Projekt SYKOM - raport

Wojciech Królak-Kurkus
indeks 331499

4 maja 2025

1 Specyfikacja zadania

Zadaniem projektowym było utworzenie modułu w języku Verilog (razem z testbenchem), którego zadaniem jest obliczanie sumy kontrolnej CRC32/ISCSI [1], następnie skompilowanie go przy użyciu danych narzędzi i uruchomienie go w QEMU. Aby można było obsłużyć moduł 'sprzętowy' należało utworzyć moduł jądra Linux oraz program testowy sprawdzający poprawność działania całego utworzonego systemu. Pełna, szczegółowa, treść zadania znajduje się w katalogu `/projekt`. Moduł wykorzystuje sygnał zegara i sygnał resetu, który asynchronicznie przywraca wszystkie wartości

2 Moduł Verilog

2.1 Założenia działania modułu

Moduł ma dwa rejestry wejściowe i dwa rejestry wyjściowe. Obsługiwane one są szyną adresu i szyną danych. W Tabeli 1 przedstawione są szczegóły dotyczące rejestrów.

Offset	Ilość bitów	We/wy	Nazwa rejestru	Przeznaczenie
0x640	8	Wejście	`IN`	Rejestr do wprowadzania danych do obliczania CRC
0x658	2	Wejście	`CTRL`	Rejestr do wydawania poleceń modułowi
0x648	3	Wyjście	`STATE`	Rejestr do pokazywania aktualnego statusu modułu
0x650	32	Wyjście	`RESULT`	Rejestr na wynikową sumę kontrolną

Tabela 1: Specyfikacje rejestrów

2.1.1 Instrukcje modułu

Użytkownik ma do dyspozycji trzy instrukcje (Tabela 2) (wewnętrznie istnieje jeszcze czwarta, ale jest to twór wewnętrzny symbolizujący aktualny brak instrukcji)

2.1.2 Moduł jako automat

Moduł, zgodnie z treścią zadania powinien działać na zasadzie automatu stanów. Założone stany automatu i ich opisy przestawione są w Tabeli 3.
Domyślnym stanem, niekonwencjonalnie, jest stan nr 1, tj. `STATE_READ`. Oznacza on, że moduł jest gotowy do przyjęcia kolejnego bajtu danych wejściowych.
`STATE_FULL` sygnalizuje, że bufor danych został zapełniony. Póki bufor nie jest zapełniony, informacja, na którym dokładnie bajcie obecnie się znajdujemy, nie jest udostępniana.

Nr instrukcji	Nazwa instrukcji	Opis instrukcji
0	`brak instrukcji`	
1	`PUT`	Kopiuje zawartość rejestru `IN` w odpowiednie miejsce w buforze
2	`GET`	Instrukcja sygnalizująca, że moduł ma zacząć liczyć sumę kontrolną
3	`CLR`	Czyści bufor (Użytkownik sygnalizuje, że chce zacząć wpisywać dane od nowa)

Tabela 2: Instrukcje modułu

Nr stanu	Nazwa stanu	Opis stanu
0	`STATE_BUSY`	Moduł jest zajęty - oblicza sumę kontrolną
1	`STATE_READ`	Moduł jest gotowy do odczytu danych
2	`STATE_FULL`	Bufor danych wejściowych modułu jest pełny - nie wpisuj kolejnych, bo wyrzucę błąd
3	`STATE_READY`	Moduł obliczył sumę kontrolną i jest gotowy do podania jej na wyjście po otrzymaniu odpowiedniej instrukcji
4	`STATE_ERROR`	Stan błędu - wejście jest zablokowane, można tylko zresetować moduł

Tabela 3: Specyfikacje stanów

Więcej wyjaśnienia wymaga stan błędu `STATE_ERROR`. Błąd wyrzucany jest w dwóch przypadkach, acz motywacja jest jedna - chcemy mieć pewność, że *wszystkie* dane wprowadzone przez użytkownika będą składały się na wejście algorytmu. Szyna danych ma 32 bity, ale rejestr `IN` przyjmuje tylko ośmiobitowe wartości, więc w przypadku, gdy na szynie podana zostanie wartość większa od 255, dane są odrzucane i sygnalizowany jest błąd. To samo dzieje się w przypadku, gdy bufor wejściowy jest pełny, a podany zostanie kolejny bajt danych.
Pozostałe stany są trywialne do zrozumienia.

Moduł działa na zasadzie automatu sterowanego zewnętrznymi, zmiennymi w czasie parametrami. Poglądowy schemat automatu przedstawiony jest na Ilustracji 1. Przeskoki ze stanu na stan dzieją się z taktem zegara. Czytając schemat, należy mieć w pamięci to, że domyślnie brak instrukcji oznacza pozostanie na tym samym stanie.

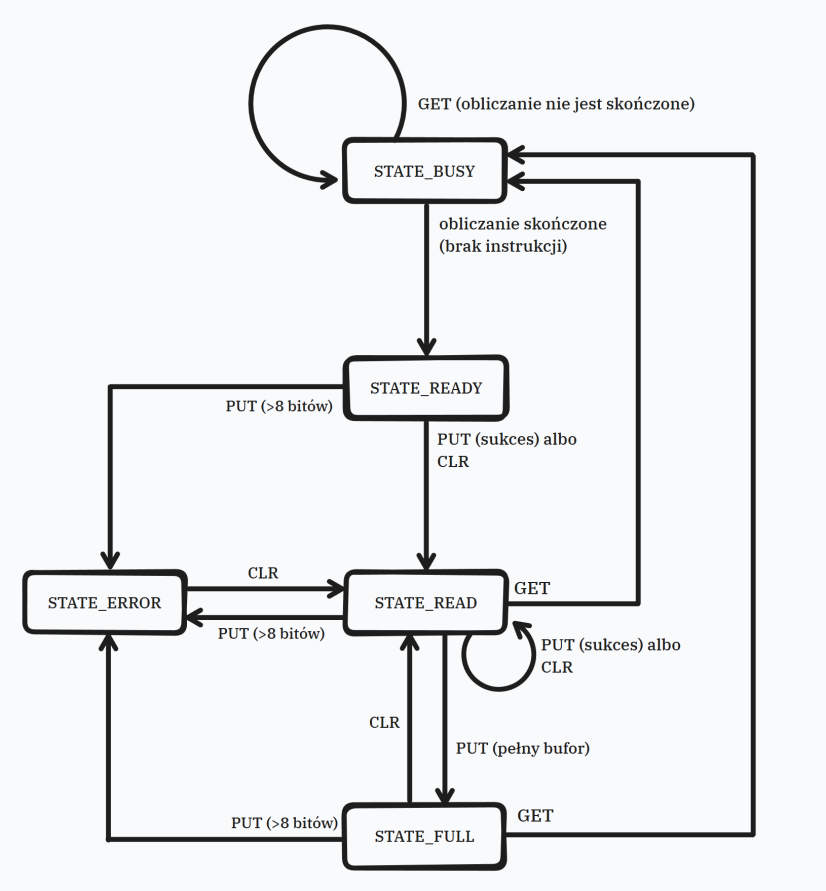
2.1.3 Sposób obliczania sumy kontrolnej

Suma kontrolna CRC32ISCSI (znana również jako CRC32C, CRC32Castagnoli, CRC32NVME)[1] jest obliczana przez moduł poprzez XORowanie danych z wielomianem charakterystycznym odmianie CRC32ISCSI, przesuwanym po każej iteracji w stronę najmłodszego bitu do najbliższej jedynki pozostałej w buforze. Sposób bardzo ładnie jest opisany w [2]. Specyfikacja CRC32ISCSI wymaga jednak paru dodatkowych kroków:

1. Odwracanie kolejności bitów każdego bajtu danych wejściowych
2. XOR najstarszych 32 bitów z `0xffffffff` przed rozpoczęciem właściwego obliczania sumy
3. Po dojściu do końca obliczeń, wynikowa suma kontrolna jest dodatkowo XORowana z `0xffffffff`, a kolejność *wszystkich 32* bitów jest odwracana.

2.2 Testowanie modułu

Aby upewnić się, że moduł działa poprawnie, utworzyłem testbench. Do testów, które nie wypisują wyniku w konsoli dołączyłem przebiegi sygnałów i zawartości rejestrów. Testy sprawdzają następujące

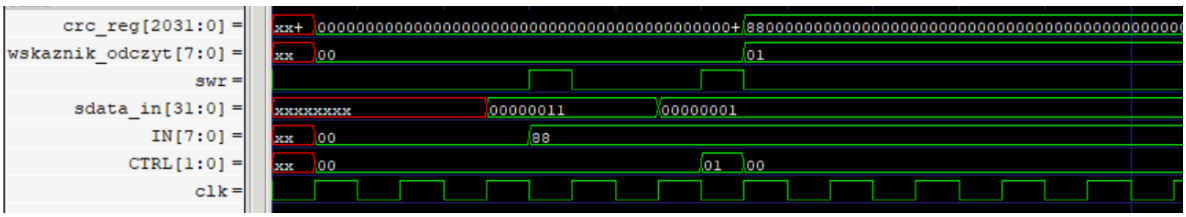


Rysunek 1: Schemat automatu

scenariusze:

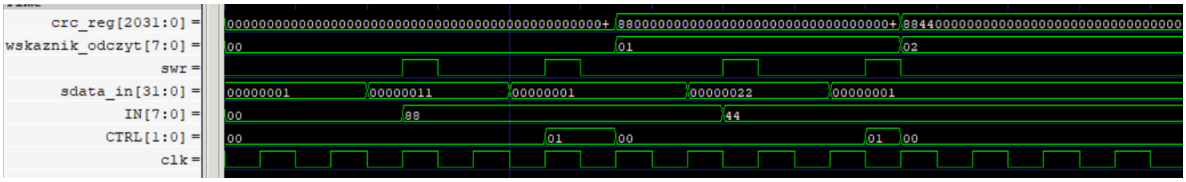
1. **Test instrukcji `PUT` po resecie**

   (a) Wprowadzenie jednego bajtu `0x11`



   Rejestr `crc_reg` dostaje od razu bajty z odwróconymi kolejnościami bitów, dlatego po wprowadzeniu `0x11 (00010001)`, do rejestru zapisywane jest `0x88 (10001000)`.

   (b) Wprowadzenie dwóch bajtów `0x1122`

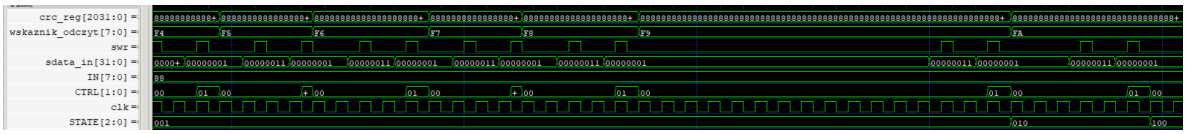


   Rejestr `wskaznik_odczyt` zwiększa z każdym nowym bajtem swoją wartość o 1,

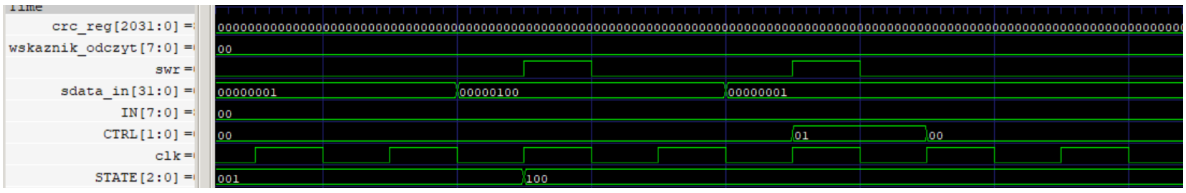
   (c) Wprowadzenie 249 bajtów

(d) Wprowadzenie 250 bajtów (stan powinien stać się `STATE_FULL 0x3`)

(e) Wprowadzenie 251 bajtów (stan powinien zmienić się na `STATE_ERROR 0x4`)



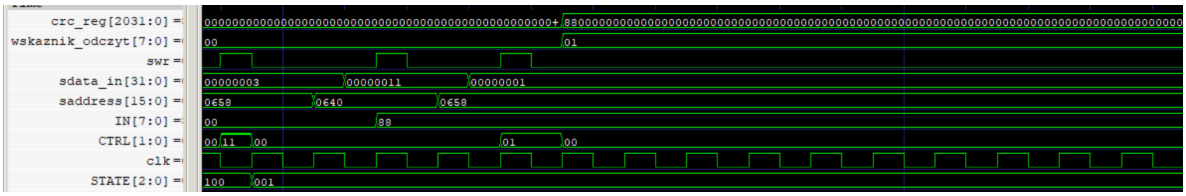
(f) Wprowadzenie wartości dziewięciobitowej `0x100` (256_{dec}) (stan powinien zmienić się na `STATE_ERROR 0x4`)



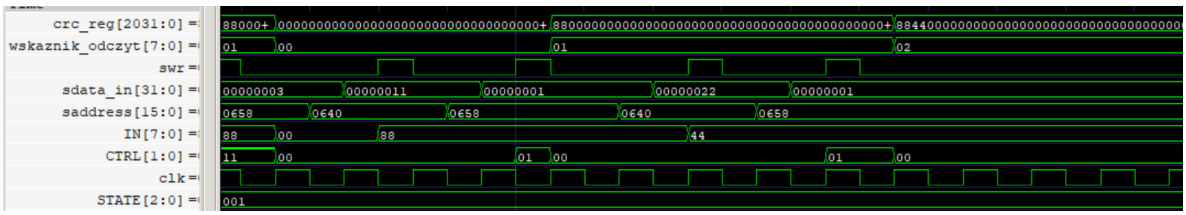
Błąd wyrzucany jest już w momencie, gdy na szynie danych znajdzie się za długa wiadomość. Dla żadnej sytuacji nie jest to stan prawidłowy, więc moduł nie czeka z ustawieniem stanu błędu na instrukcję `PUT`.

2. **Test instrukcji `PUT` po `CLR`**

(a) Wprowadzenie jednego bajtu `0x11`



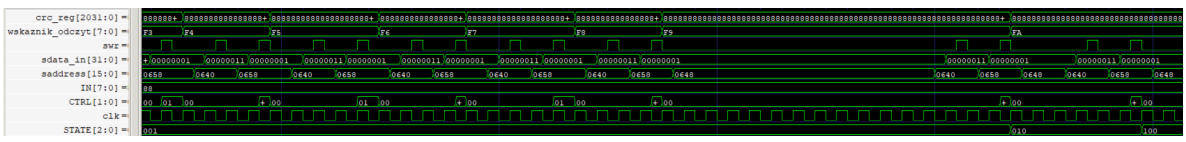
(b) Wprowadzenie dwóch bajtów `0x1122`



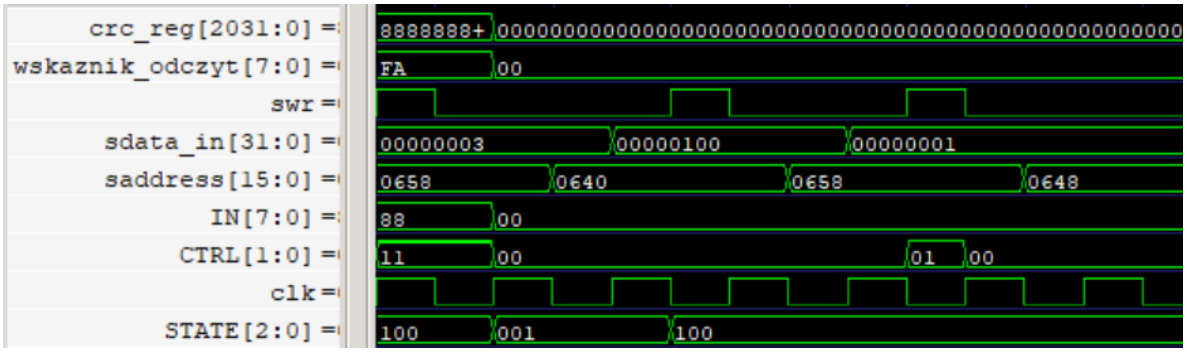
(c) Wprowadzenie 249 bajtów

(d) Wprowadzenie 250 bajtów (stan powinien stać się `STATE_FULL`)

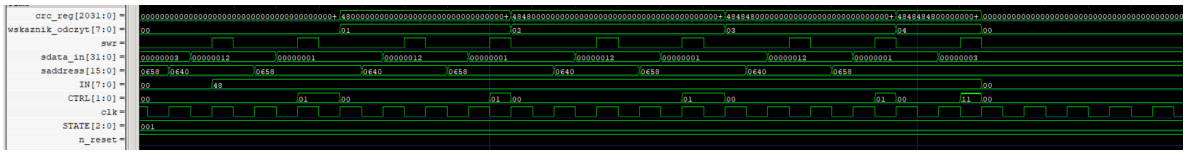
(e) Wprowadzenie 251 bajtów (stan powinien zmienić się na `STATE_ERROR`)



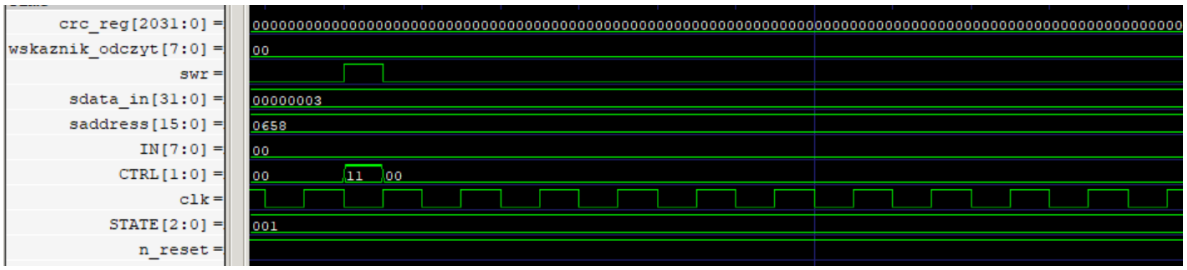
(f) Wprowadzenie wartości dziewięciobitowej `0x100` (256_{dec}) (stan powinien zmienić się na `STATE_ERROR`)



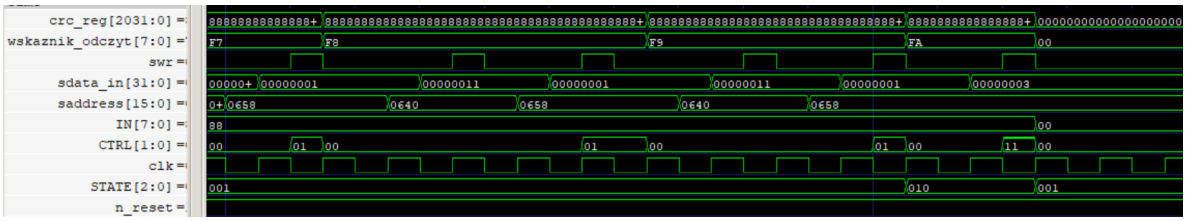
3. **Poprawność obliczania sum CRC32ISCSI:**
   (a) Obliczenie sumy kontrolnej ciągu `0xACDC`
   (b) Obliczenie sumy kontrolnej ciągu `0x53180080`
   (c) Obliczenie sumy kontrolnej ciągu `0x00` (jeden bajt)
   (d) Obliczenie sumy kontrolnej ciągu `0x00000000` (cztery bajty)
   (e) Obliczenie sumy kontrolnej z pustego bufora
   (f) Obliczenie sumy kontrolnej z bufora pełnego danych
   (g) Obliczenie sumy kontrolnej z bufora pełnego zer (250 bajtów zerowych)
4. **Test instrukcji** `CLR` (powinien czyścić się bufor wejściowy i stan ustawić się na `STATE_READ`)
   (a) Podczas wpisywania danych



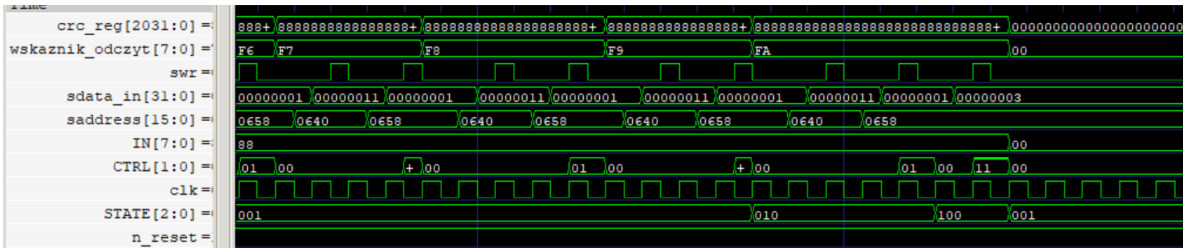
   (b) Przy pustym buforze



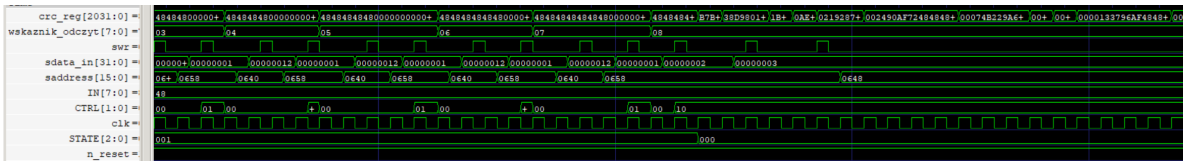
   (c) Przy pełnym buforze



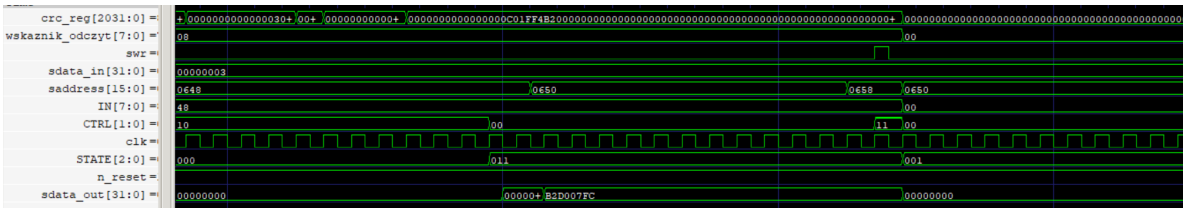
(d) Przy błędzie



(e) Podczas obliczania (powinno nic się nie stać)

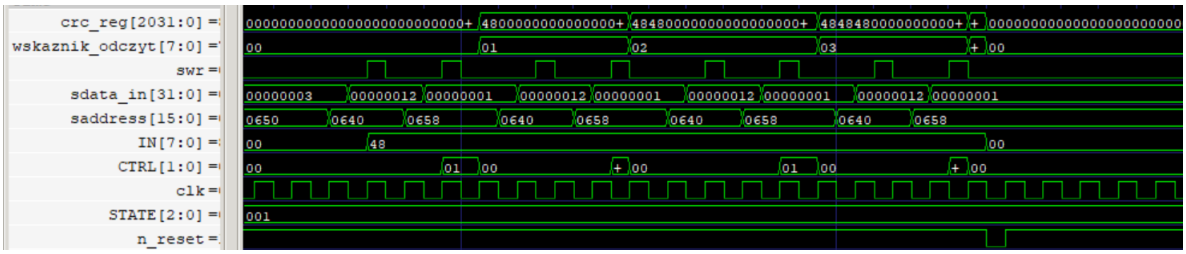


(f) Po obliczeniu sumy kontrolnej

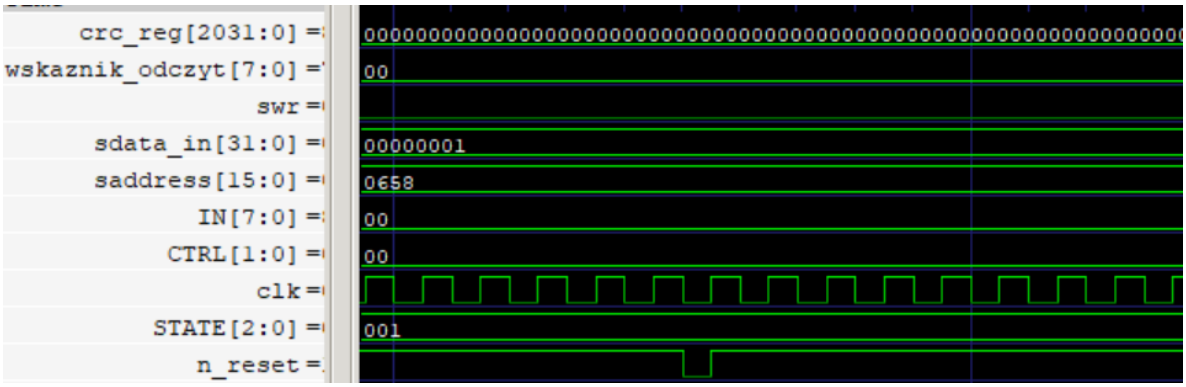


5. **Test sygnału resetu** (powinny czyścić się wszystkie rejestry i stan ustawić się na `STATE_READ`)

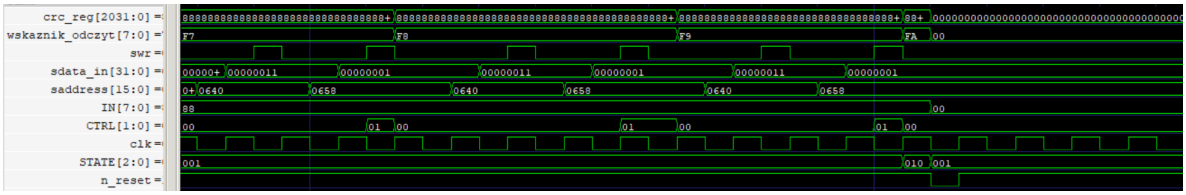
   (a) Podczas wpisywania danych



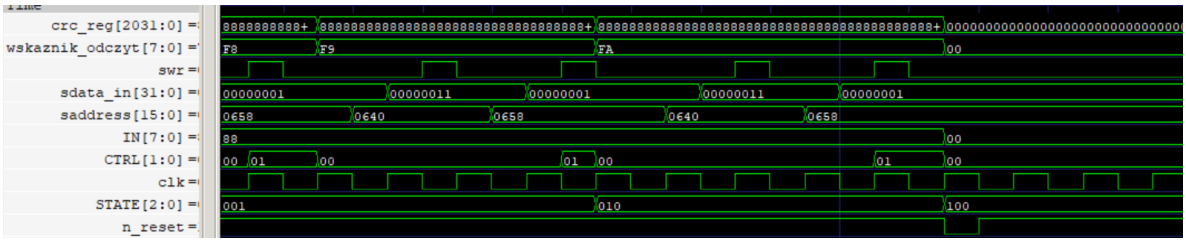
   (b) Przy pustym buforze



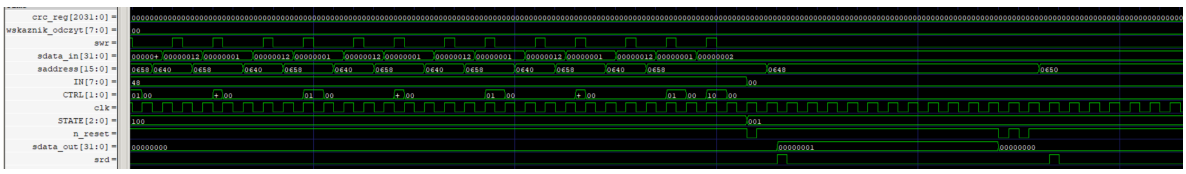
   (c) Przy pełnym buforze



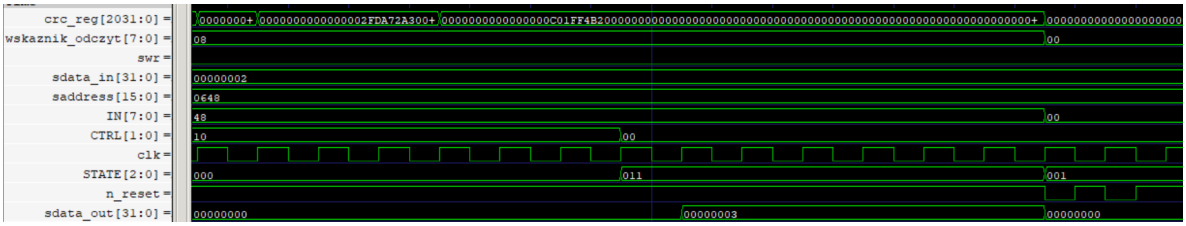
   (d) Przy błędzie



   (e) Podczas obliczania (suma powinna przestać się liczyć)

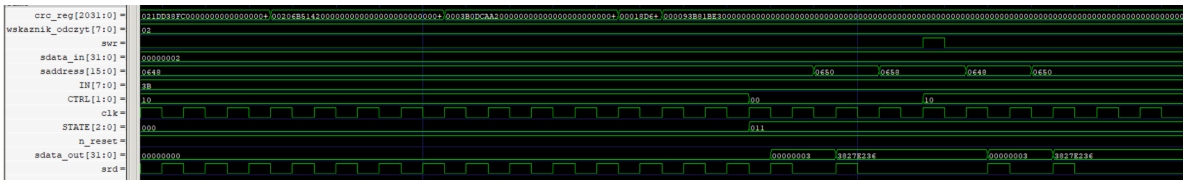


   (f) Po obliczeniu sumy kontrolnej



6. **Test innych ważnych scenariuszy**

   (a) Instrukcja `GET` przy stanie `STATE_READY` (Powinna nic nie robić - na wyjściu wciąż jest wynik)



Dane zwracane przez testbench dla testów automatycznych niewymagających analizy przebiegów:

```
Test 1. c)
STATE: 1
Oczekiwany: 1

Test 1. d)
STATE: 2
Oczekiwany: 2

Test 1. e)
STATE: 4
Oczekiwany: 4

Test 1. f)
STATE: 4
Oczekiwany: 4

Test 2. c)
STATE: 1
Oczekiwany: 1

Test 2. d)
STATE: 2
Oczekiwany: 2

Test 2. e)
STATE: 4
Oczekiwany: 4

Test 2. f)
STATE: 4
Oczekiwany: 4

Test 3. a)
Wynik: 3827e236
Oczekiwane 3827e236
```

```
Test 3. b)
Wynik: a2825413
Oczekiwane a2825413

Test 3. c)
Wynik: 527d5351
Oczekiwane 527d5351

Test 3. d)
Wynik: 48674bc7
Oczekiwane 48674bc7

Test 3. e)
Wynik: 0
Oczekiwane 00000000

Test 3. f)
Wynik: 3c96196e
Oczekiwane 3c96196e

Test 3. g)
Wynik: a51552b8
Oczekiwane a51552b8

Test 4. e)
Wynik: b2d007fc
Oczekiwane b2d007fc

Test 4. f)
Wynik: 0
Oczekiwane 00000000

Test 5. e)
STATE: 1
Oczekiwany: 1

Test 6. a)
Wynik: 3827e236
Oczekiwane 3827e236
Wynik: 3827e236
Oczekiwane 3827e236
```

3 Moduł jądra linux

Na temat modułu jądra nie ma co się rozpisywać - obsługuje zapis do `dskrwo` i `dtkrwo` oraz odczyt z `dckrwo` i `drkrwo`. Wykorzystuje on system plików `sysfs` i montuje ww. pliki w katalogu `/sys/kernel/sykom/`. Moduł sprawdza poprawność wprowadzanych do plików danych i w przypadku niepoprawności, wyrzuca błąd zapisu.

4 Aplikacja testująca

Aplikacja testująca zapisuje do i czyta z utworzonych przez `kernel_module` plików. W ten sposób wysyła dane i instrukcje do modułu 'sprzętowego'. Najpierw automatycznie oblicza przy użyciu modułu predefiniowane wektory testowe i porównuje wyniki z prawidłowymi wynikami, a potem daje użytkownikowi możliwość wpisania własnego stringa do wyliczenia jego CRC.

```
# ./main
Compiled at May  4 2025 18:39:36
Obliczono poprawnie sume '': 00000000
Czas obliczania sumy: 61 ms.

Obliczono poprawnie sume 'test': 86A072C0
Czas obliczania sumy: 484 ms.

Obliczono poprawnie sume 'sykom': 5D293ED5
Czas obliczania sumy: 595 ms.

Obliczono poprawnie sume 'wojciech': B0CBCA92
Czas obliczania sumy: 905 ms.

Obliczono poprawnie sume 'sykomsykomsykomsykomsykomsykomsykomsykomsykomsykomsyk
omsykomsykomsykomsykomsykomsykomsykomsykomsykomsykomsykomsykomsykomsykomsy
komsykomsykomsykomsykomsykomsykomsykomsykomsykomsykomsykomsykomsykomsykoms
ykomsykomsykomsykomsykomsykomsykom': C8ECA170
Czas obliczania sumy: 28959 ms.

Stan modulu po wpisaniu 251 bajtow: 00000004
Wpisz tekst, ktorego CRC32ISCSI chcesz obliczyc (aby wyjsc, wcisnij enter bez w
pisywania niczego): testtest
CRC32ISCSI ciagu znakow 'testtest' to: 46557539
Wpisz tekst, ktorego CRC32ISCSI chcesz obliczyc (aby wyjsc, wcisnij enter bez w
pisywania niczego): kocham sykom
CRC32ISCSI ciagu znakow 'kocham sykom' to: D13D6A84
Wpisz tekst, ktorego CRC32ISCSI chcesz obliczyc (aby wyjsc, wcisnij enter bez w
pisywania niczego): wojciech
CRC32ISCSI ciagu znakow 'wojciech' to: B0CBCA92
Wpisz tekst, ktorego CRC32ISCSI chcesz obliczyc (aby wyjsc, wcisnij enter bez w
pisywania niczego):
```

Dane przekazywane są, zgodnie z treścią zadania, w formacie oktalnym. Wynik ostatniej sumy, która dała wynik `0xB0CBCA92`, w formacie oktalnym wynosi `0260313312222`, co dowodzi oktalnej naturze zapisu przekazywanych w plikach sysfs danych.

```
# cat drkrwo
26062745222#
```

Rozwiązanie nie jest jednak idealne, gdyż istnieje szansa na niepoprawne wprowadzenie danych przez program, spowodowane zastosowanym mechanizmem wprowadzania danych. Ciągły odczyt i zapis plików, zwłaszcza z dużą częstotliwością, jest mało optymalny - lepiej działałoby na przykład wprowadzenie do pliku `sysfs`owego całego bloku danych do przetworzenia, przy `kernel_module` wysyłającym

bajt po bajcie do 'sprzętu'. Przy dużych odstępach między zapisami do pliku, czas wprowadzania i obliczania znacząco rośnie (zależność niby jest liniowa, ale 250 bajtów trwa aż pół minuty).

Literatura

[1] https://reveng.sourceforge.io/crc-catalogue/17plus.htm#crc.cat-bits.32.

[2] https://en.wikipedia.org/wiki/Cyclic_redundancy_check#Computation.