

MEMORY SYNTHESIS

**HIGH LEVEL DESIGN AND MODELING
CS 812**

Submitted by:

Avneesh Singh Verma (2001VLS010)

Deependra Kumar Jain (2001VLS012)

INDEX

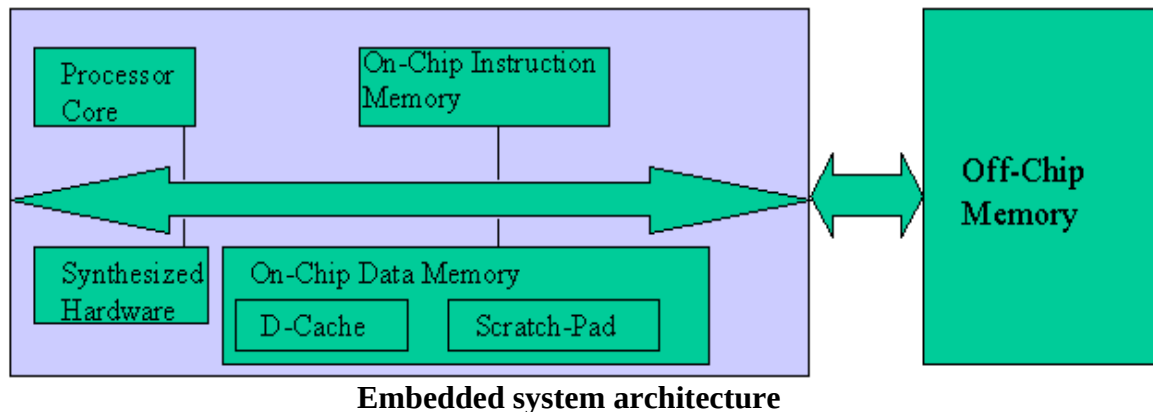
- INTRODUCTION
- OFF CHIP MEMORY
 - DRAM ORGANISATION
 - DRAM ACCESS MODES
 - OPTIMIZATION TECHNIQUES FOR DRAM
 - POWER OPTIMIZATION
- ON CHIP MEMORY
 - MEMORY ORGANIZATION
 - EFFICIENT UTILIZATION OF SCRATCH-PAD
 - POWER OPTIMIZATION
- CONCLUSION
- REFERENCES

INTRODUCTION

The design of modern digital system is influenced by several technological and market trends, including the ability to manufacture ever more complex chips, but under increasingly shorter time to market. This particular combination of increasing design complexity coupled with shrinking product design cycle times has fueled the need for design reuse in the IC-design industry. System-level design reuse is enabled by modern design libraries, which frequently consists of pre-designed mega cells. System –on-chip or system on silicon are the term used to describe product containing and integrating such mega cells onto a single silicon chip. This trend is expected to continue with the shrinking device sizes and with the capability of embedding DRAMS with logic on the same chip.

Digital circuits can be classified in two categories one is general purpose systems and another is application specific or Embedded systems.

Architecture of embedded system is consists of software and the hardware part along with memory modules.



Three important aspects of memory for Embedded systems are :

Area : 50-70% of ASIC/ASIP may be memory.

Performance : 10-90% of system performance may be memory related.

Power : 25-40% of system power may be memory related.

So, it may be seen that memory plays a very crucial role in embedded system designs. In recent years there is considerable increase in the processor speed but increase in speed of memory is very less. That's why memory is bottleneck of all the systems. Therefore we require optimization methods for increasing the performance and reducing the power of both on-chip and off-chip memories. In rest of the report these topics are dealt in detail.

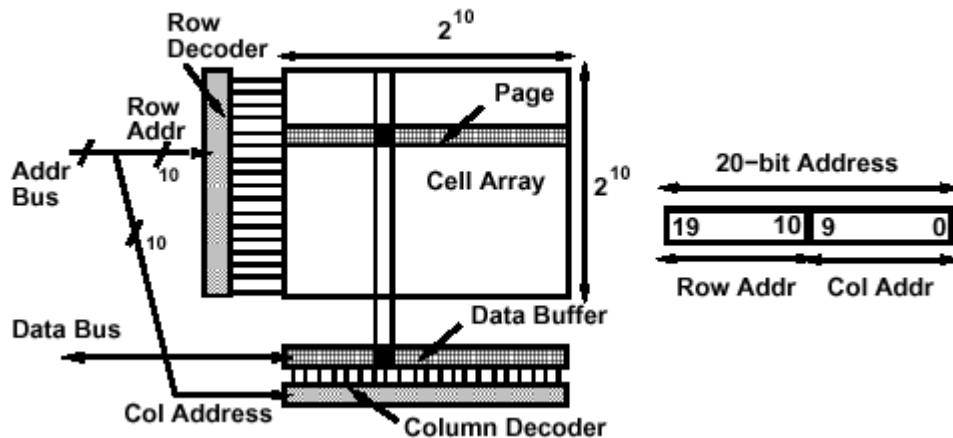
CHAPTER 2

OFF-CHIP MEMORY

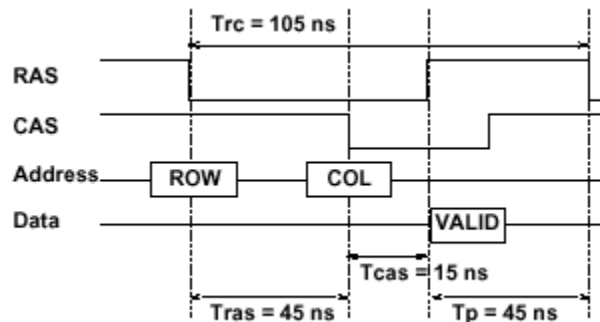
DRAM ORGANIZATION

In DRAMS Rows and columns addresses are used at different points of time, the same address pins are used to time-multiplex between row and column addresses.

The DRAM access is shown with 20 bit multiplexed address Row address from 19-10 and Column address from 9-0



DRAM ORGANISATION



TIMING DIAGRAM FOR MEMORY OPERATIONS

DIFFERENT ACCESS MODES

Read Mode – single word read, involving both row-decode and column-decode.

Write Mode – single word write, involving both row-decode and column-decode.

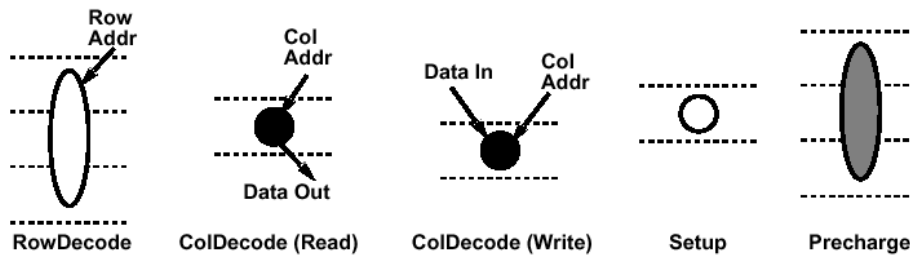
Read-Modify-Write (R-M-W) Mode – single word, involving read from an address, followed by write to the same address. This mode involves one row-decode and column-decode stages each, and is faster than two separate Read and Write accesses.

Page Mode Read – successive reads to multiple words in the same page.

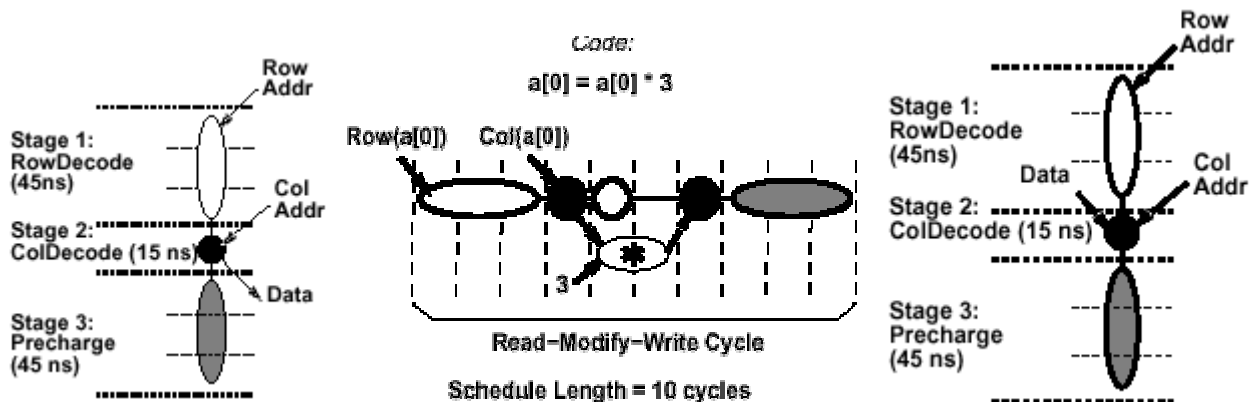
Page Mode Write – successive writes to multiple words in the same page.

Page Mode Read-Modify-Write – successive R-M-W updates to multiple words in the same page.

Memory read and write operation can be represented by control distribution flow models for DRAMS. We can model different operations by nodes in CDFG.



So we can represent READ, WRITE and R-M-W operations.



Memory read

Read-Modify-Write

Memory-Write

OPTIMIZATION TECHNIQUES FOR DRAMS

1. CLUSTERING OF VARIABLES:

scalar variables are normally assigned to on-chip registers. However, if the number of such variables is large, it might be necessary to store these variables in memory. A related optimization problem that arises in this address assignment is that, consecutive accesses to two different scalar

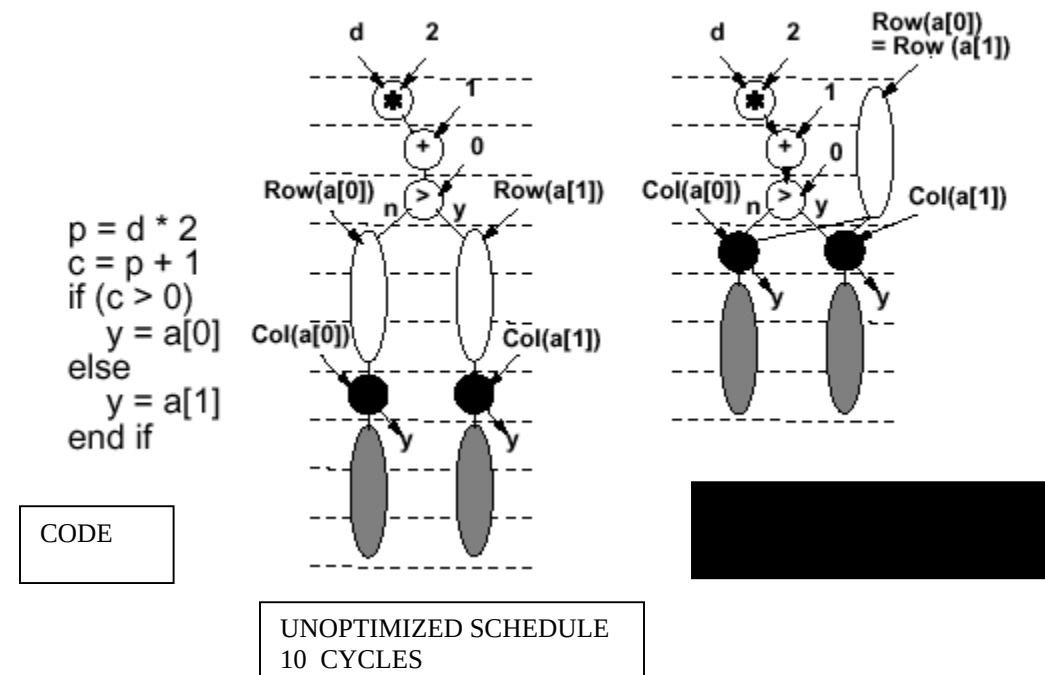
variables can be implemented as a single page mode operation if both are located in the same memory page.

2 REORDERING OF MEMORY ACCESSES

The correct ordering of memory accesses is critical for exploiting efficient memory access modes such as R-M-W. For example, in the code: “ $a[i] = b[i] + a[i]$ ”, the sequence of accesses: “Read $b[i]$! Read $a[i]$! Write $a[i]$ ” allows the utilization of the R-M-W mode for the address $a[i]$, while the sequence “Read $a[i]$! Read $b[i]$! Write $a[i]$ ” does not allow the mode, because of the intervening “Read $b[i]$ ” operation.

3. HOISTING OPTIMIZATION

Due to the time-multiplexing of the memory address bus between row and column addresses, a scheduling optimization is possible when two addresses in the same memory page are accessed from different paths of a conditional branch.

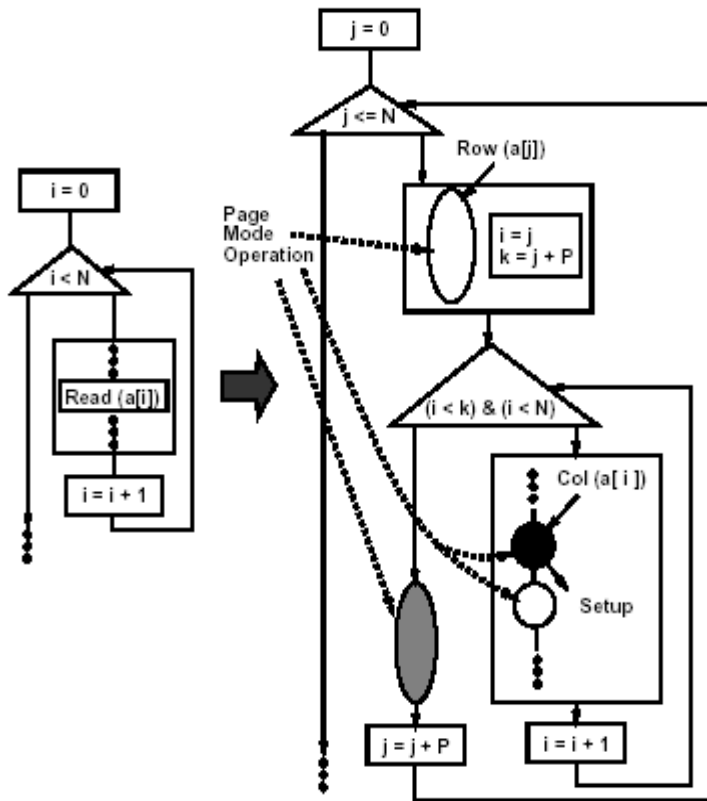


4. Loop Transformation

In order to utilize the page mode operations, the CDFG for the behavior has to be transformed to reflect the page mode operation. Since most of the computation occurs in the innermost loops of nested loop structures, concentration should be on the memory accesses in the innermost loops.

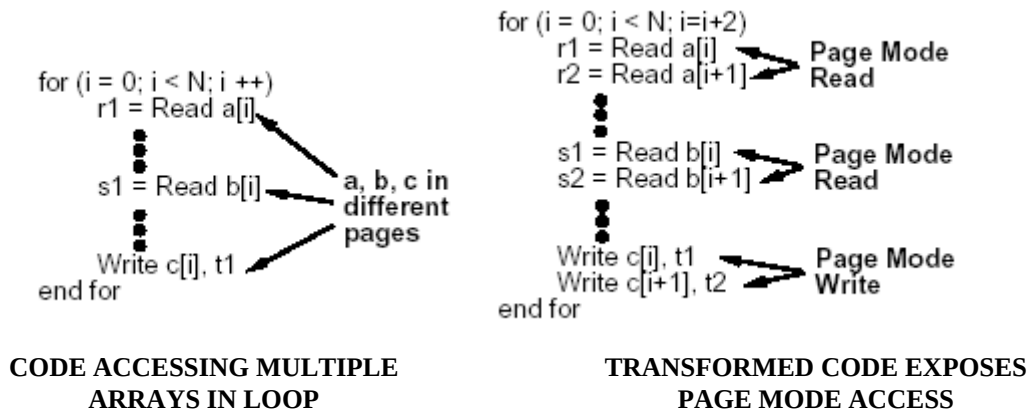
If a loop accesses locations from only a single memory page Per-iteration, e.g., there is only one read operation (*Read $a[i]$*) per iteration, the page mode operations can be applied

directly by restructuring the loop so that it iterates over the array in blocks of P iterations (where P is the page size, in words), so that we have one page mode read for every P iterations. Figure shows a section of the CDFG of an example loop with a single memory access ($a[i]$) in one iteration. We introduce an inner loop in which up to P elements from the same page are accessed. The transformed CDFG is shown in Figure . Note that the Row Decode and Pre charge nodes enclose the inner CDFG loop, forming one complete page mode operation.



CDFG OF ORIGINAL CODE

TRANSFORMED CDFG



POWER OPTIMIZATION IN MEMORY ACCESSES

POWER minimization efforts at the behavioral level usually attempt to reduce signal transition count, particularly off-chip transitions, owing to dynamic power dissipation accounting for a significant fraction of the total power dissipation in CMOS circuits. Power reduction in memory-intensive applications is done by analyzing the access patterns of behavioral arrays in the specification and organizing the arrays in memory so as to minimize transitions on the memory address buses. Examples of memory-intensive applications that exhibit regular access patterns exploitable by this technique are digital signal processing, and image and audio applications such as filters, relaxation algorithms, and compression algorithms such as discrete cosine transform (DCT) and inverse discrete cosine transform (IDCT). Given that off-chip capacitances are three orders of magnitude larger than typical on-chip capacitances, we can effect significant power savings by reducing the switched capacitance of the off-chip address bus drivers through reduction of transition activity on the memory address bus. Furthermore, reduced activity in the address bus also leads to reduced activity in the memory address buffers and decoding circuitry. Studies have shown that power dissipation in the address decoder and address buffers of typical memory chips constitute a significant portion of the power consumed (up to 50%) in the memory chip. Hence, design techniques leading to decrease in power dissipation in this part of the memory will significantly reduce the overall power dissipation of the application.

This technique takes as input the behavioral specification that is to be synthesized; the output is the assignment of arrays in the specification to physical addresses in memory. For each array, output is the expression for accessing an arbitrary memory element as a function of its behavioral array index and the array dimensions that corresponds to the best memory mapping strategy for the array.

We first assign the arrays in the specification to locations in a *logical memory*, and employ a Gray code converter (GCC) in the *address generator* to map logical memory addresses to physical memory locations. The GCC converts a logical address into *Gray code*, thereby ensuring that access of consecutive logical memory addresses results in then transition of exactly one bit on the memory address bus. The GCC helps bring down the address bus transition count to a minimum, but these memory mapping strategies are

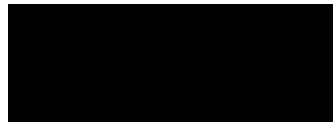
valid techniques for ensuring reduction in transition count even in the absence of the GCC from the architecture.

```
/* a,b,c : array [0..2] */
for i = 0 to 2
  c[i] = a[i] + b[i]
```

a[0]
a[1]
a[2]
b[0]
b[1]
b[2]
c[0]
c[1]
c[2]

a[0]
b[0]
c[0]
a[1]
b[1]
c[1]
a[2]
b[2]
c[2]

SAMPLE BEHAVIOUR



INTERLEAVED MAPPING

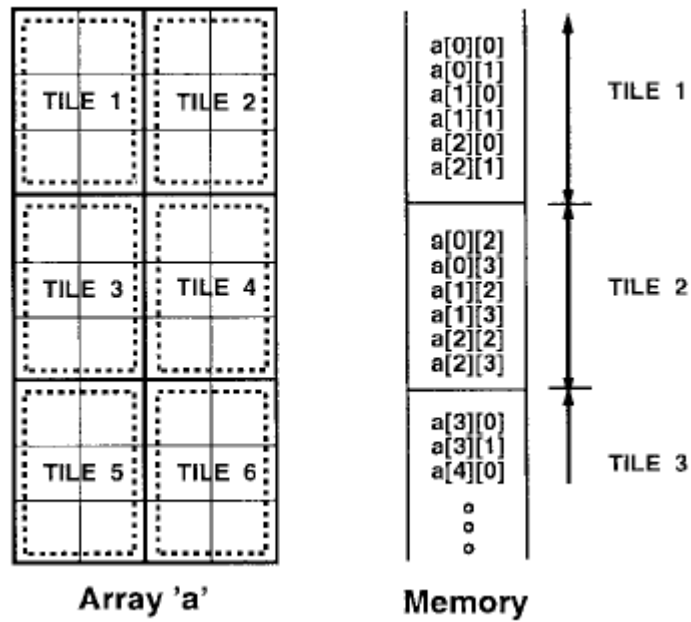
This approach can be shown by a sample behavior, shown in Fig. shows a sequential mapping of the three arrays a ,b , c . In this case, the sequence of logical memory locations accessed is: 0, 3, 6, 1, 4, 7, 2, 5, and 8. For the same example, as shown in c, an interleaved organization of arrays into logical memory, where the array elements have been stored in the order in which they are accessed in the loop. This mapping, when converted into Gray code, ensures the transition of exactly one bit on the memory address bus between consecutive accesses, resulting in a significant decrease in address bus transitions; for instance, the transition count decreases from 18 in fig b to 8 in fig c, assuming that both base addresses start at zero.

A. Array Mapping Schemes

This gives mapping strategies for two-dimensional arrays. For the effective mapping following three strategies has been applied for arrays into memory.

- 1) *Row Major*: A simple way of mapping a logical array to physical memory is to store the elements in row major form , i.e., the elements of the first row are placed in consecutive memory locations in order of increasing column index. This is followed by the elements of the second row in the same order, and so on.
- 2) *Column Major*: In column-major mapping , the elements of the two-dimensional array are stored column by column.
- 3) *Tile Based*: This mapping style first partitions the bigger array into tiles of smaller rectangles. Here, the elements of a row (or column) of tiles are stored in consecutive memory locations, with the elements of the tiles themselves being stored in either row or column major format . Similar ideas have been used in the context of compilers, where

cache reuse is improved by dividing loop iteration space into tiles and transforming the loop nesting structure to iterate over the tiles.



TILE MAPPING STRATEGY

Block decomposition of arrays in a multiprocessor environment is also based on a similar concept.

B. EXTRACTION OF ACCESS PATTERN

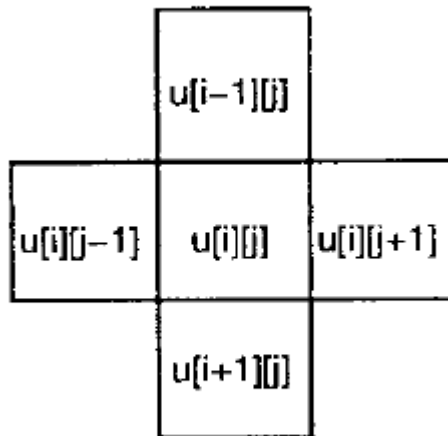
```

for (i = 1; i < N-1; i++)
    for (j = 1; j < N-1; j = j + 2) {
        res = a[i][j] * u[i+1][j] + b[i][j] * u[i-1][j] +
              c[i][j] * u[i][j+1] + d[i][j] * u[i][j-1] +
              e[i][j] * u[i][j] - f[i][j];
        u[i][j] = u[i][j] - (K * res) / e[i][j];
    }

```

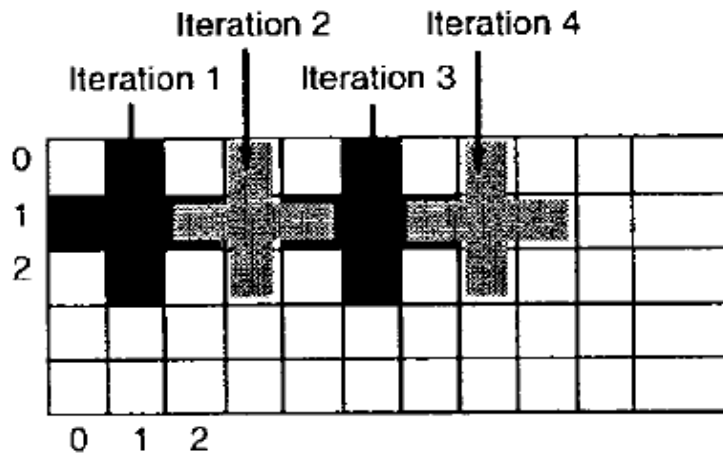
(a)

EXAMPLE



(b)

ACCESS PATTERN CONTOUR



INNER LOOP EXECUTION TRACE



NEW ELEMENT IN
EACH TRACE



TILE

Fig. A shows a simplified version of the code kernel for a successive over relaxation (SOR) algorithm , which is often used in the domain of image-processing applications. The plus-shaped contour in Fig. B shows the basic access pattern of the elements of array in the inner loop. Fig. C shows that this shape of accesses moves across by two columns to the right as we iterate through the inner loop. The second iteration of the outer loop causes the first pattern of the previous loop to move vertically down by one row. The remaining inner loop accesses are as before. The one element that is Common to successive patterns can be stored in a register instead of being accessed from memory again in the next iteration hence, the effective access pattern for this example is as shown in Fig. D. This pattern determines the dimensions of the tile to be used in the tile based mapping scheme.

C. Analysis of Mapping Schemes

This analysis will determine which mapping styles should be used. This analysis helps determine the mapping style that minimizes the total transition count. The observation that, in general, there is a low Hamming distance between elements in the same tile or adjacent tiles, and a possibly high Hamming distance between elements in distant tiles. Define a *maximal transition* as occurring when two logical addresses with a large difference are accessed in succession. A *minimal transition* occurs when this difference is small. In this case, *large* means comparable to the dimension of the array. For example, we treat all consecutive accesses to elements in the same tile as minimal transitions. **To conclude we can use following heuristic**

```
Let the inner loop index be  $i$  and outer loop index be  $j$ 
Extract the basic repeating shape from the access patterns
Determine the enclosing rectangle  $R$ .
Let  $i$ -dimension of  $R$  be  $L_i$  and  $j$ -dimension be  $L_j$ 
If increment of  $j$  in outer loop  $= L_i$  and  $j$ -dimension be  $L_j$ 
If increment of  $j$  in the outer loop  $= L_j$  or  $\min(L_i, L_j) > 2$ 
Then use tile based mapping
Else
    If  $L_j > L_i$ 
        Then use column-major mapping
    Else
        Use row-major mapping
    End if
End if
```

CHATER 3

ON-CHIP MEMORY

ORGANISATION:

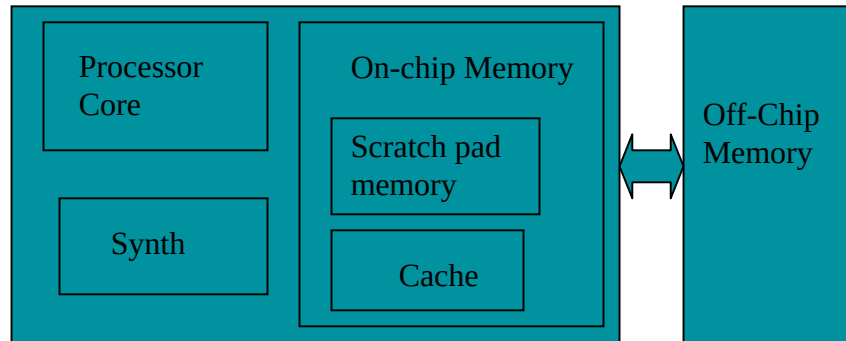


Figure shows a simplified view of an embedded processor core-based system, consisting of a processor core, on-chip memory, and synthesized block, interfacing with off-chip DRAM. The synthesized hardware block, often the result of behavioral synthesis, performs the functions specific to the application that are mapped to hardware, possibly for performance considerations. This decision of mapping different parts of a design into hardware and software is taken in a prior Hardware/Software partitioning step. Here, we assume that the partitioning has already been performed. The on-chip memory can be implemented as a combination of cache and Scratch-pad SRAM.

Data cache is fast, on-chip memory forming an interface between the processor and the off-chip DRAM, that reduces the effective memory access time by storing recently accessed data. Internally, the cache is divided into blocks, or cache lines, which constitute the smallest unit of interaction between the cache and the off-chip memory. Scratch-pad SRAM is on-chip memory, to which the assignment of data is compiler-controlled. In brief, a portion of the total data memory space is mapped to on-chip SRAM (typically used to store critical data), with the advantage of guaranteed fast access, unlike the cache, where hardware-controlled storage and replacement strategies could flush out data into the off-chip memory, resulting in cache misses that stall the processor.

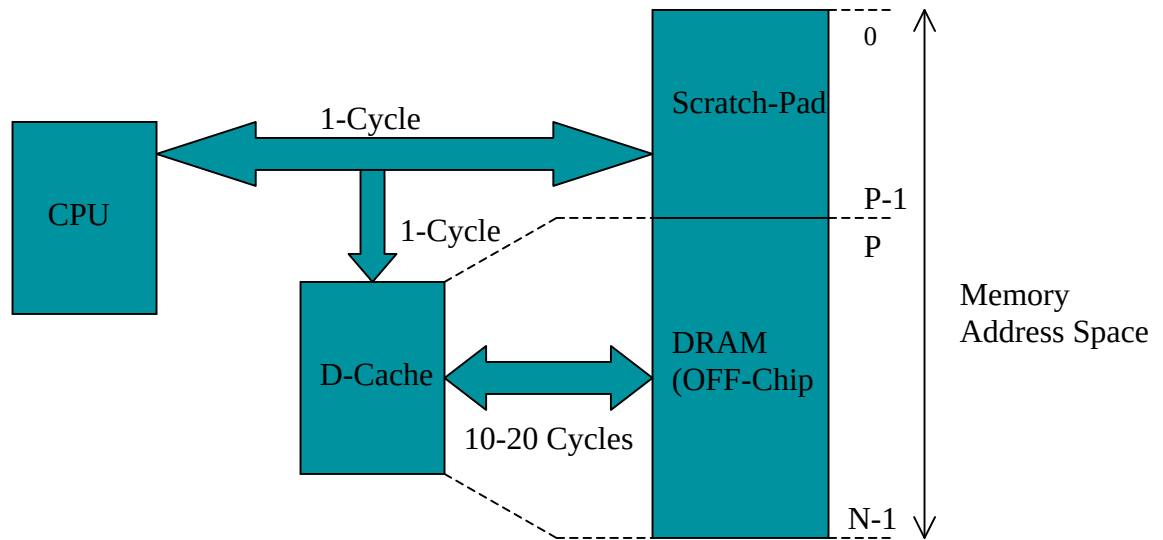
Efficient Utilization Of Scratch-Pad:

Efficient utilization of on-chip memory space is extremely important in modern embedded system applications based on microprocessor cores.

The on-chip SRAM, termed *Scratch-Pad memory*, is a small, high-speed data memory that is mapped into an address space disjoint from the off-chip memory, but connected to the same address and data buses. Both the cache

and Scratch-Pad SRAM have a single processor cycle access latency, whereas an access to the off-chip memory (usually DRAM) takes several (typically 10–20) processor cycles. The main difference between the Scratch-Pad SRAM and data cache is that the SRAM guarantees a single-cycle access time, whereas an access to cache is subject to compulsory, capacity, and conflict misses.

When an embedded application is compiled, the accessed data can now be stored either in the Scratch-Pad memory or in off-chip memory. In the second case, it is accessed by the processor through the data cache.



The data address space mapping is shown in Figure for a memory of size N data words. Memory addresses $0 \dots P-1$ are mapped into the Scratch-Pad memory, and have a single processor cycle access time. Thus, S HIT would be asserted whenever the processor attempts to access any address in the range $0 \dots P-1$. Memory addresses $P \dots N-1$ are mapped into the off-chip DRAM, and are accessed by the CPU through the data cache. A cache hit for an address in this range results in a single cycle delay, whereas a cache miss, which leads to a block transfer between off-chip and cache memory, results in a delay of 10-20 processor cycles.

The aim is to maximize the overall memory access performance by carefully mapping of each scalar and array variable. So, there is the problem of partitioning the variables between Scratch-Pad and off-chip DRAM. This is done by mapping those variables into SRAM that are estimated to cause the maximum number of conflicts in the data cache.

Features Affecting Partitioning:

Scalar Variables And Constants:

If scalars are assigned to DRAM, it is impossible to avoid the conflicts with arrays. That's why it is better to assign them to SRAM.

Size of arrays:

If the size of array is larger than size of SRAM, then that should be mapped into data-cache. Otherwise there will be only a part of array into SRAM. Since loops generally access array elements, there is no gain in mapping different parts of same array to memory with different characteristics.

Lifetime Of Variables:

The variables having different lifetimes can be stored in same memory location. So if two arrays or variable have conflicting lifetime, then cache miss can be avoided by just putting one of them into SRAM.

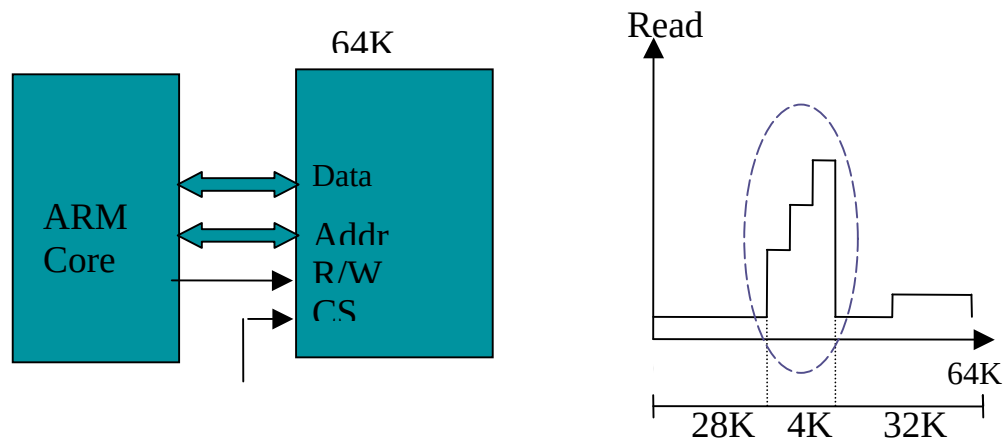
Access frequency of variables:

The access frequency of variables also affect the decision of partitioning data into SRAM and d-cache. If the frequency of access is less then it is preferable to put into DRAM. Otherwise it would lead to inefficient utilization of SRAM.

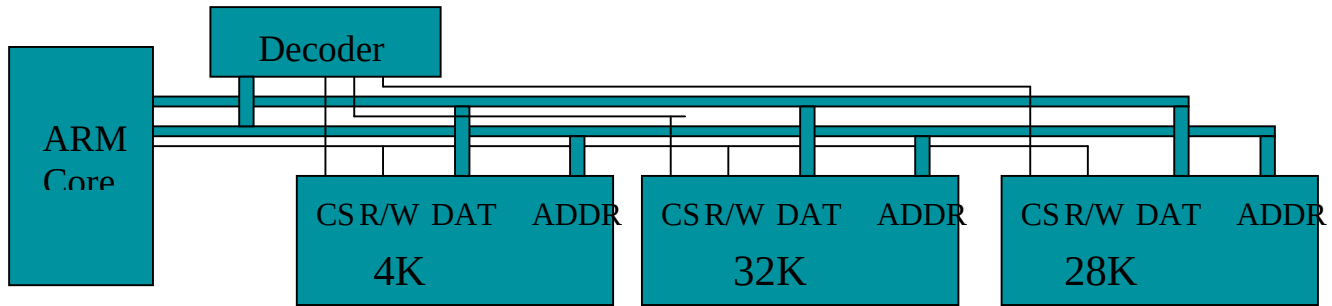
Power Optimization In On-chip SRAM:

Low power is really hot issue in today's system design. In the case of embedded systems, memory access patterns can typically be profiled at design time. This information can be used to reduce the power consumption into SRAM. One solution consists of mapping the most frequently accessed addresses onto the on-chip SRAM to guarantee power and performance efficiency.

The approach can be understood by a simple example. Suppose the fig shows a processor memory system and it's memory access profile. We see here most of the memory accesses take place between middle 4K range.



The best possible way to minimize the power in this case is shown in fig below.



Here we have separated the most frequently accessed 4K from rest of memory. Since the power dissipation is smaller in the smaller sized memory. The performance of the system greatly increases by dividing the memory into different banks.

For this following steps are taken:

1. Dynamic access Profile: This is obtained by a simulator, the output of which is the memory access pattern of different accesses.
2. Partitioning: The partitioning algorithm requires, which takes the dynamic access profile as input and gives the cuts into the memory for minimum power.
3. Decoder Generation: This is required to generate the control signals to select a particular memory bank. This takes as a input the address and gives the control signal and physical address as output.
4. Memory Generation: This is done by memory synthesis tool which takes different ranges of address as input and generates the memory block according to that.
5. Placement And Routing: Finally placement and routing of all the blocks is done. So the output is complete layout.

CONCLUSION:

The design of embedded systems –on-chip(SOC) is influenced by several evolutionary trends, such as the increase in design complexity made through shrinking feature sizes coupled with larger die sizes for chips. On the other hand, increasing complexity of building blocks leads to a revolutionary challenge in the emerging system design process that combines complex Intellectual-property(IP) library blocks to create specialized embedded SOC under tight time-to market deadline.

In this report we have included organization of modern DRAMS. We included models for representations of memory accesses, and transformation techniques for incorporating page mode ,and other efficient access features during synthesis. We then included a strategy for minimizing memory access related power by suitably mapping behavioral arrays to memory.

We then addressed architectural modifications of embedded SOC's memory subsystem. The application specific nature of embedded systems permit the incorporation of relatively unconventional components compared to general purpose processing domain. One such component is Scratch-Pad memory-on chip memory to which assignment of data is compiler controlled.

Finally we outlined a Layout –Driven memory synthesis for Embedded systems on-chip. In this we have explained a Strategy for partitioning of Scratch-Pad SRAM into multiple banks, so that overall energy consumption is minimized.

As the technology is advancing there is more possibility of memory optimization. Since now embedded DRAMs are coming, which can be fabricated with the same technology which is used for logic design, hence we can find more efficient modes for performance and power optimization techniques.

REFERENCES:

- Preeti Ranjan Panda, Nikhil D Dutt, Alexandru Nicolau, “ *Memory issues in embedded system on chip*” Kluwer Academic Publishers
- Dr. Preeti Ranjan Panda, “ *Memory optimization in embedded systems*”, Embedded system design workshop 2002, IIT Delhi
- Preeti Ranjan Panda, Nikhil D Dutt, Alexandru Nicolau, “*Architectural Exploration and optimization of local memory in embedded systems*”, IEEE 1997
- Preeti Ranjan Panda, Nikhil D Dutt, Alexandru Nicolau, “ *Exploiting off-chip memory access modes in high level synthesis*” IEEE 97
- Preeti Ranjan Panda, Nikhil D Dutt, Alexandru Nicolau, “*Memory data organization for improved cache performance in embedded systems*”, ACM transaction of design automation, Vol 2, No 4, Oct97
- Luca Benini, Luca Macchiarulo, Alberto Macii, Massimo Poncino “*Layout-driven memory synthesis for embedded system-on-chip*”, IEEE transaction on VLSI, VOL 10, No 21, April 2002