Systemy komputerowe: architektura i programowanie

Projekt

Tworzenie układów SoC z peryferiami wytworzonymi przez siebie i emulowanymi przez spersonalizowany program QEMU oraz testowanie tego systemu z wykorzystaniem tworzonej dla tego środowiska dystrybucji systemu Linux i odpowiednich sprzętowi sterowników systemowych

Patryk Figiel 325270

Politechnika Warszawska

2 czerwca 2024

Spis treści

1. Wstęp

Celem niniejszego projektu jest implementacja oraz testowanie systemu umożliwiającego wyznaczanie liczb pierwszych w czasie rzeczywistym. Projekt składa się z trzech głównych części: modułu w języku Verilog, modułu jądra systemu Linux oraz aplikacji użytkowej.

W module Verilog zaimplementowano algorytm wyznaczania liczb pierwszych, który jest uruchamiany na płycie FPGA. Moduł ten komunikuje się z systemem operacyjnym poprzez interfejs AXI, dzięki czemu możliwa jest kontrola oraz odczyt wyników z poziomu systemu.

Moduł jądra Linux odpowiedzialny jest za obsługę komunikacji między aplikacją użytkową a modułem Verilog. Za pomocą interfejsu sysfs umożliwia odczyt oraz zapis wartości używanych przez algorytm w module Verilog.

Aplikacja użytkowa jest odpowiedzialna za przeprowadzenie testów systemu, których celem jest zweryfikowanie poprawności działania algorytmu wyznaczania liczb pierwszych. Testy te wykonują się automatycznie, przekazując różne wartości wejściowe do modułu jądra i porównując otrzymane wyniki z oczekiwanymi.

Zakres projektu obejmuje implementację każdej z tych części oraz przeprowadzenie testów w celu potwierdzenia poprawności działania systemu. Niniejszy raport zawiera opisy każdej z części projektu oraz wyniki przeprowadzonych testów.

2. Moduł w Verilogu

2.1. Kod

```
/* verilator lint_off UNUSED */
/* verilator lint_off UNDRIVEN */
/* verilator lint_off MULTIDRIVEN */
/* verilator lint_off COMBDLY */
/* verilator lint_off WIDTH */
/* verilator lint_off BLKSEQ */

module gpioemu(
    input           n_reset,
    input [15:0]    saddress,
    input           srd,
    input           swr,
    input [31:0]    sdata_in,
    output [31:0]   sdata_out,
    input [31:0]    gpio_in,
    input           gpio_latch,
    output [31:0]   gpio_out,
    input           clk,
    output [31:0]   gpio_in_s_insp
);

    reg [31:0] sdata_out_s;     //stan magistrali danych - wyjscie
    reg [31:0] gpio_in_s;       //stan peryferii wyjsciowych (do polaczenia
                                //z np.: klawiszami)

    reg [31:0] gpio_out_s;      //stan peryferii wejsciowych (stan wyjsc
                                //- ale nie laczony z np.: LED'ami)

    reg [2:0]  state;           // status
    reg [31:0] A;               // ilosc liczb pierwszych do wyznaczenia
    reg [31:0] S;               // stan automatu -> 0-inicjacja,
                                //1-liczenie, 2-wyliczono

    reg [31:0] W;               // wynik, po znalezieniu liczby pierwszej

    integer counter;
    integer is_prime; // 1-prime, 0-not prime
    integer b;
    integer prime;
```

```
40      integer i;
41
42
43      // Zerowanie i inicjacja zmiennych
44      always @(negedge n_reset) begin
45          gpio_in_s <= 0;
46          gpio_out_s <= 0;
47          sdata_out_s <= 0;
48          W <= 0;
49          A <= 0;
50          i = 0;
51          counter = 0;
52      end
53
54      // Obsluga zapisu
55      always @(posedge swr) begin
56          if (saddress == 16'h100) begin
57              A <= sdata_in;
58              W <= 0;
59              S <= 0;
60              state <= 0;
61          end
62      end
63
64      // Obsluga odczytu
65      always @(posedge srd) begin
66          case (saddress)
67              16'h110: sdata_out_s <= W[31:0];
68              16'h120: sdata_out_s <= S[31:0];
69              default:   sdata_out_s <= 0;
70
71              endcase
72      end
73
74      // Obsluga logiki automatu
75      always @(posedge clk) begin
76          case(state)
77              0: begin
78                  S <= 1;
79                  is_prime = 1;
80                  b = 2;
81                  state <= 1;
82                  i = 2;
83              end
84              1: begin
85                  if (counter < A) begin
86                      is_prime = 1;
87                      b = 2;
88                      while (b <= i**(0.5)) begin
89                          if (i % b == 0) begin
90                              is_prime = 0;
91                          end
92                          b = b + 1;
93                      end
94                      if (is_prime) begin
95                          prime = i;
96                          W <= prime;
97                          counter = counter + 1;
98                      end
99                      i = i + 1;
100                 end else begin
101                     state <= 2;
102                 end
103             end
104             2: begin
```

```
                S <= 2;
                W <= prime;
                gpio_out_s <= gpio_out_s + counter;
                counter = 0;
                prime = 0;
                is_prime = 0;
                b = 0;
                i = 0;
                state <= 3;
            end
        endcase
    end

    // Przypisanie sygnalow wyjsciowych
    assign gpio_out = {16'h0, gpio_out_s[15:0]};
    assign gpio_in_s_insp = gpio_in_s;
    assign sdata_out = sdata_out_s;

endmodule
```

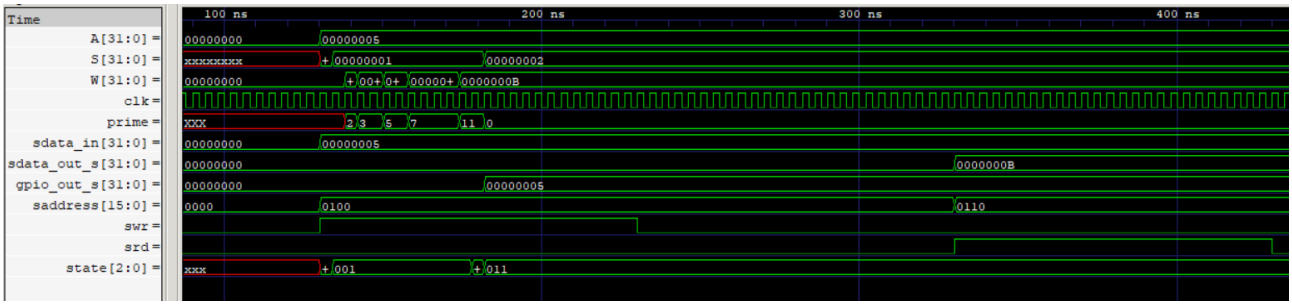
2.2. Testy

Do sprawdzenia poprawności przygotowanego modułu stworzyłem test bench sprawdzający poprawność znajdywania A-tych liczb pierwszych. Wykonałem 6 testów dla liczb: 5, 10, 25, 60, 100, 450.

Graficzna analiza symulacji potwiedziła działanie algorytmu. Do rejestru sdata_in podawana jest liczba, stan swr ustawiany jest na 1 i automat zaczyna swoją pracę. Po zainicjowaniu wartości w stanie 0 przechodzi on na stan drugi i rozpczyna wyliczanie A-tej liczby pierwszej, po skończonej pracy stan ustawiany jest na 2. Wyniki są dostępne w rejestrze W, ostatni zapis to wyliczona N-ta liczba pierwsza. Na wyjściu gpio_out_s dostępna jest informacja o liczbie wyliczonych liczb pierwszych od momentu włączenia systemu.

Test dla A = 5

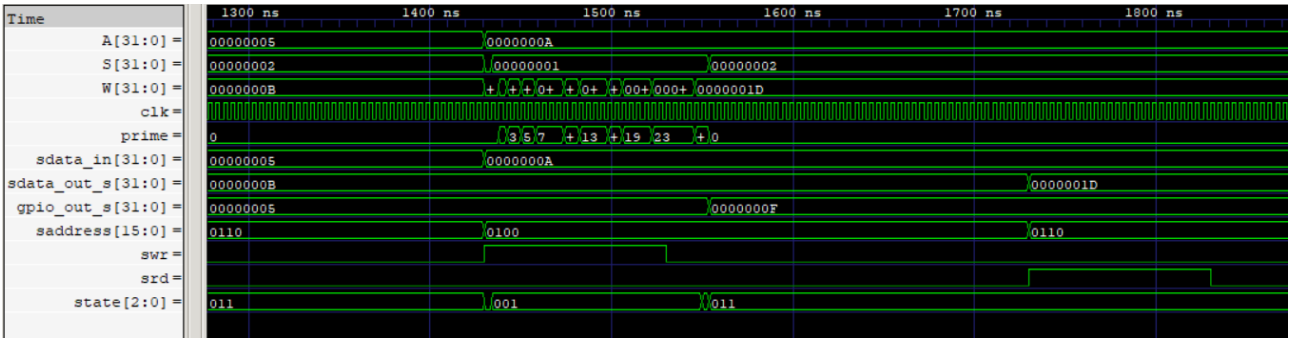
Na wejście A podano liczbę HEX 5. Otrzymano wynik B, czyli po przeliczeniu 11, co jest poprawnym wynikiem. Ilość zliczonych liczb od włączenia systemu wynosi: 5



Rysunek 1. Wynik symulacji dla testu 1

Test dla N = 10

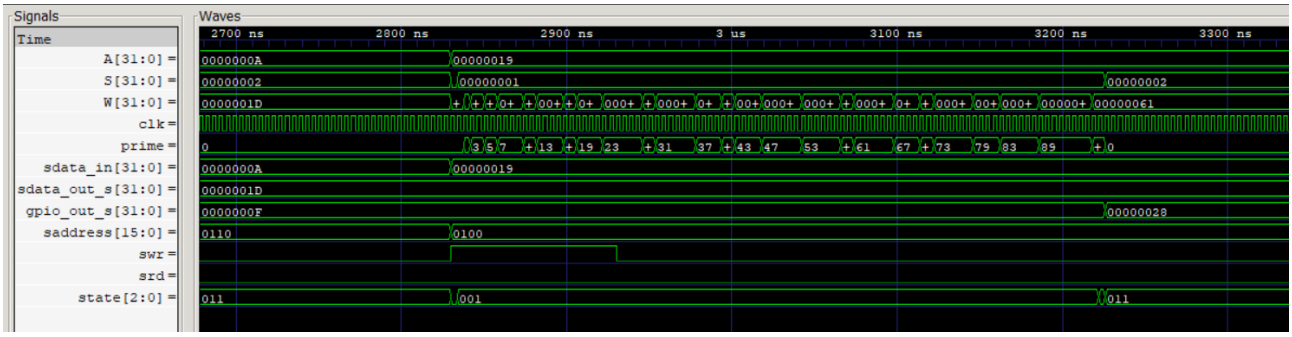
Na wejście A podano liczbę HEX A (dziesiętnie: 10). Otrzymano wynik 1D, czyli po przeliczeniu 29, co jest poprawnym wynikiem. Ilość zliczonych liczb od włączenia systemu wynosi HEX F czyli dziesiętnie 15.



Rysunek 2. Wynik symulacji dla testu 2

Test dla N = 25

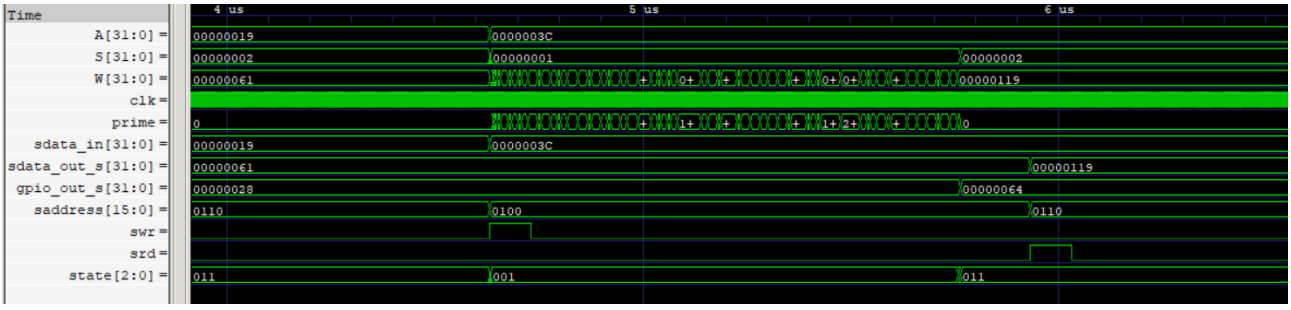
Na wejście A podano liczbę HEX 19 (dziesiętnie: 25). Otrzymano wynik 61, czyli po przeliczeniu 97, co jest poprawnym wynikiem. Ilość zliczonych liczb od włączenia systemu wynosi HEX 28 czyli dziesiętnie 40.



Rysunek 3. Wynik symulacji dla testu 3

Test dla N = 60

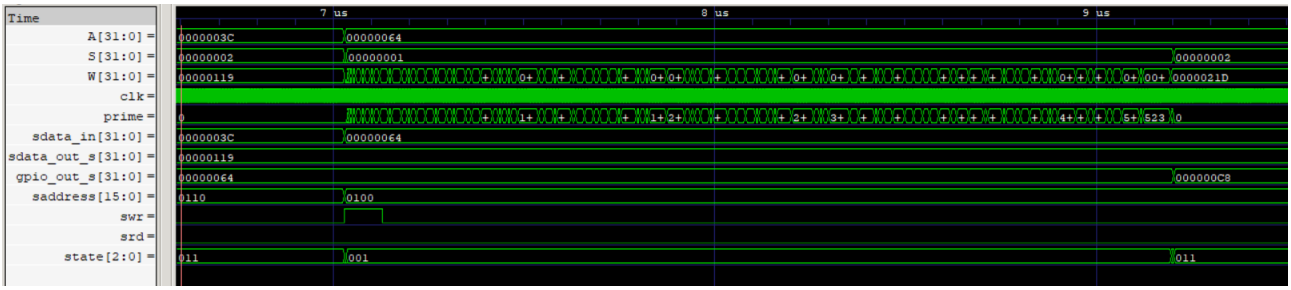
Na wejście A podano liczbę HEX 3C (dziesiętnie: 60). Otrzymano wynik 119, czyli po przeliczeniu 281, co jest poprawnym wynikiem. Ilość zliczonych liczb od włączenia systemu wynosi HEX 64 czyli dziesiętnie 100.



Rysunek 4. Wynik symulacji dla testu 4

Test dla N = 100

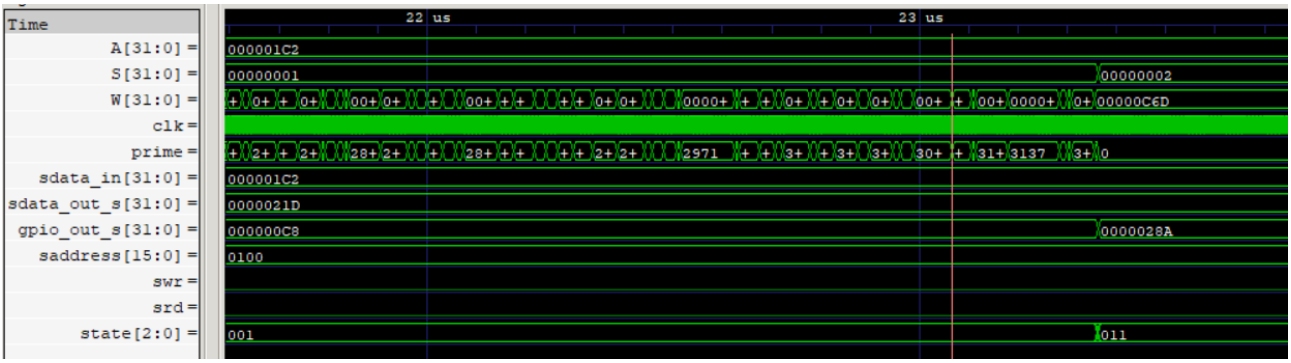
Na wejście A podano liczbę HEX 64 (dziesiętnie: 100). Otrzymano wynik 21D, czyli po przeliczeniu 541, co jest poprawnym wynikiem. Ilość zliczonych liczb od włączenia systemu wynosi HEX C8 czyli dziesiętnie 200.



Rysunek 5. Wynik symulacji dla testu 5

Test dla N = 450

Na wejście A podano liczbę HEX 1C2 (dziesiętnie: 450). Otrzymano wynik C6D, czyli po przeliczeniu 3181, co jest poprawnym wynikiem. Ilość zliczonych liczb od włączenia systemu wynosi HEX 28A czyli dziesiętnie 650.



Rysunek 6. Wynik symulacji dla testu 6

3. Moduł jądra

Funkcjonowanie modułu opiera się na interakcji poprzez pliki do komunikacji, które są obsługiwane przez system plików sysfs. W momencie inicjalizacji modułu, funkcja my_init_module tworzy niezbędne pliki do komunikacji, umożliwiając odczyt i zapis danych. Te pliki są następnie używane do przekazywania danych między modułem a użytkownikiem. Głównym zadaniem modułu jest przeniesienie danych z tych plików do odpowiednich obszarów w pamięci systemu. Po zakończeniu działania, funkcja my_cleanup_module usuwa stworzone pliki do komunikacji, zapewniając sprzątnięcie po sobie i zwolnienie zasobów systemowych.

3.1. Implementacja

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/ioport.h>
#include <linux/types.h>
#include <asm/errno.h>
#include <asm/io.h>

MODULE_INFO(intree, "Y");
MODULE_LICENSE("GPL v2");
MODULE_AUTHOR("Patryk Figiel");
MODULE_DESCRIPTION("Simple kernel module for SYKOM project");
MODULE_VERSION("1.0");

#define SYKT_GPIO_BASE_ADDR  (0x00100000)
#define SYKT_GPIO_SIZE       (0x8000)
#define SYKT_EXIT            (031463)
#define SYKT_EXIT_CODE       (0177)

#define SYKT_GPIO_ADDR_SPACE (baseptr)
#define A (SYKT_GPIO_ADDR_SPACE + 0x100)
#define W (SYKT_GPIO_ADDR_SPACE + 0x110)
#define S (SYKT_GPIO_ADDR_SPACE + 0x120)

void __iomem *baseptr;
static struct kobject *kobj_ref;

static int rejAValue;
static int rejWValue;
static int rejSValue;

// Write to A
static ssize_t rejA_store(struct kobject *kobj, struct kobj_attribute *attr,
const char *buf, size_t count)
{
    int ret;
    // Parse the input buffer for an octal integer
    ret = sscanf(buf, "%o", &rejAValue);
    if (ret != 1) {
        printk(KERN_ERR "Failed to parse input. Expected one octal number,
        got %d\n", ret);
        return -EINVAL; // Error: invalid number of arguments
    }

    // Write the parsed value to the hardware register A
    writel(rejAValue, A);
    return count;
}

// Read from A
static ssize_t rejA_show(struct kobject *kobj, struct kobj_attribute *attr,
char *buf)
{
    // Read the value from the hardware register A
```

```
    rejAValue = readl(A);
    return sprintf(buf, "%o", rejAValue);
}

// Read from W
static ssize_t rejW_show(struct kobject *kobj, struct kobj_attribute *attr,
char *buf)
{
    // Read the value from the hardware register W
    rejWValue = readl(W);
    return sprintf(buf, "%o", rejWValue);
}

// Read from S
static ssize_t rejS_show(struct kobject *kobj, struct kobj_attribute *attr,
char *buf)
{
    // Read the value from the hardware register S
    rejSValue = readl(S);
    return sprintf(buf, "%o", rejSValue);
}

// Sysfs attributes
static struct kobj_attribute rejA_attr = __ATTR_RW(rejA);
static struct kobj_attribute rejW_attr = __ATTR_RO(rejW);
static struct kobj_attribute rejS_attr = __ATTR_RO(rejS);

int my_init_module(void) // Initialization
{
    int ret;

    printk(KERN_INFO "Init my module.\n");

    // Remap the GPIO base address
    baseptr = ioremap_nocache(SYKT_GPIO_BASE_ADDR, SYKT_GPIO_SIZE);
    if (!baseptr) {
        printk(KERN_ERR "Failed to remap GPIO address space\n");
        return -ENXIO;
    }

    // Create a kobject and add it to the sysfs
    kobj_ref = kobject_create_and_add("proj4figpat", kernel_kobj);
    if (!kobj_ref) {
        printk(KERN_ERR "Failed to create sysfs entry\n");
        iounmap(baseptr);
        return -ENOMEM;
    }

    // Create sysfs files
    ret = sysfs_create_file(kobj_ref, &rejA_attr.attr);
    if (ret) {
        printk(KERN_ERR "Cannot create sysfs file rejA\n");
        goto error_cleanup;
    }

    ret = sysfs_create_file(kobj_ref, &rejW_attr.attr);
    if (ret) {
        printk(KERN_ERR "Cannot create sysfs file rejW\n");
        goto error_cleanup;
    }

    ret = sysfs_create_file(kobj_ref, &rejS_attr.attr);
    if (ret) {
        printk(KERN_ERR "Cannot create sysfs file rejS\n");
        goto error_cleanup;
```

```
    }

    return 0;

error_cleanup:
    // Cleanup on error
    sysfs_remove_file(kobj_ref, &rejA_attr.attr);
    sysfs_remove_file(kobj_ref, &rejW_attr.attr);
    sysfs_remove_file(kobj_ref, &rejS_attr.attr);
    kobject_put(kobj_ref);
    iounmap(baseptr);
    return ret;
}

void my_cleanup_module(void) // Cleanup
{
    printk(KERN_INFO "Cleanup my module, Bye\n");

    // Write the exit code to the hardware register
    writel(SYKT_EXIT | (SYKT_EXIT_CODE << 16), baseptr);

    // Remove sysfs files and kobject
    sysfs_remove_file(kobj_ref, &rejA_attr.attr);
    sysfs_remove_file(kobj_ref, &rejW_attr.attr);
    sysfs_remove_file(kobj_ref, &rejS_attr.attr);
    kobject_put(kobj_ref);

    // Unmap the GPIO base address
    iounmap(baseptr);
}

module_init(my_init_module)
module_exit(my_cleanup_module)
```

Listing 1. Kod w języku C

3.2. Test

Przetestowałem działanie jądra z użyciem komend echo i cat zapisując liczby w postaci oktalnej do rejA oraz odczytując je również oktalnie z rejestru rejW. Sprawdziłem poprawność dla przykładowych liczb, skrypt przedstawia się następująco:

```
cd sys/kernel/proj4figpat
echo 5 > rejA
cat rejW
13#
echo 12 > rejA
cat rejW
35#
echo 31 > rejA
cat rejW
141#
echo 74 > rejA
cat rejW
431#
```

Listing 2. Skrypt w terminalu

Sprawdziłem także odporność na podawanie błędnych danych, próbując podać pusty argument lub litery. Wyświetlany jest komunikat o niepoprawnym argumencie co zapewnia stabilność systemu.

```
# echo > rejA
sh: write error: Invalid argument
# echo rr > rejA
sh: write error: Invalid argument
```

Rysunek 7. Sprawdzenie odporności systemu na błedy podanych danych

4. Aplikacja

Po potwierdzeniu poprawności działania za pomocą symulacji graficznej oraz po stwierdzeniu poprawnego odczytu i zapisu danych do scieżek stworzyłem aplikację, która sama przeprowadza testy systemu.

4.1. Kod aplikacji

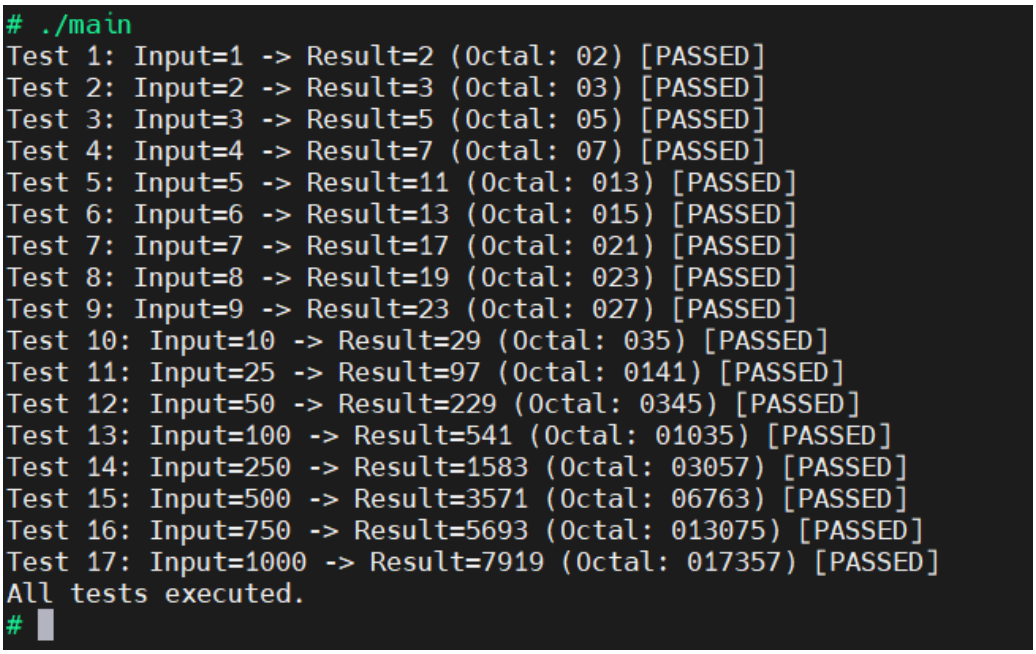
```c
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>

#define MAX_BUFFER 1024
#define FILE_A_PATH "/sys/kernel/proj4figpat/rejA"
#define FILE_W_PATH "/sys/kernel/proj4figpat/rejW"
#define FILE_S_PATH "/sys/kernel/proj4figpat/rejS"

unsigned int read_from_file(const char *file_path);
int write_to_file(const char *file_path, unsigned int value);
void run_tests(void);
unsigned int calculate_prime(unsigned int input_value);

int main(void) {
    run_tests();
    return 0;
}

unsigned int read_from_file(const char *file_path) {
    FILE *file = fopen(file_path, "r");
    if (file == NULL) {
        perror("Error opening file");
        exit(EXIT_FAILURE);
    }

    char buffer[MAX_BUFFER];
    if (fgets(buffer, MAX_BUFFER, file) == NULL) {
        perror("Error reading from file");
        fclose(file);
        exit(EXIT_FAILURE);
    }
    fclose(file);

    return strtoul(buffer, NULL, 8); // Use base 8 for octal numbers
}

int write_to_file(const char *file_path, unsigned int value) {
    FILE *file = fopen(file_path, "w");
    if (file == NULL) {
        perror("Error opening file");
        exit(EXIT_FAILURE);
    }

```

```
    if (fprintf(file, "%o", value) < 0) { // Use %o format for octal numbers
        perror("Error writing to file");
        fclose(file);
        exit(EXIT_FAILURE);
    }
    fclose(file);
    return 0;
}

unsigned int calculate_prime(unsigned int input_value) {
    write_to_file(FILE_A_PATH, input_value);

    unsigned int status;
    unsigned int result;

    while (1) {
        status = read_from_file(FILE_S_PATH);
        if (status == 2) {
            result = read_from_file(FILE_W_PATH);
            return result;
        }
    }
    return result; // Never reached, but added for completeness
}

void run_tests(void) {
    unsigned int input_values[] =    {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 25,
                                50, 100, 250, 500, 750, 1000};
    unsigned int expected_values[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29,
                                97, 229, 541, 1583, 3571, 5693, 7919};


    for (int i = 0; i < sizeof(input_values) / sizeof(input_values[0]); i++) {
        unsigned int result = calculate_prime(input_values[i]);
        if (result == expected_values[i]) {
            printf("Test %d: Input=%u -> Result=%u (Octal: 0%o)
                    [PASSED]\n", i + 1, input_values[i], result, result);
        } else {
            printf("Test %d: Input=%u -> Expected=%u, Got=%u
                [FAILED]\n", i + 1, input_values[i], expected_values[i], result);
        }
    }
    printf("All tests executed.\n");
}
```

Listing 3. Skrypt w terminalu

4.2. Test

W celu weryfikacji poprawności aplikacja wykonała 17 testów dla róznych liczb. Wszystkie test wskazały na poprawność wykonania systemu co prezentuje się poniżej:



Rysunek 8. Wynik testu aplikacji

5. Podsumowanie

Zakończam raport projektowy, z zadowoleniem informując, że udało mi się zrealizować postawione zadanie w zakładanym czasie. To doświadczenie pozwoliło mi praktycznie zastosować wiedzę zdobytą w obszarze tworzenia jądra systemu Linux, co było doskonałą okazją do ugruntowania moich umiejętności w obszarze manipulacji plikami w tym środowisku.