Efficient Digital to Analog Encoding [*]

Michael Gibson Jehoshua Bruck

California Institute of Technology
Pasadena, CA 91125
gibson@cns.caltech.edu, bruck@paradise.caltech.edu

Abstract

An important issue in analog circuit design is the problem of digital to analog conversion, namely, the encoding of Boolean variables into a single analog value which contains enough information to reconstruct the values of the Boolean variables. A natural question is: *What is the complexity of implementing the digital to analog encoding function?* That question was recently answered in [5], where matching lower and upper bounds on the size of the circuit for the encoding function were proven. In particular, it was proven that $\left\lceil \frac{3n-1}{2} \right\rceil$ 2-input arithmetic gates are necessary and sufficient for implementing the encoding function of n Boolean variables. However, the proof of the upper bound is not constructive.

In this paper, we present an explicit construction of a digital to analog encoder that is optimal in the number of 2-input arithmetic gates. In addition, we present an efficient analog to digital decoding algorithm. Namely, given the encoded analog value, our decoding algorithm reconstructs the original Boolean values. Our construction is suboptimal in that it uses constants of maximum size $n \log n$ bits while the non-constructive proof uses constants of maximum size $2n + \lceil \log n \rceil$ bits.

1 Introduction

Analog elements have recently been advanced as a way to compute Boolean functions with lower circuit complexity than traditional digital approaches. For example, analog VLSI has been used for hardware implementation of neural networks[1, 2]. With all the interest in analog computation, it is natural to consider the pros and cons of analog circuits at a theoretical level[3, 4].

The computing power of analog elements depends on the basis used. In this paper, the basis $+, -, \times$ will be considered. Circuits using this basis are called *arithmetic* circuits.

All Boolean functions, implemented arithmetically, require size at most $O(2^{n/2})$ [3]. For most Boolean functions, this is a lower bound as well. The construction which achieves this bound, requires that $2^{n/2}$ Boolean functions of $n/2$ variables be encoded into real numbers. More formally,

Definition 1 (Encoding Function)
An encoding function is an injective (one-to-one) mapping $f : \{0,1\}^n \mapsto \Re$.

In this paper we consider fan-in 2 arithmetic circuits, hence, the encoding problem is:

[*] Supported in part by a National Science Foundation Graduate Research Fellowship, by NSF Young Investigator Award CCR-9457811 and by a Sloan Research Fellowship.

Definition 2 (Encoding problem)
Let the basis $(+, -, \times)$ be given, where each operation has fan-in 2. Assuming that real-valued constants are available for free, what is the minimum number of operations necessary to create an encoding function?

Example 1 (Simple Case)
One simple encoding function is $\sum_{i=0}^{n-1} 2^i x_{i+1}$. This can be implemented with $(n-1)$ 2-input multiplications and $(n-1)$ 2-input additions; its size is $(2n-2)$. Note that this can be done with all constants of 2, i.e. the size of the constants does not depend on n.

Wegener [5] showed that the lower bound for the size of fan-in 2, fan-out 1 arithmetic circuit implementing the encoding function is $\lceil \frac{3n-1}{2} \rceil$. He also proved the existence of a circuit which achieves that bound and has constants of size $2^{2n+\lceil \log n \rceil}$.

The main contribution in this paper is an explicit construction, with constants of size $2^{n \log n}$, which achieves Wegener's lower bound on the number of operations needed in digital to analog encoding. In addition, we present an efficient analog to digital decoding algorithm. Namely, given the encoded analog value, our decoding algorithm reconstructs the original Boolean values.

The rest of the paper is organized as follows: in Section 2 the construction and decoding algorithm, as well as examples of each, are presented. In Section 3, the correctness of the decoding algorithm is proved; this establishes that the construction produces an encoding function. In Section 4, the optimality of our construction as well as an upper bound on the constants size are proved.

2 Encoding Function and Decoding Algorithm

In this section, we will introduce our construction for an encoding function, introduce the decoding algorithm, and show examples of each. We start off with a bit of notation. For the rest of the paper, let x^n denote some arbitrary element of $\{0,1\}^n$. The construction is as follows.

Construction 1 (The Encoding Function)
Let $f_n : (\{0,1\}^n, \Re) \mapsto \Re$ be defined recursively by

$$f_n(x^n, C) = \begin{cases} 2 & \text{if } n = 0 \\ x_1 + 2 & \text{if } n = 1 \\ \left(f_{n-2}(x^{n-2}, C+1)\right)(x_{n-1} + C) + x_n & \text{if } n \geq 2 \end{cases}$$

Let

$$K_n = f_n(1^n, 0) + 1$$

Then $f_n(\cdot, K_n)$ is an encoding function. See Figure 1 for a schematic representation of the inductive case.

Example 2 (An Encoding Function)
For example, consider the case $n = 5$.

$$\begin{aligned} f_5(x^5, C) &= ((x_1 + 2)(x_2 + (C+1)) + x_3)(x_4 + C) + x_5 \\ K_5 &= f_5(1^5, 0) + 1 \\ &= [((1+2)(1+1) + 1)(1+0) + 1] + 1 \\ &= 9 \end{aligned}$$

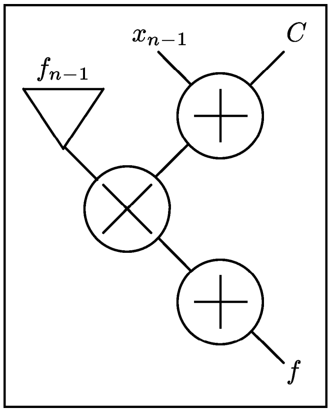

Figure 1: Schematic representation of the inductive case of Construction 1.

Thus,

$$f_5(x^5, 9) = ((x_1 + 2)(x_2 + 10) + x_3)\,(x_4 + 9) + x_5$$

is an encoding function. Its 32 values appear in Table 1. Notice that the 32 values of $f_5(x^5, 9)$ are different.

To show that $f_n(\cdot, K_n)$ is an encoding function, it suffices to find a decoding algorithm. Formally, a function f is injective if and only if it has a left inverse, i.e. there is a function $f^{-1} : \Re \mapsto \{0,1\}^n$ such that

$$\forall x^n \in \{0,1\}^n, (f^{-1} \circ f)(x^n) = x^n$$

The following algorithm is a left inverse for $f_n(\cdot, K_n)$; the proof appears in Section 3.

Algorithm 1 (Decoding Algorithm)

$$\begin{array}{lcl} f_0^{-1}(C) & = & \emptyset \\ f_1^{-1}(x_1 + 2) & = & (x_1 + 2) - 2 \end{array}$$

If $n \geq 2$, then

1. $$x_{n-1} = \begin{cases} 0 & \text{if } f_n(x^n, C) \bmod C \in \{0,1\} \\ 1 & \text{otherwise} \end{cases}$$

2. $$x_n = f_n(x^n, C) \bmod (x_{n-1} + C)$$

3. Apply this algorithm recursively with

$$f_{n-2}(x^{n-2}, C+1) = \frac{f_n(x^n, C) - x_n}{x_{n-1} + C}$$

to get $x_1, \ldots, x_{n-2}$.

$x_1x_2x_3x_4x_5$	$f_5(x^5, 9)$	f_5 mod 9
00000	180	0
00001	181	1
00010	200	2
00011	201	3
00100	189	0
00101	190	1
00110	210	3
00111	211	4
01000	198	0
01001	199	1
01010	220	4
01011	221	5
01100	207	0
01101	208	1
01110	230	5
01111	231	6
10000	270	0
10001	271	1
10010	300	3
10011	301	4
10100	279	0
10101	280	1
10110	310	4
10111	311	5
11000	297	0
11001	298	1
11010	330	6
11011	331	7
11100	306	0
11101	307	1
11110	340	7
11111	341	8

Table 1: Values for the function in Example 2

Next, we give examples of the decoding algorithm in action. For the first step in the decoding algorithm, see Table 1.

Example 3 (Decoding)
Let us consider decoding the 6th entry in the table, namely

$$f_5(0, 0, 1, 0, 1, 9) = 190$$

We have 190 mod 9 = 1. Thus $x_4 = 0$ and $x_5 = 1$. We apply the algorithm recursively with

$$\begin{aligned} f_3(x^3, 10) &= \tfrac{190-1}{9} \\ &= 21 \end{aligned}$$

We have 21 mod 10 = 1. Thus $x_2 = 0$ and $x_3 = 1$. Again, we apply the algorithm recursively, this time with

$$\begin{aligned} f_1(x^1, 11) &= \tfrac{21-1}{10} \\ &= 2 \end{aligned}$$

This is a base case, so $x_1 = 2 - 2 = 0$.

Example 4 (More decoding)
Let us consider another example, the 28th entry in the table.

$$f_5(1, 1, 0, 1, 1, 9) = 331$$

We note that 331 mod 9 = 7. Thus $x_4 = 1$. It follows that

$$\begin{aligned} x_5 &= 331 \bmod (1 + 9) \\ &= 1 \end{aligned}$$

We apply the algorithm recursively with

$$\begin{aligned} f_3(x^3, 10) &= \tfrac{331-1}{1+9} \\ &= 33 \end{aligned}$$

We find that 33 mod 10 $= 3$. Thus $x_2 = 1$.

$$\begin{aligned} x_3 &= 33 \bmod (1+10) \\ &= 0 \end{aligned}$$

Again, we apply the algorithm recursively, this time with

$$\begin{aligned} f_1(x^1, 11) &= \tfrac{33-0}{1+10} \\ &= 3 \end{aligned}$$

This is a base case, so $x_1 = 3 - 2 = 1$.

3 Correctness of Decoding Algorithm

In this section, we prove the correctness of the decoding algorithm (Algorithm 1). The basic idea is to use $f_n(x^n, C) \bmod C$ to recover the values of x_{n-1} and x_n then apply the algorithm recursively. Provided that C is "large enough," we will show that

$$f_n(x^n, C) \bmod C \in \{0, 1\} \qquad \Leftrightarrow \qquad x_{n-1} = 0 \tag{1}$$

This will allow us to recover the required values and apply the algorithm recursively. First, we prove a technical lemma, which is necessary for our proof of Equation 1.

Lemma 3 (Bounds on f mod C)
For all $C \geq K_n$ and $x^{n-2} \in \{0,1\}^{n-2}$, the following holds

$$f_{n-2}(x^{n-2}, 1) \bmod C \quad \in \quad \{2 \ldots C-2\}$$

Proof. We shall consider the case where n is odd. The case for n even is similar. For $n \geq 3$,

$$\begin{aligned} f_{n-2}(x^{n-2}, C+1) &= \big(\big((x_1+2)(x_2+C+\tfrac{n-3}{2})+x_3\big)\ldots(x_{n-3}+C+1)\big)+x_{n-2} \\ &= \text{terms without } C + \text{terms with } C \\ &= \big(\big((x_1+2)(x_2+\tfrac{n-3}{2})+x_3\big)\ldots(x_{n-3}+1)\big)+x_{n-2}+\text{terms with } C \\ &= f_{n-2}(x^{n-2}, 1) + \text{terms with } C \end{aligned}$$

The terms with C disappear modulo C, so

$$f_{n-2}(x^{n-2}, C+1) \bmod C \quad = \quad f_{n-2}(x^{n-2}, 1) \bmod C$$

By the monotonicity of f,

$$f_{n-2}(0^{n-2}, 1) \quad \leq \quad f_{n-2}(x^{n-2}, 1) \quad \leq \quad f_{n-2}(1^{n-2}, 1)$$

Thus, we have

$$\begin{aligned} f_{n-2}(x^{n-2}, 1) &\leq f_{n-2}(1^{n-2}, 1) \\ &= K_{n-2} - 1 \\ &\leq C - 2 \end{aligned}$$

As for the lower bound,

$$\begin{array}{rcl} f_{n-2}(x^{n-2},1) & \geq & f_{n-2}(0^{n-2},1) \\ & \geq & 2 \end{array}$$

Hence,

$$2 \quad \leq \quad f_{n-2}(x^{n-2},C+1) \bmod C \quad \leq \quad C-2$$

□

Theorem 4 (Correctness of decoding algorithm)
The decoding algorithm is correct, i.e. given $f_n(x^n, C)$, it will uniquely and correctly determine x^n.

Proof. (By induction)
The base cases are trivial.
If $n \geq 2$,

$$\begin{array}{rcl} f_n(x^n,C) \bmod C & = & \big(f_{n-2}(x^{n-2},C+1)(x_{n-1}+C)+x_n\big) \bmod C \\ & = & \big(f_{n-2}(x^{n-2},C+1)\times x_{n-1}+x_n\big) \bmod C \end{array}$$

If $x_{n-1}=0$, then $f_n(x^n,C) \bmod C = x_n \in \{0,1\}$
If $x_{n-1}=1$, then

$$\begin{array}{rcl} f_n(x^n,C) \bmod C & = & \big(f_{n-2}(x^{n-2},C+1)\times 1+x_n\big) \bmod C \\ & = & \big(f_{n-2}(x^{n-2},1)\times 1+x_n\big) \\ & \in & \{2,\ldots,C-1\} \end{array}$$

by Lemma 3. Thus Equation 1 holds, i.e.

$$f_n(x^n,C) \bmod C \ \in \ \{0,1\} \qquad \Leftrightarrow \qquad x_{n-1}=0$$

Hence the first step of the algorithm correctly determines x_{n-1} from $f_n(x^n,C)$.

Given x_{n-1}, the second step of the algorithm correctly determines x_n from $f_n(x^n,C)$.

Finally,

$$\begin{array}{rcl} C+1 & > & C \\ & \geq & K_n \\ & > & K_{n-2} \end{array}$$

So we may apply the algorithm recursively. By the induction hypothesis, the algorithm is able to decode $f_{n-2}(x^{n-2},C+1)$ correctly. □

4 Complexity Issues

In this section, we show that our construction achieves Wegener's lower bound on the number of operations of an encoding function[5]. We then prove a bound on the size of the constants involved, which is suboptimal in that Wegener proved (non-constructively) the existence of smaller constants creating an encoding function.

Theorem 5 (Number of operations)
This construction produces a formula with an optimal number of operations.

Proof (By induction)

Let $|f_n|$ denote the number of arithmetic operations in f_n. Then

$$\begin{aligned} |f_0| &= 0 \\ &= \left\lceil \tfrac{3(0)-1}{2} \right\rceil \\[1em] |f_1| &= 1 \\ &= \left\lceil \tfrac{3(1)-1}{2} \right\rceil \end{aligned}$$

For $n \geq 2$,

$$\begin{aligned} |f_n| &= |f_{n-2}| + 3 \\ &= \left\lceil \tfrac{3(n-2)-1}{2} \right\rceil + 3 \\ &= \left\lceil \tfrac{3n-6-1}{2} + 3 \right\rceil \\ &= \left\lceil \tfrac{3n-1}{2} \right\rceil \end{aligned}$$

Which matches Wegener's lower bound[5]. □

Theorem 6 (Constant Size)
For $n \geq 2$, the largest constant in the above construction is of size $\leq 2^{n \log n}$.

Proof. Again, we shall consider the case where n is odd. The even case is similar.

$$\begin{aligned} Const_{max} &= K_n + \tfrac{n-3}{2} \\ &= \Big[\Big(\big((1+2)(1+\tfrac{n-3}{2}) + 1\big) \ldots (1+1) + 1 \Big) (1+0) + 1 + 1 \Big] + \tfrac{n-3}{2} \\ &\leq n^n \\ &= 2^{n \log n} \end{aligned}$$

□

Wegener [5] showed that there exist constants of maximum size $2^{2n+\lceil \log n \rceil}$, which produce an encoding function; hence this construction is suboptimal.

5 Conclusions

We have extended Wegener's result on the optimal size of a formula which solves the encoding problem by presenting an explicit construction. Further, we have devised an algorithm to decode the real numbers that our function produces.

Our constants are larger than one might hope, based on Wegener's [5] existence proof. One interesting extension of this work would be to find a different construction which decreases the size of the constants involved. We note that the encoding function in Example 1, although it has more operations than the function in Construction 1, has constants which do not depend on the number of variables n. It is not clear if there is a formula with an optimal number of gates which uses a set of constants independent of n.

The original interest in encoding is motivated by the construction of formulas for arbitrary Boolean functions with size of $O\big(2^{n/2}\big)$ [3]. In that paper, the encoding is done up front, and only the final output, the real number, is used. The complexity in that case comes from the *decoding.* In this paper, we've given a decoding algorithm, but no arithmetic circuit to implement it. A possible area of further research is the trade-off between encoding complexity and decoding complexity for a given basis set.

Finally, in this paper only formulas with fan-in 2 were considered. A more general analysis would include circuit complexity (i.e. fan-out ≥ 1) and would include bounded fan-in greater than 2.

References

[1] B. E. Boser, E. Säckinger, J. Bromley, Y. L. Cun and L. D. Jackel, *An Analog Neural Network Processor with Programmable Topology,* IEEE Journal of Solid-State Circuits, Vol. 26, No. 12, pp. 2017–2025, Dec. 1991.

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

[3] G. Turán and F. Vatan, *On the Computation of Boolean Functions by Analog Circuits of Bounded Fan-in (Extended Abstract)*, Proceedings of the 35th Annual Symposium on the Foundations of Computer Science (FOCS), pp.553–564, 1994.

[4] A. Vergis, K. Steiglitz and B. Dickinson, *The Complexity of Analog Computation*, Mathematics and Computers in Simulation, **28**, pp.91–113, 1986.

[5] I. Wegener, *On the Complexity of Encoding in Analog Circuits*, to appear in Information Processing Letters.