

CPU Area & Performance Optimization

Juan Pedro Rodríguez Gracia & Rami Aqquid

ASIP Lab

- 1 Motivation
- 2 Area Optimization
 - Take out the unused instructions
 - Take out the unused Modules
 - Apply the Xilinx strategies
- 3 Performance Optimization
 - Creation of new Instructions
 - Instruction Step 2
 - Instruction Step 5
 - Instruction Step 4
 - Instruction Step 4a
 - Instruction Step 4b
 - GCC optimizations
- 4 Looking into the future
- 5 Statistics

Why Area?

- Biggest Limitation
- Makes everything easy

Why Performance?

- Speed matters
- Physical limits (GHz)

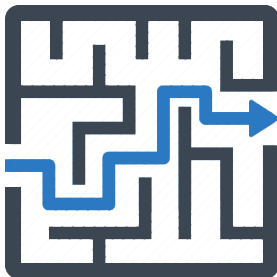
Area Optimization

Goal

Our goal is to reduce the amount of Slices and LUT needed on the FPGA.

Points to focus on

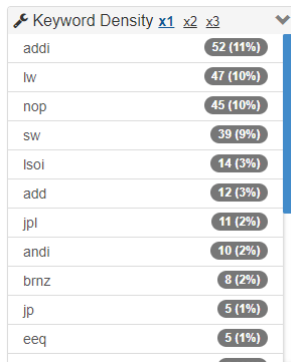
- Take out the unused Instructions
- Take out the unused Modules
- Apply the Xilinx strategies



Take out the unused instructions

Steps

- 1 Go to "<https://wordcounter.net/>"
- 2 Paste the .s text of the files
- 3 Delete all the instructions that doesn't appear



The screenshot shows a web application titled "Keyword Density" with three tabs: x1, x2, and x3. A dropdown menu is open, displaying a list of instructions and their corresponding counts and percentages. The instructions are sorted in descending order of frequency.

Instruction	Count	Percentage
addi	52	(11%)
lw	47	(10%)
nop	45	(10%)
sw	39	(9%)
lsoi	14	(3%)
add	12	(3%)
jpl	11	(2%)
andi	10	(2%)
brnz	8	(2%)
jp	5	(1%)
eeq	5	(1%)
...	5	(1%)

Take out the unused Modules

Steps

- 1 Open ASIP meister
- 2 Check unused modules
- 3 Delete them

A good Example

The **MUL** instruction and so its module aren't used, we can delete it

Apply the Xilinx strategies

As Xilinx has strategies for different purposes and we use this program, we are allowed to take advantage of all its potential.

Performance Optimization

Goal

Our Goal is to reduce the amount of clock cycles needed

Points to focus on

- Creation of new Instructions
- Addition of new Components
- Apply the GCC optimizations



Which instructions should I create?

As the algorithm is divided in steps I will create new instructions based on this classification



Instruction Step 2

The thought process

- ① Check if ($index < 0$)
 - ① If the 32nd bit is 1, the number is negative
 - ② Then assign depending on the comparison result
- ② Check if ($index > 88$)
 - ① We use the ALU to compare with 88
 - ② Then assign depending on the comparison result

Step 2 MicroInstruction

```
/* Step 2 - Find new index value (for later) */  
index += indexTable[delta];  
if ( index < 0 ) index = 0;  
if ( index > 88 ) index = 88;
```

Instruction Step 5

Quick note

32767 = 0000 0000 0000 0000 0111 1111 1111 1111

-32768 = 1111 1111 1111 1111 1000 0000 0000 0000

The thought process

- ① Check if (*valpred* > 32767)
 - ① Check if I have any 1 on the full of 0s part
 - ② If I have, assign the top-limit value
- ② Check if (*valpred* < -32768)
 - ① Check if I have any 0 on the full of 1s part
 - ② If I have, assign the bottom-limit value

```
/* Step 5 - clamp output value */  
if ( valpred > 32767 )  
    valpred = 32767;  
else if ( valpred < -32768 )  
    valpred = -32768;
```

Instruction Step 4

The division into two different instructions

As we see two clearly differentiated parts, we are going to separate it in two Micro Instructions.



Instruction Step 4a

The thought process

- ① We calculate all shifts
- ② We make the assignation
- ③ On every shift, If the condition is not satisfied, overwrite with 0
- ④ Quadruple add all the shifted values with the initial one

Step 4a Micro Instruction

```
vpdiff = step >> 3;  
if ( delta & 4 ) vpdiff += step;  
if ( delta & 2 ) vpdiff += step>>1;  
if ( delta & 1 ) vpdiff += step>>2;
```

Quick note

We need two new ALU modules to create the Quadruple Adder Macro

Instruction Step 4b

The thought process

- 1 We manually compliment by 2 the vpdiff value
- 2 According to the condition the final variable will be added to \pm vpdiff

Step 4a Micro Instruction

```
if ( sign )  
    valpred -= vpdiff;  
else  
    valpred += vpdiff;
```

The thought process

We simply use the *make* command, as always, but now we add the GCC -O flag `GCC_PARAM = -O3`



Looking into the future

Future Optimizations

- Area
 - Manual mapping on Xilinx
 - Change .s code
 - Deleting registers
- Performance
 - New modules for greater instructions
 - Finish the Step-based instructions job
 - Introducing parallelism



CPUs	Clock Cycles	Register Slices	LUT Slices
Basic CPU	121892	3568	5142
Performance CPU	59242	3637	5953
Area CPU	121892	3056	3507

CPU Table: The clock cycles captured are under the -O3 GCC Flag

Problems

- Time constraints problem
- Custom Area Strategy Xilinx

Any Questions?

