WolfNet 6502 WorkBench Computer Emulator

beta

Generated by WolfNet Computing using Doxygen 1.9.5

1 Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

2 Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

3 Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

4 File Index

4.1 File List

Here is a list of all files with brief descriptions:

5 Namespace Documentation

5.1 Emulator Namespace Reference

Namespaces

- namespace Model
- namespace ViewModel

Classes

- class App

  Interaction logic for App.xaml
- class ExitCodes
- interface IClosable
- class MainWindow

  Interaction logic for MainWindow.xaml
- class MultiThreadedObservableCollection

  A MultiThreaedObservableCollection. This allows multiple threads to access the same observable collection in a safe manner.
- class SaveFile

  SaveFile
- class Settings

  Settings
- class SettingsFile
- class Versioning

5.2 Emulator.Model Namespace Reference

Classes

- class Breakpoint
  A Representation of a Breakpoint
- class BreakpointType
  The Type of Breakpoint
- class MemoryRowModel
  A Model of a Single Page of memory
- class OutputLog
  The OutputLog Model. Used by the outputlog grid to show a history of operations performed by the CPU
- class RomFileModel
  The Model used when Loading a Program.
- class SettingsModel
  Model that contains the required information needed to save the current settings to disk
- class StateFileModel
  Model that contains the required information needed to save the current state of the processor to disk

5.3 Emulator.ViewModel Namespace Reference

Classes

- class MainViewModel
  The Main ViewModel
- class SaveFileViewModel
  The ViewModel Used by the SaveFileView
- class SettingsViewModel
  The ViewModel Used by the SaveFileView
- class ViewModelLocator
  This class contains static references to all the view models in the application and provides an entry point for the bindings.

5.4 Hardware Namespace Reference

Classes

- class AT28CXX
  An implementation of a W65C02 Processor.
- class Disassembly
  Used to help simulating. This class contains the disassembly properties.
- class HM62256
- class MemoryMap
- class Utility
- class W65C02
  An implementation of a W65C02 Processor.
- class W65C22
  An implementation of a W65C22 VIA.
- class W65C51
  An implementation of a W65C51 ACIA.

Enumerations

- enum AddressingMode

  The addressing modes used by the 6502 Processor

5.4.1 Enumeration Type Documentation

5.4.1.1 AddressingMode `enum Hardware.AddressingMode`

The addressing modes used by the 6502 Processor

Definition at line 6 of file AddressingMode.cs.

```
    {
/// <summary>
/// In this mode a full address is given to operation on IE: Memory byte[] { 0x60, 0x00, 0xFF }
/// would perform an ADC operation and Add the value at ADDRESS 0xFF00 to the accumulator.
/// The address is always LSB first
/// </summary>
        Absolute = 1,
/// <summary>
/// In this mode a full address is given to operation on IE: Memory byte[] { 0x7D, 0x00, 0xFF } The full value would then be added to the X Register.
/// If the X register was 0x01 then the address would be 0xFF01.  and the value stored there would have an ADC operation performed on it and the value would
/// be added to the accumulator.
/// </summary>
        AbsoluteX = 2,
/// <summary>
/// In this mode a full address is given to operation on IE: Memory byte[] { 0x79, 0x00, 0xFF } The full value would then be added to the Y Register.
/// If the Y register was 0x01 then the address would be 0xFF01.  and the value stored there would have an ADC operation performed on it and the value would
/// be added to the accumulator
/// </summary>
        AbsoluteY = 3,
/// <summary>
/// In this mode the instruction operates on the accumulator.  No operands are needed.
/// </summary>
        Accumulator = 4,
/// <summary>
/// In this mode, the value to operate on immediately follows the instruction.  IE: Memory byte[] { 0x69, 0x01 }
/// would perform an ADC operation and Add 0x01 directly to the accumulator
/// </summary>
        Immediate = 5,
/// <summary>
/// No address is needed for this mode.  EX: BRK (Break), CLC (Clear Carry Flag) etc
/// </summary>
        Implied = 6,
/// <summary>
/// In this mode assume the following
/// Memory = { 0x61, 0x02, 0x04, 0x00, 0x03 }
/// RegisterX = 0x01
/// 1.  Take the sum of the X Register and the value after the opcode 0x01 + 0x01 = 0x02.
/// 2.  Starting at position 0x02 get an address (0x04,0x00) = 0x0004
/// 3.  Perform the ADC operation and Add the value at 0x0005 to the accumulator
/// Note:  if the Zero Page address is greater than 0xff then roll over the value.  IE 0x101 rolls over to 0x01
/// </summary>
        IndirectX = 7,
/// <summary>
/// In this mode assume the following
/// Memory = { 0x61, 0x02, 0x04, 0x00, 0x03 }
/// RegisterY = 0x01
/// 1.  Starting at position 0x02 get an address (0x04,0x00) = 0x0004
/// 2.  Take the sum of the Y Register and the absolute address 0x01+0x0004 = 0x0005
/// 3.  Perform the ADC operation and Add the value at 0x0005 to the accumulator
/// Note:  if the address is great that 0xffff then roll over IE: 0x10001 rolls over to 0x01
/// </summary>
        IndirectY = 8,
/// <summary>
/// JMP is the only operation that uses this mode.  In this mode an absolute address is specified that points to the location of the absolute address we want to jump to.
/// </summary>
```

```
        Indirect = 9,
/// <summary>
/// This Mode Changes the PC. It allows the program to change the location of the PC by 127 in either direction.
/// </summary>
        Relative = 10,
/// <summary>
/// In this mode, a zero page address of the value to operate on is specified.  This mode can only operation on values between 0x0 and 0xFF, or those that sit on the zero page of memory.  IE: Memory byte[] { 0x69, 0x02, 0x01 }
/// would perform an ADC operation and Add 0x01 directly to the Accumulator
/// </summary>
        ZeroPage = 11,
/// <summary>
/// In this mode, a zero page address of the value to operate on is specified, however the value of the X register is added to the address IE: Memory byte[] { 0x86, 0x02, 0x01, 0x67, 0x04, 0x01 }
/// In this example we store a value of 0x01 into the X register, then we would perform an ADC operation using the addres of 0x04+0x01=0x05 and Add the result of 0x01 directly to the Accumulator
/// </summary>
        ZeroPageX = 12,
/// <summary>
/// This works the same as ZeroPageX except it uses the Y register instead of the X register.
/// </summary>
        ZeroPageY = 13,
    }
```

5.5 XamlGeneratedNamespace Namespace Reference

Classes

- class GeneratedApplication

  GeneratedApplication
- class GeneratedInternalTypeHelper

  GeneratedInternalTypeHelper

6 Class Documentation

6.1 Hardware.MemoryMap.Devices.ACIA Class Reference

Static Public Attributes

- static int Length = 0x03
- static byte Offset = 0x10

6.1.1 Detailed Description

Definition at line 58 of file MemoryMap.cs.

6.1.2 Member Data Documentation

6.1.2.1 Length `int Hardware.MemoryMap.Devices.ACIA.Length = 0x03 [static]`

Definition at line 60 of file MemoryMap.cs.

6.1.2.2 Offset `byte Hardware.MemoryMap.Devices.ACIA.Offset = 0x10 [static]`

Definition at line 61 of file MemoryMap.cs.

The documentation for this class was generated from the following file:

- Hardware/Classes/MemoryMap.cs

6.2 Emulator.App Class Reference

Interaction logic for App.xaml

6.2.1 Detailed Description

Interaction logic for App.xaml

Definition at line 6 of file App.xaml.cs.

The documentation for this class was generated from the following file:

- Emulator/App.xaml.cs

6.3 Hardware.AT28CXX Class Reference

An implementation of a W65C02 Processor.

Public Member Functions

- AT28CXX (int offset, int length, byte banks)

  Default Constructor, Instantiates a new instance of the processor.
- void Load (byte[ ][ ] data)

  Loads a program into ROM.
- void Load (byte bank, byte[ ] data)

  Loads a program into ROM.
- byte[ ][ ] ReadFile (string filename)
- byte Read (int address)

  Returns the byte at a given address without incrementing the cycle. Useful for test harness.
- void Write (int address, byte data)

  Writes data to the given address without incrementing the cycle.
- byte[ ][ ] DumpMemory ()

  Dumps the entire memory object. Used when saving the memory state
- byte[ ] DumpMemory (byte bank)

  Dumps the selected ROM bank.
- void Clear ()

  Clears the ROM.

Properties

- byte[ ][ ] Memory `[get, private set]`

  The ROM.
- byte Banks `[get, private set]`

  The total number of banks on the ROM.
- byte CurrentBank `[get, private set]`

  The bank the ROM is currently using.
- int Offset `[get, private set]`

  The memory offset
- int End `[get]`

  The end of memory
- int Length `[get, private set]`

  The memory length
- W65C02 Processor `[get, private set]`

  The processor reference

6.3.1 Detailed Description

An implementation of a W65C02 Processor.

Definition at line 10 of file AT28CXX.cs.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 AT28CXX() `Hardware.AT28CXX.AT28CXX (`

```
int offset,
int length,
byte banks )  [inline]
```

Default Constructor, Instantiates a new instance of the processor.

Definition at line 54 of file AT28CXX.cs.

```
        {
            Memory = new byte[banks][];
            for (int i = 0; i < banks; i++)
            {
                Memory[i] = new byte[length + 1];
            }
            Offset = offset;
            Length = length;
            Banks = banks;
            CurrentBank = 0;
        }
```

6.3.3 Member Function Documentation

6.3.3.1 Clear() `void Hardware.AT28CXX.Clear ( ) [inline]`

Clears the ROM.

Definition at line 165 of file AT28CXX.cs.

```
        {
            for (byte i = 0; i < Banks; i++)
            {
                for (int j = 0; j < Length; j++)
                {
                    Memory[i][j] = 0x00;
                }
            }
        }
```

6.3.3.2 DumpMemory() [1/2] `byte[ ][ ] Hardware.AT28CXX.DumpMemory ( ) [inline]`

Dumps the entire memory object. Used when saving the memory state

Returns

2 dimensional array of data analogous to the ROM of the computer.

Definition at line 143 of file AT28CXX.cs.

```
        {
            return Memory;
        }
```

6.3.3.3 DumpMemory() [2/2] `byte[ ] Hardware.AT28CXX.DumpMemory ( byte bank ) [inline]`

Dumps the selected ROM bank.

Parameters

bank	The bank to dump data from.

Returns

Array that represents the selected ROM bank.

Definition at line 153 of file AT28CXX.cs.

```
        {
            byte[] _tempMemory = new byte[MemoryMap.BankedRom.Length + 1];
            for (var i = 0; i < MemoryMap.BankedRom.Length; i++) {
                _tempMemory[i] = Memory[bank][i];
            }
            return _tempMemory;
        }
```

6.3.3.4 Load() [1/2]

```
void Hardware.AT28CXX.Load (
        byte bank,
        byte[] data )  [inline]
```

Loads a program into ROM.

Parameters

bank	The bank to load data to.
data	The data to be loaded to ROM.

Definition at line 84 of file AT28CXX.cs.

```
        {
            for (int i = 0; i <= Length; i++)
            {
                Memory[bank][i] = data[i];
            }
        }
```

6.3.3.5 Load() [2/2] void Hardware.AT28CXX.Load (byte *data[ ][ ]*) [inline]

Loads a program into ROM.

Parameters

data	The program to be loaded

Definition at line 71 of file AT28CXX.cs.

```
        {
            for (byte i = 0; i < Banks; i++)
            {
                Load(i, data[i]);
            }
        }
```

6.3.3.6 Read() byte Hardware.AT28CXX.Read (int *address*) [inline]

Returns the byte at a given address without incrementing the cycle. Useful for test harness.

Parameters

bank	The bank to read data from.
address	

Returns

the byte being returned

Definition at line 121 of file AT28CXX.cs.

```
        {
            return Memory[CurrentBank][address - Offset];
        }
```

6.3.3.7 ReadFile() `byte[][] Hardware.AT28CXX.ReadFile ( string filename ) [inline]`

Definition at line 92 of file AT28CXX.cs.

```
        {
            byte[][] bios = new byte[Banks][];
            try
            {
                FileStream file = new FileStream(filename, FileMode.Open, FileAccess.Read);
                for (int i = 0; i < Banks; i++)
                {
                    bios[i] = new byte[Length + 1];
                    for (int j = 0; j <= Length; j++)
                    {
                        bios[i][j] = new byte();
                        bios[i][j] = (byte)file.ReadByte();
                    }
                }
            }
            catch (Exception)
            {
                return null;
            }
            return bios;
        }
```

6.3.3.8 Write() `void Hardware.AT28CXX.Write ( int address, byte data ) [inline]`

Writes data to the given address without incrementing the cycle.

Parameters

bank	The bank to load data to.
address	The address to write data to
data	The data to write

Definition at line 132 of file AT28CXX.cs.

```
        {
            _ = address;
            _ = data;
            return;
        }
```

6.3.4 Property Documentation

6.3.4.1 Banks `byte Hardware.AT28CXX.Banks [get], [private set]`

The total number of banks on the ROM.

Definition at line 22 of file AT28CXX.cs.

```
{ get; private set; }
```

6.3.4.2 CurrentBank `byte Hardware.AT28CXX.CurrentBank [get], [private set]`

The bank the ROM is currently using.

Definition at line 27 of file AT28CXX.cs.
```
00027 { get; private set; }
```

6.3.4.3 End `int Hardware.AT28CXX.End [get]`

The end of memory

Definition at line 37 of file AT28CXX.cs.
```
00037 { get { return Offset + Length; } }
```

6.3.4.4 Length `int Hardware.AT28CXX.Length [get], [private set]`

The memory length

Definition at line 42 of file AT28CXX.cs.
```
00042 { get; private set; }
```

6.3.4.5 Memory `byte [][] Hardware.AT28CXX.Memory [get], [private set]`

The ROM.

Definition at line 17 of file AT28CXX.cs.
```
00017 { get; private set; }
```

6.3.4.6 Offset `int Hardware.AT28CXX.Offset [get], [private set]`

The memory offset

Definition at line 32 of file AT28CXX.cs.
```
00032 { get; private set; }
```

6.3.4.7 Processor `W65C02 Hardware.AT28CXX.Processor [get], [private set]`

The processor reference

Definition at line 47 of file AT28CXX.cs.
```
00047 { get; private set; }
```

The documentation for this class was generated from the following file:

- Hardware/AT28CXX.cs

6.4 Hardware.MemoryMap.BankedRam Class Reference

Static Public Attributes

- static int TotalLength = (BankSize ∗ TotalBanks) - 1
- static int BankSize = (int)(Length + 1)
- static byte TotalBanks = 16

Properties

- static int Offset `[get]`
- static int Length `[get]`

Static Private Attributes

- static int _Offset = 0x0000
- static int _Length = 0x7FFF

6.4.1 Detailed Description

Definition at line 8 of file MemoryMap.cs.

6.4.2 Member Data Documentation

6.4.2.1 _Length `int Hardware.MemoryMap.BankedRam._Length = 0x7FFF [static], [private]`

Definition at line 11 of file MemoryMap.cs.

6.4.2.2 _Offset `int Hardware.MemoryMap.BankedRam._Offset = 0x0000 [static], [private]`

Definition at line 10 of file MemoryMap.cs.

6.4.2.3 BankSize `int Hardware.MemoryMap.BankedRam.BankSize = (int)(Length + 1) [static]`

Definition at line 14 of file MemoryMap.cs.

6.4.2.4 TotalBanks `byte Hardware.MemoryMap.BankedRam.TotalBanks = 16 [static]`

Definition at line 15 of file MemoryMap.cs.

6.4.2.5 TotalLength `int Hardware.MemoryMap.BankedRam.TotalLength = (BankSize * TotalBanks) - 1 [static]`

Definition at line 13 of file MemoryMap.cs.

6.4.3 Property Documentation

6.4.3.1 Length `int Hardware.MemoryMap.BankedRam.Length [static], [get]`

Definition at line 18 of file MemoryMap.cs.

```
00018 { get { return _Length; } }
```

6.4.3.2 Offset `int Hardware.MemoryMap.BankedRam.Offset [static], [get]`

Definition at line 17 of file MemoryMap.cs.

```
00017 { get { return _Offset; } }
```

The documentation for this class was generated from the following file:

- Hardware/Classes/MemoryMap.cs

6.5 Hardware.MemoryMap.BankedRom Class Reference

Static Public Attributes

- static byte TotalBanks = 16

Properties

- static int Offset `[get]`
- static int Length `[get]`

Static Private Attributes

- static int _Offset = 0x8000
- static int _Length = 0x3FFF

6.5.1 Detailed Description

Definition at line 34 of file MemoryMap.cs.

6.5.2 Member Data Documentation

6.5.2.1 _Length `int Hardware.MemoryMap.BankedRom._Length = 0x3FFF [static], [private]`

Definition at line 37 of file MemoryMap.cs.

6.5.2.2 _Offset `int Hardware.MemoryMap.BankedRom._Offset = 0x8000 [static], [private]`

Definition at line 36 of file MemoryMap.cs.

6.5.2.3 TotalBanks `byte Hardware.MemoryMap.BankedRom.TotalBanks = 16 [static]`

Definition at line 39 of file MemoryMap.cs.

6.5.3 Property Documentation

6.5.3.1 Length `int Hardware.MemoryMap.BankedRom.Length [static], [get]`

Definition at line 42 of file MemoryMap.cs.

```
00042 { get { return _Length; } }
```

6.5.3.2 Offset `int Hardware.MemoryMap.BankedRom.Offset [static], [get]`

Definition at line 41 of file MemoryMap.cs.

```
00041 { get { return _Offset; } }
```

The documentation for this class was generated from the following file:

- Hardware/Classes/MemoryMap.cs

6.6 Emulator.Model.Breakpoint Class Reference

A Representation of a Breakpoint

Properties

- bool IsEnabled `[get, set]`
  Is the Breakpoint enabled or disabled
- string Value `[get, set]`
  The Value of the Breakpoint
- string Type `[get, set]`
  The Type of breakpoint being set
- List< string > AllTypes `[get]`

6.6.1 Detailed Description

A Representation of a Breakpoint

Definition at line 8 of file Breakpoint.cs.

6.6.2 Property Documentation

6.6.2.1 AllTypes `List<string> Emulator.Model.Breakpoint.AllTypes [get]`

Definition at line 25 of file Breakpoint.cs.

```
        {
            get { return BreakpointType.AllTypes; }
        }
```

6.6.2.2 IsEnabled `bool Emulator.Model.Breakpoint.IsEnabled [get], [set]`

Is the Breakpoint enabled or disabled

Definition at line 13 of file Breakpoint.cs.

```
{ get; set; }
```

6.6.2.3 Type `string Emulator.Model.Breakpoint.Type [get], [set]`

The Type of breakpoint being set

Definition at line 23 of file Breakpoint.cs.

```
{ get; set; }
```

6.6.2.4 Value `string Emulator.Model.Breakpoint.Value [get], [set]`

The Value of the Breakpoint

Definition at line 18 of file Breakpoint.cs.

```
00018 { get; set; }
```

The documentation for this class was generated from the following file:

- Emulator/Model/Breakpoint.cs

6.7 Emulator.Model.BreakpointType Class Reference

The Type of Breakpoint

Static Public Attributes

- static List< string > AllTypes

  A Listing of all of the Current Types
- const string ProgramCounterType = "Program Counter"

  The ProgamCounter Breakpoint Type
- const string NumberOfCycleType = "Number of Cycles"

  The CycleCount Breakpoint Type

6.7.1 Detailed Description

The Type of Breakpoint

Definition at line 8 of file BreakpointType.cs.

6.7.2 Member Data Documentation

6.7.2.1 AllTypes `List<string> Emulator.Model.BreakpointType.AllTypes [static]`

Initial value:

```
= new List<string>
            {
                ProgramCounterType,
                NumberOfCycleType
            }
```

A Listing of all of the Current Types

Definition at line 13 of file BreakpointType.cs.

6.7.2.2 NumberOfCycleType `const string Emulator.Model.BreakpointType.NumberOfCycleType = "Number of Cycles" [static]`

The CycleCount Breakpoint Type

Definition at line 27 of file BreakpointType.cs.

6.7.2.3 ProgramCounterType `const string Emulator.Model.BreakpointType.ProgramCounterType = "Program Counter" [static]`

The ProgamCounter Breakpoint Type

Definition at line 22 of file BreakpointType.cs.

The documentation for this class was generated from the following file:

- Emulator/Model/BreakpointType.cs

6.8 Hardware.MemoryMap.DeviceArea Class Reference

Properties

- static int End `[get]`

  The end of memory
- static int Offset `[get]`
- static int Length `[get]`

Static Private Attributes

- static int _Offset = 0xD000
- static int _Length = 0x00FF

6.8.1 Detailed Description

Definition at line 21 of file MemoryMap.cs.

6.8.2 Member Data Documentation

6.8.2.1 _Length `int Hardware.MemoryMap.DeviceArea._Length = 0x00FF [static], [private]`

Definition at line 24 of file MemoryMap.cs.

6.8.2.2 _Offset `int Hardware.MemoryMap.DeviceArea._Offset = 0xD000 [static], [private]`

Definition at line 23 of file MemoryMap.cs.

6.8.3 Property Documentation

6.8.3.1 End `int Hardware.MemoryMap.DeviceArea.End [static], [get]`

The end of memory

Definition at line 29 of file MemoryMap.cs.

```
00029 { get { return Offset + Length; } }
```

6.8.3.2 Length `int Hardware.MemoryMap.DeviceArea.Length [static], [get]`

Definition at line 31 of file MemoryMap.cs.

```
00031 { get { return _Length; } }
```

6.8.3.3 Offset `int Hardware.MemoryMap.DeviceArea.Offset [static], [get]`

Definition at line 30 of file MemoryMap.cs.

```
00030 { get { return _Offset; } }
```

The documentation for this class was generated from the following file:

- Hardware/Classes/MemoryMap.cs

6.9 Hardware.MemoryMap.Devices Class Reference

Classes

- class ACIA
- class GPIO
- class MM65SIB

6.9.1 Detailed Description

Definition at line 56 of file MemoryMap.cs.

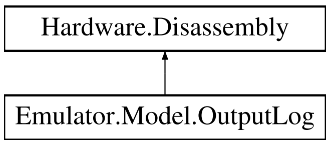
The documentation for this class was generated from the following file:

- Hardware/Classes/MemoryMap.cs

6.10 Hardware.Disassembly Class Reference

Used to help simulating. This class contains the disassembly properties.

Inheritance diagram for Hardware.Disassembly:



Properties

- string LowAddress `[get, set]`

  The low Address
- string HighAddress `[get, set]`

  The High Address
- string OpCodeString `[get, set]`

  The string representation of the OpCode
- string DisassemblyOutput `[get, set]`

  The disassembly of the current step

6.10.1 Detailed Description

Used to help simulating. This class contains the disassembly properties.

Definition at line 9 of file Disassembly.cs.

6.10.2 Property Documentation

6.10.2.1 DisassemblyOutput `string Hardware.Disassembly.DisassemblyOutput [get], [set]`

The disassembly of the current step

Definition at line 29 of file Disassembly.cs.
```
00029 { get; set; }
```

6.10.2.2 HighAddress `string Hardware.Disassembly.HighAddress [get], [set]`

The High Address

Definition at line 19 of file Disassembly.cs.
```
00019 { get; set; }
```

6.10.2.3 LowAddress `string Hardware.Disassembly.LowAddress [get], [set]`

The low Address

Definition at line 14 of file Disassembly.cs.

```
00014 { get; set; }
```

6.10.2.4 OpCodeString `string Hardware.Disassembly.OpCodeString [get], [set]`

The string representation of the OpCode

Definition at line 24 of file Disassembly.cs.

```
00024 { get; set; }
```

The documentation for this class was generated from the following file:

- Hardware/Classes/Disassembly.cs

6.11 Emulator.ExitCodes Class Reference

Static Public Attributes

- static readonly int NO_ERROR = 0x00
- static readonly int USER_ERROR = 0x01
- static readonly int NO_BIOS = 0x02
- static readonly int LOAD_BIOS_FILE_ERROR = 0x03
- static readonly int BIOS_LOADPROGRAM_ERROR = 0x04
- static readonly int LOAD_ROM_FILE_ERROR = 0x05
- static readonly int ROM_LOADPROGRAM_ERROR = 0x06
- static readonly int LOAD_STATE_ERROR = 0x07

6.11.1 Detailed Description

Definition at line 9 of file ExitCodes.cs.

6.11.2 Member Data Documentation

6.11.2.1 BIOS_LOADPROGRAM_ERROR `readonly int Emulator.ExitCodes.BIOS_LOADPROGRAM_ERROR = 0x04 [static]`

Definition at line 17 of file ExitCodes.cs.

6.11.2.2 LOAD_BIOS_FILE_ERROR `readonly int Emulator.ExitCodes.LOAD_BIOS_FILE_ERROR = 0x03 [static]`

Definition at line 16 of file ExitCodes.cs.

6.11.2.3 LOAD_ROM_FILE_ERROR `readonly int Emulator.ExitCodes.LOAD_ROM_FILE_ERROR = 0x05 [static]`

Definition at line 18 of file ExitCodes.cs.

6.11.2.4 LOAD_STATE_ERROR `readonly int Emulator.ExitCodes.LOAD_STATE_ERROR = 0x07 [static]`

Definition at line 20 of file ExitCodes.cs.

6.11.2.5 NO_BIOS `readonly int Emulator.ExitCodes.NO_BIOS = 0x02 [static]`

Definition at line 15 of file ExitCodes.cs.

6.11.2.6 NO_ERROR `readonly int Emulator.ExitCodes.NO_ERROR = 0x00 [static]`

Definition at line 11 of file ExitCodes.cs.

6.11.2.7 ROM_LOADPROGRAM_ERROR `readonly int Emulator.ExitCodes.ROM_LOADPROGRAM_ERROR = 0x06 [static]`

Definition at line 19 of file ExitCodes.cs.

6.11.2.8 USER_ERROR `readonly int Emulator.ExitCodes.USER_ERROR = 0x01 [static]`

Definition at line 13 of file ExitCodes.cs.

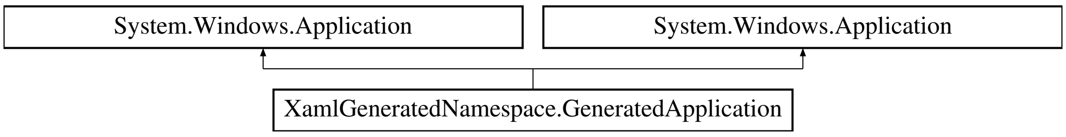
The documentation for this class was generated from the following file:

- Emulator/Classes/ExitCodes.cs

6.12 XamlGeneratedNamespace.GeneratedApplication Class Reference

GeneratedApplication

Inheritance diagram for XamlGeneratedNamespace.GeneratedApplication:


System.Windows.Application
System.Windows.Application
XamlGeneratedNamespace.GeneratedApplication

Public Member Functions

- void InitializeComponent ()

  InitializeComponent
- void InitializeComponent ()

  InitializeComponent

Static Public Member Functions

- static void Main ()

  Application Entry Point.
- static void Main ()

  Application Entry Point.

Private Attributes

- bool _contentLoaded

6.12.1 Detailed Description

GeneratedApplication

Definition at line 41 of file App.g.cs.

6.12.2 Member Function Documentation

6.12.2.1 InitializeComponent() [1/2] void XamlGeneratedNamespace.GeneratedApplication.Initialize↩ Component ( ) [inline]

InitializeComponent

Definition at line 50 of file App.g.cs.
```
                                    {
            if (_contentLoaded) {
                return;
            }
            _contentLoaded = true;

#line 2 "..\..\..\App.xaml"
            this.StartupUri = new System.Uri("MainWindow.xaml", System.UriKind.Relative);

#line default
#line hidden
            System.Uri resourceLocater = new System.Uri("/Emulator;component/app.xaml",
System.UriKind.Relative);

#line 1 "..\..\..\App.xaml"
            System.Windows.Application.LoadComponent(this, resourceLocater);

#line default
#line hidden
        }
```

6.12.2.2 InitializeComponent() [2/2] void XamlGeneratedNamespace.GeneratedApplication.Initialize↩ Component ( ) [inline]

InitializeComponent

Definition at line 50 of file App.g.i.cs.
```
                                    {
            if (_contentLoaded) {
                return;
            }
            _contentLoaded = true;

#line 2 "..\..\..\App.xaml"
            this.StartupUri = new System.Uri("MainWindow.xaml", System.UriKind.Relative);

#line default
#line hidden
            System.Uri resourceLocater = new System.Uri("/Emulator;component/app.xaml",
System.UriKind.Relative);

#line 1 "..\..\..\App.xaml"
            System.Windows.Application.LoadComponent(this, resourceLocater);

#line default
#line hidden
        }
```

6.12.2.3 Main() [1/2] static void XamlGeneratedNamespace.GeneratedApplication.Main ( ) [inline], [static]

Application Entry Point.

Definition at line 76 of file App.g.cs.
```
                            {
            SplashScreen splashScreen = new SplashScreen("splashscreen.png");
            splashScreen.Show(true);
            XamlGeneratedNamespace.GeneratedApplication app = new
XamlGeneratedNamespace.GeneratedApplication();
            app.InitializeComponent();
            app.Run();
        }
```

6.12.2.4 Main() [2/2] `static void XamlGeneratedNamespace.GeneratedApplication.Main ( ) [inline], [static]`

Application Entry Point.

Definition at line 76 of file App.g.i.cs.

```
                          {
            SplashScreen splashScreen = new SplashScreen("splashscreen.png");
            splashScreen.Show(true);
            XamlGeneratedNamespace.GeneratedApplication app = new
      XamlGeneratedNamespace.GeneratedApplication();
            app.InitializeComponent();
            app.Run();
        }
```

6.12.3 Member Data Documentation

6.12.3.1 _contentLoaded `bool XamlGeneratedNamespace.GeneratedApplication._contentLoaded [private]`

Definition at line 43 of file App.g.cs.

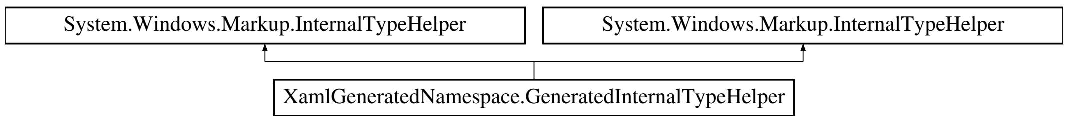
The documentation for this class was generated from the following files:

- Emulator/obj/x86/Debug/App.g.cs
- Emulator/obj/x86/Debug/App.g.i.cs

6.13 XamlGeneratedNamespace.GeneratedInternalTypeHelper Class Reference

GeneratedInternalTypeHelper

Inheritance diagram for XamlGeneratedNamespace.GeneratedInternalTypeHelper:


System.Windows.Markup.InternalTypeHelper
System.Windows.Markup.InternalTypeHelper
XamlGeneratedNamespace.GeneratedInternalTypeHelper

Protected Member Functions

- override object CreateInstance (System.Type type, System.Globalization.CultureInfo culture)

  CreateInstance
- override object GetPropertyValue (System.Reflection.PropertyInfo propertyInfo, object target, System.↩ Globalization.CultureInfo culture)

  GetPropertyValue
- override void SetPropertyValue (System.Reflection.PropertyInfo propertyInfo, object target, object value, System.Globalization.CultureInfo culture)

  SetPropertyValue
- override System.Delegate CreateDelegate (System.Type delegateType, object target, string handler)

  CreateDelegate

- override void AddEventHandler (System.Reflection.EventInfo eventInfo, object target, System.Delegate handler)

  AddEventHandler
- override object CreateInstance (System.Type type, System.Globalization.CultureInfo culture)

  CreateInstance
- override object GetPropertyValue (System.Reflection.PropertyInfo propertyInfo, object target, System.↩ Globalization.CultureInfo culture)

  GetPropertyValue
- override void SetPropertyValue (System.Reflection.PropertyInfo propertyInfo, object target, object value, System.Globalization.CultureInfo culture)

  SetPropertyValue
- override System.Delegate CreateDelegate (System.Type delegateType, object target, string handler)

  CreateDelegate
- override void AddEventHandler (System.Reflection.EventInfo eventInfo, object target, System.Delegate handler)

  AddEventHandler

6.13.1 Detailed Description

GeneratedInternalTypeHelper

Definition at line 20 of file GeneratedInternalTypeHelper.g.cs.

6.13.2 Member Function Documentation

6.13.2.1 AddEventHandler() `[1/2]` `override void XamlGeneratedNamespace.GeneratedInternalType↩ Helper.AddEventHandler (`

```
            System.Reflection.EventInfo eventInfo,
            object target,
            System.Delegate handler )  [inline], [protected]
```

AddEventHandler

Definition at line 57 of file GeneratedInternalTypeHelper.g.cs.

```
        {
            eventInfo.AddEventHandler(target, handler);
        }
```

6.13.2.2 AddEventHandler() `[2/2]` `override void XamlGeneratedNamespace.GeneratedInternalType↩ Helper.AddEventHandler (`

```
            System.Reflection.EventInfo eventInfo,
            object target,
            System.Delegate handler )  [inline], [protected]
```

AddEventHandler

Definition at line 57 of file GeneratedInternalTypeHelper.g.i.cs.

```
        {
            eventInfo.AddEventHandler(target, handler);
        }
```

6.13.2.3 CreateDelegate() [1/2] `override System.Delegate XamlGeneratedNamespace.Generated↩ InternalTypeHelper.CreateDelegate ( System.Type delegateType, object target, string handler ) [inline], [protected]`

CreateDelegate

Definition at line 47 of file GeneratedInternalTypeHelper.g.cs.

```
{
            return ((System.Delegate)(target.GetType().InvokeMember("_CreateDelegate",
(System.Reflection.BindingFlags.InvokeMethod
                                | (System.Reflection.BindingFlags.NonPublic |
System.Reflection.BindingFlags.Instance)), null, target, new object[] {
                            delegateType,
                            handler}, null)));
        }
```

6.13.2.4 CreateDelegate() [2/2] `override System.Delegate XamlGeneratedNamespace.Generated↩ InternalTypeHelper.CreateDelegate ( System.Type delegateType, object target, string handler ) [inline], [protected]`

CreateDelegate

Definition at line 47 of file GeneratedInternalTypeHelper.g.i.cs.

```
{
            return ((System.Delegate)(target.GetType().InvokeMember("_CreateDelegate",
(System.Reflection.BindingFlags.InvokeMethod
                                | (System.Reflection.BindingFlags.NonPublic |
System.Reflection.BindingFlags.Instance)), null, target, new object[] {
                            delegateType,
                            handler}, null)));
        }
```

6.13.2.5 CreateInstance() [1/2] `override object XamlGeneratedNamespace.GeneratedInternalType↩ Helper.CreateInstance ( System.Type type, System.Globalization.CultureInfo culture ) [inline], [protected]`

CreateInstance

Definition at line 25 of file GeneratedInternalTypeHelper.g.cs.

```
{
            return System.Activator.CreateInstance(type, ((System.Reflection.BindingFlags.Public |
System.Reflection.BindingFlags.NonPublic)
                                | (System.Reflection.BindingFlags.Instance |
System.Reflection.BindingFlags.CreateInstance)), null, null, culture);
        }
```

6.13.2.6 CreateInstance() [2/2] override object XamlGeneratedNamespace.GeneratedInternalType←Helper.CreateInstance (System.Type *type*, System.Globalization.CultureInfo *culture*) [inline], [protected]

CreateInstance

Definition at line 25 of file GeneratedInternalTypeHelper.g.i.cs.

```
{
            return System.Activator.CreateInstance(type, ((System.Reflection.BindingFlags.Public |
System.Reflection.BindingFlags.NonPublic)
                             | (System.Reflection.BindingFlags.Instance |
System.Reflection.BindingFlags.CreateInstance)), null, null, culture);
        }
```

6.13.2.7 GetPropertyValue() [1/2] override object XamlGeneratedNamespace.GeneratedInternalType←Helper.GetPropertyValue (System.Reflection.PropertyInfo *propertyInfo*, object *target*, System.Globalization.CultureInfo *culture*) [inline], [protected]

GetPropertyValue

Definition at line 33 of file GeneratedInternalTypeHelper.g.cs.

```
{
            return propertyInfo.GetValue(target, System.Reflection.BindingFlags.Default, null, null,
culture);
        }
```

6.13.2.8 GetPropertyValue() [2/2] override object XamlGeneratedNamespace.GeneratedInternalType←Helper.GetPropertyValue (System.Reflection.PropertyInfo *propertyInfo*, object *target*, System.Globalization.CultureInfo *culture*) [inline], [protected]

GetPropertyValue

Definition at line 33 of file GeneratedInternalTypeHelper.g.i.cs.

```
{
            return propertyInfo.GetValue(target, System.Reflection.BindingFlags.Default, null, null,
culture);
        }
```

6.13.2.9 SetPropertyValue() [1/2] override void XamlGeneratedNamespace.GeneratedInternalType←Helper.SetPropertyValue (System.Reflection.PropertyInfo *propertyInfo*, object *target*, object *value*, System.Globalization.CultureInfo *culture*) [inline], [protected]

SetPropertyValue

Definition at line 40 of file GeneratedInternalTypeHelper.g.cs.

```
{
            propertyInfo.SetValue(target, value, System.Reflection.BindingFlags.Default, null, null,
culture);
        }
```

6.13.2.10 SetPropertyValue() [2/2]

```
override void XamlGeneratedNamespace.GeneratedInternalType←
Helper.SetPropertyValue (
            System.Reflection.PropertyInfo propertyInfo,
            object target,
            object value,
            System.Globalization.CultureInfo culture )  [inline], [protected]
```

SetPropertyValue

Definition at line 40 of file GeneratedInternalTypeHelper.g.i.cs.

```
     {
            propertyInfo.SetValue(target, value, System.Reflection.BindingFlags.Default, null, null,
     culture);
        }
```

The documentation for this class was generated from the following files:

- Emulator/obj/x86/Debug/GeneratedInternalTypeHelper.g.cs
- Emulator/obj/x86/Debug/GeneratedInternalTypeHelper.g.i.cs

6.14 Hardware.MemoryMap.Devices.GPIO Class Reference

Static Public Attributes

- static int Length = 0x0F
- static byte Offset = 0x20

6.14.1 Detailed Description

Definition at line 64 of file MemoryMap.cs.

6.14.2 Member Data Documentation

6.14.2.1 Length

```
int Hardware.MemoryMap.Devices.GPIO.Length = 0x0F  [static]
```

Definition at line 66 of file MemoryMap.cs.

6.14.2.2 Offset

```
byte Hardware.MemoryMap.Devices.GPIO.Offset = 0x20  [static]
```

Definition at line 67 of file MemoryMap.cs.

The documentation for this class was generated from the following file:

- Hardware/Classes/MemoryMap.cs

6.15 Hardware.HM62256 Class Reference

Public Member Functions

- HM62256 (byte banks, int offset, int length)

  Called whenever a new 62256 object is required.
- void Reset ()

  Called whenever the emulated computer is reset.
- void Clear ()

  Clears the memory.
- byte Read (int address)

  Returns the byte at a given address without incrementing the cycle. Useful for test harness.
- void Write (int address, byte data)

  Writes data to the given address without incrementing the cycle.
- byte[ ][ ] DumpMemory ()

  Dumps the entire memory object. Used when saving the memory state

Properties

- byte[ ][ ] Memory `[get, set]`

  The memory area.
- int Offset `[get, set]`

  The memory offset.
- int Length `[get, set]`

  The memory length.
- int End `[get]`

  The location of the end of memory.
- byte Banks `[get, set]`

  The number of banks the memory has.
- byte CurrentBank `[get, set]`

  The currently selected bank.

6.15.1 Detailed Description

Definition at line 5 of file HM62256.cs.

6.15.2 Constructor & Destructor Documentation

6.15.2.1 HM62256()

```
Hardware.HM62256.HM62256 (
        byte banks,
        int offset,
        int length )  [inline]
```

Called whenever a new 62256 object is required.

Parameters

banks	Number of banks the new memory will have.
offset	Offset of the new memory in the address space.
length	Length of each bank of memory.

Definition at line 43 of file HM62256.cs.

```
        {
            Memory = new byte[banks][];
            for (int i = 0; i < banks; i++)
            {
                Memory[i] = new byte[length + 1];
            }
            Length = length;
            Banks = banks;
            Offset = offset;
            CurrentBank = 0;
        }
```

6.15.3 Member Function Documentation

6.15.3.1 Clear() `void Hardware.HM62256.Clear ( ) [inline]`

Clears the memory.

Definition at line 67 of file HM62256.cs.

```
        {
            for (var i = 0; i < Banks; i++)
            {
                for (var j = 0; j < Memory.Length; j++)
                {
                    Memory[i][j] = 0x00;
                }
            }
        }
```

6.15.3.2 DumpMemory() `byte[ ][ ] Hardware.HM62256.DumpMemory ( ) [inline]`

Dumps the entire memory object. Used when saving the memory state

Returns

Jagged array representing the banked memory.

Definition at line 104 of file HM62256.cs.

```
        {
            return Memory;
        }
```

6.15.3.3 Read() `byte Hardware.HM62256.Read ( int address ) [inline]`

Returns the byte at a given address without incrementing the cycle. Useful for test harness.

Parameters

bank	The bank to read data from.
address	

Returns

The byte being read.

Definition at line 84 of file HM62256.cs.
```
        {
            return Memory[CurrentBank][address - Offset];
        }
```

6.15.3.4 Reset() `void Hardware.HM62256.Reset ( ) [inline]`

Called whenever the emulated computer is reset.

Definition at line 59 of file HM62256.cs.
```
        {
            Clear();
        }
```

6.15.3.5 Write() `void Hardware.HM62256.Write ( int address, byte data ) [inline]`

Writes data to the given address without incrementing the cycle.

Parameters

bank	The bank to load data to.
address	The address to write data to
data	The data to write

Definition at line 95 of file HM62256.cs.
```
        {
            Memory[CurrentBank][address - Offset] = data;
        }
```

6.15.4 Property Documentation

6.15.4.1 Banks `byte Hardware.HM62256.Banks [get], [set]`

The number of banks the memory has.

Definition at line 30 of file HM62256.cs.
```
{ get; set; }
```

6.15.4.2 CurrentBank `byte Hardware.HM62256.CurrentBank [get], [set]`

The currently selected bank.

Definition at line 35 of file HM62256.cs.

```
00035 { get; set; }
```

6.15.4.3 End `int Hardware.HM62256.End [get]`

The location of the end of memory.

Definition at line 25 of file HM62256.cs.

```
00025 { get { return Offset + Length; } }
```

6.15.4.4 Length `int Hardware.HM62256.Length [get], [set]`

The memory length.

Definition at line 20 of file HM62256.cs.

```
00020 { get; set; }
```

6.15.4.5 Memory `byte [][] Hardware.HM62256.Memory [get], [set]`

The memory area.

Definition at line 10 of file HM62256.cs.

```
00010 { get; set; }
```

6.15.4.6 Offset `int Hardware.HM62256.Offset [get], [set]`

The memory offset.

Definition at line 15 of file HM62256.cs.

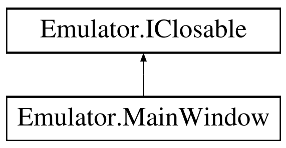
```
00015 { get; set; }
```

The documentation for this class was generated from the following file:

- Hardware/HM62256.cs

6.16 Emulator.IClosable Interface Reference

Inheritance diagram for Emulator.IClosable:



Public Member Functions

- void Close ()

6.16.1 Detailed Description

Definition at line 9 of file IClosable.cs.

6.16.2 Member Function Documentation

6.16.2.1 Close() `void Emulator.IClosable.Close ( )`

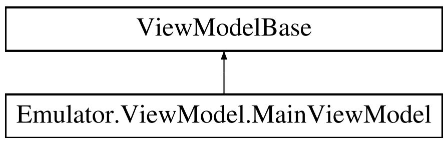
The documentation for this interface was generated from the following file:

- Emulator/Interfaces/IClosable.cs

6.17 Emulator.ViewModel.MainViewModel Class Reference

The Main ViewModel

Inheritance diagram for Emulator.ViewModel.MainViewModel:



Public Member Functions

- MainViewModel ()

    Creates a new Instance of the MainViewModel.
- void OnLoad (Object sender, RoutedEventArgs e)
- void OnClose (Object sender, CancelEventArgs e)

Properties

- HM62256 HM62256 `[get, set]`

  The 62256 RAM.
- W65C02 W65C02 `[get, private set]`

  The 65C02 Processor.
- W65C22 W65C22 `[get, private set]`

  General Purpose I/O, Shift Registers and Timers.
- W65C22 MM65SIB `[get, private set]`

  Memory management and 65SIB.
- W65C51 W65C51 `[get, private set]`

  The ACIA serial interface.
- AT28CXX AT28C64 `[get, private set]`

  The AT28C010 ROM.
- AT28CXX AT28C010 `[get, private set]`

  The AT28C010 ROM.
- MultiThreadedObservableCollection< MemoryRowModel > MemoryPage `[get, set]`

  The Current Memory Page
- MultiThreadedObservableCollection< OutputLog > OutputLog `[get, private set]`

  The output log
- MultiThreadedObservableCollection< Breakpoint > Breakpoints `[get, set]`

  The Breakpoints
- Breakpoint SelectedBreakpoint `[get, set]`

  The Currently Selected Breakpoint
- RomFileModel RomFile `[get, set]`

  The currently loaded binary file. (If it is indeed loaded, that is.)
- string CurrentDisassembly `[get]`

  The Current Disassembly
- int NumberOfCycles `[get, private set]`

  The number of cycles.
- string MemoryPageOffset `[get, set]`

  The Memory Page number.
- bool IsRunning `[get, set]`

  Is the Prorgam Running
- bool IsRomLoaded `[get, set]`

  Is the banked ROM Loaded.
- int CpuSpeed `[get, set]`

  The Slider CPU Speed
- static SettingsModel SettingsModel `[get, set]`

  The Model used for saving, loading and using data from Settings.xml
- RelayCommand StepCommand `[get, set]`

  RelayCommand for Stepping through the progam one instruction at a time.
- RelayCommand ResetCommand `[get, set]`

  Relay Command to Reset the Program back to its initial state.
- RelayCommand RunPauseCommand `[get, set]`

  Relay Command that Run/Pauses Execution
- RelayCommand UpdateMemoryMapCommand `[get, set]`

  Relay Command that updates the Memory Map when the Page changes
- RelayCommand AddBreakPointCommand `[get, set]`

  The Relay Command that adds a new breakpoint
- RelayCommand AboutCommand `[get, set]`

The Relay Command that opens the About window.

- RelayCommand RemoveBreakPointCommand `[get, set]`

  The Relay Command that Removes an existing breakpoint.
- RelayCommand SettingsCommand `[get, set]`

  The Command that loads or saves the settings.
- RelayCommand< IClosable > CloseCommand `[get, private set]`

  The Command that loads or saves the settings.
- string CurrentSerialPort `[get]`

  The current serial port object name.
- string WindowTitle `[get]`

  The title for the main window.

Private Member Functions

- void Close (IClosable window)
- void BinaryLoadedNotification (NotificationMessage< RomFileModel > notificationMessage)
- void StateLoadedNotifcation (NotificationMessage< StateFileModel > notificationMessage)
- void GenericNotifcation (NotificationMessage notificationMessage)
- void SettingsAppliedNotifcation (NotificationMessage< SettingsModel > notificationMessage)
- void UpdateMemoryPage ()
- void Reset ()
- void Step ()
- void UpdateUi ()
- void StepProcessor ()
- OutputLog GetOutputLog ()
- void RunPause ()
- void BackgroundWorkerDoWork (object sender, DoWorkEventArgs e)
- bool IsBreakPointTriggered ()
- int GetLogModValue ()
- int GetSleepValue ()
- void About ()
- void Settings ()
- void AddBreakPoint ()
- void RemoveBreakPoint ()

Private Attributes

- int _memoryPageOffset
- readonly BackgroundWorker _backgroundWorker
- bool _breakpointTriggered

6.17.1 Detailed Description

The Main ViewModel

Definition at line 27 of file MainViewModel.cs.

6.17.2 Constructor & Destructor Documentation

6.17.2.1 MainViewModel() `Emulator.ViewModel.MainViewModel.MainViewModel ( ) [inline]`

Creates a new Instance of the MainViewModel.

Definition at line 231 of file MainViewModel.cs.

```
        {
            var _formatter = new XmlSerializer(typeof(SettingsModel));
            Stream _stream = new FileStream(FileLocations.SettingsFile, FileMode.OpenOrCreate);
            if (!((_stream == null) || (0 >= _stream.Length)))
            {
                SettingsModel = (SettingsModel)_formatter.Deserialize(_stream);
                if ((SettingsModel.SettingsVersionMajor < Versioning.SettingsFile.Major) ||
                    (SettingsModel.SettingsVersionMinor < Versioning.SettingsFile.Minor) ||
                    (SettingsModel.SettingsVersionBuild < Versioning.SettingsFile.Build) ||
                    (SettingsModel.SettingsVersionRevision < Versioning.SettingsFile.Revision))
                {
#if !DEBUG
                    throw new NotImplementedException(String.Format("Unable to handle problem: Settings File version is less than {0}.{1}.{2}.{3}", Versioning.SettingsFile.Major, Versioning.SettingsFile.Minor, Versioning.SettingsFile.Revision, Versioning.SettingsFile.Build));
#else
                    MessageBox.Show("Settings file contains old information...\nDeleting old settings file...",
                                    "Settings file stale!", MessageBoxButton.OKCancel, MessageBoxImage.Warning,
                                    MessageBoxResult.OK);
                    // Close the file, then delete it.
                    _stream.Close();
                    File.Delete(FileLocations.SettingsFile);
                    SettingsModel = SettingsFile.CreateNew();
#endif
                }
            }
            else
            {
                MessageBox.Show("Creating new settings file...");
                SettingsModel = SettingsFile.CreateNew();
            }
            _stream.Close();

            HM62256 = new HM62256(MemoryMap.BankedRam.TotalBanks, MemoryMap.BankedRam.Offset, MemoryMap.BankedRam.Length);
            AT28C64 = new AT28CXX(MemoryMap.SharedRom.Offset, MemoryMap.SharedRom.Length, 1);
            AT28C010 = new AT28CXX(MemoryMap.BankedRom.Offset, MemoryMap.BankedRom.Length, MemoryMap.BankedRom.TotalBanks);
            W65C02 = new W65C02();
            W65C51 = new W65C51(W65C02, MemoryMap.Devices.ACIA.Offset);
            W65C51.Init(SettingsModel.ComPortName.ToString());
            W65C22 = new W65C22(W65C02, MemoryMap.Devices.GPIO.Offset, MemoryMap.Devices.GPIO.Length);
            W65C22.Init(1000);
            MM65SIB = new W65C22(W65C02, MemoryMap.Devices.MM65SIB.Offset, MemoryMap.Devices.MM65SIB.Length);
            MM65SIB.Init(1000);

            MemoryMap.Init(W65C02, W65C22, MM65SIB, W65C51, HM62256, AT28C010, AT28C64);

            // Now we can load the BIOS.
            byte[][] _bios = AT28C64.ReadFile(FileLocations.BiosFile);
            if (_bios == null)
            {
                Environment.Exit(ExitCodes.NO_BIOS);
            }
            AT28C64.Load(_bios);

            AboutCommand = new RelayCommand(About);
            AddBreakPointCommand = new RelayCommand(AddBreakPoint);
            CloseCommand = new RelayCommand<IClosable>(Close);
            RemoveBreakPointCommand = new RelayCommand(RemoveBreakPoint);
            ResetCommand = new RelayCommand(Reset);
            RunPauseCommand = new RelayCommand(RunPause);
            SettingsCommand = new RelayCommand(Settings);
            StepCommand = new RelayCommand(Step);
            UpdateMemoryMapCommand = new RelayCommand(UpdateMemoryPage);

            Messenger.Default.Register<NotificationMessage>(this, GenericNotifcation);
            Messenger.Default.Register<NotificationMessage<RomFileModel»(this, BinaryLoadedNotification);
            Messenger.Default.Register<NotificationMessage<SettingsModel»(this, SettingsAppliedNotifcation);
            Messenger.Default.Register<NotificationMessage<StateFileModel»(this, StateLoadedNotifcation);

            MemoryPage = new MultiThreadedObservableCollection<MemoryRowModel>();
            OutputLog = new MultiThreadedObservableCollection<OutputLog>();
            Breakpoints = new MultiThreadedObservableCollection<Breakpoint>();

            UpdateMemoryPage();

            _backgroundWorker = new BackgroundWorker { WorkerSupportsCancellation = true,
WorkerReportsProgress = false };
            _backgroundWorker.DoWork += BackgroundWorkerDoWork;
            Application.Current.MainWindow.Closing += new CancelEventHandler(OnClose);
            Application.Current.MainWindow.Loaded += new RoutedEventHandler(OnLoad);

            Reset();
        }
```

6.17.3 Member Function Documentation

6.17.3.1 About() `void Emulator.ViewModel.MainViewModel.About ( ) [inline], [private]`

Definition at line 756 of file MainViewModel.cs.

```
        {
            IsRunning = false;

            if (_backgroundWorker.IsBusy)
                _backgroundWorker.CancelAsync();

            MessageBox.Show(string.Format("{0}\n{1}\nVersion:  {2}\nCompany:  {3}",
Versioning.Product.Name, Versioning.Product.Description, Versioning.Product.VersionString,
Versioning.Product.Company), Versioning.Product.Title);
        }
```

6.17.3.2 AddBreakPoint() `void Emulator.ViewModel.MainViewModel.AddBreakPoint ( ) [inline], [private]`

Definition at line 776 of file MainViewModel.cs.

```
        {
            Breakpoints.Add(new Breakpoint());
            RaisePropertyChanged("Breakpoints");
        }
```

6.17.3.3 BackgroundWorkerDoWork() `void Emulator.ViewModel.MainViewModel.BackgroundWorkerDoWork ( object sender, DoWorkEventArgs e ) [inline], [private]`

Definition at line 632 of file MainViewModel.cs.

```
        {
            var worker = sender as BackgroundWorker;
            var outputLogs = new List<OutputLog>();

            while (true)
            {
                if (worker != null && worker.CancellationPending || IsBreakPointTriggered())
                {
                    e.Cancel = true;

                    RaisePropertyChanged("W65C02");

                    foreach (var log in outputLogs)
                        OutputLog.Insert(0, log);

                    UpdateMemoryPage();
                    return;
                }
                StepProcessor();
                outputLogs.Add(GetOutputLog());

                if (NumberOfCycles % GetLogModValue() == 0)
                {
                    foreach (var log in outputLogs)
                        OutputLog.Insert(0, log);

                    outputLogs.Clear();
                    UpdateUi();
                }
                Thread.Sleep(GetSleepValue());
            }
        }
```

6.17.3.4 BinaryLoadedNotification() `void Emulator.ViewModel.MainViewModel.BinaryLoadedNotification ( NotificationMessage< RomFileModel > notificationMessage ) [inline], [private]`

Definition at line 374 of file MainViewModel.cs.

```
        {
            if (notificationMessage.Notification != "FileLoaded")
            {
                return;
            }

            // Load Banked ROM
            AT28C010.Load(notificationMessage.Content.Rom);
            IsRomLoaded = true;
            RaisePropertyChanged("IsRomLoaded");

            Reset();
        }
```

6.17.3.5 Close() `void Emulator.ViewModel.MainViewModel.Close ( IClosable window ) [inline], [private]`

Definition at line 366 of file MainViewModel.cs.

```
        {
            if ((window != null) && (!IsRunning))
            {
                Environment.Exit(ExitCodes.NO_ERROR);
            }
        }
```

6.17.3.6 GenericNotifcation() `void Emulator.ViewModel.MainViewModel.GenericNotifcation ( NotificationMessage notificationMessage ) [inline], [private]`

Definition at line 416 of file MainViewModel.cs.

```
        {
            if (notificationMessage.Notification == "CloseFile")
            {
                AT28C010.Clear();
                if (IsRunning) { RunPause(); }
                IsRomLoaded = false;
                RaisePropertyChanged("IsRomLoaded");
                return;
            }
            else if (notificationMessage.Notification == "LoadFile")
            {
                var dialog = new OpenFileDialog {   DefaultExt = ".bin", Filter =
                                                    "All Files (*.bin, *.65C02)|*.bin;*.65C02|Binary
      Assembly (*.bin)|" +
                                                     "*.bin|WolfNet 65C02 Emulator Save State
(*.65C02)|*.65C02" };
                var result = dialog.ShowDialog();
                if (result != true)
                {
                    return;
                }

                if (Path.GetExtension(dialog.FileName.ToUpper()) == ".BIN")
                {
                    byte[][] _rom = AT28C010.ReadFile(dialog.FileName);

                    Messenger.Default.Send(new NotificationMessage<RomFileModel>(new RomFileModel
                    {
                        Rom = _rom,
                        RomBanks = AT28C010.Banks,
                        RomBankSize = AT28C010.Length,
                        RomFilePath = dialog.FileName,
                        RomFileName = Path.GetFileName(dialog.FileName),
                    }, "FileLoaded"));
                }
                else if (Path.GetExtension(dialog.FileName.ToUpper()) == ".6502")
                {
                    var formatter = new BinaryFormatter();
                    Stream stream = new FileStream(dialog.FileName, FileMode.Open);
                    var fileModel = (StateFileModel)formatter.Deserialize(stream);

                    stream.Close();

                    Messenger.Default.Send(new NotificationMessage<StateFileModel>(fileModel,
"StateLoaded"));
                }
            }
            else if (notificationMessage.Notification == "SaveState")
            {
                var dialog = new SaveFileDialog {   DefaultExt = ".65C02", Filter =
                                                     "WolfNet W65C02 Emulator Save State
(*.65C02)|*.65C02" };
                var result = dialog.ShowDialog();

                if (result != true)
                {
                    return;
                }

                var formatter = new BinaryFormatter();
                Stream stream = new FileStream(dialog.FileName, FileMode.Create, FileAccess.Write,
FileShare.None);

                formatter.Serialize(stream, new StateFileModel
                {
                    NumberOfCycles = NumberOfCycles,
                    OutputLog = OutputLog,
                    W65C02 = W65C02,
                    W65C22 = W65C22,
                    MM65SIB = MM65SIB,
                    W65C51 = W65C51,
                    AT28C010 = AT28C010,
                    AT28C64 = AT28C64,
            });
                stream.Close();
            }
            else
            {
                return;
            }
        }
```

6.17.3.7 GetLogModValue() `int Emulator.ViewModel.MainViewModel.GetLogModValue ( ) [inline], [private]`

Definition at line 699 of file MainViewModel.cs.

```
        {
            switch (CpuSpeed)
            {
                case 0:
                case 1:
                case 2:
                case 3:
                case 4:
```

```
                case 5:
                    return 1;
                case 6:
                    return 5;
                case 7:
                    return 20;
                case 8:
                    return 30;
                case 9:
                    return 40;
                case 10:
                    return 50;
                default:
                    return 5;
            }
        }
```

6.17.3.8 GetOutputLog() `OutputLog Emulator.ViewModel.MainViewModel.GetOutputLog ( ) [inline], [private]`

Definition at line 601 of file MainViewModel.cs.

```
        {
            if (W65C02.CurrentDisassembly == null)
            {
                return new OutputLog(new Disassembly());
            }

            return new OutputLog(W65C02.CurrentDisassembly)
            {
                XRegister = W65C02.XRegister.ToString("X").PadLeft(2, '0'),
                YRegister = W65C02.YRegister.ToString("X").PadLeft(2, '0'),
                Accumulator = W65C02.Accumulator.ToString("X").PadLeft(2, '0'),
                NumberOfCycles = NumberOfCycles,
                StackPointer = W65C02.StackPointer.ToString("X").PadLeft(2, '0'),
                ProgramCounter = W65C02.ProgramCounter.ToString("X").PadLeft(4, '0'),
                CurrentOpCode = W65C02.CurrentOpCode.ToString("X").PadLeft(2, '0')
            };
        }
```

6.17.3.9 GetSleepValue() `int Emulator.ViewModel.MainViewModel.GetSleepValue ( ) [inline], [private]`

Definition at line 725 of file MainViewModel.cs.

```
        {
            switch (CpuSpeed)
            {
                case 0:
                    return 550;
                case 1:
                    return 550;
                case 2:
                    return 440;
                case 3:
                    return 330;
                case 4:
                    return 220;
                case 5:
                    return 160;
                case 6:
                    return 80;
                case 7:
                    return 40;
                case 8:
                    return 20;
                case 9:
                    return 10;
                case 10:
                    return 5;
                default:
                    return 5;
            }
        }
```

6.17.3.10 IsBreakPointTriggered() `bool Emulator.ViewModel.MainViewModel.IsBreakPointTriggered ( ) [inline], [private]`

Definition at line 667 of file MainViewModel.cs.

```
        {
            //This prevents the Run Command from getting stuck after reaching a breakpoint
            if (_breakpointTriggered)
            {
                _breakpointTriggered = false;
                return false;
            }

            foreach (var breakpoint in Breakpoints.Where(x => x.IsEnabled))
            {
                if (!int.TryParse(breakpoint.Value, NumberStyles.AllowHexSpecifier,
      CultureInfo.InvariantCulture, out int value))
                    continue;

                if (breakpoint.Type == BreakpointType.NumberOfCycleType && value == NumberOfCycles)
                {
                    _breakpointTriggered = true;
                    RunPause();
                    return true;
                }

                if (breakpoint.Type == BreakpointType.ProgramCounterType && value ==
      W65C02.ProgramCounter)
                {
                    _breakpointTriggered = true;
                    RunPause();
                    return true;
                }
            }

            return false;
        }
```

6.17.3.11 OnClose() `void Emulator.ViewModel.MainViewModel.OnClose ( Object sender, CancelEventArgs e ) [inline]`

Definition at line 332 of file MainViewModel.cs.

```
        {
            e.Cancel = false;
            if (IsRunning)
            {
                MessageBox.Show("You can’t quit the emulator while it is actively running!",
                                "You can’t do that!", MessageBoxButton.OK, MessageBoxImage.Stop);
                e.Cancel = true;
                return;
            }
#if !DEBUG
            else
            {
                var result = MessageBox.Show(   "Are you sure you want to quit the emulator?",
                                                "To quit, or not to quit -- that is the question.",
                                                MessageBoxButton.YesNo, MessageBoxImage.Question,
                                                MessageBoxResult.No);
                if (result == MessageBoxResult.No)
                {
                    e.Cancel = true;
                    return;
                }
            }
#endif
            Stream stream = new FileStream(FileLocations.SettingsFile, FileMode.Create,
      FileAccess.Write, FileShare.None);
            XmlSerializer XmlFormatter = new XmlSerializer(typeof(SettingsModel));
            XmlFormatter.Serialize(stream, MainViewModel.SettingsModel);
            stream.Flush();
            stream.Close();
            W65C51.Fini();
        }
```

6.17.3.12 OnLoad()
`void Emulator.ViewModel.MainViewModel.OnLoad ( Object sender, RoutedEventArgs e ) [inline]`

Definition at line 313 of file MainViewModel.cs.

```
        {
#if !DEBUG
            if (Versioning.Product.Major < 1)
            {
                var result = MessageBox.Show(String.Format("Thank you for using {0}\n" +
                                                        "Be warned that this is a beta build.\n" +
                                                        "It may break or have bugs.",
      Versioning.Product.Name),
                                                        Versioning.Product.Title,
      MessageBoxButton.OKCancel,
                                                        MessageBoxImage.Warning,
      MessageBoxResult.None);
                if (result == MessageBoxResult.Cancel)
                {
                    // Exit without making any changes.
                    Environment.Exit(ExitCodes.NO_ERROR);
                }
            }
#endif
        }
```

6.17.3.13 RemoveBreakPoint()
`void Emulator.ViewModel.MainViewModel.RemoveBreakPoint ( ) [inline], [private]`

Definition at line 782 of file MainViewModel.cs.

```
        {
            if (SelectedBreakpoint == null)
                return;

            Breakpoints.Remove(SelectedBreakpoint);
            SelectedBreakpoint = null;
            RaisePropertyChanged("SelectedBreakpoint");
        }
```

6.17.3.14 Reset()
`void Emulator.ViewModel.MainViewModel.Reset ( ) [inline], [private]`

Definition at line 540 of file MainViewModel.cs.

```
        {
            IsRunning = false;

            if (_backgroundWorker.IsBusy)
                _backgroundWorker.CancelAsync();

            // "Reset" the Hardware...
            W65C02.Reset();
            RaisePropertyChanged("W65C02");
            W65C22.Reset();
            RaisePropertyChanged("W65C22");
            MM65SIB.Reset();
            RaisePropertyChanged("MM65SIB");
            W65C51.Reset();
            RaisePropertyChanged("W65C51");
            HM62256.Reset();
            RaisePropertyChanged("HM62256");

            IsRunning = false;
            NumberOfCycles = 0;
            RaisePropertyChanged("NumberOfCycles");

            UpdateMemoryPage();
            RaisePropertyChanged("MemoryPage");

            OutputLog.Clear();
            RaisePropertyChanged("CurrentDisassembly");

            OutputLog.Insert(0, GetOutputLog());
            UpdateUi();
        }
```

6.17.3.15 RunPause() `void Emulator.ViewModel.MainViewModel.RunPause ( ) [inline], [private]`

Definition at line 620 of file MainViewModel.cs.

```
        {
            var isRunning = !IsRunning;

            if (isRunning)
                _backgroundWorker.RunWorkerAsync();
            else
                _backgroundWorker.CancelAsync();

            IsRunning = !IsRunning;
        }
```

6.17.3.16 Settings() `void Emulator.ViewModel.MainViewModel.Settings ( ) [inline], [private]`

Definition at line 766 of file MainViewModel.cs.

```
        {
            IsRunning = false;

            if (_backgroundWorker.IsBusy)
                _backgroundWorker.CancelAsync();

            Messenger.Default.Send(new NotificationMessage<SettingsModel>(SettingsModel,
      "SettingsWindow"));
        }
```

6.17.3.17 SettingsAppliedNotifcation() `void Emulator.ViewModel.MainViewModel.SettingsApplied↩ Notifcation ( NotificationMessage< SettingsModel > notificationMessage ) [inline], [private]`

Definition at line 494 of file MainViewModel.cs.

```
        {
            if (notificationMessage.Notification != "SettingsApplied")
            {
                return;
            }

            SettingsModel = notificationMessage.Content;
            W65C51.Init(notificationMessage.Content.ComPortName);
            RaisePropertyChanged("SettingsModel");
            UpdateUi();
        }
```

6.17.3.18 StateLoadedNotifcation() `void Emulator.ViewModel.MainViewModel.StateLoadedNotifcation ( NotificationMessage< StateFileModel > notificationMessage ) [inline], [private]`

Definition at line 389 of file MainViewModel.cs.

```
        {
            if (notificationMessage.Notification != "StateLoaded")
            {
                return;
            }

            Reset();

            OutputLog = new
      MultiThreadedObservableCollection<OutputLog>(notificationMessage.Content.OutputLog);
            RaisePropertyChanged("OutputLog");

            NumberOfCycles = notificationMessage.Content.NumberOfCycles;

            W65C02 = notificationMessage.Content.W65C02;
            W65C22 = notificationMessage.Content.W65C22;
            MM65SIB = notificationMessage.Content.MM65SIB;
            W65C51 = notificationMessage.Content.W65C51;
            AT28C010 = notificationMessage.Content.AT28C010;
            AT28C64 = notificationMessage.Content.AT28C64;
            UpdateMemoryPage();
            UpdateUi();

            IsRomLoaded = true;
            RaisePropertyChanged("IsRomLoaded");
        }
```

6.17.3.19 Step() `void Emulator.ViewModel.MainViewModel.Step ( ) [inline], [private]`

Definition at line 573 of file MainViewModel.cs.

```
        {
            IsRunning = false;

            if (_backgroundWorker.IsBusy)
                _backgroundWorker.CancelAsync();

            StepProcessor();
            UpdateMemoryPage();

            OutputLog.Insert(0, GetOutputLog());
            UpdateUi();
        }
```

6.17.3.20 StepProcessor() `void Emulator.ViewModel.MainViewModel.StepProcessor ( ) [inline], [private]`

Definition at line 595 of file MainViewModel.cs.

```
        {
            W65C02.NextStep();
            NumberOfCycles = W65C02.GetCycleCount();
        }
```

6.17.3.21 UpdateMemoryPage() `void Emulator.ViewModel.MainViewModel.UpdateMemoryPage ( ) [inline], [private]`

Definition at line 507 of file MainViewModel.cs.

```
        {
            MemoryPage.Clear();
            var offset = _memoryPageOffset * 256;

            var multiplyer = 0;
            for (ushort i = (ushort)offset; i < 256 * (_memoryPageOffset + 1); i++)
            {

                MemoryPage.Add(new MemoryRowModel
                {
                    Offset = ((16 * multiplyer) + offset).ToString("X").PadLeft(4, '0'),
                    Location00 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location01 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location02 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location03 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location04 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location05 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location06 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location07 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location08 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location09 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location0A = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location0B = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location0C = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location0D = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location0E = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location0F = MemoryMap.ReadWithoutCycle(i).ToString("X").PadLeft(2, '0'),
                });
                multiplyer++;
            }
        }
```

6.17.3.22 UpdateUi() `void Emulator.ViewModel.MainViewModel.UpdateUi ( ) [inline], [private]`

Definition at line 587 of file MainViewModel.cs.

```
        {
            RaisePropertyChanged("W65C02");
            RaisePropertyChanged("NumberOfCycles");
            RaisePropertyChanged("CurrentDisassembly");
            RaisePropertyChanged("MemoryPage");
        }
```

6.17.4 Member Data Documentation

6.17.4.1 _backgroundWorker `readonly BackgroundWorker Emulator.ViewModel.MainViewModel._←backgroundWorker [private]`

Definition at line 31 of file MainViewModel.cs.

6.17.4.2 _breakpointTriggered `bool Emulator.ViewModel.MainViewModel._breakpointTriggered [private]`

Definition at line 32 of file MainViewModel.cs.

6.17.4.3 _memoryPageOffset `int Emulator.ViewModel.MainViewModel._memoryPageOffset [private]`

Definition at line 30 of file MainViewModel.cs.

6.17.5 Property Documentation

6.17.5.1 AboutCommand `RelayCommand Emulator.ViewModel.MainViewModel.AboutCommand [get], [set]`

The Relay Command that opens the About window.

Definition at line 193 of file MainViewModel.cs.

```
{ get; set; }
```

6.17.5.2 AddBreakPointCommand `RelayCommand Emulator.ViewModel.MainViewModel.AddBreakPoint←Command [get], [set]`

The Relay Command that adds a new breakpoint

Definition at line 188 of file MainViewModel.cs.

```
{ get; set; }
```

6.17.5.3 AT28C010 `AT28CXX Emulator.ViewModel.MainViewModel.AT28C010 [get], [private set]`

The AT28C010 ROM.

Definition at line 69 of file MainViewModel.cs.
```
00069 { get; private set; }
```

6.17.5.4 AT28C64 `AT28CXX Emulator.ViewModel.MainViewModel.AT28C64 [get], [private set]`

The AT28C010 ROM.

Definition at line 64 of file MainViewModel.cs.
```
00064 { get; private set; }
```

6.17.5.5 Breakpoints `MultiThreadedObservableCollection<Breakpoint> Emulator.ViewModel.Main↩ ViewModel.Breakpoints [get], [set]`

The Breakpoints

Definition at line 84 of file MainViewModel.cs.
```
00084 { get; set; }
```

6.17.5.6 CloseCommand `RelayCommand<IClosable> Emulator.ViewModel.MainViewModel.CloseCommand [get], [private set]`

The Command that loads or saves the settings.

Definition at line 208 of file MainViewModel.cs.
```
00208 { get; private set; }
```

6.17.5.7 CpuSpeed `int Emulator.ViewModel.MainViewModel.CpuSpeed [get], [set]`

The Slider CPU Speed

Definition at line 158 of file MainViewModel.cs.
```
00158 { get; set; }
```

6.17.5.8 CurrentDisassembly `string Emulator.ViewModel.MainViewModel.CurrentDisassembly [get]`

The Current Disassembly

Definition at line 99 of file MainViewModel.cs.

```
00100         {
00101             get
00102             {
00103                 if (W65C02.CurrentDisassembly != null)
00104                 {
00105                     return string.Format("{0} {1}", W65C02.CurrentDisassembly.OpCodeString,
      W65C02.CurrentDisassembly.DisassemblyOutput);
00106                 }
00107                 else
00108                 {
00109                     return string.Empty;
00110                 }
00111             }
00112         }
```

6.17.5.9 CurrentSerialPort `string Emulator.ViewModel.MainViewModel.CurrentSerialPort [get]`

The current serial port object name.

Definition at line 213 of file MainViewModel.cs.

```
00214         {
00215             get
00216             {
00217                 return W65C51.ObjectName;
00218             }
00219         }
```

6.17.5.10 HM62256 `HM62256 Emulator.ViewModel.MainViewModel.HM62256 [get], [set], [private]`

The 62256 RAM.

Definition at line 39 of file MainViewModel.cs.

```
00039 { get; set; }
```

6.17.5.11 IsRomLoaded `bool Emulator.ViewModel.MainViewModel.IsRomLoaded [get], [set]`

Is the banked ROM Loaded.

Definition at line 153 of file MainViewModel.cs.

```
00153 { get; set; }
```

6.17.5.12 IsRunning `bool Emulator.ViewModel.MainViewModel.IsRunning [get], [set]`

Is the Prorgam Running

Definition at line 140 of file MainViewModel.cs.

```
00141         {
00142             get { return W65C02.isRunning; }
00143             set
00144             {
00145                 W65C02.isRunning = value;
00146                 RaisePropertyChanged("IsRunning");
00147             }
00148         }
```

6.17.5.13 MemoryPage `MultiThreadedObservableCollection<MemoryRowModel> Emulator.ViewModel.↩ MainViewModel.MemoryPage [get], [set]`

The Current Memory Page

Definition at line 74 of file MainViewModel.cs.

```
{ get; set; }
```

6.17.5.14 MemoryPageOffset `string Emulator.ViewModel.MainViewModel.MemoryPageOffset [get], [set]`

The Memory Page number.

Definition at line 122 of file MainViewModel.cs.

```
        {
            get { return _memoryPageOffset.ToString("X"); }
            set
            {
                if (string.IsNullOrEmpty(value))
                    return;
                try
                {
                    _memoryPageOffset = Convert.ToInt32(value, 16);
                }
                catch { }
            }
        }
```

6.17.5.15 MM65SIB `W65C22 Emulator.ViewModel.MainViewModel.MM65SIB [get], [private set]`

Memory management and 65SIB.

Definition at line 54 of file MainViewModel.cs.

```
{ get; private set; }
```

6.17.5.16 NumberOfCycles `int Emulator.ViewModel.MainViewModel.NumberOfCycles [get], [private set]`

The number of cycles.

Definition at line 117 of file MainViewModel.cs.

```
{ get; private set; }
```

6.17.5.17 OutputLog `MultiThreadedObservableCollection<OutputLog> Emulator.ViewModel.Main↩ ViewModel.OutputLog [get], [private set]`

The output log

Definition at line 79 of file MainViewModel.cs.

```
{ get; private set; }
```

6.17.5.18 RemoveBreakPointCommand `RelayCommand Emulator.ViewModel.MainViewModel.Remove←BreakPointCommand [get], [set]`

The Relay Command that Removes an existing breakpoint.

Definition at line 198 of file MainViewModel.cs.

```
00198 { get; set; }
```

6.17.5.19 ResetCommand `RelayCommand Emulator.ViewModel.MainViewModel.ResetCommand [get], [set]`

Relay Command to Reset the Program back to its initial state.

Definition at line 173 of file MainViewModel.cs.

```
00173 { get; set; }
```

6.17.5.20 RomFile `RomFileModel Emulator.ViewModel.MainViewModel.RomFile [get], [set]`

The currently loaded binary file. (If it is indeed loaded, that is.)

Definition at line 94 of file MainViewModel.cs.

```
00094 { get; set; }
```

6.17.5.21 RunPauseCommand `RelayCommand Emulator.ViewModel.MainViewModel.RunPauseCommand [get], [set]`

Relay Command that Run/Pauses Execution

Definition at line 178 of file MainViewModel.cs.

```
00178 { get; set; }
```

6.17.5.22 SelectedBreakpoint `Breakpoint Emulator.ViewModel.MainViewModel.SelectedBreakpoint [get], [set]`

The Currently Selected Breakpoint

Definition at line 89 of file MainViewModel.cs.

```
00089 { get; set; }
```

6.17.5.23 SettingsCommand `RelayCommand Emulator.ViewModel.MainViewModel.SettingsCommand [get], [set]`

The Command that loads or saves the settings.

Definition at line 203 of file MainViewModel.cs.

```
00203 { get; set; }
```

6.17.5.24 SettingsModel `SettingsModel Emulator.ViewModel.MainViewModel.SettingsModel [static], [get], [set]`

The Model used for saving, loading and using data from Settings.xml

Definition at line 163 of file MainViewModel.cs.

```
00163 { get; set; }
```

6.17.5.25 StepCommand `RelayCommand Emulator.ViewModel.MainViewModel.StepCommand [get], [set]`

RelayCommand for Stepping through the progam one instruction at a time.

Definition at line 168 of file MainViewModel.cs.

```
00168 { get; set; }
```

6.17.5.26 UpdateMemoryMapCommand `RelayCommand Emulator.ViewModel.MainViewModel.Update↩ MemoryMapCommand [get], [set]`

Relay Command that updates the Memory Map when the Page changes

Definition at line 183 of file MainViewModel.cs.

```
00183 { get; set; }
```

6.17.5.27 W65C02 `W65C02 Emulator.ViewModel.MainViewModel.W65C02 [get], [private set]`

The 65C02 Processor.

Definition at line 44 of file MainViewModel.cs.

```
00044 { get; private set; }
```

6.17.5.28 W65C22 `W65C22 Emulator.ViewModel.MainViewModel.W65C22 [get], [private set]`

General Purpose I/O, Shift Registers and Timers.

Definition at line 49 of file MainViewModel.cs.

```
00049 { get; private set; }
```

6.17.5.29 W65C51 `W65C51 Emulator.ViewModel.MainViewModel.W65C51 [get], [private set]`

The ACIA serial interface.

Definition at line 59 of file MainViewModel.cs.

```
00059 { get; private set; }
```

6.17.5.30 WindowTitle `string Emulator.ViewModel.MainViewModel.WindowTitle [get]`

The title for the main window.

Definition at line 224 of file MainViewModel.cs.

```
00224 { get { return Versioning.Product.Title; } }
```

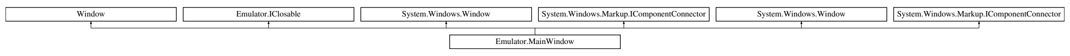
The documentation for this class was generated from the following file:

- Emulator/ViewModel/MainViewModel.cs

6.18 Emulator.MainWindow Class Reference

Interaction logic for MainWindow.xaml

Inheritance diagram for Emulator.MainWindow:



Public Member Functions

- MainWindow ()
- void InitializeComponent ()

  InitializeComponent
- void InitializeComponent ()

  InitializeComponent

Private Member Functions

- void ToClose (Object sender, EventArgs e)
- void LoadFile (Object sender, EventArgs e)
- void SaveFile (Object sender, EventArgs e)
- void CloseFile (Object sender, EventArgs e)
- void NotificationMessageReceived (NotificationMessage notificationMessage)
- void NotificationMessageReceived (NotificationMessage< StateFileModel > notificationMessage)
- void NotificationMessageReceived (NotificationMessage< SettingsModel > notificationMessage)
- void System.Windows.Markup.IComponentConnector. Connect (int connectionId, object target)
- void System.Windows.Markup.IComponentConnector. Connect (int connectionId, object target)

Private Attributes

- bool _contentLoaded

6.18.1 Detailed Description

Interaction logic for MainWindow.xaml

MainWindow

Definition at line 16 of file MainWindow.xaml.cs.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 MainWindow() `Emulator.MainWindow.MainWindow ( ) [inline]`

Definition at line 18 of file MainWindow.xaml.cs.

```
        {
            InitializeComponent();
            Messenger.Default.Register<NotificationMessage>(this, NotificationMessageReceived);
            Messenger.Default.Register<NotificationMessage<StateFileModel»(this,
NotificationMessageReceived);
            Messenger.Default.Register<NotificationMessage<SettingsModel»(this,
NotificationMessageReceived);
        }
```

6.18.3 Member Function Documentation

6.18.3.1 CloseFile() `void Emulator.MainWindow.CloseFile (`

```
            Object sender,
            EventArgs e )  [inline], [private]
```

Definition at line 41 of file MainWindow.xaml.cs.

```
        {
            Messenger.Default.Send(new NotificationMessage("CloseFile"));
        }
```

6.18.3.2 Connect() [1/2] `void System.Windows.Markup.IComponentConnector. Emulator.MainWindow.↩ Connect (`

```
            int connectionId,
            object target )  [inline], [private]
```

Definition at line 373 of file MainWindow.g.cs.

```
                                                                                                   {
            switch (connectionId)
            {
            case 1:
            this.EmulatorWindow = ((Emulator.MainWindow)(target));
            return;
            case 2:

#line 72 "..\..\..\MainWindow.xaml"
            ((System.Windows.Controls.MenuItem)(target)).Click += new
System.Windows.RoutedEventHandler(this.LoadFile);

#line default
#line hidden
            return;
            case 3:

#line 73 "..\..\..\MainWindow.xaml"
            ((System.Windows.Controls.MenuItem)(target)).Click += new
System.Windows.RoutedEventHandler(this.SaveFile);

#line default
#line hidden
            return;
            case 4:

#line 74 "..\..\..\MainWindow.xaml"
            ((System.Windows.Controls.MenuItem)(target)).Click += new
System.Windows.RoutedEventHandler(this.CloseFile);
```

```

#line default
#line hidden
            return;
            case 5:

#line 76 "..\..\..\MainWindow.xaml"
            ((System.Windows.Controls.MenuItem)(target)).Click += new
System.Windows.RoutedEventHandler(this.ToClose);

#line default
#line hidden
            return;
            case 6:
            this.OutputLog = ((System.Windows.Controls.DataGrid)(target));
            return;
            case 7:
            this.Run = ((System.Windows.Controls.Button)(target));
            return;
            case 8:
            this.Step = ((System.Windows.Controls.Button)(target));
            return;
            case 9:
            this.Reset = ((System.Windows.Controls.Button)(target));
            return;
            case 10:
            this.RomFileNameText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 11:
            this.ComPortNameText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 12:
            this.Breakpoints = ((System.Windows.Controls.DataGrid)(target));
            return;
            case 13:
            this.YRegister = ((System.Windows.Controls.TextBox)(target));
            return;
            case 14:
            this.XRegister = ((System.Windows.Controls.TextBox)(target));
            return;
            case 15:
            this.Accumulator = ((System.Windows.Controls.TextBox)(target));
            return;
            case 16:
            this.StackPointer = ((System.Windows.Controls.TextBox)(target));
            return;
            case 17:
            this.ProgramCounter = ((System.Windows.Controls.TextBox)(target));
            return;
            case 18:
            this.Dissambly = ((System.Windows.Controls.TextBox)(target));
            return;
            case 19:
            this.CycleCount = ((System.Windows.Controls.TextBox)(target));
            return;
            case 20:
            this.XRegisterText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 21:
            this.YRegisterText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 22:
            this.StackPointerRegisterText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 23:
            this.AText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 24:
            this.CurrentInstructionText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 25:
            this.ProgramCounterText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 26:
            this.CycleCountText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 27:
            this.CarryFlag = ((System.Windows.Controls.CheckBox)(target));
            return;
            case 28:
            this.CarryFlagText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 29:
            this.ZeroFlag = ((System.Windows.Controls.CheckBox)(target));
            return;
            case 30:
            this.ZeroFlagText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 31:
            this.InterruptFlag = ((System.Windows.Controls.CheckBox)(target));
            return;
            case 32:
            this.InterruptFlagText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 33:
            this.BcdFlag = ((System.Windows.Controls.CheckBox)(target));
            return;
            case 34:
            this.BcdFlagText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 35:
            this.BreakFlag = ((System.Windows.Controls.CheckBox)(target));
            return;
            case 36:
            this.BreakFlagText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 37:
            this.OverflowFlag = ((System.Windows.Controls.CheckBox)(target));
            return;
            case 38:
            this.OverflowFlagText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 39:
            this.NegativeFlag = ((System.Windows.Controls.CheckBox)(target));
            return;
            case 40:
            this.NegativeFlagText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 41:
            this.CpuSpeed = ((System.Windows.Controls.Slider)(target));
            return;
            case 42:
            this.SpeedText = ((System.Windows.Controls.TextBlock)(target));
            return;
            }
            this._contentLoaded = true;
        }
```

6.18.3.3 Connect() [2/2]

`void System.Windows.Markup.IComponentConnector. Emulator.MainWindow.↩ Connect (int connectionId, object target ) [inline], [private]`

Definition at line 373 of file MainWindow.g.i.cs.

```
                                                                                        {
            switch (connectionId)
            {
            case 1:
            this.EmulatorWindow = ((Emulator.MainWindow)(target));
            return;
            case 2:

#line 72 "..\..\..\MainWindow.xaml"
            ((System.Windows.Controls.MenuItem)(target)).Click += new
System.Windows.RoutedEventHandler(this.LoadFile);

#line default
#line hidden
            return;
            case 3:

#line 73 "..\..\..\MainWindow.xaml"
            ((System.Windows.Controls.MenuItem)(target)).Click += new
System.Windows.RoutedEventHandler(this.SaveFile);

#line default
#line hidden
            return;
            case 4:

#line 74 "..\..\..\MainWindow.xaml"
            ((System.Windows.Controls.MenuItem)(target)).Click += new
System.Windows.RoutedEventHandler(this.CloseFile);

#line default
#line hidden
```

```
            return;
            case 5:

#line 76 "..\..\..\MainWindow.xaml"
            ((System.Windows.Controls.MenuItem)(target)).Click += new
System.Windows.RoutedEventHandler(this.ToClose);

#line default
#line hidden
            return;
            case 6:
            this.OutputLog = ((System.Windows.Controls.DataGrid)(target));
            return;
            case 7:
            this.Run = ((System.Windows.Controls.Button)(target));
            return;
            case 8:
            this.Step = ((System.Windows.Controls.Button)(target));
            return;
            case 9:
            this.Reset = ((System.Windows.Controls.Button)(target));
            return;
            case 10:
            this.RomFileNameText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 11:
            this.ComPortNameText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 12:
            this.Breakpoints = ((System.Windows.Controls.DataGrid)(target));
            return;
            case 13:
            this.YRegister = ((System.Windows.Controls.TextBox)(target));
            return;
            case 14:
            this.XRegister = ((System.Windows.Controls.TextBox)(target));
            return;
            case 15:
            this.Accumulator = ((System.Windows.Controls.TextBox)(target));
            return;
            case 16:
            this.StackPointer = ((System.Windows.Controls.TextBox)(target));
            return;
            case 17:
            this.ProgramCounter = ((System.Windows.Controls.TextBox)(target));
            return;
            case 18:
            this.Dissambly = ((System.Windows.Controls.TextBox)(target));
            return;
            case 19:
            this.CycleCount = ((System.Windows.Controls.TextBox)(target));
            return;
            case 20:
            this.XRegisterText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 21:
            this.YRegisterText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 22:
            this.StackPointerRegisterText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 23:
            this.AText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 24:
            this.CurrentInstructionText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 25:
            this.ProgramCounterText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 26:
            this.CycleCountText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 27:
            this.CarryFlag = ((System.Windows.Controls.CheckBox)(target));
            return;
            case 28:
            this.CarryFlagText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 29:
            this.ZeroFlag = ((System.Windows.Controls.CheckBox)(target));
            return;
            case 30:
            this.ZeroFlagText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 31:
            this.InterrupFlag = ((System.Windows.Controls.CheckBox)(target));
```

```
00488             return;
00489             case 32:
00490             this.InterruptFlagText = ((System.Windows.Controls.TextBlock)(target));
00491             return;
00492             case 33:
00493             this.BcdFlag = ((System.Windows.Controls.CheckBox)(target));
00494             return;
00495             case 34:
00496             this.BcdFlagText = ((System.Windows.Controls.TextBlock)(target));
00497             return;
00498             case 35:
00499             this.BreakFlag = ((System.Windows.Controls.CheckBox)(target));
00500             return;
00501             case 36:
00502             this.BreakFlagText = ((System.Windows.Controls.TextBlock)(target));
00503             return;
00504             case 37:
00505             this.OverflowFlag = ((System.Windows.Controls.CheckBox)(target));
00506             return;
00507             case 38:
00508             this.OverflowFlagText = ((System.Windows.Controls.TextBlock)(target));
00509             return;
00510             case 39:
00511             this.NegativeFlag = ((System.Windows.Controls.CheckBox)(target));
00512             return;
00513             case 40:
00514             this.NegativeFlagText = ((System.Windows.Controls.TextBlock)(target));
00515             return;
00516             case 41:
00517             this.CpuSpeed = ((System.Windows.Controls.Slider)(target));
00518             return;
00519             case 42:
00520             this.SpeedText = ((System.Windows.Controls.TextBlock)(target));
00521             return;
00522             }
00523             this._contentLoaded = true;
00524         }
```

6.18.3.4 InitializeComponent() [1/2] `void Emulator.MainWindow.InitializeComponent ( ) [inline]`

InitializeComponent

Definition at line 353 of file MainWindow.g.cs.

```
00353                                             {
00354             if (_contentLoaded) {
00355                 return;
00356             }
00357             _contentLoaded = true;
00358             System.Uri resourceLocater = new System.Uri("/Emulator;component/mainwindow.xaml",
      System.UriKind.Relative);
00359
00360 #line 1 "..\..\..\MainWindow.xaml"
00361             System.Windows.Application.LoadComponent(this, resourceLocater);
00362
00363 #line default
00364 #line hidden
00365         }
```

6.18.3.5 InitializeComponent() [2/2] `void Emulator.MainWindow.InitializeComponent ( ) [inline]`

InitializeComponent

Definition at line 353 of file MainWindow.g.i.cs.

```
00353                                             {
00354             if (_contentLoaded) {
00355                 return;
00356             }
00357             _contentLoaded = true;
00358             System.Uri resourceLocater = new System.Uri("/Emulator;component/mainwindow.xaml",
      System.UriKind.Relative);
00359
00360 #line 1 "..\..\..\MainWindow.xaml"
00361             System.Windows.Application.LoadComponent(this, resourceLocater);
00362
00363 #line default
00364 #line hidden
00365         }
```

6.18.3.6 LoadFile() `void Emulator.MainWindow.LoadFile ( Object sender, EventArgs e ) [inline], [private]`

Definition at line 31 of file MainWindow.xaml.cs.

```
        {
            Messenger.Default.Send(new NotificationMessage("LoadFile"));
        }
```

6.18.3.7 NotificationMessageReceived() [1/3] `void Emulator.MainWindow.NotificationMessage↩ Received ( NotificationMessage notificationMessage ) [inline], [private]`

Definition at line 46 of file MainWindow.xaml.cs.

```
        {
            if (notificationMessage.Notification == "CloseWindow")
            {
                Close();
            }
        }
```

6.18.3.8 NotificationMessageReceived() [2/3] `void Emulator.MainWindow.NotificationMessage↩ Received ( NotificationMessage< SettingsModel > notificationMessage ) [inline], [private]`

Definition at line 63 of file MainWindow.xaml.cs.

```
        {
            if (notificationMessage.Notification == "SettingsWindow")
            {
                var settingsFile = new Settings { DataContext = new
      SettingsViewModel(notificationMessage.Content) };
                settingsFile.ShowDialog();
            }
        }
```

6.18.3.9 NotificationMessageReceived() [3/3] `void Emulator.MainWindow.NotificationMessage↩ Received ( NotificationMessage< StateFileModel > notificationMessage ) [inline], [private]`

Definition at line 54 of file MainWindow.xaml.cs.

```
        {
            if (notificationMessage.Notification == "SaveFileWindow")
            {
                var saveFile = new SaveFile { DataContext = new
      SaveFileViewModel(notificationMessage.Content) };
                saveFile.ShowDialog();
            }
        }
```

6.18.3.10 SaveFile() `void Emulator.MainWindow.SaveFile ( Object sender, EventArgs e ) [inline], [private]`

Definition at line 36 of file MainWindow.xaml.cs.

```
        {
            Messenger.Default.Send(new NotificationMessage("SaveState"));
        }
```

6.18.3.11 ToClose() `void Emulator.MainWindow.ToClose ( Object sender, EventArgs e ) [inline], [private]`

Definition at line 26 of file MainWindow.xaml.cs.

```
        {
            Close();
        }
```

6.18.4 Member Data Documentation

6.18.4.1 _contentLoaded `bool Emulator.MainWindow._contentLoaded [private]`

Definition at line 346 of file MainWindow.g.cs.

The documentation for this class was generated from the following files:

- Emulator/MainWindow.xaml.cs
- Emulator/obj/x86/Debug/MainWindow.g.cs
- Emulator/obj/x86/Debug/MainWindow.g.i.cs

6.19 Hardware.MemoryMap Class Reference

Classes

- class BankedRam
- class BankedRom
- class DeviceArea
- class Devices
- class SharedRom

Static Public Member Functions

- static void Init (W65C02 processor, W65C22 gpio, W65C22 mm65sib, W65C51 acia, HM62256 bankedRam, AT28CXX bankedRom, AT28CXX sharedRom)
- static byte Read (int address)

  Returns the byte at the given address.
- static byte ReadWithoutCycle (int address)

  Returns the byte at the given address without incrementing the cycle count.
- static void Write (int address, byte data)

  Writes data to the given address.
- static void WriteWithoutCycle (int address, byte data)

  Writes data to the given address without incrementing the cycle count.

Static Public Attributes

- static readonly int Length = 0xFFFF

Properties

- static W65C02 Processor `[get, set]`
- static W65C22 GPIO `[get, set]`
- static W65C22 MM65SIB `[get, set]`
- static W65C51 ACIA `[get, set]`
- static AT28CXX SharedROM `[get, set]`
- static AT28CXX BankedROM `[get, set]`
- static HM62256 BankedRAM `[get, set]`

6.19.1 Detailed Description

Definition at line 6 of file MemoryMap.cs.

6.19.2 Member Function Documentation

6.19.2.1 Init()

```
static void Hardware.MemoryMap.Init (
        W65C02 processor,
        W65C22 gpio,
        W65C22 mm65sib,
        W65C51 acia,
        HM62256 bankedRam,
        AT28CXX bankedRom,
        AT28CXX sharedRom )  [inline], [static]
```

Definition at line 87 of file MemoryMap.cs.

```
        {
            Processor = processor;
            GPIO = gpio;
            MM65SIB = mm65sib;
            ACIA = acia;
            SharedROM = sharedRom;
            BankedROM = bankedRom;
            BankedRAM = bankedRam;
        }
```

6.19.2.2 Read()

```
static byte Hardware.MemoryMap.Read (
        int address )  [inline], [static]
```

Returns the byte at the given address.

Parameters

address	The address to return

Returns

the byte being returned

Definition at line 103 of file MemoryMap.cs.

```
        {
            var value = ReadWithoutCycle(address);
            Processor.IncrementCycleCount();
            return value;
        }
```

6.19.2.3 ReadWithoutCycle() `static byte Hardware.MemoryMap.ReadWithoutCycle ( int address ) [inline], [static]`

Returns the byte at the given address without incrementing the cycle count.

Parameters

address	The address to return

Returns

the byte being returned

Definition at line 115 of file MemoryMap.cs.

```
        {
            int _address = address;
            if ((ACIA.Offset <= _address) && (_address <= (ACIA.Offset + ACIA.Length)))
            {
                return ACIA.Read(address);
            }
            else if ((GPIO.Offset <= _address) && (_address <= (GPIO.Offset + GPIO.Length)))
            {
                return GPIO.Read(_address);
            }
            else if ((DeviceArea.Offset <= _address) && (_address <= DeviceArea.End))
            {
                throw new ArgumentOutOfRangeException("Device area accessed where there is no
    device!");
            }
            else if ((SharedROM.Offset <= _address) && (_address <= SharedROM.End))
            {
                return SharedROM.Read(_address);
            }
            else if ((BankedROM.Offset <= _address) && (_address <= BankedROM.End))
            {
                return BankedROM.Read(_address);
            }
            else if ((BankedRAM.Offset <= _address) && (_address <= BankedRAM.End))
            {
                return BankedRAM.Read(_address);
            }
            else
            {
                return 0x00;
            }
        }
```

6.19.2.4 Write() `static void Hardware.MemoryMap.Write ( int address, byte data ) [inline], [static]`

Writes data to the given address.

Parameters

address	The address to write data to.
data	The data to write.

Definition at line 153 of file MemoryMap.cs.

```
        {
            Processor.IncrementCycleCount();
            WriteWithoutCycle(address, data);
        }
```

6.19.2.5 WriteWithoutCycle() `static void Hardware.MemoryMap.WriteWithoutCycle ( int address, byte data ) [inline], [static]`

Writes data to the given address without incrementing the cycle count.

Parameters

address	The address to write data to.
data	The data to write.

Definition at line 164 of file MemoryMap.cs.

```
        {
            if ((ACIA.Offset <= address) && (address <= (ACIA.Offset + ACIA.Length)))
            {
                ACIA.Write(address, data);
            }
            else if ((GPIO.Offset <= address) && (address <= (GPIO.Offset + GPIO.Length)))
            {
                GPIO.Write(address, data);
            }
            else if ((SharedROM.Offset <= address) && (address <= (SharedROM.Offset +
        SharedROM.Length)))
            {
                SharedROM.Write(address, data);
            }
            else if ((BankedROM.Offset <= address) && (address <= (BankedROM.Offset +
        BankedROM.Length)))
            {
                BankedROM.Write(address, data);
            }
            else if ((BankedRAM.Offset <= address) && (address <= (BankedRAM.Offset +
        BankedRAM.Length)))
            {
                BankedRAM.Write(address, data);
            }
            else
            {
                throw new ApplicationException(String.Format("Cannot write to address:  {0}",
        address));
            }
        }
```

6.19.3 Member Data Documentation

6.19.3.1 Length `readonly int Hardware.MemoryMap.Length = 0xFFFF [static]`

Definition at line 77 of file MemoryMap.cs.

6.19.4 Property Documentation

6.19.4.1 ACIA `W65C51 Hardware.MemoryMap.ACIA [static], [get], [set], [private]`

Definition at line 82 of file MemoryMap.cs.

```
00082 { get; set; }
```

6.19.4.2 BankedRAM `HM62256 Hardware.MemoryMap.BankedRAM [static], [get], [set], [private]`

Definition at line 85 of file MemoryMap.cs.

```
00085 { get; set; }
```

6.19.4.3 BankedROM `AT28CXX Hardware.MemoryMap.BankedROM [static], [get], [set], [private]`

Definition at line 84 of file MemoryMap.cs.

```
00084 { get; set; }
```

6.19.4.4 GPIO `W65C22 Hardware.MemoryMap.GPIO [static], [get], [set], [private]`

Definition at line 80 of file MemoryMap.cs.

```
00080 { get; set; }
```

6.19.4.5 MM65SIB `W65C22 Hardware.MemoryMap.MM65SIB [static], [get], [set], [private]`

Definition at line 81 of file MemoryMap.cs.

```
00081 { get; set; }
```

6.19.4.6 Processor `W65C02 Hardware.MemoryMap.Processor [static], [get], [set], [private]`

Definition at line 79 of file MemoryMap.cs.

```
00079 { get; set; }
```

6.19.4.7 SharedROM `AT28CXX Hardware.MemoryMap.SharedROM [static], [get], [set], [private]`

Definition at line 83 of file MemoryMap.cs.

```
00083 { get; set; }
```

The documentation for this class was generated from the following file:

- Hardware/Classes/MemoryMap.cs

6.20 Emulator.Model.MemoryRowModel Class Reference

A Model of a Single Page of memory

Properties

- string Offset `[get, set]`
  The offset of this row. Expressed in hex
- string Location00 `[get, set]`
  The memory at the location offset + 00
- string Location01 `[get, set]`
  The memory at the location offset + 01
- string Location02 `[get, set]`
  The memory at the location offset + 02
- string Location03 `[get, set]`
  The memory at the location offset + 03
- string Location04 `[get, set]`
  The memory at the location offset + 04
- string Location05 `[get, set]`
  The memory at the location offset + 05
- string Location06 `[get, set]`
  The memory at the location offset + 06
- string Location07 `[get, set]`
  The memory at the location offset + 07
- string Location08 `[get, set]`
  The memory at the location offset + 08
- string Location09 `[get, set]`
  The memory at the location offset + 09
- string Location0A `[get, set]`
  The memory at the location offset + 0A
- string Location0B `[get, set]`
  The memory at the location offset + 0B
- string Location0C `[get, set]`
  The memory at the location offset + 0C
- string Location0D `[get, set]`
  The memory at the location offset + 0D
- string Location0E `[get, set]`
  The memory at the location offset + 0E
- string Location0F `[get, set]`
  The memory at the location offset + 0F

6.20.1 Detailed Description

A Model of a Single Page of memory

Definition at line 6 of file MemoryRowModel.cs.

6.20.2 Property Documentation

6.20.2.1 Location00 `string Emulator.Model.MemoryRowModel.Location00 [get], [set]`

The memory at the location offset + 00

Definition at line 15 of file MemoryRowModel.cs.

```
00015 { get; set; }
```

6.20.2.2 Location01 `string Emulator.Model.MemoryRowModel.Location01 [get], [set]`

The memory at the location offset + 01

Definition at line 19 of file MemoryRowModel.cs.

```
00019 { get; set; }
```

6.20.2.3 Location02 `string Emulator.Model.MemoryRowModel.Location02 [get], [set]`

The memory at the location offset + 02

Definition at line 23 of file MemoryRowModel.cs.

```
00023 { get; set; }
```

6.20.2.4 Location03 `string Emulator.Model.MemoryRowModel.Location03 [get], [set]`

The memory at the location offset + 03

Definition at line 27 of file MemoryRowModel.cs.

```
00027 { get; set; }
```

6.20.2.5 Location04 `string Emulator.Model.MemoryRowModel.Location04 [get], [set]`

The memory at the location offset + 04

Definition at line 31 of file MemoryRowModel.cs.

```
00031 { get; set; }
```

6.20.2.6 Location05 `string Emulator.Model.MemoryRowModel.Location05 [get], [set]`

The memory at the location offset + 05

Definition at line 35 of file MemoryRowModel.cs.

```
00035 { get; set; }
```

6.20.2.7 Location06 `string Emulator.Model.MemoryRowModel.Location06 [get], [set]`

The memory at the location offset + 06

Definition at line 39 of file MemoryRowModel.cs.

```
00039 { get; set; }
```

6.20.2.8 Location07 `string Emulator.Model.MemoryRowModel.Location07 [get], [set]`

The memory at the location offset + 07

Definition at line 43 of file MemoryRowModel.cs.

```
00043 { get; set; }
```

6.20.2.9 Location08 `string Emulator.Model.MemoryRowModel.Location08 [get], [set]`

The memory at the location offset + 08

Definition at line 47 of file MemoryRowModel.cs.

```
00047 { get; set; }
```

6.20.2.10 Location09 `string Emulator.Model.MemoryRowModel.Location09 [get], [set]`

The memory at the location offset + 09

Definition at line 51 of file MemoryRowModel.cs.

```
00051 { get; set; }
```

6.20.2.11 Location0A `string Emulator.Model.MemoryRowModel.Location0A [get], [set]`

The memory at the location offset + 0A

Definition at line 55 of file MemoryRowModel.cs.

```
00055 { get; set; }
```

6.20.2.12 Location0B `string Emulator.Model.MemoryRowModel.Location0B [get], [set]`

The memory at the location offset + 0B

Definition at line 59 of file MemoryRowModel.cs.
```
00059 { get; set; }
```

6.20.2.13 Location0C `string Emulator.Model.MemoryRowModel.Location0C [get], [set]`

The memory at the location offset + 0C

Definition at line 63 of file MemoryRowModel.cs.
```
00063 { get; set; }
```

6.20.2.14 Location0D `string Emulator.Model.MemoryRowModel.Location0D [get], [set]`

The memory at the location offset + 0D

Definition at line 67 of file MemoryRowModel.cs.
```
00067 { get; set; }
```

6.20.2.15 Location0E `string Emulator.Model.MemoryRowModel.Location0E [get], [set]`

The memory at the location offset + 0E

Definition at line 71 of file MemoryRowModel.cs.
```
00071 { get; set; }
```

6.20.2.16 Location0F `string Emulator.Model.MemoryRowModel.Location0F [get], [set]`

The memory at the location offset + 0F

Definition at line 75 of file MemoryRowModel.cs.
```
00075 { get; set; }
```

6.20.2.17 Offset `string Emulator.Model.MemoryRowModel.Offset [get], [set]`

The offset of this row. Expressed in hex

Definition at line 11 of file MemoryRowModel.cs.
```
00011 { get; set; }
```

The documentation for this class was generated from the following file:

- Emulator/Model/MemoryRowModel.cs

6.21 Hardware.MemoryMap.Devices.MM65SIB Class Reference

Static Public Attributes

- static int Length = 0x0F
- static byte Offset = 0x30

6.21.1 Detailed Description

Definition at line 70 of file MemoryMap.cs.

6.21.2 Member Data Documentation

6.21.2.1 Length `int Hardware.MemoryMap.Devices.MM65SIB.Length = 0x0F [static]`

Definition at line 72 of file MemoryMap.cs.

6.21.2.2 Offset `byte Hardware.MemoryMap.Devices.MM65SIB.Offset = 0x30 [static]`

Definition at line 73 of file MemoryMap.cs.

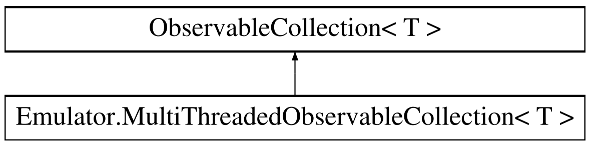
The documentation for this class was generated from the following file:

- Hardware/Classes/MemoryMap.cs

6.22 Emulator.MultiThreadedObservableCollection< T > Class Template Reference

A MultiThreaedObservableCollection. This allows multiple threads to access the same observable collection in a safe manner.

Inheritance diagram for Emulator.MultiThreadedObservableCollection< T >:


ObservableCollection< T >
Emulator.MultiThreadedObservableCollection< T >

Public Member Functions

- MultiThreadedObservableCollection ()

  Instantiates a new instance of the MultiThreadedObservableCollection
- MultiThreadedObservableCollection (IEnumerable< T > collection)

  Instantiates a new instance of the MultiThreadedObservableCollection
- MultiThreadedObservableCollection (List< T > list)

  Instantiates a new instance of the MultiThreadedObservableCollection

Protected Member Functions

- override void OnCollectionChanged (NotifyCollectionChangedEventArgs e)

    The NotifyCollectionChangedEventHandler, Notifies the listeners in a thread safe manner

Events

- override NotifyCollectionChangedEventHandler CollectionChanged

    The NotifyCollectionChangedEventHandler, Sends a notification anytime the collection has been modified.

6.22.1 Detailed Description

A MultiThreaedObservableCollection. This allows multiple threads to access the same observable collection in a safe manner.

Template Parameters

T	

Definition at line 14 of file MultiThreadedCollection.cs.

6.22.2 Constructor & Destructor Documentation

6.22.2.1 MultiThreadedObservableCollection() `[1/3]` `Emulator.MultiThreadedObservableCollection< T >.MultiThreadedObservableCollection ( ) [inline]`

Instantiates a new instance of the MultiThreadedObservableCollection

Definition at line 19 of file MultiThreadedCollection.cs.

```
        {
        
        }
```

6.22.2.2 MultiThreadedObservableCollection() `[2/3]` `Emulator.MultiThreadedObservableCollection< T >.MultiThreadedObservableCollection ( IEnumerable< T > collection ) [inline]`

Instantiates a new instance of the MultiThreadedObservableCollection

Parameters

collection	The initial collection to be loaded

Definition at line 28 of file MultiThreadedCollection.cs.

```
            : base(collection)
        {
        
        }
```

6.22.2.3 MultiThreadedObservableCollection() [3/3] Emulator.MultiThreadedObservableCollection< T >.MultiThreadedObservableCollection (List< T > *list*) [inline]

Instantiates a new instance of the MultiThreadedObservableCollection

Parameters

list	The initial list to be loaded

Definition at line 38 of file MultiThreadedCollection.cs.

```
            : base(list)
        {
        
        }
```

6.22.3 Member Function Documentation

6.22.3.1 OnCollectionChanged() override void Emulator.MultiThreadedObservableCollection< T >.OnCollectionChanged (NotifyCollectionChangedEventArgs *e*) [inline], [protected]

The NotifyCollectionChangedEventHandler, Notifies the listeners in a thread safe manner

Definition at line 53 of file MultiThreadedCollection.cs.

```
        {
            var collectionChanged = CollectionChanged;
            if (collectionChanged != null)
                foreach (NotifyCollectionChangedEventHandler nh in
collectionChanged.GetInvocationList())
                {
                    var dispObj = nh.Target as DispatcherObject;
                    if (dispObj != null)
                    {
                        var dispatcher = dispObj.Dispatcher;
                        if (dispatcher != null && !dispatcher.CheckAccess())
                        {
                            var nh1 = nh;
                            dispatcher.BeginInvoke(
                                (Action)(() => nh1.Invoke(this,
                                    new
NotifyCollectionChangedEventArgs(NotifyCollectionChangedAction.Reset))),
                                DispatcherPriority.DataBind);
                            continue;
                        }
                    }
                    nh.Invoke(this, e);
                }
        }
```

6.22.4 Event Documentation

6.22.4.1 CollectionChanged `override NotifyCollectionChangedEventHandler Emulator.MultiThreadedObservableColle` `T >.CollectionChanged`

The NotifyCollectionChangedEventHandler, Sends a notification anytime the collection has been modified.

Definition at line 47 of file MultiThreadedCollection.cs.

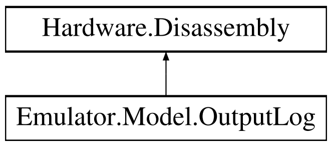
The documentation for this class was generated from the following file:

- Emulator/MultiThreadedCollection.cs

6.23 Emulator.Model.OutputLog Class Reference

The OutputLog Model. Used by the outputlog grid to show a history of operations performed by the CPU

Inheritance diagram for Emulator.Model.OutputLog:



Public Member Functions

- OutputLog (Disassembly disassembly)

Properties

- string ProgramCounter `[get, set]`
  The Program Counter Value
- string CurrentOpCode `[get, set]`
  The Current Ope Code
- string XRegister `[get, set]`
  The X Register
- string YRegister `[get, set]`
  The Y Register
- string Accumulator `[get, set]`
  The Accummulator
- string StackPointer `[get, set]`
  The Stack Pointer
- int NumberOfCycles `[get, set]`
  The number of cycles executed since the last load or reset

6.23.1 Detailed Description

The OutputLog Model. Used by the outputlog grid to show a history of operations performed by the CPU

Definition at line 10 of file OutputLog.cs.

6.23.2 Constructor & Destructor Documentation

6.23.2.1 OutputLog() `Emulator.Model.OutputLog.OutputLog ( Disassembly *disassembly* ) [inline]`

Definition at line 12 of file OutputLog.cs.

```
        {
            DisassemblyOutput = disassembly.DisassemblyOutput;
            HighAddress = disassembly.HighAddress;
            LowAddress = disassembly.LowAddress;
            OpCodeString = disassembly.OpCodeString;
        }
```

6.23.3 Property Documentation

6.23.3.1 Accumulator `string Emulator.Model.OutputLog.Accumulator [get], [set]`

The Accummulator

Definition at line 39 of file OutputLog.cs.

```
{ get; set; }
```

6.23.3.2 CurrentOpCode `string Emulator.Model.OutputLog.CurrentOpCode [get], [set]`

The Current Ope Code

Definition at line 27 of file OutputLog.cs.

```
{ get; set; }
```

6.23.3.3 NumberOfCycles `int Emulator.Model.OutputLog.NumberOfCycles [get], [set]`

The number of cycles executed since the last load or reset

Definition at line 47 of file OutputLog.cs.

```
{ get; set; }
```

6.23.3.4 ProgramCounter `string Emulator.Model.OutputLog.ProgramCounter [get], [set]`

The Program Counter Value

Definition at line 23 of file OutputLog.cs.

```
{ get; set; }
```

6.23.3.5 StackPointer `string Emulator.Model.OutputLog.StackPointer [get], [set]`

The Stack Pointer

Definition at line 43 of file OutputLog.cs.

```
00043 { get; set; }
```

6.23.3.6 XRegister `string Emulator.Model.OutputLog.XRegister [get], [set]`

The X Register

Definition at line 31 of file OutputLog.cs.

```
00031 { get; set; }
```

6.23.3.7 YRegister `string Emulator.Model.OutputLog.YRegister [get], [set]`

The Y Register

Definition at line 35 of file OutputLog.cs.

```
00035 { get; set; }
```

The documentation for this class was generated from the following file:

- Emulator/Model/OutputLog.cs

6.24 Emulator.Versioning.Product Class Reference

Static Public Attributes

- const int Major = 0
- const int Minor = 1
- const int Build = 3
- const int Revision = 1
- const string Title = Name
- const string Name = "WolfNet 65C02 WorkBench Computer Emulator"
- const string Company = "WolfNet Computing"
- const string Copyright = "Copyright ľ WolfNet Computing 2022"
- const string VersionString = "0.2.3.1"
- const string Description = "Emulator for the WolfNet 65C02 WorkBench Computer coded in C# using the .NET Framework"

6.24.1 Detailed Description

Definition at line 9 of file Versioning.cs.

6.24.2 Member Data Documentation

6.24.2.1 Build `const int Emulator.Versioning.Product.Build = 3 [static]`

Definition at line 13 of file Versioning.cs.

6.24.2.2 Company `const string Emulator.Versioning.Product.Company = "WolfNet Computing" [static]`

Definition at line 17 of file Versioning.cs.

6.24.2.3 Copyright `const string Emulator.Versioning.Product.Copyright = "Copyright ľ WolfNet Computing 2022" [static]`

Definition at line 18 of file Versioning.cs.

6.24.2.4 Description `const string Emulator.Versioning.Product.Description = "Emulator for the WolfNet 65C02 WorkBench Computer coded in C# using the .NET Framework" [static]`

Definition at line 20 of file Versioning.cs.

6.24.2.5 Major `const int Emulator.Versioning.Product.Major = 0 [static]`

Definition at line 11 of file Versioning.cs.

6.24.2.6 Minor `const int Emulator.Versioning.Product.Minor = 1 [static]`

Definition at line 12 of file Versioning.cs.

6.24.2.7 Name `const string Emulator.Versioning.Product.Name = "WolfNet 65C02 WorkBench Computer Emulator" [static]`

Definition at line 16 of file Versioning.cs.

6.24.2.8 Revision `const int Emulator.Versioning.Product.Revision = 1 [static]`

Definition at line 14 of file Versioning.cs.

6.24.2.9 Title `const string Emulator.Versioning.Product.Title = Name [static]`

Definition at line 15 of file Versioning.cs.

6.24.2.10 VersionString `const string Emulator.Versioning.Product.VersionString = "0.2.3.1" [static]`

Definition at line 19 of file Versioning.cs.

The documentation for this class was generated from the following file:

- Emulator/Classes/Versioning.cs

6.25 Hardware.Versioning.Product Class Reference

Static Public Attributes

- const string Title = Name
- const string Name = "WolfNet 65C02 Hardware Library"
- const string Company = "WolfNet Computing"
- const string Copyright = "Copyright l’ WolfNet Computing 2022"
- const string Version = "1.3.0.0"
- const string Description = "65C02 Hardware Library, coded in C# using the .NET Framework"

6.25.1 Detailed Description

Definition at line 5 of file Versioning.cs.

6.25.2 Member Data Documentation

6.25.2.1 Company `const string Hardware.Versioning.Product.Company = "WolfNet Computing" [static]`

Definition at line 9 of file Versioning.cs.

6.25.2.2 Copyright `const string Hardware.Versioning.Product.Copyright = "Copyright 1’ WolfNet Computing 2022" [static]`

Definition at line 10 of file Versioning.cs.

6.25.2.3 Description `const string Hardware.Versioning.Product.Description = "65C02 Hardware Library, coded in C# using the .NET Framework" [static]`

Definition at line 12 of file Versioning.cs.

6.25.2.4 Name `const string Hardware.Versioning.Product.Name = "WolfNet 65C02 Hardware Library" [static]`

Definition at line 8 of file Versioning.cs.

6.25.2.5 Title `const string Hardware.Versioning.Product.Title = Name [static]`

Definition at line 7 of file Versioning.cs.

6.25.2.6 Version `const string Hardware.Versioning.Product.Version = "1.3.0.0" [static]`

Definition at line 11 of file Versioning.cs.

The documentation for this class was generated from the following file:

- Hardware/Classes/Versioning.cs

6.26 Emulator.Model.RomFileModel Class Reference

The Model used when Loading a Program.

Properties

- byte[ ][ ] Rom `[get, set]`
  The Program Converted into Hex.
- byte RomBanks `[get, set]`
  The path of the Program that was loaded.
- int RomBankSize `[get, set]`
  The name of the Program that was loaded.
- string RomFileName `[get, set]`
  The name of the Program that was loaded.
- string RomFilePath `[get, set]`
  The path of the Program that was loaded.

6.26.1 Detailed Description

The Model used when Loading a Program.

Definition at line 6 of file RomFileModel.cs.

6.26.2 Property Documentation

6.26.2.1 Rom `byte [][] Emulator.Model.RomFileModel.Rom [get], [set]`

The Program Converted into Hex.

Definition at line 11 of file RomFileModel.cs.

```
00011 { get; set; }
```

6.26.2.2 RomBanks `byte Emulator.Model.RomFileModel.RomBanks [get], [set]`

The path of the Program that was loaded.

Definition at line 16 of file RomFileModel.cs.

```
00016 { get; set; }
```

6.26.2.3 RomBankSize `int Emulator.Model.RomFileModel.RomBankSize [get], [set]`

The name of the Program that was loaded.

Definition at line 21 of file RomFileModel.cs.

```
00021 { get; set; }
```

6.26.2.4 RomFileName `string Emulator.Model.RomFileModel.RomFileName [get], [set]`

The name of the Program that was loaded.

Definition at line 26 of file RomFileModel.cs.

```
00026 { get; set; }
```

6.26.2.5 RomFilePath `string Emulator.Model.RomFileModel.RomFilePath [get], [set]`

The path of the Program that was loaded.

Definition at line 31 of file RomFileModel.cs.

```
00031 { get; set; }
```

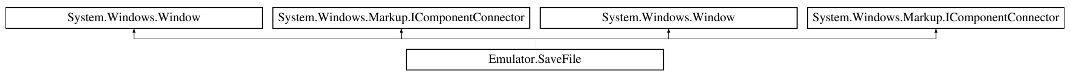
The documentation for this class was generated from the following file:

- Emulator/Model/RomFileModel.cs

6.27 Emulator.SaveFile Class Reference

SaveFile

Inheritance diagram for Emulator.SaveFile:


System.Windows.Window
System.Windows.Markup.IComponentConnector
System.Windows.Window
System.Windows.Markup.IComponentConnector
Emulator.SaveFile

Public Member Functions

- void InitializeComponent ()

  InitializeComponent
- void InitializeComponent ()

  InitializeComponent
- SaveFile ()

Private Member Functions

- void System.Windows.Markup.IComponentConnector. Connect (int connectionId, object target)
- void System.Windows.Markup.IComponentConnector. Connect (int connectionId, object target)
- void NotificationMessageReceived (NotificationMessage notificationMessage)

Private Attributes

- bool _contentLoaded

6.27.1 Detailed Description

SaveFile

Interaction logic for SaveState.xaml

Definition at line 40 of file SaveFile.g.cs.

6.27.2 Constructor & Destructor Documentation

6.27.2.1 SaveFile() `Emulator.SaveFile.SaveFile ( ) [inline]`

Definition at line 10 of file SaveFile.xaml.cs.

```
        {
            InitializeComponent();
            Messenger.Default.Register<NotificationMessage>(this, NotificationMessageReceived);
        }
```

6.27.3 Member Function Documentation

6.27.3.1 Connect() [1/2] `void System.Windows.Markup.IComponentConnector. Emulator.SaveFile.↩ Connect (`

```
int connectionId,
object target ) [inline], [private]
```

Definition at line 109 of file SaveFile.g.cs.

```
00109                                                                                                         {
00110             switch (connectionId)
00111             {
00112             case 1:
00113             this.SelectFile = ((System.Windows.Controls.Button)(target));
00114             return;
00115             case 2:
00116             this.FilePath = ((System.Windows.Controls.TextBox)(target));
00117             return;
00118             case 3:
00119             this.PathText = ((System.Windows.Controls.TextBlock)(target));
00120             return;
00121             case 4:
00122             this.CancelButton = ((System.Windows.Controls.Button)(target));
00123             return;
00124             case 5:
00125             this.LoadButton = ((System.Windows.Controls.Button)(target));
00126             return;
00127             }
00128             this._contentLoaded = true;
00129         }
```

6.27.3.2 Connect() [2/2] `void System.Windows.Markup.IComponentConnector. Emulator.SaveFile.↩ Connect (`

```
int connectionId,
object target ) [inline], [private]
```

Definition at line 109 of file SaveFile.g.i.cs.

```
00109                                                                                                         {
00110             switch (connectionId)
00111             {
00112             case 1:
00113             this.SelectFile = ((System.Windows.Controls.Button)(target));
00114             return;
00115             case 2:
00116             this.FilePath = ((System.Windows.Controls.TextBox)(target));
00117             return;
00118             case 3:
00119             this.PathText = ((System.Windows.Controls.TextBlock)(target));
00120             return;
00121             case 4:
00122             this.CancelButton = ((System.Windows.Controls.Button)(target));
00123             return;
00124             case 5:
00125             this.LoadButton = ((System.Windows.Controls.Button)(target));
00126             return;
00127             }
00128             this._contentLoaded = true;
00129         }
```

6.27.3.3 InitializeComponent() [1/2] void Emulator.SaveFile.InitializeComponent ( ) [inline]

InitializeComponent

Definition at line 89 of file SaveFile.g.cs.

```
                                {
            if (_contentLoaded) {
                return;
            }
            _contentLoaded = true;
            System.Uri resourceLocater = new System.Uri("/Emulator;component/savefile.xaml",
      System.UriKind.Relative);

#line 1 "..\..\..\SaveFile.xaml"
            System.Windows.Application.LoadComponent(this, resourceLocater);

#line default
#line hidden
        }
```

6.27.3.4 InitializeComponent() [2/2] void Emulator.SaveFile.InitializeComponent ( ) [inline]

InitializeComponent

Definition at line 89 of file SaveFile.g.i.cs.

```
                                {
            if (_contentLoaded) {
                return;
            }
            _contentLoaded = true;
            System.Uri resourceLocater = new System.Uri("/Emulator;component/savefile.xaml",
      System.UriKind.Relative);

#line 1 "..\..\..\SaveFile.xaml"
            System.Windows.Application.LoadComponent(this, resourceLocater);

#line default
#line hidden
        }
```

6.27.3.5 NotificationMessageReceived() void Emulator.SaveFile.NotificationMessageReceived (NotificationMessage *notificationMessage*) [inline], [private]

Definition at line 16 of file SaveFile.xaml.cs.

```
        {
            if (notificationMessage.Notification == "CloseSaveFileWindow")
                Close();
        }
```

6.27.4 Member Data Documentation

6.27.4.1 _contentLoaded bool Emulator.SaveFile._contentLoaded [private]

Definition at line 82 of file SaveFile.g.cs.

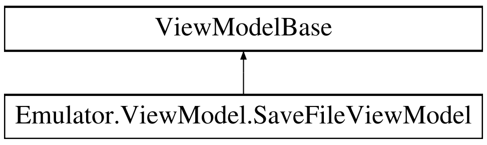
The documentation for this class was generated from the following files:

- Emulator/obj/x86/Debug/SaveFile.g.cs
- Emulator/obj/x86/Debug/SaveFile.g.i.cs
- Emulator/SaveFile.xaml.cs

6.28 Emulator.ViewModel.SaveFileViewModel Class Reference

The ViewModel Used by the SaveFileView

Inheritance diagram for Emulator.ViewModel.SaveFileViewModel:



Public Member Functions

- SaveFileViewModel ()

  Instantiates a new instance of the SaveFileViewModel. This is used by the IOC to create the default instance.
- SaveFileViewModel (StateFileModel stateFileModel)

  Instantiates a new instance of the SaveFileViewModel

Properties

- RelayCommand SaveFileCommand `[get, set]`

  The Relay Command called when saving a file
- RelayCommand CloseCommand `[get, set]`

  The Relay Command called when closing a file
- RelayCommand SelectFileCommand `[get, set]`

  The Relay Command called when Selecting a file
- string Filename `[get, set]`

  The file to be saved
- bool SaveEnabled `[get]`

  Tells the UI that that a file has been selected and can be saved.

Private Member Functions

- void Save ()
- void Select ()

Static Private Member Functions

- static void Close ()

Private Attributes

- readonly StateFileModel _stateFileModel

6.28.1 Detailed Description

The ViewModel Used by the SaveFileView

Definition at line 15 of file SaveFileViewModel.cs.

6.28.2 Constructor & Destructor Documentation

6.28.2.1 SaveFileViewModel() [1/2] Emulator.ViewModel.SaveFileViewModel.SaveFileViewModel ( ) [inline]

Instantiates a new instance of the SaveFileViewModel. This is used by the IOC to create the default instance.

Definition at line 51 of file SaveFileViewModel.cs.

```
        {
        }
```

6.28.2.2 SaveFileViewModel() [2/2] Emulator.ViewModel.SaveFileViewModel.SaveFileViewModel (StateFileModel *stateFileModel*) [inline]

Instantiates a new instance of the SaveFileViewModel

Parameters

stateFileModel	The StateFileModel to be serialized to a file

Definition at line 60 of file SaveFileViewModel.cs.

```
        {
            SaveFileCommand = new RelayCommand(Save);
            CloseCommand = new RelayCommand(Close);
            SelectFileCommand = new RelayCommand(Select);
            _stateFileModel = stateFileModel;
        }
```

6.28.3 Member Function Documentation

6.28.3.1 Close() static void Emulator.ViewModel.SaveFileViewModel.Close ( ) [inline], [static], [private]

Definition at line 80 of file SaveFileViewModel.cs.

```
        {
            Messenger.Default.Send(new NotificationMessage("CloseSaveFileWindow"));
        }
```

6.28.3.2 Save() `void Emulator.ViewModel.SaveFileViewModel.Save ( ) [inline], [private]`

Definition at line 70 of file SaveFileViewModel.cs.

```
        {
            var formatter = new BinaryFormatter();
            Stream stream = new FileStream(Filename, FileMode.Create, FileAccess.Write,
    FileShare.None);
            formatter.Serialize(stream, _stateFileModel);
            stream.Close();

            Close();
        }
```

6.28.3.3 Select() `void Emulator.ViewModel.SaveFileViewModel.Select ( ) [inline], [private]`

Definition at line 85 of file SaveFileViewModel.cs.

```
        {
            var dialog = new SaveFileDialog { DefaultExt = ".6502", Filter = "WolfNet W65C02 Emulator
    Save State (*.6502)|*.6502" };

            var result = dialog.ShowDialog();

            if (result != true)
                return;

            Filename = dialog.FileName;
            RaisePropertyChanged("Filename");
            RaisePropertyChanged("SaveEnabled");

        }
```

6.28.4 Member Data Documentation

6.28.4.1 _stateFileModel `readonly StateFileModel Emulator.ViewModel.SaveFileViewModel._state↩ FileModel [private]`

Definition at line 17 of file SaveFileViewModel.cs.

6.28.5 Property Documentation

6.28.5.1 CloseCommand `RelayCommand Emulator.ViewModel.SaveFileViewModel.CloseCommand [get], [set]`

The Relay Command called when closing a file

Definition at line 28 of file SaveFileViewModel.cs.

```
00028 { get; set; }
```

6.28.5.2 Filename `string Emulator.ViewModel.SaveFileViewModel.Filename [get], [set]`

The file to be saved

Definition at line 38 of file SaveFileViewModel.cs.

```
00038 { get; set; }
```

6.28.5.3 SaveEnabled `bool Emulator.ViewModel.SaveFileViewModel.SaveEnabled [get]`

Tells the UI that that a file has been selected and can be saved.

Definition at line 43 of file SaveFileViewModel.cs.

```
00043 { get { return !string.IsNullOrEmpty(Filename); }}
```

6.28.5.4 SaveFileCommand `RelayCommand Emulator.ViewModel.SaveFileViewModel.SaveFileCommand [get], [set]`

The Relay Command called when saving a file

Definition at line 23 of file SaveFileViewModel.cs.

```
00023 { get; set; }
```

6.28.5.5 SelectFileCommand `RelayCommand Emulator.ViewModel.SaveFileViewModel.SelectFile↩ Command [get], [set]`

The Relay Command called when Selecting a file

Definition at line 33 of file SaveFileViewModel.cs.

```
00033 { get; set; }
```

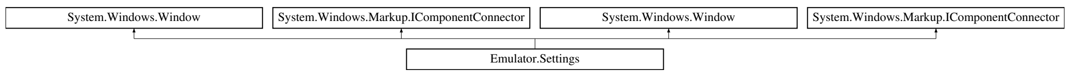
The documentation for this class was generated from the following file:

- Emulator/ViewModel/SaveFileViewModel.cs

6.29 Emulator.Settings Class Reference

Settings

Inheritance diagram for Emulator.Settings:


System.Windows.Window
System.Windows.Markup.IComponentConnector
System.Windows.Window
System.Windows.Markup.IComponentConnector
Emulator.Settings

Public Member Functions

- void InitializeComponent ()

  InitializeComponent
- void InitializeComponent ()

  InitializeComponent
- Settings ()

Private Member Functions

- void System.Windows.Markup.IComponentConnector. Connect (int connectionId, object target)
- void System.Windows.Markup.IComponentConnector. Connect (int connectionId, object target)
- void NotificationMessageReceived (NotificationMessage notificationMessage)
- void NotificationMessageReceived (NotificationMessage< SettingsModel > notificationMessage)
- void PortSelectionDropDownClosed (object sender, EventArgs e)

Private Attributes

- bool _contentLoaded

6.29.1 Detailed Description

Settings

Interaction logic for Settings.xaml

Definition at line 40 of file Settings.g.cs.

6.29.2 Constructor & Destructor Documentation

6.29.2.1 Settings() `Emulator.Settings.Settings ( ) [inline]`

Definition at line 14 of file Settings.xaml.cs.

```
        {
            InitializeComponent();
            Messenger.Default.Register<NotificationMessage>(this, NotificationMessageReceived);
            Messenger.Default.Register<NotificationMessage<SettingsModel»(this,
    NotificationMessageReceived);
        }
```

6.29.3 Member Function Documentation

6.29.3.1 Connect() [1/2] void System.Windows.Markup.IComponentConnector. Emulator.Settings.↩ Connect (

```
            int connectionId,
            object target )  [inline], [private]
```

Definition at line 101 of file Settings.g.cs.

```
                                                                                {
            switch (connectionId)
            {
            case 1:
            this.ComPortCombo = ((System.Windows.Controls.ComboBox)(target));

#line 7 "..\..\..\Settings.xaml"
            this.ComPortCombo.DropDownClosed += new
      System.EventHandler(this.PortSelectionDropDownClosed);

#line default
#line hidden
            return;
            case 2:
            this.PortText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 3:
            this.ApplyButton = ((System.Windows.Controls.Button)(target));
            return;
            case 4:
            this.CloseButton = ((System.Windows.Controls.Button)(target));
            return;
            }
            this._contentLoaded = true;
        }
```

6.29.3.2 Connect() [2/2] void System.Windows.Markup.IComponentConnector. Emulator.Settings.↩ Connect (

```
            int connectionId,
            object target )  [inline], [private]
```

Definition at line 101 of file Settings.g.i.cs.

```
                                                                                {
            switch (connectionId)
            {
            case 1:
            this.ComPortCombo = ((System.Windows.Controls.ComboBox)(target));

#line 7 "..\..\..\Settings.xaml"
            this.ComPortCombo.DropDownClosed += new
      System.EventHandler(this.PortSelectionDropDownClosed);

#line default
#line hidden
            return;
            case 2:
            this.PortText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 3:
            this.ApplyButton = ((System.Windows.Controls.Button)(target));
            return;
            case 4:
            this.CloseButton = ((System.Windows.Controls.Button)(target));
            return;
            }
            this._contentLoaded = true;
        }
```

6.29.3.3 InitializeComponent() [1/2] void Emulator.Settings.InitializeComponent ( ) [inline]

InitializeComponent

Definition at line 81 of file Settings.g.cs.

```
                                         {
            if (_contentLoaded) {
                return;
            }
            _contentLoaded = true;
            System.Uri resourceLocater = new System.Uri("/Emulator;component/settings.xaml",
      System.UriKind.Relative);

#line 1 "..\..\..\Settings.xaml"
            System.Windows.Application.LoadComponent(this, resourceLocater);

#line default
#line hidden
        }
```

6.29.3.4 InitializeComponent() [2/2] void Emulator.Settings.InitializeComponent ( ) [inline]

InitializeComponent

Definition at line 81 of file Settings.g.i.cs.

```
                                         {
            if (_contentLoaded) {
                return;
            }
            _contentLoaded = true;
            System.Uri resourceLocater = new System.Uri("/Emulator;component/settings.xaml",
      System.UriKind.Relative);

#line 1 "..\..\..\Settings.xaml"
            System.Windows.Application.LoadComponent(this, resourceLocater);

#line default
#line hidden
        }
```

6.29.3.5 NotificationMessageReceived() [1/2] void Emulator.Settings.NotificationMessageReceived (

NotificationMessage *notificationMessage*) [inline], [private]

Definition at line 21 of file Settings.xaml.cs.

```
        {
            if (notificationMessage.Notification == "CloseSettingsWindow")
            {
                Close();
            }
        }
```

6.29.3.6 NotificationMessageReceived() [2/2] void Emulator.Settings.NotificationMessageReceived (

NotificationMessage< SettingsModel > *notificationMessage*) [inline], [private]

Definition at line 29 of file Settings.xaml.cs.

```
        {
            if (notificationMessage.Notification == "SettingsWindow")
            {
                SettingsViewModel.SettingsModel = notificationMessage.Content;
                ComPortCombo.SelectedItem = notificationMessage.Content.ComPortName;
            }
        }
```

6.29.3.7 PortSelectionDropDownClosed() `void Emulator.Settings.PortSelectionDropDownClosed ( object sender, EventArgs e ) [inline], [private]`

Definition at line 38 of file Settings.xaml.cs.

```
        {
            if (!(ComPortCombo.SelectedValue == null))
            {
                string port = ComPortCombo.SelectedValue.ToString();
                SettingsViewModel.ComPortSelection = port;
            }
        }
```

6.29.4 Member Data Documentation

6.29.4.1 _contentLoaded `bool Emulator.Settings._contentLoaded [private]`

Definition at line 74 of file Settings.g.cs.

The documentation for this class was generated from the following files:

- Emulator/obj/x86/Debug/Settings.g.cs
- Emulator/obj/x86/Debug/Settings.g.i.cs
- Emulator/Settings.xaml.cs

6.30 Emulator.SettingsFile Class Reference

Static Public Member Functions

- static SettingsModel CreateNew ()

6.30.1 Detailed Description

Definition at line 6 of file SettingsFile.cs.

6.30.2 Member Function Documentation

6.30.2.1 CreateNew() `static SettingsModel Emulator.SettingsFile.CreateNew ( ) [inline], [static]`

Definition at line 8 of file SettingsFile.cs.

```
        {
            // Create new settings file.
            SettingsModel _settings = new SettingsModel
            {
                SettingsVersionMajor = Versioning.SettingsFile.Major,
                SettingsVersionMinor = Versioning.SettingsFile.Minor,
                SettingsVersionBuild = Versioning.SettingsFile.Build,
                SettingsVersionRevision = Versioning.SettingsFile.Revision,
#if DEBUG
                ComPortName = "COM9",
#else
                ComPortName = "COM1",
#endif
            };
            return _settings;
        }
```

The documentation for this class was generated from the following file:

- Emulator/Classes/SettingsFile.cs

6.31 Emulator.Versioning.SettingsFile Class Reference

Static Public Attributes

- const byte Major = 1
- const byte Minor = 0
- const byte Build = 0
- const byte Revision = 0

6.31.1 Detailed Description

Definition at line 22 of file Versioning.cs.

6.31.2 Member Data Documentation

6.31.2.1 Build `const byte Emulator.Versioning.SettingsFile.Build = 0 [static]`

Definition at line 26 of file Versioning.cs.

6.31.2.2 Major `const byte Emulator.Versioning.SettingsFile.Major = 1 [static]`

Definition at line 24 of file Versioning.cs.

6.31.2.3 Minor `const byte Emulator.Versioning.SettingsFile.Minor = 0 [static]`

Definition at line 25 of file Versioning.cs.

6.31.2.4 Revision `const byte Emulator.Versioning.SettingsFile.Revision = 0 [static]`

Definition at line 27 of file Versioning.cs.

The documentation for this class was generated from the following file:

- Emulator/Classes/Versioning.cs

6.32 Emulator.Model.SettingsModel Class Reference

Model that contains the required information needed to save the current settings to disk

Properties

- byte SettingsVersionMajor `[get, set]`
  The version of the file that is being saved
- byte SettingsVersionMinor `[get, set]`
  The version of the file that is being saved
- byte SettingsVersionBuild `[get, set]`
  The version of the file that is being saved
- byte SettingsVersionRevision `[get, set]`
  The version of the file that is being saved
- string ComPortName `[get, set]`
  The PC port that is being saved

6.32.1 Detailed Description

Model that contains the required information needed to save the current settings to disk

Definition at line 11 of file SettingsModel.cs.

6.32.2 Property Documentation

6.32.2.1 ComPortName `string Emulator.Model.SettingsModel.ComPortName [get], [set]`

The PC port that is being saved

Definition at line 36 of file SettingsModel.cs.

```
00036 { get; set; }
```

6.32.2.2 SettingsVersionBuild `byte Emulator.Model.SettingsModel.SettingsVersionBuild [get], [set]`

The version of the file that is being saved

Definition at line 26 of file SettingsModel.cs.

```
00026 { get; set; }
```

6.32.2.3 SettingsVersionMajor `byte Emulator.Model.SettingsModel.SettingsVersionMajor [get], [set]`

The version of the file that is being saved

Definition at line 16 of file SettingsModel.cs.

```
00016 { get; set; }
```

6.32.2.4 SettingsVersionMinor `byte Emulator.Model.SettingsModel.SettingsVersionMinor [get], [set]`

The version of the file that is being saved

Definition at line 21 of file SettingsModel.cs.

```
00021 { get; set; }
```

6.32.2.5 SettingsVersionRevision `byte Emulator.Model.SettingsModel.SettingsVersionRevision [get], [set]`

The version of the file that is being saved

Definition at line 31 of file SettingsModel.cs.

```
00031 { get; set; }
```

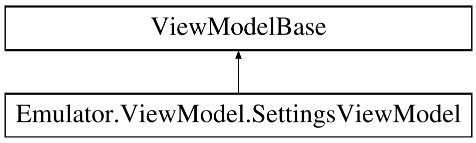
The documentation for this class was generated from the following file:

- Emulator/Model/SettingsModel.cs

6.33 Emulator.ViewModel.SettingsViewModel Class Reference

The ViewModel Used by the SaveFileView

Inheritance diagram for Emulator.ViewModel.SettingsViewModel:



Public Member Functions

- SettingsViewModel ()

  Instantiates a new instance of the SettingsViewModel. This is used by the IOC to create the default instance.
- SettingsViewModel (SettingsModel settingsModel)

  Instantiates a new instance of the SettingsViewModel
- void UpdatePortList ()

  Updates PortList with the COM ports available to the computer

Properties

- RelayCommand ApplyCommand `[get, set]`

  The Relay Command called when saving a file
- RelayCommand CloseCommand `[get, set]`

  The Relay Command called when closing a file
- bool ApplyEnabled `[get]`

  Tells the UI that that a file has been selected and can be saved.
- ObservableCollection< string > PortList `[get]`

  Creates a new instance of PortList, the list of all COM ports available to the computer
- static string ComPortSelection `[get, set]`
- static SettingsModel SettingsModel `[get, set]`

Private Member Functions

- void Apply ()

Static Private Member Functions

- static void Close ()

Private Attributes

- readonly ObservableCollection< string > _PortList = new ObservableCollection<string>()

6.33.1 Detailed Description

The ViewModel Used by the SaveFileView

Definition at line 17 of file SettingsViewModel.cs.

6.33.2 Constructor & Destructor Documentation

6.33.2.1 SettingsViewModel() [1/2] Emulator.ViewModel.SettingsViewModel.SettingsViewModel ( ) [inline]

Instantiates a new instance of the SettingsViewModel. This is used by the IOC to create the default instance.

Definition at line 51 of file SettingsViewModel.cs.

```
        {
        }
```

6.33.2.2 SettingsViewModel() [2/2] Emulator.ViewModel.SettingsViewModel.SettingsViewModel (SettingsModel *settingsModel*) [inline]

Instantiates a new instance of the SettingsViewModel

Parameters

settingsModel	The SettingsFileModel to be serialized to a file

Definition at line 60 of file SettingsViewModel.cs.

```
        {
            ApplyCommand = new RelayCommand(Apply);
            CloseCommand = new RelayCommand(Close);
            ComPortSelection = settingsModel.ComPortName;

            UpdatePortList();
        }
```

6.33.3 Member Function Documentation

6.33.3.1 Apply() `void Emulator.ViewModel.SettingsViewModel.Apply ( ) [inline], [private]`

Definition at line 84 of file SettingsViewModel.cs.

```
        {
            Messenger.Default.Send(new NotificationMessage<SettingsModel>(new SettingsModel
            {
                SettingsVersionMajor = Versioning.SettingsFile.Major,
                SettingsVersionMinor = Versioning.SettingsFile.Minor,
                SettingsVersionBuild = Versioning.SettingsFile.Build,
                SettingsVersionRevision = Versioning.SettingsFile.Revision,
                ComPortName = ComPortSelection,
            }, "SettingsApplied"));
            Messenger.Default.Send(new NotificationMessage("CloseSettingsWindow"));
        }
```

6.33.3.2 Close() `static void Emulator.ViewModel.SettingsViewModel.Close ( ) [inline], [static], [private]`

Definition at line 97 of file SettingsViewModel.cs.

```
        {
            Messenger.Default.Send(new NotificationMessage("CloseSettingsWindow"));
        }
```

6.33.3.3 UpdatePortList() `void Emulator.ViewModel.SettingsViewModel.UpdatePortList ( ) [inline]`

Updates PortList with the COM ports available to the computer

Definition at line 72 of file SettingsViewModel.cs.

```
        {
            PortList.Clear();
            foreach (string s in SerialPort.GetPortNames())
            {
                PortList.Add(s);
            }
            RaisePropertyChanged("PortList");
        }
```

6.33.4 Member Data Documentation

6.33.4.1 _PortList `readonly ObservableCollection<string> Emulator.ViewModel.SettingsView↩ Model._PortList = new ObservableCollection<string>() [private]`

Definition at line 40 of file SettingsViewModel.cs.

6.33.5 Property Documentation

6.33.5.1 ApplyCommand `RelayCommand Emulator.ViewModel.SettingsViewModel.ApplyCommand [get], [set]`

The Relay Command called when saving a file

Definition at line 23 of file SettingsViewModel.cs.

```
00023 { get; set; }
```

6.33.5.2 ApplyEnabled `bool Emulator.ViewModel.SettingsViewModel.ApplyEnabled [get]`

Tells the UI that that a file has been selected and can be saved.

Definition at line 33 of file SettingsViewModel.cs.

```
00033 { get { return !string.IsNullOrEmpty(Emulator.FileLocations.SettingsFile); } }
```

6.33.5.3 CloseCommand `RelayCommand Emulator.ViewModel.SettingsViewModel.CloseCommand [get], [set]`

The Relay Command called when closing a file

Definition at line 28 of file SettingsViewModel.cs.

```
00028 { get; set; }
```

6.33.5.4 ComPortSelection `string Emulator.ViewModel.SettingsViewModel.ComPortSelection [static], [get], [set]`

Definition at line 42 of file SettingsViewModel.cs.

```
00042 { get; set; }
```

6.33.5.5 PortList `ObservableCollection<string> Emulator.ViewModel.SettingsViewModel.PortList [get]`

Creates a new instance of PortList, the list of all COM ports available to the computer

Definition at line 39 of file SettingsViewModel.cs.

```
00039 { get { return _PortList; } }
```

6.33.5.6 SettingsModel `SettingsModel Emulator.ViewModel.SettingsViewModel.SettingsModel [static], [get], [set]`

Definition at line 43 of file SettingsViewModel.cs.

```
00043 { get; set; }
```

The documentation for this class was generated from the following file:

- Emulator/ViewModel/SettingsViewModel.cs

6.34 Hardware.MemoryMap.SharedRom Class Reference

Static Public Attributes

- static byte TotalBanks = 1

Properties

- static int Offset `[get]`
- static int Length `[get]`

Static Private Attributes

- static int _Offset = 0xE000
- static int _Length = 0x1FFF

6.34.1 Detailed Description

Definition at line 45 of file MemoryMap.cs.

6.34.2 Member Data Documentation

6.34.2.1 _Length `int Hardware.MemoryMap.SharedRom._Length = 0x1FFF [static], [private]`

Definition at line 48 of file MemoryMap.cs.

6.34.2.2 _Offset `int Hardware.MemoryMap.SharedRom._Offset = 0xE000 [static], [private]`

Definition at line 47 of file MemoryMap.cs.

6.34.2.3 TotalBanks `byte Hardware.MemoryMap.SharedRom.TotalBanks = 1 [static]`

Definition at line 50 of file MemoryMap.cs.

6.34.3 Property Documentation

6.34.3.1 Length `int Hardware.MemoryMap.SharedRom.Length [static], [get]`

Definition at line 53 of file MemoryMap.cs.

```
00053 { get { return _Length; } }
```

6.34.3.2 Offset `int Hardware.MemoryMap.SharedRom.Offset [static], [get]`

Definition at line 52 of file MemoryMap.cs.

```
00052 { get { return _Offset; } }
```

The documentation for this class was generated from the following file:

- Hardware/Classes/MemoryMap.cs

6.35 Emulator.Model.StateFileModel Class Reference

Model that contains the required information needed to save the current state of the processor to disk

Properties

- int NumberOfCycles `[get, set]`

  The Number of Cycles the Program has Ran so Far
- IList< OutputLog > OutputLog `[get, set]`

  The output of the program
- Hardware.W65C02 W65C02 `[get, set]`

  The Processor Object that is being saved
- Hardware.W65C22 W65C22 `[get, set]`

  The first VIA Object that is being saved
- Hardware.W65C22 MM65SIB `[get, set]`

  The second VIA Object that is being saved
- Hardware.W65C51 W65C51 `[get, set]`

  The ACIA Object that is being saved
- Hardware.AT28CXX AT28C010 `[get, set]`

  The Shared ROM Object that is being saved
- Hardware.AT28CXX AT28C64 `[get, set]`

  The Banked ROM Object that is being saved

6.35.1 Detailed Description

Model that contains the required information needed to save the current state of the processor to disk

Definition at line 10 of file StateFileModel.cs.

6.35.2 Property Documentation

6.35.2.1 AT28C010 `Hardware.AT28CXX Emulator.Model.StateFileModel.AT28C010 [get], [set]`

The Shared ROM Object that is being saved

Definition at line 45 of file StateFileModel.cs.
```
00045 { get; set; }
```

6.35.2.2 AT28C64 `Hardware.AT28CXX Emulator.Model.StateFileModel.AT28C64 [get], [set]`

The Banked ROM Object that is being saved

Definition at line 50 of file StateFileModel.cs.
```
00050 { get; set; }
```

6.35.2.3 MM65SIB `Hardware.W65C22 Emulator.Model.StateFileModel.MM65SIB [get], [set]`

The second VIA Object that is being saved

Definition at line 35 of file StateFileModel.cs.
```
00035 { get; set; }
```

6.35.2.4 NumberOfCycles `int Emulator.Model.StateFileModel.NumberOfCycles [get], [set]`

The Number of Cycles the Program has Ran so Far

Definition at line 15 of file StateFileModel.cs.
```
00015 { get; set; }
```

6.35.2.5 OutputLog `IList<OutputLog> Emulator.Model.StateFileModel.OutputLog [get], [set]`

The output of the program

Definition at line 20 of file StateFileModel.cs.
```
00020 { get; set; }
```

6.35.2.6 W65C02 `Hardware.W65C02 Emulator.Model.StateFileModel.W65C02 [get], [set]`

The Processor Object that is being saved

Definition at line 25 of file StateFileModel.cs.
```
00025 { get; set; }
```

6.35.2.7 W65C22 `Hardware.W65C22 Emulator.Model.StateFileModel.W65C22 [get], [set]`

The first VIA Object that is being saved

Definition at line 30 of file StateFileModel.cs.

```
00030 { get; set; }
```

6.35.2.8 W65C51 `Hardware.W65C51 Emulator.Model.StateFileModel.W65C51 [get], [set]`

The ACIA Object that is being saved

Definition at line 40 of file StateFileModel.cs.

```
00040 { get; set; }
```

The documentation for this class was generated from the following file:

- Emulator/Model/StateFileModel.cs

6.36 Hardware.Utility Class Reference

Static Public Member Functions

- static string ConvertOpCodeIntoString (this int i)

6.36.1 Detailed Description

Definition at line 5 of file Utility.cs.

6.36.2 Member Function Documentation

6.36.2.1 ConvertOpCodeIntoString() `static string Hardware.Utility.ConvertOpCodeIntoString ( this int i ) [inline], [static]`

Definition at line 7 of file Utility.cs.

```
        {
            switch (i)
            {
                case 0x69:    //ăADCăImmediate
                case 0x65:    //ăADCăZeroăPage
                case 0x75:    //ăADCăZeroăPageăX
                case 0x6D:  //ăADCăAbsolute
                case 0x7D:  //ăADCăAbsoluteăX
                case 0x79:    //ăADCăAbsoluteăY
                case 0x61:    //ăADCăIndrectăX
                case 0x71:    //ăADCăIndirectăY
                    {
                        return "ADC";
                    }
                case 0x29:    //ăANDăImmediate
                case 0x25:    //ăANDăZeroăPage
                case 0x35:    //ăANDăZeroăPageăX
                case 0x2D:  //ăANDăAbsolute
                case 0x3D:  //ăANDăAbsoluteăX
                case 0x39:    //ăANDăAbsoluteăY
                case 0x21:    //ăANDăIndirectăX
                case 0x31:    //ăANDăIndirectăY
                    {
                        return "AND";
                    }
                case 0x0A:  //ăASLăAccumulator
                case 0x06:    //ăASLăZeroăPage
                case 0x16:    //ăASLăZeroăPageăX
                case 0x0E:  //ăASLăAbsolute
                case 0x1E:  //ăASLăAbsoluteăX
                    {
                        return "ASL";
                    }
                case 0x90:    //ăBCCăRelative
                    {
                        return "BCC";
                    }
                case 0xB0:    //ăBCSăRelative
                    {
                        return "BCS";
                    }
                case 0xF0:    //ăBEQăRelative
                    {
                        return "BEQ";
                    }
                case 0x24:    //ăBITăZeroăPage
                case 0x2C:  //ăBITăAbsolute
                    {
                        return "BIT";
                    }
                case 0x30:    //ăBMIăRelative
                    {
                        return "BMI";
                    }
                case 0xD0:    //ăBNEăRelative
                    {
                        return "BNE";
                    }
                case 0x10:    //ăBPLăRelative
                    {
                        return "BPL";
                    }
                case 0x00:    //ăBRKăImplied
                    {
                        return "BRK";
                    }
                case 0x50:  // BVC Relative
                    {
                        return "BCV";
                    }
                case 0x70:  //BVS Relative
                    {
                        return "BVS";
                    }
                case 0x18:    //ăCLCăImplied
                    {
                        return "CLC";
                    }
                case 0xD8:    //ăCLDăImplied
                    {
                        return "CLD";
                    }
                case 0x58:    //ăCLIăImplied
                    {
                        return "CLI";
                    }
                case 0xB8:    //ăCLVăImplied
                    {
                        return "CLV";
                    }
                case 0xC9:    //ăCMPăImmediate
                case 0xC5:    //ăCMPăZeroPage
                case 0xD5:    //ăCMPăZeroăPageăX
                case 0xCD:  //ăCMPăAbsolute
                case 0xDD:  //ăCMPăAbsoluteăX
                case 0xD9:    //ăCMPăAbsoluteăY
                case 0xC1:    //ăCMPăIndirectăX
                case 0xD1:    //ăCMPăIndirectăY
                    {
                        return "CMP";
                    }
                case 0xE0:    //ăCPXăImmediate
                case 0xE4:    //ăCPXăZeroPage
                case 0xEC:  //ăCPXăAbsolute
                    {
                        return "CPX";
                    }
                case 0xC0:    //ăCPYăImmediate
                case 0xC4:    //ăCPYăZeroPage
                case 0xCC:  //ăCPYăAbsolute
                    {
                        return "CPY";
                    }
                case 0xC6:    //ăDECăZeroăPage
                case 0xD6:    //ăDECăZeroăPageăX
                case 0xCE:  //ăDECăAbsolute
                case 0xDE:  //ăDECăAbsoluteăX
                    {
                        return "DEC";
                    }
                case 0xCA:  //ăDEXăImplied
                    {
                        return "DEX";
                    }
                case 0x88:    //ăDEYăImplied
                    {
                        return "DEY";
                    }
                case 0x49:    //ăEORăImmediate
                case 0x45:    //ăEORăZeroăPage
                case 0x55:    //ăEORăZeroăPageăX
                case 0x4D:  //ăEORăAbsolute
                case 0x5D:  //ăEORăAbsoluteăX
                case 0x59:    //ăEORăAbsoluteăY
                case 0x41:    //ăEORăIndrectăX
                case 0x51:    //ăEORăIndirectăY
                    {
                        return "EOR";
                    }
                case 0xE6:    //ăINCăZeroăPage
                case 0xF6:    //ăINCăZeroăPageăX
                    {
                        return "INC";
                    }
                case 0xE8:    //ăINXăImplied
                    {
                        return "INX";
                    }
                case 0xC8:    //ăINYăImplied
                    {
                        return "INY";
                    }
                case 0xEE:  //ăINCăAbsolute
                case 0xFE:  //ăINCăAbsoluteăX
                    {
                        return "INC";
                    }
                case 0x4C:  //ăJMPăAbsolute
                case 0x6C:  //ăJMPăIndirect
                    {
                        return "JMP";
                    }
                case 0x20:    //ăJSRăAbsolute
                    {
                        return "JSR";
                    }
                case 0xA9:    //ăLDAăImmediate
                case 0xA5:    //ăLDAăZeroăPage
                case 0xB5:    //ăLDAăZeroăPageăX
                case 0xAD:  //ăLDAăAbsolute
                case 0xBD:  //ăLDAăAbsoluteăX
                case 0xB9:    //ăLDAăAbsoluteăY
                case 0xA1:    //ăLDAăIndirectăX
                case 0xB1:    //ăLDAăIndirectăY
                    {
                        return "LDA";
                    }
                case 0xA2:    //ăLDXăImmediate
                case 0xA6:    //ăLDXăZeroăPage
                case 0xB6:    //ăLDXăZeroăPageăY
                case 0xAE:  //ăLDXăAbsolute
                case 0xBE:  //ăLDXăAbsoluteăY
                    {
                        return "LDX";
                    }
                case 0xA0:    //ăLDYăImmediate
                case 0xA4:    //ăLDYăZeroăPage
                case 0xB4:    //ăLDYăZeroăPageăY
                case 0xAC:  //ăLDYăAbsolute
                case 0xBC:  //ăLDYăAbsoluteăY
                    {
                        return "LDY";
                    }
                case 0x4A:  //ăLSRăAccumulator
                case 0x46:    //ăLSRăZeroăPage
                case 0x56:    //ăLSRăZeroăPageăX
                case 0x4E:  //ăLSRăAbsolute
                case 0x5E:  //ăLSRăAbsoluteăX
                    {
                        return "LSR";
                    }
                case 0xEA:  //ăNOPăImplied
                    {
                        return "NOP";
                    }
                case 0x09:    //ăORAăImmediate
                case 0x05:    //ăORAăZeroăPage
                case 0x15:    //ăORAăZeroăPageăX
                case 0x0D:  //ăORAăAbsolute
                case 0x1D:  //ăORAăAbsoluteăX
                case 0x19:    //ăORAăAbsoluteăY
                case 0x01:    //ăORAăIndirectăX
                case 0x11:    //ăORAăIndirectăY
                    {
                        return "ORA";
                    }
                case 0x48:    //ăPHAăImplied
                    {
                        return "PHA";
                    }
                case 0x08:    //ăPHPăImplied
                    {
                        return "PHP";
                    }
                case 0x68:    //ăPLAăImplied
                    {
                        return "PLA";
                    }
                case 0x28:    //ăPLPăImplied
                    {
                        return "PLP";
                    }
                case 0x2A:  //ăROLăAccumulator
                case 0x26:    //ăROLăZeroăPage
                case 0x36:    //ăROLăZeroăPageăX
                case 0x2E:  //ăROLăAbsolute
                case 0x3E:  //ăROLăAbsoluteăX
                    {
                        return "ROL";
                    }
                case 0x6A:  //ăRORăAccumulator
                case 0x66:    //ăRORăZeroăPage
                case 0x76:    //ăRORăZeroăPageăX
                case 0x6E:  //ăRORăAbsolute
                case 0x7E:  //ăRORăAbsoluteăX
                    {
                        return "ROR";
                    }
                case 0x40:    //ăRTIăImplied
                    {
                        return "RTI";
                    }
                case 0x60:    //ăRTSăImplied
                    {
                        return "RTS";
                    }
                case 0xE9:    //ăSBCăImmediate
                case 0xE5:    //ăSBCăZeroăPage
                case 0xF5:    //ăSBCăZeroăPageăX
                case 0xED:  //ăSBCăAbsolute
                case 0xFD:  //ăSBCăAbsoluteăX
                case 0xF9:    //ăSBCăAbsoluteăY
                case 0xE1:    //ăSBCăIndrectăX
                case 0xF1:    //ăSBCăIndirectăY
                    {
                        return "SBC";
                    }
                case 0x38:    //ăSECăImplied
                    {
                        return "SEC";
                    }
                case 0xF8:    //ăSEDăImplied
                    {
                        return "SED";
                    }
                case 0x78:    //ăSEIăImplied
                    {
                        return "SEI";
                    }
                case 0x85:    //ăSTAăZeroPage
                case 0x95:    //ăSTAăZeroăPageăX
                case 0x8D:  //ăSTAăAbsolute
                case 0x9D:  //ăSTAăAbsoluteăX
                case 0x99:    //ăSTAăAbsoluteăY
                case 0x81:    //ăSTAăIndirectăX
                case 0x91:    //ăSTAăIndirectăY
                    {
                        return "STA";
                    }
                case 0x86:    //ăSTXăZeroăPage
                case 0x96:    //ăSTXăZeroăPageăY
                case 0x8E:  //ăSTXăAbsolute
                    {
                        return "STX";
                    }
                case 0x84:    //ăSTYăZeroăPage
                case 0x94:    //ăSTYăZeroăPageăX
                case 0x8C:  //ăSTYăAbsolute
                    {
                        return "STY";
                    }
                case 0xAA:  //ăTAXăImplied
                    {
                        return "TAX";
                    }
                case 0xA8:    //ăTAYăImplied
                    {
                        return "TAY";
                    }
                case 0xBA:  //ăTSXăImplied
                    {
                        return "TSX";
                    }
                case 0x8A:  //ăTXAăImplied
                    {
                        return "TXA";
                    }
                case 0x9A:  //ăTXSăImplied
                    {
                        return "TXS";
                    }
                case 0x98:    //ăTYAăImplied
                    {
                        return "TYA";
                    }
                default:
                    throw new InvalidEnumArgumentException(string.Format("A Valid Conversion does not
exist for OpCode {0}", i.ToString("X")));

            }
        }
```

The documentation for this class was generated from the following file:

- Hardware/Classes/Utility.cs

6.37 Emulator.Versioning Class Reference

Classes

- class Product
- class SettingsFile

6.37.1 Detailed Description

Definition at line 7 of file Versioning.cs.

The documentation for this class was generated from the following file:

- Emulator/Classes/Versioning.cs

6.38 Emulator.ViewModel.ViewModelLocator Class Reference

This class contains static references to all the view models in the application and provides an entry point for the bindings.

Public Member Functions

- ViewModelLocator ()

  Initializes a new instance of the ViewModelLocator class.

Static Public Member Functions

- static void Cleanup ()

  The Cleanup Method

Properties

- MainViewModel Main `[get]`

  The MainViewModel Instance
- SaveFileViewModel SaveFile `[get]`

  The SaveFileViewModel Instance
- SettingsViewModel Settings `[get]`

  The SaveFileViewModel Instance

6.38.1 Detailed Description

This class contains static references to all the view models in the application and provides an entry point for the bindings.

Definition at line 24 of file ViewModelLocator.cs.

6.38.2 Constructor & Destructor Documentation

6.38.2.1 ViewModelLocator() `Emulator.ViewModel.ViewModelLocator.ViewModelLocator ( ) [inline]`

Initializes a new instance of the ViewModelLocator class.

Definition at line 29 of file ViewModelLocator.cs.

```
        {
            ServiceLocator.SetLocatorProvider(() => SimpleIoc.Default);

            SimpleIoc.Default.Register<MainViewModel>();
            SimpleIoc.Default.Register<SaveFileViewModel>();
            SimpleIoc.Default.Register<SettingsViewModel>();
        }
```

6.38.3 Member Function Documentation

6.38.3.1 Cleanup() `static void Emulator.ViewModel.ViewModelLocator.Cleanup ( ) [inline], [static]`

The Cleanup Method

<todo> Clear the ViewModels </todo>

Definition at line 65 of file ViewModelLocator.cs.

```
        {
/// <todo>
/// Clear the ViewModels
/// </todo>
        }
```

6.38.4 Property Documentation

6.38.4.1 Main `MainViewModel Emulator.ViewModel.ViewModelLocator.Main [get]`

The MainViewModel Instance

Definition at line 41 of file ViewModelLocator.cs.

```
        {
            get { return ServiceLocator.Current.GetInstance<MainViewModel>(); }
        }
```

6.38.4.2 SaveFile `SaveFileViewModel Emulator.ViewModel.ViewModelLocator.SaveFile [get]`

The SaveFileViewModel Instance

Definition at line 49 of file ViewModelLocator.cs.

```
        {
            get { return ServiceLocator.Current.GetInstance<SaveFileViewModel>(); }
        }
```

6.38.4.3 Settings `SettingsViewModel Emulator.ViewModel.ViewModelLocator.Settings [get]`

The SaveFileViewModel Instance

Definition at line 57 of file ViewModelLocator.cs.

```
        {
            get { return ServiceLocator.Current.GetInstance<SettingsViewModel>(); }
        }
```

The documentation for this class was generated from the following file:

- Emulator/ViewModel/ViewModelLocator.cs

6.39 Hardware.W65C02 Class Reference

An implementation of a W65C02 Processor.

Public Member Functions

- W65C02 ()
  Default Constructor, Instantiates a new instance of the processor.
- void Reset ()
  Initializes the processor to its default state.
- void NextStep ()
  Performs the next step on the processor
- void InterruptRequest ()
  The InterruptRequest or IRQ
- int GetCycleCount ()
  Gets the Number of Cycles that have elapsed
- void IncrementCycleCount ()
  Increments the Cycle Count, causes a CycleCountIncrementedAction to fire.
- void ResetCycleCount ()
  Resets the Cycle Count back to 0
- void AslOperation (AddressingMode addressingMode)
  The ASL - Shift Left One Bit (Memory or Accumulator)

Public Attributes

- bool isRunning
  Checks shether the emulated computer is running or not.

Protected Member Functions

- void SetNegativeFlag (int value)

    Sets the IsSignNegative register
- void SetZeroFlag (int value)

    Sets the IsResultZero register
- int GetAddressByAddressingMode (AddressingMode addressingMode)

    Uses the AddressingMode to return the correct address based on the mode. Note: This method will not increment the program counter for any mode. Note: This method will return an error if called for either the immediate or accumulator modes.
- void AddWithCarryOperation (AddressingMode addressingMode)

    The ADC - Add Memory to Accumulator with Carry Operation
- void SubtractWithBorrowOperation (AddressingMode addressingMode)

    The SBC operation. Performs a subtract with carry operation on the accumulator and a value in memory.

Properties

- int Accumulator `[get, protected set]`

    The Accumulator. This value is implemented as an integer intead of a byte. This is done so we can detect wrapping of the value and set the correct number of cycles.
- int XRegister `[get, private set]`

    The X Index Register
- int YRegister `[get, private set]`

    The Y Index Register
- int CurrentOpCode `[get, private set]`

    The Current Op Code being executed by the system
- Disassembly CurrentDisassembly `[get, private set]`

    The disassembly of the current operation. This value is only set when the CPU is built in debug mode.
- int ProgramCounter `[get, private set]`

    Points to the Current Address of the instruction being executed by the system. The PC wraps when the value is greater than 65535, or less than 0.
- int StackPointer `[get, private set]`

    Points to the Current Position of the Stack. This value is a 00-FF value but is offset to point to the location in memory where the stack resides.
- Action CycleCountIncrementedAction `[get, set]`

    An external action that occurs when the cycle count is incremented
- bool CarryFlag `[get, protected set]`

    This is the carry flag. when adding, if the result is greater than 255 or 99 in BCD Mode, then this bit is enabled. In subtraction this is reversed and set to false if a borrow is required IE the result is less than 0
- bool ZeroFlag `[get, private set]`

    Is true if one of the registers is set to zero.
- bool DisableInterruptFlag `[get, private set]`

    This determines if Interrupts are currently disabled. This flag is turned on during a reset to prevent an interrupt from occuring during startup/Initialization. If this flag is true, then the IRQ pin is ignored.
- bool DecimalFlag `[get, private set]`

    Binary Coded Decimal Mode is set/cleared via this flag. when this mode is in effect, a byte represents a number from 0-99.
- bool OverflowFlag `[get, protected set]`

    This property is set when an overflow occurs. An overflow happens if the high bit(7) changes during the operation. Remember that values from 128-256 are negative values as the high bit is set to 1. Examples: 64 + 64 = -128 -128 + -128 = 0
- bool NegativeFlag `[get, private set]`

Set to true if the result of an operation is negative in ADC and SBC operations. Remember that 128-256 represent negative numbers when doing signed math. In shift operations the sign holds the carry.

- bool TriggerNmi `[get, set]`

  Set to true when an NMI should occur

- bool TriggerIRQ `[get, private set]`

  Set to true when an IRQ has occurred and is being processed by the CPU.

Private Member Functions

- void ExecuteOpCode ()

  Executes an Opcode

- void MoveProgramCounterByRelativeValue (byte valueToMove)

  Moves the ProgramCounter in a given direction based on the value inputted

- byte PeekStack ()

  Returns a the value from the stack without changing the position of the stack pointer

- void PokeStack (byte value)

  Write a value directly to the stack without modifying the Stack Pointer

- byte ConvertFlagsToByte (bool setBreak)

  Coverts the Flags into its byte representation.

- void SetDisassembly ()
- int WrapProgramCounter (int value)
- AddressingMode GetAddressingMode ()
- void AndOperation (AddressingMode addressingMode)

  The AND - Compare Memory with Accumulator operation

- void BranchOperation (bool performBranch)

  Performs the different branch operations.

- void BitOperation (AddressingMode addressingMode)

  The bit operation, does an & comparison between a value in memory and the accumulator

- void CompareOperation (AddressingMode addressingMode, int comparisonValue)

  The compare operation. This operation compares a value in memory with a value passed into it.

- void ChangeMemoryByOne (AddressingMode addressingMode, bool decrement)

  Changes a value in memory by 1

- void ChangeRegisterByOne (bool useXRegister, bool decrement)

  Changes a value in either the X or Y register by 1

- void EorOperation (AddressingMode addressingMode)

  The EOR Operation, Performs an Exclusive OR Operation against the Accumulator and a value in memory

- void LsrOperation (AddressingMode addressingMode)

  The LSR Operation. Performs a Left shift operation on a value in memory

- void OrOperation (AddressingMode addressingMode)

  The Or Operation. Performs an Or Operation with the accumulator and a value in memory

- void RolOperation (AddressingMode addressingMode)

  The ROL operation. Performs a rotate left operation on a value in memory.

- void RorOperation (AddressingMode addressingMode)

  The ROR operation. Performs a rotate right operation on a value in memory.

- void PushFlagsOperation ()

  The PSP Operation. Pushes the Status Flags to the stack

- void PullFlagsOperation ()

  The PLP Operation. Pull the status flags off the stack on sets the flags accordingly.

- void JumpToSubRoutineOperation ()

  The JSR routine. Jumps to a subroutine.

- void ReturnFromSubRoutineOperation ()

The RTS routine. Called when returning from a subroutine.

- void BreakOperation (bool isBrk, int vector)

  The BRK routine. Called when a BRK occurs.
- void ReturnFromInterruptOperation ()

  The RTI routine. Called when returning from a BRK opertion. Note: when called after a BRK operation the Program Counter is not set to the location after the BRK, it is set +1
- void ProcessNMI ()

  This is ran anytime an NMI occurrs
- void ProcessIRQ ()

  This is ran anytime an IRQ occurrs

Private Attributes

- readonly ILogger _logger = LogManager.GetLogger("Processor")
- int _programCounter
- int _stackPointer
- int _cycleCount
- bool _previousInterrupt
- bool _interrupt

6.39.1 Detailed Description

An implementation of a W65C02 Processor.

Definition at line 13 of file W65C02.cs.

6.39.2 Constructor & Destructor Documentation

6.39.2.1 W65C02() `Hardware.W65C02.W65C02 ( ) [inline]`

Default Constructor, Instantiates a new instance of the processor.

Definition at line 143 of file W65C02.cs.

```
        {
            StackPointer = 0x100;
            CycleCountIncrementedAction = () => { };
        }
```

6.39.3 Member Function Documentation

6.39.3.1 AddWithCarryOperation() `void Hardware.W65C02.AddWithCarryOperation ( AddressingMode addressingMode ) [inline], [protected]`

The ADC - Add Memory to Accumulator with Carry Operation

Parameters

addressingMode	The addressing mode used to perform this operation.

Definition at line 1888 of file W65C02.cs.

```
        {
            //Accumulator, Carry = Accumulator + ValueInMemoryLocation + Carry
            var memoryValue = MemoryMap.Read(GetAddressByAddressingMode(addressingMode));
            var newValue = memoryValue + Accumulator + (CarryFlag ?  1 :  0);


            OverflowFlag = (((Accumulator ^ newValue) & 0x80) != 0) && (((Accumulator ^ memoryValue) &
    0x80) == 0);

            if (DecimalFlag)
            {
                newValue = int.Parse(memoryValue.ToString("x")) + int.Parse(Accumulator.ToString("x"))
    + (CarryFlag ?  1 :  0);

                if (newValue > 99)
                {
                    CarryFlag = true;
                    newValue -= 100;
                }
                else
                {
                    CarryFlag = false;
                }

                newValue = (int)Convert.ToInt64(string.Concat("0x", newValue), 16);
            }
            else
            {
                if (newValue > 255)
                {
                    CarryFlag = true;
                    newValue -= 256;
                }
                else
                {
                    CarryFlag = false;
                }
            }

            SetZeroFlag(newValue);
            SetNegativeFlag(newValue);

            Accumulator = newValue;
        }
```

6.39.3.2 AndOperation() `void Hardware.W65C02.AndOperation ( AddressingMode addressingMode ) [inline], [private]`

The AND - Compare Memory with Accumulator operation

Parameters

addressingMode	The addressing mode being used

Definition at line 1936 of file W65C02.cs.

```
        {
            Accumulator = MemoryMap.Read(GetAddressByAddressingMode(addressingMode)) & Accumulator;

            SetZeroFlag(Accumulator);
            SetNegativeFlag(Accumulator);
        }
```

6.39.3.3 AslOperation() void Hardware.W65C02.AslOperation (AddressingMode *addressingMode*) [inline]

The ASL - Shift Left One Bit (Memory or Accumulator)

Parameters

addressingMode	The addressing Mode being used

Definition at line 1948 of file W65C02.cs.

```
        {
            int value;
            var memoryAddress = 0;
            if (addressingMode == AddressingMode.Accumulator)
            {
                MemoryMap.Read(ProgramCounter + 1);
                value = Accumulator;
            }
            else
            {
                memoryAddress = GetAddressByAddressingMode(addressingMode);
                value = MemoryMap.Read(memoryAddress);
            }

            //Dummy Write
            if (addressingMode != AddressingMode.Accumulator)
            {
                MemoryMap.Write(memoryAddress, (byte)value);
            }

            //If the 7th bit is set, then we have a carry
            CarryFlag = ((value & 0x80) != 0);

            //The And here ensures that if the value is greater than 255 it wraps properly.
            value = (value « 1) & 0xFE;

            SetNegativeFlag(value);
            SetZeroFlag(value);


            if (addressingMode == AddressingMode.Accumulator)
                Accumulator = value;
            else
            {
                MemoryMap.Write(memoryAddress, (byte)value);
            }
        }
```

6.39.3.4 BitOperation() void Hardware.W65C02.BitOperation (AddressingMode *addressingMode*) [inline], [private]

The bit operation, does an & comparison between a value in memory and the accumulator

Parameters

addressingMode	

Definition at line 2008 of file W65C02.cs.

```
        {

            var memoryValue = MemoryMap.Read(GetAddressByAddressingMode(addressingMode));
            var valueToCompare = memoryValue & Accumulator;

            OverflowFlag = (memoryValue & 0x40) != 0;

            SetNegativeFlag(memoryValue);
            SetZeroFlag(valueToCompare);
        }
```

6.39.3.5 BranchOperation() `void Hardware.W65C02.BranchOperation ( bool performBranch ) [inline], [private]`

Performs the different branch operations.

Parameters

performBranch	Is a branch required

Definition at line 1991 of file W65C02.cs.

```
        {
            var value = MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.Relative));

            if (!performBranch)
            {
                ProgramCounter++;
                return;
            }

            MoveProgramCounterByRelativeValue(value);
        }
```

6.39.3.6 BreakOperation() `void Hardware.W65C02.BreakOperation ( bool isBrk, int vector ) [inline], [private]`

The BRK routine. Called when a BRK occurs.

Definition at line 2354 of file W65C02.cs.

```
        {
            MemoryMap.Read(++ProgramCounter);

            //Put the high value on the stack
            //When we RTI the address will be incremented by one, and the address after a break will
    not be used.
            PokeStack((byte)(((ProgramCounter) » 8) & 0xFF));
            StackPointer--;
            IncrementCycleCount();

            //Put the low value on the stack
            PokeStack((byte)((ProgramCounter) & 0xFF));
            StackPointer--;
            IncrementCycleCount();

            //We only set the Break Flag is a Break Occurs
            if (isBrk)
                PokeStack((byte)(ConvertFlagsToByte(true) | 0x10));
            else
                PokeStack(ConvertFlagsToByte(false));

            StackPointer--;
            IncrementCycleCount();

            DisableInterruptFlag = true;

            ProgramCounter = (MemoryMap.Read(vector + 1) « 8) | MemoryMap.Read(vector);

            _previousInterrupt = false;
        }
```

6.39.3.7 ChangeMemoryByOne() `void Hardware.W65C02.ChangeMemoryByOne ( AddressingMode addressingMode, bool decrement ) [inline], [private]`

Changes a value in memory by 1

Parameters

addressingMode	The addressing mode to use
decrement	If the operation is decrementing or incrementing the vaulue by 1

Definition at line 2044 of file W65C02.cs.

```
        {
            var memoryLocation = GetAddressByAddressingMode(addressingMode);
            var memory = MemoryMap.Read(memoryLocation);

            MemoryMap.Write(memoryLocation, memory);

            if (decrement)
                memory -= 1;
            else
                memory += 1;

            SetZeroFlag(memory);
            SetNegativeFlag(memory);


            MemoryMap.Write(memoryLocation, memory);
        }
```

6.39.3.8 ChangeRegisterByOne()

```
void Hardware.W65C02.ChangeRegisterByOne (
        bool useXRegister,
        bool decrement ) [inline], [private]
```

Changes a value in either the X or Y register by 1

Parameters

useXRegister	If the operation is using the X or Y register
decrement	If the operation is decrementing or incrementing the vaulue by 1

Definition at line 2068 of file W65C02.cs.

```
        {
            var value = useXRegister ? XRegister : YRegister;

            if (decrement)
                value -= 1;
            else
                value += 1;

            if (value < 0x00)
                value += 0x100;
            else if (value > 0xFF)
                value -= 0x100;

            SetZeroFlag(value);
            SetNegativeFlag(value);
            IncrementCycleCount();

            if (useXRegister)
                XRegister = value;
            else
                YRegister = value;
        }
```

6.39.3.9 CompareOperation()

```
void Hardware.W65C02.CompareOperation (
        AddressingMode addressingMode,
        int comparisonValue ) [inline], [private]
```

The compare operation. This operation compares a value in memory with a value passed into it.

Parameters

addressingMode	The addressing mode to use
comparisonValue	The value to compare against memory

Definition at line 2025 of file W65C02.cs.

```
        {
            var memoryValue = MemoryMap.Read(GetAddressByAddressingMode(addressingMode));
            var comparedValue = comparisonValue - memoryValue;

            if (comparedValue < 0)
                comparedValue += 0x10000;

            SetZeroFlag(comparedValue);

            CarryFlag = memoryValue <= comparisonValue;
            SetNegativeFlag(comparedValue);
        }
```

6.39.3.10 ConvertFlagsToByte()

`byte Hardware.W65C02.ConvertFlagsToByte ( bool setBreak ) [inline], [private]`

Coverts the Flags into its byte representation.

Parameters

setBreak	Determines if the break flag should be set during conversion. IRQ does not set the flag on the stack, but PHP and BRK do

Returns

Definition at line 1522 of file W65C02.cs.

```
        {
            return (byte)((CarryFlag ? 0x01 : 0) + (ZeroFlag ? 0x02 : 0) + (DisableInterruptFlag ?
    0x04 : 0) +
                  (DecimalFlag ? 8 : 0) + (setBreak ? 0x10 : 0) + 0x20 + (OverflowFlag ? 0x40 : 0)
    + (NegativeFlag ? 0x80 : 0));
        }
```

6.39.3.11 EorOperation()

`void Hardware.W65C02.EorOperation ( AddressingMode addressingMode ) [inline], [private]`

The EOR Operation, Performs an Exclusive OR Operation against the Accumulator and a value in memory

Parameters

addressingMode	The addressing mode to use

Definition at line 2096 of file W65C02.cs.

```
        {
            Accumulator = Accumulator ^ MemoryMap.Read(GetAddressByAddressingMode(addressingMode));
            SetNegativeFlag(Accumulator);
            SetZeroFlag(Accumulator);
        }
```

6.39.3.12 ExecuteOpCode() `void Hardware.W65C02.ExecuteOpCode ( ) [inline], [private]`

Executes an Opcode

Definition at line 239 of file W65C02.cs.

```
        {
            //The x+ cycles denotes that if a page wrap occurs, then an additional cycle is consumed.
            //The x++ cycles denotes that 1 cycle is added when a branch occurs and it on the same page, and two cycles are added if its on a different page./
            //This is handled inside the GetValueFromMemory Method
            switch (CurrentOpCode)
            {
#region Add / Subtract Operations
                //ADC Add With Carry, Immediate, 2 Bytes, 2 Cycles
                case 0x69:
                    {
                        AddWithCarryOperation(AddressingMode.Immediate);
                        break;
                    }
                //ADC Add With Carry, Zero Page, 2 Bytes, 3 Cycles
                case 0x65:
                    {
                        AddWithCarryOperation(AddressingMode.ZeroPage);
                        break;
                    }
                //ADC Add With Carry, Zero Page X, 2 Bytes, 4 Cycles
                case 0x75:
                    {
                        AddWithCarryOperation(AddressingMode.ZeroPageX);
                        break;
                    }
                //ADC Add With Carry, Absolute, 3 Bytes, 4 Cycles
                case 0x6D:
                    {
                        AddWithCarryOperation(AddressingMode.Absolute);
                        break;
                    }
                //ADC Add With Carry, Absolute X, 3 Bytes, 4+ Cycles
                case 0x7D:
                    {
                        AddWithCarryOperation(AddressingMode.AbsoluteX);
                        break;
                    }
                //ADC Add With Carry, Absolute Y, 3 Bytes, 4+ Cycles
                case 0x79:
                    {
                        AddWithCarryOperation(AddressingMode.AbsoluteY);
                        break;
                    }
                //ADC Add With Carry, Indexed Indirect, 2 Bytes, 6 Cycles
                case 0x61:
                    {
                        AddWithCarryOperation(AddressingMode.IndirectX);
                        break;
                    }
                //ADC Add With Carry, Indexed Indirect, 2 Bytes, 5+ Cycles
                case 0x71:
                    {
                        AddWithCarryOperation(AddressingMode.IndirectY);
                        break;
                    }
                //SBC Subtract with Borrow, Immediate, 2 Bytes, 2 Cycles
                case 0xE9:
                    {
                        SubtractWithBorrowOperation(AddressingMode.Immediate);
                        break;
                    }
                //SBC Subtract with Borrow, Zero Page, 2 Bytes, 3 Cycles
                case 0xE5:
                    {
                        SubtractWithBorrowOperation(AddressingMode.ZeroPage);
                        break;
                    }
                //SBC Subtract with Borrow, Zero Page X, 2 Bytes, 4 Cycles
                case 0xF5:
                    {
                        SubtractWithBorrowOperation(AddressingMode.ZeroPageX);
                        break;
                    }
                //SBC Subtract with Borrow, Absolute, 3 Bytes, 4 Cycles
                case 0xED:
                    {
                        SubtractWithBorrowOperation(AddressingMode.Absolute);
                        break;
                    }
                //SBC Subtract with Borrow, Absolute X, 3 Bytes, 4+ Cycles
                case 0xFD:
                    {
                        SubtractWithBorrowOperation(AddressingMode.AbsoluteX);
                        break;
                    }
                //SBC Subtract with Borrow, Absolute Y, 3 Bytes, 4+ Cycles
                case 0xF9:
                    {
                        SubtractWithBorrowOperation(AddressingMode.AbsoluteY);
                        break;
                    }
                //SBC Subtract with Borrow, Indexed Indirect, 2 Bytes, 6 Cycles
                case 0xE1:
                    {
                        SubtractWithBorrowOperation(AddressingMode.IndirectX);
                        break;
                    }
                //SBC Subtract with Borrow, Indexed Indirect, 2 Bytes, 5+ Cycles
                case 0xF1:
                    {
                        SubtractWithBorrowOperation(AddressingMode.IndirectY);
                        break;
                    }
#endregion

#region Branch Operations
                //BCC Branch if Carry is Clear, Relative, 2 Bytes, 2++ Cycles
                case 0x90:
                    {
                        BranchOperation(!CarryFlag);
                        break;

                    }
                //BCS Branch if Carry is Set, Relative, 2 Bytes, 2++ Cycles
                case 0xB0:
                    {
                        BranchOperation(CarryFlag);
                        break;
                    }
                //BEQ Branch if Zero is Set, Relative, 2 Bytes, 2++ Cycles
                case 0xF0:
                    {
                        BranchOperation(ZeroFlag);
                        break;
                    }

                // BMI Branch if Negative Set
                case 0x30:
                    {
                        BranchOperation(NegativeFlag);
                        break;
                    }
                //BNE Branch if Zero is Not Set, Relative, 2 Bytes, 2++ Cycles
                case 0xD0:
                    {
                        BranchOperation(!ZeroFlag);
                        break;
                    }
                // BPL Branch if Negative Clear, 2 Bytes, 2++ Cycles
                case 0x10:
                    {
                        BranchOperation(!NegativeFlag);
                        break;
                    }
                // BVC Branch if Overflow Clear, 2 Bytes, 2++ Cycles
                case 0x50:
                    {
                        BranchOperation(!OverflowFlag);
                        break;
                    }
                // BVS Branch if Overflow Set, 2 Bytes, 2++ Cycles
                case 0x70:
                    {
                        BranchOperation(OverflowFlag);
                        break;
                    }
#endregion

#region BitWise Comparison Operations
                //AND Compare Memory with Accumulator, Immediate, 2 Bytes, 2 Cycles
                case 0x29:
                    {
                        AndOperation(AddressingMode.Immediate);
                        break;
                    }
                //AND Compare Memory with Accumulator, Zero Page, 2 Bytes, 3 Cycles
                case 0x25:
                    {
                        AndOperation(AddressingMode.ZeroPage);
                        break;
                    }
                //AND Compare Memory with Accumulator, Zero PageX, 2 Bytes, 3 Cycles
                case 0x35:
                    {
                        AndOperation(AddressingMode.ZeroPageX);
                        break;
                    }
                //AND Compare Memory with Accumulator, Absolute,  3 Bytes, 4 Cycles
                case 0x2D:
                    {
                        AndOperation(AddressingMode.Absolute);
                        break;
                    }
                //AND Compare Memory with Accumulator, AbsolueteX 3 Bytes, 4+ Cycles
                case 0x3D:
                    {
                        AndOperation(AddressingMode.AbsoluteX);
                        break;
                    }
                //AND Compare Memory with Accumulator, AbsoluteY, 3 Bytes, 4+ Cycles
                case 0x39:
                    {
                        AndOperation(AddressingMode.AbsoluteY);
                        break;
                    }
                //AND Compare Memory with Accumulator, IndexedIndirect, 2 Bytes, 6 Cycles
                case 0x21:
                    {
                        AndOperation(AddressingMode.IndirectX);
                        break;
                    }
                //AND Compare Memory with Accumulator, IndirectIndexed, 2 Bytes, 5 Cycles
                case 0x31:
                    {
                        AndOperation(AddressingMode.IndirectY);
                        break;
                    }
                //BIT Compare Memory with Accumulator, Zero Page, 2 Bytes, 3 Cycles
                case 0x24:
                    {
                        BitOperation(AddressingMode.ZeroPage);
                        break;
                    }
                //BIT Compare Memory with Accumulator, Absolute, 2 Bytes, 4 Cycles
                case 0x2C:
                    {
                        BitOperation(AddressingMode.Absolute);
                        break;
                    }
                //EOR Exclusive OR Memory with Accumulator, Immediate, 2 Bytes, 2 Cycles
                case 0x49:
                    {
                        EorOperation(AddressingMode.Immediate);
                        break;
                    }
                //EOR Exclusive OR Memory with Accumulator, Zero Page, 2 Bytes, 3 Cycles
                case 0x45:
                    {
                        EorOperation(AddressingMode.ZeroPage);
                        break;
                    }
                //EOR Exclusive OR Memory with Accumulator, Zero Page X, 2 Bytes, 4 Cycles
                case 0x55:
                    {
                        EorOperation(AddressingMode.ZeroPageX);
                        break;
                    }
                //EOR Exclusive OR Memory with Accumulator, Absolute, 3 Bytes, 4 Cycles
                case 0x4D:
                    {
                        EorOperation(AddressingMode.Absolute);
                        break;
                    }
                //EOR Exclusive OR Memory with Accumulator, Absolute X, 3 Bytes, 4+ Cycles
                case 0x5D:
                    {
                        EorOperation(AddressingMode.AbsoluteX);
                        break;
                    }
                //EOR Exclusive OR Memory with Accumulator, Absolute Y, 3 Bytes, 4+ Cycles
                case 0x59:
                    {
                        EorOperation(AddressingMode.AbsoluteY);
                        break;
                    }
                //EOR Exclusive OR Memory with Accumulator, IndexedIndirect, 2 Bytes 6 Cycles
                case 0x41:
                    {
                        EorOperation(AddressingMode.IndirectX);
                        break;
                    }
                //EOR Exclusive OR Memory with Accumulator, IndirectIndexed, 2 Bytes 5 Cycles
                case 0x51:
                    {
                        EorOperation(AddressingMode.IndirectY);
                        break;
                    }
                //ORA Compare Memory with Accumulator, Immediate, 2 Bytes, 2 Cycles
                case 0x09:
                    {
                        OrOperation(AddressingMode.Immediate);
                        break;
                    }
                //ORA Compare Memory with Accumulator, Zero Page, 2 Bytes, 2 Cycles
                case 0x05:
                    {
                        OrOperation(AddressingMode.ZeroPage);
                        break;
                    }
                //ORA Compare Memory with Accumulator, Zero PageX, 2 Bytes, 4 Cycles
                case 0x15:
                    {
                        OrOperation(AddressingMode.ZeroPageX);
                        break;
                    }
                //ORA Compare Memory with Accumulator, Absolute,  3 Bytes, 4 Cycles
                case 0x0D:
                    {
                        OrOperation(AddressingMode.Absolute);
                        break;
                    }
                //ORA Compare Memory with Accumulator, AbsolueteX 3 Bytes, 4+ Cycles
                case 0x1D:
                    {
                        OrOperation(AddressingMode.AbsoluteX);
                        break;
                    }
                //ORA Compare Memory with Accumulator, AbsoluteY, 3 Bytes, 4+ Cycles
                case 0x19:
                    {
                        OrOperation(AddressingMode.AbsoluteY);
                        break;
                    }
                //ORA Compare Memory with Accumulator, IndexedIndirect, 2 Bytes, 6 Cycles
                case 0x01:
                    {
                        OrOperation(AddressingMode.IndirectX);
                        break;
                    }
                //ORA Compare Memory with Accumulator, IndirectIndexed, 2 Bytes, 5 Cycles
                case 0x11:
                    {
                        OrOperation(AddressingMode.IndirectY);
                        break;
                    }
#endregion

#region Clear Flag Operations
                //CLC Clear Carry Flag, Implied, 1 Byte, 2 Cycles
                case 0x18:
                    {
                        CarryFlag = false;
                        IncrementCycleCount();
                        break;
                    }
                //CLD Clear Decimal Flag, Implied, 1 Byte, 2 Cycles
                case 0xD8:
                    {
                        DecimalFlag = false;
                        IncrementCycleCount();
                        break;

                    }
                //CLI Clear Interrupt Flag, Implied, 1 Byte, 2 Cycles
                case 0x58:
                    {
                        DisableInterruptFlag = false;
                        IncrementCycleCount();
                        break;

                    }
                //CLV Clear Overflow Flag, Implied, 1 Byte, 2 Cycles
                case 0xB8:
                    {
                        OverflowFlag = false;
                        IncrementCycleCount();
                        break;
                    }

#endregion

#region Compare Operations
                //CMP Compare Accumulator with Memory, Immediate, 2 Bytes, 2 Cycles
                case 0xC9:
                    {
                        CompareOperation(AddressingMode.Immediate, Accumulator);
                        break;
                    }
                //CMP Compare Accumulator with Memory, Zero Page, 2 Bytes, 3 Cycles
                case 0xC5:
                    {
                        CompareOperation(AddressingMode.ZeroPage, Accumulator);
                        break;
                    }
                //CMP Compare Accumulator with Memory, Zero Page x, 2 Bytes, 4 Cycles
                case 0xD5:
                    {
                        CompareOperation(AddressingMode.ZeroPageX, Accumulator);
                        break;
                    }
                //CMP Compare Accumulator with Memory, Absolute, 3 Bytes, 4 Cycles
                case 0xCD:
                    {
                        CompareOperation(AddressingMode.Absolute, Accumulator);
                        break;
                    }
                //CMP Compare Accumulator with Memory, Absolute X, 2 Bytes, 4 Cycles
                case 0xDD:
                    {
                        CompareOperation(AddressingMode.AbsoluteX, Accumulator);
                        break;
                    }
                //CMP Compare Accumulator with Memory, Absolute Y, 2 Bytes, 4 Cycles
                case 0xD9:
                    {
                        CompareOperation(AddressingMode.AbsoluteY, Accumulator);
                        break;
                    }
                //CMP Compare Accumulator with Memory, Indirect X, 2 Bytes, 6 Cycles
                case 0xC1:
                    {
                        CompareOperation(AddressingMode.IndirectX, Accumulator);
                        break;
                    }
                //CMP Compare Accumulator with Memory, Indirect Y, 2 Bytes, 5 Cycles
                case 0xD1:
                    {
                        CompareOperation(AddressingMode.IndirectY, Accumulator);
                        break;
                    }
                //CPX Compare Accumulator with X Register, Immediate, 2 Bytes, 2 Cycles
                case 0xE0:
                    {
                        CompareOperation(AddressingMode.Immediate, XRegister);
                        break;
                    }
                //CPX Compare Accumulator with X Register, Zero Page, 2 Bytes, 3 Cycles
                case 0xE4:
                    {
                        CompareOperation(AddressingMode.ZeroPage, XRegister);
                        break;
                    }
                //CPX Compare Accumulator with X Register, Absolute, 3 Bytes, 4 Cycles
                case 0xEC:
                    {
                        CompareOperation(AddressingMode.Absolute, XRegister);
                        break;
                    }
                //CPY Compare Accumulator with Y Register, Immediate, 2 Bytes, 2 Cycles
                case 0xC0:
                    {
                        CompareOperation(AddressingMode.Immediate, YRegister);
                        break;
                    }
                //CPY Compare Accumulator with Y Register, Zero Page, 2 Bytes, 3 Cycles
                case 0xC4:
                    {
                        CompareOperation(AddressingMode.ZeroPage, YRegister);
                        break;
                    }
                //CPY Compare Accumulator with Y Register, Absolute, 3 Bytes, 4 Cycles
                case 0xCC:
                    {
                        CompareOperation(AddressingMode.Absolute, YRegister);
                        break;
                    }
#endregion

#region Increment/Decrement Operations
                //DEC Decrement Memory by One, Zero Page, 2 Bytes, 5 Cycles
                case 0xC6:
                    {
                        ChangeMemoryByOne(AddressingMode.ZeroPage, true);
                        break;
                    }
                //DEC Decrement Memory by One, Zero Page X, 2 Bytes, 6 Cycles
                case 0xD6:
                    {
                        ChangeMemoryByOne(AddressingMode.ZeroPageX, true);
                        break;
                    }
                //DEC Decrement Memory by One, Absolute, 3 Bytes, 6 Cycles
                case 0xCE:
                    {
                        ChangeMemoryByOne(AddressingMode.Absolute, true);
                        break;
                    }
                //DEC Decrement Memory by One, Absolute X, 3 Bytes, 7 Cycles
                case 0xDE:
                    {
                        ChangeMemoryByOne(AddressingMode.AbsoluteX, true);
                        IncrementCycleCount();
                        break;
                    }
                //DEX Decrement X Register by One, Implied, 1 Bytes, 2 Cycles
                case 0xCA:
                    {
                        ChangeRegisterByOne(true, true);
                        break;
                    }
                //DEY Decrement Y Register by One, Implied, 1 Bytes, 2 Cycles
                case 0x88:
                    {
                        ChangeRegisterByOne(false, true);
                        break;
                    }
                //INC Increment Memory by One, Zero Page, 2 Bytes, 5 Cycles
                case 0xE6:
                    {
                        ChangeMemoryByOne(AddressingMode.ZeroPage, false);
                        break;
                    }
                //INC Increment Memory by One, Zero Page X, 2 Bytes, 6 Cycles
                case 0xF6:
                    {
                        ChangeMemoryByOne(AddressingMode.ZeroPageX, false);
                        break;
                    }
                //INC Increment Memory by One, Absolute, 3 Bytes, 6 Cycles
                case 0xEE:
                    {
                        ChangeMemoryByOne(AddressingMode.Absolute, false);
                        break;
                    }
                //INC Increment Memory by One, Absolute X, 3 Bytes, 7 Cycles
                case 0xFE:
                    {
                        ChangeMemoryByOne(AddressingMode.AbsoluteX, false);
                        IncrementCycleCount();
                        break;
                    }
                //INX Increment X Register by One, Implied, 1 Bytes, 2 Cycles
                case 0xE8:
                    {
                        ChangeRegisterByOne(true, false);
                        break;
                    }
                //INY Increment Y Register by One, Implied, 1 Bytes, 2 Cycles
                case 0xC8:
                    {
                        ChangeRegisterByOne(false, false);
                        break;
                    }
#endregion

#region GOTO and GOSUB Operations
                //JMP Jump to New Location, Absolute 3 Bytes, 3 Cycles
                case 0x4C:
                    {
                        ProgramCounter = GetAddressByAddressingMode(AddressingMode.Absolute);
                        break;
                    }
                //JMP Jump to New Location, Indirect 3 Bytes, 5 Cycles
                case 0x6C:
                    {
                        ProgramCounter = GetAddressByAddressingMode(AddressingMode.Absolute);

                        if ((ProgramCounter & 0xFF) == 0xFF)
                        {
                            //Get the first half of the address
                            int address = MemoryMap.Read(ProgramCounter);

                            //Get the second half of the address, due to the issue with page boundary it reads from the wrong location!
                            address += 256 * MemoryMap.Read(ProgramCounter - 255);
                            ProgramCounter = address;
                        }
                        else
                        {
                            ProgramCounter = GetAddressByAddressingMode(AddressingMode.Absolute);
                        }

                        break;
                    }
                //JSR Jump to SubRoutine, Absolute, 3 Bytes, 6 Cycles
                case 0x20:
                    {
                        JumpToSubRoutineOperation();
                        break;
                    }
                //BRK Simulate IRQ, Implied, 1 Byte, 7 Cycles
                case 0x00:
                    {
                        BreakOperation(true, 0xFFFE);
                        break;
                    }
                //RTI Return From Interrupt, Implied, 1 Byte, 6 Cycles
                case 0x40:
                    {
                        ReturnFromInterruptOperation();
                        break;
                    }
                //RTS Return From Subroutine, Implied, 1 Byte, 6 Cycles
                case 0x60:
                    {
                        ReturnFromSubRoutineOperation();
                        break;
                    }
#endregion

#region Load Value From Memory Operations
                //LDA Load Accumulator with Memory, Immediate, 2 Bytes, 2 Cycles
                case 0xA9:
                    {
                        Accumulator = MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.Immediate));
                        SetZeroFlag(Accumulator);
                        SetNegativeFlag(Accumulator);
                        break;
                    }
                //LDA Load Accumulator with Memory, Zero Page, 2 Bytes, 3 Cycles
                case 0xA5:
                    {
                        Accumulator = MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.ZeroPage));
                        SetZeroFlag(Accumulator);
                        SetNegativeFlag(Accumulator);
                        break;
                    }
                //LDA Load Accumulator with Memory, Zero Page X, 2 Bytes, 4 Cycles
                case 0xB5:
                    {
                        Accumulator =
        MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.ZeroPageX));
                        SetZeroFlag(Accumulator);
                        SetNegativeFlag(Accumulator);
                        break;
                    }
                //LDA Load Accumulator with Memory, Absolute, 3 Bytes, 4 Cycles
                case 0xAD:
                    {
                        Accumulator =
        MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.Absolute));
                        SetZeroFlag(Accumulator);
                        SetNegativeFlag(Accumulator);
                        break;
                    }
                //LDA Load Accumulator with Memory, Absolute X, 3 Bytes, 4+ Cycles
                case 0xBD:
                    {
                        Accumulator =
        MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.AbsoluteX));
                        SetZeroFlag(Accumulator);
                        SetNegativeFlag(Accumulator);
                        break;
                    }
                //LDA Load Accumulator with Memory, Absolute Y, 3 Bytes, 4+ Cycles
                case 0xB9:
                    {
                        Accumulator =
        MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.AbsoluteY));
                        SetZeroFlag(Accumulator);
                        SetNegativeFlag(Accumulator);
                        break;
                    }
                //LDA Load Accumulator with Memory, Index Indirect, 2 Bytes, 6 Cycles
                case 0xA1:
                    {
                        Accumulator =
        MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.IndirectX));
                        SetZeroFlag(Accumulator);
                        SetNegativeFlag(Accumulator);
                        break;
                    }
                //LDA Load Accumulator with Memory, Indirect Index, 2 Bytes, 5+ Cycles
                case 0xB1:
                    {
                        Accumulator =
        MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.IndirectY));
                        SetZeroFlag(Accumulator);
                        SetNegativeFlag(Accumulator);
                        break;
                    }
                //LDX Load X with memory, Immediate, 2 Bytes, 2 Cycles
                case 0xA2:
                    {
                        XRegister =
        MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.Immediate));
                        SetZeroFlag(XRegister);
                        SetNegativeFlag(XRegister);
                        break;
                    }
                //LDX Load X with memory, Zero Page, 2 Bytes, 3 Cycles
                case 0xA6:
                    {
                        XRegister =
        MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.ZeroPage));
                        SetZeroFlag(XRegister);
                        SetNegativeFlag(XRegister);
                        break;
                    }
                //LDX Load X with memory, Zero Page Y, 2 Bytes, 4 Cycles
                case 0xB6:
                    {
                        XRegister =
        MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.ZeroPageY));
                        SetZeroFlag(XRegister);
                        SetNegativeFlag(XRegister);
                        break;
                    }
                //LDX Load X with memory, Absolute, 3 Bytes, 4 Cycles
                case 0xAE:
                    {
                        XRegister =
        MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.Absolute));
                        SetZeroFlag(XRegister);
                        SetNegativeFlag(XRegister);
                        break;
                    }
                //LDX Load X with memory, Absolute Y, 3 Bytes, 4+ Cycles
                case 0xBE:
                    {
                        XRegister =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.AbsoluteY));
                        SetZeroFlag(XRegister);
                        SetNegativeFlag(XRegister);
                        break;
                    }
                //LDY Load Y with memory, Immediate, 2 Bytes, 2 Cycles
                case 0xA0:
                    {
                        YRegister =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.Immediate));
                        SetZeroFlag(YRegister);
                        SetNegativeFlag(YRegister);
                        break;
                    }
                //LDY Load Y with memory, Zero Page, 2 Bytes, 3 Cycles
                case 0xA4:
                    {
                        YRegister =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.ZeroPage));
                        SetZeroFlag(YRegister);
                        SetNegativeFlag(YRegister);
                        break;
                    }
                //LDY Load Y with memory, Zero Page X, 2 Bytes, 4 Cycles
                case 0xB4:
                    {
                        YRegister =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.ZeroPageX));
                        SetZeroFlag(YRegister);
                        SetNegativeFlag(YRegister);
                        break;
                    }
                //LDY Load Y with memory, Absolute, 3 Bytes, 4 Cycles
                case 0xAC:
                    {
                        YRegister =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.Absolute));
                        SetZeroFlag(YRegister);
                        SetNegativeFlag(YRegister);
                        break;
                    }
                //LDY Load Y with memory, Absolue X, 3 Bytes, 4+ Cycles
                case 0xBC:
                    {
                        YRegister =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.AbsoluteX));
                        SetZeroFlag(YRegister);
                        SetNegativeFlag(YRegister);
                        break;
                    }
#endregion

#region Push/Pull Stack
                //PHA Push Accumulator onto Stack, Implied, 1 Byte, 3 Cycles
                case 0x48:
                    {
                        MemoryMap.Read(ProgramCounter + 1);

                        PokeStack((byte)Accumulator);
                        StackPointer--;
                        IncrementCycleCount();
                        break;

                    }
                //PHP Push Flags onto Stack, Implied, 1 Byte, 3 Cycles
                case 0x08:
                    {
                        MemoryMap.Read(ProgramCounter + 1);

                        PushFlagsOperation();
                        StackPointer--;
                        IncrementCycleCount();
                        break;
                    }
                //PLA Pull Accumulator from Stack, Implied, 1 Byte, 4 Cycles
                case 0x68:
                    {
                        MemoryMap.Read(ProgramCounter + 1);
                        StackPointer++;
                        IncrementCycleCount();

                        Accumulator = PeekStack();
                        SetNegativeFlag(Accumulator);
                        SetZeroFlag(Accumulator);

                        IncrementCycleCount();
                        break;
                    }
                //PLP Pull Flags from Stack, Implied, 1 Byte, 4 Cycles
                case 0x28:
                    {
                        MemoryMap.Read(ProgramCounter + 1);

                        StackPointer++;
                        IncrementCycleCount();

                        PullFlagsOperation();

                        IncrementCycleCount();
                        break;
                    }
                //TSX Transfer Stack Pointer to X Register, 1 Bytes, 2 Cycles
                case 0xBA:
                    {
                        XRegister = StackPointer;

                        SetNegativeFlag(XRegister);
                        SetZeroFlag(XRegister);
                        IncrementCycleCount();
                        break;
                    }
                //TXS Transfer X Register to Stack Pointer, 1 Bytes, 2 Cycles
                case 0x9A:
                    {
                        StackPointer = (byte)XRegister;
                        IncrementCycleCount();
                        break;
                    }
#endregion

#region Set Flag Operations
                //SEC Set Carry, Implied, 1 Bytes, 2 Cycles
                case 0x38:
                    {
                        CarryFlag = true;
                        IncrementCycleCount();
                        break;
                    }
                //SED Set Interrupt, Implied, 1 Bytes, 2 Cycles
                case 0xF8:
                    {
                        DecimalFlag = true;
                        IncrementCycleCount();
                        break;
                    }
                //SEI Set Interrupt, Implied, 1 Bytes, 2 Cycles
                case 0x78:
                    {
                        DisableInterruptFlag = true;
                        IncrementCycleCount();
                        break;
                    }
#endregion

#region Shift/Rotate Operations
                //ASL Shift Left 1 Bit Memory or Accumulator, Accumulator, 1 Bytes, 2 Cycles
                case 0x0A:
                    {
                        AslOperation(AddressingMode.Accumulator);
                        break;
                    }
                //ASL Shift Left 1 Bit Memory or Accumulator, Zero Page, 2 Bytes, 5 Cycles
                case 0x06:
                    {
                        AslOperation(AddressingMode.ZeroPage);
                        break;
                    }
                //ASL Shift Left 1 Bit Memory or Accumulator, Zero PageX, 2 Bytes, 6 Cycles
                case 0x16:
                    {
                        AslOperation(AddressingMode.ZeroPageX);
                        break;
                    }
                //ASL Shift Left 1 Bit Memory or Accumulator, Absolute, 3 Bytes, 6 Cycles
                case 0x0E:
                    {
                        AslOperation(AddressingMode.Absolute);
                        break;
                    }
                //ASL Shift Left 1 Bit Memory or Accumulator, AbsoluteX, 3 Bytes, 7 Cycles
                case 0x1E:
                    {
                        AslOperation(AddressingMode.AbsoluteX);
                        IncrementCycleCount();
                        break;
                    }
                //LSR Shift Left 1 Bit Memory or Accumulator, Accumulator, 1 Bytes, 2 Cycles
                case 0x4A:
                    {
                        LsrOperation(AddressingMode.Accumulator);
                        break;
                    }
                //LSR Shift Left 1 Bit Memory or Accumulator, Zero Page, 2 Bytes, 5 Cycles
                case 0x46:
                    {
                        LsrOperation(AddressingMode.ZeroPage);
                        break;
                    }
                //LSR Shift Left 1 Bit Memory or Accumulator, Zero PageX, 2 Bytes, 6 Cycles
                case 0x56:
                    {
                        LsrOperation(AddressingMode.ZeroPageX);
                        break;
                    }
                //LSR Shift Left 1 Bit Memory or Accumulator, Absolute, 3 Bytes, 6 Cycles
                case 0x4E:
                    {
                        LsrOperation(AddressingMode.Absolute);
                        break;
                    }
                //LSR Shift Left 1 Bit Memory or Accumulator, AbsoluteX, 3 Bytes, 7 Cycles
                case 0x5E:
                    {
                        LsrOperation(AddressingMode.AbsoluteX);
                        IncrementCycleCount();
                        break;
                    }
                //ROL Rotate Left 1 Bit Memory or Accumulator, Accumulator, 1 Bytes, 2 Cycles
                case 0x2A:
                    {
                        RolOperation(AddressingMode.Accumulator);
                        break;
                    }
                //ROL Rotate Left 1 Bit Memory or Accumulator, Zero Page, 2 Bytes, 5 Cycles
                case 0x26:
                    {
                        RolOperation(AddressingMode.ZeroPage);
                        break;
                    }
                //ROL Rotate Left 1 Bit Memory or Accumulator, Zero PageX, 2 Bytes, 6 Cycles
                case 0x36:
                    {
                        RolOperation(AddressingMode.ZeroPageX);
                        break;
                    }
                //ROL Rotate Left 1 Bit Memory or Accumulator, Absolute, 3 Bytes, 6 Cycles
                case 0x2E:
                    {
                        RolOperation(AddressingMode.Absolute);
                        break;
                    }
                //ROL Rotate Left 1 Bit Memory or Accumulator, AbsoluteX, 3 Bytes, 7 Cycles
                case 0x3E:
                    {
                        RolOperation(AddressingMode.AbsoluteX);
                        IncrementCycleCount();
                        break;
                    }
                //ROR Rotate Right 1 Bit Memory or Accumulator, Accumulator, 1 Bytes, 2 Cycles
                case 0x6A:
                    {
                        RorOperation(AddressingMode.Accumulator);
                        break;
                    }
                //ROR Rotate Right 1 Bit Memory or Accumulator, Zero Page, 2 Bytes, 5 Cycles
                case 0x66:
                    {
                        RorOperation(AddressingMode.ZeroPage);
                        break;
                    }
                //ROR Rotate Right 1 Bit Memory or Accumulator, Zero PageX, 2 Bytes, 6 Cycles
                case 0x76:
                    {
                        RorOperation(AddressingMode.ZeroPageX);
                        break;
                    }
                //ROR Rotate Right 1 Bit Memory or Accumulator, Absolute, 3 Bytes, 6 Cycles
                case 0x6E:
                    {
                        RorOperation(AddressingMode.Absolute);
                        break;
                    }
                //ROR Rotate Right 1 Bit Memory or Accumulator, AbsoluteX, 3 Bytes, 7 Cycles
                case 0x7E:
                    {
                        RorOperation(AddressingMode.AbsoluteX);
                        IncrementCycleCount();
                        break;
                    }
#endregion

#region Store Value In Memory Operations
                //STA Store Accumulator In Memory, Zero Page, 2 Bytes, 3 Cycles
                case 0x85:
                    {
                        MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.ZeroPage),
      (byte)Accumulator);
                        break;
                    }
                //STA Store Accumulator In Memory, Zero Page X, 2 Bytes, 4 Cycles
                case 0x95:
                    {
                        MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.ZeroPageX),
      (byte)Accumulator);
                        break;
                    }
                //STA Store Accumulator In Memory, Absolute, 3 Bytes, 4 Cycles
                case 0x8D:
                    {
                        MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.Absolute),
      (byte)Accumulator);
                        break;
                    }
                //STA Store Accumulator In Memory, Absolute X, 3 Bytes, 5 Cycles
                case 0x9D:
                    {
                        MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.AbsoluteX),
      (byte)Accumulator);
                        IncrementCycleCount();
                        break;
                    }
                //STA Store Accumulator In Memory, Absolute Y, 3 Bytes, 5 Cycles
                case 0x99:
                    {
                        MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.AbsoluteY),
      (byte)Accumulator);
                        IncrementCycleCount();
                        break;
                    }
                //STA Store Accumulator In Memory, Indexed Indirect, 2 Bytes, 6 Cycles
                case 0x81:
                    {
                        MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.IndirectX),
      (byte)Accumulator);
                        break;
                    }
                //STA Store Accumulator In Memory, Indirect Indexed, 2 Bytes, 6 Cycles
                case 0x91:
                    {
                        MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.IndirectY),
      (byte)Accumulator);
                        IncrementCycleCount();
                        break;
                    }
                //STX Store Index X, Zero Page, 2 Bytes, 3 Cycles
                case 0x86:
                    {
                        MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.ZeroPage),
      (byte)XRegister);
                        break;
                    }
                //STX Store Index X, Zero Page Y, 2 Bytes, 4 Cycles
                case 0x96:
                    {
                        MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.ZeroPageY),
      (byte)XRegister);
                        break;
                    }
                //STX Store Index X, Absolute, 3 Bytes, 4 Cycles
                case 0x8E:
                    {
                        MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.Absolute),
(byte)XRegister);
                        break;
                    }
                //STY Store Index Y, Zero Page, 2 Bytes, 3 Cycles
                case 0x84:
                    {
                        MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.ZeroPage),
(byte)YRegister);
                        break;
                    }
                //STY Store Index Y, Zero Page X, 2 Bytes, 4 Cycles
                case 0x94:
                    {
                        MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.ZeroPageX),
(byte)YRegister);
                        break;
                    }
                //STY Store Index Y, Absolute, 2 Bytes, 4 Cycles
                case 0x8C:
                    {
                        MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.Absolute),
(byte)YRegister);
                        break;
                    }
#endregion

#region Transfer Operations
                //TAX Transfer Accumulator to X Register, Implied, 1 Bytes, 2 Cycles
                case 0xAA:
                    {
                        IncrementCycleCount();
                        XRegister = Accumulator;

                        SetNegativeFlag(XRegister);
                        SetZeroFlag(XRegister);
                        break;
                    }
                //TAY Transfer Accumulator to Y Register, 1 Bytes, 2 Cycles
                case 0xA8:
                    {
                        IncrementCycleCount();
                        YRegister = Accumulator;

                        SetNegativeFlag(YRegister);
                        SetZeroFlag(YRegister);
                        break;
                    }
                //TXA Transfer X Register to Accumulator, Implied, 1 Bytes, 2 Cycles
                case 0x8A:
                    {
                        IncrementCycleCount();
                        Accumulator = XRegister;

                        SetNegativeFlag(Accumulator);
                        SetZeroFlag(Accumulator);
                        break;
                    }
                //TYA Transfer Y Register to Accumulator, Implied, 1 Bytes, 2 Cycles
                case 0x98:
                    {
                        IncrementCycleCount();
                        Accumulator = YRegister;

                        SetNegativeFlag(Accumulator);
                        SetZeroFlag(Accumulator);
                        break;
                    }
#endregion

                //NOP Operation, Implied, 1 Byte, 2 Cycles
                case 0xEA:
                    {
                        IncrementCycleCount();
                        break;
                    }

                default:
                    throw new NotSupportedException(string.Format("The OpCode {0} is not supported",
CurrentOpCode));
            }
        }
```

6.39.3.13 GetAddressByAddressingMode() `int Hardware.W65C02.GetAddressByAddressingMode (`

AddressingMode *addressingMode*) [inline], [protected]

Uses the AddressingMode to return the correct address based on the mode. Note: This method will not increment the program counter for any mode. Note: This method will return an error if called for either the immediate or accumulator modes.

Parameters

addressingMode	The addressing Mode to use

Returns

The memory Location

Definition at line 1340 of file W65C02.cs.

```
        {
            int address;
            int highByte;
            switch (addressingMode)
            {
                case (AddressingMode.Absolute):
                    {
                        return (MemoryMap.Read(ProgramCounter++) | (MemoryMap.Read(ProgramCounter++) « 8));
                    }
                case AddressingMode.AbsoluteX:
                    {
                        //Get the low half of the address
                        address = MemoryMap.Read(ProgramCounter++);

                        //Get the high byte
                        highByte = MemoryMap.Read(ProgramCounter++);

                        //We crossed a page boundry, so an extra read has occurred.
                        //However, if this is an ASL, LSR, DEC, INC, ROR, ROL or STA operation, we do not decrease it by 1.
                        if (address + XRegister > 0xFF)
                        {
                            switch (CurrentOpCode)
                            {
                                case 0x1E:
                                case 0xDE:
                                case 0xFE:
                                case 0x5E:
                                case 0x3E:
                                case 0x7E:
                                case 0x9D:
                                    {
                                        //This is a MemoryMap.Read Fetch Write Operation, so we don’t make the extra read.
                                        return ((highByte « 8 | address) + XRegister) & 0xFFFF;
                                    }
                                default:
                                    {
                                        MemoryMap.Read((((highByte « 8 | address) + XRegister) - 0xFF) & 0xFFFF);
                                        break;
                                    }
                            }
                        }

                        return ((highByte « 8 | address) + XRegister) & 0xFFFF;
                    }
                case AddressingMode.AbsoluteY:
                    {
                        //Get the low half of the address
                        address = MemoryMap.Read(ProgramCounter++);

                        //Get the high byte
                        highByte = MemoryMap.Read(ProgramCounter++);

                        //We crossed a page boundry, so decrease the number of cycles by 1 if the operation is not STA
                        if (address + YRegister > 0xFF && CurrentOpCode != 0x99)
                        {
                            MemoryMap.Read((((highByte « 8 | address) + YRegister) - 0xFF) & 0xFFFF);
                        }
                        //Bitshift the high byte into place, AND with FFFF to handle wrapping.
                        return ((highByte « 8 | address) + YRegister) & 0xFFFF;
                    }
                case AddressingMode.Immediate:
                    {
                        return ProgramCounter++;
                    }
                case AddressingMode.IndirectX:
                    {
                        //Get the location of the address to retrieve
                        address = MemoryMap.Read(ProgramCounter++);
                        MemoryMap.Read(address);

                        address += XRegister;

                        //Now get the final Address.  The is not a zero page address either.
                        var finalAddress = MemoryMap.Read((address & 0xFF)) | (MemoryMap.Read((address
+ 1) & 0xFF) « 8);
                        return finalAddress;
                    }
                case AddressingMode.IndirectY:
                    {
                        address = MemoryMap.Read(ProgramCounter++);

                        var finalAddress = MemoryMap.Read(address) + (MemoryMap.Read((address + 1) &
0xFF) « 8);

                        if ((finalAddress & 0xFF) + YRegister > 0xFF && CurrentOpCode != 0x91)
                        {
                            MemoryMap.Read((finalAddress + YRegister - 0xFF) & 0xFFFF);
                        }

                        return (finalAddress + YRegister) & 0xFFFF;
                    }
                case AddressingMode.Relative:
                    {
                        return ProgramCounter;
                    }
                case (AddressingMode.ZeroPage):
                    {
                        address = MemoryMap.Read(ProgramCounter++);
                        return address;
                    }
                case (AddressingMode.ZeroPageX):
                    {
                        address = MemoryMap.Read(ProgramCounter++);
                        MemoryMap.Read(address);

                        address += XRegister;
                        address &= 0xFF;

                        //This address wraps if its greater than 0xFF
                        if (address > 0xFF)
                        {
                            address -= 0x100;
                            return address;
                        }

                        return address;
                    }
                case (AddressingMode.ZeroPageY):
                    {
                        address = MemoryMap.Read(ProgramCounter++);
                        MemoryMap.Read(address);

                        address += YRegister;
                        address &= 0xFF;

                        return address;
                    }
                default:
                    throw new InvalidOperationException(string.Format("The Address Mode '{0}' does not
require an address", addressingMode));
            }
        }
```

6.39.3.14 GetAddressingMode() AddressingMode Hardware.W65C02.GetAddressingMode ( ) [inline], [private]

Definition at line 1685 of file W65C02.cs.

```
        {
            switch (CurrentOpCode)
            {
                case 0x0D:  //ORA
                case 0x2D:  //AND
                case 0x4D:  //EOR
                case 0x6D:  //ADC
                case 0x8D:  //STA
                case 0xAD:  //LDA
                case 0xCD:  //CMP
                case 0xED:  //SBC
                case 0x0E:  //ASL
                case 0x2E:  //ROL
                case 0x4E:  //LSR
                case 0x6E:  //ROR
                case 0x8E:  //SDX
                case 0xAE:  //LDX
                case 0xCE:  //DEC
                case 0xEE:  //INC
                case 0x2C:  //Bit
                case 0x4C:  //JMP
                case 0x8C:  //STY
                case 0xAC:  //LDY
                case 0xCC:  //CPY
                case 0xEC:  //CPX
                case 0x20:    //JSR
                    {
                        return AddressingMode.Absolute;
                    }
                case 0x1D:  //ORA
                case 0x3D:  //AND
                case 0x5D:  //EOR
                case 0x7D:  //ADC
                case 0x9D:  //STA
                case 0xBD:  //LDA
                case 0xDD:  //CMP
                case 0xFD:  //SBC
                case 0xBC:  //LDY
                case 0xFE:  //INC
                case 0x1E:  //ASL
                case 0x3E:  //ROL
                case 0x5E:  //LSR
                case 0x7E:  //ROR
                    {
                        return AddressingMode.AbsoluteX;
                    }
                case 0x19:    //ORA
                case 0x39:    //AND
                case 0x59:    //EOR
                case 0x79:    //ADC
                case 0x99:    //STA
                case 0xB9:    //LDA
                case 0xD9:    //CMP
                case 0xF9:    //SBC
                case 0xBE:  //LDX
                    {
                        return AddressingMode.AbsoluteY;
                    }
                case 0x0A:  //ASL
                case 0x4A:  //LSR
                case 0x2A:  //ROL
                case 0x6A:  //ROR
                    {
                        return AddressingMode.Accumulator;
                    }

                case 0x09:    //ORA
                case 0x29:    //AND
                case 0x49:    //EOR
                case 0x69:    //ADC
                case 0xA0:    //LDY
                case 0xC0:    //CPY
                case 0xE0:    //CMP
                case 0xA2:    //LDX
                case 0xA9:    //LDA
                case 0xC9:    //CMP
                case 0xE9:    //SBC
                    {
                        return AddressingMode.Immediate;
                    }
                case 0x00:    //BRK
                case 0x18:    //CLC
                case 0xD8:    //CLD
                case 0x58:    //CLI
                case 0xB8:    //CLV
                case 0xDE:  //DEC
                case 0xCA:  //DEX
                case 0x88:    //DEY
                case 0xE8:    //INX
                case 0xC8:    //INY
                case 0xEA:  //NOP
                case 0x48:    //PHA
                case 0x08:    //PHP
                case 0x68:    //PLA
                case 0x28:    //PLP
                case 0x40:    //RTI
                case 0x60:    //RTS
                case 0x38:    //SEC
                case 0xF8:    //SED
                case 0x78:    //SEI
                case 0xAA:  //TAX
                case 0xA8:    //TAY
                case 0xBA:  //TSX
                case 0x8A:  //TXA
                case 0x9A:  //TXS
                case 0x98:    //TYA
                    {
                        return AddressingMode.Implied;
                    }
                case 0x6C:
                    {
                        return AddressingMode.Indirect;
                    }

                case 0x61:    //ADC
                case 0x21:    //AND
                case 0xC1:    //CMP
                case 0x41:    //EOR
                case 0xA1:    //LDA
                case 0x01:    //ORA
                case 0xE1:    //SBC
                case 0x81:    //STA
                    {
                        return AddressingMode.IndirectX;
                    }
                case 0x71:    //ADC
                case 0x31:    //AND
                case 0xD1:    //CMP
                case 0x51:    //EOR
                case 0xB1:    //LDA
                case 0x11:    //ORA
                case 0xF1:    //SBC
                case 0x91:    //STA
                    {
                        return AddressingMode.IndirectY;
                    }
                case 0x90:    //BCC
                case 0xB0:    //BCS
                case 0xF0:    //BEQ
                case 0x30:    //BMI
                case 0xD0:    //BNE
                case 0x10:    //BPL
                case 0x50:    //BVC
                case 0x70:    //BVS
                    {
                        return AddressingMode.Relative;
                    }
                case 0x65:    //ADC
                case 0x25:    //AND
                case 0x06:    //ASL
                case 0x24:    //BIT
                case 0xC5:    //CMP
                case 0xE4:    //CPX
                case 0xC4:    //CPY
                case 0xC6:    //DEC
                case 0x45:    //EOR
                case 0xE6:    //INC
                case 0xA5:    //LDA
                case 0xA6:    //LDX
                case 0xA4:    //LDY
                case 0x46:    //LSR
                case 0x05:    //ORA
                case 0x26:    //ROL
                case 0x66:    //ROR
                case 0xE5:    //SBC
                case 0x85:    //STA
                case 0x86:    //STX
                case 0x84:    //STY
                    {
                        return AddressingMode.ZeroPage;
                    }
                case 0x75:    //ADC
                case 0x35:    //AND
                case 0x16:    //ASL
                case 0xD5:    //CMP
                case 0xD6:    //DEC
                case 0x55:    //EOR
                case 0xF6:    //INC
                case 0xB5:    //LDA
                case 0xB6:    //LDX
                case 0xB4:    //LDY
                case 0x56:    //LSR
                case 0x15:    //ORA
                case 0x36:    //ROL
                case 0x76:    //ROR
                case 0xF5:    //SBC
                case 0x95:    //STA
                case 0x96:    //STX
                case 0x94:    //STY
                    {
                        return AddressingMode.ZeroPageX;
                    }
                default:
                    throw new NotSupportedException(string.Format("Opcode {0} is not supported",
      CurrentOpCode));
            }
        }
```

6.39.3.15 GetCycleCount() `int Hardware.W65C02.GetCycleCount ( ) [inline]`

Gets the Number of Cycles that have elapsed

Returns

The number of elapsed cycles

Definition at line 209 of file W65C02.cs.

```
        {
            return _cycleCount;
        }
```

6.39.3.16 IncrementCycleCount() `void Hardware.W65C02.IncrementCycleCount ( ) [inline]`

Increments the Cycle Count, causes a CycleCountIncrementedAction to fire.

Definition at line 217 of file W65C02.cs.

```
        {
            _cycleCount++;
            CycleCountIncrementedAction();

            _previousInterrupt = _interrupt;
            _interrupt = TriggerNmi || (TriggerIRQ && !DisableInterruptFlag);
        }
```

6.39.3.17 InterruptRequest() `void Hardware.W65C02.InterruptRequest ( ) [inline]`

The InterruptRequest or IRQ

Definition at line 200 of file W65C02.cs.

```
        {
            TriggerIRQ = true;
        }
```

6.39.3.18 JumpToSubRoutineOperation() `void Hardware.W65C02.JumpToSubRoutineOperation ( ) [inline], [private]`

The JSR routine. Jumps to a subroutine.

Definition at line 2313 of file W65C02.cs.

```
        {
            IncrementCycleCount();

            //Put the high value on the stack, this should be the address after our operation -1
            //The RTS operation increments the PC by 1 which is why we don’t move 2
            PokeStack((byte)(((ProgramCounter + 1) » 8) & 0xFF));
            StackPointer--;
            IncrementCycleCount();

            PokeStack((byte)((ProgramCounter + 1) & 0xFF));
            StackPointer--;
            IncrementCycleCount();

            ProgramCounter = GetAddressByAddressingMode(AddressingMode.Absolute);
        }
```

6.39.3.19 LsrOperation() `void Hardware.W65C02.LsrOperation ( AddressingMode addressingMode ) [inline], [private]`

The LSR Operation. Performs a Left shift operation on a value in memory

Parameters

addressingMode	The addressing mode to use

Definition at line 2108 of file W65C02.cs.

```
        {
            int value;
            var memoryAddress = 0;
            if (addressingMode == AddressingMode.Accumulator)
            {
                MemoryMap.Read(ProgramCounter + 1);
                value = Accumulator;
            }
            else
            {
                memoryAddress = GetAddressByAddressingMode(addressingMode);
                value = MemoryMap.Read(memoryAddress);
            }

            //Dummy Write
            if (addressingMode != AddressingMode.Accumulator)
            {
                MemoryMap.Write(memoryAddress, (byte)value);
            }

            NegativeFlag = false;

            //If the Zero bit is set, we have a carry
            CarryFlag = (value & 0x01) != 0;

            value = (value » 1);

            SetZeroFlag(value);
            if (addressingMode == AddressingMode.Accumulator)
                Accumulator = value;
            else
            {
                MemoryMap.Write(memoryAddress, (byte)value);
            }
        }
```

6.39.3.20 MoveProgramCounterByRelativeValue() `void Hardware.W65C02.MoveProgramCounterBy↩ RelativeValue ( byte valueToMove ) [inline], [private]`

Moves the ProgramCounter in a given direction based on the value inputted

Definition at line 1476 of file W65C02.cs.

```
        {
            var movement = valueToMove > 127 ?  (valueToMove - 255) :  valueToMove;

            var newProgramCounter = ProgramCounter + movement;

            //This makes sure that we always land on the correct spot for a positive number
            if (movement >= 0)
                newProgramCounter++;

            //We Crossed a Page Boundary.  So we increment the cycle counter by one.  The +1 is
      because we always check from the end of the instruction not the beginning
            if (((ProgramCounter + 1 ^ newProgramCounter) & 0xff00) != 0x0000)
            {
                IncrementCycleCount();
            }

            ProgramCounter = newProgramCounter;
            MemoryMap.Read(ProgramCounter);
        }
```

6.39.3.21 NextStep() `void Hardware.W65C02.NextStep ( ) [inline]`

Performs the next step on the processor

Definition at line 171 of file W65C02.cs.

```
        {
            SetDisassembly();

            //Have to read this first otherwise it causes tests to fail on a NES
            CurrentOpCode = MemoryMap.Read(ProgramCounter);

            ProgramCounter++;

            ExecuteOpCode();

            if (_previousInterrupt)
            {
                if (TriggerNmi)
                {
                    ProcessNMI();
                    TriggerNmi = false;
                }
                else if (TriggerIRQ)
                {
                    ProcessIRQ();
                    TriggerIRQ = false;
                }
            }
        }
```

6.39.3.22 OrOperation() `void Hardware.W65C02.OrOperation ( AddressingMode addressingMode ) [inline], [private]`

The Or Operation. Performs an Or Operation with the accumulator and a value in memory

Parameters

addressingMode	The addressing mode to use

Definition at line 2149 of file W65C02.cs.

```
        {
            Accumulator = Accumulator | MemoryMap.Read(GetAddressByAddressingMode(addressingMode));

            SetNegativeFlag(Accumulator);
            SetZeroFlag(Accumulator);
        }
```

6.39.3.23 PeekStack() `byte Hardware.W65C02.PeekStack ( ) [inline], [private]`

Returns a the value from the stack without changing the position of the stack pointer

Returns

The value at the current Stack Pointer

Definition at line 1500 of file W65C02.cs.

```
        {
            //The stack lives at 0x100-0x1FF, but the value is only a byte so it needs to be
    translated
            return MemoryMap.Read(StackPointer + 0x100);
        }
```

6.39.3.24 PokeStack() `void Hardware.W65C02.PokeStack ( byte value ) [inline], [private]`

Write a value directly to the stack without modifying the Stack Pointer

Parameters

value	The value to be written to the stack

Definition at line 1511 of file W65C02.cs.

```
        {
            //The stack lives at 0x100-0x1FF, but the value is only a byte so it needs to be
    translated
            MemoryMap.Write(StackPointer + 0x100, value);
        }
```

6.39.3.25 ProcessIRQ() `void Hardware.W65C02.ProcessIRQ ( ) [inline], [private]`

This is ran anytime an IRQ occurrs

Definition at line 2425 of file W65C02.cs.

```
        {
            if (DisableInterruptFlag)
                return;

            ProgramCounter--;
            BreakOperation(false, 0xFFFE);
            CurrentOpCode = MemoryMap.Read(ProgramCounter);

            SetDisassembly();
        }
```

6.39.3.26 ProcessNMI() `void Hardware.W65C02.ProcessNMI ( ) [inline], [private]`

This is ran anytime an NMI occurrs

Definition at line 2413 of file W65C02.cs.

```
        {
            ProgramCounter--;
            BreakOperation(false, 0xFFFA);
            CurrentOpCode = MemoryMap.Read(ProgramCounter);

            SetDisassembly();
        }
```

6.39.3.27 PullFlagsOperation() `void Hardware.W65C02.PullFlagsOperation ( ) [inline], [private]`

The PLP Operation. Pull the status flags off the stack on sets the flags accordingly.

Definition at line 2297 of file W65C02.cs.

```
        {
            var flags = PeekStack();
            CarryFlag = (flags & 0x01) != 0;
            ZeroFlag = (flags & 0x02) != 0;
            DisableInterruptFlag = (flags & 0x04) != 0;
            DecimalFlag = (flags & 0x08) != 0;
            OverflowFlag = (flags & 0x40) != 0;
            NegativeFlag = (flags & 0x80) != 0;


        }
```

6.39.3.28 PushFlagsOperation() `void Hardware.W65C02.PushFlagsOperation ( ) [inline], [private]`

The PSP Operation. Pushes the Status Flags to the stack

Definition at line 2289 of file W65C02.cs.

```
        {
            PokeStack(ConvertFlagsToByte(true));
        }
```

6.39.3.29 Reset() `void Hardware.W65C02.Reset ( ) [inline]`

Initializes the processor to its default state.

Definition at line 152 of file W65C02.cs.

```
        {
            ResetCycleCount();
            StackPointer = 0x1FD;
            //Set the Program Counter to the Reset Vector Address.
            ProgramCounter = 0xFFFC;
            //Reset the Program Counter to the Address contained in the Reset Vector
            ProgramCounter = (MemoryMap.Read(ProgramCounter) | (MemoryMap.Read(ProgramCounter + 1) «
      8));
            CurrentOpCode = MemoryMap.Read(ProgramCounter);
            //SetDisassembly();
            DisableInterruptFlag = true;
            _previousInterrupt = false;
            TriggerNmi = false;
            TriggerIRQ = false;
        }
```

6.39.3.30 ResetCycleCount() `void Hardware.W65C02.ResetCycleCount ( ) [inline]`

Resets the Cycle Count back to 0

Definition at line 229 of file W65C02.cs.

```
        {
            _cycleCount = 0;
        }
```

6.39.3.31 ReturnFromInterruptOperation() `void Hardware.W65C02.ReturnFromInterruptOperation ( ) [inline], [private]`

The RTI routine. Called when returning from a BRK opertion. Note: when called after a BRK operation the Program Counter is not set to the location after the BRK, it is set +1

Definition at line 2390 of file W65C02.cs.

```
        {
            MemoryMap.Read(++ProgramCounter);
            StackPointer++;
            IncrementCycleCount();

            PullFlagsOperation();
            StackPointer++;
            IncrementCycleCount();

            var lowBit = PeekStack();
            StackPointer++;
            IncrementCycleCount();

            var highBit = PeekStack() « 8;
            IncrementCycleCount();

            ProgramCounter = (highBit | lowBit);
        }
```

6.39.3.32 ReturnFromSubRoutineOperation() `void Hardware.W65C02.ReturnFromSubRoutineOperation ( ) [inline], [private]`

The RTS routine. Called when returning from a subroutine.

Definition at line 2333 of file W65C02.cs.

```
        {
            MemoryMap.Read(++ProgramCounter);
            StackPointer++;
            IncrementCycleCount();

            var lowBit = PeekStack();
            StackPointer++;
            IncrementCycleCount();

            var highBit = PeekStack() « 8;
            IncrementCycleCount();

            ProgramCounter = (highBit | lowBit) + 1;
            IncrementCycleCount();
        }
```

6.39.3.33 RolOperation() `void Hardware.W65C02.RolOperation ( AddressingMode addressingMode ) [inline], [private]`

The ROL operation. Performs a rotate left operation on a value in memory.

Parameters

addressingMode	The addressing mode to use

Definition at line 2161 of file W65C02.cs.

```
        {
            int value;
            var memoryAddress = 0;
            if (addressingMode == AddressingMode.Accumulator)
            {
                //Dummy MemoryMap.Read
                MemoryMap.Read(ProgramCounter + 1);
                value = Accumulator;
            }
            else
            {
                memoryAddress = GetAddressByAddressingMode(addressingMode);
                value = MemoryMap.Read(memoryAddress);
            }

            //Dummy Write
            if (addressingMode != AddressingMode.Accumulator)
            {
                MemoryMap.Write(memoryAddress, (byte)value);
            }

            //Store the carry flag before shifting it
            var newCarry = (0x80 & value) != 0;

            //The And here ensures that if the value is greater than 255 it wraps properly.
            value = (value « 1) & 0xFE;

            if (CarryFlag)
                value = value | 0x01;

            CarryFlag = newCarry;

            SetZeroFlag(value);
            SetNegativeFlag(value);


            if (addressingMode == AddressingMode.Accumulator)
                Accumulator = value;
            else
            {
                MemoryMap.Write(memoryAddress, (byte)value);
            }
        }
```

6.39.3.34 RorOperation()

`void Hardware.W65C02.RorOperation ( AddressingMode addressingMode ) [inline], [private]`

The ROR operation. Performs a rotate right operation on a value in memory.

Parameters

addressingMode	The addressing mode to use

Definition at line 2210 of file W65C02.cs.

```
        {
            int value;
            var memoryAddress = 0;
            if (addressingMode == AddressingMode.Accumulator)
            {
                //Dummy MemoryMap.Read
                MemoryMap.Read(ProgramCounter + 1);
                value = Accumulator;
            }
            else
            {
                memoryAddress = GetAddressByAddressingMode(addressingMode);
                value = MemoryMap.Read(memoryAddress);
            }

            //Dummy Write
            if (addressingMode != AddressingMode.Accumulator)
            {
                MemoryMap.Write(memoryAddress, (byte)value);
            }

            //Store the carry flag before shifting it
            var newCarry = (0x01 & value) != 0;

            value = (value » 1);

            //If the carry flag is set then 0x
            if (CarryFlag)
                value = value | 0x80;

            CarryFlag = newCarry;

            SetZeroFlag(value);
            SetNegativeFlag(value);

            if (addressingMode == AddressingMode.Accumulator)
                Accumulator = value;
            else
            {
                MemoryMap.Write(memoryAddress, (byte)value);
            }
        }
```

6.39.3.35 SetDisassembly() `void Hardware.W65C02.SetDisassembly ( ) [inline], [private]`

Definition at line 1529 of file W65C02.cs.

```
        {
            if (!_logger.IsDebugEnabled)
                return;

            var addressMode = GetAddressingMode();

            var currentProgramCounter = ProgramCounter;

            currentProgramCounter = WrapProgramCounter(++currentProgramCounter);
            int? address1 = MemoryMap.Read(currentProgramCounter);

            currentProgramCounter = WrapProgramCounter(++currentProgramCounter);
            int? address2 = MemoryMap.Read(currentProgramCounter);

            string disassembledStep = string.Empty;

            switch (addressMode)
            {
                case AddressingMode.Absolute:
                    {
                        disassembledStep = string.Format("${0}{1}",
address2.Value.ToString("X").PadLeft(2, '0'), address1.Value.ToString("X").PadLeft(2, '0'));
                        break;
                    }
                case AddressingMode.AbsoluteX:
                    {
                        disassembledStep = string.Format("${0}{1},X",
address2.Value.ToString("X").PadLeft(2, '0'), address1.Value.ToString("X").PadLeft(2, '0'));
                        break;
                    }
                case AddressingMode.AbsoluteY:
                    {
                        disassembledStep = string.Format("${0}{1},Y",
address2.Value.ToString("X").PadLeft(2, '0'), address1.Value.ToString("X").PadLeft(2, '0'));
                        break;
                    }
                case AddressingMode.Accumulator:
                    {
                        address1 = null;
                        address2 = null;

                        disassembledStep = "A";
                        break;
                    }
                case AddressingMode.Immediate:
                    {
                        disassembledStep = string.Format("#${0}",
address1.Value.ToString("X").PadLeft(4, '0'));
                        address2 = null;
                        break;
                    }
                case AddressingMode.Implied:
                    {
                        address1 = null;
                        address2 = null;
                        break;
                    }
                case AddressingMode.Indirect:
                    {
                        disassembledStep = string.Format("(${0}{1})",
address2.Value.ToString("X").PadLeft(2, '0'), address1.Value.ToString("X").PadLeft(2, '0'));
                        break;
                    }
                case AddressingMode.IndirectX:
                    {
                        address2 = null;

                        disassembledStep = string.Format("(${0},X)",
address1.Value.ToString("X").PadLeft(2, '0'));
                        break;
                    }
                case AddressingMode.IndirectY:
                    {
                        address2 = null;

                        disassembledStep = string.Format("(${0}),Y",
address1.Value.ToString("X").PadLeft(2, '0'));
                        break;
                    }
                case AddressingMode.Relative:
                    {
                        var valueToMove = (byte)address1.Value;

                        var movement = valueToMove > 127 ?  (valueToMove - 255) :  valueToMove;

                        var newProgramCounter = ProgramCounter + movement;

                        //This makes sure that we always land on the correct spot for a positive
number
                        if (movement >= 0)
                            newProgramCounter++;

                        var stringAddress = ProgramCounter.ToString("X").PadLeft(4, '0');

                        address1 = int.Parse(stringAddress.Substring(0, 2),
NumberStyles.AllowHexSpecifier);
                        address2 = int.Parse(stringAddress.Substring(2, 2),
NumberStyles.AllowHexSpecifier);

                        disassembledStep = string.Format("${0}",
newProgramCounter.ToString("X").PadLeft(4, '0'));

                        break;
                    }
                case AddressingMode.ZeroPage:
                    {
                        address2 = null;

                        disassembledStep = string.Format("${0}",
address1.Value.ToString("X").PadLeft(2, '0'));
                        break;
                    }
                case AddressingMode.ZeroPageX:
                    {
                        address2 = null;

                        disassembledStep = string.Format("${0},X",
address1.Value.ToString("X").PadLeft(2, '0'));
                        break;
                    }
                case AddressingMode.ZeroPageY:
                    {
                        address2 = null;

                        disassembledStep = string.Format("${0},Y",
address1.Value.ToString("X").PadLeft(4, '0'));
                        break;
                    }
                default:
                    throw new InvalidEnumArgumentException("Invalid Addressing Mode");

            }

```

```

            CurrentDisassembly = new Disassembly
            {
                HighAddress = address2.HasValue ?  address2.Value.ToString("X").PadLeft(2, '0') : string.Empty,
                LowAddress = address1.HasValue ?  address1.Value.ToString("X").PadLeft(2, '0') : string.Empty,
                OpCodeString = CurrentOpCode.ConvertOpCodeIntoString(),
                DisassemblyOutput = disassembledStep
            };

            _logger.Debug("{0} :  {1}{2}{3} {4} {5} A: {6} X: {7} Y: {8} SP {9} N: {10} V: {11} B: {12} D: {13} I: {14} Z: {15} C: {16}",
                 ProgramCounter.ToString("X").PadLeft(4, '0'),
                 CurrentOpCode.ToString("X").PadLeft(2, '0'),
                 CurrentDisassembly.LowAddress,
                 CurrentDisassembly.HighAddress,

                 CurrentDisassembly.OpCodeString,
                 CurrentDisassembly.DisassemblyOutput.PadRight(10, ' '),

                 Accumulator.ToString("X").PadLeft(3, '0'),
                     XRegister.ToString("X").PadLeft(3, '0'),
                     YRegister.ToString("X").PadLeft(3, '0'),
                     StackPointer.ToString("X").PadLeft(3, '0'),
                     Convert.ToInt16(NegativeFlag),
                     Convert.ToInt16(OverflowFlag),
                     0,
                     Convert.ToInt16(DecimalFlag),
                     Convert.ToInt16(DisableInterruptFlag),
                     Convert.ToInt16(ZeroFlag),
                     Convert.ToInt16(CarryFlag));
        }
```

6.39.3.36 SetNegativeFlag() `void Hardware.W65C02.SetNegativeFlag ( int value ) [inline], [protected]`

Sets the IsSignNegative register

Parameters

value	

Definition at line 1318 of file W65C02.cs.

```
        {
            //on the 6502, any value greater than 127 is negative.  128 = 1000000 in Binary.  the 8th bit is set, therefore the number is a negative number.
            NegativeFlag = value > 127;
        }
```

6.39.3.37 SetZeroFlag() `void Hardware.W65C02.SetZeroFlag ( int value ) [inline], [protected]`

Sets the IsResultZero register

Parameters

value	

Definition at line 1328 of file W65C02.cs.

```
        {
            ZeroFlag = value == 0;
        }
```

6.39.3.38 SubtractWithBorrowOperation() `void Hardware.W65C02.SubtractWithBorrowOperation ( AddressingMode addressingMode ) [inline], [protected]`

The SBC operation. Performs a subtract with carry operation on the accumulator and a value in memory.

Parameters

addressingMode	The addressing mode to use

Definition at line 2258 of file W65C02.cs.

```
        {
            var memoryValue = MemoryMap.Read(GetAddressByAddressingMode(addressingMode));
            var newValue = DecimalFlag ?  int.Parse(Accumulator.ToString("x")) -
      int.Parse(memoryValue.ToString("x")) - (CarryFlag ?  0 :  1) :  Accumulator - memoryValue - (CarryFlag
      ?  0 :  1);

            CarryFlag = newValue >= 0;

            if (DecimalFlag)
            {
                if (newValue < 0)
                    newValue += 100;

                newValue = (int)Convert.ToInt64(string.Concat("0x", newValue), 16);
            }
            else
            {
                OverflowFlag = (((Accumulator ^ newValue) & 0x80) != 0) && (((Accumulator ^
      memoryValue) & 0x80) != 0);

                if (newValue < 0)
                    newValue += 256;
            }

            SetNegativeFlag(newValue);
            SetZeroFlag(newValue);

            Accumulator = newValue;
        }
```

6.39.3.39 WrapProgramCounter() `int Hardware.W65C02.WrapProgramCounter ( int value ) [inline], [private]`

Definition at line 1680 of file W65C02.cs.

```
        {
            return value & 0xFFFF;
        }
```

6.39.4 Member Data Documentation

6.39.4.1 _cycleCount `int Hardware.W65C02._cycleCount [private]`

Definition at line 19 of file W65C02.cs.

6.39.4.2 _interrupt `bool Hardware.W65C02._interrupt [private]`

Definition at line 21 of file W65C02.cs.

6.39.4.3 _logger `readonly ILogger Hardware.W65C02._logger = LogManager.GetLogger("Processor") [private]`

Definition at line 16 of file W65C02.cs.

6.39.4.4 _previousInterrupt `bool Hardware.W65C02._previousInterrupt [private]`

Definition at line 20 of file W65C02.cs.

6.39.4.5 _programCounter `int Hardware.W65C02._programCounter [private]`

Definition at line 17 of file W65C02.cs.

6.39.4.6 _stackPointer `int Hardware.W65C02._stackPointer [private]`

Definition at line 18 of file W65C02.cs.

6.39.4.7 isRunning `bool Hardware.W65C02.isRunning`

Checks shether the emulated computer is running or not.

Definition at line 26 of file W65C02.cs.

6.39.5 Property Documentation

6.39.5.1 Accumulator `int Hardware.W65C02.Accumulator [get], [protected set]`

The Accumulator. This value is implemented as an integer intead of a byte. This is done so we can detect wrapping of the value and set the correct number of cycles.

Definition at line 34 of file W65C02.cs.

```
00034 { get; protected set; }
```

6.39.5.2 CarryFlag `bool Hardware.W65C02.CarryFlag [get], [protected set]`

This is the carry flag. when adding, if the result is greater than 255 or 99 in BCD Mode, then this bit is enabled. In subtraction this is reversed and set to false if a borrow is required IE the result is less than 0

Definition at line 94 of file W65C02.cs.
```
00094 { get; protected set; }
```

6.39.5.3 CurrentDisassembly `Disassembly Hardware.W65C02.CurrentDisassembly [get], [private set]`

The disassembly of the current operation. This value is only set when the CPU is built in debug mode.

Definition at line 54 of file W65C02.cs.
```
00054 { get; private set; }
```

6.39.5.4 CurrentOpCode `int Hardware.W65C02.CurrentOpCode [get], [private set]`

The Current Op Code being executed by the system

Definition at line 49 of file W65C02.cs.
```
00049 { get; private set; }
```

6.39.5.5 CycleCountIncrementedAction `Action Hardware.W65C02.CycleCountIncrementedAction [get], [set]`

An external action that occurs when the cycle count is incremented

Definition at line 87 of file W65C02.cs.
```
00087 { get; set; }
```

6.39.5.6 DecimalFlag `bool Hardware.W65C02.DecimalFlag [get], [private set]`

Binary Coded Decimal Mode is set/cleared via this flag. when this mode is in effect, a byte represents a number from 0-99.

Definition at line 112 of file W65C02.cs.
```
00112 { get; private set; }
```

6.39.5.7 DisableInterruptFlag `bool Hardware.W65C02.DisableInterruptFlag [get], [private set]`

This determines if Interrupts are currently disabled. This flag is turned on during a reset to prevent an interrupt from occuring during startup/Initialization. If this flag is true, then the IRQ pin is ignored.

Definition at line 106 of file W65C02.cs.
```
00106 { get; private set; }
```

6.39.5.8 NegativeFlag `bool Hardware.W65C02.NegativeFlag [get], [private set]`

Set to true if the result of an operation is negative in ADC and SBC operations. Remember that 128-256 represent negative numbers when doing signed math. In shift operations the sign holds the carry.

Definition at line 128 of file W65C02.cs.

```
{ get; private set; }
```

6.39.5.9 OverflowFlag `bool Hardware.W65C02.OverflowFlag [get], [protected set]`

This property is set when an overflow occurs. An overflow happens if the high bit(7) changes during the operation. Remember that values from 128-256 are negative values as the high bit is set to 1. Examples: 64 + 64 = -128 -128 + -128 = 0

Definition at line 121 of file W65C02.cs.

```
{ get; protected set; }
```

6.39.5.10 ProgramCounter `int Hardware.W65C02.ProgramCounter [get], [private set]`

Points to the Current Address of the instruction being executed by the system. The PC wraps when the value is greater than 65535, or less than 0.

Definition at line 60 of file W65C02.cs.

```
        {
            get { return _programCounter; }
            private set { _programCounter = WrapProgramCounter(value); }
        }
```

6.39.5.11 StackPointer `int Hardware.W65C02.StackPointer [get], [private set]`

Points to the Current Position of the Stack. This value is a 00-FF value but is offset to point to the location in memory where the stack resides.

Definition at line 70 of file W65C02.cs.

```
        {
            get { return _stackPointer; }
            private set
            {
                if (value > 0xFF)
                    _stackPointer = value - 0x100;
                else if (value < 0x00)
                    _stackPointer = value + 0x100;
                else
                    _stackPointer = value;
            }
        }
```

6.39.5.12 TriggerIRQ `bool Hardware.W65C02.TriggerIRQ [get], [private set]`

Set to true when an IRQ has occurred and is being processed by the CPU.

Definition at line 136 of file W65C02.cs.

```
{ get; private set; }
```

6.39.5.13 TriggerNmi `bool Hardware.W65C02.TriggerNmi [get], [set]`

Set to true when an NMI should occur

Definition at line 133 of file W65C02.cs.

```
00133 { get; set; }
```

6.39.5.14 XRegister `int Hardware.W65C02.XRegister [get], [private set]`

The X Index Register

Definition at line 39 of file W65C02.cs.

```
00039 { get; private set; }
```

6.39.5.15 YRegister `int Hardware.W65C02.YRegister [get], [private set]`

The Y Index Register

Definition at line 44 of file W65C02.cs.

```
00044 { get; private set; }
```

6.39.5.16 ZeroFlag `bool Hardware.W65C02.ZeroFlag [get], [private set]`

Is true if one of the registers is set to zero.

Definition at line 99 of file W65C02.cs.

```
00099 { get; private set; }
```

The documentation for this class was generated from the following file:

- Hardware/W65C02.cs

6.40 Hardware.W65C22 Class Reference

An implementation of a W65C22 VIA.

Public Member Functions

- W65C22 (W65C02 processor, byte offset, int length)
- void Reset ()

  Reset routine called whenever the emulated computer is reset.
- void Init (double timer)

  Initialization routine for the VIA.
- void T1Init (double value)

  T1 counter initialization routine.
- void T2Init (double value)

  T2 counter initialization routine.
- byte Read (int address)

  Routine to read from local memory.
- void Write (int address, byte data)

  Writes data to the specified address in local memory.

Public Attributes

- readonly bool T1IsIRQ = false
- readonly bool T2IsIRQ = true
- int T1CL = 0x04
- int T1CH = 0x05
- int T2CL = 0x08
- int T2CH = 0x09
- int ACR = 0x0B
- int IFR = 0x0D
- int IER = 0x0E
- byte ACR_T1TC = (byte)(1 << 7)
- byte ACR_T2TC = (byte)(1 << 6)
- byte IFR_T2 = (byte)(1 << 5)
- byte IFR_T1 = (byte)(1 << 6)
- byte IFR_INT = (byte)(1 << 7)
- byte IER_T2 = (byte)(1 << 5)
- byte IER_T1 = (byte)(1 << 6)
- byte IER_EN = (byte)(1 << 7)

Properties

- byte[ ] Memory `[get, set]`

  The memory area.
- int Offset `[get, set]`

  The memory offset of the device.
- int Length `[get, set]`

  The length of the device memory.
- int End `[get]`

  The end of memory
- bool T1TimerControl `[get, set]`

  T1 timer control
- bool T2TimerControl `[get, set]`

  T2 timer control.
- bool T1IsEnabled `[get, set]`

  Enable or check whether timer 1 is enabled or not.
- bool T2IsEnabled `[get, set]`

  Enable or check whether timer 2 is enabled or not.
- double T1Interval `[get]`

  Set or check the timer 1 interval.
- double T2Interval `[get]`

  Set or check the timer 2 interval.
- Timer T1Object `[get, set]`

  Set or get the timer 1 object.
- Timer T2Object `[get, set]`

  Set or get the timer 2 object.
- W65C02 Processor `[get, set]`

  Local referemce to the processor object.

Private Member Functions

- void OnT1Timeout (object sender, ElapsedEventArgs e)

  Called whenever System.Timers.Timer event elapses.
- void OnT2Timeout (object sender, ElapsedEventArgs e)

  Called whenever System.Timers.Timer event elapses

6.40.1 Detailed Description

An implementation of a W65C22 VIA.

Definition at line 11 of file W65C22.cs.

6.40.2 Constructor & Destructor Documentation

6.40.2.1 W65C22()

```
Hardware.W65C22.W65C22 (
        W65C02 processor,
        byte offset,
        int length )  [inline]
```

Definition at line 123 of file W65C22.cs.

```
        {
            if (offset > MemoryMap.DeviceArea.Length)
                throw new ArgumentException(String.Format("The offset:  {0} is greater than the device area:  {1}", offset, MemoryMap.DeviceArea.Length));
            T1Init(1000);
            T2Init(1000);

            Offset = MemoryMap.DeviceArea.Offset | offset;
            Memory = new byte[length + 1];
            Length = length;
            Processor = processor;
        }
```

6.40.3 Member Function Documentation

6.40.3.1 Init()

```
void Hardware.W65C22.Init (
        double timer )  [inline]
```

Initialization routine for the VIA.

Parameters

timer	Amount of time to set timers for.

Definition at line 151 of file W65C22.cs.

```
        {
            T1Init(timer);
            T2Init(timer);
        }
```

6.40.3.2 OnT1Timeout()

`void Hardware.W65C22.OnT1Timeout (`
`object sender,`
`ElapsedEventArgs e ) [inline], [private]`

Called whenever System.Timers.Timer event elapses.

Parameters

sender	
e	

Definition at line 248 of file W65C22.cs.

```
        {
            if (Processor.isRunning)
            {
                if (T1IsEnabled)
                {
                    Write(IFR, (byte)(IFR_T1 & IFR_INT));
                    if (T1IsIRQ)
                    {
                        Processor.InterruptRequest();
                    }
                    else
                    {
                        Processor.TriggerNmi = true;
                    }
                }
            }
        }
```

6.40.3.3 OnT2Timeout()

`void Hardware.W65C22.OnT2Timeout (`
`object sender,`
`ElapsedEventArgs e ) [inline], [private]`

Called whenever System.Timers.Timer event elapses

Parameters

sender	
e	

Definition at line 273 of file W65C22.cs.

```
        {
            if (Processor.isRunning)
            {
                if (T2IsEnabled)
                {
                    Write(IFR, (byte)(IFR_T2 & IFR_INT));
                    if (T2IsIRQ)
                    {
                        Processor.InterruptRequest();
                    }
                    else
                    {
                        Processor.TriggerNmi = true;
                    }
                }
            }
        }
```

6.40.3.4 Read() `byte Hardware.W65C22.Read ( int address ) [inline]`

Routine to read from local memory.

Parameters

address	Address to read from.

Returns

Byte value stored in the local memory.

Definition at line 192 of file W65C22.cs.

```
        {
            if ((Offset <= address) && (address <= End))
            {
                byte data = 0x00;
                if (T1TimerControl)
                {
                    data = (byte)(data | ACR_T1TC);
                }
                else if (T2TimerControl)
                {
                    data = (byte)(data | ACR_T2TC);
                }
                return data;
            }
            else
            {
                return Memory[address - Offset];
            }
        }
```

6.40.3.5 Reset() `void Hardware.W65C22.Reset ( ) [inline]`

Reset routine called whenever the emulated computer is reset.

Definition at line 139 of file W65C22.cs.

```
        {
            T1TimerControl = false;
            T1IsEnabled = false;
            T2TimerControl = false;
            T2IsEnabled = false;
        }
```

6.40.3.6 T1Init() `void Hardware.W65C22.T1Init ( double value ) [inline]`

T1 counter initialization routine.

Parameters

value	Timer initialization value in milliseconds.

Definition at line 162 of file W65C22.cs.

```
        {
            T1Object = new Timer(value);
            T1Object.Start();
            T1Object.Elapsed += OnT1Timeout;
            T1TimerControl = true;
            T1IsEnabled = false;
        }
```

6.40.3.7 T2Init()

`void Hardware.W65C22.T2Init ( double value ) [inline]`

T2 counter initialization routine.

Parameters

value	Timer initialization value in milliseconds.

Definition at line 176 of file W65C22.cs.

```
        {
            T2Object = new Timer(value);
            T2Object.Start();
            T2Object.Elapsed += OnT2Timeout;
            T2TimerControl = true;
            T2IsEnabled = false;
        }
```

6.40.3.8 Write()

`void Hardware.W65C22.Write ( int address, byte data ) [inline]`

Writes data to the specified address in local memory.

Parameters

address	The address to write data to.
data	The data to be written.

Definition at line 219 of file W65C22.cs.

```
        {
            if ((address == Offset + ACR) && ((data | ACR_T1TC) == ACR_T1TC))
            {
                T1TimerControl = true;
            }
            else if ((address == Offset + ACR) && ((data | ACR_T2TC) == ACR_T2TC))
            {
                T2TimerControl = true;
            }
            else if ((address == Offset + IER) && ((data | IER_T1) == IER_T1) && ((data | IER_EN) ==
    IER_EN))
            {
                T1Init(T1Interval);
            }
            else if ((address == Offset + IER) && ((data | IER_T2) == IER_T2) && ((data | IER_EN) ==
      IER_EN))
            {
                T2Init(T2Interval);
            }
            Memory[address - Offset] = data;
        }
```

6.40.4 Member Data Documentation

6.40.4.1 ACR `int Hardware.W65C22.ACR = 0x0B`

Definition at line 20 of file W65C22.cs.

6.40.4.2 ACR_T1TC `byte Hardware.W65C22.ACR_T1TC = (byte)(1 << 7)`

Definition at line 24 of file W65C22.cs.

6.40.4.3 ACR_T2TC `byte Hardware.W65C22.ACR_T2TC = (byte)(1 << 6)`

Definition at line 25 of file W65C22.cs.

6.40.4.4 IER `int Hardware.W65C22.IER = 0x0E`

Definition at line 22 of file W65C22.cs.

6.40.4.5 IER_EN `byte Hardware.W65C22.IER_EN = (byte)(1 << 7)`

Definition at line 33 of file W65C22.cs.

6.40.4.6 IER_T1 `byte Hardware.W65C22.IER_T1 = (byte)(1 << 6)`

Definition at line 32 of file W65C22.cs.

6.40.4.7 IER_T2 `byte Hardware.W65C22.IER_T2 = (byte)(1 << 5)`

Definition at line 31 of file W65C22.cs.

6.40.4.8 IFR `int Hardware.W65C22.IFR = 0x0D`

Definition at line 21 of file W65C22.cs.

6.40.4.9 IFR_INT `byte Hardware.W65C22.IFR_INT = (byte)(1 << 7)`

Definition at line 29 of file W65C22.cs.

6.40.4.10 IFR_T1 `byte Hardware.W65C22.IFR_T1 = (byte)(1 << 6)`

Definition at line 28 of file W65C22.cs.

6.40.4.11 IFR_T2 `byte Hardware.W65C22.IFR_T2 = (byte)(1 << 5)`

Definition at line 27 of file W65C22.cs.

6.40.4.12 T1CH `int Hardware.W65C22.T1CH = 0x05`

Definition at line 17 of file W65C22.cs.

6.40.4.13 T1CL `int Hardware.W65C22.T1CL = 0x04`

Definition at line 16 of file W65C22.cs.

6.40.4.14 T1IsIRQ `readonly bool Hardware.W65C22.T1IsIRQ = false`

Definition at line 14 of file W65C22.cs.

6.40.4.15 T2CH `int Hardware.W65C22.T2CH = 0x09`

Definition at line 19 of file W65C22.cs.

6.40.4.16 T2CL `int Hardware.W65C22.T2CL = 0x08`

Definition at line 18 of file W65C22.cs.

6.40.4.17 T2IsIRQ `readonly bool Hardware.W65C22.T2IsIRQ = true`

Definition at line 15 of file W65C22.cs.

6.40.5 Property Documentation

6.40.5.1 End `int Hardware.W65C22.End [get]`

The end of memory

Definition at line 55 of file W65C22.cs.
```
00055 { get { return Offset + Length; } }
```

6.40.5.2 Length `int Hardware.W65C22.Length [get], [set]`

The length of the device memory.

Definition at line 50 of file W65C22.cs.
```
00050 { get; set; }
```

6.40.5.3 Memory `byte [] Hardware.W65C22.Memory [get], [set]`

The memory area.

Definition at line 40 of file W65C22.cs.
```
00040 { get; set; }
```

6.40.5.4 Offset `int Hardware.W65C22.Offset [get], [set]`

The memory offset of the device.

Definition at line 45 of file W65C22.cs.

```
00045 { get; set; }
```

6.40.5.5 Processor `W65C02 Hardware.W65C22.Processor [get], [set], [private]`

Local referemce to the processor object.

Definition at line 119 of file W65C22.cs.

```
00119 { get; set; }
```

6.40.5.6 T1Interval `double Hardware.W65C22.T1Interval [get]`

Set or check the timer 1 interval.

Definition at line 96 of file W65C22.cs.

```
00096 { get { return (int)(Read(T1CL) | (Read(T1CH) « 8)); } }
```

6.40.5.7 T1IsEnabled `bool Hardware.W65C22.T1IsEnabled [get], [set]`

Enable or check whether timer 1 is enabled or not.

Definition at line 78 of file W65C22.cs.

```
00079         {
00080             get { return T1Object.Enabled; }
00081             set { T1Object.Enabled = value; }
00082         }
```

6.40.5.8 T1Object `Timer Hardware.W65C22.T1Object [get], [set]`

Set or get the timer 1 object.

Definition at line 109 of file W65C22.cs.

```
00109 { get; set; }
```

6.40.5.9 T1TimerControl `bool Hardware.W65C22.T1TimerControl [get], [set]`

T1 timer control

Definition at line 60 of file W65C22.cs.

```
00061         {
00062             get { return T1Object.AutoReset; }
00063             set { T1Object.AutoReset = value; }
00064         }
```

6.40.5.10 T2Interval `double Hardware.W65C22.T2Interval [get]`

Set or check the timer 2 interval.

Definition at line 101 of file W65C22.cs.

```
00102         {
00103             get { return (int)(Read(T2CL) | (Read(T2CH) « 8)); }
00104         }
```

6.40.5.11 T2IsEnabled `bool Hardware.W65C22.T2IsEnabled [get], [set]`

Enable or check whether timer 2 is enabled or not.

Definition at line 87 of file W65C22.cs.

```
00088         {
00089             get { return T2Object.Enabled; }
00090             set { T2Object.Enabled = value; }
00091         }
```

6.40.5.12 T2Object `Timer Hardware.W65C22.T2Object [get], [set]`

Set or get the timer 2 object.

Definition at line 114 of file W65C22.cs.

```
00114 { get; set; }
```

6.40.5.13 T2TimerControl `bool Hardware.W65C22.T2TimerControl [get], [set]`

T2 timer control.

Definition at line 69 of file W65C22.cs.

```
00070         {
00071             get { return T2Object.AutoReset; }
00072             set { T2Object.AutoReset = value; }
00073         }
```

The documentation for this class was generated from the following file:

- Hardware/W65C22.cs

6.41 Hardware.W65C51 Class Reference

An implementation of a W65C51 ACIA.

Public Member Functions

- W65C51 (W65C02 processor, byte offset)
- void Reset ()
- void Init (string port)

    Default Constructor, Instantiates a new instance of COM Port I/O.
- void Init (string port, int baudRate)

    Default Constructor, Instantiates a new instance of COM Port I/O.
- void Fini ()

    Called when the window is closed.
- byte Read (int address)

    Returns the byte at a given address.
- void Write (int address, byte data)

    Writes data to the given address.
- void WriteCOM (byte data)

    Called in order to write to the serial port.

Public Attributes

- readonly int defaultBaudRate = 115200
- byte byteIn

Properties

- byte[ ] Memory `[get, set]`
- bool IsEnabled `[get, set]`
- SerialPort Object `[get, set]`
- string ObjectName `[get, set]`
- W65C02 Processor `[get, set]`
- BackgroundWorker _backgroundWorker `[get, set]`
- int Offset `[get, set]`
- int Length `[get, set]`
- bool DataRead `[get, set]`
- bool EchoMode `[get, set]`
- bool InterruptDisabled `[get, set]`
- bool Interrupted `[get, set]`
- bool Overrun `[get, set]`
- bool ParityEnabled `[get, set]`
- bool ReceiverFull `[get, set]`
- byte RtsControl `[get, set]`

Private Member Functions

- void ComInit (SerialPort serialPort)

    Called whenever the ACIA is initialized.
- void ComFini (SerialPort serialPort)

    Called when the window is closed.
- void SerialDataReceived (object sender, SerialDataReceivedEventArgs e)

    Called whenever SerialDataReceivedEventHandler event occurs.
- void HardwarePreWrite (int address, byte data)
- void HardwarePreRead (int address)
- void CommandRegister (byte data)
- void CommandRegisterUpdate ()
- void ControlRegister (byte data)
- void ControlRegisterUpdate ()
- void StatusRegisterUpdate ()
- void BackgroundWorkerDoWork (object sender, DoWorkEventArgs e)

6.41.1 Detailed Description

An implementation of a W65C51 ACIA.

Definition at line 13 of file W65C51.cs.

6.41.2 Constructor & Destructor Documentation

6.41.2.1 W65C51() `Hardware.W65C51.W65C51 ( W65C02 processor, byte offset ) [inline]`

Definition at line 41 of file W65C51.cs.

```
        {
            if (offset > MemoryMap.DeviceArea.Length)
                throw new ArgumentException(String.Format("The offset:  {0} is greater than the device
    area:  {1}", offset, MemoryMap.DeviceArea.Length));

            Processor = processor;

            Offset = MemoryMap.DeviceArea.Offset | offset;
            Length = 0x04;
            Memory = new byte[Length + 1];

            _backgroundWorker = new BackgroundWorker
            {
                WorkerSupportsCancellation = true
            };
            _backgroundWorker.DoWork += BackgroundWorkerDoWork;
            _backgroundWorker.RunWorkerAsync();
        }
```

6.41.3 Member Function Documentation

6.41.3.1 BackgroundWorkerDoWork() `void Hardware.W65C51.BackgroundWorkerDoWork ( object sender, DoWorkEventArgs e ) [inline], [private]`

Definition at line 678 of file W65C51.cs.

```
        {
            var worker = sender as BackgroundWorker;

            while (true)
            {
                if (worker != null && worker.CancellationPending)
                {
                    e.Cancel = true;
                    return;
                }

                if (Processor.isRunning)
                {
                    if (ReceiverFull || Overrun)
                    {
                        Memory[Offset + 1] = (byte)(Memory[Offset + 1] | 0x80);
                        Interrupted = true;
                        Processor.InterruptRequest();
                    }

                    if (DataRead)
                    {
                        ReceiverFull = false;
                        Interrupted = false;
                        Overrun = false;
                        DataRead = false;
                    }
                }
            }
        }
```

6.41.3.2 ComFini() `void Hardware.W65C51.ComFini ( SerialPort serialPort ) [inline], [private]`

Called when the window is closed.

Parameters

serialPort	SerialPort Object to close

Definition at line 196 of file W65C51.cs.

```
        {
            if (serialPort != null)
            {
                serialPort.Close();
            }

            _backgroundWorker.CancelAsync();
            _backgroundWorker.DoWork -= BackgroundWorkerDoWork;
        }
```

6.41.3.3 ComInit() `void Hardware.W65C51.ComInit ( SerialPort serialPort ) [inline], [private]`

Called whenever the ACIA is initialized.

Parameters

serialPort	SerialPort object to initialize.

Definition at line 148 of file W65C51.cs.

```
        {
            try
            {
                serialPort.Open();
            }
            catch (UnauthorizedAccessException w)
            {
                FileStream file = new FileStream(FileLocations.ErrorFile, FileMode.OpenOrCreate,
      FileAccess.ReadWrite);
                StreamWriter stream = new StreamWriter(file);
                stream.WriteLine(w.Message);
                stream.WriteLine(w.Source);
                stream.Flush();
                file.Flush();
                stream.Close();
                file.Close();
                return;
            }
            serialPort.ReadTimeout = 50;
            serialPort.WriteTimeout = 50;
            serialPort.DataReceived += new SerialDataReceivedEventHandler(SerialDataReceived);
            try
            {
                serialPort.Write("---------------------------\r\n");
                serialPort.Write(" WolfNet 6502 WBC Emulator\r\n");
                serialPort.Write("---------------------------\r\n");
                serialPort.Write("\r\n");
            }
            catch (TimeoutException t)
            {
                _ = t;
                FileStream file = new FileStream(FileLocations.ErrorFile, FileMode.OpenOrCreate,
      FileAccess.ReadWrite);
                StreamWriter stream = new StreamWriter(file);
                stream.WriteLine("Read/Write error:  Port timed out!");
                stream.WriteLine("Please ensure all cables are connected properly!");
                stream.Flush();
                file.Flush();
                stream.Close();
                file.Close();
                return;
            }
        }
```

6.41.3.4 CommandRegister() `void Hardware.W65C51.CommandRegister ( byte data ) [inline], [private]`

Definition at line 297 of file W65C51.cs.

```
        {
            byte test = (byte)(data & 0x20);
            if (test == 0x20)
            {
                throw new ArgumentException("Parity must NEVER be enabled!");
            }

            test = (byte)(data & 0x10);
            if (test == 0x10)
            {
                EchoMode = true;
            }
            else
            {
                EchoMode = false;
            }

            test = (byte)(data & 0x0C);
            if (test == 0x00)
            {
                Object.Handshake = Handshake.None;
                Object.RtsEnable = true;
                Object.Handshake = Handshake.RequestToSend;
            }
            else if (test == 0x04)
            {
                Object.Handshake = Handshake.None;
                Object.RtsEnable = false;
            }
            else if ((test == 0x08) || (test == 0x0C))
            {
                throw new NotImplementedException("This cannot be emulated on windows!");
            }
            else
            {
                throw new ArgumentOutOfRangeException("RtsControl is invalid!");
            }

            test = (byte)(data & 0x02);
            if (test == 0x02)
            {
                InterruptDisabled = true;
            }
            else
            {
                InterruptDisabled = false;
            }

            test = (byte)(data & 0x01);
            if (test == 0x01)
            {
                Object.DtrEnable = true;
            }
            else
            {
                Object.DtrEnable= false;
            }
        }
```

6.41.3.5 CommandRegisterUpdate() `void Hardware.W65C51.CommandRegisterUpdate ( ) [inline], [private]`

Definition at line 357 of file W65C51.cs.

```
        {
            byte data = Memory[Offset + 2];

            if (ParityEnabled)
            {
                data |= 0x20;
            }
            else
            {
                data &= 0xD0;
            }

            if (EchoMode)
            {
                data |= 0x10;
            }
            else
            {
                data &= 0xE0;
            }

            data &= RtsControl;

            if (InterruptDisabled)
            {
                data |= 0x02;
            }
            else
            {
                data &= 0x0D;
            }
            if (Object.DtrEnable)
            {
                data |= 0x01;
            }
            else
            {
                data &= 0x0E;
            }

            Memory[Offset + 2] = data;
        }
```

6.41.3.6 ControlRegister() `void Hardware.W65C51.ControlRegister ( byte data ) [inline], [private]`

Definition at line 401 of file W65C51.cs.

```
        {
            byte test = (byte)(data & 0x80);
            if (test == 0x80)
            {
                test = (byte)(data & 0x60);
                if (test == 0x60)
                {
                    Object.StopBits = StopBits.OnePointFive;
                }
                else
                {
                    Object.StopBits = StopBits.Two;
                }
            }
            else
            {
                Object.StopBits = StopBits.One;
            }

            test = (byte)(data & 0x60);
            if (test == 0x20)
            {
                Object.DataBits = 7;
            }
            else if (test == 0x40)
            {
                Object.DataBits = 6;
            }
            else if (test == 0x60)
            {
                Object.DataBits = 5;
            }
            else
            {
                Object.DataBits = 8;
            }

            test = (byte)(data & 0x10);
            if (!(test == 0x10))
            {
                throw new ArgumentException("External clock rate not available on the WolfNet 65C02
      WBC!");
            }

            test = (byte)(data & 0x0F);
            if (test == 0x00)
            {
                Object.BaudRate = 115200;
            }
            else if (test == 0x01)
            {
                Object.BaudRate = 50;
            }
            else if (test == 0x02)
            {
                Object.BaudRate = 75;
            }
            else if (test == 0x03)
            {
                Object.BaudRate = 110;
            }
            else if (test == 0x04)
            {
                Object.BaudRate = 135;
            }
            else if (test == 0x05)
            {
                Object.BaudRate = 150;
            }
            else if (test == 0x06)
            {
                Object.BaudRate = 300;
            }
            else if (test == 0x07)
            {
                Object.BaudRate = 600;
            }
            else if (test == 0x08)
            {
                Object.BaudRate = 1200;
            }
            else if (test == 0x09)
            {
                Object.BaudRate = 1800;
            }
            else if (test == 0x0A)
            {
                Object.BaudRate = 2400;
            }
            else if (test == 0x0B)
            {
                Object.BaudRate = 3600;
            }
            else if (test == 0x0C)
            {
                Object.BaudRate = 4800;
            }
            else if (test == 0x0D)
            {
                Object.BaudRate = 7200;
            }
            else if (test == 0x0E)
            {
                Object.BaudRate = 9600;
            }
            else
            {
                Object.BaudRate = 19200;
            }
        }
```

6.41.3.7 ControlRegisterUpdate() `void Hardware.W65C51.ControlRegisterUpdate ( ) [inline], [private]`

Definition at line 512 of file W65C51.cs.

```
        {
            byte controlRegister = Memory[Offset + 3];

            if (Object.StopBits == StopBits.Two)
            {
                controlRegister |= 0x80;
            }
            else if ((Object.StopBits == StopBits.OnePointFive) && (Object.DataBits == 5) ||
(Object.StopBits == StopBits.One))
            {
                controlRegister &= 0x7F;
            }
            else
            {
                throw new ArgumentOutOfRangeException("StopBits or combination of StopBits and
DataBits is invalid!");
            }

            if (Object.DataBits == 8)
            {
                controlRegister &= 0x9F;
            }
            else if (Object.DataBits == 7)
            {
                controlRegister |= 0x20;
            }
            else if (Object.DataBits == 6)
            {
                controlRegister |= 0x40;
            }
            else if (Object.DataBits == 5)
            {
                controlRegister |= 0x60;
            }
            else
            {
                throw new ArgumentOutOfRangeException("DataBits is out of range!");
            }

            if (Object.BaudRate == 115200)
            {
                controlRegister &= 0xF0;
            }
            else if (Object.BaudRate == 50)
            {
                controlRegister |= 0x01;
            }
            else if (Object.BaudRate == 75)
            {
                controlRegister |= 0x02;
            }
            else if (Object.BaudRate == 110)
            {
                controlRegister |= 0x03;
            }
            else if (Object.BaudRate == 135)
            {
                controlRegister |= 0x04;
            }
            else if (Object.BaudRate == 150)
            {
                controlRegister |= 0x05;
            }
            else if (Object.BaudRate == 300)
            {
                controlRegister |= 0x06;
            }
            else if (Object.BaudRate == 600)
            {
                controlRegister |= 0x07;
            }
            else if (Object.BaudRate == 1200)
            {
                controlRegister |= 0x08;
            }
            else if (Object.BaudRate == 1800)
            {
                controlRegister |= 0x09;
            }
            else if (Object.BaudRate == 2400)
            {
                controlRegister |= 0x0A;
            }
            else if (Object.BaudRate == 3600)
            {
                controlRegister |= 0x0B;
            }
            else if (Object.BaudRate == 4800)
            {
                controlRegister |= 0x0C;
            }
            else if (Object.BaudRate == 7200)
            {
                controlRegister |= 0x0D;
            }
            else if (Object.BaudRate == 9600)
            {
                controlRegister |= 0x0E;
            }
            else if (Object.BaudRate == 19200)
            {
                controlRegister |= 0x0F;
            }
            else
            {
                throw new ArgumentOutOfRangeException("BaudRate is outside the range of Baud Rates supported by the W65C51!");
            }

            Memory[Offset + 3] = controlRegister;
        }
```

6.41.3.8 Fini() `void Hardware.W65C51.Fini ( ) [inline]`

Called when the window is closed.

Definition at line 95 of file W65C51.cs.

```
        {
            ComFini(Object);
        }
```

6.41.3.9 HardwarePreRead() `void Hardware.W65C51.HardwarePreRead ( int address ) [inline], [private]`

Definition at line 274 of file W65C51.cs.

```
        {
            if (address == Offset)
            {
                Interrupted = false;
                Overrun = false;
                ReceiverFull = false;

            }
            else if (address == Offset + 1)
            {
                StatusRegisterUpdate();
            }
            else if (address == Offset + 2)
            {
                CommandRegisterUpdate();
            }
            else if (address == Offset + 3)
            {
                ControlRegisterUpdate();
            }
        }
```

6.41.3.10 HardwarePreWrite() `void Hardware.W65C51.HardwarePreWrite ( int address, byte data ) [inline], [private]`

Definition at line 254 of file W65C51.cs.

```
        {
            if (address == Offset)
            {
                WriteCOM(data);
            }
            else if (address == Offset + 1)
            {
                Reset();
            }
            else if (address == Offset + 2)
            {
                CommandRegister(data);
            }
            else if (address == Offset + 3)
            {
                ControlRegister(data);
            }
        }
```

6.41.3.11 Init() [1/2] `void Hardware.W65C51.Init ( string port ) [inline]`

Default Constructor, Instantiates a new instance of COM Port I/O.

Parameters

port	COM Port to use for I/O

Definition at line 70 of file W65C51.cs.

```
        {
            Object = new SerialPort(port, defaultBaudRate, Parity.None, 8, StopBits.One);
            ObjectName = port;

            ComInit(Object);
        }
```

6.41.3.12 Init() [2/2] `void Hardware.W65C51.Init ( string port, int baudRate ) [inline]`

Default Constructor, Instantiates a new instance of COM Port I/O.

Parameters

port	COM Port to use for I/O
baudRate	Baud Rate to use for I/O

Definition at line 84 of file W65C51.cs.

```
        {
            Object = new SerialPort(port, baudRate, Parity.None, 8, StopBits.One);
            ObjectName = port;

            ComInit(Object);
        }
```

6.41.3.13 Read() `byte Hardware.W65C51.Read ( int address ) [inline]`

Returns the byte at a given address.

Parameters

address	

Returns

the byte being returned

Definition at line 107 of file W65C51.cs.

```
        {
            HardwarePreRead(address);
            byte data = Memory[address - Offset];
            DataRead = true;
            return data;
        }
```

6.41.3.14 Reset() `void Hardware.W65C51.Reset ( ) [inline]`

Definition at line 60 of file W65C51.cs.

```
        {
            IsEnabled = false;
        }
```

6.41.3.15 SerialDataReceived() `void Hardware.W65C51.SerialDataReceived ( object sender, SerialDataReceivedEventArgs e ) [inline], [private]`

Called whenever SerialDataReceivedEventHandler event occurs.

Parameters

sender	
e	

Definition at line 213 of file W65C51.cs.

```
        {
            try
            {
                if (EchoMode)
                {
                    WriteCOM(Convert.ToByte(Object.ReadByte()));
                }
                else
                {
                    if (!ReceiverFull)
                    {
                        ReceiverFull = true;
                    }
                    else
                    {
                        Overrun = true;
                    }
                    Memory[0] = Convert.ToByte(Object.ReadByte());
                }

                if (!InterruptDisabled)
                {
                    Interrupted = true;
                    Processor.InterruptRequest();
                }
            }
            catch (Win32Exception w)
            {
                FileStream file = new FileStream(FileLocations.ErrorFile, FileMode.OpenOrCreate, FileAccess.ReadWrite);
                StreamWriter stream = new StreamWriter(file);
                stream.WriteLine(w.Message);
                stream.WriteLine(w.ErrorCode.ToString());
                stream.WriteLine(w.Source);
                stream.Flush();
                stream.Close();
                file.Flush();
                file.Close();
            }
        }
```

6.41.3.16 StatusRegisterUpdate() `void Hardware.W65C51.StatusRegisterUpdate ( )  [inline], [private]`

Definition at line 622 of file W65C51.cs.

```
        {
            byte statusRegister = Memory[Offset + 1];

            if (Interrupted)
            {
                statusRegister |= 0x80;
            }
            else
            {
                statusRegister &= 0x7F;
            }

            if (Object.DsrHolding == false)
            {
                statusRegister |= 0x40;
            }
            else
            {
                statusRegister &= 0xBF;
            }

            if (Object.CDHolding)
            {
                statusRegister |= 0x20;
            }
            else
            {
                statusRegister &= 0xDF;
            }

            statusRegister |= 0x10;

            if (ReceiverFull)
            {
                statusRegister |= 0x08;
            }
            else
            {
                statusRegister &= 0xF7;
            }

            if (Overrun)
            {
                statusRegister |= 0x04;
            }
            else
            {
                statusRegister &= 0xFB;
            }

            statusRegister &= 0xFC;

            Memory[Offset + 1] = statusRegister;
        }
```

6.41.3.17 Write()

```
void Hardware.W65C51.Write (
    int address,
    byte data ) [inline]
```

Writes data to the given address.

Parameters

address	The address to write data to
data	The data to write

Definition at line 121 of file W65C51.cs.

```
        {
            HardwarePreWrite(address, data);
            if (!((address == Offset) || (address == Offset + 1)))
            {
                Memory[address - Offset] = data;
            }
        }
```

6.41.3.18 WriteCOM()

```
void Hardware.W65C51.WriteCOM (
    byte data ) [inline]
```

Called in order to write to the serial port.

Parameters

data	Byte of data to send

Definition at line 135 of file W65C51.cs.

```
        {
            byte[] writeByte = new byte[] { data };
            Object.Write(writeByte, 0, 1);
        }
```

6.41.4 Member Data Documentation

6.41.4.1 byteIn

```
byte Hardware.W65C51.byteIn
```

Definition at line 17 of file W65C51.cs.

6.41.4.2 defaultBaudRate `readonly int Hardware.W65C51.defaultBaudRate = 115200`

Definition at line 16 of file W65C51.cs.

6.41.5 Property Documentation

6.41.5.1 _backgroundWorker `BackgroundWorker Hardware.W65C51._backgroundWorker [get], [set], [private]`

Definition at line 26 of file W65C51.cs.

```
00026 { get; set; }
```

6.41.5.2 DataRead `bool Hardware.W65C51.DataRead [get], [set], [private]`

Definition at line 30 of file W65C51.cs.

```
00030 { get; set; }
```

6.41.5.3 EchoMode `bool Hardware.W65C51.EchoMode [get], [set], [private]`

Definition at line 31 of file W65C51.cs.

```
00031 { get; set; }
```

6.41.5.4 InterruptDisabled `bool Hardware.W65C51.InterruptDisabled [get], [set], [private]`

Definition at line 32 of file W65C51.cs.

```
00032 { get; set; }
```

6.41.5.5 Interrupted `bool Hardware.W65C51.Interrupted [get], [set], [private]`

Definition at line 33 of file W65C51.cs.

```
00033 { get; set; }
```

6.41.5.6 IsEnabled `bool Hardware.W65C51.IsEnabled [get], [set]`

Definition at line 22 of file W65C51.cs.

```
00022 { get; set; }
```

6.41.5.7 Length `int Hardware.W65C51.Length [get], [set]`

Definition at line 28 of file W65C51.cs.
```
00028 { get; set; }
```

6.41.5.8 Memory `byte [ ] Hardware.W65C51.Memory [get], [set]`

Definition at line 21 of file W65C51.cs.
```
00021 { get; set; }
```

6.41.5.9 Object `SerialPort Hardware.W65C51.Object [get], [set]`

Definition at line 23 of file W65C51.cs.
```
00023 { get; set; }
```

6.41.5.10 ObjectName `string Hardware.W65C51.ObjectName [get], [set]`

Definition at line 24 of file W65C51.cs.
```
00024 { get; set; }
```

6.41.5.11 Offset `int Hardware.W65C51.Offset [get], [set]`

Definition at line 27 of file W65C51.cs.
```
00027 { get; set; }
```

6.41.5.12 Overrun `bool Hardware.W65C51.Overrun [get], [set], [private]`

Definition at line 34 of file W65C51.cs.
```
00034 { get; set; }
```

6.41.5.13 ParityEnabled `bool Hardware.W65C51.ParityEnabled [get], [set], [private]`

Definition at line 35 of file W65C51.cs.
```
00035 { get; set; }
```

6.41.5.14 Processor `W65C02 Hardware.W65C51.Processor [get], [set], [private]`

Definition at line 25 of file W65C51.cs.

```
{ get; set; }
```

6.41.5.15 ReceiverFull `bool Hardware.W65C51.ReceiverFull [get], [set], [private]`

Definition at line 36 of file W65C51.cs.

```
{ get; set; }
```

6.41.5.16 RtsControl `byte Hardware.W65C51.RtsControl [get], [set], [private]`

Definition at line 37 of file W65C51.cs.

```
{ get; set; }
```

The documentation for this class was generated from the following file:

- Hardware/W65C51.cs

7 File Documentation

7.1 Emulator/App.xaml.cs File Reference

Classes

- class Emulator.App

  Interaction logic for App.xaml

Namespaces

- namespace Emulator

7.2 App.xaml.cs

Go to the documentation of this file.

```
namespace Emulator
{
/// <summary>
/// Interaction logic for App.xaml
/// </summary>
    public partial class App
    {
    }
}
```

7.3 Emulator/Classes/ExitCodes.cs File Reference

Classes

- class Emulator.ExitCodes

Namespaces

- namespace Emulator

7.4 ExitCodes.cs

Go to the documentation of this file.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Emulator
{
    public class ExitCodes
    {
        public static readonly int NO_ERROR = 0x00;

        public static readonly int USER_ERROR = 0x01;

        public static readonly int NO_BIOS = 0x02;
        public static readonly int LOAD_BIOS_FILE_ERROR = 0x03;
        public static readonly int BIOS_LOADPROGRAM_ERROR = 0x04;
        public static readonly int LOAD_ROM_FILE_ERROR = 0x05;
        public static readonly int ROM_LOADPROGRAM_ERROR = 0x06;
        public static readonly int LOAD_STATE_ERROR = 0x07;
    }
}
```

7.5 Emulator/Classes/FileLocations.cs File Reference

Namespaces

- namespace Emulator

7.6 FileLocations.cs

Go to the documentation of this file.

```
namespace Emulator
{
    internal class FileLocations
    {
#region Fields
        public static string SettingsFile = "./Settings.xml";
        public static string ErrorFile = "./Errors.log";
#if DEBUG
            public static string BiosFile = "../../../bios.bin";
#else
            public static string BiosFile = "./bios.bin";
#endif
#endregion
    }
}
```

7.7 Hardware/Classes/FileLocations.cs File Reference

Namespaces

- namespace Hardware

7.8 FileLocations.cs

Go to the documentation of this file.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Hardware
{
    internal class FileLocations
    {
#region Fields
        public static string ErrorFile = "./Hardware_Library_Errors.log";
        public static string MemoryDump = "./Hardware_Library_Memory_Dump.log";
#endregion
    }
}
```

7.9 Emulator/Classes/SettingsFile.cs File Reference

Classes

- class Emulator.SettingsFile

Namespaces

- namespace Emulator

7.10 SettingsFile.cs

Go to the documentation of this file.

```
using Emulator.Model;
using GalaSoft.MvvmLight.Messaging;

namespace Emulator
{
    public static class SettingsFile
    {
        public static SettingsModel CreateNew()
        {
            // Create new settings file.
            SettingsModel _settings = new SettingsModel
            {
                SettingsVersionMajor = Versioning.SettingsFile.Major,
                SettingsVersionMinor = Versioning.SettingsFile.Minor,
                SettingsVersionBuild = Versioning.SettingsFile.Build,
                SettingsVersionRevision = Versioning.SettingsFile.Revision,
#if DEBUG
                ComPortName = "COM9",
#else
                ComPortName = "COM1",
#endif
            };
            return _settings;
        }
    }
}
```

7.11 Emulator/Classes/Versioning.cs File Reference

Classes

- class Emulator.Versioning
- class Emulator.Versioning.Product
- class Emulator.Versioning.SettingsFile

Namespaces

- namespace Emulator

7.12 Versioning.cs

Go to the documentation of this file.

```
using System.Deployment;
using System.Reflection;
using System;

namespace Emulator
{
    public static class Versioning
    {
        public class Product
        {
            public const int Major = 0;
            public const int Minor = 1;
            public const int Build = 3;
            public const int Revision = 1;
            public const string Title = Name;
            public const string Name = "WolfNet 65C02 WorkBench Computer Emulator";
            public const string Company = "WolfNet Computing";
            public const string Copyright = "Copyright l’ WolfNet Computing 2022";
            public const string VersionString = "0.2.3.1";
            public const string Description = "Emulator for the WolfNet 65C02 WorkBench Computer coded in C# using the .NET Framework";
        }
        public class SettingsFile
        {
            public const byte Major = 1;
            public const byte Minor = 0;
            public const byte Build = 0;
            public const byte Revision = 0;

        }
    }
}
```

7.13 Hardware/Classes/Versioning.cs File Reference

Classes

- class Hardware.Versioning.Product

Namespaces

- namespace Hardware

7.14 Versioning.cs

Go to the documentation of this file.

```
namespace Hardware
{
    internal class Versioning
    {
        public class Product
        {
            public const string Title = Name;
            public const string Name = "WolfNet 65C02 Hardware Library";
            public const string Company = "WolfNet Computing";
            public const string Copyright = "Copyright l’ WolfNet Computing 2022";
            public const string Version = "1.3.0.0";
            public const string Description = "65C02 Hardware Library, coded in C# using the .NET Framework";
        }
    }
}
```

7.15 Emulator/Interfaces/IClosable.cs File Reference

Classes

- interface Emulator.IClosable

Namespaces

- namespace Emulator

7.16 IClosable.cs

Go to the documentation of this file.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Emulator
{
    public interface IClosable
    {
        void Close();
    }
}
```

7.17 Emulator/MainWindow.xaml.cs File Reference

Classes

- class Emulator.MainWindow

  Interaction logic for MainWindow.xaml

Namespaces

- namespace Emulator

7.18 MainWindow.xaml.cs

Go to the documentation of this file.

```
using GalaSoft.MvvmLight.Messaging;
using Emulator.Model;
using Emulator.ViewModel;
using System;
using System.ComponentModel;
using System.Windows;
using Hardware;
using System.IO;
using System.Xml.Serialization;

namespace Emulator
{
/// <summary>
/// Interaction logic for MainWindow.xaml
/// </summary>
    public partial class MainWindow :  Window, IClosable
    {
        public MainWindow()
        {
            InitializeComponent();
            Messenger.Default.Register<NotificationMessage>(this, NotificationMessageReceived);
            Messenger.Default.Register<NotificationMessage<StateFileModel»(this, NotificationMessageReceived);
            Messenger.Default.Register<NotificationMessage<SettingsModel»(this, NotificationMessageReceived);
        }

        private void ToClose(Object sender, EventArgs e)
        {
            Close();
        }

        private void LoadFile(Object sender, EventArgs e)
        {
            Messenger.Default.Send(new NotificationMessage("LoadFile"));
        }

        private void SaveFile(Object sender, EventArgs e)
        {
            Messenger.Default.Send(new NotificationMessage("SaveState"));
        }

        private void CloseFile(Object sender, EventArgs e)
        {
            Messenger.Default.Send(new NotificationMessage("CloseFile"));
        }

        private void NotificationMessageReceived(NotificationMessage notificationMessage)
        {
            if (notificationMessage.Notification == "CloseWindow")
            {
                Close();
            }
        }

        private void NotificationMessageReceived(NotificationMessage<StateFileModel> notificationMessage)
        {
            if (notificationMessage.Notification == "SaveFileWindow")
            {
                var saveFile = new SaveFile { DataContext = new SaveFileViewModel(notificationMessage.Content) };
                saveFile.ShowDialog();
            }
        }

        private void NotificationMessageReceived(NotificationMessage<SettingsModel> notificationMessage)
        {
            if (notificationMessage.Notification == "SettingsWindow")
            {
                var settingsFile = new Settings { DataContext = new SettingsViewModel(notificationMessage.Content) };
                settingsFile.ShowDialog();
            }
        }
    }
}
```

7.19 Emulator/Model/Breakpoint.cs File Reference

Classes

- class Emulator.Model.Breakpoint

  A Representation of a Breakpoint

Namespaces

- namespace Emulator
- namespace Emulator.Model

7.20 Breakpoint.cs

Go to the documentation of this file.

```
using System.Collections.Generic;

namespace Emulator.Model
{
/// <summary>
/// A Representation of a Breakpoint
/// </summary>
    public class Breakpoint
    {
/// <summary>
/// Is the Breakpoint enabled or disabled
/// </summary>
        public bool IsEnabled { get; set; }

/// <summary>
/// The Value of the Breakpoint
/// </summary>
        public string Value { get; set; }

/// <summary>
/// The Type of breakpoint being set
/// </summary>
        public string Type { get; set; }

        public List<string> AllTypes
        {
            get { return BreakpointType.AllTypes; }
        }
    }
}
```

7.21 Emulator/Model/BreakpointType.cs File Reference

Classes

- class Emulator.Model.BreakpointType

  The Type of Breakpoint

Namespaces

- namespace Emulator
- namespace Emulator.Model

7.22 BreakpointType.cs

Go to the documentation of this file.

```
using System.Collections.Generic;

namespace Emulator.Model
{
/// <summary>
/// The Type of Breakpoint
/// </summary>
    public class BreakpointType
    {
/// <summary>
/// A Listing of all of the Current Types
/// </summary>
        public static List<string> AllTypes = new List<string>
            {
                ProgramCounterType,
                NumberOfCycleType
            };

/// <summary>
/// The ProgamCounter Breakpoint Type
/// </summary>
        public const string ProgramCounterType = "Program Counter";

/// <summary>
/// The CycleCount Breakpoint Type
/// </summary>
        public const string NumberOfCycleType = "Number of Cycles";

    }
}
```

7.23 Emulator/Model/MemoryRowModel.cs File Reference

Classes

- class Emulator.Model.MemoryRowModel

  A Model of a Single Page of memory

Namespaces

- namespace Emulator
- namespace Emulator.Model

7.24 MemoryRowModel.cs

Go to the documentation of this file.

```
namespace Emulator.Model
{
/// <summary>
/// A Model of a Single Page of memory
/// </summary>
    public class MemoryRowModel
    {
/// <summary>
/// The offset of this row.  Expressed in hex
/// </summary>
        public string Offset { get; set; }
/// <summary>
/// The memory at the location offset + 00
/// </summary>
        public string Location00 { get; set; }
/// <summary>
/// The memory at the location offset + 01
/// </summary>
        public string Location01 { get; set; }
/// <summary>
/// The memory at the location offset + 02
/// </summary>
        public string Location02 { get; set; }
/// <summary>
/// The memory at the location offset + 03
/// </summary>
        public string Location03 { get; set; }
/// <summary>
/// The memory at the location offset + 04
/// </summary>
        public string Location04 { get; set; }
/// <summary>
/// The memory at the location offset + 05
/// </summary>
        public string Location05 { get; set; }
/// <summary>
/// The memory at the location offset + 06
/// </summary>
        public string Location06 { get; set; }
/// <summary>
/// The memory at the location offset + 07
/// </summary>
        public string Location07 { get; set; }
/// <summary>
/// The memory at the location offset + 08
/// </summary>
        public string Location08 { get; set; }
/// <summary>
/// The memory at the location offset + 09
/// </summary>
        public string Location09 { get; set; }
/// <summary>
/// The memory at the location offset + 0A
/// </summary>
        public string Location0A { get; set; }
/// <summary>
/// The memory at the location offset + 0B
/// </summary>
        public string Location0B { get; set; }
/// <summary>
/// The memory at the location offset + 0C
/// </summary>
        public string Location0C { get; set; }
/// <summary>
/// The memory at the location offset + 0D
/// </summary>
        public string Location0D { get; set; }
/// <summary>
/// The memory at the location offset + 0E
/// </summary>
        public string Location0E { get; set; }
/// <summary>
/// The memory at the location offset + 0F
/// </summary>
        public string Location0F { get; set; }
    }
}
```

7.25 Emulator/Model/OutputLog.cs File Reference

Classes

- class Emulator.Model.OutputLog

  The OutputLog Model. Used by the outputlog grid to show a history of operations performed by the CPU

Namespaces

- namespace Emulator
- namespace Emulator.Model

7.26 OutputLog.cs

Go to the documentation of this file.

```
using System;
using Hardware;

namespace Emulator.Model
{
/// <summary>
/// The OutputLog Model.  Used by the outputlog grid to show a history of operations performed by the
    CPU
/// </summary>
    [Serializable]
    public class OutputLog :  Disassembly
    {
        public OutputLog(Disassembly disassembly)
        {
            DisassemblyOutput = disassembly.DisassemblyOutput;
            HighAddress = disassembly.HighAddress;
            LowAddress = disassembly.LowAddress;
            OpCodeString = disassembly.OpCodeString;
        }

/// <summary>
/// The Program Counter Value
/// </summary>
        public string ProgramCounter { get; set; }
/// <summary>
/// The Current Ope Code
/// </summary>
        public string CurrentOpCode { get; set; }
/// <summary>
/// The X Register
/// </summary>
        public string XRegister { get; set; }
/// <summary>
/// The Y Register
/// </summary>
        public string YRegister { get; set; }
/// <summary>
/// The Accummulator
/// </summary>
        public string Accumulator { get; set; }
/// <summary>
/// The Stack Pointer
/// </summary>
        public string StackPointer { get; set; }
/// <summary>
/// The number of cycles executed since the last load or reset
/// </summary>
        public int NumberOfCycles { get; set; }
    }
}
```

7.27 Emulator/Model/RomFileModel.cs File Reference

Classes

- class Emulator.Model.RomFileModel

  The Model used when Loading a Program.

Namespaces

- namespace Emulator
- namespace Emulator.Model

7.28 RomFileModel.cs

Go to the documentation of this file.
```
namespace Emulator.Model
{
/// <summary>
/// The Model used when Loading a Program.
/// </summary>
    public class RomFileModel
    {
/// <summary>
/// The Program Converted into Hex.
/// </summary>
        public byte[][] Rom { get; set; }

/// <summary>
/// The path of the Program that was loaded.
/// </summary>
        public byte RomBanks { get; set; }

/// <summary>
/// The name of the Program that was loaded.
/// </summary>
        public int RomBankSize { get; set; }

/// <summary>
/// The name of the Program that was loaded.
/// </summary>
        public string RomFileName { get; set; }

/// <summary>
/// The path of the Program that was loaded.
/// </summary>
        public string RomFilePath { get; set; }
    }
}
```

7.29 Emulator/Model/SettingsModel.cs File Reference

Classes

- class Emulator.Model.SettingsModel

    Model that contains the required information needed to save the current settings to disk

Namespaces

- namespace Emulator
- namespace Emulator.Model

7.30 SettingsModel.cs

Go to the documentation of this file.
```
using System;
using System.Xml.Serialization;

namespace Emulator.Model
{
/// <summary>
/// Model that contains the required information needed to save the current settings to disk
/// </summary>
    [Serializable]
    [XmlRootAttribute("SettingsFileModel", Namespace="Emulator.Model", IsNullable = false)]
    public class SettingsModel
    {
/// <summary>
/// The version of the file that is being saved
/// </summary>
        public byte SettingsVersionMajor { get; set; }

/// <summary>
/// The version of the file that is being saved
/// </summary>
        public byte SettingsVersionMinor { get; set; }

/// <summary>
/// The version of the file that is being saved
/// </summary>
        public byte SettingsVersionBuild { get; set; }

/// <summary>
/// The version of the file that is being saved
/// </summary>
        public byte SettingsVersionRevision { get; set; }

/// <summary>
/// The PC port that is being saved
/// </summary>
        public string ComPortName { get; set; }
    }
}
```

7.31 Emulator/Model/StateFileModel.cs File Reference

Classes

- class Emulator.Model.StateFileModel

  Model that contains the required information needed to save the current state of the processor to disk

Namespaces

- namespace Emulator
- namespace Emulator.Model

7.32 StateFileModel.cs

Go to the documentation of this file.

```
using System;
using System.Collections.Generic;

namespace Emulator.Model
{
/// <summary>
/// Model that contains the required information needed to save the current state of the processor to
disk
/// </summary>
    [Serializable]
    public class StateFileModel
    {
/// <summary>
/// The Number of Cycles the Program has Ran so Far
/// </summary>
        public int NumberOfCycles { get; set; }

/// <summary>
/// The output of the program
/// </summary>
        public IList<OutputLog> OutputLog { get; set; }

/// <summary>
/// The Processor Object that is being saved
/// </summary>
        public Hardware.W65C02 W65C02 { get; set; }

/// <summary>
/// The first VIA Object that is being saved
/// </summary>
        public Hardware.W65C22 W65C22 { get; set; }

/// <summary>
/// The second VIA Object that is being saved
/// </summary>
        public Hardware.W65C22 MM65SIB { get; set; }

/// <summary>
/// The ACIA Object that is being saved
/// </summary>
        public Hardware.W65C51 W65C51 { get; set; }

/// <summary>
/// The Shared ROM Object that is being saved
/// </summary>
        public Hardware.AT28CXX AT28C010 { get; set; }

/// <summary>
/// The Banked ROM Object that is being saved
/// </summary>
        public Hardware.AT28CXX AT28C64 { get; set; }
    }
}
```

7.33 Emulator/MultiThreadedCollection.cs File Reference

Classes

- class Emulator.MultiThreadedObservableCollection< T >

  A MultiThreaedObservableCollection. This allows multiple threads to access the same observable collection in a safe manner.

Namespaces

- namespace Emulator

7.34 MultiThreadedCollection.cs

Go to the documentation of this file.

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Collections.Specialized;
using System.Windows.Threading;

namespace Emulator
{
/// <summary>
/// A MultiThreaedObservableCollection.
/// This allows multiple threads to access the same observable collection in a safe manner.
/// </summary>
/// <typeparam name="T"></typeparam>
    public class MultiThreadedObservableCollection<T> :  ObservableCollection<T>
    {
/// <summary>
/// Instantiates a new instance of the MultiThreadedObservableCollection
/// </summary>
        public MultiThreadedObservableCollection()
        {

        }

/// <summary>
///  Instantiates a new instance of the MultiThreadedObservableCollection
/// </summary>
/// <param name="collection">The initial collection to be loaded</param>
        public MultiThreadedObservableCollection(IEnumerable<T> collection)
            :  base(collection)
        {

        }

/// <summary>
///  Instantiates a new instance of the MultiThreadedObservableCollection
/// </summary>
/// <param name="list">The initial list to be loaded</param>
        public MultiThreadedObservableCollection(List<T> list)
            :  base(list)
        {

        }

/// <summary>
/// The NotifyCollectionChangedEventHandler, Sends a notification anytime the collection has been
modified.
/// </summary>
        public override event NotifyCollectionChangedEventHandler CollectionChanged;


/// <summary>
/// The NotifyCollectionChangedEventHandler, Notifies the listeners in a thread safe manner
/// </summary>
        protected override void OnCollectionChanged(NotifyCollectionChangedEventArgs e)
        {
            var collectionChanged = CollectionChanged;
            if (collectionChanged != null)
                foreach (NotifyCollectionChangedEventHandler nh in
collectionChanged.GetInvocationList())
                {
                    var dispObj = nh.Target as DispatcherObject;
                    if (dispObj != null)
                    {
                        var dispatcher = dispObj.Dispatcher;
                        if (dispatcher != null && !dispatcher.CheckAccess())
                        {
                            var nh1 = nh;
                            dispatcher.BeginInvoke(
                                (Action)(() => nh1.Invoke(this,
                                    new
NotifyCollectionChangedEventArgs(NotifyCollectionChangedAction.Reset))),
                                DispatcherPriority.DataBind);
                            continue;
                        }
                    }
                    nh.Invoke(this, e);
            }
        }
    }
}
```

7.35 Emulator/obj/x86/Debug/.NETFramework,Version=v4.8.AssemblyAttributes.cs File Reference

7.36 .NETFramework,Version=v4.8.AssemblyAttributes.cs

Go to the documentation of this file.

```
// <autogenerated />
using System;
using System.Reflection;
[assembly:  global::System.Runtime.Versioning.TargetFrameworkAttribute(".NETFramework,Version=v4.8",
FrameworkDisplayName = ".NET Framework 4.8")]
```

7.37 Hardware/obj/Debug/.NETFramework,Version=v4.8.AssemblyAttributes.cs File Reference

7.38 .NETFramework,Version=v4.8.AssemblyAttributes.cs

Go to the documentation of this file.

```
// <autogenerated />
using System;
using System.Reflection;
[assembly:  global::System.Runtime.Versioning.TargetFrameworkAttribute(".NETFramework,Version=v4.8",
FrameworkDisplayName = ".NET Framework 4.8")]
```

7.39 Emulator/obj/x86/Debug/App.g.cs File Reference

Classes

- class XamlGeneratedNamespace.GeneratedApplication

  GeneratedApplication

Namespaces

- namespace XamlGeneratedNamespace

7.40 App.g.cs

Go to the documentation of this file.

```
#pragma checksum "..\..\..\App.xaml" "{8829d00f-11b8-4213-878b-770e8597ac16}"
"3C3B83350F313F767CDD9CA458D577D426BB4EF0F6F94CE9866749BCB08F1D0F"
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:4.0.30319.42000
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

using Emulator.ViewModel;
using System;
using System.Diagnostics;
using System.Windows;
using System.Windows.Automation;
using System.Windows.Controls;
using System.Windows.Controls.Primitives;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Markup;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Media.Effects;
using System.Windows.Media.Imaging;
using System.Windows.Media.Media3D;
using System.Windows.Media.TextFormatting;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Windows.Shell;


namespace XamlGeneratedNamespace {


/// <summary>
/// GeneratedApplication
/// </summary>
    public partial class GeneratedApplication :  System.Windows.Application {

        private bool _contentLoaded;

/// <summary>
/// InitializeComponent
/// </summary>
        [System.Diagnostics.DebuggerNonUserCodeAttribute()]
        [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
        public void InitializeComponent() {
            if (_contentLoaded) {
                return;
            }
            _contentLoaded = true;

#line 2 "..\..\..\App.xaml"
            this.StartupUri = new System.Uri("MainWindow.xaml", System.UriKind.Relative);

#line default
#line hidden
```

```
            System.Uri resourceLocater = new System.Uri("/Emulator;component/app.xaml",
System.UriKind.Relative);

#line 1 "..\..\..\App.xaml"
            System.Windows.Application.LoadComponent(this, resourceLocater);

#line default
#line hidden
        }

/// <summary>
/// Application Entry Point.
/// </summary>
        [System.STAThreadAttribute()]
        [System.Diagnostics.DebuggerNonUserCodeAttribute()]
        [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
        public static void Main() {
            SplashScreen splashScreen = new SplashScreen("splashscreen.png");
            splashScreen.Show(true);
            XamlGeneratedNamespace.GeneratedApplication app = new
XamlGeneratedNamespace.GeneratedApplication();
            app.InitializeComponent();
            app.Run();
        }
    }
}

```

7.41 Emulator/obj/x86/Debug/App.g.i.cs File Reference

Classes

- class XamlGeneratedNamespace.GeneratedApplication

  GeneratedApplication

Namespaces

- namespace XamlGeneratedNamespace

7.42 App.g.i.cs

Go to the documentation of this file.

```
#pragma checksum "..\..\..\App.xaml" "{8829d00f-11b8-4213-878b-770e8597ac16}"
"3C3B83350F313F767CDD9CA458D577D426BB4EF0F6F94CE9866749BCB08F1D0F"
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:4.0.30319.42000
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

using Emulator.ViewModel;
using System;
using System.Diagnostics;
using System.Windows;
using System.Windows.Automation;
using System.Windows.Controls;
using System.Windows.Controls.Primitives;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Markup;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Media.Effects;
using System.Windows.Media.Imaging;
using System.Windows.Media.Media3D;
using System.Windows.Media.TextFormatting;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Windows.Shell;


namespace XamlGeneratedNamespace {


/// <summary>
/// GeneratedApplication
/// </summary>
    public partial class GeneratedApplication :  System.Windows.Application {

        private bool _contentLoaded;

/// <summary>
/// InitializeComponent
/// </summary>
        [System.Diagnostics.DebuggerNonUserCodeAttribute()]
        [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
        public void InitializeComponent() {
            if (_contentLoaded) {
                return;
            }
            _contentLoaded = true;

#line 2 "..\..\..\App.xaml"
            this.StartupUri = new System.Uri("MainWindow.xaml", System.UriKind.Relative);

#line default
#line hidden
            System.Uri resourceLocater = new System.Uri("/Emulator;component/app.xaml",
System.UriKind.Relative);

#line 1 "..\..\..\App.xaml"
            System.Windows.Application.LoadComponent(this, resourceLocater);

#line default
#line hidden
        }

/// <summary>
/// Application Entry Point.
/// </summary>
        [System.STAThreadAttribute()]
        [System.Diagnostics.DebuggerNonUserCodeAttribute()]
        [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
        public static void Main() {
            SplashScreen splashScreen = new SplashScreen("splashscreen.png");
            splashScreen.Show(true);
            XamlGeneratedNamespace.GeneratedApplication app = new
XamlGeneratedNamespace.GeneratedApplication();
            app.InitializeComponent();
            app.Run();
        }
    }
}

```

7.43 Emulator/obj/x86/Debug/Emulator_Content.g.cs File Reference

7.44 Emulator_Content.g.cs

Go to the documentation of this file.

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:4.0.30319.42000
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

[assembly:  System.Windows.Resources.AssemblyAssociatedContentFileAttribute("nlog.config")]


```

7.45 Emulator/obj/x86/Debug/Emulator_Content.g.i.cs File Reference

7.46 Emulator_Content.g.i.cs

Go to the documentation of this file.

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:4.0.30319.42000
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

[assembly:  System.Windows.Resources.AssemblyAssociatedContentFileAttribute("nlog.config")]


```

7.47 Emulator/obj/x86/Debug/GeneratedInternalTypeHelper.g.cs File Reference

Classes

- class XamlGeneratedNamespace.GeneratedInternalTypeHelper

  GeneratedInternalTypeHelper

Namespaces

- namespace XamlGeneratedNamespace

7.48 GeneratedInternalTypeHelper.g.cs

Go to the documentation of this file.

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:4.0.30319.42000
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

namespace XamlGeneratedNamespace {


/// <summary>
/// GeneratedInternalTypeHelper
/// </summary>
    [System.Diagnostics.DebuggerNonUserCodeAttribute()]
    [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
    [System.ComponentModel.EditorBrowsableAttribute(System.ComponentModel.EditorBrowsableState.Never)]
    public sealed class GeneratedInternalTypeHelper :  System.Windows.Markup.InternalTypeHelper {

/// <summary>
/// CreateInstance
/// </summary>
        protected override object CreateInstance(System.Type type, System.Globalization.CultureInfo
culture) {
            return System.Activator.CreateInstance(type, ((System.Reflection.BindingFlags.Public |
System.Reflection.BindingFlags.NonPublic)
                            | (System.Reflection.BindingFlags.Instance |
System.Reflection.BindingFlags.CreateInstance)), null, null, culture);
        }

/// <summary>
/// GetPropertyValue
/// </summary>
```

```
        protected override object GetPropertyValue(System.Reflection.PropertyInfo propertyInfo, object target, System.Globalization.CultureInfo culture) {
            return propertyInfo.GetValue(target, System.Reflection.BindingFlags.Default, null, null, culture);
        }

/// <summary>
/// SetPropertyValue
/// </summary>
        protected override void SetPropertyValue(System.Reflection.PropertyInfo propertyInfo, object target, object value, System.Globalization.CultureInfo culture) {
            propertyInfo.SetValue(target, value, System.Reflection.BindingFlags.Default, null, null, culture);
        }

/// <summary>
/// CreateDelegate
/// </summary>
        protected override System.Delegate CreateDelegate(System.Type delegateType, object target, string handler) {
            return ((System.Delegate)(target.GetType().InvokeMember("_CreateDelegate", (System.Reflection.BindingFlags.InvokeMethod
                            | (System.Reflection.BindingFlags.NonPublic | System.Reflection.BindingFlags.Instance)), null, target, new object[] {
                        delegateType,
                        handler}, null)));
        }

/// <summary>
/// AddEventHandler
/// </summary>
        protected override void AddEventHandler(System.Reflection.EventInfo eventInfo, object target, System.Delegate handler) {
            eventInfo.AddEventHandler(target, handler);
        }
    }
}

```

7.49 Emulator/obj/x86/Debug/GeneratedInternalTypeHelper.g.i.cs File Reference

Classes

- class XamlGeneratedNamespace.GeneratedInternalTypeHelper

  GeneratedInternalTypeHelper

Namespaces

- namespace XamlGeneratedNamespace

7.50 GeneratedInternalTypeHelper.g.i.cs

Go to the documentation of this file.

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:4.0.30319.42000
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

namespace XamlGeneratedNamespace {


/// <summary>
/// GeneratedInternalTypeHelper
/// </summary>
    [System.Diagnostics.DebuggerNonUserCodeAttribute()]
    [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
    [System.ComponentModel.EditorBrowsableAttribute(System.ComponentModel.EditorBrowsableState.Never)]
    public sealed class GeneratedInternalTypeHelper :  System.Windows.Markup.InternalTypeHelper {

/// <summary>
/// CreateInstance
/// </summary>
        protected override object CreateInstance(System.Type type, System.Globalization.CultureInfo
culture) {
            return System.Activator.CreateInstance(type, ((System.Reflection.BindingFlags.Public |
System.Reflection.BindingFlags.NonPublic)
                            | (System.Reflection.BindingFlags.Instance |
System.Reflection.BindingFlags.CreateInstance)), null, null, culture);
        }

/// <summary>
/// GetPropertyValue
/// </summary>
        protected override object GetPropertyValue(System.Reflection.PropertyInfo propertyInfo, object
target, System.Globalization.CultureInfo culture) {
            return propertyInfo.GetValue(target, System.Reflection.BindingFlags.Default, null, null,
culture);
        }

/// <summary>
/// SetPropertyValue
/// </summary>
        protected override void SetPropertyValue(System.Reflection.PropertyInfo propertyInfo, object
target, object value, System.Globalization.CultureInfo culture) {
            propertyInfo.SetValue(target, value, System.Reflection.BindingFlags.Default, null, null,
culture);
        }

/// <summary>
/// CreateDelegate
/// </summary>
        protected override System.Delegate CreateDelegate(System.Type delegateType, object target,
string handler) {
            return ((System.Delegate)(target.GetType().InvokeMember("_CreateDelegate",
(System.Reflection.BindingFlags.InvokeMethod
                            | (System.Reflection.BindingFlags.NonPublic |
System.Reflection.BindingFlags.Instance)), null, target, new object[] {
                        delegateType,
                        handler}, null)));
        }

/// <summary>
/// AddEventHandler
/// </summary>
        protected override void AddEventHandler(System.Reflection.EventInfo eventInfo, object target,
System.Delegate handler) {
            eventInfo.AddEventHandler(target, handler);
        }
    }
}

```

7.51 Emulator/obj/x86/Debug/MainWindow.g.cs File Reference

Classes

- class Emulator.MainWindow

  Interaction logic for MainWindow.xaml

Namespaces

- namespace Emulator

7.52 MainWindow.g.cs

Go to the documentation of this file.

```
#pragma checksum "..\..\..\MainWindow.xaml" "{8829d00f-11b8-4213-878b-770e8597ac16}"
"661FDF568BC60BE24BAB53613C08D1DB3A4A19717F25E8058F980E67E6302D8D"
//------------------------------------------------------------------------------
```

```
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:4.0.30319.42000
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

using System;
using System.Diagnostics;
using System.Windows;
using System.Windows.Automation;
using System.Windows.Controls;
using System.Windows.Controls.Primitives;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Markup;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Media.Effects;
using System.Windows.Media.Imaging;
using System.Windows.Media.Media3D;
using System.Windows.Media.TextFormatting;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Windows.Shell;


namespace Emulator {


/// <summary>
/// MainWindow
/// </summary>
    public partial class MainWindow :  System.Windows.Window,
System.Windows.Markup.IComponentConnector {


#line 2 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal Emulator.MainWindow EmulatorWindow;

#line default
#line hidden


#line 89 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.DataGrid OutputLog;

#line default
#line hidden


#line 106 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.Button Run;

#line default
#line hidden


#line 107 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.Button Step;

#line default
#line hidden


#line 108 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.Button Reset;

#line default
#line hidden


#line 110 "..\..\..\MainWindow.xaml"
```

```
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock RomFileNameText;

#line default
#line hidden


#line 111 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock ComPortNameText;

#line default
#line hidden


#line 112 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.DataGrid Breakpoints;

#line default
#line hidden


#line 137 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBox YRegister;

#line default
#line hidden


#line 138 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBox XRegister;

#line default
#line hidden


#line 139 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBox Accumulator;

#line default
#line hidden


#line 140 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBox StackPointer;

#line default
#line hidden


#line 141 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBox ProgramCounter;

#line default
#line hidden


#line 142 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBox Dissambly;

#line default
#line hidden


#line 143 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBox CycleCount;

#line default
#line hidden
```

```


#line 144 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock XRegisterText;

#line default
#line hidden


#line 145 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock YRegisterText;

#line default
#line hidden


#line 146 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock StackPointerRegisterText;

#line default
#line hidden


#line 147 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock AText;

#line default
#line hidden


#line 148 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock CurrentInstructionText;

#line default
#line hidden


#line 149 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock ProgramCounterText;

#line default
#line hidden


#line 150 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock CycleCountText;

#line default
#line hidden


#line 151 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.CheckBox CarryFlag;

#line default
#line hidden


#line 152 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock CarryFlagText;

#line default
#line hidden


#line 153 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.CheckBox ZeroFlag;
```

```

#line default
#line hidden


#line 154 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock ZeroFlagText;

#line default
#line hidden


#line 155 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.CheckBox InterrupFlag;

#line default
#line hidden


#line 156 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock InterruptFlagText;

#line default
#line hidden


#line 157 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.CheckBox BcdFlag;

#line default
#line hidden


#line 158 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock BcdFlagText;

#line default
#line hidden


#line 159 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.CheckBox BreakFlag;

#line default
#line hidden


#line 160 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock BreakFlagText;

#line default
#line hidden


#line 161 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.CheckBox OverflowFlag;

#line default
#line hidden


#line 162 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock OverflowFlagText;

#line default
#line hidden


#line 163 "..\..\..\MainWindow.xaml"
```

```
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.CheckBox NegativeFlag;

#line default
#line hidden


#line 164 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock NegativeFlagText;

#line default
#line hidden


#line 165 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.Slider CpuSpeed;

#line default
#line hidden


#line 166 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock SpeedText;

#line default
#line hidden

        private bool _contentLoaded;

/// <summary>
/// InitializeComponent
/// </summary>
        [System.Diagnostics.DebuggerNonUserCodeAttribute()]
        [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
        public void InitializeComponent() {
            if (_contentLoaded) {
                return;
            }
            _contentLoaded = true;
            System.Uri resourceLocater = new System.Uri("/Emulator;component/mainwindow.xaml", System.UriKind.Relative);

#line 1 "..\..\..\MainWindow.xaml"
            System.Windows.Application.LoadComponent(this, resourceLocater);

#line default
#line hidden
        }

        [System.Diagnostics.DebuggerNonUserCodeAttribute()]
        [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
        [System.ComponentModel.EditorBrowsableAttribute(System.ComponentModel.EditorBrowsableState.Never)]
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Design", "CA1033:InterfaceMethodsShouldBeCallableByChildTypes")]
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Maintainability", "CA1502:AvoidExcessiveComplexity")]
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1800:DoNotCastUnnecessarily")]
        void System.Windows.Markup.IComponentConnector.Connect(int connectionId, object target) {
            switch (connectionId)
            {
            case 1:
            this.EmulatorWindow = ((Emulator.MainWindow)(target));
            return;
            case 2:

#line 72 "..\..\..\MainWindow.xaml"
            ((System.Windows.Controls.MenuItem)(target)).Click += new System.Windows.RoutedEventHandler(this.LoadFile);

#line default
#line hidden
            return;
            case 3:

#line 73 "..\..\..\MainWindow.xaml"
            ((System.Windows.Controls.MenuItem)(target)).Click += new System.Windows.RoutedEventHandler(this.SaveFile);

```

```
#line default
#line hidden
            return;
            case 4:

#line 74 "..\..\..\MainWindow.xaml"
            ((System.Windows.Controls.MenuItem)(target)).Click += new
System.Windows.RoutedEventHandler(this.CloseFile);

#line default
#line hidden
            return;
            case 5:

#line 76 "..\..\..\MainWindow.xaml"
            ((System.Windows.Controls.MenuItem)(target)).Click += new
System.Windows.RoutedEventHandler(this.ToClose);

#line default
#line hidden
            return;
            case 6:
            this.OutputLog = ((System.Windows.Controls.DataGrid)(target));
            return;
            case 7:
            this.Run = ((System.Windows.Controls.Button)(target));
            return;
            case 8:
            this.Step = ((System.Windows.Controls.Button)(target));
            return;
            case 9:
            this.Reset = ((System.Windows.Controls.Button)(target));
            return;
            case 10:
            this.RomFileNameText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 11:
            this.ComPortNameText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 12:
            this.Breakpoints = ((System.Windows.Controls.DataGrid)(target));
            return;
            case 13:
            this.YRegister = ((System.Windows.Controls.TextBox)(target));
            return;
            case 14:
            this.XRegister = ((System.Windows.Controls.TextBox)(target));
            return;
            case 15:
            this.Accumulator = ((System.Windows.Controls.TextBox)(target));
            return;
            case 16:
            this.StackPointer = ((System.Windows.Controls.TextBox)(target));
            return;
            case 17:
            this.ProgramCounter = ((System.Windows.Controls.TextBox)(target));
            return;
            case 18:
            this.Dissambly = ((System.Windows.Controls.TextBox)(target));
            return;
            case 19:
            this.CycleCount = ((System.Windows.Controls.TextBox)(target));
            return;
            case 20:
            this.XRegisterText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 21:
            this.YRegisterText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 22:
            this.StackPointerRegisterText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 23:
            this.AText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 24:
            this.CurrentInstructionText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 25:
            this.ProgramCounterText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 26:
            this.CycleCountText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 27:
            this.CarryFlag = ((System.Windows.Controls.CheckBox)(target));
            return;
```

```
                case 28:
                this.CarryFlagText = ((System.Windows.Controls.TextBlock)(target));
                return;
                case 29:
                this.ZeroFlag = ((System.Windows.Controls.CheckBox)(target));
                return;
                case 30:
                this.ZeroFlagText = ((System.Windows.Controls.TextBlock)(target));
                return;
                case 31:
                this.InterrupFlag = ((System.Windows.Controls.CheckBox)(target));
                return;
                case 32:
                this.InterruptFlagText = ((System.Windows.Controls.TextBlock)(target));
                return;
                case 33:
                this.BcdFlag = ((System.Windows.Controls.CheckBox)(target));
                return;
                case 34:
                this.BcdFlagText = ((System.Windows.Controls.TextBlock)(target));
                return;
                case 35:
                this.BreakFlag = ((System.Windows.Controls.CheckBox)(target));
                return;
                case 36:
                this.BreakFlagText = ((System.Windows.Controls.TextBlock)(target));
                return;
                case 37:
                this.OverflowFlag = ((System.Windows.Controls.CheckBox)(target));
                return;
                case 38:
                this.OverflowFlagText = ((System.Windows.Controls.TextBlock)(target));
                return;
                case 39:
                this.NegativeFlag = ((System.Windows.Controls.CheckBox)(target));
                return;
                case 40:
                this.NegativeFlagText = ((System.Windows.Controls.TextBlock)(target));
                return;
                case 41:
                this.CpuSpeed = ((System.Windows.Controls.Slider)(target));
                return;
                case 42:
                this.SpeedText = ((System.Windows.Controls.TextBlock)(target));
                return;
                }
                this._contentLoaded = true;
            }
        }
}

```

7.53 Emulator/obj/x86/Debug/MainWindow.g.i.cs File Reference

Classes

- class Emulator.MainWindow

  Interaction logic for MainWindow.xaml

Namespaces

- namespace Emulator

7.54 MainWindow.g.i.cs

Go to the documentation of this file.

```
#pragma checksum "..\..\..\MainWindow.xaml" "{8829d00f-11b8-4213-878b-770e8597ac16}"
"661FDF568BC60BE24BAB53613C08D1DB3A4A19717F25E8058F980E67E6302D8D"
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:4.0.30319.42000
```

```
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

using System;
using System.Diagnostics;
using System.Windows;
using System.Windows.Automation;
using System.Windows.Controls;
using System.Windows.Controls.Primitives;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Markup;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Media.Effects;
using System.Windows.Media.Imaging;
using System.Windows.Media.Media3D;
using System.Windows.Media.TextFormatting;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Windows.Shell;


namespace Emulator {


/// <summary>
/// MainWindow
/// </summary>
    public partial class MainWindow :  System.Windows.Window,
System.Windows.Markup.IComponentConnector {


#line 2 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal Emulator.MainWindow EmulatorWindow;

#line default
#line hidden


#line 89 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.DataGrid OutputLog;

#line default
#line hidden


#line 106 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.Button Run;

#line default
#line hidden


#line 107 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.Button Step;

#line default
#line hidden


#line 108 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.Button Reset;

#line default
#line hidden


#line 110 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock RomFileNameText;
```

```

#line default
#line hidden


#line 111 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock ComPortNameText;

#line default
#line hidden


#line 112 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.DataGrid Breakpoints;

#line default
#line hidden


#line 137 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBox YRegister;

#line default
#line hidden


#line 138 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBox XRegister;

#line default
#line hidden


#line 139 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBox Accumulator;

#line default
#line hidden


#line 140 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBox StackPointer;

#line default
#line hidden


#line 141 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBox ProgramCounter;

#line default
#line hidden


#line 142 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBox Dissambly;

#line default
#line hidden


#line 143 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBox CycleCount;

#line default
#line hidden


#line 144 "..\..\..\MainWindow.xaml"
```

```
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock XRegisterText;

#line default
#line hidden


#line 145 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock YRegisterText;

#line default
#line hidden


#line 146 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock StackPointerRegisterText;

#line default
#line hidden


#line 147 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock AText;

#line default
#line hidden


#line 148 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock CurrentInstructionText;

#line default
#line hidden


#line 149 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock ProgramCounterText;

#line default
#line hidden


#line 150 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock CycleCountText;

#line default
#line hidden


#line 151 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.CheckBox CarryFlag;

#line default
#line hidden


#line 152 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock CarryFlagText;

#line default
#line hidden


#line 153 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.CheckBox ZeroFlag;

#line default
#line hidden
```

```


#line 154 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock ZeroFlagText;

#line default
#line hidden


#line 155 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.CheckBox InterrupFlag;

#line default
#line hidden


#line 156 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock InterruptFlagText;

#line default
#line hidden


#line 157 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.CheckBox BcdFlag;

#line default
#line hidden


#line 158 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock BcdFlagText;

#line default
#line hidden


#line 159 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.CheckBox BreakFlag;

#line default
#line hidden


#line 160 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock BreakFlagText;

#line default
#line hidden


#line 161 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.CheckBox OverflowFlag;

#line default
#line hidden


#line 162 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock OverflowFlagText;

#line default
#line hidden


#line 163 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.CheckBox NegativeFlag;
```

```

#line default
#line hidden


#line 164 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock NegativeFlagText;

#line default
#line hidden


#line 165 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.Slider CpuSpeed;

#line default
#line hidden


#line 166 "..\..\..\MainWindow.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock SpeedText;

#line default
#line hidden

        private bool _contentLoaded;

/// <summary>
/// InitializeComponent
/// </summary>
        [System.Diagnostics.DebuggerNonUserCodeAttribute()]
        [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
        public void InitializeComponent() {
            if (_contentLoaded) {
                return;
            }
            _contentLoaded = true;
            System.Uri resourceLocater = new System.Uri("/Emulator;component/mainwindow.xaml",
System.UriKind.Relative);

#line 1 "..\..\..\MainWindow.xaml"
            System.Windows.Application.LoadComponent(this, resourceLocater);

#line default
#line hidden
        }

        [System.Diagnostics.DebuggerNonUserCodeAttribute()]
        [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]

[System.ComponentModel.EditorBrowsableAttribute(System.ComponentModel.EditorBrowsableState.Never)]
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Design",
"CA1033:InterfaceMethodsShouldBeCallableByChildTypes")]
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Maintainability",
"CA1502:AvoidExcessiveComplexity")]
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1800:DoNotCastUnnecessarily")]
        void System.Windows.Markup.IComponentConnector.Connect(int connectionId, object target) {
            switch (connectionId)
            {
            case 1:
            this.EmulatorWindow = ((Emulator.MainWindow)(target));
            return;
            case 2:

#line 72 "..\..\..\MainWindow.xaml"
            ((System.Windows.Controls.MenuItem)(target)).Click += new
System.Windows.RoutedEventHandler(this.LoadFile);

#line default
#line hidden
            return;
            case 3:

#line 73 "..\..\..\MainWindow.xaml"
            ((System.Windows.Controls.MenuItem)(target)).Click += new
System.Windows.RoutedEventHandler(this.SaveFile);

#line default
#line hidden
            return;
            case 4:

#line 74 "..\..\..\MainWindow.xaml"
            ((System.Windows.Controls.MenuItem)(target)).Click += new
System.Windows.RoutedEventHandler(this.CloseFile);

#line default
#line hidden
            return;
            case 5:

#line 76 "..\..\..\MainWindow.xaml"
            ((System.Windows.Controls.MenuItem)(target)).Click += new
System.Windows.RoutedEventHandler(this.ToClose);

#line default
#line hidden
            return;
            case 6:
            this.OutputLog = ((System.Windows.Controls.DataGrid)(target));
            return;
            case 7:
            this.Run = ((System.Windows.Controls.Button)(target));
            return;
            case 8:
            this.Step = ((System.Windows.Controls.Button)(target));
            return;
            case 9:
            this.Reset = ((System.Windows.Controls.Button)(target));
            return;
            case 10:
            this.RomFileNameText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 11:
            this.ComPortNameText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 12:
            this.Breakpoints = ((System.Windows.Controls.DataGrid)(target));
            return;
            case 13:
            this.YRegister = ((System.Windows.Controls.TextBox)(target));
            return;
            case 14:
            this.XRegister = ((System.Windows.Controls.TextBox)(target));
            return;
            case 15:
            this.Accumulator = ((System.Windows.Controls.TextBox)(target));
            return;
            case 16:
            this.StackPointer = ((System.Windows.Controls.TextBox)(target));
            return;
            case 17:
            this.ProgramCounter = ((System.Windows.Controls.TextBox)(target));
            return;
            case 18:
            this.Dissambly = ((System.Windows.Controls.TextBox)(target));
            return;
            case 19:
            this.CycleCount = ((System.Windows.Controls.TextBox)(target));
            return;
            case 20:
            this.XRegisterText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 21:
            this.YRegisterText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 22:
            this.StackPointerRegisterText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 23:
            this.AText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 24:
            this.CurrentInstructionText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 25:
            this.ProgramCounterText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 26:
            this.CycleCountText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 27:
            this.CarryFlag = ((System.Windows.Controls.CheckBox)(target));
            return;
            case 28:
            this.CarryFlagText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 29:
            this.ZeroFlag = ((System.Windows.Controls.CheckBox)(target));
            return;
            case 30:
            this.ZeroFlagText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 31:
            this.InterrupFlag = ((System.Windows.Controls.CheckBox)(target));
            return;
            case 32:
            this.InterruptFlagText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 33:
            this.BcdFlag = ((System.Windows.Controls.CheckBox)(target));
            return;
            case 34:
            this.BcdFlagText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 35:
            this.BreakFlag = ((System.Windows.Controls.CheckBox)(target));
            return;
            case 36:
            this.BreakFlagText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 37:
            this.OverflowFlag = ((System.Windows.Controls.CheckBox)(target));
            return;
            case 38:
            this.OverflowFlagText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 39:
            this.NegativeFlag = ((System.Windows.Controls.CheckBox)(target));
            return;
            case 40:
            this.NegativeFlagText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 41:
            this.CpuSpeed = ((System.Windows.Controls.Slider)(target));
            return;
            case 42:
            this.SpeedText = ((System.Windows.Controls.TextBlock)(target));
            return;
            }
            this._contentLoaded = true;
        }
    }
}

```

7.55 Emulator/obj/x86/Debug/SaveFile.g.cs File Reference

Classes

- class Emulator.SaveFile

  SaveFile

Namespaces

- namespace Emulator

7.56 SaveFile.g.cs

Go to the documentation of this file.

```
#pragma checksum "..\..\..\SaveFile.xaml" "{8829d00f-11b8-4213-878b-770e8597ac16}"
"34689CE75633CB3BE5E4FDF3C6E7ECDD6274F88E3F05662C41A2D31C677175A9"
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:4.0.30319.42000
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

using System;
using System.Diagnostics;
using System.Windows;
using System.Windows.Automation;
using System.Windows.Controls;
using System.Windows.Controls.Primitives;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Markup;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Media.Effects;
using System.Windows.Media.Imaging;
using System.Windows.Media.Media3D;
using System.Windows.Media.TextFormatting;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Windows.Shell;


namespace Emulator {


/// <summary>
/// SaveFile
/// </summary>
    public partial class SaveFile :  System.Windows.Window, System.Windows.Markup.IComponentConnector
{


#line 7 "..\..\..\SaveFile.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.Button SelectFile;

#line default
#line hidden


#line 8 "..\..\..\SaveFile.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBox FilePath;

#line default
#line hidden


#line 9 "..\..\..\SaveFile.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock PathText;

#line default
#line hidden


#line 10 "..\..\..\SaveFile.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.Button CancelButton;

#line default
#line hidden


#line 11 "..\..\..\SaveFile.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.Button LoadButton;

#line default
#line hidden

        private bool _contentLoaded;

/// <summary>
/// InitializeComponent
/// </summary>
        [System.Diagnostics.DebuggerNonUserCodeAttribute()]
        [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
        public void InitializeComponent() {
            if (_contentLoaded) {
                return;
            }
            _contentLoaded = true;
            System.Uri resourceLocater = new System.Uri("/Emulator;component/savefile.xaml",
System.UriKind.Relative);

#line 1 "..\..\..\SaveFile.xaml"
            System.Windows.Application.LoadComponent(this, resourceLocater);

#line default
#line hidden
        }

        [System.Diagnostics.DebuggerNonUserCodeAttribute()]
        [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]

[System.ComponentModel.EditorBrowsableAttribute(System.ComponentModel.EditorBrowsableState.Never)]
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Design",
"CA1033:InterfaceMethodsShouldBeCallableByChildTypes")]
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Maintainability",
"CA1502:AvoidExcessiveComplexity")]
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1800:DoNotCastUnnecessarily")]
        void System.Windows.Markup.IComponentConnector.Connect(int connectionId, object target) {
            switch (connectionId)
            {
            case 1:
            this.SelectFile = ((System.Windows.Controls.Button)(target));
            return;
            case 2:
            this.FilePath = ((System.Windows.Controls.TextBox)(target));
            return;
            case 3:
            this.PathText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 4:
            this.CancelButton = ((System.Windows.Controls.Button)(target));
            return;
            case 5:
            this.LoadButton = ((System.Windows.Controls.Button)(target));
            return;
            }
            this._contentLoaded = true;
        }
    }
}

```

7.57 Emulator/obj/x86/Debug/SaveFile.g.i.cs File Reference

Classes

- class Emulator.SaveFile

  SaveFile

Namespaces

- namespace Emulator

7.58 SaveFile.g.i.cs

Go to the documentation of this file.

```
#pragma checksum "..\..\..\SaveFile.xaml" "{8829d00f-11b8-4213-878b-770e8597ac16}"
"34689CE75633CB3BE5E4FDF3C6E7ECDD6274F88E3F05662C41A2D31C677175A9"
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:4.0.30319.42000
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

using System;
using System.Diagnostics;
using System.Windows;
using System.Windows.Automation;
using System.Windows.Controls;
using System.Windows.Controls.Primitives;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Markup;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Media.Effects;
using System.Windows.Media.Imaging;
using System.Windows.Media.Media3D;
using System.Windows.Media.TextFormatting;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Windows.Shell;


namespace Emulator {


/// <summary>
/// SaveFile
/// </summary>
    public partial class SaveFile :  System.Windows.Window, System.Windows.Markup.IComponentConnector {


#line 7 "..\..\..\SaveFile.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.Button SelectFile;

#line default
#line hidden


#line 8 "..\..\..\SaveFile.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBox FilePath;

#line default
#line hidden


#line 9 "..\..\..\SaveFile.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock PathText;

#line default
#line hidden


#line 10 "..\..\..\SaveFile.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.Button CancelButton;

#line default
#line hidden


#line 11 "..\..\..\SaveFile.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.Button LoadButton;

#line default
#line hidden

        private bool _contentLoaded;

/// <summary>
/// InitializeComponent
/// </summary>
        [System.Diagnostics.DebuggerNonUserCodeAttribute()]
        [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
        public void InitializeComponent() {
            if (_contentLoaded) {
                return;
            }
            _contentLoaded = true;
            System.Uri resourceLocater = new System.Uri("/Emulator;component/savefile.xaml",
System.UriKind.Relative);

#line 1 "..\..\..\SaveFile.xaml"
            System.Windows.Application.LoadComponent(this, resourceLocater);

#line default
#line hidden
        }

        [System.Diagnostics.DebuggerNonUserCodeAttribute()]
        [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]

[System.ComponentModel.EditorBrowsableAttribute(System.ComponentModel.EditorBrowsableState.Never)]
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Design",
"CA1033:InterfaceMethodsShouldBeCallableByChildTypes")]
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Maintainability",
"CA1502:AvoidExcessiveComplexity")]
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1800:DoNotCastUnnecessarily")]
        void System.Windows.Markup.IComponentConnector.Connect(int connectionId, object target) {
            switch (connectionId)
            {
            case 1:
            this.SelectFile = ((System.Windows.Controls.Button)(target));
            return;
            case 2:
            this.FilePath = ((System.Windows.Controls.TextBox)(target));
            return;
            case 3:
            this.PathText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 4:
            this.CancelButton = ((System.Windows.Controls.Button)(target));
            return;
            case 5:
            this.LoadButton = ((System.Windows.Controls.Button)(target));
            return;
            }
            this._contentLoaded = true;
        }
    }
}

```

7.59 Emulator/obj/x86/Debug/Settings.g.cs File Reference

Classes

- class Emulator.Settings

  Settings

Namespaces

- namespace Emulator

7.60 Settings.g.cs

Go to the documentation of this file.

```
#pragma checksum "..\..\..\Settings.xaml" "{8829d00f-11b8-4213-878b-770e8597ac16}"
"5C331E215A507ACA3F7FF07CFD574A81287117C06061A7F3A96858A63F0BA78B"
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:4.0.30319.42000
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

using System;
using System.Diagnostics;
using System.Windows;
using System.Windows.Automation;
using System.Windows.Controls;
using System.Windows.Controls.Primitives;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Markup;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Media.Effects;
using System.Windows.Media.Imaging;
using System.Windows.Media.Media3D;
using System.Windows.Media.TextFormatting;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Windows.Shell;


namespace Emulator {


/// <summary>
/// Settings
/// </summary>
    public partial class Settings :  System.Windows.Window, System.Windows.Markup.IComponentConnector {


#line 7 "..\..\..\Settings.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.ComboBox ComPortCombo;

#line default
#line hidden


#line 8 "..\..\..\Settings.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock PortText;

#line default
#line hidden


#line 9 "..\..\..\Settings.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.Button ApplyButton;

#line default
#line hidden


#line 10 "..\..\..\Settings.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.Button CloseButton;

#line default
#line hidden

        private bool _contentLoaded;

/// <summary>
/// InitializeComponent
/// </summary>
        [System.Diagnostics.DebuggerNonUserCodeAttribute()]
        [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
        public void InitializeComponent() {
            if (_contentLoaded) {
                return;
            }
            _contentLoaded = true;
            System.Uri resourceLocater = new System.Uri("/Emulator;component/settings.xaml", System.UriKind.Relative);

#line 1 "..\..\..\Settings.xaml"
            System.Windows.Application.LoadComponent(this, resourceLocater);

#line default
#line hidden
        }

        [System.Diagnostics.DebuggerNonUserCodeAttribute()]
        [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
[System.ComponentModel.EditorBrowsableAttribute(System.ComponentModel.EditorBrowsableState.Never)]
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Design",
"CA1033:InterfaceMethodsShouldBeCallableByChildTypes")]
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Maintainability",
"CA1502:AvoidExcessiveComplexity")]
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1800:DoNotCastUnnecessarily")]
        void System.Windows.Markup.IComponentConnector.Connect(int connectionId, object target) {
            switch (connectionId)
            {
            case 1:
            this.ComPortCombo = ((System.Windows.Controls.ComboBox)(target));

#line 7 "..\..\..\Settings.xaml"
            this.ComPortCombo.DropDownClosed += new
System.EventHandler(this.PortSelectionDropDownClosed);

#line default
#line hidden
            return;
            case 2:
            this.PortText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 3:
            this.ApplyButton = ((System.Windows.Controls.Button)(target));
            return;
            case 4:
            this.CloseButton = ((System.Windows.Controls.Button)(target));
            return;
            }
            this._contentLoaded = true;
        }
    }
}

```

7.61 Emulator/obj/x86/Debug/Settings.g.i.cs File Reference

Classes

- class Emulator.Settings

  Settings

Namespaces

- namespace Emulator

7.62 Settings.g.i.cs

Go to the documentation of this file.

```
#pragma checksum "..\..\..\Settings.xaml" "{8829d00f-11b8-4213-878b-770e8597ac16}"
"5C331E215A507ACA3F7FF07CFD574A81287117C06061A7F3A96858A63F0BA78B"
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:4.0.30319.42000
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

using System;
using System.Diagnostics;
using System.Windows;
using System.Windows.Automation;
using System.Windows.Controls;
using System.Windows.Controls.Primitives;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Markup;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Media.Effects;
using System.Windows.Media.Imaging;
using System.Windows.Media.Media3D;
using System.Windows.Media.TextFormatting;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Windows.Shell;


namespace Emulator {


/// <summary>
/// Settings
/// </summary>
    public partial class Settings :  System.Windows.Window, System.Windows.Markup.IComponentConnector {


#line 7 "..\..\..\Settings.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.ComboBox ComPortCombo;

#line default
#line hidden


#line 8 "..\..\..\Settings.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.TextBlock PortText;

#line default
#line hidden


#line 9 "..\..\..\Settings.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.Button ApplyButton;

#line default
#line hidden


#line 10 "..\..\..\Settings.xaml"
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.Button CloseButton;

#line default
#line hidden

        private bool _contentLoaded;

/// <summary>
/// InitializeComponent
/// </summary>
        [System.Diagnostics.DebuggerNonUserCodeAttribute()]
        [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
        public void InitializeComponent() {
            if (_contentLoaded) {
                return;
            }
            _contentLoaded = true;
            System.Uri resourceLocater = new System.Uri("/Emulator;component/settings.xaml", System.UriKind.Relative);

#line 1 "..\..\..\Settings.xaml"
            System.Windows.Application.LoadComponent(this, resourceLocater);

#line default
#line hidden
        }

        [System.Diagnostics.DebuggerNonUserCodeAttribute()]
        [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
[System.ComponentModel.EditorBrowsableAttribute(System.ComponentModel.EditorBrowsableState.Never)]
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Design",
"CA1033:InterfaceMethodsShouldBeCallableByChildTypes")]
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Maintainability",
"CA1502:AvoidExcessiveComplexity")]
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1800:DoNotCastUnnecessarily")]
        void System.Windows.Markup.IComponentConnector.Connect(int connectionId, object target) {
            switch (connectionId)
            {
            case 1:
            this.ComPortCombo = ((System.Windows.Controls.ComboBox)(target));

#line 7 "..\..\..\Settings.xaml"
            this.ComPortCombo.DropDownClosed += new
System.EventHandler(this.PortSelectionDropDownClosed);

#line default
#line hidden
            return;
            case 2:
            this.PortText = ((System.Windows.Controls.TextBlock)(target));
            return;
            case 3:
            this.ApplyButton = ((System.Windows.Controls.Button)(target));
            return;
            case 4:
            this.CloseButton = ((System.Windows.Controls.Button)(target));
            return;
            }
            this._contentLoaded = true;
        }
    }
}

```

7.63 Emulator/Properties/AssemblyInfo.cs File Reference

7.64 AssemblyInfo.cs

Go to the documentation of this file.

```
using System.Resources;
using System.Reflection;
using System.Runtime.InteropServices;
using System.Windows;
using Emulator;

// General Information about an assembly is controlled through the following
// set of attributes.  Change these attribute values to modify the information
// associated with an assembly.
[assembly:  AssemblyTitle(Versioning.Product.Title)]
[assembly:  AssemblyDescription(Versioning.Product.Description)]
[assembly:  AssemblyConfiguration("")]
[assembly:  AssemblyCompany(Versioning.Product.Company)]
[assembly:  AssemblyProduct(Versioning.Product.Name)]
[assembly:  AssemblyCopyright(Versioning.Product.Copyright)]
[assembly:  AssemblyTrademark("")]
[assembly:  AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not visible
// to COM components.    If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
[assembly:  ComVisible(false)]

//In order to begin building localizable applications, set
//<UICulture>CultureYouAreCodingWith</UICulture> in your .csproj file
//inside a <PropertyGroup>.    For example, if you are using US english
//in your source files, set the <UICulture> to en-US.   Then uncomment
//the NeutralResourceLanguage attribute below.    Update the "en-US" in
//the line below to match the UICulture setting in the project file.

//[assembly:  NeutralResourcesLanguage("en-US", UltimateResourceFallbackLocation.Satellite)]


[assembly:  ThemeInfo(
    ResourceDictionaryLocation.None, //where theme specific resource dictionaries are located
    //(used if a resource is not found in the page,
    // or application resource dictionaries)
    ResourceDictionaryLocation.SourceAssembly //where the generic resource dictionary is located
    //(used if a resource is not found in the page,
    // app, or any theme specific resource dictionaries)
)]


// Version information for an assembly consists of the following four values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the values or you can default the Build and Revision Numbers
// by using the '*' as shown below:
// [assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyVersion(Versioning.Product.VersionString)]
[assembly: AssemblyFileVersion(Versioning.Product.VersionString)]
[assembly: NeutralResourcesLanguage("en-GB")]
```

7.65 Hardware/Properties/AssemblyInfo.cs File Reference

7.66 AssemblyInfo.cs

Go to the documentation of this file.

```
using System.Resources;
using System.Reflection;
using System.Runtime.InteropServices;
using Hardware;

// General Information about an assembly is controlled through the following
// set of attributes.  Change these attribute values to modify the information
// associated with an assembly.
[assembly: AssemblyTitle(Versioning.Product.Title)]
[assembly: AssemblyDescription(Versioning.Product.Description)]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany(Versioning.Product.Company)]
[assembly: AssemblyProduct("")]
[assembly: AssemblyCopyright(Versioning.Product.Copyright)]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not visible
// to COM components.    If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]

// The following GUID is for the ID of the typelib if this project is exposed to COM
[assembly: Guid("f4afef76-2e8f-4497-86c6-c903aa70eebd")]

// Version information for an assembly consists of the following four values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the values or you can default the Build and Revision Numbers
// by using the '*' as shown below:
// [assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyVersion(Versioning.Product.Version)]
[assembly: AssemblyFileVersion(Versioning.Product.Version)]
[assembly: NeutralResourcesLanguage("")]
```

7.67 Emulator/SaveFile.xaml.cs File Reference

Classes

- class Emulator.SaveFile

  SaveFile

Namespaces

- namespace Emulator

7.68 SaveFile.xaml.cs

Go to the documentation of this file.

```
using GalaSoft.MvvmLight.Messaging;

namespace Emulator
{
/// <summary>
/// Interaction logic for SaveState.xaml
/// </summary>
    public partial class SaveFile
    {
        public SaveFile()
        {
            InitializeComponent();
            Messenger.Default.Register<NotificationMessage>(this, NotificationMessageReceived);
        }

        private void NotificationMessageReceived(NotificationMessage notificationMessage)
        {
            if (notificationMessage.Notification == "CloseSaveFileWindow")
                Close();
        }
    }
}
```

7.69 Emulator/Settings.xaml.cs File Reference

Classes

- class Emulator.Settings

  Settings

Namespaces

- namespace Emulator

7.70 Settings.xaml.cs

Go to the documentation of this file.

```
using GalaSoft.MvvmLight.Messaging;
using Emulator.Model;
using Emulator.ViewModel;
using System;
using System.Windows;

namespace Emulator
{
/// <summary>
/// Interaction logic for Settings.xaml
/// </summary>
    public partial class Settings
    {
        public Settings()
        {
            InitializeComponent();
            Messenger.Default.Register<NotificationMessage>(this, NotificationMessageReceived);
            Messenger.Default.Register<NotificationMessage<SettingsModel»(this,
NotificationMessageReceived);
        }

        private void NotificationMessageReceived(NotificationMessage notificationMessage)
        {
            if (notificationMessage.Notification == "CloseSettingsWindow")
            {
                Close();
            }
        }

        private void NotificationMessageReceived(NotificationMessage<SettingsModel>
notificationMessage)
        {
            if (notificationMessage.Notification == "SettingsWindow")
            {
                SettingsViewModel.SettingsModel = notificationMessage.Content;
                ComPortCombo.SelectedItem = notificationMessage.Content.ComPortName;
            }
        }

        private void PortSelectionDropDownClosed(object sender, EventArgs e)
        {
            if (!(ComPortCombo.SelectedValue == null))
            {
                string port = ComPortCombo.SelectedValue.ToString();
                SettingsViewModel.ComPortSelection = port;
            }
        }
    }
}
```

7.71 Emulator/ViewModel/MainViewModel.cs File Reference

Classes

- class Emulator.ViewModel.MainViewModel

  The Main ViewModel

Namespaces

- namespace Emulator
- namespace Emulator.ViewModel

Typedefs

- using W65C02 = Hardware.W65C02
- using W65C22 = Hardware.W65C22
- using W65C51 = Hardware.W65C51

7.71.1 Typedef Documentation

7.71.1.1 W65C02 `using W65C02 = Hardware.W65C02`

Definition at line 16 of file MainViewModel.cs.

7.71.1.2 W65C22 `using W65C22 = Hardware.W65C22`

Definition at line 17 of file MainViewModel.cs.

7.71.1.3 W65C51 `using W65C51 = Hardware.W65C51`

Definition at line 18 of file MainViewModel.cs.

7.72 MainViewModel.cs

Go to the documentation of this file.

```
00001 using Microsoft.Win32;
00002 using System;
00003 using System.Collections.Generic;
00004 using System.ComponentModel;
00005 using System.Globalization;
00006 using System.IO;
00007 using System.Linq;
00008 using System.Threading;
00009 using System.Windows;
00010 using System.Xml.Serialization;
00011 using GalaSoft.MvvmLight;
00012 using GalaSoft.MvvmLight.Command;
00013 using GalaSoft.MvvmLight.Messaging;
00014 using Hardware;
00015 using Emulator.Model;
00016 using W65C02 = Hardware.W65C02;
00017 using W65C22 = Hardware.W65C22;
00018 using W65C51 = Hardware.W65C51;
00019 using System.Runtime.Serialization.Formatters.Binary;
00020 using System.Windows.Navigation;
00021
00022 namespace Emulator.ViewModel
00023 {
00024 /// <summary>
00025 /// The Main ViewModel
00026 /// </summary>
00027     public class MainViewModel :  ViewModelBase
00028     {
00029 #region Fields
00030         private int _memoryPageOffset;
00031         private readonly BackgroundWorker _backgroundWorker;
00032         private bool _breakpointTriggered;
00033 #endregion
00034
00035 #region Properties
00036 /// <summary>
00037 /// The 62256 RAM.
00038 /// </summary>
00039         private HM62256 HM62256 { get; set; }
00040
00041 /// <summary>
00042 /// The 65C02 Processor.
00043 /// </summary>
00044         public W65C02 W65C02 { get; private set; }
00045
00046 /// <summary>
00047 /// General Purpose I/O, Shift Registers and Timers.
00048 /// </summary>
00049         public W65C22 W65C22 { get; private set; }
00050
00051 /// <summary>
00052 /// Memory management and 65SIB.
00053 /// </summary>
00054         public W65C22 MM65SIB { get; private set; }
00055
00056 /// <summary>
00057 /// The ACIA serial interface.
00058 /// </summary>
00059         public W65C51 W65C51 { get; private set; }
00060
00061 /// <summary>
00062 /// The AT28C010 ROM.
00063 /// </summary>
00064         public AT28CXX AT28C64 { get; private set; }
```

```

/// <summary>
/// The AT28C010 ROM.
/// </summary>
        public AT28CXX AT28C010 { get; private set; }

/// <summary>
/// The Current Memory Page
/// </summary>
        public MultiThreadedObservableCollection<MemoryRowModel> MemoryPage { get; set; }

/// <summary>
/// The output log
/// </summary>
        public MultiThreadedObservableCollection<OutputLog> OutputLog { get; private set; }

/// <summary>
/// The Breakpoints
/// </summary>
        public MultiThreadedObservableCollection<Breakpoint> Breakpoints { get; set; }

/// <summary>
/// The Currently Selected Breakpoint
/// </summary>
        public Breakpoint SelectedBreakpoint { get; set; }

/// <summary>
/// The currently loaded binary file.  (If it is indeed loaded, that is.)
/// </summary>
        public RomFileModel RomFile { get; set; }

/// <summary>
/// The Current Disassembly
/// </summary>
        public string CurrentDisassembly
        {
            get
            {
                if (W65C02.CurrentDisassembly != null)
                {
                    return string.Format("{0} {1}", W65C02.CurrentDisassembly.OpCodeString,
W65C02.CurrentDisassembly.DisassemblyOutput);
                }
                else
                {
                    return string.Empty;
                }
            }
        }

/// <summary>
/// The number of cycles.
/// </summary>
        public int NumberOfCycles { get; private set; }

/// <summary>
/// The Memory Page number.
/// </summary>
        public string MemoryPageOffset
        {
            get { return _memoryPageOffset.ToString("X"); }
            set
            {
                if (string.IsNullOrEmpty(value))
                    return;
                try
                {
                    _memoryPageOffset = Convert.ToInt32(value, 16);
                }
                catch { }
            }
        }

/// <summary>
///  Is the Prorgam Running
/// </summary>
        public bool IsRunning
        {
            get { return W65C02.isRunning; }
            set
            {
                W65C02.isRunning = value;
                RaisePropertyChanged("IsRunning");
            }
        }

/// <summary>
/// Is the banked ROM Loaded.
/// </summary>
        public bool IsRomLoaded { get; set; }

/// <summary>
/// The Slider CPU Speed
/// </summary>
        public int CpuSpeed { get; set; }

/// <summary>
/// The Model used for saving, loading and using data from Settings.xml
/// </summary>
        public static SettingsModel SettingsModel { get; set; }

/// <summary>
/// RelayCommand for Stepping through the progam one instruction at a time.
/// </summary>
        public RelayCommand StepCommand { get; set; }

/// <summary>
/// Relay Command to Reset the Program back to its initial state.
/// </summary>
        public RelayCommand ResetCommand { get; set; }

/// <summary>
/// Relay Command that Run/Pauses Execution
/// </summary>
        public RelayCommand RunPauseCommand { get; set; }

/// <summary>
/// Relay Command that updates the Memory Map when the Page changes
/// </summary>
        public RelayCommand UpdateMemoryMapCommand { get; set; }

/// <summary>
/// The Relay Command that adds a new breakpoint
/// </summary>
        public RelayCommand AddBreakPointCommand { get; set; }

/// <summary>
/// The Relay Command that opens the About window.
/// </summary>
        public RelayCommand AboutCommand { get; set; }

/// <summary>
/// The Relay Command that Removes an existing breakpoint.
/// </summary>
        public RelayCommand RemoveBreakPointCommand { get; set; }

/// <summary>
/// The Command that loads or saves the settings.
/// </summary>
        public RelayCommand SettingsCommand { get; set; }

/// <summary>
/// The Command that loads or saves the settings.
/// </summary>
        public RelayCommand<IClosable> CloseCommand { get; private set; }

/// <summary>
/// The current serial port object name.
/// </summary>
        public string CurrentSerialPort
        {
            get
            {
                return W65C51.ObjectName;
            }
        }

/// <summary>
/// The title for the main window.
/// </summary>
        public string WindowTitle { get { return Versioning.Product.Title; } }
#endregion

#region public Methods
/// <summary>
/// Creates a new Instance of the MainViewModel.
/// </summary>
        public MainViewModel()
        {
            var _formatter = new XmlSerializer(typeof(SettingsModel));
            Stream _stream = new FileStream(FileLocations.SettingsFile, FileMode.OpenOrCreate);
            if (!((_stream == null) || (0 >= _stream.Length)))
            {
                SettingsModel = (SettingsModel)_formatter.Deserialize(_stream);
                if ((SettingsModel.SettingsVersionMajor < Versioning.SettingsFile.Major) ||
                    (SettingsModel.SettingsVersionMinor < Versioning.SettingsFile.Minor) ||
                    (SettingsModel.SettingsVersionBuild < Versioning.SettingsFile.Build) ||
                    (SettingsModel.SettingsVersionRevision < Versioning.SettingsFile.Revision))
                {
#if !DEBUG
                    throw new NotImplementedException(String.Format("Unable to handle problem: Settings File version is less than {0}.{1}.{2}.{3}", Versioning.SettingsFile.Major, Versioning.SettingsFile.Minor, Versioning.SettingsFile.Revision, Versioning.SettingsFile.Build));
#else
                    MessageBox.Show("Settings file contains old information...\nDeleting old settings file...",
                                    "Settings file stale!", MessageBoxButton.OKCancel, MessageBoxImage.Warning,
                                    MessageBoxResult.OK);
                    // Close the file, then delete it.
                    _stream.Close();
                    File.Delete(FileLocations.SettingsFile);
                    SettingsModel = SettingsFile.CreateNew();
#endif
                }
            }
            else
            {
                MessageBox.Show("Creating new settings file...");
                SettingsModel = SettingsFile.CreateNew();
            }
            _stream.Close();

            HM62256 = new HM62256(MemoryMap.BankedRam.TotalBanks, MemoryMap.BankedRam.Offset, MemoryMap.BankedRam.Length);
            AT28C64 = new AT28CXX(MemoryMap.SharedRom.Offset, MemoryMap.SharedRom.Length, 1);
            AT28C010 = new AT28CXX(MemoryMap.BankedRom.Offset, MemoryMap.BankedRom.Length, MemoryMap.BankedRom.TotalBanks);
            W65C02 = new W65C02();
            W65C51 = new W65C51(W65C02, MemoryMap.Devices.ACIA.Offset);
            W65C51.Init(SettingsModel.ComPortName.ToString());
            W65C22 = new W65C22(W65C02, MemoryMap.Devices.GPIO.Offset, MemoryMap.Devices.GPIO.Length);
            W65C22.Init(1000);
            MM65SIB = new W65C22(W65C02, MemoryMap.Devices.MM65SIB.Offset, MemoryMap.Devices.MM65SIB.Length);
            MM65SIB.Init(1000);

            MemoryMap.Init(W65C02, W65C22, MM65SIB, W65C51, HM62256, AT28C010, AT28C64);

            // Now we can load the BIOS.
            byte[][] _bios = AT28C64.ReadFile(FileLocations.BiosFile);
            if (_bios == null)
            {
                Environment.Exit(ExitCodes.NO_BIOS);
            }
            AT28C64.Load(_bios);

            AboutCommand = new RelayCommand(About);
            AddBreakPointCommand = new RelayCommand(AddBreakPoint);
            CloseCommand = new RelayCommand<IClosable>(Close);
            RemoveBreakPointCommand = new RelayCommand(RemoveBreakPoint);
            ResetCommand = new RelayCommand(Reset);
            RunPauseCommand = new RelayCommand(RunPause);
            SettingsCommand = new RelayCommand(Settings);
            StepCommand = new RelayCommand(Step);
            UpdateMemoryMapCommand = new RelayCommand(UpdateMemoryPage);

            Messenger.Default.Register<NotificationMessage>(this, GenericNotifcation);
            Messenger.Default.Register<NotificationMessage<RomFileModel»(this, BinaryLoadedNotification);
            Messenger.Default.Register<NotificationMessage<SettingsModel»(this, SettingsAppliedNotifcation);
            Messenger.Default.Register<NotificationMessage<StateFileModel»(this, StateLoadedNotifcation);

            MemoryPage = new MultiThreadedObservableCollection<MemoryRowModel>();
            OutputLog = new MultiThreadedObservableCollection<OutputLog>();
            Breakpoints = new MultiThreadedObservableCollection<Breakpoint>();

            UpdateMemoryPage();

            _backgroundWorker = new BackgroundWorker { WorkerSupportsCancellation = true, WorkerReportsProgress = false };
            _backgroundWorker.DoWork += BackgroundWorkerDoWork;
            Application.Current.MainWindow.Closing += new CancelEventHandler(OnClose);
            Application.Current.MainWindow.Loaded += new RoutedEventHandler(OnLoad);

            Reset();
        }

        public void OnLoad(Object sender, RoutedEventArgs e)
        {
#if !DEBUG
            if (Versioning.Product.Major < 1)
            {
                var result = MessageBox.Show(String.Format("Thank you for using {0}\n" +
                                                        "Be warned that this is a beta build.\n" +
                                                        "It may break or have bugs.", Versioning.Product.Name),
                                                        Versioning.Product.Title, MessageBoxButton.OKCancel,
                                                        MessageBoxImage.Warning, MessageBoxResult.None);
                if (result == MessageBoxResult.Cancel)
                {
                    // Exit without making any changes.
                    Environment.Exit(ExitCodes.NO_ERROR);
                }
            }
#endif
        }

        public void OnClose(Object sender, CancelEventArgs e)
        {
            e.Cancel = false;
            if (IsRunning)
            {
                MessageBox.Show("You can't quit the emulator while it is actively running!",
                                "You can't do that!", MessageBoxButton.OK, MessageBoxImage.Stop);
                e.Cancel = true;
                return;
            }
#if !DEBUG
            else
            {
                var result = MessageBox.Show(   "Are you sure you want to quit the emulator?",
                                                "To quit, or not to quit -- that is the question.",
                                                MessageBoxButton.YesNo, MessageBoxImage.Question,
                                                MessageBoxResult.No);
                if (result == MessageBoxResult.No)
                {
                    e.Cancel = true;
                    return;
                }
            }
#endif
            Stream stream = new FileStream(FileLocations.SettingsFile, FileMode.Create, FileAccess.Write, FileShare.None);
            XmlSerializer XmlFormatter = new XmlSerializer(typeof(SettingsModel));
            XmlFormatter.Serialize(stream, MainViewModel.SettingsModel);
            stream.Flush();
            stream.Close();
            W65C51.Fini();
        }
#endregion

#region Private Methods
        private void Close(IClosable window)
        {
            if ((window != null) && (!IsRunning))
            {
                Environment.Exit(ExitCodes.NO_ERROR);
            }
        }

        private void BinaryLoadedNotification(NotificationMessage<RomFileModel> notificationMessage)
        {
            if (notificationMessage.Notification != "FileLoaded")
            {
                return;
            }

            // Load Banked ROM
            AT28C010.Load(notificationMessage.Content.Rom);
            IsRomLoaded = true;
            RaisePropertyChanged("IsRomLoaded");

            Reset();
        }

        private void StateLoadedNotifcation(NotificationMessage<StateFileModel> notificationMessage)
        {
            if (notificationMessage.Notification != "StateLoaded")
            {
                return;
            }

            Reset();

            OutputLog = new MultiThreadedObservableCollection<OutputLog>(notificationMessage.Content.OutputLog);
            RaisePropertyChanged("OutputLog");

            NumberOfCycles = notificationMessage.Content.NumberOfCycles;

            W65C02 = notificationMessage.Content.W65C02;
            W65C22 = notificationMessage.Content.W65C22;
            MM65SIB = notificationMessage.Content.MM65SIB;
            W65C51 = notificationMessage.Content.W65C51;
            AT28C010 = notificationMessage.Content.AT28C010;
            AT28C64 = notificationMessage.Content.AT28C64;
            UpdateMemoryPage();
            UpdateUi();

            IsRomLoaded = true;
            RaisePropertyChanged("IsRomLoaded");
        }

        private void GenericNotifcation(NotificationMessage notificationMessage)
        {
            if (notificationMessage.Notification == "CloseFile")
            {
                AT28C010.Clear();
                if (IsRunning) { RunPause(); }
                IsRomLoaded = false;
                RaisePropertyChanged("IsRomLoaded");
                return;
            }
            else if (notificationMessage.Notification == "LoadFile")
            {
                var dialog = new OpenFileDialog {   DefaultExt = ".bin", Filter =
                                                    "All Files (*.bin, *.65C02)|*.bin;*.65C02|Binary Assembly (*.bin)|" +
                                                    "*.bin|WolfNet 65C02 Emulator Save State (*.65C02)|*.65C02" };
                var result = dialog.ShowDialog();
                if (result != true)
                {
                    return;
                }

                if (Path.GetExtension(dialog.FileName.ToUpper()) == ".BIN")
                {
                    byte[][] _rom = AT28C010.ReadFile(dialog.FileName);

                    Messenger.Default.Send(new NotificationMessage<RomFileModel>(new RomFileModel
                    {
                        Rom = _rom,
                        RomBanks = AT28C010.Banks,
                        RomBankSize = AT28C010.Length,
                        RomFilePath = dialog.FileName,
                        RomFileName = Path.GetFileName(dialog.FileName),
                    }, "FileLoaded"));
                }
                else if (Path.GetExtension(dialog.FileName.ToUpper()) == ".6502")
                {
                    var formatter = new BinaryFormatter();
                    Stream stream = new FileStream(dialog.FileName, FileMode.Open);
                    var fileModel = (StateFileModel)formatter.Deserialize(stream);

                    stream.Close();

                    Messenger.Default.Send(new NotificationMessage<StateFileModel>(fileModel, "StateLoaded"));
                }
            }
            else if (notificationMessage.Notification == "SaveState")
            {
                var dialog = new SaveFileDialog {   DefaultExt = ".65C02", Filter =
                                                    "WolfNet W65C02 Emulator Save State (*.65C02)|*.65C02" };
                var result = dialog.ShowDialog();

                if (result != true)
                {
                    return;
                }

                var formatter = new BinaryFormatter();
                Stream stream = new FileStream(dialog.FileName, FileMode.Create, FileAccess.Write, FileShare.None);

                formatter.Serialize(stream, new StateFileModel
                {
                    NumberOfCycles = NumberOfCycles,
                    OutputLog = OutputLog,
                    W65C02 = W65C02,
                    W65C22 = W65C22,
                    MM65SIB = MM65SIB,
                    W65C51 = W65C51,
                    AT28C010 = AT28C010,
                    AT28C64 = AT28C64,
            });
                stream.Close();
            }
            else
            {
                return;
            }
        }

        private void SettingsAppliedNotifcation(NotificationMessage<SettingsModel>
notificationMessage)
        {
            if (notificationMessage.Notification != "SettingsApplied")
            {
                return;
            }

            SettingsModel = notificationMessage.Content;
            W65C51.Init(notificationMessage.Content.ComPortName);
            RaisePropertyChanged("SettingsModel");
            UpdateUi();
        }

        private void UpdateMemoryPage()
        {
            MemoryPage.Clear();
            var offset = _memoryPageOffset * 256;

            var multiplyer = 0;
            for (ushort i = (ushort)offset; i < 256 * (_memoryPageOffset + 1); i++)
            {

                MemoryPage.Add(new MemoryRowModel
                {
                    Offset = ((16 * multiplyer) + offset).ToString("X").PadLeft(4, '0'),
                    Location00 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location01 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location02 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location03 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location04 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location05 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location06 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location07 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location08 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location09 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location0A = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location0B = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location0C = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location0D = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location0E = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
                    Location0F = MemoryMap.ReadWithoutCycle(i).ToString("X").PadLeft(2, '0'),
                });
                multiplyer++;
            }
        }

        private void Reset()
        {
            IsRunning = false;

            if (_backgroundWorker.IsBusy)
                _backgroundWorker.CancelAsync();

            // "Reset" the Hardware...
            W65C02.Reset();
            RaisePropertyChanged("W65C02");
            W65C22.Reset();
            RaisePropertyChanged("W65C22");
            MM65SIB.Reset();
            RaisePropertyChanged("MM65SIB");
            W65C51.Reset();
            RaisePropertyChanged("W65C51");
            HM62256.Reset();
            RaisePropertyChanged("HM62256");

            IsRunning = false;
            NumberOfCycles = 0;
            RaisePropertyChanged("NumberOfCycles");

            UpdateMemoryPage();
            RaisePropertyChanged("MemoryPage");

            OutputLog.Clear();
            RaisePropertyChanged("CurrentDisassembly");

            OutputLog.Insert(0, GetOutputLog());
            UpdateUi();
        }

        private void Step()
        {
            IsRunning = false;

            if (_backgroundWorker.IsBusy)
                _backgroundWorker.CancelAsync();

            StepProcessor();
            UpdateMemoryPage();

            OutputLog.Insert(0, GetOutputLog());
            UpdateUi();
        }

        private void UpdateUi()
        {
            RaisePropertyChanged("W65C02");
            RaisePropertyChanged("NumberOfCycles");
            RaisePropertyChanged("CurrentDisassembly");
            RaisePropertyChanged("MemoryPage");
        }

        private void StepProcessor()
        {
            W65C02.NextStep();
            NumberOfCycles = W65C02.GetCycleCount();
        }

        private OutputLog GetOutputLog()
        {
            if (W65C02.CurrentDisassembly == null)
            {
                return new OutputLog(new Disassembly());
            }

            return new OutputLog(W65C02.CurrentDisassembly)
            {
                XRegister = W65C02.XRegister.ToString("X").PadLeft(2, '0'),
                YRegister = W65C02.YRegister.ToString("X").PadLeft(2, '0'),
                Accumulator = W65C02.Accumulator.ToString("X").PadLeft(2, '0'),
                NumberOfCycles = NumberOfCycles,
                StackPointer = W65C02.StackPointer.ToString("X").PadLeft(2, '0'),
                ProgramCounter = W65C02.ProgramCounter.ToString("X").PadLeft(4, '0'),
                CurrentOpCode = W65C02.CurrentOpCode.ToString("X").PadLeft(2, '0')
            };
        }

        private void RunPause()
        {
            var isRunning = !IsRunning;

            if (isRunning)
                _backgroundWorker.RunWorkerAsync();
            else
                _backgroundWorker.CancelAsync();

            IsRunning = !IsRunning;
        }

        private void BackgroundWorkerDoWork(object sender, DoWorkEventArgs e)
        {
            var worker = sender as BackgroundWorker;
            var outputLogs = new List<OutputLog>();

            while (true)
            {
                if (worker != null && worker.CancellationPending || IsBreakPointTriggered())
                {
                    e.Cancel = true;

                    RaisePropertyChanged("W65C02");

                    foreach (var log in outputLogs)
                        OutputLog.Insert(0, log);

                    UpdateMemoryPage();
                    return;
                }

                StepProcessor();
                outputLogs.Add(GetOutputLog());

                if (NumberOfCycles % GetLogModValue() == 0)
                {
                    foreach (var log in outputLogs)
                        OutputLog.Insert(0, log);

                    outputLogs.Clear();
                    UpdateUi();
                }
                Thread.Sleep(GetSleepValue());
            }
        }

        private bool IsBreakPointTriggered()
        {
            //This prevents the Run Command from getting stuck after reaching a breakpoint
            if (_breakpointTriggered)
            {
                _breakpointTriggered = false;
                return false;
            }

            foreach (var breakpoint in Breakpoints.Where(x => x.IsEnabled))
            {
                if (!int.TryParse(breakpoint.Value, NumberStyles.AllowHexSpecifier,
      CultureInfo.InvariantCulture, out int value))
                    continue;

                if (breakpoint.Type == BreakpointType.NumberOfCycleType && value == NumberOfCycles)
                {
                    _breakpointTriggered = true;
                    RunPause();
                    return true;
                }

                if (breakpoint.Type == BreakpointType.ProgramCounterType && value ==
      W65C02.ProgramCounter)
                {
                    _breakpointTriggered = true;
                    RunPause();
                    return true;
                }
            }

            return false;
        }

        private int GetLogModValue()
        {
            switch (CpuSpeed)
            {
                case 0:
                case 1:
                case 2:
                case 3:
                case 4:
                case 5:
                    return 1;
                case 6:
                    return 5;
                case 7:
                    return 20;
                case 8:
                    return 30;
                case 9:
                    return 40;
                case 10:
                    return 50;
                default:
                    return 5;
            }
        }

        private int GetSleepValue()
        {
            switch (CpuSpeed)
            {
                case 0:
                    return 550;
                case 1:
                    return 550;
                case 2:
                    return 440;
                case 3:
                    return 330;
                case 4:
                    return 220;
                case 5:
                    return 160;
                case 6:
                    return 80;
                case 7:
                    return 40;
                case 8:
                    return 20;
                case 9:
                    return 10;
                case 10:
                    return 5;
                default:
                    return 5;
            }
        }

        private void About()
        {
            IsRunning = false;

            if (_backgroundWorker.IsBusy)
                _backgroundWorker.CancelAsync();

            MessageBox.Show(string.Format("{0}\n{1}\nVersion:  {2}\nCompany:  {3}", Versioning.Product.Name, Versioning.Product.Description, Versioning.Product.VersionString, Versioning.Product.Company), Versioning.Product.Title);
        }

        private void Settings()
        {
            IsRunning = false;

            if (_backgroundWorker.IsBusy)
                _backgroundWorker.CancelAsync();

            Messenger.Default.Send(new NotificationMessage<SettingsModel>(SettingsModel, "SettingsWindow"));
        }

        private void AddBreakPoint()
        {
            Breakpoints.Add(new Breakpoint());
            RaisePropertyChanged("Breakpoints");
        }

        private void RemoveBreakPoint()
        {
            if (SelectedBreakpoint == null)
                return;

            Breakpoints.Remove(SelectedBreakpoint);
            SelectedBreakpoint = null;
            RaisePropertyChanged("SelectedBreakpoint");
        }
#endregion
    }
}
```

7.73 Emulator/ViewModel/SaveFileViewModel.cs File Reference

Classes

- class Emulator.ViewModel.SaveFileViewModel

  The ViewModel Used by the SaveFileView

Namespaces

- namespace Emulator
- namespace Emulator.ViewModel

7.74 SaveFileViewModel.cs

Go to the documentation of this file.

```
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;
using GalaSoft.MvvmLight;
using GalaSoft.MvvmLight.Command;
using GalaSoft.MvvmLight.Ioc;
using GalaSoft.MvvmLight.Messaging;
using Microsoft.Win32;
using Emulator.Model;

namespace Emulator.ViewModel
{
/// <summary>
/// The ViewModel Used by the SaveFileView
/// </summary>
    public class SaveFileViewModel :  ViewModelBase
    {
        private readonly StateFileModel _stateFileModel;

#region Properties
/// <summary>
/// The Relay Command called when saving a file
/// </summary>
        public RelayCommand SaveFileCommand { get; set; }

/// <summary>
/// The Relay Command called when closing a file
/// </summary>
        public RelayCommand CloseCommand { get; set; }

/// <summary>
/// The Relay Command called when Selecting a file
/// </summary>
        public RelayCommand SelectFileCommand { get; set; }

/// <summary>
/// The file to be saved
/// </summary>
        public string Filename { get; set; }

/// <summary>
/// Tells the UI that that a file has been selected and can be saved.
/// </summary>
        public bool SaveEnabled { get { return !string.IsNullOrEmpty(Filename); }}
#endregion

#region Public Methods
/// <summary>
/// Instantiates a new instance of the SaveFileViewModel.  This is used by the IOC to create the
default instance.
/// </summary>
        [PreferredConstructor]
        public SaveFileViewModel()
        {

        }

/// <summary>
/// Instantiates a new instance of the SaveFileViewModel
/// </summary>
/// <param name="stateFileModel">The StateFileModel to be serialized to a file</param>
        public SaveFileViewModel(StateFileModel stateFileModel)
        {
            SaveFileCommand = new RelayCommand(Save);
            CloseCommand = new RelayCommand(Close);
            SelectFileCommand = new RelayCommand(Select);
            _stateFileModel = stateFileModel;
        }
#endregion

#region Private Methods
        private void Save()
        {
            var formatter = new BinaryFormatter();
            Stream stream = new FileStream(Filename, FileMode.Create, FileAccess.Write,
FileShare.None);
            formatter.Serialize(stream, _stateFileModel);
            stream.Close();

            Close();
        }

        private static void Close()
        {
            Messenger.Default.Send(new NotificationMessage("CloseSaveFileWindow"));
        }

        private void Select()
        {
            var dialog = new SaveFileDialog { DefaultExt = ".6502", Filter = "WolfNet W65C02 Emulator
Save State (*.6502)|*.6502" };

            var result = dialog.ShowDialog();

            if (result != true)
                return;

            Filename = dialog.FileName;
            RaisePropertyChanged("Filename");
            RaisePropertyChanged("SaveEnabled");

        }
#endregion
    }
}
```

7.75 Emulator/ViewModel/SettingsViewModel.cs File Reference

Classes

- class Emulator.ViewModel.SettingsViewModel

  The ViewModel Used by the SaveFileView

Namespaces

- namespace Emulator
- namespace Emulator.ViewModel

7.76 SettingsViewModel.cs

Go to the documentation of this file.

```
using System;
using System.Collections.ObjectModel;
using System.IO;
using System.IO.Ports;
using System.Xml.Serialization;
using GalaSoft.MvvmLight;
using GalaSoft.MvvmLight.Command;
using GalaSoft.MvvmLight.Ioc;
using GalaSoft.MvvmLight.Messaging;
using Emulator.Model;

namespace Emulator.ViewModel
{
/// <summary>
/// The ViewModel Used by the SaveFileView
/// </summary>
    public class SettingsViewModel :  ViewModelBase
    {
#region Properties
/// <summary>
/// The Relay Command called when saving a file
/// </summary>
        public RelayCommand ApplyCommand { get; set; }

/// <summary>
/// The Relay Command called when closing a file
/// </summary>
        public RelayCommand CloseCommand { get; set; }

/// <summary>
/// Tells the UI that that a file has been selected and can be saved.
/// </summary>
        public bool ApplyEnabled { get { return
!string.IsNullOrEmpty(Emulator.FileLocations.SettingsFile); } }

/// <summary>
/// Creates a new instance of PortList, the list of all COM ports available to the computer
/// </summary>
///
        public ObservableCollection<string> PortList { get { return _PortList; } }
        private readonly ObservableCollection<string> _PortList = new ObservableCollection<string>();

        public static string ComPortSelection { get; set; }
        public static SettingsModel SettingsModel { get; set; }
#endregion

#region Public Methods
/// <summary>
/// Instantiates a new instance of the SettingsViewModel.  This is used by the IOC to create the
default instance.
/// </summary>
        [PreferredConstructor]
        public SettingsViewModel()
        {

        }

/// <summary>
/// Instantiates a new instance of the SettingsViewModel
/// </summary>
/// <param name="settingsModel">The SettingsFileModel to be serialized to a file</param>
        public SettingsViewModel(SettingsModel settingsModel)
        {
            ApplyCommand = new RelayCommand(Apply);
            CloseCommand = new RelayCommand(Close);
            ComPortSelection = settingsModel.ComPortName;

            UpdatePortList();
        }

/// <summary>
/// Updates PortList with the COM ports available to the computer
/// </summary>
        public void UpdatePortList()
        {
            PortList.Clear();
            foreach (string s in SerialPort.GetPortNames())
            {
                PortList.Add(s);
            }
            RaisePropertyChanged("PortList");
        }
#endregion

#region Private Methods
        private void Apply()
        {
            Messenger.Default.Send(new NotificationMessage<SettingsModel>(new SettingsModel
            {
                SettingsVersionMajor = Versioning.SettingsFile.Major,
                SettingsVersionMinor = Versioning.SettingsFile.Minor,
                SettingsVersionBuild = Versioning.SettingsFile.Build,
                SettingsVersionRevision = Versioning.SettingsFile.Revision,
                ComPortName = ComPortSelection,
            }, "SettingsApplied"));
            Messenger.Default.Send(new NotificationMessage("CloseSettingsWindow"));
        }

        private static void Close()
        {
            Messenger.Default.Send(new NotificationMessage("CloseSettingsWindow"));
        }
#endregion
    }
}
```

7.77 Emulator/ViewModel/ViewModelLocator.cs File Reference

Classes

- class Emulator.ViewModel.ViewModelLocator

  This class contains static references to all the view models in the application and provides an entry point for the bindings.

Namespaces

- namespace Emulator
- namespace Emulator.ViewModel

7.78 ViewModelLocator.cs

Go to the documentation of this file.

```
/*
In App.xaml:
<Application.Resources>
<vm:ViewModelLocator xmlns:vm="clr-namespace:Emulator"
x:Key="Locator" />
</Application.Resources>

In the View:
DataContext="{Binding Source={StaticResource Locator}, Path=ViewModelName}"

You can also use Blend to do all this with the tool's support.
See http://www.galasoft.ch/mvvm
*/

using GalaSoft.MvvmLight.Ioc;
using Microsoft.Practices.ServiceLocation;

namespace Emulator.ViewModel
{
/// <summary>
/// This class contains static references to all the view models in the
/// application and provides an entry point for the bindings.
/// </summary>
    public class ViewModelLocator
    {
/// <summary>
/// Initializes a new instance of the ViewModelLocator class.
/// </summary>
        public ViewModelLocator()
        {
            ServiceLocator.SetLocatorProvider(() => SimpleIoc.Default);

            SimpleIoc.Default.Register<MainViewModel>();
            SimpleIoc.Default.Register<SaveFileViewModel>();
            SimpleIoc.Default.Register<SettingsViewModel>();
        }

/// <summary>
/// The MainViewModel Instance
/// </summary>
        public MainViewModel Main
        {
            get { return ServiceLocator.Current.GetInstance<MainViewModel>(); }
        }

/// <summary>
/// The SaveFileViewModel Instance
/// </summary>
        public SaveFileViewModel SaveFile
        {
            get { return ServiceLocator.Current.GetInstance<SaveFileViewModel>(); }
        }

/// <summary>
/// The SaveFileViewModel Instance
/// </summary>
        public SettingsViewModel Settings
        {
            get { return ServiceLocator.Current.GetInstance<SettingsViewModel>(); }
        }

/// <summary>
/// The Cleanup Method
/// </summary>
        public static void Cleanup()
        {
/// <todo>
/// Clear the ViewModels
/// </todo>
        }
    }
}
```

7.79 Hardware/AT28CXX.cs File Reference

Classes

- class Hardware.AT28CXX

  An implementation of a W65C02 Processor.

Namespaces

- namespace Hardware

7.80 AT28CXX.cs

Go to the documentation of this file.

```
using System;
using System.IO;

namespace Hardware
{
/// <summary>
/// An implementation of a W65C02 Processor.
/// </summary>
    [Serializable]
    public class AT28CXX
    {
        //All of the properties here are public and read only to facilitate ease of debugging and
testing.
#region Properties
/// <summary>
/// The ROM.
/// </summary>
        public byte[][] Memory { get; private set; }

/// <summary>
/// The total number of banks on the ROM.
/// </summary>
        public byte Banks { get; private set; }

/// <summary>
/// The bank the ROM is currently using.
/// </summary>
        public byte CurrentBank { get; private set; }

/// <summary>
/// The memory offset
/// </summary>
        public int Offset { get; private set; }

/// <summary>
/// The end of memory
/// </summary>
        public int End { get { return Offset + Length; } }

/// <summary>
/// The memory length
/// </summary>
        public int Length { get; private set; }

/// <summary>
/// The processor reference
/// </summary>
        public W65C02 Processor{ get; private set; }
#endregion

#region Public Methods
/// <summary>
/// Default Constructor, Instantiates a new instance of the processor.
/// </summary>
        public AT28CXX(int offset, int length, byte banks)
        {
            Memory = new byte[banks][];
            for (int i = 0; i < banks; i++)
            {
                Memory[i] = new byte[length + 1];
            }
            Offset = offset;
            Length = length;
            Banks = banks;
            CurrentBank = 0;
        }

/// <summary>
/// Loads a program into ROM.
/// </summary>
/// <param name="data">The program to be loaded</param>
        public void Load(byte[][] data)
        {
            for (byte i = 0; i < Banks; i++)
            {
                Load(i, data[i]);
            }
        }

/// <summary>
/// Loads a program into ROM.
/// </summary>
/// <param name="bank">The bank to load data to.</param>
/// <param name="data">The data to be loaded to ROM.</param>
        public void Load(byte bank, byte[] data)
        {
            for (int i = 0; i <= Length; i++)
            {
                Memory[bank][i] = data[i];
            }
        }

        public byte[][] ReadFile(string filename)
        {
            byte[][] bios = new byte[Banks][];
            try
            {
                FileStream file = new FileStream(filename, FileMode.Open, FileAccess.Read);
                for (int i = 0; i < Banks; i++)
                {
                    bios[i] = new byte[Length + 1];
                    for (int j = 0; j <= Length; j++)
                    {
                        bios[i][j] = new byte();
                        bios[i][j] = (byte)file.ReadByte();
                    }
                }
            }
            catch (Exception)
            {
                return null;
            }
            return bios;
        }

/// <summary>
/// Returns the byte at a given address without incrementing the cycle.  Useful for test harness.
/// </summary>
/// <param name="bank">The bank to read data from.</param>
/// <param name="address"></param>
/// <returns>the byte being returned</returns>
        public byte Read(int address)
        {
            return Memory[CurrentBank][address - Offset];
        }

/// <summary>
/// Writes data to the given address without incrementing the cycle.
/// </summary>
/// <param name="bank">The bank to load data to.</param>
/// <param name="address">The address to write data to</param>
/// <param name="data">The data to write</param>
        public void Write(int address, byte data)
        {
            _ = address;
            _ = data;
            return;
        }

/// <summary>
/// Dumps the entire memory object.  Used when saving the memory state
/// </summary>
/// <returns>2 dimensional array of data analogous to the ROM of the computer.</returns>
        public byte[][] DumpMemory()
        {
            return Memory;
        }

```

```
/// <summary>
/// Dumps the selected ROM bank.
/// </summary>
/// <param name="bank">The bank to dump data from.</param>
/// <returns>Array that represents the selected ROM bank.</returns>
        public byte[] DumpMemory(byte bank)
        {
            byte[] _tempMemory = new byte[MemoryMap.BankedRom.Length + 1];
            for (var i = 0; i < MemoryMap.BankedRom.Length; i++) {
                _tempMemory[i] = Memory[bank][i];
            }
            return _tempMemory;
        }

/// <summary>
/// Clears the ROM.
/// </summary>
        public void Clear()
        {
            for (byte i = 0; i < Banks; i++)
            {
                for (int j = 0; j < Length; j++)
                {
                    Memory[i][j] = 0x00;
                }
            }
        }
#endregion
    }
}
```

7.81 Hardware/Classes/AddressingMode.cs File Reference

Namespaces

- namespace Hardware

Enumerations

- enum Hardware.AddressingMode

  The addressing modes used by the 6502 Processor

7.82 AddressingMode.cs

Go to the documentation of this file.

```
namespace Hardware
{
/// <summary>
/// The addressing modes used by the 6502 Processor
/// </summary>
    public enum AddressingMode
    {
/// <summary>
/// In this mode a full address is given to operation on IE: Memory byte[] { 0x60, 0x00, 0xFF }
/// would perform an ADC operation and Add the value at ADDRESS 0xFF00 to the accumulator.
/// The address is always LSB first
/// </summary>
        Absolute = 1,
/// <summary>
/// In this mode a full address is given to operation on IE: Memory byte[] { 0x7D, 0x00, 0xFF } The
      full value would then be added to the X Register.
/// If the X register was 0x01 then the address would be 0xFF01.  and the value stored there would
      have an ADC operation performed on it and the value would
/// be added to the accumulator.
/// </summary>
        AbsoluteX = 2,
/// <summary>
/// In this mode a full address is given to operation on IE: Memory byte[] { 0x79, 0x00, 0xFF } The
      full value would then be added to the Y Register.
/// If the Y register was 0x01 then the address would be 0xFF01.  and the value stored there would
      have an ADC operation performed on it and the value would
/// be added to the accumulator
/// </summary>
        AbsoluteY = 3,
/// <summary>
/// In this mode the instruction operates on the accumulator.  No operands are needed.
/// </summary>
        Accumulator = 4,
/// <summary>
/// In this mode, the value to operate on immediately follows the instruction.  IE: Memory byte[] {
0x69, 0x01 }
/// would perform an ADC operation and Add 0x01 directly to the accumulator
/// </summary>
        Immediate = 5,
/// <summary>
/// No address is needed for this mode.  EX: BRK (Break), CLC (Clear Carry Flag) etc
/// </summary>
        Implied = 6,
/// <summary>
/// In this mode assume the following
/// Memory = { 0x61, 0x02, 0x04, 0x00, 0x03 }
/// RegisterX = 0x01
/// 1.  Take the sum of the X Register and the value after the opcode 0x01 + 0x01 = 0x02.
/// 2.  Starting at position 0x02 get an address (0x04,0x00) = 0x0004
/// 3.  Perform the ADC operation and Add the value at 0x0005 to the accumulator
/// Note:  if the Zero Page address is greater than 0xff then roll over the value.  IE 0x101 rolls
over to 0x01
/// </summary>
        IndirectX = 7,
/// <summary>
/// In this mode assume the following
/// Memory = { 0x61, 0x02, 0x04, 0x00, 0x03 }
/// RegisterY = 0x01
/// 1.  Starting at position 0x02 get an address (0x04,0x00) = 0x0004
/// 2.  Take the sum of the Y Register and the absolute address 0x01+0x0004 = 0x0005
/// 3.  Perform the ADC operation and Add the value at 0x0005 to the accumulator
/// Note:  if the address is great that 0xffff then roll over IE: 0x10001 rolls over to 0x01
/// </summary>
        IndirectY = 8,
/// <summary>
/// JMP is the only operation that uses this mode.  In this mode an absolute address is specified that
points to the location of the absolute address we want to jump to.
/// </summary>
        Indirect = 9,
/// <summary>
/// This Mode Changes the PC. It allows the program to change the location of the PC by 127 in either
direction.
/// </summary>
        Relative = 10,
/// <summary>
/// In this mode, a zero page address of the value to operate on is specified.  This mode can only
operation on values between 0x0 and 0xFF, or those that sit on the zero page of memory.  IE: Memory
byte[] { 0x69, 0x02, 0x01 }
/// would perform an ADC operation and Add 0x01 directly to the Accumulator
/// </summary>
        ZeroPage = 11,
/// <summary>
/// In this mode, a zero page address of the value to operate on is specified, however the value of
the X register is added to the address IE: Memory byte[] { 0x86, 0x02, 0x01, 0x67, 0x04, 0x01 }
/// In this example we store a value of 0x01 into the X register, then we would perform an ADC
operation using the addres of 0x04+0x01=0x05 and Add the result of 0x01 directly to the Accumulator
/// </summary>
        ZeroPageX = 12,
/// <summary>
/// This works the same as ZeroPageX except it uses the Y register instead of the X register.
/// </summary>
        ZeroPageY = 13,
    }
}
```

7.83 Hardware/Classes/Disassembly.cs File Reference

Classes

- class Hardware.Disassembly

  Used to help simulating. This class contains the disassembly properties.

Namespaces

- namespace Hardware

7.84 Disassembly.cs

Go to the documentation of this file.

```
using System;

namespace Hardware
{
/// <summary>
/// Used to help simulating.  This class contains the disassembly properties.
/// </summary>
    [Serializable]
    public class Disassembly
    {
/// <summary>
/// The low Address
/// </summary>
        public string LowAddress { get; set; }

/// <summary>
/// The High Address
/// </summary>
        public string HighAddress { get; set; }

/// <summary>
/// The string representation of the OpCode
/// </summary>
        public string OpCodeString { get; set; }

/// <summary>
/// The disassembly of the current step
/// </summary>
        public string DisassemblyOutput { get; set; }
    }
}
```

7.85 Hardware/Classes/MemoryMap.cs File Reference

Classes

- class Hardware.MemoryMap
- class Hardware.MemoryMap.BankedRam
- class Hardware.MemoryMap.DeviceArea
- class Hardware.MemoryMap.BankedRom
- class Hardware.MemoryMap.SharedRom
- class Hardware.MemoryMap.Devices
- class Hardware.MemoryMap.Devices.ACIA
- class Hardware.MemoryMap.Devices.GPIO
- class Hardware.MemoryMap.Devices.MM65SIB

Namespaces

- namespace Hardware

7.86 MemoryMap.cs

Go to the documentation of this file.

```
using System;
using System.IO;

namespace Hardware
{
    public class MemoryMap
    {
        public static class BankedRam
        {
            private static int _Offset = 0x0000;
            private static int _Length = 0x7FFF;

            public static int TotalLength = (BankSize * TotalBanks) - 1;
            public static int BankSize = (int)(Length + 1);
            public static byte TotalBanks = 16;

            public static int Offset { get { return _Offset; } }
            public static int Length { get { return _Length; } }
        }

        public static class DeviceArea
        {
            private static int _Offset = 0xD000;
            private static int _Length = 0x00FF;

/// <summary>
/// The end of memory
/// </summary>
            public static int End { get { return Offset + Length; } }
            public static int Offset { get { return _Offset; } }
            public static int Length { get { return _Length; } }
        }

        public static class BankedRom
        {
            private static int _Offset = 0x8000;
            private static int _Length = 0x3FFF;

            public static byte TotalBanks = 16;

            public static int Offset { get { return _Offset; } }
            public static int Length { get { return _Length; } }
        }

        public static class SharedRom
        {
            private static int _Offset = 0xE000;
            private static int _Length = 0x1FFF;

            public static byte TotalBanks = 1;

            public static int Offset { get { return _Offset; } }
            public static int Length { get { return _Length; } }
        }

        public static class Devices
        {
            public static class ACIA
            {
                public static int Length = 0x03;
                public static byte Offset = 0x10;
            }

            public static class GPIO
            {
                public static int Length = 0x0F;
                public static byte Offset = 0x20;
            }

            public static class MM65SIB
            {
                public static int Length = 0x0F;
                public static byte Offset = 0x30;
            }
        }

        public static readonly int Length = 0xFFFF;

        private static W65C02 Processor { get; set; }
        private static W65C22 GPIO { get; set; }
        private static W65C22 MM65SIB { get; set; }
        private static W65C51 ACIA { get; set; }
        private static AT28CXX SharedROM { get; set; }
        private static AT28CXX BankedROM { get; set; }
        private static HM62256 BankedRAM { get; set; }

        public static void Init(W65C02 processor, W65C22 gpio, W65C22 mm65sib, W65C51 acia, HM62256
bankedRam, AT28CXX bankedRom, AT28CXX sharedRom)
        {
            Processor = processor;
            GPIO = gpio;
            MM65SIB = mm65sib;
            ACIA = acia;
            SharedROM = sharedRom;
            BankedROM = bankedRom;
            BankedRAM = bankedRam;
        }
```

```

/// <summary>
/// Returns the byte at the given address.
/// </summary>
/// <param name="address">The address to return</param>
/// <returns>the byte being returned</returns>
        public static byte Read(int address)
        {
            var value = ReadWithoutCycle(address);
            Processor.IncrementCycleCount();
            return value;
        }

/// <summary>
/// Returns the byte at the given address without incrementing the cycle count.
/// </summary>
/// <param name="address">The address to return</param>
/// <returns>the byte being returned</returns>
        public static byte ReadWithoutCycle(int address)
        {
            int _address = address;
            if ((ACIA.Offset <= _address) && (_address <= (ACIA.Offset + ACIA.Length)))
            {
                return ACIA.Read(address);
            }
            else if ((GPIO.Offset <= _address) && (_address <= (GPIO.Offset + GPIO.Length)))
            {
                return GPIO.Read(_address);
            }
            else if ((DeviceArea.Offset <= _address) && (_address <= DeviceArea.End))
            {
                throw new ArgumentOutOfRangeException("Device area accessed where there is no device!");
            }
            else if ((SharedROM.Offset <= _address) && (_address <= SharedROM.End))
            {
                return SharedROM.Read(_address);
            }
            else if ((BankedROM.Offset <= _address) && (_address <= BankedROM.End))
            {
                return BankedROM.Read(_address);
            }
            else if ((BankedRAM.Offset <= _address) && (_address <= BankedRAM.End))
            {
                return BankedRAM.Read(_address);
            }
            else
            {
                return 0x00;
            }
        }

/// <summary>
/// Writes data to the given address.
/// </summary>
/// <param name="address">The address to write data to.</param>
/// <param name="data">The data to write.</param>
        public static void Write(int address, byte data)
        {
            Processor.IncrementCycleCount();
            WriteWithoutCycle(address, data);
        }

/// <summary>
/// Writes data to the given address without incrementing the cycle count.
/// </summary>
/// <param name="address">The address to write data to.</param>
/// <param name="data">The data to write.</param>
        public static void WriteWithoutCycle(int address, byte data)
        {
            if ((ACIA.Offset <= address) && (address <= (ACIA.Offset + ACIA.Length)))
            {
                ACIA.Write(address, data);
            }
            else if ((GPIO.Offset <= address) && (address <= (GPIO.Offset + GPIO.Length)))
            {
                GPIO.Write(address, data);
            }
            else if ((SharedROM.Offset <= address) && (address <= (SharedROM.Offset + SharedROM.Length)))
            {
                SharedROM.Write(address, data);
            }
            else if ((BankedROM.Offset <= address) && (address <= (BankedROM.Offset + BankedROM.Length)))
            {
                BankedROM.Write(address, data);
            }
            else if ((BankedRAM.Offset <= address) && (address <= (BankedRAM.Offset + BankedRAM.Length)))
            {
                BankedRAM.Write(address, data);
            }
            else
            {
                throw new ApplicationException(String.Format("Cannot write to address:  {0}", address));
            }
        }
    }
}
```

7.87 Hardware/Classes/Utility.cs File Reference

Classes

- class Hardware.Utility

Namespaces

- namespace Hardware

7.88 Utility.cs

Go to the documentation of this file.

```
using System.ComponentModel;

namespace Hardware
{
    public static class Utility
    {
        public static string ConvertOpCodeIntoString(this int i)
        {
            switch (i)
            {
                case 0x69:    //ăADCăImmediate
                case 0x65:    //ăADCăZeroăPage
                case 0x75:    //ăADCăZeroăPageăX
                case 0x6D:  //ăADCăAbsolute
                case 0x7D:  //ăADCăAbsoluteăX
                case 0x79:    //ăADCăAbsoluteăY
                case 0x61:    //ăADCăIndrectăX
                case 0x71:    //ăADCăIndirectăY
                    {
                        return "ADC";
                    }
                case 0x29:    //ăANDăImmediate
                case 0x25:    //ăANDăZeroăPage
                case 0x35:    //ăANDăZeroăPageăX
                case 0x2D:  //ăANDăAbsolute
                case 0x3D:  //ăANDăAbsoluteăX
                case 0x39:    //ăANDăAbsoluteăY
                case 0x21:    //ăANDăIndirectăX
                case 0x31:    //ăANDăIndirectăY
                    {
                        return "AND";
                    }
                case 0x0A:  //ăASLăAccumulator
                case 0x06:    //ăASLăZeroăPage
                case 0x16:    //ăASLăZeroăPageăX
                case 0x0E:  //ăASLăAbsolute
                case 0x1E:  //ăASLăAbsoluteăX
                    {
                        return "ASL";
                    }
                case 0x90:    //ăBCCăRelative
                    {
                        return "BCC";
                    }
                case 0xB0:    //ăBCSăRelative
```

```
                    {
                        return "BCS";
                    }
                case 0xF0:    //ăBEQăRelative
                    {
                        return "BEQ";
                    }
                case 0x24:    //ăBITăZeroăPage
                case 0x2C:  //ăBITăAbsolute
                    {
                        return "BIT";
                    }
                case 0x30:    //ăBMIăRelative
                    {
                        return "BMI";
                    }
                case 0xD0:    //ăBNEăRelative
                    {
                        return "BNE";
                    }
                case 0x10:    //ăBPLăRelative
                    {
                        return "BPL";
                    }
                case 0x00:    //ăBRKăImplied
                    {
                        return "BRK";
                    }
                case 0x50:  // BVC Relative
                    {
                        return "BCV";
                    }
                case 0x70:  //BVS Relative
                    {
                        return "BVS";
                    }
                case 0x18:    //ăCLCăImplied
                    {
                        return "CLC";
                    }
                case 0xD8:    //ăCLDăImplied
                    {
                        return "CLD";
                    }
                case 0x58:    //ăCLIăImplied
                    {
                        return "CLI";
                    }
                case 0xB8:    //ăCLVăImplied
                    {
                        return "CLV";
                    }
                case 0xC9:    //ăCMPăImmediate
                case 0xC5:    //ăCMPăZeroPage
                case 0xD5:    //ăCMPăZeroăPageăX
                case 0xCD:  //ăCMPăAbsolute
                case 0xDD:  //ăCMPăAbsoluteăX
                case 0xD9:    //ăCMPăAbsoluteăY
                case 0xC1:    //ăCMPăIndirectăX
                case 0xD1:    //ăCMPăIndirectăY
                    {
                        return "CMP";
                    }
                case 0xE0:    //ăCPXăImmediate
                case 0xE4:    //ăCPXăZeroPage
                case 0xEC:  //ăCPXăAbsolute
                    {
                        return "CPX";
                    }
                case 0xC0:    //ăCPYăImmediate
                case 0xC4:    //ăCPYăZeroPage
                case 0xCC:  //ăCPYăAbsolute
                    {
                        return "CPY";
                    }
                case 0xC6:    //ăDECăZeroăPage
                case 0xD6:    //ăDECăZeroăPageăX
                case 0xCE:  //ăDECăAbsolute
                case 0xDE:  //ăDECăAbsoluteăX
                    {
                        return "DEC";
                    }
                case 0xCA:  //ăDEXăImplied
                    {
                        return "DEX";
                    }
                case 0x88:    //ăDEYăImplied
```

```
                    {
                        return "DEY";
                    }
                case 0x49:    //ăEORăImmediate
                case 0x45:    //ăEORăZeroăPage
                case 0x55:    //ăEORăZeroăPageăX
                case 0x4D:  //ăEORăAbsolute
                case 0x5D:  //ăEORăAbsoluteăX
                case 0x59:    //ăEORăAbsoluteăY
                case 0x41:    //ăEORăIndrectăX
                case 0x51:    //ăEORăIndirectăY
                    {
                        return "EOR";
                    }
                case 0xE6:    //ăINCăZeroăPage
                case 0xF6:    //ăINCăZeroăPageăX
                    {
                        return "INC";
                    }
                case 0xE8:    //ăINXăImplied
                    {
                        return "INX";
                    }
                case 0xC8:    //ăINYăImplied
                    {
                        return "INY";
                    }
                case 0xEE:  //ăINCăAbsolute
                case 0xFE:  //ăINCăAbsoluteăX
                    {
                        return "INC";
                    }
                case 0x4C:  //ăJMPăAbsolute
                case 0x6C:  //ăJMPăIndirect
                    {
                        return "JMP";
                    }
                case 0x20:    //ăJSRăAbsolute
                    {
                        return "JSR";
                    }
                case 0xA9:    //ăLDAăImmediate
                case 0xA5:    //ăLDAăZeroăPage
                case 0xB5:    //ăLDAăZeroăPageăX
                case 0xAD:  //ăLDAăAbsolute
                case 0xBD:  //ăLDAăAbsoluteăX
                case 0xB9:    //ăLDAăAbsoluteăY
                case 0xA1:    //ăLDAăIndirectăX
                case 0xB1:    //ăLDAăIndirectăY
                    {
                        return "LDA";
                    }
                case 0xA2:    //ăLDXăImmediate
                case 0xA6:    //ăLDXăZeroăPage
                case 0xB6:    //ăLDXăZeroăPageăY
                case 0xAE:  //ăLDXăAbsolute
                case 0xBE:  //ăLDXăAbsoluteăY
                    {
                        return "LDX";
                    }
                case 0xA0:    //ăLDYăImmediate
                case 0xA4:    //ăLDYăZeroăPage
                case 0xB4:    //ăLDYăZeroăPageăY
                case 0xAC:  //ăLDYăAbsolute
                case 0xBC:  //ăLDYăAbsoluteăY
                    {
                        return "LDY";
                    }
                case 0x4A:  //ăLSRăAccumulator
                case 0x46:    //ăLSRăZeroăPage
                case 0x56:    //ăLSRăZeroăPageăX
                case 0x4E:  //ăLSRăAbsolute
                case 0x5E:  //ăLSRăAbsoluteăX
                    {
                        return "LSR";
                    }
                case 0xEA:  //ăNOPăImplied
                    {
                        return "NOP";
                    }
                case 0x09:    //ăORAăImmediate
                case 0x05:    //ăORAăZeroăPage
                case 0x15:    //ăORAăZeroăPageăX
                case 0x0D:  //ăORAăAbsolute
                case 0x1D:  //ăORAăAbsoluteăX
                case 0x19:    //ăORAăAbsoluteăY
                case 0x01:    //ăORAăIndirectăX
                case 0x11:    //ăORAăIndirectăY
                    {
                        return "ORA";
                    }
                case 0x48:    //ăPHAăImplied
                    {
                        return "PHA";
                    }
                case 0x08:    //ăPHPăImplied
                    {
                        return "PHP";
                    }
                case 0x68:    //ăPLAăImplied
                    {
                        return "PLA";
                    }
                case 0x28:    //ăPLPăImplied
                    {
                        return "PLP";
                    }
                case 0x2A:  //ăROLăAccumulator
                case 0x26:    //ăROLăZeroăPage
                case 0x36:    //ăROLăZeroăPageăX
                case 0x2E:  //ăROLăAbsolute
                case 0x3E:  //ăROLăAbsoluteăX
                    {
                        return "ROL";
                    }
                case 0x6A:  //ăRORăAccumulator
                case 0x66:    //ăRORăZeroăPage
                case 0x76:    //ăRORăZeroăPageăX
                case 0x6E:  //ăRORăAbsolute
                case 0x7E:  //ăRORăAbsoluteăX
                    {
                        return "ROR";
                    }
                case 0x40:    //ăRTIăImplied
                    {
                        return "RTI";
                    }
                case 0x60:    //ăRTSăImplied
                    {
                        return "RTS";
                    }
                case 0xE9:    //ăSBCăImmediate
                case 0xE5:    //ăSBCăZeroăPage
                case 0xF5:    //ăSBCăZeroăPageăX
                case 0xED:  //ăSBCăAbsolute
                case 0xFD:  //ăSBCăAbsoluteăX
                case 0xF9:    //ăSBCăAbsoluteăY
                case 0xE1:    //ăSBCăIndrectăX
                case 0xF1:    //ăSBCăIndirectăY
                    {
                        return "SBC";
                    }
                case 0x38:    //ăSECăImplied
                    {
                        return "SEC";
                    }
                case 0xF8:    //ăSEDăImplied
                    {
                        return "SED";
                    }
                case 0x78:    //ăSEIăImplied
                    {
                        return "SEI";
                    }
                case 0x85:    //ăSTAăZeroPage
                case 0x95:    //ăSTAăZeroăPageăX
                case 0x8D:  //ăSTAăAbsolute
                case 0x9D:  //ăSTAăAbsoluteăX
                case 0x99:    //ăSTAăAbsoluteăY
                case 0x81:    //ăSTAăIndirectăX
                case 0x91:    //ăSTAăIndirectăY
                    {
                        return "STA";
                    }
                case 0x86:    //ăSTXăZeroăPage
                case 0x96:    //ăSTXăZeroăPageăY
                case 0x8E:  //ăSTXăAbsolute
                    {
                        return "STX";
                    }
                case 0x84:    //ăSTYăZeroăPage
                case 0x94:    //ăSTYăZeroăPageăX
                case 0x8C:  //ăSTYăAbsolute
                    {
                        return "STY";
                    }
                case 0xAA:  //ăTAXăImplied
                    {
                        return "TAX";
                    }
                case 0xA8:    //ăTAYăImplied
                    {
                        return "TAY";
                    }
                case 0xBA:  //ăTSXăImplied
                    {
                        return "TSX";
                    }
                case 0x8A:  //ăTXAăImplied
                    {
                        return "TXA";
                    }
                case 0x9A:  //ăTXSăImplied
                    {
                        return "TXS";
                    }
                case 0x98:    //ăTYAăImplied
                    {
                        return "TYA";
                    }
                default:
                    throw new InvalidEnumArgumentException(string.Format("A Valid Conversion does not exist for OpCode {0}", i.ToString("X")));

            }
        }
    }
}
```

7.89 Hardware/HM62256.cs File Reference

Classes

- class Hardware.HM62256

Namespaces

- namespace Hardware

7.90 HM62256.cs

Go to the documentation of this file.

```
using System;

namespace Hardware
{
    public class HM62256
    {
/// <summary>
/// The memory area.
/// </summary>
        public byte[][] Memory { get; set; }

/// <summary>
/// The memory offset.
/// </summary>
        public int Offset { get; set; }

/// <summary>
/// The memory length.
/// </summary>
        public int Length { get; set; }

/// <summary>
/// The location of the end of memory.
/// </summary>
        public int End { get { return Offset + Length; } }

/// <summary>
/// The number of banks the memory has.
/// </summary>
        public byte Banks { get; set; }

/// <summary>
/// The currently selected bank.
/// </summary>
        public byte CurrentBank { get; set; }

/// <summary>
/// Called whenever a new 62256 object is required.
/// </summary>
/// <param name="banks">Number of banks the new memory will have.</param>
/// <param name="offset">Offset of the new memory in the address space.</param>
/// <param name="length">Length of each bank of memory.</param>
        public HM62256(byte banks, int offset, int length)
        {
            Memory = new byte[banks][];
            for (int i = 0; i < banks; i++)
            {
                Memory[i] = new byte[length + 1];
            }
            Length = length;
            Banks = banks;
            Offset = offset;
            CurrentBank = 0;
        }

/// <summary>
/// Called whenever the emulated computer is reset.
/// </summary>
        public void Reset()
        {
            Clear();
        }

/// <summary>
/// Clears the memory.
/// </summary>
        public void Clear()
        {
            for (var i = 0; i < Banks; i++)
            {
                for (var j = 0; j < Memory.Length; j++)
                {
                    Memory[i][j] = 0x00;
                }
            }
        }

/// <summary>
/// Returns the byte at a given address without incrementing the cycle.  Useful for test harness.
/// </summary>
/// <param name="bank">The bank to read data from.</param>
/// <param name="address"></param>
/// <returns>The byte being read.</returns>
        public byte Read(int address)
        {
            return Memory[CurrentBank][address - Offset];
        }

/// <summary>
/// Writes data to the given address without incrementing the cycle.
/// </summary>
/// <param name="bank">The bank to load data to.</param>
/// <param name="address">The address to write data to</param>
/// <param name="data">The data to write</param>
        public void Write(int address, byte data)
        {
            Memory[CurrentBank][address - Offset] = data;
        }

/// <summary>
/// Dumps the entire memory object.  Used when saving the memory state
/// </summary>
/// <returns>Jagged array representing the banked memory.</returns>
        public byte[][] DumpMemory()
        {
            return Memory;
        }
    }
}
```

7.91 Hardware/W65C02.cs File Reference

Classes

- class Hardware.W65C02

  An implementation of a W65C02 Processor.

Namespaces

- namespace Hardware

7.92 W65C02.cs

Go to the documentation of this file.

```
using NLog;
using System;
using System.ComponentModel;
using System.Diagnostics;
using System.Globalization;

namespace Hardware
{
/// <summary>
/// An implementation of a W65C02 Processor.
/// </summary>
    [Serializable]
    public class W65C02
    {
#region Fields
        private readonly ILogger _logger = LogManager.GetLogger("Processor");
        private int _programCounter;
        private int _stackPointer;
        private int _cycleCount;
        private bool _previousInterrupt;
        private bool _interrupt;

/// <summary>
/// Checks shether the emulated computer is running or not.
/// </summary>
        public bool isRunning;
#endregion

#region Properties
/// <summary>
/// The Accumulator.  This value is implemented as an integer intead of a byte.
/// This is done so we can detect wrapping of the value and set the correct number of cycles.
/// </summary>
        public int Accumulator { get; protected set; }

/// <summary>
/// The X Index Register
/// </summary>
        public int XRegister { get; private set; }

/// <summary>
/// The Y Index Register
/// </summary>
        public int YRegister { get; private set; }

/// <summary>
/// The Current Op Code being executed by the system
/// </summary>
        public int CurrentOpCode { get; private set; }

/// <summary>
/// The disassembly of the current operation.  This value is only set when the CPU is built in debug
mode.
/// </summary>
        public Disassembly CurrentDisassembly { get; private set; }

/// <summary>
/// Points to the Current Address of the instruction being executed by the system.
/// The PC wraps when the value is greater than 65535, or less than 0.
/// </summary>
        public int ProgramCounter
        {
            get { return _programCounter; }
            private set { _programCounter = WrapProgramCounter(value); }
        }

/// <summary>
/// Points to the Current Position of the Stack.
/// This value is a 00-FF value but is offset to point to the location in memory where the stack resides.
/// </summary>
        public int StackPointer
        {
            get { return _stackPointer; }
            private set
            {
                if (value > 0xFF)
                    _stackPointer = value - 0x100;
                else if (value < 0x00)
                    _stackPointer = value + 0x100;
                else
                    _stackPointer = value;
            }
        }

/// <summary>
/// An external action that occurs when the cycle count is incremented
/// </summary>
        public Action CycleCountIncrementedAction { get; set; }

        //Status Registers
/// <summary>
/// This is the carry flag.  when adding, if the result is greater than 255 or 99 in BCD Mode, then this bit is enabled.
/// In subtraction this is reversed and set to false if a borrow is required IE the result is less than 0
/// </summary>
        public bool CarryFlag { get; protected set; }

/// <summary>
/// Is true if one of the registers is set to zero.
/// </summary>
        public bool ZeroFlag { get; private set; }

/// <summary>
/// This determines if Interrupts are currently disabled.
/// This flag is turned on during a reset to prevent an interrupt from occuring during startup/Initialization.
/// If this flag is true, then the IRQ pin is ignored.
/// </summary>
        public bool DisableInterruptFlag { get; private set; }

/// <summary>
/// Binary Coded Decimal Mode is set/cleared via this flag.
/// when this mode is in effect, a byte represents a number from 0-99.
/// </summary>
        public bool DecimalFlag { get; private set; }

/// <summary>
/// This property is set when an overflow occurs.  An overflow happens if the high bit(7) changes during the operation.  Remember that values from 128-256 are negative values
/// as the high bit is set to 1.
/// Examples:
/// 64 + 64 = -128
/// -128 + -128 = 0
/// </summary>
        public bool OverflowFlag { get; protected set; }

/// <summary>
/// Set to true if the result of an operation is negative in ADC and SBC operations.
/// Remember that 128-256 represent negative numbers when doing signed math.
/// In shift operations the sign holds the carry.
/// </summary>
        public bool NegativeFlag { get; private set; }

/// <summary>
/// Set to true when an NMI should occur
/// </summary>
        public bool TriggerNmi { get; set; }

/// Set to true when an IRQ has occurred and is being processed by the CPU
        public bool TriggerIRQ { get; private set; }
#endregion

#region Public Methods
/// <summary>
/// Default Constructor, Instantiates a new instance of the processor.
/// </summary>
```

```
        public W65C02()
        {
            StackPointer = 0x100;
            CycleCountIncrementedAction = () => { };
        }

/// <summary>
/// Initializes the processor to its default state.
/// </summary>
        public void Reset()
        {
            ResetCycleCount();
            StackPointer = 0x1FD;
            //Set the Program Counter to the Reset Vector Address.
            ProgramCounter = 0xFFFC;
            //Reset the Program Counter to the Address contained in the Reset Vector
            ProgramCounter = (MemoryMap.Read(ProgramCounter) | (MemoryMap.Read(ProgramCounter + 1) «
      8));
            CurrentOpCode = MemoryMap.Read(ProgramCounter);
            //SetDisassembly();
            DisableInterruptFlag = true;
            _previousInterrupt = false;
            TriggerNmi = false;
            TriggerIRQ = false;
        }

/// <summary>
/// Performs the next step on the processor
/// </summary>
        public void NextStep()
        {
            SetDisassembly();

            //Have to read this first otherwise it causes tests to fail on a NES
            CurrentOpCode = MemoryMap.Read(ProgramCounter);

            ProgramCounter++;

            ExecuteOpCode();

            if (_previousInterrupt)
            {
                if (TriggerNmi)
                {
                    ProcessNMI();
                    TriggerNmi = false;
                }
                else if (TriggerIRQ)
                {
                    ProcessIRQ();
                    TriggerIRQ = false;
                }
            }
        }

/// <summary>
/// The InterruptRequest or IRQ
/// </summary>
        public void InterruptRequest()
        {
            TriggerIRQ = true;
        }

/// <summary>
/// Gets the Number of Cycles that have elapsed
/// </summary>
/// <returns>The number of elapsed cycles</returns>
        public int GetCycleCount()
        {
            return _cycleCount;
        }

/// <summary>
/// Increments the Cycle Count, causes a CycleCountIncrementedAction to fire.
/// </summary>
        public void IncrementCycleCount()
        {
            _cycleCount++;
            CycleCountIncrementedAction();

            _previousInterrupt = _interrupt;
            _interrupt = TriggerNmi || (TriggerIRQ && !DisableInterruptFlag);
        }

/// <summary>
/// Resets the Cycle Count back to 0
/// </summary>
```

```
        public void ResetCycleCount()
        {
            _cycleCount = 0;
        }
#endregion

#region Private Methods
/// <summary>
/// Executes an Opcode
/// </summary>
        private void ExecuteOpCode()
        {
            //The x+ cycles denotes that if a page wrap occurs, then an additional cycle is consumed.
            //The x++ cycles denotes that 1 cycle is added when a branch occurs and it on the same
page, and two cycles are added if its on a different page./
            //This is handled inside the GetValueFromMemory Method
            switch (CurrentOpCode)
            {
#region Add / Subtract Operations
                //ADC Add With Carry, Immediate, 2 Bytes, 2 Cycles
                case 0x69:
                    {
                        AddWithCarryOperation(AddressingMode.Immediate);
                        break;
                    }
                //ADC Add With Carry, Zero Page, 2 Bytes, 3 Cycles
                case 0x65:
                    {
                        AddWithCarryOperation(AddressingMode.ZeroPage);
                        break;
                    }
                //ADC Add With Carry, Zero Page X, 2 Bytes, 4 Cycles
                case 0x75:
                    {
                        AddWithCarryOperation(AddressingMode.ZeroPageX);
                        break;
                    }
                //ADC Add With Carry, Absolute, 3 Bytes, 4 Cycles
                case 0x6D:
                    {
                        AddWithCarryOperation(AddressingMode.Absolute);
                        break;
                    }
                //ADC Add With Carry, Absolute X, 3 Bytes, 4+ Cycles
                case 0x7D:
                    {
                        AddWithCarryOperation(AddressingMode.AbsoluteX);
                        break;
                    }
                //ADC Add With Carry, Absolute Y, 3 Bytes, 4+ Cycles
                case 0x79:
                    {
                        AddWithCarryOperation(AddressingMode.AbsoluteY);
                        break;
                    }
                //ADC Add With Carry, Indexed Indirect, 2 Bytes, 6 Cycles
                case 0x61:
                    {
                        AddWithCarryOperation(AddressingMode.IndirectX);
                        break;
                    }
                //ADC Add With Carry, Indexed Indirect, 2 Bytes, 5+ Cycles
                case 0x71:
                    {
                        AddWithCarryOperation(AddressingMode.IndirectY);
                        break;
                    }
                //SBC Subtract with Borrow, Immediate, 2 Bytes, 2 Cycles
                case 0xE9:
                    {
                        SubtractWithBorrowOperation(AddressingMode.Immediate);
                        break;
                    }
                //SBC Subtract with Borrow, Zero Page, 2 Bytes, 3 Cycles
                case 0xE5:
                    {
                        SubtractWithBorrowOperation(AddressingMode.ZeroPage);
                        break;
                    }
                //SBC Subtract with Borrow, Zero Page X, 2 Bytes, 4 Cycles
                case 0xF5:
                    {
                        SubtractWithBorrowOperation(AddressingMode.ZeroPageX);
                        break;
                    }
                //SBC Subtract with Borrow, Absolute, 3 Bytes, 4 Cycles
                case 0xED:
                    {
                        SubtractWithBorrowOperation(AddressingMode.Absolute);
                        break;
                    }
                //SBC Subtract with Borrow, Absolute X, 3 Bytes, 4+ Cycles
                case 0xFD:
                    {
                        SubtractWithBorrowOperation(AddressingMode.AbsoluteX);
                        break;
                    }
                //SBC Subtract with Borrow, Absolute Y, 3 Bytes, 4+ Cycles
                case 0xF9:
                    {
                        SubtractWithBorrowOperation(AddressingMode.AbsoluteY);
                        break;
                    }
                //SBC Subtract with Borrow, Indexed Indirect, 2 Bytes, 6 Cycles
                case 0xE1:
                    {
                        SubtractWithBorrowOperation(AddressingMode.IndirectX);
                        break;
                    }
                //SBC Subtract with Borrow, Indexed Indirect, 2 Bytes, 5+ Cycles
                case 0xF1:
                    {
                        SubtractWithBorrowOperation(AddressingMode.IndirectY);
                        break;
                    }
#endregion

#region Branch Operations
                //BCC Branch if Carry is Clear, Relative, 2 Bytes, 2++ Cycles
                case 0x90:
                    {
                        BranchOperation(!CarryFlag);
                        break;

                    }
                //BCS Branch if Carry is Set, Relative, 2 Bytes, 2++ Cycles
                case 0xB0:
                    {
                        BranchOperation(CarryFlag);
                        break;
                    }
                //BEQ Branch if Zero is Set, Relative, 2 Bytes, 2++ Cycles
                case 0xF0:
                    {
                        BranchOperation(ZeroFlag);
                        break;
                    }

                // BMI Branch if Negative Set
                case 0x30:
                    {
                        BranchOperation(NegativeFlag);
                        break;
                    }
                //BNE Branch if Zero is Not Set, Relative, 2 Bytes, 2++ Cycles
                case 0xD0:
                    {
                        BranchOperation(!ZeroFlag);
                        break;
                    }
                // BPL Branch if Negative Clear, 2 Bytes, 2++ Cycles
                case 0x10:
                    {
                        BranchOperation(!NegativeFlag);
                        break;
                    }
                // BVC Branch if Overflow Clear, 2 Bytes, 2++ Cycles
                case 0x50:
                    {
                        BranchOperation(!OverflowFlag);
                        break;
                    }
                // BVS Branch if Overflow Set, 2 Bytes, 2++ Cycles
                case 0x70:
                    {
                        BranchOperation(OverflowFlag);
                        break;
                    }
#endregion

#region BitWise Comparison Operations
                //AND Compare Memory with Accumulator, Immediate, 2 Bytes, 2 Cycles
                case 0x29:
                    {
                        AndOperation(AddressingMode.Immediate);
                        break;
                    }
                //AND Compare Memory with Accumulator, Zero Page, 2 Bytes, 3 Cycles
                case 0x25:
                    {
                        AndOperation(AddressingMode.ZeroPage);
                        break;
                    }
                //AND Compare Memory with Accumulator, Zero PageX, 2 Bytes, 3 Cycles
                case 0x35:
                    {
                        AndOperation(AddressingMode.ZeroPageX);
                        break;
                    }
                //AND Compare Memory with Accumulator, Absolute,  3 Bytes, 4 Cycles
                case 0x2D:
                    {
                        AndOperation(AddressingMode.Absolute);
                        break;
                    }
                //AND Compare Memory with Accumulator, AbsolueteX 3 Bytes, 4+ Cycles
                case 0x3D:
                    {
                        AndOperation(AddressingMode.AbsoluteX);
                        break;
                    }
                //AND Compare Memory with Accumulator, AbsoluteY, 3 Bytes, 4+ Cycles
                case 0x39:
                    {
                        AndOperation(AddressingMode.AbsoluteY);
                        break;
                    }
                //AND Compare Memory with Accumulator, IndexedIndirect, 2 Bytes, 6 Cycles
                case 0x21:
                    {
                        AndOperation(AddressingMode.IndirectX);
                        break;
                    }
                //AND Compare Memory with Accumulator, IndirectIndexed, 2 Bytes, 5 Cycles
                case 0x31:
                    {
                        AndOperation(AddressingMode.IndirectY);
                        break;
                    }
                //BIT Compare Memory with Accumulator, Zero Page, 2 Bytes, 3 Cycles
                case 0x24:
                    {
                        BitOperation(AddressingMode.ZeroPage);
                        break;
                    }
                //BIT Compare Memory with Accumulator, Absolute, 2 Bytes, 4 Cycles
                case 0x2C:
                    {
                        BitOperation(AddressingMode.Absolute);
                        break;
                    }
                //EOR Exclusive OR Memory with Accumulator, Immediate, 2 Bytes, 2 Cycles
                case 0x49:
                    {
                        EorOperation(AddressingMode.Immediate);
                        break;
                    }
                //EOR Exclusive OR Memory with Accumulator, Zero Page, 2 Bytes, 3 Cycles
                case 0x45:
                    {
                        EorOperation(AddressingMode.ZeroPage);
                        break;
                    }
                //EOR Exclusive OR Memory with Accumulator, Zero Page X, 2 Bytes, 4 Cycles
                case 0x55:
                    {
                        EorOperation(AddressingMode.ZeroPageX);
                        break;
                    }
                //EOR Exclusive OR Memory with Accumulator, Absolute, 3 Bytes, 4 Cycles
                case 0x4D:
                    {
                        EorOperation(AddressingMode.Absolute);
                        break;
                    }
                //EOR Exclusive OR Memory with Accumulator, Absolute X, 3 Bytes, 4+ Cycles
                case 0x5D:
                    {
                        EorOperation(AddressingMode.AbsoluteX);
                        break;
                    }
```

```
                //EOR Exclusive OR Memory with Accumulator, Absolute Y, 3 Bytes, 4+ Cycles
                case 0x59:
                    {
                        EorOperation(AddressingMode.AbsoluteY);
                        break;
                    }
                //EOR Exclusive OR Memory with Accumulator, IndexedIndirect, 2 Bytes 6 Cycles
                case 0x41:
                    {
                        EorOperation(AddressingMode.IndirectX);
                        break;
                    }
                //EOR Exclusive OR Memory with Accumulator, IndirectIndexed, 2 Bytes 5 Cycles
                case 0x51:
                    {
                        EorOperation(AddressingMode.IndirectY);
                        break;
                    }
                //ORA Compare Memory with Accumulator, Immediate, 2 Bytes, 2 Cycles
                case 0x09:
                    {
                        OrOperation(AddressingMode.Immediate);
                        break;
                    }
                //ORA Compare Memory with Accumulator, Zero Page, 2 Bytes, 2 Cycles
                case 0x05:
                    {
                        OrOperation(AddressingMode.ZeroPage);
                        break;
                    }
                //ORA Compare Memory with Accumulator, Zero PageX, 2 Bytes, 4 Cycles
                case 0x15:
                    {
                        OrOperation(AddressingMode.ZeroPageX);
                        break;
                    }
                //ORA Compare Memory with Accumulator, Absolute,  3 Bytes, 4 Cycles
                case 0x0D:
                    {
                        OrOperation(AddressingMode.Absolute);
                        break;
                    }
                //ORA Compare Memory with Accumulator, AbsolueteX 3 Bytes, 4+ Cycles
                case 0x1D:
                    {
                        OrOperation(AddressingMode.AbsoluteX);
                        break;
                    }
                //ORA Compare Memory with Accumulator, AbsoluteY, 3 Bytes, 4+ Cycles
                case 0x19:
                    {
                        OrOperation(AddressingMode.AbsoluteY);
                        break;
                    }
                //ORA Compare Memory with Accumulator, IndexedIndirect, 2 Bytes, 6 Cycles
                case 0x01:
                    {
                        OrOperation(AddressingMode.IndirectX);
                        break;
                    }
                //ORA Compare Memory with Accumulator, IndirectIndexed, 2 Bytes, 5 Cycles
                case 0x11:
                    {
                        OrOperation(AddressingMode.IndirectY);
                        break;
                    }
#endregion

#region Clear Flag Operations
                //CLC Clear Carry Flag, Implied, 1 Byte, 2 Cycles
                case 0x18:
                    {
                        CarryFlag = false;
                        IncrementCycleCount();
                        break;
                    }
                //CLD Clear Decimal Flag, Implied, 1 Byte, 2 Cycles
                case 0xD8:
                    {
                        DecimalFlag = false;
                        IncrementCycleCount();
                        break;

                    }
                //CLI Clear Interrupt Flag, Implied, 1 Byte, 2 Cycles
                case 0x58:
                    {
                        DisableInterruptFlag = false;
                        IncrementCycleCount();
                        break;

                    }
                //CLV Clear Overflow Flag, Implied, 1 Byte, 2 Cycles
                case 0xB8:
                    {
                        OverflowFlag = false;
                        IncrementCycleCount();
                        break;
                    }

#endregion

#region Compare Operations
                //CMP Compare Accumulator with Memory, Immediate, 2 Bytes, 2 Cycles
                case 0xC9:
                    {
                        CompareOperation(AddressingMode.Immediate, Accumulator);
                        break;
                    }
                //CMP Compare Accumulator with Memory, Zero Page, 2 Bytes, 3 Cycles
                case 0xC5:
                    {
                        CompareOperation(AddressingMode.ZeroPage, Accumulator);
                        break;
                    }
                //CMP Compare Accumulator with Memory, Zero Page x, 2 Bytes, 4 Cycles
                case 0xD5:
                    {
                        CompareOperation(AddressingMode.ZeroPageX, Accumulator);
                        break;
                    }
                //CMP Compare Accumulator with Memory, Absolute, 3 Bytes, 4 Cycles
                case 0xCD:
                    {
                        CompareOperation(AddressingMode.Absolute, Accumulator);
                        break;
                    }
                //CMP Compare Accumulator with Memory, Absolute X, 2 Bytes, 4 Cycles
                case 0xDD:
                    {
                        CompareOperation(AddressingMode.AbsoluteX, Accumulator);
                        break;
                    }
                //CMP Compare Accumulator with Memory, Absolute Y, 2 Bytes, 4 Cycles
                case 0xD9:
                    {
                        CompareOperation(AddressingMode.AbsoluteY, Accumulator);
                        break;
                    }
                //CMP Compare Accumulator with Memory, Indirect X, 2 Bytes, 6 Cycles
                case 0xC1:
                    {
                        CompareOperation(AddressingMode.IndirectX, Accumulator);
                        break;
                    }
                //CMP Compare Accumulator with Memory, Indirect Y, 2 Bytes, 5 Cycles
                case 0xD1:
                    {
                        CompareOperation(AddressingMode.IndirectY, Accumulator);
                        break;
                    }
                //CPX Compare Accumulator with X Register, Immediate, 2 Bytes, 2 Cycles
                case 0xE0:
                    {
                        CompareOperation(AddressingMode.Immediate, XRegister);
                        break;
                    }
                //CPX Compare Accumulator with X Register, Zero Page, 2 Bytes, 3 Cycles
                case 0xE4:
                    {
                        CompareOperation(AddressingMode.ZeroPage, XRegister);
                        break;
                    }
                //CPX Compare Accumulator with X Register, Absolute, 3 Bytes, 4 Cycles
                case 0xEC:
                    {
                        CompareOperation(AddressingMode.Absolute, XRegister);
                        break;
                    }
                //CPY Compare Accumulator with Y Register, Immediate, 2 Bytes, 2 Cycles
                case 0xC0:
                    {
                        CompareOperation(AddressingMode.Immediate, YRegister);
                        break;
                    }
                //CPY Compare Accumulator with Y Register, Zero Page, 2 Bytes, 3 Cycles
                case 0xC4:
                    {
                        CompareOperation(AddressingMode.ZeroPage, YRegister);
                        break;
                    }
                //CPY Compare Accumulator with Y Register, Absolute, 3 Bytes, 4 Cycles
                case 0xCC:
                    {
                        CompareOperation(AddressingMode.Absolute, YRegister);
                        break;
                    }
#endregion

#region Increment/Decrement Operations
                //DEC Decrement Memory by One, Zero Page, 2 Bytes, 5 Cycles
                case 0xC6:
                    {
                        ChangeMemoryByOne(AddressingMode.ZeroPage, true);
                        break;
                    }
                //DEC Decrement Memory by One, Zero Page X, 2 Bytes, 6 Cycles
                case 0xD6:
                    {
                        ChangeMemoryByOne(AddressingMode.ZeroPageX, true);
                        break;
                    }
                //DEC Decrement Memory by One, Absolute, 3 Bytes, 6 Cycles
                case 0xCE:
                    {
                        ChangeMemoryByOne(AddressingMode.Absolute, true);
                        break;
                    }
                //DEC Decrement Memory by One, Absolute X, 3 Bytes, 7 Cycles
                case 0xDE:
                    {
                        ChangeMemoryByOne(AddressingMode.AbsoluteX, true);
                        IncrementCycleCount();
                        break;
                    }
                //DEX Decrement X Register by One, Implied, 1 Bytes, 2 Cycles
                case 0xCA:
                    {
                        ChangeRegisterByOne(true, true);
                        break;
                    }
                //DEY Decrement Y Register by One, Implied, 1 Bytes, 2 Cycles
                case 0x88:
                    {
                        ChangeRegisterByOne(false, true);
                        break;
                    }
                //INC Increment Memory by One, Zero Page, 2 Bytes, 5 Cycles
                case 0xE6:
                    {
                        ChangeMemoryByOne(AddressingMode.ZeroPage, false);
                        break;
                    }
                //INC Increment Memory by One, Zero Page X, 2 Bytes, 6 Cycles
                case 0xF6:
                    {
                        ChangeMemoryByOne(AddressingMode.ZeroPageX, false);
                        break;
                    }
                //INC Increment Memory by One, Absolute, 3 Bytes, 6 Cycles
                case 0xEE:
                    {
                        ChangeMemoryByOne(AddressingMode.Absolute, false);
                        break;
                    }
                //INC Increment Memory by One, Absolute X, 3 Bytes, 7 Cycles
                case 0xFE:
                    {
                        ChangeMemoryByOne(AddressingMode.AbsoluteX, false);
                        IncrementCycleCount();
                        break;
                    }
                //INX Increment X Register by One, Implied, 1 Bytes, 2 Cycles
                case 0xE8:
                    {
                        ChangeRegisterByOne(true, false);
                        break;
                    }
                //INY Increment Y Register by One, Implied, 1 Bytes, 2 Cycles
                case 0xC8:
                    {
                        ChangeRegisterByOne(false, false);
                        break;
                    }
#endregion

#region GOTO and GOSUB Operations
                //JMP Jump to New Location, Absolute 3 Bytes, 3 Cycles
                case 0x4C:
                    {
                        ProgramCounter = GetAddressByAddressingMode(AddressingMode.Absolute);
                        break;
                    }
                //JMP Jump to New Location, Indirect 3 Bytes, 5 Cycles
                case 0x6C:
                    {
                        ProgramCounter = GetAddressByAddressingMode(AddressingMode.Absolute);

                        if ((ProgramCounter & 0xFF) == 0xFF)
                        {
                            //Get the first half of the address
                            int address = MemoryMap.Read(ProgramCounter);

                            //Get the second half of the address, due to the issue with page boundary
it reads from the wrong location!
                            address += 256 * MemoryMap.Read(ProgramCounter - 255);
                            ProgramCounter = address;
                        }
                        else
                        {
                            ProgramCounter = GetAddressByAddressingMode(AddressingMode.Absolute);
                        }

                        break;
                    }
                //JSR Jump to SubRoutine, Absolute, 3 Bytes, 6 Cycles
                case 0x20:
                    {
                        JumpToSubRoutineOperation();
                        break;
                    }
                //BRK Simulate IRQ, Implied, 1 Byte, 7 Cycles
                case 0x00:
                    {
                        BreakOperation(true, 0xFFFE);
                        break;
                    }
                //RTI Return From Interrupt, Implied, 1 Byte, 6 Cycles
                case 0x40:
                    {
                        ReturnFromInterruptOperation();
                        break;
                    }
                //RTS Return From Subroutine, Implied, 1 Byte, 6 Cycles
                case 0x60:
                    {
                        ReturnFromSubRoutineOperation();
                        break;
                    }
#endregion

#region Load Value From Memory Operations
                //LDA Load Accumulator with Memory, Immediate, 2 Bytes, 2 Cycles
                case 0xA9:
                    {
                        Accumulator =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.Immediate));
                        SetZeroFlag(Accumulator);
                        SetNegativeFlag(Accumulator);
                        break;
                    }
                //LDA Load Accumulator with Memory, Zero Page, 2 Bytes, 3 Cycles
                case 0xA5:
                    {
                        Accumulator =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.ZeroPage));
                        SetZeroFlag(Accumulator);
                        SetNegativeFlag(Accumulator);
                        break;
                    }
                //LDA Load Accumulator with Memory, Zero Page X, 2 Bytes, 4 Cycles
                case 0xB5:
                    {
                        Accumulator =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.ZeroPageX));
                        SetZeroFlag(Accumulator);
                        SetNegativeFlag(Accumulator);
                        break;
                    }
                //LDA Load Accumulator with Memory, Absolute, 3 Bytes, 4 Cycles
                case 0xAD:
                    {
                        Accumulator = MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.Absolute));
                        SetZeroFlag(Accumulator);
                        SetNegativeFlag(Accumulator);
                        break;
                    }
                //LDA Load Accumulator with Memory, Absolute X, 3 Bytes, 4+ Cycles
                case 0xBD:
                    {
                        Accumulator = MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.AbsoluteX));
                        SetZeroFlag(Accumulator);
                        SetNegativeFlag(Accumulator);
                        break;
                    }
                //LDA Load Accumulator with Memory, Absolute Y, 3 Bytes, 4+ Cycles
                case 0xB9:
                    {
                        Accumulator = MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.AbsoluteY));
                        SetZeroFlag(Accumulator);
                        SetNegativeFlag(Accumulator);
                        break;
                    }
                //LDA Load Accumulator with Memory, Index Indirect, 2 Bytes, 6 Cycles
                case 0xA1:
                    {
                        Accumulator = MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.IndirectX));
                        SetZeroFlag(Accumulator);
                        SetNegativeFlag(Accumulator);
                        break;
                    }
                //LDA Load Accumulator with Memory, Indirect Index, 2 Bytes, 5+ Cycles
                case 0xB1:
                    {
                        Accumulator = MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.IndirectY));
                        SetZeroFlag(Accumulator);
                        SetNegativeFlag(Accumulator);
                        break;
                    }
                //LDX Load X with memory, Immediate, 2 Bytes, 2 Cycles
                case 0xA2:
                    {
                        XRegister = MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.Immediate));
                        SetZeroFlag(XRegister);
                        SetNegativeFlag(XRegister);
                        break;
                    }
                //LDX Load X with memory, Zero Page, 2 Bytes, 3 Cycles
                case 0xA6:
                    {
                        XRegister = MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.ZeroPage));
                        SetZeroFlag(XRegister);
                        SetNegativeFlag(XRegister);
                        break;
                    }
                //LDX Load X with memory, Zero Page Y, 2 Bytes, 4 Cycles
                case 0xB6:
                    {
                        XRegister = MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.ZeroPageY));
                        SetZeroFlag(XRegister);
                        SetNegativeFlag(XRegister);
                        break;
                    }
                //LDX Load X with memory, Absolute, 3 Bytes, 4 Cycles
                case 0xAE:
                    {
                        XRegister = MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.Absolute));
                        SetZeroFlag(XRegister);
                        SetNegativeFlag(XRegister);
                        break;
                    }
                //LDX Load X with memory, Absolute Y, 3 Bytes, 4+ Cycles
                case 0xBE:
                    {
                        XRegister = MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.AbsoluteY));
                        SetZeroFlag(XRegister);
                        SetNegativeFlag(XRegister);
                        break;
                    }
                //LDY Load Y with memory, Immediate, 2 Bytes, 2 Cycles
                case 0xA0:
                    {
                        YRegister = MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.Immediate));
                        SetZeroFlag(YRegister);
                        SetNegativeFlag(YRegister);
                        break;
                    }
                //LDY Load Y with memory, Zero Page, 2 Bytes, 3 Cycles
                case 0xA4:
                    {
                        YRegister = MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.ZeroPage));
                        SetZeroFlag(YRegister);
                        SetNegativeFlag(YRegister);
                        break;
                    }
                //LDY Load Y with memory, Zero Page X, 2 Bytes, 4 Cycles
                case 0xB4:
                    {
                        YRegister = MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.ZeroPageX));
                        SetZeroFlag(YRegister);
                        SetNegativeFlag(YRegister);
                        break;
                    }
                //LDY Load Y with memory, Absolute, 3 Bytes, 4 Cycles
                case 0xAC:
                    {
                        YRegister = MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.Absolute));
                        SetZeroFlag(YRegister);
                        SetNegativeFlag(YRegister);
                        break;
                    }
                //LDY Load Y with memory, Absolue X, 3 Bytes, 4+ Cycles
                case 0xBC:
                    {
                        YRegister = MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.AbsoluteX));
                        SetZeroFlag(YRegister);
                        SetNegativeFlag(YRegister);
                        break;
                    }
#endregion

#region Push/Pull Stack
                //PHA Push Accumulator onto Stack, Implied, 1 Byte, 3 Cycles
                case 0x48:
                    {
                        MemoryMap.Read(ProgramCounter + 1);

                        PokeStack((byte)Accumulator);
                        StackPointer--;
                        IncrementCycleCount();
                        break;

                    }
                //PHP Push Flags onto Stack, Implied, 1 Byte, 3 Cycles
                case 0x08:
                    {
                        MemoryMap.Read(ProgramCounter + 1);

                        PushFlagsOperation();
                        StackPointer--;
                        IncrementCycleCount();
                        break;
                    }
                //PLA Pull Accumulator from Stack, Implied, 1 Byte, 4 Cycles
                case 0x68:
                    {
                        MemoryMap.Read(ProgramCounter + 1);
                        StackPointer++;
                        IncrementCycleCount();

                        Accumulator = PeekStack();
                        SetNegativeFlag(Accumulator);
                        SetZeroFlag(Accumulator);

                        IncrementCycleCount();
                        break;
                    }
                //PLP Pull Flags from Stack, Implied, 1 Byte, 4 Cycles
                case 0x28:
                    {
                        MemoryMap.Read(ProgramCounter + 1);

                        StackPointer++;
                        IncrementCycleCount();

                        PullFlagsOperation();

                        IncrementCycleCount();
                        break;
                    }
                //TSX Transfer Stack Pointer to X Register, 1 Bytes, 2 Cycles
                case 0xBA:
                    {
                        XRegister = StackPointer;

                        SetNegativeFlag(XRegister);
                        SetZeroFlag(XRegister);
                        IncrementCycleCount();
                        break;
                    }
                //TXS Transfer X Register to Stack Pointer, 1 Bytes, 2 Cycles
                case 0x9A:
                    {
                        StackPointer = (byte)XRegister;
                        IncrementCycleCount();
                        break;
                    }
#endregion

#region Set Flag Operations
                //SEC Set Carry, Implied, 1 Bytes, 2 Cycles
                case 0x38:
                    {
                        CarryFlag = true;
                        IncrementCycleCount();
                        break;
                    }
                //SED Set Interrupt, Implied, 1 Bytes, 2 Cycles
                case 0xF8:
                    {
                        DecimalFlag = true;
                        IncrementCycleCount();
                        break;
                    }
                //SEI Set Interrupt, Implied, 1 Bytes, 2 Cycles
                case 0x78:
                    {
                        DisableInterruptFlag = true;
                        IncrementCycleCount();
                        break;
                    }
#endregion

#region Shift/Rotate Operations
                //ASL Shift Left 1 Bit Memory or Accumulator, Accumulator, 1 Bytes, 2 Cycles
                case 0x0A:
                    {
                        AslOperation(AddressingMode.Accumulator);
                        break;
                    }
                //ASL Shift Left 1 Bit Memory or Accumulator, Zero Page, 2 Bytes, 5 Cycles
                case 0x06:
                    {
                        AslOperation(AddressingMode.ZeroPage);
                        break;
                    }
                //ASL Shift Left 1 Bit Memory or Accumulator, Zero PageX, 2 Bytes, 6 Cycles
                case 0x16:
                    {
                        AslOperation(AddressingMode.ZeroPageX);
                        break;
                    }
                //ASL Shift Left 1 Bit Memory or Accumulator, Absolute, 3 Bytes, 6 Cycles
                case 0x0E:
                    {
                        AslOperation(AddressingMode.Absolute);
                        break;
                    }
                //ASL Shift Left 1 Bit Memory or Accumulator, AbsoluteX, 3 Bytes, 7 Cycles
                case 0x1E:
                    {
                        AslOperation(AddressingMode.AbsoluteX);
                        IncrementCycleCount();
                        break;
                    }
                //LSR Shift Left 1 Bit Memory or Accumulator, Accumulator, 1 Bytes, 2 Cycles
                case 0x4A:
                    {
                        LsrOperation(AddressingMode.Accumulator);
                        break;
                    }
                //LSR Shift Left 1 Bit Memory or Accumulator, Zero Page, 2 Bytes, 5 Cycles
                case 0x46:
                    {
                        LsrOperation(AddressingMode.ZeroPage);
                        break;
                    }
                //LSR Shift Left 1 Bit Memory or Accumulator, Zero PageX, 2 Bytes, 6 Cycles
                case 0x56:
                    {
                        LsrOperation(AddressingMode.ZeroPageX);
                        break;
                    }
                //LSR Shift Left 1 Bit Memory or Accumulator, Absolute, 3 Bytes, 6 Cycles
                case 0x4E:
                    {
                        LsrOperation(AddressingMode.Absolute);
                        break;
                    }
                //LSR Shift Left 1 Bit Memory or Accumulator, AbsoluteX, 3 Bytes, 7 Cycles
                case 0x5E:
                    {
                        LsrOperation(AddressingMode.AbsoluteX);
                        IncrementCycleCount();
                        break;
                    }
                //ROL Rotate Left 1 Bit Memory or Accumulator, Accumulator, 1 Bytes, 2 Cycles
                case 0x2A:
                    {
                        RolOperation(AddressingMode.Accumulator);
                        break;
                    }
                //ROL Rotate Left 1 Bit Memory or Accumulator, Zero Page, 2 Bytes, 5 Cycles
                case 0x26:
                    {
                        RolOperation(AddressingMode.ZeroPage);
                        break;
                    }
                //ROL Rotate Left 1 Bit Memory or Accumulator, Zero PageX, 2 Bytes, 6 Cycles
                case 0x36:
                    {
                        RolOperation(AddressingMode.ZeroPageX);
                        break;
                    }
                //ROL Rotate Left 1 Bit Memory or Accumulator, Absolute, 3 Bytes, 6 Cycles
                case 0x2E:
                    {
                        RolOperation(AddressingMode.Absolute);
                        break;
                    }
                //ROL Rotate Left 1 Bit Memory or Accumulator, AbsoluteX, 3 Bytes, 7 Cycles
                case 0x3E:
                    {
                        RolOperation(AddressingMode.AbsoluteX);
                        IncrementCycleCount();
                        break;
                    }
                //ROR Rotate Right 1 Bit Memory or Accumulator, Accumulator, 1 Bytes, 2 Cycles
                case 0x6A:
                    {
                        RorOperation(AddressingMode.Accumulator);
                        break;
                    }
                //ROR Rotate Right 1 Bit Memory or Accumulator, Zero Page, 2 Bytes, 5 Cycles
                case 0x66:
                    {
                        RorOperation(AddressingMode.ZeroPage);
                        break;
                    }
                //ROR Rotate Right 1 Bit Memory or Accumulator, Zero PageX, 2 Bytes, 6 Cycles
                case 0x76:
                    {
                        RorOperation(AddressingMode.ZeroPageX);
                        break;
                    }
                //ROR Rotate Right 1 Bit Memory or Accumulator, Absolute, 3 Bytes, 6 Cycles
                case 0x6E:
                    {
                        RorOperation(AddressingMode.Absolute);
                        break;
                    }
                //ROR Rotate Right 1 Bit Memory or Accumulator, AbsoluteX, 3 Bytes, 7 Cycles
                case 0x7E:
                    {
                        RorOperation(AddressingMode.AbsoluteX);
                        IncrementCycleCount();
                        break;
                    }
#endregion

#region Store Value In Memory Operations
                //STA Store Accumulator In Memory, Zero Page, 2 Bytes, 3 Cycles
                case 0x85:
                    {
                        MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.ZeroPage),
(byte)Accumulator);
                        break;
                    }
                //STA Store Accumulator In Memory, Zero Page X, 2 Bytes, 4 Cycles
                case 0x95:
                    {
                        MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.ZeroPageX),
(byte)Accumulator);
                        break;
                    }
                //STA Store Accumulator In Memory, Absolute, 3 Bytes, 4 Cycles
                case 0x8D:
                    {
                        MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.Absolute),
(byte)Accumulator);
                        break;
                    }
                //STA Store Accumulator In Memory, Absolute X, 3 Bytes, 5 Cycles
                case 0x9D:
                    {
                        MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.AbsoluteX),
(byte)Accumulator);
                        IncrementCycleCount();
                        break;
                    }
                //STA Store Accumulator In Memory, Absolute Y, 3 Bytes, 5 Cycles
                case 0x99:
                    {
                        MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.AbsoluteY),
(byte)Accumulator);
                        IncrementCycleCount();
                        break;
                    }
                //STA Store Accumulator In Memory, Indexed Indirect, 2 Bytes, 6 Cycles
                case 0x81:
                    {
                        MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.IndirectX),
(byte)Accumulator);
                        break;
                    }
                //STA Store Accumulator In Memory, Indirect Indexed, 2 Bytes, 6 Cycles
                case 0x91:
                    {
                        MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.IndirectY),
(byte)Accumulator);
                        IncrementCycleCount();
                        break;
                    }
                //STX Store Index X, Zero Page, 2 Bytes, 3 Cycles
                case 0x86:
                    {
                        MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.ZeroPage),
(byte)XRegister);
                        break;
                    }
                //STX Store Index X, Zero Page Y, 2 Bytes, 4 Cycles
                case 0x96:
                    {
                        MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.ZeroPageY),
(byte)XRegister);
                        break;
                    }
                //STX Store Index X, Absolute, 3 Bytes, 4 Cycles
                case 0x8E:
                    {
                        MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.Absolute),
(byte)XRegister);
                        break;
                    }
                //STY Store Index Y, Zero Page, 2 Bytes, 3 Cycles
                case 0x84:
                    {
                        MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.ZeroPage),
(byte)YRegister);
                        break;
                    }
                //STY Store Index Y, Zero Page X, 2 Bytes, 4 Cycles
                case 0x94:
                    {
                        MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.ZeroPageX),
(byte)YRegister);
                        break;
                    }
                //STY Store Index Y, Absolute, 2 Bytes, 4 Cycles
                case 0x8C:
                    {
                        MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.Absolute),
(byte)YRegister);
                        break;
                    }
#endregion

#region Transfer Operations
                //TAX Transfer Accumulator to X Register, Implied, 1 Bytes, 2 Cycles
                case 0xAA:
                    {
                        IncrementCycleCount();
                        XRegister = Accumulator;

                        SetNegativeFlag(XRegister);
                        SetZeroFlag(XRegister);
                        break;
                    }
                //TAY Transfer Accumulator to Y Register, 1 Bytes, 2 Cycles
                case 0xA8:
                    {
                        IncrementCycleCount();
                        YRegister = Accumulator;

                        SetNegativeFlag(YRegister);
                        SetZeroFlag(YRegister);
                        break;
                    }
                //TXA Transfer X Register to Accumulator, Implied, 1 Bytes, 2 Cycles
                case 0x8A:
                    {
                        IncrementCycleCount();
                        Accumulator = XRegister;

                        SetNegativeFlag(Accumulator);
                        SetZeroFlag(Accumulator);
                        break;
                    }
                //TYA Transfer Y Register to Accumulator, Implied, 1 Bytes, 2 Cycles
                case 0x98:
                    {
                        IncrementCycleCount();
                        Accumulator = YRegister;

                        SetNegativeFlag(Accumulator);
                        SetZeroFlag(Accumulator);
                        break;
                    }
#endregion

                //NOP Operation, Implied, 1 Byte, 2 Cycles
                case 0xEA:
                    {
                        IncrementCycleCount();
                        break;
                    }

                default:
                    throw new NotSupportedException(string.Format("The OpCode {0} is not supported",
CurrentOpCode));
            }
        }

/// <summary>
/// Sets the IsSignNegative register
/// </summary>
/// <param name="value"></param>
        protected void SetNegativeFlag(int value)
        {
            //on the 6502, any value greater than 127 is negative.  128 = 1000000 in Binary.  the 8th
bit is set, therefore the number is a negative number.
            NegativeFlag = value > 127;
        }

/// <summary>
/// Sets the IsResultZero register
/// </summary>
/// <param name="value"></param>
        protected void SetZeroFlag(int value)
        {
            ZeroFlag = value == 0;
        }

/// <summary>
/// Uses the AddressingMode to return the correct address based on the mode.
/// Note:  This method will not increment the program counter for any mode.
/// Note:  This method will return an error if called for either the immediate or accumulator modes.
/// </summary>
/// <param name="addressingMode">The addressing Mode to use</param>
/// <returns>The memory Location</returns>
        protected int GetAddressByAddressingMode(AddressingMode addressingMode)
        {
            int address;
            int highByte;
            switch (addressingMode)
            {
                case (AddressingMode.Absolute):
                    {
                        return (MemoryMap.Read(ProgramCounter++) | (MemoryMap.Read(ProgramCounter++) « 8));
                    }
                case AddressingMode.AbsoluteX:
                    {
                        //Get the low half of the address
                        address = MemoryMap.Read(ProgramCounter++);

                        //Get the high byte
                        highByte = MemoryMap.Read(ProgramCounter++);

                        //We crossed a page boundry, so an extra read has occurred.
                        //However, if this is an ASL, LSR, DEC, INC, ROR, ROL or STA operation, we do not decrease it by 1.
                        if (address + XRegister > 0xFF)
                        {
                            switch (CurrentOpCode)
                            {
                                case 0x1E:
                                case 0xDE:
                                case 0xFE:
                                case 0x5E:
                                case 0x3E:
                                case 0x7E:
                                case 0x9D:
                                    {
                                        //This is a MemoryMap.Read Fetch Write Operation, so we don't make the extra read.
                                        return ((highByte « 8 | address) + XRegister) & 0xFFFF;
                                    }
                                default:
                                    {
                                        MemoryMap.Read((((highByte « 8 | address) + XRegister) - 0xFF) & 0xFFFF);
                                        break;
                                    }
                            }
                        }

                        return ((highByte « 8 | address) + XRegister) & 0xFFFF;
                    }
                case AddressingMode.AbsoluteY:
                    {
                        //Get the low half of the address
                        address = MemoryMap.Read(ProgramCounter++);

                        //Get the high byte
                        highByte = MemoryMap.Read(ProgramCounter++);

                        //We crossed a page boundry, so decrease the number of cycles by 1 if the operation is not STA
                        if (address + YRegister > 0xFF && CurrentOpCode != 0x99)
                        {
                            MemoryMap.Read((((highByte « 8 | address) + YRegister) - 0xFF) & 0xFFFF);
                        }

                        //Bitshift the high byte into place, AND with FFFF to handle wrapping.
                        return ((highByte « 8 | address) + YRegister) & 0xFFFF;
                    }
                case AddressingMode.Immediate:
                    {
                        return ProgramCounter++;
                    }
                case AddressingMode.IndirectX:
                    {
                        //Get the location of the address to retrieve
                        address = MemoryMap.Read(ProgramCounter++);
                        MemoryMap.Read(address);

                        address += XRegister;

                        //Now get the final Address.  The is not a zero page address either.
                        var finalAddress = MemoryMap.Read((address & 0xFF)) | (MemoryMap.Read((address + 1) & 0xFF) « 8);
                        return finalAddress;
                    }
                case AddressingMode.IndirectY:
                    {
                        address = MemoryMap.Read(ProgramCounter++);

                        var finalAddress = MemoryMap.Read(address) + (MemoryMap.Read((address + 1) & 0xFF) « 8);

                        if ((finalAddress & 0xFF) + YRegister > 0xFF && CurrentOpCode != 0x91)
                        {
                            MemoryMap.Read((finalAddress + YRegister - 0xFF) & 0xFFFF);
                        }

                        return (finalAddress + YRegister) & 0xFFFF;
                    }
                case AddressingMode.Relative:
                    {
                        return ProgramCounter;
                    }
                case (AddressingMode.ZeroPage):
                    {
                        address = MemoryMap.Read(ProgramCounter++);
                        return address;
                    }
                case (AddressingMode.ZeroPageX):
                    {
                        address = MemoryMap.Read(ProgramCounter++);
                        MemoryMap.Read(address);

                        address += XRegister;
                        address &= 0xFF;

                        //This address wraps if its greater than 0xFF
                        if (address > 0xFF)
                        {
                            address -= 0x100;
                            return address;
                        }

                        return address;
                    }
                case (AddressingMode.ZeroPageY):
                    {
                        address = MemoryMap.Read(ProgramCounter++);
                        MemoryMap.Read(address);

                        address += YRegister;
                        address &= 0xFF;

                        return address;
                    }
                default:
                    throw new InvalidOperationException(string.Format("The Address Mode '{0}' does not require an address", addressingMode));
            }
        }

/// <summary>
/// Moves the ProgramCounter in a given direction based on the value inputted
///
/// </summary>
        private void MoveProgramCounterByRelativeValue(byte valueToMove)
        {
            var movement = valueToMove > 127 ?  (valueToMove - 255) :  valueToMove;

            var newProgramCounter = ProgramCounter + movement;

            //This makes sure that we always land on the correct spot for a positive number
            if (movement >= 0)
                newProgramCounter++;

            //We Crossed a Page Boundary.  So we increment the cycle counter by one.  The +1 is because we always check from the end of the instruction not the beginning
            if (((ProgramCounter + 1 ^ newProgramCounter) & 0xff00) != 0x0000)
            {
                IncrementCycleCount();
            }

            ProgramCounter = newProgramCounter;
            MemoryMap.Read(ProgramCounter);
        }

/// <summary>
/// Returns a the value from the stack without changing the position of the stack pointer
/// </summary>
/// <returns>The value at the current Stack Pointer</returns>
        private byte PeekStack()
        {
            //The stack lives at 0x100-0x1FF, but the value is only a byte so it needs to be translated
            return MemoryMap.Read(StackPointer + 0x100);
        }

/// <summary>
/// Write a value directly to the stack without modifying the Stack Pointer
/// </summary>
///
/// <param name="value">The value to be written to the stack</param>
        private void PokeStack(byte value)
        {
            //The stack lives at 0x100-0x1FF, but the value is only a byte so it needs to be translated
            MemoryMap.Write(StackPointer + 0x100, value);
        }

/// <summary>
/// Coverts the Flags into its byte representation.
/// </summary>
/// <param name="setBreak">Determines if the break flag should be set during conversion.  IRQ does not set the flag on the stack, but PHP and BRK do</param>
/// <returns></returns>
        private byte ConvertFlagsToByte(bool setBreak)
        {
            return (byte)((CarryFlag ?  0x01 :  0) + (ZeroFlag ?  0x02 :  0) + (DisableInterruptFlag ? 0x04 :  0) +
                (DecimalFlag ?  8 :  0) + (setBreak ?  0x10 :  0) + 0x20 + (OverflowFlag ?  0x40 :  0) + (NegativeFlag ?  0x80 :  0));
        }

        [Conditional("DEBUG")]
        private void SetDisassembly()
        {
            if (!_logger.IsDebugEnabled)
                return;

            var addressMode = GetAddressingMode();

            var currentProgramCounter = ProgramCounter;

            currentProgramCounter = WrapProgramCounter(++currentProgramCounter);
            int?  address1 = MemoryMap.Read(currentProgramCounter);

            currentProgramCounter = WrapProgramCounter(++currentProgramCounter);
            int?  address2 = MemoryMap.Read(currentProgramCounter);

            string disassembledStep = string.Empty;

            switch (addressMode)
            {
                case AddressingMode.Absolute:
                    {
                        disassembledStep = string.Format("${0}{1}", address2.Value.ToString("X").PadLeft(2, '0'), address1.Value.ToString("X").PadLeft(2, '0'));
                        break;
                    }
                case AddressingMode.AbsoluteX:
                    {
                        disassembledStep = string.Format("${0}{1},X", address2.Value.ToString("X").PadLeft(2, '0'), address1.Value.ToString("X").PadLeft(2, '0'));
                        break;
                    }
                case AddressingMode.AbsoluteY:
                    {
                        disassembledStep = string.Format("${0}{1},Y", address2.Value.ToString("X").PadLeft(2, '0'), address1.Value.ToString("X").PadLeft(2, '0'));
                        break;
                    }
                case AddressingMode.Accumulator:
                    {
                        address1 = null;
                        address2 = null;

                        disassembledStep = "A";
                        break;
                    }
                case AddressingMode.Immediate:
                    {
                        disassembledStep = string.Format("#${0}", address1.Value.ToString("X").PadLeft(4, ’0’));
                        address2 = null;
                        break;
                    }
                case AddressingMode.Implied:
                    {
                        address1 = null;
                        address2 = null;
                        break;
                    }
                case AddressingMode.Indirect:
                    {
                        disassembledStep = string.Format("(${0}{1})", address2.Value.ToString("X").PadLeft(2, ’0’), address1.Value.ToString("X").PadLeft(2, ’0’));
                        break;
                    }
                case AddressingMode.IndirectX:
                    {
                        address2 = null;

                        disassembledStep = string.Format("(${0},X)", address1.Value.ToString("X").PadLeft(2, ’0’));
                        break;
                    }
                case AddressingMode.IndirectY:
                    {
                        address2 = null;

                        disassembledStep = string.Format("(${0}),Y", address1.Value.ToString("X").PadLeft(2, ’0’));
                        break;
                    }
                case AddressingMode.Relative:
                    {
                        var valueToMove = (byte)address1.Value;

                        var movement = valueToMove > 127 ?  (valueToMove - 255) :  valueToMove;

                        var newProgramCounter = ProgramCounter + movement;

                        //This makes sure that we always land on the correct spot for a positive number
                        if (movement >= 0)
                            newProgramCounter++;

                        var stringAddress = ProgramCounter.ToString("X").PadLeft(4, ’0’);

                        address1 = int.Parse(stringAddress.Substring(0, 2), NumberStyles.AllowHexSpecifier);
                        address2 = int.Parse(stringAddress.Substring(2, 2), NumberStyles.AllowHexSpecifier);

                        disassembledStep = string.Format("${0}", newProgramCounter.ToString("X").PadLeft(4, ’0’));

                        break;
                    }
                case AddressingMode.ZeroPage:
                    {
                        address2 = null;

                        disassembledStep = string.Format("${0}", address1.Value.ToString("X").PadLeft(2, ’0’));
                        break;
                    }
                case AddressingMode.ZeroPageX:
                    {
                        address2 = null;

                        disassembledStep = string.Format("${0},X", address1.Value.ToString("X").PadLeft(2, ’0’));
                        break;
                    }
                case AddressingMode.ZeroPageY:
                    {
                        address2 = null;

                        disassembledStep = string.Format("${0},Y", address1.Value.ToString("X").PadLeft(4, ’0’));
                        break;
                    }
                default:
                    throw new InvalidEnumArgumentException("Invalid Addressing Mode");

            }


            CurrentDisassembly = new Disassembly
            {
                HighAddress = address2.HasValue ?  address2.Value.ToString("X").PadLeft(2, '0') :
      string.Empty,
                LowAddress = address1.HasValue ?  address1.Value.ToString("X").PadLeft(2, '0') :
      string.Empty,
                OpCodeString = CurrentOpCode.ConvertOpCodeIntoString(),
                DisassemblyOutput = disassembledStep
            };

            _logger.Debug("{0} :  {1}{2}{3} {4} {5} A: {6} X: {7} Y: {8} SP {9} N: {10} V: {11} B:
      {12} D: {13} I: {14} Z: {15} C: {16}",
                 ProgramCounter.ToString("X").PadLeft(4, '0'),
                 CurrentOpCode.ToString("X").PadLeft(2, '0'),
                 CurrentDisassembly.LowAddress,
                 CurrentDisassembly.HighAddress,

                 CurrentDisassembly.OpCodeString,
                 CurrentDisassembly.DisassemblyOutput.PadRight(10, ' '),

                 Accumulator.ToString("X").PadLeft(3, '0'),
                     XRegister.ToString("X").PadLeft(3, '0'),
                     YRegister.ToString("X").PadLeft(3, '0'),
                     StackPointer.ToString("X").PadLeft(3, '0'),
                     Convert.ToInt16(NegativeFlag),
                     Convert.ToInt16(OverflowFlag),
                     0,
                     Convert.ToInt16(DecimalFlag),
                     Convert.ToInt16(DisableInterruptFlag),
                     Convert.ToInt16(ZeroFlag),
                     Convert.ToInt16(CarryFlag));
        }

        private int WrapProgramCounter(int value)
        {
            return value & 0xFFFF;
        }

        private AddressingMode GetAddressingMode()
        {
            switch (CurrentOpCode)
            {
                case 0x0D:  //ORA
                case 0x2D:  //AND
                case 0x4D:  //EOR
                case 0x6D:  //ADC
                case 0x8D:  //STA
                case 0xAD:  //LDA
                case 0xCD:  //CMP
                case 0xED:  //SBC
                case 0x0E:  //ASL
                case 0x2E:  //ROL
                case 0x4E:  //LSR
                case 0x6E:  //ROR
                case 0x8E:  //SDX
                case 0xAE:  //LDX
                case 0xCE:  //DEC
                case 0xEE:  //INC
                case 0x2C:  //Bit
                case 0x4C:  //JMP
                case 0x8C:  //STY
                case 0xAC:  //LDY
                case 0xCC:  //CPY
                case 0xEC:  //CPX
                case 0x20:    //JSR
                    {
                        return AddressingMode.Absolute;
                    }
                case 0x1D:  //ORA
                case 0x3D:  //AND
                case 0x5D:  //EOR
                case 0x7D:  //ADC
                case 0x9D:  //STA
                case 0xBD:  //LDA
                case 0xDD:  //CMP
                case 0xFD:  //SBC
                case 0xBC:  //LDY
                case 0xFE:  //INC
                case 0x1E:  //ASL
                case 0x3E:  //ROL
                case 0x5E:  //LSR
                case 0x7E:  //ROR
```

```
                    {
                        return AddressingMode.AbsoluteX;
                    }
                case 0x19:    //ORA
                case 0x39:    //AND
                case 0x59:    //EOR
                case 0x79:    //ADC
                case 0x99:    //STA
                case 0xB9:    //LDA
                case 0xD9:    //CMP
                case 0xF9:    //SBC
                case 0xBE:  //LDX
                    {
                        return AddressingMode.AbsoluteY;
                    }
                case 0x0A:  //ASL
                case 0x4A:  //LSR
                case 0x2A:  //ROL
                case 0x6A:  //ROR
                    {
                        return AddressingMode.Accumulator;
                    }

                case 0x09:    //ORA
                case 0x29:    //AND
                case 0x49:    //EOR
                case 0x69:    //ADC
                case 0xA0:    //LDY
                case 0xC0:    //CPY
                case 0xE0:    //CMP
                case 0xA2:    //LDX
                case 0xA9:    //LDA
                case 0xC9:    //CMP
                case 0xE9:    //SBC
                    {
                        return AddressingMode.Immediate;
                    }
                case 0x00:    //BRK
                case 0x18:    //CLC
                case 0xD8:    //CLD
                case 0x58:    //CLI
                case 0xB8:    //CLV
                case 0xDE:  //DEC
                case 0xCA:  //DEX
                case 0x88:    //DEY
                case 0xE8:    //INX
                case 0xC8:    //INY
                case 0xEA:  //NOP
                case 0x48:    //PHA
                case 0x08:    //PHP
                case 0x68:    //PLA
                case 0x28:    //PLP
                case 0x40:    //RTI
                case 0x60:    //RTS
                case 0x38:    //SEC
                case 0xF8:    //SED
                case 0x78:    //SEI
                case 0xAA:  //TAX
                case 0xA8:    //TAY
                case 0xBA:  //TSX
                case 0x8A:  //TXA
                case 0x9A:  //TXS
                case 0x98:    //TYA
                    {
                        return AddressingMode.Implied;
                    }
                case 0x6C:
                    {
                        return AddressingMode.Indirect;
                    }

                case 0x61:    //ADC
                case 0x21:    //AND
                case 0xC1:    //CMP
                case 0x41:    //EOR
                case 0xA1:    //LDA
                case 0x01:    //ORA
                case 0xE1:    //SBC
                case 0x81:    //STA
                    {
                        return AddressingMode.IndirectX;
                    }
                case 0x71:    //ADC
                case 0x31:    //AND
                case 0xD1:    //CMP
                case 0x51:    //EOR
                case 0xB1:    //LDA
                case 0x11:    //ORA
                case 0xF1:    //SBC
                case 0x91:    //STA
                    {
                        return AddressingMode.IndirectY;
                    }
                case 0x90:    //BCC
                case 0xB0:    //BCS
                case 0xF0:    //BEQ
                case 0x30:    //BMI
                case 0xD0:    //BNE
                case 0x10:    //BPL
                case 0x50:    //BVC
                case 0x70:    //BVS
                    {
                        return AddressingMode.Relative;
                    }
                case 0x65:    //ADC
                case 0x25:    //AND
                case 0x06:    //ASL
                case 0x24:    //BIT
                case 0xC5:    //CMP
                case 0xE4:    //CPX
                case 0xC4:    //CPY
                case 0xC6:    //DEC
                case 0x45:    //EOR
                case 0xE6:    //INC
                case 0xA5:    //LDA
                case 0xA6:    //LDX
                case 0xA4:    //LDY
                case 0x46:    //LSR
                case 0x05:    //ORA
                case 0x26:    //ROL
                case 0x66:    //ROR
                case 0xE5:    //SBC
                case 0x85:    //STA
                case 0x86:    //STX
                case 0x84:    //STY
                    {
                        return AddressingMode.ZeroPage;
                    }
                case 0x75:    //ADC
                case 0x35:    //AND
                case 0x16:    //ASL
                case 0xD5:    //CMP
                case 0xD6:    //DEC
                case 0x55:    //EOR
                case 0xF6:    //INC
                case 0xB5:    //LDA
                case 0xB6:    //LDX
                case 0xB4:    //LDY
                case 0x56:    //LSR
                case 0x15:    //ORA
                case 0x36:    //ROL
                case 0x76:    //ROR
                case 0xF5:    //SBC
                case 0x95:    //STA
                case 0x96:    //STX
                case 0x94:    //STY
                    {
                        return AddressingMode.ZeroPageX;
                    }
                default:
                    throw new NotSupportedException(string.Format("Opcode {0} is not supported",
CurrentOpCode));
            }
        }

#region Op Code Operations
/// <summary>
/// The ADC - Add Memory to Accumulator with Carry Operation
/// </summary>
/// <param name="addressingMode">The addressing mode used to perform this operation.</param>
        protected void AddWithCarryOperation(AddressingMode addressingMode)
        {
            //Accumulator, Carry = Accumulator + ValueInMemoryLocation + Carry
            var memoryValue = MemoryMap.Read(GetAddressByAddressingMode(addressingMode));
            var newValue = memoryValue + Accumulator + (CarryFlag ?  1 :  0);


            OverflowFlag = (((Accumulator ^ newValue) & 0x80) != 0) && (((Accumulator ^ memoryValue) &
0x80) == 0);

            if (DecimalFlag)
            {
                newValue = int.Parse(memoryValue.ToString("x")) + int.Parse(Accumulator.ToString("x"))
+ (CarryFlag ?  1 :  0);
```

```

                if (newValue > 99)
                {
                    CarryFlag = true;
                    newValue -= 100;
                }
                else
                {
                    CarryFlag = false;
                }

                newValue = (int)Convert.ToInt64(string.Concat("0x", newValue), 16);
            }
            else
            {
                if (newValue > 255)
                {
                    CarryFlag = true;
                    newValue -= 256;
                }
                else
                {
                    CarryFlag = false;
                }
            }

            SetZeroFlag(newValue);
            SetNegativeFlag(newValue);

            Accumulator = newValue;
        }

/// <summary>
/// The AND - Compare Memory with Accumulator operation
/// </summary>
/// <param name="addressingMode">The addressing mode being used</param>
        private void AndOperation(AddressingMode addressingMode)
        {
            Accumulator = MemoryMap.Read(GetAddressByAddressingMode(addressingMode)) & Accumulator;

            SetZeroFlag(Accumulator);
            SetNegativeFlag(Accumulator);
        }

/// <summary>
/// The ASL - Shift Left One Bit (Memory or Accumulator)
/// </summary>
/// <param name="addressingMode">The addressing Mode being used</param>
        public void AslOperation(AddressingMode addressingMode)
        {
            int value;
            var memoryAddress = 0;
            if (addressingMode == AddressingMode.Accumulator)
            {
                MemoryMap.Read(ProgramCounter + 1);
                value = Accumulator;
            }
            else
            {
                memoryAddress = GetAddressByAddressingMode(addressingMode);
                value = MemoryMap.Read(memoryAddress);
            }

            //Dummy Write
            if (addressingMode != AddressingMode.Accumulator)
            {
                MemoryMap.Write(memoryAddress, (byte)value);
            }

            //If the 7th bit is set, then we have a carry
            CarryFlag = ((value & 0x80) != 0);

            //The And here ensures that if the value is greater than 255 it wraps properly.
            value = (value « 1) & 0xFE;

            SetNegativeFlag(value);
            SetZeroFlag(value);


            if (addressingMode == AddressingMode.Accumulator)
                Accumulator = value;
            else
            {
                MemoryMap.Write(memoryAddress, (byte)value);
            }
        }

```

```
/// <summary>
/// Performs the different branch operations.
/// </summary>
/// <param name="performBranch">Is a branch required</param>
        private void BranchOperation(bool performBranch)
        {
            var value = MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.Relative));

            if (!performBranch)
            {
                ProgramCounter++;
                return;
            }

            MoveProgramCounterByRelativeValue(value);
        }

/// <summary>
/// The bit operation, does an & comparison between a value in memory and the accumulator
/// </summary>
/// <param name="addressingMode"></param>
        private void BitOperation(AddressingMode addressingMode)
        {

            var memoryValue = MemoryMap.Read(GetAddressByAddressingMode(addressingMode));
            var valueToCompare = memoryValue & Accumulator;

            OverflowFlag = (memoryValue & 0x40) != 0;

            SetNegativeFlag(memoryValue);
            SetZeroFlag(valueToCompare);
        }

/// <summary>
/// The compare operation.  This operation compares a value in memory with a value passed into it.
/// </summary>
/// <param name="addressingMode">The addressing mode to use</param>
/// <param name="comparisonValue">The value to compare against memory</param>
        private void CompareOperation(AddressingMode addressingMode, int comparisonValue)
        {
            var memoryValue = MemoryMap.Read(GetAddressByAddressingMode(addressingMode));
            var comparedValue = comparisonValue - memoryValue;

            if (comparedValue < 0)
                comparedValue += 0x10000;

            SetZeroFlag(comparedValue);

            CarryFlag = memoryValue <= comparisonValue;
            SetNegativeFlag(comparedValue);
        }

/// <summary>
/// Changes a value in memory by 1
/// </summary>
/// <param name="addressingMode">The addressing mode to use</param>
/// <param name="decrement">If the operation is decrementing or incrementing the vaulue by 1 </param>
        private void ChangeMemoryByOne(AddressingMode addressingMode, bool decrement)
        {
            var memoryLocation = GetAddressByAddressingMode(addressingMode);
            var memory = MemoryMap.Read(memoryLocation);

            MemoryMap.Write(memoryLocation, memory);

            if (decrement)
                memory -= 1;
            else
                memory += 1;

            SetZeroFlag(memory);
            SetNegativeFlag(memory);


            MemoryMap.Write(memoryLocation, memory);
        }

/// <summary>
/// Changes a value in either the X or Y register by 1
/// </summary>
/// <param name="useXRegister">If the operation is using the X or Y register</param>
/// <param name="decrement">If the operation is decrementing or incrementing the vaulue by 1 </param>
        private void ChangeRegisterByOne(bool useXRegister, bool decrement)
        {
            var value = useXRegister ?  XRegister :  YRegister;

            if (decrement)
                value -= 1;
            else
                value += 1;

            if (value < 0x00)
                value += 0x100;
            else if (value > 0xFF)
                value -= 0x100;

            SetZeroFlag(value);
            SetNegativeFlag(value);
            IncrementCycleCount();

            if (useXRegister)
                XRegister = value;
            else
                YRegister = value;
        }

/// <summary>
/// The EOR Operation, Performs an Exclusive OR Operation against the Accumulator and a value in
memory
/// </summary>
/// <param name="addressingMode">The addressing mode to use</param>
        private void EorOperation(AddressingMode addressingMode)
        {
            Accumulator = Accumulator ^ MemoryMap.Read(GetAddressByAddressingMode(addressingMode));

            SetNegativeFlag(Accumulator);
            SetZeroFlag(Accumulator);
        }

/// <summary>
/// The LSR Operation.  Performs a Left shift operation on a value in memory
/// </summary>
/// <param name="addressingMode">The addressing mode to use</param>
        private void LsrOperation(AddressingMode addressingMode)
        {
            int value;
            var memoryAddress = 0;
            if (addressingMode == AddressingMode.Accumulator)
            {
                MemoryMap.Read(ProgramCounter + 1);
                value = Accumulator;
            }
            else
            {
                memoryAddress = GetAddressByAddressingMode(addressingMode);
                value = MemoryMap.Read(memoryAddress);
            }

            //Dummy Write
            if (addressingMode != AddressingMode.Accumulator)
            {
                MemoryMap.Write(memoryAddress, (byte)value);
            }

            NegativeFlag = false;

            //If the Zero bit is set, we have a carry
            CarryFlag = (value & 0x01) != 0;

            value = (value » 1);

            SetZeroFlag(value);
            if (addressingMode == AddressingMode.Accumulator)
                Accumulator = value;
            else
            {
                MemoryMap.Write(memoryAddress, (byte)value);
            }
        }

/// <summary>
/// The Or Operation.  Performs an Or Operation with the accumulator and a value in memory
/// </summary>
/// <param name="addressingMode">The addressing mode to use</param>
        private void OrOperation(AddressingMode addressingMode)
        {
            Accumulator = Accumulator | MemoryMap.Read(GetAddressByAddressingMode(addressingMode));

            SetNegativeFlag(Accumulator);
            SetZeroFlag(Accumulator);
        }

/// <summary>
/// The ROL operation.  Performs a rotate left operation on a value in memory.
/// </summary>
/// <param name="addressingMode">The addressing mode to use</param>
        private void RolOperation(AddressingMode addressingMode)
        {
            int value;
            var memoryAddress = 0;
            if (addressingMode == AddressingMode.Accumulator)
            {
                //Dummy MemoryMap.Read
                MemoryMap.Read(ProgramCounter + 1);
                value = Accumulator;
            }
            else
            {
                memoryAddress = GetAddressByAddressingMode(addressingMode);
                value = MemoryMap.Read(memoryAddress);
            }

            //Dummy Write
            if (addressingMode != AddressingMode.Accumulator)
            {
                MemoryMap.Write(memoryAddress, (byte)value);
            }

            //Store the carry flag before shifting it
            var newCarry = (0x80 & value) != 0;

            //The And here ensures that if the value is greater than 255 it wraps properly.
            value = (value « 1) & 0xFE;

            if (CarryFlag)
                value = value | 0x01;

            CarryFlag = newCarry;

            SetZeroFlag(value);
            SetNegativeFlag(value);


            if (addressingMode == AddressingMode.Accumulator)
                Accumulator = value;
            else
            {
                MemoryMap.Write(memoryAddress, (byte)value);
            }
        }

/// <summary>
/// The ROR operation.  Performs a rotate right operation on a value in memory.
/// </summary>
/// <param name="addressingMode">The addressing mode to use</param>
        private void RorOperation(AddressingMode addressingMode)
        {
            int value;
            var memoryAddress = 0;
            if (addressingMode == AddressingMode.Accumulator)
            {
                //Dummy MemoryMap.Read
                MemoryMap.Read(ProgramCounter + 1);
                value = Accumulator;
            }
            else
            {
                memoryAddress = GetAddressByAddressingMode(addressingMode);
                value = MemoryMap.Read(memoryAddress);
            }

            //Dummy Write
            if (addressingMode != AddressingMode.Accumulator)
            {
                MemoryMap.Write(memoryAddress, (byte)value);
            }

            //Store the carry flag before shifting it
            var newCarry = (0x01 & value) != 0;

            value = (value » 1);

            //If the carry flag is set then 0x
            if (CarryFlag)
                value = value | 0x80;

            CarryFlag = newCarry;

            SetZeroFlag(value);
            SetNegativeFlag(value);

            if (addressingMode == AddressingMode.Accumulator)
```

```
                Accumulator = value;
            else
            {
                MemoryMap.Write(memoryAddress, (byte)value);
            }
        }

/// <summary>
/// The SBC operation.  Performs a subtract with carry operation on the accumulator and a value in memory.
/// </summary>
/// <param name="addressingMode">The addressing mode to use</param>
        protected void SubtractWithBorrowOperation(AddressingMode addressingMode)
        {
            var memoryValue = MemoryMap.Read(GetAddressByAddressingMode(addressingMode));
            var newValue = DecimalFlag ? int.Parse(Accumulator.ToString("x")) - int.Parse(memoryValue.ToString("x")) - (CarryFlag ? 0 : 1) : Accumulator - memoryValue - (CarryFlag ? 0 : 1);

            CarryFlag = newValue >= 0;

            if (DecimalFlag)
            {
                if (newValue < 0)
                    newValue += 100;

                newValue = (int)Convert.ToInt64(string.Concat("0x", newValue), 16);
            }
            else
            {
                OverflowFlag = (((Accumulator ^ newValue) & 0x80) != 0) && (((Accumulator ^ memoryValue) & 0x80) != 0);

                if (newValue < 0)
                    newValue += 256;
            }

            SetNegativeFlag(newValue);
            SetZeroFlag(newValue);

            Accumulator = newValue;
        }

/// <summary>
/// The PSP Operation.  Pushes the Status Flags to the stack
/// </summary>
        private void PushFlagsOperation()
        {
            PokeStack(ConvertFlagsToByte(true));
        }

/// <summary>
/// The PLP Operation.  Pull the status flags off the stack on sets the flags accordingly.
/// </summary>
        private void PullFlagsOperation()
        {
            var flags = PeekStack();
            CarryFlag = (flags & 0x01) != 0;
            ZeroFlag = (flags & 0x02) != 0;
            DisableInterruptFlag = (flags & 0x04) != 0;
            DecimalFlag = (flags & 0x08) != 0;
            OverflowFlag = (flags & 0x40) != 0;
            NegativeFlag = (flags & 0x80) != 0;


        }

/// <summary>
/// The JSR routine.  Jumps to a subroutine.
/// </summary>
        private void JumpToSubRoutineOperation()
        {
            IncrementCycleCount();

            //Put the high value on the stack, this should be the address after our operation -1
            //The RTS operation increments the PC by 1 which is why we don't move 2
            PokeStack((byte)(((ProgramCounter + 1) » 8) & 0xFF));
            StackPointer--;
            IncrementCycleCount();

            PokeStack((byte)((ProgramCounter + 1) & 0xFF));
            StackPointer--;
            IncrementCycleCount();

            ProgramCounter = GetAddressByAddressingMode(AddressingMode.Absolute);
        }

```

```
/// <summary>
/// The RTS routine.  Called when returning from a subroutine.
/// </summary>
        private void ReturnFromSubRoutineOperation()
        {
            MemoryMap.Read(++ProgramCounter);
            StackPointer++;
            IncrementCycleCount();

            var lowBit = PeekStack();
            StackPointer++;
            IncrementCycleCount();

            var highBit = PeekStack() « 8;
            IncrementCycleCount();

            ProgramCounter = (highBit | lowBit) + 1;
            IncrementCycleCount();
        }


/// <summary>
/// The BRK routine.  Called when a BRK occurs.
/// </summary>
        private void BreakOperation(bool isBrk, int vector)
        {
            MemoryMap.Read(++ProgramCounter);

            //Put the high value on the stack
            //When we RTI the address will be incremented by one, and the address after a break will not be used.
            PokeStack((byte)(((ProgramCounter) » 8) & 0xFF));
            StackPointer--;
            IncrementCycleCount();

            //Put the low value on the stack
            PokeStack((byte)((ProgramCounter) & 0xFF));
            StackPointer--;
            IncrementCycleCount();

            //We only set the Break Flag is a Break Occurs
            if (isBrk)
                PokeStack((byte)(ConvertFlagsToByte(true) | 0x10));
            else
                PokeStack(ConvertFlagsToByte(false));

            StackPointer--;
            IncrementCycleCount();

            DisableInterruptFlag = true;

            ProgramCounter = (MemoryMap.Read(vector + 1) « 8) | MemoryMap.Read(vector);

            _previousInterrupt = false;
        }

/// <summary>
/// The RTI routine.  Called when returning from a BRK opertion.
/// Note:  when called after a BRK operation the Program Counter is not set to the location after the BRK,
/// it is set +1
/// </summary>
        private void ReturnFromInterruptOperation()
        {
            MemoryMap.Read(++ProgramCounter);
            StackPointer++;
            IncrementCycleCount();

            PullFlagsOperation();
            StackPointer++;
            IncrementCycleCount();

            var lowBit = PeekStack();
            StackPointer++;
            IncrementCycleCount();

            var highBit = PeekStack() « 8;
            IncrementCycleCount();

            ProgramCounter = (highBit | lowBit);
        }

/// <summary>
/// This is ran anytime an NMI occurrs
/// </summary>
        private void ProcessNMI()
        {
            ProgramCounter--;
            BreakOperation(false, 0xFFFA);
            CurrentOpCode = MemoryMap.Read(ProgramCounter);

            SetDisassembly();
        }

/// <summary>
/// This is ran anytime an IRQ occurrs
/// </summary>
        private void ProcessIRQ()
        {
            if (DisableInterruptFlag)
                return;

            ProgramCounter--;
            BreakOperation(false, 0xFFFE);
            CurrentOpCode = MemoryMap.Read(ProgramCounter);

            SetDisassembly();
        }
#endregion

#endregion
    }
}
```

7.93 Hardware/W65C22.cs File Reference

Classes

- class Hardware.W65C22

    An implementation of a W65C22 VIA.

Namespaces

- namespace Hardware

7.94 W65C22.cs

Go to the documentation of this file.

```
using System;
using System.IO;
using System.Timers;

namespace Hardware
{
/// <summary>
/// An implementation of a W65C22 VIA.
/// </summary>
    [Serializable]
    public class W65C22
    {
#region Fields
        public readonly bool T1IsIRQ = false;
        public readonly bool T2IsIRQ = true;
        public int T1CL = 0x04;
        public int T1CH = 0x05;
        public int T2CL = 0x08;
        public int T2CH = 0x09;
        public int ACR = 0x0B;
        public int IFR = 0x0D;
        public int IER = 0x0E;

        public byte ACR_T1TC = (byte)(1 « 7);
        public byte ACR_T2TC = (byte)(1 « 6);

        public byte IFR_T2 = (byte)(1 « 5);
        public byte IFR_T1 = (byte)(1 « 6);
        public byte IFR_INT = (byte)(1 « 7);

        public byte IER_T2 = (byte)(1 « 5);
        public byte IER_T1 = (byte)(1 « 6);
        public byte IER_EN = (byte)(1 « 7);
#endregion

#region Properties
/// <summary>
/// The memory area.
/// </summary>
        public byte[] Memory { get; set; }

/// <summary>
/// The memory offset of the device.
/// </summary>
        public int Offset { get; set; }

/// <summary>
/// The length of the device memory.
/// </summary>
        public int Length { get; set; }

/// <summary>
/// The end of memory
/// </summary>
        public int End { get { return Offset + Length; } }

/// <summary>
/// T1 timer control
/// </summary>
        public bool T1TimerControl
        {
            get { return T1Object.AutoReset; }
            set { T1Object.AutoReset = value; }
        }

/// <summary>
/// T2 timer control.
/// </summary>
        public bool T2TimerControl
        {
            get { return T2Object.AutoReset; }
            set { T2Object.AutoReset = value; }
        }

/// <summary>
/// Enable or check whether timer 1 is enabled or not.
/// </summary>
        public bool T1IsEnabled
        {
            get { return T1Object.Enabled; }
            set { T1Object.Enabled = value; }
        }

/// <summary>
/// Enable or check whether timer 2 is enabled or not.
/// </summary>
        public bool T2IsEnabled
        {
            get { return T2Object.Enabled; }
            set { T2Object.Enabled = value; }
        }

/// <summary>
/// Set or check the timer 1 interval.
/// </summary>
        public double T1Interval { get { return (int)(Read(T1CL) | (Read(T1CH) « 8)); } }

/// <summary>
/// Set or check the timer 2 interval.
/// </summary>
        public double T2Interval
        {
            get { return (int)(Read(T2CL) | (Read(T2CH) « 8)); }
        }

/// <summary>
/// Set or get the timer 1 object.
/// </summary>
        public Timer T1Object { get; set; }

/// <summary>
/// Set or get the timer 2 object.
/// </summary>
        public Timer T2Object { get; set; }

/// <summary>
/// Local referemce to the processor object.
/// </summary>
```

```
        private W65C02 Processor { get; set; }
#endregion

#region Public Methods
        public W65C22(W65C02 processor, byte offset, int length)
        {
            if (offset > MemoryMap.DeviceArea.Length)
                throw new ArgumentException(String.Format("The offset:  {0} is greater than the device
area:  {1}", offset, MemoryMap.DeviceArea.Length));
            T1Init(1000);
            T2Init(1000);

            Offset = MemoryMap.DeviceArea.Offset | offset;
            Memory = new byte[length + 1];
            Length = length;
            Processor = processor;
        }

/// <summary>
/// Reset routine called whenever the emulated computer is reset.
/// </summary>
        public void Reset()
        {
            T1TimerControl = false;
            T1IsEnabled = false;
            T2TimerControl = false;
            T2IsEnabled = false;
        }

/// <summary>
/// Initialization routine for the VIA.
/// </summary>
/// <param name="timer">Amount of time to set timers for.</param>
        public void Init(double timer)
        {
            T1Init(timer);
            T2Init(timer);
        }

/// <summary>
/// T1 counter initialization routine.
/// </summary>
///
/// <param name="value">Timer initialization value in milliseconds.</param>
        public void T1Init(double value)
        {
            T1Object = new Timer(value);
            T1Object.Start();
            T1Object.Elapsed += OnT1Timeout;
            T1TimerControl = true;
            T1IsEnabled = false;
        }

/// <summary>
/// T2 counter initialization routine.
/// </summary>
///
/// <param name="value">Timer initialization value in milliseconds.</param>
        public void T2Init(double value)
        {
            T2Object = new Timer(value);
            T2Object.Start();
            T2Object.Elapsed += OnT2Timeout;
            T2TimerControl = true;
            T2IsEnabled = false;
        }

/// <summary>
/// Routine to read from local memory.
/// </summary>
///
/// <param name="address">Address to read from.</param>
///
/// <returns>Byte value stored in the local memory.</returns>
        public byte Read(int address)
        {
            if ((Offset <= address) && (address <= End))
            {
                byte data = 0x00;
                if (T1TimerControl)
                {
                    data = (byte)(data | ACR_T1TC);
                }
                else if (T2TimerControl)
                {
                    data = (byte)(data | ACR_T2TC);
                }
                return data;
            }
            else
            {
                return Memory[address - Offset];
            }
        }

/// <summary>
/// Writes data to the specified address in local memory.
/// </summary>
///
/// <param name="address">The address to write data to.</param>
/// <param name="data">The data to be written.</param>
        public void Write(int address, byte data)
        {
            if ((address == Offset + ACR) && ((data | ACR_T1TC) == ACR_T1TC))
            {
                T1TimerControl = true;
            }
            else if ((address == Offset + ACR) && ((data | ACR_T2TC) == ACR_T2TC))
            {
                T2TimerControl = true;
            }
            else if ((address == Offset + IER) && ((data | IER_T1) == IER_T1) && ((data | IER_EN) ==
IER_EN))
            {
                T1Init(T1Interval);
            }
            else if ((address == Offset + IER) && ((data | IER_T2) == IER_T2) && ((data | IER_EN) ==
IER_EN))
            {
                T2Init(T2Interval);
            }
            Memory[address - Offset] = data;
        }
#endregion

#region Private Methods
/// <summary>
/// Called whenever System.Timers.Timer event elapses.
/// </summary>
///
/// <param name="sender"></param>
/// <param name="e"></param>
        private void OnT1Timeout(object sender, ElapsedEventArgs e)
        {
            if (Processor.isRunning)
            {
                if (T1IsEnabled)
                {
                    Write(IFR, (byte)(IFR_T1 & IFR_INT));
                    if (T1IsIRQ)
                    {
                        Processor.InterruptRequest();
                    }
                    else
                    {
                        Processor.TriggerNmi = true;
                    }
                }
            }
        }

/// <summary>
/// Called whenever System.Timers.Timer event elapses
/// </summary>
///
/// <param name="sender"></param>
/// <param name="e"></param>
        private void OnT2Timeout(object sender, ElapsedEventArgs e)
        {
            if (Processor.isRunning)
            {
                if (T2IsEnabled)
                {
                    Write(IFR, (byte)(IFR_T2 & IFR_INT));
                    if (T2IsIRQ)
                    {
                        Processor.InterruptRequest();
                    }
                    else
                    {
                        Processor.TriggerNmi = true;
                    }
                }
            }
        }
#endregion
    }
}
```

7.95 Hardware/W65C51.cs File Reference

Classes

- class Hardware.W65C51

  An implementation of a W65C51 ACIA.

Namespaces

- namespace Hardware

7.96 W65C51.cs

Go to the documentation of this file.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.IO;
using System.IO.Ports;

namespace Hardware
{
/// <summary>
/// An implementation of a W65C51 ACIA.
/// </summary>
    [Serializable]
    public class W65C51
    {
#region Fields
        public readonly int defaultBaudRate = 115200;
        public byte byteIn;
#endregion

#region Properties
        public byte[] Memory { get; set; }
        public bool IsEnabled { get; set; }
        public SerialPort Object { get; set; }
        public string ObjectName { get; set; }
        private W65C02 Processor { get; set; }
        private BackgroundWorker _backgroundWorker { get; set; }
        public int Offset { get; set; }
        public int Length { get; set; }

        private bool DataRead { get; set; }
        private bool EchoMode { get; set; }
        private bool InterruptDisabled { get; set; }
        private bool Interrupted { get; set; }
        private bool Overrun { get; set; }
        private bool ParityEnabled { get; set; }
        private bool ReceiverFull { get; set; }
        private byte RtsControl { get; set; }
#endregion

#region Public Methods
        public W65C51(W65C02 processor, byte offset)
        {
            if (offset > MemoryMap.DeviceArea.Length)
                throw new ArgumentException(String.Format("The offset:  {0} is greater than the device
area:  {1}", offset, MemoryMap.DeviceArea.Length));

            Processor = processor;

            Offset = MemoryMap.DeviceArea.Offset | offset;
            Length = 0x04;
            Memory = new byte[Length + 1];

            _backgroundWorker = new BackgroundWorker
            {
                WorkerSupportsCancellation = true
            };
            _backgroundWorker.DoWork += BackgroundWorkerDoWork;
            _backgroundWorker.RunWorkerAsync();
        }

        public void Reset()
        {
            IsEnabled = false;
        }

/// <summary>
/// Default Constructor, Instantiates a new instance of COM Port I/O.
/// </summary>
///
/// <param name="port"> COM Port to use for I/O</param>
        public void Init(string port)
        {
            Object = new SerialPort(port, defaultBaudRate, Parity.None, 8, StopBits.One);
            ObjectName = port;

            ComInit(Object);
        }

/// <summary>
/// Default Constructor, Instantiates a new instance of COM Port I/O.
/// </summary>
///
/// <param name="port">COM Port to use for I/O</param>
/// <param name="baudRate">Baud Rate to use for I/O</param>
        public void Init(string port, int baudRate)
        {
            Object = new SerialPort(port, baudRate, Parity.None, 8, StopBits.One);
            ObjectName = port;

            ComInit(Object);
        }

/// <summary>
/// Called when the window is closed.
/// </summary>
        public void Fini()
        {
            ComFini(Object);
        }

/// <summary>
/// Returns the byte at a given address.
/// </summary>
///
/// <param name="address"></param>
///
/// <returns>the byte being returned</returns>
        public byte Read(int address)
        {
            HardwarePreRead(address);
            byte data = Memory[address - Offset];
            DataRead = true;
            return data;
        }

/// <summary>
/// Writes data to the given address.
/// </summary>
///
/// <param name="address">The address to write data to</param>
/// <param name="data">The data to write</param>
        public void Write(int address, byte data)
        {
            HardwarePreWrite(address, data);
            if (!((address == Offset) || (address == Offset + 1)))
            {
                Memory[address - Offset] = data;
            }
        }

/// <summary>
/// Called in order to write to the serial port.
/// </summary>
///
/// <param name="data">Byte of data to send</param>
        public void WriteCOM(byte data)
        {
            byte[] writeByte = new byte[] { data };
            Object.Write(writeByte, 0, 1);
        }
```

```
#endregion

#region Private Methods
/// <summary>
/// Called whenever the ACIA is initialized.
/// </summary>
///
/// <param name="serialPort">SerialPort object to initialize.</param>
        private void ComInit(SerialPort serialPort)
        {
            try
            {
                serialPort.Open();
            }
            catch (UnauthorizedAccessException w)
            {
                FileStream file = new FileStream(FileLocations.ErrorFile, FileMode.OpenOrCreate,
FileAccess.ReadWrite);
                StreamWriter stream = new StreamWriter(file);
                stream.WriteLine(w.Message);
                stream.WriteLine(w.Source);
                stream.Flush();
                file.Flush();
                stream.Close();
                file.Close();
                return;
            }
            serialPort.ReadTimeout = 50;
            serialPort.WriteTimeout = 50;
            serialPort.DataReceived += new SerialDataReceivedEventHandler(SerialDataReceived);
            try
            {
                serialPort.Write("---------------------------\r\n");
                serialPort.Write(" WolfNet 6502 WBC Emulator\r\n");
                serialPort.Write("---------------------------\r\n");
                serialPort.Write("\r\n");
            }
            catch (TimeoutException t)
            {
                _ = t;
                FileStream file = new FileStream(FileLocations.ErrorFile, FileMode.OpenOrCreate,
FileAccess.ReadWrite);
                StreamWriter stream = new StreamWriter(file);
                stream.WriteLine("Read/Write error:  Port timed out!");
                stream.WriteLine("Please ensure all cables are connected properly!");
                stream.Flush();
                file.Flush();
                stream.Close();
                file.Close();
                return;
            }
        }

/// <summary>
/// Called when the window is closed.
/// </summary>
///
/// <param name="serialPort">SerialPort Object to close</param>
        private void ComFini(SerialPort serialPort)
        {
            if (serialPort != null)
            {
                serialPort.Close();
            }

            _backgroundWorker.CancelAsync();
            _backgroundWorker.DoWork -= BackgroundWorkerDoWork;
        }

/// <summary>
/// Called whenever SerialDataReceivedEventHandler event occurs.
/// </summary>
///
/// <param name="sender"></param>
/// <param name="e"></param>
        private void SerialDataReceived(object sender, SerialDataReceivedEventArgs e)
        {
            try
            {
                if (EchoMode)
                {
                    WriteCOM(Convert.ToByte(Object.ReadByte()));
                }
                else
                {
                    if (!ReceiverFull)
                    {
                        ReceiverFull = true;
                    }
                    else
                    {
                        Overrun = true;
                    }
                    Memory[0] = Convert.ToByte(Object.ReadByte());
                }

                if (!InterruptDisabled)
                {
                    Interrupted = true;
                    Processor.InterruptRequest();
                }
            }
            catch (Win32Exception w)
            {
                FileStream file = new FileStream(FileLocations.ErrorFile, FileMode.OpenOrCreate,
      FileAccess.ReadWrite);
                StreamWriter stream = new StreamWriter(file);
                stream.WriteLine(w.Message);
                stream.WriteLine(w.ErrorCode.ToString());
                stream.WriteLine(w.Source);
                stream.Flush();
                stream.Close();
                file.Flush();
                file.Close();
            }
        }

        private void HardwarePreWrite(int address, byte data)
        {
            if (address == Offset)
            {
                WriteCOM(data);
            }
            else if (address == Offset + 1)
            {
                Reset();
            }
            else if (address == Offset + 2)
            {
                CommandRegister(data);
            }
            else if (address == Offset + 3)
            {
                ControlRegister(data);
            }
        }

        private void HardwarePreRead(int address)
        {
            if (address == Offset)
            {
                Interrupted = false;
                Overrun = false;
                ReceiverFull = false;

            }
            else if (address == Offset + 1)
            {
                StatusRegisterUpdate();
            }
            else if (address == Offset + 2)
            {
                CommandRegisterUpdate();
            }
            else if (address == Offset + 3)
            {
                ControlRegisterUpdate();
            }
        }

        private void CommandRegister(byte data)
        {
            byte test = (byte)(data & 0x20);
            if (test == 0x20)
            {
                throw new ArgumentException("Parity must NEVER be enabled!");
            }

            test = (byte)(data & 0x10);
            if (test == 0x10)
            {
                EchoMode = true;
            }
            else
            {
                EchoMode = false;
            }

            test = (byte)(data & 0x0C);
            if (test == 0x00)
            {
                Object.Handshake = Handshake.None;
                Object.RtsEnable = true;
                Object.Handshake = Handshake.RequestToSend;
            }
            else if (test == 0x04)
            {
                Object.Handshake = Handshake.None;
                Object.RtsEnable = false;
            }
            else if ((test == 0x08) || (test == 0x0C))
            {
                throw new NotImplementedException("This cannot be emulated on windows!");
            }
            else
            {
                throw new ArgumentOutOfRangeException("RtsControl is invalid!");
            }

            test = (byte)(data & 0x02);
            if (test == 0x02)
            {
                InterruptDisabled = true;
            }
            else
            {
                InterruptDisabled = false;
            }

            test = (byte)(data & 0x01);
            if (test == 0x01)
            {
                Object.DtrEnable = true;
            }
            else
            {
                Object.DtrEnable= false;
            }
        }

        private void CommandRegisterUpdate()
        {
            byte data = Memory[Offset + 2];

            if (ParityEnabled)
            {
                data |= 0x20;
            }
            else
            {
                data &= 0xD0;
            }

            if (EchoMode)
            {
                data |= 0x10;
            }
            else
            {
                data &= 0xE0;
            }

            data &= RtsControl;

            if (InterruptDisabled)
            {
                data |= 0x02;
            }
            else
            {
                data &= 0x0D;
            }
            if (Object.DtrEnable)
            {
                data |= 0x01;
            }
            else
            {
                data &= 0x0E;
            }

            Memory[Offset + 2] = data;
        }

        private void ControlRegister(byte data)
        {
            byte test = (byte)(data & 0x80);
            if (test == 0x80)
            {
                test = (byte)(data & 0x60);
                if (test == 0x60)
                {
                    Object.StopBits = StopBits.OnePointFive;
                }
                else
                {
                    Object.StopBits = StopBits.Two;
                }
            }
            else
            {
                Object.StopBits = StopBits.One;
            }

            test = (byte)(data & 0x60);
            if (test == 0x20)
            {
                Object.DataBits = 7;
            }
            else if (test == 0x40)
            {
                Object.DataBits = 6;
            }
            else if (test == 0x60)
            {
                Object.DataBits = 5;
            }
            else
            {
                Object.DataBits = 8;
            }

            test = (byte)(data & 0x10);
            if (!(test == 0x10))
            {
                throw new ArgumentException("External clock rate not available on the WolfNet 65C02
      WBC!");
            }

            test = (byte)(data & 0x0F);
            if (test == 0x00)
            {
                Object.BaudRate = 115200;
            }
            else if (test == 0x01)
            {
                Object.BaudRate = 50;
            }
            else if (test == 0x02)
            {
                Object.BaudRate = 75;
            }
            else if (test == 0x03)
            {
                Object.BaudRate = 110;
            }
            else if (test == 0x04)
            {
                Object.BaudRate = 135;
            }
            else if (test == 0x05)
            {
                Object.BaudRate = 150;
            }
            else if (test == 0x06)
            {
                Object.BaudRate = 300;
            }
            else if (test == 0x07)
            {
                Object.BaudRate = 600;
            }
            else if (test == 0x08)
            {
                Object.BaudRate = 1200;
            }
            else if (test == 0x09)
            {
                Object.BaudRate = 1800;
            }
            else if (test == 0x0A)
            {
                Object.BaudRate = 2400;
            }
            else if (test == 0x0B)
            {
                Object.BaudRate = 3600;
            }
            else if (test == 0x0C)
            {
                Object.BaudRate = 4800;
            }
            else if (test == 0x0D)
            {
                Object.BaudRate = 7200;
            }
            else if (test == 0x0E)
            {
                Object.BaudRate = 9600;
            }
            else
            {
                Object.BaudRate = 19200;
            }
        }

        private void ControlRegisterUpdate()
        {
            byte controlRegister = Memory[Offset + 3];

            if (Object.StopBits == StopBits.Two)
            {
                controlRegister |= 0x80;
            }
            else if ((Object.StopBits == StopBits.OnePointFive) && (Object.DataBits == 5) ||
      (Object.StopBits == StopBits.One))
            {
                controlRegister &= 0x7F;
            }
            else
            {
                throw new ArgumentOutOfRangeException("StopBits or combination of StopBits and
      DataBits is invalid!");
            }

            if (Object.DataBits == 8)
            {
                controlRegister &= 0x9F;
            }
            else if (Object.DataBits == 7)
            {
                controlRegister |= 0x20;
            }
            else if (Object.DataBits == 6)
            {
                controlRegister |= 0x40;
            }
            else if (Object.DataBits == 5)
            {
                controlRegister |= 0x60;
            }
            else
            {
                throw new ArgumentOutOfRangeException("DataBits is out of range!");
            }

            if (Object.BaudRate == 115200)
            {
                controlRegister &= 0xF0;
            }
            else if (Object.BaudRate == 50)
            {
                controlRegister |= 0x01;
            }
            else if (Object.BaudRate == 75)
            {
                controlRegister |= 0x02;
            }
            else if (Object.BaudRate == 110)
            {
                controlRegister |= 0x03;
            }
            else if (Object.BaudRate == 135)
            {
                controlRegister |= 0x04;
            }
            else if (Object.BaudRate == 150)
            {
                controlRegister |= 0x05;
            }
            else if (Object.BaudRate == 300)
            {
                controlRegister |= 0x06;
            }
            else if (Object.BaudRate == 600)
            {
                controlRegister |= 0x07;
            }
            else if (Object.BaudRate == 1200)
            {
                controlRegister |= 0x08;
            }
            else if (Object.BaudRate == 1800)
            {
                controlRegister |= 0x09;
            }
            else if (Object.BaudRate == 2400)
            {
                controlRegister |= 0x0A;
            }
            else if (Object.BaudRate == 3600)
            {
                controlRegister |= 0x0B;
            }
            else if (Object.BaudRate == 4800)
            {
                controlRegister |= 0x0C;
            }
            else if (Object.BaudRate == 7200)
            {
                controlRegister |= 0x0D;
            }
            else if (Object.BaudRate == 9600)
            {
                controlRegister |= 0x0E;
            }
            else if (Object.BaudRate == 19200)
            {
                controlRegister |= 0x0F;
            }
            else
            {
                throw new ArgumentOutOfRangeException("BaudRate is outside the range of Baud Rates
supported by the W65C51!");
            }

            Memory[Offset + 3] = controlRegister;
        }

        private void StatusRegisterUpdate()
        {
            byte statusRegister = Memory[Offset + 1];

            if (Interrupted)
            {
                statusRegister |= 0x80;
            }
            else
            {
                statusRegister &= 0x7F;
            }

            if (Object.DsrHolding == false)
            {
                statusRegister |= 0x40;
            }
            else
            {
                statusRegister &= 0xBF;
            }

            if (Object.CDHolding)
            {
                statusRegister |= 0x20;
            }
            else
            {
                statusRegister &= 0xDF;
            }

            statusRegister |= 0x10;

            if (ReceiverFull)
            {
                statusRegister |= 0x08;
            }
            else
            {
                statusRegister &= 0xF7;
            }

            if (Overrun)
            {
                statusRegister |= 0x04;
            }
            else
            {
                statusRegister &= 0xFB;
            }

            statusRegister &= 0xFC;

            Memory[Offset + 1] = statusRegister;
        }

        private void BackgroundWorkerDoWork(object sender, DoWorkEventArgs e)
        {
            var worker = sender as BackgroundWorker;

            while (true)
            {
                if (worker != null && worker.CancellationPending)
                {
                    e.Cancel = true;
                    return;
                }

                if (Processor.isRunning)
                {
                    if (ReceiverFull || Overrun)
                    {
                        Memory[Offset + 1] = (byte)(Memory[Offset + 1] | 0x80);
                        Interrupted = true;
                        Processor.InterruptRequest();
                    }

                    if (DataRead)
                    {
                        ReceiverFull = false;
                        Interrupted = false;
                        Overrun = false;
                        DataRead = false;
                    }
                }
            }
        }
#endregion
    }
}
```

Index