Function-Level Processor (FLP): Raising Efficiency by Operating at Function Granularity for Market-Oriented MPSoC

Hamed Tabkhi*, Robert Bushey^, Gunar Schirner*

*Electrical and Computer Engineering
Northeastern University
Boston (MA), USA

^ Embedded Systems Products and Technology
Analog Devices Inc. (ADI)
Norwood (MA), USA
High-performance Low-Power Computing

• Increasing demand for high performance low power computing
  – Tens billions of operations per second
  – Less than few watts power
  – Example Markets:
    • Embedded vision
    • Software Define Radio (SDR)
    • Cyber Physical Systems (CPS)

• Instruction-Level Processors (ILP)
  – Great flexibility
  – Very high power / operation
    • 3% of power for computation [Keckler’11]
    • Remain: inst/data fetch, memory hierarchy, decode
Multi-Processor System-on-Chips (MP-SoCs)

- Heterogeneous composition for performance / power-efficiency
  - ILP (e.g. CPU, DSP, or even GPU)
  - Flexibility
  - Custom-HW for compute-intensive kernels
  - Power efficiency

- Recover NRE: aim for market (domain of applications)
- E.g. Vision, SDR, Radar

Just increase number of accelerators?
ILP + HW Accel. Scaling Limitations

• Approach:
  – HW-ACCs
    • Executes Compute-intense kernels
  – ILPs
    • Executes remaining application
    • Orchestrates HWACCs / coordinate data movement
  – On-chip scratchpad memory
    • Keep data between ILP and ACCs on-chip
      – Avoid costly off-chip memory access
  – Increasing # HWACCs not scalable
    – Large on-chip scratch-pad memory
      – But often under-utilized
        [Lyons’10] reports 30% utilization
    – High volume of on-/off-chip traffic
    – Significant synchronization overhead on ILPs

➢ Need new solutions balancing flexibility / efficiency.
Outline

• Intro: High-performance low-power computing
• Flexibility / Efficiency trade-off
• Related work
• Function-Level Programming
• Function-Level Processor (FLP)
• Experimental results
• Conclusions
Flexibility / Efficiency Trade-off

- **Architecture Class**

  - **HW IPs**
  - **FPGAs**
  - **ILPs**
  - **Control Processors**
  - **DSPs**
  - **GPUs**
  - **Application**
  - **Instruction**
  - **Gate**

**Flexibility**

- Need new solutions to fill flexibility / efficiency gap.
- High complexity but low performance efficiency
- Difficult to program
- Complex instruction/data fetch
- Custom HW IPs – Least flexible, but most efficient
- Has potential for most efficient implementation through fixed, application-specific data path.
- e.g. ASIC H.264 is 500x and 25x more energy efficient than RISC-like and optimized VLIW/SIMD implementations [Hameed’10]

**Efficiency**

- [GOPs/Watt]

- Flexibility/Efficiency Gap

- **HW IPs**
- **FPGAs**
- **ILPs**
- **GPUs**
- **DSPs**
- **Control Processors**
Outline

• Intro: High-performance low-power computing
• Flexibility / Efficiency trade-off
• Related work
• Function-Level Programming
• Function-Level Processor (FLP)
• Experimental results
• Conclusions
Related Work

- **Systems with many accelerators**
  - Compose larger applications out of many accelerators
  - e.g. Accelerator-Store [Lyons’10], Accelerator-Rich CMPs [Cong’12], CHARM [Cong’12]
    - Problem: Redundant on-chip traffic and scratch-pad for exchanging data between accelerators.

- **Coarse-Grain Reconfigurable Architectures (CGRAs)**
  - Interconnect homogeneous compute units via mesh/crossbar
  - Great for instruction-level parallelism (SW Loop-accelerations) [Hartenstein’01][Park’11]
  - Challenges:
    - Higher power consumption than HW ACCs
    - Tightly depends on host ILPs

- **Application-Specific Instruction Processors (ASIPs)**
  - Extend ILP data-path with custom-HW to execute C-functions, e.g. [Venkatesh’14]
    - Higher efficiency with fairly good flexibility
  - Problem: suffer from the limitation of ILPs
Outline

• Intro: High-performance low-power computing
• Flexibility / Efficiency trade-off
• Related work
• Function-Level Programming
• Function-Level Processor (FLP)
• Experimental results
• Conclusions
Raising Abstraction to Function-level

• Programming v.s. Architecture Abstraction

• Function-Level Processor (FLP):
  – Match programming abstraction at function-level granularity
    • Increase efficiency (coarser programmability)
    • Maintain flexibility
    • Simplify application composition
  – Focus on Market-oriented MPSoCs (for group of apps)
    • Main target: Streaming applications
      – e.g. Vision, SDR, Radar

H. Tabkhi, R. Bushey, G. Schirner
Function-level programming

- Applications composed of functions
- Requirements:
  - Identify common functions within market applications
    - Similar to libraries, such as OpenCV, OpenSDR
  - Identify common function interactions in desired applications

Applications of Market (Domain)

Function-Level Processor (FLP)
  - Architecture for function-level programming
Outline

• Intro: High-performance low-power computing
• Flexibility / Efficiency trade-off
• Related work
• Function-Level Programming
• Function-Level Processor (FLP)
• Experimental results
• Conclusions
• **FLP Overview**
  
  1. **Optimized Function Blocks (FBs)**
  2. MUX-based / market-specific communication
  3. Separation of data traffic
  4. Autonomous control and synchronization

• **FLP Principles:**
  - Target stream processing applications
    - E.g. Vision, SDR, Radar
  - Construct macro pipeline out of functions to build application
    - Self controlled function blocks
    - Power gate unused blocks
  - Compute contiguously inside FLP
    - Avoid costly data movement
  - Limited ILP interaction
    - Independent processing unit

• **FLP Architecture Components:**
  1. Optimized Function Blocks (FBs)
  2. MUX-based / market-specific communication
  3. Separation of data traffic
  4. Autonomous control and synchronization
Function Block: Traffic Separation

- **Insight:** Not all traffic is equal!
  - Routes, roles and importance
  - Access patterns

**Streaming:** input / output data stream
- Independent of computation realization
- Can be hidden by FB-to-FB connection

**Operational:** traffic generated by algorithm itself (other than output)
- Part of algorithm (algorithm-intrinsic)
  - E.g. model of stream
- Reuse potential: cache?
- Size? May be emitted to mem hierarchy
Function Block (FB)

- **Decomposition**
  
  (a) Function-specific (but configurable) data-path
  - Implements function \( y = f(x) \)
  - e.g., CNV, Histogram, Gaussian

  (b) Configuration / control registers
  - Alter / Parameterize functionality
  - e.g., threshold values, dimensions

  (c) Stream-in / stream-out
  - Stream data for direct FB-to-FB communication
  - e.g., data samples, frame pixels

  (d) Stream buffer
  - Temporary storage
  - e.g., integral operation

  (e) Operational data / local buffer
  - Access the memory hierarchy
  - e.g. Gaussian parameters, CNV co-efficient values
**MUX-Based Interconnection**

- **Point-to-point communication between FBs**
  - Streaming data
  - Set of configurable MUXs
    - Hides stream traffic from memory hierarchy

- **Sparse connectivity**
  - Compatible data types
  - Connectivity required by application set
  - Arranged in pipeline stages with forwarding and backwarding

- **Data type and flow control**
  - Marshaling / de-marshaling of streaming data
  - Side-channel signals for flow control
Autonomous Control / FSA

- **Autonomous Control**
  - FLP with centralized Control Unit (CU)
    - Fetches configuration through DMA
    - Transitions across configurations
    - Distributes configuration
    - Interconnect and FB configuration
    - Coordinates FBs
  - Minimizes or even eliminates need for ILP-centric control

- **Function-Set Architecture (FSA)**
  - Exposes
    - Type and number of FBs
    - FB configuration
    - FBs connectivity
  - Coarse-grained programmability at the function-level
    - Significantly more abstract than ISA
    - Comparable to developing applications out of API function

---

**Function-Level Processor (FLP)**

H. Tabkhi, R. Bushey, G. Schirner

17
FLP Architecture

- **Efficiency contributors:**
  - Optimized data-path per function block
  - Eliminate instruction fetch overhead
  - Eliminate system-level traffic for streaming data
  - Direct system inputs/outputs
  - Dedicating memory hierarchy only for reusable data (operational traffic)
  - Minimize ILP interaction
System Integration

- FLP as independent processor
- Pair with ILP cores
  - Create complete control and analytic applications.
    - FLP for compute intense processing
    - ILPs for the top-level adaptive control and intelligence.
Outline

• Intro: High-performance low-power computing
• Flexibility / Efficiency trade-off
• Related work
• Function-Level Programming
• Function-Level Processor (FLP)
• Experimental results
• Conclusions
Experimental Setup

• Pipeline Vision Processor (PVP)
  - Contains 11 FBs
    • Manually selected (app. analysis)
  - Vision filters: Edge detection / enhancement
  - Target market:
    • Advanced Deriver Assistance Systems (ADAS)
  - Fabricated at 65nm @ 83MHz
  - Part of Blackfin 60x series
    • + 2x Blackfin DSP cores

• FLP is generalization of PVP
  - Result of joint work with
    PVP chief architect (2nd author)
    • Generalized & improved
  - PVP can be considered early instance FLP
    • Use PVP for component results (due to similarity)
Flexibility: Application Mapping

- Example: Parallel Execution of:
  - High-beam / Low-beam detection
  - Canny edge detection

- Mapping Flexibility:
  - 10 ADAS apps mapped
  - Up to 2 parallel streams
Comparison Setup

1. ILP-BFDSP
   - Blackfin DSPs

2. ILP-BFDSP+HWACC
   - Blackfin DSPs
   - Custom-HW Accelerators
     - Convolution
     - Color extraction

3. FLP-PVP
   - PVP only

• VERY optimistic assumptions for ILP:
  - Fused implementation
  - Optimal pipeline utilization (*no stalls*)
  - 90% of data traffic is hidden from off-chip memory
  - Stream data ACC <-> ILP in scratch pad

• Power:
  - PVP for function blocks from fabricated chip: TSMC 65nm LP
  - Single Blackfin DSP core 280mW
Results Comparison

- **Computation:**
  - FLP-PVP \(\leq 22.5\) GOPs (HB/LB + TSR)
  - ILP+ACC requires 2 ILP cores
  - ILP requires 7 ILP cores

- **Communication:**
  - Off-chip traffic
  - FLP-PVP offer:
    - \(1/5^{th}\) of ILP
    - \(2/5^{th}\) of ILP+ACC

- **Power:**
  - FLP-PVP offer:
    - 14x-18x less than ILP
    - 5x less than ILP+ACC
Outline

• Intro: High-performance low-power computing
• Flexibility / Efficiency trade-off
• Related work
• Function-Level Programming
• Function-Level Processor (FLP)
• Experimental results
• Conclusions
• Conclusions

• Function-level Processor (FLP)
  – Raises architecture abstraction to functions
    • Targets streaming application markets (domain specific)
  – Avoids inefficiency of traditional ILP+HWACCs
    • No instruction fetch (reduced flexibility at function-level)
    • Custom traffic management (stream v.s. operational)
    • Stream traffic hidden from system fabric
    • Autonomous control, higher independence

• Research Challenges / Future Work
  – Selection of FBs & their potential composition
    • A minimal but sufficiently contiguous set of FBs
      ➢ Need for tools explore and analyze the market applications
  – Enhance from spatial multi-stream to timed multi-stream
  – Simplify FLP programming / traffic optimization
Thank you!
PVP Function Blocks

- 4x 2-D Convolution blocks (CNV),
- 1x General-purpose Arithmetic Unit (ACU),
- 1x Cartesian to polar converter (PMA),
- 1x 2-D derivations Pixel-Edge Classifier (PEC),
- 2x range reduction Threshold-Histogram-Compression (THC),
- 2x 2-D Integral Image Manipulators (IIM).
### Application Mapping Flexibility

- High/Low Bean Adjustment (HB/LB)
- Canny Edge Detection (Canny)
- Sobel Edge Detection (Sobel)
- Laplacian on Gaussian (LoG)
- Threshold Binary Edge Map (TBEM)

<table>
<thead>
<tr>
<th></th>
<th>Stage 1</th>
<th>Stage 2</th>
<th>Stage 3</th>
<th>Stage 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CNV0 CNV1 CNV2 CNV3</td>
<td>ACU PMA PEC THC0 THC1</td>
<td>IIM0 IIM1</td>
<td></td>
</tr>
<tr>
<td>HB/LB</td>
<td>Step0</td>
<td></td>
<td></td>
<td>Step1</td>
</tr>
<tr>
<td>Canny</td>
<td>Step0</td>
<td>Step1</td>
<td>Step1</td>
<td></td>
</tr>
<tr>
<td>Sobel</td>
<td>Step0</td>
<td>Step0</td>
<td></td>
<td>Step2</td>
</tr>
<tr>
<td>LoG</td>
<td>Step0</td>
<td>Step0</td>
<td>Step1</td>
<td>Step2</td>
</tr>
<tr>
<td>TBEM</td>
<td>Step0</td>
<td>Step0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LH</td>
<td>Step0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBEM</td>
<td>Step0</td>
<td>Step0</td>
<td>Step1</td>
<td>Step2</td>
</tr>
<tr>
<td>IoV</td>
<td>Step0</td>
<td></td>
<td></td>
<td>Step3</td>
</tr>
<tr>
<td>DoG</td>
<td>Step0</td>
<td>Step1</td>
<td>Step1</td>
<td>Step3</td>
</tr>
<tr>
<td>HoG</td>
<td>Step0</td>
<td>Step0</td>
<td>Step1</td>
<td>Step2</td>
</tr>
</tbody>
</table>

H. Tabkhi, R. Bushey, G. Schirner

Function-Level Processor (FLP)
Computation Demand

- VERY optimistic assumptions for ILP:
  - Fused implementation
  - Optimal pipeline utilization (no stalls)
  - 90% of data traffic is hidden from off-chip memory
  - Stream data ACC <-> ILP in scratch pad

<table>
<thead>
<tr>
<th>Applications</th>
<th>FLP-PVP FLP Oper. [GOPs/s]</th>
<th>ILP Util. [%]</th>
<th>ILP-BFDSP+ACCs ACC Oper. [GOPs/s]</th>
<th>ILP Util. [%]</th>
<th>ILP-BFDSP ACC Oper. [GOPs/s]</th>
<th>ILP Util. [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>HB/LB</td>
<td>7.387</td>
<td>0</td>
<td>4.316</td>
<td>159</td>
<td>0</td>
<td>370</td>
</tr>
<tr>
<td>TSR(Canny)</td>
<td>15.106</td>
<td>0</td>
<td>12.616</td>
<td>135</td>
<td>0</td>
<td>756</td>
</tr>
<tr>
<td>HB/LB+TSR</td>
<td>22.493</td>
<td>0</td>
<td>16.932</td>
<td>294</td>
<td>0</td>
<td>1125</td>
</tr>
<tr>
<td>Sobel</td>
<td>9.628</td>
<td>0</td>
<td>8.466</td>
<td>66</td>
<td>0</td>
<td>482</td>
</tr>
<tr>
<td>LoG</td>
<td>10.126</td>
<td>0</td>
<td>8.466</td>
<td>91</td>
<td>0</td>
<td>507</td>
</tr>
<tr>
<td>Sobel+LoG</td>
<td>19.754</td>
<td>0</td>
<td>16.932</td>
<td>157</td>
<td>0</td>
<td>988</td>
</tr>
<tr>
<td>IBEM</td>
<td>12.699</td>
<td>0</td>
<td>8.466</td>
<td>220</td>
<td>0</td>
<td>635</td>
</tr>
<tr>
<td>IoV</td>
<td>1.162</td>
<td>0</td>
<td>0.166</td>
<td>53</td>
<td>0</td>
<td>59</td>
</tr>
<tr>
<td>IBEM+IoV</td>
<td>13.861</td>
<td>0</td>
<td>8.632</td>
<td>272</td>
<td>0</td>
<td>694</td>
</tr>
<tr>
<td>TBEM</td>
<td>12.367</td>
<td>0</td>
<td>8.466</td>
<td>203</td>
<td>0</td>
<td>619</td>
</tr>
<tr>
<td>LH</td>
<td>7.387</td>
<td>0</td>
<td>4.316</td>
<td>159</td>
<td>0</td>
<td>370</td>
</tr>
<tr>
<td>TBEM+LH</td>
<td>19.754</td>
<td>0</td>
<td>12.782</td>
<td>362</td>
<td>0</td>
<td>988</td>
</tr>
<tr>
<td>DoG</td>
<td>14.608</td>
<td>0</td>
<td>12.616</td>
<td>110</td>
<td>0</td>
<td>731</td>
</tr>
<tr>
<td>HG</td>
<td>11.288</td>
<td>0</td>
<td>8.466</td>
<td>149</td>
<td>0</td>
<td>565</td>
</tr>
<tr>
<td>AVG</td>
<td>12.687</td>
<td>0</td>
<td>9.402</td>
<td>173</td>
<td>0</td>
<td>634</td>
</tr>
</tbody>
</table>

Function-Level Processor (FLP)  
H. Tabkhi, R. Bushey, G. Schirner  
30
Off-Chip Communication

Assumptions:
- 128 KB on-chip memory per HWACCs
- Required on-chip traffic:
  - 1 GB/s on average and up to 1.7 GB/s to move data between HWACCs and Blackfin cores.

<table>
<thead>
<tr>
<th>Applications</th>
<th>FLP-PVP [GB/s]</th>
<th>ILP-BFDSP+ACCs [GB/s]</th>
<th>ILP-BFDSP [GB/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>HB/LB</td>
<td>0.148</td>
<td>0.325</td>
<td>1.77</td>
</tr>
<tr>
<td>TSR(Canny)</td>
<td>0.148</td>
<td>0.413</td>
<td>2.655</td>
</tr>
<tr>
<td>HB/LB+TSR</td>
<td>0.295</td>
<td>0.59</td>
<td>4.424</td>
</tr>
<tr>
<td>Sobel</td>
<td>0.148</td>
<td>0.325</td>
<td>2.065</td>
</tr>
<tr>
<td>LoG</td>
<td>0.148</td>
<td>0.325</td>
<td>2.065</td>
</tr>
<tr>
<td>Sobel+LoG</td>
<td>0.295</td>
<td>0.502</td>
<td>4.129</td>
</tr>
<tr>
<td>IBEM</td>
<td>0.148</td>
<td>0.502</td>
<td>2.655</td>
</tr>
<tr>
<td>IoV</td>
<td>0.148</td>
<td>0.413</td>
<td>1.77</td>
</tr>
<tr>
<td>IBEM+IoV</td>
<td>0.295</td>
<td>0.767</td>
<td>4.424</td>
</tr>
<tr>
<td>TBEM</td>
<td>0.148</td>
<td>0.413</td>
<td>2.36</td>
</tr>
<tr>
<td>LH</td>
<td>0.148</td>
<td>0.325</td>
<td>1.77</td>
</tr>
<tr>
<td>TBEM+LH</td>
<td>0.295</td>
<td>0.59</td>
<td>4.129</td>
</tr>
<tr>
<td>DoG</td>
<td>0.148</td>
<td>0.413</td>
<td>2.655</td>
</tr>
<tr>
<td>HG</td>
<td>0.148</td>
<td>0.502</td>
<td>2.655</td>
</tr>
<tr>
<td>AVG</td>
<td>0.19</td>
<td>0.458</td>
<td>2.823</td>
</tr>
</tbody>
</table>
Power

- Assumptions:
  - PVP for function blocks from fabricated chip: TSMC 65nm LP
  - Single Blackfin DSP core 280mW
  - LPDDR2 memory interface: 40pJ per bit transfer [Malladi’12]
  - Off-chip memory access: 15pJ per bit for SDRAM [Keckler’11]