Performance-Area Trade-Off of Address Generators for Address Decoder-Decoupled Memory

Sambuddhi Hettiaratchi, Peter Y.K. Cheung, Thomas J.W. Clarke

Department of Electrical and Electronic Engineering
Imperial College of Science, Technology and Medicine, London
E-mail: s.hetti@ic.ac.uk, p.cheung@ic.ac.uk, t.clarke@ic.ac.uk

Abstract

Multimedia applications are characterized by a large number of data accesses and complex array index manipulations. The built-in address decoder in the RAM memory model commonly used by most memory synthesis tools, unnecessarily restricts the freedom of address generator synthesis. Therefore a memory model in which the address decoder is decoupled from the memory cell array is proposed. In order to demonstrate the benefits and limitations of this alternative memory model, synthesis results for a Shift Register based Address Generator that does not require address decoding are compared to those for a counter-based address generator that requires address decoding. Results show that delay can be nearly halved at the expense of increased area.

1. Introduction

Multimedia applications such as video and image processing are often characterized by a large number of data accesses. Many image processing applications such as the block matching motion estimation algorithm also contain complex index manipulations, resulting in complex address patterns. Furthermore, the high throughput required for such applications imposes tight timing constraints on address generation. Therefore, optimization of address generation in data transfer intensive applications is critical to system synthesis.

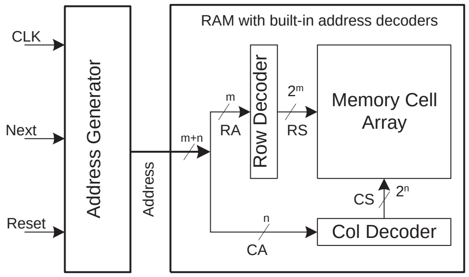

Figure 1. Conventional RAM model with address generator

Although memory synthesis and optimization have been studied elsewhere, and a good overview can be found in [9], most of this research is based on a random access memory (RAM) model as shown in Figure 1. This common RAM model accepts a binary coded address and decodes it using built-in decoders into row and column select signals. Such a memory architecture is advantageous, or even necessary, for applications where the sequence of access to memory is not predictable. For many application-specific integrated circuits (ASICs), the address sequences are usually known *a priori* and the actual sequence often follows a reasonably regular pattern. In this paper, we examine the possible impact on area and performance of memory access related circuitry in eliminating the row and column address decoders from the memory by incorporating any necessary decoding with the address generation circuit. The novel contributions of this paper are: (1) to propose an *"address decoder-decoupled memory"* model that separates the built-in address decoder from memory and incorporates the address generation and any address decoding logic into one or more finite state machines (FSMs); (2) to evaluate such a generic model for address sequencing in the context of circuit synthesis and assess its limitations; (3) to propose a shift register based architecture with "two-hot" encoding as an efficient implementation of such FSMs for regular address sequences; (4) to compare the proposed approach with those using conventional address counter/decoding methods in terms of area and performance.

This paper is organized as follows. Section 2 gives a brief summary of the previous work. Section 3 presents a generalized model for the address generator targeted at the address decoder-decoupled memory (ADDM). Section 4 describes a novel shift register based address generator architecture which does not use address decoders. Section 5 describes the procedure used to map an address sequence to this particular architecture. Section 6 reports synthesis results and Section 7 contains conclusions and possibilities for future work.

2. Related work

Memory synthesis optimizations such as *in-place mapping, loop transformations* and *array clustering* typically add extra complexity to the address sequences in the non-optimized application code [9]. Furthermore, most modern memory architectures are based on *memory hierarchy* [2]. This involves adding extra smaller and faster layers of memory between larger memory and computational units and introduces extra data transfers [12]. Explicit transfer memory hierarchy, a memory hierarchy whose address sequences are known and synthesized at compile time, requires extra address generation [12].

In many digital signal processing applications the array access patterns are regular and periodic [4]. In these cases it becomes feasible and efficient to generate the necessary address patterns either directly from a dedicated counter or via circuit transformations applied to a counter output [4]. PHIDEO silicon compiler is targeted at stream-based video applications [6]. The address allocation task of PHIDEO trades off memory size against addressing cost. ZIPPO tool which is integrated with PHIDEO considers several address streams accessing different memory modules on-chip, and synthesises an address generator that is area optimized by sharing hardware [5].

The combined ADOPT methodology for address generator synthesis described in [8] is a general framework for optimizing address generators. ADOPT considers both counter-based and arithmetic-based address generator styles and also considers hardware sharing. Schmit and Thomas employ some interesting properties of the exclusive-OR function to overcome the offset addition problem that is introduced by *array clustering* memory optimization [10].

All these works use a conventional RAM model. Access to memory is specified by a binary coded address which is split to form row and column addresses. These are then decoded to access two dimensional arrays of memory cells. Such memory architecture has the advantage of reducing the number of signals needed to access memory, in particular for off-chip devices. It also decouples memory accessing from address sequencing, making the device suitable for any application.

However, some researchers have argued that the built-in address decoder which is present in the conventional RAM model restricts the freedom of memory system optimizers to arrive at an optimal memory architecture for a given algorithm [1, 3, 11]. This is particularly true for application-specific integrated circuits where the address sequences are deterministic and often regular. Under such circumstances, decoupling the address decoder from the memory array and incorporating it into the address generator circuit may offer more scope for optimization.

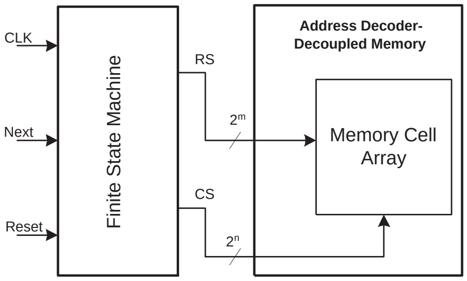

Figure 2. Finite state machine address generator model for the address decoder-decoupled memory

3. Generalized address generator model for the address decoder-decoupled memory

In the case of deterministic access patterns the address generator for the address decoder-decoupled memory can be generalized as an FSM (Figure 2). The problem of address generator optimization can then be formulated as a generic state encoding and assignment problem. Unfortunately, for a repetitive address sequence of length N, an FSM with N states is required. For a relatively long sequence, which is common in many applications, this presents an intractable problem to a logic optimizer, resulting in large circuits that take a very long time to synthesis. Alternatively, the general FSM architecture can be replaced by a structured solution based on shift registers. These two alternatives were investigated for a simple incremental access sequence (i.e. 0,1,2,...,N-1). Both circuits were synthesized with Synopsys' Design Compiler for a 0.18μ CMOS process. Figures 3 and 4 show the delay and area for FSM based and the shift register based solutions respectively.

The shift register, in general, is over twice as fast as the binary encoded symbolic state machine with only 10% increase in area. Moreover, as the sequence length increases, the synthesis time for the symbolic state machine becomes impractical. (For N=256, FSM synthesis took over 6 hours compared to 36 minutes for the shift register solution on a SUN Ultra-5 workstation.)

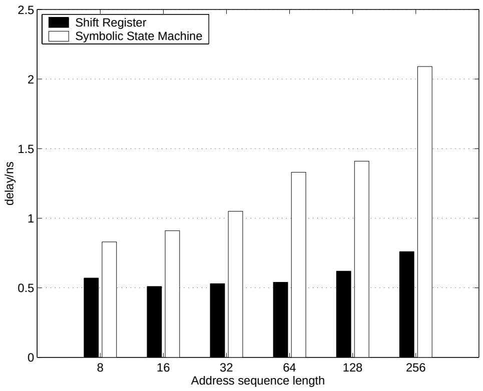

Figure 3. The delay through the address generator for different address sequences lengths

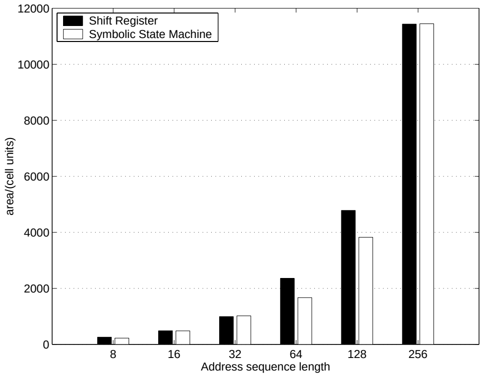

Figure 4. The area of the address generator for different address sequences lengths

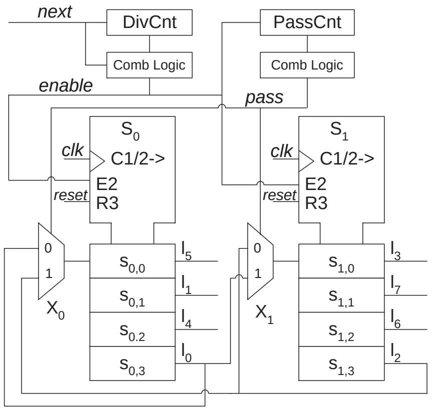

Figure 5. SRAG architecture

4. Shift register based address generator architecture

A shift register can be used to generate a simple incrementing sequence of addresses for the decoder-decoupled memory. For more complex address sequences, a more sophisticated architecture is required. We propose a Shift Register based Address Generator (SRAG) architecture, as shown in Figure 5, which can generate many regular address sequences, especially those found in block-based image processing algorithms. This design is an improvement on the Sequential FIFO memory (SFM) proposed by Aloqeely [1]. SFM works on the same principle as a RAM but the address decoder is replaced by two single-bit shift registers, one for read and the other for write (Figure 6). SFM, despite its advantages, suffers from three main limitations. Firstly, SFM assumes that memory is one dimensional whereas memory arrays are two dimensional. Secondly, SFM uses one-hot encoding for its address generator state machine. Finally, SFM is a first-in first-out (FIFO) memory and cannot be applied to other types of address sequences such as block access.

In contrast, our SRAG architecture works explicitly with a two-dimensional memory. Two dimensional memory allows us to use “two-hot” encoding, “one-hot” in each of the row and column address selections, which takes up far less area than one-hot encoding. The two dimensional arrangement of the memory array naturally implements decoding for two-hot encoded addresses and therefore two-hot encoding does not incur any delay penalty over one-hot encoding. Finally, as we demonstrate in this paper SRAG is capable of implementing access patterns other than FIFO type access.

The complete SRAG is composed of a row SRAG and a column SRAG controlling the row select (RS) and the column select (CS) lines respectively. The row and col-

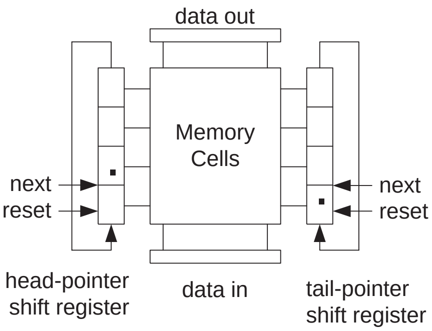

Figure 6. Sequential FIFO memory

umn SRAGs are identical in architecture. Therefore, for the sake of simplicity, we will consider address generation for one dimension of the memory array which has just one set of select lines. Figure 5 shows the schematic for such an SRAG containing two shift registers and two multiplexors. All shift registers have a clock input $C1$, an enable input $E2$, and a reset input $R3$.

The general architecture of the SRAG is composed of a set of shift registers, $S = (S_0, S_1, \ldots, S_{N-1})$, a set of two input multiplexors, X = $(X_0, X_1, \ldots, X_{N-1})$, and two synchronous binary counters, DivCnt and PassCnt. The SRAG has inputs *clk*, *next* and *reset*, internal control signals *enable* and *pass* and an output set of select lines, $L = (l_0, l_1, \ldots, l_{w-1})$. Each shift register $S_i \in S$ is defined as a set of flip-flops, $(s_{i,0}, s_{i,1}, \ldots, s_{i,M_i-1})$.

When $enable = 0$ all shift registers are disabled and retain their previous states. When $enable = 1$, at every rising edge of *clk*, $s_{i,j} := s_{i,j-1}$ where $i = 0 \ldots N-1, j = 1 \ldots M_i - 1$. The value assigned to the first flip-flop of each shift register is controlled by the *pass* signal. If $pass = 1$ then $s_{(i+1) mod N,0} := s_{i,M_i-1}$ where $i = 0 \ldots N-1$, otherwise if $pass = 0$ then $s_{i,0} := s_{i,M_i-1}$ where $i = 0 \ldots N-1$. The set of multiplexors implements the controlled assignments to $s_{i,0}$ where $i = 0 \ldots N-1$. If $N = 1$ multiplexors are not required. At any given time only one flip-flop in the set S can output the value 1, which we call the *token*.

The output of each flip-flop of each shift register $s_{i,j}$ is mapped onto a distinct select line, $l_k \in L$. The token travels from one flip-flop to the next, thus activating the select lines in the desired order. The subscripts of the select lines correspond to the one-dimensional memory address, a_n. The mapping of one-dimensional addresses to the flip-flops is described in detail in the next section.

In general, address generators are driven by a *next* signal. Every *next* signal advances the address generator from the current position in the address sequence a_n to the next position in the address sequence a_{n+1}. Therefore, the *next* signal controls the *enable* signal. If the current address in the address sequence is identical to the next address, i.e, $a_n = a_{n+1}$, then it is necessary to pass the *next* signal through a divider. DivCnt counts the number of *next* signals up to a predetermined value d_C. The connected combinational logic block asserts *enable* when the counter value equals $d_C - 1$ and the input *next* is also high. The SRAG only has one DivCnt, this imposes the restriction that the length of each repetition of each address in the address sequence should be equal. For example, the SRAG shown in Figure 5, with $d_C = 2$ and assuming the *pass* signal is always asserted, gives the address sequence $5, 5, 1, 1, 4, 4, 0, 0, 3, 3, 7, 7, 6, 6, 2, 2$. In contrast, the sequence $5, 5, 5, 1, 1, 4, 4, 0, 0, 3, 3, 7, 7, 6, 6, 2, 2$ has a d_C of 3 for address 5 and a d_C of 2 for all other addresses and violates the DivCnt restriction.

PassCnt controls the number of iterations for which a particular shift register $S_i \in S$ retains the token by controlling the *pass* signal that switches the two input multiplexors. It does so by counting the number of *enable* signals. When the counter value reaches a predetermined value $p_C - 1$ the connected combinational logic asserts the *pass* signal. There is only one PassCnt in the SRAG. This imposes the restriction that the length of each shift register multiplied by the number of iterations of that shift register has to be the same for all shift registers. For example, the SRAG shown in Figure 5, with $p_C = 8$ and $d_C = 1$ gives the sequence $5, 1, 4, 0, 5, 1, 4, 0, 3, 7, 6, 2, 3, 7, 6, 2$. In contrast, the sequence $5, 1, 4, 0, 5, 1, 4, 0, 5, 1, 4, 0, 3, 7, 6, 2, 3, 7, 6, 2$ has a p_C of 12 for S_0 and 8 for S_1 and therefore would violate the PassCnt restriction.

The restrictions on DivCnt and PassCnt limit the spectrum of addressing patterns that can be implemented with SRAG. This can be relaxed by using multiple counters that provide more flexibility in the sequences that can be generated. Furthermore, it is not necessary to use counters for deriving the *enable* and the *pass* signals. It is possible to use shift registers or interacting FSMs to derive these signals.

5. Automatic Mapping Procedure

The input to the mapping procedure is a row address (RA) and a column address (CA) sequence. The mapper works on the RA and the CA sequences separately. The function of the mapper is to identify the shift register set S, the common division count d_C and the common pass count p_C for a given address sequence.

In order to better explain the mapping procedure we use the two dimensional array $new\_img[I_0][I_1]$ from the block matching motion estimation algorithm (Figure 7) where $img\_width = 4, img\_height = 4, mb\_width = 2, mb\_height = 2$ and $m = 0$. Table 1 shows the resulting linear address sequence (LinAS), row address sequence (RowAS), and column address sequence (ColAS). In this paper, we assume

Name	Address Sequence
LinAS	$0,1,4,5,2,3,6,7,8,9,12,13,10,11,14,15$
RowAS	$0,0,1,1,0,0,1,1,2,2,3,3,2,2,3,3$
ColAS	$0,1,0,1,2,3,2,3,0,1,0,1,2,3,2,3$

Table 1. Address sequences

Parameter	Value
I	$0,0,1,1,0,0,1,1,2,2,3,3,2,2,3,3$
D	$2,2,2,2,2,2,2,2$
R	$0,1,0,1,2,3,2,3$
U	$0,1,2,3$
O	$2,2,2,2$
Z	$0,1,4,5$
S	$(0,1),(2,3)$
P	$4,4$
d_C	2
p_C	4

Table 2. Mapping parameters for column address sequence

that the *new_img* array is row-major mapped to the memory array so that $RA = I_0$, $CA = I_1$, and linear address, $LA = I_0 \times img\_width + I_1$. Data organization within the memory cell array can greatly affect the available regularity at the RowAS and ColAS level. The optimal data organization within the memory cell array for a given arbitrary address sequence is not addressed in this paper. Since the same procedure applies to both the RowAS and the ColAS we will now focus on the RowAS alone.

Table 2 shows the mapping parameters for the RowAS. The input to the procedure is the address sequence I. The outputs are S, d_C and p_C. The following describes the sequence of steps involved in the mapping.

- The division count set D is found by counting the consecutive repetitions of individual addresses in I. The restriction on DivCnt means that all 8 elements of D should be equal. In this case, d_C is equal to any element of the sequence D.

- Once d_C is known, I is reduced by replacing the repeated blocks by a single element to give the reduced address sequence R, e.g., 00 is replaced by 0.

- Then the mapper examines R to find the unique address sequence U. An address appearing in R is recorded in U if that address is not already recorded in U. The order in which addresses are recorded in U corresponds to the order in which they first appear in R.

```
for(g=0; g < img_height/n; g++)
  for(h=0; h <img_width/n; h++) {
    m_vect[g][h] = big;
    for(i=-m; i<m ;i++)
      for(j=-m; j<m ;j++) {
        diff = 0;
        for(k=0;k<mb_height;k++)
          for(l=0;l<mb_width;l++) {
            diff += abs(new_img[g*mb_height+k][h*mb_width+l]
                  - old_img[g*mb_height+i+k][h*mb_width+j+l]);
          }
        m_vect[g][h] = min(diff, m_vect[g][h]);
      }
    }
```

Figure 7. Block matching motion estimation algorithm

- For each of the unique addresses in U, the number of times they occur and the position of their first occurrence in R is recorded in the sets O and Z respectively.

- The grouping and mapping of select lines to shift registers is done in two steps. First, initial grouping is done by checking if two consecutive unique addresses, $u_k, u_{k+1} \in U$, that occur the same number of times also have consecutive first appearances in R. If they do then the select lines, $l_{u_k}, l_{u_{k+1}} \in L$ are grouped and mapped to the same shift register, $s_{i,j} \leftarrow l_{u_k}, s_{i,j+1} \leftarrow l_{u_{k+1}}$ else $s_{i,M_i-1} \leftarrow l_{u_k}, s_{i+1,0} \leftarrow l_{u_{k+1}}$. Initial grouping may fail for certain address sequences such as $1,2,3,4,3,2,1,4$, therefore, a verification step follows. The number of shift registers N is simply the number of groups and the number of flip-flops in each group M_i is the number of elements in each group.

- Finally, for each shift register pass count p_C is found and recorded in P. Pass count is calculated by finding the length of R that is produced by each of the shift registers. The restriction on the PassCnt means that all elements in P should be equal. In this case p_C is equal to any element of the sequence P.

We have implemented these mapping rules in our SRAdGen tool. The tool accepts a sequence of one-dimensional addresses and, if mapping is successful, produces synthesisable VHDL code describing the corresponding SRAG.

6. Experimental Results

In this section, we present synthesis results in terms of area and delay of the SRAG architecture and compare them to those for a counter-based address generator using address decoders (CntAG). Both types of address generator were synthesized with Synopsys' Design Compiler for a 0.18μ CMOS process. The counter-based style was chosen as the benchmark because, for regular access patterns, it performs better than arithmetic-based address generators [7]. Note that we cannot compare SRAG with SFM because SFM is only a FIFO memory. The address sequences used are the write and read sequences for the array *new_img* in the block matching motion estimation algorithm shown in Figure 7. This code segment defines the read sequence for *new_img* which exhibits block-based memory access. However, it does not define the production order (i.e. write sequence) for *new_img*. Therefore, we assume that the write sequence is such that LinAS is incremental (i.e. 0,1,2,...,N).

Delay results in Figure 8 indicate that for both read and write accesses (excluding the delay through the memory cell array) , the SRAG is on average approximately twice as fast as the CntAG. The delay through the SRAGs increases slowly with array size. This is because the delay is contributed mainly by the counters in the control circuit, which is dependent on the repetitive pattern in the address sequence and not on the size of the array. In contrast, the delay in the CntAG increases much faster with array size. This is because as *N* gets larger, the size and delay of the address decoder dominates. Figure 9 shows the delay through each component of the CntAG, the total delay is the sum of the counter delay and the worst of the row or the column decoder delay. This shows that as the array size increases the decoder delay begins to dominate.

The area results in Figure 10 illustrate the *performance-area* trade off. SRAG may be twice as fast as CntAG, but it is also approximately three times larger in area. However, since the address generator/decoder circuit is in general a small fraction of the total area of the memory, this area penalty is usually unimportant when compared to the speed increase.

Table 3 shows the average delay reduction and area increase factors for several other examples. *dct* refers to an access sequence from a separable discrete-cosine transform (DCT). *zoombytow* is a sequence from an image zooming algorithm. And *motion_est* and *fifo* are the read and write sequences for our motion estimation example, respectively.

7. Conclusion and Future Work

The results, presented in this paper, have shown the performance gains achievable from address decoder decoupling. However, it is clear that the increase in performance comes at a cost in area. It is possible to reduce the area of SRAG through enhancements such as reuse of control circuitry between the row and the column address sequences or exploiting the interaction between the row and the column address generators. However, the contribution of this paper is not only the SRAG architecture but also the encour-

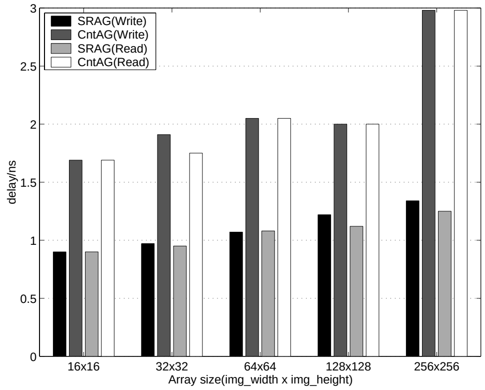

Figure 8. Address generator delay for different array size

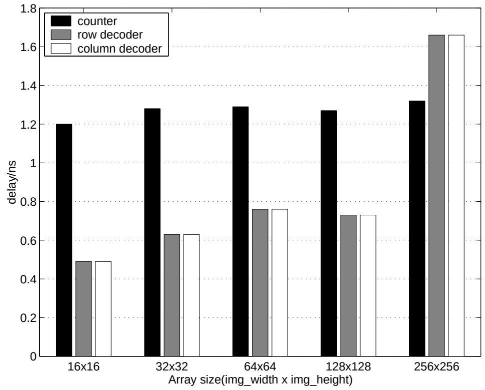

Figure 9. Delay figures for the components of the CntAG

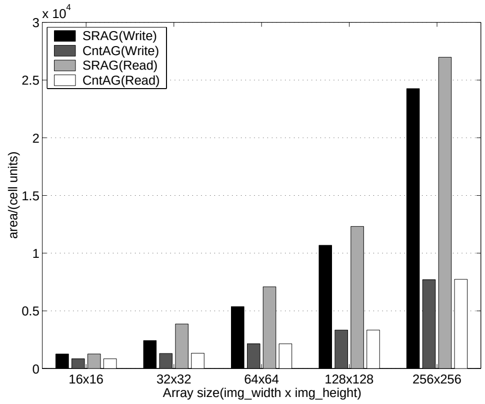

Figure 10. Address generator area for different array size

Example	Avg. Delay reduction factor	Avg. Area increase factor
dct	1.7	3.2
zoombytwo	1.7	3.1
motion_est	1.8	3.0
fifo	1.9	2.4

Table 3. Average delay reduction and area increase for different examples

aging results it gives for the address decoder decoupling approach to array based memory access in general. For applications where the access patterns are regular, such as image and video processing, this approach can effectively eliminate the need for address decoding. However, the physical-level viability and reliability of the ADDM has to be demonstrated. For example, it must be guaranteed that no two row select lines will be asserted at the same time as this could corrupt data in the memory.

This particular work operates on a single address sequence at a time. Therefore, any reuse of address generation circuity between different address sequences is not considered. The reuse of address circuity between different address sequences in space and time can greatly reduce the area resources required. As most modern high-performance memory systems are based on distributed memory architectures, the interconnect and routing costs should also be considered [7].

Although we expect this decoder decoupling approach to reduce power dissipation, in this work we have not carried out a rigorous study of it. Furthermore, SRAG architecture is somewhat limited in its application and is very much targeted towards block-based image processing applications and FIFO type access. If SRAG is not applicable to a particular access pattern then , for example, CntAG architecture or an arithmetic-based architecture can be used. We have not demonstrated the impact of delay reduction achieved through decoder decoupling on the overall memory access delay due to lack of data for the memory cell array.

Our final goal is to discover algorithms and heuristics which can explore the vast design space opened up by address decoder decoupling at a high level of abstraction and choose the best architecture for low level circuit optimization.

Acknowledgement

This work was funded by LSI Logic Corporation and Overseas Research Students Awards Scheme. The authors would like to acknowledge the contributions of Mike Brookes, George Constantinides, Andreas Dante, Nishanth Kulasekeram, Andrew Royal and Marcus von Scotti.

References

[1] M. Aloqeely. A simple alternative for storage allocation in high-level synthesis. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, volume 6, pages 377 – 380, June 1998.

[2] L. Benini and G. de Micheli. System-level power optimization: Techniques and tools. *ACM Transations on Design Automation of Electronic Systems*, 5(2):115–192, Apr. 2000.

[3] S. Gerez and E. Woutersen. Assignment of storage values to sequential read-write memories. In *Proceedings of the European Design Automation Conference*, pages 302 – 308, Sept. 1996.

[4] D. Grant, P. B. Denyer, and I. Finlay. Synthesis of address generators. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 116 – 119, Nov. 1989.

[5] D. Grant, J. V. Meerbergen, and P. Lippens. Optimization of address generator hardware. In *Proceedings of the European Design and Test Conference*, pages 325 – 329, Feb. 1994.

[6] P. Lippens, J. V. Meerbergan, A. V. der Werf, and W. Verhaegh. PHIDEO: a silicon compiler for high speed algorithms. In *Proceedings of the European Conference on Design Automation*, pages 436 – 441, Feb 1991.

[7] M. Miranda, F. Catthoor, M. Janssen, and H. D. Man. ADOPT: efficient hardware address generation in distributed memory architectures. In *Proceedings of the International Symposium on System Synthesis*, pages 20 – 25, Nov. 1996.

[8] M. Miranda, F. Catthoor, and M. J. H. D. Man. High-level address optimization and synthesis techniques for data-transfer-intensive applications. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 6(4):677 – 686, Dec. 1998.

[9] P. R. Panda, F. Catthoor, N. D. Dutt, K. Danckaert, E. Brockmeyer, C. Kulkarni, A. Vandercappelle, and P. G. Kjeldsberg. Data and memory optimization techniques for embedded systems. *ACM Transactions on Design Automation of Electronic Systems*, 6(2):149 – 206, Apr. 2001.

[10] H. Schmit and D. E. J. Thomas. Address generation for memories containing multiple arrays. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(5):377–385, May 1998.

[11] J. van Sas, F. Catthoor, L. Inzé, and H. D. Man. Testability strategy for registers and memories in a multi-processor architecture. In *Proceedings of the European Test Conference*, pages 294–303, Apr. 1989.

[12] S. Wuytack, Jean-Philippe, F. V. Catthoor, and H. J. D. Man. Formalized methodology for data reuse exploration for low-power hierarchical memory mappings. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 6(4):529–537, Dec. 1998.