VLSI ARCHITECTURE OPTIMIZATION FOR REAL TIME DSP

ANINDYA SUNDAR DHAR

Basic VLSI Design Flow

- SYSTEM SPECIFICATION *(Algorithmic Level)*
- ARCHITECTURAL DESIGN *(Register Transfer Level)*
- LOGIC DESIGN *(Gate Level)*
- CIRCUIT DESIGN *(Transistor Level)*
- DEVICE DESIGN
- LAYOUT
  ....

Optimization is possible at every level

- ALGORITHMIC LEVEL
- ARCHITECTURAL LEVEL
- LOGIC LEVEL
- CIRCUIT LEVEL
- DEVICE LEVEL
  .....
OPTIMIZATION PROBLEM

PRIMARY OBJECTIVE FUNCTIONS

- SPEED
- AREA
- POWER

- The actual cost function involving these factors depends on specific application.
- Often they are considered to be constraint-satisfying problem rather than optimization problem.
- Design challenge is to have a better trade-off.

Implementation options:

- Application Specific Integrated Circuit (ASIC)
- DSP Microprocessor (DSP μP)
- Field Programmable Gate Array (FPGA)

ASIC vs. DSP Microprocessor

<table>
<thead>
<tr>
<th>OPTIMIZED DESIGN</th>
<th>PROGRAMMABILITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEED</td>
<td>(Unoptimized &amp; Underutilized)</td>
</tr>
<tr>
<td>POWER</td>
<td></td>
</tr>
<tr>
<td>AREA</td>
<td></td>
</tr>
</tbody>
</table>

Field Programmable Gate Array (FPGA) may strike a balance between Optimization and Flexibility.
**Basic Sequential Circuit Design Philosophy**

**OPTIMIZATION AT ARCHITECTURAL LEVEL**

```plaintext
input N
S = 0
for i = 1 to N
   S = S + i
next i
output S
```

**OPTIMIZATION AT LOGIC LEVEL**

```plaintext
input N
S = 0
for i = N downto 1
   S = S + i
next i
output S
```

20 Transistors

14 Transistors

2:1 MULTIPLEXER
OPTIMIZATION AT CIRCUIT LEVEL

2:1 MULTIPLEXER USING TRANSMISSION GATE LOGIC

6 Transistors (including 2 for inverting S)

Y = (AB + C)D

Optimized transistor level realization of Boolean function

18 Transistors

8 Transistors

OPTIMIZATION WITH RESPECT TO VARIOUS OBJECTIVE FUNCTIONS

- **SPEED@LOGIC LEVEL**
- **POWER@LOGIC LEVEL**
- **Processing of stored data**

  ![Diagram](image)

  *Memory (RAM) -> PROCESSOR*

  *(for non real time applications)*

- **Processing of continuous data stream**

  ![Diagram](image)

  *Use of ping-pong buffer*

  ![Diagram](image)

  *MEMORY (RAM 1) -> PROCESSOR -> MEMORY (RAM 2) (for real time applications)*

- **Processing of continuous data stream**

  ![Diagram](image)

  *On line processing*

  ![Diagram](image)

  *INPUT Data Acq. Unit -> PROCESSOR -> Memory (RAM) Intermediate data storage*
SPEED-POWER-AREA TRADE OFF

- Requirements are generally contradictory.

- However, *clever solutions* sometimes could satisfy contradictory requirements!
How to arrive at a clever solution

• Look for the variables which are not getting altered during the course of computation

• Identify the set of variables which are having mutually exclusive scopes

• Identify the elements which are remaining idle for a considerable period of time

• Visualize the operations in terms of binary number system

Variables not getting altered during the course of computation

Generally there exists some alternative way to handle that variable in the algorithm

It is not required to keep that variable stored for the total duration of computation

⇒ Reduced Area
⇒ Reduced Power

Set of variables having mutually exclusive scopes

It is possible to reduce the storage requirement by reusing the same storage elements for exclusive set of variables

⇒ Reduced Area
⇒ Reduced Power
Elements remaining idle for considerable period of time

It is generally possible to club the duties of several processors to have 100% utilization and thus redundant processors can be eliminated

⇒ Reduced Area
⇒ Reduced Power

Visualizing the operations in terms of binary number system

Sometimes it is possible to decompose the operations in terms of powers of two, which significantly simplifies the implementation

Examples:
• Multiplication or division by 2, 4, etc. (powers of 2)
• Multiplication or Division by a fixed number
• Usage of alternative number representation

⇒ Reduced Area
⇒ Reduced Power
⇒ Increased Speed

Examples and Case study

• Usage of redundant arithmetic
• Usage of alternative number representation (normalized / Gray coded)
• Usage of running transforms

• Design of an alternative arithmetic unit (e.g. CORDIC)
• Design of an FFT address generator
**REDUCING THE NUMBER OF TRANSITIONS IN THE ADDRESS BUS**

**Conventional**

15 = 01111

1 = 00001

16 = 10000

**Redundant**

15 = 10001

1 = 00001

16 = 10000 (Carry free)

**ADDITION in REDUNDANT NUMBER SYSTEM**

**ALTERNATIVE ARITHMETIC UNIT**

CO-ORDINATE ROTATION DIGITAL COMPUTER (CORDIC)

\[
x' = x \cos \theta + y \sin \theta \\
y' = -x \sin \theta + y \cos \theta
\]
/* Algorithm for address generation to compute N = 2^n point FFT */

For stage = n-1 downto 0
    ncluster = 2^(n-1 - stage)
    npair = 2^stage
    For cluster = ncluster-1 downto 0
        index = cluster*2^(stage+1)
        rot = 0
        For pair = npair-1 downto 0
            Call Butterfly(index, index+npair, rot)
            index = index+1
            rot = rot+ncluster
        Next pair
    Next cluster
Next stage
End

FFT BUTTERFLY & ADDRESS GENERATION
CONCLUSION

- Try to Believe that
  There *ALWAYS EXISTS A BETTER SOLUTION*
  than the present one and *WE’LL* be able to find it out!

- But
  There *ALWAYS EXISTS A BETTER SOLUTION*
  than what we can think of!