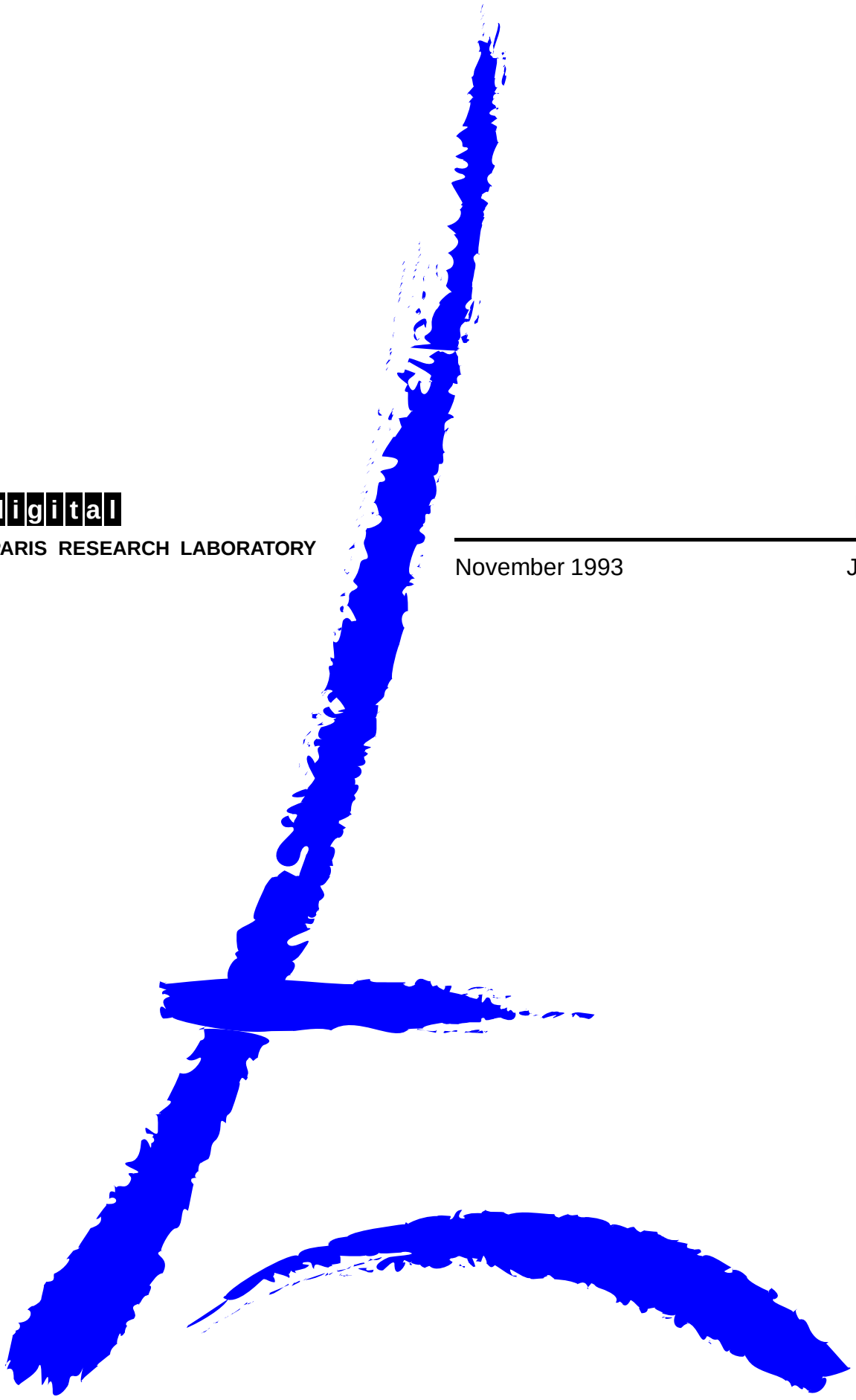


digital

PARIS RESEARCH LABORATORY

25

On Circuits and Numbers

November 1993

Jean Vuillemin

25

On Circuits and Numbers

Jean Vuillemin

November 1993

Publication Notes

The author may be contacted at the following address: Digital Equipment Corporation, Paris Research Laboratory, 85 Av. Victor Hugo, 92563 Rueil-Malmaison, Cedex France.

Abstract

We establish the following correspondences between the ring of *2adic integers* ${}_2\mathbf{Z}$ from arithmetics and *digital circuits* (finite and infinite, combinational and synchronous) from electronics (Theorems 1 and 2):

1. A function is computed by a *combinational* circuit if and only if it is *continuous* over the 2adic integers ${}_2\mathbf{Z}$:

$$\forall n \in \mathbf{N}, x \in {}_2\mathbf{Z}, \exists m \in \mathbf{N} : \; f(x) = f(x \bmod 2^m) \pmod{2^n}.$$

2. A function is computed by a *synchronous* circuit if and only if it is *on-line* over ${}_2\mathbf{Z}$:

$$\forall n \in \mathbf{N}, x \in {}_2\mathbf{Z} : \; f(x) = f(x \bmod 2^n) \pmod{2^n}.$$

   The proof of this result provides a SDD *normal form for synchronous circuits* which generalizes the BDD and TDG constructions (Algorithm 1) proposed by [A78], [B86] and [B87] for processing finite boolean functions. We show that the circuit SDD(f) synthesized by Algorithm 2 is *finite* if and only if function f may be realized by some finite state machine (Proposition 3).

From simple identities in the *ring* of 2adic integers, we derive both classical and new bit-serial circuits for computing: $(+, -, \times, 1/(1+2x), \sqrt{1+8x})$. All but the adders $(+, -)$ are *infinite* synchronous circuits (Proposition 4). In each case, the *correctness* of the circuit directly follows from the 2adic definition of the corresponding operator.

We demonstrate that our *2adic semantics* is fully general and may be fruitfully applied to *any* digital circuit. In particular, we characterize when the multiplexer and register commute with arbitrary logic; the retiming property, expressed by: $\forall x \in {}_2\mathbf{Z} : \; f(2x) = 2f(x)$, holds of an on-line function $f \in {}_2\mathbf{Z} \to {}_2\mathbf{Z}$ if and only if $f(0) = 0$ (Proposition 6). We also provide a simple characterization of *reversible* synchronous circuits (Proposition 7).

We introduce a general procedure (Algorithm 3) which transforms any synchronous circuit into an equivalent infinite parallel combinational implementation. Conversely, we show that every continuous function is computable by a synchronous circuit with output enable (Theorem 3).

We use reset signals in order to pipeline finite integer computations through arbitrary 2adic networks (Theorem 4); in this context, all arithmetic circuits become *finite*.

Résumé

Nous établissons les correspondances suivantes entre l'anneau des entiers 2adiques ${}_2\mathbf{Z}$ issu de l'arithmétique, et les circuits digitaux (finis et infinis, combinatoires et synchrones) issus de l'électronique (Théorèmes 1 et 2).

1. Une fonction est calculée par un circuit combinatoire si et seulement si elle est continue en tout $x \in {}_2\mathbf{Z}$ entier 2adique :

$$\forall n \in \mathbf{N}, x \in {}_2\mathbf{Z}, \exists m \in \mathbf{N} : \ f(x) = f(x \bmod 2^m) \pmod{2^n}.$$

2. une fonction est calculée par un circuit synchrone si et seulement si elle est en-ligne :

$$\forall n \in \mathbf{N}, x \in {}_2\mathbf{Z} : \ f(x) = f(x \bmod 2^n) \pmod{2^n}.$$

   La démonstration de ce résultat fournit une forme normale SDD pour les circuits synchrones, qui généralise les constructions BDD et TGD (Algorithme 1) proposées par [A78], [B86] et [B87] pour traiter des fonctions booléennes finies. Nous montrons que le circuit SDD synthétisé par l'Algorithme 2 est fini si et seulement si il peut être réalisé par une machine à état fini (Proposition 3).

A partir d'identités simples sur les entiers 2adiques, nous trouvons des circuits en série, à la fois classiques et nouveaux, pour calculer : $(+, -, \times, 1/(1+2x), \sqrt{1+8x})$. Mis à part les additionneurs $(+, -)$, tous ces circuits synchrones sont des circuits infinis (Proposition 4). Dans chaque cas, l'exactitude du circuit découle de la définition 2adique de l'opérateur correspondant.

Nous démontrons que notre sémantique 2adique est générale et peut s'appliquer avec profit à tout circuit digital. Plus précisément, nous caractérisons quand le multiplexeur et le registre binaire commutent avec une logique arbitraire. La propriété de "retiming" est exprimée par : $\forall x \in {}_2\mathbf{Z} : \ f(2x) = 2f(x)$. Elle vaut pour une fonction en-ligne $f \in {}_2\mathbf{Z} \to {}_2\mathbf{Z}$ si et seulement si $f(0) = 0$ (Proposition 6). Nous donnons aussi une caractérisation simple des circuits synchrones qui sont réversibles (Proposition 7).

Nous présentons une méthode générale qui transforme tout circuit synchrone en une réalisation combinatoire parallèle infinie. Inversement, nous montrons que toute fonction continue peut être calculée par un circuit synchrone avec validation des sorties (enable).

Nous utilisons des signaux de remise à zéro (reset synchrone) afin d'enchaîner (pipe-line) des calculs entiers finis au travers de réseaux 2adiques arbitraires. De cette façon, tous les circuits arithmétiques considérés deviennent finis.

Keywords

Synchronous circuits, combinatorial circuits, circuit synthesis, 2adic arithmetic, retiming, enable, reset, finite and infinite state machine.

Acknowledgements

This work has benefited from contributions by Patrice Bertin and Gérard Berry. I am indebted to Hervé Touati and Francois Bourdoncle for their sharp help with the proofs of Proposition 6 and Lemma 2. Thanks to Martin Abadi for pointing out the norm characterization of on-line functions.

Contents

1 Introduction

Modern electronic circuits fall in two categories, *analog* and *digital*.

The dynamic analysis of analog circuits involves physical parameters, such as currents and voltages, whose value $v_t \in \mathbf{R}$ varies continuously with *real time* $t \in \mathbf{R}$. Carver Mead's book [M89] provides an excellent introduction to analog circuits.

Digital circuits are characterized by a finite number of physical variables, whose value $v_t \in \mathbf{B} = \{0, 1\}$ may be identified with either zero or one, when properly discretized (say 0 when voltage $< 1V$ and 1 when voltage $> 2V$) at integer multiples ($t = n\Delta t$ for $n \in \mathbf{N}$) of the clock period Δt. Setting $\Delta t = 1$ through a suitable choice of the physical units allows us to identify *digital time* $t \in \mathbf{N} = \{0, 1, 2, \cdots\}$ with the set of natural numbers.

The present work is exclusively concerned with digital circuits, which we introduce mathematically as follows:

Definition 1 (Digital Circuit) *In a digital circuit C, the value of any variable* $v \in \mathcal{V}(C)$ *is a* bit $v_t \in \mathbf{B}$ *which may only change at integer time* $t \in \mathbf{N}$*:*

$$\forall v \in \mathcal{V}(C), t \in \mathbf{R}, \exists n = \lfloor t \rfloor \in \mathbf{N} : \quad v_t = v_n \in \mathbf{B}.$$

A direct consequence of Definition 1 is that all delays in a digital circuit are *exact integers*. In particular, *combinational* circuits have *zero delay*: the output response to changes in the inputs is instantaneous, and time plays no part in their mathematical analysis. With *synchronous* circuits, changes in the digital values of variables are equally instantaneous and they precisely occur at digital time $t \in \mathbf{N}$.

Of course, any physical implementation of our mathematical digital circuits has (hopefully very) small delays, but (certainly) not zero delays. As a consequence, the physical behaviour matches its mathematical idealization only when operated with a clock whose period $\Delta t > \delta$ exceeds the maximum physical delay δ (critical path).

Synchronous circuits operate upon *infinite* binary sequences: in any computation performed by circuit *C*, each variable $v \in \mathcal{V}(C)$ takes consecutive binary values $v_0, v_1, \cdots v_t, \cdots \in \mathbf{B}$ as digital time progresses through the natural numbers $t \in \mathbf{N}$. So, synchronous circuits naturally map infinite binary sequences, representing the successive input values at each clock tick $t \in \mathbf{N}$, into infinite binary sequences, representing the corresponding output values.

Infinite binary sequences have a rich mathematical structure, namely that of the *2adic integers* ${}_2\mathbf{Z}$, whose algebraic properties are presented in Section 2 (Propositions 1 and 2). Operations over ${}_2\mathbf{Z}$ contain most [1] of the usual operations over the ordinary integers $\mathbf{Z} = \{\cdots, -2, -1, 0, 1, 2, \cdots\}$ hence the name. In addition, they contain all set operations over the subsets $2^{\mathbf{N}}$ of the natural numbers. In short, the 2adic integers ${}_2\mathbf{Z}$ form both a *ring* $(0, +, -, 1, \times)$ and a *boolean algebra* $(\emptyset, \overline{\ }, \cup, \cap)$.

[1] but not all, as we loose integer comparison and division by a number which is not a power of 2;

Each digital circuit defines an operator over the 2adic integers, and the correspondence works in two *dual* ways:

1. In a combinational circuit $C \in \mathcal{C}(?)$ (Definition 6), where time takes no part, input variables $i[0], \cdots, i[n], \cdots \in \mathcal{V}(C)$ and output variables $o[0], \cdots, o[n], \cdots \in \mathcal{V}(C)$ are grouped *in space* so as to form the 2adic integers $I = \sum_{n \in \mathbf{N}} i[n]2^n$ and $O = \sum_{n \in \mathbf{N}} o[n]2^n$. We show (Theorem 1) that a function $O = C(I)$ may be so realized by a combinational circuit if and only if it is a *continuous* mapping over ${}_2\mathbf{Z}$.

2. In a synchronous circuit (Definition 8), the successive boolean values $v_0, \cdots, v_t, \cdots \in \mathbf{B}$ taken by each variable $v \in \mathcal{V}(C)$ at each clock tick $t \in \mathbf{N}$ are grouped *in time*, so as to form the 2adic integer $V = \sum_{t \in \mathbf{N}} v_t 2^t$. We show (Theorem 2) that a function may be so realized by a synchronous circuit if and only if it is an *on-line* mapping over ${}_2\mathbf{Z}$.

Combining these two results provides a simple and effective characterization of which functions *may* or *may not* be directly implemented by synchronous and combinational circuits, with many forthcoming concrete examples.

2 The 2adic integers ${}_2\mathbf{Z}$

The p-adic integers were introduced around 1900 by K. Hensel [H13] (for each prime $p \in \mathbf{N}$) and they play a central role in arithmetic (see [A75] and [K77]). Such numbers are obtained by extending indefinitely the ordinary base p representation, as computed by the rule:

$$\mathcal{B}_p(n) \quad \Rightarrow \quad (n \cdot|\cdot p)\, \mathcal{B}_p(n \div p). \tag{1}$$

We use $n \div p$ to represent the *quotient* and $n \cdot|\cdot p$ the *remainder* in the integer division of n by $p \neq 0$: $n = p \times (n \div p) + (n \cdot|\cdot p)$ with $0 \leq (n \cdot|\cdot p) < p$. For example, we compute the infinite binary ($p = 2$) representation of decimal number 22 by:

$$\mathcal{B}_2(22) \stackrel{*}{\Rightarrow} 011010\cdots0\cdots = {}_201101(0).$$

In the above equality, subscript ${}_2$ indicate the representation base 2 as well as the reading order, from low order bits to high order bits; the (0) denotes an infinite (periodic) sequence of zeroes. Similarly, we find:

$$\begin{array}{rrcll} & \mathcal{B}_2(-7) & \stackrel{*}{\Rightarrow} & 1001\cdots1\cdots & = {}_2100(1), \\ \text{and} & \mathcal{B}_2(\tfrac{1}{3}) & \stackrel{*}{\Rightarrow} & 110\cdots10\cdots & = {}_21(10). \end{array}$$

One may correctly infer from these three motivating examples that:

1. The infinite binary representation of a natural number, such as $0 = {}_2(0)$, is obtained by appending infinitely many non-significant zeroes to its ordinary finite binary representation.

2. The infinite binary representation of a negative integer, such as $-1 = {}_2(1)$, is obtained by extending indefinitely the most significant 1 bit from its ordinary two's complement finite representation.

3. Rational numbers having an *even* denominator, such as $\frac{1}{2}$, are *not* members of ${}_2\mathbf{Z}$: applying Rule (1) to these numbers fails to deliver a meaningful binary representation.

4. Rational numbers having an *odd* denominator, such as $-\frac{1}{7} = {}_2(100)$, are characterized by their *ultimately periodic* infinite binary representation. Equivalently, the *odd rationals* $\mathbf{Z}/1 + 2\mathbf{N}$ are precisely the members of ${}_2\mathbf{Z}$ which admit a *finite* notation upon adopting our parenthesis convention (Proposition 2); they are associated with finite synchronous circuits having constant inputs, or any equivalent explicit representation of finite automata.

5. The 2adic integers also contain non-periodic numbers, which the practical designer may choose to ignore since they are exclusively associated with *infinite* circuits, such as $\sqrt{-7} = {}_2 1010110100000010\cdots$

While Hensel's construction applies for any prime p, we need only concern ourselves with the case $p = 2$ of the 2adic integers ${}_2\mathbf{Z}$ for the purpose of studying digital circuits. The *arithmetic* properties of ${}_2\mathbf{Z}$ are (almost) similar to those of the p-adic integers for $p > 2$. The *logical* properties of ${}_2\mathbf{Z}$ are unique, a fact which we emphasize in the following definition.

Definition 2 *A 2adic integer $B \in {}_2\mathbf{Z}$ is the limit of three equivalent infinite sequences*

$$B = \lim_{n\mapsto\infty} {}_2 b_0 \cdots b_n = \lim_{n\mapsto\infty} b(n) = \lim_{n\mapsto\infty} b\{n\},$$

each composed of (for $n \in N$):

1. *bits $b_n \in \mathbf{B}$, with* $b_n = \big(B \div 2^n\big) \cdot|\cdot 2$;
2. *natural numbers $b(n) \in \mathbf{N}$, with* $b(n) = B \cdot|\cdot 2^{n+1} = \sum_{0\le k\le n} b_k 2^k$, *for $n \in \mathbf{N}$;*
3. *finite sets of integers $b\{n\} \subseteq \{0, 1, \cdots, n\}$, with* $b\{n\} = \{k \le n : \ b_k = 1\}$.

Let us make explicit the meaning of the word *limit* in Definition 2, by introducing a *distance* over binary sequences.

Definition 3 *The* valuation $v_2(b) \in \mathbf{N}$ *of a 2adic integer $b \in {}_2\mathbf{Z}$ is:*

1. *the index of the first non-zero bit in the binary representation of b;*
2. *the largest power of two which divides $b(n)$, for all $n \in \mathbf{N}$;*
3. *the smallest element in the set $x\{n\}$, for all $n \in \mathbf{N}$.*

The norm *of a 2adic integer* $x \in {}_2\mathbf{Z}$ *is defined by* ${}_2|x| = 2^{-v_2(x)}$.

The distance *between two 2adic integers* $x, y \in {}_2\mathbf{Z}$ *is the norm of their difference:* ${}_2|x-y|$.

Note that $v_2(0) = \infty$ so ${}_2|x| = 0$ if and only if $x = 0$, and $0 < {}_2|x| \leq 1$ for $x \neq 0$. Norm ${}_2|x|$ satisfies:

$$\begin{array}{ll}(i) & {}_2|x+y| \leq max\{{}_2|x|, {}_2|y|\};\\ (ii) & {}_2|x \times y| = {}_2|x| \times {}_2|y|;\end{array}$$

Property (i), which is characteristic of *ultra-metric* norms (see [K77]) is *stronger* than the classical triangle inequality for real numbers $\mathbf{R}$: $|x+y| \leq |x| + |y|$.

It follows from Definition 3 of the distance between 2adic integers that the *finite* approximants ${}_2b_0 \cdots b_n = b(n) = b\{n\}$ converge to number $B \in {}_2\mathbf{Z}$ according to:

$$\forall n \in \mathbf{N}: \ {}_2|B - {}_2b_0 \cdots b_n| = {}_2|B - b(n)| = {}_2|B - b\{n\}| \leq 2^{-n+1}.$$

So, each infinite binary sequence $A = (a_0 \cdots a_n \cdots)$ with $a_n \in \mathbf{B}$ for $n \in \mathbf{N}$ is *equal* to the *unique* 2adic integer $A = \sum_{n\in\mathbf{N}} a_n 2^n$ of which it is the base 2 representation, as computed by Rule (1) for $p = 2$. Two infinite binary sequences A, B are *equal* when ${}_2|A - B| = 0$, which is equivalent to each of the following:

$$A = B \Leftrightarrow \forall n \in \mathbf{N}: \ a_n = b_n \Leftrightarrow \forall n \in \mathbf{N}: \ a(n) = b(n) \Leftrightarrow \forall n \in \mathbf{N}: \ a\{n\} = b\{n\}.$$

This justifies to use of the following notations for representing a 2adic integer $B \in {}_2\mathbf{Z}$:

$$B \ = \ {}_2b_0 \cdots b_n \cdots \ = \ \sum_{n\in\mathbf{N}} b_n 2^n = \ \bigvee_{n\in\mathbf{N}} b(n) \ = \bigcup_{n\in\mathbf{N}} b\{n\}.$$

One may choose either of the proposed representations - bits or integers or sets - in order to introduce operations over ${}_2\mathbf{Z}$.

Definition 4 *Let* $A = \bigvee_{n\in\mathbf{N}} a(n) = \bigcup_{n\in\mathbf{N}} a\{n\}$ *and* $B = \bigvee_{n\in\mathbf{N}} b(n) = \bigcup_{n\in\mathbf{N}} b\{n\}$ *be two 2adic integers* $A, B \in {}_2\mathbf{Z}$. *We define the operations:*

- *complement* $\neg A = \bigcup_{n\in\mathbf{N}} \{k \leq n : \ k \notin a\{n\}\}$,
- *or* $A \vee B = \bigcup_{n\in\mathbf{N}} (a\{n\} \cup b\{n\})$,
- *and* $A \wedge B = \bigcup_{n\in\mathbf{N}} (a\{n\} \cap b\{n\})$.
- *sum* $A + B = \bigvee_{n\in\mathbf{N}} (a(n) + b(n)) \cdot|\cdot 2^{n+1}$,
- *difference* $A - B = \bigvee_{n\in\mathbf{N}} (a(n) - b(n)) \cdot|\cdot 2^{n+1}$,
- *product* $A \times B = \bigvee_{n\in\mathbf{N}} (a(n) \times b(n)) \cdot|\cdot 2^{n+1}$.

It follows from elementary set theory and arithmetic that:

Proposition 1 (Structure of ${}_2\mathbf{Z}$)

1. *The 2adic integers* $\left({}_2\mathbf{Z}, \neg, \vee, \wedge\right)$ *form a* boolean algebra, *isomorphic to* $\left(2^{\mathbf{N}}, \overline{}, \cup, \cap\right)$.
2. *The 2adic integers* $\left({}_2\mathbf{Z}, +, -, \times\right)$ *form a* ring *which contains the ordinary integers* $\mathbf{Z}$ *and the odd rationals* $\mathbf{Z}/1 + 2\mathbf{N}$ *as proper sub-rings.*

Let us characterize further the set inclusions $\mathbf{Z} \subset \mathbf{Z}/1 + 2\mathbf{N} \subset {}_2\mathbf{Z}$, both arithmetically and in terms of synchronous circuits (forward Definition 8):

Proposition 2 *Assertions (i), (ii), (iii) and (iv) are equivalent:*

(i) *A 2adic integer* $B \in {}_2\mathbf{Z}$ *is an ordinary integer* $B \in \mathbf{Z}$.

(ii) $\exists z \in \mathbf{Z}, \forall k \in \mathbf{N} :\ B = z \pmod{2^{k+1}}$.

(iii) *The binary representation of* $B = {}_2b_0 \cdots b_{l-1}\left(b_l\right)$ *is* ultimately constant*:*
$\exists l \in \mathbf{N}, \forall k \geq l :\ b_k = b_l$*; here,* $l + 1$ *is the (ordinary) binary length of* $z \in \mathbf{Z}$.

(iv) $B = \sum_{t \in \mathbf{N}} b_t 2^t$ *is the output of some* finite acyclic *synchronous circuit with constant inputs, either* $0 = {}_2\left(0\right)$ *or* $-1 = {}_2\left(1\right)$.

Assertions (v), (vi), (vii) and (viii) are equivalent:

(v) *A 2adic integer* $B \in {}_2\mathbf{Z}$ *is an odd rational* $B \in \mathbf{Z}/1 + 2\mathbf{N}$.

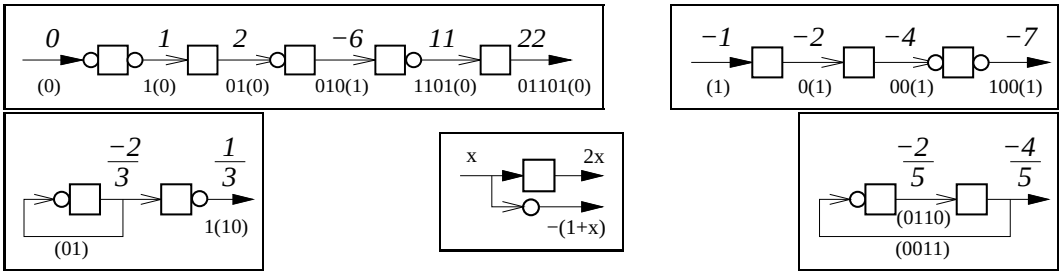
(vi) $\exists z \in \mathbf{Z}, n \in \mathbf{N}, \forall k \in \mathbf{N} :\ B \times \left(1 + 2n\right) = z \pmod{2^{k+1}}$.

(vii) *The binary representation of* $B = {}_2b_0 \cdots b_{i-1}\left(b_i \cdots b_{i+p-1}\right)$ *is* ultimately periodic*:*
$\exists i \in \mathbf{N}, p \in \mathbf{N} + 1\ \forall k \geq i :\ b_k = b_{k+p}$.

(viii) $B = \sum_{t \in \mathbf{N}} b_t 2^t$ *is the output of some* finite *synchronous circuit with constant inputs.*

As a consequence of this fact, we can systematically label each variable in a synchronous circuit having constant inputs with odd rationals, starting from $0 = {}_2\left(0\right)$ for the electrical *ground* and $-1 = {}_2\left(1\right)$ for the electrical *power supply*. In the following example, labels are given both in decimal and in binary,(low order bit first, periodic part in parenthesis).

Example 2.1 The numbers $22, -7, \frac{1}{3}$ and $-\frac{4}{5}$ as circuits:

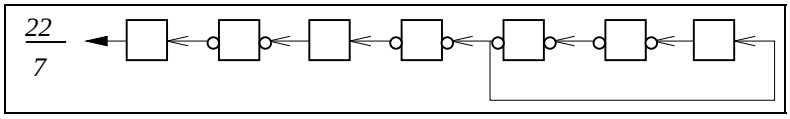


In our schemas: circles denote inverters and squares denote binary registers,*a.k.a.* synchronous flip-flop with initial value *zero* (Example 2.1). A register with inverters on both sides is easily recognized as a synchronous flip-flops with initial value *one* (Example 4.2).

The circuits presented in the proof (Section 8.1) of the Paragraphs (iii) and (vi) in Proposition 2 are precisely those constructed by the SDD procedure (Algorithm 2) upon constant (arity=0) input $f() = z \in \mathbf{Z}$ and $f() = \frac{z}{1+2n} \in \mathbf{Z}/1+2\mathbf{N}$. It establishes a direct correspondence between the ultimately periodic binary representation of an odd rational, such as

$$\frac{22}{7} = {}_20101(110),$$

and its realization by a finite synchronous circuit containing exactly *one* loop, such as:



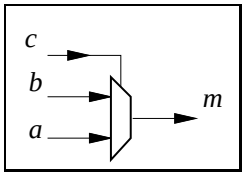
3 Combinational Circuits

Combinational circuits can all be built from a single atomic gate, the *multiplexer*.

Definition 5 (The Multiplexer $? \in \mathbf{B}^3 \to \mathbf{B}$) *mux*
The value of output $m \in \mathbf{B}$ *is determined from the three input values* $c, b, a \in \mathbf{B}$ *by:*

$$m = ?(c, b, a) = \begin{cases} b & \text{if } c = 1, \\ a & \text{if } c = 0. \end{cases}$$

$\boxed{m = ?(c,b,a)}$



Note the relations: $?(c, b, a) = (c \wedge b) \vee (\neg c \wedge a) =$ if c then b else a.
Together with 0 and 1, the multiplexer is a *basis* for boolean algebra, as seen from:

$\boxed{\neg b = ?(b, 0, 1)}$ $\boxed{a \vee b = ?(a, a, b)}$ $\boxed{a \wedge b = ?(a, b, a)}$

The *exclusive or* $a \oplus b = (a \wedge \neg b) \vee (\neg a \wedge b)$ is realized with two muxes:

$$\boxed{a \oplus b = ?(a, ?(b, 0, 1), b)}$$

We construct arbitrarily complex boolean functions by wiring together mux gates, subject to the following caveat:

1. the graph of the circuit is acyclic;

2. every path between inputs and outputs is finite.

Condition 1 eliminates from consideration cyclic structures, such as $u = ?\big(u, 0, 1\big)$ whose output value is *undefined* in the boolean domain. Condition 2 guarantees that the value of each variable may be computed within a *finite* combinational delay. In particular, this ensures that any *finite* combinational circuit may be operated reliably with a synchronous clock having a finite period $\Delta t > \delta$ greater than the circuit's largest delay δ (critical path).

Definition 6 (Combinational circuits $\mathcal{C}(?)$) *A* combinational circuit $C \in \mathcal{C}\big(\mathcal{I}, ?\big)$ *is a set* $\mathcal{V}\big(C\big) = \mathcal{I} \cup \mathcal{M}$ *of digital variables partitioned into:*

- *Inputs* $\mathcal{I} = \{i[0] \cdots i[n] \cdots\}$*; they may take arbitrary boolean values* $i[n] \in \mathbf{B}$.
- *Multiplexers* $\mathcal{M} = \{m[0] \cdots m[n] \cdots\}$*; each is defined by a mux equation:*

$$\forall m \in \mathcal{M}, \exists c, a, b \in \mathcal{V}\big(C\big) : \quad m = ?\big(c, b, a\big).$$

The mux-ordering $\prec$ *induced over the variables* $\mathcal{V}\big(C\big)$ *by*

$$m = ?\big(c, b, a\big) \text{ implies } c \prec m, b \prec m \text{ and } a \prec m,$$

must be well-founded: *every descending chain* $v[1] \succ \cdots \succ v[n] \succ \cdots$ *is* finite.

The circuit's outputs $\mathcal{O} = \{o[0] \cdots o[n] \cdots\} \subseteq \mathcal{V}\big(C\big)$ *form a subset of the variables.*

Note that Definition 6 allows for infinite as well as finite circuits. From an arbitrary assignment of boolean values to the inputs, corresponding to the 2adic integer $I = \sum_{n \in \mathbf{N}} i[n] 2^n$, we can follow the mux-ordering in order to compute the value of each variable in $\mathcal{M}$, so as to obtain the 2adic integer $O = \sum_{n \in \mathbf{N}} o[n] 2^n$ which represents the circuit's output response.

Theorem 1 *A function is computable by some combinational circuit* $f \in \mathcal{C}\big(?\big)$ *if and only if the mapping* $f \in {}_2\mathbf{Z} \to {}_2\mathbf{Z}$ *defined by* $f\big(\sum_{n \geq 0} i[n] 2^n\big) = \sum_{n \geq 0} o[n] 2^n$ *is* continuous*, that is:*

$$\forall I \in {}_2\mathbf{Z}, \epsilon \in \mathbf{R} > 0, \exists \eta \in \mathbf{R}, \forall I' \in {}_2\mathbf{Z} : \ {}_2|I - I'| < \eta \text{ implies } {}_2|f\big(I\big) - f\big(I'\big)| < \epsilon. \qquad (2)$$

The proof of this result is given in Section 8.2.

Any finite boolean function, such as the full-adder from Example 3.1, is continuous; it is well known (Algorithm 1) that each may be computed by a finite combinational circuit.

On-line functions (Theorem 2) such as $+, -, \times$ are continuous. The corresponding infinite combinational circuit may be directly constructed from Theorem 1. We may also start from a bit-serial synchronous circuit implementing the on-line function (Theorem 2), and apply the parallelization procedure of Section 5 (Algorithm 3) in order to obtain an equivalent

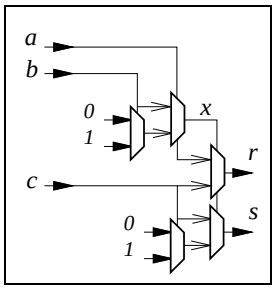
(infinite) combinational implementation. The parallel adder from Example 3.2 is compiled by Algorithm 3 from the serial adder of Section 5.1, and the parallel multiplier (Example 6.1) from the serial-parallel multiplier in Section 5.3.

Example 3.3 presents examples of continuous functions, the Peano projections, which are not on-line: they may be realized by an infinite combinational circuit; they may not be realized by any synchronous circuit.

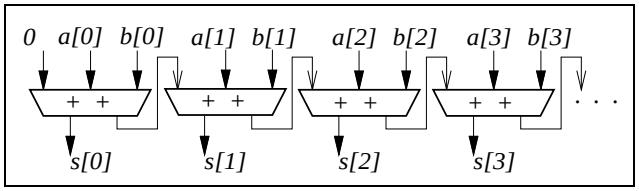
The test for zero function $z \in {}_2\mathbf{Z} \to {}_2\mathbf{Z}$ defined by $z(0) = 0$ and $z(x) = -1$ for $x \neq 0$ is *not* continuous at 0, therefore, is is not computable by *any* digital circuit. The related z' defined by $z'(0) = 0$ and $z'(2^v(1+2x)) = -2^v$ is continuous (and on-line); so, it is computable by a combinational circuit $z' \in \mathcal{C}(?)$ (and by a synchronous circuit $z' \in \mathcal{C}(?, 2\times)$).

Example 3.1 (Full Adder)
The full-adder computes the unique boolean solution to the equation $\boxed{a+b+c = s+2r}$, namely $s = a \oplus b \oplus c$ and $r = (a \wedge b) \vee (b \wedge c) \vee (c \wedge a)$. The following circuit implements the function with a *minimal* number of 5 muxes:

$$
\begin{array}{lrcl}
\mathit{FullAdd}(a,b,c) & = & (s,r) \\
\text{where} & \{\overline{b} & = & ?(b,0,1), \\
& x & = & ?(a,b,\overline{b}), \\
& \overline{c} & = & ?(c,0,1), \\
& r & = & ?(x,a,c), \\
& s & = & ?(x,c,\overline{c})\}
\end{array}
$$



Example 3.2 (Parallel Adder) From here on, a trapezoid with two + signs represents a full-adder:



While this circuit contains an infinite carry chain, the longest path of muxes leading from inputs to each output is finite: $n+3$ for $n \geq 1$, with the full-adder from Example 3.1.

Example 3.3 (Peano Pairing) Define the *cartesian product* $\pi \in {}_2\mathbf{Z} \times {}_2\mathbf{Z} \to {}_2\mathbf{Z}$ of two 2adic integers by *interleaving* the binary representations of each operand. The inverse first projection $\pi_0 \in {}_2\mathbf{Z} \to {}_2\mathbf{Z}$ extracts the *even* bits, and the second $\pi_1 \in {}_2\mathbf{Z} \to {}_2\mathbf{Z}$ the *odd* bits. They are computed by the recursive system:

$$\pi(a,b) \;=\; a \cdot|\cdot\, 2 + 2 \times \pi(b, a \div 2),$$

$$\begin{aligned}\pi_0(a) &= a \cdot|\cdot 2 + 2 \times \pi_0(a \div 4),\\ \pi_1(a) &= \pi_0(a \div 2).\end{aligned}$$

We easily verify that $\forall a, b \in {}_2\mathbf{Z}:\ \pi_0(\pi(a,b)) = a,\ \pi_1(\pi(a,b)) = b$ and that each of $\pi, \pi_1, \pi_2 \in \mathcal{C}(?)$ may be realized by a *mux-less* combinational circuit (just wires). Note that π establishes a one to one mapping $\mathbf{N} \leftrightarrow \mathbf{N} \times \mathbf{N}$ between natural numbers and pairs of natural numbers; similarly for the odd rationals $\mathbf{Z}/1 + 2\mathbf{N}$ and the 2adic integers ${}_2\mathbf{Z}$.

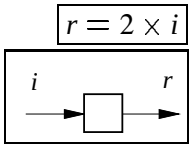
4 Synchronous Circuits

Synchronous circuits are built from two atomic gates, the *multiplexer* and the *register*, also known as flip-flop. When operating in a synchronous environment, the multiplexer retains its zero-delay (mathematical) property: the output value $m_t \in \mathbf{B}$ is determined, at all times $t \in \mathbf{N}$, from the input values $c_t, b_t, a_t \in \mathbf{B}$ by:

$$m_t = ?(c_t, b_t, a_t) = \text{ if } c_t \text{ then } b_t \text{ else } a_t.$$

We need to change our Definition of the logical complement to $\boxed{\neg b = ?(b, 0, -1)}$. This accounts for the difference between $-1 = {}_2(1)$, the electrical power supply, and $1 = {}_21(0)$, the *boot* signal. The register introduces a *unit time delay*:

Definition 7 (The Register $2\times \in {}_2\mathbf{Z} \to {}_2\mathbf{Z}$) *reg*
The output r_t of a register is 0 at initial time: $r_0 = 0$; for $t \geq 1$, it is equal to the value i_{t-1} of its input, sampled at the previous clock tick: $\forall t \in \mathbf{N} + 1,\ r_t = i_{t-1}$.



As we represent the input sequence by the 2adic integer $i = \sum_{t\in\mathbf{N}} i^t 2^t$, the 2adic integer $o = \sum_{t\in\mathbf{N}} o^t 2^t$ representing the output of the register is equal to $r = 2 \times i$.

Complex synchronous circuits are realized by wiring together a number of muxes and registers. All registers are *synchronous* in that they share the same clock signal. As before, no infinite combinational path is allowed.

Definition 8 *A* synchronous circuit $C \in \mathcal{C}(?, 2\times)$ *is a set $\mathcal{V} = \mathcal{V}(C) = \mathcal{I} \cup \mathcal{M} \cup \mathcal{R}$ of digital variables made of:*

- *Inputs $\mathcal{I} = \{i[0] \cdots i[i-1]\}$ in* finite *number $i = |\mathcal{I}|$; they may take arbitrary 2adic integer values.*
- *Muxes $\mathcal{M} = \{m[0] \cdots m[n] \cdots\}$; each is defined by a mux equation:*

  $$\forall m \in \mathcal{M}, \exists c, a, b \in \mathcal{V}(C):\ \ m = ?(c, b, a).$$

  The mux-ordering must be well-founded, as in Definition 6.

- *Registers* $\mathcal{R} = \{r[0] \cdots r[n] \cdots\}$*; each is defined by a* reg *equation:*

$$\forall r \in \mathcal{R}, \exists i \in \mathcal{V}(C) : \; r = 2 \times i.$$

The outputs $\mathcal{O} = \{o_0 \cdots o_{o-1}\} \subseteq \mathcal{V}(C)$ *form a* finite *subset* $o = |\mathcal{O}|$ *of the variables.*

Like combinational circuits, synchronous circuits may be finite or infinite; unlike combinational circuits, synchronous circuits have only finitely many inputs and outputs.

When the graph of a synchronous circuit contains loops, it is straightforward to verify from the well-founded mux ordering that each loop through the circuit must traverse at least one register.

From an arbitrary assignment of boolean values for each time $t \in \mathbf{N}$ to the inputs, corresponding to the 2adic integers $I[j] = \sum_{t \in \mathbf{N}} i_t[j] 2^t$ for $0 \leq j < i = |\mathcal{I}|$, we may compute, from $t = 0, 1, 2, \cdots$ on, the values of each mux and reg in circuit $C \in \mathcal{C}(?, 2\times)$, so as to obtain the 2adic integers $O[j] = \sum_{t \in \mathbf{N}} o_t[j] 2^t$ for $0 \leq j < o = |\mathcal{O}|$ which represent the circuit's output responses.

Theorem 2 *A function is computed by k inputs, one output synchronous circuit* $f \in \mathcal{C}(?, 2\times)$ *if and only if the mapping* $f \in {}_2\mathbf{Z}^k \to {}_2\mathbf{Z}$ *defined by*

$$f\Big(\sum_{t \in \mathbf{N}} i_t[0] 2^t, \cdots, \sum_{t \in \mathbf{N}} i_t[k-1] 2^t\Big) = \sum_{t \in \mathbf{N}} o_t 2^t$$

is on-line*, that is:*

$$\forall t \in \mathbf{N}, I[0], \cdots, I[k-1] \in {}_2\mathbf{Z} :$$
$$f\big(I[0] \cdot|\cdot 2^{t+1}, \cdots, I[k-1] \cdot|\cdot 2^{t+1}\big) = f\big(I[0], \cdots, I[k-1]\big) \pmod{2^{t+1}}.$$

The direct implication simply expresses that the outputs of a synchronous circuit at time $t \in \mathbf{N}$ may only depend upon the values of its inputs during the first t clock cycles.

The converse implication is proved in Section 8.3 by constructing *synchronous decision diagrams* (SDD Algorithm 2) which synthesize synchronous circuits. The SDD generalize the *binary decision diagrams* (BDD Algorithm 1), introduced by [A78], [B86] and [B87] for representing and manipulating finite boolean circuits. While it is by nature an *infinite* process, algorithm $SDD(f)$ generates a finite synchronous circuit (in finite time) if and only if function f may be defined by a *finite automaton.*

Proposition 3 (Finite State Machines) *An on-line function f is representable by some finite state machine if and only if the SDD (Algorithm 2) generates a* finite *synchronous circuit* $SDD(f) \in \mathcal{C}(?, 2\times)$.

When applied to a finite purely combinational function (no registers, finitely many muxes and constants), the SDD procedure (Algorithm 2) generates the same circuit as the BDD procedure (Algorithm 1).

The SDD procedure provides a *normal form* for synchronous circuits. The technical restriction imposed on circuits by this normal form are:

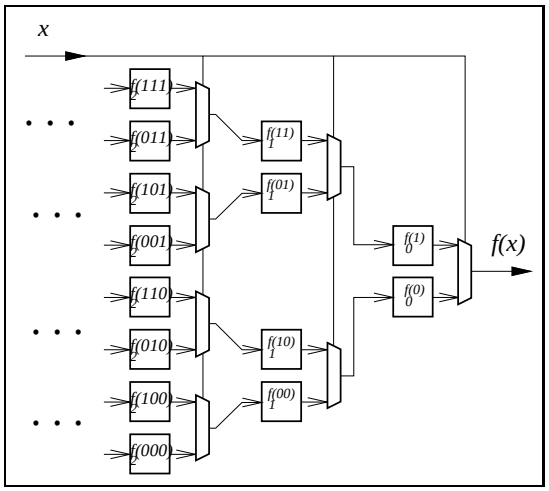
1. the control input c of each mux $m = ?(c, b, a)$ must be one of the circuit's primary inputs $c \in \mathcal{I}$;
2. along with the multiplexer and the register, we take the logical negation $\neg x$ in our set of primitive operators; as usual, it is schematized by a small circle.

Condition 1 is used by Akers [A78] and Bryant [B86] for establishing the BDD normal form. The negation is introduced by Billon [B87] who defines TDG (typed decision graphs) and shows that TDGs keep all the advantages of BDDs, and further reduce the circuit's size.

The structure of the *universal* synchronous circuit for computing an arbitrary on-line function

$$f(x) = \sum_{t \in \mathbf{N}} f_t(x_0, \cdots, x_t) 2^t,$$

where $f_t \in \mathbf{B}^{t+1} \to \mathbf{B}$ for $t \in \mathbf{N}$, is the following infinite binary tree structured circuit, result of SDD(f); the labels inside the squares represent the initial value (0 or 1) of the corresponding register.



Condition 1 guarantees that all SDD circuits having a small number of primary inputs are *electrically fast*. Though the method has not yet been thoroughly tested, it gives almost optimal results when hand-applied to the synthesis of the simple arithmetic functions $1 + 2x$, $1 + x$ and $x + y$ (Examples 4.2, 4.3, 4.4). The procedure works equally well on 2x and $1 + 2x$ (Example 4.2).

Example 4.1 shows that the BDD procedure is not optimal: it synthesizes 7 muxes for the full adder, instead of 5 in Example 3.1. Similarly, Example 4.5 shows that SDD(3x) has 8 muxes and 5 registers, compared to 5 muxes and 1 register in Section 5.1.

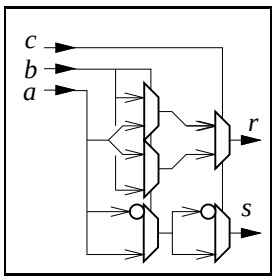
The arithmetic functions $\times$, $i(x) = \frac{1}{1+2x}$ and $r(x) = \sqrt{1+8x}$ are on-line. Although they are amenable to the SDD algorithm, we construct in Section 5 simpler bit-serial circuits for implementing these arithmetic operations.

Function $f(x) = x \div 2$ is *not* on-line since $f(2) \neq f(2 \cdot|\cdot 2) \pmod 2$. Indeed, this *anti-flop* is not computable by *any* synchronous circuit. The related $f(x) = 2 \times (x \div 2) = x - (x \cdot|\cdot 2)$ is an on-line function.

While Peano's cartesian product $\pi \in \mathcal{C}(?, 2\times)$ is synchronous, neither is π_0 nor π_1. We leave it as an interesting design exercise (for the reader) to realize π by a synchronous circuit.

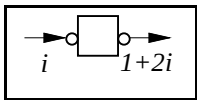
Example 4.1 (FullAdder) The following 7 muxes result from applying the BDD procedure (Algorithm 1) to the synthesis of the full-adder.

$$
\begin{array}{lrcl}
\mathit{FullAdd}(a,b,c) & = & (s,r) \\
\text{where} & \{o & = & ?(b,b,a), \\
& e & = & ?(b,a,b), \\
& r & = & ?(c,o,e), \\
& \overline{a} & = & \neg a, \\
& x & = & ?(b,\overline{a},a), \\
& \overline{x} & = & \neg x, \\
& s & = & ?(c,\overline{x},x)\}
\end{array}
$$



Example 4.2 (Register with initial value 1)
The circuit to the right computes $o = 1 + 2 \times i$

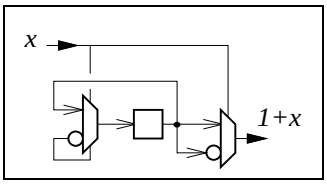
$$
\begin{array}{lrcl}
& \{\overline{i} & = & ?(i,0,-1), \\
\text{where} & 2\overline{i} & = & 2 \times \overline{i}, \\
& o & = & ?(2\overline{i},0,-1)\}
\end{array}
$$



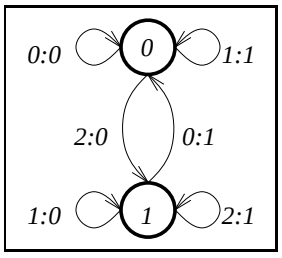
Example 4.3 (Serial increment) The following circuit results from applying the SDD procedure to the synthesis of the increment function $1 + x$.

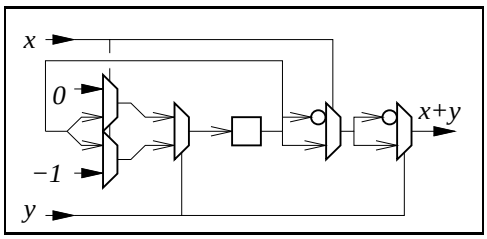
$$
\begin{array}{lrcl}
\{\mathrm{SDD}(1+x) & = & ?(x,v,\neg v), \\
v & = & 2\times ?(x,v,\neg x)\}
\end{array}
$$

Note the relation $v = -2^{v_2(x)+1}$, where $v_2(x)$ is the valuation of x (Definition 3).



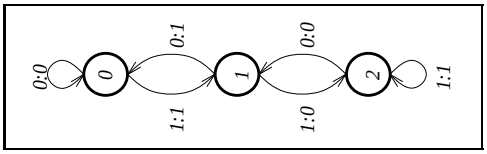
Example 4.4 (SDD Adder) Apply the SDD procedure to the synthesis of a serial adder (see Section 5.1) computing $x + y$. The 2 states finite automaton and SDD($x + y$) are:

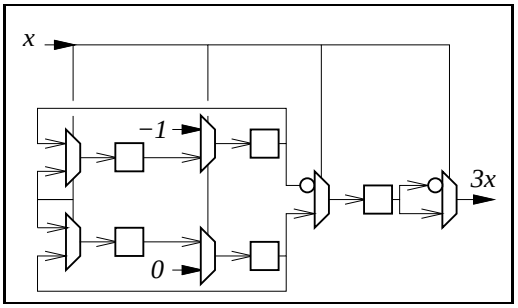




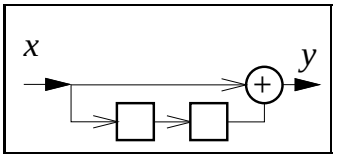
In our finite state diagrams, circles represent states (numbered inside); the 2:0 label on a transition arc means that both inputs are one, and the output zero (0:1 means that both inputs are 0, and the output is 1).

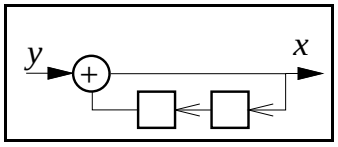
Example 4.5 (Times 3) The following circuit, to be compared with the simpler realization in Example 5.1, results from computing SDD(3x).





Example 4.6 ($\otimes$ 5) The reversible circuit $y = C(x) = x \oplus 4x$ implements a convolution code with generator $g(x) = 1 + x^2$; its inverse by Proposition 7 is $x = C^{-}(y) = y \oplus 4x$.



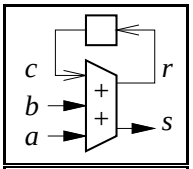


5 Arithmetic circuits

We now introduce bit-serial synchronous circuits for computing the arithmetic operations $+, -, \times, \frac{1}{1+2x}$ and $\sqrt{1+8x}$. From each of these synchronous circuits, we may derive a fully parallel combinational implementation through a systematic circuit transformation procedure (Algorithm 3). The *only* finite circuits in this section are the serial + and $-$, since:

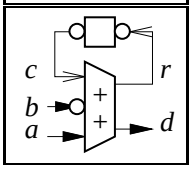
Proposition 4 *Any synchronous circuit for squaring a 2adic integer contains infinitely many registers.*

5.1 Addition



The basic arithmetic invariant of the full-adder is: $a+b+c = s+2r$. We solve this system by letting $c = 2r$, and define addition by: $s = a + b$ where $(s, c) = \mathit{FullAdd}(a, b, 2 \times c)$.

5.2 Subtraction



Binary substraction is computed as $a - b = a + (-b)$, with $-b = 1 + \neg b$ the opposite of b. So, define substraction by $d = a - b$ where $(d, c) = \mathit{FullAdd}(a, \neg b, 1 + 2 \times c)$.

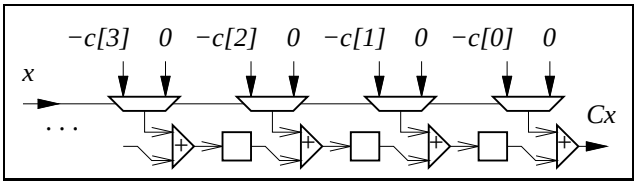
From now on, a triangle with a + inside denotes a serial adder, a serial substracter with a $-$.

5.3 Serial-Parallel Multiplier

In order to construct a synchronous circuit which multiplies input $x \in {}_2\mathbf{Z}$ by some fixed 2adic integer $C = \sum_{k\in\mathbf{N}} c[k]2^k$, consider the following elementary identity:

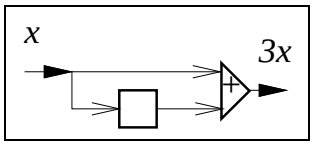
$$C \times x = \big(c[0] + 2\big(C \div 2\big)\big) \times x = \big(-c[0]\big) \wedge x + \big(C \div 2\big) \times 2x.$$

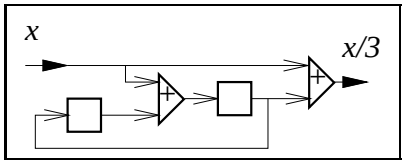
This provides a direct recursive definition for the following infinite multiplier:



When C is constant, slices corresponding to $c[n] = 0$ simplify to a single register, and all muxes may be eliminated. When $C = {}_2c_0 \cdots c_{i-1}\big(c_i \cdots c_{i+p-1}\big)$ is a constant odd rational, the multiplier becomes finite with $i + p$ slices: the input to the last slice is the output from the i-th slice. When $C = {}_2c_0 \cdots c_{e-1}\big(c_e\big)$ is a constant integer, the multiplier becomes identical to the classical two's complement Lyon's multiplier [Ly81].

Example 5.1 The following circuits compute respectively $3 \times x = {}_211\big(0\big) \times x$ and $x/3 = {}_21\big(10\big) \times x$:





5.4 Serial-Serial Multiplier

Let $x = {}_2x_0 \cdots x_t \cdots$ and $y = {}_2y_0 \cdots y_t \cdots$ be the operands to be multiplied in order to compute serially the product $p = x \times y = {}_2p_0 \cdots p_t \cdots$. The invariant of this synchronous multiplier $M \in \mathcal{C}\big(?, 2\times\big)$ is:

$$M\big(2^t, x, y\big) = 2^t\big(x \div 2^t\big) \times \big(y \div 2^t\big).$$

From the elementary identity in the 2adic ring,

$$\big(x \div 2^t\big) \times \big(y \div 2^t\big) = x_t y_t + 2x_t\big(y \div 2^{t+1}\big) + 2y_t\big(x \div 2^{t+1}\big) + 4\big(x \div 2^{t+1}\big) \times \big(y \div 2^{t+1}\big),$$

we derive the recurrence relation,

$$M\big(2^t, x, y\big) = A\big(2^t, x, y\big) + 2M\big(2^{t+1}, x, y\big),$$

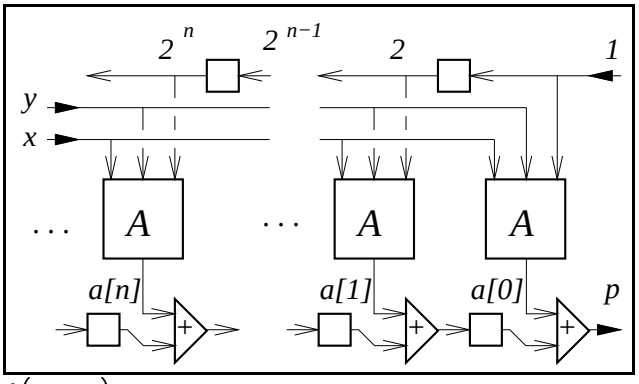
where A is the auxiliary function:

$$2^{-t}A\big(2^t, x, y\big) = a[t] = x_t y_t + 2x_t\big(y \div 2^{t+1}\big) + 2y_t\big(x \div 2^{t+1}\big). \qquad (3)$$

The final product is obtained as $p = M(1, x, y) = \sum_{t\in\mathbf{N}} a[t]2^t$, where:

$$M(c, x, y) = A(c, x, y) + 2M(2 \times c, x, y).$$

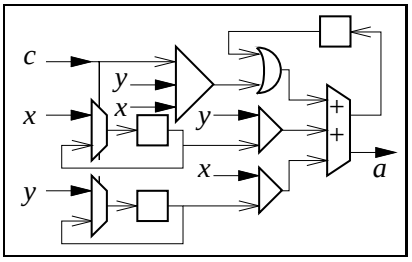
The resulting cellular structure looks like:



In order to design cell A, rewrite (3) as $A(c, x, y) =$

$$(x \wedge y \wedge c) + x \wedge (-2(c \wedge y)) + y \wedge (-2(c \wedge x))$$

which is equal to $2^t a[t]$, provided that $c = 2^t$. This equation translates to the finite circuit to the right, where triangles denote *and* gates, and half-circles *or* gates.



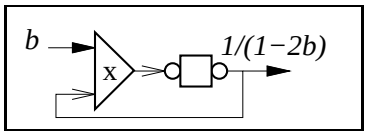
This design leads to more economical circuits than the Atrubin (see [K81]) or the Chen and Willoner constructions [CW79].

5.5 Odd Inverse

An even 2adic integer $b = 2b' \in {}_2\mathbf{Z}$ has *no* inverse $b^- \in {}_2\mathbf{Z}$: indeed $2b \times b^-$ is even and $2b \times b^- \neq 1$ for all b^-. So ${}_2\mathbf{Z}$ is not a field, but it comes close: we can define the *odd inverse* $i = 1/(1 - 2b) \in {}_2\mathbf{Z}$ of any 2adic integer $b \in {}_2\mathbf{Z}$ by the formula:

$$i = \frac{1}{1-2b} = \sum_{k\in\mathbf{N}} (2b)^k. \tag{4}$$

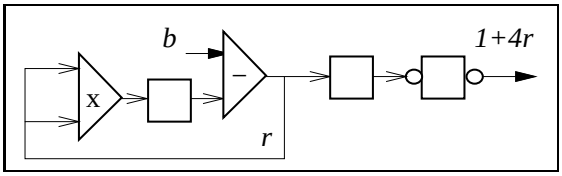
Rewriting (4) as $i(1-2b) = 1$, we obtain $i = 1+2ib$ which translates to the synchronous circuit to the right. While it looks finite in the picture, this is yet another infinite circuit since it contains the serial multiplier from the previous section.



5.6 Square Root

An odd 2adic integer has a square root if and only if it is congruent to one modulo 8 (Proposition 8). Such a number $1 + 8b \in {}_2\mathbf{Z}$ has exactly two square roots: $(+\sqrt{1+8b}) \cdot | \cdot 4 = 1$

and $\left(-\sqrt{1+8b}\right)\cdot|\cdot 4 = 3$. We compute the former $\sqrt{1+8b} = 1+4r$ so as to verify: $\left(1+4r\right)^2 = 1+8r+16r^2 = 1+8b$. Simplifying this last expression to $r+2r^2 = b$, we see that the square root $1+4r = +\sqrt{1+8b}$ is given by $r = b - 2r^2$ which translates to the following (infinite) synchronous circuit:



6 Applications

This section groups various applications of the 2adic theory to practical circuit design problems.

The following corollary to Theorems 1 and 2 provides an explicit representation of the functions computed by digital circuits in terms of finite boolean functions. It is stated for unary ($k = 1$) functions, the generalization ($k > 1$) being straightforward. A combinational circuit is said to be *finite* if it has finitely many inputs and outputs.

Proposition 5 *Function $f \in {}_2\mathbf{Z} \to {}_2\mathbf{Z}$ is computable by*

- *a* finite *combinational circuit if and only if it is* bitwise*:*

$$\exists f' \in \mathbf{B} \to \mathbf{B} : \; f\big(\sum_{t\in\mathbf{N}} x_t 2^t\big) = \sum_{t\in\mathbf{N}} f'\big(x_t\big)2^t;$$

- *a synchronous circuit if and only if it is* on-line*:*

$$\forall t \in N, \exists f_t \in \mathbf{B}^{t+1} \to \mathbf{B} : \; f\big(\sum_{t\in\mathbf{N}} x_t 2^t\big) = \sum_{t\in\mathbf{N}} f_t\big(x_0,\cdots,x_t\big)2^t;$$

- *an* infinite *combinational circuit if and only if it is* continuous*:*

$$\forall t \in \mathbf{N}, \exists m\big(t\big) \in \mathbf{N}, f_t \in \mathbf{B}^{m(t)+1} \to \mathbf{B} : f\big(\sum_{t\in\mathbf{N}} x_t 2^t\big) = \sum_{t\in\mathbf{N}} f_t\big(x_0,\cdots,x_{m(t)}\big)2^t.$$

6.1 Commutation with mux, reg and retiming

The following general commutation properties between mux, reg and arbitrary functions play an important role in the optimization of electrical delays in digital circuits (see [LS91]). To simplify notations, we state them for functions having $k = 2$ arguments, the generalization to arbitrary $k \geq 0$ being straightforward.

Proposition 6 • *A continuous function* $f \in {}_2\mathbf{Z}^2 \to {}_2\mathbf{Z}$ *commutes with the mux*

$$\forall a, b, c, d, e \in {}_2\mathbf{Z} : ?\big(a, f\big(b, c\big), f\big(d, e\big)\big) = f\big(?\big(a, b, d\big), ?\big(a, c, e\big)\big)$$

if and only if it is a finite boolean function.

• *An on-line function* $f \in {}_2\mathbf{Z}^2 \to {}_2\mathbf{Z}$ *commutes with the register*

$$\forall a, b \in {}_2\mathbf{Z} : \; 2 \times f\big(a, b\big) = f\big(2 \times a, 2 \times b\big)$$

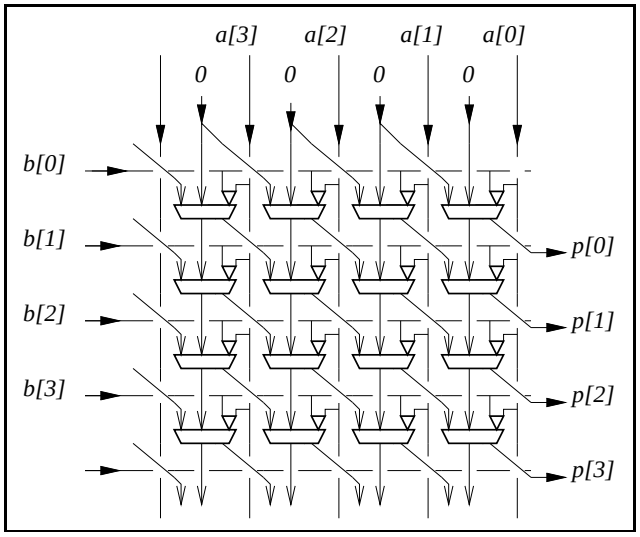
if and only if $f\big(0, 0\big) = 0$.

6.2 Parallelization of synchronous circuits

Since any *on-line* function is also *continuous*, it may be computed by a synchronous circuit as well as by an infinite combinational circuit. The procedure (Algorithm 3) which translates one into the other by *unfolding time into space* is detailed in Section 8.4. The application of this procedure to the synthesis of a parallel adder (from the serial adder of Section 5.1) is given in Example 3.2; the synthesis of a parallel multiplier (from the serial-parallel multiplier in Section 5.3) is given in Example 6.1. In both cases, the longest path (electrical delay) between inputs and the *n*-th output is a linear function of *n*; in both cases, faster and bigger designs exist for solving the problem within a *logarithmic* electrical delay (see [GV82] and [V83]).

Example 6.1 (Parallel Multiplier)

The circuit to the right, is a fragment of the (infinite) *high-school* parallel multiplier obtained by unfolding the serial-parallel multiplier in Section 5.3 through Algorithm 3:



6.3 Reversible Synchronous Circuits

Consider one last characterization of synchronous (*a.k.a.* on-line) functions as *norm contractions*; we state the definition in the unary (one input/output $f \in {}_2\mathbf{Z} \to {}_2\mathbf{Z}$) case only:

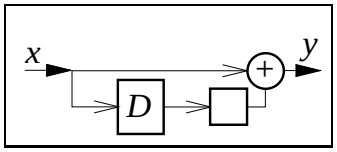
$$\forall x, y \in {}_2\mathbf{Z} : \qquad {}_2|f\big(x\big) - f\big(y\big)| \leq {}_2|x - y|. \tag{5}$$

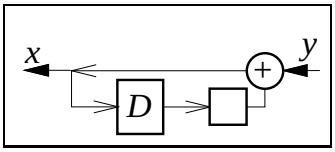
Relation (5) is easily seen to be equivalent to each of the previously stated characterizations of synchronous functions. With this in mind, we make a short incursion into *coding theory*:

Proposition 7 *A synchronous circuit $C \in \mathcal{C}(?, 2\times)$ computing $f \in {}_2\mathbf{Z} \to {}_2\mathbf{Z}$ is* reversible *if and only if each of the following equivalent properties hold:*

One to One	$\exists C^- \in \mathcal{C}(?, 2\times), \forall x \in {}_2\mathbf{Z}:$	$C^-\big(C(x)\big) = x;$
Norm Preserving	$\forall x, y \in {}_2\mathbf{Z}:$	${}_2\lvert f(x) - f(y)\rvert = {}_2\lvert x - y\rvert;$
Galois Sum	$\exists d \in \mathbf{B}, D \in \mathcal{C}(?, 2\times):$	$C(x) = x \oplus d \oplus 2D(x).$

Since $y = C(x)$ has an inverse C^- if and only if $\neg y = \overline{C}(x)$ has an inverse, we may choose $d = 0$ in the expression $y = x \oplus 2G(x)$. The inverse $x = C^-(y)$ is given by $x = y \oplus 2D(x)$, as illustrated by Examples 4.5 and 4.6, and the schemas below.





6.4 Synchronous circuits with output enable

By Theorem 2, we know that there exists no synchronous circuit which computes the output sequence $x \div 2 = {}_2x_1x_2\cdots$ in response to the input $x = {}_2x_0x_1x_2\cdots$. In general, no synchronous circuit is capable of producing strictly less output bits than it consumes inputs. To get around this problem, experienced designers add a signal $en \in \mathcal{V}(C)$ which is used to *enable* the outputs from the synchronous circuit C: it is set to $en_t = 1$ on cycles $t \in \mathbf{N}$ when the outputs of C are significant; it is set to $en_t = 0$ on cycles when the outputs of C are irrelevant.

By this convention, we may compute $x \div 2$ through the identity circuit with output enable $en = {}_20(1) = -2$. The same identity circuit with output enable $en = {}_2(10) = -1/3$ computes Peano's first projection $\pi_0(x_0x_1x_2\cdots x_t\cdots) = x_0x_2\cdots x_{2t}\cdots$. Indeed, these are special cases of the following general result:

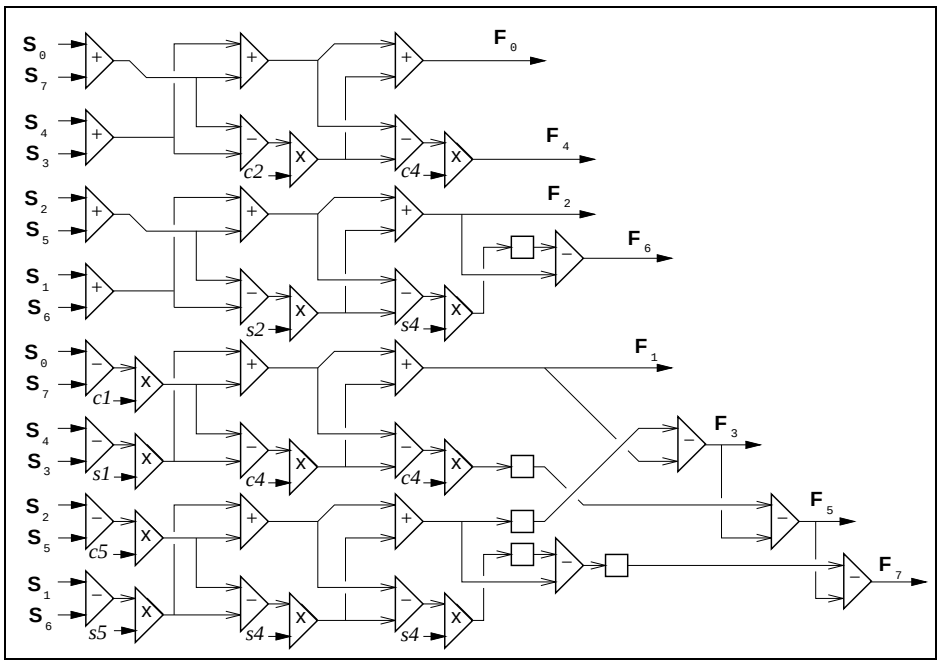
Theorem 3 *Every continuous function $f \in {}_2\mathbf{Z} \to {}_2\mathbf{Z}$ is computable by some synchronous circuit with output enable.*

Let us describe how to compose circuits with output enables. Suppose that some synchronous circuit $A \in \mathcal{C}(?, 2\times)$ has output enable en$\mathcal{A}$. We want to connect the inputs of A to the outputs $\mathcal{I}$ of some other synchronous circuit with output enable en$\mathcal{I}$. The rules are:

1. Replace every register $a = 2 \times b$ in A by the enabled register $a = 2\times?(en\mathcal{I}, b, a)$.
2. Set the output enable of the results of A to $en = en\mathcal{A} \wedge en\mathcal{I}$.

6.5 Synchronous circuits with reset

The arithmetic circuits in Section 5 all have a finite implementation when we interpret each operation modulo an odd number $1 + 2p$. In particular, we show in [SV93] how to derive from equation (4) a *finite* multiplier modulo $1 + 2p$, which operates from low order bits to high order bits. Combined with other techniques presented in [SV93], this modular multiplier is the key to the record breaking performances of this design.



Consider an arbitrary synchronous network, such as the one drawn above which computes the fast discrete cosine transform for appropriate coefficients $s1, \cdots, s7, c1, \cdots, c7 \in \mathbf{N}$. How can we pipe-line a sequence of finite precision integer computations through such a network? By simply adjoining a (synchronous) *reset* input, and replacing each register equation $r = 2 \times i$ in the network by the register with reset:

$$r = 2 \times ?\big(reset, 0, i\big).$$

Theorem 4 *In order to pipe-line a network $f \in {}_2\mathbf{Z} \to {}_2\mathbf{Z}$ over integer input sequences $i_0\, i_1 \cdots i_t \cdots \in \mathbf{Z}$ whose binary lengths vary in time $\forall t \in \mathbf{N}, \exists m\big(n\big) \in \mathbf{N} : i_t < 2^{m(n)}$, we adjoin the following reset signal to all registers:*

$$rst = 1 + \sum_{t \in \mathbf{N}} 2^{m(t)}.$$

With this reset, network f now computes:

$$F\big(\sum_{t \in \mathbf{N}} i_t z^t\big) = \sum_{t \in \mathbf{N}} \big(f\big(i_t\big) \cdot | \cdot 2^{m(t)}\big) z^t.$$

Note that all the operators which have so far received an infinite definition, such as multiplication, become *finite* as soon as the number $m = max_{n \in \mathbf{N}} m(n)$ is itself finite; indeed, we can truncate the whole network at m bits since all final results are correct modulo 2^{m+1}. The reset signals of two circuits get or-ed together, during composition.

7 Conclusions

We expect that the 2adic semantics introduced here for synchronous circuits will have an impact on current CAD systems, for the following reasons.

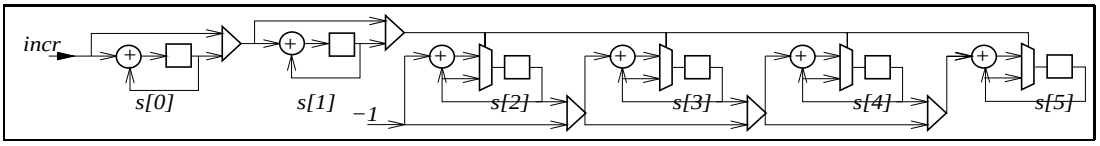
7.1 Synchronous Circuits Description

With F. Bourdoncle and G. Berry, we are attempting to map this 2adic theory into a *language* called 2*Z* for describing synchronous circuits, as they naturally occur in practical PAM designs (see [BRV89]), such as the ones reported in [BRV93]. For example, the following 2*Z* source code generates the counters from [V91] whose operating speed is, for all practical purposes, *independent* of the counter's length.

```
SlowCounter[n](incr) = (s[n],ovfl)
where
  c[0]=incr;
  for k<n do
    c[k+1] = c[k] and s[k];
    s[k]   = reg(c[k] xor s[k])
  end for;
  ovfl = c[n]
end where;

FastCounter[n](incr) = s[n]
where
  k=2;    // this parameter is technology-dependent //
  if n<k
    then (s,ovfl) = SlowCounter[n](incr);
    else (s[0..k-1],en) = SlowCounter[k](incr);
         enable en in
           (s[k..n-1],cn) = SlowCounter[n-k](-1);
         end enable;
         ovfl = cn and en;
  end if
end where
```

The schemas resulting from to the execution of the 2*Z* expression *FastCounter*[6] are:



7.2 Synchronous Circuits Synthesis

As pointed out earlier, the SDD procedure is an interesting candidate for compiling from finite state machine descriptions (hopefully produced by higher level systems) into real hardware (silicon or FPGA). It should be instructive in this respect to compare the resulting SDD implementation with the direct technique reported by [B92], and others.

7.3 Synchronous Circuits Verification

Finally, we expect some CAD systems to incorporate rules for circuit verification having to do with the *ring* properties of 2adic algebra, not just the boolean part. The need for such tools is clear when one consider the problem of proving functionally equivalent, such structurally different multipliers as the ones from Example 6.1 and Reference [V83]. To prove further that such parallel multipliers compute the same function as their serial counterparts (Sections 5.3 and 5.4), we seem to need the full 2adic apparatus introduced here; any proof attempt through independent means has to discover, prove and use the ring laws somewhere along the line.

8 Proofs

This section regroups the mathematical definitions, lemmas and constructions required to demonstrate the various claims made in this paper.

8.1 2adic integers

Proof of Proposition 2

(ii) Let $l \in \mathbf{N}$ be the least integer such that $|z| < 2^l$. After computing l bits of z by Rule 1, we reach the state: $\mathcal{B}_2(z) \stackrel{l}{\Rightarrow} z_0 \cdots z_{l-1}\mathcal{B}_2(z \div 2^l)$. When $z \geq 0$, we have $z \div 2^l = 0$, so $\mathcal{B}_2(z \div 2^l) = {}_2(0)$; when $z < 0$, we have $z \div 2^l = -1$, so $\mathcal{B}_2(z \div 2^l) = {}_2(1)$.

(iii) Conversely ${}_2z_0 \cdots z_{l-1}(z_l) = \sum_{k<l} z_k 2^k - 2^l z_l$ is an integer $z \in \mathbf{Z}$.

(iv) Integer $z = {}_2z_0 \cdots z_{l-1}(z_l)$ is computed by the following acyclic synchronous circuit, with l registers: $r[l-1] = z_{l-1}+2\times z_l$, $r[l-2] = z_{l-1}+2\times r[l-1], \cdots, r[0] = z_0+2\times r[1]$. The inputs $-z_l = {}_2(z_l), 0, -1$ are constant, and the output is $r[0]$.

Conversely, it follows from Definitions 5,7 of mux and reg that an acyclic synchronous circuit with $l \in \mathbf{N}$ registers produces a constant output after at most l cycles, upon constant input.

(vi) Let $B = \frac{z}{1+2n}$ be an odd rational, with $z \in \mathbf{Z}$ and $n \in \mathbf{N}+1$. Define the *period* $p = p_2(1+2n)$ of the denominator $1+2n$ to be the order of 2 in the multiplicative group $\mathbf{Z} \cdot|\cdot (1+2n)$ of the integers modulo $1+2n$, namely the smallest natural number such that:

$$2^p = 1 \pmod{1+2n}.$$

Let the quotient be $q = (2^p - 1) \div (1+2n)$, the corresponding remainder being 0.

Compute $I = zq \div (2^p - 1)$ the quotient, and $P = zq \cdot|\cdot (2^p - 1) = {}_2p_0 \cdots p_{p-1}$ the remainder in the integer division of $z \times q$ by $2^p - 1$, so as to write:

$$B = \frac{z}{1+2n} = I - \frac{P}{2^p - 1}.$$

The binary representation of $\frac{-P}{2^p-1} = {}_2(p_0 \cdots p_{p-1})$ is purely periodic. It follows that the binary representation of B is ultimately periodic: $b_k = b_{k+p}$ for $k \geq i+p$.

(vii) Let number $B = {}_2b_0 \cdots b_{i-1}(b_i \cdots b_{i+p-1})$ be *ultimately periodic*, and consider the integers $I = {}_2b_0 \cdots b_{i-1}$ and $P = {}_2b_i \cdots b_{i+p-1}$. Number $B \in \mathbf{Z}/1+2\mathbf{N}$ is the odd rational:

$$B = I - \frac{2^i P}{2^p - 1}.$$

(viii) The periodic rational $B = {}_2b_0 \cdots b_{i-1}(b_i \cdots b_{i+p-1})$ is computed by the following synchronous circuit, with $i+p$ registers:

$$r[i+p-1] = b_{i+p-1} + 2 \times r[i],\ r[i+p-2] = b_{i+p-2} + 2 \times r[i+p-3],\ \cdots,\ r[0] = b_0 + 2 \times r[1].$$

The inputs 0 and -1 are constants, and the output is $r[0]$.

Conversely, let $r[0], \cdots, r[n-1]$ be the n registers of a finite synchronous circuit $C \in \mathcal{C}(?, 2\times)$. Define the *state* of C at time t by the integer $S_t = \sum_{0 \leq k < n} r_t[k] 2^k$. This number is obviously bounded by $S_t < 2^n$; so, there must exist two instants $0 \leq t_0 < t_1 < 2^N$ where we find the circuit in the *same* state: $S_{t_0} = S_{t_1}$. With a constant input, the output must therefore be periodic $B = {}_2b_0 \cdots b_{i-1}(b_i \cdots b_{i+p-1})$ with $i = t_0$ and $p = t_1 - t_0$. ∎

8.2 Combinational circuits

Definition 9 *The* truth table *of a finite boolean function $f \in \mathbf{B}^n \to \mathbf{B}$ is the natural number $\mathbf{f} = {}_2\mathbf{f}_0 \cdots \mathbf{f}_{2^n-1} \in \mathbf{N}$ defined by:*

$$\mathbf{f} = \sum_{b_0, \cdots, b_{n-1} \in \mathbf{B}} f(b_0, \cdots, b_{n-1}) 2^{2^{b_0, \cdots, b_{n-1}}}.$$

The BDD [B86] and TDG [B87] algorithms produce a combinational circuit $f \in \mathcal{C}(?)$ which computes a finite boolean function presented by its truth table as follows:

1. Recursively decompose f into a tree of muxes by Shannon's formula:

$$f\big(x_0, \cdots, x_{n-2}, x_{n-1}\big) = ?\big(x_{n-1}, f\big(x_0, \cdots, x_{n-2}, 1\big), f\big(x_0, \cdots, x_{n-2}, 0\big)\big).$$

2. Share all equal sub-expressions generated during this decomposition.

Within our framework, this procedure may be expressed as follows.

Algorithm 1 (BDD) *Let $f \in \mathbf{B}^n \to \mathbf{B}$ be any finite boolean function presented by its truth table $\mathbf{f} \in \mathbf{N}$. The finite combinational circuit $BDD\big(f\big) \in \mathcal{C}\big(?\big)$ which computes f has:*

1. *Inputs $\{i[0], \cdots, i[n-1]\}$.*
2. *Output $o = m[\mathbf{f}]$.*
3. *A set $\mathcal{M}\big(n, \mathbf{f}\big)$ of muxes where each mux $m[k] \in \mathcal{M}\big(p, k\big)$ implements the truth table $k \in \mathbf{N}$ over $p > 0$ inputs by:*

$$\begin{array}{lcll} m[k] & = & ?\big(i[p], m[k \div 2^p], m[k \cdot|\cdot 2^p]\big) & \text{if } k \text{ is even;} \\ m[k] & = & \neg m[\neg k] & \text{if } k \text{ is odd.} \end{array}$$

These equations are simplified when either of the following applies:

$$\begin{array}{lcl} ?\big(c, b, b\big) & \Rightarrow & b, \\ ?\big(c, 1, 0\big) & \Rightarrow & c. \end{array}$$

Proof of Theorem 1

The proof is decomposed in three lemmas:

1. function f is continuous if and only if it is uniformly continuous (Lemma 2);
2. function f is uniformly continuous if and only if each output depends upon finitely many inputs (Lemma 3);
3. each output of function f depends upon finitely many inputs if and only if f is computable by some combinational circuit (Lemma 1). ■

The well-founded ordering of the variables in a combinational circuit (Definition 6) implies that each output only depends upon a *finite* subset of the inputs.

Lemma 1 *A function $f \in {}_2\mathbf{Z} \to {}_2\mathbf{Z}$ over the 2adic integers is computed by a combinational circuit $f \in \mathcal{C}\big(?\big)$ if and only if it can be expressed as a sum*

$$f\big(\sum_{n \in \mathbf{N}} i[n] 2^n\big) \quad = \quad \sum_{n \in \mathbf{N}} f_n\big(i[0] \cdots i[m\big(n\big)]\big) 2^n, \tag{6}$$

where for all $n \in \mathbf{N}$, number $m\big(n\big) \in \mathbf{N}$ is an integer and $f_n \in \mathbf{B}^{m(n)+1} \to \mathbf{B}$ is a boolean function with $m\big(n\big) + 1$ inputs.

Proof: Through topological sort, we may assume *w.l.o.g.* that the mux numbering is compatible with the mux ordering in Definition 6: $m[i] \prec m[j]$ implies $i < j$. The first mux output $m[0]$ is determined by $m[0] = ?\big(c_0, b_0, a_0\big)$ which depends upon (at most) 3 different inputs $c_0, b_0, a_0 \in \mathcal{I}$; similarly, $m[1]$ is computed through (at most) two multiplexers involving (at most) 5 different inputs in $\mathcal{I}$. In general, $m[n-1]$ may be drawn as a *tree* of multiplexers having depth less than n, and at most $3n$ different leaves in $\mathcal{I}$. It follows that output $o[n]$ may be expressed, for each $n \in \mathbf{N}$, as a boolean function $o[n] = f_n\big(i[0] \cdots i[m(n)]\big)$ where $m\big(n\big) \in \mathbf{N}$ is the highest input index which appears in the leaves of the mux tree defining $o[n]$. Conversely, we know from the BDD (Algorithm 1) how to realize boolean function f_n from Expression (6) by a finite combinational circuit $\mathcal{C}_n$, for each $n \in \mathbf{N}$; the union $f = \cup_{n\geq 0}\mathcal{C}_n \in \mathcal{C}\big(?\big)$ of all such circuits realizes any function f given by an expression of the form (6). ■

Our next result is a special case of Heine's theorem, which says that a function is continuous over a topologically compact set (namely the whole of ${}_2\mathbf{Z}$) if and only if it is uniformly continuous (see *e.g.* [A75]).

Lemma 2 *A function* $f \in {}_2\mathbf{Z} \to {}_2\mathbf{Z}$ *is* continuous *if and only if it is* uniformly continuous*:*

$$\forall n \in \mathbf{N}, \exists m \in \mathbf{N}, \forall x, y \in {}_2\mathbf{Z} : \ {}_2|x - y| < 2^{-n} \textit{ implies } {}_2|f\big(x\big) - f\big(y\big)| < 2^{-m}. \qquad (7)$$

Lemma 3 *A function* $f \in {}_2\mathbf{Z} \to {}_2\mathbf{Z}$ *is* uniformly continuous *over the 2adic integers if and only if it is computed by some combinational circuit* $C_f \in \mathcal{C}\big(?\big)$.

Proof: Let us first express uniform continuity in the equivalent form:

$$\forall n \in \mathbf{N}, \exists m\big(n\big) \in \mathbf{N}, \forall x \in {}_2\mathbf{Z} : \ {}_2|f\big(x\big) - f\big(x \cdot|\cdot 2^m\big)| < 2^{-n}. \qquad (8)$$

Clearly (7) implies (8); conversely, ${}_2|f\big(x+2^m y\big) - f\big(x\big)| = {}_2|f\big(x+2^m y\big) - f\big(x \cdot|\cdot 2^m\big) + f\big(x \cdot|\cdot 2^m\big) - f\big(x\big)| \leq max\{\, {}_2|f\big(x + 2^m y\big) - f\big(\big(x + 2^m y\big) \cdot|\cdot 2^m\big)|, \text{ so we have } {}_2|f\big(x\big) - f\big(x \cdot|\cdot 2^m\big)|\} < 2^{-n}$ for any $y \in {}_2\mathbf{Z}$ by (7) and the ultra-metric property (i). Expression (8) says that we can determine the first n bits of $f\big(x\big) \cdot|\cdot 2^n = {}_2f_0 \cdots f_{n-1} = f\big(x \cdot|\cdot 2^m\big) \cdot|\cdot 2^n$ from the first $m = m\big(n\big)$ bits of the argument $x \in {}_2\mathbf{Z}$. In particular, the first output bit $f_0 = f\big(x\big) \cdot|\cdot 2$ is given by $f_0 = f_0\big(x_0, \cdots, x_{m(0)-1}\big)$, for some boolean function $f_0 \in \mathbf{B}^{m(0)} \to \mathbf{B}$ defined by $f_0\big(x_0, \cdots, x_{m(0)-1}\big) = f\big(\sum_{0\leq k<m(0)} x_k 2^k\big) \cdot|\cdot 2$. In general, function $f \in {}_2\mathbf{Z} \to {}_2\mathbf{Z}$ is uniformly continuous if and only if it can be expressed as a sum of the form (6), in which boolean function $f_n \in \mathbf{B}^{m(n)} \to \mathbf{B}$ is defined for each integer $n \in \mathbf{N}$ by $f_n\big(x_0, \cdots, x_{m(n)-1}\big) = \big(f\big(\sum_{0\leq k<m(n)} x_k 2^k\big) \div 2^{n-1}\big) \cdot|\cdot 2$. We conclude this proof by invoquing Lemma 1. ■

8.3 Synchronous circuits

To simplify notations, the SDD procedure is presented in the generic case of a one input ($k = 1$), one output on-line function. The generalization to an arbitrary number of inputs $k \geq 0$ is direct; Example 2.1 shows applications of the procedure with zero input ($k = 0$); Examples 4.1,4.3,4.5 have one input ($k = 1$) and Example 4.4 has two inputs ($k = 2$).

The basic step in the SDD construction is to express each on-line function f in the form

$$f(x) = ?(x, b_1 + 2f^{(1)}(x),\ b_0 + 2f^{(0)}(x))$$

for some booleans $b_0, b_1 \in \mathbf{B}$ and on-line functions $f^{(0)}, f^{(1)} \in {}_2\mathbf{Z} \to {}_2\mathbf{Z}$. From the expression

$$f(x) = \sum_{t\in\mathbf{N}} f_t(x_0, \cdots, x_t)2^t,$$

we see that $b_0 = f_0(0)$, $b_1 = f_0(1)$ and, for $b \in \mathbf{B}$:

$$f^{(b)}(x) = \sum_{t\in\mathbf{N}} f_t(x_0, \cdots, x_t, b)2^t.$$

Algorithm 2 (SDD) *Let $f \in {}_2\mathbf{Z} \to {}_2\mathbf{Z}$ be any on-line function defined by*

$$f(x) = \sum_{t\in\mathbf{N}} f_t(x_0, \cdots, x_t)2^t,$$

with $f_t \in \mathbf{B}^{t+1} \to \mathbf{B}$ for $t \in \mathbf{N}$. The synchronous circuit SDD$(f) \in \mathcal{C}(?, 2\times)$ which computes f is constructed as follows.:

1. *It has input $x = {}_2x_0 \cdots x_t \cdots \in {}_2\mathbf{Z}$.*
2. *Its output is defined by $o = ?(x, v[1], v[0])$.*
3. *It has an infinite set $\mathcal{V}(f) = \{v[b_0 \cdots b_{p-1}] :\ p \in \mathbf{N}, b_0, \cdots, b_{p-1} \in \mathbf{B}\}$ of variables, indexed by all possible finite binary strings. Each variable computes the function*

$$v[b_0 \cdots b_{p-1}] \quad = \quad \sum_{t\in\mathbf{N}} f_{t+p}(x_0, \cdots, x_t, b_0, \cdots, b_{p-1})2^t$$

   which is defined (in the absence of simplification) by the equation:

$$v[b_0 \cdots b_{p-1}] \quad = \quad f_{p-1}(b_0, \cdots, b_{p-1}) + 2\times?(x, v[b_0 \cdots b_{p-1}1], v[b_0 \cdots b_{p-1}0]).$$

4. *For each equality $v = v'$ or $v = \neg v'$ between two different variables $v \ll v'$, replace the equation defining v' by:*

$$v' = v \quad or \quad v' = \neg v.$$

   The replaced variable v' is chosen to be the largest of v and v' in the lexicographic ordering: $v[b_0 \cdots b_{p-1}] \ll v[b'_0 \cdots b'_{p'-1}]$ if $p < p'$ or ${}_2b_0 \cdots b_{p-1} < {}_2b'_0 \cdots b'_{p-1}$.
5. *Simplify each mux sub-expression according to the four rules:*

$$\begin{aligned} ?(c, b, b) &\Rightarrow b, \\ ?(c, -1, 0) &\Rightarrow c, \\ ?(c, \neg b, \neg a) &\Rightarrow \neg?(c, b, a), \\ ?(c, 2\times b, 2\times a) &\Rightarrow 2\times?(c, b, a). \end{aligned}$$

Let us apply the SDD construction to the simple synchronous circuit defined by the equations:

$$r = 2 \times \mathcal{S}(x, r),\ f(x) = \mathcal{O}(x, r).$$

Register r contains the current state, exclusively determined from the input x by the combinational state function $\mathcal{S}$. The output $f(x)$ is given by the combinational function $\mathcal{O}$. One step of the SDD construction introduces the functions $f^{(0)}$ and $f^{(1)}$ related to f by:

$$f(x) = ?(x, f_0(1) + 2f^{(1)}(x),\ f_0(0) + 2f^{(0)}(x)).$$

Through elementary 2adic transformations, circuit $f^{(b)}$ for $b \in \mathbf{B}$ has the expression:

$$r = 2 \times \mathcal{S}(x, r),\ f^{(b)}(x) = \mathcal{O}^{(b)}(x, r).$$

The circuit defining $f^{(b)}$ has the *same* state function $\mathcal{S}$ as f, and output function:

$$\mathcal{O}^{(b)}(x, r) = \mathcal{O}(-b, \mathcal{S}(x, r)).$$

Each variable $v = f^{(b_0 \cdots b_{p-1})}$ in the SDD construction is defined by a circuit of the form

$$r = 2 \times \mathcal{S}(x, r),\ f^{(b_0 \cdots b_{p-1})}(x) = \mathcal{O}^{(b_0 \cdots b_{p-1})}(x, r),$$

for some output function $\mathcal{O}^{(b_0 \cdots b_{p-1})}$. Termination of the SDD procedure follows in this case, as there are finitely many such combinational functions.

We now state the argument in a formal way.

Definition 10 *Let $f \in \mathcal{C}(?, 2\times)$ be a synchronous circuit, with registers*

$$\mathcal{R}(f) = \{r[0], \cdots, r[n], \cdots\}.$$

1. *The* state $\mathcal{S}(f, x, k) \in {}_2\mathbf{Z}$ *of f at time $k \in \mathbf{N}$ on input $x \in {}_2\mathbf{Z}$ is the 2adic integer:*

$$\mathcal{S}(f, x, k) = \sum_{r[n] \in \mathcal{R}(f)} r_k[n] 2^n,$$

   where $r_k[n] \in \mathbf{B}$ is the value of register $r[n] \in \mathcal{R}(f)$ at cycle $k \in \mathbf{N}$, in the computation of $f(x)$, starting from the initial zero state $\mathcal{S}(f, x, 0) = 0$.

2. *To each state $s_t = \mathcal{S}(f, x, k) \in {}_2\mathbf{Z}$, we associate the* state function $f[s_t] \in {}_2\mathbf{Z} \to {}_2\mathbf{Z}$ *defined by*

$$\forall x, z \in {}_2\mathbf{Z}, k \in \mathbf{N} :\ f[\mathcal{S}(f, x, k)](z) = f(x + z2^k) \div 2^k.$$

3. *An on-line function $f \in {}_2\mathbf{Z} \to {}_2\mathbf{Z}$ is* finite *when the set*

$$\mathcal{S}(f) = \bigcup_{x \in {}_2\mathbf{Z}, k \in \mathbf{N}} \mathcal{S}(f, x, k)$$

   of states reachable *from the initial zero state is finite: $|\mathcal{S}(f)| = s \in \mathbf{N}$.*

Note that the state function is independent of the specific input (x, k) leading to that state, since $\mathcal{S}(f, x, k) = \mathcal{S}(f, x', k')$ implies $f[\mathcal{S}(f, x, k)] = f[\mathcal{S}(f, x', k')]$.

Proof of Proposition 3 If $SDD(f) \in \mathcal{C}(?, 2\times)$ is a finite synchronous circuit, then $f \in {}_2\mathbf{Z} \to {}_2\mathbf{Z}$ is clearly a finite on-line function. To prove the converse, we must show that, for $p \in \mathbf{N}$ large enough, each variable

$$v[b_0 \cdots b_{p-1}] \quad = \quad \sum_{t \in \mathbf{N}} f_{t+p}(x_0, \cdots, x_t, b_0, \cdots, b_{p-1}) 2^t \tag{9}$$

in the SDD algorithm is equal to some $v[b'_0 \cdots b'_{p'-1}]$, with $b'_0 \cdots b'_{p'-1} \ll b_0 \cdots b_{p-1}$. We may ignore for this proof the use of negation in the SDD construction, since it may only reduce the size of the generated logic. Through state functions, we rewrite (9) as:

$$v[b_0 \cdots b_{p-1}] \quad = \quad \sum_{t \in \mathbf{N}} f_p[s_t](b_0, \cdots, b_{p-1}) 2^t, \tag{10}$$

where $s_t = \mathcal{S}(f, x, t) \in \{0, \cdots, s-1\}$, $s = |\mathcal{S}(f)|$ and $f_p[s_t](b) = \big(f[s_t](b) \div 2^p\big) \cdot | \cdot 2$. In order to prove $v[b'_0 \cdots b'_{p'-1}] = v[b_0 \cdots b_{p-1}]$, we see from (10) that it is sufficient to establish:

$$\forall s_t \in \mathcal{S}(f) : \; f[s_t](b_0, \cdots, b_{p-1}) \quad = \quad f[s_t](b'_0, \cdots, b'_{p'-1}). \tag{11}$$

Equation (11) must certainly be satisfied in a non-trivial manner as soon as $p > s^2$, since there are at most s^2 pairs of state functions. ∎

8.4 Arithmetic circuits

Proof of Proposition 4: Any circuit for squaring arbitrary 2adic integers must contain infinitely many registers. Suppose the contrary, namely some circuit $C \in \mathcal{C}(?, 2\times)$ with n registers produces output x^2 for each input $x \in {}_2\mathbf{Z}$. Circuit C may reach at most 2^n different states S. There are 2^{n+1} integers in the set $I = \{y \in \mathbf{N} : \; y = 2^{n+1}x, x \leq 2^{n+1}\}$. Take inputs to circuit C from set I, and consider what happens at time $t = 2n + 2$:

1. all the outputs produced up to this point are zero, since C is squaring;
2. there are more elements in I than possible states, hence there must exist two *different* $y \neq y'$ numbers in I which bring C into the *same* state $S_{2n+2}(y) = S_{2n+2}(y')$, on inputs y and y'. Since all subsequent inputs are zero, all subsequent outputs must necessarily be equal; so $C(y) = C(y')$ yet $y^2 \neq y'^2$, a contradiction. ∎

Proposition 8 *A non zero 2adic integer $b' \in {}_2\mathbf{Z}$ is a square if and only if it is of the form $b' = 2^{2v}(1 + 8b)$ with even valuation $v_2(b') = 2v$ and odd part $1 + 8b$ congruent to 1 modulo 8.*

Proof: Any non zero 2adic integer $b \in {}_2\mathbf{Z}$ can be written as $b = 2^v(1 + 2b_1)$ with valuation $v = v_2(b) \in \mathbf{N}$ and odd part $1 + 2b_1 + 4B_2$, where $b_1 \in \mathbf{B}$ and $B_2 \in {}_2\mathbf{Z}$. The square $S = b^2$

of b has the form: $S = 2^{2v}\big(1 + 8\big(b_1 + b_2\big)\big(1 + 2b_2\big)\big)$. Conversely, a number of the form $S = 2^{2v}\big(1 + 8b\big)$ has the two square roots $\pm 2^v\sqrt{1 + 8b}$, one of which being constructed by the circuit in Section 5.6. ∎

8.5 Applications

Algorithm 3 (Parallelization of a Synchronous circuit)

- *The input to this algorithm is a synchronous circuit* $C \in \mathcal{C}\big(?, 2\times\big)$, *whose set of variables* $\mathcal{V}\big(C\big) = \mathcal{I} \cup \mathcal{R} \cup \mathcal{M}$ *consists of:*

  1. *Inputs* $\mathcal{I} = \{i[0], \cdots, i[i-1]\}$ *for* $i = |\mathcal{I}|$.
  2. *Muxes* $\mathcal{M} = \{m[0], \cdots, m[n], \cdots\}$, *each defined by a mux equation* $m[n] = ?\big(c_n, b_n, a_n\big)$ *for some* $c_n, b_n, a_n \in \mathcal{V}\big(C\big)$.
  3. *Registers* $\mathcal{R} = \{r[0], \cdots, r[n], \cdots\}$, *each defined by a reg equation* $r[n] = 2 \times i_n$ *for some* $i_n \in \mathcal{V}\big(C\big)$.

  The outputs of C form a finite subset of the variables $\mathcal{O} = \{o[0], \cdots o[o-1]\} \subseteq \mathcal{V}\big(C\big)$ *and the mux ordering* $\prec$ *is well-founded.*

- *The output from this algorithm is an infinite combinational circuit* $C' \in \mathcal{C}\big(?\big)$. *To each variable* $v \in \mathcal{V}\big(C\big)$, *we associate an infinite set of variables in* C'*:*
  $\mathcal{V}\big(C'\big) = \{v[t] : \ v \in \mathcal{V}\big(C\big), t \in \mathbf{N}\}$.

  1. *The inputs of* C' *are* $\mathcal{I}' = \{i[j][t] : i[j] \in \mathcal{I}, t \in \mathbf{N}.\}$
  2. *The muxes of* C' *are* $\mathcal{M}' = \{m[j][t] : m[j] \in \mathcal{M}, t \in \mathbf{N}.\}$, *each defined by the mux equation* $m[n][t] = ?\big(c_n[t], b_n[t], a_n[t]\big)$.
  3. *To each* $r[n] \in \mathcal{R}$ *correspond the equalities:* $r[n][0] = 0$ *and* $r[n][t+1] = i_n[t]$ *for* $t \in \mathbf{N}$.

  The outputs of C' *are* $\mathcal{O}' = \{o[j][t] : o[j] \in \mathcal{O}, t \in \mathbf{N}.\}$ *The mux ordering* $\prec'$ *is (lexicographically) well-founded since* $v_1[t_1] \prec' v_2[t_2]$ *implies* $t_1 < t_2$ *or* $t_1 = t_2$ *and* $v_1 \prec v_2$.

Proof of Theorem 3 By Proposition 5, we may express f as

$$f\big(x\big) = \sum_{k \in \mathbf{N}} 2^k f_k\big(x_0 \cdots x_{m(k)}\big).$$

Since a boolean function with k inputs may be considered as a function with $k+1$ inputs which ignores the last one, we may assume that $m\big(k\big) < m\big(k+1\big)$ for all $k \in \mathbf{N}$.

For $j \in \mathbf{N}$, define $g_j\big(x_0 \cdots x_j\big) = f_k\big(x_0 \cdots x_{m(k)}\big)$ where index $k = k\big(j\big)$ is determined by the relation $m\big(k\big) \le j < m\big(k+1\big)$.

Consider the on-line function g defined by:

$$g(x) = \sum_{j \in \mathbf{N}} 2^j g_j(x_0 \cdots x_j)$$

We know from Theorem 2 that g may be computed by some synchronous circuit C_g. By construction, function f is thus computed by circuit C_g with output enable:

$$en = \sum_{k \in \mathbf{N}} 2^{m(k)}.$$

■

References

[A75] Y. Amice. Les nombres p-adiques. *Presses Universitaires de France*, 1975.

[A78] S. B. Akers. Binary decision diagrams. *IEEE Trans. Computers*, 27:509–516, 1978.

[BRV89] P. Bertin, D.Roncin and J. Vuillemin. Introduction to Programmable Active Memories. *Systolic Array Processors*, Prentice-Hall, 300–309, 1989. Also available as *PRL Research Report 3*, Digital Equipment Corp., Paris Research Laboratory, 85, Av. Victor Hugo, 92563 Rueil-Malmaison Cedex, France, 1989.

[BRV93] P. Bertin, D.Roncin and J. Vuillemin. Programmable Active Memories: a Performance Assessment. *To appear in Symposium on Integrated Systems*, U. of Washington, 1993. Also available as *PRL Research Report 24*, Digital Equipment Corp., Paris Research Laboratory, 85, Av. Victor Hugo. 92563 Rueil-Malmaison Cedex, France, 1993.

[B92] G. Berry. A Hardware Implementation of Pure Esterel. *Academy Proceedings in Engineering Sciences, Indian Academy of Sciences*, Sadhana, 17:1:95–130, 1992. Also available as *PRL Research Report 15*, Digital Equipment Corp., Paris Research Laboratory, 85, Av. Victor Hugo. 92563 Rueil-Malmaison Cedex, France, (1989).

[B86] R. E. Bryant. Graph-based Algorithms for Boolean Function Manipulation. *IEEE Trans. Computers*, 35:8:677–691, 1986.

[B87] J. P. Billon. Perfect Normal Forms for Discrete Function. *BULL Research Report*, 87019, 1987.

[CW79] I.N. Chen, R. Willoner. An O(n) parallel multiplier with bit-sequential input and output. *IEEE Trans. Computers*, 28:10, Oct. 1979.

[GV82] L. Guibas, J. Vuillemin. On fast binary addition in n-MOS technologies. *Proceedings of IEEE Conference*, New York ICCC 82, 147–151.

[H13] K. Hensel. Zahlentheorie. *Gőshen, Berlin-Leiptzig*, 1913.

[K77] N. Koblitz. p-adic Numbers, p-adic Analysis and Zeta Functions. *Springer-Verlag*, 1977.

[K81] D.E. Knuth. Seminumerical Algorithms. *The Art of Computer Programming*, Addison Wesley, 2, 1981.

[LS91] C. Leiserson, J. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6:1:5–35, 1991.

[L76] R. F. Lyon. Two's complement pipeline multipliers. *IEEE Trans. Comm.*,24:418–425, 1976.

[M89] C. Mead. Analog VLSI and Neural Systems. *Addison-Wesley*, 1989.

[SV93] M. Shand, J. Vuillemin. Fast Implementations of RSA Cryptography. *11-th IEEE Symposium on Computer Arithmetic*, 1993.

[V83] J. Vuillemin. A very fast multiplication algorithm for VLSI implementation. *INTEGRATION, the VLSI Journal*, 1:1, Mars 1983.

[V91] J. Vuillemin. Constant Time Arbitrary Length Synchronous Binary Counters. *10-th IEEE Symposium on Computer Arithmetic*, 301–309, 1991.

PRL Research Reports

The following documents may be ordered by regular mail from:

Librarian – Research Reports
Digital Equipment Corporation
Paris Research Laboratory
85, avenue Victor Hugo
92563 Rueil-Malmaison Cedex
France.

It is also possible to obtain them by electronic mail. For more information, send a message whose subject line is `help` to `doc-server@prl.dec.com` or, from within Digital, to `decprl::doc-server`.

Research Report 1: *Incremental Computation of Planar Maps*. Michel Gangnet, Jean-Claude Hervé, Thierry Pudet, and Jean-Manuel Van Thong. May 1989.

Research Report 2: *BigNum: A Portable and Efficient Package for Arbitrary-Precision Arithmetic*. Bernard Serpette, Jean Vuillemin, and Jean-Claude Hervé. May 1989.

Research Report 3: *Introduction to Programmable Active Memories*. Patrice Bertin, Didier Roncin, and Jean Vuillemin. June 1989.

Research Report 4: *Compiling Pattern Matching by Term Decomposition*. Laurence Puel and Ascánder Suárez. January 1990.

Research Report 5: *The WAM: A (Real) Tutorial*. Hassan Aït-Kaci. January 1990.†

Research Report 6: *Binary Periodic Synchronizing Sequences*. Marcin Skubiszewski. May 1991.

Research Report 7: *The Siphon: Managing Distant Replicated Repositories*. Francis J. Prusker and Edward P. Wobber. May 1991.

Research Report 8: *Constructive Logics. Part I: A Tutorial on Proof Systems and Typed λ-Calculi*. Jean Gallier. May 1991.

Research Report 9: *Constructive Logics. Part II: Linear Logic and Proof Nets*. Jean Gallier. May 1991.

Research Report 10: *Pattern Matching in Order-Sorted Languages*. Delia Kesner. May 1991.

---

† This report is no longer available from PRL. A revised version has now appeared as a book: "Hassan Aït-Kaci, *Warren's Abstract Machine: A Tutorial Reconstruction. MIT Press, Cambridge, MA (1991).*"

Research Report 11: *Towards a Meaning of LIFE*. Hassan Aït-Kaci and Andreas Podelski. June 1991 (Revised, October 1992).

Research Report 12: *Residuation and Guarded Rules for Constraint Logic Programming*. Gert Smolka. June 1991.

Research Report 13: *Functions as Passive Constraints in LIFE*. Hassan Aït-Kaci and Andreas Podelski. June 1991 (Revised, November 1992).

Research Report 14: *Automatic Motion Planning for Complex Articulated Bodies*. Jérôme Barraquand. June 1991.

Research Report 15: *A Hardware Implementation of Pure Esterel*. Gérard Berry. July 1991.

Research Report 16: *Contribution à la Résolution Numérique des Équations de Laplace et de la Chaleur*. Jean Vuillemin. February 1992.

Research Report 17: *Inferring Graphical Constraints with Rockit*. Solange Karsenty, James A. Landay, and Chris Weikart. March 1992.

Research Report 18: *Abstract Interpretation by Dynamic Partitioning*. François Bourdoncle. March 1992.

Research Report 19: *Measuring System Performance with Reprogrammable Hardware*. Mark Shand. August 1992.

Research Report 20: *A Feature Constraint System for Logic Programming with Entailment*. Hassan Aït-Kaci, Andreas Podelski, and Gert Smolka. November 1992.

Research Report 21: *The Genericity Theorem and the Notion of Parametricity in the Polymorphic λ-calculus*. Giuseppe Longo, Kathleen Milsted, and Sergei Soloviev. December 1992.

Research Report 22: *Sémantiques des langages impératifs d'ordre supérieur et interprétation abstraite*. François Bourdoncle. January 1993.

Research Report 23: *Dessin à main levée et courbes de Bézier : comparaison des algorithmes de subdivision, modélisation des épaisseurs variables*. Thierry Pudet. January 1993.

Research Report 24: *Programmable Active Memories: a Performance Assessment*. Patrice Bertin, Didier Roncin, and Jean Vuillemin. March 1993.

Research Report 25: *On Circuits and Numbers*. Jean Vuillemin. November 1993.

Research Report 26: *Numerical Valuation of High Dimensional Multivariate European Securities*. Jérôme Barraquand. March 1993.

Research Report 27: *A Database Interface for Complex Objects*. Marcel Holsheimer, Rolf A. de By, and Hassan Aït-Kaci. March 1993.

Research Report 28: *Feature Automata and Sets of Feature Trees*. Joachim Niehren and Andreas Podelski. March 1993.

Research Report 29: *Real Time Fitting of Pressure Brushstrokes*. Thierry Pudet. March 1993.

Research Report 30: *Rollit: An Application Builder*. Solange Karsenty and Chris Weikart. April 1993.

Research Report 31: *Label-Selective λ-Calculus*. Hassan Aït-Kaci and Jacques Garrigue. May 1993.

Research Report 32: *Order-Sorted Feature Theory Unification*. Hassan Aït-Kaci, Andreas Podelski, and Seth Copen Goldstein. May 1993.

Research Report 33: *Path Planning through Variational Dynamic Programming*. Jérôme Barraquand and Pierre Ferbach. September 1993.

Research Report 34: *A penalty function method for constrained motion planning*. Pierre Ferbach and Jérôme Barraquand. September 1993.

Research Report 35: *The Typed Polymorphic Label-Selective λ-Calculus*. Jacques Garrigue and Hassan Aït-Kaci. October 1993.

25

On Circuits and Numbers

Jean Vuillemin

digital

PARIS RESEARCH LABORATORY

85, Avenue Victor Hugo
92563 RUEIL MALMAISON CEDEX
FRANCE