Efficient Digital-to-Analog Encoding

Michael A. Gibson and Jehoshua Bruck, *Senior Member, IEEE*

***Abstract*— An important issue in analog circuit design is the problem of digital-to-analog conversion, i.e., 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 by Wegener, who proved matching lower and upper bounds on the size of the circuit for the encoding function. In particular, it was proven that $\lceil (3n-1)/2 \rceil$ 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; the nonconstructive proof uses constants of maximum size $2n + \lceil \log n \rceil$ bits.

***Index Terms*— Complexity, construction, decoding, digital-to-analog, encoding.**

I. 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 that 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 $f\colon \{0,\ 1\}^n \mapsto \Re$.

Manuscript received November 1, 1996; revised November 1, 1998. This work was supported in part by a National Science Foundation Graduate Research Fellowship, a National Science Foundation Young Investigator Award CCR-9457811, and a Sloan Research Fellowship. The material in this paper was presented in part at the IEEE International Symposium on Information Theory, MIT, Cambridge, MA, August 16–21, 1998.

The authors are with the California Institute of Technology, MS 136-93, Pasadena, CA 91125 USA (e-mail: {gibson; bruck}@paradise.caltech.edu).

Communicated by R. Laroia, Associate Editor for Source Coding.

Publisher Item Identifier S 0018-9448(99)04358-8.

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

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 $\lceil (3n-1)/2 \rceil$. He also proved the existence of a circuit that 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 $2^{n \log n}$ that 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 II the construction and decoding algorithm are presented, and examples of each are given. In Section III, the correctness of the decoding algorithm is proven; this establishes that the construction produces an encoding function. In Section IV, the optimality of our construction and an upper bound on the size of the constants are proven.

II. Encoding Function and Decoding Algorithm

In this section, we introduce our construction for an encoding function, introduce the decoding algorithm, and show examples of each. We start 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

$$f_n\colon (\{0, 1\}^n, \Re) \mapsto \Re$$

be defined recursively by

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

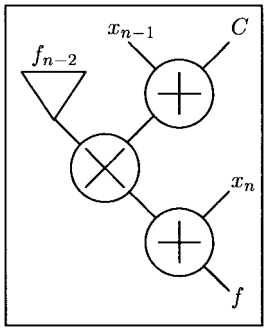

Fig. 1. Schematic representation of the inductive case of construction.

Let $K_n = f_n(1^n, 0) + 1$. Then $f_n(\cdot, K_n)$ is an encoding function. See Fig. 1 for a schematic representation of the inductive case.

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

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

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 I. Notice that the 32 values of $f_5(x^5, 9)$ are distinct.

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 III.

Algorithm 1. Decoding Algorithm:

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

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 $x_1, \cdots, x_{n-2}$.

TABLE I
VALUES FOR THE FUNCTION $f_5(x^5, 9)$ IN EXAMPLE 2. THE VALUES $(f_5 \bmod 9)$ ARE USED IN EXAMPLES 3 AND 4

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

Next, we give examples of the decoding algorithm in action. For the first step in the decoding algorithm, please refer to Table I.

Example 3. Decoding: Let us consider decoding the sixth entry in the table, namely, $f_5(0,0,1,0,1,9) = 190$. We have $190 \bmod 9 = 1$; thus $x_4 = 0$, and $x_5 = 1$. We apply the algorithm recursively with $f_3(x^3, 10) = (190-1)/9 = 21$. We have $21 \bmod 10 = 1$; thus $x_2 = 0$, and $x_3 = 1$. Again, we apply the algorithm recursively, this time with $f_1(x^1, 11) = (21-1)/10 = 2$. 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 $331 \bmod 9 = 7$; thus $x_4 = 1$. It follows that $x_5 = 331 \bmod (1+9) = 1$. We apply the algorithm recursively with $f_3(x^3, 10) = (331-1)/(1+9) = 33$. We find that $33 \bmod 10 = 3$; thus $x_2 = 1$, and $x_3 = 33 \bmod (1+10) = 0$. Again, we apply the algorithm recursively, this time with $f_1(x^1, 11) = (33-0)/(1+10) = 3$. This is a base case, so $x_1 = 3-2 = 1$.

III. 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 shall show that

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

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

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

$$f_{n-2}(x^{n-2}, C+1) \bmod C \in \{2 \cdots C-2\}.$$

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

$$\begin{aligned} & f_{n-2}(x^{n-2}, C+1) \\ &= \left(\left((x_1+2)\left(x_2+C+\frac{n-3}{2}\right)+x_3\right)\cdots \right. \\ &\qquad \left. (x_{n-3}+C+1)\right)+x_{n-2} \\ &= \text{terms without } C + \text{ terms with } C \\ &= \left(\left((x_1+2)\left(x_2+\frac{n-3}{2}\right)+x_3\right)\cdots(x_{n-3}+1)\right) \\ &\quad + x_{n-2} + \text{ terms with } C \\ &= f_{n-2}(x^{n-2}, 1) + \text{ terms with } C. \end{aligned}$$

The terms with C disappear modulo C, so

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

By the monotonicity of f, we have

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

for all x^{n-2}. Thus we have

$$\begin{aligned} f_{n-2}(x^{n-2}, 1) \leq f_{n-2}(1^{n-2}, 1) &= f_n(1^n, 0) - 1 \\ &= K_n - 2 \leq C - 2. \end{aligned}$$

As for the lower bound

$$f_{n-2}(x^{n-2}, 1) \geq f_{n-2}(0^{n-2}, 1) \geq 2.$$

Hence

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

Theorem 4. Correctness of Decoding Algorithm: The decoding algorithm is correct: given $f_n(x^n, C)$, it determines x^n uniquely and correctly.

Proof. By Induction: The base cases are trivial.
If $n \geq 2$, then

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

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

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

which follows from Lemma 3. Thus (1) holds, i.e.,

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

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, $C+1 > C \geq K_n > K_{n-2}$, 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. $\square$

IV. 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. The size of the constants is suboptimal, in that Wegener proved (nonconstructively) the existence of smaller constants that create 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 $|f_0| = 0 = \lceil (3(0)-1)/2 \rceil$. Further, $|f_1| = 1 = \lceil (3(1)-1)/2 \rceil$.

For $n \geq 2$

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

which matches Wegener's lower bound [5]. $\square$

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.

$$\begin{aligned} \text{Const}_{\max} &= K_n + \frac{n-3}{2} \\ &= \left[\left(\left((1+2)\left(1+\frac{n-3}{2}\right)+1\right)\cdots(1+1)+1\right)\right. \\ &\qquad \left. \cdots(1+0)+1+1\right] + \frac{n-3}{2} \\ &\leq n^n \\ &= 2^{n \log n}. \end{aligned}$$

$\square$

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

V. Conclusions

We have presented an explicit construction that extends Wegener's result on the optimal size of a formula that solves the encoding problem. 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 that decreases the size of the constants involved. We note that although the encoding function in Example 1 has more operations than the function in Construction 1, it has constants that do not depend

on the number of variables, n. It is not clear if there is a formula with an optimal number of gates that 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 $O(2^{n/2})$ [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 have given a decoding algorithm, but no arithmetic circuit to implement it. A possible area of further research is the tradeoff 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 J. Solid-State Circuits*, vol. 26, pp. 2017–2025, Dec. 1991.

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

[3] G. Turán and F. Vatan, "On the computation of Boolean functions by analog circuits of bounded fan-in (extended abstract)," in *Proc. 35th Annu. Symp. Foundations of Computer Science (FOCS)*, 1994, pp. 553–564.

[4] A. Vergis, K. Steiglitz, and B. Dickinson, "The complexity of analog computation," *Math. and Comput. in Simulation*, vol. 28, pp. 91–113, 1986.

[5] I. Wegener, "On the complexity of encoding in analog circuits," *Inform. Processing Lett.*, vol. 60, no. 1, pp. 49–52, Oct. 14, 1996.