Annales Univ. Sci. Budapest., Sect. Comp. 17 (1998) 381-389

ON THE SUPERPOSITION OF BOOLEAN FUNCTIONS

M. Tombak (Tartu, Estonia)
I. Sillitoe (Borås, Sweden)

Abstract. The use of memory devices as the means of implementing arbitrarily complex Boolean functions has been applied in a number of fields, including digital signal processing, control and memory based learning. The approach applies the literals of the function to be implemented to the memory address lines and programs, the state of the corresponding memory cell to provide the correct value. Since the propagation delay of the implementation is independent of the form of the function for a given number of literals, the penalty for this flexibility is the number of additional devices and corresponding increase in the size of the device. The exact cost depends upon the precise method of implementation, but it is governed by the number of memory cells, which will be in general $O(2^n)$, where n is the number of literals.

Commercial solutions to this problem, such as the configurable logic block found in family of Xilinx FPGAs, employ a cascaded RAM network, the first layer of which consists of two 16 bit RAMs each with 4 independent inputs and a single 8 bit RAM which forms the final products using the results of previous layer's RAMs and an additional external input. Thus the network is able to implement a limited subset of all possible 9 variable functions at a greatly reduced hardware cost.

The objective of this paper is to investigate the formal properties of cascaded RAM networks and in particular those functions which are expressible with RAM networks sharing common literals and for which $n > 4$. The use of common inputs does not necessarily produce a network which will implement a given function at minimum cost, but does radically increase the number of functions which can be expressed for a relatively small increased cost. This work was inspired by the need for the specification of functions which could be implemented using a 9 variable

Supported by the Estonian Science Foundation Research Grant Nr. 1203.

RAM network.

General method for testing representability of given function via superposition of functions, sharing its variables, is given.

1. Introduction

The use of memory devices as the means of implementing arbitrarily complex boolean functions has found application in a large number of fields, including digital signal processing, control and more recently memory based learning (e.g. [1]). Rather than forming a direct combinational implementation, the technique applies the variables of the function to memory devices address lines and programs the state of the corresponding memory cell to provide the correct value [2]. In this way, the optimality of a direct combinational implementation of the function, measured in terms of the propagation delay and gate count, is exchanged for the flexibility of the sequential design and the unique ability (if non-volatile memories devices are used) to specify dynamically the function.

Since the propagation delay of the implementation is independent of the form of the function for a given number of literals, the penalty for this flexibility is the number of additional devices and corresponding increase in the size of the device. The exact cost depends upon the precise method of implementation, but is governed by the number of memory cells, which will be in general $O(2^n)$, where n is the number of variables. Thus the use of this approach is often limited to applications where n is small (typically ≤ 4) or when there is no requirement for the function to be replicated and the additional cost of a small number of large scale memory devices is justified. However, with the reemergence of modular designs, which are formed of large numbers of regularly connected programmable function blocks within a single chip, the balance between the flexibility of expression available with a memory based implementation and the cost of corresponding hardware is no longer straightforward to assess (e.g. systolic arrays, field programmable gate arrays (FPGAs)).

Commercial solutions to this problem, such as the configurable logic block found in FPGAs, employ a cascaded RAM network, the first layer of which consists of two 16 bit RAMs each with 4 independent inputs and a single 8 bit RAM. The 8 bit RAM forms the final product using the results of previous layer's RAMs and an additional external input; thus the network is able to realize a limited subset of all possible 9 variable functions at a greatly reduced hardware cost.

The objective of this paper is to investigate the formal properties of cascaded RAM networks and in particular those functions which are expressible with RAM networks sharing common variables, and for which $n > 4$. It will be shown that whilst the use of common inputs does not necessarily result in an implementation of a particular function with a minimum gate cost as would be possible with a direct implementation, they can be used to realize large groups of functions at costs significantly lower than those which would be incured with a single layer exhaustive RAM approach.

This work was inspired by the need for the specification of the functions which could be realized by a 9 variable 2 layer RAM network. The network was an intuitive solution to the design of a binary convolution mask, which provided the necessary functionality at a fraction of the cost of an exhaustive RAM implementation, but for which the complete set of functions could not be readily found.

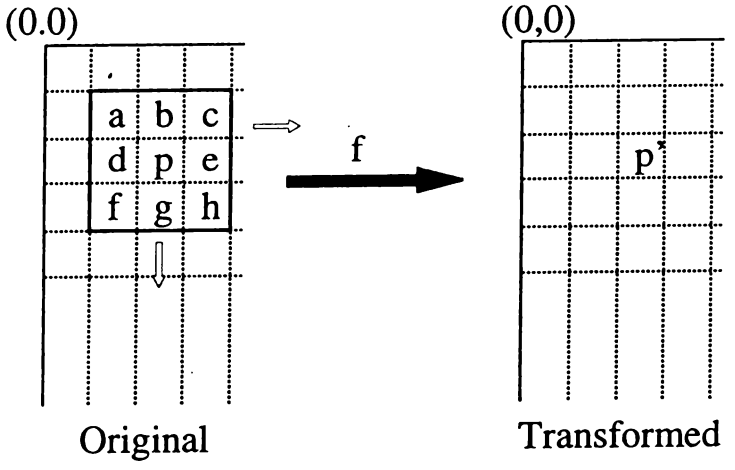

Fig.1. Generalized binary image processing operations

2. Binary image processing

Once an image has been captured and quantized it can be regarded as an array of pixels, where each pixel represents the averege light intensity of a small rectangular region of the image plane. Binary images are in which the value of light intensity is quantized and restricted to either one of two states, black or white; thus a binary image can be regarded as an array of Boolean variables.

This simplified form of representation reduces the image storage requirements and renders the processing to a sequence of Boolean mappings. The effect of these simplifications also reduces the processing time, required by conventional sequential machines, allowing binary image processing to be applied to many real time imaging applications. However, in applications such robotics and document processing the reduction in processing time is still insufficient and special purpose hardware is required.

Operations on binary images can be generalized as the mappings of neighbourhoods of pixels in the original image, through a Boolean function, to a single pixel in the transformed image (see Figure 1). In practice [5] the majority of the neighbourhoods use 9 pixels, consisting of a central pixel (i.e. the p pixel at position $i(x, y)$ within the image in Figure 1) and its 8-connected neighbours $(a, ..., g)$, and the transformed pixel p' is placed at $i'(x, y)$ within the transformed image. Thus binary image processing lends itself to simple hardware implementations using RAM based look up tables.

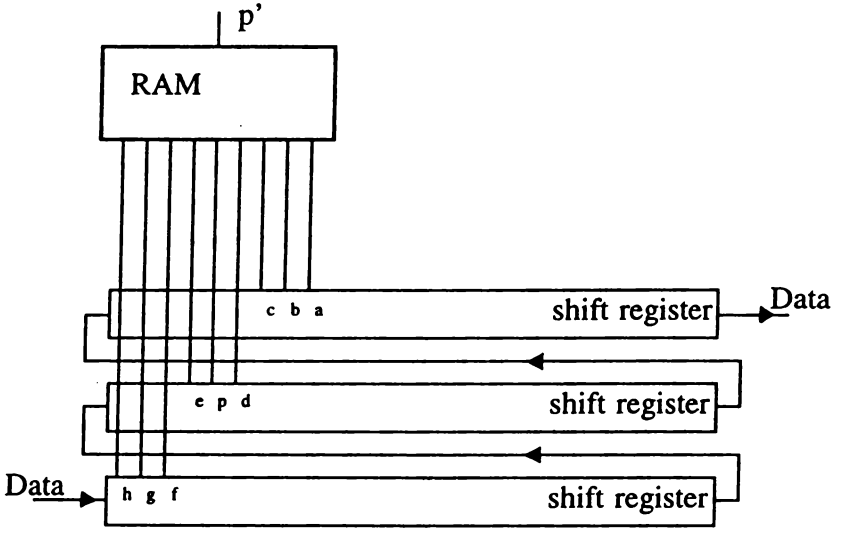

Fig.2. A simplified BIP architecture

3. The architecture of the binary image processor (BIP)

Pixels are sampled from the image in sequence row by row starting at the top left hand corner of the image, and so form a serial data stream. BIP stores 3 rows of pixels and generates the transformed pixel using a RAM look up table as shown in Figure 2 (see [4] for a similar approach).

This implies that for a conventional 9 variable neighbourhood a single look up table would require 512 addressable memory cells, corresponding to approximately 4100 gates. If however, the number of look up tables were increased, so that multiple neighbourhoods might be operated upon in parallel, the number of gates required by this approach becomes a severe limitation.

An alternative approach, designed specifically to avoid this limitation, replaced the single RAM with 5 16 bit RAMs organized as a two layer network, such that adjacent RAMs in the first layers shared a common pair of inputs (Figure 3).

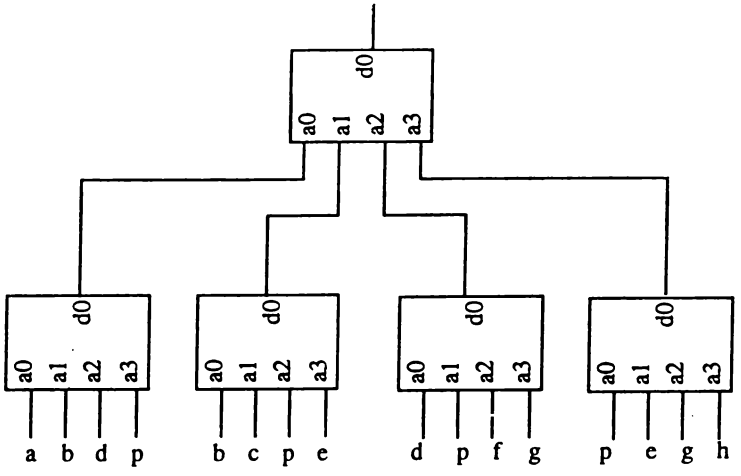

Fig.3. The structure of the two layer RAM network

Thus each of the first layer RAMs operated on an overlapping region of the overall 9 variable mask (see Figure 4).

This had the effect of reducing the simulated gate count to 640, whilst still allowing the necessary functions to be implemented. However, it was not possible simply to determine the complete set of realizable functions or the effects of modifying the network's shared variables; questions whose answers

are necessary before the approach may be generalized. The remainder of the paper describes some of the initial results discovered on the way to answering these questions.

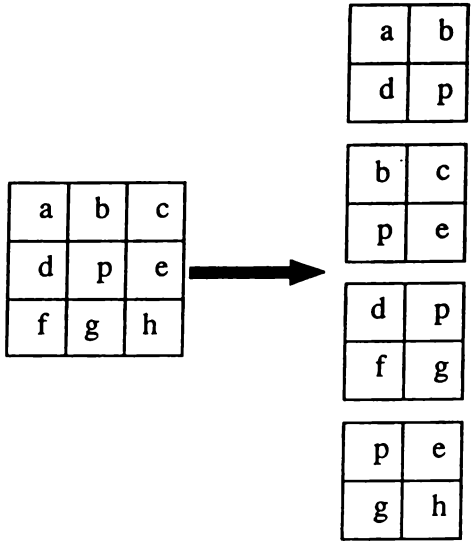

Fig.4. The partition of the 3x3 mask

4. The formal model

The problem of the representability of given 9-variable Boolean function as a superposition of four 4-variable functions, sharing variables accordingly to Figure 4 can be formulated in general fashion.

Given $f(x_1, ..., x_n) : \{0,1\}^n \rightarrow \{0,1\}$. Does there exist functions $g : \{0,1\}^k \rightarrow \{0,1\}$ and $h_1(x_{11}, ..., x_{1n_1}), ..., h_k(x_{k1}, ..., x_{kn_k})$, where $k \leq n$, $1 \leq \leq n_i \leq n$ and $x_{ij} \in \{x_1, ..., x_n\}$ $(1 \leq i \leq k,\ 1 \leq j \leq n_i)$, such that

$$f(x_1, ..., x_n) = g(h_1(x_{11}, ..., x_{1n_1}), ..., h_k(x_{k1}, ..., x_{kn_k}))? \quad (1)$$

We will use for representation of Boolean functions their natural enumeration - we enumerate n-variable Boolean functions with numbers $0...2^{2^n}$ so that binary representation of the number of function f is its truth-table ([3]). Let $[f^n]$ be the number of the function f in the enumeration of n-variable Boolean

functions (we omit n if it is known from context). Let $[f^n]_0, ..., [f^n]_{2^n-1}$ be binary digits of $[f^n]$.

Theorem 4.1. *Variable x_j $(1 \leq j \leq n)$ is fictive for function $f(x_1, ..., x_n)$ iff for every r, p $(0 \leq r < 2^{j-1},\ 0 \leq p < 2^{n-j})$ $[f^n]_l = [f^n]_{l+m}$, where $m = 2^{n-j},\ l = 2rm + p$.*

Proof. Variable x_j is fictive for $f(x_1, ..., x_n)$

$\Longleftrightarrow$ $f(x_1, ..., x_{j-1}, 0, x_{j+1}, ..., x_n) = f(x_1, ..., x_{j-1}, 1, x_{j+1}, ..., x_n)$ for every truth assignment $\alpha_1, ..., \alpha_{j-1}, \alpha_{j+1}, ..., \alpha_n \in \{0,1\}^{n-1}$ to the variables $x_1, ..., x_{j-1}, x_{j+1}, ..., x_n$;

$\Longleftrightarrow$ in the truth-table of the function f the entries, corresponding to the argument vectors $\alpha_1, ..., \alpha_{j-1}, 0, \alpha_{j+1}, ..., \alpha_n$ and $\alpha_1, ..., \alpha_{j-1}, 1, \alpha_{j+1}, ..., \alpha_n$ must be equal for every $\alpha_1, ..., \alpha_{j-1}, \alpha_{j+1}, ..., \alpha_n \in \{0,1\}^{n-1}$;

$\Longleftrightarrow$ bits with (binary) numbers $\alpha_1 ... \alpha_{j-1} 0 \alpha_{j+1} ... \alpha_n$ and $\alpha_1 ... \alpha_{j-1} 1 \alpha_{j+1} ...$ $... \alpha_n$ in the number of the function f, $[f^n]$, must be equal for every $\alpha_1, ..., \alpha_{j-1}$, $\alpha_{j+1}, ..., \alpha_n \in \{0,1\}^{n-1}$ i.e. $[f^n]_{\alpha_1 ... \alpha_{j-1} 0 \alpha_{j+1} ... \alpha_n} = [f^n]_{\alpha_1 ... \alpha_{j-1} 1 \alpha_{j+1} ... \alpha_n}$;

$\Longleftrightarrow$ in decimal:

$$(2) \qquad [f^n]_{\alpha_1 \cdot 2^{n-1} + ... + \alpha_{j-1} \cdot 2^{n-(j-1)} + 0 \cdot 2^{n-j} + \alpha_{j+1} \cdot 2^{n-(j+1)} + ... + \alpha_n \cdot 2^0} =$$

$$[f^n]_{\alpha_1 \cdot 2^{n-1} + ... + \alpha_{j-1} \cdot 2^{n-(j-1)} + 1 \cdot 2^{n-j} + \alpha_{j+1} \cdot 2^{n-(j+1)} + ... + \alpha_n \cdot 2^0}.$$

Let r and p be numbers with binary representations $\alpha_1 ... \alpha_{j-1}$ and $\alpha_{j+1} ... \alpha_n$ correspondingly. We can rewrite (2): $[f^n]_{r \cdot 2^{n-j+1} + p} = [f^n]_{r \cdot 2^{n-j+1} + 2^{n-j} + p}$. If we take $m = 2^{n-j}$ and $l = 2mr + p$, we get $[f^n]_l = [f^n]_{l+m}$. The condition (2) holds for every $\alpha_1, ..., \alpha_{j-1}, \alpha_{j+1}, ..., \alpha_n \in \{0,1\}^{n-1}$ and, therefore, our conclusion is true for every r, p $(0 \leq r < 2^{j-1},\ 0 \leq p < 2^{n-j})$.

Function $h_i(x_{i1}, ..., x_{in_i})$ from (1) has $n - n_i$ fictive variables and Theorem 4.1 splits the bits of its number, $[h_i]$, into 2^{n_i} equivalence classes, 2^{n-n_i} elements in each class. If we sort these classes by least numbers of bits in each class, we get surjection, determined by the list of fictive variables of the function h_i — $\varphi_i : [0, 2^n - 1] \to [0, 2^{n_i} - 1]$, or in binary $\varphi_i : \{0,1\}^n \to \{0,1\}^{n_i}$.

Theorem 4.2. *Function $f(x_1, ..., x_n) : \{0,1\}^n \to \{0,1\}$ is representable via superposition of function $g : \{0,1\}^k \to \{0,1\}$ and $h_1(x_{11}, ..., x_{1n_1}), ...$ $..., h_k(x_{k1}, ..., x_{kn_k})$, where $k \leq n,\ 1 \leq n_i \leq n$ and $x_{ij} \in \{x_1, ..., x_n\}$ $(1 \leq i \leq k,\ 1 \leq j \leq n_i)$, in form*

$$(3) \qquad f(x_1, ..., x_n) = g(h_1(x_{11}, ..., x_{1n_1}), ..., h_k(x_{k1}, ..., x_{kn_k}))$$

if and only if there exist binary strings $\alpha^1, ..., \alpha^k \in \{0,1\}^n$ so that

$$(4) \qquad \text{if} \quad \varphi_i(l) = \varphi_i(m), \text{ then } \alpha^i_l = \alpha^i_m,$$

$$\{(\alpha_l^1, ..., \alpha_l^k) : f(l) = 1\} \cap \{(\alpha_m^1, ..., \alpha_m^k) : f(m) = 0\} = \emptyset. \tag{5}$$

Proof. $\Longrightarrow$ Suppose there exist functions $g, h_1, ..., h_k$ so that (3) holds. We take $\alpha^1 = [h_1^n], ..., \alpha^k = [h_k^n]$. Condition (4) holds due to Theorem 4.1. Suppose that condition (5) is violated for some l, m, i.e. $[h_1^n]_l = [h_1^n]_m, ..., [h_k^n]_l = [h_k^n]_m$, but $f(l) \neq f(m)$. Then $h_1(l) = h_1(m), ..., h_k(l) = h_k(m)$ and $g(h_1(x_{11}, ..., x_{1n_1}), ..., h_k(x_{k1}, ..., x_{kn_k})) = g(h_1(x_1, ..., x_n), ..., h_k(x_1, ..., x_n))$ cannot agree with $f(x_1, ..., x_n)$ for both bitstrings l and m.

$\Longleftarrow$ Let binary strings $\alpha^1, ..., \alpha^k \in \{0,1\}^n$ satisfy conditions (4) and (5). Due to condition (4) α^i can be considered as a natural number of n-variable Boolean function h_i, which depends essentially on variables $x_{i1}, ..., x_{in_i}$. Define $g(y_1, ..., y_k) = \exists \beta \in \{0,1\}^n [f(\beta) = 1 \& y_1 = h_1(\beta) \& ... \& y_k = h_k(\beta)]$. Obviously $g(h_1(\beta), ..., h_k(\beta)) = f(\beta)$ and condition (5) guarantees that g is a function.

We define propositional variables $h_{lm}^n \equiv$ "$[h_l^n]_m = 1$", where $1 \leq l \leq k$ and $1 \leq m \leq 2^n$, and express the condition (5) in the form of propositional formula:

$$\bigwedge_{i \in f^{-1}(1)} \bigwedge_{j \in f^{-1}(0)} \left(\bigvee_{l=1}^{k} (h_{li}^n \oplus h_{lj}^n) \right). \tag{6}$$

After transforming formula (6) into conjunctive normal form we get

$$\bigwedge_{i \in f^{-1}(1)} \bigwedge_{j \in f^{-1}(0)} \bigwedge_{\beta \in \{0,1\}^k} \left(\bigvee_{l=1}^{k} \left((h_{li}^n)^{\beta_l} \vee (h_{lj}^n)^{\beta_l} \right) \right). \tag{7}$$

Condition (4) demands equivalence of some variables. We introduce a new (smaller) set of variables $h_{lm} (1 \leq l \leq k,\ 1 \leq m \leq 2^{n_l})$ to satisfy (4), using the surjection $\varphi(h_{lm}^n) = h_{l\varphi(m)}$:

$$\bigwedge_{i \in f^{-1}(1)} \bigwedge_{j \in f^{-1}(0)} \bigwedge_{\beta \in \{0,1\}^k} \left(\bigvee_{l=1}^{k} \left((h_{l\phi(i)})^{\beta_l} \vee (h_{l\phi(j)})^{\beta_l} \right) \right). \tag{8}$$

Every satisfying assignment for (6) gives a tuple of binary strings, for which the conditions of Theorem 4.2 hold. The Boolean functions $h_1, ..., h_k$ we are searching for are functions with numbers $[h_i^{n_i}] = h_{i1} ... h_{in_i}$ and function g can

be computed using the second part of Theorem 4.2. Formula (8) consists of $|f^{-1}(1)| \cdot |f^{-1}(0)| \cdot 2^k$ clauses, $2k$ literals in each clause and $\sum_{i=1}^{k} n_i$ different variables. Satisfiability problem with such parameters is feasible for our 9 variable 2 layer RAM network problem.

References

[1] **Aleksander I. and Morton H.,** *An Introduction to Neural Computing*, Chapman Hall, 1990.

[2] **Lewin D. and Protheroe D.,** *Design of Logic Systems*, Chapman Hall.

[3] **Regan K. W., Sivakumar D. and Cai J.,** *Pseudorandom Generators, Measure Theory and Natural Proofs,* Technical Report UB-CS-TR 95-02, Computer Science Dept., University at Buffalo, 1995.

[4] **Sterberg S. R.,** Biomedical Image Processing, *IEEE Computer*, (1983), 22-34.

[5] **Veron D.,** *Machine Vision: Automated Visual Inspection and Robot Vision*, Prentice Hall.

M. Tombak
Department of Computer Science
University of Tartu
2. J. Liivi Str.
EE-2400 Tartu, Estonia
mati@cs.ut.ee

I. Sillitoe
University of Borås
Sweden