Politechnika Świętokrzyska w Kielcach Wydział Elektrotechniki, Automatyki i Informatyki	
Systemy odporne na błędy - projekt	
Temat 7: Nadmiarowość TMR	**Autorzy:** Joanna Gmyr Zbigniew Bielecki Bartosz Dygas **Grupa**: 1ID21A

Spis treści

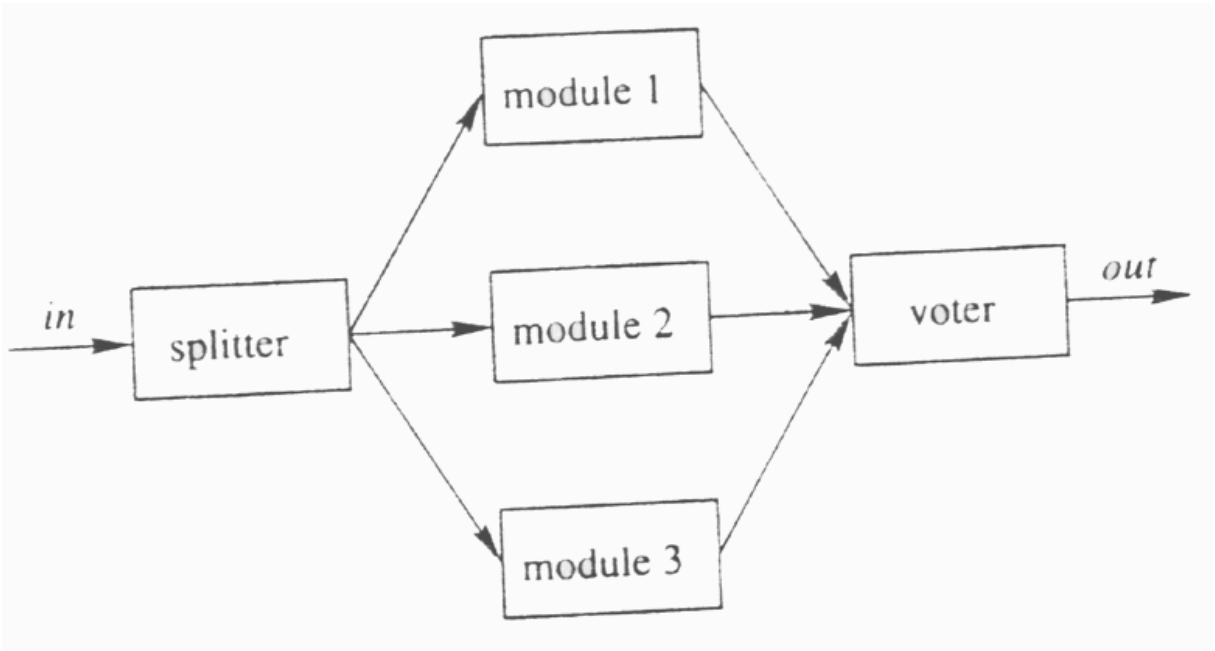
1. Opis użytych technologii. 1
2. Opis zastosowanych algorytmów. 1
3. Diagramy głównych klas. 4
4. Diagramy przypadków użycia. 4
5. Przedstawienie działania aplikacji. 4
6. Wnioski. 6

1. Opis użytych technologii.

Program zaimplementowano w języku C#. Użyto pliku mapowania w pamięci (ang. *Memory-Mapped File*), który zawiera zawartość pliku w pamięci wirtualnej. Mapowanie między plikiem i obszarem pamięci umożliwia aplikacji, w tym wielu procesom, modyfikowanie pliku przez odczytywanie i zapisywanie bezpośrednio w pamięci. Kodu zarządzanego można używać do uzyskiwania dostępu do plików mapowanych w pamięci w taki sam sposób, jak funkcje natywne Windows. Dostęp do plików mapowanych w pamięci, zgodnie z opisem są w bibliotece zarządzania plikami Memory-Mapped (ang. *Managing Memory-Mapped Files*).

2. Opis zastosowanych algorytmów.

Potrójna redundancja modularna to popularna technika tolerancji i detekcji błędów. Dzięki niej można poprawić niezawodność systemu przez połączenie trzech niezależnych instancji systemu poprzez rozdzielacz na wejściu i wybierak na wyjściu. W przypadku niezgodności sygnałów pochodzących od równoważnych trzech źródeł, wybierak rozstrzyga o prawidłowej wartości sygnału na podstawie "głosowania”. Zaletą TMR jest maskowanie przed użytkownikiem zarówno przelotnych jak i trwałych błędów.



Rysunek 1. Podstawowa wersja TMR.

W aplikacji kontroler dokonuje wyboru poprzez zastosowanie prostych instrukcji warunkowych. Istnieje także możliwość, iż wszystkie 3 wejścia okażą się inne od siebie - w takim przypadku aplikacja wyświetli okno z opisem błędu. Na sam koniec funkcja wyświetli wybraną przez kontroler liczbę.

```
private void buttonControlerStart_Click(object sender, EventArgs e)
{
    String res = "";
    if (textBoxFModule.Text == textBoxSModule.Text)
    {
        res = textBoxFModule.Text;
    }
    else if (textBoxSModule.Text == textBoxTModule.Text)
    {
        res = textBoxSModule.Text;
    }
    else if (textBoxFModule.Text == textBoxTModule.Text)
    {
        res = textBoxFModule.Text;
    }
    else
    {
        MessageBox.Show("All three modules values are diffrent!", "Comparing ERROR",
          MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    byte[] bytes = StringToBytes(res);
    int convertedInt = BitConverter.ToInt32(bytes, 0);
    textBoxControler.Text = convertedInt.ToString();
}
```

Rysunek 2. Funkcja *buttonControlerStartClick*.

Zapis i odczyt z pamięci polegają na użyciu zmiennej *mmf*, która pozwala na dostęp do danego punktu pamięci przy zastosowaniu nazwy. Jeśli nie ma zajętego miejsca w pamięci przypisanego do danej nazwy to zostanie ono utworzone.

Implementacja zapisu:

```
private void buttonSave_Click(object sender, EventArgs e)
{
    //byte[] bytes = StringToBytes(textBoxControler.Text);
    byte[] buffer = Encoding.UTF8.GetBytes(textBoxControler.Text);
    MemoryMappedViewAccessor mmva = mmf.CreateViewAccessor();
    mmva.Write(0, buffer.Length);
    mmva.WriteArray<byte>(4, buffer, 0, buffer.Length);
    mmva.Flush();
}
```

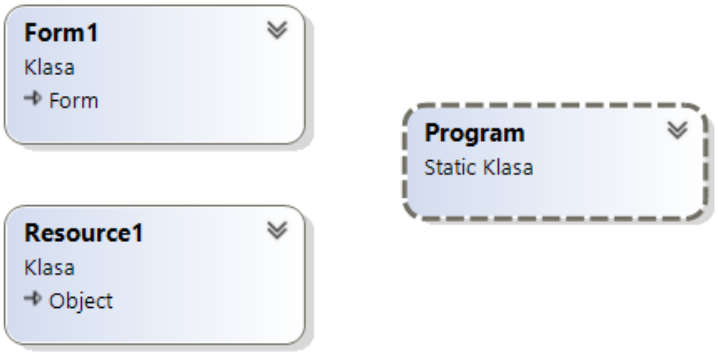
Rysunek 3. Funkcja *buttonSave_Click*.

Implementacja odczytu:

```
private void buttonStartRead_Click(object sender, EventArgs e)
{
    MemoryMappedViewAccessor mmva = mmf.CreateViewAccessor();
    byte[] buffer = new byte[mmva.ReadInt32(0)];
    mmva.ReadArray<byte>(4, buffer, 0, buffer.Length);
    String s = Encoding.UTF8.GetString(buffer);
    Console.WriteLine("buffer: {0}", s);
    labelReadValue.Text = s;
}
```

Rysunek 4. Funkcja *buttonStartRead_Click*.

3. Diagramy głównych klas.

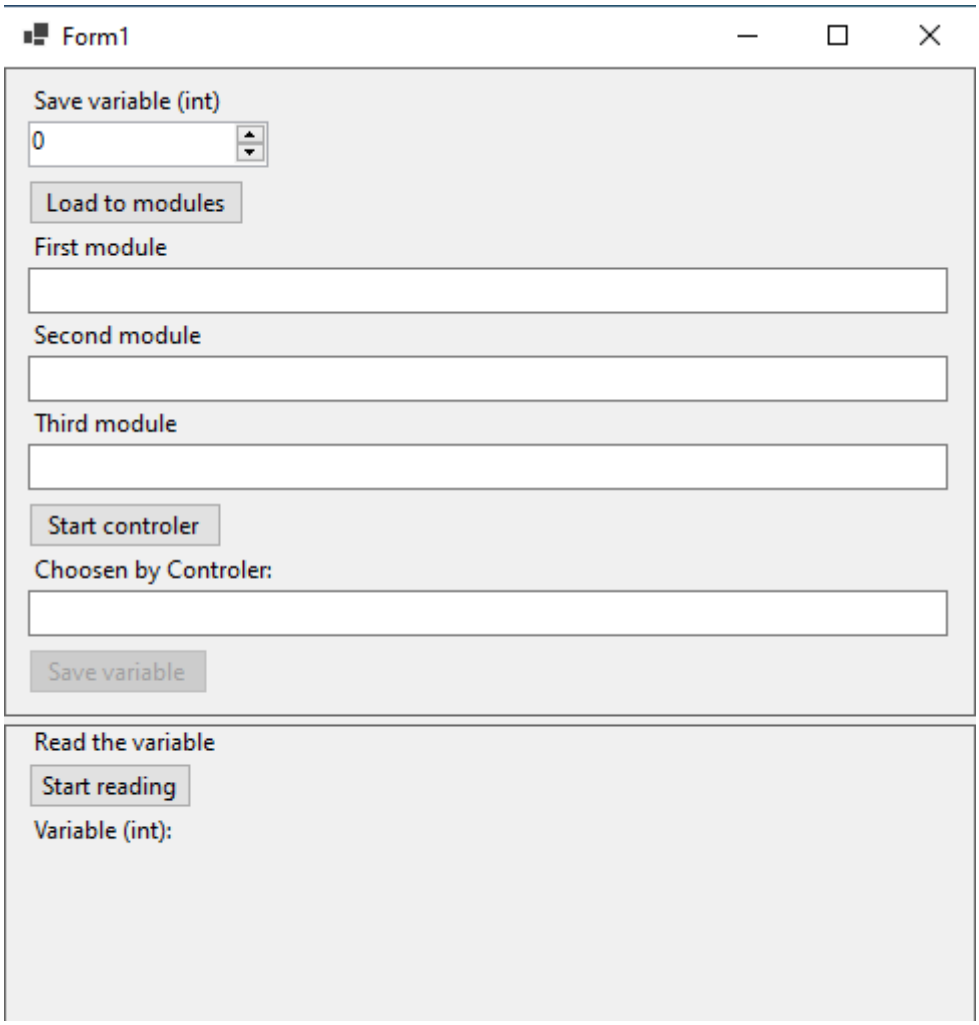


Rysunek 5. Diagram głównych klas.

4. Diagramy przypadków użycia.

5. Przedstawienie działania aplikacji.

Aplikacja prezentuje się następująco:



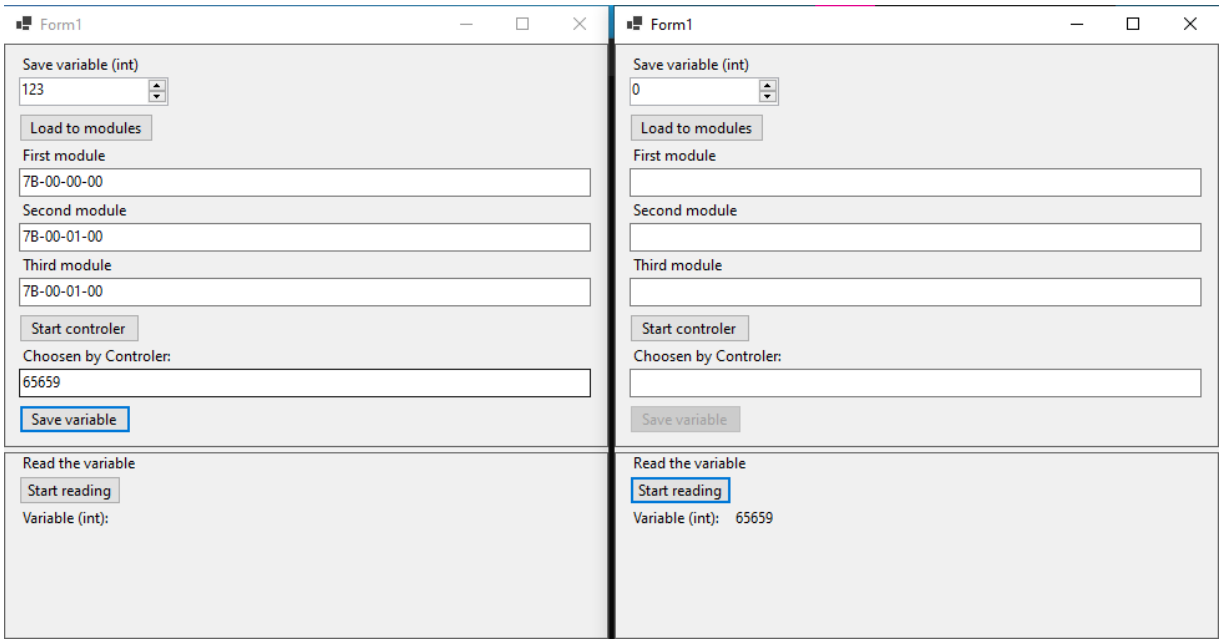
Rysunek 6. Wygląd aplikacji.

Użycie aplikacji:

1. Wpisuje się liczbę do pola *Save variable (int).*
2. Klika się przycisk *Load to modules*, aby zobaczyć podgląd binarny liczby.
3. Można zmienić bity, aby kontrolować wynik kontrolera.
4. Przycisk *Start controler* dokona porównania z każdego z trzech modułów i wyświetli wynik w *Choosen by Controler.*
5. Przycisk *Save variable* zapisze liczbę do pamięci.
6. Za pomocą przycisku *Start reading* pobierze się liczbę z pamięci i się ją wyświetli.

W przypadku działania dla dwóch aplikacji, dla testu wpisano liczbę 123 i załadowano do modułów. W modułach dokonano zmiany 3-ciego bitu dla drugiego i trzeciego modułu. Po

sprawdzeniu kontrolerem okazało się, że liczba do zapisania w pamięci jest inna od tej początkowo zakładanej. Próba odczytu z pamięci sprawi, że aplikacja otrzyma nieprawidłową liczbę. Zrzut ekranu przedstawiono poniżej:



Rysunek 7. Aplikacja z nieprawidłową liczbą.

6. Wnioski.

Po przeanalizowaniu otrzymanych wyników dochodzimy do wniosku, że użycie w powyższy sposób potrójnej redundancji modularnej może nie dać oczekiwanych skutków ze względu na możliwość uszkodzenia kontrolera lub któregoś z modułów. Aby poprawić tę niedogodność należałoby dodać wykrywanie uszkodzeń kontrolera lub modułu i odpowiednie reagowanie programu – na przykład zastąpienie uszkodzonego modułu przez inny, zapasowy.