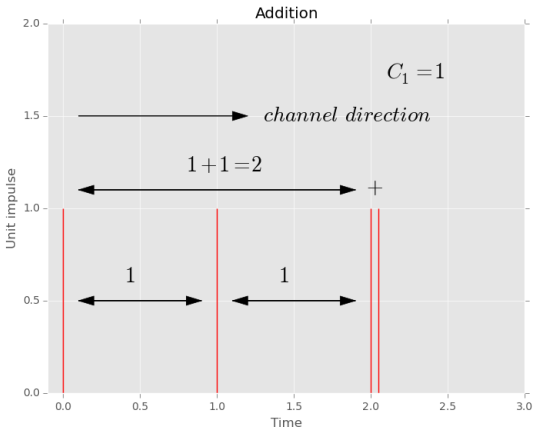

Fig. 2. Addition: 1+1, two values are added by ignoring the middle impulses, the full concatenation forms the addition. The flow of the processing is the signal moving from left to right

3.2 Subtraction

Subtraction relies on the processing unit sending two signals starting with the same "head", so both the signals are sent at the same time. Absolute subtraction

[3] `http://docs.oracle.com/javase/specs/jvms/se7/html/`

then becomes the time between the two "tails", whilst performing true subtraction requires attaching a "tag" pattern of impulses, to ascertain order. This tag is in effect a known pattern in the look-up table that is indicative of the order of presentation of the operands. Figure 3 visualises $2 - 1$, the two signals are overlaid and subtraction becomes their difference:

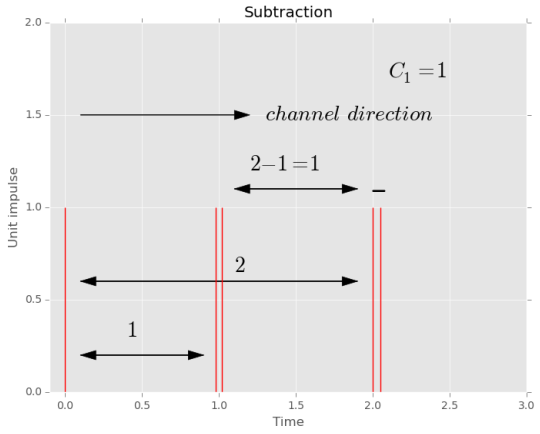

Fig. 3. Subtraction: 2-1. The header indicates order and the resultant difference be calculated through timing. The processing flows from left to right

3.3 Multiplication and Division

Multiplication and division can be performed by manipulating the clock speed relative to the system wide clock. To perform multiplication we slow the clock down by the first operand ($op1 * op2$: $\frac{1}{op1}T$) synchronised to the System wide clock so that the second operand takes longer to transmit across the channel. Conversely, for division, we speed the transmission up by increasing clock speed by the first operand and hence the second operand becomes shorter. Figures 4 and 5 visualise how this works for $2 * 1$ and $4/2$:

3.4 Logical Operators

Once arithmetic operators are implemented it is trivial to implement the AND (addition) and OR (multiplication). The NOT operator is performed by look-up, with a send/don't send switch in the operator look-up table.

3.5 Comparison Operators

The implementation of the minus operator gives rise to a natural method of comparing operands. A similar operation to negation occurs but the arrival of

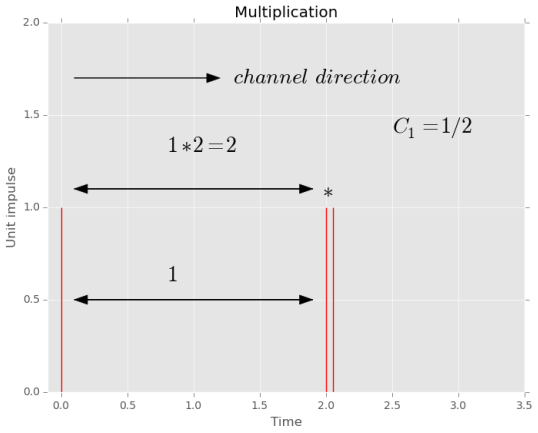

Fig. 4. Multiplication: 2*1. Changing the clock speeds relative to a central clock allows the signal to be scaled and hence multiplied. Processing is again performed from the left to the right.

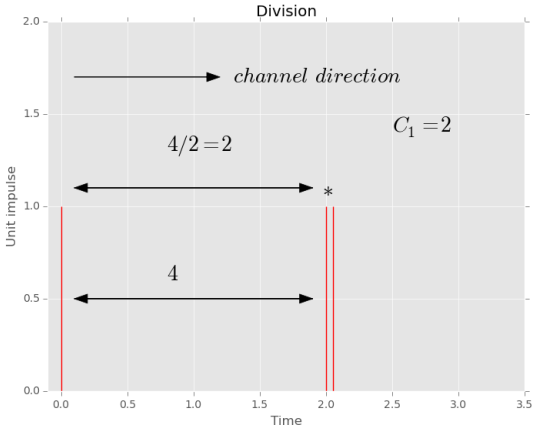

Fig. 5. Division: 4/2. Using the relative clock speed, clock scaling can perform division as well as multiplication.

the tag impulse is recorded, if it arrives before the second impulse the first operand is the greatest, if it arrives after then the first operand is smaller and if they arrive at the same time then there is equality. Figure 6 demonstrates this with a comparison between the values 2 and 1:

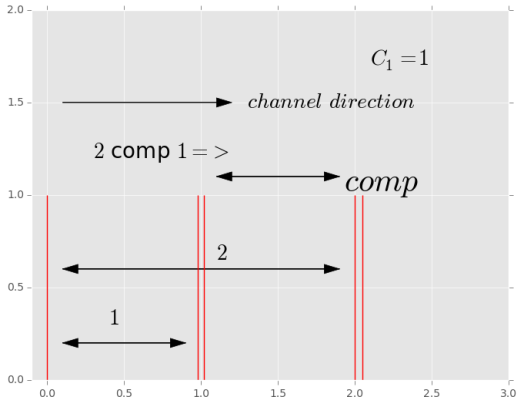

Fig. 6. A comparison operator, here we compare the values 2 and 1. Given that subtraction can be performed natively it is easy to implement comparison using a similar method.

3.6 Multiple Operations

To build more complex statements, the operations can be arranged into the traditional Reverse Polish queue. The whole calculation then becomes a procession through a *general* processing unit. Figure 7 shows how this queue might work, implicit in this is a method to deliver the multiplication and division operand to the clock. The figure presents an example for the expression (3+1)/2:

4 Time Delay Storage

Time delay storage through the re-circulation of a signal is not a new idea - in fact it was the pre-eminent storage method before transistors and integrated circuitry [5]. Several implementation methods exist [6]. The main disadvantage of this approach is that re-circulation slows down read/write access, and electrical circuitry requires fast op-amps[4]. Additionally, and more recently, alternative work with memory enabled resisitors, so-called *memristors*, has also focused on strategies for non-transistor based computation [7].

[4] `http://electronicdesign.com/analog/accurate-analog-delay-circuit`

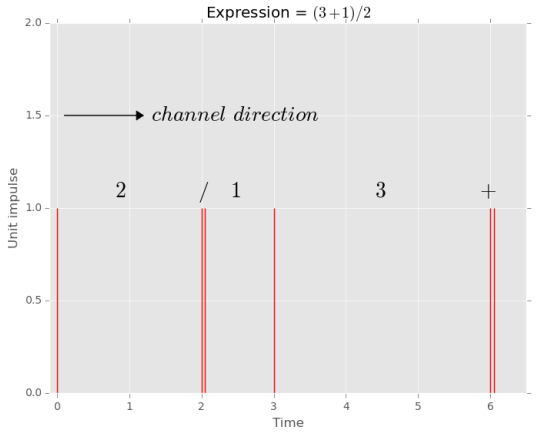

Fig. 7. A more complex arithmetical operation. However processing is still performed on one channel proceeding from left to right with the right acting as a receiver

5 Practical Considerations

5.1 Large Values

Large values present a problem for processing as they are formed from long delays, this is akin to large amounts of infrastructure to store large values in traditional processors. One way to mitigate against this is to use more channels to represent large numerical values, so effectively have an analogue equivalent of a bit-width, which we will refer to as the *resolution*.

As a trivial example we could arbitrarily set the resolution of a time-delay channel to one thousand, so a channel will have a time interval of a thousand clock cycles, and hence any number up to a thousand can be represented. Two channels could be used to encode all values up to one million by representing the upper and lower 3 digits (0-999) by individual time-delay channels. Addition and subtraction will function with a carry operation, and only a slight modification of interleaving the multiplication/division operands, for example $12 * 12$ will equal $10 * 10 + 2 * 10 + 10 * 2 + 2 * 2$.

5.2 Clock Drift

It may be necessary in this system to use more than one clock, and the limits on the precision of the transmitter and receiver clocks are readily computed. N is limited as follows:

– Rx clock is faster than Tx clock:

$$N < \frac{1}{2(\frac{\rho_{Tx}}{\rho_{Rx}} - 1)} \tag{1}$$

– Rx Clock is slower than Tx clock:

$$N < \frac{1}{2(1 - \frac{\rho_{Tx}}{\rho_{Rx}})} \tag{2}$$

Surprisingly, the limits on N are independent of δT, the time quantum or the nominal clock frequency and depend only on the relative clock drift rate (ρ_{Tx}/ρ_{Rx}). Table 1 records the calculated limiting values against clock tolerances:

Tolerance ppm	Rx faster than Tx	Tx faster than Rx
10	25000	25000
20	12500	12500
30	8333	8334
40	6250	6250
50	5000	5000
60	4166	4167
70	3571	3572
80	3125	3125
90	2778	2778
100	2500	2500

Table 1. Table of maximum usable values against clock tolerances

6 Conclusions and Further Work

The high level architecture described above represents a strong deviation from current Von-Neumann implementations, however it has several advantages. The above method blurs the lines between storage and processing. The circuitry required to process *and* store data is essentially homogeneous as memory units are similar to computation units (apart from the concept of re-circulation). Furthermore, there is also no need to implement specialist hardware for individual processing operations (e.g. multiplicative circuitry). However, there are still strong disadvantages. Speed of processing is relative to data value size (even with multiple channels for numerical encoding) and conditional on clock resolution. Storage requires amplification and is limited to re-circulation time.

In the short term, our next avenue for investigation is to implement the methods described in this paper as a virtual MISC processor. It is hoped that this will enable direct comparison with alternative architectures, and illuminate the selection of a medium for hardware implementation.

Longer term aims are to assess in more detail, and on a more practical level, the comparative advantages of this approach compared to the established norm.

There are clearly areas of computation [3] that lend themselves to analogue interpretation and the authors are interested in developing systems that model these in greater detail. Many biological system perform tasks that are presently proving difficult for digital technology. Perhaps moving to a fundamentally different representation as offered by encoding in *time* will make these problems more amenable and we will arrive at the best of both worlds - analogue and digital processing where best suited.

References

1. Shannon, C.E.: A symbolic analysis of relay and switching circuits. Electrical Engineering **57**(12) (1938) 713–723
2. Maass, W., Bishop, C.M., eds.: Pulsed Neural Networks. MIT Press, Cambridge, MA, USA (1999)
3. Hopfield, J.J., Brody, C.D., Roweis, S.: Computing with action potentials. In: Adv. Neural Inf. Processing 10, MIT Press (1998) 166–172
4. Burrell, M.: Fundamentals of Computer Architecture. Palgrave (2003)
5. Wilkes, M.V.: Computers then and now. J. ACM **15**(1) (January 1968) 1–7
6. Buckwalter, J., Hajimiri, A.: An active analog delay and the delay reference loop. In: Proc. of IEEE RFIC Symposium. (2004) 17–20
7. Gale, E., de Lacy Costello, B., Adamatzky, A.: Boolean logic gates from a single memristor via low-level sequential logic. CoRR - Computing Research Repository **abs/1402.4046** (2014)