ARM+FPGA Heterogeneous Platform:
All Programmable SoC
All Programmable SoC
Heterogeneous Processing Models (PS and PL)

“All Programmable” refers to ability to not only program an ASSP-like processing system (PS) via software methods, but to the programmability (and re-programmability) of Co-processing Engines as hardware accelerators in Programmable Logic (PL) fabric.
Techniques Span HW, SW & System Engineers
All Programmable Abstractions

▶ Accelerates development for hardware engineers

▶ Enables software engineers to take advantage of custom hardware performance

▶ Allows systems engineers to evaluate and optimize algorithm performance in hardware & software
Design flows that let developers generate hardware IP from algorithms without RTL design experience

- Supports popular algorithm modeling environments
- Design architecture exploration
- Generates IP for the Vivado IP catalog
Vivado IP Integrator

- Accelerates HW design productivity through design reuse
  - Graphical IP assembly
  - Correct-by-construction
  - System centric

- Generates IP subsystems
  - Supports multiple plug-and-play IP formats
  - Generates software drivers and APIs

- Platform and silicon aware
  - Built in support for Xilinx development boards
Accelerating System Integration

- IP and System-centric integration with fast verification

- Platform aware
  - Abstracts board peripheral connections

- Optimized for silicon
  - IP Aware to maximize interconnect performance

Diagram:

- C, C++ or SystemC
- C Libraries
- Vivado IP Catalog - Standardized IP-XACT
- Subsystems, Xilinx, 3rd Party, User
- Vivado IP Integrator
- Development Board
HLS Overview
High-Level Synthesis

Accelerates Algorithmic C to Co-Processing Accelerator Integration
Design Decisions

Decisions made: by Designer

- **Functionality**
  - As implicit state machine

- **Performance**
  - Latency, throughput

- **Interfaces**

- **Storage architecture**
  - Memories, registers banks etc...

- **Partitioning into modules**

- **Design Exploration**

Decisions made: HLS Tool

- **State machine**
  - Structure, encoding

- **Pipelining**
  - Pipeline registers allocation

- **Scheduling**
  - Memory I/O
  - Interface I/O
  - Functional operations
Vivado HLS – Interfaces

Vivado™ HLS

C based code

Directives / Pragmas

• AXI
• FIFO Interface
• BRAM Interface
• Default HLS protocols
• Handshake Interface

Interfaceable IP

Abstract, untimed

Ready to use IP block

Accelerates Algorithmic C to RTL Creation
Vivado HLS – Synthesis

- **Directives / Pragmas**
- **Constraints**
- **Libraries**
  - Arbitrary Precision
  - Video
  - Math
  - Linear algebra
  - IP: FFT and FIR

C, C++, SystemC
OpenCL C

Vivado™ HLS

**FSM**

**Datapath**
- MUL
- ADD
- DIV
- FPU

Interfaceable IP / Verified RTL

Abstract, untimed

Target optimized, timed, Connectivity ready

Accelerates Algorithmic C to RTL Creation
Vivado HLS – Verification

- C, C++ Testbench and C, C++, SystemC
- C simulation:
  - GCC
  - G++
  - SystemC
- C / RTL Co-simulation:
  - Xsim
  - ISim
  - Questa SIM
  - VCS
  - NCSim
  - Riviera
  - OSCI

Single testbench for C-Sim and RTL-Sim

Creating a testbench is highly recommended!

Interfaceable IP / Verified RTL

Abstract, untimed testbench

Automatic generation of RTL testbench

Accelerates Algorithmic C to RTL Creation
The HLS Productivity Boost...

» Quick algorithmic convergence
  - Fast simulation and synthesis
  - Automatic generation of RTL testbench

» Optimized RTL Output
  - Code is architecture-aware

» Interfaces
  - Included in generated RTL

• Long iterations in RTL
• Exploration is hard

• Fast iterations...
• Easy exploration
• RTL is verified

Faster C-based Acceleration Verification
Vivado HLS System IP Integration Flow

**C-based IP Creation**
- C, C++, SystemC
- Libraries
  - Arbitrary Precision
  - Video
  - Math
  - Linear algebra
  - IP: FFT and FIR
- VHDL or Verilog

**System Integration**
- Vivado IP Integrator
- Vivado RTL
- System Generator for DSP

**Vivado® HLS Integrates into System Flows**
HLS Accelerates Verification Productivity

Optical flow video example

<table>
<thead>
<tr>
<th>Input</th>
<th>RTL Simulation Time</th>
<th>C Simulation Time</th>
<th>Acceleration</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 frames of video data</td>
<td>~2 days</td>
<td>10 seconds</td>
<td>~12,000X</td>
</tr>
</tbody>
</table>

>100x Faster C-based Acceleration Verification
Heterogeneous Platform Partitioning Use Case:
Co-processing Acceleration in Imaging Systems
Trend towards “at-the-edge” image processing
– Applicable to wide range of applications
– e.g., Smart cameras, in vehicle, gimbal, or telescoping robotic arm

Specific requirement for porting customer algorithm
– from OpenCV algorithm to embedded platform
– e.g. **Edge detection for object recognition** and positional tracking

Processes video from a miniature GigE Vision camera

High-performance required for low-latency refresh
Edge Detection Fundamentals: Canny Algorithm

1. Gaussian Blur to perform Noise Filtering
   - Utilize 2D Filters to smooth the image

2. Sobel Filter to determine Intensity Gradient of Image
   - Convolution masks for horizontal and vertical edges

3. Gradient Magnitude and Direction
   - Determine gradient magnitude & angle
     at each pixel location

4. Candidate Edge Selection
   - Suppress pixels not considered as part of an edge
     NMS: Non-maximum suppression
   - Keep thin line candidate edges

5. Statistical Analysis - Hysteresis
   - Computing the hysteresis thresholds based on the histogram of the magnitudes of
     the gradients of the entire image.
   - Performing hysteresis thresholding to determine output edge map
Algorithm Partitioning

- Dual-core processors:
  - Core 0: Object Recognition Algorithm
  - Core 1: Command & control:
    - System management

- Custom Linux AXI drivers & APIs
  - move data from PL to PS

- Algorithm re-complied
  - from OpenCV to ARM

- Portions of algorithm ported
  - use NEON extensions for additional acceleration

- Data flow management
  - Handled by the programmable logic
HLS Case Study:
Heterogeneous Processing
All Programmable SoC

ARM Cortex A9 + FPGA
Case Study: Canny Edge Detect Algorithm

GigE Vision Video Input
720p @ 30 Hz

HDMI Video Output

AXI

Horizontal Edge Detect

Vertical Edge Detect

Gaussian Noise Reduction

Edge Direction & Mag.

Non-Maxima Suppress.

Hysteresis Thresholding

Histogram / Compute Threshold

PL

Candidate Edges

Magnitude

Direction

Edge Map

PS

AXI

2.5 FPS
Case Study: Canny Edge Detect using HLS

Observations

– Canny edge detection algorithm is computationally intensive
– C-code port to ARM® CPU yields 2 fps → System Requirement of 30 fps

Actions

– Need to accelerate algorithm
– Partition functions use Processing System & Programmable Logic
– Keep decision-making/statistical functions on the CPU (PS)
– Move remaining functions to Programmable Logic (PL)

Use Vivado High-Level Synthesis (HLS)

– Reduce development time
Canny Edge Detect: Partitioning Approach

- **Re-structure code**
  - Using Vivado HLS coding methodologies

- **Verify HLS code operation**
  - Execute on CPU

- **Synthesize C-code blocks**
  - Use Vivado HLS - connect blocks to design using Vivado IPI

- **Modify original source**
  - Use accelerated output versus raw video input

- **Verify operation and results**
  - Make improvements as needed
Partitioning Advantages

CPU Offload
- Offloaded Functions operation in // with ARM
- Frees up processor for additional compute capability

Optimized partitioning
- Easily add filtering / Image enhancement coprocessing engines

Primary Edge Detect Functions Offloaded to FPGA
1. Gaussian Blur to perform Noise Filtering
2. Sobel Filter to determine Intensity Gradient of Image
3. Gradient Magnitude and Direction

Statistical Functions running on the ARM Processor
4. Edge Selection
5. Hysteresis Thresholding
Canny Edge Detect with Acceleration

Gaussian Noise Reduction

Horizontal Edge Detect

Edge Direction & Mag.

Non-Maxima Suppress.

Edge Map

Histogram / Compute Threshold

Hysteresis Thresholding

Candidate Edges

Need to Achieve 30 FPS

22 FPS

IMPROVE: Accelerate, Reduce Bandwidth

To HDMI

From GigE Vision

PL

PS

AXI

AXI

AXI

Thresholds

Direction

Magnitude

22 FPS

IMPROVE: Accelerate, Reduce Bandwidth
Canny Edge Acceleration Improved

From GigE Vision

Gaussian Noise Reduction

Horizontal Edge Detect

Vertical Edge Detect

Edge Direction & Mag.

Non-Maxima Suppress.

Histogram

Compute Threshold

Hysteresis Threshold-ing

Candidate Edges

Histogram

Direction

Magnitude

To HDMI

PL

PS

30 FPS
Alternative AP SoC Programming & Partitioning Workflows
MathWorks Guided Workflow for AP SoC

- Single system modeling environment from requirements to rapid prototype
- Guided workflow for hardware & software development
  - Programmable Logic bitstream generation
  - Software build file generation
- HW/SW Partitioning and Code Generation
- Automated AP SOC system assembly and implementation
- Evaluate system performance
  - Repartition the system if needed without the need to hand code C or HDL
High-Level AP SoC Design Work Flow

User defines partitioning

MathWorks automates code and interface-model generation

MathWorks automates the build and download through the Xilinx tools

RESEARCH > REQUIREMENTS

DESIGN

Top-Level System Model
Software Model
Hardware Model

IMPLEMENTATION

Embedded Coder®
C-Code

HDL Coder™
HDL Code

Zynq Template
Xilinx Embedded System Integration

Real-Time Parameter Tuning and Verification

MathWorks automates the build and download through the Xilinx tools

MathWorks automates code and interface-model generation

User defines partitioning
Automatic C and HDL code generation from the same graphical syntax in LabVIEW IDE

Automatically generates a hardware implementation to meet requirements, abstracting Xilinx tool flow

Comprehensive software, hardware and I/O platform for creating control and monitoring systems

LabVIEW FPGA IP Builder leveraging Vivado HLS

– NI CompactRIO-9068 architecture for Zynq® All Programmable SoC.
– “The most flexible CompactRIO ever”
Increasing Productivity for All Programmable SoCs

- Abstraction
  - Software Based Programming: C/C++ and OpenCL
  - System Level Programming: MathWorks and NI
  - IP/C Based HW Programming: Vivado IP Integrator & HLS

- Automation
  - Ease of Programming
  - Software & IP Reuse
  - Faster Time to Market
Programming Techniques for AP SoC Partitioning Benefit HW, SW & System Designers

➤ Design highly optimized embedded processing systems
  – Increase overall system performance
  – Leverage co-processor offload
  – Optimize functional partitioning

➤ Achieve Significant Reduction in Development time
  – Rapidly evaluate partitioning tradeoffs with HLS
  – Move from algorithm development → simulation → testing on real hardware
  – Reduce overall system design time & hardware development cycles

➤ Differentiate with reconfigurable programmable logic
  – Flexibility to interchange algorithms from SW to programmable logic
  – At time of configuration and during run time operation

Multiple WorkFlows Supported – Vivado HLS, Matlab, Labview
Backup
Datapath Synthesis

Example: \( y = a \times x + b + c; \)

1. HLS begins by extracting a data flow graph (DFG), a functional representation

2. Accounts for target Fmax to determine minimum required pipelining (not yet the optimal implementation)

3. Expression balancing for latency reduction

4. Restructuring to optimize fabric resources

5. Restructuring for optimized DSP48

Structured for DSP inference or leveraging floating point IPs
C function arguments become RTL interface ports

```c
f(int in[20], int out[20]) {
    int a, b, c, x, y;
    for(int i = 0; i < 20; i++) {
        x = in[i];
        y = a*x + b + c; out[i] = y;
    }
}
```

The State Machine Automatically Adapts to the Design Interface
Vivado High-Level Synthesis: Accelerated IP Development and Design Space Exploration

» C libraries:
  • Arbitrary precision
  • Floating-point math.h
  • OpenCV video functions

» Accelerated verification
  – >100X faster than RTL design

» Fast compilation and design exploration
  – Algorithm feasibility
  – Architecture Iteration

Customer proven results