Design of Embedded Hardware and Firmware

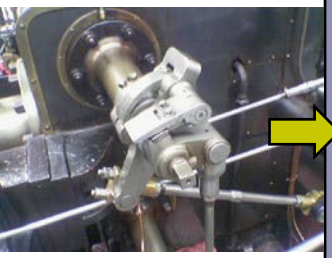
Custom Instructions

Andrea Guerrieri

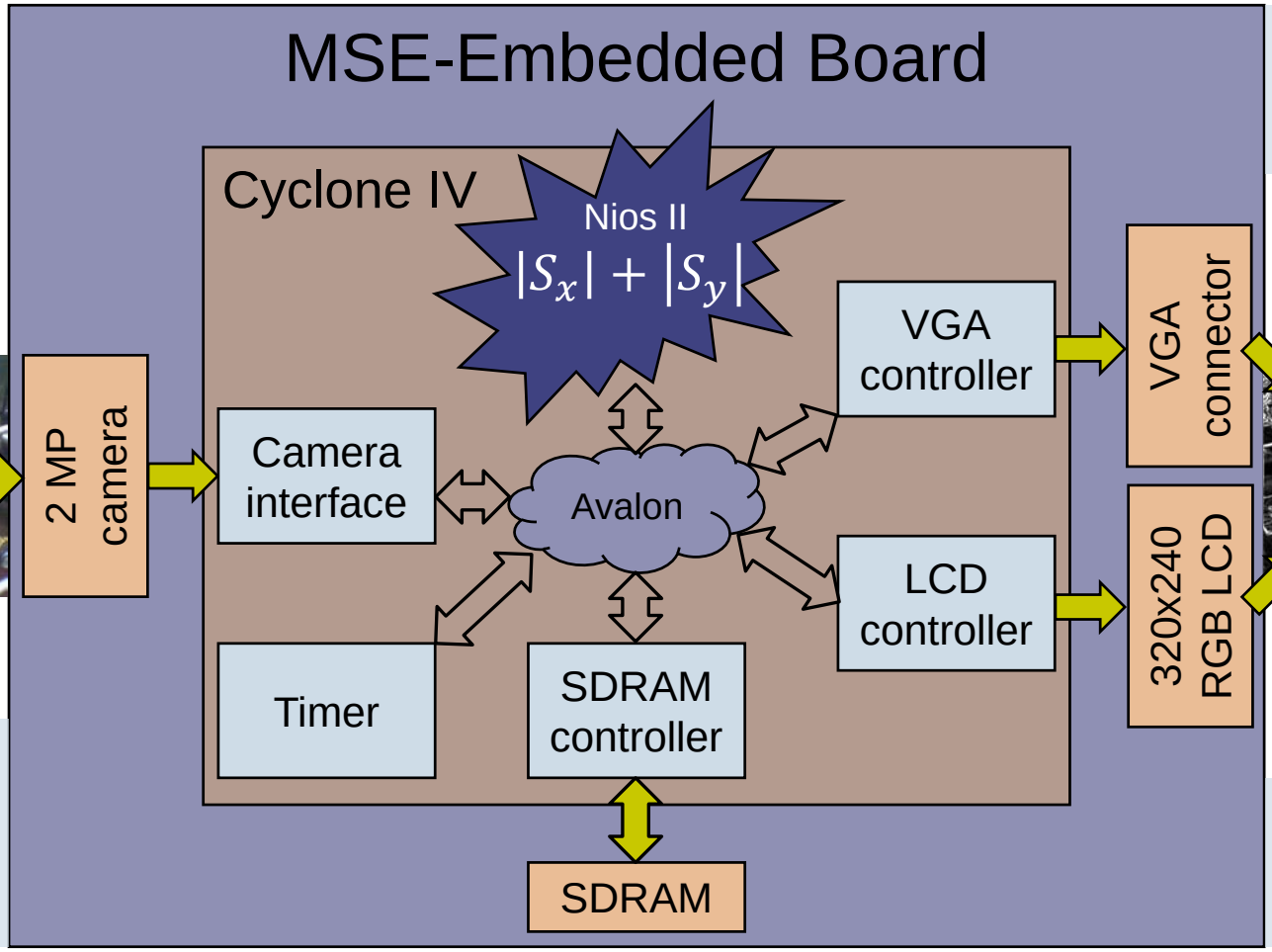
HES-SO//Genève

andrea.guerrieri@hesge.ch

Problem Specifications

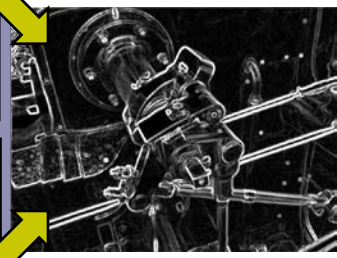


Camera:
- **RGB565**
- **512x384**
- **5-15 fps**



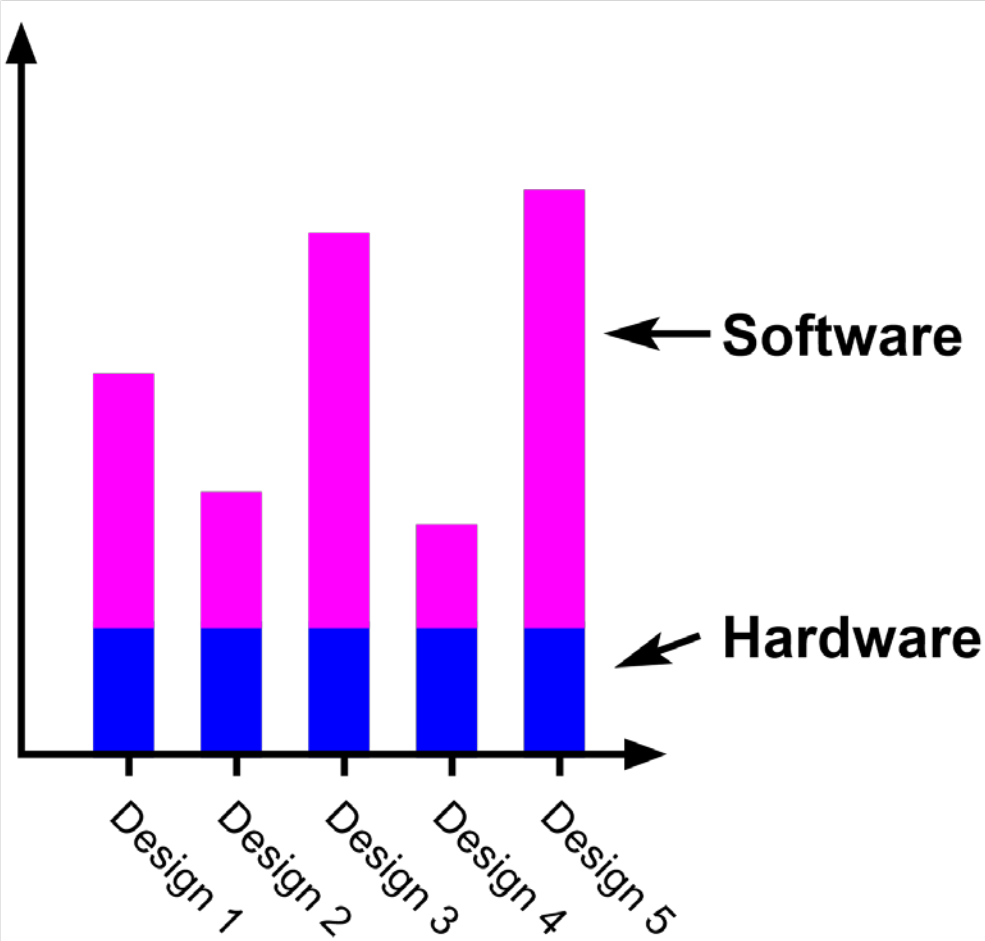
VGA:
- **RGB10-10-10**
- **1024x768**

?? fps

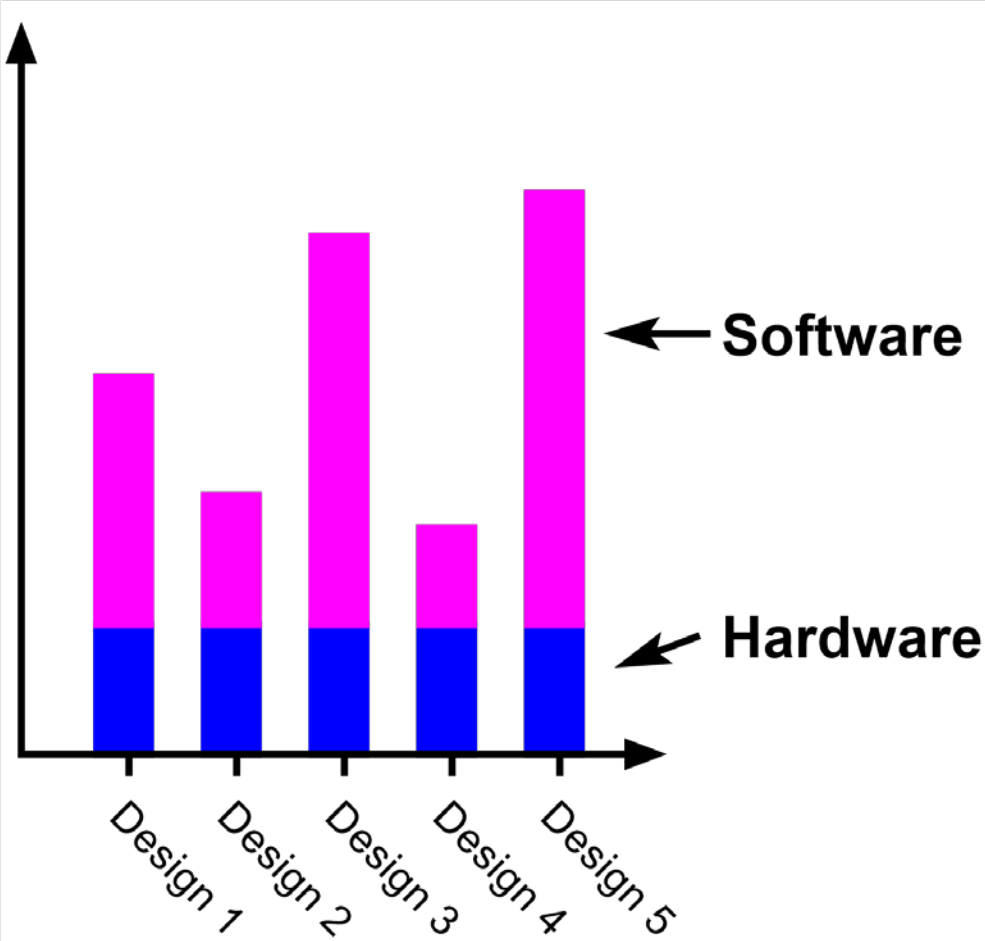


LCD:
- **RGB565**
- **240x320**

Software Optimization

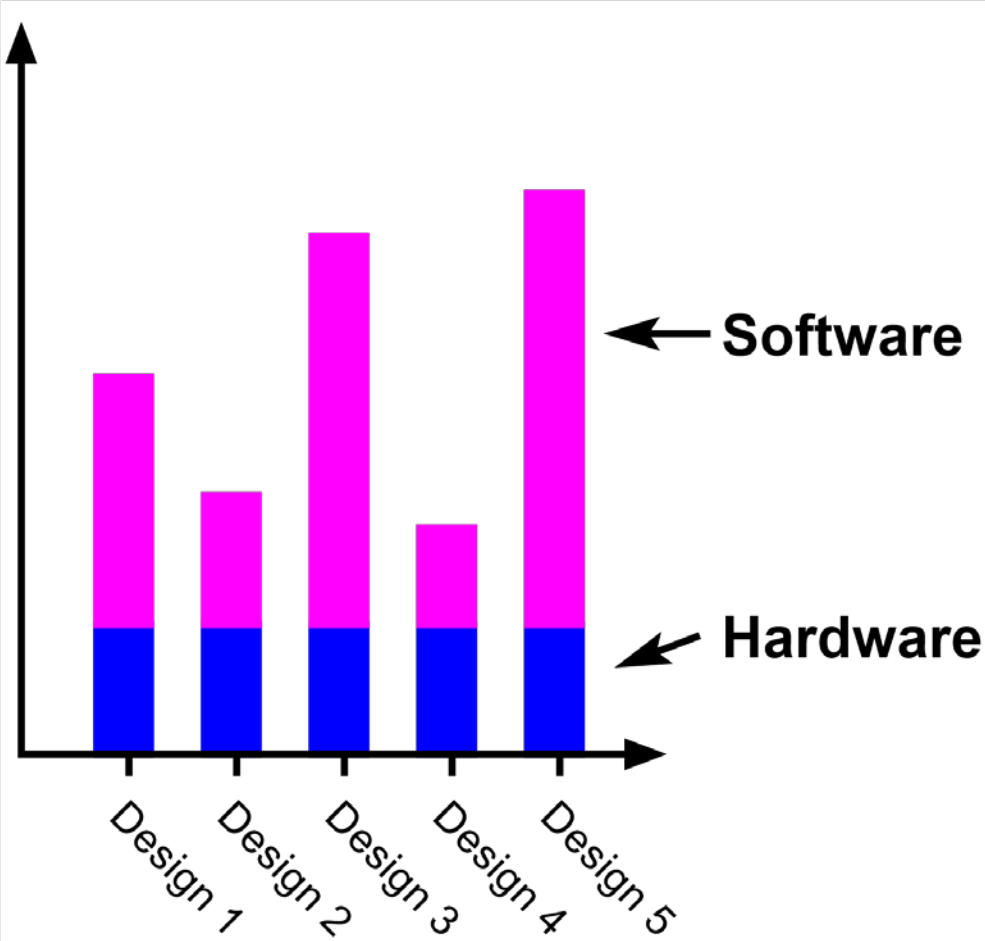


Software Optimization



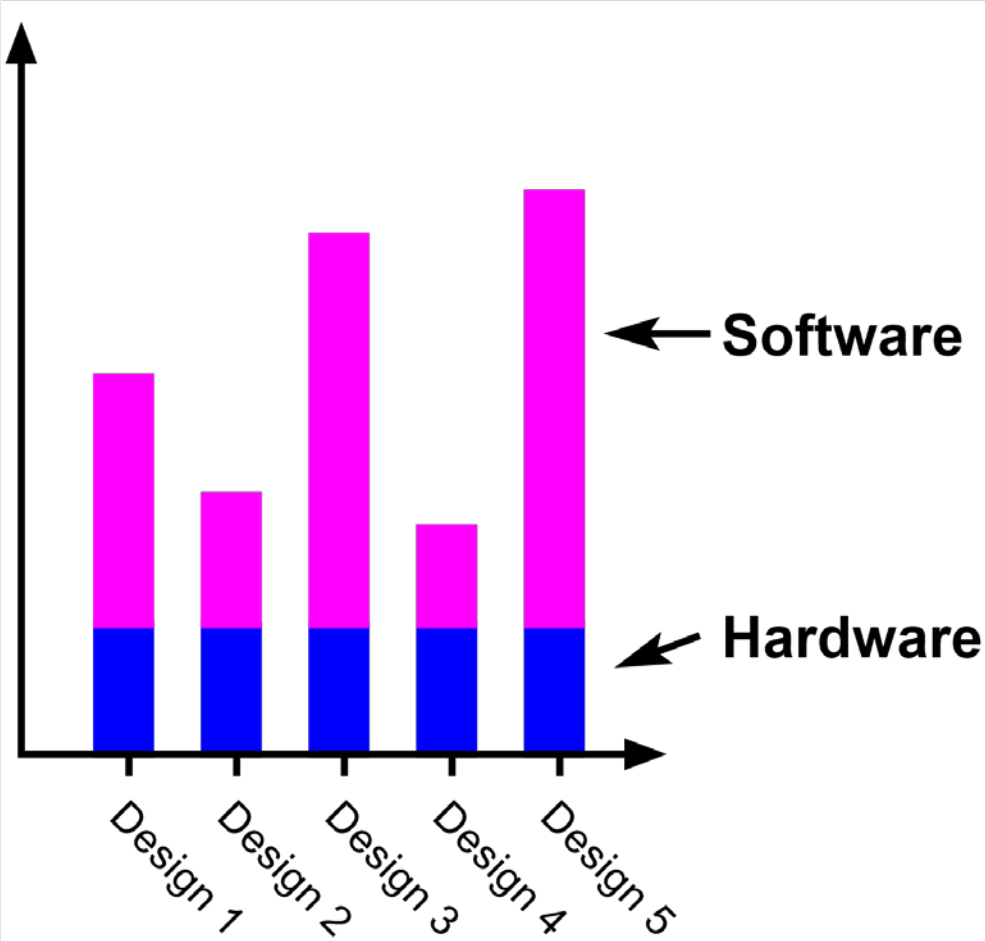
- Profiling

Software Optimization



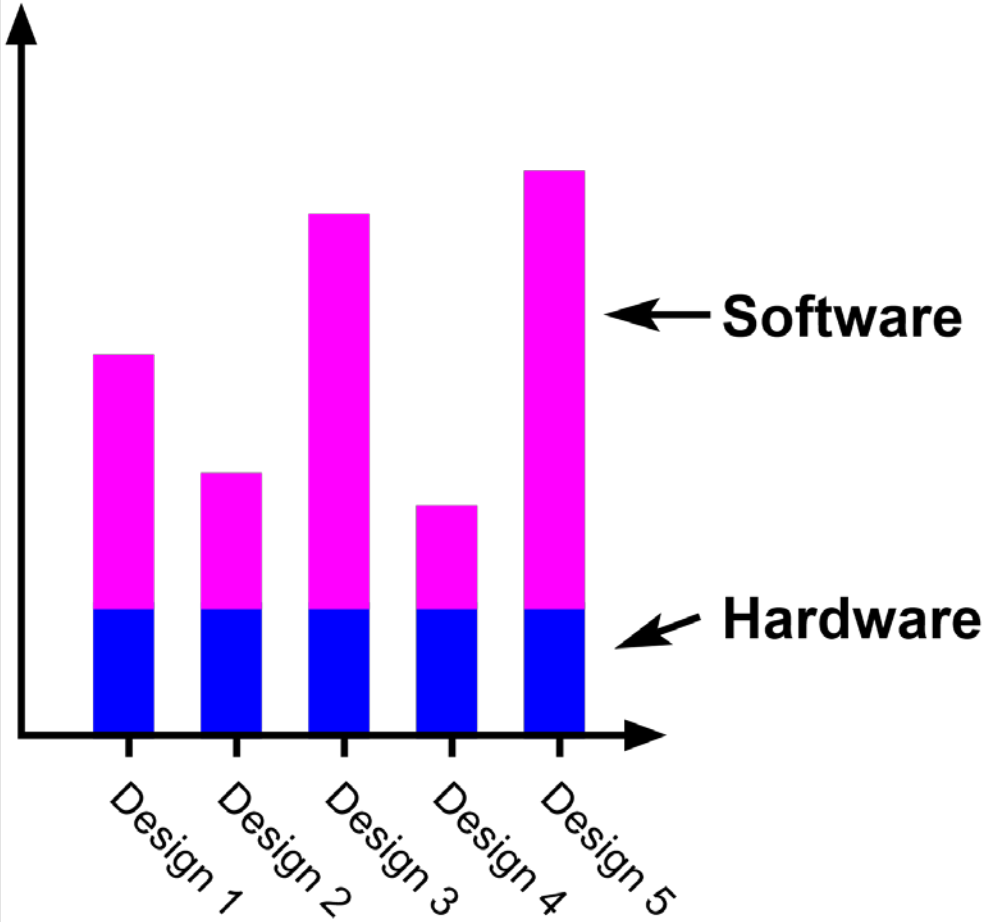
- Profiling
- General purpose
  - Loop unrolling
  - In-lining

Software Optimization



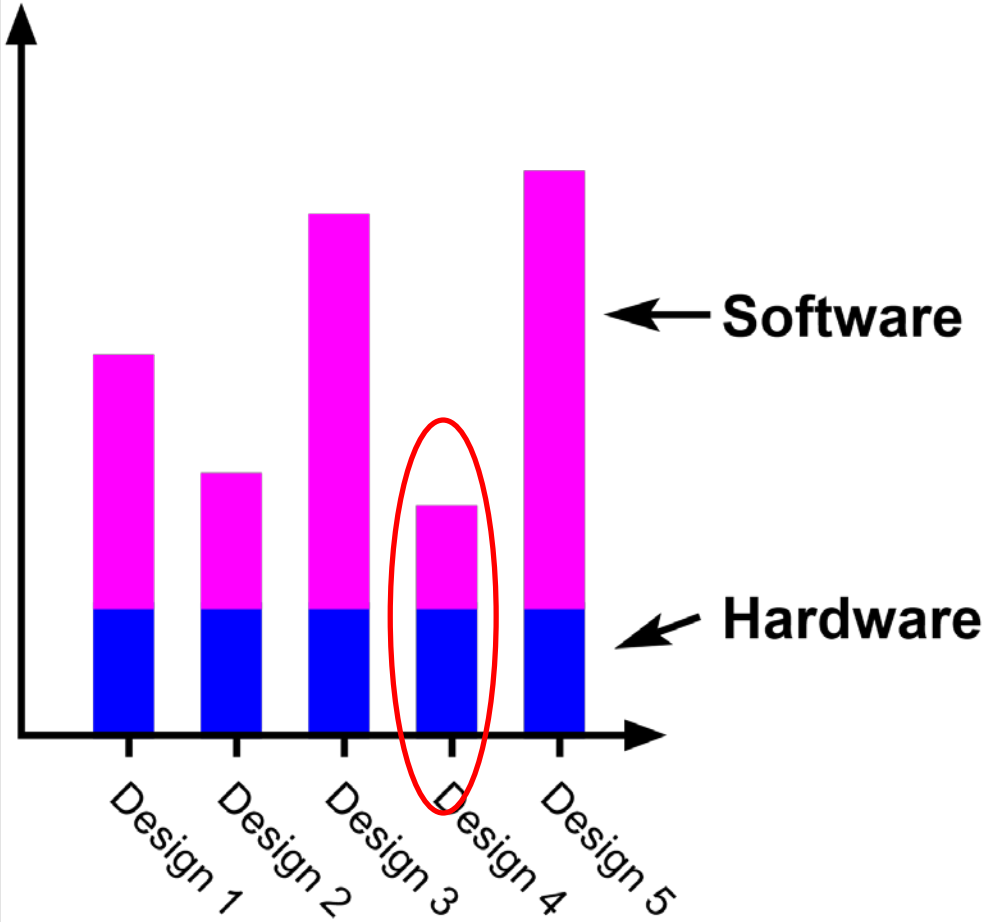
- Profiling
- General purpose
  - Loop unrolling
  - In-lining
- Algorithmic

Software Optimization



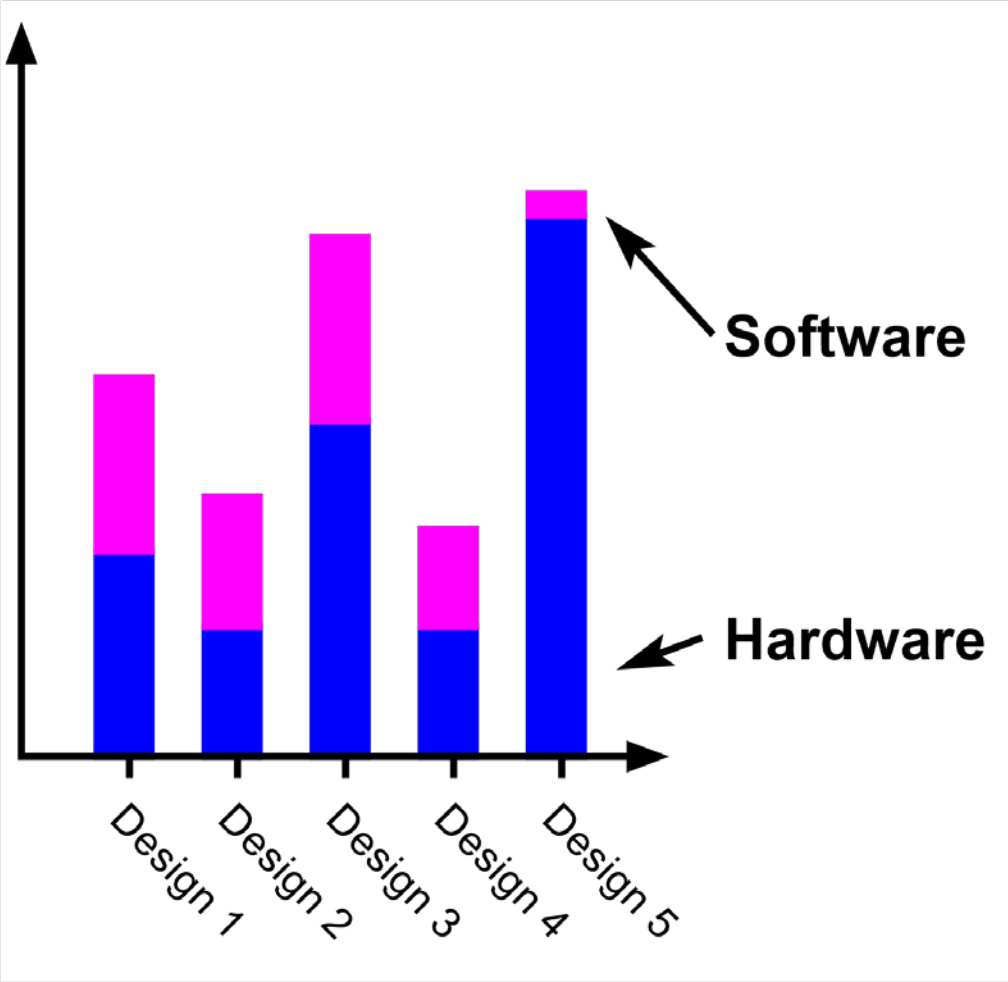
- Profiling
- General purpose
  - Loop unrolling
  - In-lining
- Algorithmic
- Cache

Software Optimization

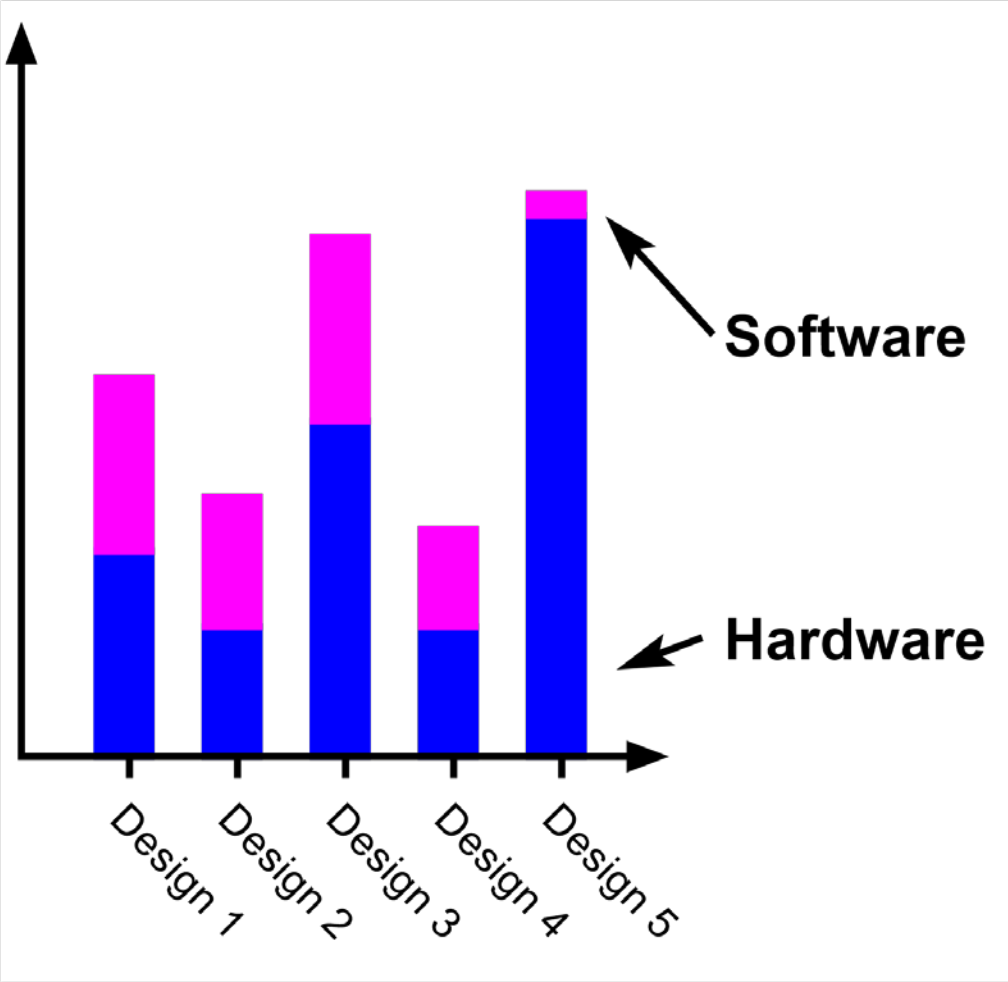


- Profiling
- General purpose
  - Loop unrolling
  - In-lining
- Algorithmic
- Cache

Hardware-Software Optimization

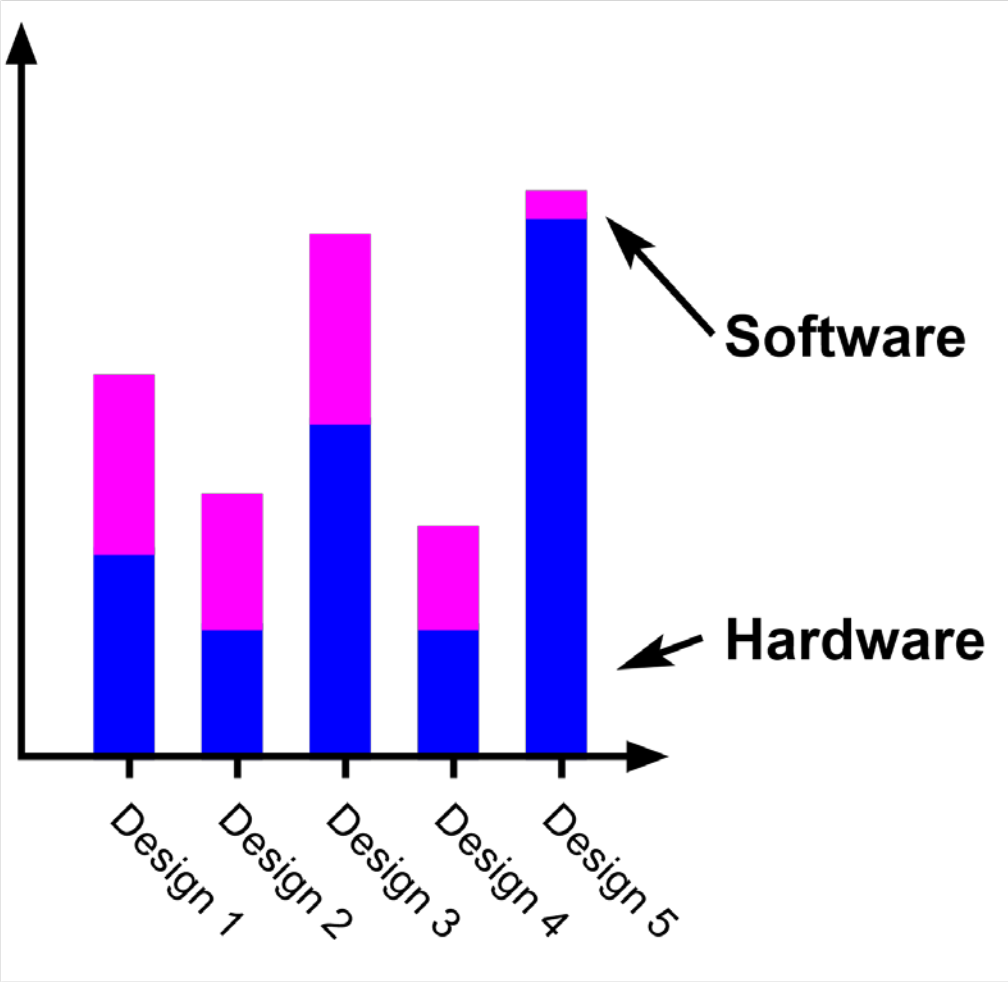


Hardware-Software Optimization



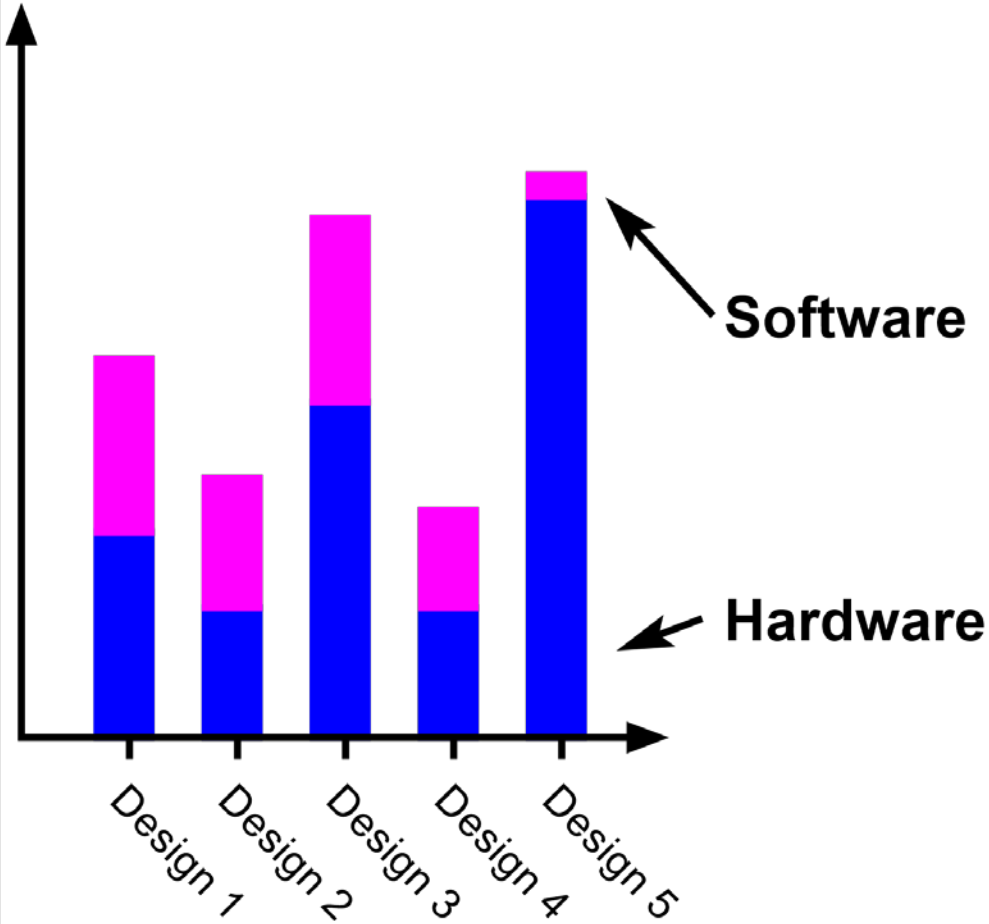
- Better processor (DSPs, VLIW..)

Hardware-Software Optimization



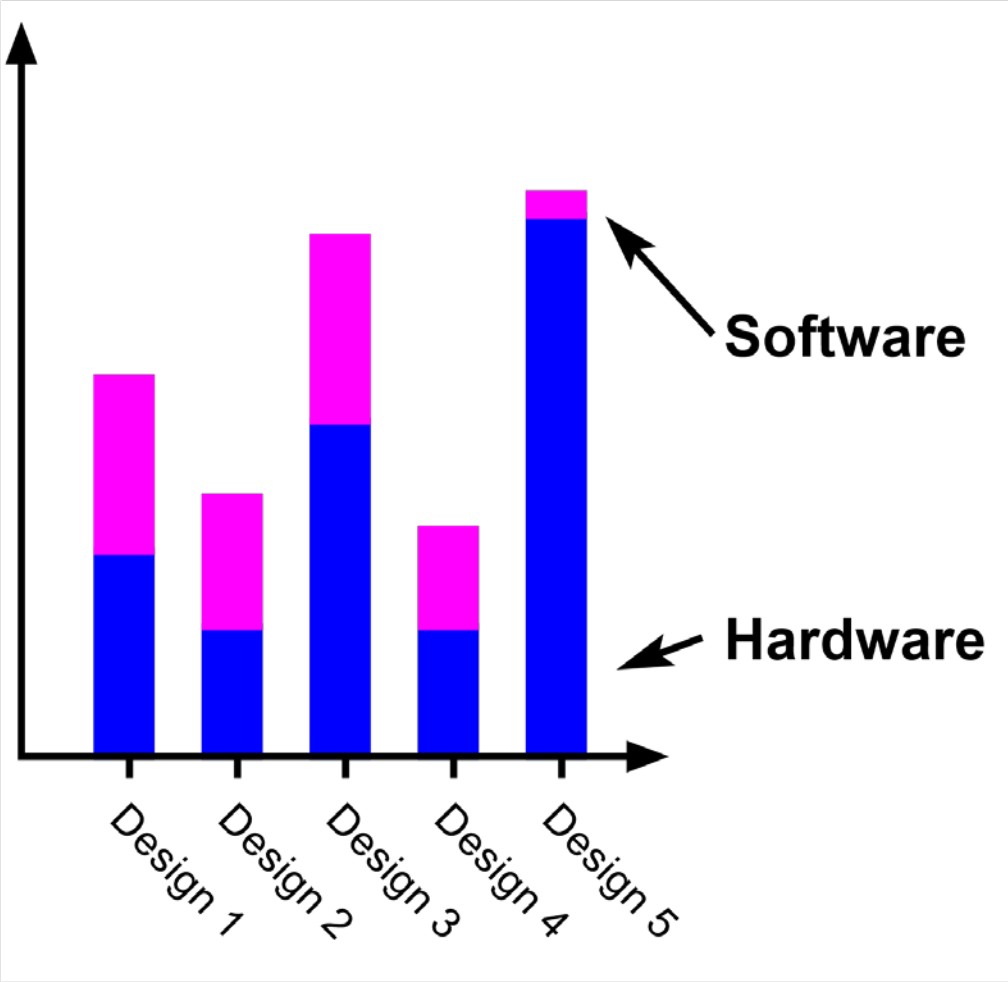
- Better processor (DSPs, VLIW..)
- Multiprocessor (Multi-Core)

Hardware-Software Optimization



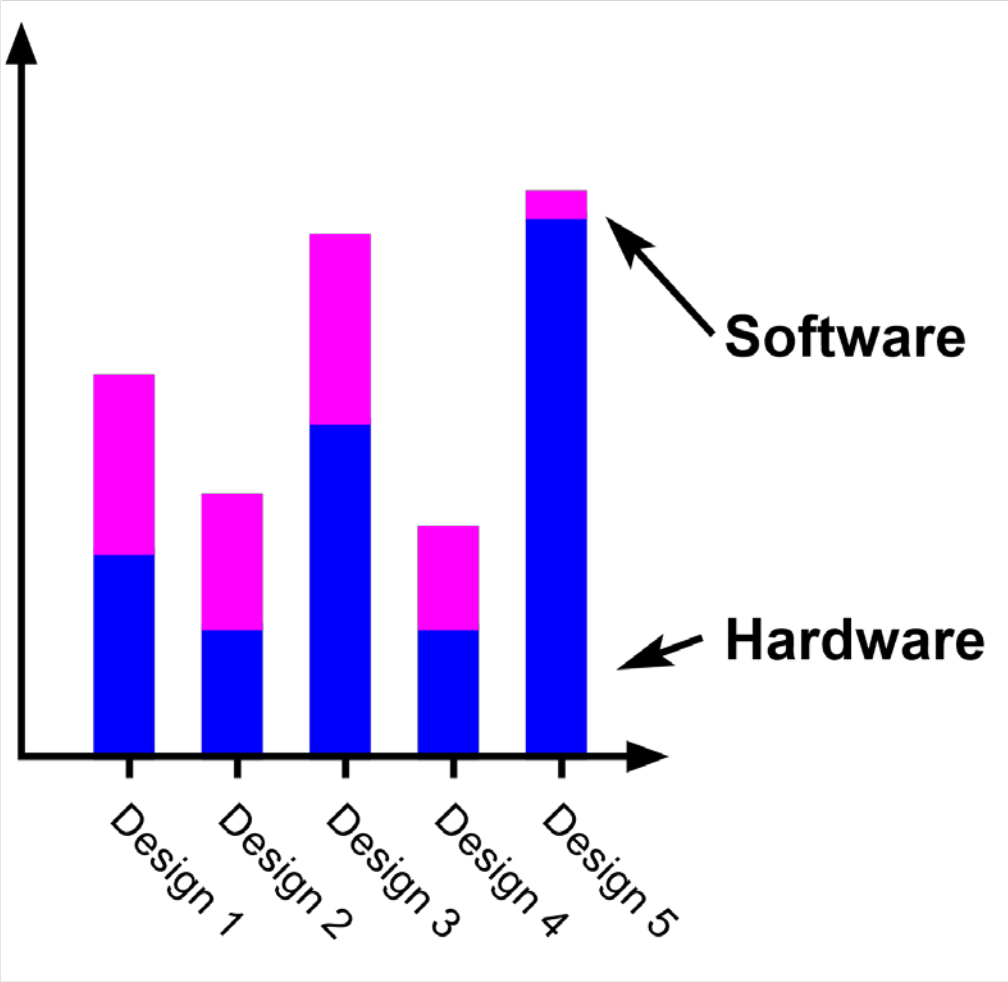
- Better processor (DSPs, VLIW..)
- Multiprocessor (Multi-Core)
- Specialized (co)processor (ASIP)

Hardware-Software Optimization



- Better processor (DSPs, VLIW..)
- Multiprocessor (Multi-Core)
- Specialized (co)processor (ASIP)
- Transfer part of the code to an accelerator (FPGA)

Hardware-Software Optimization



- Better processor (DSPs, VLIW..)
- Multiprocessor (Multi-Core)
- **Specialized (co)processor (ASIP)**
- Transfer part of the code to an accelerator (FPGA)

ASIP (Application Specific Instruction-Set Processor)

Definition

"It's a CPU which uses a set of customized instruction, designed with the purpose to accomplish a specific task required by the application, and supported by the hardware resources provided on-chip"

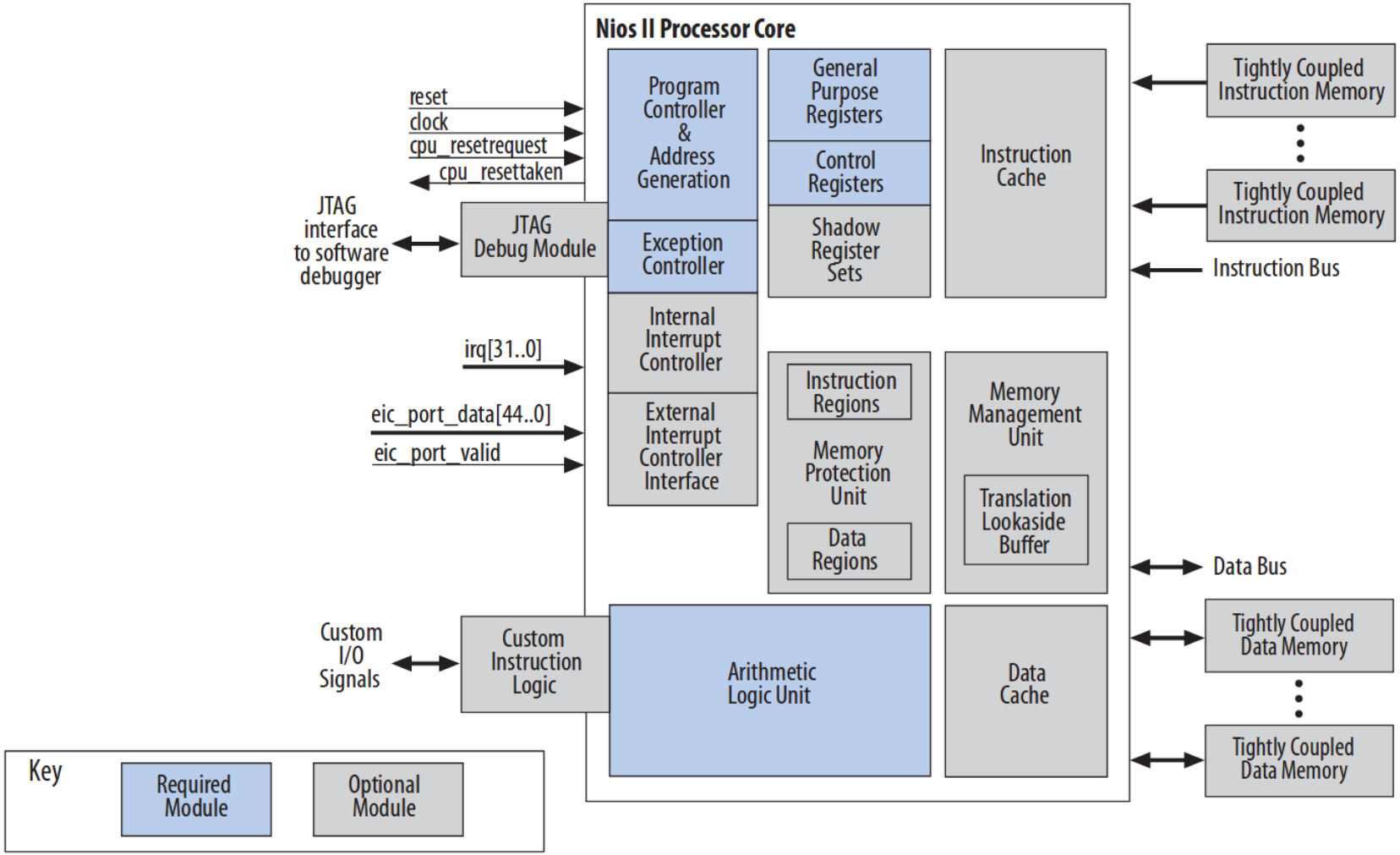
ASIP are a valid tradeoff between application specific hardware (ASIC) and general-purpose computing (CPU)

Perfect solution for improving performance in soft-core CPUs!

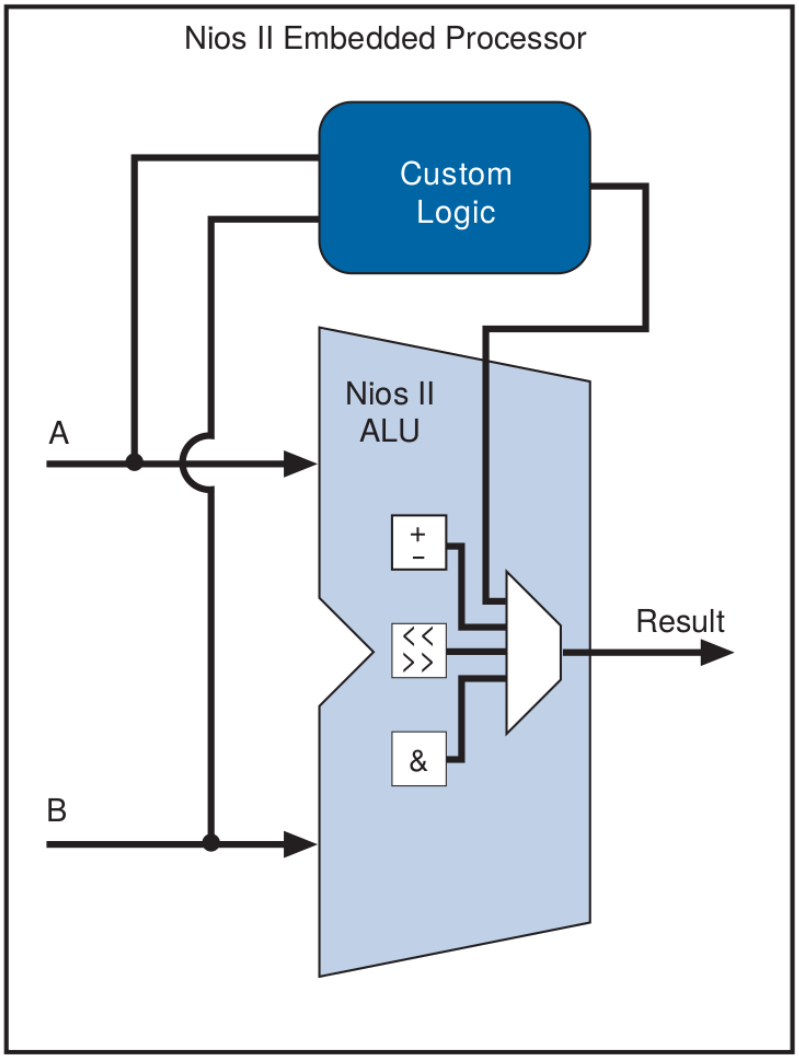
Nios-II Processor

- Nios-II is a General Purpose RISC processor
- Full instruction set available is efficient in general cases
- Up to 256 user-defined (custom) instructions

Nios-II Processor

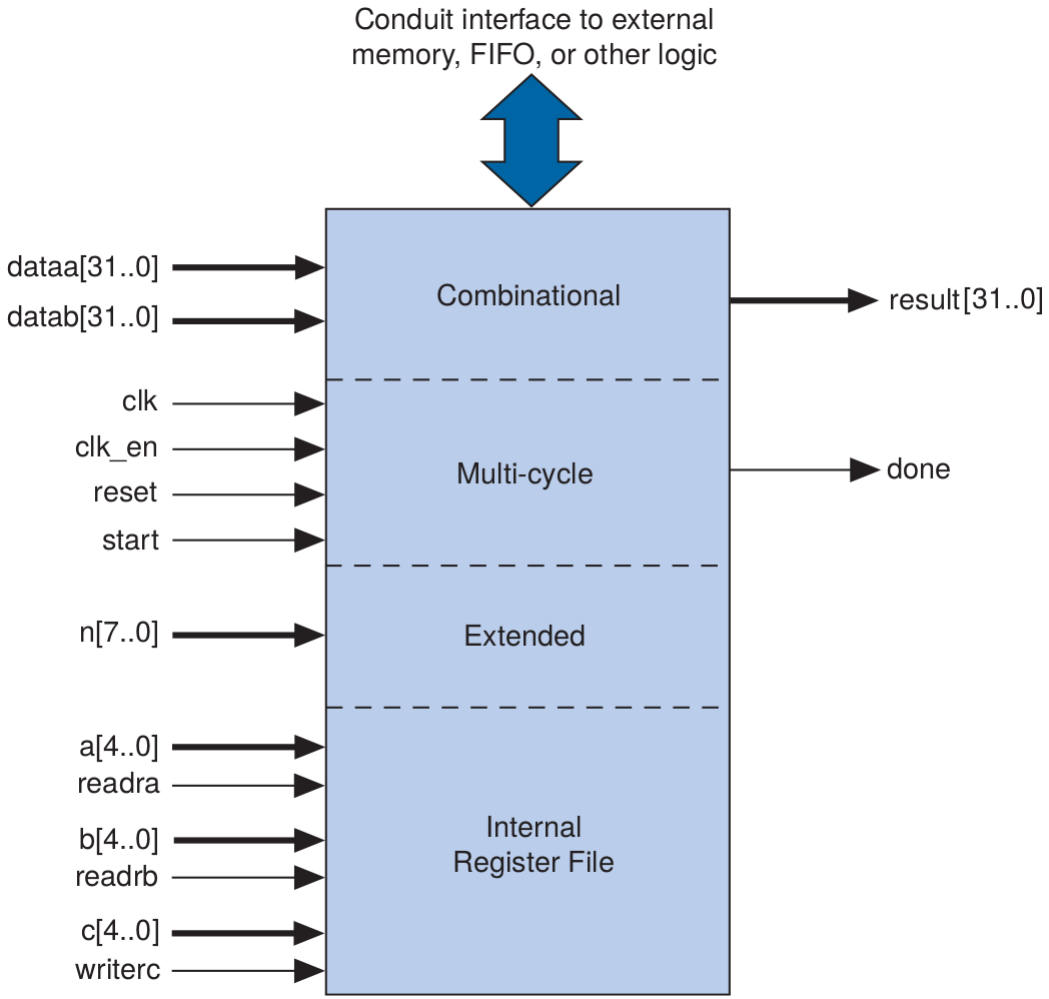


Nios-II Custom Instructions



CI Types

- Combinational
- Multi-cycle
- Extended (up to 256)
- With internal Register File
- With external access



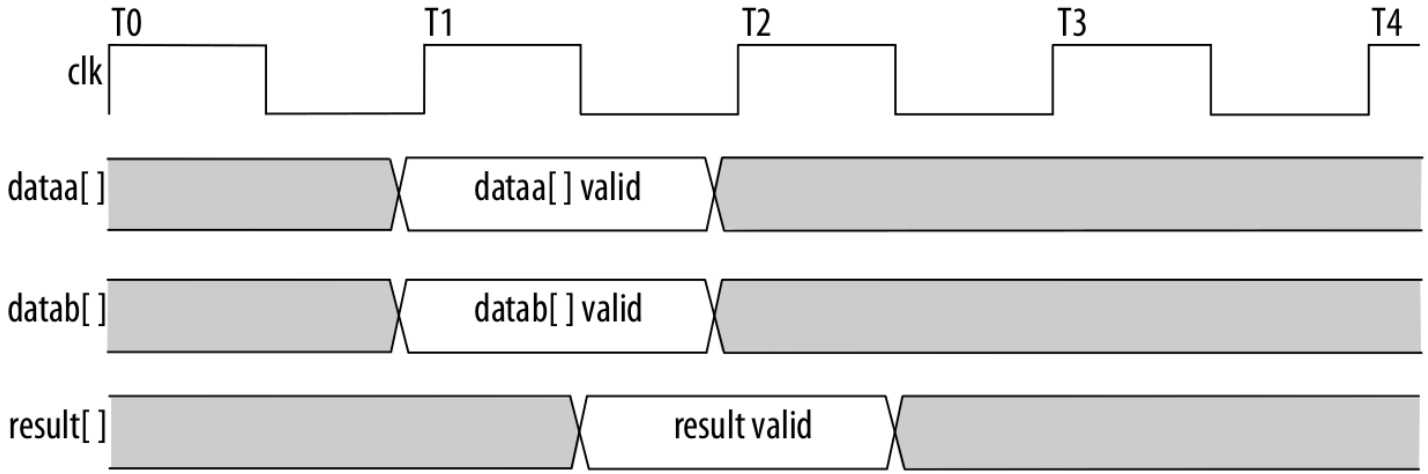
Combinational CI

- 2 * 32 bits data, 32 bits result
- 1 clock cycle

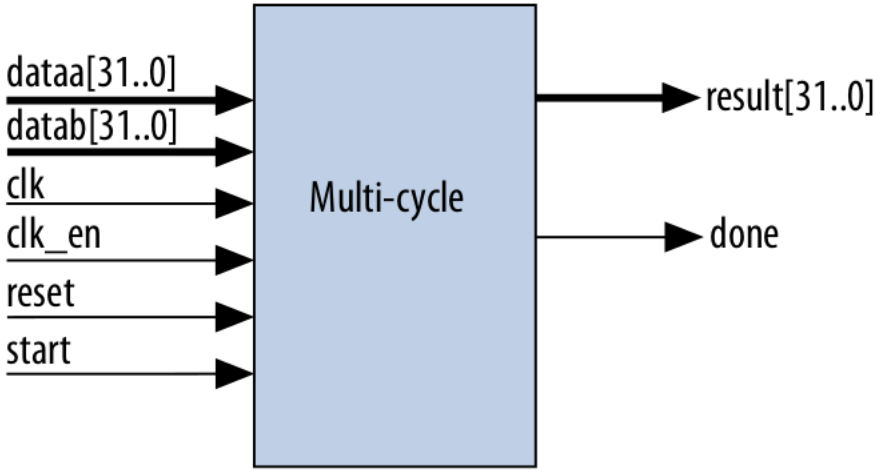
Port Name	Direction	Required	Description
`dataa[31:0]`	Input	No	Input operand to custom instruction
`datab[31:0]`	Input	No	Input operand to custom instruction
`result[31:0]`	Output	Yes	Result of custom instruction

Combinational CI

- 2 * 32 bits data, 32 bits result
- 1 clock cycle



Multi-cycle CI

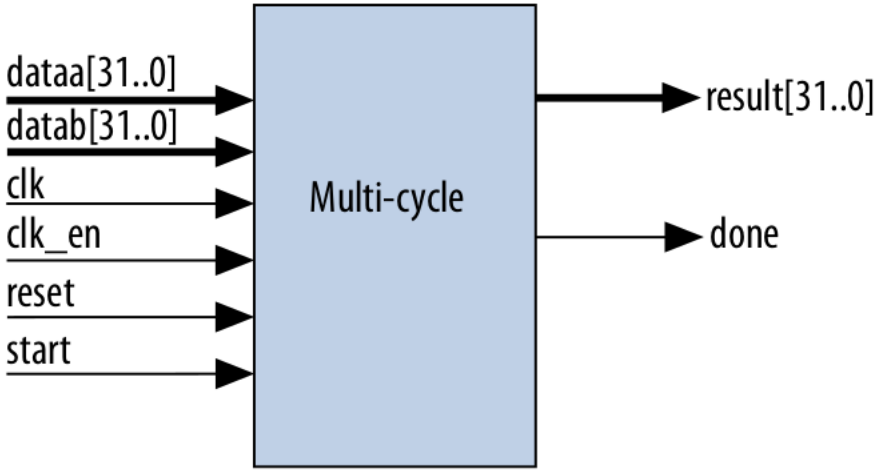


- 2 * 32 bits data, 32 bits result
- N clock cycles (fixed/variable)
- start-done (handshake)

Port Name	Direction	Required	Description
clk	Input	Yes	System clock
clk_en	Input	Yes	Clock enable
reset	Input	Yes	Synchronous reset
start	Input	No	Commands custom instruction logic to start execution
done	Output	No	Custom instruction logic indicates to the processor that execution is complete

continued...

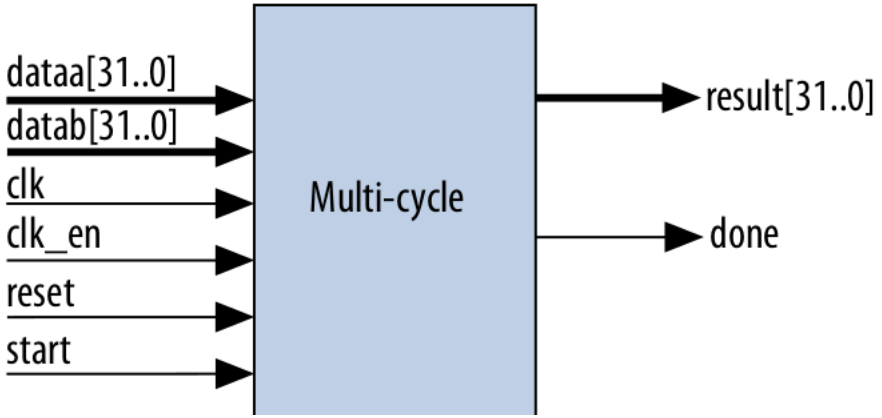
Multi-cycle CI



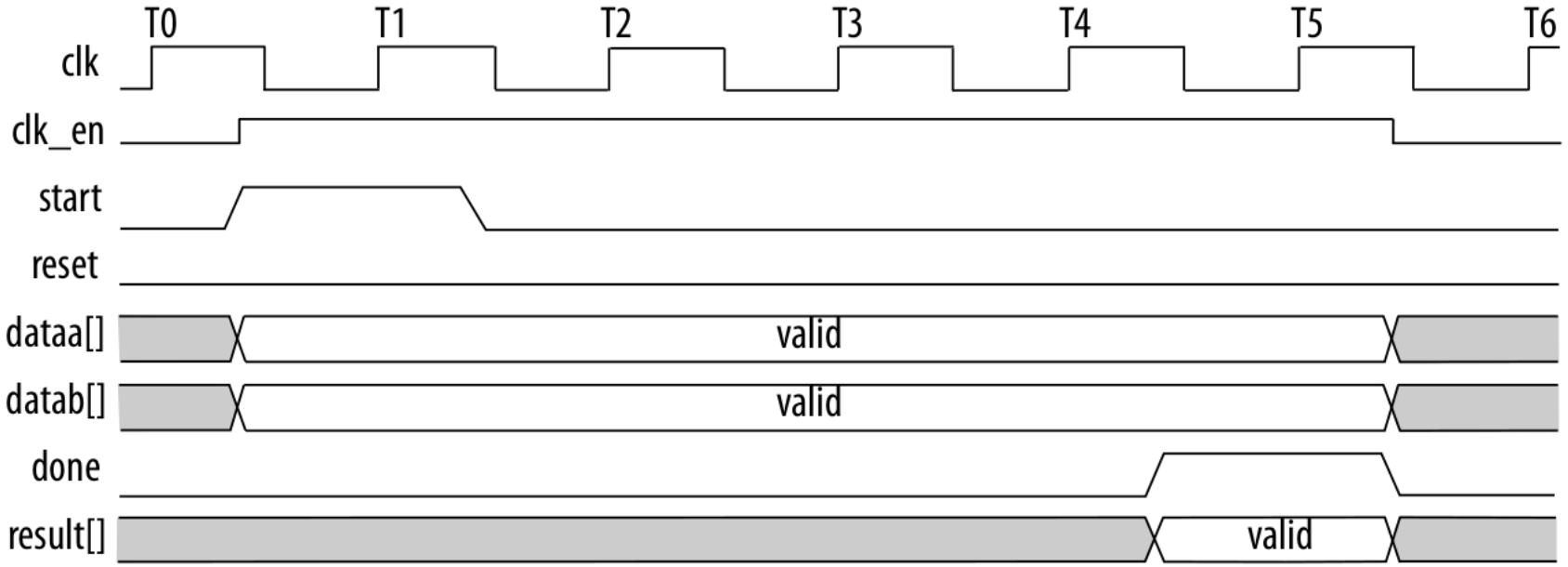
- 2 * 32 bits data, 32 bits result
- N clock cycles (fixed/variable)
- start-done (handshake)

Port Name	Direction	Required	Description
`dataa[31:0]`	Input	No	Input operand to custom instruction
`datab[31:0]`	Input	No	Input operand to custom instruction
`result[31:0]`	Output	No	Result of custom instruction

Multi-cycle CI

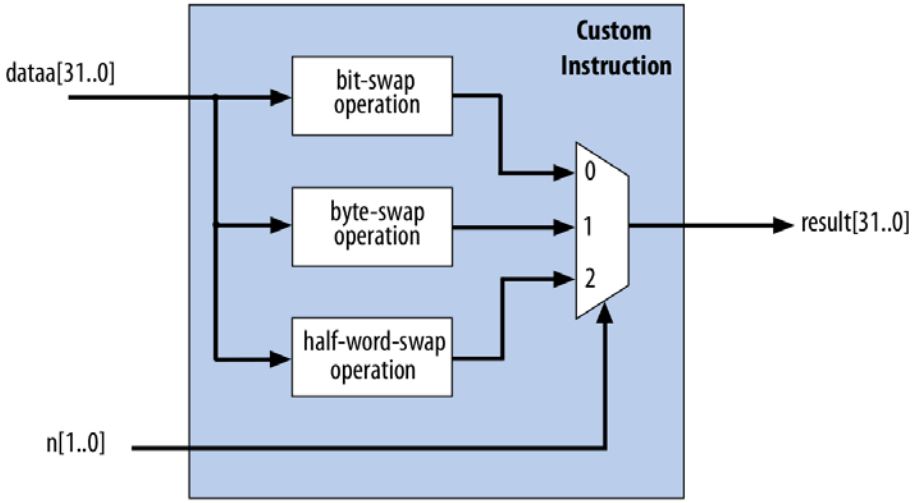


- 2 * 32 bits data, 32 bits result
- N clock cycles (fixed/variable)
- start-done (handshake)



Extended CI

- Up to 256 instructions with the same block
- n: instruction index
- Combinational or multi-cycle



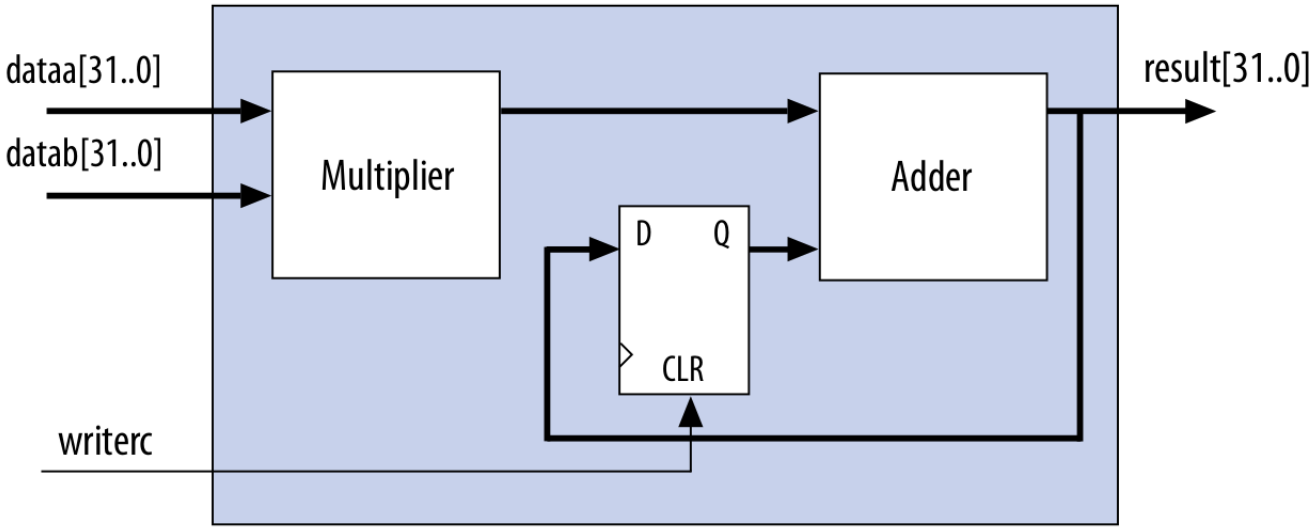
Internal Register File CI

- Using up to 32 internal registers
- Source registers: ra & rb
- Destination register: rc

Port Name	Direction	Required	Description
readra	Input	No	If readra is high, Nios II processor register a supplies dataa. If readra is low, custom instruction logic reads internal register a.
readrb	Input	No	If readrb is high, Nios II processor register b supplies datab. If readrb is low, custom instruction logic reads internal register b.
writerc	Input	No	If writerc is high, the Nios II processor writes the value on the result port to register c. If writerc is low, custom instruction logic writes to internal register c.
a[4:0]	Input	No	Custom instruction internal register number for data source A.
b[4:0]	Input	No	Custom instruction internal register number for data source B.
c[4:0]	Input	No	Custom instruction internal register number for data destination.

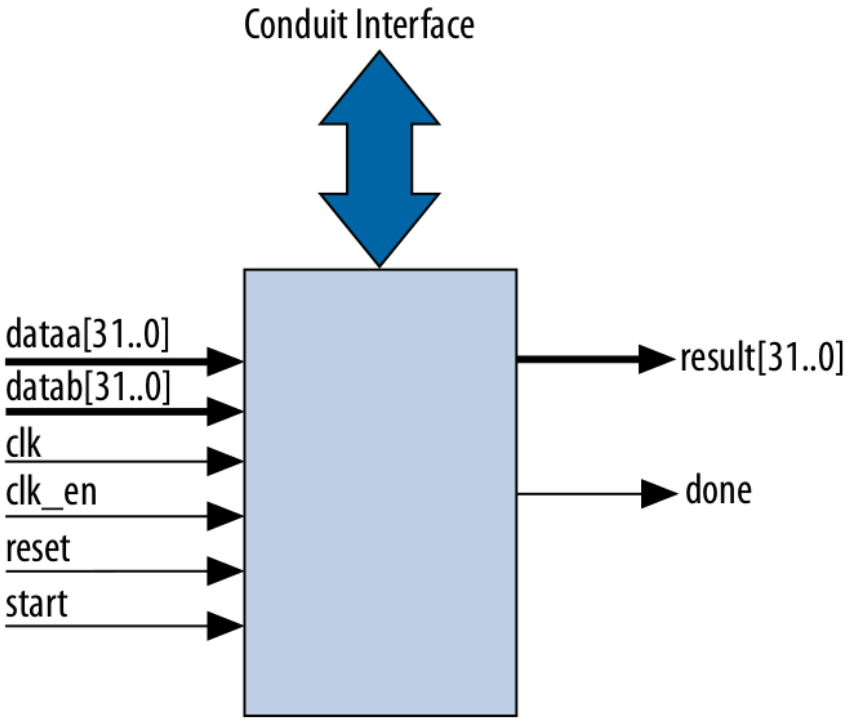
Internal Register File CI

- Using up to 32 internal registers
- Source registers: ra & rb
- Destination register: rc



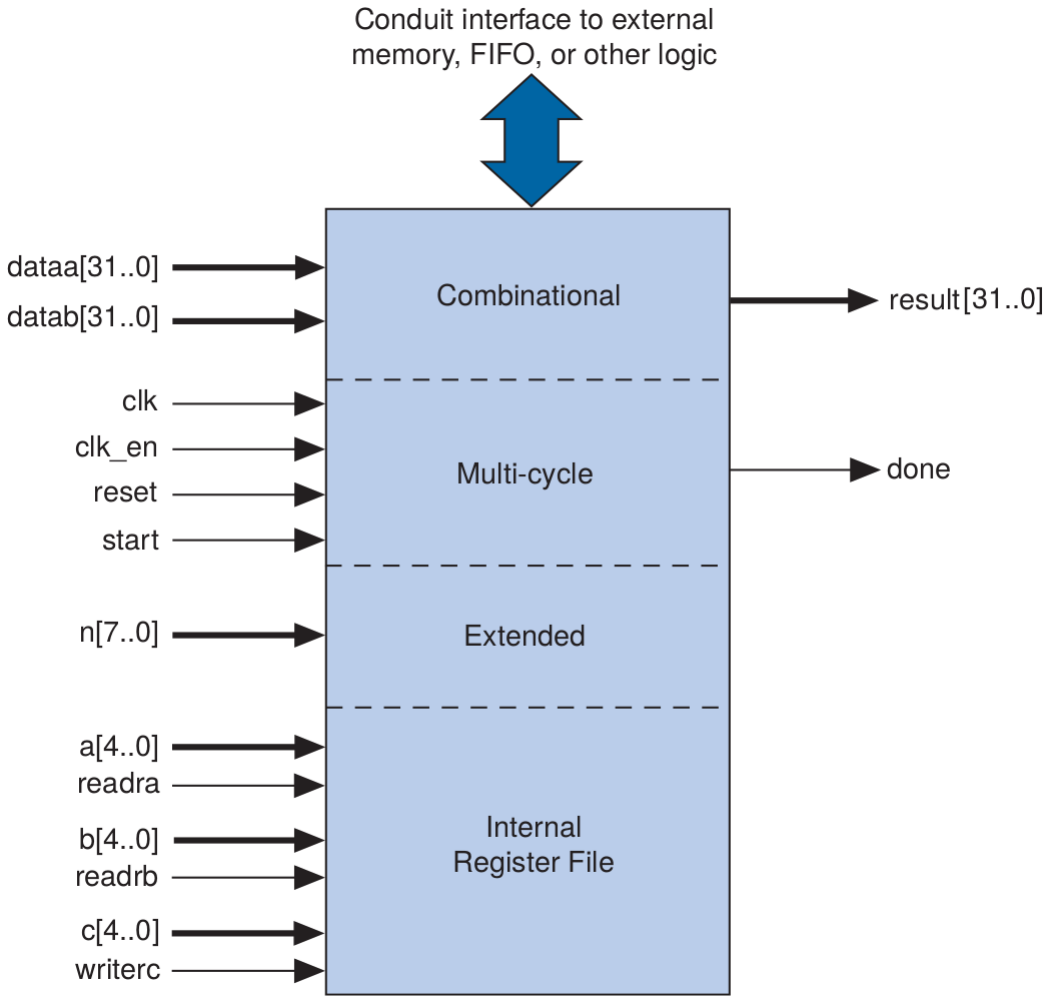
External Interface CI

- Storage of intermediate results, memory access, FIFO…



CI Types

- Combinational
- Multi-cycle
- Extended (up to 256)
- With internal Register File
- With external access



CI usage example

- Once the instruction is defined, a C macro is automatically created in *system.h*

```
#define ALT_CI_BITSWAP_N 0x00
#define ALT_CI_BITSWAP(A) __builtin_custom_ini(ALT_CI_BITSWAP_N,(A))
```

- Usage:

```
#include "system.h"

int main (void)
{
    int a = 0x12345678;
    int a_swap = 0;

    a_swap = ALT_CI_BITSWAP(a);
    return 0;
}
```

Built-in functions

- The compiler uses GCC built-in functions to map custom instructions:

  `__builtin_custom_`*<return type>*`n`*<parameter types>*

- *<return type>* and *<parameter types>* are decoded as:
  - *i* integer
  - *f* float
  - *p* void *
  - *(empty)* void

Example of built-in functions

```
1. /* define void udef_macro1(float data); */
2. #define UDEF_MACRO1_N 0x00
3. #define UDEF_MACRO1(A) __builtin_custom_nf(UDEF_MACRO1_N, (A));
4. /* define float udef_macro2(void *data); */
5. #define UDEF_MACRO2_N 0x01
6. #define UDEF_MACRO2(B) __builtin_custom_fnp(UDEF_MACRO2_N, (B));
7.
8. int main (void)
9. {
10.     float a = 1.789;
11.     float b = 0.0;
12.     float *pt_a = &a;
13.
14.     UDEF_MACRO1(a);
15.     b = UDEF_MACRO2((void *)pt_a);
16.     return 0;
17. }
```

CI Assembly Language Syntax

- Syntax:

  `custom` <*selection index*>, <*Destination*>, <*Source A*>, <*Source B*>

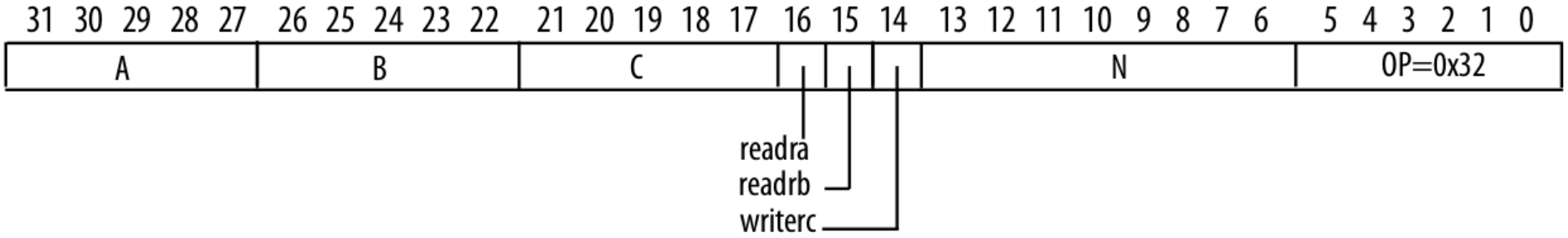
- <*Destination*>, <*Source A*> and <*Source B*> can be:

  `r`<*i*>—Nios II register <*i*>

  `c`<*i*>—Custom register <*i*>

- Examples:
  - *custom 0, r6, r7, r8*
  - *custom 3, c1, r2, c4*
  - *custom 4, r6, c9, r2*

CI Word Instruction format



Field Name	Purpose	Corresponding Signal
A	Register address of input operand A	
B	Register address of input operand B	
C	Register address of output operand C	
readra	Register file selector for input operand A	readra
readrb	Register file selector for input operand B	readrb
writerc	Register file selector for ouput operand C	writerc
N	Custom instruction select index (optionally includes an extension index)	
OP	custom opcode, 0x32	n/a

Example

```
 8 #include <stdlib.h>
 9 #include <io.h>
10 #include <system.h>
11
12 unsigned char *grayscale_array;
13 int grayscale_width = 0;
14 int grayscape_height = 0;
15
16 void conv_grayscale(void *picture,
17                     int width,
18                     int height) {
19     int x,y,gray;
20     unsigned short *pixels = (unsigned short *)picture , rgb;
21     grayscale_width = width;
22     grayscape_height = height;
23     if (grayscale_array != NULL)
24         free(grayscale_array);
25     grayscale_array = (unsigned char *) malloc(width*height);
26     for (y = 0 ; y < height ; y++) {
27         for (x = 0 ; x < width ; x++) {
28             rgb = pixels[y*width+x];
29             gray = (((rgb>>11)&0x1F)<<3)*21; // red part
30             gray += (((rgb>>5)&0x3F)<<2)*72; // green part
31             gray += (((rgb>>0)&0x1F)<<3)*7; // blue part
32             gray /= 100;
33             IOWR_8DIRECT(grayscale_array,y*width+x,gray);
34         }
35     }
36 }
37
```

Design Tradeoffs