

SYKOM

Projekt

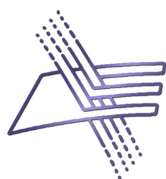
Tworzenie układów SoC z peryferiami emulowanymi przez program QEMU
oraz testowanie systemu z wykorzystaniem dystrybucji systemu Linux i
odpowiednich sterowników systemowych

Jakub Strzelczyk 325325

2 czerwca 2024

Spis treści

1. Cel projektu	2
2. Moduł gpioemu.v	2
2.1. Opis działania	2
2.2. Kod źródłowy modułu	2
2.3. Przeprowadzone testy	4
2.4. Kod źródłowy testbenchu	8
3. Moduł jądra systemu Linux	11
3.1. Przeprowadzone testy	14
4. Aplikacja testująca w języku C	14
4.1. Krótki opis	14
4.2. Kod źródłowy	14
4.3. Przeprowadzone testy	16
5. Podsumowanie	17



**Wydział Elektroniki
i Technik Informatycznych**

POLITECHNIKA WARSZAWSKA

1. Cel projektu

Celem projektu jest zaprojektowanie i wdrożenie systemu na chipie (SoC) z własnoręcznie stworzonymi modułami peryferyjnymi emulowanymi w programie QEMU. Projekt obejmuje stworzenie modułu Verilog obliczającego n-tą liczbę pierwszą, sterownika jądra Linux oraz aplikacji testowej do weryfikacji działania modułu i jego integracji z jądrem systemu.

2. Moduł gpioemu.v

2.1. Opis działania

Moduł Verilog, który zaprojektowałem, służy do wyznaczania N-tej liczby pierwszej, opierając się na automacie stanowym operującym w trzech stanach. W stanie inicjalizacji moduł resetuje wszystkie wewnętrzne rejestry i zmienne, ustawiając licznik liczb pierwszych na 1 (dla liczby 2) i rozpoczynając iterację od liczby 3. Jeśli licznik osiągnie wartość N, przechodzi do stanu końcowego. W stanie liczenia moduł iteruje tylko po liczbach nieparzystych, sprawdzając, czy aktualna liczba jest pierwsza. Jeśli liczba jest pierwsza, zwiększa licznik znalezionych liczb pierwszych i aktualizuje ostatnią znalezioną liczbę pierwszą. Gdy licznik liczb pierwszych osiągnie wartość N, moduł przechodzi do stanu końcowego. W stanie zakończenia obliczeń moduł zapisuje wynik do rejestru W, aktualizuje stan na "obliczone" i resetuje zmienne pomocnicze. Dodatkowo moduł implementuje sygnał statusu (S) z trzema możliwymi stanami: 0 - reset, 1 - w trakcie liczenia, 2 - wynik dostępny.

2.2. Kod źródłowy modułu

Listing 1. GpioEmu.v

```
1  /* verilator lint_off UNUSED */
2  /* verilator lint_off UNDRIVEN */
3  /* verilator lint_off BLKSEQ */
4  /* verilator lint_off MULTIDRIVEN */
5  /* verilator lint_off COMBDLY */
6  /* verilator lint_off WIDTH */
7  module gpioemu(
8      n_reset,
9      saddress[15:0],
10     srd,
11     swr,
12     sdata_in[31:0],
13     sdata_out[31:0],
14     gpio_in[31:0],
15     gpio_latch,
16     gpio_out[31:0],
17     clk,
18     gpio_in_s_insp[31:0]
19 );
20
21 // Deklaracje portow
22 input n_reset;
23 input [15:0] saddress;
24 input srd;
25 input swr;
26 input [31:0] sdata_in;
27 output [31:0] sdata_out;
28
29 input [31:0] gpio_in;
30 input gpio_latch;
31
32 output [31:0] gpio_out;
33 reg [31:0] gpio_in_s;
34 reg [31:0] gpio_out_s;
35 reg [31:0] sdata_out_s;
36 reg [2:0] state;
37
```

```

38 output [31:0] gpio_in_s_insp;
39
40 input clk;
41
42 // Wewnętrzne rejestry do obliczania liczby pierwszej
43 reg [31:0] A; // Numer N-tej liczby pierwszej do znalezienia
44 reg [31:0] S; // Aktualny status maszyny stanów
45 (0: inicjalizacja, 1: liczenie, 2: obliczone)
46 reg [31:0] W; // Wynikowa liczba pierwsza
47
48 reg [31:0] counter; // Licznik dzielników bieżącej liczby
49
50 reg [31:0] prime_number; // Znaleziona liczba pierwsza
51 reg [31:0] current_number; // Aktualna liczba do sprawdzenia
52 reg [31:0] prime_counter; // Licznik znalezionych liczb pierwszych
53
54 reg is_prime; // Zmienna pomocnicza do sprawdzania pierwszości
55 integer i; // Zmienna pomocnicza
56
57 // Zerowanie zmiennych przy zboczu opadającym sygnału n_reset
58 always @(negedge n_reset) begin
59     A <= 0;
60     S <= 0;
61     W <= 0;
62     counter <= 0;
63     prime_number <= 0;
64     current_number <= 0;
65     prime_counter <= 0;
66
67     gpio_in_s <= 0;
68     gpio_out_s <= 0;
69     sdata_out_s <= 0;
70     state <= 3;
71 end
72
73 // Odczytanie danych przy zboczu narastającym sygnału swr
74 always @(posedge swr) begin
75     if (saddress == 16'h224) begin
76         A <= sdata_in[31:0];
77         S <= 0;
78         W <= 0;
79         state <= 0;
80         current_number <= 2;
81         prime_counter <= 0;
82         prime_number <= 0;
83     end
84 end
85
86 // Wypisanie danych na sdata_out przy zboczu narastającym srd
87 always @(posedge srd) begin
88     case (saddress)
89         16'h234: sdata_out_s <= W[31:0]; // Adres wyniku
90         16'h23C: sdata_out_s <= S[31:0]; // Adres statusu
91         default: sdata_out_s <= 0;
92     endcase
93 end
94
95 // Maszyna stanów do obliczania N-tej liczby pierwszej
96 always @(posedge clk) begin
97     case (state)
98     0: begin // Stan inicjalizacji
99         S <= 1; // Ustawienie statusu na "liczenie"
100         if (A == 1) begin // Jeśli A = 1,
101             bezpośrednio ustaw pierwszą liczbę pierwszą
102             prime_counter <= 1;

```

```

103         prime_number <= 2;
104         state <= 2;
105     end else begin
106         current_number <= 2;
107         prime_counter <= 0;
108         state <= 1;
109     end
110 end
111
112 1: begin // Stan liczenia
113     is_prime = 1;
114     for (i = 2; (i * i <= current_number) && (is_prime == 1); i = i + 1) begin
115         if (current_number % i == 0) begin
116             //Sprawdzenie, czy liczba jest pierwsza
117             is_prime = 0;
118         end
119     end
120     if (is_prime) begin
121         // Aktualizacja licznika i ostatniej znalezionej liczby pierwszej
122         prime_counter <= prime_counter + 1;
123         prime_number <= current_number;
124     end
125     if (A == prime_counter) begin
126         // Sprawdzenie, czy znaleziono N-ta liczbe pierwsza
127         state <= 2;
128     end else begin
129         current_number <= current_number + 1; // Przejdź do następnej liczby
130     end
131 end
132
133 2: begin // Stan zakończenia obliczeń
134     W <= prime_number; // Zapisanie wyniku
135     gpio_out_s <= gpio_out_s + prime_counter; // Aktualizacja wyjścia GPIO
136     S <= 2; // Ustawienie statusu na "obliczone"
137     state <= 3; // Przejście do stanu oczekiwania
138 end
139 endcase
140 end
141
142 assign gpio_out = {16'h0, gpio_out_s[15:0]};
143 assign gpio_in_s_insp = gpio_in_s;
144 assign sdata_out = sdata_out_s;
145
146 endmodule

```

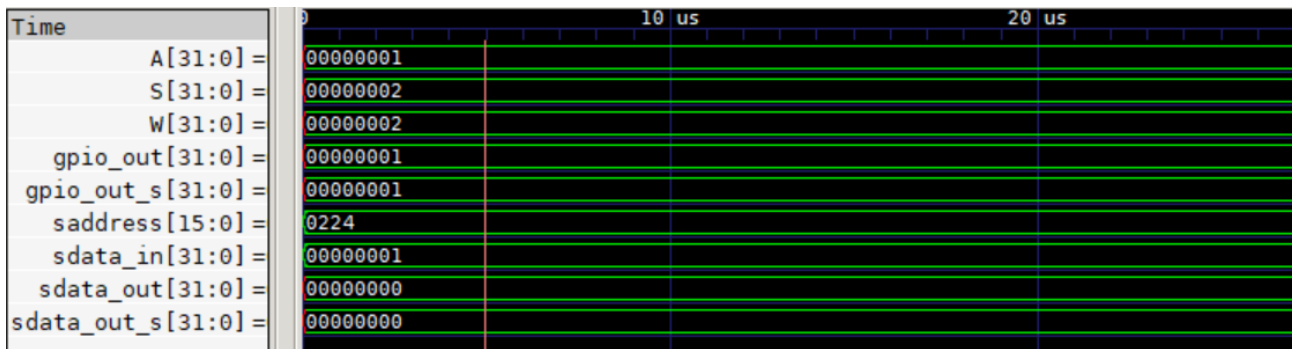
2.3. Przeprowadzone testy

W celu przeprowadzenia testów poprawności działania modułu napisałem testbench, sprawdziłem działanie korzystając z możliwości wizualizacji wyników za pomocą programu gtkwave zintegrowanego ze środowiskiem Iverilog. Poniżej przedstawiam tabelkę, w której zawarte są dane, za pomocą których przetestowałem działanie modułu.

N - dziesiętnie	1	2	7	10	28	100	255	500	750	850	1000
N - heksadecymalnie	0x1	0x2	0x7	0xA	0x1C	0x64	0xFF	0x1F4	0x2EE	0x352	0x3E8
Przewidywany wynik - dziesiętnie	2	3	17	29	107	541	1613	3571	5693	6571	7919
Przewidywany wynik - heksadecymalnie	0x2	0x3	0x11	0x1D	0x6B	0x21D	0x64D	0xDF3	0x163D	0x19AB	0x1EEF

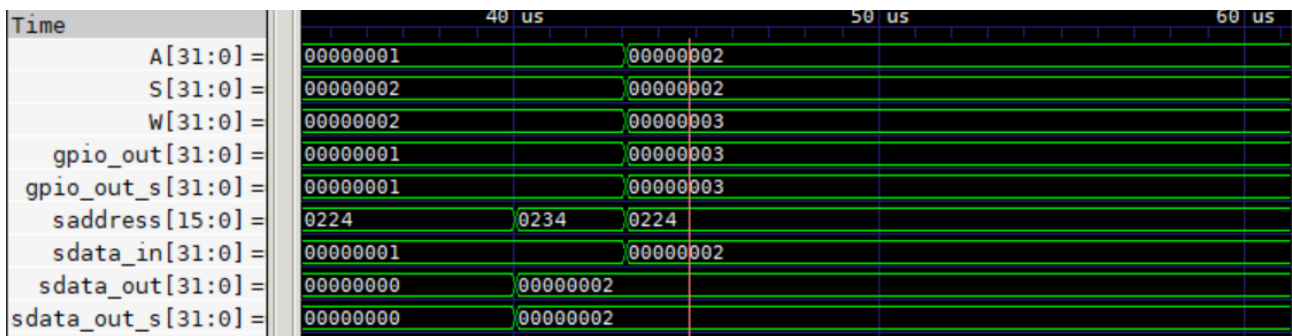
Rysunek 1. Tabela - przewidywane wyniki

1. Dla wartości $A = 1$ otrzymałem wartość $W = 2$, co jest zgodne z oczekiwaniami - wynik symulacji widoczny jest na rysunku 2.



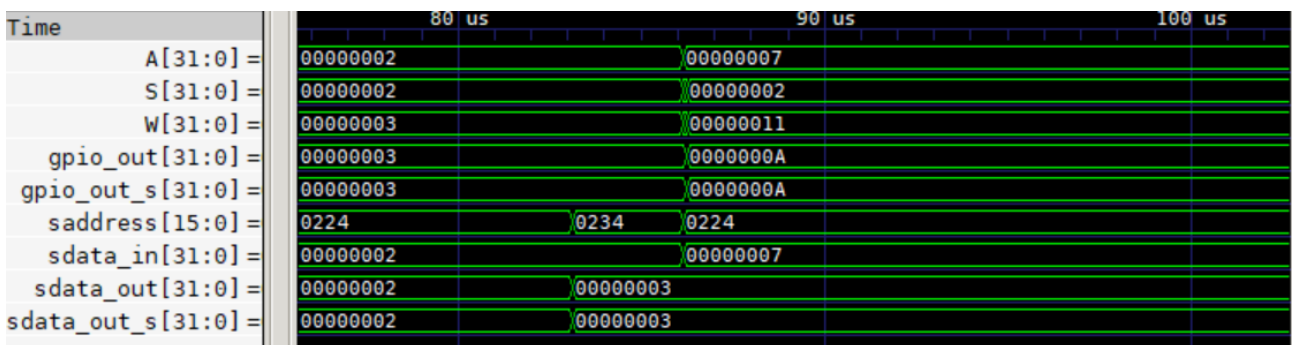
Rysunek 2. Wynik symulacji dla 1. testu

2. Dla wartości $A = 2$ otrzymałem wartość $W = 3$, co jest zgodne z oczekiwaniami - wynik symulacji widoczny jest na rysunku 3.



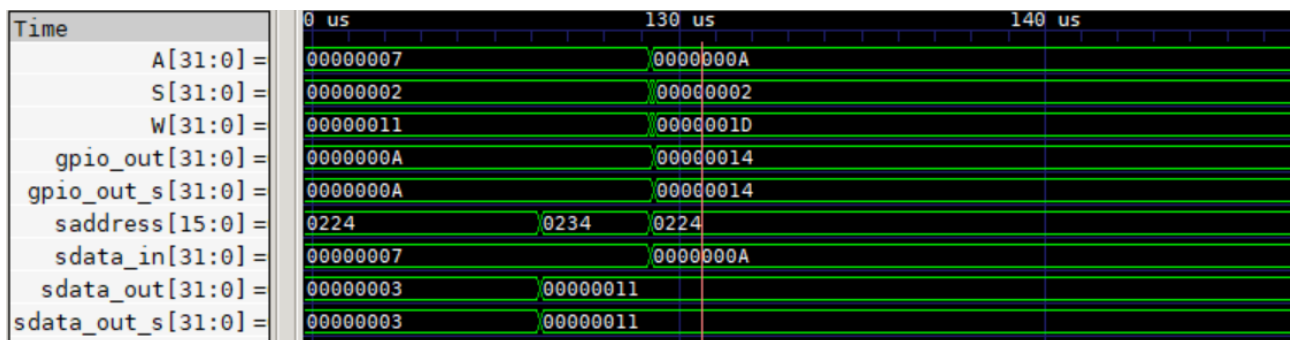
Rysunek 3. Wynik symulacji dla 2. testu

3. Dla wartości $A = 7$ otrzymałem wartość $W = 11$, co jest zgodne z oczekiwaniami - wynik symulacji widoczny jest na rysunku 4.



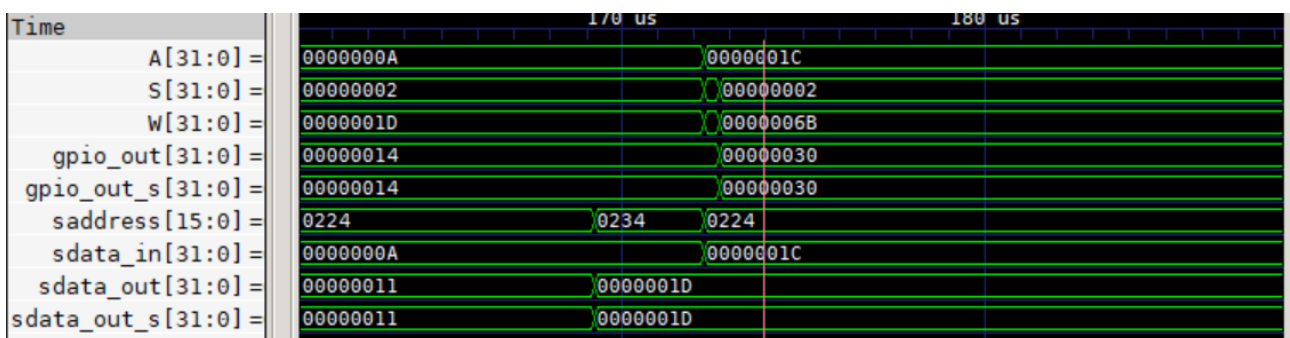
Rysunek 4. Wynik symulacji dla 3. testu

4. Dla wartości $A = A$ otrzymałem wartość $W = 1D$, co jest zgodne z oczekiwaniami - wynik symulacji widoczny jest na rysunku 5.



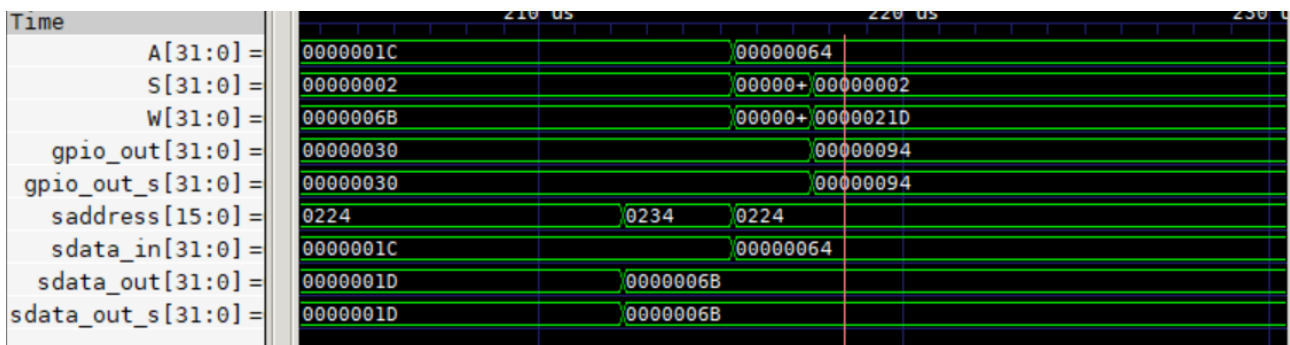
Rysunek 5. Wynik symulacji dla 4. testu

5. Dla wartości $A = 1C$ otrzymałem wartość $W = 6B$, co jest zgodne z oczekiwaniami - wynik symulacji widoczny jest na rysunku 6.



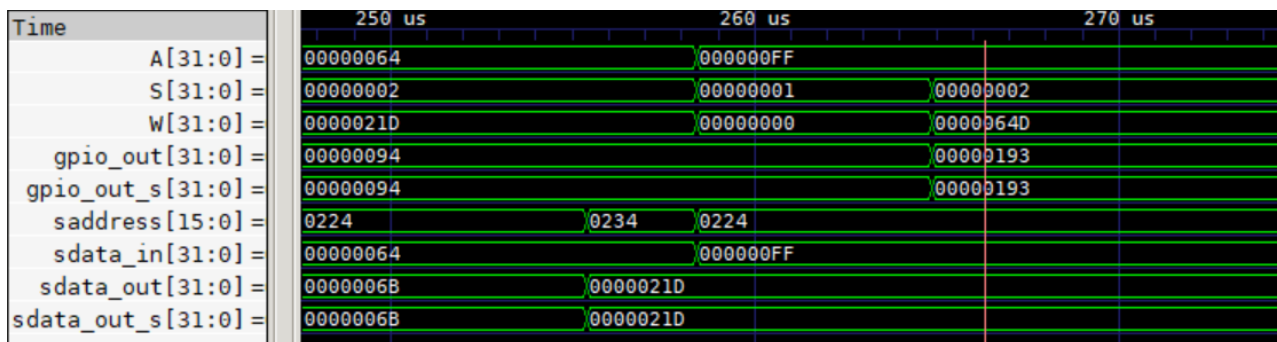
Rysunek 6. Wynik symulacji dla 5. testu

6. Dla wartości $A = 64$ otrzymałem wartość $W = 21D$, co jest zgodne z oczekiwaniami - wynik symulacji widoczny jest na rysunku 7.



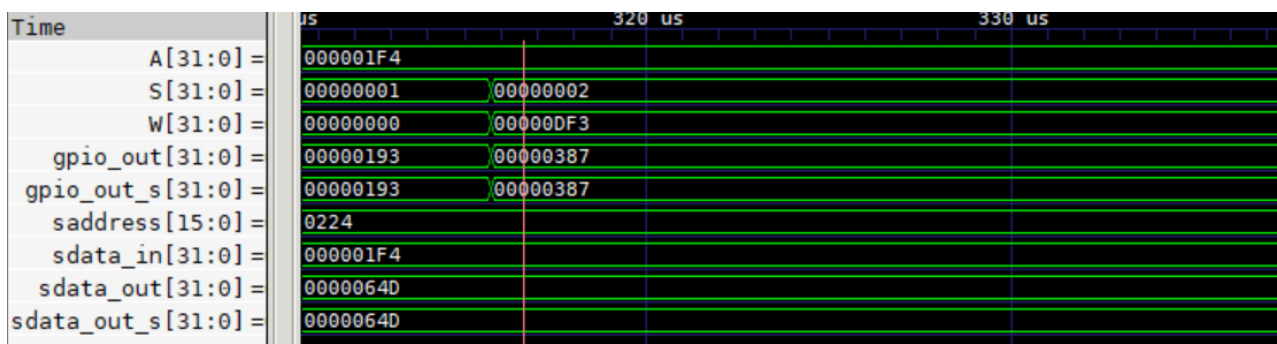
Rysunek 7. Wynik symulacji dla 6. testu

7. Dla wartości $A = FF$ otrzymałem wartość $W = 64D$, co jest zgodne z oczekiwaniami - wynik symulacji widoczny jest na rysunku 8.



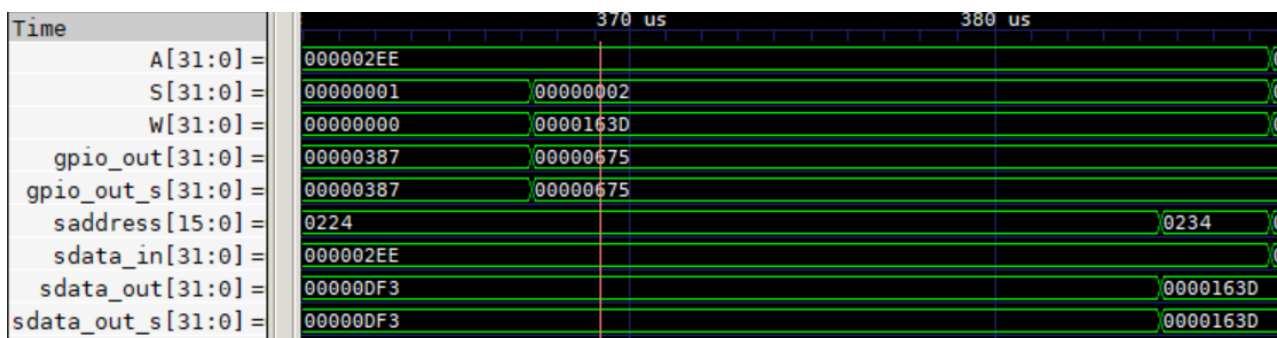
Rysunek 8. Wynik symulacji dla 7. testu

8. Dla wartości A = 1F4 otrzymałem wartość W = DF3, co jest zgodne z oczekiwaniami - wynik symulacji widoczny jest na rysunku 9.



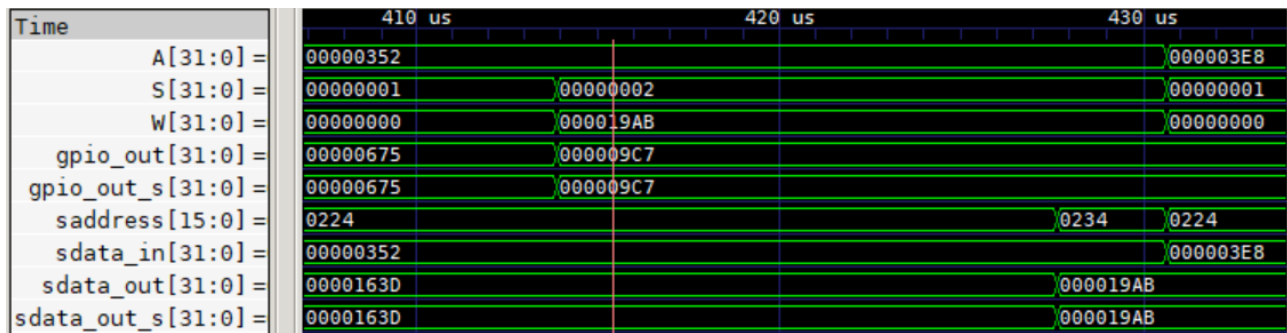
Rysunek 9. Wynik symulacji dla 8. testu

9. Dla wartości A = 2EE otrzymałem wartość W = 163D, co jest zgodne z oczekiwaniami - wynik symulacji widoczny jest na rysunku 10.



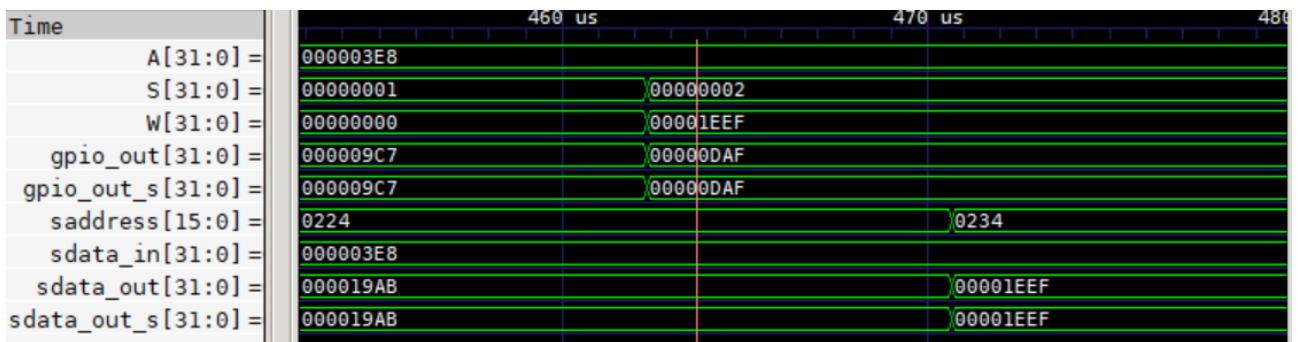
Rysunek 10. Wynik symulacji dla 9. testu

10. Dla wartości A = 352 otrzymałem wartość W = 19AB, co jest zgodne z oczekiwaniami - wynik symulacji widoczny jest na rysunku 11.



Rysunek 11. Wynik symulacji dla 10. testu

11. Dla wartości $A = 3E8$ otrzymałem wartość $W = 1EFF$, co jest zgodne z oczekiwaniami - wynik symulacji widoczny jest na rysunku 12.



Rysunek 12. Wynik symulacji dla 11. testu

Wszystkie wyniki są zgodne z przewidywanymi - zamieszczonymi na Rysunku 1.

2.4. Kod źródłowy testbenchu

Listing 2. GpioEmu.v

```

1  `timescale 1 ns/10 ps
2  module gpioemu_tb;
3
4  // Deklaracja rejestrów i zmiennych
5  reg n_reset = 1;
6  reg [15:0] saddress = 0;
7  reg srd = 0;
8  reg swr = 0;
9  reg [31:0] sdata_in = 0;
10 reg [31:0] gpio_in = 0;
11 reg gpio_latch = 0;
12 reg clk = 0;
13 integer i;
14
15 // Deklaracja wyjść
16 output [31:0] gpio_in_s_insp;
17 output [31:0] gpio_out;
18
19 // Inicjalizacja sygnałów debugowania
20 initial begin
21     $dumpfile("gpioemu.vcd");
22     $dumpvars(0, gpioemu_tb);
23     clk = 0;
24 end
25

```



```

26 // Generowanie sygnału zegarowego
27 initial begin
28     for(i = 0; i < 2500000; i = i + 1)
29         #2 clk = ~clk;
30 end
31
32 // Główna sekwencja testowa
33 initial begin
34     // Resetowanie systemu
35     #5 n_reset = 0;
36     #5 n_reset = 1;
37
38     // Test dla każdej wartości A
39     // Test dla A = 1
40     #5 sdata_in = 32'd1;
41     #5 saddress = 16'h224;
42     #5 swr = 1;
43     #5 swr = 0;
44     #40000;
45     #30 saddress = 16'h234;
46     #5 srd = 1;
47     #5 srd = 0;
48     #3000;
49
50     // Test dla A = 2
51     #5 sdata_in = 32'd2;
52     #5 saddress = 16'h224;
53     #5 swr = 1;
54     #5 swr = 0;
55     #40000;
56     #30 saddress = 16'h234;
57     #5 srd = 1;
58     #5 srd = 0;
59     #3000;
60
61     // Test dla A = 7
62     #5 sdata_in = 32'd7;
63     #5 saddress = 16'h224;
64     #5 swr = 1;
65     #5 swr = 0;
66     #40000;
67     #30 saddress = 16'h234;
68     #5 srd = 1;
69     #5 srd = 0;
70     #3000;
71
72     // Test dla A = 10
73     #5 sdata_in = 32'd10;
74     #5 saddress = 16'h224;
75     #5 swr = 1;
76     #5 swr = 0;
77     #40000;
78     #30 saddress = 16'h234;
79     #5 srd = 1;
80     #5 srd = 0;
81     #3000;
82
83     // Test dla A = 28
84     #5 sdata_in = 32'd28;
85     #5 saddress = 16'h224;
86     #5 swr = 1;
87     #5 swr = 0;
88     #40000;
89     #30 saddress = 16'h234;
90     #5 srd = 1;

```

```

91     #5 srd = 0;
92     #3000;
93
94     // Test dla A = 100
95     #5 sdata_in = 32'd100;
96     #5 saddress = 16'h224;
97     #5 swr = 1;
98     #5 swr = 0;
99     #40000;
100    #30 saddress = 16'h234;
101    #5 srd = 1;
102    #5 srd = 0;
103    #3000;
104
105    // Test dla A = 255
106    #5 sdata_in = 32'd255;
107    #5 saddress = 16'h224;
108    #5 swr = 1;
109    #5 swr = 0;
110    #40000;
111    #30 saddress = 16'h234;
112    #5 srd = 1;
113    #5 srd = 0;
114    #3000;
115
116    // Test dla A = 500
117    #5 sdata_in = 32'd500;
118    #5 saddress = 16'h224;
119    #5 swr = 1;
120    #5 swr = 0;
121    #40000;
122    #30 saddress = 16'h234;
123    #5 srd = 1;
124    #5 srd = 0;
125    #3000;
126
127    // Test dla A = 750
128    #5 sdata_in = 32'd750;
129    #5 saddress = 16'h224;
130    #5 swr = 1;
131    #5 swr = 0;
132    #40000;
133    #30 saddress = 16'h234;
134    #5 srd = 1;
135    #5 srd = 0;
136    #3000;
137
138    // Test dla A = 850
139    #5 sdata_in = 32'd850;
140    #5 saddress = 16'h224;
141    #5 swr = 1;
142    #5 swr = 0;
143    #40000;
144    #30 saddress = 16'h234;
145    #5 srd = 1;
146    #5 srd = 0;
147    #3000;
148
149    // Test dla A = 1000
150    #5 sdata_in = 32'd1000;
151    #5 saddress = 16'h224;
152    #5 swr = 1;
153    #5 swr = 0;
154    #40000;
155    #30 saddress = 16'h234;

```

```

156     #5 srd = 1;
157     #5 srd = 0;
158     #20000;
159
160     $finish;
161 end
162
163 // Deklaracja sygnałów wyjściowych
164 wire [31:0] gpio_out_test;
165 wire [31:0] sdata_out_test;
166 wire [31:0] gpio_in_s_insp_test;
167 wire [63:0] is_prime;
168 wire [63:0] comparison_value;
169
170 // Instancja testowanego modułu
171 gpioemu test(n_reset, saddress, srd, swr, sdata_in, sdata_out_test, gpio_in,
172 gpio_latch, gpio_out, clk, gpio_in_s_insp);
173
174 endmodule

```

3. Moduł jądra systemu Linux

Moduł obsługuje odczyt i zapis danych przez pliki w procfs, które następnie przekazuje do odpowiednich rejestrów w przestrzeni GPIO. Pliki te są tworzone w funkcji `my_init_module`, a usuwane w funkcji `my_cleanup_module`. Poniżej przedstawiam kod modułu jądra systemu Linux:

Listing 3. GpioEmu.v

```

1 #include <linux/module.h>
2 #include <linux/kernel.h>
3 #include <linux/ioport.h>
4 #include <linux/proc_fs.h>
5 #include <linux/uaccess.h>
6 #include <asm/io.h>
7
8 // Informacje o module
9 MODULE_INFO(intree, "Y");
10 MODULE_LICENSE("GPL");
11 MODULE_AUTHOR("Jakub Strzelczyk");
12 MODULE_DESCRIPTION("Simple kernel module for SYKOM project");
13 MODULE_VERSION("0.01");
14
15 // Definicje konfiguracyjne dla zarządzania przestrzenią GPIO
16 i procesami kończącymi moduł
17 #define SYKT_GPIO_BASE_ADDR (0x00100000)
18 #define SYKT_GPIO_SIZE (0x8000)
19 #define SYKT_EXIT (0x3333)
20 #define SYKT_EXIT_CODE (0x7F)
21
22 // Definicje adresów dla różnych rejestrów w przestrzeni GPIO,
23 wykorzystują bazowy adres
24 void __iomem *baseptr;
25 #define SYKT_GPIO_ADDR_SPACE (baseptr)
26 #define REG_A (SYKT_GPIO_ADDR_SPACE + 0x224)
27 #define REG_S (SYKT_GPIO_ADDR_SPACE + 0x23C)
28 #define REG_W (SYKT_GPIO_ADDR_SPACE + 0x234)
29
30 // Deklaracja zmiennych
31 static struct proc_dir_entry *proc_dir;
32 static unsigned int regAValue, regSValue, regWValue;
33
34 // Funkcje zapisu wartości do rejestrów
35 static ssize_t regA_write(struct file *file, const char __user *buffer,

```

```

36 size_t count, loff_t *pos)
37 {
38     char buf[13];
39     if (copy_from_user(buf, buffer, min(count, sizeof(buf) - 1)))
40         return -EFAULT;
41     buf[min(count, sizeof(buf) - 1)] = '\0';
42     if (kstrtouint(buf, 8, &regAValue))
43         return -EINVAL;
44     writel(regAValue, REG_A);
45     return count;
46 }
47
48 static ssize_t regS_write(struct file *file, const char __user *buffer,
49 size_t count, loff_t *pos)
50 {
51     char buf[13];
52     if (copy_from_user(buf, buffer, min(count, sizeof(buf) - 1)))
53         return -EFAULT;
54     buf[min(count, sizeof(buf) - 1)] = '\0';
55     if (kstrtouint(buf, 8, &regSValue))
56         return -EINVAL;
57     writel(regSValue, REG_S);
58     return count;
59 }
60
61 static ssize_t regW_write(struct file *file, const char __user *buffer,
62 size_t count, loff_t *pos)
63 {
64     char buf[13];
65     if (copy_from_user(buf, buffer, min(count, sizeof(buf) - 1)))
66         return -EFAULT;
67     buf[min(count, sizeof(buf) - 1)] = '\0';
68     if (kstrtouint(buf, 8, &regWValue))
69         return -EINVAL;
70     writel(regWValue, REG_W);
71     return count;
72 }
73
74 // Funkcje odczytu wartosci z rejestrow
75 static ssize_t regA_read(struct file *file, char __user *buffer,
76 size_t count, loff_t *pos)
77 {
78     char buf[13];
79     regAValue = readl(REG_A);
80     int len = snprintf(buf, sizeof(buf), "%o\n", regAValue);
81     if (*pos >= len)
82         return 0;
83     if (copy_to_user(buffer, buf, len))
84         return -EFAULT;
85     *pos += len;
86     return len;
87 }
88
89 static ssize_t regS_read(struct file *file, char __user *buffer,
90 size_t count, loff_t *pos)
91 {
92     char buf[13];
93     regSValue = readl(REG_S);
94     int len = snprintf(buf, sizeof(buf), "%o\n", regSValue);
95     if (*pos >= len)
96         return 0;
97     if (copy_to_user(buffer, buf, len))
98         return -EFAULT;
99     *pos += len;
100    return len;

```

```

101 }
102
103 static ssize_t regW_read(struct file *file, char __user *buffer,
104 size_t count, loff_t *pos)
105 {
106     char buf[13];
107     regWValue = readl(REG_W);
108     int len = snprintf(buf, sizeof(buf), "%o\n", regWValue);
109     if (*pos >= len)
110         return 0;
111     if (copy_to_user(buffer, buf, len))
112         return -EFAULT;
113     *pos += len;
114     return len;
115 }
116
117 //Definicje interfejsow plikow w /proc
118 static const struct file_operations proc_regA_ops = {
119     .owner = THIS_MODULE,
120     .read = regA_read,
121     .write = regA_write,
122 };
123
124 static const struct file_operations proc_regS_ops = {
125     .owner = THIS_MODULE,
126     .read = regS_read,
127     .write = regS_write,
128 };
129
130 static const struct file_operations proc_regW_ops = {
131     .owner = THIS_MODULE,
132     .read = regW_read,
133     .write = regW_write,
134 };
135
136 //Inicjalizacja modulu
137 static int __init my_init_module(void)
138 {
139     printk(KERN_INFO "Init my module.\n");
140     baseptr = ioremap(SYKT_GPIO_BASE_ADDR, SYKT_GPIO_SIZE);
141     if (!baseptr) {
142         printk(KERN_ERR "Error while mapping base address.\n");
143         return -ENOMEM;
144     }
145
146     proc_dir = proc_mkdir("sykom", NULL);
147     if (!proc_dir) {
148         printk(KERN_ERR "Error while creating /proc/sykom directory.\n");
149         iounmap(baseptr);
150         return -ENOMEM;
151     }
152
153     if (!proc_create("rejstrjakA", 0644, proc_dir, &proc_regA_ops) ||
154         !proc_create("rejstrjakS", 0644, proc_dir, &proc_regS_ops) ||
155         !proc_create("rejstrjakW", 0444, proc_dir, &proc_regW_ops)) {
156         printk(KERN_ERR "Cannot create /proc/sykom entries.\n");
157         remove_proc_entry("rejstrjakA", proc_dir);
158         remove_proc_entry("rejstrjakS", proc_dir);
159         remove_proc_entry("rejstrjakW", proc_dir);
160         remove_proc_entry("sykom", NULL);
161         iounmap(baseptr);
162         return -ENOMEM;
163     }
164
165     return 0;

```

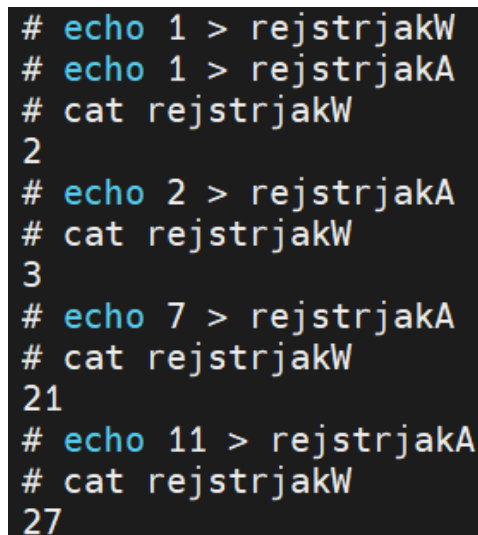
```

166 }
167
168 // Czyszczenie modulu
169 static void __exit my_cleanup_module(void)
170 {
171     printk(KERN_INFO "Cleanup my module.\n");
172     writel(SYKT_EXIT | ((SYKT_EXIT_CODE) << 16), baseptr);
173     remove_proc_entry("rejstrjakA", proc_dir);
174     remove_proc_entry("rejstrjakS", proc_dir);
175     remove_proc_entry("rejstrjakW", proc_dir);
176     remove_proc_entry("sykom", NULL);
177     iounmap(baseptr);
178 }
179
180 module_init(my_init_module);
181 module_exit(my_cleanup_module);

```

3.1. Przeprowadzone testy

Przeprowadziłem testy sprawdzające poprawność działania jądra systemu Linux. W tym celu skorzystałem z następujących komend:



```

# echo 1 > rejstrjakW
# echo 1 > rejstrjakA
# cat rejstrjakW
2
# echo 2 > rejstrjakA
# cat rejstrjakW
3
# echo 7 > rejstrjakA
# cat rejstrjakW
21
# echo 11 > rejstrjakA
# cat rejstrjakW
27

```

Rysunek 13. Wyniki przeprowadzonych testów

Uzyskane wyniki są poprawne, zgodne z oczekiwaniami. Gdy zamienimy 2, 3, 21 i 27 z notacji oktalnej na dziesiętną, wówczas otrzymamy 2, 3, 17 i 31.

4. Aplikacja testująca w języku C

4.1. Krótki opis

Aplikacja w języku C testuje moduł systemowy poprzez komunikację z plikami specjalnymi w katalogu /proc. Definiuje ścieżki do plików rejestrów i używa funkcji do odczytu (read_from_file) i zapisu (write_to_file) wartości oktalnych. Funkcja calculate_prime zapisuje wartość a do rejestru A, oczekuje na zakończenie obliczeń przez moduł, a następnie odczytuje wynik z rejestru W. test_module porównuje wyniki obliczeń z oczekiwanymi wartościami dla różnych a, wyświetlając rezultaty testów na konsoli. Funkcja main uruchamia test_module, aby zweryfikować działanie modułu.

4.2. Kod źródłowy

Poniżej przedstawiam kod źródłowy aplikacji testującej:

Listing 4. GpioEmu.v

```

1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <sys/stat.h>
4 #include <fcntl.h>
5 #include <unistd.h>
6 #include <errno.h>
7 #include <stdlib.h>
8 #include <string.h>
9
10 #define MAX_BUFFER 1024
11 #define REG_A "/proc/sykom/rejstrjakA" // Sciezka do rejestru A
12 #define REG_W "/proc/sykom/rejstrjakW" // Sciezka do rejestru W
13 #define REG_S "/proc/sykom/rejstrjakS" // Sciezka do rejestru S
14
15 // Funkcja do odczytu wartosci z pliku
16 unsigned int read_from_file(const char *);
17 int write_to_file(const char *, unsigned int);
18 void test_module();
19 unsigned int calculate_prime(unsigned int a);
20
21 int main(void)
22 {
23     test_module();
24     return 0;
25 }
26
27 // Funkcja do odczytu wartosci z pliku
28 unsigned int read_from_file(const char *filePath) {
29     FILE *file = fopen(filePath, "r");
30     if (file == NULL) {
31         printf("Error opening file: %d\n", errno);
32         exit(EXIT_FAILURE);
33     }
34
35     char buffer[MAX_BUFFER];
36     size_t n = fread(buffer, 1, MAX_BUFFER - 1, file);
37     if (n == 0) {
38         if (feof(file)) {
39             printf("End of file reached.\n");
40         } else {
41             printf("Error reading from file\n");
42             fclose(file);
43             exit(EXIT_FAILURE);
44         }
45     }
46     buffer[n] = '\0';
47     fclose(file);
48
49     unsigned int result;
50     if (sscanf(buffer, "%o", &result) != 1) { // %o dla formatu oktalnego
51         printf("Error parsing file content\n");
52         exit(EXIT_FAILURE);
53     }
54
55     return result;
56 }
57
58 // Funkcja do zapisu wartosci do pliku
59 int write_to_file(const char *filePath, unsigned int input) {
60     FILE *file = fopen(filePath, "w");
61     if (file == NULL) {
62         printf("Error opening file: %d\n", errno);
63         exit(EXIT_FAILURE);

```

```

64     }
65
66     if (fprintf(file, "%o", input) < 0) { // %o dla formatu oktalnego
67         printf("Error writing to file\n");
68         fclose(file);
69         exit(EXIT_FAILURE);
70     }
71     fclose(file);
72     return 0;
73 }
74
75 // Funkcja do obliczenia liczby pierwszej na podstawie wartosci a
76 unsigned int calculate_prime(unsigned int a) {
77     write_to_file(REG_A, a); // Zapisujemy wartosc a do rejestru A
78     unsigned int status;
79     unsigned int result;
80
81     // Petla oczekujaca na zakonczenie obliczen
82     while (1) {
83         status = read_from_file(REG_S); // Odczyt statusu z rejestru S
84         if (status == 2) { // Jesli status wskazuje zakonczenie obliczen
85             result = read_from_file(REG_W); // Odczyt wyniku z rejestru W
86             return result;
87         }
88         usleep(10000); // Opoznienie 10ms
89     }
90     return 0;
91 }
92
93 // Funkcja testujaca modul
94 void test_module() {
95     unsigned int a[] = {1, 2, 7, 10, 28, 100, 255, 500, 750, 850, 1000};
96     // Testowane wartosci a
97     unsigned int results[] = {02, 03, 021, 035, 0153, 01035, 03135, 07043,
98     013501, 015435, 017537}; // Oczekiwane wyniki w formacie oktalnym
99     unsigned int wynik;
100
101     // Petla wykonujaca testy dla
102     kazdej wartosci a
103     for (int i = 0; i < sizeof(a) / sizeof(a[0]); i++) {
104         wynik = calculate_prime(a[i]);
105         // Obliczenie liczby pierwszej dla danej wartosci a
106         if (wynik == results[i]) {
107             printf("TEST PASSED for N=%u: expected=%o, got=%o\n", a[i],
108             results[i], wynik); // Sukces testu
109         } else {
110             printf("TEST FAILED for N=%u: expected=%o, got=%o\n", a[i],
111             results[i], wynik); // Niepowodzenie testu
112         }
113     }
114     printf("Tests completed.\n");
115 }

```

4.3. Przeprowadzone testy

Aby weryfikować poprawność działania projektu wykonałem następujące testy przy użyciu aplikacji:

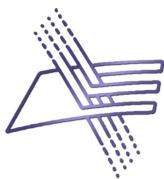

```
Welcome to Buildroot
buildroot login: root
# modprobe kernel_module
[ 10.398131] Init my module.
# ./main
TEST PASSED for N=1: expected=2, got=2
TEST PASSED for N=2: expected=3, got=3
TEST PASSED for N=7: expected=21, got=21
TEST PASSED for N=10: expected=35, got=35
TEST PASSED for N=28: expected=153, got=153
TEST PASSED for N=30: expected=161, got=161
TEST PASSED for N=40: expected=255, got=255
TEST PASSED for N=50: expected=345, got=345
TEST PASSED for N=55: expected=401, got=401
TEST PASSED for N=70: expected=535, got=535
TEST PASSED for N=88: expected=711, got=711
TEST PASSED for N=100: expected=1035, got=1035
Tests completed.
```

Rysunek 14. Wyniki przeprowadzonych testów

Wszystkie przeprowadzone testy zakończyły się powodzeniem, co oznacza, że aplikacja działa prawidłowo.

5. Podsumowanie

Projekt zakończył się sukcesem, a wszystkie jego elementy, od modułu Verilog, przez moduł jądra systemu Linux, aż po aplikację testującą, działały zgodnie z oczekiwaniami, co potwierdzają przeprowadzone testy.



**Wydział Elektroniki
i Technik Informatycznych**

POLITECHNIKA WARSZAWSKA