CEG5010: Variable Radix Variable Wordlength Architectures

From PhD presentation of Leong Monk Ping, Norris

September 25, 2001

Overview

• Introduction
• fp design environment
• Variable-radix, variable-wordlength architecture
• Applications
• Conclusion
Introduction

• Limitations of microprocessor systems
  – Software programs executing on microprocessor systems are essentially sequences of instructions supported by the microprocessor architecture
  – Large footprint, high power requirements and large heat dissipation, applications to mobile devices and embedded systems are often not feasible
• Hardware implementation as a solution to these problems
  – Hardware parallelism and dedicated logic
  – Customized for a specific application, better power efficiency and higher computational density
  – Non-recurrent engineering cost associated with ASIC

Introduction

• FPGAs are a solution to these problems
  – Reconfigurable nature allows multiple designs to be programmed on the same device at different times and reduces cost
  – Density and speed of FPGA devices are increasing
  – Shorter design time
  – Moore’s law guarantees the availability of larger and faster devices, as well as reducing design time
• The most straightforward deployment is to couple the FPGA with a microprocessor
• This dissertation focused on the challenges in the design of FPGA-based coprocessors
Motivation and Aims

• Challenges in designing an FPGA-based coprocessor
  – Computationally intensive portions of a program are extracted into hardware which is executed on an FPGA device, increasing design complexity
  – Resources on an FPGA are limited, designers need control over the tradeoff between area and performance

Motivation and Aims

<table>
<thead>
<tr>
<th>Challenges</th>
<th>Aims</th>
</tr>
</thead>
<tbody>
<tr>
<td>Increased design complexity due to the division of a program's execution into hardware and software</td>
<td>Methods that can partially or entirely automate the process of translating a program into an efficient hardware design</td>
</tr>
<tr>
<td>Hardware resources are limited and designers should have the option to sacrifice performance to fit a design into a device</td>
<td>Designers may explore the tradeoff between area and performance. A single description leads to multiple implementations with different requirements</td>
</tr>
</tbody>
</table>
Contributions

• *fp* design environment
  – translates floating-point software descriptions to fixed-point hardware implementations
  – optimizes area or precision of the resultant implementation
• Variable-radix, variable-wordlength (VRVW) architecture
  – an architecture which incorporates the advantages of both multiple wordlength and digit-serial architectures
  – the tradeoff among precision, area, latency and throughput can be more efficiently explored

*fp* Design Environment – Background

• Algorithms are designed in floating-point, but fixed-point hardware implementations are preferred
  – Uses less resources
  – Reduced design complexity, lower design cost
  – Higher throughput, lower latency
• Major difficulty in designing fixed-point hardware arises from quantization effects
• Each floating-point variable in the algorithm needs to be converted to fixed-point and be assigned sufficiently large fractional wordlength
**fp Design Environment – Background**

- In a multiple wordlength architecture, integer wordlengths of variables can also be individually assigned
  - Meaningful in FPGA designs due to the capability of bit-level processing
  - Better area efficiency
- Problem – how should the translation of floating-point variables to fixed-point be efficiently carried out?
- Considerations – design automation, ease of use, optimality of resultant implementation

**fp Design Environment – Outline**

- Primary input is a floating-point algorithmic description in C (sequential)
- Output is fixed-point hardware described in VHDL (parallel)
- Four major stages
  - Compilation
  - Simulation
  - Optimization
  - Hardware generation
**fp Design Environment – Parallelism**

- Assumption – designers have identified the innermost loop (the most computationally intensive portion) of a program
- Compilation converts the dataflow in the innermost loop identified by designers into a DAG
  - The DAG extracted essentially resolves the dependencies of operators in the algorithm, operator-level (coarse-grain) parallelism is elaborated
  - Can be directly mapped to a non-feedback pipelined hardware design, bit-level (fine-grain) parallelism is elaborated
- Suitable for high throughput hardware designs

**fp Design Environment – Optimization**

- Wordlength of every variable needs to be individually adjusted
  - Sufficient fractional wordlength so that the resultant quantization error of the algorithm satisfy certain accuracy criteria
  - Sufficient integer wordlength so that variables do not overflow
- Large and complex design space, especially when there are a lot of variables
- Two approaches were previously proposed – analytical-based and simulation-based
**fp Design Environment – Optimization**

- Wordlength optimization used a simulation-based approach
  - The dynamic ranges and quantization errors of variables depends on the input data
  - Analytical-based methods are too pessimistic, dynamic ranges were overestimated and the properties of input data were not utilized
  - Empirical results show a saving of 75% area at the expense of 5dB decrease in SNR
  - A global optimization on an integer vector of wordlengths
  - Long compilation time due to large and complex optimization space, but only done once per design

**fp Design Environment – Hardware Generation**

- Based on module libraries
- Each arithmetic operation may have multiple implementations (such as pipelined and multi-cycle), designers choose the most suitable implementation for the application
- Datapath is automatically synthesized, significantly reduce design effort
- Machine generated implementation is usually more optimized
- The implementation details are transparent to designers
**fp Design Environment – Summary**

- A design tool that automatically produces hardware designs from a software description
- Input is C language, output is VHDL
- Designers may specify the tradeoff between area and precision by giving different weights to these two components during optimization of wordlengths
- Problem – having determined the optimal wordlengths, can we produce a set of implementations with different performance/area tradeoffs?

---

**VRVW Architecture – Background**

- Digit-serial architectures
  - A widely used technique in hardware design
  - Radix (digit size) is a common parameter for all variables and operators in a traditional digit-serial architecture
  - The radix controls the level of parallelism, enabling designers to explore the tradeoff between area and performance
- Wordlength of a variable or an operator is the product of its digit size \(d\) and number of digits \(n\), \(w = n \times d\)
VRVW Architecture – Background

![Diagram showing clk, ctrl_x, and x over time with n cycles]

VRVW Architecture – Conflicting Design Methodologies

- An $n$-digit variable takes at least $n$ cycles to be processed in a digit-serial architecture.
- The traditional digit-serial architecture assumes all variables have the same wordlength.
- $n$ is common and every operator has the same throughput and is fully utilized.
VRVW Architecture – Conflicting Design Methodologies

• The multiple wordlength design methodology assigns every variable a different wordlength

• Problem
  – Both the design methodologies attempt to reduce area
  – Digit-serial architectures assume a common wordlength and area reduction is based on time-multiplexing of operators
  – Multiple wordlength architectures carry out area reduction by individually adjusting wordlengths of variables
  – Conflicting approaches
VRVW Architecture – Combining the Design Methodologies

- Variable-radix, variable-wordlength architecture
  - The number of digits $n$ remains fixed
  - The digit size $d$ can be different for every variable
- Traditional digit-serial architecture is a special case of VRVW architecture in which $d$ is constant
- Variables are parameterized by triplets $(f, w, d)$
  - $f$ – fractional wordlength
  - $w$ – total (integer and fractional) wordlength
  - $d$ – digit size

For example, $(3, 11, 4)$ and $n = 3$ means

```
Digit 2      Digit 1      Digit 0
b_7  b_7  b_6  b_5  b_4  b_3  b_2  b_1  b_0  b_{-1}  b_{-2}  b_{-3}
|----------------|
Sign extension  Fractional part
```
VRVW Architecture – Combining the Design Methodologies

- **Advantages**
  - All the operators have the same throughput, higher utilization is achieved
  - As digit size can be individually adjusted, the area overhead of increasing the wordlength of a variable is localized

- **Disadvantages**
  - Operators require operand to have the same radix, thus need radix conversion
  - An additional parameter to be determined, larger search space

- **Empirical results** (in the DCT application) shows as much as 87% improved area efficiency

---

VRVW Architecture – Radix Conversion

![Radix Conversion Diagram]

(3, 11, 4) → (5, 14, 5)

Digit 2 → Digit 2
Digit 1 → Digit 1
Digit 0 → Digit 0
VRVW Architecture –
Radix Conversion

• Needed if the operands to an operator are of different radices
• Radix conversion modules are placed near the associated operator, small impact on clock rate
• FPGAs are rich in registers, and experiments show that conversion modules match the FPGA architecture well
• During \( fp \)'s optimization
  – The area occupied by the radix conversion modules is taken into account
  – The optimization process can determine whether a radix conversion would lead to a global advantage

VRVW Architecture – Summary

• A methodology to incorporate the advantages of two commonly used but seldom simultaneously applied design methodologies
  – Multiple wordlength architecture
  – Digit-serial architecture
• This design space offers great flexibility and the tradeoff among precision, area and performance is addressed
• With \( fp \), a single algorithmic description may generate multiple VRVW implementations that satisfy different performance requirements
Applications – Post-Rendering 3D Warping

• An algorithm that renders a scene based on an input scene and a depth map
• Suitable for small changes in viewing parameters, usually used in conjunction with geometrical rendering
• Complexity depends only on resolution, independent of scene complexity
Applications – Post-Rendering 3D Warping

- An algorithm that renders a scene based on an input scene and a depth map
- Suitable for small changes in viewing parameters, usually used in conjunction with geometrical rendering
- Complexity depends only on resolution, independent of scene complexity

![FPGA Algorithm function](image1)

<table>
<thead>
<tr>
<th>Pixel 3D coordinates</th>
<th>X-Y-Z coordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel color values</td>
<td>R-G-B values</td>
</tr>
<tr>
<td>Output image buffer</td>
<td>Write data</td>
</tr>
<tr>
<td>External memory</td>
<td>Write address</td>
</tr>
</tbody>
</table>

![Projected coordinate on the new viewplane](image2)
Applications –
Post-Rendering 3D Warping

- Uses fp’s multiple wordlength design methodology to implement the pixel re-projection inner loop of the algorithm
- Multiple hardware implementations (each with 20000 lines of VHDL code) were generated from 25 lines of C code

```c
#define EQ(w, x, y, z) {w[0] = x; w[1] = y; w[2] = z;}

void normalize(fixed *n)
{
    fixed m;
    EQ(n, n[0] / m, n[1] / m, n[2] / m);
}

void warp(fixed *v, fixed *vd, fixed *p, fixed &h, fixed &k)
{
    fixed vp[3], v00[3], v01[3], v10[3], v11[3], nvp, l;
    EQ(vp, vd[2], 0, -vd[0]);
    normalize(vp);
    EQ(v00, vd[0] + vp[0], vd[1] + 1.0, vd[2] + vp[2]);
    EQ(v01, vd[0] + vp[0], vd[1] - 1.0, vd[2] + vp[2]);
    EQ(v11, vd[0] - vp[0], vd[1] - 1.0, vd[2] - vp[2]);
    normalize(nvp);
    h = (l * nvp[0] - v00[0]) / (v10[0] - v00[0]);
    k = (l * nvp[1] - v00[1]) / (v10[1] - v00[1]);
}
```
Applications - Discrete Cosine Transform

- Systolic computation of DCT via computation of discrete moments (DM)
- Setting up the relationship between DCT and DM
- The $N$-point DCT is defined by the equations

\[
X(k) = c_k \sum_{n=0}^{N-1} x(n) \cos \left( \frac{\pi (2n+1)k}{2N} \right), \quad 0 \leq k \leq N-1,
\]

\[
c_k = \begin{cases} 
1/\sqrt{N} & \text{if } k = 0, \\
\sqrt{2/N} & \text{otherwise.}
\end{cases}
\]

Discrete Cosine Transform

- The equation is transformable into

\[
X(k) = c_k \left( x_{k,0} + \sum_{r=0}^{p} a_r m_{k,2r} \right) + R_p, \quad 0 \leq k \leq N-1
\]

- $m_{k,2r}$ is the result of DM, $a_r$ is a vector of constants, $c_k$ is a constant, $R_p$ is the Taylor’s remainder term
- Ignoring $R_p$, DCT is transformed into a dot product of $m_{k,2r}$ and $a_r$, plus $x_{k,0}$ and times $c_k$
- Only involves multiplications and additions
Discrete Cosine Transform

- DM is computed by cascaded $p$-networks
- A $p$-network resembles a Pascal triangle

\[
\begin{align*}
(1+x)^n &= 1 + nx + \frac{n(n-1)}{2!}x^2 + \frac{n(n-1)(n-2)}{3!}x^3 + \cdots + x^n \\
&= \sum_{k=0}^{n} \binom{n}{k} x^k
\end{align*}
\]
Applications – Systolic Structure for the Computation of DCT

- Used fp’s area optimization feature and the VRVW architecture
- Tested implementations is for computing 8-point DCT, consisting of 157 adders and 5 multipliers
- A set of 70 implementations were experimented
  - Almost continuous area requirements from 366 to 4013 slices
  - Throughput from 1.6M DCT/s to 120.5M DCT/s
  - SNR from 19.7dB to 44.5dB

```c
void dct(fixed &xp, fixed *X)
{
    fixed x[2 * p + 1];
    double a;
    int i, j, k;
    x[2 * p] = xp[N - 1] + xp[N - 1];
    for (i = 2 * p - 1; i >= 0; i--)
        x[i] = x[i + 1] + x[i + 1];
    for (i = N - 2; i > 1; i--)
        for (j = 0; j <= 2 * p; j++)
            x[j] += xp[i];
        for (j = 2 * p - 1; j >= 0; j--)
            for (k = 0; k <= j; k++)
                x[k] += x[k + 1];
    for (i = 0; i <= 2 * p; i += 2)
        x[k] += xp[1];
    X[0] = xp[0] + x[2 * p];
    for (i = 0; i < 2 * p; i += 2)
        x[k] += xp[1];
    X[0] = xp[0] + x[2 * p];
    k = 1;
    for (i = 2 * p - 2; i >= 0; i -= 2)
        a = pow(M_PI, (double) (2 * k)) / pow(2 * N);
        a /= (double) j;
        if (k++ % 2)
            a = -a;
        X[0] += a * x[i];
    X[1] = X[0] * sqrt(2 * N);
    X[0] /= sqrt(N);
}
```

- Compactly described in 33 lines of C code, and generates 50000 lines of VHDL code for each of the 70 implementations
Applications – Systolic Structure for the Computation of DCT

Area requirements against number of digits

Performance against number of digits

Applications – Systolic Structure for the Computation of DCT
Applications – Systolic Structure for the Computation of DCT

• Results show VRVW has up to 87% improved area efficiency
  – Digit-serial – all variables and operators must have the same wordlength
  – VRVW – all variables and operators can have different wordlengths
Conclusion

• *fp* and VRVW architecture are tools to simplify the FPGA-based coprocessor design process
  – Automatic translation from a high-level floating-point algorithmic description to optimized fixed-point hardware
  – Help designers to explore the tradeoffs among precision, area and performance
  – Designers can concentrate on the higher-level algorithmic issues during development, generate implementations with different area, precision and performance tradeoffs, and later choose the most suitable implementation