Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий
институт
Кафедра «Информатика»
кафедра

ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ №4

Контролируемая среда тестирования
тема

Преподаватель _____ А. С. Кузнецов
подпись, дата инициалы, фамилия

Студент КИ18-17/1б 031830504 _____ Е.В. Железкин
номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2021

1 Цель работы

Изучить возможности создания контролируемой среды тестирования программного обеспечения, проведения регрессионного тестирования.

2 Задача работы

Научиться:

- Пользоваться подставными объектами
- Провести регрессионное тестирование

Вариант 5

Битовый вектор (реализовать в виде массива, хранящего значения true-false), при этом должны быть реализованы и протестированы следующие побитовые (поэлементные) логические операции:

- Сравнение
- И
- ИЛИ
- НЕ
- Импликация

2.1 Инструкция по запуску

Необходимо установить .Net SDK:

- Страница загрузки для Windows: https://dotnet.microsoft.com/download/dotnet/5.0
- Для Linux:

```
$ sudo apt-get update; \
$ sudo apt-get install -y apt-transport-https && \
```

*$ sudo apt-get update && *

$ sudo apt-get install -y dotnet-sdk-5.0

Далее на любой из двух систем выполнить в папке проекта:

$ dotnet test

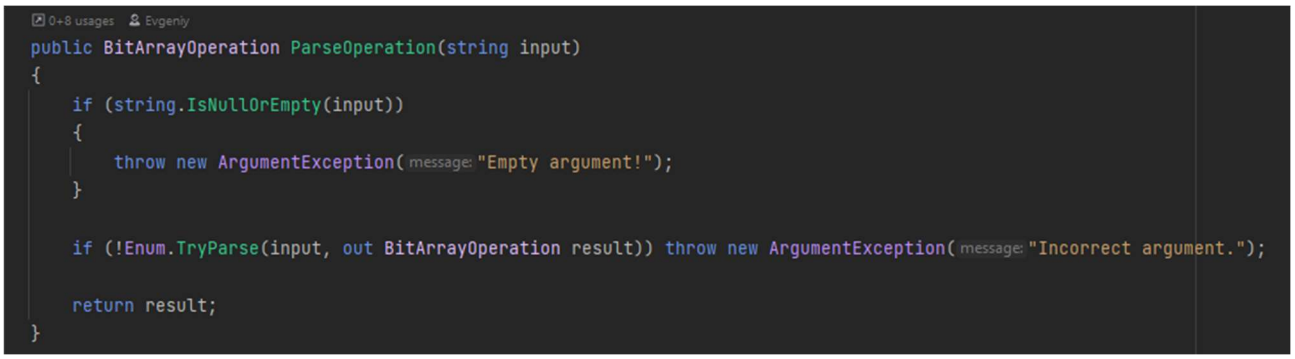
$ dotnet test -v n (для более подробного вывода)

3 Ход работы

В ходе выполнения данной практической работы был произведён рефакторинг класса из предыдущей работы. Теперь метод парсинга строки внесен в отдельный класс, так же создан конечный автомат с 3мя состояниями, для выполнения операций над битовыми векторами. (Рисунки 1-3)

```
public class BitArrayParser : IBitArrayParser
{
    /// <summary>
    /// Extension method allows translate a string to BitArray instance
    /// </summary>
    /// <param name="input">String to convert</param>
    /// <returns>BitArray instance obtained as a result of conversion</returns>
    /// <exception cref="ArgumentException">Empty or invalid input string format</exception>
    0+15 usages  Evgeniy *
    public BitArray ParseArrayValue(string input)
    {
        if (string.IsNullOrEmpty(input))
        {
            throw new ArgumentException( message: "Empty argument!");
        }

        input = input.Replace( oldValue: " ", newValue: "");
        InputType type;

        if (Regex.IsMatch(input, pattern: @"^[01]*$", options: RegexOptions.Compiled | RegexOptions.IgnoreCase))
        {
            type = InputType.Numeric;
        }
        else
        {
            if (Regex.IsMatch(input, pattern: @"^(true,|false,)*(true|false){1}$",
                options: RegexOptions.Compiled | RegexOptions.IgnoreCase))
            {
                type = InputType.Boolean;
            }
            else
            {
                throw new ArgumentException( message: "Incorrect argument.");
            }
        }

        // BitArray result = new BitArray(0);

        switch (type)
        {
            case InputType.Boolean:

                var temp :string[] = input.Split( separator: ',');
                var bMediator = new bool[temp.Length];

                for (var i = 0; i < temp.Length; i++)
                {
                    if (temp[i].ToLower().Equals("false"))
                    {
                        bMediator[i] = false;
                    }
                    else
                    {
                        bMediator[i] = true;
                    }
                }
```

Рисунок 1 – Функция парсинга строки в аргумент (битовый вектор)



```
0+8 usages  Evgeniy
public BitArrayOperation ParseOperation(string input)
{
    if (string.IsNullOrEmpty(input))
    {
        throw new ArgumentException( message: "Empty argument!");
    }

    if (!Enum.TryParse(input, out BitArrayOperation result)) throw new ArgumentException( message: "Incorrect argument.");

    return result;
}
```

Рисунок 2 – Функция парсинга операции

```
10 usages  Evgeniy
public class BitArrayCalc
{
    private readonly IBitArrayParser _parser;
    private BitArrayCalcState _state = BitArrayCalcState.WaitingForStart;
    private BitArrayOperation _currentOperation;
    private BitArray _currentArray;

    10 usages  Evgeniy
    public BitArrayCalc(IBitArrayParser parser)
    {
        _parser = parser;
    }

    3 usages  Evgeniy
    public BitArray GetArray()
    {
        return _currentArray;
    }

    21 usages  Evgeniy
    public BitArrayCalcStatus Step(string input)
    {
        try
        {
            switch (_state)
            {
                case BitArrayCalcState.WaitingForStart:
                    _currentArray = _parser.ParseArrayValue(input);
                    break;

                case BitArrayCalcState.WaitingForOp:
                    _currentOperation = _parser.ParseOperation(input);
                    if (_currentOperation == BitArrayOperation.Not)
                    {
                        MakeNegative();
                    }
                    break;

                case BitArrayCalcState.WaitingForArg:
                    var arg:BitArray = _parser.ParseArrayValue(input);
                    MakeOperation(arg);
                    break;

                default:
                    throw new ArgumentOutOfRangeException();
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
            return BitArrayCalcStatus.Error;
        }

        NextState();
```

Рисунок 3 – Класс конечного автомата

Для наглядности мок-тестирования, в классе-парсере всегда возвращается значение *null,* чтобы нагляднее отображалась независимость тестирования от внешних модулей (Рисунок 4).

```
    // BitArray result = new BitArray(0);

    switch (type)
    {
        case InputType.Boolean:

            var temp :string[] = input.Split( separator: ',');
            var bMediator = new bool[temp.Length];

            for (var i = 0; i < temp.Length; i++)
            {
                if (temp[i].ToLower().Equals("false"))
                {
                    bMediator[i] = false;
                }
                else
                {
                    bMediator[i] = true;
                }
            }

            // result = new BitArray(bMediator);

            break;

        case InputType.Numeric:

            var nMediator = new bool[input.Length];

            for (var i = 0; i < input.Length; i++)
            {
                if (input[i] == '0')
                {
                    nMediator[i] = false;
                }
                else
                {
                    nMediator[i] = true;
                }
            }

            // result = new BitArray(nMediator);

            break;
    }

    return null;
}
```

Рисунок 4 – Функция парсинга аргумента возвращает *null*

```
Evgeniy *
public class Tests
{
    private Mock<IBitArrayParser> _mockParser;

    [SetUp]
    Evgeniy *
    public void Setup()
    {
        _mockParser = new Mock<IBitArrayParser>();
        _mockParser.Setup( expression: p :IBitArrayParser => p.ParseArrayValue( input: "101")).Returns(new BitArray( values: new[] {true, false, true}));
        _mockParser.Setup( expression: p :IBitArrayParser => p.ParseArrayValue( input: "1001")).Returns(new BitArray( values: new[] {true, false, false, true}));
        _mockParser.Setup( expression: p :IBitArrayParser => p.ParseArrayValue( input: "1100")).Returns(new BitArray( values: new[] {true, true, false, false}));
        _mockParser.Setup( expression: p :IBitArrayParser => p.ParseArrayValue( input: "1010")).Returns(new BitArray( values: new[] {true, false, true, false}));
        _mockParser.Setup( expression: p :IBitArrayParser => p.ParseArrayValue( input: "1")).Returns(new BitArray( values: new[] {true}));
        _mockParser.Setup( expression: p :IBitArrayParser => p.ParseArrayValue( input: "true,false")).Returns(new BitArray( values: new[] {true, false}));

        _mockParser.Setup( expression: p :IBitArrayParser => p.ParseOperation( input: "And")).Returns(BitArrayOperation.And);
        _mockParser.Setup( expression: p :IBitArrayParser => p.ParseOperation( input: "Impl")).Returns(BitArrayOperation.Impl);
        _mockParser.Setup( expression: p :IBitArrayParser => p.ParseOperation( input: "Not")).Returns(BitArrayOperation.Not);
        _mockParser.Setup( expression: p :IBitArrayParser => p.ParseOperation( input: "Comp")).Returns(BitArrayOperation.Comp);
        _mockParser.Setup( expression: p :IBitArrayParser => p.ParseOperation( input: "Or")).Returns(BitArrayOperation.Or);

        _mockParser.Setup( expression: p :IBitArrayParser => p.ParseArrayValue( input: null)).Throws(new ArgumentException( message: "Empty argument!"));
    }
```

Рисунок 5 – Конфигурация мок-сущности

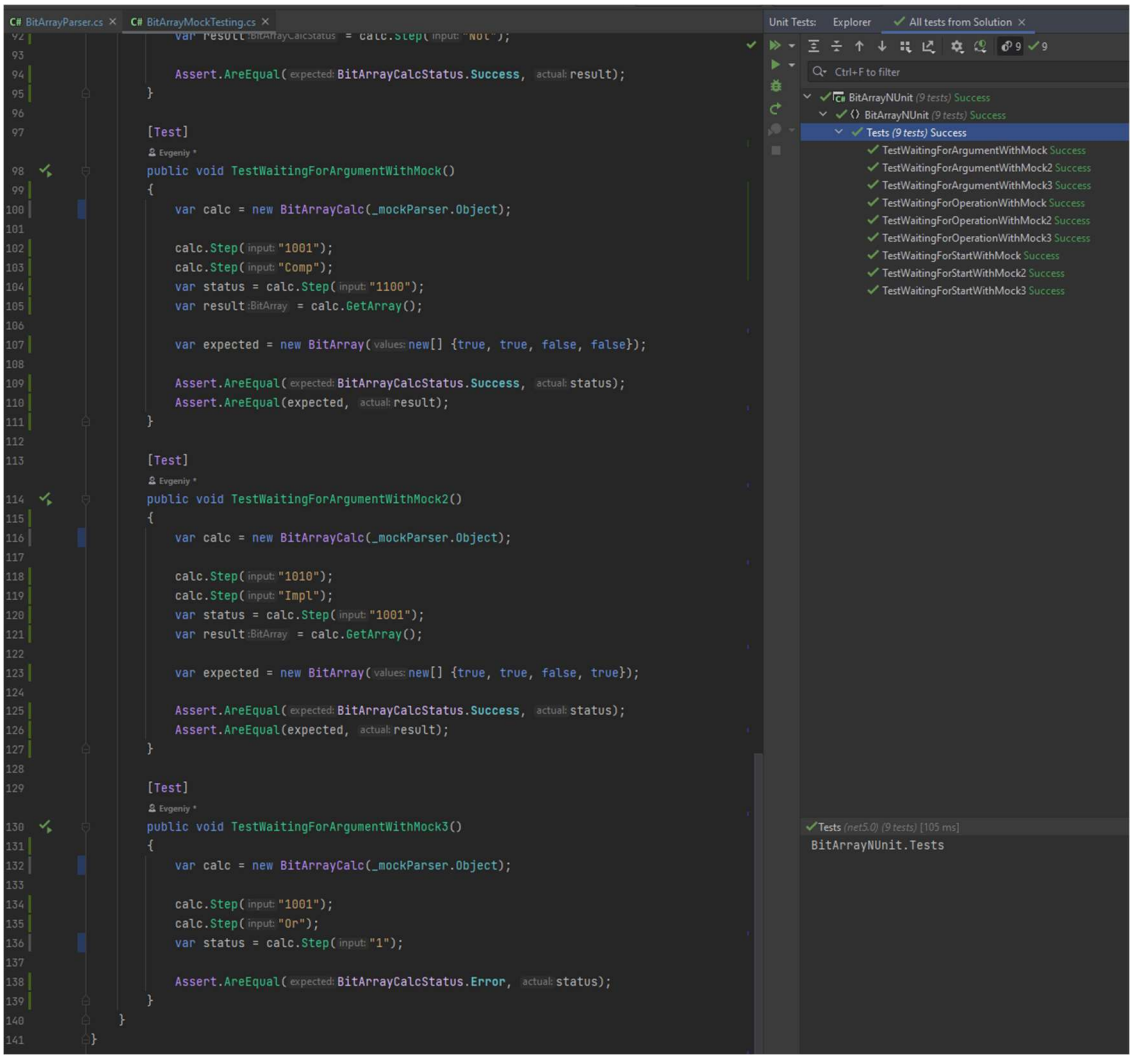


Рисунок 5 – Непосредственно мок-тестирование

Во время реализации было принято решение заменить токены на альтернативу «Enum», так как токены добавляли лишние сложности). Тем не менее классы токенов были реализованы для тестов, но на практике не применялись. (Рисунок 7)

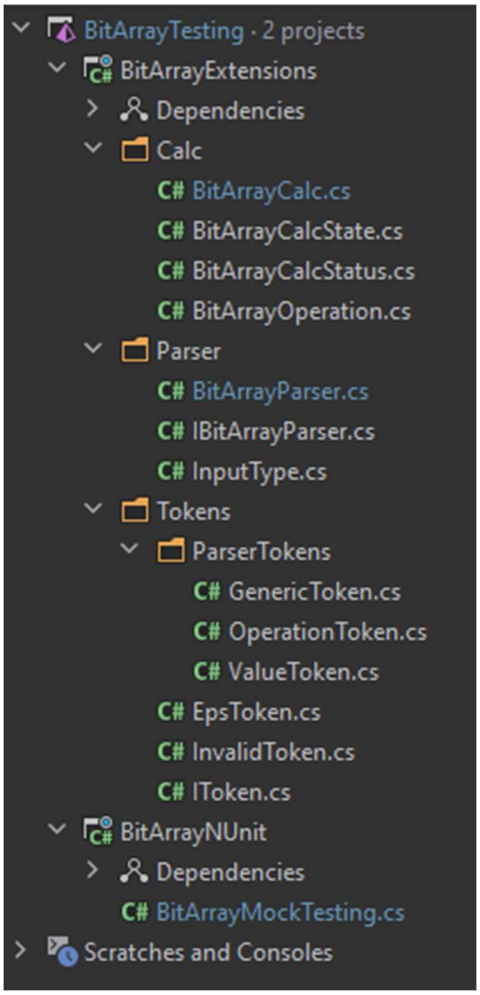


Рисунок 7 – Структура проекта

В качестве примера регрессионного тестирования покроем тестами весь конечный автомат(файл BitArrayCalc.cs). (Рисунки 8-17)

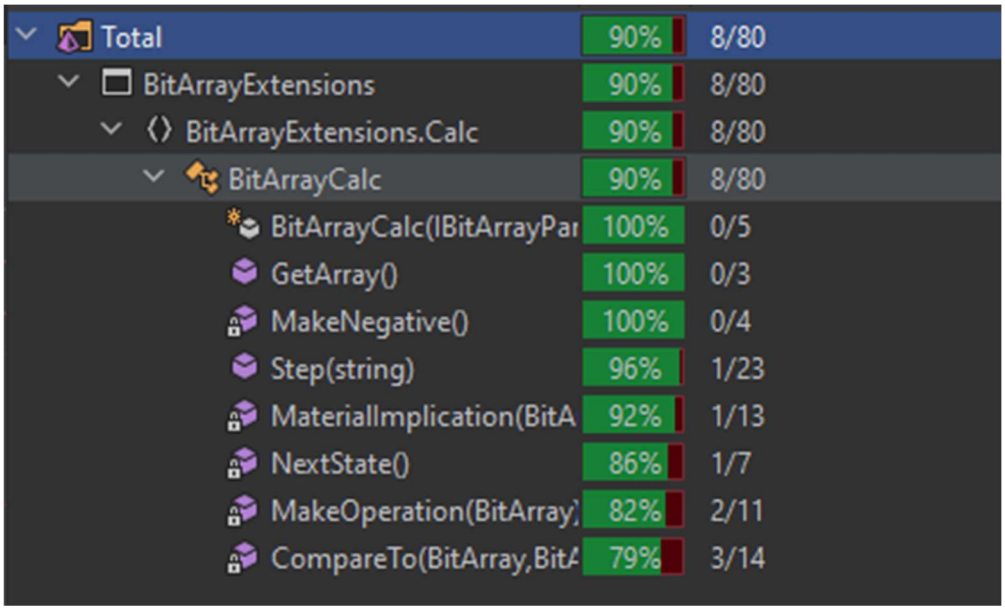


Рисунок 8 – Исходное покрытие тестами

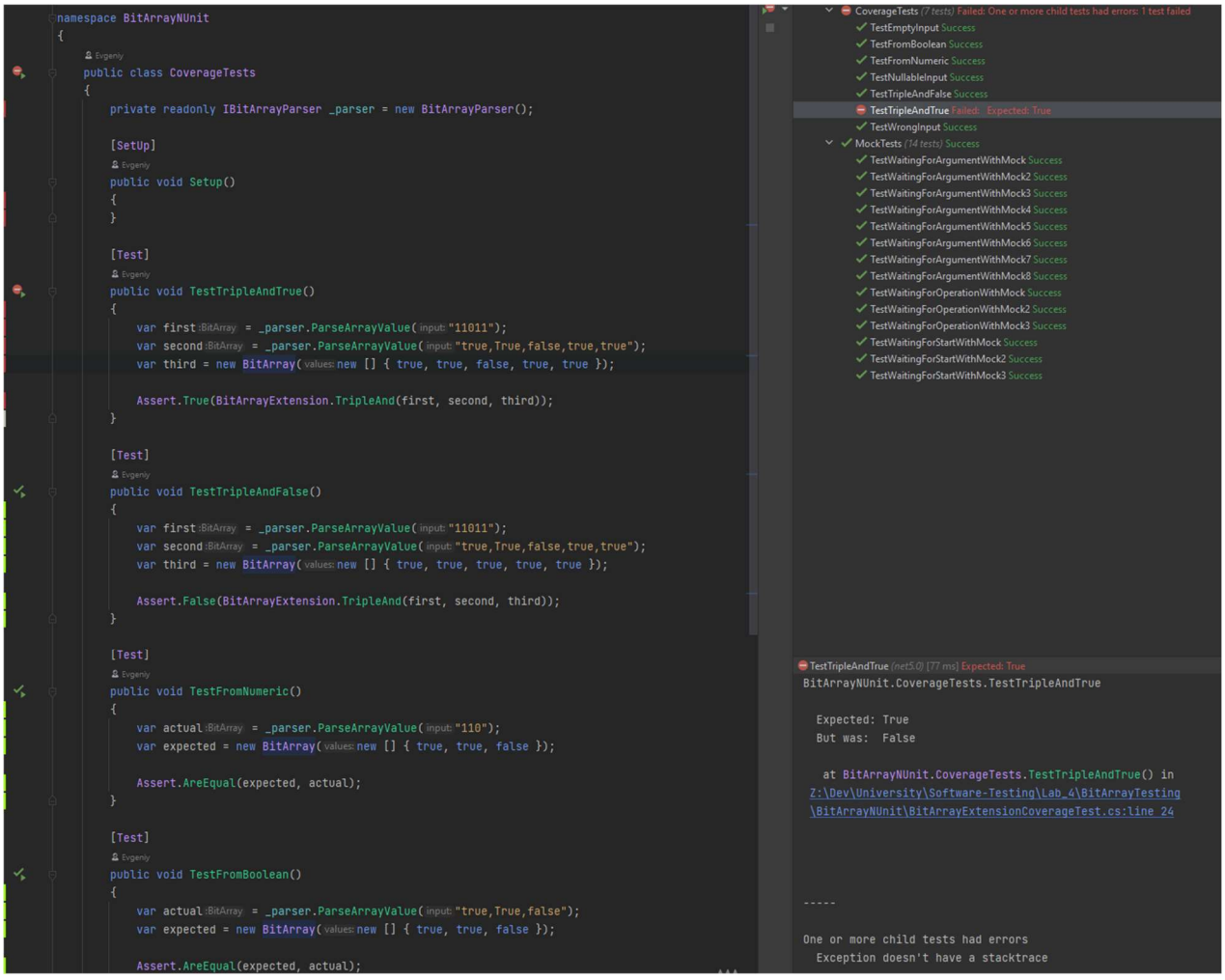
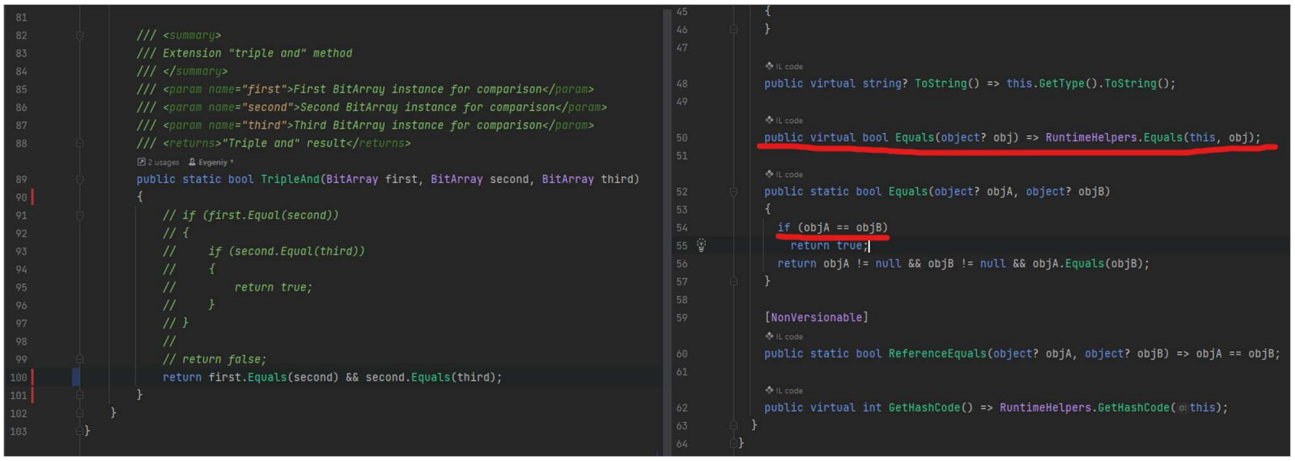


Рисунок 9 – ВНЕЗАПНАЯ регрессионная ошибка, после добавления старых тестов



```
/// <summary>
/// Extension "triple and" method
/// </summary>
/// <param name="first">First BitArray instance for comparison</param>
/// <param name="second">Second BitArray instance for comparison</param>
/// <param name="third">Third BitArray instance for comparison</param>
/// <returns>"Triple and" result</returns>
2 usages  Evgeniy *
public static bool TripleAnd(BitArray first, BitArray second, BitArray third)
{
    // if (first.Equal(second))
    // {
    //     if (second.Equal(third))
    //     {
    //         return true;
    //     }
    // }
    //
    // return false;
    return first.Equals(second) && second.Equals(third);
}
}
}
```

```
{
}

IL code
public virtual string? ToString() => this.GetType().ToString();

IL code
public virtual bool Equals(object? obj) => RuntimeHelpers.Equals(this, obj);

IL code
public static bool Equals(object? objA, object? objB)
{
  if (objA == objB)
    return true;
  return objA != null && objB != null && objA.Equals(objB);
}

[NonVersionable]
IL code
public static bool ReferenceEquals(object? objA, object? objB) => objA == objB;

IL code
public virtual int GetHashCode() => RuntimeHelpers.GetHashCode(this);
}
}
```

Рисунок 11 – По всей видимости системный метод *Equals* сравнивает ссылки объектов класса *BitArray*

```
/// <summary>
/// Extension method adding Equality functionality
/// </summary>
/// <param name="firstArray">First array of bits to compute</param>
/// <param name="secondArray">Second array of bits to compute</param>
/// <returns>Equality result</returns>
/// <exception cref="ArgumentException">Array lengths must be the same</exception>
2 usages  Evgeniy
public static bool Equal(this BitArray firstArray, BitArray secondArray)
{
    if (firstArray.Length != secondArray.Length)
        throw new ArgumentException( message: "Array lengths must be the same.");

    for (var i = 0; i < firstArray.Length; i++)
    {
        if (firstArray[i] != secondArray[i])
            return false;
    }

    return true;
}
```

Рисунок 12 – Был определён собственный метод Equal, работающий корректно

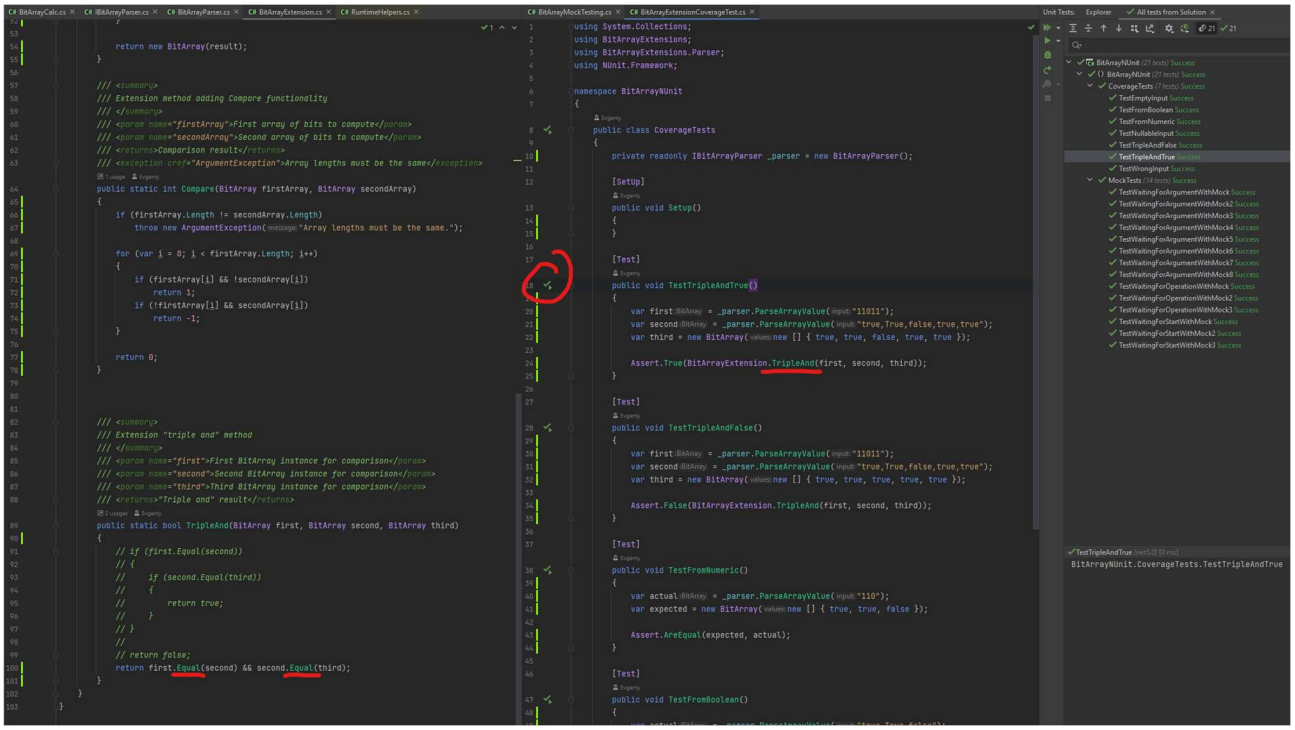


Рисунок 13 – Успешное прохождение тестов, после устранения неполадки

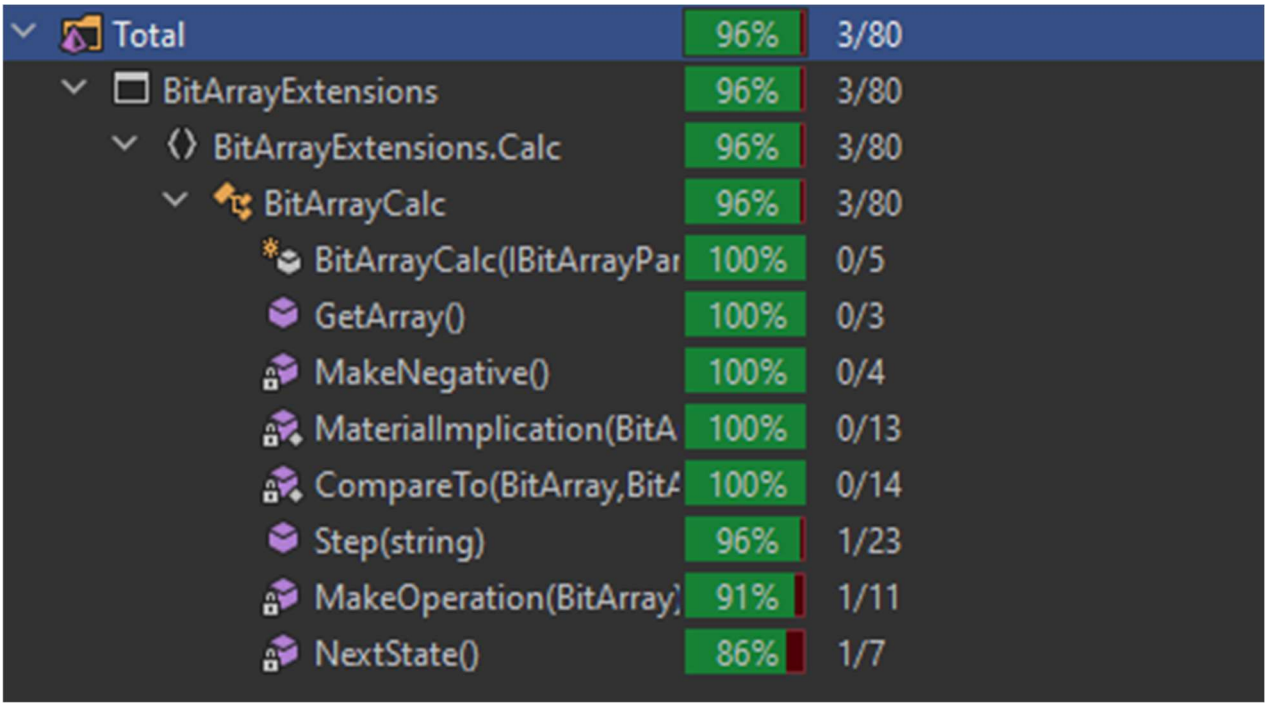


Рисунок 14 – Покрытие, после добавления необходимых тестов

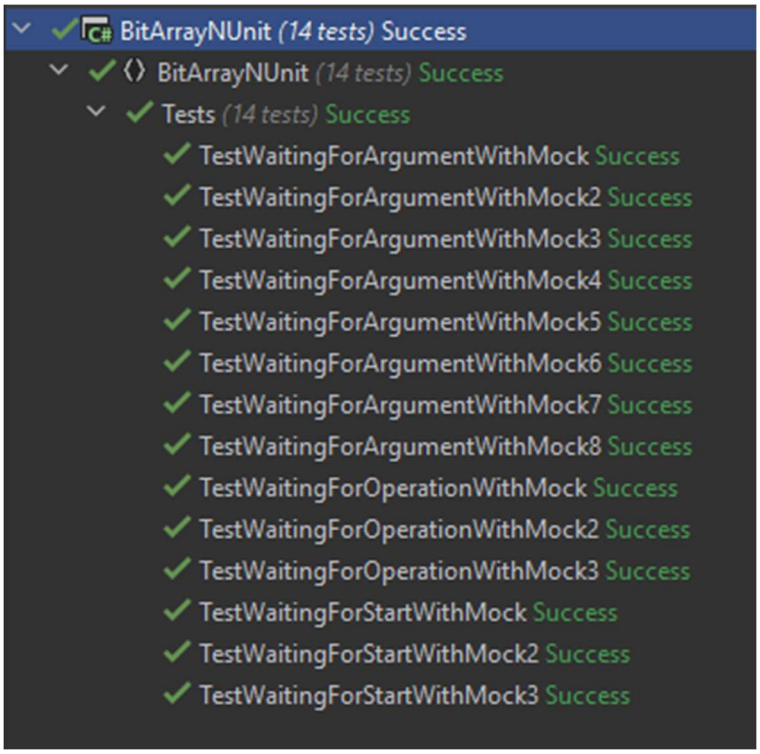


Рисунок 15 – Финальный набор тестов

В данном случае невозможно достигнуть 100% покрытия тестов, потому что некоторые строки кода не могут быть достигнуты ни при каких обстоятельствах, но написаны, чтобы статический анализатор не подсвечивал их жёлтеньким. К примеру функция *NextState*, в ней происходит смена состояния автомата. Так как у автомата всего 3 состояния и рассмотрены все возможные случаи, то функция никогда не дойдёт до (1)25 строки: (Рисунки 16-17)

Рисунок 16 – Оригинальная функция *NextState*

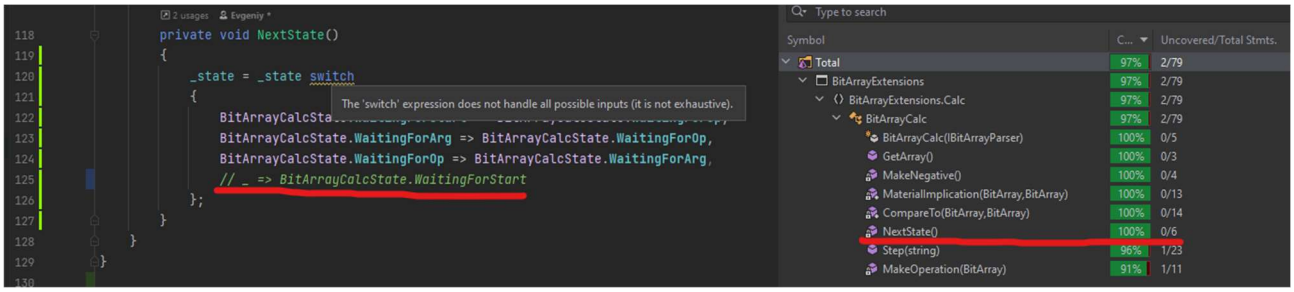


Рисунок 17 – Функция *NextState* со 100% покрытием

По итогам проведения регрессионного тестирования была выявлена и исправлена ошибка в системном методе *Equals*, теперь модуль ведёт себя исправно.

4 Вывод

В ходе данной лабораторной работы были изучены возможности создания контролируемой среды тестирования программного обеспечения, проведения регрессионного тестирования.