|  |  |
| --- | --- |
|  | 🞂Wykrywanie błędów transmisji  Raport dotyczący zebranych informacji |
|  |  |
|  | Maciej Nowak 4T3 Sebastian Łuczak 4T3  Prowadzący: mgr inż. Tomasz Wojciechowski |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Spis treści

[1. Informacje ogólne 2](#_Toc232659382)

[2. Wykrywanie błędów – metoda CRC 2](#_Toc232659383)

[3. Opis pakietu 2](#_Toc232659384)

[4. Opis raportu 3](#_Toc232659385)

[5. Schemat ideowy 4](#_Toc232659386)

[6. Implementacyjny schemat układu i połączeń 5](#_Toc232659387)

[us.vhd 6](#_Toc232659388)

[buforin.vhd 6](#_Toc232659389)

[flowcontrol.vhd 6](#_Toc232659390)

[crccalc.vhd 6](#_Toc232659391)

[comparator.vhd 7](#_Toc232659392)

[buforout.vhd 7](#_Toc232659393)

[7. Opis interfejsu 7](#_Toc232659394)

[Wejścia 7](#_Toc232659395)

[Wyjścia 7](#_Toc232659396)

[8. Sterowanie układem 8](#_Toc232659397)

[Opis działania automatu main\_fsm 8](#_Toc232659398)

[Opis działania automatu flow\_fsm 9](#_Toc232659399)

[Opis działania automatu proc\_fsm 9](#_Toc232659400)

[Opis działania automatu calc\_fsm 9](#_Toc232659401)

[Opis działania automatu bufout\_fsm 9](#_Toc232659402)

[9. Opis procedury testowej 10](#_Toc232659403)

[10. Wyniki syntezy 10](#_Toc232659404)

[11. Podsumowanie 11](#_Toc232659405)

[Propozycje ulepszenia/optymalizacji 11](#_Toc232659406)

[12. Literatura i źródła 11](#_Toc232659408)

Wykrywanie błędów transmisji

Raport dotyczący zebranych informacji

# Informacje ogólne

Projekt dotyczy wykrywania błędów transmisji za pomocą algorytmu CRC (ang. Cyclic Redundancy Check) w implementacji 16-bitowej.   
Przy transmisji danych między płytką laboratoryjną a komputerem posługiwać się będziemy własnym protokołem transmisji.

# Wykrywanie błędów – metoda CRC

Algorytm CRC 16bit jest stosowany do wyznaczania matematycznych sum kontrolnych dla dowolnych danych wejściowych. Jest to n-bitowa liczba całkowita określająca poprawność danych ze wzorcem. CRC stosowane jest do zabezpieczania przed błędami plików i przesyłanych danych np. w Internecie.   
CRC używane w naszym projekcie jest resztą z binarnego dzielenia ciągu danych przez wielomian x16+x15+x2+1 o długości 16 bitów.

Wielomian ten zapisywany jest jako liczba całkowita w następujący sposób:

* jeśli współczynnik przy *x* w *k*-tej potędze jest równy 1, wówczas ustawiamy bit *k*-ty na 1
* jeśli współczynnik przy *x* w *k*-tej potędze jest równy 0, wówczas ustawiamy bit *k*-ty na 0

x16+x15+x2+1 -> 10000000000001012Obliczając wartość CRC posłużyliśmy się pakietem zewnętrznym wygenerowanym na stronie [[1]](#footnote-1). Pakiet ten realizuje równoległą generację wartości CRC co daje oszczędność 16 cykli zegara, gdyż nie trzeba oczekiwać na ustalenie się wartości w rejestrze wyjściowym.

# Opis pakietu

* 1. Porządek Bitów i Bajtów

Bity są przesyłane od najmniej znaczącego do najbardziej znaczącego.

* 1. Pola

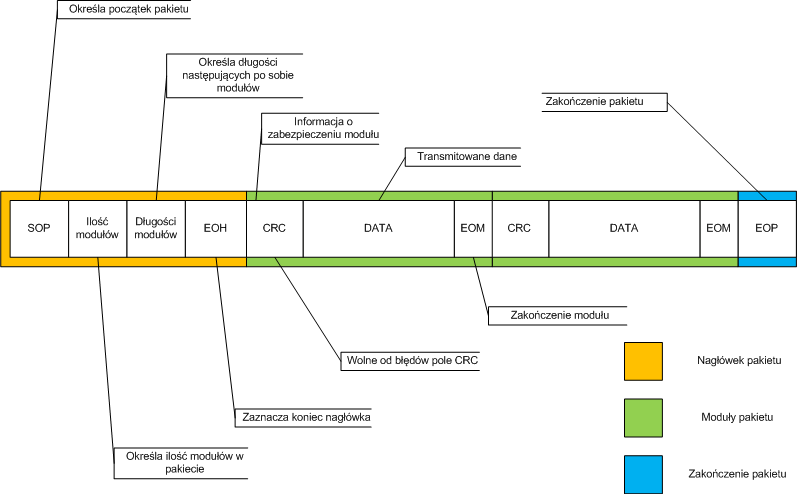
Opisane na schemacie

* 1. Oznaczenia pól kluczowych

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nazwa pola: | SOP | EOH | EOM | EOP |
| Ciąg bitów: | 00000010 | 00000110 | 00000011 | 00000100 |

* 1. Długości poszczególnych pól pakietu:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Nazwa pola: | SOP | Liczba  Modułów | Długość  Modułów | EOH | CRC | DATA | EOM | EOP |
| Długość | 2x8bit | 8bit | 4x16bit | 8bit | 16bit | 1KB | 2x8bit | 8bit |



# Opis raportu

* 1. Porządek Bitów i Bajtów

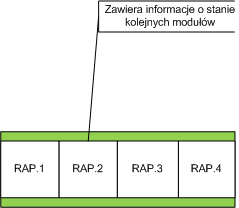
Bity są przesyłane od najmniej znaczącego do najbardziej znaczącego.

* 1. Pola:

Opisane na schemacie

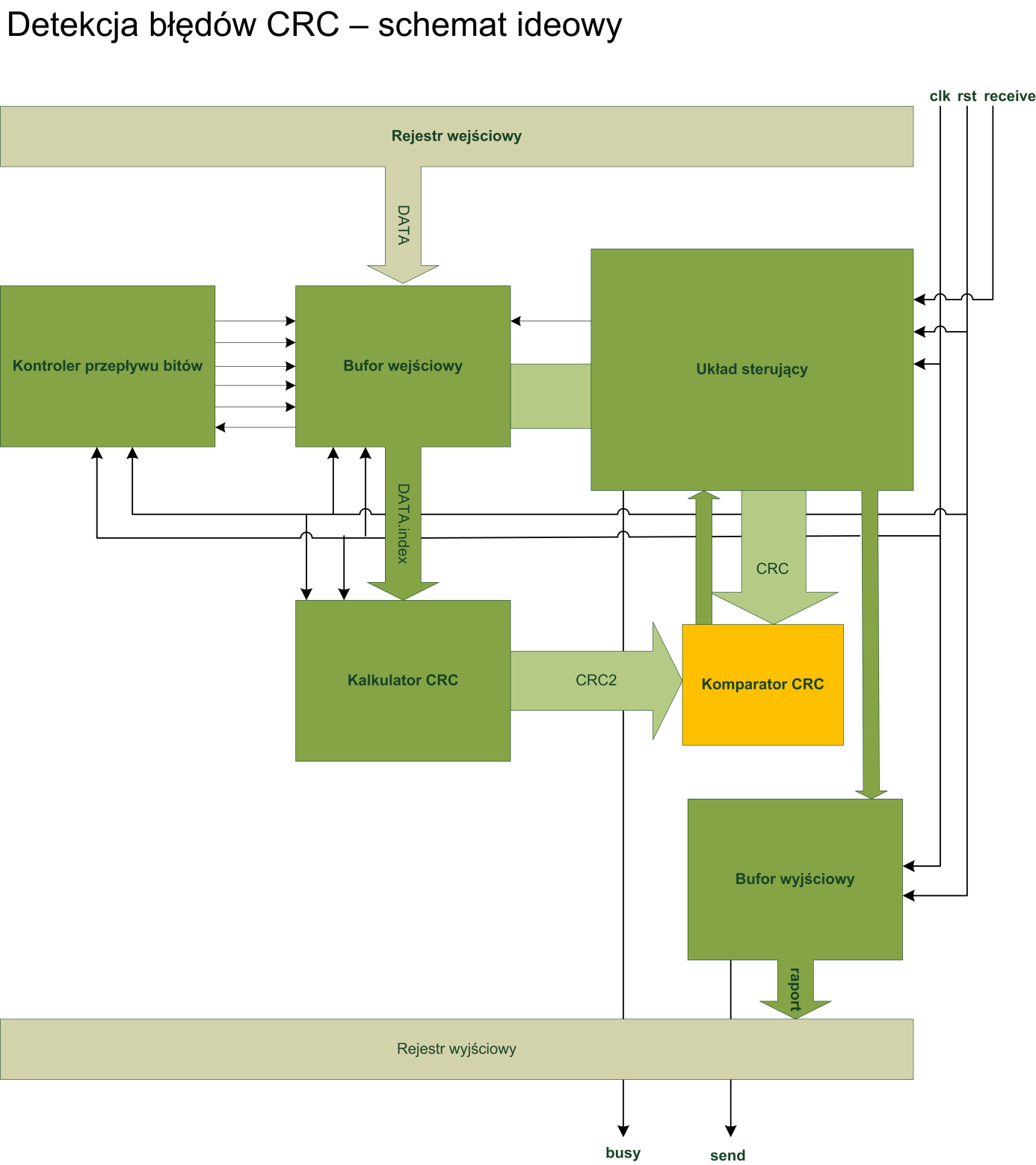
* 1. Długości poszczególnych pól raportu:

|  |  |
| --- | --- |
| Nazwa pola: | RAP |
| Długość | 4x2bit |

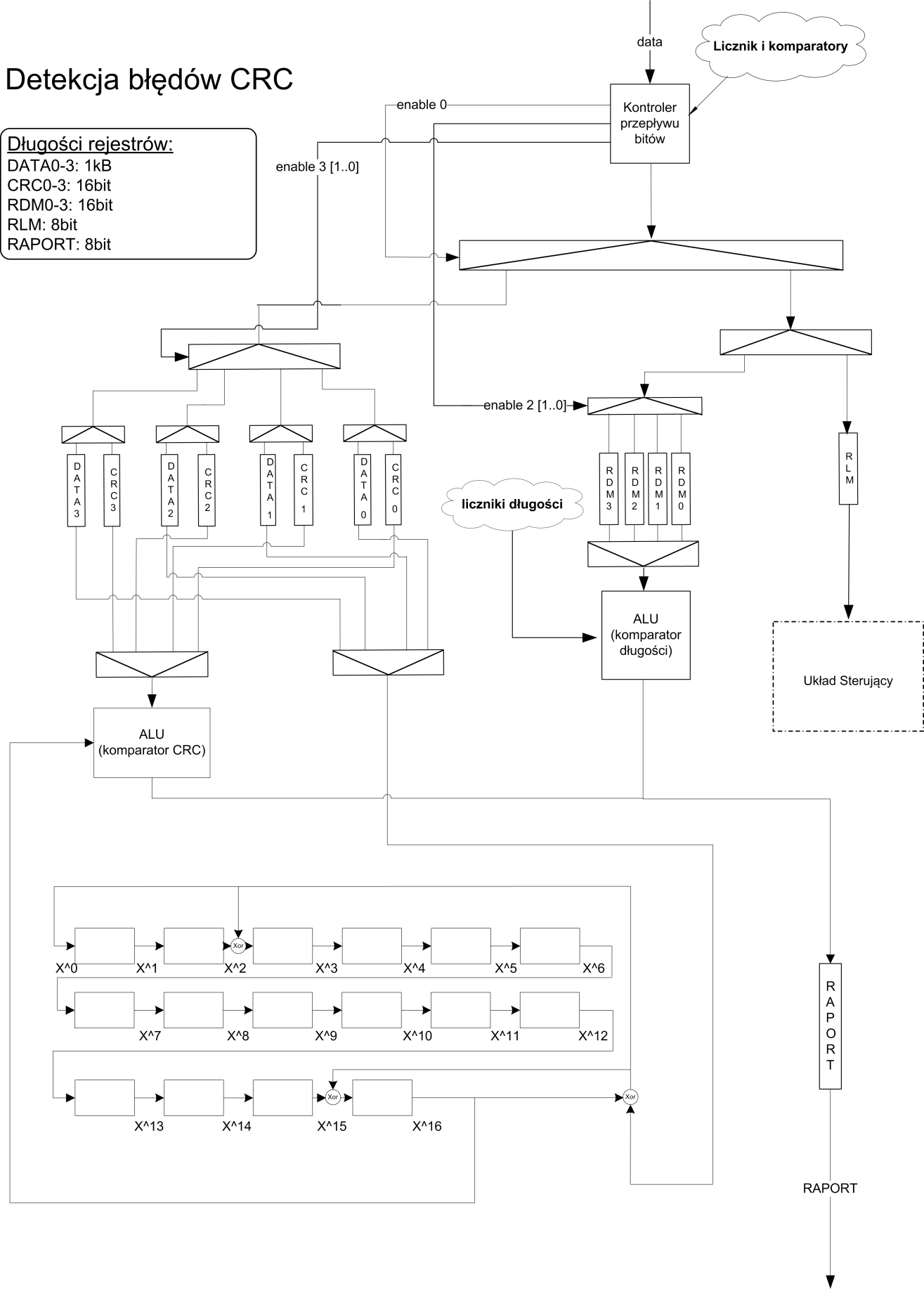




# Schemat ideowy



# Implementacyjny schemat układu i połączeń



Dane podawane na szynie 8 bitowej docierają do rejestru odpowiadającego za ich synchronizację z cyklem zegara. Po opuszczeniu tego rejestru blok kontroli przepływu danych zaczyna rozmieszczać dane w odpowiadających im rejestrach i pamięci RAM. Równolegle z odbiorem danych po otrzymaniu pierwszego modułu bez błędu długości, układ zaczyna go przetwarzać. Na ten proces składa się obliczenie CRC, porównanie wyniku z CRC przesłanym w module i przekazanie statusu modułu do bufora wyjściowego.

Opisy poszczególnych komponentów przedstawione są poniżej.

## us.vhd

Zadania:

Układ sterujący odpowiedzialny jest za sterowanie działaniem całego układu. Działanie automatów jest opisane poniżej (8. Sterowanie układem). W zależności od ilości modułów, ich zgodności z deklarowaną długością i poprawności co do CRC skaluje on zakres działania całego układu.

## buforin.vhd

Zadania:

Moduł odpowiedzialny za prawidłowe rozpropagowanie i zapamiętanie danych wejściowych. Zawiera odrębne rejestry na wartości z nagłówka pakietu, CRC, a także pamięć RAM na dane pochodzące z modułów. Odrębny moduł flowcontrol.vhd zarządza procesem zapisu danych.

### flowcontrol.vhd

Zadania:

Blok kontroli przepływu bitów kontroluje punkty charakterystyczne pakietu. W zależności od danych wejściowych ( miejsca w transmitowanym module ) wysyła sygnały *enable* i s*elect* do odpowiednich demultiplekserów, multiplekserów i rejestrów, oraz sygnały *wren* i *adresy* do pamięci RAM. Kontroluje zgodność modułów co do długości, a także informuje układ sterujący o kończeniu transmisji kolejnych modułów.

## crccalc.vhd

Zadania:

Moduł ten liczy wartość CRC w oparciu o zewnętrzny pakiet wygenerowany na stronie i. Posiada własny automat, który zarządza operacjami niezbędnymi do uzyskania wartości CRC dla konkretnego modułu.

## comparator.vhd

Zadania:

Element czysto kombinacyjny. Porównuje wartość CRC daną i obliczoną. Wyodrębniony jedynie dla przejrzystości.

## buforout.vhd

Zadania:

Przyjmuje i przechowuje dane o stanie modułów oraz wysyła je po skompletowaniu wszystkich.

# Opis interfejsu

Poniższa tabela przedstawia wejścia i wyjścia układu:

|  |  |  |  |
| --- | --- | --- | --- |
| Wejścia i wyjścia układu | | | |
| Nazwa | Typ | Rozmiar | Krótki opis |
| clk | IN | 1 bit | Wejście zegara |
| rst | IN | 1 bit | Reset asynchroniczny |
| receive | IN | 1 bit | Rozpoczęcie przyjmowania danych |
| data | IN | 8 bit | Dane wejściowe |
| busy | OUT | 1 bit | Informacja o stanie zajętości układu |
| send | OUT | 1 bit | Informacja o rozpoczęciu wysyłania raportu |
| raport | OUT | 8 bit | Raport o stanie danych wejściowych |

## Wejścia

* **clk ­­­­­­­**–wejście zegara
* **rst** – reset asynchroniczny
* **receive** – sygnał sterujący pracą układu, rozpoczyna przyjmowanie danych
* **data** – ośmiobitowa szyna doprowadzająca dane do układu

## Wyjścia

* **busy** – sygnał informujący o stanie zajętości układu. Stan wysoki oznacza, że układ nie jest gotowy na przyjęcie następnych danych
* **send** – sygnał informujący o rozpoczęciu wysyłania raportu. Raport po pojawieniu się tego sygnału jest zerowany, a cały układ przechodzi w stan oczekiwania
* **raport** – szyna wyprowadzająca z układu raport o stanie modułów w kolejności od pierwszego do ostatniego  
   *kodowanie: 11 - moduł bezbłędny*

*10 - błąd transmisji (niezgodne crc)*

*01 - błąd długości modułu przesłanego*

*00 - moduł nieprzetworzony*

# Sterowanie układem

Układ sterowany jest przez pięć automatów skończonych (FSM) o różnym przeznaczeniu.  
Ich grafy dołączone są do dokumentacji.

Automat główny (**main\_fsm**) odpowiada za sterowanie pracą całego układu. Jest on taktowany zegarem podłączonym do wejścia **clk**. Wejście układu **receive** steruje rozpoczęciem pracy układu. Stan wysoki tego wejścia występujący podczas narastającego zbocza zegara powoduje rozpoczęcie pracy układu (porzucenie stanu main\_fsb\_idle).

## Opis działania automatu main\_fsm

* *main\_fsb\_idle –* stan oczekiwania (bezczynności) automatu; warunkiem przejścia jest sygnał data\_incoming => **receive**
* *main\_fsb\_receive* – stan rozpoczęcia odbioru danych; warunkiem przejścia jest zakończenie transmisji pierwszego modułu
* *main\_fsb\_busyx –* stany przejściowe (**x oznacza numer modułu)**; w zależności od zgodności otrzymanych danych co do długości (z dokładnością do 1 bajta), moduł jest przetwarzany lub pomijany; warunkiem rozpoczęcia przetwarzania jest zakończenie transmisji x modułu
* *main\_fsb\_procx ­–* stany przetwarzania modułów (**x oznacza numer modułu)**; stan następny ustalany jest w zależności od liczby modułów; warunkiem przejścia jest zakończenie przetwarzania *x* modułu
* *main\_fsb\_send –*  stan zakończenia pracy automatu; ustawia automat bufout\_fsm w odpowiednim stanie i wysyła sygnał rozpoczęcia wysyłania raportu do bufora wyjściowego

## Opis działania automatu flow\_fsm

Po otrzymaniu sygnału **flow\_in** automat opuszcza stan flow\_idle i rozpoczyna przyjmowanie danych. Stan *flow\_sop* wykrywa znacznik rozpoczęcia pakietu i przechodzi w stan następny Stany *flow\_eoh* i *flow\_eomx* są charakterystycznymi etapami transmisji danych. Oczekują na odpowiednie ciągi bitów (patrz - 3. Opis pakietu) w danych wejściowych. Po ich napotkaniu przechodzą do stanów następnych. W sytuacji napotkania nieodpowiedniej sekwencji przerywają pracę automatu. Stan *flow\_eop* oczekuje na znacznik końca transmisji.  
Pozostałe stany automatu odpowiedzialne są za ustawianie odpowiednich sygnałów enable i select dla multiplekserów, demultiplekserów, rejestrów i pamięci RAM, w celu odpowiedniego rozpropagowania i zapisania danych w układzie.

## Opis działania automatu proc\_fsm

* *proc\_fsb\_idle –* stan bezczynności (oczekiwania)
* *proc\_fsb\_calc* – stan w którym układ oblicza crc modułu; warunkiem przejścia jest zakończenie obliczeń
* *proc\_fsb\_comp –* stan w którymnastępuje porównanie crc dołączonego do modułu z obliczonym; przejście bezwarunkowe
* *proc\_fsb\_transmit ­–* stan w którym do bufora wyjściowego przesyłany jest element raportu związany z konkretnym modułem; warunkiem przejścia jest sygnał z bufora wyjściowego

## Opis działania automatu calc\_fsm

* *calc\_idle –* stan bezczynności (oczekiwania)
* *calc\_wait* – stan w którym układ odczekuje jeden cykl na dane z pamięci RAM
* *calc\_calculate –* stan w którymnastępuje obliczenie wartości CRC dla danych wejściowych do modułu crccalc; warunkiem przejścia jest napotkanie w ciągu danych wejściowych znacznika zakończenia modułu

## Opis działania automatu bufout\_fsm

* *bufout\_fsb\_idle –* stan bezczynności (oczekiwania)
* *bufout \_fsb\_receiving* – stan w którym bufor wyjściowy przyjmuje dane do rejestru
* *bufout \_fsb\_sending –* stan w którym raport gotowy jest do wysłania; przejście bezwarunkowe
* *bufout\_fsb\_clearing* – stan w którym po wysłaniu raportu czyszczony jest rejestr

# Opis procedury testowej

Środowisko testowe przygotowane zostało w programie ModelSim ALTERA STARTER EDITION 6.4a.

Odrębne procesy stimulator i monitor podłączone zostały do interfejsu układu.  
*Stimulator* w zależności od zajętości układu podaje, lub nie, dane zaczerpnięte z pliku \*.dat .   
Wymagany jest odpowiedni format pliku, zgodny z konstrukcją pakietu zdefiniowaną w niniejszej dokumentacji. Plik może zawierać wiele pakietów następujących po sobie.

*Monitor* odbiera dane od układu po otrzymaniu sygnału *send*, a następnie sporządza raport tekstowy o stanie modułów zapisywany w pliku sim\_res.dat .   
Dane do testów są ciągami znaków ASCII zaczerpniętymi z różnych źródeł tekstowych. Długości ciągów znaków zostały policzone korzystając z funkcjonalności pobocznej edytora Notepad++.  
Wartości CRC obliczone zostały przy pomocy funkcjonalności pobocznej edytora szesnastkowego HxD. Prawidłowość jego działania została wcześniej sprawdzona na podstawie ręcznych obliczeń.

**Należy zwrócić uwagę na fakt, że w przypadku transmisji pakietu z liczbą modułów x < 4, raport będzie zawierał tylko informacje o modułach z identyfikatorem > x.**

# Wyniki syntezy

Synteza jednostki przeprowadzona została dla układu EP2C35F672C6 z rodziny CycloneII przy pomocy pakietu Quartus II 9.0 Web Edition.  
W poniższej tabeli zawarte zostały wyniki syntezy w zależności od typu optymalizacji. Algorytmem kodowania stanów automatów jest **One-Hot.**

|  |  |  |
| --- | --- | --- |
| Metoda optymalizacji: | Total LE's | clock fmax |
| Speed | 471 | 120,39 MHz |
| Balanced | 475 | 110,04 MHz |
| Area | 472 | 111,99 MHz |

Skorzystanie z metody optymalizacji **Speed** wywołało poprawę zarówno w liczbie zajętych elementów logicznych jak i w częstotliwości pracy układu. Z tego powodu uważamy, że jest to najlepszy typ optymalizacji dla tego układu. Ustawienie **Balanced** przyniosło najgorsze rezultaty.

Poniższa tabela przedstawia wyniki syntezy dla dwóch różnych algorytmów kodowania stanów automatów i metody optymalizacji **Balanced**.

|  |  |  |
| --- | --- | --- |
| Kodowanie stanów | Total LE's | clock fmax |
| One-Hot | 475 | 110,04 MHz |
| Minimal Bits | 504 | 124,36 MHz |

Oba algorytmy kodowania stanów dają przeciwstawne sobie efekty. Z tego powodu ich dobór zależy od tego, czy priorytetem jest poziom zajętości zasobów układu, czy też jego maksymalna częstotliwość pracy.

Ponieważ układ zawiera 5 automatów z których 2 mają bardziej rozbudowaną funkcję przejść-wyjść, różnice między algorytmami kodowania stanów są wyraźnie widoczne.   
Do syntezy jednostki powinien być wykorzystywany algorytm Minimal Bits. Co prawda zwiększa on liczbę przerzutników niezbędną do kodowania stanów o ok 6%, jednak maksymalna częstotliwość pracy wzrasta o 13%.

# Podsumowanie

## Propozycje ulepszenia/optymalizacji

Układ jest w tym momencie nieelastyczny. Pozwala na obsługę pakietów zawierających do 4 modułów o odgórnie ustalonym rozmiarze. Wprowadzenie kilku parametrów ogólnych, definiujących te wymiary, a także unifikacja automatu głównego, pozwoliłoby skalować układ w zależności od jego zastosowania.

Należy rozważyć również zwiększenie przepływności danych w układzie poprzez zastosowanie pełnego potokowania. W chwili obecnej część układu jest zbudowana w sposób który umożliwia jego implementację, często jednak część jednostek pozostaje bezczynnych.

# Literatura i źródła

­

1. Łuba T.(red), Rawski M., Tomaszewicz P., Zbierzchowski B.: *Programowalne Układy Przetwarzania Sygnałów i Informacji.* WKŁ, Warszawa 2008.
2. Wykłady z przedmiotu "Układy Cyfrowe" prowadzonych na WEiTI PW przez dr Mariusza Rawskiego - *<http://www.zpt.tele.pw.edu.pl>*­.
3. Zbysiński P., *"CRC w VHDL"*, "Elektronika Praktyczna" 2005, nr 4, s. 41-42.
4. *[Cyclic Redundancy Check](http://www.relisoft.com/science/CrcMath.html)* [w:] „[http://www.relisoft.com/](http://www.relisoft.com/science/CrcMath.html)science”, 17 października 2007 (dostęp 1 kwiecień 2009).

1. http://www.easics.com/webtools/crctool [↑](#footnote-ref-1)