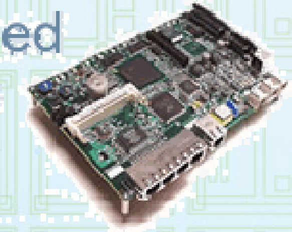
ALTERA.

Automating Hardware Acceleration of Embedded Systems

May 2007

the embedded masterclass



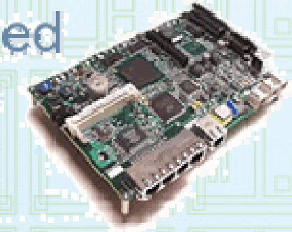
Agenda

- Introduction - FPGA for Embedded
- The C2H Compiler
  - Overview
  - How the C2H compiler works
  - Optimising performance/size
- More information

ALTERA.

Introduction

the embedded
masterclass



the embedded masterclass

Today’s FPGA Devices Meet Embedded System Requirements

- Wide range of fast I/O
- High-performance Digital Signal Processing (DSP) blocks
- Abundant logic
- Substantial embedded memory
- Low Cost FPGA and Structured ASIC families
- Soft Processor cores

QUARTUS II
SOPC Builder
Nios II IDE
Custom Logic
C/C++
Middle-ware
10/100 MAC
RTOS
ALTERA
USB
Flash Controller
PCI
Trace Debug
UART
SDRAM Controller
DMA
Drivers
DDRAM Controller
Nios II

Stratix III Stratix II GX Cyclone III HardCopy II

LG Electronics 3G Base Station

Application:

3G Base Station

Industry:

Wireless Communications

Altera Value Proposition:

- *Highest FPGA Capacity Enabled One-Chip Solution*
- *FPGA Flexibility Delivered Short Time-to-Market*
- *HardCopy Structured ASIC Provided Path to Cost Reduction*

"Altera HardCopy Stratix devices provide a low-risk, cost-optimized, high-volume solution for our next generation 3G base station, eliminating the need for us to use an ASIC or standard product. By offering industry-leading density and a seamless migration path from Stratix FPGAs to HardCopy devices, Altera improves our time-to-market and lowers our costs, enabling us to penetrate new markets."

–Bong-Bin Park, Senior Vice President, CDMA Research Lab

Altera Products Chosen:

Stratix™

HARDCOPY™

ALTERA.

Toolrama
DiabloSport Predator

Application:

Automotive Diagnostic Tool

Industry:

Automotive

Altera Value Proposition:

- *Nios Processor + SOPC Builder + Low Cost Cyclone FPGA = Perfect Microprocessor Solution*
- *FPGA Flexibility Enables Acceleration via Custom Peripherals*

> *"In our Predator product, the industry-leading low cost and flexibility of the Altera solution enabled us to replace an off-the-shelf processor, add features, and increase performance. Based on our successful experience with Cyclone devices, we will be adopting the Cyclone II family for all of our new product development, which will enable us to deliver even greater functionality at lower cost."*
>
> *–Ivan Kotzig, Chief Engineer*

Altera Products Chosen:

Cyclone

Nios II

ALTERA.

Tait

TM8100, TM8200, TM9100, and TP9100 Radios

“Altera products provide us with the flexibility to configure combinations of complex embedded and signal processing functions that are not available in a single off-the-shelf processor solution. With Cyclone devices we create the exact combination of peripherals and functions we require, including complex multi-rate channel filters, automatic gain control, frequency control loops, and complex high performance modems.”

–Tony Berggren, Radio Architectures Technology Leader

Application:

Digital Portable Radio

Industry:

Wireless Communications

Altera Value Proposition:

- *Flexible Platform for Rapid Development of Multiple Product Lines*
- *Customization Not Possible with Standard Products*
- *Adopting Nios II Processor Reduces Power Consumption*

Altera Products Chosen:

Cyclone II

Nios II

ALTERA

Navman
TRACKFISH 6600

Application:

Marine Navigation Instrument

Industry:

Digital Consumer

Altera Value Proposition:

- ***Reduced Component Cost and Power Consumption by Integrating Off-the-Shelf Processor***
- ***Nios-Based Custom Microcontrollers Address Specific Design Requirements***

"Replacing an off-the-shelf processor with a Cyclone series device running a Nios processor enabled us to achieve the highest visual quality, as well as bring these improvements to a wide variety of display sizes. These benefits, combined with power consumption and cost savings, have led us to adopt the Nios processor as our preferred embedded solution."

–Shane Dooley, Marine GPS Product Manager

Altera Products Chosen:

Cyclone

Nios II

ALTERA.

Intevac NightVista

Application:

Night Vision Camera

Industry:

Industrial

Altera Value Proposition:

- *Lower Cost Solution than DSP Processor*
- *High Integration, Small Form Factor*
- *Low Power*

> *"Replacing an off-the-shelf DSP processor allowed us to reduce five separate boards of components to a single board occupied mainly by a Cyclone device running an embedded Nios processor. The Altera-based approach reduced our component costs by at least 20 percent and decreased our power consumption to a fifth of what it had been."*
>
> *–David Main, Engineering Group Manager*

Altera Products Chosen:

Cyclone

Nios II

ALTERA.

Sanyo
PLV-Z5 Home Theater Projector

PROJECTOR CENTRAL ✓ HIGHLY RATED

Projector Reviews .com HOT Product AWARD

HiVi Best Buy 2006

"By using the Stratix FPGA plus HardCopy structured ASIC solution for the PLV-Z5, we can design new features and functions more quickly and cost-effectively than alternative silicon solutions allow."

-Kazuto Sugimura, General Manager of Technology Unit, Projector Central Business Unit

Application:

LCD Projector

Industry:

Digital Consumer

Altera Value Proposition:

- *Rapid, Cost-Effective Product Development Not Possible with Alternative Solutions*
- *Altera Device + Nios II Implements Image Enhancement Functions Resulting in Award-Winning Product*

Altera Products Chosen:

Stratix

HARDCOPY™

Nios II

ALTERA

Loewe
Spheros and Xelos LCD Televisions

“For several years Loewe has leveraged the flexibility of FPGAs, enabling us to respond quickly to differing display requirements without major PCB modifications. The density of the Cyclone II family now enables us to integrate our Image+ picture improvement algorithms at an attractive cost. Cyclone II's timescales matched our development schedule and Altera's on time delivery allowed us to meet our launch target for the Image+ equipped TV sets.”

–Roland Bohl, Director R&D

Application:

LCD TVs

Industry:

Digital Consumer

Altera Value Proposition:

- *Cyclone Series FPGAs Deliver Low-Cost Image Enhancement Functions*
- *Flexible LCD Timing Broadens Pool of Available Panels, Reducing Manufacturing Costs*

Altera Products Chosen:

Cyclone™ II

Cyclone™

ALTERA.

Blaupunkt
TravelPilot Rome Car Audio / Sat.Nav. Unit

“The combination of Altera’s programmable solutions for the automotive industry and excellent development support shortened our design time by six months. We were able to reduce design complexity by replacing multiple standard components with a single Cyclone device hosting a Nios II embedded processor, which eased our development effort and increased our product quality and reliability.”

–Georg Sandhaus, Director of System Engineering

Application:

Auto Navigation

Industry:

Automotive

Altera Value Proposition:

- *Low-Cost, Flexible Graphics Processing and Control Functions*
- *Shortened Design Time by Six Months*
- *Cyclone + Nios II Processor Enable Platform for Rapid Product Development*

Altera Products Chosen:

Cyclone

Nios II

ALTERA.

Host Automation H2-EBC100 and H2-ECOM100

"Utilizing the Nios processor and Cyclone FPGA approach, we can get the exact mix of peripherals we need, in a package that we need, at a reasonable cost. In addition, we can reduce the number of unique parts in our inventory by using the same hardware platform for all of our designs."

–Bob Palermo

Senior Design Engineer

Application:

100 Base-T Ethernet Controllers for a PLC

Industry:

Industrial Automation

Altera Value Proposition:

- *Nios Processor + Low Cost Cyclone FPGA = Perfect Microprocessor Solution*
- *FPGA Flexibility Enables Connection to Proprietary PLC Backplane and Custom Microcontroller Peripheral Set*

Altera Products Chosen:

Cyclone

Nios II

ALTERA

Phoenix Contact ILC150 PLC

Application:

Process Logic Controllers

Industry:

Industrial

Altera Value Proposition:

- *Scaleable Platform*
- *Low Cost*
- *Obsolescence free*

Altera Products Chosen:

Cyclone II

Nios II

ALTERA

"Phoenix Contact have been using Altera and the Nios Soft CPU since 2002 to develop a scaleable hardware platform used in products like the ILC 350 ETH and ILC 390 PN 2TX-IB. We have also been able to develop a new, highly compact generation of controller called the ILC 150 ETH. The combination of an entirely FPGA-based platform with the NIOS soft CPU has enabled us to deliver a very small but powerful controller with integrated Ethernet and INTERBUS interfaces at a extremely competitive market price"

- Roland Bent,
Executive VP Marketing and Development, Phoenix Contact

Siemens AG
SIMATIC MV220

Application:

Color area sensor for manufacturing and packaging applications

Industry:

Industrial

Altera Value Proposition:

- *High Performance*
- *Low Cost*
- *Flexibility*

Altera Products Chosen:

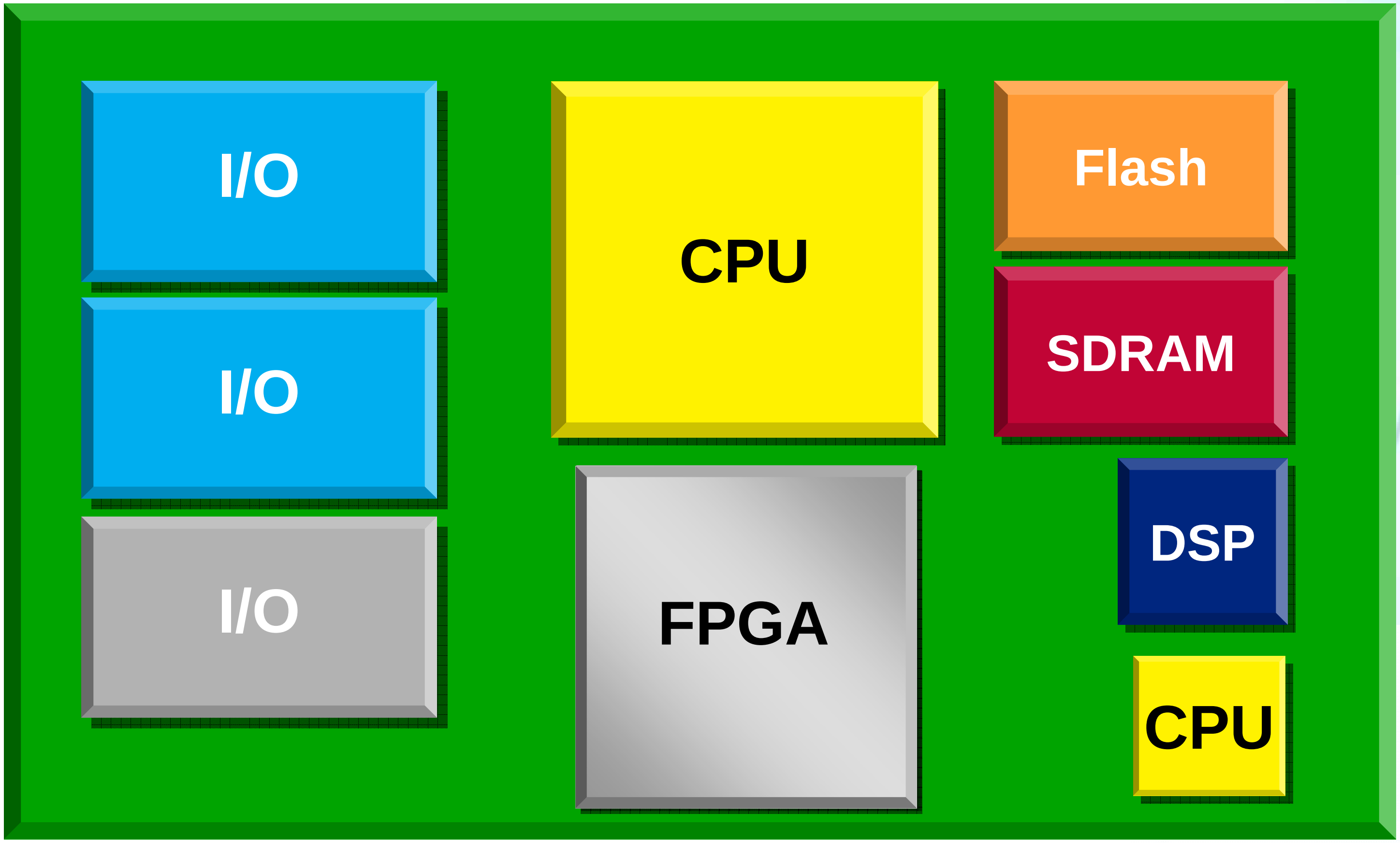
Cyclone™.

Nios®

ALTERA.

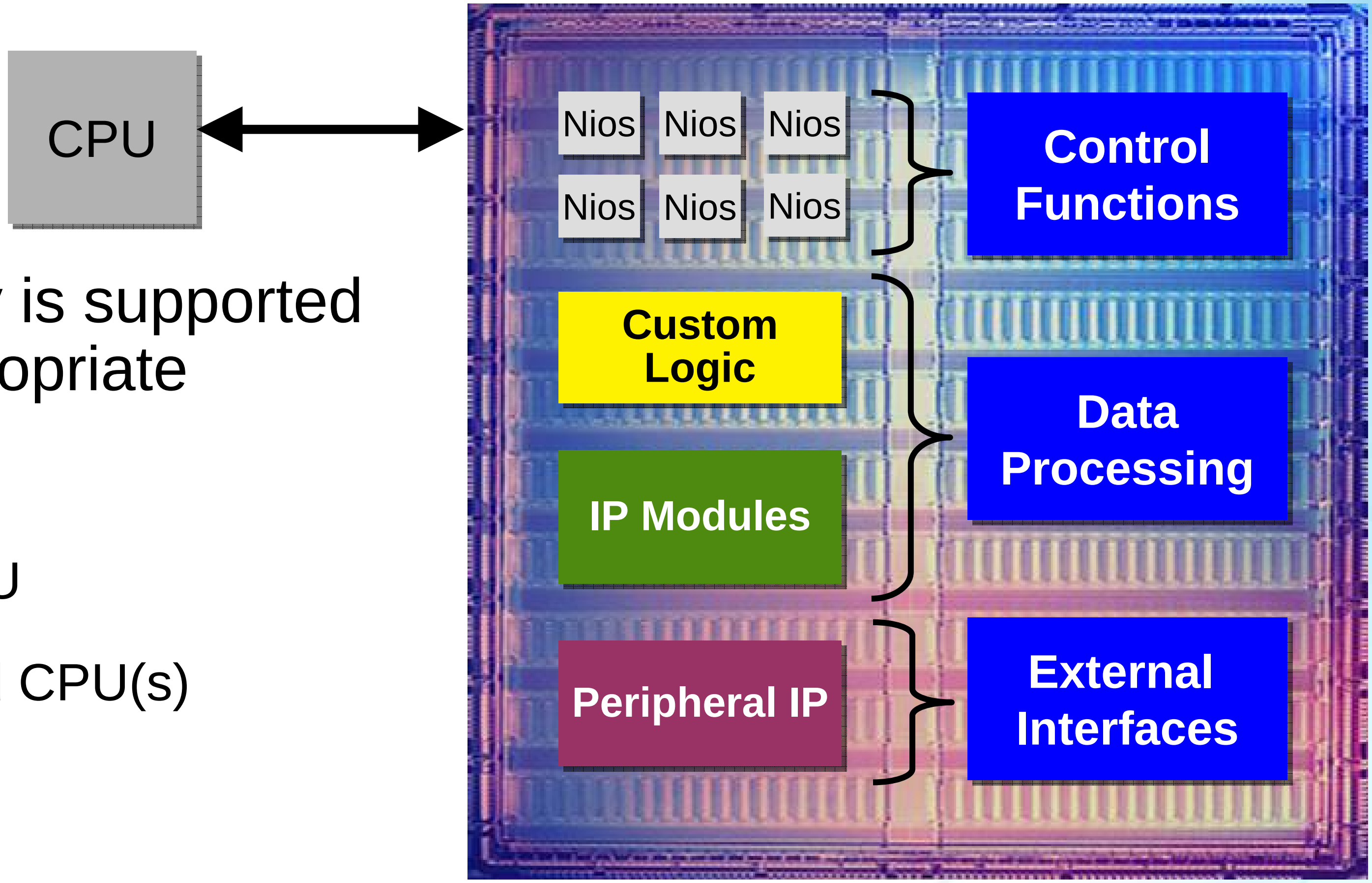
> *"For the colour area Sensor SIMATIC MV220* we *needed to integrate a complete image processing system into a compact form factor, presenting serious performance, heat dissipation and cost challenges. Using the flexible combination of Cyclone series FPGAs with the Nios II embedded processor we were able to achieve our goals and deliver a high performance, highly integrated and cost effective solution."*
>
> *– Jens Hauffe, Product Manager*
> *– Dr. Peter Thamm, Project Leader*
> *Siemens AG*

Reducing System Costs - Integration



Replace External Devices with Programmable Logic

FPGA Provides Hybrid Approach

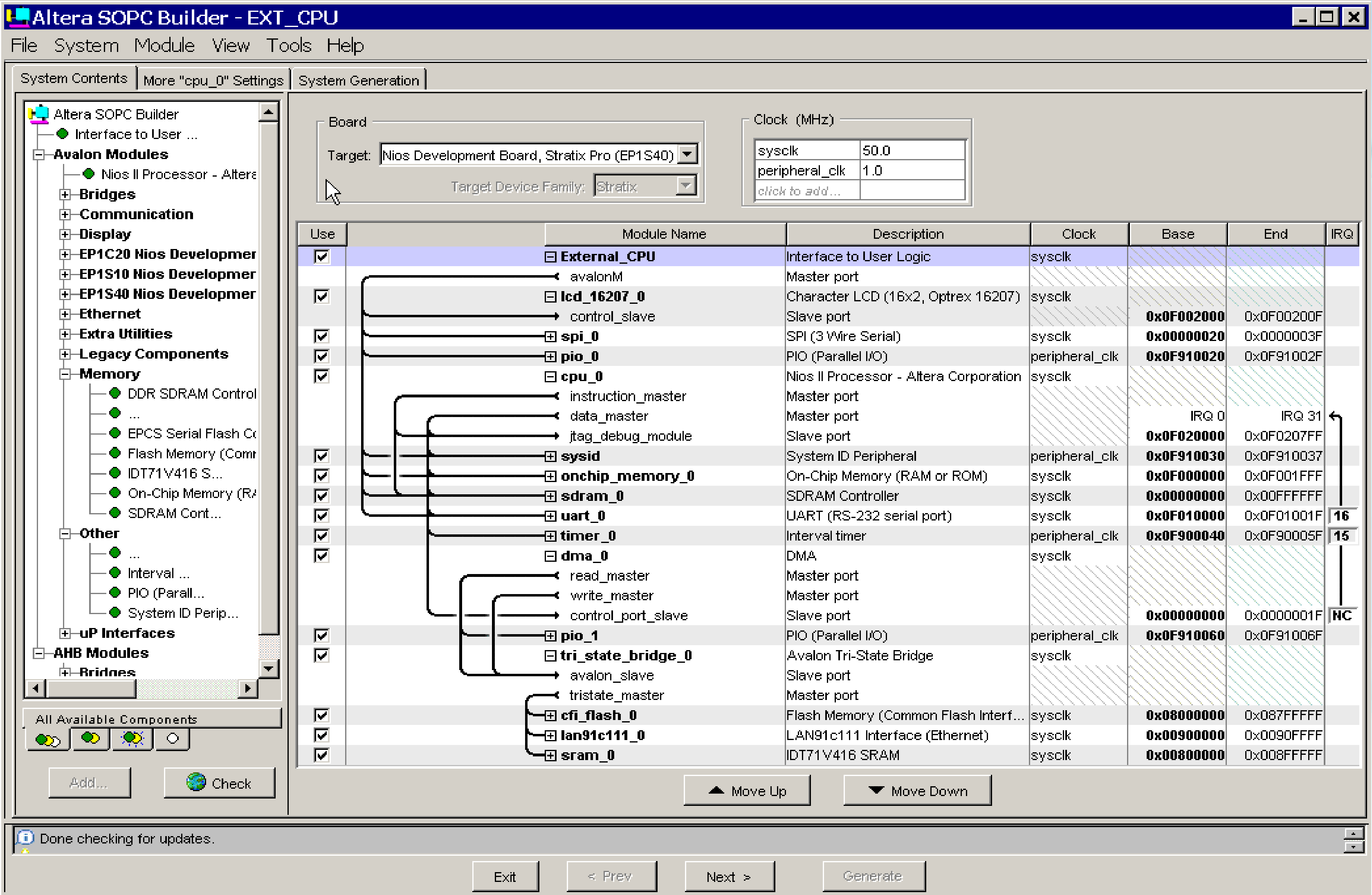


Functionality is supported in most appropriate location:

- External CPU
- FPGA based CPU(s)
- FPGA Logic

Differentiate Your System With FPGA Based Functionality

SOPC Builder



- Altera SOPC Builder
  - Interface to User ...
- Avalon Modules
  - Nios II Processor - Altera
  - Bridges
  - Communication
  - Display
  - EP1C20 Nios Developmer
  - EP1S10 Nios Developmer
  - EP1S40 Nios Developmer
  - Ethernet
  - Extra Utilities
  - Legacy Components
  - Memory
    - DDR SDRAM Control
    - ...
    - EPCS Serial Flash Co
    - Flash Memory (Comr
    - IDT71V416 S...
    - On-Chip Memory (RA
    - SDRAM Cont...
  - Other
    - ...
    - Interval ...
    - PIO (Parall...
    - System ID Perip...
  - uP Interfaces
- AHB Modules
  - Bridges

Board

Target: Nios Development Board, Stratix Pro (EP1S40)

Target Device Family: Stratix

Clock (MHz)

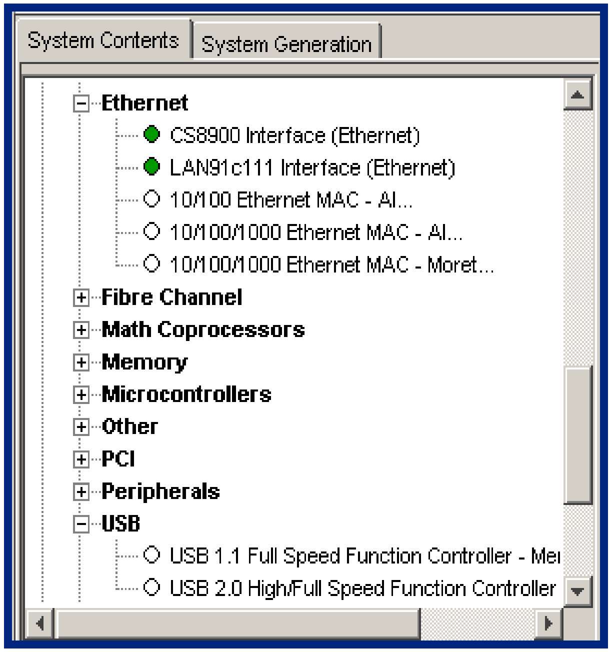
sysclk	50.0
peripheral_clk	1.0
click to add...	

Use	Module Name	Description	Clock	Base	End	IRQ
☑	External_CPU	Interface to User Logic	sysclk			
	avalonM	Master port				
☑	lcd_16207_0	Character LCD (16x2, Optrex 16207)	sysclk			
	control_slave	Slave port		**0x0F002000**	0x0F00200F	
☑	spi_0	SPI (3 Wire Serial)	sysclk	**0x00000020**	0x0000003F	
☑	pio_0	PIO (Parallel I/O)	peripheral_clk	**0x0F910020**	0x0F91002F	
☑	cpu_0	Nios II Processor - Altera Corporation	sysclk			
	instruction_master	Master port				
	data_master	Master port		IRQ 0	IRQ 31	
	jtag_debug_module	Slave port		**0x0F020000**	0x0F0207FF	
☑	sysid	System ID Peripheral	peripheral_clk	**0x0F910030**	0x0F910037	
☑	onchip_memory_0	On-Chip Memory (RAM or ROM)	sysclk	**0x0F000000**	0x0F001FFF	
☑	sdram_0	SDRAM Controller	sysclk	**0x00000000**	0x00FFFFFF	
☑	uart_0	UART (RS-232 serial port)	sysclk	**0x0F010000**	0x0F01001F	16
☑	timer_0	Interval timer	peripheral_clk	**0x0F900040**	0x0F90005F	15
☑	dma_0	DMA	sysclk			
	read_master	Master port				
	write_master	Master port				
	control_port_slave	Slave port		**0x00000000**	0x0000001F	NC
☑	pio_1	PIO (Parallel I/O)	peripheral_clk	**0x0F910060**	0x0F91006F	
☑	tri_state_bridge_0	Avalon Tri-State Bridge	sysclk			
	avalon_slave	Slave port				
	tristate_master	Master port				
☑	cfi_flash_0	Flash Memory (Common Flash Interf...	sysclk	**0x08000000**	0x087FFFFF	
☑	lan91c111_0	LAN91c111 Interface (Ethernet)	sysclk	**0x00900000**	0x0090FFFF	
☑	sram_0	IDT71V416 SRAM	sysclk	**0x00800000**	0x008FFFFF	

Done checking for updates.

SOPC Builder System Design

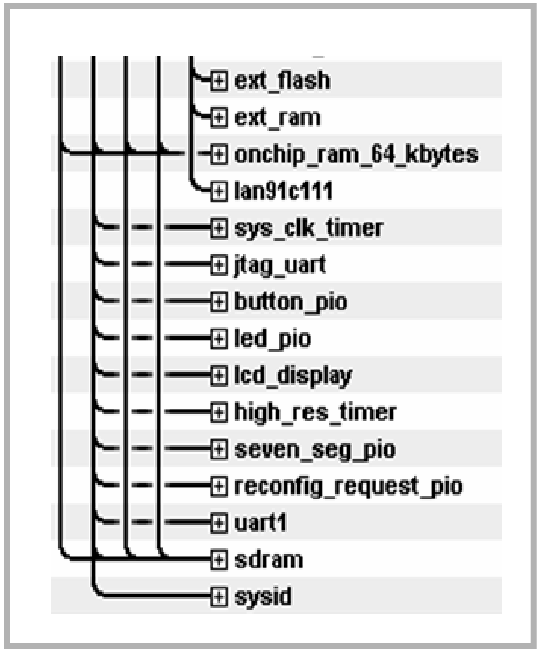
1. Select & Configure IP



System Contents | System Generation

- Ethernet
  - ● CS8900 Interface (Ethernet)
  - ● LAN91c111 Interface (Ethernet)
  - ○ 10/100 Ethernet MAC - Al...
  - ○ 10/100/1000 Ethernet MAC - Al...
  - ○ 10/100/1000 Ethernet MAC - Moret...
- Fibre Channel
- Math Coprocessors
- Memory
- Microcontrollers
- Other
- PCI
- Peripherals
- USB
  - ○ USB 1.1 Full Speed Function Controller - Mei
  - ○ USB 2.0 High/Full Speed Function Controller

2. Select Connections



- ext_flash
- ext_ram
- onchip_ram_64_kbytes
- lan91c111
- sys_clk_timer
- jtag_uart
- button_pio
- led_pio
- lcd_display
- high_res_timer
- seven_seg_pio
- reconfig_request_pio
- uart1
- sdram
- sysid

3. Generate System

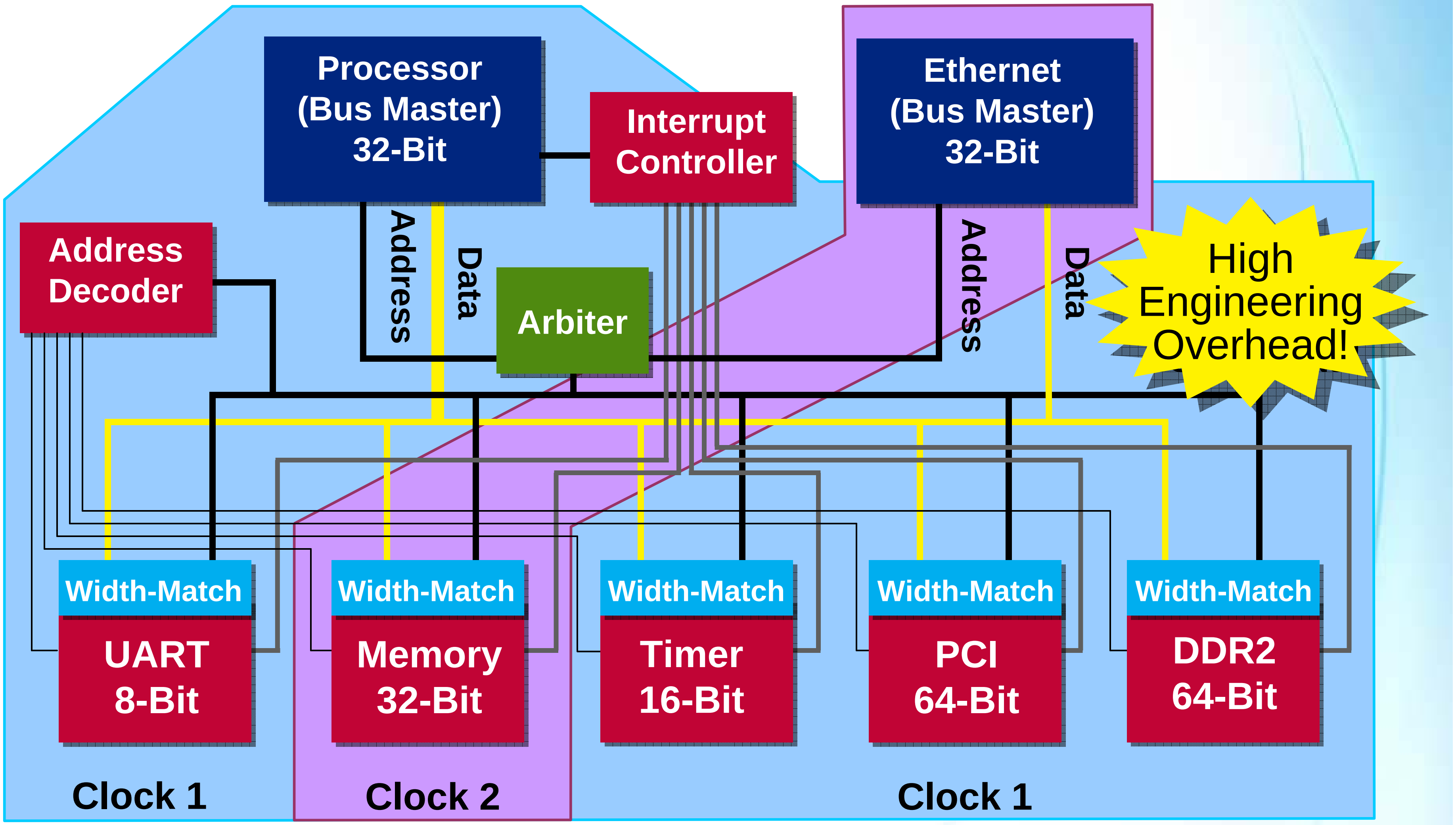
QUARTUS® II

HDL Simulator

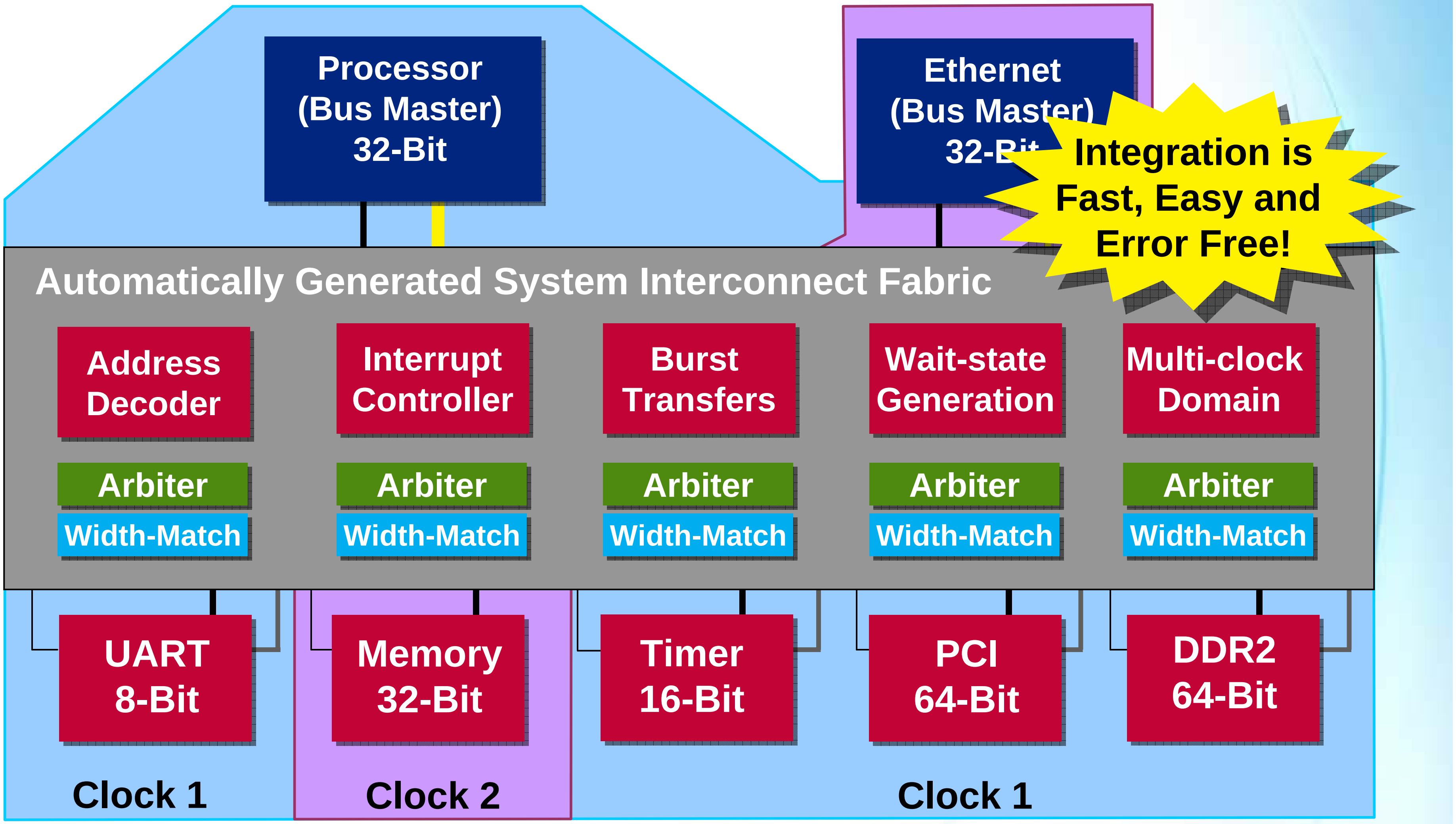
ALTERA FPGA

Cuts Weeks Off Development Time

Traditional System Design

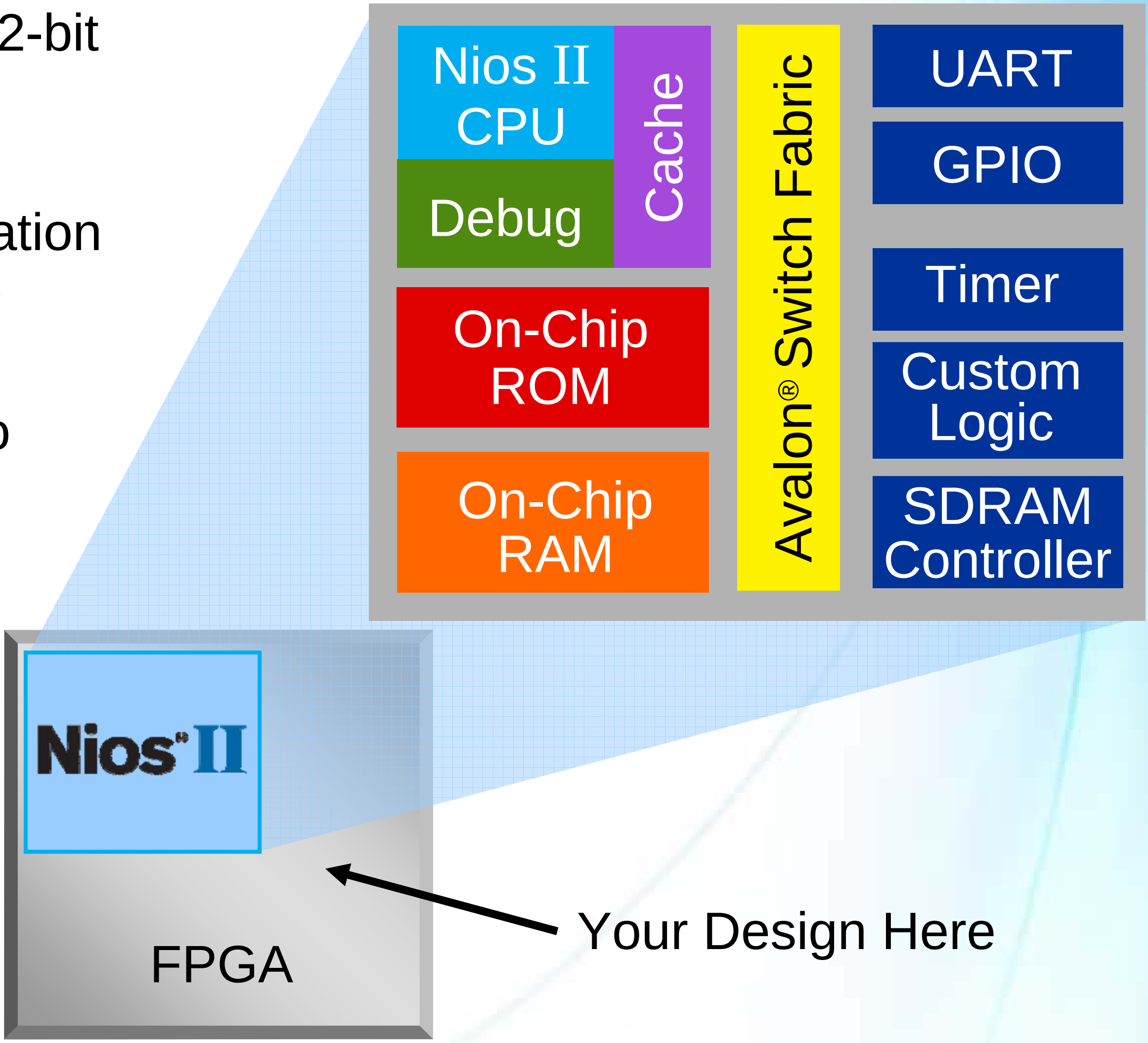


SOPC Builder Integration

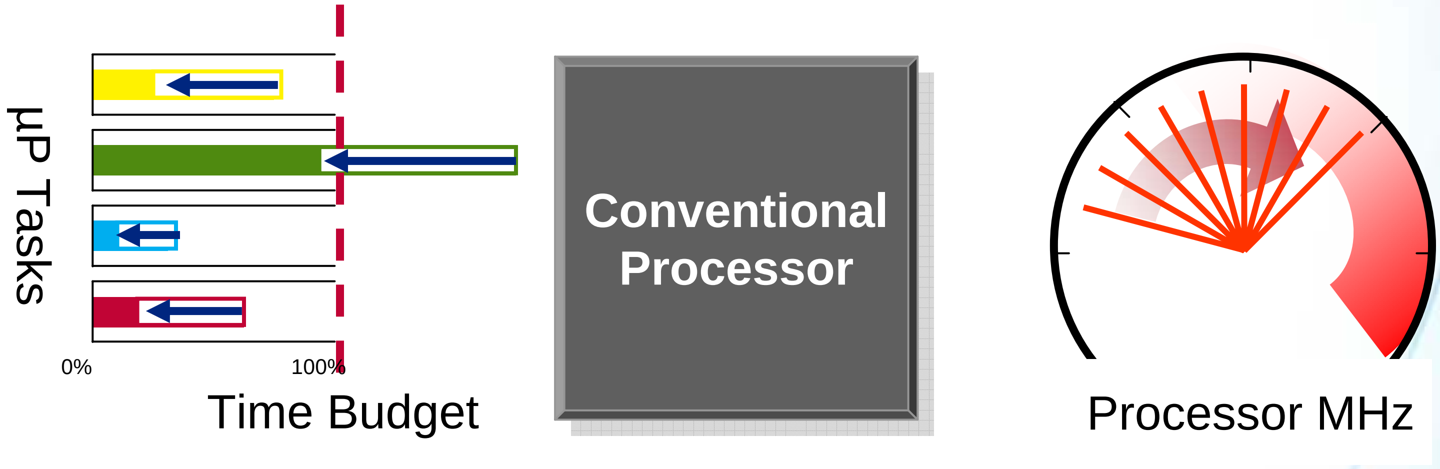


Nios II Processor Overview

- Family of configurable 32-bit RISC processors
- Automated processor configuration and integration of peripherals via SOPC Builder
- Integrate custom logic to add custom features and boost performance
- Performance up to 300 DMIPs (/f, Stratix III)
- Cost as low as 25¢ of logic (/e, Cyclone III)

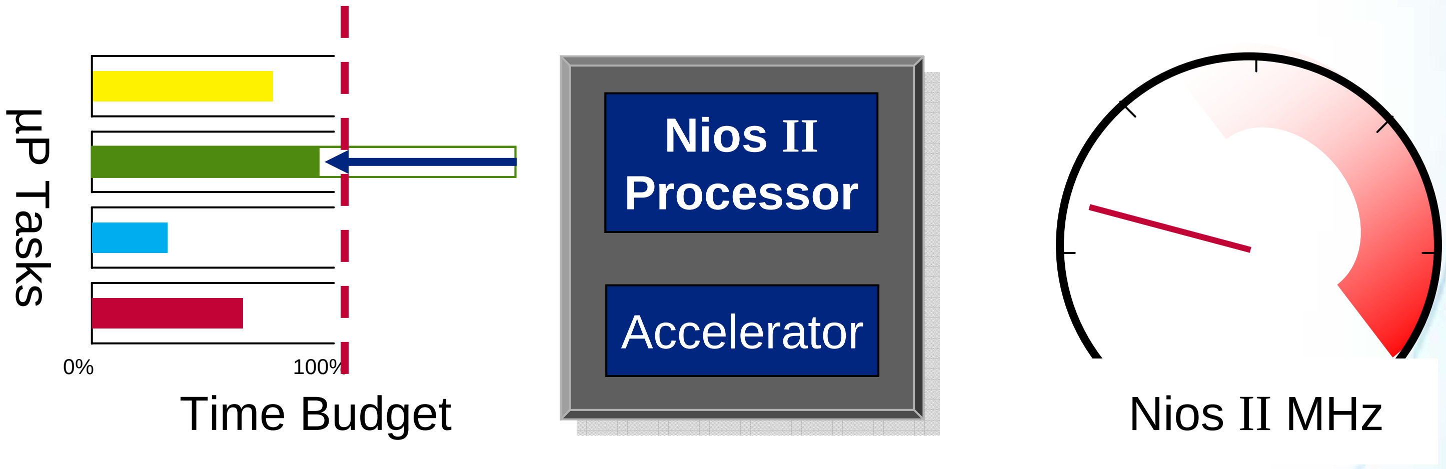


Traditional Processor Acceleration



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

Accelerate Only What’s Needed

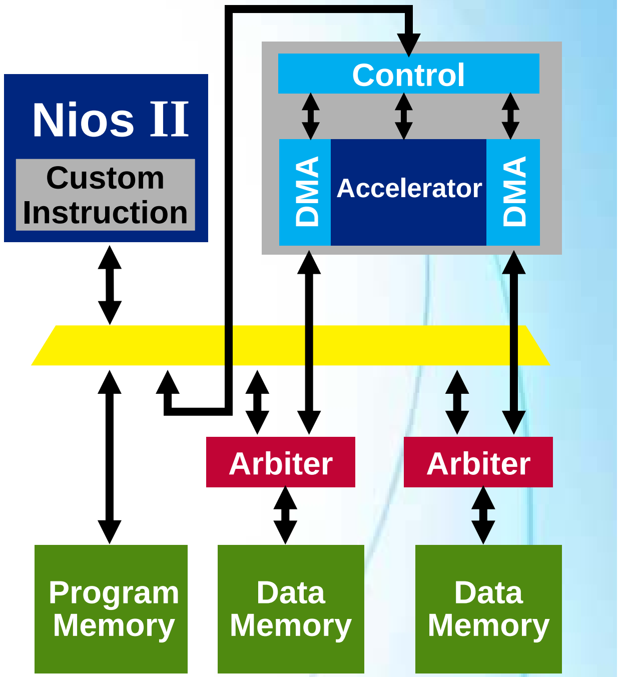


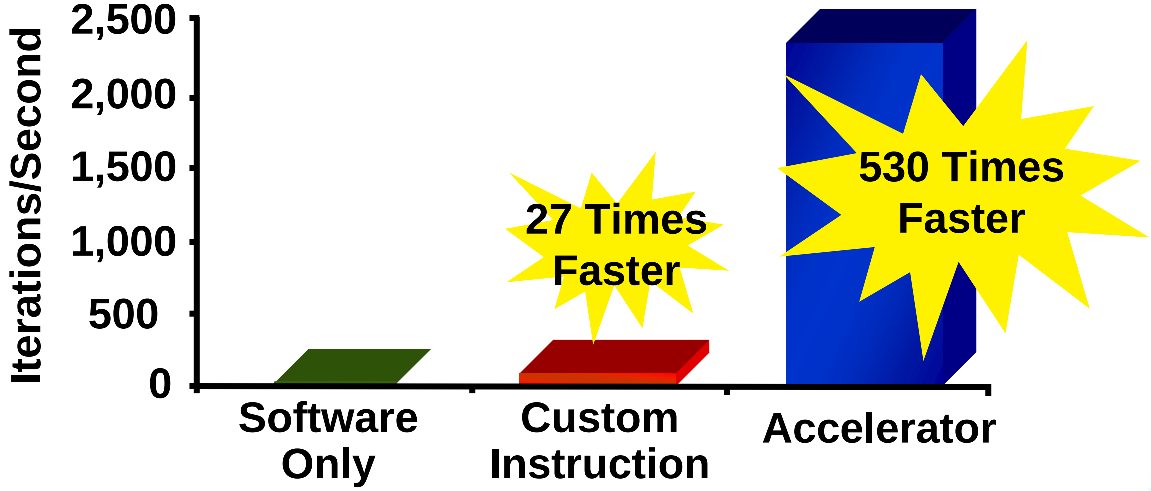
Transfer Processing to Hardware
Highly Effective – Minimal Impact to System

(eg. Image Rotation - performance of 95MHz Nios II with C2H Accelerator equivalent to 1.4 GHz PowerPC)

Accelerating Software in FPGA

- Add custom instruction
  - Ideal for discrete operations
- Add hardware accelerator
  - Processor & accelerator can run concurrently
  - More work per clock
  - Lower f_{MAX}, power, cost
  - Ideal for block operations





* Accelerator running 64Kb CRC at 100 MHz

SW->HW Accelerator Integration

Standard Flow

- Profile Code
- Identify Bottlenecks
- Re-partition Memory
- Create an Accelerator
  - HDL
    - Design
    - Simulate
    - Integrate
  - Software driver
    - Write function
    - Integrate
    - Re-build ‘C’ code
- Verify
- Repeat based upon results

Altera Flow

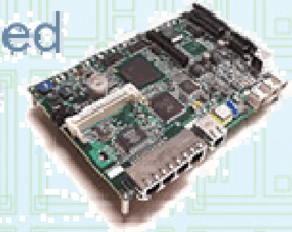
- Profile Code
- Identify Bottlenecks
- Re-partition Memory
- Create an Accelerator

CB23 8EU

  - HDL
    - Design
  - Software driver
    - Write function
  - Integrate HW and SW with SOPC Builder
- Verify
- Repeat based upon results

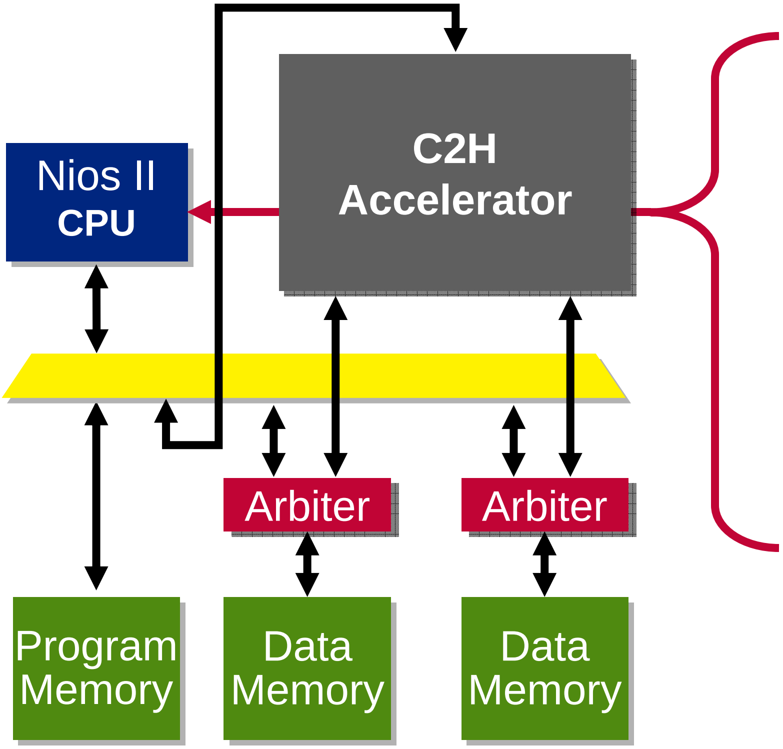
C2H Compiler For Nios II

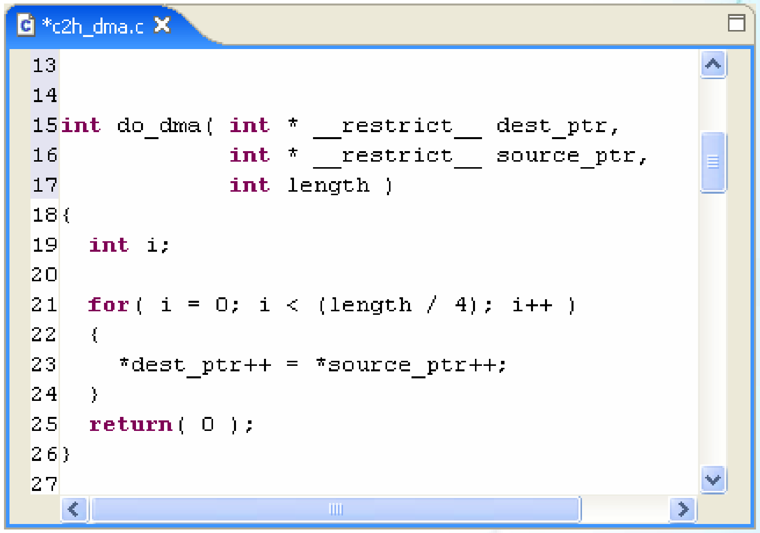
the embedded masterclass



Altera C2H – The SW/HW Accelerator Solution

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

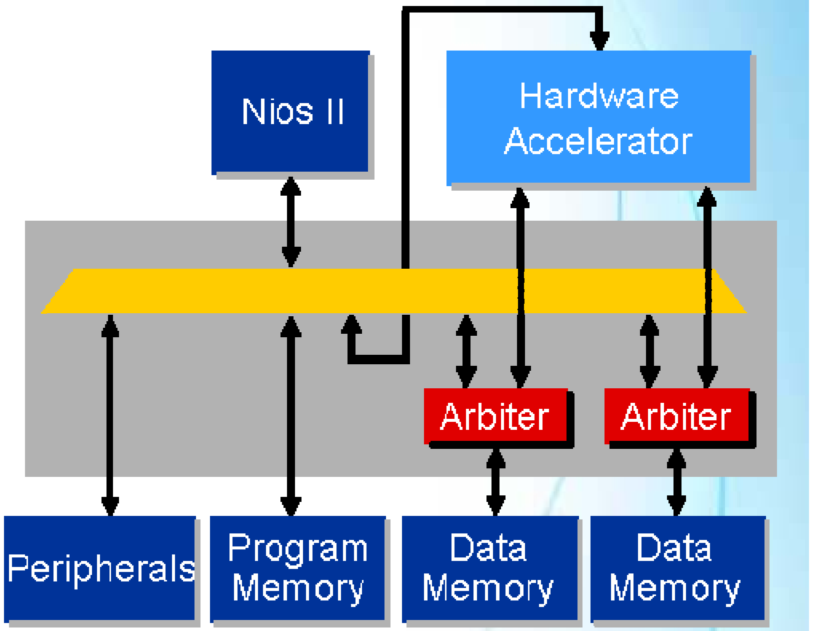




```


int do_dma( int * __restrict__ dest_ptr,
            int * __restrict__ source_ptr,
            int length )
{
  int i;

  for( i = 0; i < (length / 4); i++ )
  {
    *dest_ptr++ = *source_ptr++;
  }
  return( 0 );
}

```

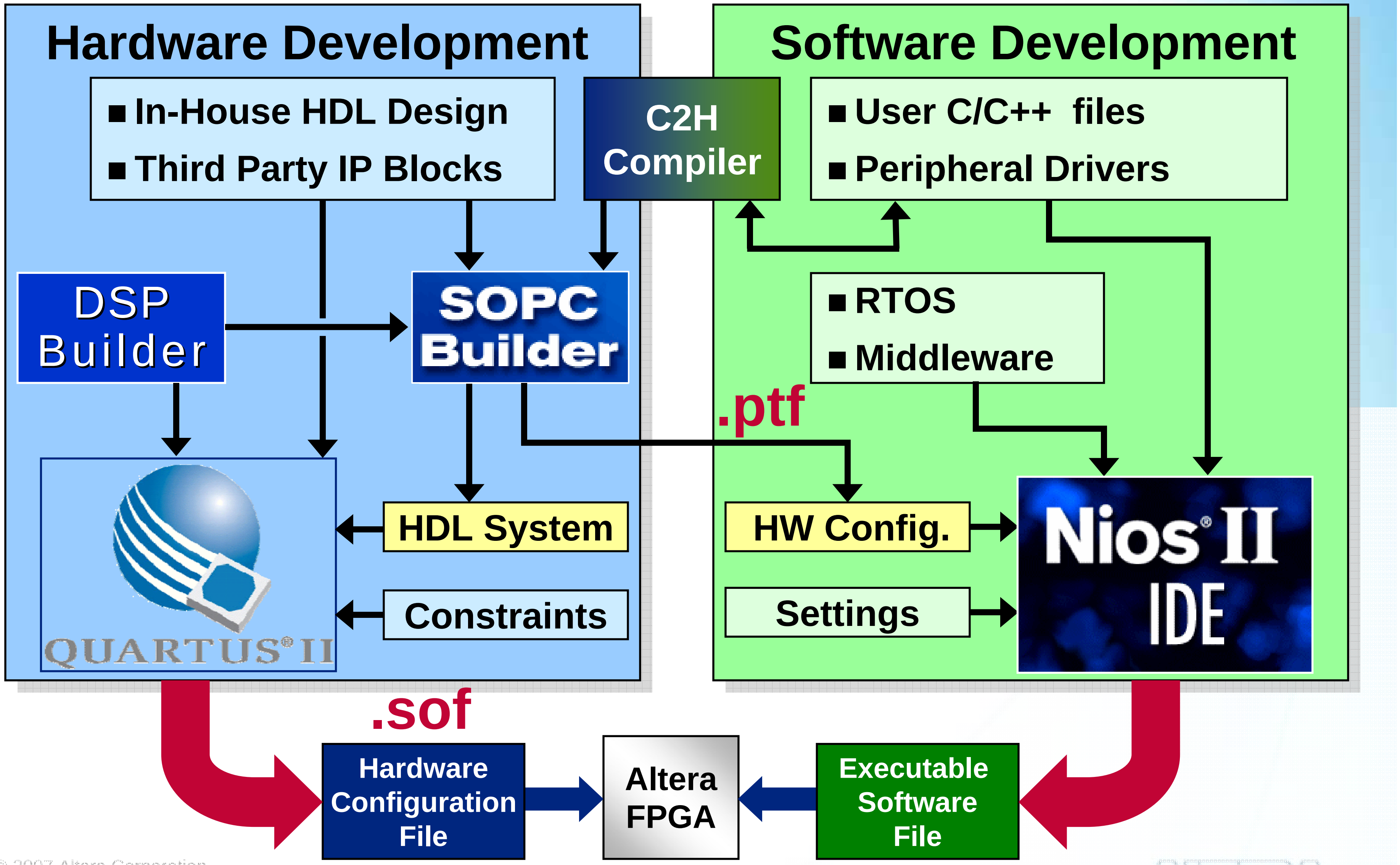
“Secret Sauce” Behind C2H Compiler

- SOPC Builder
  - Automatically connects CPU, accelerator & memory
  - Understands memory latencies (for HW scheduling)
  - Memory connection gives accelerator access to variable data and allows de-referencing of ‘C’ pointers
- Avalon System Interconnect Fabric
  - Automatically generated
  - High bandwidth custom interconnect
  - Master-Slave based design
  - Dedicated connections with slave-side arbitration
  - Any slave can connect to any master



C2H Leverages Over 30 Man-Years of Investment in SOPC Builder & Avalon

C2H Design Flow



Hardware Development

- In-House HDL Design
- Third Party IP Blocks

C2H Compiler

DSP Builder

SOPC Builder

QUARTUS® II

HDL System

Constraints

.ptf

.sof

Software Development

- User C/C++ files
- Peripheral Drivers

- RTOS
- Middleware

HW Config.

Settings

Nios® II IDE

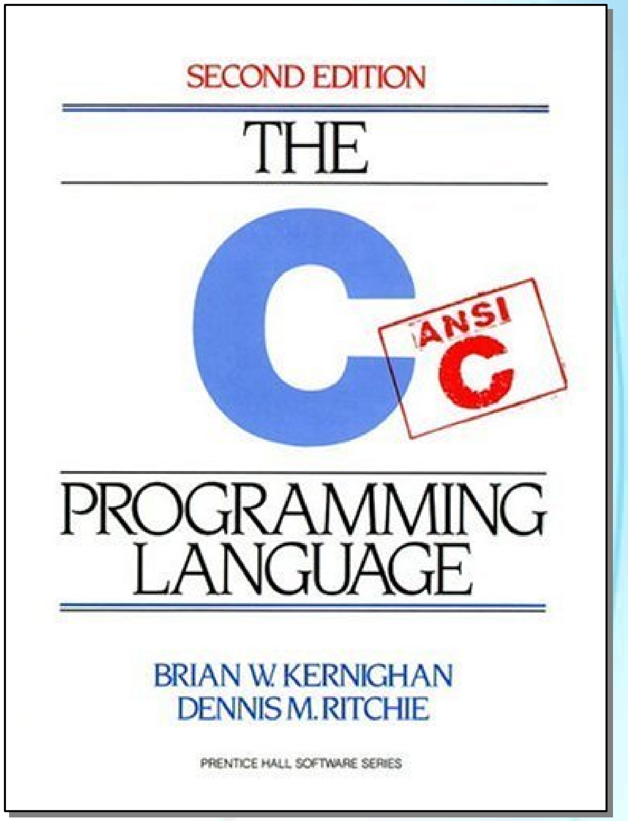
Hardware Configuration File

Altera FPGA

Executable Software File

ANSI C Language Support

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



SECOND EDITION

THE C PROGRAMMING LANGUAGE

ANSI C

BRIAN W. KERNIGHAN
DENNIS M. RITCHIE

PRENTICE HALL SOFTWARE SERIES

C2H Works Best With…

- Time consuming loops (block based data)
  - Take data from buffer/s
  - Process (maths/computation intensive)
  - Write data to buffer/s
- Systems with multiple memories
  - Access to only one external RAM can become a bottleneck
- Use custom instructions for HW operations that do not involve blocks of data

C2H Is Not A Good Fit For…

- Developers with no intention of using Nios II
- Applications that need the **very** highest performance (development effort/time not an issue)
- Applications that require smallest implementation (number of LEs)
- Exception:
  - C2H can be used to confirm bottleneck analysis and then replace with hand coded HDL later

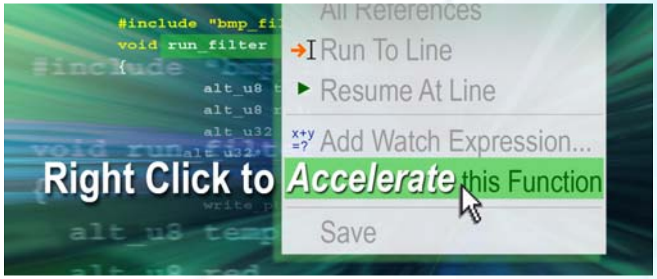
Hand Crafted Accelerator vs. C2H

Standard Flow

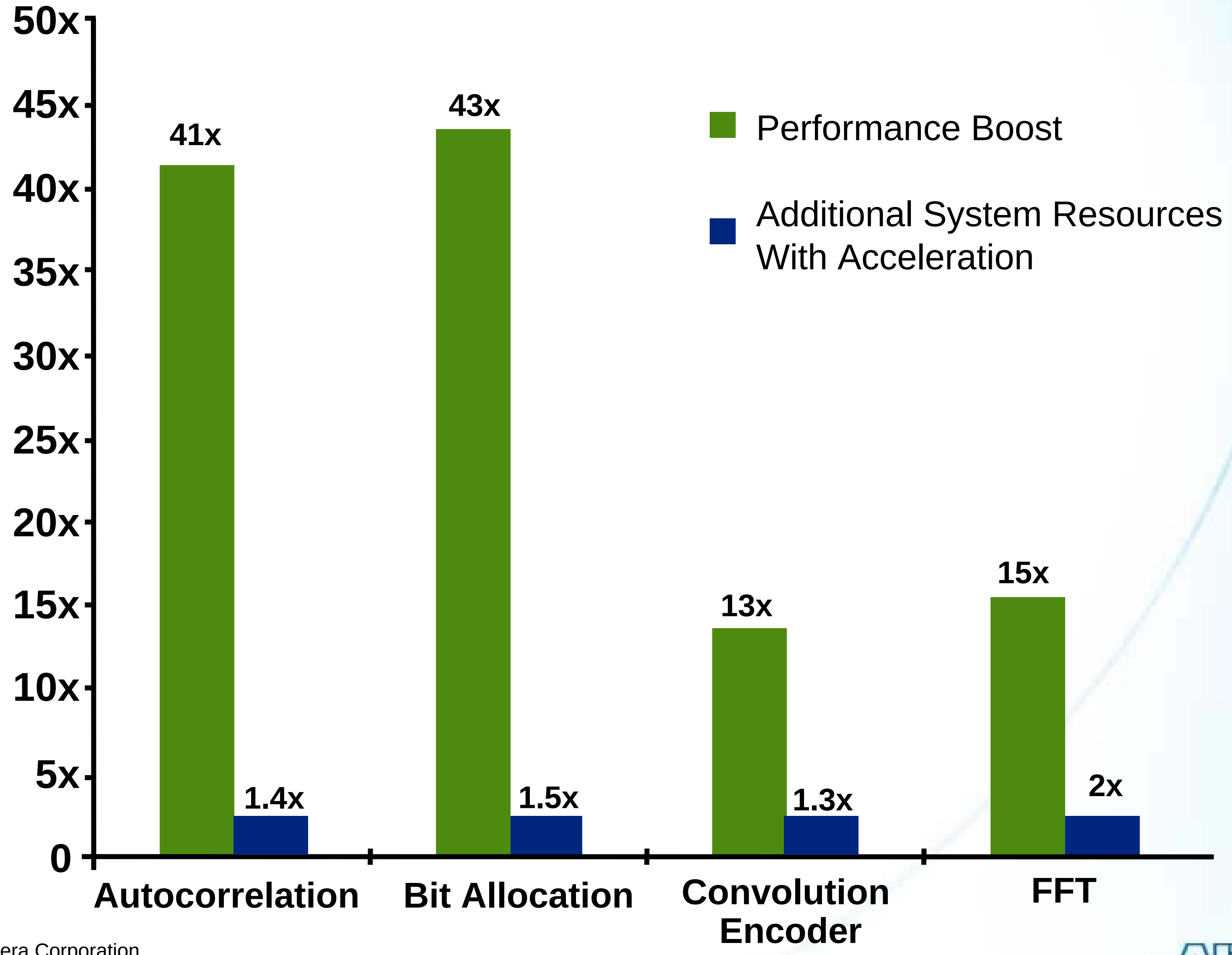
- Profile Code
- Identify Bottlenecks
- Re-partition Memory
- Create/Modify an Accelerator
  - HDL
    - Design
    - Simulate
    - Integrate
  - Software driver
    - Write function
    - Integrate
    - Re-build ‘C’ code
  - Import into SOPC Builder
- Repeat based upon results

C2H Flow

- Profile Code
- Identify Bottlenecks
- Re-partition Memory
- Select C function
  - Right click to accelerate
  - Build
- Repeat based upon results

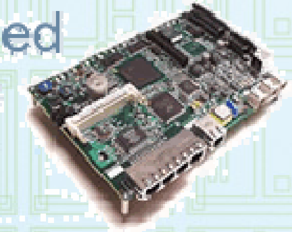


Dramatic Performance Boost



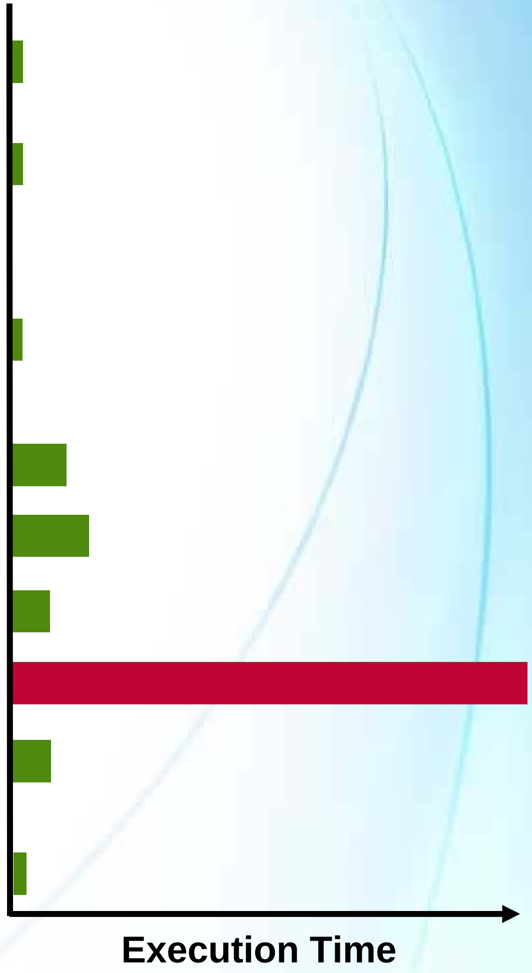
How To Use The C2H Compiler

the embedded masterclass

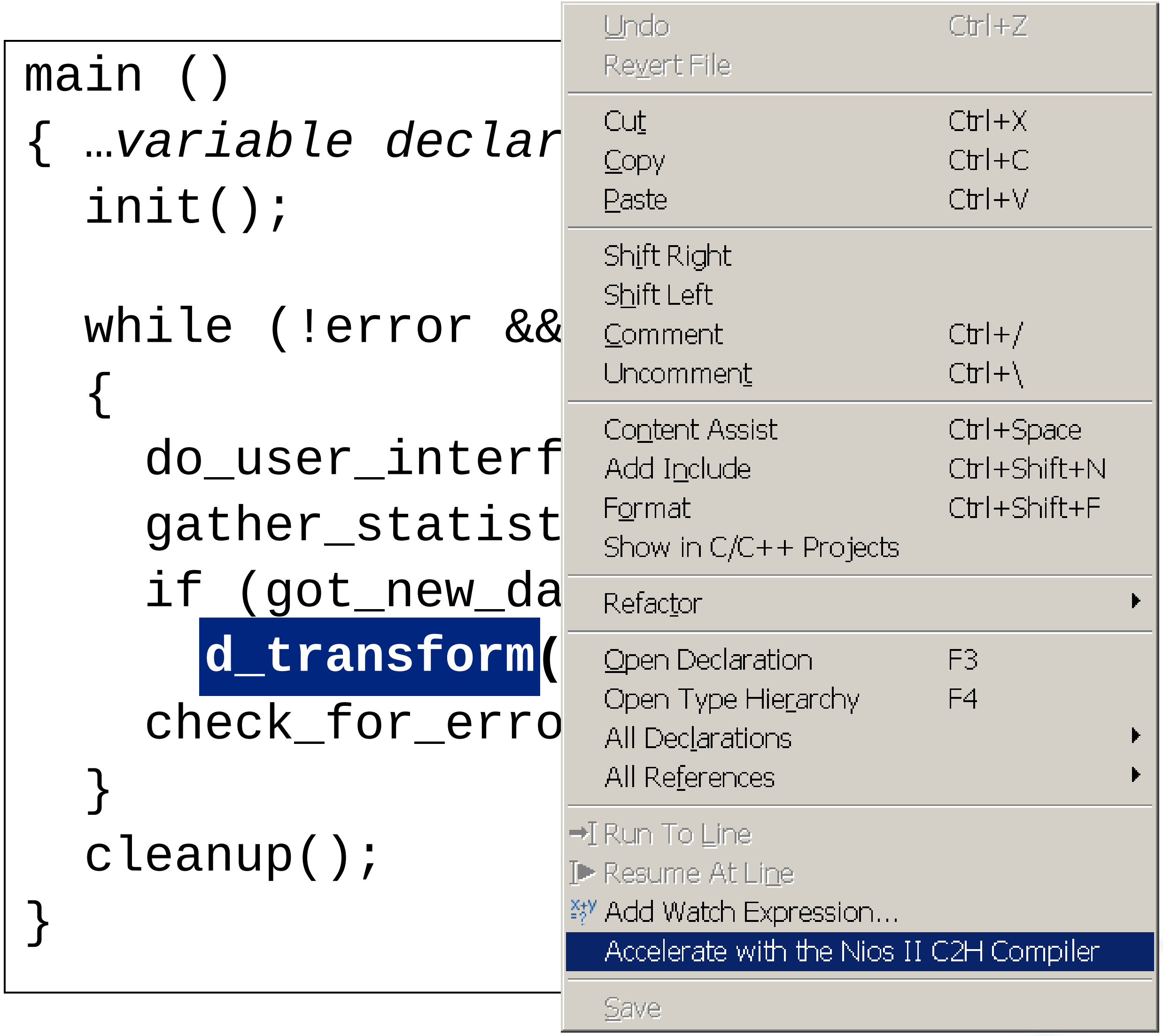


Step 1: Identify Software Bottlenecks

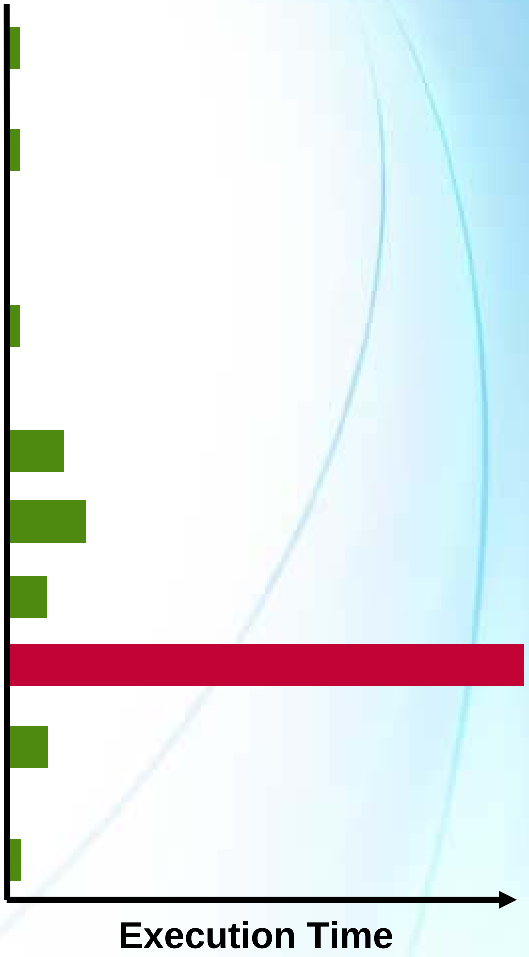
```
main ()
{ …variable declarations…
  init();

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



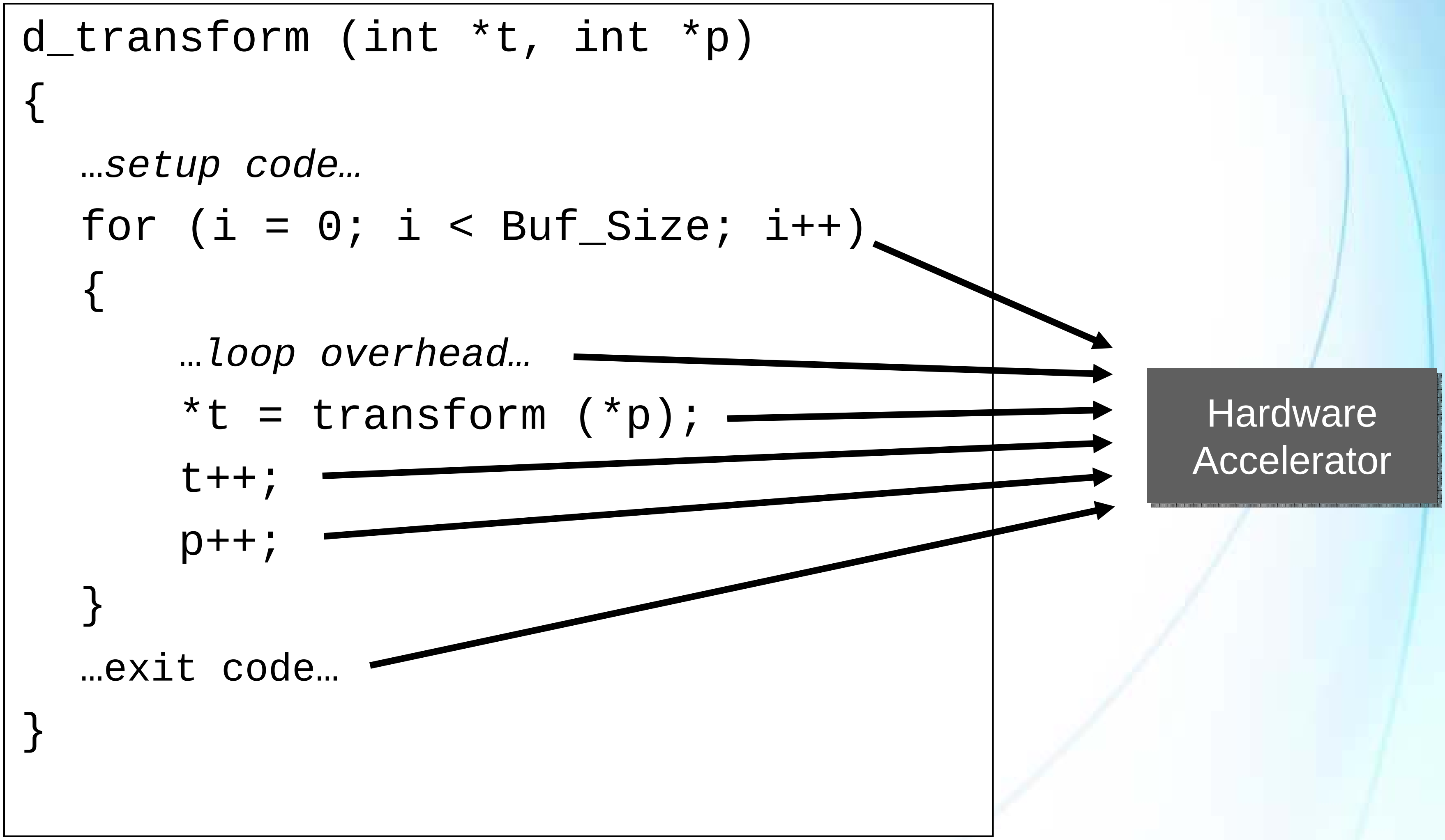
Step 2: Right-Click to *Accelerate*



```
main ()
{ …variable declar
  init();

  while (!error &&
  {
    do_user_interf
    gather_statist
    if (got_new_da
      d_transform(
    check_for_erro
  }
  cleanup();
}
```

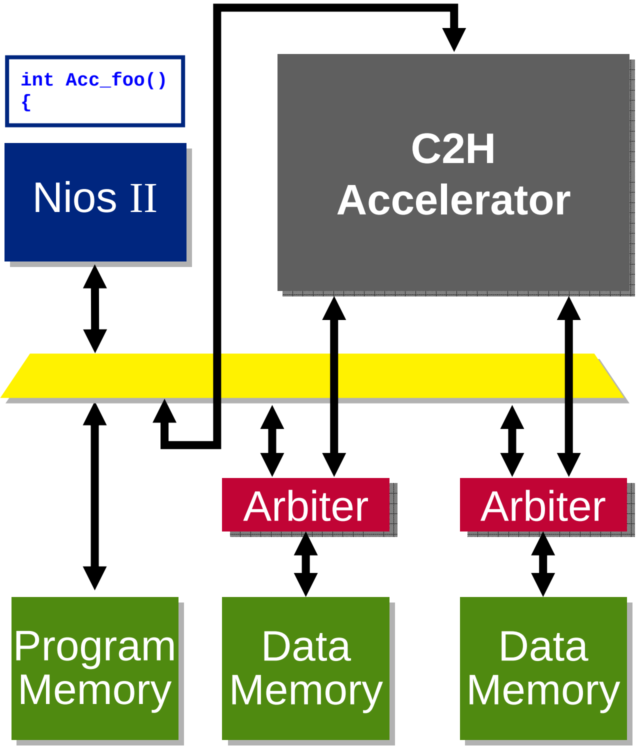


Build Hardware Unit for Function



```
d_transform (int *t, int *p)
{
   …setup code…
   for (i = 0; i < Buf_Size; i++)
   {
       …loop overhead…
       *t = transform (*p);
       t++;
       p++;
   }
   …exit code…
}
```

Integrate Into System



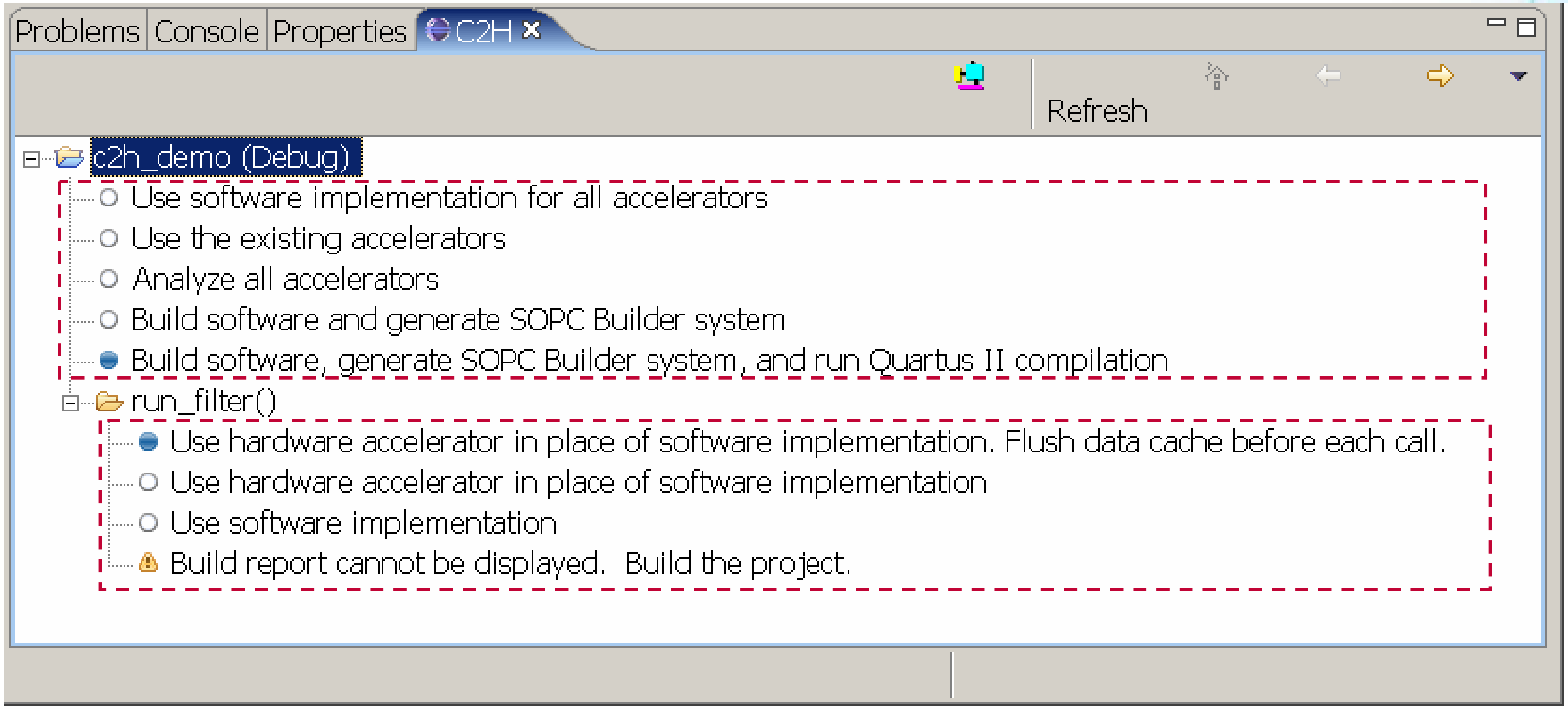
1. Generate SOPC Component
2. Integrate into HW system
3. Generate SW control function
4. Integrate into SW build
5. Build Software and/or Hardware

SOPC Builder
Concept to System in Minutes

QUARTUS® II

Select C2H Build and Run Options

- Accelerated functions tab gets created
  - Select desired options then **Build project**



c2h_demo (Debug)

- ○ 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

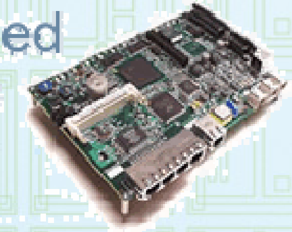
run_filter()

- ● Use hardware accelerator in place of software implementation. Flush data cache before each call.
- ○ Use hardware accelerator in place of software implementation
- ○ Use software implementation
- ⚠ Build report cannot be displayed. Build the project.

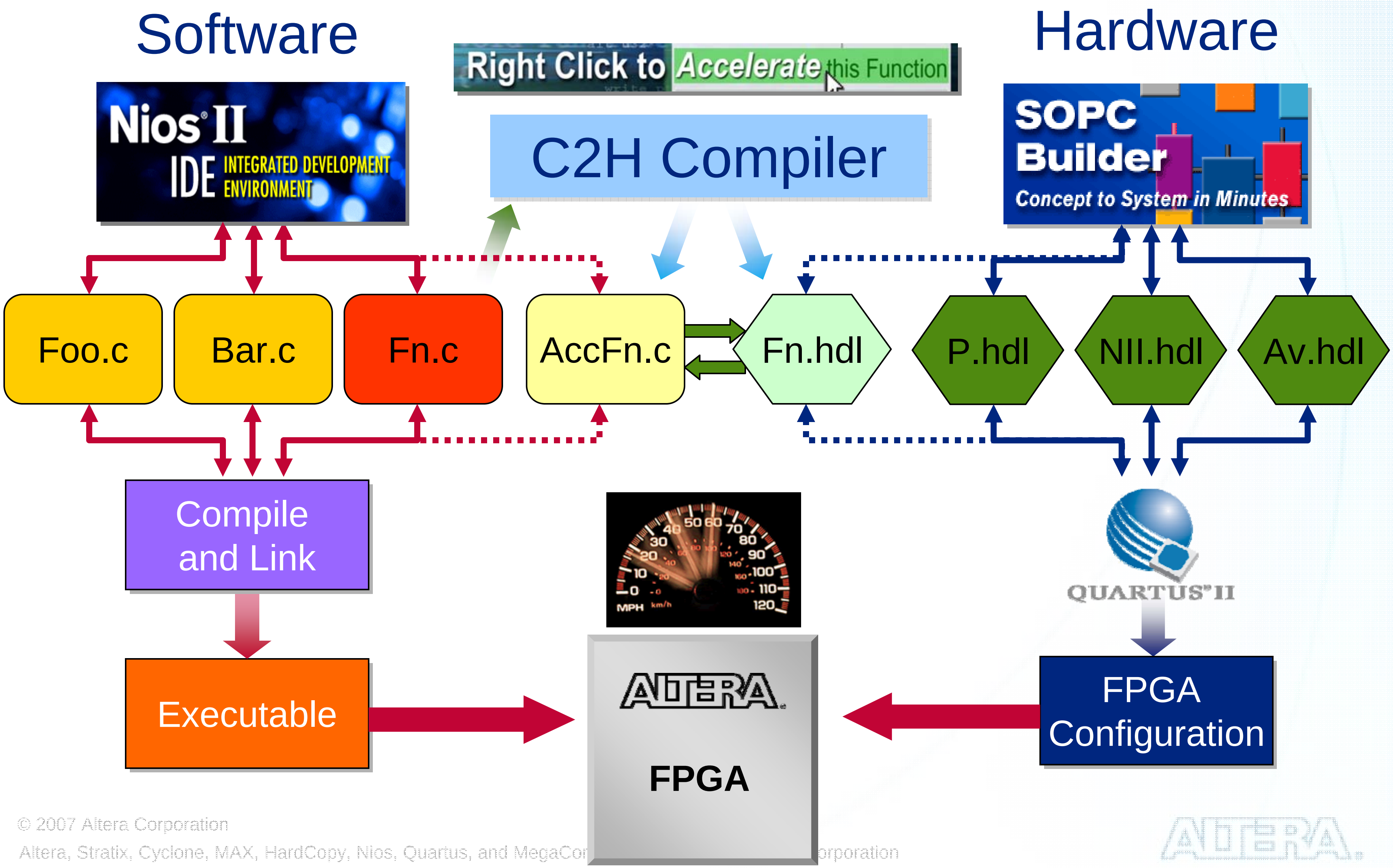
ALTERA

How The C2H Compiler Works

the embedded masterclass



Automated Acceleration With C2H



Software

Hardware

Right Click to Accelerate this Function

C2H Compiler

Nios II IDE INTEGRATED DEVELOPMENT ENVIRONMENT

SOPC Builder Concept to System in Minutes

Foo.c

Bar.c

Fn.c

AccFn.c

Fn.hdl

P.hdl

NII.hdl

Av.hdl

Compile and Link

Executable

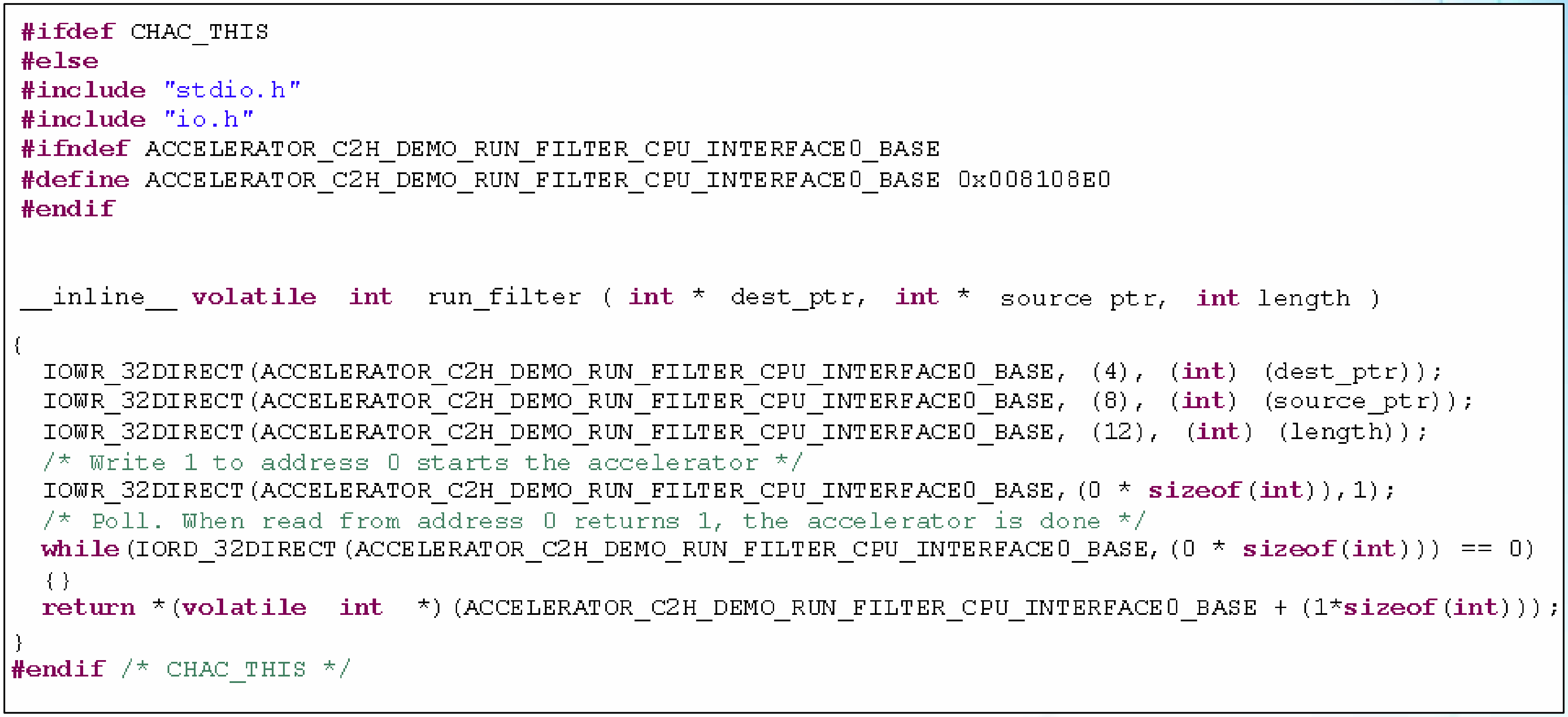
QUARTUS II

FPGA Configuration

ALTERA FPGA

HW Accelerator Software Wrapper File

- In **Application** or **Release** directory in Nios II IDE



```
#ifdef CHAC_THIS
#else
#include "stdio.h"
#include "io.h"
#ifndef ACCELERATOR_C2H_DEMO_RUN_FILTER_CPU_INTERFACE0_BASE
#define ACCELERATOR_C2H_DEMO_RUN_FILTER_CPU_INTERFACE0_BASE 0x008108E0
#endif


__inline__ volatile  int  run_filter ( int * dest_ptr,  int *  source ptr,  int length )
{
  IOWR_32DIRECT(ACCELERATOR_C2H_DEMO_RUN_FILTER_CPU_INTERFACE0_BASE, (4), (int) (dest_ptr));
  IOWR_32DIRECT(ACCELERATOR_C2H_DEMO_RUN_FILTER_CPU_INTERFACE0_BASE, (8), (int) (source_ptr));
  IOWR_32DIRECT(ACCELERATOR_C2H_DEMO_RUN_FILTER_CPU_INTERFACE0_BASE, (12), (int) (length));
  /* Write 1 to address 0 starts the accelerator */
  IOWR_32DIRECT(ACCELERATOR_C2H_DEMO_RUN_FILTER_CPU_INTERFACE0_BASE,(0 * sizeof(int)),1);
  /* Poll. When read from address 0 returns 1, the accelerator is done */
  while(IORD_32DIRECT(ACCELERATOR_C2H_DEMO_RUN_FILTER_CPU_INTERFACE0_BASE,(0 * sizeof(int))) == 0)
  {}
  return *(volatile  int  *)(ACCELERATOR_C2H_DEMO_RUN_FILTER_CPU_INTERFACE0_BASE + (1*sizeof(int)));
}
#endif /* CHAC_THIS */
```

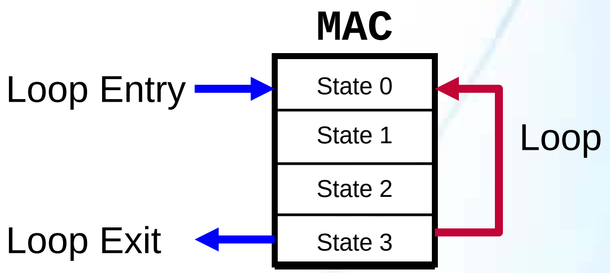
Logic Generation

- “What you type is what you get”
- Every arithmetic operator you type…
  - Makes equivalent hardware unit in accelerator
    - *, +, %, >>
- Control-flow statements generate control state machine
- Pointer and array references create memory master interfaces
- Results strongly depend on target algorithm and coding style
  - Some algorithms accelerate better than others
  - Coding style impacts size and speed of accelerator

Example:

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

- ◼ 1 - 32x32 multiplier
- 🟩 3 - 32-bit incrementers
- 🟦 1 - 64-bit adder
- 🟥 1 - 32-bit comparator
- 🟨 2 - read-masters
- ■ Nominal control logic

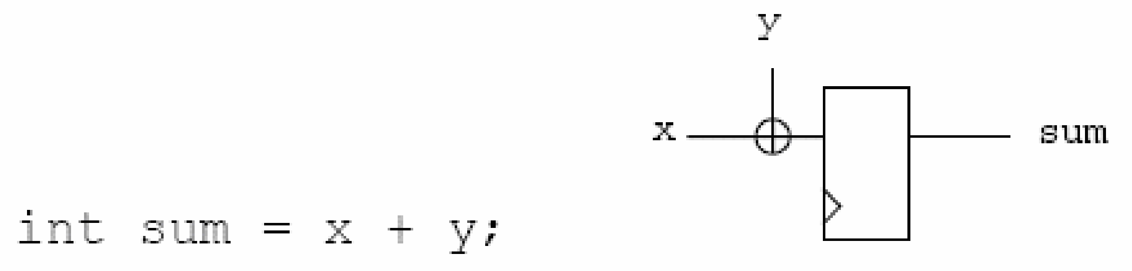


Every converted software function has its own dedicated hardware state machine

Assignments

General rule:

- “=” operator translates into a registered HW operation
- Calculation of the value takes one clock cycle

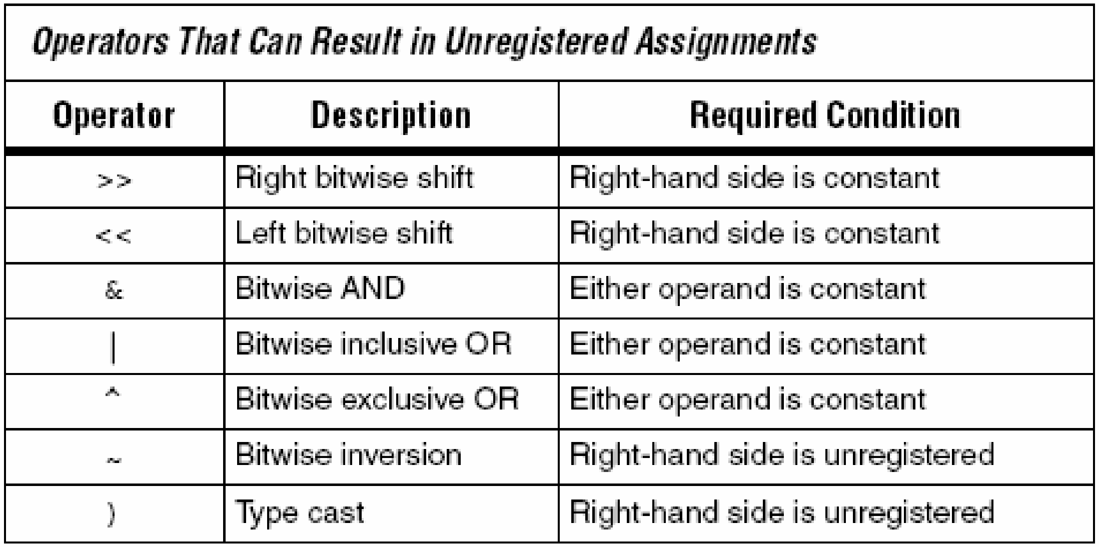


```
int sum = x + y;
```

- Two exceptions:
  - Assignments that require zero logic elements in hardware (0 cycles)
  - Assignments that use complex arithmetic,
    these require logic that can take multiple cycles (>1 cycle)

Unregistered Operations / Assignments

- Certain logical and bitwise operations involving constants are trivial and require no logic
  - In hardware, they are performed by manipulating wires
  - If an assignment consists solely of such operations (see table below), then its result is not registered



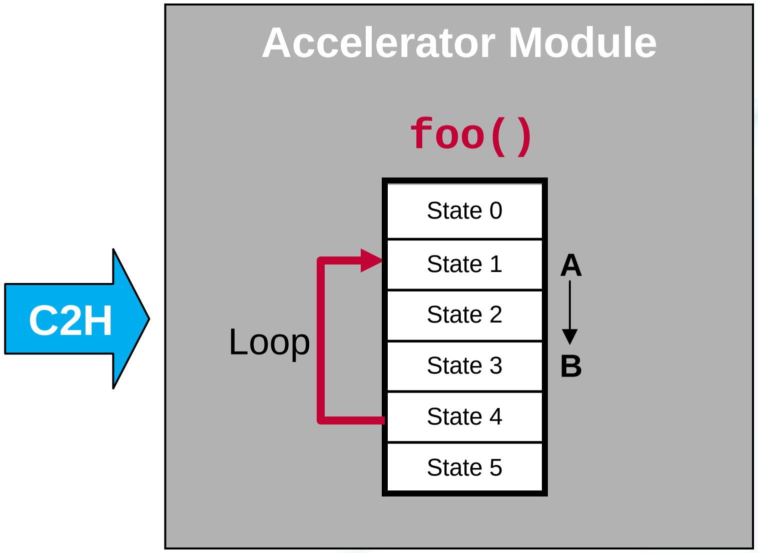
Operators That Can Result in Unregistered Assignments

Operator	Description	Required Condition
>>	Right bitwise shift	Right-hand side is constant
<<	Left bitwise shift	Right-hand side is constant
&	Bitwise AND	Either operand is constant
|	Bitwise inclusive OR	Either operand is constant
^	Bitwise exclusive OR	Either operand is constant
~	Bitwise inversion	Right-hand side is unregistered
)	Type cast	Right-hand side is unregistered

Mapping Software To Hardware

- Software operations are assigned to hardware states
- Multiple software operations can be assigned to single hardware state (executed in parallel)
- Parallel execution is limited by data dependencies
  - Eg. If operation B depends on a value calculated in operation A, then B cannot execute until A has completed

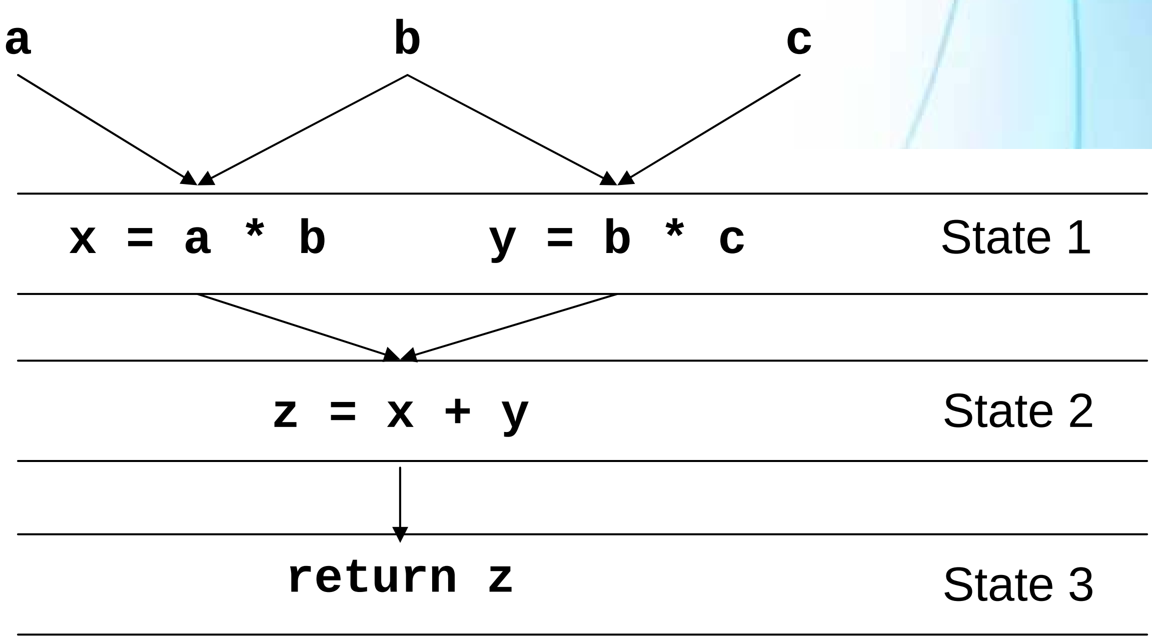
```
int foo(int data_count)
{ …variable declarations…
  count = data_count;
  while (count)
  {
    ..
    ..
    ..
    count--;
  }
  return result/data_count;
}
```



Introduction To Data Dependencies

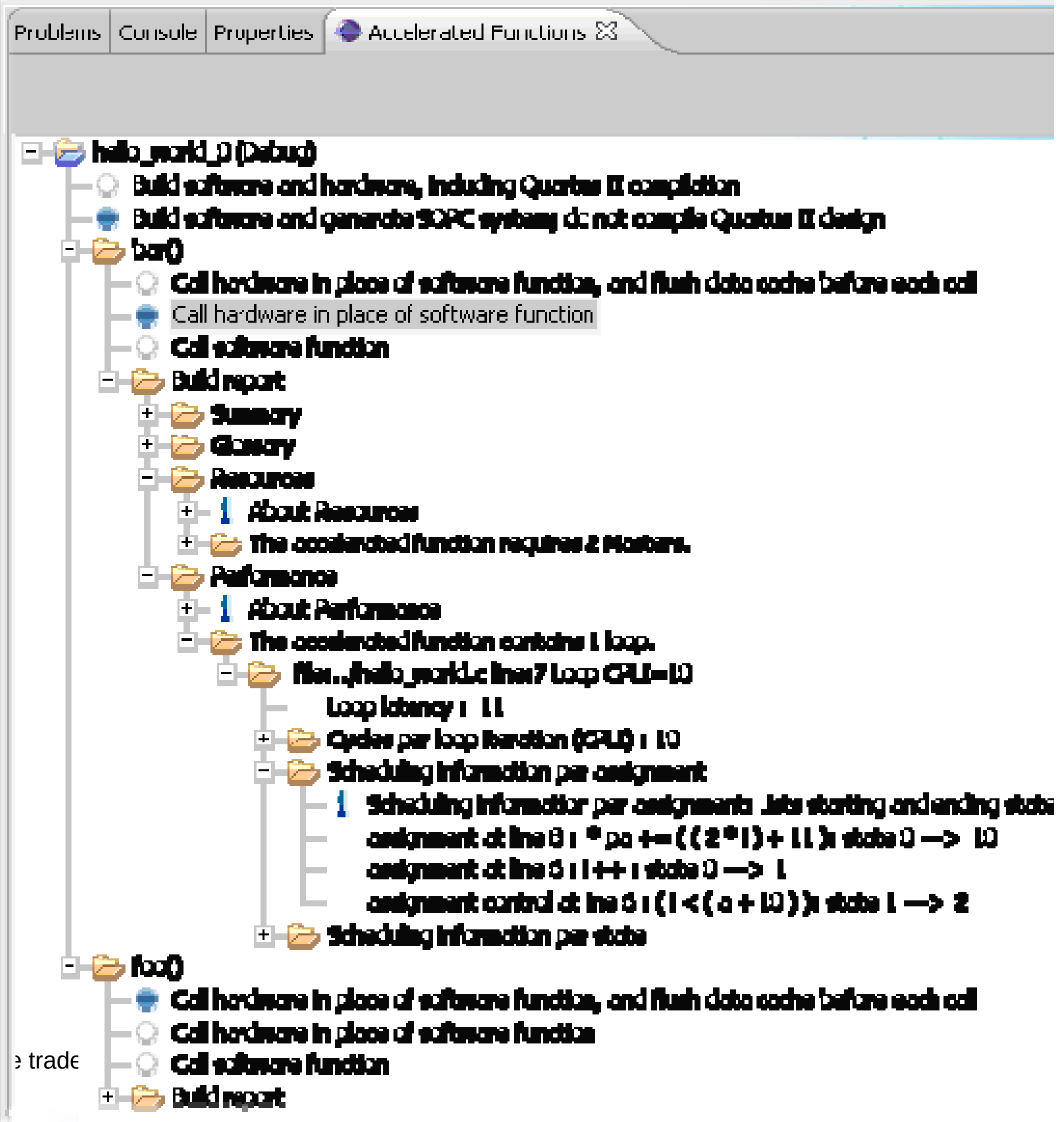
```
int foo(int a, int b, int c)
{
  int x ,y, z;
  x = a * b;
  y = b * c;
  z = x + y;
  return z;
}
```

States assigned directly from dependency graph



Performance/Resource Report

- C2H compiler results:
  - Logic created
    - Masters
    - Multipliers
  - Mapping of ‘C’ code to hardware states
  - Loop latency
  - Clocks per loop iteration (CPLI)

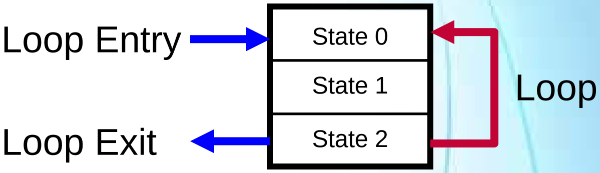


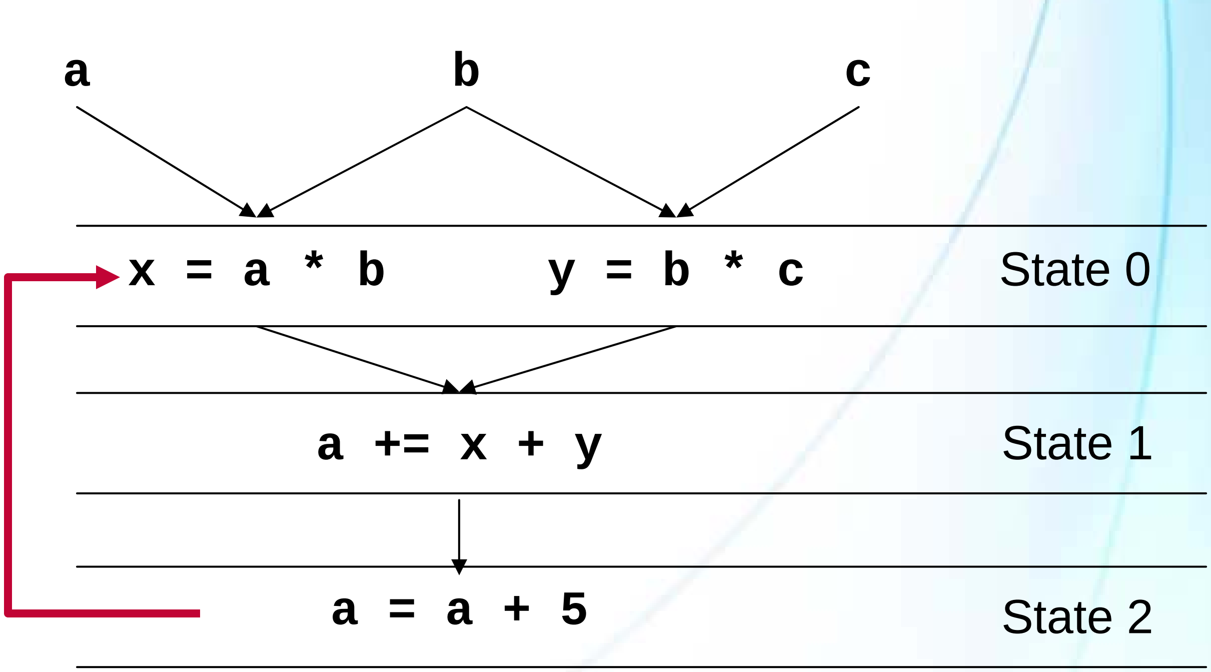
```
hello_world_0 (Debug)
  ○ Build software and hardware, including Quartus II compilation
  ● Build software and generate SOPC system; do not compile Quartus II design
  bar()
    ○ Call hardware in place of software function, and flush data cache before each call
    ● Call hardware in place of software function
    ○ Call software function
    Build report
      Summary
      Glossary
      Resources
        About Resources
        The accelerated function requires 2 Masters.
      Performance
        About Performance
        The accelerated function contains 1 loop.
          file:../hello_world.c line:7 Loop CPLI=10
            Loop latency : 11
            Cycles per loop iteration (CPLI) : 10
            Scheduling information per assignment
              Scheduling information per assignments: Lists starting and ending state
              assignment at line 8 : *pa += ((2*i) + 11); state 0 --> 10
              assignment at line 6 : i++ ; state 0 --> 1
              assignment control at line 6 : (i < (a + 10)); state 1 --> 2
            Scheduling information per state
  foo()
    ● Call hardware in place of software function, and flush data cache before each call
    ○ Call hardware in place of software function
    ○ Call software function
    Build report
```

Looping Data Dependencies

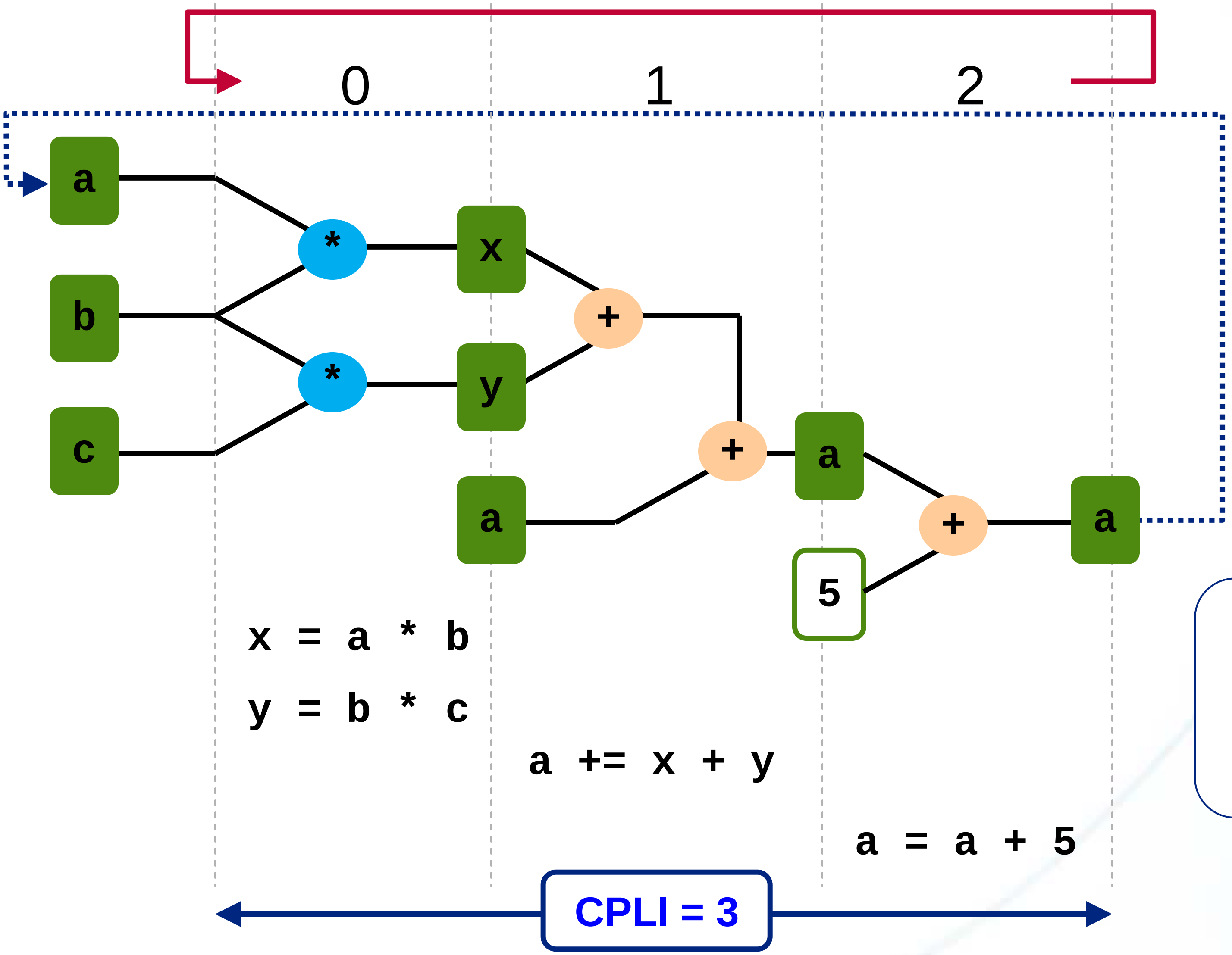
```
int foo( int a,
         int b,
         int c )
{
  int x, y;
  int i = 0;

  while (i < 5)
  {
    x = a * b;
    y = b * c;
    a += x + y;
    a = a + 5;
    i++;
  }
  return a;
}
```

States assigned directly from dependency graph





CPLI = Clocks Per Loop Iteration



x = a * b

y = b * c

a += x + y

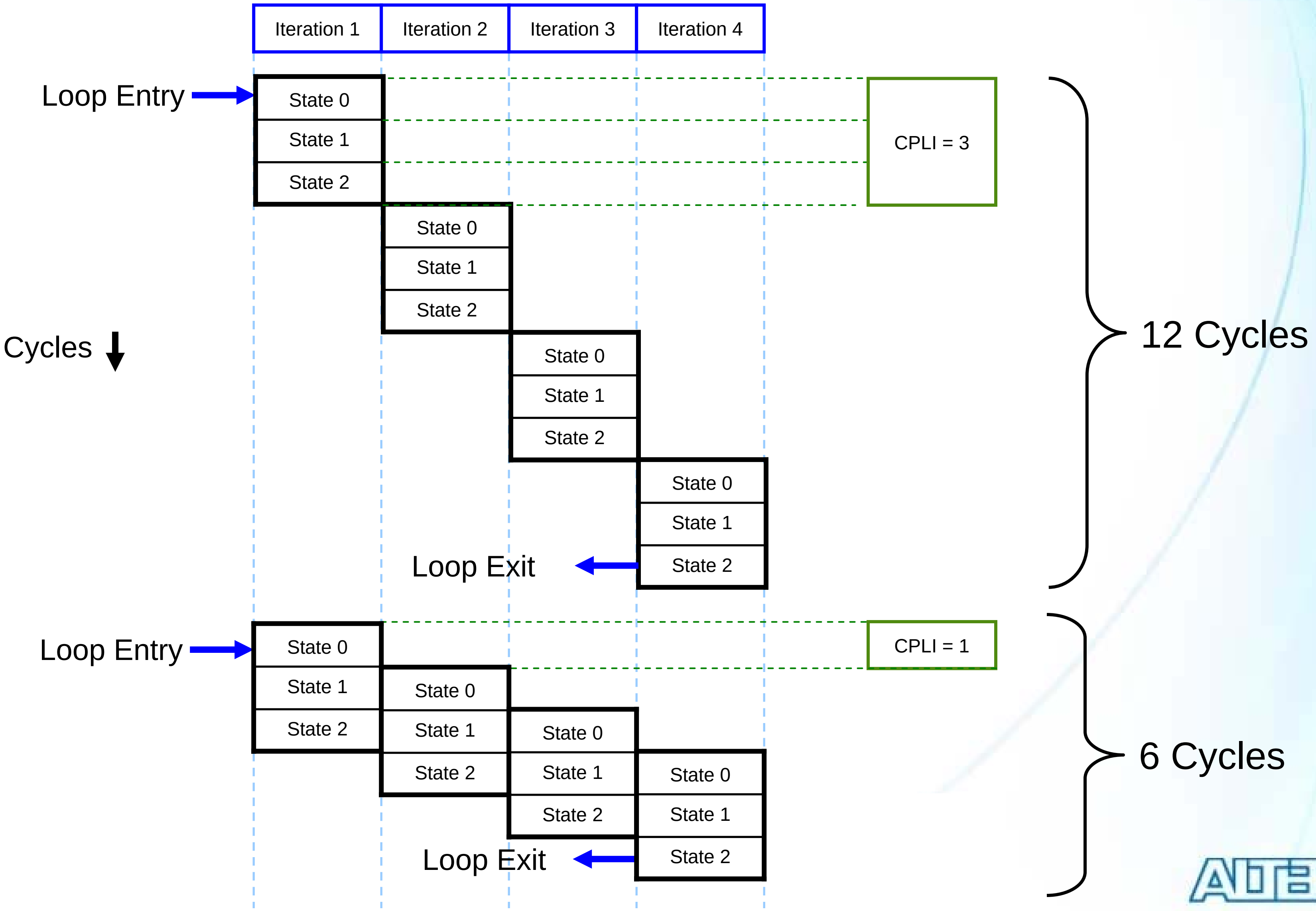
a = a + 5

CPLI = 3

2 Multipliers

3 Adders

Hardware Pipelines And CPLI



Iteration 1 | Iteration 2 | Iteration 3 | Iteration 4

Loop Entry

State 0
State 1
State 2

State 0
State 1
State 2

State 0
State 1
State 2

State 0
State 1
State 2

Loop Exit

CPLI = 3

12 Cycles

Cycles

Loop Entry

State 0
State 1
State 2

State 0
State 1
State 2

State 0
State 1
State 2

State 0
State 1
State 2

Loop Exit

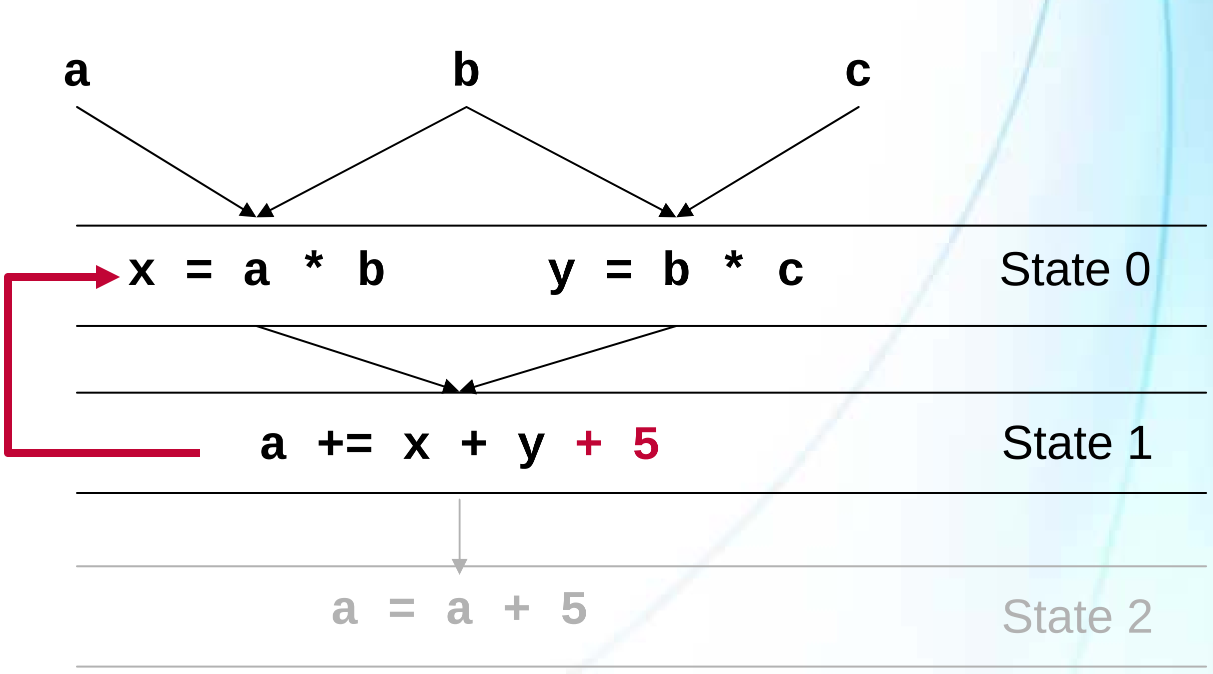
CPLI = 1

6 Cycles

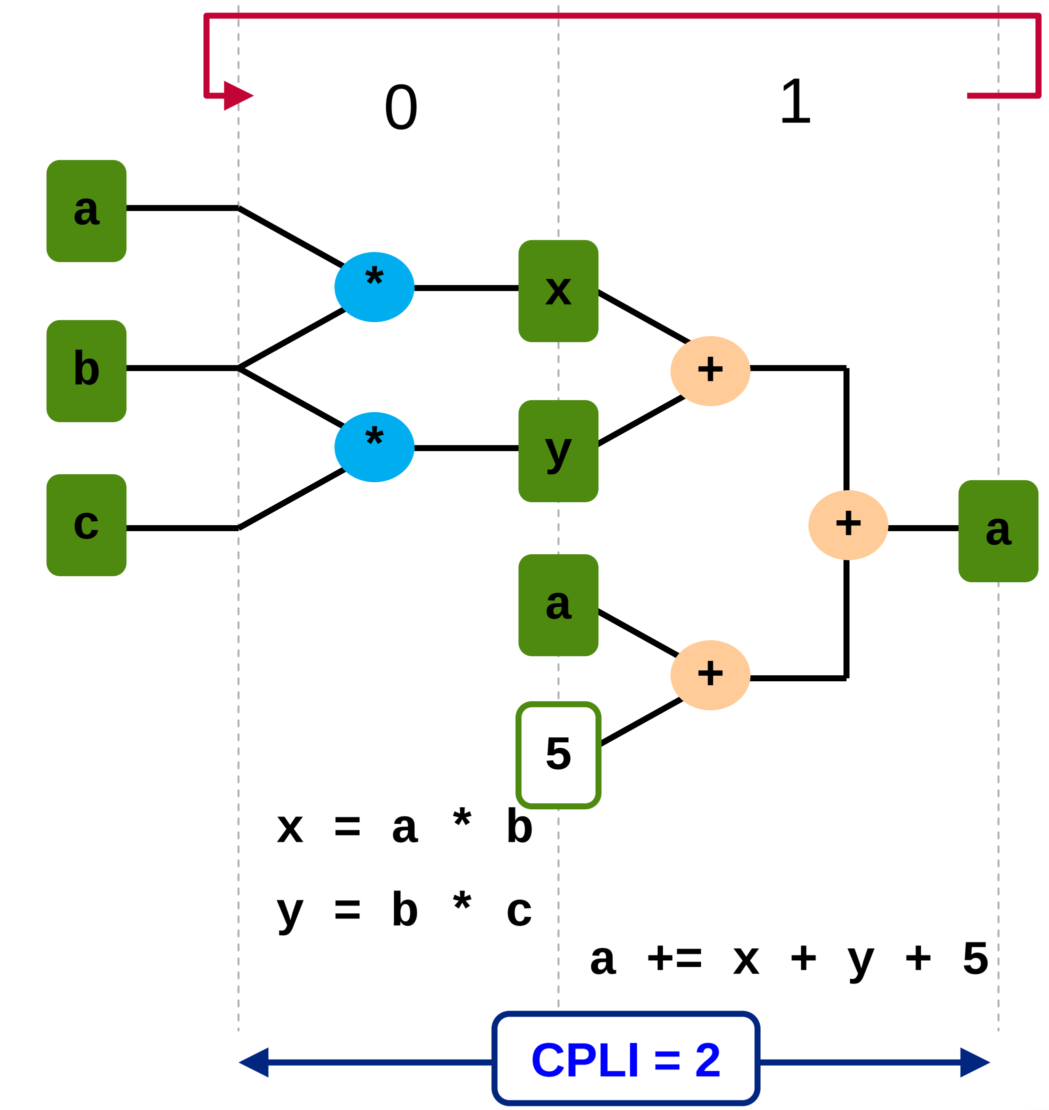
Optimising Data Dependencies

```
int foo( int a,
         int b,
         int c)
{
  int x, y;
  int i = 0;

  while (i < 5)
  {
    x = a * b;
    y = b * c;
    a += x + y + 5;
//  a = a + 5;
    i++;
  }
  return a;
}
```

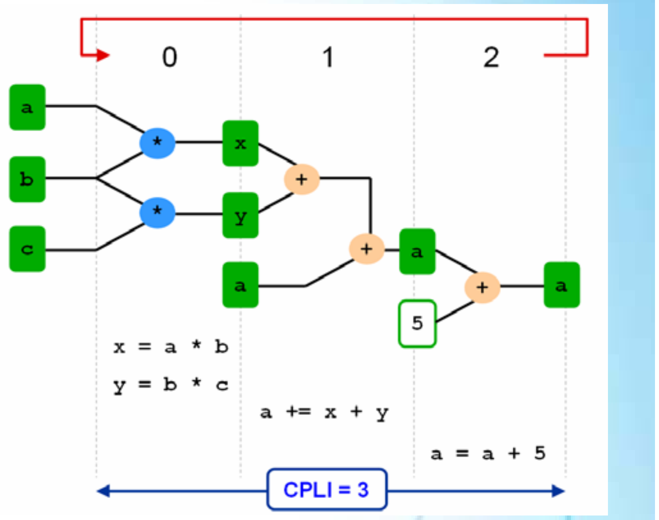
Change code to eliminate last stage



CPLI Optimised



```
x = a * b
y = b * c
a += x + y + 5
```



```
x = a * b
y = b * c
a += x + y
a = a + 5
```

CPLI = 2

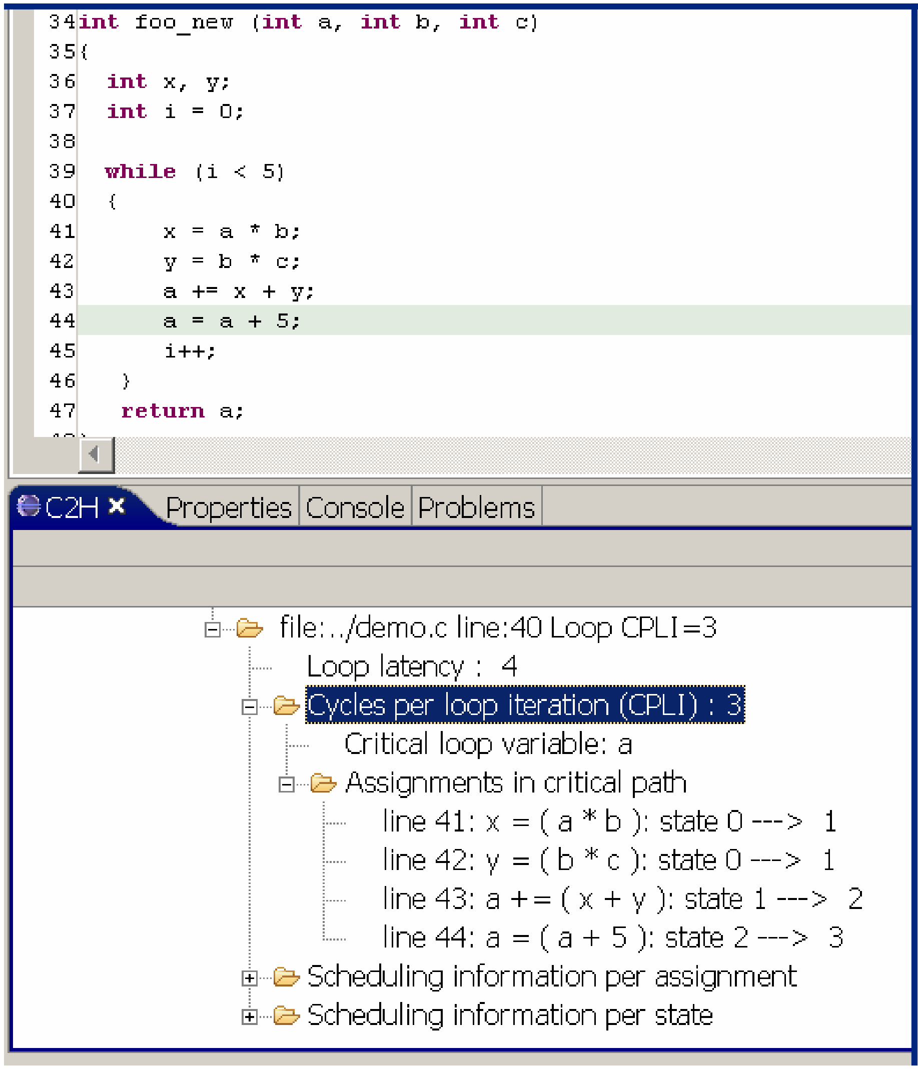
2 Multipliers

3 Adders

Optimising CPLI

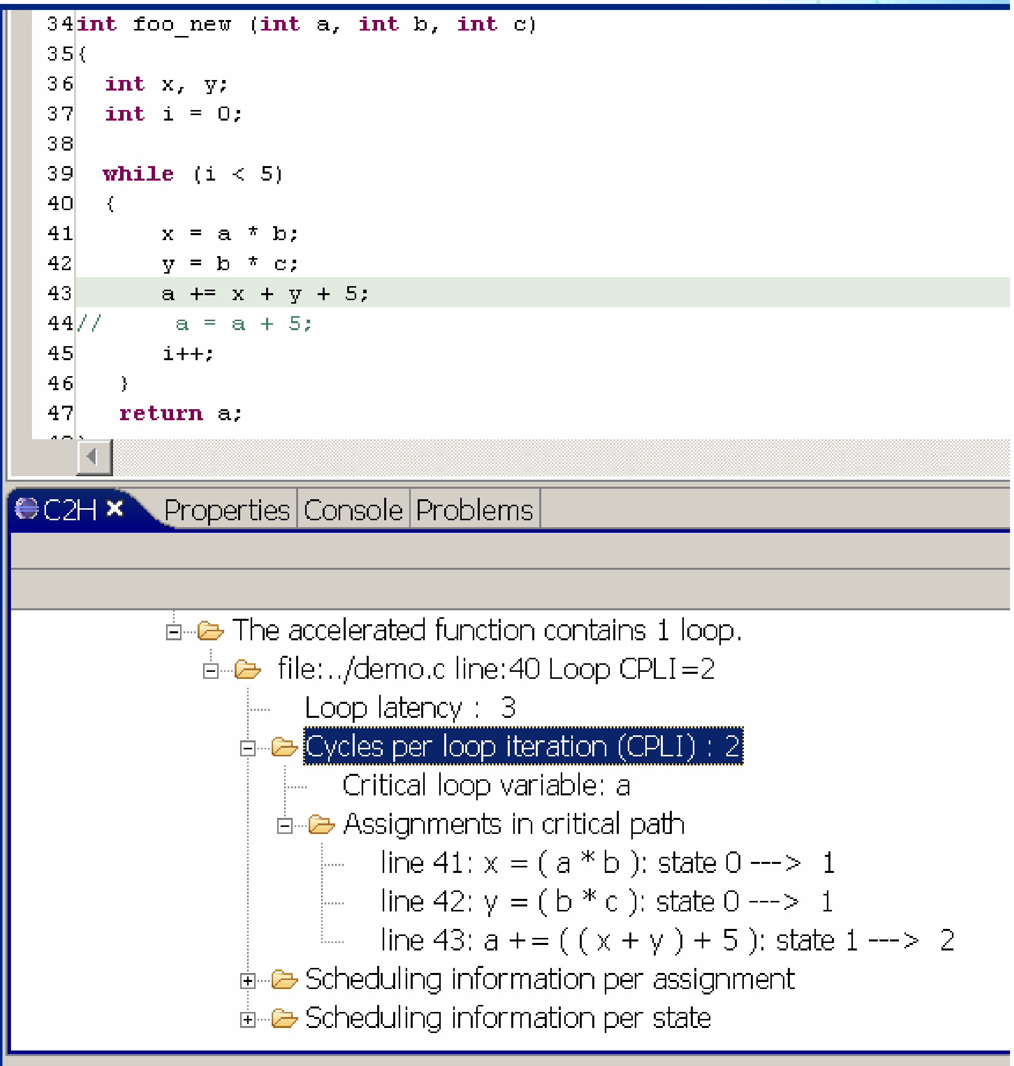
CPLI = 3

CPLI = 2



```
34 int foo_new (int a, int b, int c)
35 {
36   int x, y;
37   int i = 0;
38
39   while (i < 5)
40   {
41       x = a * b;
42       y = b * c;
43       a += x + y;
44       a = a + 5;
45       i++;
46    }
47    return a;
```

```
file:../demo.c line:40 Loop CPLI=3
    Loop latency :  4
    Cycles per loop iteration (CPLI) : 3
        Critical loop variable: a
        Assignments in critical path
            line 41: x = ( a * b ): state 0 ---> 1
            line 42: y = ( b * c ): state 0 ---> 1
            line 43: a += ( x + y ): state 1 ---> 2
            line 44: a = ( a + 5 ): state 2 ---> 3
    Scheduling information per assignment
    Scheduling information per state
```



```
34 int foo_new (int a, int b, int c)
35 {
36   int x, y;
37   int i = 0;
38
39   while (i < 5)
40   {
41       x = a * b;
42       y = b * c;
43       a += x + y + 5;
44 //     a = a + 5;
45       i++;
46    }
47    return a;
```

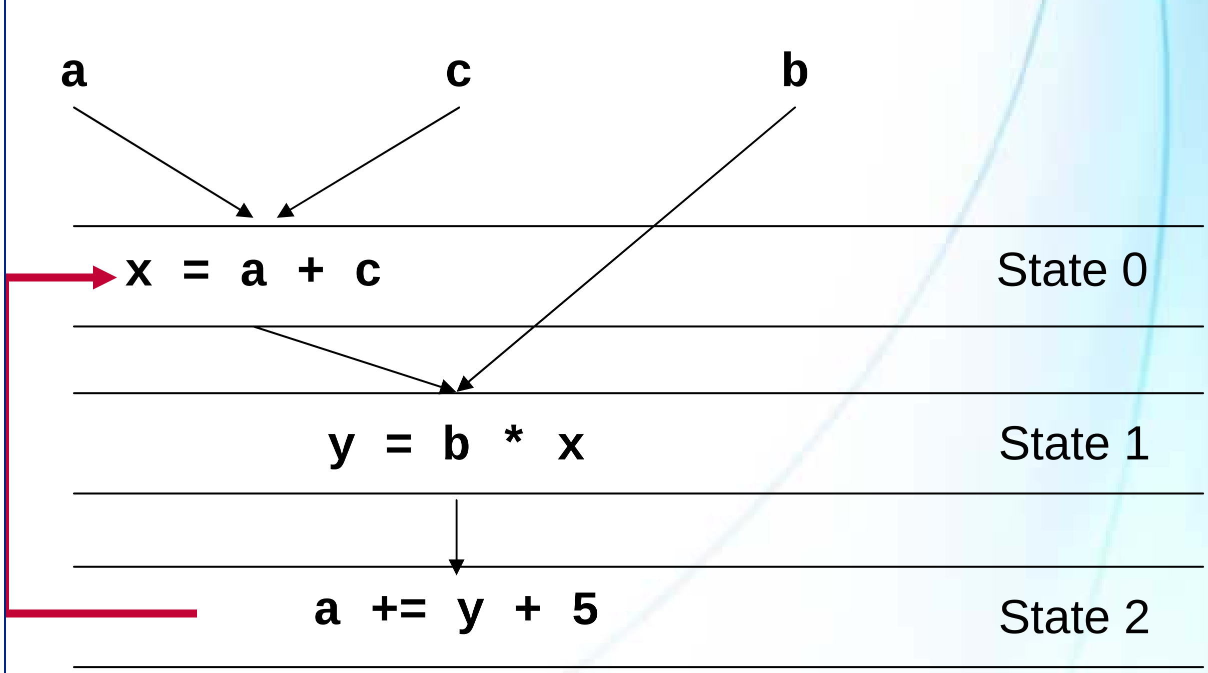
```
The accelerated function contains 1 loop.
    file:../demo.c line:40 Loop CPLI=2
        Loop latency :  3
        Cycles per loop iteration (CPLI) : 2
            Critical loop variable: a
            Assignments in critical path
                line 41: x = ( a * b ): state 0 ---> 1
                line 42: y = ( b * c ): state 0 ---> 1
                line 43: a += ( ( x + y ) + 5 ): state 1 ---> 2
        Scheduling information per assignment
        Scheduling information per state
```

Optimising Hardware Resource

```
int foo( int a,
         int b,
         int c )
{
  int x, y;
  int i = 0;

  while (i < 5)
  {
    x = a * b;
    y = b * c;
    a += x + y;
    a = a + 5;
    i++;
  }
  return a;
}
```

Change code to reduce number of multipliers

`(a * b) + (b * c) = b * (a + c)`



a

c

b

x = a + c

State 0

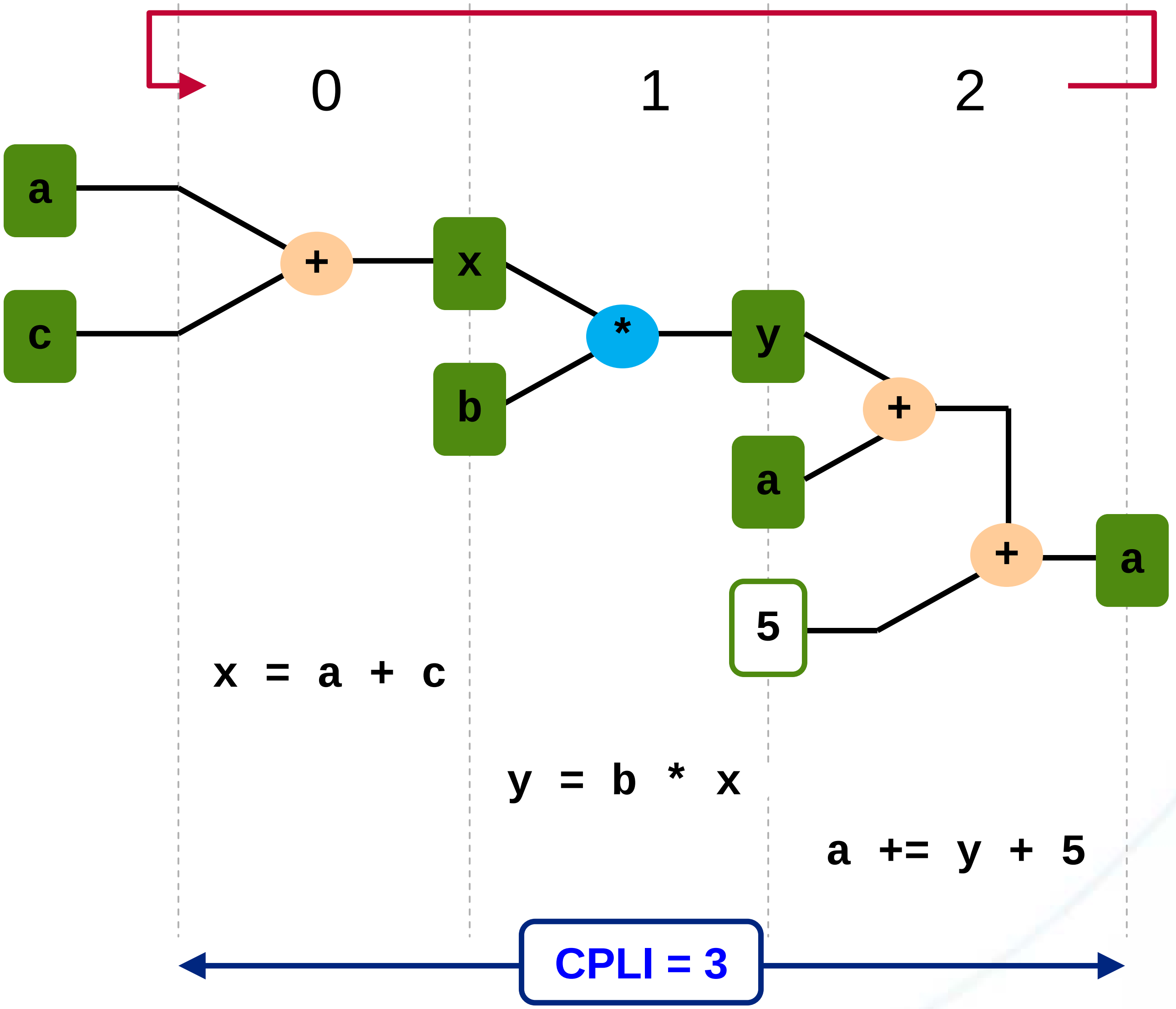
y = b * x

State 1

a += y + 5

State 2

Resource Optimised



x = a + c

y = b * x

a += y + 5

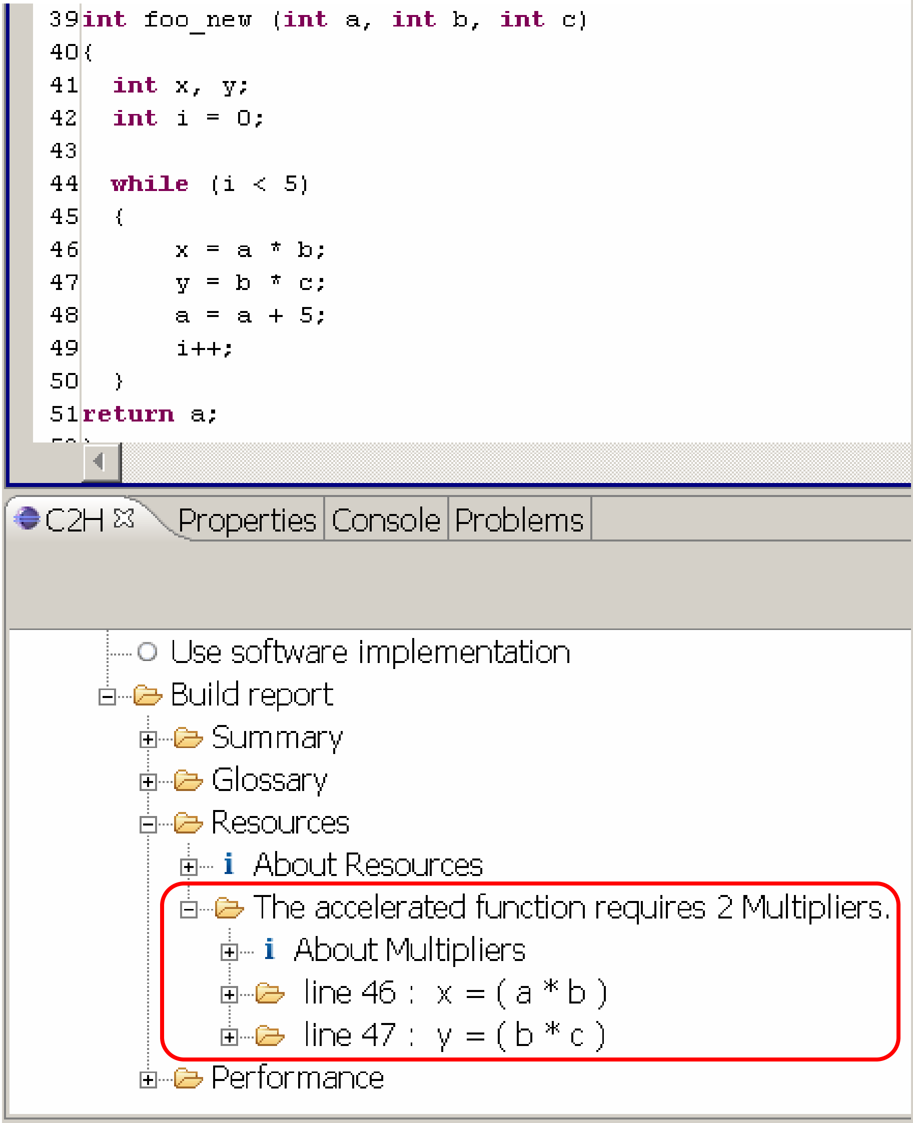
CPLI = 3

CPLI = 3

1 Multiplier

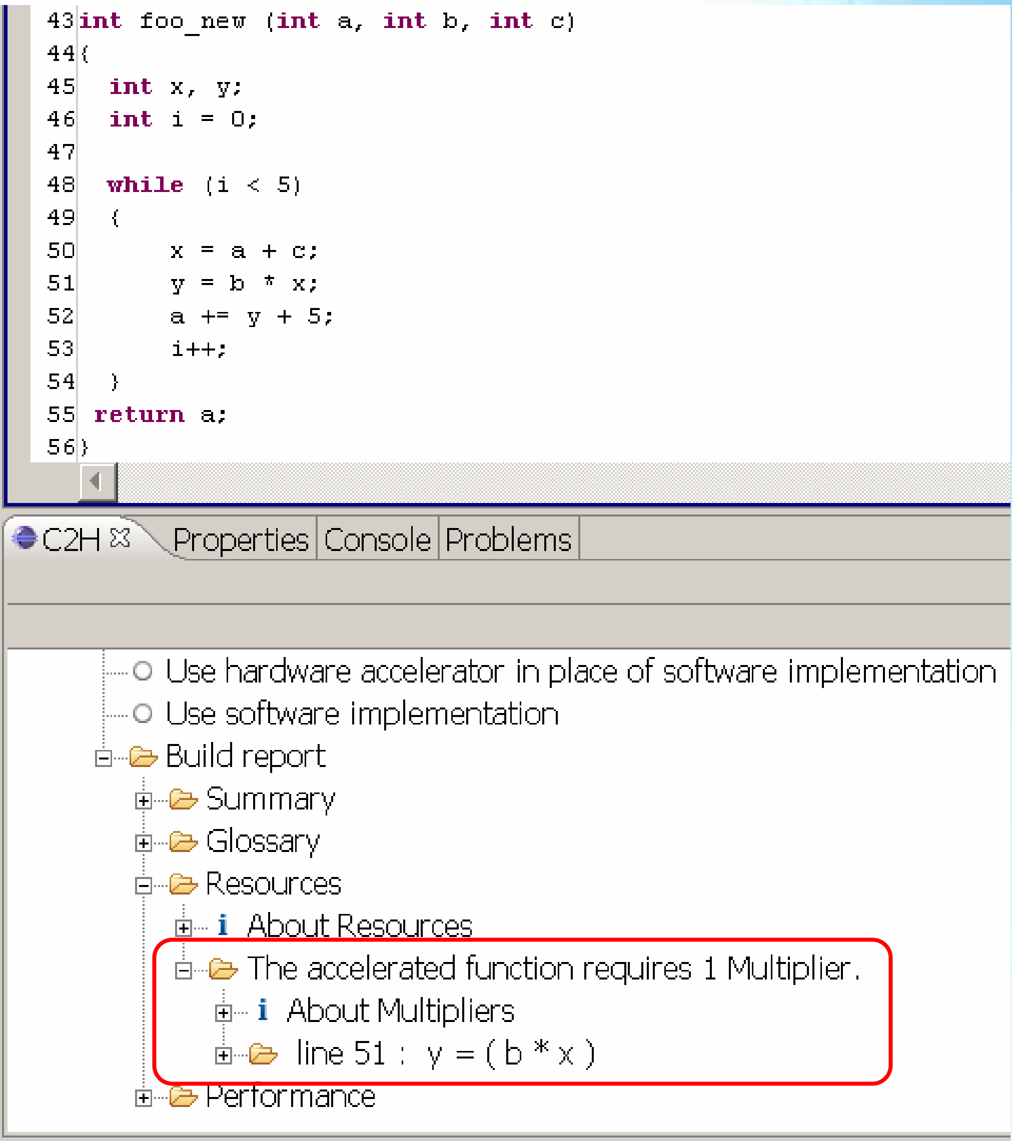
3 Adders

Optimising Resources



```
39int foo_new (int a, int b, int c)
40{
41   int x, y;
42   int i = 0;
43
44   while (i < 5)
45   {
46       x = a * b;
47       y = b * c;
48       a = a + 5;
49       i++;
50   }
51return a;
```

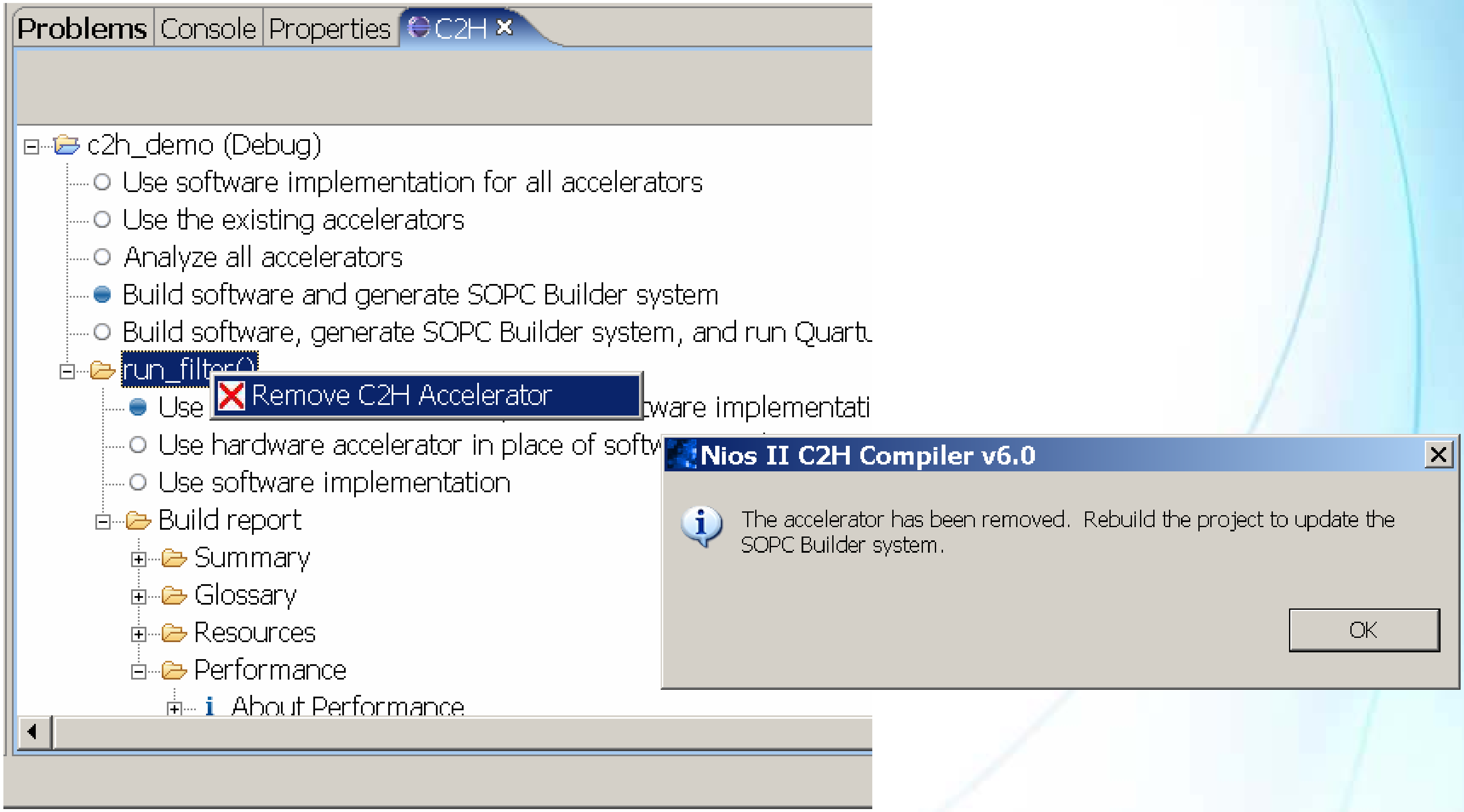
- ○ Use software implementation
- Build report
  - Summary
  - Glossary
  - Resources
    - i About Resources
    - The accelerated function requires 2 Multipliers.
      - i About Multipliers
      - line 46 : x = (a * b)
      - line 47 : y = (b * c)
  - Performance



```
43int foo_new (int a, int b, int c)
44{
45   int x, y;
46   int i = 0;
47
48   while (i < 5)
49   {
50       x = a + c;
51       y = b * x;
52       a += y + 5;
53       i++;
54   }
55  return a;
56}
```

- ○ Use hardware accelerator in place of software implementation
- ○ Use software implementation
- Build report
  - Summary
  - Glossary
  - Resources
    - i About Resources
    - The accelerated function requires 1 Multiplier.
      - i About Multipliers
      - line 51 : y = (b * x)
  - Performance

Removing the Accelerator



- c2h_demo (Debug)
  - ○ 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 Quartu
  - run_filter()
    - ● Use ... tware implementati
    - ○ Use hardware accelerator in place of softw
    - ○ Use software implementation
    - Build report
      - Summary
      - Glossary
      - Resources
      - Performance
        - About Performance

Nios II C2H Compiler v6.0

The accelerator has been removed. Rebuild the project to update the SOPC Builder system.

Benefits of C2H Compiler

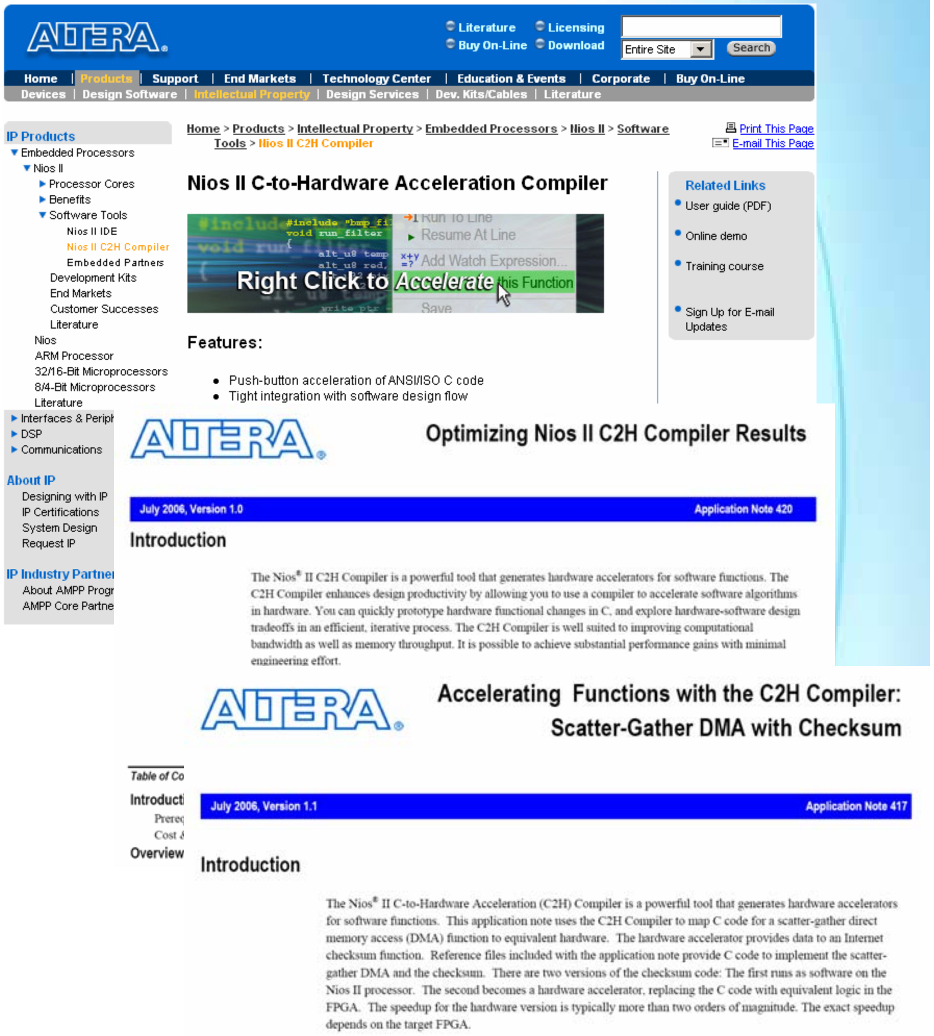
Improves productivity:

- Automated process
- Hit the performance target quicker
- Get more value out of the FPGA
- Finish designs sooner
- Accelerate legacy systems that are struggling to support new features

C2H Compiler:
High Productivity Hardware Acceleration

More Information on C2H

- www.altera.com/C2H
  - C2H Overview
  - C2H White paper
  - Online demonstration
  - C2H User Guide
  - Image rotation and FIR design examples
- 1 hour tutorial **"Nios II C2H Compiler Fundamentals"** www.altera.com/training
- **AN420:** Optimizing Nios II C2H Compiler Results (including design files)
- **AN 417:** Accelerating Functions with the C2H Compiler: Scatter-Gather DMA with Checksum (including design files)



Home > Products > Intellectual Property > Embedded Processors > Nios II > Software Tools > Nios II C2H Compiler

Nios II C-to-Hardware Acceleration Compiler

Features:

- Push-button acceleration of ANSI/ISO C code
- Tight integration with software design flow

Optimizing Nios II C2H Compiler Results

July 2006, Version 1.0 Application Note 420

Introduction

The Nios® II C2H Compiler is a powerful tool that generates hardware accelerators for software functions. The C2H Compiler enhances design productivity by allowing you to use a compiler to accelerate software algorithms in hardware. You can quickly prototype hardware functional changes in C, and explore hardware-software design tradeoffs in an efficient, iterative process. The C2H Compiler is well suited to improving computational bandwidth as well as memory throughput. It is possible to achieve substantial performance gains with minimal engineering effort.

Accelerating Functions with the C2H Compiler: Scatter-Gather DMA with Checksum

July 2006, Version 1.1 Application Note 417

Introduction

The Nios® II C-to-Hardware Acceleration (C2H) Compiler is a powerful tool that generates hardware accelerators for software functions. This application note uses the C2H Compiler to map C code for a scatter-gather direct memory access (DMA) function to equivalent hardware. The hardware accelerator provides data to an Internet checksum function. Reference files included with the application note provide C code to implement the scatter-gather DMA and the checksum. There are two versions of the checksum code: The first runs as software on the Nios II processor. The second becomes a hardware accelerator, replacing the C code with equivalent logic in the FPGA. The speedup for the hardware version is typically more than two orders of magnitude. The exact speedup depends on the target FPGA.

Thank You….

the embedded
masterclass