Chapter 5

Hardware/Software Partitioning and Performance Analysis

Professor Trong-Yen Lee

e-mail: tylee@ntut.edu.tw

http://www.ntut.edu.tw/~tylee

NTUT/EN
Outline

5.1 Hardware/Software Partitioning Methodology
5.2 The Principle of Profiling Tool
5.3 System Performance Measure and Analysis
5.4 Performance Analysis of Hardware Accelerator
5.5 Performance Prediction of Integration of Hardware and Software
5.1 Hardware/Software Partitioning Methodology

5.2 The Principle of Profiling Tool

5.3 System Performance Measure and Analysis

5.4 Performance Analysis of Hardware Accelerator

5.5 Performance Prediction of Integration of Hardware and Software
5.1 Hardware/Software Partitioning Methodology (1/6)

- **System Partitioning**
  - **System functionality** is implemented on system components.
    - ASICs, processors, memories, buses
  - **Two design tasks:**
    - Allocate system components or ASIC constraints
    - Partition functionality among components
  - **Constraints**
    - Cost, performance, size, power
  - Partitioning is a central system design task.

```plaintext
System functionality is implemented on system components.
- ASICs, processors, memories, buses
Two design tasks:
- Allocate system components or ASIC constraints
- Partition functionality among components
Constraints
- Cost, performance, size, power
Partitioning is a central system design task.
```
5.1 Hardware/Software Partitioning Methodology (2/6)

◆ Hardware/Software Partitioning

- **Informal Definition**
  - The process of deciding, for each subsystem, whether the required functionality is more advantageously implemented in hardware or software.

- **Goal**
  - To achieve a partition that will give us the required performance within the overall system requirements (in size, weight, power, cost, etc.).

- This is a multivariate optimization problem that when automated, is an **NP-hard problem**.
5.1 Hardware/Software Partitioning Methodology (3/6)

◆ **HW/SW partitioning formal definition**

- A hardware/software partition is defined using two sets $H$ and $S$, where $H \subseteq O$, $S \subseteq O$, $H \cup S = O$, $H \cap S = \emptyset$

- Associated metrics:
  - $Hsize(H)$ is the size of the hardware needed to implement the functions in $H$ (e.g., number of transistors).
  - $Performance(G)$ is the total execution time for the group of functions in $G$ for a given partition $\{H,S\}$.
  - Set of performance constraints, $Cons = (C_1, \ldots, C_m)$, where $C_j = \{G, timecon\}$, indicates the maximum execution time allowed for all the functions in group $G$ and $G \subseteq O$. 
5.1 Hardware/Software Partitioning Methodology (4/6)

◆ Structural vs. Functional partitioning

■ Structural: Implement structure, then partition
  ● Good for the hardware (size & pin) estimation.
  ● Size/performance tradeoffs are difficult.
  ● Suffer for large possible number of objects.
  ● Difficult for HW/SW tradeoff.

■ Functional: Partition function, then implement
  ● Enables better size/performance tradeoffs.
  ● Uses fewer objects, better for algorithms/humans.
  ● Permits hardware/software solutions.
  ● But, it’s harder than graph partitioning.
5.1 Hardware/Software Partitioning Methodology (5/6)

◆ Partitioning Approaches

- **Start** with all functionality in software and move portions into hardware which are time-critical and can not be allocated to software (software-oriented partitioning).

- **Start** with all functionality in hardware and move portions into software implementation (hardware-oriented partitioning).
5.1 Hardware/Software Partitioning Methodology (6/6)

◆ System partitioning (Functional partitioning)
  ■ System partitioning in the context of hardware/software codesign is also referred to as functional partitioning.
  ■ Partitioning functional objects among system components is done as follows.
    ● The system’s functionality is described as collection of indivisible functional objects.
    ● Each system component’s functionality is implemented in either hardware or software.
  ■ An important advantage of functional partitioning is that it allows hardware/software solutions.
5.1 Basic Partitioning Issues (1/3)

- Specification-abstraction level: input definition
  - Executable languages becoming a requirement.
    - Although natural languages common in practice.
  - Just indicating the language is insufficient.
  - Abstraction-level indicates amount of design already done.
    - e.g. task DFG, tasks, CDFG, FSMD

- Granularity: specification size in each object
  - Fine granularity yields more possible designs
  - Coarse granularity better for computation, designer interaction
    - e.g. tasks, procedures, statement blocks, statements

- Component allocation: types and numbers
  - e.g. ASICs, processors, memories, buses
5.1 Basic Partitioning Issues (2/3)

- **Metrics and estimations**: "good" partition attributes
  - e.g. cost, speed, power, size, pins, testability, reliability
  - Estimates derived from quick, rough implementation
  - Speed and accuracy are competing goals of estimation.

- **Objective and closeness functions**
  - Combines multiple metric values
  - Closeness used for grouping before complete partition
  - Weighted sum common
  - e.g. \( k_1 F(\text{area}, c) + k_2 F(\text{delay}, c) + k_3 F(\text{power}, c) \)

- **Output: format and uses**
  - e.g. new specification, hints to synthesis tool

- **Flow of control and designer interaction**
5.1 Basic Partitioning Issues (3/3)

- High Level Abstraction

- Decomposition of functional objects
  - Metrics and estimations
  - Partitioning algorithms
  - Objective and closeness functions

- Component allocation

- Output
5.2 The Principle of Profiling Tool

5.1 Hardware/Software Partitioning Methodology

5.2 The Principle of Profiling Tool

5.3 System Performance Measure and Analysis

5.4 Performance Analysis of Hardware Accelerator

5.5 Performance Prediction of Integration of Hardware and Software
5.2 The Principle of Profiling Tool

- Profiling is to **check** a program that those are **called out** most frequently or **executed time** longest.
- Profiling is good way to **find out** optimization and loss time.
- Profiling must be non-intrusive for the analyzed of the SW code.
  - This means it should not disturb or influence SW execution.
- Profiling enables the designer to iteratively **detect critical parts** of the functional SW code and to **estimate the exact performance** of the optimized design.
5.2 The Principle of Profiling Tool

◆ In general, profiling approach covers two design aspects important for efficient partitioning.
  - Performance analysis
  - Performance estimation

◆ SW oriented profiling option is based on emulation or simulation of processor systems on independent host platforms before the physical implementation.
  - The benefit is that the SW code and its execution are not altered.

◆ HW oriented profiling approach is the use of a logic analyzer for instruction/data bus monitoring.
  - The drawbacks are high cost, ineffectiveness and limited storage capacity of logic analyzers.
5.2 The Principle of Profiling Tool

- Some profiling systems also take advantage of additional special on-chip HW for profiling support.
  - e.g. Intel VTune

- Profilers with additional HW offer better accuracy when it comes to time-related performance measurements and CPU execution-related parameters.

- Obtaining an efficient performance analysis and estimation of potential partitioning solutions with a clock-cycle accuracy is required.

- Profiling approach covers two design domains.
  - Simulation domain
  - Implementation domain
5.2 The Principle of Profiling Tool

- Simulation domain

  - The **advantage** is that does not require any physical HW platform for implementation of the analyzed processor system.

  - The performance analysis and estimation are performed with the use of the related profiler on the basis of the simulation output of a single simulation cycle.

  - For each critical part of the functional SW code, the designer can redefine timing execution parameters and estimate the system performance **without actually implementing** any additional HW components for co-processing.
5.2 The Principle of Profiling Tool

- Simulation domain (cont’d)
  - A clock-cycle accurate insight into the performance improvement can be obtained.
  - Different partitioning possibilities can be experimented with without the need to re-simulate the design.

- Implementation domain
  - Partitioning in this domain can be performed by implementing a basic targeted processor system configuration on any adequate development board capable of executing the SW code or its functional parts.
### 5.2 The Principle of Profiling Tool

#### Comparison of several profiling systems.

<table>
<thead>
<tr>
<th>System</th>
<th>Overhead</th>
<th>Scope</th>
<th>Grain</th>
<th>Stalls</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixie</td>
<td>high</td>
<td>app</td>
<td>inst count</td>
<td>none</td>
</tr>
<tr>
<td>gprof</td>
<td>high</td>
<td>app</td>
<td>proc count</td>
<td>none</td>
</tr>
<tr>
<td>jprof</td>
<td>high</td>
<td>app</td>
<td>proc count</td>
<td>none</td>
</tr>
<tr>
<td>quartz</td>
<td>high</td>
<td>app</td>
<td>proc count</td>
<td>none</td>
</tr>
<tr>
<td>MTOOL</td>
<td>high</td>
<td>app</td>
<td>inst count/time</td>
<td>inaccurate</td>
</tr>
<tr>
<td>SimOS</td>
<td>high</td>
<td>sys</td>
<td>inst time</td>
<td>accurate</td>
</tr>
<tr>
<td>SpeedShop (pixie)</td>
<td>high</td>
<td>app</td>
<td>inst count</td>
<td>none</td>
</tr>
<tr>
<td>VTune (dynamic)</td>
<td>high</td>
<td>app</td>
<td>inst time</td>
<td>accurate</td>
</tr>
<tr>
<td>prof</td>
<td>low</td>
<td>app</td>
<td>inst time</td>
<td>none</td>
</tr>
<tr>
<td>iprobe</td>
<td>high</td>
<td>sys</td>
<td>inst time</td>
<td>inaccurate</td>
</tr>
<tr>
<td>Morph</td>
<td>low</td>
<td>sys</td>
<td>inst time</td>
<td>none</td>
</tr>
<tr>
<td>VTune (sampler)</td>
<td>low</td>
<td>sys</td>
<td>inst time</td>
<td>inaccurate</td>
</tr>
<tr>
<td>SpeedShop (timer and counters)</td>
<td>low</td>
<td>sys</td>
<td>inst time</td>
<td>inaccurate</td>
</tr>
<tr>
<td>DCPI</td>
<td>low</td>
<td>sys</td>
<td>inst time</td>
<td>accurate</td>
</tr>
</tbody>
</table>
5.2 The Principle of Profiling Tool

◆ The column of overhead describes how much profiling slows down the target program.

◆ The “scope” column shows whether the profiling system is restricted to a single application (app) or can measure full system activity (sys).

◆ The “grain” column indicates the range over which an individual measurement applies.

◆ The “stalls” column indicates whether and how well the system can subdivide the time spent at an instruction into components like cache miss latency, branch misprediction delays, etc.
5.3 System Performance Measurement and Analysis

5.1 Hardware/Software Partitioning Methodology
5.2 The Principle of Profiling Tool
5.3 System Performance Measure and Analysis
5.4 Performance Analysis of Hardware Accelerator
5.5 Performance Prediction of Integration of Hardware and Software
5.3 System Performance Measurement and Analysis

- Why profiling?
  - **Adjust** the program to most efficiency and performance.
  - Function optimization
  - A suit of software to analyse the efficiency of the program.

- GNU gprof
  - This manual describes the GNU profiler
  - Show efficiency test information (profiling) of program.
  - Must add “–pg” before program is complied.
  - After the executive program, will produce gmon.out.
    - gmon.out include the result of efficiency test.
  - The file of gmon.out must use gprof instruction to read.
5.3 System Performance Measurement and Analysis – Profiling Flow

1. Modify makefile and add efficiency test instruction
2. Re-make
3. Execute program
4. Read efficiency test file
5. Observation test efficiency

Add -pg parameter after cc

Produce mpeg2encode executable

Produce gmon.out after execute mpeg2encode

gprof mpeg2encode > prof

Vi prof
5.3 System Performance Measurement and Analysis – Addition Efficiency Test Instruction

- File path: mpeg2/src/mpeg2enc/Makefile
  - #
  - # GNU gcc
  - #
  - CC = gcc
  - CFLAGS = -O -pg
  - OBJ = mpeg2enc.o conform.o putseq.o putpic.o ......
  - all: mpeg2encode
  - clean:
    - rm -f *.o *% core mpeg2encode
    - mpeg2encode: $(OBJ)
    - $(CC) $(CFLAGS) -o mpeg2encode $(OBJ) -lm
5.3 System Performance Measurement and Analysis - Re-Make

◆ Make -B: Compile by force

[Image of a terminal window with a code snippet]

```
make -B
```

```
gcc -pg -O2 -c -o mpeg2enc.o mpeg2enc.c
gcc -pg -O2 -c -o conform.o conform.c
gcc -pg -O2 -c -o putseq.o putseq.c
gcc -pg -O2 -c -o putpic.o putpic.c
gcc -pg -O2 -c -o puthdr.o puthdr.c
gcc -pg -O2 -c -o putmpg.o putmpg.c
gcc -pg -O2 -c -o putvlc.o putvlc.c
gcc -pg -O2 -c -o putbits.o putbits.c
gcc -pg -O2 -c -o motion.o motion.c
gcc -pg -O2 -c -o predict.o predict.c
gcc -pg -O2 -c -o readpic.o readpic.c
gcc -pg -O2 -c -o writepic.o writepic.c
gcc -pg -O2 -c -o transform.o transform.c
gcc -pg -O2 -c -o fdctref.o fdctref.c
gcc -pg -O2 -c -o idct.o idct.c
gcc -pg -O2 -c -o quantize.o quantize.c
gcc -pg -O2 -c -o ratec1.o ratec1.c
gcc -pg -O2 -c -o stats.o stats.c
gcc -pg -O2 -c -o mpeg2encode.o conform.o putseq.o putpic.o puthdr.o putmpg.o putvlc.o putbits.o motion.o predict.o readpic.o writepic.o transform.o fdctref.o idct.o quantize.o ratec1.o stats.o -lm
```

[root@localhost mpeg2enc]
```
5.3 System Performance Measurement and Analysis – Execution Program

- ./mpeg2encode MPEG-1.par out

MPEG-1.par is parameter setup file of MPEG-2.
5.3 System Performance Measurement and Analysis – Read Efficiency Test File

- gprof mpeg2encode > prof
  - The prof name can be named wantonly.
## 5.3 System Performance Measurement and Analysis - Observation Test Efficiency

- Vi prof

### Flat profile:

Each sample counts as 0.01 seconds.

<table>
<thead>
<tr>
<th>% cumulative</th>
<th>self seconds</th>
<th>calls</th>
<th>self s/call</th>
<th>total s/call</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>seconds</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31.90</td>
<td>3.85</td>
<td>3.85</td>
<td>35436288</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>25.35</td>
<td>6.91</td>
<td>3.06</td>
<td>712800</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>14.58</td>
<td>8.67</td>
<td>1.76</td>
<td>651024</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>4.97</td>
<td>9.27</td>
<td>0.60</td>
<td>187308</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>3.07</td>
<td>9.64</td>
<td>0.37</td>
<td>712800</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>2.98</td>
<td>10.00</td>
<td>0.36</td>
<td>78804</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>2.90</td>
<td>10.35</td>
<td>0.35</td>
<td>900</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>2.49</td>
<td>10.65</td>
<td>0.30</td>
<td>217008</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>2.07</td>
<td>10.90</td>
<td>0.25</td>
<td>300</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>1.82</td>
<td>11.12</td>
<td>0.22</td>
<td>138963</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>1.41</td>
<td>11.29</td>
<td>0.17</td>
<td>300</td>
<td>0.00</td>
<td>0.01</td>
</tr>
<tr>
<td>1.33</td>
<td>11.45</td>
<td>0.16</td>
<td>61776</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>1.08</td>
<td>11.58</td>
<td>0.13</td>
<td>475200</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.99</td>
<td>11.70</td>
<td>0.12</td>
<td>651024</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>
5.3 System Performance Measurement and Analysis – Observation Test Efficiency

- Cumulative seconds
  - Accumulation time
- Self seconds
  - Function execution time
- Calls
  - Number of times that function is called out.
5.4 Performance Analysis of Hardware Accelerator

5.1 Hardware/Software Partitioning Methodology
5.2 The Principle of Profiling Tool
5.3 System Performance Measure and Analysis
5.4 Performance Analysis of Hardware Accelerator
5.5 Performance Prediction of Integration of Hardware and Software
5.4 Performance Analysis of Hardware Accelerator

- Impact of hardware acceleration on system software
- Inverse DCT
- Row-column decomposition of 2D-IDCT
- Matrix factorization of 1D-IDCT
- Algorithm optimization for programmable logic
- Implementation details
  - 2D-IDCT slice block diagram
  - Data path widths for main paths in the IDCT design
- Minimum throughput requirements
- Transpose RAM
- Hardware IDCT performance
5.4.1 Impact of Hardware Acceleration on System Software

- Distribution of MPEG-2 decoder processing tasks for non-accelerated and accelerated scenarios.

![Diagram showing distribution of tasks for non-accelerated and accelerated scenarios.](image)
5.4.2 Inverse DCT

- The **inverse DCT** function used in the MPEG-2 specification is a 2-dimensional IDCT (2D-IDCT), where an 8x8 set of frequency domain coefficients is transformed into an 8x8 block of pixels.

- The IDCT process for an **8x8 block** of coefficients is defined by the following equation:

  \[
  P[x, y] = \frac{1}{4} \sum_{u=0}^{7} \sum_{v=0}^{7} C[u] C[v] F[u, v] \cos \left( \frac{(2x+1)u\pi}{16} \right) \cos \left( \frac{(2y+1)v\pi}{16} \right)
  \]

  where \( C[k] = \frac{1}{\sqrt{2}} \) for \( k = 0 \), \( C[k] = 1 \) otherwise.

- A direct implementation of this equation requires an excessive amount of operations.
5.4.3 Row-Column Decomposition of 2D-IDCT

- The number of computations is easily reduced by performing 1D-IDCT on each row, then performing it again on each column as illustrated in the following figure.

- An 8-point 1D-IDCT is given by the following equation:

  \[ P[m] = \frac{1}{2} \sum_{n=0}^{7} C[n]F[n] \cos \frac{(2m+1)n\pi}{16} \]

- A direct implementation of this 1D-IDCT equation requires 64 multiplications and 56 additions per row of eight elements.

- Implementing the 2D-IDCT using this approach would require 1,024 multiplications and 896 additions for the whole 8x8 block.
5.4.4 Matrix Factorization of 1D-IDCT

Employing matrix factorization similar to that employed in Chen's algorithm, the 8x8 matrix multiplication is reduced to the sum and differences of two 4x4 matrix-vector multiplications given as figure (b).

$$
\begin{align*}
\begin{bmatrix}
P[0] \\
P[1] \\
P[2] \\
P[3] \\
P[4] \\
P[5] \\
P[6] \\
P[7]
\end{bmatrix}
\begin{bmatrix}
A & B & C & D & E & F & G & H \\
A & D & G & \_H & \_E & \_B & \_C & \_F \\
A & F & \_G & \_B & \_E & H & C & D \\
A & H & \_C & \_F & E & D & \_G & \_B \\
A & \_H & \_C & \_F & E & \_D & \_G & B \\
A & \_F & \_G & \_B & \_E & H & C & \_D \\
A & \_D & G & H & \_E & B & \_C & F \\
A & \_B & C & \_D & E & \_F & G & \_H \\
\end{bmatrix}
= 
\begin{bmatrix}
F[0] \\
F[1] \\
F[2] \\
F[3] \\
F[4] \\
F[5] \\
F[6] \\
F[7]
\end{bmatrix}
\begin{bmatrix}
A & C & E & G \\
A & G & \_E & \_C \\
A & \_G & \_E & C \\
A & \_C & \_E & \_G \\
B & D & F & H \\
D & \_H & \_B & \_F \\
F[2] \\
F[3]
\end{bmatrix}
+ 
\begin{bmatrix}
B & D & F & H \\
D & \_H & \_B & \_F \\
F[4] \\
F[5] \\
H & \_F & D & \_B \\
F[7]
\end{bmatrix}
\end{align*}
$$

Figure (a). Non-factorized matrix

<table>
<thead>
<tr>
<th>1D IDCT</th>
<th>multiplications</th>
<th>additions</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>56</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2D IDCT</th>
<th>multiplications</th>
<th>additions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>896</td>
<td></td>
</tr>
</tbody>
</table>

Figure (b). Factorized matrix

<table>
<thead>
<tr>
<th>1D IDCT</th>
<th>multiplications</th>
<th>additions</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>32</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2D IDCT</th>
<th>multiplications</th>
<th>additions</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>512</td>
<td></td>
</tr>
</tbody>
</table>
5.4.5 Algorithm Optimization for Programmable Logic

- The factorized matrix version of the IDCT is a reasonably optimal algorithm for programmable logic implementation.
- An algorithm that favors addition over multiplication would be **considered** more optimal than one that has more multiplication because a multiplier requires more standard cell logic than an adder.
- In the QuickMIPS programmable logic, ECU blocks are provided where the cost of a multiply is the same as addition.
- With the ECUs in the programmable logic, a multiplier and an adder are packed into each ECU unit.
5.4.5 Algorithm Optimization for Programmable Logic

◆ The goal for choosing or optimizing algorithm shifts away from minimizing the number of multiplies towards minimizing the total number of operations.

◆ The goal is to transform the algorithm into iterations of regular/uniform operations, i.e., having the multiplier/adder functions originate from a contiguous data source so that an operation can be executed every clock cycle.

◆ Iteration of regular/uniform operations simplifies control logic and data flow design, resulting in higher speed with fewer logic cells.
5.4.6 2D-IDCT Slice Block Diagram

- A simple block diagram of a single slice of the hardware implementation using two multipliers is shown in the flow.

- To achieve the desired throughput, two sets of multipliers and adders are used.

- Consequently, data can be processed as odd/even pairs. This allows 512 multiply/adds to be performed in half the time it would take otherwise.
5.4.6 Data Path Widths for Main Paths in the IDCT Design

The word lengths in the following table are used in the design to satisfy precision requirements.

<table>
<thead>
<tr>
<th>Path</th>
<th>Word Length (Bits)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Samples (Row)</td>
<td>12</td>
<td>Input samples from inverse quantizer block—This is internally shifted up by four to get 16-bit samples, minimizing multiplexing in the output word shifting.</td>
</tr>
<tr>
<td>Coefficients</td>
<td>16</td>
<td>Although 13-bit coefficients would be sufficient, additional 3 bits were included for improved precision because the ECU's multipliers works on an 8-bit block.</td>
</tr>
<tr>
<td>Accumulators</td>
<td>19</td>
<td>Only 16 of these 19 bits will continue to the next level.</td>
</tr>
<tr>
<td>Input Samples (Col)</td>
<td>16</td>
<td>This is the output from the row 1D iDCT process.</td>
</tr>
<tr>
<td>Output</td>
<td>9</td>
<td>$-256 \leq x \leq 255$</td>
</tr>
</tbody>
</table>
5.4.7 Minimum Throughput Requirements

- For a DVD formatted MPEG-2 stream (720x480 @ 30 fps) the minimum throughput requirement that the 2D-IDCT should meet is calculated as follows:

\[
\frac{720 \times 480 \text{ pixels}}{\text{frame}} \times \frac{30 \text{ frames}}{\text{sec}} \times \frac{\text{block}}{64 \text{ pixels}} \times 1.5 = 243,000 \text{blocks/sec}
\]

- As was discussed in “Matrix Factorization of 1D-IDCT” on page 37, the 2D-IDCT process for a block of 8x8 pixels require 512 multiplications, so the above throughput can be expressed in terms of number of multiplications as follows:

\[
\frac{243,000 \text{blocks/sec}}{\text{sec}} \times \frac{512 \text{multiplications}}{\text{block}} = 124.4 \text{Mmultiplications/sec}
\]

- Based on these estimates, the 2D-IDCT block has to be able to do at least two multiplications per cycle to set its frequency target to a reasonable speed.

- At two multiplications/cycle, the minimum frequency the block needs to run, without considering initial latencies and overheads, is 62.2 MHz.
5.4.8 Transpose RAM

- On the *first pass*, IDCT operates on the incoming 8x8 block of pixels on a per row basis and stores the intermediate results into the Transpose RAM.

- In the *second pass*, the IDCT data path works on these results on a per column basis and generates the final output block, as illustrated in the right figure.
5.4.8 Transpose RAM

- To simplify the control logic and data path to/from the IDCT circuit, an alternate mapping of the Transpose RAM was used in the implementation, as shown in the right figure.

- As in the previous illustration, the first IDCT works on the 8x8 block on a per row basis and stores the intermediate results into the Transpose RAM, but writes in a per column basis.

- The second IDCT works on the transposed block also in a per row basis and stores the final results into the output FIFO, also writing in a per column basis.
5.4.9 Active Portion of the IDCT Block During Row Processing

◆ Row Processing
5.4.9 Active Portion of the IDCT Block

During Column Processing

柱状处理

- 输入RAM偶数（4-32x16）
  - 系数
  - 从iQ/iZZ块
- 输入RAM偶数（4-32x16）
  - 乘法
  - 右移19位
  - 加法
  - 右移9位
  - 除法
  - 15位
  - 转置RAM奇数样本（32x16）
- 转置RAM偶数样本（32x16）
- 输出RAM偶数（8-32x9）
  - 右移9位
  - 转置RAM偶数样本（8-32x9）
  - 输出
    - 送至时间混合器

- 输出RAM偶数（32x16）
  - 右移9位
  - 转置RAM偶数样本（32x16）
- 输出RAM偶数（8-32x9）
  - 右移9位
  - 转置RAM偶数样本（8-32x9）
  - 输出
    - 送至时间混合器
5.4.9 Hardware IDCT Performance

◆ If the incoming FIFO is kept filled, the 2D-IDCT design would be able to process 8x8 block of information every 256 cycles.

◆ A 10-cycle penalty is incurred every time the incoming FIFO becomes empty.
5.5 Performance Prediction of Integration of Hardware and Software

5.1 Hardware/Software Partitioning Methodology
5.2 The Principle of Profiling Tool
5.3 System Performance Measure and Analysis
5.4 Performance Analysis of Hardware Accelerator
5.5 Performance Prediction of Integration of Hardware and Software
5.5 Performance Prediction of Integration of Hardware and Software

- The concepts of hardware acceleration are applied to the acceleration of an MPEG-2 decoder.
- The topic is discussed in relation to hardware and software implementation.
5.5 Performance Prediction of Integration of Hardware and Software

◆ Decoding MPEG-2 Overview
◆ Development Environment
◆ Software Implementation of an MPEG-2 Decoder
  ■ Performance Profile of the Software MPEG-2 Decoder
◆ Implementing an MPEG-2 Decoder in QuickMIPS
◆ Results
  ■ Device Performance and Logic Utilization
  ■ Implementation Constraint
  ■ Mapping Constraint
  ■ Benchmark Performance
  ■ Power Measurement
◆ Conclusion
5.5.1 Decoding MPEG-2 Overview

- MPEG-2 Systems
- MPEG-2 Video
- MPEG-2 Audio
- MPEG-2 Software
- MPEG-2 10-bit Video
- MPEG-2 Real Time Interface
5.5.1 MPEG-2 System
5.5.1 MPEG-2 Program Stream

- The MPEG-2 systems define two types of streams: the **program stream** and the **transport stream**.
- The program stream is similar to the MPEG-1 system stream, but uses a modified syntax and new functions to **support advanced functionalities**.
- Program streams decoders typically employ long and variable-length packets.
- The program stream is **computable with MPEG-1**.
- Suitable for **disk storage**.
The second type of stream supported by MPEG-2 system is the transport stream, which differs significantly from MPEG-1 systems as well as the program stream.

The transport stream offers robustness necessary for noisy channels as well as the ability to include multiple programs in a single stream.

Suitable for communication.

A basic data structure that is common to the organization of both the program stream and transport stream data is called the Packetized Elementary Stream (PES) packet.
5.5.1 MPEG-2 Video

- MPEG-2 Non-Scalable Video Codec
5.5.1 MPEG-2 Audio (1/4)

- MPEG-2 Scalable Video Codec
5.5.1 MPEG-2 Audio (2/4)

- The MPEG-2 Audio standard consists of two parts, one that allows coding of multi-channel audio signals in a forward and backward compatible manner with MPEG-1.

- Forward compatibility means that the MPEG-2 multi-channel audio decoder can decode MPEG-1 mono of stereo audio signals.

- Backward compatibility (BC) means that an MPEG-1 stereo decoder can reproduce a meaningful down mix of the original five channels from the MPEG-2 audio bit stream.
5.5.1 MPEG-2 Audio (3/4)

◆ Multi-channel Audio CODEC
5.5.1 MPEG-2 Audio (4/4)

- A Multi-channel Audio consisting of five signals, left (L), and right surround (Rs).

- To verify the syntax two type of tests were conducted:
  - First bit streams produced by software encoding were decoded in non-real-time by software decoders, and second, bit streams produced by software encoders were decoded in real-time.
5.5.1 MPEG-2 Software

- The MPEG-2 Systems Software consists of software for encoding and decoding of both the Program and the Transport Streams.
- The MPEG-2 Video Software mainly consists of software for encoding and decoding of non-scalable video.
- In addition, it also includes some software for scalable encoding and decoding.
5.5.1 MPEG-2 Real Time Interface

- The RTI is an optional supplement to the System’s Transport Stream System Target Decoder (STD) model.
- Configuration for illustration Real-Time Interface (RTI)
Both unaccelerated (software-only) and hardware accelerated versions of the MPEG-2 decoder were developed for this section.

The unaccelerated performance results are used as a baseline to compare with the performance results of the hardware accelerated version.
5.5.2 Development Environment (2/2)

- In both cases, the QuickMIPS Development Platform (QDP) was used.
- 902QDP Board used for hardware and software development.
To minimize the development time of the MPEG-2 decoder, an open source software decoder is used as a starting point.

The first criteria is that the software has to be command line driven so that when profiled, it does not contain user interface code that would influence the profile results.

User interface code could skew the results so that it would not be possible to accurately determine what routines needed to be accelerated in hardware.
5.5.3 Software Implementation of an MPEG-2 Decoder (2/4)

- **The second** criteria for MPEG-2 software selection is that the software decoder does not attempt to synchronize the video stream by dropping frames.
  - The dropping of frames can also skew the results obtained when trying to determine the amount of acceleration achieved by the use of the hardware acceleration.

- **The software** from the libmpeg2.sourceforge.net website meets the criteria detailed above and is used in the development of this section.

- **The software** is in the form of a reusable library and comes with several sample applications, including a full MPEG video decoder, called `mpeg2dec`, that can play MPEG files and DVD VOB files.
5.5.3 Software Implementation of an MPEG-2 Decoder (3/4)

◆ An additional benefit of the `libmpeg2` decoder is that it provides several ways to handle the display of frames so that the effects of displaying the results can be negated from the performance measurements.

◆ Normally, the MPEG-2 decoder creates YCrCb planes that must be converted to RGB for display purposes by converting the color space.

◆ In this section, color space conversion is not implemented in hardware.

◆ Consequently, the process of color space conversion and display needs to be turned off for the profiling.
5.5.3 Software Implementation of an MPEG-2 Decoder (4/4)

- The libmpeg2 software provides the capability of frame per second (FPS) data for the cases of:
  - Decode to YCrCb, no display
  - Decode to RGB, no display
  - Decode to RGB and display

- For the purposes of this section, the first case (decode to YCrCb, no display) is used as the baseline for software-only performance.

- The ratio of frame rates between the software only and the software/hardware hybrid is used to determine the acceleration factor.
The figure in the next page shows a histogram of the DVD reference stream for the top ranked functions up to the 99% level.

The results show that over 87% of CPU time is occupied by the two operations.

- IDCT and Motion Compensation
5.5.3 Performance Profile of the Software MPEG-2 Decoder (2/4)

- Histogram of MPEG-2 Software Decoder for the DVD Reference Stream
5.5.3 Performance Profile of the Software MPEG-2 Decoder (3/4)

- MPEG-2 decoder functions
  - Grouping the functions according to their role in the MPEG-2 decoder.
    - Parse — where most of the upper layers of MPEG stream are decoded
    - VLD/iQ/iZZ — Variable Length Decode, inverse Quantization, and inverse Zig-Zag
    - IDCT — inverse DCT
    - MC — Motion Compensation
5.5.3 Performance Profile of the Software MPEG-2 Decoder (4/4)

- Histogram of MPEG-2 Software Decoder for the DVD Reference Stream With Functions Grouped

- It is clear the IDCT and Motion Compensation functions should be considered as candidates for hardware acceleration.
5.5.4 Implementing an MPEG-2 Decoder in QuickMIPS

- The processing pipeline begins with the RLD that implements a subset of the VLD.
- The RLD block feeds the iQ/iZZ block without the need for any data buffering. The output of the iQ/iZZ block feeds the IDCT through a RAM buffer.
- The motion compensation function is broken into the following two components:
  - Spatial Blender
  - Temporal Blender
The decoder implementations, both with and without iQ/iZZ, use the resources listed in the following table, as reported by SpDE:

<table>
<thead>
<tr>
<th>Resource</th>
<th>No iQ/iZZ</th>
<th>With iQ/iZZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Utilized Cells</td>
<td>1823 (90.4%)</td>
<td>2015 (100%)</td>
</tr>
<tr>
<td>Utilized Logic Cell Frags</td>
<td>6898 (57%)</td>
<td>8788 (72.7%)</td>
</tr>
<tr>
<td>RAM</td>
<td>23</td>
<td>28</td>
</tr>
<tr>
<td>ECU</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>Registers</td>
<td>1807 (44.8%)</td>
<td>2129 (52.8%)</td>
</tr>
<tr>
<td>Decoder Max Freq.</td>
<td>63 MHz</td>
<td>69 MHz</td>
</tr>
<tr>
<td>AMBA Max Freq.</td>
<td>90 MHz</td>
<td>85 MHz</td>
</tr>
</tbody>
</table>
To understand how the resource utilization is distributed between the blocks, each block within the MPEG-2 decoder was individually synthesized (except for the item OTHERs) using SpDE.

The results are listed in the following table.

### Resource Utilization of the MPEG-2 Decoder by Block

<table>
<thead>
<tr>
<th>Block</th>
<th>Logic Cells (Total of 2016)</th>
<th>Flip-Flops(^a) (Total of 4032)</th>
<th>ECUs (Total of 18)</th>
<th>RAMs (Total of 36)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPEG decoder (all blocks)</td>
<td>2015</td>
<td>2129</td>
<td>14</td>
<td>28</td>
</tr>
<tr>
<td>RLD</td>
<td>227</td>
<td>110</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>IQ_IZZ</td>
<td>401</td>
<td>209</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>IDCT</td>
<td>409</td>
<td>510</td>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td>MC</td>
<td>688</td>
<td>411</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>FRAME_CTRL</td>
<td>327</td>
<td>231</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>OTHERs (ahb_slave, parser, etc.)</td>
<td>–</td>
<td>658(^b)</td>
<td>0</td>
<td>9</td>
</tr>
</tbody>
</table>
To **improve** the overall performance of the implemented design, the following constraints should be applied in Synplify.

- Frequency target relaxed to a very low value (i.e., 10 MHz)
- Buffering option turned off
5.5.5 Mapping constraints

◆ The mapped netlist is fed into SpDE with the following constraints:
  - Over-constrain speed to 100 MHz.
  - Set necessary timing exceptions (e.g., false paths on RESETs, inter-clock domain signals, etc.).
  - Set a target window placement on a particular block (i.e., the ahb_slave block was placed near the interface to the standard cell portion of the QL902M).
  - Pull the flops into I/Os to improve interface timing.
  - Use available global clock networks to route timing critical signals.
  - Perform a second iteration to improve timing—The first run revealed the critical paths, which are properly constrained in the second iteration. This step resulted in up to 8 MHz improvement.
5.5.5 Benchmark Performance (1/5)

- Two versions of the hardware were created and programmed into separate QL902M devices:
  - With RLD/iQ/iZZ — One version contains all of the blocks described.
  - Without RLD/iQ/iZZ — Another stripped-down version contains everything except the RLD and iQ/iZZ blocks. For this version, the RLD and iQ/iZZ functions are performed in software.
Each version was tested and run under the following conditions:

- CPU core frequency of 175 MHz
- AHB System Bus operating at one-half of the CPU core frequency (87.5 MHz)
- Accelerator circuit operating at 66 MHz
Software-only and each hardware version were measured for frame rate of two MPEG video streams to determine the performance:

- **Reds**—Low bit rate, low resolution MPEG-1 stream.
- **VE**—High bit rate 720x480 MPEG-2 DVD stream, referred to earlier as the DVD reference stream, used in the initial software profiling.
5.5.5 Benchmark Performance (4/5)

- Comparison of MPEG-2 decoding for two different MPEG streams showing frame rates for three operating conditions.
Comparison of MPEG-2 decoding for two different MPEG streams showing the improvement multiplier of two hardware accelerator options.
5.5.5 Power Measurement (1/2)

◆ Measurements were taken of the I/O, core voltages and current to determine the power consumption of the three versions of the decoder (i.e., software-only, software/hardware without RLD/iQ/iZZ, and software/hardware with RLD/iQ/iZZ).

◆ These measurements were taken with the high bit-rate MPEG-2 VE stream.

Power Consumption Data for the MPEG-2 Decoder

<table>
<thead>
<tr>
<th>Condition</th>
<th>Core+I/O Power (mW)</th>
<th>Frame Rate (fps)</th>
<th>Energy-per-Frame (mWatt seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software-only on QuickMIPS running at 175 MHz CPU clock</td>
<td>930</td>
<td>3.0</td>
<td>310</td>
</tr>
<tr>
<td>Software/Hardware without RLD/iQ/iZZ on QuickMIPS running at 175 MHz CPU and 66 MHz Fabric clock</td>
<td>1077</td>
<td>25.4</td>
<td>42</td>
</tr>
<tr>
<td>Software/Hardware with RLD/iQ/iZZ on QuickMIPS running at 175 MHz CPU and 66 MHz Fabric clock</td>
<td>1134</td>
<td>30.4</td>
<td>37</td>
</tr>
</tbody>
</table>
The results in the following show that the software decoder provides just over half an hour of DVD quality video.

By comparison, the iQ/iZZ hardware version provides 270 minutes of play time.

That is more than 8 times longer than the software-only decoder!
The use of hardware acceleration offers the system designer two specific options:

- **Optimization for power and system cost**
  - Using a slower CPU reduces power dissipation, potentially eliminating the requirement for heatsinks and supplemental cooling.

- **Delivering value-added functionality**
  - Off-loading the CPU core of cycle-consuming functions provides the opportunity to offer additional system functions.
The magnitude of the performance gain depends on two primary elements:

- **Algorithm characteristics**
  - Efficiency with which the code base segments can be migrated to hardware, avoiding the time-consuming data transfers between modules.

- **Parallelism**
  - Our experience with the MPEG algorithm indicated that a 3–4x performance improvement over a software-only implementation can be achieved by replacing software subroutines with a hardware equivalent.
  - The tenfold improvement was achieved by rewriting the algorithm to invoke concurrent processing in the CPU core and the hardware acceleration logic.
5.5.6 Conclusions (3/4)

- Implications for Cost and Power Consumption
  - Power consumption
  - Cost
  - Reliability
5.5.6 Conclusions (4/4)

- Implications for System Design
  - A well designed, optimized solution needs to examine the following:
  - Dataflow through the system
  - Partitioning
  - Parallel or sequential
  - Localization of data
  - Programmable logic architecture

Reference (2/2)

