An Implementation of Hardware Accelerator using Dynamically Reconfigurable Architecture

Takashi Yoshikawa, Yutaka Yamada and Shigehiro Asano
Toshiba R&D Center
Outline

- Motivation
- Architecture overview
- Application example (H.264 decode)
- Evaluation
  - Performance
  - Area consumption
- Conclusion
Motivation

Need better solution for multimedia applications

- Processor solution is not efficient
  - Lots of data preparation before actual operations
    - Data alignment, Shuffle, Shift
  - SIMD operations don’t always fit multimedia applications
    - Sometimes an instruction requires two or more operations (ex. ++-- ++--) to maximize its efficiency
  - Difficult to add newly defined instructions to an existing ISA for supporting these operations
    - 100, 200, or even more instructions needs to be added

- Hardware engine is efficient enough but not flexible
  - Needs a new design for each application
  - Not easy to fix bugs
Motivation (Cont’d)

- Reconfigurable logic may solve these problems but….
  - Utilization of logic/network must be high enough
  - Overhead of loading configuration must be covered with processing time

- We propose dynamically reconfigurable HW engine optimized for multimedia applications
Outline

- Motivation
- Architecture overview
- Application example (H.264 decode)
- Evaluation
  - Performance
  - Area consumption
- Conclusion
Architecture overview (entire block diagram)
Architecture overview (data/code transfer)

- **Host Processor**
- **DMAC**
- **System Memory**
- **I/O Buffer**
  - **I/O Buffer Ctrl**
  - **Write Control**
  - **Input Ctrl**
- **Formatter0**
  - **Output Ctrl**
- **Formatter1**
  - **Output Ctrl**
- **Code Buffer**
  - **Input Ctrl**
- **Inter-Unit Buffer**
  - **Input Ctrl**
  - **AUX1**
    - **Output Ctrl**
  - **AUX0**
    - **Output Ctrl**
- **Request**

Data flow:
- Red line indicates data flow.
- Dashed green line indicates code flow.

(if not in Code Buffer)
Architecture overview (data/code transfer)

- DMAC controls the transfer of data/codes between a system memory and I/O Buffer/Code Buffer
  - The code consists of the configuration bits and the control code
    - The control code specifies which configuration bits should be applied to the reconfigurable units at any given cycle
- DMAC also initiates code transfer from Code Buffer to the reconfigurable units
  - Reconfigurable units: Formatters, AUXs and Write Control Unit
- A host processor issues a command with a system memory address to DMAC as a request
  - Issued through the control bus
Architecture overview (Formatter)

- Host Processor
- DMAC
- System Memory
- Processor
- Memory
- I/O Buffer
  - I/O Buffer Ctrl
  - Write Control
  - Input Ctrl
- Inter-Unit Buffer
  - Input Ctrl
  - AUX1
  - Output Ctrl
- AUX0
  - Output Ctrl
- Formatter0
  - Output Ctrl
- Formatter1
  - Output Ctrl

Data flow:
- Control bus
- Data flow

HotChips 2006
Formatter Units

- Each consists of Input Control, Output Control, full crossbar switches and 5 stages of processing element (PE)
  - Next two slides describe PE in detail
  - Input Control reads data from I/O buffer (or Inter Unit Buffer) and send the data along with a Context ID
    - Context ID is a pointer to one of the configurations of PE

- Output Control receives the data from the last stage of PE and writes it into Inter Unit Buffer
Architecture overview (Formatter)

- **Formatter**: 16x8 (8bit) crossbars available at Formatter0
- **Each PE has its own configuration memory**
- **Only 4 patterns are available**
- **Data, context ID and valid (described later) goes down the pipeline in sync**
- **8x8 (16bit) crossbar available at Formatter1**

- **16-bit ALU * 8**
- **ConfigMem**
- **Shuffle**
- **data A**
- **data B**
- **ID**
- **valid**

- **PE0**
- **PE1 (same as PE0)**
- **PE2 (same as PE0)**
- **PE3 (same as PE0)**
- **PE4 (same as PE0)**

- **XB**
Architecture overview (Processing Element)

- **Processing Element (PE)**

  Each ALU can be configured separately.

  - 2 outputs each
  - From Previous PE
  - data A
  - data B
  - ID
  - valid

  To Next PE

  ConfigMem

  HotChips 2006
Architecture overview (Inter-Unit Buffer)

From Formatter0  From Formatter1  From AUX0  From AUX1

data  select

MUX  MUX  MUX  MUX  MUX  MUX  MUX  MUX
buf  buf  buf  buf  buf  buf  buf  buf

To Formatter1  To AUX0  To AUX1  To Write Ctrl

HotChips 2006
Architecture overview (Inter-Unit Buffer)

- Inter-Unit Buffer (IUB)
  - Consists of 14 data buffers (size: 128bit each)
  - Formatters and AUX write output data into one of these buffers
    - The buffer ID is associated with a context ID from those units
  - Conflict of buffer writes has been statically avoided by the control code
    - All the timing of buffer writes is deterministic
  - Each buffer has a Valid Bit
    - Described in detail in the application example
Architecture overview (AUX units)

- AUX (Auxiliary) Units
  - Perform operations which can’t be handled by Formatter
    - Multiplication, 32bit operations, etc.
  - Each AUX unit consists of Input Control, Output Control and single reconfigurable SIMD unit
    - SIMD unit consists of eight 32-bit integer units
      - Multiplier is 16-bit wide
      - Result is held in a 32x8 bit accumulator
    - All of the operation units share the same configuration
    - One AUX can transfer its output data to the other AUX directly
      - Inter-Unit Buffer may also be used if necessary
HotChips 2006

Architecture overview (Write Control Unit)

- **Host Processor**
- **DMAC**
- **System Memory**
- **I/O Buffer**
  - I/O Buffer Ctrl
  - Write Control
  - Input Ctrl
- **Inter-Unit Buffer**
  - Input Ctrl
  - AUX1
    - Output Ctrl
  - AUX0
    - Output Ctrl
  - Formatter0
    - Output Ctrl
  - Formatter1
    - Output Ctrl

**Note:** HotChips 2006
Architecture overview (Write Control Unit)

- **Write Control Unit**
  - Reads 128bit data from Inter-Unit Buffer and writes it into I/O buffer
    - Data shuffle available (8x8 [16bit] full crossbar)
  - Context IDs are issued in the same manner as Input Control of Formatter/AUX
  - The Context ID is associated with:
    - The buffer containing write data
    - 128bit-aligned write address
      - The address is given in conjunction with the control code
    - Byte enable (for partial write)
    - Configuration of a crossbar switch
Architecture overview (Timing Chart)

- Timing chart example

Number represents a Context ID for each unit

| Load0 | Load1 | Formatter0 (PE0) | Formatter0 (PE1) | Formatter0 (PE2) | Formatter0 (PE3) | Formatter0 (PE4) | IUB Formatter1 (A0) | IUB Formatter1 (B0) | Formatter1 (PE0) | Formatter1 (PE1) | Formatter1 (PE2) | Formatter1 (PE3) | Formatter1 (PE4) | IUB AUX0 (A0) | IUB AUX0 (B0) | IUB AUX1 (A0) | IUB AUX1 (B0) | IUB AUX1 (A1) | IUB AUX1 (B1) | AUX0 | AUX1 | IUB Write Control (0) | IUB Write Control (1) | Store0 | Store1 |
| 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 |

Pattern of Context ID change is statically specified in the control code.

Starts when buffer A0 & B0 both become valid.

4 context IDs are used at the same cycle.

Context ID changes as frequently as every cycle.

HotChips 2006
Outline

- Motivation
- Architecture overview
- Application example (H.264 decode)
- Evaluation
  - Performance
  - Area consumption
- Conclusion
Application example (H.264 decode: intra16x16 prediction [plane])

Context Dependency Graph

Original Code

unsigned char Org[768]
void intra16x16_plane_prediction(uchar *left, uchar *top) {
    int H,V,a,b,c,t,i,pt=132,p;
    V=H=-top[7]*8;
    for (b=1;b<8;b++)
    {
        H+=top[15]*8; V+=left[15]*8;
        a=(top[15]+left[15])<<4;
        for (i = 0; i < 256; i++)
        {
            p=t>>5; t+=b;
            Org[pt++]=p;
            if ((i & 15) == 15)
            {
                t+=c-b*16; pt+=16;
            }
        }
    }
    for (i = 0; i < 256; i++)
    {
        p=t>>5; t+=b;
        Org[pt++]=p;
        if ((i & 15) == 15)
        {
            t+=c-b*16; pt+=16;
        }
    }
}
Application example (operation graph)

ALU & full crossbar/shuffle configuration for Formatter0 context 0&1

A=top[0..15]  B=top[0..15]

0123456789ABCDEF
A=F89ABCDE  B=76543210

Shuffle pattern 1
01234567 -> 10325476

Shuffle pattern 2
01234567 -> 10325476

* blue arrow is subtractor
* both outputs are same by default

* bold arrow forwarded to red arrow

○ unused operation
(unused data paths are not shown)
Application example (Timing Chart)

- Timing Chart (96 cycles in total)

- Execution images of these cycles are shown in the next animation

- IUB AUX1 is not used (AUX0 transfers data to AUX1 directly)
A new Context ID is issued to PE0 when all the input data required for the context become valid. The ID is issued with data and valid=1.
**Application example (Execution Image)**

When PE receives a valid Context ID, it changes its configuration accordingly and performs calculation & data shuffle.

<table>
<thead>
<tr>
<th>PE</th>
<th>Inst</th>
<th>ALU * 8</th>
<th>ID</th>
<th>valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>data A</td>
<td>data B</td>
<td>Mem</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>data A</td>
<td>data B</td>
<td>Mem</td>
</tr>
</tbody>
</table>

**Data A**

**Data B**

**ID**

**valid**
At the next cycle, both data remain valid, therefore the next Context ID is issued.
The change of configuration and data calculation/shuffle are performed in a pipeline fashion. PE0 & PE1 use different Context ID.
Outline

- Motivation
- Architecture overview
- Application example (H.264 decode)
- Evaluation
  - Performance
  - Area consumption
- Conclusion
**Evaluation (Performance)**

- **Application: H.264 decode**
  - Estimated number of cycles required for decoding one macroblock

<table>
<thead>
<tr>
<th>Name</th>
<th>Cycles</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our Accelerator (ideal)</td>
<td>3792 (intra) 5705 (inter)</td>
<td>ideal cycles expected for decoding the most demanding macroblock</td>
</tr>
<tr>
<td>Our Accelerator (real)</td>
<td>4923 (intra) 7108 (inter)</td>
<td>Actual cycles required for decoding the most demanding macroblock</td>
</tr>
<tr>
<td>Our Accelerator (typical)</td>
<td>3660 (average)</td>
<td>When 2Mbps VGA stream is decoded</td>
</tr>
<tr>
<td>PowerPC G5 @2GHz</td>
<td>16576 (average)</td>
<td>dual G5@2Ghz recommended by Apple for 8Mbps, 24fps HD stream</td>
</tr>
</tbody>
</table>

More than 4 times faster than PowerPC G5 in terms of number of cycles required
Evaluation (Performance Breakdown)

- Timing chart (real, intra)

1100 more cycles are required than the ideal cycles
- The reason: configuration transfer cannot be completely hidden by signal processing
  - It can be hidden if the number of configuration memories increases (currently two buffers)
# Evaluation (Area consumption)

## Area (logic gate) consumption

<table>
<thead>
<tr>
<th>Component</th>
<th>Logic [gate size]</th>
<th>F/F Memory [gate size]</th>
<th>Memory Size [bit]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formatter (x2)</td>
<td>87,500</td>
<td>31,000</td>
<td>3,136</td>
</tr>
<tr>
<td>- Processing Element (x5 in Form.)</td>
<td>18,700</td>
<td>7,500</td>
<td>656</td>
</tr>
<tr>
<td>- ALU (x8 in PE)</td>
<td>1,700</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>- Shuffle (x1 in PE)</td>
<td>1,000</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>- Buffer (x1 in PE)</td>
<td>2,200</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Micro Controller (x5)</td>
<td>800</td>
<td>12,000</td>
<td>1,152</td>
</tr>
<tr>
<td>Total Inter Unit Ctrl. (Input &amp; Output)</td>
<td>20,000</td>
<td>136,000</td>
<td>13,696</td>
</tr>
<tr>
<td>Crossbar (Formatter0)</td>
<td>4,000</td>
<td>22,000</td>
<td>2,112</td>
</tr>
<tr>
<td>Crossbar (Formatter1, Write Ctrl Unit)</td>
<td>3,000</td>
<td>7,600</td>
<td>768</td>
</tr>
<tr>
<td>AUX (x2)</td>
<td>45,000</td>
<td>9,500</td>
<td>960</td>
</tr>
<tr>
<td>I/O Buffer Interface (Read &amp; Write)</td>
<td>6,000</td>
<td>7,600</td>
<td>800</td>
</tr>
<tr>
<td>Inter Unit Buffer</td>
<td>20,000</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>319,000</strong></td>
<td><strong>314,200</strong></td>
<td><strong>31,584</strong></td>
</tr>
</tbody>
</table>
Evaluation (Chip layout)
Outline

- Motivation
- Architecture overview
- Application example (H.264 decode)
- Evaluation
  - Performance
  - Area consumption
- Conclusion
Conclusion

- Presented an implementation of hardware accelerator using dynamically reconfigurable architecture

  - Efficient signal processing can be achieved by reconfiguring ALUs and crossbars dynamically
    - More efficient than a general SIMD unit because each crossbar/16bit-ALU can be reconfigured differently
  - Not limited to the acceleration of H.264 decode
    - Can also accelerate H.264 encode and other codecs by applying different configurations
Conclusion (cont’d)

- **Performance**
  - Expects 30 frames/sec with 3 accelerators running at 300MHz for decoding Full HD H.264 video
    - More than 4 times faster than PowerPC G5 in terms of number of cycles required for decoding one macroblock
- **Area consumption (number of logic gates)**
  - 319K Gates (excluding memory)
    - Small enough considering its performance
    - Size of buffer memory will be 24KB or less
Evaluation (Comparison with PowerPC G5)

- Application: H.264 decoder
  - Full HD frame (1920x1080 pixels) consists of 8100 macroblocks
    - one macroblock = 16x16 pixels
  - To achieve frame rate = 24 Full HD frame/sec, one macroblock must be decoded in 5.14 sec
    - Requires dual 2.0GHz Power Mac G5 (Apple’s recommendation)
      - decoded w/ Quicktime7.1 (AltiVec used)
      - 5.14 sec = 10288 cycles at 2.0GHz
### Evaluation (Function level performance results)

<table>
<thead>
<tr>
<th>Name</th>
<th>Cycles</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>IQ/IDCT</td>
<td>221</td>
<td>204 if intra16x16 prediction is not used</td>
</tr>
<tr>
<td>Intra prediction (luma/chroma)</td>
<td>~ 448 / 75</td>
<td>motion compensation (luma) included</td>
</tr>
<tr>
<td>Motion compensation</td>
<td>133</td>
<td>69 if motion compensation (luma) excluded</td>
</tr>
<tr>
<td>Inter prediction (luma/chroma)</td>
<td>~ 1579/~163</td>
<td>Worst case (6/4-tap filter applied max. times)</td>
</tr>
<tr>
<td>Deblocking filter luma</td>
<td>~ 2112</td>
<td>Worst case (filter applied maximum times)</td>
</tr>
<tr>
<td>Deblocking filter chroma</td>
<td>~ 784</td>
<td>Worst case (filter applied maximum times)</td>
</tr>
<tr>
<td>Store macroblock</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>