Impulse C-to-FPGA Workshop

Software-to-FPGA Solutions for an Accelerated World

Presented by David Pellerin, Founder and CTO
Impulse Accelerated Technologies

www.ImpulseAccelerated.com
Training Agenda

- FPGA Tools History
- FPGA-Based Computing
- Introduction to Impulse C
- FPGA Platform Overview
- C Programming Methods and Models
- Demonstrations and Explorations
Programmable Logic Tools

A brief history
Early Programmable Logic Devices

- FPLAs, PALs, GALS, complex PLDs
- Sum-of-products architecture for Boolean functions
- Used to integrate TTL functions for reduced chip counts
- Also capable of implementing complex logic functions
  - State machines
  - Decoders
  - CRC
  - Etc.
Devices are programmed by preserving connections in a logic array
- Via fusible links or via EPROM technology

Sum-of-products logic array

D-type register
Tools for PLD Design

- **PALASM**
  - Sum-of-product equations
  - Logic validation using test vectors

- **ABEL and CUPL**
  - More flexible Boolean equation languages
  - State machine languages
  - More compiler-like optimization capabilities

- **Other tools**
  - Including graphical block diagrams and state machines
MODULE PWM_Generator
  TITLE '8-bit PWM Generator'
  clk PIN;
  d7..d0 PIN; " Inputs
  duty = [d7..d0];
  pwm_out PIN ISTYPE 'reg, buffer'; " Output

  " Internal nodes
  c7..c0 NODE ISTYPE 'reg, buffer';
  count = [c7..c0];

  EQUATIONS
  pwm_out.clk = clk;
  count.clk = clk;
  count := count + 1; " Counter runs continuously
  WHEN (pwm_out == 0) THEN
    pwm_out := (count == 0) & (duty != 0);
  ELSE WHEN (pwm_out == 1) THEN
    pwm_out := !((count == duty) & (duty != 255));
  END
MODULE SequenceDetect
TITLE 'Sequence Detector for pattern 1011'

x, clk, rst PIN;
z PIN ISTYPE 'com';
q1, q0 PIN ISTYPE 'reg';

SREG = [q1,q0]; " State registers 
S0 = [0,0]; S1 = [0,1]; S2 = [1,0]; S3 = [1,1];

EQUATIONS
[q1,q0].ar = rst;
[q1,q0].clk =clk;

STATE_DIAGRAM SREG
STATE S0: IF x THEN S1 WITH z=0; ELSE S0;
STATE S1: IF x THEN S1 WITH z=0; ELSE S2;
STATE S2: IF x THEN S3 WITH z=0; ELSE S0;
STATE S3: IF x THEN S1 WITH z=1; ELSE S2;

END
Field Programmable Gate Arrays

- Completely different structure than simple PLDs
  - More complex, higher density
  - More registers
  - More flexible interconnects

- Logic synthesis and HDLs become important
  - VHDL and Verilog are the hardware languages of choice for FPGAs

- Synthesis tools raise the level of abstraction
  - Applications start to look more like software
A higher-level language, but still requires deep hardware knowledge

- VHDL descriptions are register transfer logic (RTL), not software abstractions

```vhdl
Library IEEE; use IEEE.std_logic_1164.all;
entity fsm is
port (Clk, Rst, A, B, C, D, E: in STD_LOGIC;
   SINGLE, MULTI, CONTIG: out STD_LOGIC);
end fsm;
architecture rtl of fsm is
type STATE_TYPE is (S1, S2, S3, S4, S5, S6, S7);
signal CS, NS: STATE_TYPE;
begn
   process (Clk, Rst)
   begin
      if (Rst='1') then
         CS <= S1;
      elsif (Clk'event and Clk = '1') then
         CS <= NS;
      end if;
   end process;
   process (CS, A, B, C, D, E)
   begin
      case CS is
      when S1 =>
         MULTI <= '0';
         CONTIG <= '0';
         SINGLE <= '0';
      if (A = '1' and B = '0' and C = '1') then
         NS <= S2;
      elsif (A = '1' and B = '1' and C = '0') then
         NS <= S4;
      end if;
   end process;
```

...
VHDL

- VHDL is an international IEEE standard language (IEEE 1076-1993) for describing digital hardware
- VHDL is an acronym for VHSIC (Very High Speed Integrated Circuit) Hardware Description Language
- VHDL enables hardware modeling from the gate to system level
- VHDL was originally design for modeling/simulation, and only later became a language for logic synthesis
RTL

- Register Transfer Logic
- VHDL (the synthesizable subset, that is) is designed specifically for RTL

If the logic has feedback, then it is a state machine

```vhdl
process (Clk, Rst)
begin
    if (Rst='1') then
        Output_Data <= '0';
    elsif (Clk'event and Clk = '1') then
        Output_Data <= some_function;
    end if;
end process;
```
VHDL Process Statements

- Basic granularity of concurrency is the *process*
- Multiple processes are executed concurrently
- Within the process, statements execute sequentially
- In most synthesizable VHDL processes, there are clocks and resets that trigger each iteration:

```vhdl
ARCHITECTURE statemachine OF myproc IS
BEGIN
  fsm_proc : PROCESS (rst, clk)
  BEGIN
    IF (rst = '1') THEN
      z <= 0;
    ELSIF rising_edge(clk) THEN
      z <= y;
    END IF;
  END PROCESS fsm_proc;
END sequential;
```
VHDL Summary

- Provides tight control over hardware implementation
  - Down to the level of registers, the input-forming logic and the I/O

- Designs can be verified using HDL simulators
  - The HDL can be a test language as well as a design language

- Very little uncertainty about synthesis results
  - Although there are some things to watch for (latches, multiplexer chains, etc)
Drawbacks to HDLs?

- Not a very productive way to describe and debug complex systems
  - Hand-optimizing logic can take many hours, even for simple designs
  - Debugging hardware timing issues can be painful

- Designs may not be portable between different FPGA types
  - Portable designs may be non-optimal
  - Optimized designs may require specialized skills for the selected FPGA

- It may be difficult to validate correctness of results against an original software model
  - Writing HDL test benches may take more time than writing the HDL description
FPGA-Based Computing

Featuring Impulse C
What drives demand for FPGA-accelerated solutions?
- EFFICIENCY – More computation with less power consumption
- FPGA-based computing offers the best ratio of GFLOPS/watt

What has limited the use of reconfigurable computing?
- PLATFORM INSTABILITY – Too many choices, too much risk
- LOW PRODUCTIVITY – Historic difficulty of programming
- UNCERTAINTY – Too much changing, too fast

The solution?
- PLATFORM PORTABILITY – Insulate the programming from the hardware
- HIGHER LEVEL PROGRAMMING TOOLS – Universal, familiar, accessible
Why C-to-FPGA Tools?

Moving C applications to FPGAs

- Image processing and DSP
- Embedded processing acceleration
- Desktop/server applications
- Financial processing
- Etc

Software Coprocessing

Systems on FPGA

hardware accelerated processing from C-language

Increasing FPGA Complexity...

Glue Logic

Control Logic

DSP


Moving C applications to FPGAs

- Image processing and DSP
- Embedded processing acceleration
- Desktop/server applications
- Financial processing
- Etc
Hardware Design is a Bottleneck

Problem 1: HW/SW interfaces must be frozen before hardware design can begin.

Problem 2: Hardware development is a bottleneck.

System modeling

Design partitioning (HW/SW hand-off)

Software development (embedded S/W tools)

Hardware development (HDL tools)

Validation (S/W and H/W tools)

High-level specification

Software complete

Prototype complete

Product complete

Prototype complete
Hardware/Software Co-Design

The solution:
Provide software-to-hardware tools along with hardware/software co-design capabilities

The result:
Faster time to a working prototype, more time available for design optimization and performance improvements.

System modeling

Design partitioning

Software development (embedded S/W tools)

Hardware development (C-to-hardware and HDL tools)

Software complete

Design refinement and testing (S/W and H/W tools)

Prototype complete

High-level specification
Uses for C-to-FPGA

1. **C-to-HDL Hardware Module**
   - Create a hardware module

2. **Embedded CPU Core**
   - Accelerate an embedded CPU

3. **Application Hardware Accelerator**
   - Accelerate an external/host CPU
The FPGA as an Accelerator

- Compile C code into a processor-attached accelerator

- Processor may be embedded within the FPGA, or external
  - PowerPC
  - MicroBlaze
  - NIOS II
  - Intel
  - AMD

*Note: a processor is not required to use Impulse C*
A Wide Variety of Applications

*Commercial, defense and scientific applications...*
Accelerated Image Enhancement

FPGA with multiple scalable processes

void edge_filter(co_stream pixels_in, co_stream pixels_out) {
    float nPixel;
    ...
    do {
        co_stream_open(pixels_in, O_RDONLY, FLOAT_TYPE);
        co_stream_open(pixels_out, O_WRONLY, FLOAT_TYPE);
        while ( co_stream_read(pixels_in, &nPixel, sizeof(float)) == 0 ) {
            ...
            co_stream_write(pixels_out, &nPixel, sizeof(float));
        }
    } while(1);
}
Accelerated Financial Simulation

Simulation Parameters
- time to maturity
- strike price
- risk-free rate
- current underlying price
- etc.

Multiple pipelined processes are scaled to take advantage of available FPGA resources.

```c
void blackscholes(co_stream params_in, co_stream rnd_in, co_stream st_out){
...
  for(i=0; i < m; i++) {
    // Init stock prices to be log of the price (Geometric mean)
    IS1 = log(S);  IS2 = IS1;
    for(j=0; j < n; j++) {
      #pragma CO pipeline
      if(co_stream_read(rnd_in, &u1, sizeof(co_uint32) ) != co_err_none) break;
      randnZ = u2n(u1);
      // Simulate the stock paths. using Weiner process
      IS1 = IS1 + uudt + vdt * randnZ;
      IS2 = IS2 + uudt + vdt*(-randnZ);  // antithetic path and variance reduction
    }
  }
```
What is Impulse C?

- **Not a new language!**
- **ANSI C for FPGA programming**
  - For embedded and HPC applications
  - Supports standard C development tools
  - Supports multi-process partitioning
  - Used with or without an embedded or host processor
- **A software-to-hardware compiler**
  - Optimizes C code for parallelism
  - Generates HDL, ready for FPGA synthesis
  - Also generates hardware/software interfaces
- **Purpose**
  - Describe hardware accelerators using C
  - Move compute-intensive functions to FPGAs

---

www.ImpulseC.com
Impulse C Design Philosophy

- Give C programmers access to FPGAs
- Allow the use of standard C tools, including debuggers
- Allow for the easy creation and control of parallelism at all levels:
  - At the level of individual C-statements
  - For larger blocks of statements
  - Within loops, including pipelined loops
  - Allow the use of communicating H/W processes
  - Support hardware and software partitioning
- Support FPGA-based platforms
  - Not just the FPGA
  - Allow the compiler to be extended via scripting
Impulse C Goals

Use Impulse C to create highly parallel, dataflow-oriented applications.

Ideal for DSP, image processing and many other application domains.

Use with or without an embedded processor.
A Software-to-FPGA Compiler

- Familiar compiler design flow
  - C pre-processor (gcc)
  - C parser/compiler
    - Including common compiler optimizations
    - Common sub-expressions, constant folding, etc.
- Parallel optimizer
  - Includes instruction scheduling, pipelining, unrolling
- Hardware generator
  - Generates VHDL or Verilog
- Scriptable, extendable Platform Support Packages
  - For FPGA-based platforms

- Fully compatible with standard C development tools
  - C source-level debuggers for testing and verification
  - Profilers, configuration management tools, etc.
  - Does not significantly change your existing design flow
Why Use Impulse C?

Reduced application development times
- Faster, more agile application development
- Faster time-to-prototype and reduced risk
- More opportunity for design optimization and experimentation

Reduced project costs
- Reduce or eliminate costly, high-risk hardware design phases
- Get your prototypes working faster in FPGA hardware

Example: image processing for defense/aerospace
- Advanced, embedded image processing algorithm for machine vision
- Customer saved an estimated three developer-months of effort
- Customer was able to try applications never before considered for an FPGA
FPGA-Based Platforms

For embedded and HPC
FPGA-Based Platforms

What defines an FPGA-based platform?

- **TYPE OF FPGA**
  - What resources are available for computation?
- **HOST PROCESSOR**
  - Embedded within the FPGA, discrete, or none at all?
- **BUS ARCHITECTURE**
  - How is the FPGA connected to the processor?
- **OTHER PERIPHERALS**
  - What devices will the FPGA application be communicating with?
- **OPERATING SYSTEM**
  - For the host processor?

How many types of platforms are there?

- **UNLIMITED!**
  - For embedded computing
  - For desktop prototyping
  - For high performance computing
  - Customized for specific applications

www.ImpulseC.com
Sample FPGA Platforms

[Images of various FPGA platforms and components]
Impulse C Platform Support

Designed for hardware/software co-design
- Compatible with standard development tools including Visual Studio and GCC/GDB
- Supports multi-process partitioning through C-compatible API library
- Flexible hardware generation
  - Can be extended to support a wide variety of reconfigurable platforms
  - Custom platforms can be supported – ask us for details

A software-to-hardware compiler
- Optimizes C code for parallelism
- Generates hardware/software interfaces
C Programming Methods

Impulse C Programming Models
From Software to FPGA Hardware

C-based design
- Emphasizing iterative methods of programming

Desktop simulation
- Using standard C tools

C-to-FPGA compilation
- VHDL or Verilog

Interactive optimization
- For high performance
Iterative optimization example: Multiple Sequence Alignment

<table>
<thead>
<tr>
<th>Processor/FPGA</th>
<th>Clock Rate</th>
<th>Execution time (seconds)</th>
<th>FPGA slices (Virtex-4 LX25)</th>
<th>Acceleration vs. MicroBlaze</th>
<th>Acceleration vs. 2GHz PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>MicroBlaze with on-chip BRAM</td>
<td>100MHz</td>
<td>41.3</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Pentium PC</td>
<td>2GHz</td>
<td>.454</td>
<td></td>
<td>91X</td>
<td>1</td>
</tr>
<tr>
<td>FPGA, 1 PE</td>
<td>100MHz</td>
<td>.209</td>
<td>1076 (10%)</td>
<td>198X</td>
<td>2.2X</td>
</tr>
<tr>
<td>FPGA, 2 PEs</td>
<td>100MHz</td>
<td>.145</td>
<td>1509 (14%)</td>
<td>285X</td>
<td>3.1X</td>
</tr>
<tr>
<td>FPGA, 8 PEs</td>
<td>100MHz</td>
<td>.102</td>
<td>4229 (39%)</td>
<td>405X</td>
<td>4.5X</td>
</tr>
<tr>
<td>FPGA, 2 optimized processes, each with 2 PEs</td>
<td>100MHz</td>
<td>.074</td>
<td>5118 (47%)</td>
<td>558X</td>
<td>6.1X</td>
</tr>
</tbody>
</table>
It’s All About Parallelism

Parallelism at the system level
- Multiple parallel processes
- System-level pipelining and/or co-processing as appropriate
- Hardware accelerators combined with embedded software

Parallelism at the C statement level
- Loop unrolling and pipelining
- Instruction scheduling
Communicating Processes

- Buffered communication channels to implement data streams
- Supports dataflow and message-based communications
- Supports parallelism at the application level and at the level of individual processes
Parallelism via Multiple Processes

Spatial parallelism

Temporal parallelism
(system-level pipelining)
Impulse C Streaming Process

```c
void img_proc(co_stream pixels_in, co_stream pixels_out) {
    int nPixel;
    ...
    do {
        co_stream_open(pixels_in, O_RDONLY, INT_TYPE(32));
        co_stream_open(pixels_out, O_WRONLY, INT_TYPE(32));
        while ( co_stream_read(pixels_in, &nPixel, sizeof(int)) == 0 ) {

            // Do a filtering operation here using standard C…
            ...

            co_stream_write(pixels_out, &nPixel, sizeof(int));
        }
        co_stream_close(pixels_in);
        co_stream_close(pixels_out);
    } while(1);  // Run forever
}
```

Impulse C streaming API functions compile automatically to generate processor bus Interfaces.
Impulse C API Functions

- co_memory_create
- co_memory_ptr
- co_memory_readblock
- co_memory_writeblock
- co_process_config
- co_process_create
- co_register_create
- co_register_get
- co_register_put
- co_register_read
- co_register_write
- co_semaphore_create
- co_semaphore_release
- co_semaphore_wait
- co_signal_create
- co_signal_post
- co_signal_wait
- co_stream_close
- co_stream_create
- co_stream_eos
- co_stream_open
- co_stream_read
- co_stream_read_nb
- co_stream_write
- co_stream_write_nb
- cosim_logwindow_create
- cosim_logwindow_fwrite
- cosim_logwindow_init
- cosim_logwindow_write
void img_proc(co_signal start, co_memory datamem, co_signal done) {
    double A[ARRAYSIZE];
    double B[ARRAYSIZE];
    int32 status;
    int32 offset = 0;
    ...
    do {
        co_signal_wait(start, (int32*)&status);
        co_memory_readblock(datamem, offset, A, ARRAYSIZE * sizeof(double));
        ...
        // Do some kind of computation here, perhaps calculating A[] into B[]
        ...
        co_memory_writeblock(datamem, offset, B, ARRAYSIZE * sizeof(double));
        co_signal_post(done, 0);
    } while(1);
}
An Impulse C Process

Processes are independently synchronized

Multiple methods of process-to-process communications are supported

Shared memory block reads/writes

Stream inputs

Signal inputs

Register inputs

App Monitor outputs

Stream outputs

Signal outputs

Register outputs
Accelerate Using Parallel Processes

Supports scaling in two dimensions…

Host interface

Impulse C scalable FPGA accelerator process

Temporal parallelism (system-level pipelining)

Spatial Parallelism (arrays of processes)

FPGA accelerator

Host interface

www.ImpulseC.com
Debug and Verify

- Use standard C tools and methods for application debugging
  - Source-level debuggers
  - C-language testing

- Use Impulse Application Monitor to test and analyze parallel dataflow
Compile and Optimize

- Generate FPGA hardware
- VHDL or Verilog

- Optimize the results using interactive tools
- Pipeline analysis
- Loop unrolling
- Instruction scheduling

www.ImpulseC.com
Impulse Flow

- Familiar IDE Interface

- Desktop simulation
  - Using standard C programming methods

- Dataflow analysis
  - Identify bottlenecks

- Graphical optimization analysis
  - Balance size and speed to meet performance goals
Streams provide one method of multi-process parallelism
- Hardware-to-hardware streams generated as FIFOs
- Hardware-to-software streams generated as system interfaces

Target platform determines system-level streams I/O
- Bus interface wrapper (e.g. PowerPC APU or MicroBlaze FSL)
- External port for direct hardware connection
- Other interfaces specific to an FPGA board or FPGA tool
entity img_arch is
  port (
    reset : in std_ulogic;
    sclk : in std_ulogic;
    clk : in std_ulogic;
    pixels_raw_en : in std_ulogic;
    pixels_raw_eos : in std_ulogic;
    pixels_raw_data : in std_ulogic_vector (23 downto 0);
    pixels_raw_rdy : out std_ulogic;
    pixels_filtered_en : in std_ulogic;
    pixels_filtered_data : out std_ulogic_vector (23 downto 0);
    pixels_filtered_eos : out std_ulogic;
    pixels_filtered_rdy : out std_ulogic);
end;
Process Stream Interfaces
Embedded Computing

Combine embedded processors with custom C-language accelerators

Easily partition your application between the embedded processor and FPGA logic

Verify using standard C debugging tools

Optimize your application for high performance

Example: Xilinx Virtex-5™ with MicroBlaze™ soft processor
PowerPC Embedded Computing

- YUV Color
- IDCT
- FCB_0
- APU_0
- PowerPC
- MPEG2 decoding and display
- PLB_0
- TFT
- GPIO
- INTC
- BRAM
- DDR2
- SysACE
- UART

www.ImpulseC.com
Dual PowerPC Embedded Computing

- Web server and TFT display control
  - PowerPC 405 (VxWorks) APU_1
  - Image Filter
  - YUV Color
  - IDCT
  - FCB_1
- MPEG2 decoding and filtering
  - PowerPC 405 (standalone) APU_0
- PLB_1
  - TEMAC
  - TFT
  - INTC
  - BRAM
  - DDR2
  - UART
- PLB_0
  - GPIO
  - INTC
  - BRAM
  - DDR2
  - SysACE
  - UART

Dual PowerPC Acceleration
- Embedded web server, camera interface and accelerated filtering
- Two PowerPC processors, both with Impulse C acceleration
- Featuring the Xilinx ML410 development board and Virtex-4 FX60 FPGA

www.ImpulseC.com
Multiple pipelined processes are scaled to take advantage of available FPGA resources.

```c
void blackscholes(co_stream params_in, co_stream rnd_in, co_stream st_out){
...
for(i=0; i < m; i++) { // Init stock prices to be log of the price (Geometric mean)
    lS1 = log(S);  lS2 = lS1;
    for(j=0; j < n; j++) {
        #pragma CO pipeline
        if(co_stream_read(rnd_in, &u1, sizeof(co_uint32) ) != co_err_none) break;
        randnZ = u2n(u1);
        // Simulate the stock paths. using Weiner process
        lS1 = lS1 + uudt + vdt * randnZ;
        lS2 = lS2 + uudt + vdt*(-randnZ); // antithetic path and variance reduction
    }
}
```
Demonstrations and Explorations

Impulse C Programming and Tools