

# Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI  
I TECHNIK INFORMACYJNYCH



przedmiot  
SYKOM 24L



Systemy komputerowe: architektura i programowanie (SYKOM)

PROJEKT

Karol Żelazowski

Numer albumu 324953

prowadzący  
mgr inz. Aleksander Pruszkowski

WARSZAWA 2 czerwca 2024

## Spis treści

<b>1. Wstęp</b>	3
1.1. Spersonalizowane wartości	3
<b>2. Moduł Verilog</b>	3
2.1. Zadanie modułu	3
2.2. Zapis do rejestru A	4
2.3. Odczyt z rejestrów	4
2.4. Algorytm wyznaczania liczby pierwszej	5
2.5. Testy	7
2.5.1. Sprawdzenie poprawności działania zapisu do rejestru A oraz działania algorytmu	7
2.5.2. Test neutralności	7
2.5.3. Odczytanie rejestru stanu	8
2.5.4. Odczytanie rejestru wyniku	8
2.5.5. Odczytanie rejestru argumentu	8
2.5.6. Próba odczytania złego adresu	9
<b>3. System Linux - jądro</b>	10
<b>4. Uruchomienie systemu oraz testy utworzonego modułu z użyciem emulatora     QEMU</b>	10
4.1. Próba wpisania niepoprawnego formatu liczby	10
4.2. Zapis i odczytanie rejestru A	11
4.3. Test poprawności działania algorytmu	11
4.4. Test podania za dużej liczby	11
4.5. Test ogólnego działania	12
<b>5. Aplikacja testująca działanie systemu</b>	13
5.1. Test poprawności działania	13
5.2. Kolejny test poprawności	14
5.3. Test nieprawidłowego argumentu	14
<b>6. Listing modułów</b>	15
6.1. gpioemu.v	15
6.2. kernel_module.c	19
<b>7. main.c</b>	24

# 1. Wstęp

Tematem projektu było stworzenie układu SoC z wytworzonymi przez siebie peryferiami i emulowanymi przez program QEMU. Do przetestowania takiego układu należało stworzyć specjalnie przygotowaną dystrybucję systemu Linux oraz odpowiednich sterowników systemowych

## 1.1. Spersonalizowane wartości

- SYKT\_GPIO\_ADDR\_SPACE - ustalony na podstawie konfiguracji wewnętrznej QEMU.
- A - wyznaczony jako SYKT\_GPIO\_ADDR\_SPACE + 0x288  
dostępny przez plik: /proc/proj4zelkar/rejA
- S - wyznaczony jako SYKT\_GPIO\_ADDR\_SPACE + 0x2A0  
dostępny przez plik: /proc/proj4zelkar/rejS
- W - wyznaczony jako SYKT\_GPIO\_ADDR\_SPACE + 0x298  
dostępny przez plik: /proc/proj4zelkar/rejW
- Moduł jądra będzie komunikował się w systemie OCT.

# 2. Moduł Verilog

## 2.1. Zadanie modułu

Moduł verilog w pliku gpioemu.v ma realizować operację wyznaczania N-tej liczby pierwszej. Jej numer ma być podawany przez rejestr argumentu (A). W założeniu jest, że argument ma być nie większy od 1000. Moment wpisania argumentu ma uruchamiać automat wyznaczający N-tą liczbę pierwszą. Aktualny stan automatu ma być dostępny po przez rejestr stanu (S). Będzie on przyjmował cztery możliwe wartości: 0 - **IDLE** czas oczekiwania na argument; 1 - **CHECKING\_PRIME** moment sprawdzania czy dana liczba jest liczbą pierwszą; 3 - **CALCULATING** stan potrzebny do przejścia do kolejnej liczby; 2 - **FOUND** - stan, w którym automat się znajdzie po znalezieniu zadanej liczby pierwszej. Po znalezieniu N'tej liczby jej wartość będzie dostępna w 32 bitowym rejestrze wyniku (W). Na wyprowadzeniu GPIO modułu gpioemu pojawia się liczba znalezionych liczb pierwszych od włączenia systemu.

### 2.2. Zapis do rejestru A

Jak widać na poniższym kodzie zapis do rejestru A jest tylko możliwy gdy podany jest sygnał swr oraz gdy saddress przyjmuje odpowiednią wartość w tym wypadku 0x288. Przy podaniu argumentu resetowany jest wynik poprzedniego liczenia, ustawiany jest odpowiedni stan automatu oraz zmienne potrzebne algorytmowi

```
always @(posedge swr)
begin
    if (saddress == 16'h288)
    begin
        W <= 0;
        A <= sdata_in[9:0];
        S <= CALCULATING;
        current_prime <= 2;
        prime_count <= 0;
    end
end
```

### 2.3. Odczyt z rejestrów

Jak widać na poniższym kodzie odczyt z rejestrów jest możliwy, gdy jest aktywny sygnał srd. Odpowiednia wartość jest przypisywana do sdata\_out\_s, gdy jest podana wartość na saddress, której odpowiada jeden z trzech dostępnych do odczytania rejestrów. W innym przypadku do sdata\_out\_s przypisywane jest 0.

```
always @(posedge srd)
begin
    if (saddress == 16'h288)
    begin
        sdata_out_s <= {22'b0, A[9:0]};
    end
    else if (saddress == 16'h2A0)
    begin
        sdata_out_s <= {29'b0, S[2:0]};
    end
    else if (saddress == 16'h298)
    begin
        sdata_out_s <= W;
    end
    else
    begin
        sdata_out_s <= 0;
    end
end
```

```

        end
    end

```

## 2.4. Algorytm wyznaczania liczby pierwszej

Algorytm działa na zasadzie sprawdzenia czy dana liczba jest pierwsza i jeśli taka jest zlicza je do momentu osiągnięcia wymaganej liczby. Aby sprawdzić czy dana liczba jest pierwsza, korzysta z dzielenia liczby  $n$  przez kolejne liczby począwszy od 2 aż do wartości pierwiastka kwadratowego z  $n$ . Gdy dana liczba nie jest podzielna przez żadną wartość z danego przedziału oznacza to, że jest pierwsza.

```

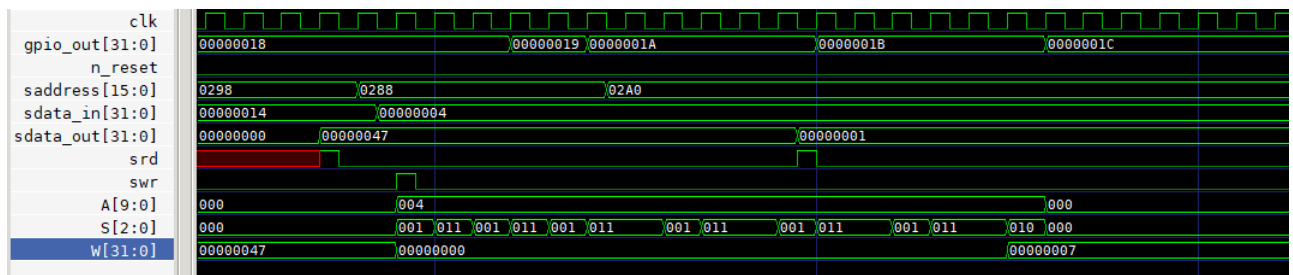
always @(posedge clk)
begin
    case(S)
        IDLE:
        begin
            if (A != 0)
            begin
                S <= CALCULATING;
                current_prime <= 2;
                prime_count <= 0;
                number_of_numbers <= 0;
            end
        end
        CALCULATING:
        begin
            S <= CHECKING_PRIME;
            i <= 2;
            prime_flag <= 1;
        end
        CHECKING_PRIME:
        begin
            if (i * i > current_prime)
            begin
                if (prime_flag)
                begin
                    if (prime_count == A - 1)
                    begin
current_prime <= current_prime + 1;
                        prime_count <= prime_count + 1;
                        number_of_numbers <= number_of_numbers + 1;
                        S <= FOUND;
                    end
                end
            end
        end
    endcase
end

```

```
        W <= current_prime;
    end
    else
    begin
        current_prime <= current_prime + 1;
        prime_count <= prime_count + 1;
        number_of_numbers <= number_of_numbers + 1;
        S <= CALCULATING;
    end
end
else
begin
    current_prime <= current_prime + 1;
    S <= CALCULATING;
end
end
else
begin
    if (current_prime % i == 0)
        prime_flag <= 0;
        i <= i + 1;
    end
end
FOUND:
begin
    prime_count <= 0;
    A <= 0;
    S <= IDLE;
end
endcase
end
```



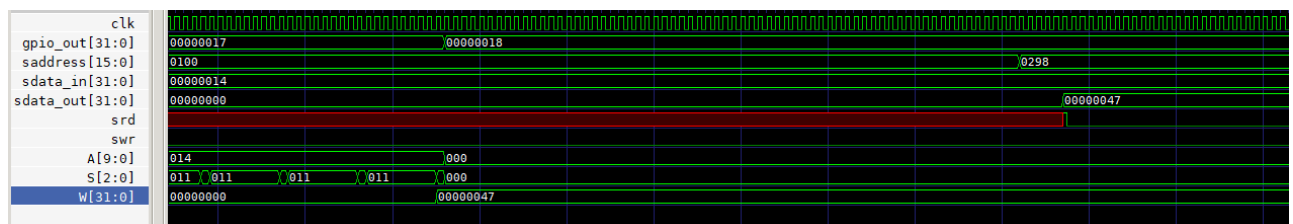
### 2.5.3. Odczytanie rejestru stanu



Rysunek 2.3. Obraz symulacji odczytania rejestru stanu z aplikacji gtkwave

W trakcie obliczania n-tej liczby pierwszej na szynę saddress podany jest adres 0x2A0, który odpowiada rejestrowi stanu. Przy tym gdy zostało podane wzniesienie srd na sdata\_out zostało przypisany aktualny stan rejestru S, czyli w tym przypadku 0b01 - CALCULATING.

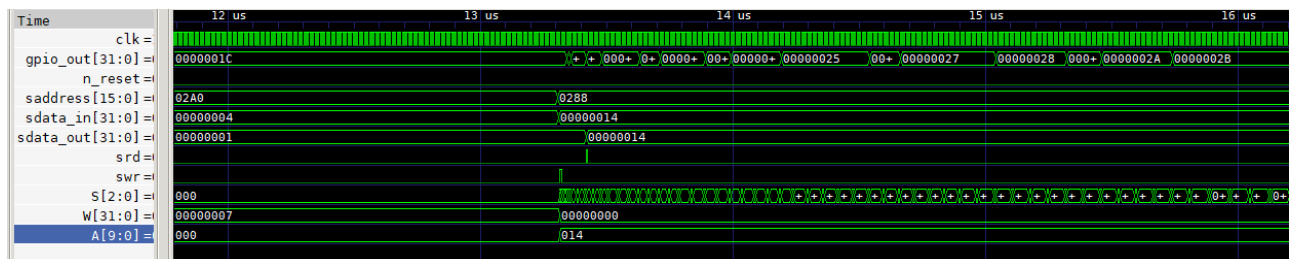
### 2.5.4. Odczytanie rejestru wyniku



Rysunek 2.4. Obraz symulacji odczytania rejestru wyniku z aplikacji gtkwave

Po obliczeniu n-tej liczby pierwszej rezultat został przypisany do rejestru W, następnie na saddress została podana wartość 0x298, co odpowiada rejestrowi wyniku i na sdata\_out, został przypisany aktualny stan rejestru W, czyli 0x47

### 2.5.5. Odczytanie rejestru argumentu

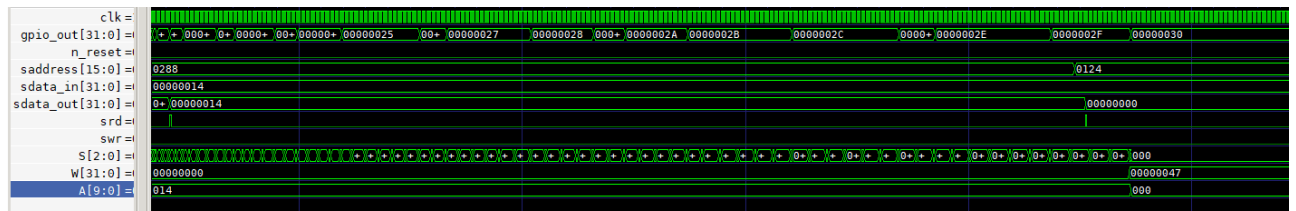


Rysunek 2.5. Obraz symulacji odczytania rejestru argumentu z aplikacji gtkwave

Widzimy, że podaniu wartości 0x298 na saddres na sdata\_out, został przypisany aktualny stan rejestru A, czyli 0x14



### 2.5.6. Próba odczytania złego adresu



**Rysunek 2.6.** Obraz symulacji odczytania złego adresu z aplikacji gtkwave

Widzimy, że po podaniu złego adresu `sdata_out` się zeruje.

### 3. System Linux - jądro

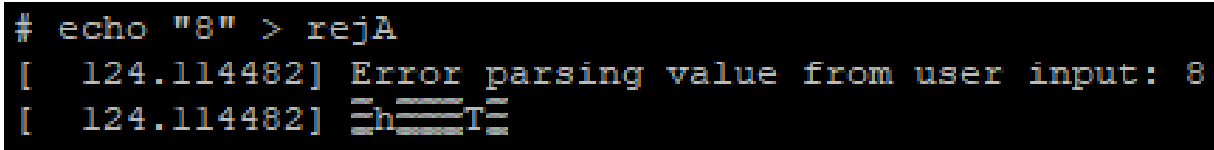
Aby poprawnie uruchomić system za pomocą emulatora QEMU należało utworzyć jądro systemu w pliku `kernel_module.c`, który będzie komunikował się z urządzeniem za pomocą systemu plików `proc-fs`. Dzięki czemu będzie można zapisywać i odczytywać wartości z odpowiednich rejestrów GPIO.

Do właściwego działania narzędzia `make_busybox_kernel_module` potrzebne było stowornie następujących plików: `Config.in` - odpowiada za konfigurację systemu., `kernel_module.mk` - konfiguruje proces budowania i instalacji `kernel_module` za pomocą narzędzia `Buildroot`, `Makefile` - plik do kompilacji modułu jądra. Podczas kompilacji jądra powstały pliki: `fw_jump.elf` - zawiera bootloader, który ma uruchomić jądro systemu operacyjnego, `Image` - binarny obraz jądra, które ładowane jest przez bootloader, `rootfs.ext2` - zawiera narzędzia niezbędne do działania systemu operacyjnego.

### 4. Uruchomienie systemu oraz testy utworzonego modułu z użyciem emulatora QEMU

Po załadowaniu modułu jądra za pomocą polecenia `modprobe kernel_module` należało przetestować poprawność działania modułu. Testy odbywały się za pomocą poleceń `echo arg > file` oraz `cat file`.

#### 4.1. Próba wpisania niepoprawnego formatu liczby



```
# echo "8" > rejA
[ 124.114482] Error parsing value from user input: 8
[ 124.114482] ==h==T==
```

Rysunek 4.1. Obsługa błędu niepoprawnego formatu

Jak widzimy, system nie chciał przyjąć nieodpowiedniego formatu. Jeśli zależało by nam na odzyskaniu ósmej liczby pierwszej należało podać wartość 010.

#### 4. Uruchomienie systemu oraz testy utworzonego modułu z użyciem emulatora QEMU

##### 4.2. Zapis i odczytanie rejestru A

```
# cd proj4zelkar/  
# echo 1000 > rejA  
# cat rejA  
[ 85.959373] Number of prime numbers having been searched  
1000
```

Rysunek 4.2. Zapis i odczytanie rejestru A

##### 4.3. Test poprawności działania algorytmu

```
# echo 12 > rejA  
# cat rejW  
[ 92.244780] Reading result of the computing:  
35
```

Rysunek 4.3. Zapis do A oraz odczytanie z W

Jak widzimy podajemy wartość oktalną 12 czyli decymalną 10. Dziesiątą liczbą pierwszą jest 29, której reprezentacja oktalna to 035

##### 4.4. Test podania za dużej liczby

```
# echo 1751 > rejA  
[ 521.731053] Error: Invalid value of argument.  
sh: write error: Invalid argument  
#
```

Rysunek 4.4. Test podania za dużej liczby

Po podaniu wartości ósemkowej 1751 czyli decymalnie 1001 dostajemy informację, że podajemy nieprawidłowy argument.

#### 4. Uruchomienie systemu oraz testy utworzonego modułu z użyciem emulatora QEMU

---

##### 4.5. Test ogólnego działania

```
# echo 100 > rejA
# cat rejA
[ 201.146980] Number of prime numbers having been searched
```

**Rysunek 4.5.** Zapis do rejestru A

Zapisujemy wartość oktalną 100 do rejestru A czyli decymalnie 64.

```
# cat rejS
[ 339.764410] Reading status of the device
0
[ 339.769037] Reading status of the device
[ 339.771668] Reading status of the device
# cat rejW
[ 342.061382] Reading result of the computing:
467
```

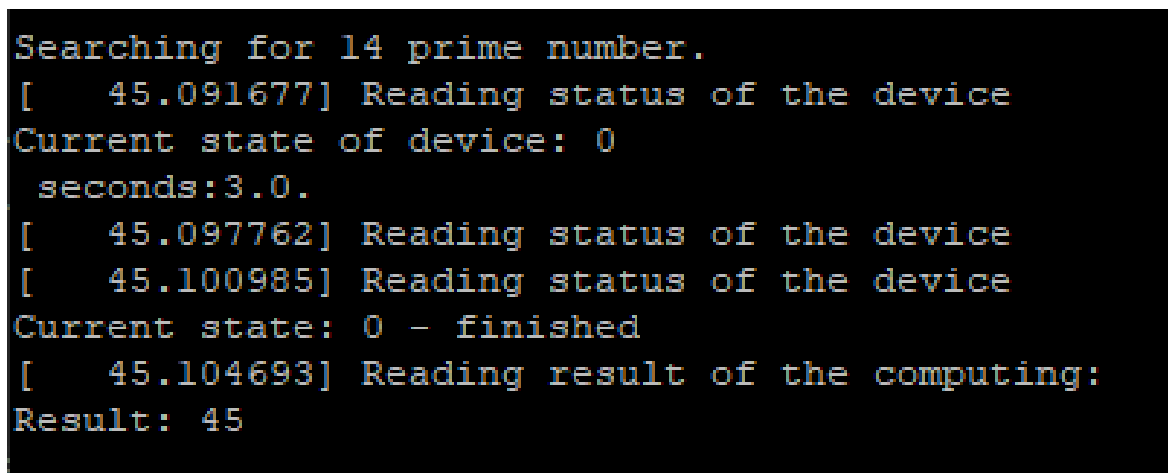
**Rysunek 4.6.** Odczytanie statusu i wyniku

Następnie odczytujemy rejestr statusu i na końcu rejestr wyniku, który zwraca nam oktalną wartość 467, która w reprezentacji decymalnej ma wartość 311, co jest 64. liczbą pierwszą

## 5. Aplikacja testująca działanie systemu

W celu przetestowania poprawności przetestowania całego systemu została użyta aplikacja testująca stworzona w pliku `main.c`. Aby przetestować funkcjonalności systemu modyfikuje ona lub odczytuje pliki znajdujące się w katalogu `/proc/proj4zelkar/`. Żeby utworzyć aplikacje w systemie macierzystym w katalogu zawierającym pliki źródłowe należało użyć polecenia `make_busybox_compile main.c`. Aby dokonać testów należało tak jak w poprzednim punkcie załadować jądro poleceniem `modprobe kernel_module`.

### 5.1. Test poprawności działania



```
Searching for 14 prime number.
[ 45.091677] Reading status of the device
Current state of device: 0
seconds:3.0.
[ 45.097762] Reading status of the device
[ 45.100985] Reading status of the device
Current state: 0 - finished
[ 45.104693] Reading result of the computing:
Result: 45
```

**Rysunek 5.1.** Wynik działania algorytmu dla argumentu: 14 - oktalnie, czyli 12 w systemie dziesiętnym

Widzimy, że system zadziałał poprawnie ponieważ zwrócona wartość w reprezentacji decymalnej ma wartość: 37. Zaznaczoną też tutaj wartością jest odczytany status po 3 sekundach, który wynosił 0 - IDLE.

### 5.2. Kolejny test poprawności

```
Searching for 100 prime number.  
[ 95.111468] Reading status of the device  
Current state of device: 0  
seconds:50.0.  
[ 95.123084] Reading status of the device  
[ 95.125602] Reading status of the device  
Current state: 0 - finished  
[ 95.130120] Reading result of the computing:  
Result: 467
```

**Rysunek 5.2.** Wynik działania algorytmu dla argumentu: 100 - oktalnie, czyli 64 w systemie dziesiętkowym

Widzimy, że system zadziałał poprawnie ponieważ zwrócona wartość w reprezentacji decymalnej ma wartość: 311. Zaznaczoną też tutaj wartością jest odczytany status po 50 sekundach, który wynosił 0 - IDLE.

### 5.3. Test nieprawidłowego argumentu

```
Searching for 200000 prime number.  
[ 66.680637] Error: Ivalid value of argument: Try between 0d0 and 0d1000.  
[ 68.718582] Reading status of the device  
Current state after 3 seconds: 2.0  
[ 70.814023] Reading status of the device  
Current state: 3 - finished  
[ 70.816423] Reading result of the computing:  
Result: 0
```

**Rysunek 5.3.** Wynik przy podaniu za dużego argumentu

Widzimy, że wynik to 0 i użytkownik został informację dlaczego argument jest nieprawny.

## 6. Listing modułów

### 6.1. gpioemu.v

```
/* verilator lint_off UNUSED */
/* verilator lint_off UNDRIVEN */
/* verilator lint_off MULTIDRIVEN */
/* verilator lint_off COMBDLY */
/* verilator lint_off WIDTH */
/* verilator lint_off CASEINCOMPLETE */
module gpioemu(n_reset,                                //magistrala z CPU
               saddress[15:0], srd, swr,
               sdata_in[31:0], sdata_out[31:0],
               gpio_in[31:0], gpio_latch,              //styk z GPIO - in
               gpio_out[31:0],                        //styk z GPIO = out
               clk,                                    //sygnaly opcjonalne - zegar 1 KHz
               gpio_in_s_insp[31:0]);                 //sygnaly testowe

input          clk;
input          n_reset;
input [15:0]    saddress;    //magistrala - adres
input          srd;          //odczyt przez CPU z mag. danych
input          swr;          //zapis przez CPU do mag. danych
input [31:0]    sdata_in;    //magistrala wejscowa CPU
output[31:0]    sdata_out;    //magistrala wyjscowa z CPU
reg [31:0]      sdata_out_s;  //stan magistrali danych -wyjscie
input[31:0]     gpio_in;      //dane z peryferii wejscie do modulu
reg[31:0]       gpio_in_s;    //stan peryferii wyjsciwych
input          gpio_latch;    //zapis danych na gpio_in
output[31:0]    gpio_in_s_insp; //debuging
output[31:0]    gpio_out;     //dane wyjsciwedo peryferii
reg[31:0]       gpio_out_s;   //stan peryferii wejsciwych

reg[31:0]       sdata_in_s;

parameter MAX_PRIME = 1000;
parameter IDLE = 2'b00;
parameter CALCULATING = 2'b01;
parameter CHECKING_PRIME = 2'b11;
parameter FOUND = 2'b10;
```

```
reg unsigned [9:0] A;
reg unsigned [2:0] S;
reg unsigned [31:0] W;
reg unsigned [31:0] number_of_numbers;
reg unsigned [31:0] current_prime;

reg unsigned [31:0] prime_count;
reg unsigned [31:0] i;
reg prime_flag;

always @(negedge n_reset)
begin
    gpio_in_s <= 0;
    gpio_out_s <= 0;
    sdata_out_s <= 0;
    A <= 0;
    S <= IDLE;
    W <= 0;
    number_of_numbers <= 0;
    prime_count <= 0;
    current_prime <= 2;
    i <= 0;
    prime_flag <= 1;
end

always @(posedge gpio_latch) begin
gpio_in_s[31:0] <= gpio_in[31:0];
end

always @(posedge swr)
begin
    if (saddress == 16'h288)
    begin
        W <= 0;
        A <= sdata_in[9:0];
        S <= CALCULATING;
        current_prime <= 2;
        prime_count <= 0;
    end
end
```



```
        end
    end

    always @(posedge srd)
    begin
        if (saddress == 16'h288)
        begin
            sdata_out_s <= {22'b0, A[9:0]};
        end
        else if (saddress == 16'h2A0)
        begin
            sdata_out_s <= {29'b0, S[2:0]};
        end
        else if (saddress == 16'h298)
        begin
            sdata_out_s <= W;
        end
        else
        begin
            sdata_out_s <= 0;
        end
    end

end

always @(posedge clk)
begin
    case(S)
        IDLE:
        begin
            if (A != 0)
            begin
                S <= CALCULATING;
                current_prime <= 2;
                prime_count <= 0;
                number_of_numbers <= 0;
            end
        end
        CALCULATING:
        begin
            S <= CHECKING_PRIME;
            i <= 2;
            prime_flag <= 1;
        end
    endcase
end
```

```
end
CHECKING_PRIME:
begin
    if (i * i > current_prime)
    begin
        if (prime_flag)
        begin
            if (prime_count == A - 1)
            begin
current_prime <= current_prime + 1;
                prime_count <= prime_count + 1;
                number_of_numbers <= number_of_numbers + 1;
                S <= FOUND;
                W <= current_prime;
            end
        end
        else
        begin
            current_prime <= current_prime + 1;
            prime_count <= prime_count + 1;
            number_of_numbers <= number_of_numbers + 1;
            S <= CALCULATING;
        end
    end
    end
    else
    begin
        current_prime <= current_prime + 1;
        S <= CALCULATING;
    end
end
end
else
begin
    if (current_prime % i == 0)
        prime_flag <= 0;
    i <= i + 1;
end
end
end
FOUND:
begin
    prime_count <= 0;
    A <= 0;
    S <= IDLE;
```

```
        end
    endcase
end

always @(posedge clk)
begin
    gpio_out_s <= number_of_numbers;

end

assign gpio_out = gpio_out_s;
assign gpio_in_s_insp = gpio_in_s;
assign sdata_out = sdata_out_s;

endmodule
```

## 6.2. kernel\_module.c

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/ioport.h>
#include <asm/errno.h>
#include <asm/io.h>
#include <linux/fs.h>
#include <linux/proc_fs.h>
#include <linux/uaccess.h>
#include <asm/io.h>

MODULE_INFO(intree, "Y");
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Karol Żelazowski");
MODULE_DESCRIPTION("Simple kernel module for SYKOM lecture");
MODULE_VERSION("0.01");

#define SYKT_GPIO_BASE_ADDR (0x00100000)
#define SYKT_GPIO_SIZE (0x8000)
#define SYKT_EXIT (0x3333)
#define SYKT_EXIT_CODE (0x7F)
```

```
#define A_REG_GPIO_ADDR_OFFSET (0x288)
#define S_REG_GPIO_ADDR_OFFSET (0x2A0)
#define W_REG_GPIO_ADDR_OFFSET (0x298)

#define LEN 32

void __iomem *baseptrA;
void __iomem *baseptrS;
void __iomem *baseptrW;
void __iomem *baseptr;

static struct proc_dir_entry *direc;

static struct proc_dir_entry *proc_file_rej_a;
static struct proc_dir_entry *proc_file_rej_s;
static struct proc_dir_entry *proc_file_rej_w;

static ssize_t my_read_A(struct file *file, char __user *ubuf, size_t len, loff_t *off)
{
    printk(KERN_INFO "Number of prime numbers having been searched\n");
    char buf[LEN];
    int not_copied, to_copy, num;
    if(*off > 0) {
        return 0;
    }
    num = readl(baseptrA);
    snprintf(buf, LEN, "%o\n", num);
    to_copy = strlen(buf);
    if (to_copy > LEN){
        to_copy = LEN;
    }

    not_copied = copy_to_user(ubuf, buf, to_copy);
    if(not_copied){
        printk(KERN_ERR "Error copying data to user space\n");
        return -EFAULT;
    }
}
```

```

*off = to_copy;
    return to_copy;
}

static ssize_t my_write_A(struct file *file, const char __user *ubuf, size_t count,
int num, i;
char buf[LEN];

if(*off>0 || count>LEN)
{
    printk(KERN_ERR "Invalid offset or count exceeds buffer length\n");
    return -EPERM;
}
if(copy_from_user(buf, ubuf, count))
{
    printk(KERN_ERR "Error copying data from user space\n");
    return -EFAULT;
}
num=sscanf(buf, "%o", &i);
if(num!=1) {
    printk(KERN_ERR "Error parsing value from user input: %s\n", buf);
    return -EFAULT;
}
if( i <= 01 || i > 01750){
    printk(KERN_ERR "Error: Ivalid value of argument: Try between 0d0 and 0d1000.\n");
}

writel(i, baseptrA);
return count;

}

static const struct file_operations A_ops = {
    .owner = THIS_MODULE,
    .read = my_read_A,
    .write = my_write_A
};

static ssize_t my_read_S(struct file *file, char __user *ubuf, size_t count, loff_t
    printk(KERN_INFO "Reading status of the device\n");
    char buf[LEN];
    int to_copy, not_copied, num;

```

```
    if (*off > 0) {
        return 0;
    }

    num = readl(baseptrS);
    snprintf(buf, LEN, "%o\n", num);
    to_copy = strlen(buf);

    if (to_copy > LEN) {
        to_copy = LEN;
    }

    not_copied = copy_to_user(ubuf, buf, to_copy);
    if (not_copied) {
        printk(KERN_ERR "Error copying data to user space\n");
        return -EFAULT;
    }

    *off = to_copy;
    return to_copy;
}
```

```
static const struct file_operations S_ops = {
    .owner = THIS_MODULE,
    .read = my_read_S
};
```

```
static ssize_t my_read_W(struct file *file, char __user *ubuf, size_t count, loff_t *
    printk(KERN_INFO "Reading result of the computing:\n");
    char buf[LEN];
    int to_copy, not_copied, num;

    if (*off > 0) {
        return 0;
    }

    num = readl(baseptrW);
    snprintf(buf, LEN, "%o\n", num);
```

```
    to_copy = strlen(buf);

    if (to_copy > LEN) {
        to_copy = LEN;
    }

    not_copied = copy_to_user(ubuf, buf, to_copy);
    if (not_copied) {
        printk(KERN_ERR "Error copying data to user space\n");
        return -EFAULT;
    }

    *off = to_copy;
    return to_copy;
}

static const struct file_operations W_ops = {
    .owner = THIS_MODULE,
    .read = my_read_W
};

int my_init_module(void){
    printk(KERN_INFO "Init my module.\n");
    baseptr=ioremap(SYKT_GPIO_BASE_ADDR, SYKT_GPIO_SIZE);

    if(baseptr == NULL){
        printk("Failed to map GPIO memory!\n");
        return -ENOMEM;
    }

    baseptrA = baseptr + A_REG_GPIO_ADDR_OFFSET;
    baseptrS = baseptr + S_REG_GPIO_ADDR_OFFSET;
    baseptrW = baseptr + W_REG_GPIO_ADDR_OFFSET;

    if(baseptrA == NULL || baseptrS == NULL || baseptrW == NULL ){
        printk("Failed to map registers memmory!\n");
        return -ENOMEM;
    }
    direc = proc_mkdir("proj4zelkar",NULL);
    if(!direc){
        printk(KERN_INFO "Error unable to creat procfs directory!\n");
    }
}
```

```

proc_remove(direc);
return -ENOMEM;
}

proc_file_rej_a = proc_create("rejA", 0666, direc, &A_ops);
proc_file_rej_s = proc_create("rejS", 0444, direc, &S_ops);
proc_file_rej_w = proc_create("rejW", 0444, direc, &W_ops);
if(!proc_file_rej_a || !proc_file_rej_s || !proc_file_rej_w){
    printk(KERN_INFO "Error unable to creat procfs files!\n");
    proc_remove(proc_file_rej_a);
    proc_remove(proc_file_rej_s);
    proc_remove(proc_file_rej_w);
    proc_remove(direc);
}
return 0;
}

void my_cleanup_module(void){
    printk(KERN_INFO "Cleanup my module.\n");
    proc_remove(proc_file_rej_a);
    proc_remove(proc_file_rej_s);
    proc_remove(proc_file_rej_w);
    remove_proc_entry("proj4karzel",NULL);
    writel(SYKT_EXIT | ((SYKT_EXIT_CODE)<<16), baseptr);
    if (baseptrA) {
        iounmap(baseptrA);
    }
    if (baseptrS) {
        iounmap(baseptrS);
    }
    if (baseptrW) {
        iounmap(baseptrW);
    }
    iounmap(baseptr);
}

module_init(my_init_module);
module_exit(my_cleanup_module);

```

## 7. main.c

```

#include <stdio.h>
#include <stdlib.h>

```



```
#include <unistd.h>

#define A_path "/proc/proj4zelkar/rejA"
#define S_path "/proc/proj4zelkar/rejS"
#define W_path "/proc/proj4zelkar/rejW"

int rd_file(char *file_path){
    int i;
    FILE *fptr;

    fptr = fopen(file_path,"r");
    if (fptr == NULL){
        printf("Error! Can not open the file.");
        exit(1);
    }

    fscanf(fptr,"%o", &i);
    fclose(fptr);

    return i;
}

void wrt_file(char *file_path, int input_number){
    FILE *fptr;

    fptr = fopen(file_path, "w");
    if(fptr == NULL){
        printf("Error! Can not open the file.");
        exit(EXIT_FAILURE);
    }

    if(fprintf(fptr, "%o", input_number) < 0){
        printf("Error while trying to write the file.");
        exit(EXIT_FAILURE);
    }

    fclose(fptr);
}

int searching_prime_test(int argument, float sleep_time){
```

```
    if (argument <= 0) {
        fprintf(stderr, "Invalid argument: %d. Argument must be a positive integer.\n",
            argument);
        return -1;
    }

    if (sleep_time < 0) {
        fprintf(stderr, "Invalid sleep time: %f. Sleep time must be non-negative.\n",
            sleep_time);
        return -1;
    }

    printf("\n");
    printf("Searching for %o prime number.\n", argument);
    wrt_file(A_path, argument);
    sleep(sleep_time);
    printf("Current state of device: %o\n seconds:%0.1lf.\n", rd_file(S_path), sleep_time);
    while(rd_file(S_path) != 0 && rd_file(S_path) != 2){
        sleep(0.1);
    }
    printf("Current state: %o - finished\n", rd_file(S_path));
    printf("Result: %o\n", rd_file(W_path));
    return rd_file(W_path);
}

void inv_argument(int argument, float sleep_time1, float sleep_time2){
    if (sleep_time1 < 0 || sleep_time2 < 0) {
        fprintf(stderr, "Invalid sleep times: %f, %f. Sleep times must be non-negative.\n",
            sleep_time1, sleep_time2);
        return;
    }

    printf("\n");
    printf("Searching for %o prime number.\n", argument);
    wrt_file(A_path, argument);
    sleep(sleep_time2);
    printf("Current state after %o seconds: %0.1lf \n", rd_file(S_path), sleep_time2);
    sleep(sleep_time1);
    printf("Current state: %o - finished1\n", rd_file(S_path));
    printf("Result: %o\n", rd_file(W_path));
}
```

```
int main(void){

    printf("\n");
    printf("Tests for correct values:");

    if( searching_prime_test(05, 0.5) == 11 ){
        printf("test passed\n");
    }else{
        printf("test not passed\n");
    }
    if( searching_prime_test(014, 4.0) == 37 ){
        printf("test passed\n");
    }else{
        printf("test not passed\n");
    }
    if( searching_prime_test(0100, 45.0) == 311 ){
        printf("test passed\n");
    }else{
        printf("test not passed\n");
    }

    inv_argument(0200000, 2.0, 2.0);
    inv_argument(-20, 2.0, 2.0);
}
```

