Предварительный отчет о реализации системы поиска подстроки на FPGA

# Общая схема системы



# Реализация двоичного поиска на C

Файл bitap.h:

#ifndef BITAP\_H\_

#define BITAP\_H\_

// Alphabet size

#define BITAP\_ASIZE 256

// Word size

#define BITAP\_WORD (sizeof(unsigned int)\*8)

typedef struct

{

unsigned int limit;

unsigned int shift;

unsigned int masks[BITAP\_ASIZE];

} bitap\_Pattern\_t;

bool bitap\_MakePattern(char\* needle, bitap\_Pattern\_t\* pattern);

int bitap\_Find(char\* haystack, bitap\_Pattern\_t\* pattern);

#endif /\* BITAP\_H\_ \*/

Файл bitap.c

#include <string.h>

#include <stdbool.h>

#include "bitap.h"

bool bitap\_MakePattern(char\* needle, bitap\_Pattern\_t\* pattern)

{

int len = strlen(needle);

if (len > BITAP\_WORD) return false;

pattern->shift = len - 1;

memset(pattern->masks, ~0, BITAP\_ASIZE \* sizeof(int));

for(int i = 0; needle[i]; i++) pattern->masks[(unsigned int)needle[i]] &= ~(1 << i);

pattern->limit = ~0 << pattern->shift;

return true;

}

int bitap\_Find(char\* haystack, bitap\_Pattern\_t\* pattern)

{

int counter = 0;

unsigned int state = ~0;

for (int j = 0; haystack[j]; j++)

{

state = (state << 1) | pattern->masks[(unsigned int)haystack[j]];

if (state < pattern->limit) counter++;

}

return counter;

}

# Реализация двоичного поиска на Verilog

module bitap

(

// Control registers stuff

input control\_regs\_clock,

input control\_regs\_we,

input [0:0] control\_regs\_addr,

input [31:0] control\_regs\_data\_in,

output reg [31:0] control\_regs\_data\_out,

// Memory for haystack

output haystack\_mem\_clock,

output [15:0] haystack\_mem\_addr,

input [7:0] haystack\_mem\_data,

// Memory for needle

output needle\_mem\_clock,

output [7:0] needle\_mem\_addr,

input [31:0] needle\_mem\_data,

// Clock and reset

input clock,

input reset

);

reg run\_enable;

reg [4:0] needle\_shift;

reg [31:0] match\_amount;

reg [31:0] state;

reg [31:0] needle\_data;

reg [31:0] control\_reg0;

reg wflag0, wflag1;

integer counter;

wire [31:0] state\_limit = 32'hFFFF\_FFFF << needle\_shift;

wire [31:0] new\_state = (state << 1) | needle\_mem\_data;

assign needle\_mem\_clock = clock;

assign needle\_mem\_addr = haystack\_mem\_data;

assign haystack\_mem\_clock = clock;

assign haystack\_mem\_addr = counter;

always @\*

begin

case(control\_regs\_addr)

1'b0: control\_regs\_data\_out <= {needle\_shift, run\_enable};

1'b1: control\_regs\_data\_out <= match\_amount;

endcase

end

always @(posedge control\_regs\_clock)

begin

if (reset)

begin

wflag0 <= 1'b0;

control\_reg0 <= 1'b0;

end

else

begin

if (control\_regs\_we)

begin

case(control\_regs\_addr)

1'b0: control\_reg0 <= control\_regs\_data\_in;

endcase

if (wflag1 != wflag0) wflag0 <= ~wflag0;

end

end

end

always @(posedge reset or posedge clock)

begin

if (reset)

begin

wflag1 <= 1'b0;

run\_enable <= 1'b0;

match\_amount <= 1'b0;

end

else

begin

if (wflag1 == wflag0)

begin

wflag1 <= ~wflag1;

{needle\_shift, run\_enable} <= control\_reg0;

if (control\_reg0[0]) // If run\_enable will be set

begin

counter <= 1'b0;

match\_amount <= 1'b0;

state <= 32'hFFFF\_FFFF;

end

end

if (run\_enable)

begin

if (counter >= 2)

begin

state <= new\_state;

if (new\_state >= state\_limit) match\_amount <= match\_amount;

else match\_amount <= match\_amount + 1'b1;

end

// if we get null-terminator then stop

if (counter >= 1 && !haystack\_mem\_data) run\_enable <= 1'b0;

counter <= counter + 1'b1;

end

end

end

endmodule

# Утилита fsearch

Для тестирования системы и сравнения скорости работы поиска на FPGA со скоростью работы на ARM была написана утилита fsearch. Для ее вызова используется следующий синтаксис:

fsearch -h haystack\_file -n needle\_file -o output\_file [-f]

где:

* haystack\_file – путь к файлу с текстами для поиска, разделенными символом переноса (UNIX-style).
* needle\_file – путь к файлу с шаблонами поиска, разделенными символом переноса (UNIX-style).
* output\_file – файл с результатами поиска. В каждой строке данного файла через пробел перечислены количества совпадений шаблона из соответствующей строки файла needle\_file в каждом из текстов файла haystack\_file.
* f – флаг, включающий режим поиска с использованием FPGA. При отсутствии данного флага поиск будет осуществлен с использованием ядра ARM.

# Результат

На данный момент остались нерешенными две проблемы:

1. Неправильное измерение времени работы поиска с использованием вызова системных функций. Данная проблема, вероятно, возникла из-за несоответствия конфигурации ОС Linux и аппаратного обеспечения платы ZedBoard.
2. Проблема целостности данных, отправляемых в блочную память модуля поиска реализованного в FPGA. Данная проблема проявляется в виде не полного соответствия результатов поиска с использованием блока в FPGA с результатом поиска с использованием ARM. Т. к. это несоответствие носит случайный характер (результаты запуска программы с одинаковыми параметрами также отличаются друг от друга), то версия о неверной реализации алгоритма поиска исключена. Данная проблема, скорее всего, обусловлена невозможностью синхронизации отправки данных в FPGA и старта поиска при текущей реализации системы. Вероятно, что написание полноценного драйвера для OC Linux полностью решит эту проблему.

Несмотря на невозможность точно сравнить время работы двух режимов поиска, субъективная оценка свидетельствует о более быстром исполнении алгоритма на FPGA по сравнению с алгоритмом на ARM на одних и тех же данных. Стоит отметить, что тактовая частота процессорной системы равна примерно 600 МГц, в то время как блок поиска на FPGA тактируется всего от 100 МГц, что также свидетельствует в пользу эффективности поиска с использованием FPGA. Напоследок, стоит отметить, что при субъективной оценке учитывалось также время загрузки шаблона и текста в FPGA, что может составлять значительную часть времени работы поиска. Решение проблемы с измерением времени позволит оценить временные затраты непосредственно на сам поиск.