**Temat projektu:**

Tematem niniejszego projektu, wykonanego w ramach przedmiotu Systemy odporne na błędy, jest Symulacja pamięci RAM ECC zabezpieczonego kodem Hamminga.

**Skład zespołu:**

|  |  |
| --- | --- |
| **Imię i nazwisko:** | **Grupa:** |
| Mateusz Błaszczyk | 21A |
| Salem Albarudy | 21A |
| Kamil Bodziechowski | 21A |

**Wstęp teoretyczny:**

Kod Hamminga, wykorzystywany jest do wykrywania przekłamania jednego bitu w obrębie słowa bitowego. Dodatkowo jego zaletą jest wskazanie miejsca, na którym wystąpił błąd. Jednak nie jest to rozwiązanie idealne. W przypadku przekłamania dwóch lub większej ilości bitów, nie da się wykryć gdzie doszło do błędu. Bity kontrolne w kodzie Hamminga zajmują pozycje które są potęgą liczby dwa.

**Przykład kodu hamminga dla 8 bitowego słowa:**

Dla 8 bitowego przykładu bity kontrolne zajmą następujące pozycje:

* 1
* 2
* 4
* 8

Natomiast bity informacyjne pozcyje:

* 3
* 5
* 5
* 6
* 9
* 10
* 11
* 12

Poniżej zaprezentowany został rozkład bitów informacyjnych oraz kontrolnych dla 8bitowego słowa.



Każdy z bitów kontrolnych jest tworzony na podstawie bitów informacyjnych.

**hc1** = **bi1** XOR **bi2** XOR **bi4** XOR **bi5** XOR **bi7**

**hc2** = **bi1** XOR **bi3** XOR **bi4** XOR **bi6** XOR **bi7**

**hc3** = **bi2** XOR **bi3** XOR **bi4** XOR **bi8**

**hc4** = **bi5** XOR **bi6** XOR **bi7** XOR **bi8**

Przykładowo dla 8bitowego słowa: 1111 1111, bity kontrolne będą miały następujące wartości:

* hc1 = 1
* hc2 = 1
* hc3 = 0
* hc4 = 0

**Technologie:**

Projekt został wykonany w frontendowym webowym frameworku: Angular. W wersji 6.

Logika aplikacji została napisana w języku TypeScript. Jest to nadzbiór języka JavaScript.

Do stworzenia czystego projektu Angular niezbędne jest posiadanie zainstalowanego środwiska Node.js oraz Angular. Następnie za pomocą wiersza poleceń możliwe jest zbudowanie czystego projektu Angular 6. Służy do tego polecenie:

* ***ng new [nazwa]***

Opcjonalnie można dodać opcjonalne parametry. W przypadku tego projektu wykorzystany został parametr *prefix*. Za jego pomocą każdy komponent będzie posiadał przedrostek.

Uruchomienia aplikacji można dokonać za pomocą polecenia:

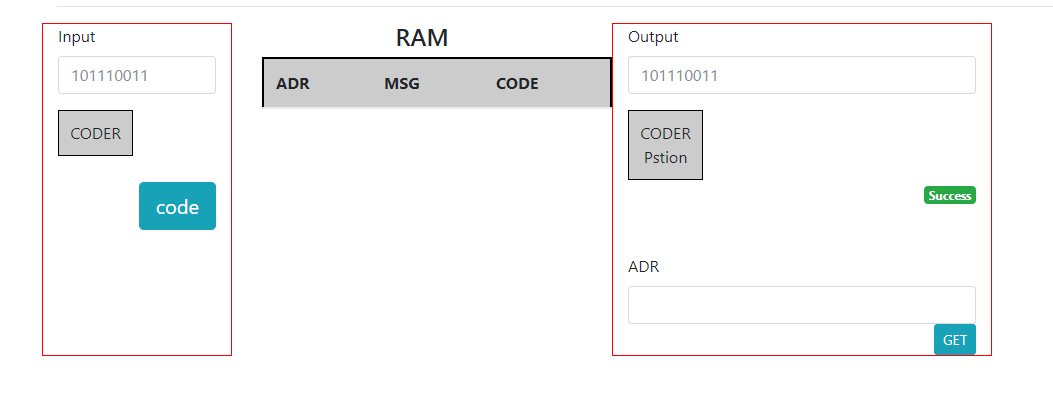
* ng serve --o

Do stworzenia widoku aplikacji posłużył HTML w wersji 5 oraz style CSS.

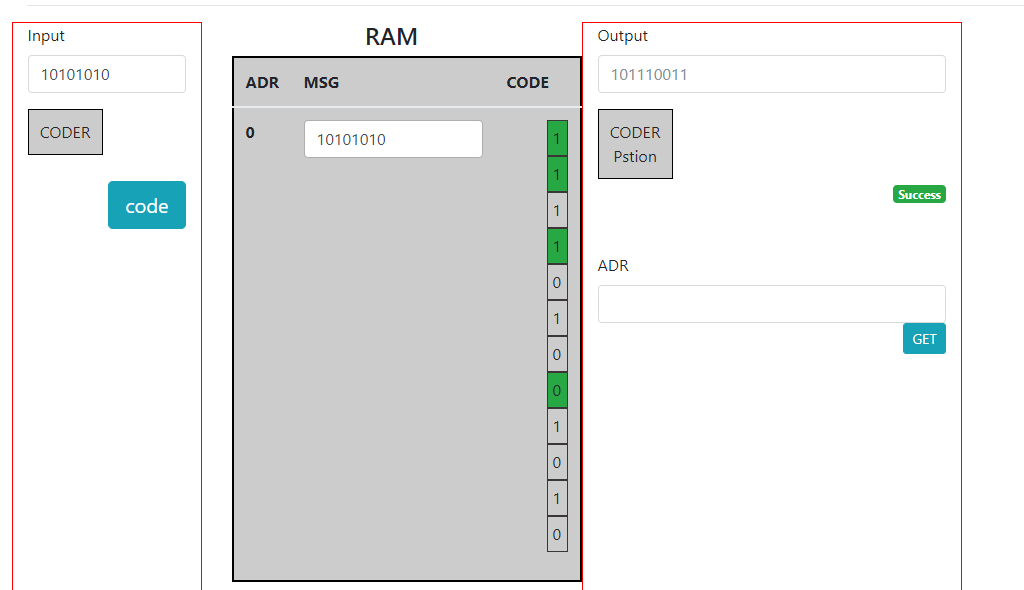
**Działanie aplikacji:**

W ramach projektu zostały stworzone dwa projekty. Pierwszy prezentuje działanie pamięci RAM ECC. Kod JS tego projektu dostępny jest na systemie kontroli wersji gitlab.Natomiast drugi projekt prezentuje bardziej wartości statystyczne. Cały kod znajduje się w systemie kontroli wersji gitlab.

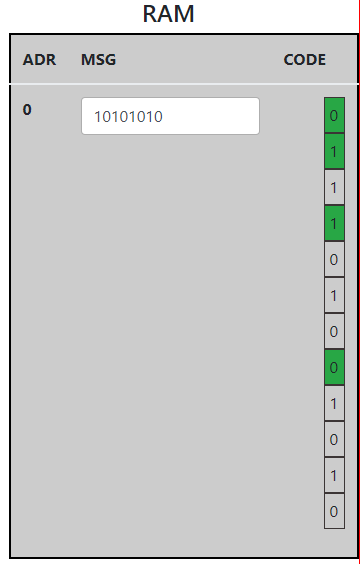
Pierwszy projekt daje możliwość ingerencji użytkownikowi w wynik jaki zostanie otrzymany na wyjściu.



Pierwszą czynnością jako musi wykonać użytkownik jest wpisanie wartości jaką chce zapisać w RAMie. Może tego dokonać poprzez wpisanie ciągu zer lub jedynek w pole oznaczone jako Input. Znajduje się ono po lewej stronie widoku. Efektem takiego działania będzie przypisanie wiadomości do pierwszego wolnego adresu (w tym przypadku adres posada numer 0), oraz zapisanie kodu Hamminga.

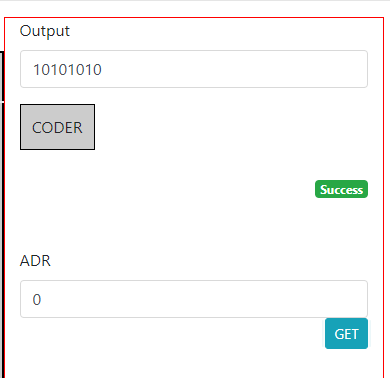


Użytkownik ma możliwość zmiany dowolnego znaku w kodzie Hamminga za pomocą naciśnięcia na wybrany bit. Efektem będzie zmiana bitu na przeciwny.

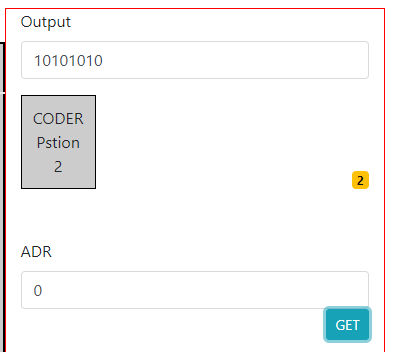


Ostatnią czynnością jako użytkownik może wykonać jest wybranie adresu. Należy wpisać konkretny adres w pole ADR, znajdujące sie w prawej części widoku, a następnie zatwierdzić przyciskiem GET.

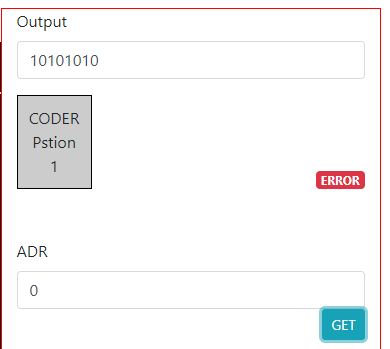
Wartość którą zwróci system może będzie wiadomość wpisana przez użytkownika. Dodatkowo system informuje o stanie RAMu. Jeżeli użytkownik nie zmieni nic w kodze Hamminga zapisanym w RAMie to system poinformuje o sukcesie.



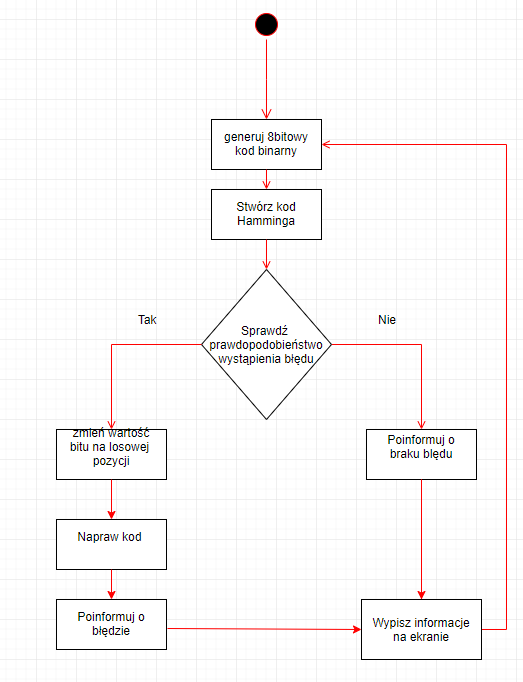
W przypadku kiedy zostanie mieniony jeden bit system zreperuje kod, ale poinformuje użytkownika o wystąpieniu błędu. Zwróci również pozycję na której wykrył nieprawidłowość.



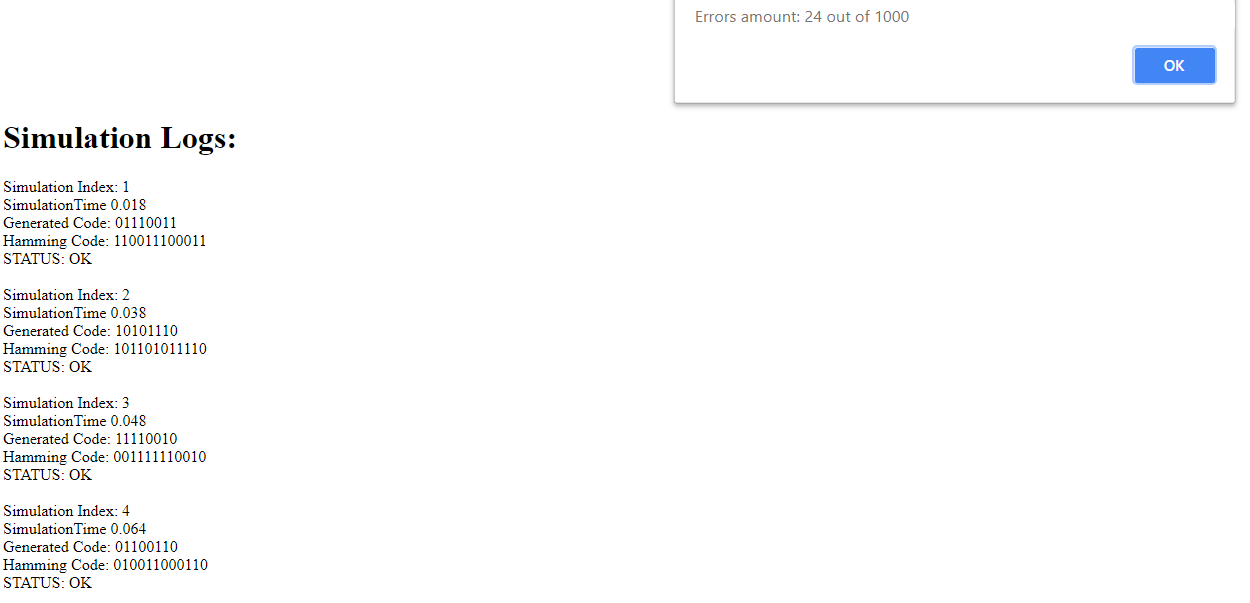
Ostatnim możliwym scenariuszem jest zmiana więcej niż jednego bitu. W takim przypadku system wykona próbę nareperowania kodu, a następnie ponownie go sprawdzi. W przypadku ponownego błędu zasygnalizuje uszkodzenie.



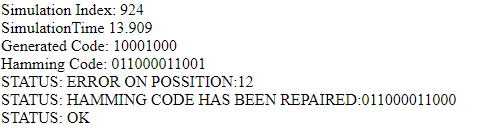
Druga wersja projektu polega na wykonaniu pewnego rodzaju symulacji RAMu ECC. System przeprowadza określoną liczbę razy (w poniższym przypadku jest to 1000) następującą sekwencję czynności:



Następnie na ekranie użytkownik otrzymuje informacje o ilości błędów oraz przebieg całej symulacji.



Błąd sygnalizowany jest w następujący sposób:



System zwraca pozycję błędu, a następnie go naprawia, po czym sprawdza ponownie.

Do wyliczenia prawdopodobieństwa używany jest rozkład Poissona, z wartością lambda równą: 1200+(indexSymulacji/5).

**Wnioski:**

- projekt działa, i spełnia swoje założenie. Prezentuje w doskonały sposób działanie i możliwości pamięci RAM ECC.

- praca nad projektem pozwoliła każdemu z nas poznać i zrozumieć działanie pamięci RAM ECC. Każdy z nas z zapałem wykorzysta nabytą wiedzę w przyszłości.

- system kontroli wersji to narzędzie znacznie zwiększające możliwości pracy w zespole. Jego największym atutem jest możliwość odzyskania zapisanych danych nawet w przypadku losowego, trwałego uszkodzenia dysku twardego w laptopie.

- RAM ECC nie może poprawić przekłamań biotwych występujących w liczbie większej niż jeden.

- RAM ECC może wykryć błąd w ciągu bitów, który pojawia sie tylko na jednej pozycji. Dodatkowo potrafi odnaleźć miejsce nieprawidłowości i poprawić cały kod.

- RAM ECC nie jest wydajny w do zabezpieczenia przed błędami krótkich słów, natomiast przy długich sprawdza się znakomicie.