

nbit_adder: adder1

GENERIC MAP (x => 8)

PORT MAP (AddSubR_n => M

multiplexer: mux2to1

GENERIC MAP (x => 7)

PORT MAP (A => Z, S => G

AddSubR_n <= (OTHERS => AddSubR_n

YOR AddSubR_n

Improving Software Performance Using Hardware Acceleration

Agenda

- What is the C-to-Hardware Acceleration (C2H) Compiler?
- How does it work?
- C2H Compiler “under the hood”
- C-based industry partnership
- Summary
- Q&A

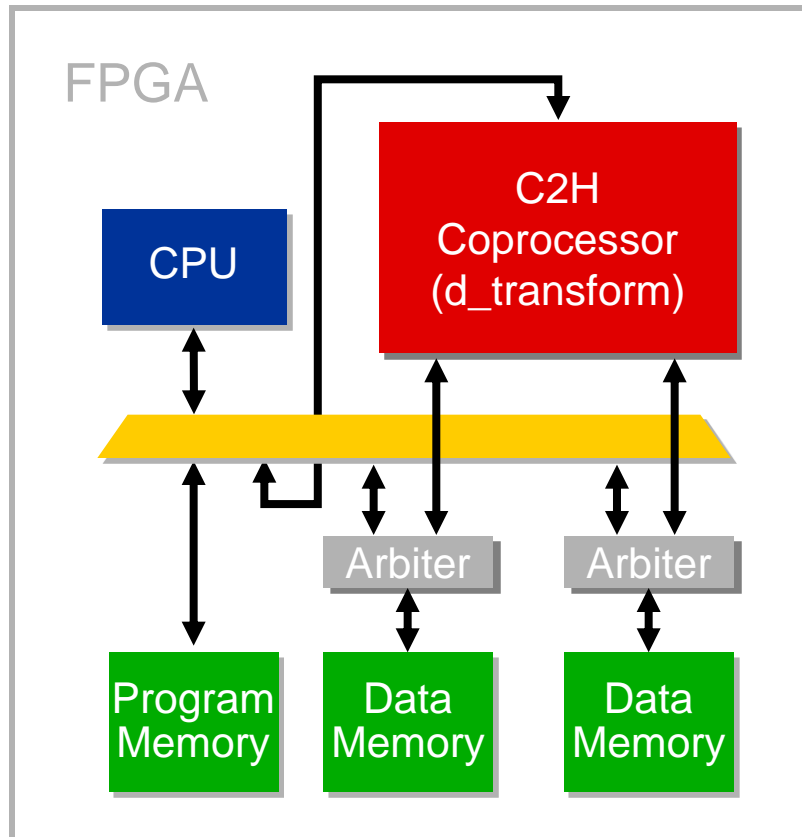
```
nbit_adder: adderx
    GENERIC MAP [ X => 8 ]
    PORT MAP ( AddSubR_n => AddSubR_n, M => M );
multiplexer: mux2to1
    GENERIC MAP [ X => 1 ]
    PORT MAP ( A1 => Z, S1 => S,
    AddSubR_n <= (OTHERS => AddSubR_n)
    M <= (OTHERS => XOR M(n-1));
```

What is the C2H Compiler?



What Is the C2H Compiler?

- A productivity tool to automatically create hardware accelerators from ANSI C code

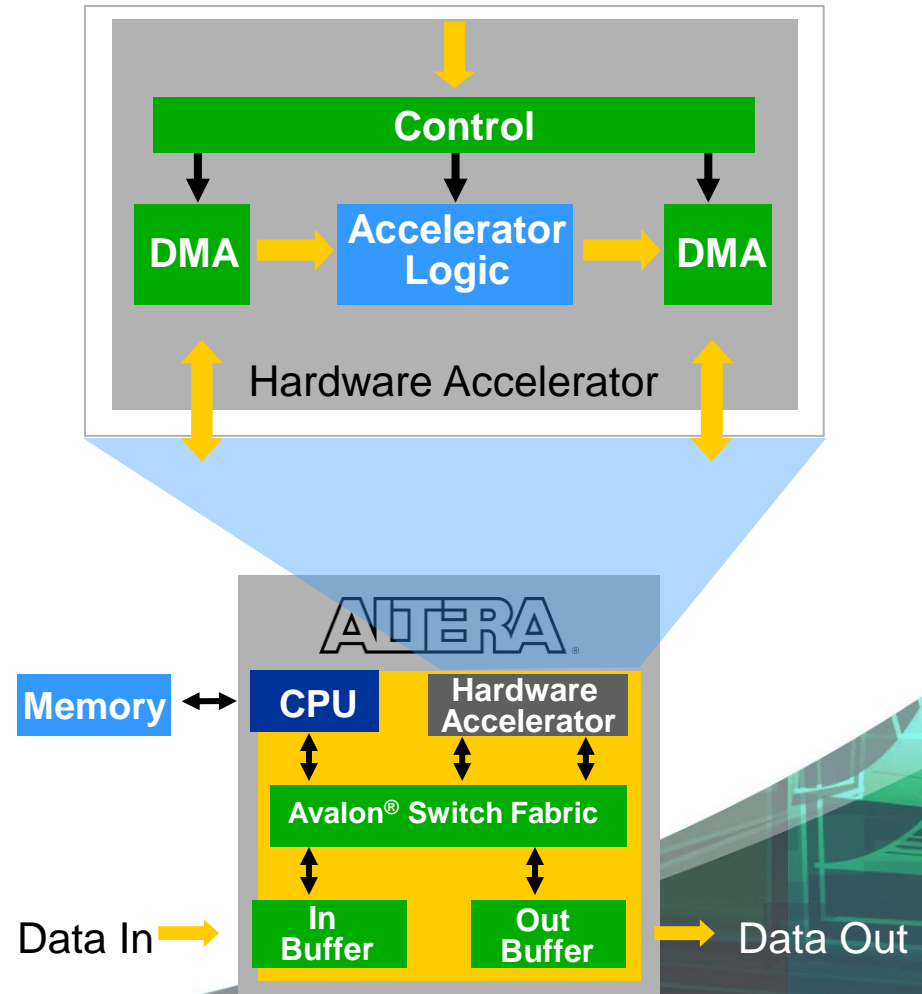


```
main ()
{ ...variable declarations...
  init();

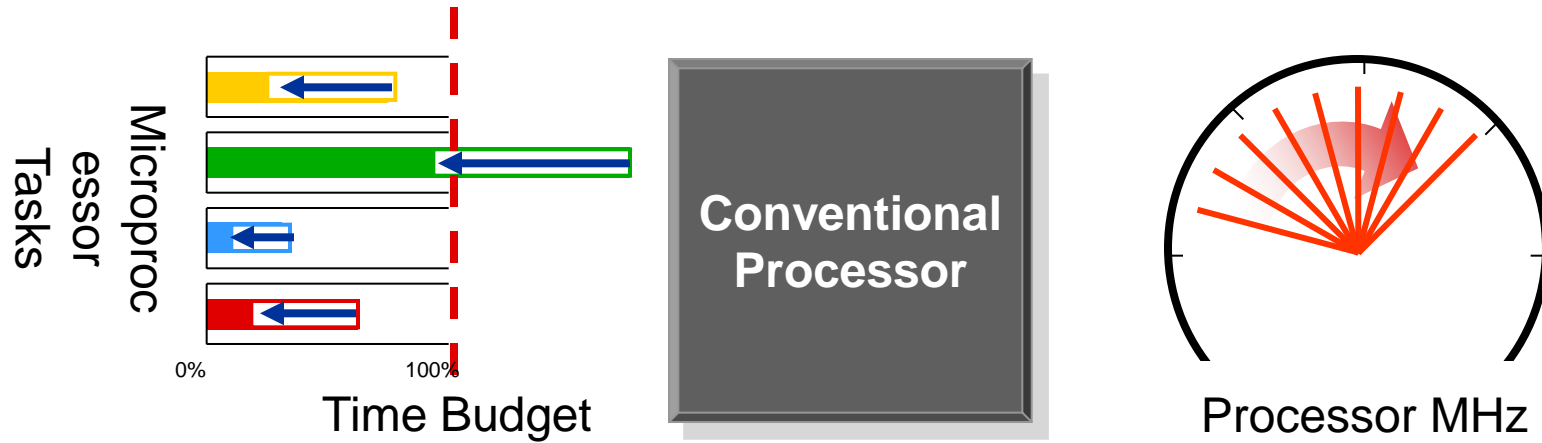
  while (!error && got_data())
  {
    do_user_interface();
    gather_statistics();
    if (got_new_data())
      d_transform(in_buf, out_buf);
    check_for_errors();
  }
  cleanup();
}
```

Nios II C2H Compiler Overview

- **Compiles** performance-critical C code into fast hardware accelerators in FPGA
- **Integrates** accelerators into Nios II system
- **Links** accelerators into Nios II software build process
- **Boosts** performance by 10X or more for many applications

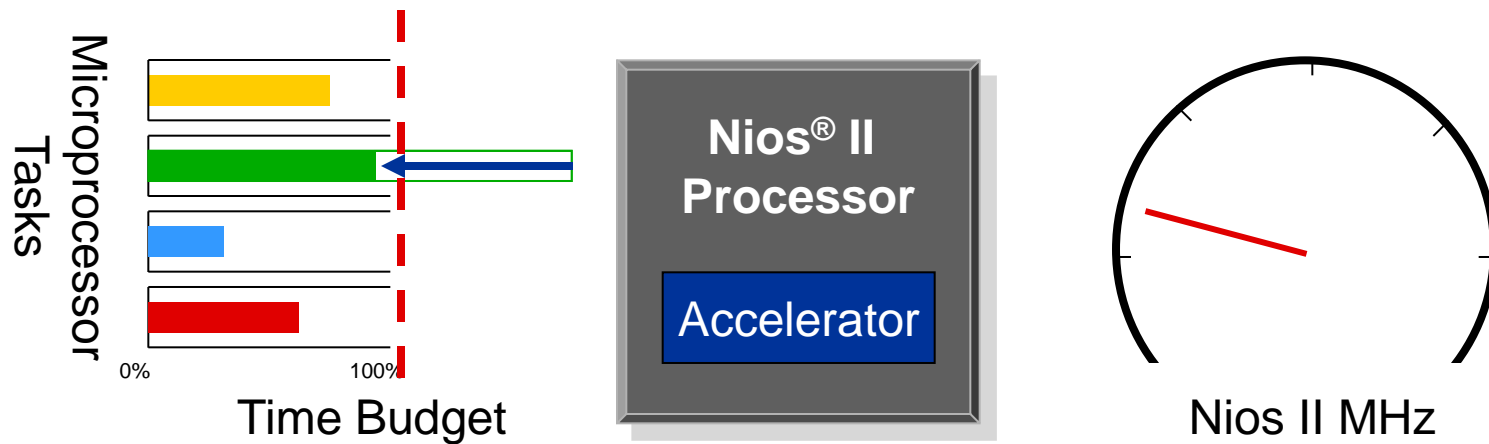


Traditional Processor Acceleration



Potential Issues With Power, Board Layout, Memory Speed, Device Cost, and Availability

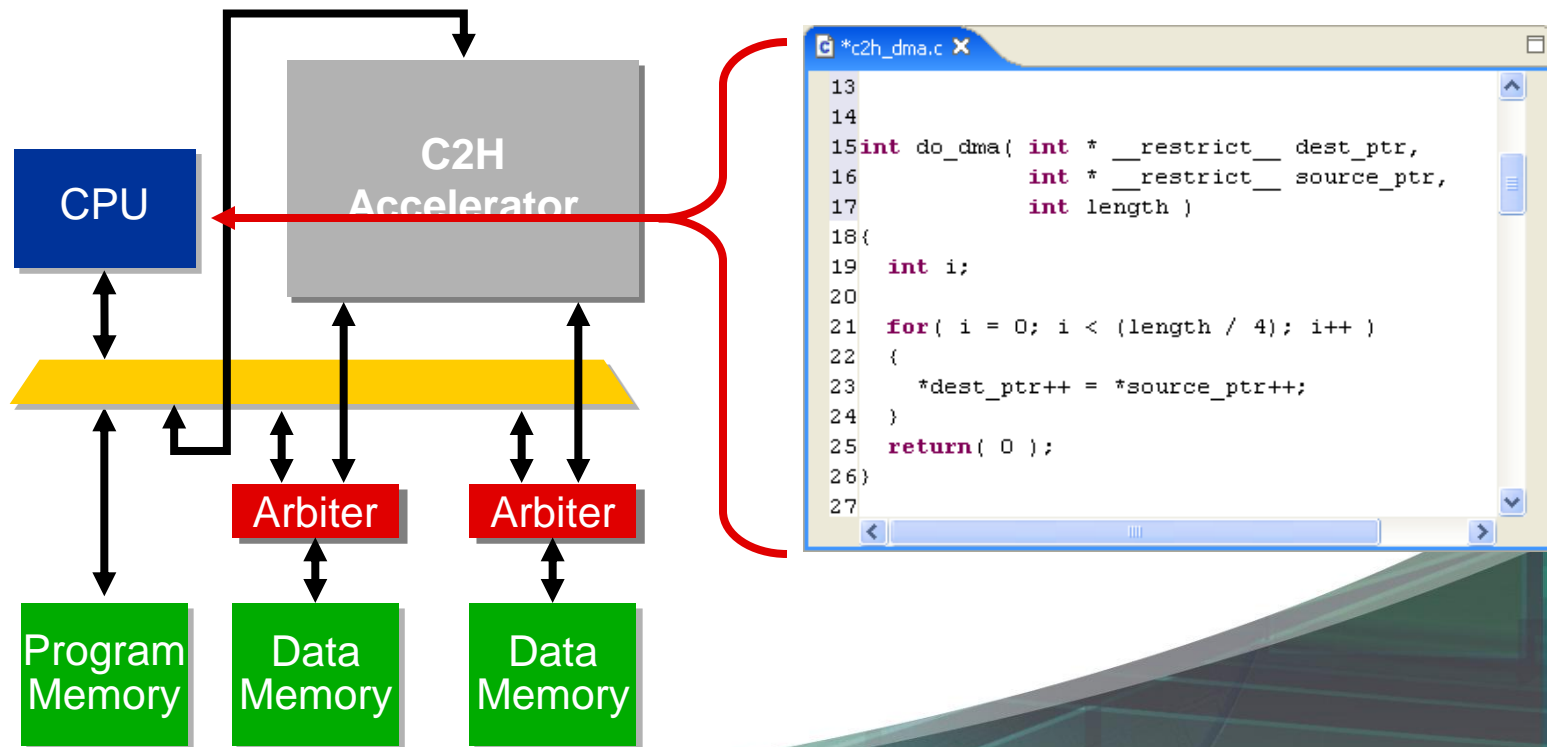
Accelerate Only What Is Needed



***Transfer Processing to Hardware
Highly Effective, Minimal Impact to System***

What Does C2H Do?

- Generates a custom hardware accelerator from an ANSI C function.



Hand-Crafted Accelerator vs. C2H Compiler

Standard Flow

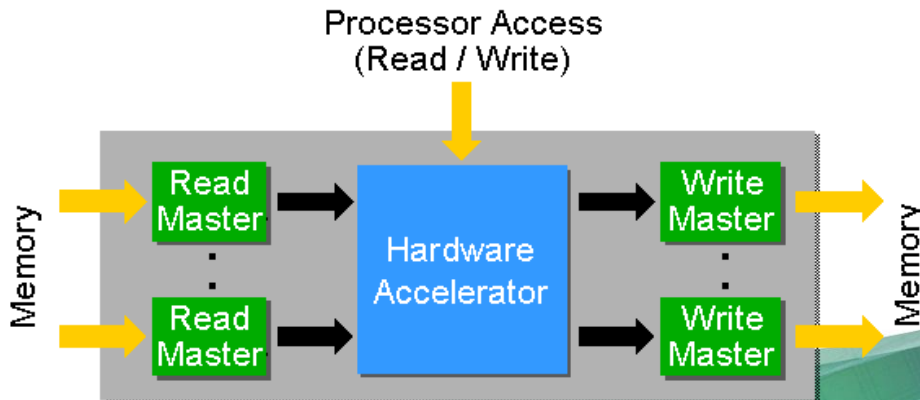
- Profile code
- Identify bottlenecks
- Repartition memory
- Create/modify an accelerator
 - HDL
 - Design
 - Simulate
 - C wrapper
 - Write function
 - Test
 - Re-build C code
 - Import into SOPC Builder
- Repeat based upon results

C2H Compiler Flow

- Profile code
- Identify bottlenecks
- Repartition memory
- Select code
 - Right-click to accelerate
 - Build
- Repeat based upon results

Anatomy of an Accelerator

- Multiple master ports to memory
 - Handles pointer dereferencing
 - Pipelined logic
 - Matched to memory latency
 - Sustain *extremely* high memory bandwidth
- Slave port for CPU control
 - Registers for parameter passing and synchronization
- Software wrapper
 - Replaces function call

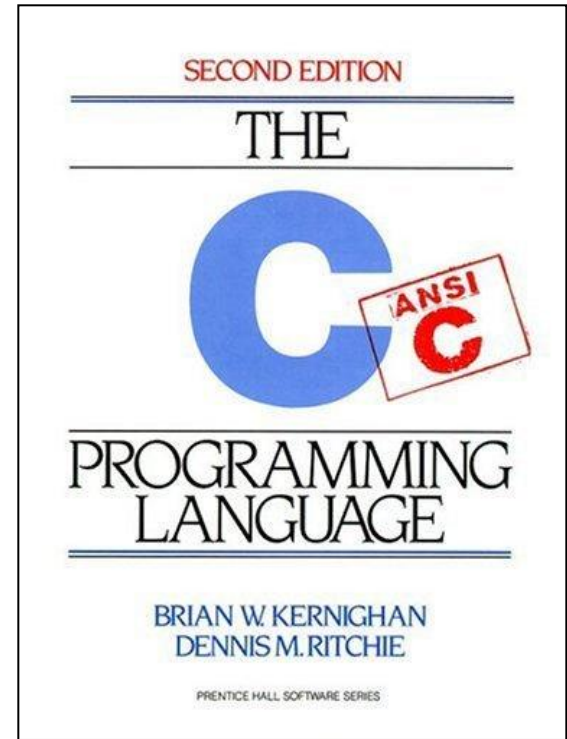


C2H Compiler Key Facts

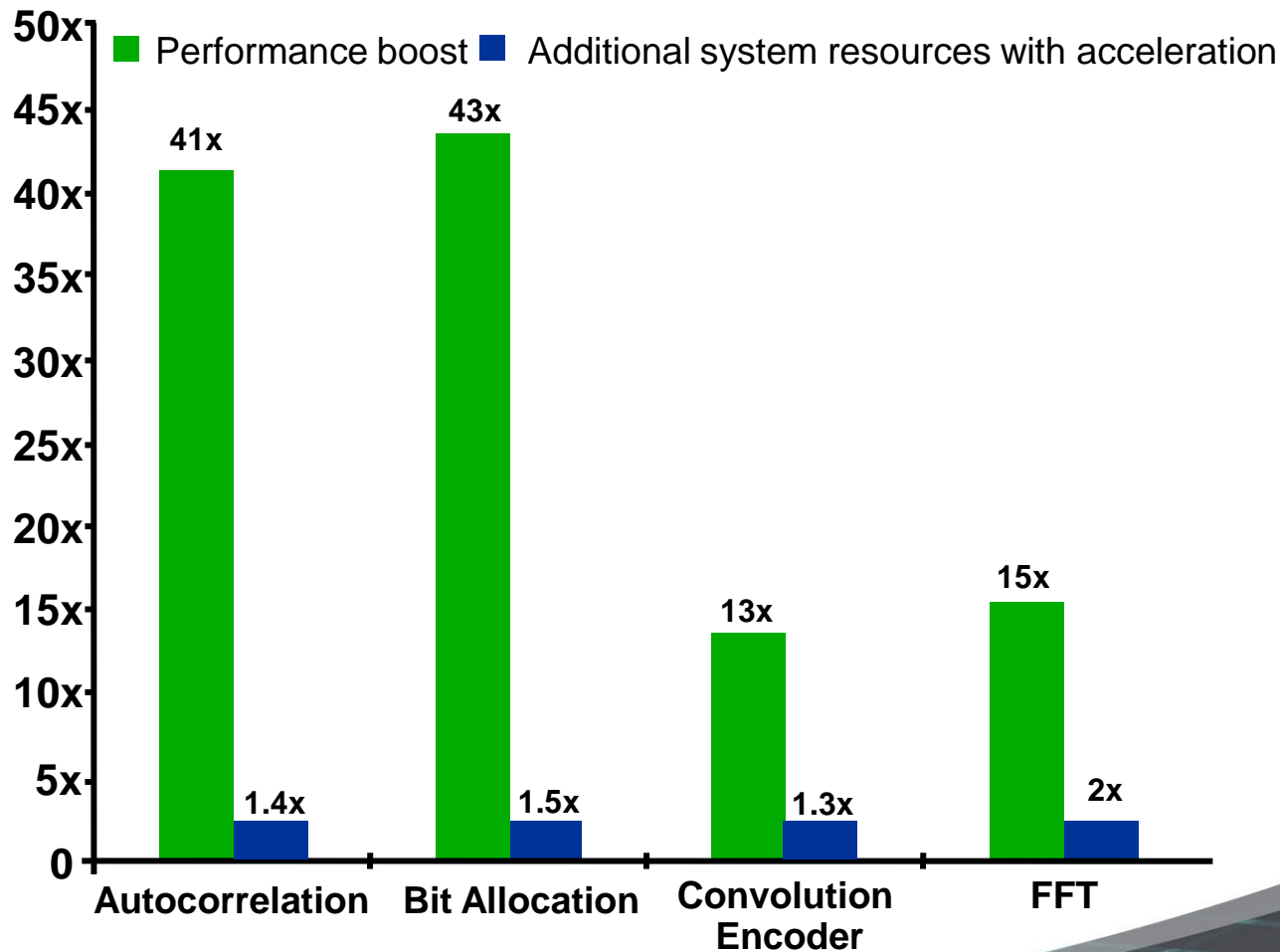
- C2H Compiler accelerates ANSI C code
 - No new language to learn
- Aimed at accelerating a critical function
 - Not whole software design
- Relies on Nios II processor
 - Built into system with SOPC Builder
- Used within Nios II Integrated Development Environment (IDE)
 - Build your application software
 - “Point and accelerate” critical functions

ANSI C Language Support

- All data types
- All operators
- All control-flow constructs
- All looping constructs
- Macros
- Function calls
- Pointer and array access

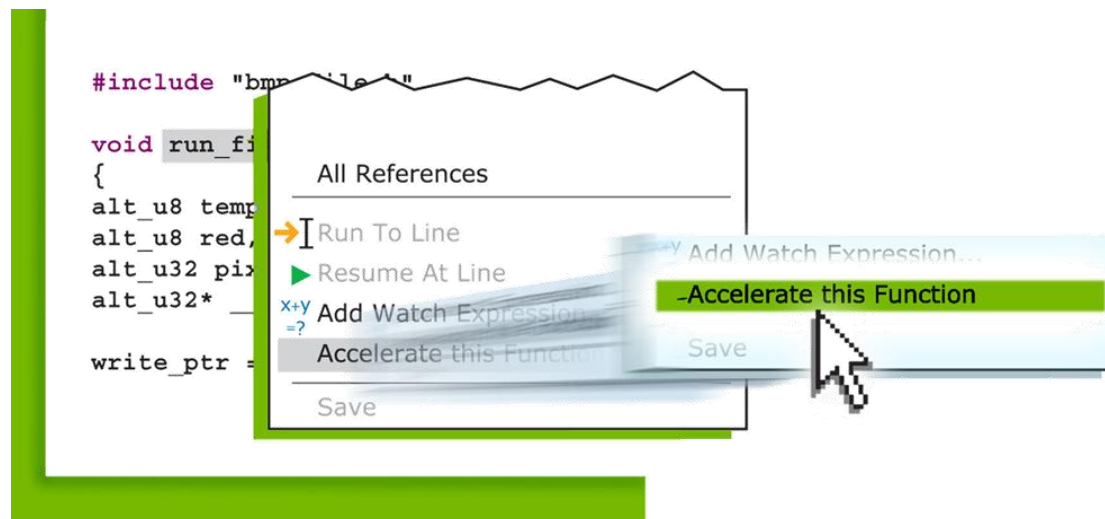


Dramatic Performance Boost



Advanced Productivity Tool

- Automates creation and integration of hardware accelerators
- Intuitive user interface streamlines C acceleration
- Fully integrated with Nios II IDE
- Tool license costs \$2,995



Right-Click to Accelerate

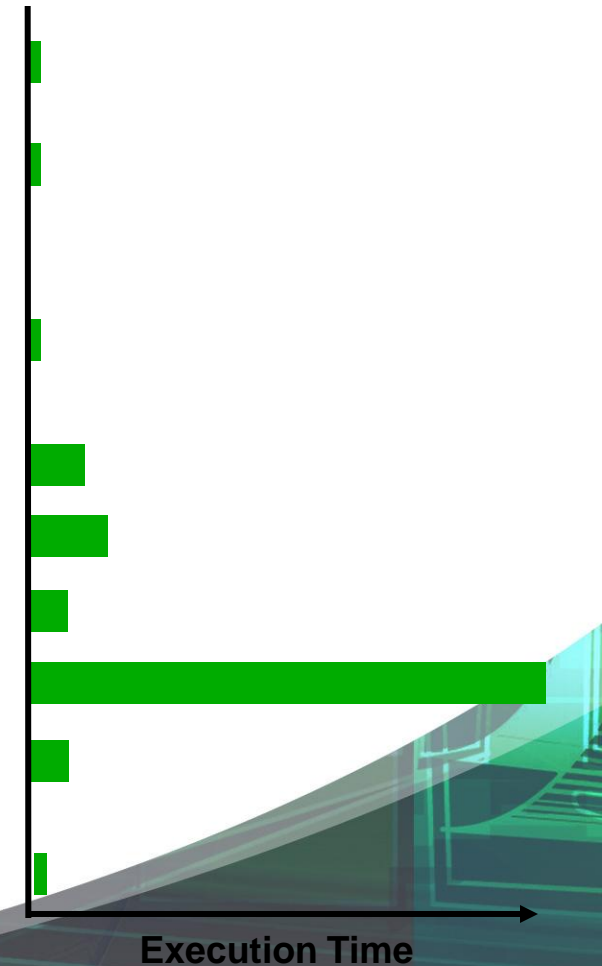
```
nbit_adder: adder1
    GENERIC MAP (K => 8)
    PORT MAP (AddSubR_n => AddSubR_n, M => M)
multiplexer: mux2to1
    GENERIC MAP (K => 8)
    PORT MAP (A1 => Z, S1 => S,
    AddSubR_n <= (OTHERS => AddSubR_n)
    M <= M XOR AddSubR_n
    carryout XOR C1 <= XOR M(0), XOR M(1), XOR M(n-1);
```

How Does It Work?

Step 1: Identify Software Bottlenecks

```
main ()
{ ...variable declarations...
  init();

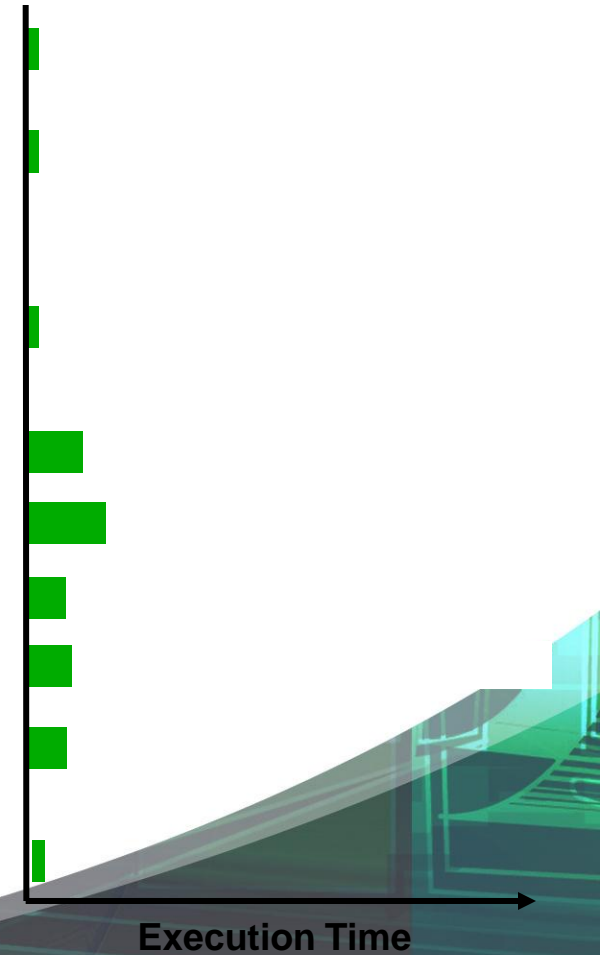
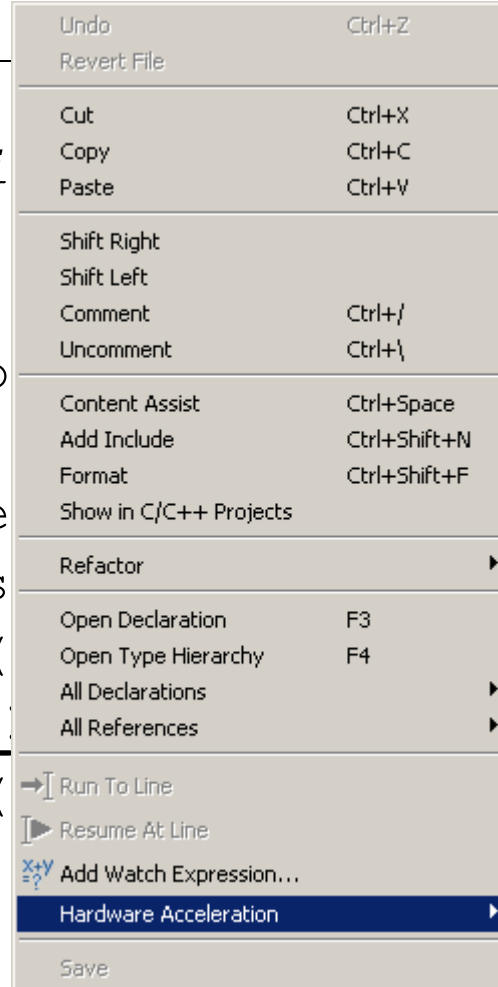
  while (!error && got_data())
  {
    do_user_interface();
    gather_statistics();
    if (got_new_data())
      d_transform(in_buf, out_buf);
    check_for_errors();
  }
  cleanup();
}
```



Step 2: Right-Click to Accelerate

```
main ()
{ ..variable declarati
  init();

  while (!error && go
  {
    do_user_interface
    gather_statistics
    if (got new data (
      d_transform (in_
    check_for_errors (
  }
  cleanup();
}
```

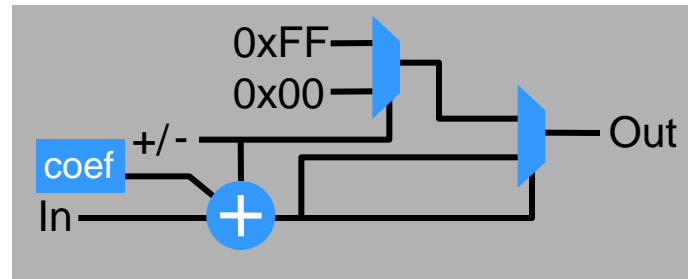


Color Filtering Demonstration

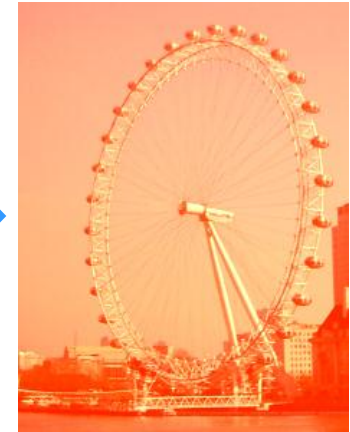
input.bmp



Color Filtering (RGB)



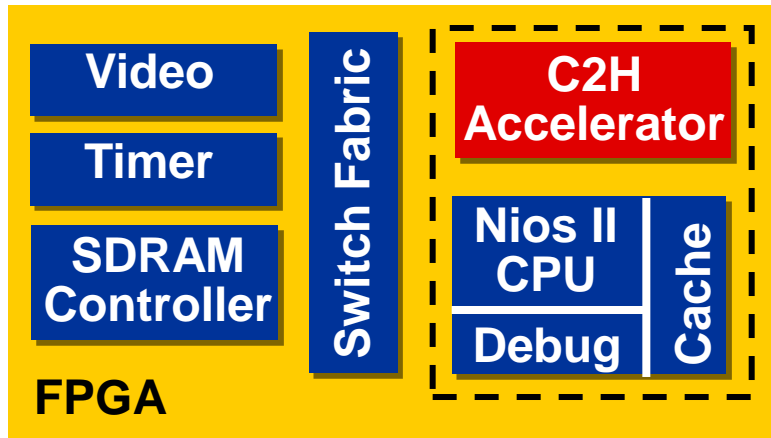
output.bmp



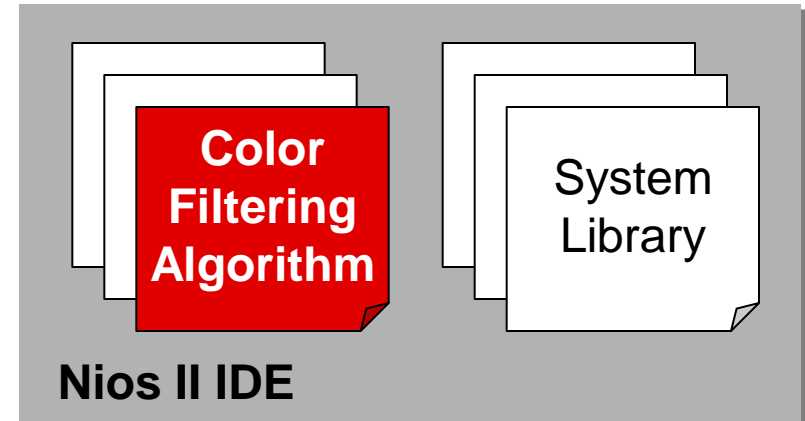
- Demo
 - Image color filtering
 - Algorithm applied to every pixel (RGB)
- Critical path
 - Color-filtering algorithm
 - Accelerated via C2H Compiler
- Timer measurement
 - Indicates time taken by software vs. C2H acceleration

Step by Step: Building the System

FPGA Hardware



Nios II Software

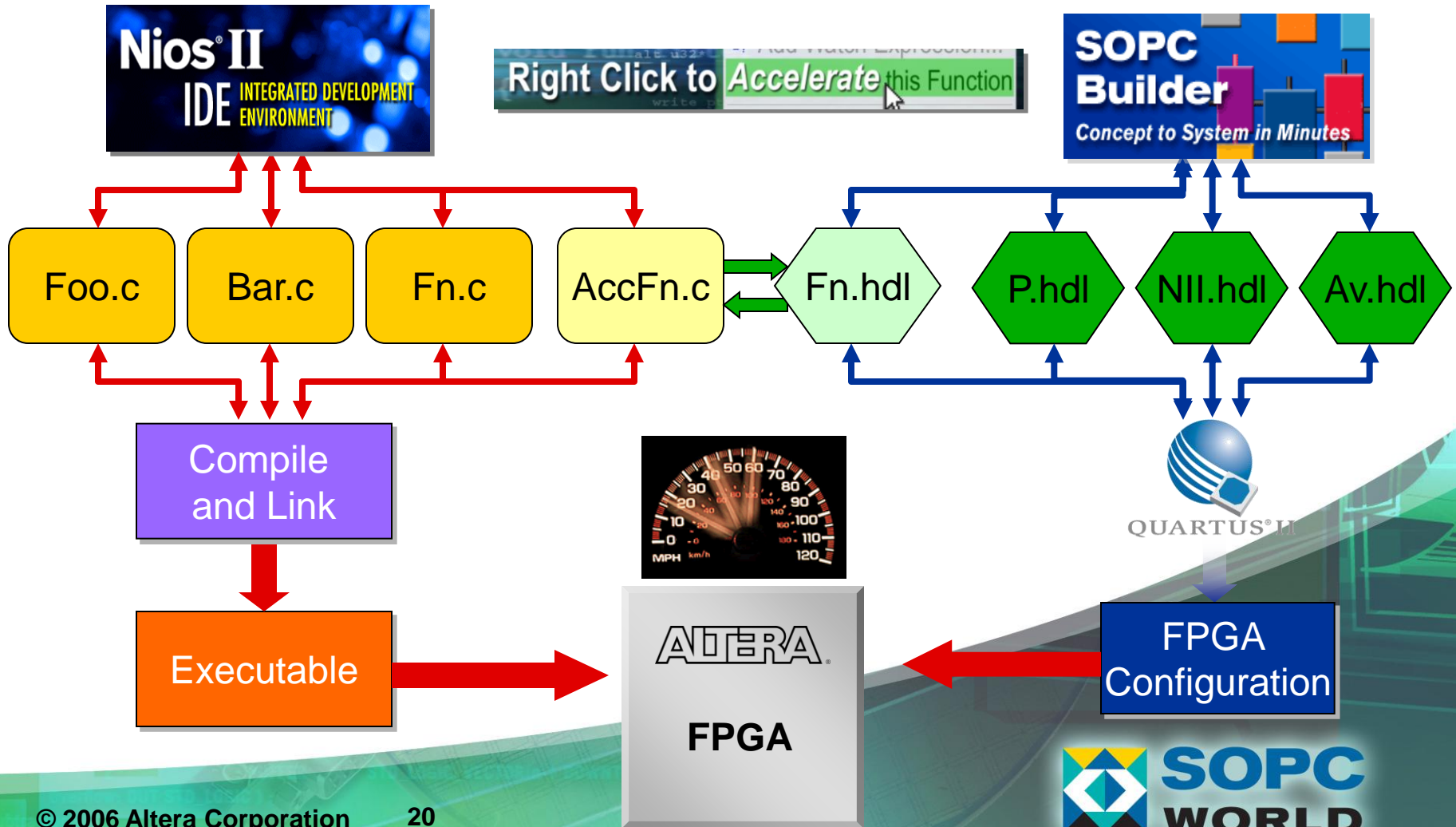


- Base system hardware
 - Already built in SOPC Builder
- C2H Compiler demonstration
 1. Create software in Nios II IDE
 2. Accelerate system performance with C2H Compiler

C2H Compiler Integration

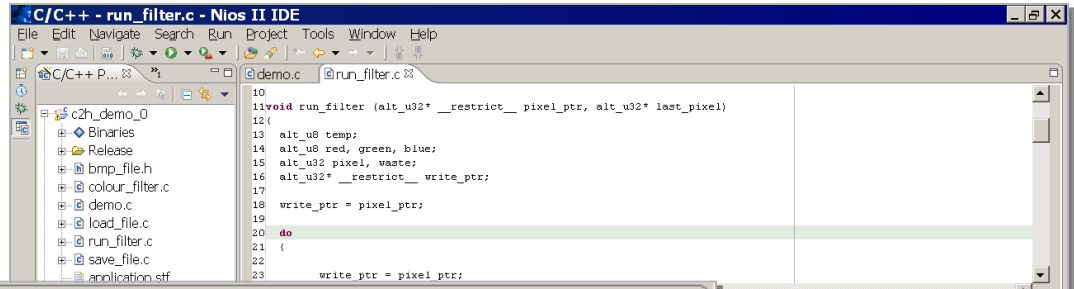
Software

Hardware



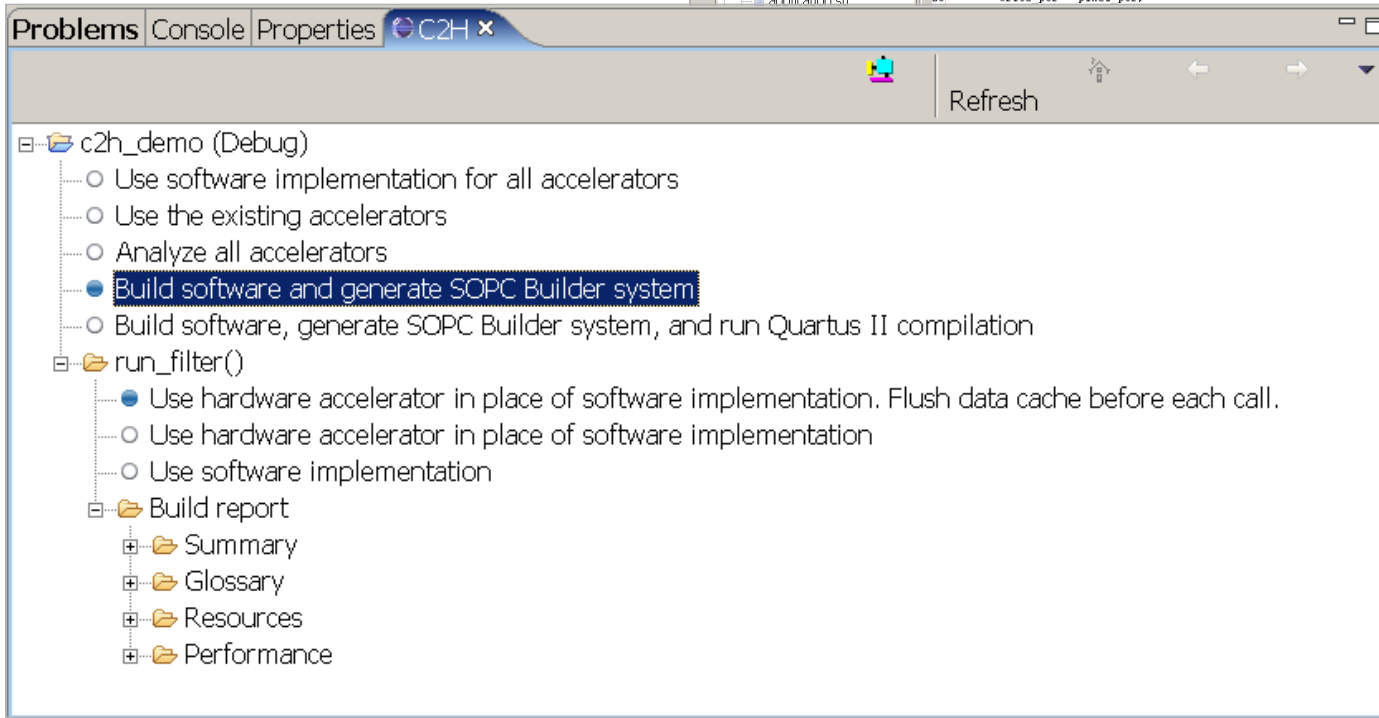
C2H Compiler Controls

- Global controls to control build settings



The screenshot shows the Nios II IDE interface. The top window displays the project structure for 'c2h_demo_0', including folders for 'Binaries', 'Release', and source files like 'bmp_file.h', 'colour_filter.c', 'demo.c', 'load_file.c', 'run_filter.c', and 'save_file.c'. The bottom window shows the source code for 'run_filter.c' with the following content:

```
10  
11 void run_filter (alt_u32* __restrict__ pixel_ptr, alt_u32* last_pixel)  
12 {  
13     alt_u0 temp;  
14     alt_u0 red, green, blue;  
15     alt_u32 pixel, waste;  
16     alt_u32* __restrict__ write_ptr;  
17  
18     write_ptr = pixel_ptr;  
19  
20     do  
21     {  
22  
23         write_ptr = pixel_ptr;
```



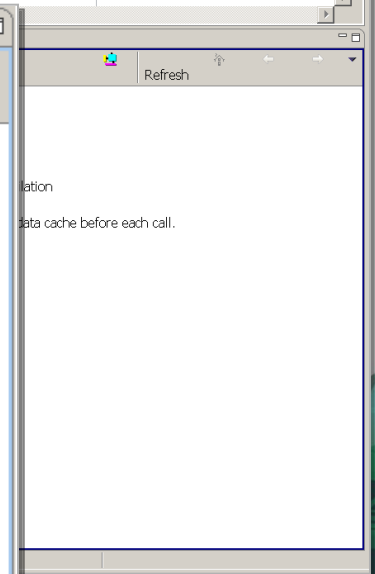
The screenshot shows the 'C2H' Properties window. The 'c2h_demo (Debug)' section has the following options:

- Use software implementation for all accelerators
- Use the existing accelerators
- Analyze all accelerators
- Build software and generate SOPC Builder system
- Build software, generate SOPC Builder system, and run Quartus II compilation

The 'run_filter()' section has the following options:

- Use hardware accelerator in place of software implementation. Flush data cache before each call.
- Use hardware accelerator in place of software implementation
- Use software implementation

Below these are folders for 'Build report', 'Summary', 'Glossary', 'Resources', and 'Performance'.



The screenshot shows a console window with a 'Refresh' button and some text, including 'ation' and 'ata cache before each call.'

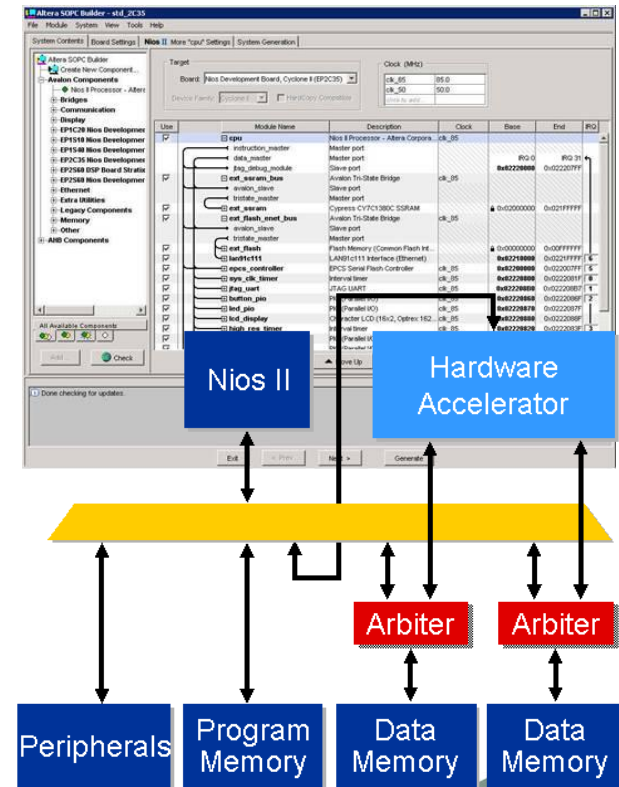
“Secret Sauce” Behind C2H Compiler

■ SOPC Builder

- Connects CPU, accelerator, and memory
- Understands pipeline depth for scheduling

■ Avalon interconnect fabric

- Resolves pointer dereferencing
- High bandwidth interconnect
- Parallel memory access



```
nbit_adder: adder1
    GENERIC MAP (N => NBIT)
    PORT MAP (AddSubR_n => AddSubR_n,
              AddSubR_n1 => AddSubR_n1,
              C_in => C_in,
              C_out => C_out);
multiplexer: mux2to1
    GENERIC MAP (N => NBIT)
    PORT MAP (A_in => A_in,
              B_in => B_in,
              S_in => S_in);
AddSubR_n1 <= (OTHERS => AddSubR_n)
              XOR AddSubR_n;
C_out <= XOR C_in, XOR C_in, XOR M(n-1);
```

C2H Compiler “Under the Hood”

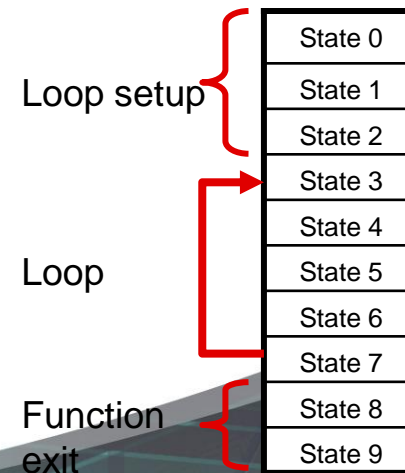
Controlling Logic Generation

- “What you type is what you get”
- Every arithmetic operator you type
 - Makes equivalent hardware unit in accelerator
 - Examples: `*`, `+`, `%`, `>>`
 - Some arithmetic requires zero logic (wires)
 - Examples: `y = x << 3`, `x &= 0xf`;
- Control-flow statements generate control state machine
- Pointer and array references create pipelined master interfaces to memory

Example: Arithmetic Hardware

```
long long MAC
(int *a, int *b, int len)
{
    long long result = 0;
    while (len > 0) {
        result += *a++ * *b++;
        len--;
    }
    return result;
}
```

- 1 - 32x32=64 multiplier
- 3 - 32-bit incrementers
- 1 - 64-bit adder
- 1 - 32-bit comparator
- 2 - read-masters
- Nominal control logic



Example: Controlling Implementation

```
void vector_function ()
{
    long long x, y;
    int *a,*b,*c,*d;
    int len;
    <... Various set-up ...>

    while (<loop-condition>) {
        <...loop overhead...>
        while (len-- > 0) {
            x += *a++ * *b++;
            y += *c++ * *d++;
        }
        <...result-saving...>
    }
}
```

- 1 accelerator unit created
 - Incorporates all arithmetic
- 2 *s typed
 - Creates 2 multipliers
- Control flow
 - X and Y computed in parallel
 - Y does not need to wait for X
- Faster unit, more hardware

```
vector_function
Accelerator module
```



Example: Controlling Implementation

```
void vector_function ()
{
    long long x, y;
    int *a,*b,*c,*d;
    int len;
    <... Various set-up ...>

    while (<loop-condition>) {
        <...loop overhead...>

        x = MAC (a, b, len);
        y = MAC (c, d, len);

        <...result-saving...>
    }
}
```

- 2 accelerator units created
 - vector_function
 - MAC
- MAC unit is *shared*
 - Only 1 multiplier (MAC)
 - Used twice
- Control flow
 - Y result *waits* for X result

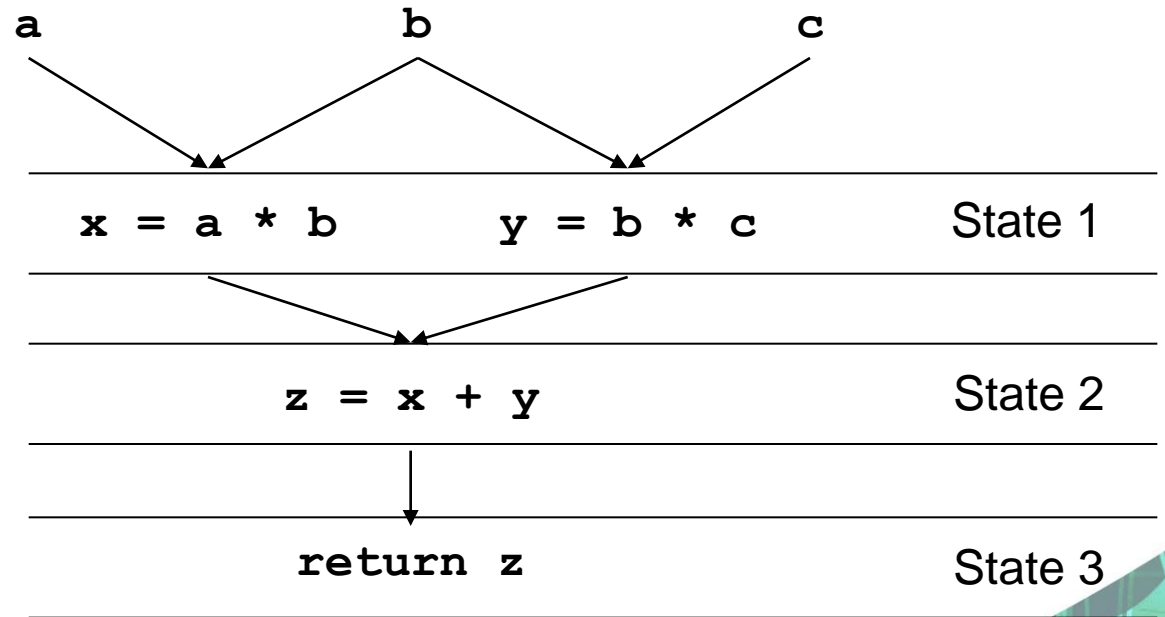
```
vector_function
Accelerator module
```

```
MAC
Sub-accelerator
Module
```

* +

Basic Data Dependencies

```
int foo(int a, int b, int c)
{
  int x ,y, z;
  x = a * b;
  y = b * c;
  z = x + y;
  return z;
}
```



States assigned directly
from the dependency
graph

Performance/Resource Report

■ Accelerator data

- Hardware used
- Loop latency
- Clocks per loop iteration
- Algorithm scheduling

Console Properties C2H x

- [-] c2h_demo_0 (Release)
 - Use software implementation for all accelerators
 - Use the existing accelerators
 - Analyze all accelerators
 - Build software and generate SOPC Builder system
 - Build software, generate SOPC Builder system, and run Quartus II compilation
- [-] run_filter()
 - Use hardware accelerator in place of software implementation. Flush data
 - Use hardware accelerator in place of software implementation
 - Use software implementation
- [-] Build report
 - [-] Summary
 - [-] Glossary
 - [-] Resources
 - [-] About Resources
 - [-] The accelerated function requires 2 Masters.
 - [-] The accelerated function requires 2 Multipliers.
 - [-] Performance
 - [-] About Performance
 - [-] The accelerated function contains 1 loop.
 - [-] file: /run_filter.c line:21 Loop CPL I=1
 - Loop latency : 27
 - Cycles per loop iteration (CPLI) : 1
 - Scheduling information per assignment
 - Scheduling information per state

Sequential Operation

- Software calls hardware accelerators sequentially
 - Processor must wait for accelerators to finish
 - Reduces benefit of dedicated hardware

```
main ()
{ ...variable declarations...
  init();
  while (!error && got_data())
  {
    do_user_interface();
    gather_statistics();
    if (got_new_data())
      edge_detect(in_buf1, out_buf1);
      image_rotate(in_buf2, out_buf2);
    check_for_errors();
  }
  cleanup();
}
```

Concurrent Operation

- Software calls accelerators as *threads*
 - Each accelerator runs concurrently
 - True multitasking system (CPU, hardware accelerators)

```
main ()
{ ...variable declarations...
  init();
  message_queue* msg;

  thread_start (&edge_detect, msg);
  thread_start (&image_rotate, msg);
  thread_start (&gather_statistics, msg);

  while (!msg->done()) { sleep(); }
}
```

```
nbit_adder: adder7
  GENERIC MAP (x => 6)
  PORT MAP (AddSubR_n => 0, M => 1)
multiplexer: mux2to3
  GENERIC MAP (x => 2)
  PORT MAP (A_in1 => 2, S_in1 => 0,
    AddSubR_n <= (OTHERS => AddSubR_n)
    M => 1, YOR AddSubR_n,
    M_n1 <= (OTHERS => YOR M(n-1));
```

C-based Industry Partnership



Best of Altera and Partner Technologies

- C language design tools available today from industry-leading suppliers



- C-based design initiative

- Open application program interface (API) for Altera[®] system generation tools
 - SOPC Builder
 - C2H Compiler
- Technology sharing and joint development
 - “Right-click to accelerate” for ESL partner tools from within Nios II IDE

```
nbit_adder: adderx
    GENERIC MAP (x => n)
    PORT MAP (AddSubR_n => AddSubR_n, M => M)
multiplexer: mux2to1
    GENERIC MAP (x => n)
    PORT MAP (A1 => Z, S1 => G,
    AddSubR_n <= (OTHERS => AddSubR_n)
    M <= M
    XOR AddSubR_n
    carryout XOR G(n-2), XOR G(n-2), XOR M(n-1);
```

Summary

Nios II C2H Compiler Offers

■ Performance

- Boosts overall FPGA computing performance
- No increase in clock frequency

■ Productivity

- Generates and adds hardware accelerators automatically
- Results in substantial time savings
- Supports existing tool chain

***Altera Is Driving Innovation
With System-Level Design Tools***

Thank You Q & A