Automatically exploiting regularity in applications to reduce reconfiguration memory requirements

Fatma Abouelella
Karel Bruneel and Dirk Stroobandt
Fatma.abouelella@ugent.be
Have you ever tried run-time reconfiguration?

• How much time and effort have you spent to design for run-time reconfiguration system?

• Can you afford the reconfiguration overhead?
OUTLINE

• FPGA configuration
• Dynamic Circuit Specialization (DCS)
• Exploitation of regularity in applications
• Experiments
• Conclusion
OUTLINE

• FPGA configuration
• Dynamic Circuit Specialization (DCS)
• Exploitation of regularity in applications
• Experiments
• Conclusion
FPGA architecture

Programmable routing

I/O block

Logic Block

Ghent University - Computer Systems Lab (CSL) - FPL 2012 - 30 August 2012
FPGA architecture

Programmable routing

I/O block

Logic Block
FPGA reconfiguration
FPGA reconfiguration

How can we limit the reconfiguration overhead?
Reconfiguration overhead

Partial configuration 1

Partial configuration 2

\vdots

Partial configuration N

Large memory

Long run-time reconfiguration time

FPGA

Configuration memory

0 1 1 0 1 0 1
1 0 1 1 0 1 1
1 0 0 1 1 0 1
OUTLINE

• FPGA configuration
• Dynamic Circuit Specialization (DCS)
• Exploitation of regularity in applications
• Experiments
• Conclusion
Parameterized Configuration (PC)

Parameters

\{ 0 \ 1 \ 0 \ \text{A+B} \ \text{AB} \ A \ 1 \}\n
Parameterized Configuration

A B
0 0
0 1
1 0
1 1

Specialized Configurations

\{ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \}
\{ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \}
\{ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 1 \}
\{ 0 \ 1 \ 0 \ 1 \ 1 \ 1 \ 1 \}

Example: Adaptive filtering

- FIR filter:
  - 16 taps
  - 8-bit input
  - 8-bit coefficients

The implementation of this generic FIR occupies large number of LUTs in the FPGA
Dynamic Circuit Specialization (DCS) technique

The implementation of a specialized FIR occupies less number of LUTs in the FPGA

But

$2^{128}$ configurations should be stored
Parameterized configuration and DCS

• By exploiting PC in DCS systems

  - One configuration needed to be stored
  - Configuration generation time is evaluation time
TLUT tool flow*

Parameterized HDL

Synthesis

TLUTMAP

Place & route

Template Configuration

Generate PC

Parameterized Configuration (PC)

Generate reconf. functions

Reconfiguration functions

At run-time

Generate Specialized configuration

Specialized configurations

Parameter values

Current problem

• Exploiting parameterized configurations in the TLUT tool flow

+ Overhead of dynamically specializing circuits is significantly reduced
  (One configuration stored + configuration generation time)

- PC memory can be very large especially when applications scale
Proposed solution achievements

For 64-tap FIR filter

<table>
<thead>
<tr>
<th></th>
<th>Before</th>
<th>After</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PC resources (Kbyte)</td>
<td>742.67</td>
<td>9.85</td>
<td>76 times less memory</td>
</tr>
</tbody>
</table>

Before vs. After comparison for PC resources (Kbyte) for a 64-tap FIR filter, showing a significant reduction in memory usage.
Proposed solution achievements

For 64-tap FIR filter

<table>
<thead>
<tr>
<th></th>
<th>Before</th>
<th>After</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PC resources (Kbyte)</td>
<td>742.67</td>
<td>9.85</td>
<td>76 times less memory</td>
</tr>
</tbody>
</table>

Local Memory

Processor

742.62 Kbyte
Proposed solution achievements

For 64-tap FIR filter

<table>
<thead>
<tr>
<th></th>
<th>Before</th>
<th>After</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PC resources (Kbyte)</td>
<td>742.67</td>
<td>9.85</td>
<td>76 times less memory</td>
</tr>
</tbody>
</table>

Before: 742.62 Kbyte

- Processor
- Local Memory
- Slow connection

After: 9.85 Kbyte

- External Memory
- 742.62 Kbyte
Proposed solution achievements

For 64-tap FIR filter

<table>
<thead>
<tr>
<th>PC resources (Kbyte)</th>
<th>Before</th>
<th>After</th>
<th>76 times less memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>742.67</td>
<td>9.85</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

9.85 Kbyte
Local Memory

Processor
OUTLINE

• FPGA configuration
• Dynamic Circuit Specialization (DCS)
• Exploitation of regularity in applications
• Experiments
• Conclusion
Main idea

Repetitive module

Configuration Data
Main idea

At run-time

Configuration Data
Main idea

At run-time

Configuration Data
Main idea

At run-time

$cd_1 \quad cd_2 \quad \ldots$

Configuration Data
Main idea

At run-time

\[ \text{cd}_1 \quad \text{cd}_2 \quad \ldots \quad \text{cd}_{16} \]

Configuration Data

How to detect regularity in applications and transfer this regularity to the reconfiguration process?
TLUT tool flow with front end

entity FIR is
port (clk : in std_logic;
    i : in std_logic_vector(DIW-1 downto 0);
    -- Start PARAM
    c : in arrayOfStdlogicVectors;
    -- End PARAM
    o : out std_logic_vector(DOW-1 downto 0));
end FIR;

architecture struct of FIR is
begin
    -- declarations
    begin
        GS_FOR : for T in 0 to N-1 generate

            GS_IF1 : if (T = 0) generate
                MULTIPLIER0 : entity work.multiplier(rtl)
                    port map (i, c(T), mult(T));
                LOAD : entity work.init(rtl)
                    port map (clk, mult(T), inter(T+1));
            end generate GS_IF1;

            GS_IF2 : if (0 < T < N) generate
                MULTIPLIER0 : entity work.multiplier(rtl)
                    port map (i, c(T), mult(T));
                ADDER0 : entity work.seqadder(rtl)
                    port map (clk, mult(T), inter(T), inter(T+1));
            end generate GS_IF2;
        end generate GS_FOR;

        o <= inter(N);
    end struct;
TLUT tool flow with front end

entity FIR is
port (clk : in std_logic;
i : in std_logic_vector(DIW-1 downto 0);
c : in arrayOfStdlogicVectors;
o : out std_logic_vector(DOW-1 downto 0));
end FIR;

architecture struct of FIR is
begin

GS_FOR : for T in 0 to N-1 generate

GS_IF1 : if (T = 0) generate
  MULTIPLIER0 : entity work.multiplier(rtl)
  port map (i, c(T), mult(T));
  LOAD : entity work.init(rtl)
  port map (clk, mult(T), inter(T+1));
end generate GS_IF1;

GS_IF2 : if (0 < T < N) generate
  MULTIPLIER0 : entity work.multiplier(rtl)
  port map (i, c(T), mult(T));
  ADDER0 : entity work.seqadder(rtl)
  port map (clk, mult(T), inter(T), inter(T+1));
end generate GS_IF2;
end generate GS_FOR;
o <= inter(N);
end struct;
TLUT tool flow with front end

Abstract Syntax Tree (AST)

Parameterized HDL Design

Detected Reg. Struct.

HDL Processing

Generate HL Reconfig. Funct.

TLUT tool flow

Generate Reconfig. Procedure

Ghent University - Computer Systems Lab (CSL) - FPL 2012 - 30 August 2012
TLUT tool flow with front end

Parameterized HDL Design

- Detect Reg. Struct.
- HDL Processing
- Generate HL Reconfig. Funct.

TLUT tool flow

Modular Reconfig. Funct.

Generate Reconfig. Procedure

Procedure

Parameterized HDL Design

- Detect Reg. Struct.
- HDL Processing
- Generate HL Reconfig. Funct.

Module

Generate Reconfig. Procedure
TLUT tool flow with front end

Static HW AST

Entity: FIR

Ports

Arch.

Body

Declaration

ForGn

Range

$0 \leq T < N$

Body

IfGn1

Cond.

$T = 0$

Body

Component

Init

Ports

Arch.

mult

(T)

inter

(T+1)

i

mult

(T)

inter

(T+1)

i

mult

(T)

inter

(T+1)

i

mult

(T)

Template Configuration

Parameterized HDL Design

Detect Reg. Struct.

HDL Processing

Generate HL Reconfig. Funct.

TLUT tool flow

Modular Reconfig. Funct.

Generate Reconfig. Procedure

Ghent University - Computer Systems Lab (CSL) - FPL 2012 - 30 August 2012
TLUT tool flow with front end

SW AST

HL-RF-FIR
- Arg.
  - c
- Body
  - Stat.
  - initialization
  - For
  - Range
  - 0 ≤ T < N
  - Body
  - If1
  - Cond. T = 0
  - Body
  - If2
  - Cond. 0 < T < N
  - Body
  - Mod-RF Multiplier
  - Arg.
  - c (T)
  - Body
  - Mod-RF Multiplier
  - Arg.
  - c (T)
  - Body

Parameterized HDL Design
- Detect Reg. Struct.
- HDL Processing
- Generate HL Reconfig. Funct.
- TLUT tool flow
- Modular Reconfig. Funct.
- Generate Reconfig. Procedure

Template Configuration

Ghent University - Computer Systems Lab (CSL) - FPL 2012 - 30 August 2012
void HL-RF-FIR(int c[Ceil(N)][DIW]){ 
  for (T = 0; T < N; T++){
    if (T == 0)
      Mod-RF-multiplier (c[T]);
    if (T>0 && T<Ceil(N))
      Mod-RF-multiplier (c[T]);
  }
}

TLUT tool flow with front end

Parameterized HDL Design

Detect Reg. Struct.

HDL Processing

Generate HL Reconfig. Funct.

High-level Reconfig. Funct.

Modular Reconfig. Funct.

Generate Reconfig. Procedure

Template Configuration
TLUT tool flow with front end

Parameterized HDL Design

- Detect Reg. Struct.

HDL Processing

- Generate HL Reconfig. Funct.

TLUT tool flow

- High-level Reconfig. Funct.

Template Configuration

- Modular Reconfig. Funct.

Generate Reconfig. Procedure

Reconfiguration Procedure
OUTLINE

• FPGA configuration
• Dynamic Circuit Specialization (DCS)
• Exploitation of regularity in applications
  • Experiments
• Conclusion
Experiments

• Platform:
  • Xilinx Virtex-II Pro (XCVP30) FPGA
  • Reconfiguration procedure:
    • Set of C functions (HL and Modular RFs)
    • Executed on the embedded PowerPC of the Xilinx Virtex-II Pro

• Implementation methodologies:
  • Conventional
  • TLUT tool flow
  • TLUT tool flow with the front end (TLUT w.f.e.)

• measurements:
  • Design area (LUTs)
  • Maximum frequency
  • Memory resources needed to store reconfiguration procedure
  • Configuration generation time
FIR filter (64-tap)

<table>
<thead>
<tr>
<th></th>
<th>Implementation</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Design area (LUTs)</td>
<td>Frequency (Mhz)</td>
</tr>
<tr>
<td>Conventional</td>
<td>7578</td>
<td>85.44</td>
</tr>
<tr>
<td>TLUT</td>
<td>5080</td>
<td>195.72</td>
</tr>
<tr>
<td>TLUT w.f.e.</td>
<td>5080</td>
<td>185.61</td>
</tr>
</tbody>
</table>

*This generation time is associated to a reconfiguration procedure that is stored in an external memory.*

32% ~50% 76x ↓ 22x ↓
RegExp matcher (13-GB)

<table>
<thead>
<tr>
<th></th>
<th>Implementation</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Design area (LUTs)</td>
<td>Frequency (Mhz)</td>
</tr>
<tr>
<td>Conventional</td>
<td>709</td>
<td>14.80</td>
</tr>
<tr>
<td>TLUT</td>
<td>392</td>
<td>78.55</td>
</tr>
<tr>
<td>TLUT w.f.e.</td>
<td>377</td>
<td>49.21</td>
</tr>
</tbody>
</table>

- 44-46%
- 70-81%
- 10x ↓
- 3x ↓
### Ternary Content Addressable Memory (TCAM) (256 entries)

#### Implementation

<table>
<thead>
<tr>
<th></th>
<th>Design area (LUTs)</th>
<th>Frequency (Mhz)</th>
<th>Recon. Procedure (Kbyte)</th>
<th>Generation time (us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conventional</td>
<td>10871</td>
<td>65.47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLUT</td>
<td>4036</td>
<td>67.38</td>
<td>543.39</td>
<td>15123.66*</td>
</tr>
<tr>
<td>TLUT w.f.e.</td>
<td>4036</td>
<td>78.98</td>
<td>3.25</td>
<td>480.44</td>
</tr>
</tbody>
</table>

#### Overhead

- 62%
- 2-17%
- 167x ↓
- 31x ↓
OUTLINE

• FPGA configuration
• Dynamic Circuit Specialization (DCS)
• Exploitation of regularity in applications
• Experiments

• Conclusion
Conclusion

• The PC memory was very large especially when applications scale.

• We introduce an automatic method that exploits regularity to reduce PC memory requirements.

• Exploiting the regularity achieves:
  – PC memory reduction factor of 76 and 176 for the FIR filter (64-tap) and TCAM (256 entries).
  – The reduction factor increases when applications scale.
Thank you