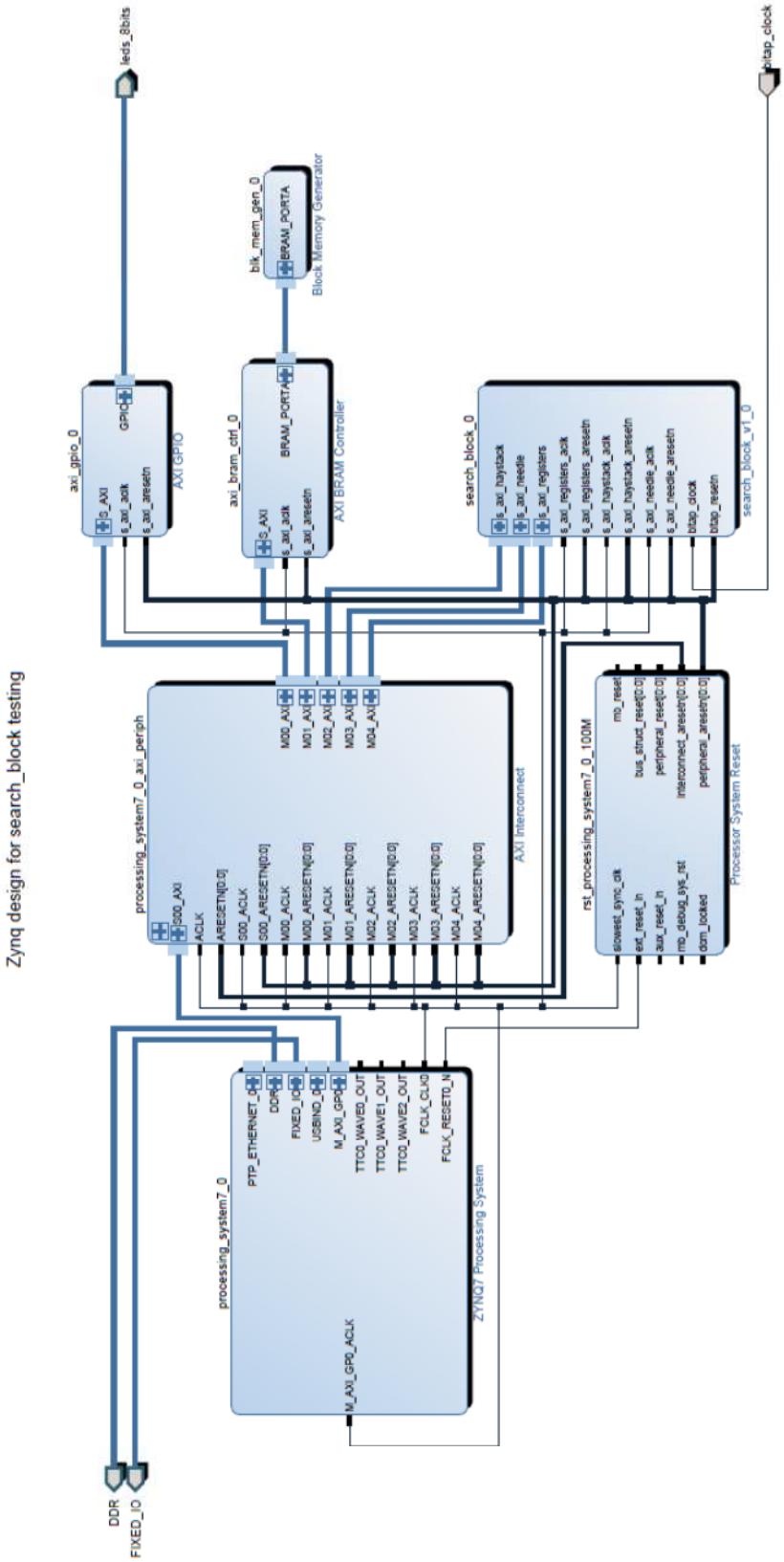
ПРЕДВАРИТЕЛЬНЫЙ ОТЧЕТ О РЕАЛИЗАЦИИ СИСТЕМЫ ПОИСКА ПОДСТРОКИ НА FPGA

ОБЩАЯ СХЕМА СИСТЕМЫ



РЕАЛИЗАЦИЯ ДВОИЧНОГО ПОИСКА НА С

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 ОС 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.