

WolfNet 6502 WorkBench Computer Emulator

beta

Generated by WolfNet Computing using Doxygen 1.9.5

1 Namespace Index	1
1.1 Namespace List	1
2 Hierarchical Index	2
2.1 Class Hierarchy	2
3 Class Index	4
3.1 Class List	4
4 File Index	5
4.1 File List	5
5 Namespace Documentation	7
5.1 Emulator Namespace Reference	7
5.2 Emulator.Model Namespace Reference	8
5.3 Emulator.ViewModel Namespace Reference	8
5.4 Hardware Namespace Reference	8
5.4.1 Enumeration Type Documentation	9
5.5 XamlGeneratedNamespace Namespace Reference	10
6 Class Documentation	10
6.1 Hardware.MemoryMap.Devices.ACIA Class Reference	10
6.1.1 Detailed Description	10
6.1.2 Member Data Documentation	10
6.2 Emulator.App Class Reference	11
6.2.1 Detailed Description	11
6.3 Hardware.AT28CXX Class Reference	11
6.3.1 Detailed Description	12
6.3.2 Constructor & Destructor Documentation	12
6.3.3 Member Function Documentation	12
6.3.4 Property Documentation	16
6.4 Hardware.MemoryMap.BankedRam Class Reference	18
6.4.1 Detailed Description	18
6.4.2 Member Data Documentation	18
6.4.3 Property Documentation	19
6.5 Hardware.MemoryMap.BankedRom Class Reference	19
6.5.1 Detailed Description	20
6.5.2 Member Data Documentation	20
6.5.3 Property Documentation	20
6.6 Emulator.Model.Breakpoint Class Reference	20
6.6.1 Detailed Description	21
6.6.2 Property Documentation	21
6.7 Emulator.Model.BreakpointType Class Reference	22
6.7.1 Detailed Description	22

6.7.2 Member Data Documentation	22
6.8 Hardware.MemoryMap.DeviceArea Class Reference	23
6.8.1 Detailed Description	23
6.8.2 Member Data Documentation	23
6.8.3 Property Documentation	24
6.9 Hardware.MemoryMap.Devices Class Reference	24
6.9.1 Detailed Description	24
6.10 Hardware.Disassembly Class Reference	25
6.10.1 Detailed Description	25
6.10.2 Property Documentation	25
6.11 Emulator.ExitCodes Class Reference	26
6.11.1 Detailed Description	26
6.11.2 Member Data Documentation	26
6.12 XamlGeneratedNamespace.GeneratedApplication Class Reference	28
6.12.1 Detailed Description	28
6.12.2 Member Function Documentation	28
6.12.3 Member Data Documentation	30
6.13 XamlGeneratedNamespace.GeneratedInternalTypeHelper Class Reference	30
6.13.1 Detailed Description	31
6.13.2 Member Function Documentation	31
6.14 Hardware.MemoryMap.Devices.GPIO Class Reference	34
6.14.1 Detailed Description	34
6.14.2 Member Data Documentation	34
6.15 Hardware.HM62256 Class Reference	35
6.15.1 Detailed Description	35
6.15.2 Constructor & Destructor Documentation	35
6.15.3 Member Function Documentation	36
6.15.4 Property Documentation	37
6.16 Emulator.IClosable Interface Reference	38
6.16.1 Detailed Description	39
6.16.2 Member Function Documentation	39
6.17 Emulator.ViewModel.MainViewModel Class Reference	39
6.17.1 Detailed Description	41
6.17.2 Constructor & Destructor Documentation	41
6.17.3 Member Function Documentation	43
6.17.4 Member Data Documentation	51
6.17.5 Property Documentation	51
6.18 Emulator.MainWindow Class Reference	57
6.18.1 Detailed Description	57
6.18.2 Constructor & Destructor Documentation	58
6.18.3 Member Function Documentation	58
6.18.4 Member Data Documentation	64

6.19 Hardware.MemoryMap Class Reference	64
6.19.1 Detailed Description	65
6.19.2 Member Function Documentation	65
6.19.3 Member Data Documentation	67
6.19.4 Property Documentation	68
6.20 Emulator.Model.MemoryRowModel Class Reference	69
6.20.1 Detailed Description	69
6.20.2 Property Documentation	70
6.21 Hardware.MemoryMap.Devices.MM65SIB Class Reference	73
6.21.1 Detailed Description	73
6.21.2 Member Data Documentation	73
6.22 Emulator.MultiThreadedObservableCollection< T > Class Template Reference	73
6.22.1 Detailed Description	74
6.22.2 Constructor & Destructor Documentation	74
6.22.3 Member Function Documentation	75
6.22.4 Event Documentation	75
6.23 Emulator.Model.OutputLog Class Reference	76
6.23.1 Detailed Description	76
6.23.2 Constructor & Destructor Documentation	77
6.23.3 Property Documentation	77
6.24 Emulator.Versioning.Product Class Reference	78
6.24.1 Detailed Description	78
6.24.2 Member Data Documentation	79
6.25 Hardware.Versioning.Product Class Reference	80
6.25.1 Detailed Description	80
6.25.2 Member Data Documentation	80
6.26 Emulator.Model.RomFileModel Class Reference	81
6.26.1 Detailed Description	82
6.26.2 Property Documentation	82
6.27 Emulator.SaveFile Class Reference	83
6.27.1 Detailed Description	83
6.27.2 Constructor & Destructor Documentation	83
6.27.3 Member Function Documentation	84
6.27.4 Member Data Documentation	85
6.28 Emulator.ViewModel.SaveFileViewModel Class Reference	86
6.28.1 Detailed Description	87
6.28.2 Constructor & Destructor Documentation	87
6.28.3 Member Function Documentation	87
6.28.4 Member Data Documentation	88
6.28.5 Property Documentation	88
6.29 Emulator.Settings Class Reference	89
6.29.1 Detailed Description	90

6.29.2 Constructor & Destructor Documentation	90
6.29.3 Member Function Documentation	90
6.29.4 Member Data Documentation	93
6.30 Emulator.SettingsFile Class Reference	93
6.30.1 Detailed Description	93
6.30.2 Member Function Documentation	93
6.31 Emulator.Versioning.SettingsFile Class Reference	94
6.31.1 Detailed Description	94
6.31.2 Member Data Documentation	94
6.32 Emulator.Model.SettingsModel Class Reference	94
6.32.1 Detailed Description	95
6.32.2 Property Documentation	95
6.33 Emulator.ViewModel.SettingsViewModel Class Reference	96
6.33.1 Detailed Description	97
6.33.2 Constructor & Destructor Documentation	97
6.33.3 Member Function Documentation	98
6.33.4 Member Data Documentation	98
6.33.5 Property Documentation	98
6.34 Hardware.MemoryMap.SharedRom Class Reference	100
6.34.1 Detailed Description	100
6.34.2 Member Data Documentation	100
6.34.3 Property Documentation	100
6.35 Emulator.Model.StateFileModel Class Reference	101
6.35.1 Detailed Description	101
6.35.2 Property Documentation	101
6.36 Hardware.Utility Class Reference	103
6.36.1 Detailed Description	103
6.36.2 Member Function Documentation	103
6.37 Emulator.Versioning Class Reference	108
6.37.1 Detailed Description	108
6.38 Emulator.ViewModel.ViewModelLocator Class Reference	108
6.38.1 Detailed Description	108
6.38.2 Constructor & Destructor Documentation	109
6.38.3 Member Function Documentation	109
6.38.4 Property Documentation	109
6.39 Hardware.W65C02 Class Reference	110
6.39.1 Detailed Description	113
6.39.2 Constructor & Destructor Documentation	113
6.39.3 Member Function Documentation	113
6.39.4 Member Data Documentation	147
6.39.5 Property Documentation	148
6.40 Hardware.W65C22 Class Reference	151

6.40.1 Detailed Description	153
6.40.2 Constructor & Destructor Documentation	153
6.40.3 Member Function Documentation	153
6.40.4 Member Data Documentation	157
6.40.5 Property Documentation	159
6.41 Hardware.W65C51 Class Reference	161
6.41.1 Detailed Description	163
6.41.2 Constructor & Destructor Documentation	163
6.41.3 Member Function Documentation	163
6.41.4 Member Data Documentation	173
6.41.5 Property Documentation	174
7 File Documentation	176
7.1 Emulator/App.xaml.cs File Reference	176
7.2 App.xaml.cs	176
7.3 Emulator/Classes/ExitCodes.cs File Reference	177
7.4 ExitCodes.cs	177
7.5 Emulator/Classes/FileLocations.cs File Reference	177
7.6 FileLocations.cs	177
7.7 Hardware/Classes/FileLocations.cs File Reference	178
7.8 FileLocations.cs	178
7.9 Emulator/Classes/SettingsFile.cs File Reference	178
7.10 SettingsFile.cs	178
7.11 Emulator/Classes/Versioning.cs File Reference	179
7.12 Versioning.cs	179
7.13 Hardware/Classes/Versioning.cs File Reference	179
7.14 Versioning.cs	180
7.15 Emulator/Interfaces/IClosable.cs File Reference	180
7.16 IClosable.cs	180
7.17 Emulator/MainWindow.xaml.cs File Reference	180
7.18 MainWindow.xaml.cs	181
7.19 Emulator/Model/Breakpoint.cs File Reference	182
7.20 Breakpoint.cs	182
7.21 Emulator/Model/BreakpointType.cs File Reference	182
7.22 BreakpointType.cs	183
7.23 Emulator/Model/MemoryRowModel.cs File Reference	183
7.24 MemoryRowModel.cs	183
7.25 Emulator/Model/OutputLog.cs File Reference	184
7.26 OutputLog.cs	185
7.27 Emulator/Model/RomFileModel.cs File Reference	185
7.28 RomFileModel.cs	186
7.29 Emulator/Model/SettingsModel.cs File Reference	186

7.30 SettingsModel.cs	186
7.31 Emulator/Model/StateFileModel.cs File Reference	187
7.32 StateFileModel.cs	187
7.33 Emulator/MultiThreadedCollection.cs File Reference	188
7.34 MultiThreadedCollection.cs	188
7.35 Emulator/obj/x86/Debug/.NETFramework,Version=v4.8.AssemblyAttributes.cs File Reference	189
7.36 .NETFramework,Version=v4.8.AssemblyAttributes.cs	189
7.37 Hardware/obj/Debug/.NETFramework,Version=v4.8.AssemblyAttributes.cs File Reference	189
7.38 .NETFramework,Version=v4.8.AssemblyAttributes.cs	189
7.39 Emulator/obj/x86/Debug/App.g.cs File Reference	190
7.40 App.g.cs	190
7.41 Emulator/obj/x86/Debug/App.g.i.cs File Reference	191
7.42 App.g.i.cs	191
7.43 Emulator/obj/x86/Debug/Emulator_Content.g.cs File Reference	192
7.44 Emulator_Content.g.cs	192
7.45 Emulator/obj/x86/Debug/Emulator_Content.g.i.cs File Reference	193
7.46 Emulator_Content.g.i.cs	193
7.47 Emulator/obj/x86/Debug/GeneratedInternalTypeHelper.g.cs File Reference	193
7.48 GeneratedInternalTypeHelper.g.cs	193
7.49 Emulator/obj/x86/Debug/GeneratedInternalTypeHelper.g.i.cs File Reference	194
7.50 GeneratedInternalTypeHelper.g.i.cs	194
7.51 Emulator/obj/x86/Debug/MainWindow.g.cs File Reference	195
7.52 MainWindow.g.cs	195
7.53 Emulator/obj/x86/Debug/MainWindow.g.i.cs File Reference	202
7.54 MainWindow.g.i.cs	202
7.55 Emulator/obj/x86/Debug/SaveFile.g.cs File Reference	209
7.56 SaveFile.g.cs	209
7.57 Emulator/obj/x86/Debug/SaveFile.g.i.cs File Reference	211
7.58 SaveFile.g.i.cs	211
7.59 Emulator/obj/x86/Debug/Settings.g.cs File Reference	213
7.60 Settings.g.cs	213
7.61 Emulator/obj/x86/Debug/Settings.g.i.cs File Reference	215
7.62 Settings.g.i.cs	215
7.63 Emulator/Properties/AssemblyInfo.cs File Reference	217
7.64 AssemblyInfo.cs	217
7.65 Hardware/Properties/AssemblyInfo.cs File Reference	218
7.66 AssemblyInfo.cs	218
7.67 Emulator/SaveFile.xaml.cs File Reference	218
7.68 SaveFile.xaml.cs	219
7.69 Emulator/Settings.xaml.cs File Reference	219
7.70 Settings.xaml.cs	219
7.71 Emulator/ViewModel/MainViewModel.cs File Reference	220

7.71.1 Typedef Documentation	220
7.72 MainViewModel.cs	221
7.73 Emulator/ViewModel/SaveFileViewModel.cs File Reference	230
7.74 SaveFileViewModel.cs	231
7.75 Emulator/ViewModel/SettingsViewModel.cs File Reference	232
7.76 SettingsViewModel.cs	232
7.77 Emulator/ViewModel/ViewModelLocator.cs File Reference	233
7.78 ViewModelLocator.cs	234
7.79 Hardware/AT28CXX.cs File Reference	235
7.80 AT28CXX.cs	235
7.81 Hardware/Classes/AddressingMode.cs File Reference	237
7.82 AddressingMode.cs	237
7.83 Hardware/Classes/Disassembly.cs File Reference	238
7.84 Disassembly.cs	239
7.85 Hardware/Classes/MemoryMap.cs File Reference	239
7.86 MemoryMap.cs	239
7.87 Hardware/Classes/Utility.cs File Reference	242
7.88 Utility.cs	242
7.89 Hardware/HM62256.cs File Reference	246
7.90 HM62256.cs	246
7.91 Hardware/W65C02.cs File Reference	248
7.92 W65C02.cs	248
7.93 Hardware/W65C22.cs File Reference	277
7.94 W65C22.cs	277
7.95 Hardware/W65C51.cs File Reference	281
7.96 W65C51.cs	281
Index	291

1 Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Emulator	7
Emulator.Model	8
Emulator.ViewModel	8
Hardware	8
XamlGeneratedNamespace	10

2 Hierarchical Index

2.1 Class Hierarchy

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

Hardware.MemoryMap.Devices.ACIA	10
Emulator.App	11
System.Windows.Application	
XamlGeneratedNamespace.GeneratedApplication	28
XamlGeneratedNamespace.GeneratedApplication	28
Hardware.AT28CXX	11
Hardware.MemoryMap.BankedRam	18
Hardware.MemoryMap.BankedRom	19
Emulator.Model.Breakpoint	20
Emulator.Model.BreakpointType	22
Hardware.MemoryMap.DeviceArea	23
Hardware.MemoryMap.Devices	24
Hardware.Disassembly	25
Emulator.Model.OutputLog	76
Emulator.ExitCodes	26
Hardware.MemoryMap.Devices.GPIO	34
Hardware.HM62256	35
Emulator.IClosable	38
Emulator.MainWindow	57
System.Windows.Markup.IComponentConnector	
Emulator.MainWindow	57
Emulator.MainWindow	57
Emulator.SaveFile	83
Emulator.SaveFile	83
Emulator.Settings	89
Emulator.Settings	89
System.Windows.Markup.InternalTypeHelper	
XamlGeneratedNamespace.GeneratedInternalTypeHelper	30
XamlGeneratedNamespace.GeneratedInternalTypeHelper	30

Hardware.MemoryMap	64
Emulator.Model.MemoryRowModel	69
Hardware.MemoryMap.Devices.MM65SIB	73
ObservableCollection	
Emulator.MultiThreadedObservableCollection< T >	73
Emulator.Versioning.Product	78
Hardware.Versioning.Product	80
Emulator.Model.RomFileModel	81
Emulator.SettingsFile	93
Emulator.Versioning.SettingsFile	94
Emulator.Model.SettingsModel	94
Hardware.MemoryMap.SharedRom	100
Emulator.Model.StateFileModel	101
Hardware.Utility	103
Emulator.Versioning	108
ViewModelBase	
Emulator.ViewModel.MainViewModel	39
Emulator.ViewModel.SaveFileViewModel	86
Emulator.ViewModel.SettingsViewModel	96
Emulator.ViewModel.ViewModelLocator	108
Hardware.W65C02	110
Hardware.W65C22	151
Hardware.W65C51	161
System.Windows.Window	
Emulator.MainWindow	57
Emulator.MainWindow	57
Emulator.SaveFile	83
Emulator.SaveFile	83
Emulator.Settings	89
Emulator.Settings	89
Window	
Emulator.MainWindow	57

3 Class Index

3.1 Class List

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

Hardware.MemoryMap.Devices.ACIA	10
Emulator.App Interaction logic for App.xaml	11
Hardware.AT28CXX An implementation of a W65C02 Processor.	11
Hardware.MemoryMap.BankedRam	18
Hardware.MemoryMap.BankedRom	19
Emulator.Model.Breakpoint A Representation of a Breakpoint	20
Emulator.Model.BreakpointType The Type of Breakpoint	22
Hardware.MemoryMap.DeviceArea	23
Hardware.MemoryMap.Devices	24
Hardware.Disassembly Used to help simulating. This class contains the disassembly properties.	25
Emulator.ExitCodes	26
XamlGeneratedNamespace.GeneratedApplication GeneratedApplication	28
XamlGeneratedNamespace.GeneratedInternalTypeHelper GeneratedInternalTypeHelper	30
Hardware.MemoryMap.Devices.GPIO	34
Hardware.HM62256	35
Emulator.IClosable	38
Emulator.ViewModel.MainViewModel The Main ViewModel	39
Emulator.MainWindow Interaction logic for MainWindow.xaml	57
Hardware.MemoryMap	64
Emulator.Model.MemoryRowModel A Model of a Single Page of memory	69
Hardware.MemoryMap.Devices.MM65SIB	73
Emulator.MultiThreadedObservableCollection< T > A MultiThreaedObservableCollection . This allows multiple threads to access the same observable collection in a safe manner.	73

Emulator.Model.OutputLog	
The OutputLog Model . Used by the outputlog grid to show a history of operations performed by the CPU	76
Emulator.Versioning.Product	78
Hardware.Versioning.Product	80
Emulator.Model.RomFileModel	
The Model used when Loading a Program.	81
Emulator.SaveFile	
SaveFile	83
Emulator.ViewModel.SaveFileViewModel	
The ViewModel Used by the SaveFileView	86
Emulator.Settings	
Settings	89
Emulator.SettingsFile	93
Emulator.Versioning.SettingsFile	94
Emulator.Model.SettingsModel	
Model that contains the required information needed to save the current settings to disk	94
Emulator.ViewModel.SettingsViewModel	
The ViewModel Used by the SaveFileView	96
Hardware.MemoryMap.SharedRom	100
Emulator.Model.StateFileModel	
Model that contains the required information needed to save the current state of the processor to disk	101
Hardware.Utility	103
Emulator.Versioning	108
Emulator.ViewModel.ViewModelLocator	
This class contains static references to all the view models in the application and provides an entry point for the bindings.	108
Hardware.W65C02	
An implementation of a W65C02 Processor.	110
Hardware.W65C22	
An implementation of a W65C22 VIA.	151
Hardware.W65C51	
An implementation of a W65C51 ACIA.	161

4 File Index

4.1 File List

Here is a list of all files with brief descriptions:

Emulator/ App.xaml.cs	176
Emulator/ MainWindow.xaml.cs	180
Emulator/ MultiThreadedCollection.cs	188
Emulator/ SaveFile.xaml.cs	218
Emulator/ Settings.xaml.cs	219
Emulator/Classes/ ExitCodes.cs	177
Emulator/Classes/ FileLocations.cs	177
Emulator/Classes/ SettingsFile.cs	178
Emulator/Classes/ Versioning.cs	179
Emulator/Interfaces/ IClosable.cs	180
Emulator/Model/ Breakpoint.cs	182
Emulator/Model/ BreakpointType.cs	182
Emulator/Model/ MemoryRowModel.cs	183
Emulator/Model/ OutputLog.cs	184
Emulator/Model/ RomFileModel.cs	185
Emulator/Model/ SettingsModel.cs	186
Emulator/Model/ StateFileModel.cs	187
Emulator/obj/x86/Debug/.NETFramework,Version=v4.8.AssemblyAttributes.cs	189
Emulator/obj/x86/Debug/ App.g.cs	190
Emulator/obj/x86/Debug/ App.g.i.cs	191
Emulator/obj/x86/Debug/ Emulator_Content.g.cs	192
Emulator/obj/x86/Debug/ Emulator_Content.g.i.cs	193
Emulator/obj/x86/Debug/ GeneratedInternalTypeHelper.g.cs	193
Emulator/obj/x86/Debug/ GeneratedInternalTypeHelper.g.i.cs	194
Emulator/obj/x86/Debug/ MainWindow.g.cs	195
Emulator/obj/x86/Debug/ MainWindow.g.i.cs	202
Emulator/obj/x86/Debug/ SaveFile.g.cs	209
Emulator/obj/x86/Debug/ SaveFile.g.i.cs	211
Emulator/obj/x86/Debug/ Settings.g.cs	213
Emulator/obj/x86/Debug/ Settings.g.i.cs	215
Emulator/Properties/ AssemblyInfo.cs	217
Emulator/ViewModel/ MainViewModel.cs	220

Emulator/ViewModel/ SaveFileViewModel.cs	230
Emulator/ViewModel/ SettingsViewModel.cs	232
Emulator/ViewModel/ ViewModelLocator.cs	233
Hardware/ AT28CXX.cs	235
Hardware/ HM62256.cs	246
Hardware/ W65C02.cs	248
Hardware/ W65C22.cs	277
Hardware/ W65C51.cs	281
Hardware/Classes/ AddressingMode.cs	237
Hardware/Classes/ Disassembly.cs	238
Hardware/Classes/ FileLocations.cs	178
Hardware/Classes/ MemoryMap.cs	239
Hardware/Classes/ Utility.cs	242
Hardware/Classes/ Versioning.cs	179
Hardware/obj/Debug/.NETFramework,Version=v4.8.AssemblyAttributes.cs	189
Hardware/Properties/ AssemblyInfo.cs	218

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

The addressing modes used by the 6502 Processor

Definition at line 6 of file AddressingMode.cs.

```

00007 {
00008 /// <summary>
00009 /// In this mode a full address is given to operation on IE: Memory byte[] { 0x60, 0x00, 0xFF }
00010 /// would perform an ADC operation and Add the value at ADDRESS 0xFF00 to the accumulator.
00011 /// The address is always LSB first
00012 /// </summary>
00013 Absolute = 1,
00014 /// <summary>
00015 /// In this mode a full address is given to operation on IE: Memory byte[] { 0x7D, 0x00, 0xFF } The
00016 /// full value would then be added to the X Register.
00017 /// If the X register was 0x01 then the address would be 0xFF01. and the value stored there would
00018 /// have an ADC operation performed on it and the value would
00019 /// be added to the accumulator.
00020 /// </summary>
00021 AbsoluteX = 2,
00022 /// <summary>
00023 /// In this mode a full address is given to operation on IE: Memory byte[] { 0x79, 0x00, 0xFF } The
00024 /// full value would then be added to the Y Register.
00025 /// If the Y register was 0x01 then the address would be 0xFF01. and the value stored there would
00026 /// have an ADC operation performed on it and the value would
00027 /// be added to the accumulator
00028 /// </summary>
00029 AbsoluteY = 3,
00030 /// <summary>
00031 /// In this mode the instruction operates on the accumulator. No operands are needed.
00032 /// </summary>
00033 Accumulator = 4,
00034 /// <summary>
00035 /// In this mode, the value to operate on immediately follows the instruction. IE: Memory byte[] {
00036 /// 0x69, 0x01 }
00037 /// would perform an ADC operation and Add 0x01 directly to the accumulator
00038 /// </summary>
00039 Immediate = 5,
00040 /// <summary>
00041 /// No address is needed for this mode. EX: BRK (Break), CLC (Clear Carry Flag) etc
00042 /// </summary>
00043 Implied = 6,
00044 /// <summary>
00045 /// In this mode assume the following
00046 /// Memory = { 0x61, 0x02, 0x04, 0x00, 0x03 }
00047 /// RegisterX = 0x01
00048 /// 1. Take the sum of the X Register and the value after the opcode 0x01 + 0x01 = 0x02.
00049 /// 2. Starting at position 0x02 get an address (0x04, 0x00) = 0x0004
00050 /// 3. Perform the ADC operation and Add the value at 0x0005 to the accumulator
00051 /// Note: if the Zero Page address is greater than 0xff then roll over the value. IE 0x101 rolls
00052 /// over to 0x01
00053 /// </summary>
00054 IndirectX = 7,
00055 /// <summary>
00056 /// In this mode assume the following
00057 /// Memory = { 0x61, 0x02, 0x04, 0x00, 0x03 }
00058 /// RegisterY = 0x01
00059 /// 1. Starting at position 0x02 get an address (0x04, 0x00) = 0x0004
00060 /// 2. Take the sum of the Y Register and the absolute address 0x01 + 0x0004 = 0x0005
00061 /// 3. Perform the ADC operation and Add the value at 0x0005 to the accumulator
00062 /// Note: if the address is great that 0xffff then roll over IE: 0x10001 rolls over to 0x01
00063 /// </summary>
00064 IndirectY = 8,
00065 /// <summary>
00066 /// JMP is the only operation that uses this mode. In this mode an absolute address is specified that
00067 /// points to the location of the absolute address we want to jump to.
00068 /// </summary>

```



```

00062         Indirect = 9,
00063     /// <summary>
00064     /// This Mode Changes the PC. It allows the program to change the location of the PC by 127 in either
    direction.
00065     /// </summary>
00066         Relative = 10,
00067     /// <summary>
00068     /// 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 }
00069     /// would perform an ADC operation and Add 0x01 directly to the Accumulator
00070     /// </summary>
00071         ZeroPage = 11,
00072     /// <summary>
00073     /// 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 }
00074     /// In this example we store a value of 0x01 into the X register, then we would perform an ADC
    operation using the address of 0x04+0x01=0x05 and Add the result of 0x01 directly to the Accumulator
00075     /// </summary>
00076         ZeroPageX = 12,
00077     /// <summary>
00078     /// This works the same as ZeroPageX except it uses the Y register instead of the X register.
00079     /// </summary>
00080         ZeroPageY = 13,
00081     }

```

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](#).

```
00055     {
00056         Memory = new byte[banks][];
00057         for (int i = 0; i < banks; i++)
00058         {
00059             Memory[i] = new byte[length + 1];
00060         }
00061         Offset = offset;
00062         Length = length;
00063         Banks = banks;
00064         CurrentBank = 0;
00065     }
```

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](#).

```
00166     {
00167         for (byte i = 0; i < Banks; i++)
00168         {
00169             for (int j = 0; j < Length; j++)
00170             {
00171                 Memory[i][j] = 0x00;
00172             }
00173         }
00174     }
```

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](#).

```
00144     {
00145         return Memory;
00146     }
```

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

Dumps the selected ROM bank.

Parameters

<i>bank</i>	The bank to dump data from.
-------------	-----------------------------

Returns

Array that represents the selected ROM bank.

Definition at line 153 of file [AT28CXX.cs](#).

```
00154     {
00155         byte[] _tempMemory = new byte[MemoryMap.BankedRom.Length + 1];
00156         for (var i = 0; i < MemoryMap.BankedRom.Length; i++) {
00157             _tempMemory[i] = Memory[bank][i];
00158         }
00159         return _tempMemory;
00160     }
```

6.3.3.4 Load() [1/2] `void Hardware.AT28CXX.Load (`
 `byte bank,`
 `byte[] data) [inline]`

Loads a program into ROM.

Parameters

<i>bank</i>	The bank to load data to.
<i>data</i>	The data to be loaded to ROM.

Definition at line 84 of file [AT28CXX.cs](#).

```
00085     {
00086         for (int i = 0; i <= Length; i++)
00087         {
00088             Memory[bank][i] = data[i];
00089         }
00090     }
```

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

Loads a program into ROM.

Parameters

<i>data</i>	The program to be loaded
-------------	--------------------------

Definition at line 71 of file [AT28CXX.cs](#).

```
00072     {
00073         for (byte i = 0; i < Banks; i++)
00074         {
00075             Load(i, data[i]);
00076         }
00077     }
```

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

<i>bank</i>	The bank to read data from.
<i>address</i>	

Returns

the byte being returned

Definition at line 121 of file [AT28CXX.cs](#).

```
00122     {
00123         return Memory[CurrentBank][address - Offset];
00124     }
```

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

Definition at line 92 of file [AT28CXX.cs](#).

```
00093     {
00094         byte[][] bios = new byte[Banks][];
00095         try
00096         {
00097             FileStream file = new FileStream(filename, FileMode.Open, FileAccess.Read);
00098             for (int i = 0; i < Banks; i++)
00099             {
00100                 bios[i] = new byte[Length + 1];
00101                 for (int j = 0; j <= Length; j++)
00102                 {
00103                     bios[i][j] = new byte();
00104                     bios[i][j] = (byte)file.ReadByte();
00105                 }
00106             }
00107         }
00108         catch (Exception)
00109         {
00110             return null;
00111         }
00112         return bios;
00113     }
```

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

<i>bank</i>	The bank to load data to.
<i>address</i>	The address to write data to
<i>data</i>	The data to write

Definition at line 132 of file [AT28CXX.cs](#).

```
00133     {
00134         _ = address;
00135         _ = data;
00136         return;
00137     }
```

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](#).

```
00022 { 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](#).

```
00026     {
00027         get { return BreakpointType.AllTypes; }
00028     }
```

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](#).

```
00013 { 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](#).

```
00023 { 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 ProgramCounter [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 ProgramCounter [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](#).

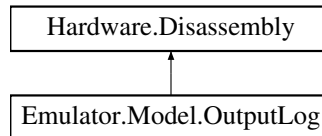
The documentation for this class was generated from the following file:

- [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](#).

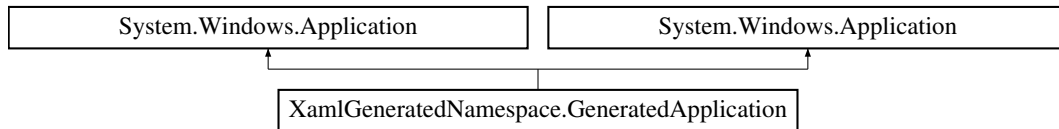
The documentation for this class was generated from the following file:

- Emulator/Classes/[ExitCodes.cs](#)

6.12 XamlGeneratedNamespace.GeneratedApplication Class Reference

GeneratedApplication

Inheritance diagram for XamlGeneratedNamespace.GeneratedApplication:



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.InitializeComponent () [inline]

InitializeComponent

Definition at line 50 of file [App.g.cs](#).

```
00050 {
00051     if (_contentLoaded) {
00052         return;
00053     }
00054     _contentLoaded = true;
00055
00056 #line 2 "..\..\..\App.xaml"
00057     this.StartupUri = new System.Uri("MainWindow.xaml", System.UriKind.Relative);
00058
00059 #line default
00060 #line hidden
00061     System.Uri resourceLocator = new System.Uri("/Emulator;component/app.xaml",
        System.UriKind.Relative);
00062
00063 #line 1 "..\..\..\App.xaml"
00064     System.Windows.Application.LoadComponent(this, resourceLocator);
00065
00066 #line default
00067 #line hidden
00068 }
```

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

InitializeComponent

Definition at line 50 of file [App.g.i.cs](#).

```
00050 {
00051     if (_contentLoaded) {
00052         return;
00053     }
00054     _contentLoaded = true;
00055
00056 #line 2 "..\..\..\App.xaml"
00057     this.StartupUri = new System.Uri("MainWindow.xaml", System.UriKind.Relative);
00058
00059 #line default
00060 #line hidden
00061     System.Uri resourceLocator = new System.Uri("/Emulator;component/app.xaml",
        System.UriKind.Relative);
00062
00063 #line 1 "..\..\..\App.xaml"
00064     System.Windows.Application.LoadComponent(this, resourceLocator);
00065
00066 #line default
00067 #line hidden
00068 }
```

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](#).

```
00076 {
00077     SplashScreen splashScreen = new SplashScreen("splashscreen.png");
00078     splashScreen.Show(true);
00079     XamlGeneratedNamespace.GeneratedApplication app = new
        XamlGeneratedNamespace.GeneratedApplication();
00080     app.InitializeComponent();
00081     app.Run();
00082 }
```

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](#).

```
00076      {
00077          SplashScreen splashScreen = new SplashScreen("splashscreen.png");
00078          splashScreen.Show(true);
00079          XamlGeneratedNamespace.GeneratedApplication app = new
XamlGeneratedNamespace.GeneratedApplication();
00080          app.InitializeComponent();
00081          app.Run();
00082      }
```

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](#).

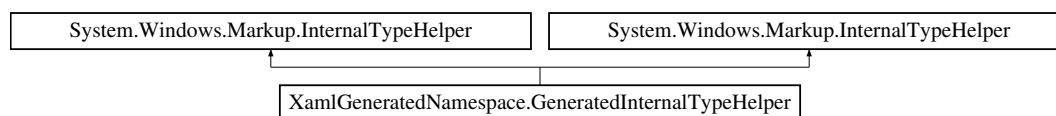
The documentation for this class was generated from the following files:

- 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:



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.GeneratedInternalTypeHelper.AddEventHandler (
 System.Reflection.EventInfo eventInfo,
 object target,
 System.Delegate handler) [inline], [protected]

AddEventHandler

Definition at line 57 of file [GeneratedInternalTypeHelper.g.cs](#).

```
00057 {
00058     eventInfo.AddEventHandler(target, handler);
00059 }
```

6.13.2.2 AddEventHandler() [2/2] override void XamlGeneratedNamespace.GeneratedInternalTypeHelper.AddEventHandler (
 System.Reflection.EventInfo eventInfo,
 object target,
 System.Delegate handler) [inline], [protected]

AddEventHandler

Definition at line 57 of file [GeneratedInternalTypeHelper.g.i.cs](#).

```
00057 {
00058     eventInfo.AddEventHandler(target, handler);
00059 }
```

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

CreateDelegate

Definition at line 47 of file [GeneratedInternalTypeHelper.g.cs](#).

```
00047 {
00048     return ((System.Delegate) (target.GetType().InvokeMember("_CreateDelegate",
00049         (System.Reflection.BindingFlags.InvokeMethod
00049         | (System.Reflection.BindingFlags.NonPublic |
00050         System.Reflection.BindingFlags.Instance)), null, target, new object[] {
00050             delegateType,
00051             handler}, null)));
00052 }
```

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

CreateDelegate

Definition at line 47 of file [GeneratedInternalTypeHelper.g.i.cs](#).

```
00047 {
00048     return ((System.Delegate) (target.GetType().InvokeMember("_CreateDelegate",
00049         (System.Reflection.BindingFlags.InvokeMethod
00049         | (System.Reflection.BindingFlags.NonPublic |
00050         System.Reflection.BindingFlags.Instance)), null, target, new object[] {
00050             delegateType,
00051             handler}, null)));
00052 }
```

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

CreateInstance

Definition at line 25 of file [GeneratedInternalTypeHelper.g.cs](#).

```
00025 {
00026     return System.Activator.CreateInstance(type, ((System.Reflection.BindingFlags.Public |
00027         System.Reflection.BindingFlags.NonPublic)
00027         | (System.Reflection.BindingFlags.Instance |
00028         System.Reflection.BindingFlags.CreateInstance)), null, null, culture);
00028 }
```

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](#).

```
00025 {  
00026     return System.Activator.CreateInstance(type, ((System.Reflection.BindingFlags.Public |  
System.Reflection.BindingFlags.NonPublic)  
00027     | (System.Reflection.BindingFlags.Instance |  
System.Reflection.BindingFlags.CreateInstance)), null, null, culture);  
00028 }
```

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](#).

```
00033 {  
00034     return propertyInfo.GetValue(target, System.Reflection.BindingFlags.Default, null, null,  
culture);  
00035 }
```

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](#).

```
00033 {  
00034     return propertyInfo.GetValue(target, System.Reflection.BindingFlags.Default, null, null,  
culture);  
00035 }
```

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](#).

```
00040 {  
00041     propertyInfo.SetValue(target, value, System.Reflection.BindingFlags.Default, null, null,  
culture);  
00042 }
```


6.13.2.10 SetPropertyValue() [2/2] `override void XamlGeneratedNamespace.GeneratedInternalTypeHelper.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](#).

```
00040 {  
00041     propertyInfo.SetValue(target, value, System.Reflection.BindingFlags.Default, null, null,  
00042     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

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

Definition at line 43 of file [HM62256.cs](#).

```

00044     {
00045         Memory = new byte[banks][];
00046         for (int i = 0; i < banks; i++)
00047         {
00048             Memory[i] = new byte[length + 1];
00049         }
00050         Length = length;
00051         Banks = banks;
00052         Offset = offset;
00053         CurrentBank = 0;
00054     }

```

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](#).

```

00068     {
00069         for (var i = 0; i < Banks; i++)
00070         {
00071             for (var j = 0; j < Memory.Length; j++)
00072             {
00073                 Memory[i][j] = 0x00;
00074             }
00075         }
00076     }

```

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](#).

```

00105     {
00106         return Memory;
00107     }

```

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

<i>bank</i>	The bank to read data from.
<i>address</i>	

Returns

The byte being read.

Definition at line 84 of file [HM62256.cs](#).

```
00085     {
00086         return Memory[CurrentBank][address - Offset];
00087     }
```

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](#).

```
00060     {
00061         Clear();
00062     }
```

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

<i>bank</i>	The bank to load data to.
<i>address</i>	The address to write data to
<i>data</i>	The data to write

Definition at line 95 of file [HM62256.cs](#).

```
00096     {
00097         Memory[CurrentBank][address - Offset] = data;
00098     }
```

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](#).

```
00030 { 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](#).

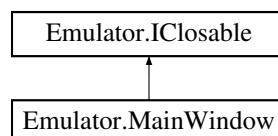
```
00015 { get; set; }
```

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 ( )
```

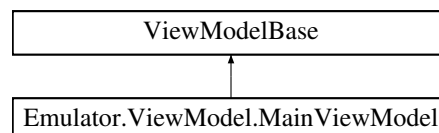
The documentation for this interface was generated from the following file:

- [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 Program 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 program 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](#).

```

00232     {
00233         var _formatter = new XmlSerializer(typeof(SettingsModel));
00234         Stream _stream = new FileStream(FileLocations.SettingsFile, FileMode.OpenOrCreate);
00235         if (!(_stream == null) || (0 >= _stream.Length))
00236         {
00237             SettingsModel = (SettingsModel)_formatter.Deserialize(_stream);
00238             if ((SettingsModel.SettingsVersionMajor < Versioning.SettingsFile.Major) ||
00239                 (SettingsModel.SettingsVersionMinor < Versioning.SettingsFile.Minor) ||
00240                 (SettingsModel.SettingsVersionBuild < Versioning.SettingsFile.Build) ||
00241                 (SettingsModel.SettingsVersionRevision < Versioning.SettingsFile.Revision))
00242             {
00243                 #if !DEBUG
00244                     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));
00245                 #else
00246                     MessageBox.Show("Settings file contains old information...\nDeleting old settings
file...",
00247                                     "Settings file stale!", MessageBoxButton.OKCancel,
00248                                     MessageBoxImage.Warning,
00249                                     MessageBoxResult.OK);
00250                     // Close the file, then delete it.
00251                     _stream.Close();
00252                     File.Delete(FileLocations.SettingsFile);
00253                     SettingsModel = SettingsFile.CreateNew();
00254                 #endif
00255             }
00256             else
00257             {
00258                 MessageBox.Show("Creating new settings file...");
00259                 SettingsModel = SettingsFile.CreateNew();
00260             }
00261             _stream.Close();
00262
00263             HM62256 = new HM62256(MemoryMap.BankedRam.TotalBanks, MemoryMap.BankedRam.Offset,
MemoryMap.BankedRam.Length);
00264             AT28C64 = new AT28CXX(MemoryMap.SharedRom.Offset, MemoryMap.SharedRom.Length, 1);
00265             AT28C010 = new AT28CXX(MemoryMap.BankedRom.Offset, MemoryMap.BankedRom.Length,
MemoryMap.BankedRom.TotalBanks);
00266             W65C02 = new W65C02();
00267             W65C51 = new W65C51(W65C02, MemoryMap.Devices.ACIA.Offset);
00268             W65C51.Init(SettingsModel.ComPortName.ToString());
00269             W65C22 = new W65C22(W65C02, MemoryMap.Devices.GPIO.Offset, MemoryMap.Devices.GPIO.Length);
00270             W65C22.Init(1000);
00271             MM65SIB = new W65C22(W65C02, MemoryMap.Devices.MM65SIB.Offset,
MemoryMap.Devices.MM65SIB.Length);
00272             MM65SIB.Init(1000);
00273
00274             MemoryMap.Init(W65C02, W65C22, MM65SIB, W65C51, HM62256, AT28C010, AT28C64);
00275
00276             // Now we can load the BIOS.
00277             byte[][] _bios = AT28C64.ReadFile(FileLocations.BiosFile);
00278             if (_bios == null)
00279             {
00280                 Environment.Exit(ExitCodes.NO_BIOS);
00281             }
00282             AT28C64.Load(_bios);
00283
00284             AboutCommand = new RelayCommand(About);
00285             AddBreakPointCommand = new RelayCommand(AddBreakPoint);
00286             CloseCommand = new RelayCommand<IClosable>(Close);
00287             RemoveBreakPointCommand = new RelayCommand(RemoveBreakPoint);
00288             ResetCommand = new RelayCommand(Reset);
00289             RunPauseCommand = new RelayCommand(RunPause);
00290             SettingsCommand = new RelayCommand(Settings);
00291             StepCommand = new RelayCommand(Step);
00292             UpdateMemoryPageCommand = new RelayCommand(UpdateMemoryPage);
00293
00294             Messenger.Default.Register<NotificationMessage>(this, GenericNotification);
00295             Messenger.Default.Register<NotificationMessage>(this, RomFileModel,
BinaryLoadedNotification);
00296             Messenger.Default.Register<NotificationMessage>(this, SettingsModel,
SettingsAppliedNotification);
00297             Messenger.Default.Register<NotificationMessage>(this, StateFileModel,
StateLoadedNotification);
00298
00299             MemoryPage = new MultiThreadedObservableCollection<MemoryRowModel>();
00300             OutputLog = new MultiThreadedObservableCollection<OutputLog>();
00301             Breakpoints = new MultiThreadedObservableCollection<Breakpoint>();

```

```

00302
00303         UpdateMemoryPage();
00304
00305         _backgroundWorker = new BackgroundWorker { WorkerSupportsCancellation = true,
WorkerReportsProgress = false };
00306         _backgroundWorker.DoWork += BackgroundWorkerDoWork;
00307         Application.Current.MainWindow.Closing += new CancelEventHandler(OnClose);
00308         Application.Current.MainWindow.Loaded += new RoutedEventHandler(OnLoad);
00309
00310         Reset();
00311     }

```

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](#).

```

00757     {
00758         IsRunning = false;
00759
00760         if (_backgroundWorker.IsBusy)
00761             _backgroundWorker.CancelAsync();
00762
00763         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);
00764     }

```

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

Definition at line 776 of file [MainViewModel.cs](#).

```

00777     {
00778         Breakpoints.Add(new Breakpoint());
00779         RaisePropertyChanged("Breakpoints");
00780     }

```

6.17.3.3 BackgroundWorkerDoWork() void Emulator.ViewModel.MainViewModel.BackgroundWorkerDo←
Work (

object sender,
DoWorkEventArgs e) [inline], [private]

Definition at line 632 of file [MainViewModel.cs](#).

```

00633     {
00634         var worker = sender as BackgroundWorker;
00635         var outputLogs = new List<OutputLog>();
00636
00637         while (true)
00638         {
00639             if (worker != null && worker.CancellationPending || IsBreakPointTriggered())
00640             {
00641                 e.Cancel = true;
00642
00643                 RaisePropertyChanged("W65C02");
00644
00645                 foreach (var log in outputLogs)
00646                     OutputLog.Insert(0, log);
00647
00648                 UpdateMemoryPage();
00649                 return;
00650             }

```

```

00651
00652         StepProcessor();
00653         outputLogs.Add(GetOutputLog());
00654
00655         if (NumberOfCycles % GetLogModValue() == 0)
00656         {
00657             foreach (var log in outputLogs)
00658                 OutputLog.Insert(0, log);
00659
00660             outputLogs.Clear();
00661             UpdateUi();
00662         }
00663         Thread.Sleep(GetSleepValue());
00664     }
00665 }

```

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

Definition at line 374 of file [MainViewModel.cs](#).

```

00375     {
00376         if (notificationMessage.Notification != "FileLoaded")
00377         {
00378             return;
00379         }
00380
00381         // Load Banked ROM
00382         AT28C010.Load(notificationMessage.Content.Rom);
00383         IsRomLoaded = true;
00384         RaisePropertyChanged("IsRomLoaded");
00385
00386         Reset();
00387     }

```

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

Definition at line 366 of file [MainViewModel.cs](#).

```

00367     {
00368         if ((window != null) && (!IsRunning))
00369         {
00370             Environment.Exit(ExitCodes.NO_ERROR);
00371         }
00372     }

```

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

Definition at line 416 of file [MainViewModel.cs](#).

```

00417     {
00418         if (notificationMessage.Notification == "CloseFile")
00419         {
00420             AT28C010.Clear();
00421             if (IsRunning) { RunPause(); }
00422             IsRomLoaded = false;
00423             RaisePropertyChanged("IsRomLoaded");
00424             return;
00425         }
00426         else if (notificationMessage.Notification == "LoadFile")
00427         {
00428             var dialog = new OpenFileDialog { DefaultExt = ".bin", Filter =
00429                 "All Files (*.bin, *.65C02)|*.bin;*.65C02|Binary
Assembly (*.bin)|" +

```

```

00430                                     "*.bin|WolfNet 65C02 Emulator Save State
(*.65C02)|*.65C02" };
00431         var result = dialog.ShowDialog();
00432         if (result != true)
00433         {
00434             return;
00435         }
00436
00437         if (Path.GetExtension(dialog.FileName.ToUpper()) == ".BIN")
00438         {
00439             byte[][] _rom = AT28C010.ReadFile(dialog.FileName);
00440
00441             Messenger.Default.Send(new NotificationMessage<RomFileModel>(new RomFileModel
00442             {
00443                 Rom = _rom,
00444                 RomBanks = AT28C010.Banks,
00445                 RomBankSize = AT28C010.Length,
00446                 RomFilePath = dialog.FileName,
00447                 RomFileName = Path.GetFileName(dialog.FileName),
00448                 }, "FileLoaded"));
00449             }
00450             else if (Path.GetExtension(dialog.FileName.ToUpper()) == ".6502")
00451             {
00452                 var formatter = new BinaryFormatter();
00453                 Stream stream = new FileStream(dialog.FileName, FileMode.Open);
00454                 var fileModel = (StateFileModel)formatter.Deserialize(stream);
00455
00456                 stream.Close();
00457
00458                 Messenger.Default.Send(new NotificationMessage<StateFileModel>(fileModel,
"StateLoaded"));
00459             }
00460         }
00461         else if (notificationMessage.Notification == "SaveState")
00462         {
00463             var dialog = new SaveFileDialog { DefaultExt = ".65C02", Filter =
00464                 (*.65C02)|*.65C02" };
00465             var result = dialog.ShowDialog();
00466
00467             if (result != true)
00468             {
00469                 return;
00470             }
00471
00472             var formatter = new BinaryFormatter();
00473             Stream stream = new FileStream(dialog.FileName, FileMode.Create, FileAccess.Write,
FileShare.None);
00474
00475             formatter.Serialize(stream, new StateFileModel
00476             {
00477                 NumberOfCycles = NumberOfCycles,
00478                 OutputLog = OutputLog,
00479                 W65C02 = W65C02,
00480                 W65C22 = W65C22,
00481                 MM65SIB = MM65SIB,
00482                 W65C51 = W65C51,
00483                 AT28C010 = AT28C010,
00484                 AT28C64 = AT28C64,
00485             });
00486             stream.Close();
00487         }
00488         else
00489         {
00490             return;
00491         }
00492     }

```

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

Definition at line 699 of file [MainViewModel.cs](#).

```

00700     {
00701         switch (CpuSpeed)
00702         {
00703             case 0:
00704             case 1:
00705             case 2:
00706             case 3:
00707             case 4:

```

```

00708         case 5:
00709             return 1;
00710         case 6:
00711             return 5;
00712         case 7:
00713             return 20;
00714         case 8:
00715             return 30;
00716         case 9:
00717             return 40;
00718         case 10:
00719             return 50;
00720         default:
00721             return 5;
00722     }
00723 }

```

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

Definition at line 601 of file [MainViewModel.cs](#).

```

00602     {
00603         if (W65C02.CurrentDisassembly == null)
00604         {
00605             return new OutputLog(new Disassembly());
00606         }
00607
00608         return new OutputLog(W65C02.CurrentDisassembly)
00609         {
00610             XRegister = W65C02.XRegister.ToString("X").PadLeft(2, '0'),
00611             YRegister = W65C02.YRegister.ToString("X").PadLeft(2, '0'),
00612             Accumulator = W65C02.Accumulator.ToString("X").PadLeft(2, '0'),
00613             NumberOfCycles = NumberOfCycles,
00614             StackPointer = W65C02.StackPointer.ToString("X").PadLeft(2, '0'),
00615             ProgramCounter = W65C02.ProgramCounter.ToString("X").PadLeft(4, '0'),
00616             CurrentOpCode = W65C02.CurrentOpCode.ToString("X").PadLeft(2, '0')
00617         };
00618     }

```

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

Definition at line 725 of file [MainViewModel.cs](#).

```

00726     {
00727         switch (CpuSpeed)
00728         {
00729             case 0:
00730                 return 550;
00731             case 1:
00732                 return 550;
00733             case 2:
00734                 return 440;
00735             case 3:
00736                 return 330;
00737             case 4:
00738                 return 220;
00739             case 5:
00740                 return 160;
00741             case 6:
00742                 return 80;
00743             case 7:
00744                 return 40;
00745             case 8:
00746                 return 20;
00747             case 9:
00748                 return 10;
00749             case 10:
00750                 return 5;
00751             default:
00752                 return 5;
00753         }
00754     }

```

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

Definition at line 667 of file [MainViewModel.cs](#).

```

00668     {
00669         //This prevents the Run Command from getting stuck after reaching a breakpoint
00670         if (_breakpointTriggered)
00671         {
00672             _breakpointTriggered = false;
00673             return false;
00674         }
00675
00676         foreach (var breakpoint in Breakpoints.Where(x => x.IsEnabled))
00677         {
00678             if (!int.TryParse(breakpoint.Value, NumberStyles.AllowHexSpecifier,
00679                 CultureInfo.InvariantCulture, out int value))
00680                 continue;
00681
00682             if (breakpoint.Type == BreakpointType.NumberOfCycleType && value == NumberOfCycles)
00683             {
00684                 _breakpointTriggered = true;
00685                 RunPause();
00686                 return true;
00687             }
00688
00689             if (breakpoint.Type == BreakpointType.ProgramCounterType && value ==
00690                 W65C02.ProgramCounter)
00691             {
00692                 _breakpointTriggered = true;
00693                 RunPause();
00694                 return true;
00695             }
00696
00697             return false;
00698         }
00699     }

```

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

Definition at line 332 of file [MainViewModel.cs](#).

```

00333     {
00334         e.Cancel = false;
00335         if (IsRunning)
00336         {
00337             MessageBox.Show("You can't quit the emulator while it is actively running!",
00338                 "You can't do that!", MessageBoxButton.OK, MessageBoxImage.Stop);
00339             e.Cancel = true;
00340             return;
00341         }
00342         #if !DEBUG
00343         else
00344         {
00345             var result = MessageBox.Show( "Are you sure you want to quit the emulator?",
00346                 "To quit, or not to quit -- that is the question.",
00347                 MessageBoxButton.YesNo, MessageBoxImage.Question,
00348                 MessageBoxResult.No);
00349             if (result == MessageBoxResult.No)
00350             {
00351                 e.Cancel = true;
00352                 return;
00353             }
00354         }
00355         #endif
00356         Stream stream = new FileStream(FileLocations.SettingsFile, FileMode.Create,
00357             FileAccess.Write, FileShare.None);
00358         XmlSerializer XmlFormatter = new XmlSerializer(typeof(SettingsModel));
00359         XmlFormatter.Serialize(stream, MainViewModel.SettingsModel);
00360         stream.Flush();
00361         stream.Close();
00362         W65C51.Fini();
00363     }

```

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

Definition at line 313 of file [MainViewModel.cs](#).

```
00314     {
00315     #if !DEBUG
00316         if (Versioning.Product.Major < 1)
00317         {
00318             var result = MessageBox.Show(String.Format("Thank you for using {0}\n" +
00319                                                         "Be warned that this is a beta build.\n" +
00320                                                         "It may break or have bugs.",
00321                                                         Versioning.Product.Name),
00322                                                         Versioning.Product.Title,
00323                                                         MessageBoxButton.OKCancel,
00324                                                         MessageBoxImage.Warning,
00325                                                         MessageBoxResult.None);
00326             if (result == MessageBoxResult.Cancel)
00327             {
00328                 // Exit without making any changes.
00329                 Environment.Exit(ExitCodes.NO_ERROR);
00330             }
00331         }
00332     #endif
00333     }
```

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

Definition at line 782 of file [MainViewModel.cs](#).

```
00783     {
00784         if (SelectedBreakpoint == null)
00785             return;
00786
00787         Breakpoints.Remove(SelectedBreakpoint);
00788         SelectedBreakpoint = null;
00789         RaisePropertyChanged("SelectedBreakpoint");
00790     }
```

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

Definition at line 540 of file [MainViewModel.cs](#).

```
00541     {
00542         IsRunning = false;
00543
00544         if (_backgroundWorker.IsBusy)
00545             _backgroundWorker.CancelAsync();
00546
00547         // "Reset" the Hardware...
00548         W65C02.Reset();
00549         RaisePropertyChanged("W65C02");
00550         W65C22.Reset();
00551         RaisePropertyChanged("W65C22");
00552         MM65SIB.Reset();
00553         RaisePropertyChanged("MM65SIB");
00554         W65C51.Reset();
00555         RaisePropertyChanged("W65C51");
00556         HM62256.Reset();
00557         RaisePropertyChanged("HM62256");
00558
00559         IsRunning = false;
00560         NumberOfCycles = 0;
00561         RaisePropertyChanged("NumberOfCycles");
00562
00563         UpdateMemoryPage();
00564         RaisePropertyChanged("MemoryPage");
00565
00566         OutputLog.Clear();
00567         RaisePropertyChanged("CurrentDisassembly");
00568
00569         OutputLog.Insert(0, GetOutputLog());
00570         UpdateUi();
00571     }
```

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

Definition at line 620 of file [MainViewModel.cs](#).

```
00621     {
00622         var isRunning = !IsRunning;
00623
00624         if (isRunning)
00625             _backgroundWorker.RunWorkerAsync();
00626         else
00627             _backgroundWorker.CancelAsync();
00628
00629         IsRunning = !IsRunning;
00630     }
```

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

Definition at line 766 of file [MainViewModel.cs](#).

```
00767     {
00768         IsRunning = false;
00769
00770         if (_backgroundWorker.IsBusy)
00771             _backgroundWorker.CancelAsync();
00772
00773         Messenger.Default.Send(new NotificationMessage<SettingsModel>(SettingsModel,
00774             "SettingsWindow"));
00775     }
```

6.17.3.17 SettingsAppliedNotification() void Emulator.ViewModel.MainViewModel.SettingsApplied↵
Notification (
NotificationMessage< SettingsModel > notificationMessage) [inline], [private]

Definition at line 494 of file [MainViewModel.cs](#).

```
00495     {
00496         if (notificationMessage.Notification != "SettingsApplied")
00497         {
00498             return;
00499         }
00500
00501         SettingsModel = notificationMessage.Content;
00502         W65C51.Init(notificationMessage.Content.ComPortName);
00503         RaisePropertyChanged("SettingsModel");
00504         UpdateUi();
00505     }
```

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

Definition at line 389 of file [MainViewModel.cs](#).

```
00390     {
00391         if (notificationMessage.Notification != "StateLoaded")
00392         {
00393             return;
00394         }
00395
00396         Reset();
00397
00398         OutputLog = new
00399             MultiThreadedObservableCollection<OutputLog>(notificationMessage.Content.OutputLog);
00400         RaisePropertyChanged("OutputLog");
00401
00402         NumberOfCycles = notificationMessage.Content.NumberOfCycles;
00403
00404         W65C02 = notificationMessage.Content.W65C02;
```



```

00404         W65C22 = notificationMessage.Content.W65C22;
00405         MM65SIB = notificationMessage.Content.MM65SIB;
00406         W65C51 = notificationMessage.Content.W65C51;
00407         AT28C010 = notificationMessage.Content.AT28C010;
00408         AT28C64 = notificationMessage.Content.AT28C64;
00409         UpdateMemoryPage();
00410         UpdateUi();
00411
00412         IsRomLoaded = true;
00413         RaisePropertyChanged("IsRomLoaded");
00414     }

```

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

Definition at line 573 of file [MainViewModel.cs](#).

```

00574     {
00575         IsRunning = false;
00576
00577         if (_backgroundWorker.IsBusy)
00578             _backgroundWorker.CancelAsync();
00579
00580         StepProcessor();
00581         UpdateMemoryPage();
00582
00583         OutputLog.Insert(0, GetOutputLog());
00584         UpdateUi();
00585     }

```

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

Definition at line 595 of file [MainViewModel.cs](#).

```

00596     {
00597         W65C02.NextStep();
00598         NumberOfCycles = W65C02.GetCycleCount();
00599     }

```

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

Definition at line 507 of file [MainViewModel.cs](#).

```

00508     {
00509         MemoryPage.Clear();
00510         var offset = _memoryPageOffset * 256;
00511
00512         var multiplier = 0;
00513         for (ushort i = (ushort)offset; i < 256 * (_memoryPageOffset + 1); i++)
00514         {
00515             MemoryPage.Add(new MemoryRowModel
00516             {
00517                 Offset = ((16 * multiplier) + offset).ToString("X").PadLeft(4, '0'),
00518                 Location00 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00519                 Location01 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00520                 Location02 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00521                 Location03 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00522                 Location04 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00523                 Location05 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00524                 Location06 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00525                 Location07 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00526                 Location08 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00527                 Location09 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00528                 Location0A = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00529                 Location0B = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00530                 Location0C = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00531                 Location0D = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00532                 Location0E = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00533                 Location0F = MemoryMap.ReadWithoutCycle(i).ToString("X").PadLeft(2, '0'),
00534             });
00535             multiplier++;
00536         }
00537     }
00538 }

```

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

Definition at line 587 of file [MainViewModel.cs](#).

```
00588     {
00589         RaisePropertyChanged("W65C02");
00590         RaisePropertyChanged("NumberOfCycles");
00591         RaisePropertyChanged("CurrentDisassembly");
00592         RaisePropertyChanged("MemoryPage");
00593     }
```

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 RelayCommand that opens the About window.

Definition at line 193 of file [MainViewModel.cs](#).

```
00193 { get; set; }
```

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

The RelayCommand that adds a new breakpoint

Definition at line 188 of file [MainViewModel.cs](#).

```
00188 { 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.MainViewModel.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 Program 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](#).

```
00074 { 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](#).

```
00123     {
00124         get { return _memoryPageOffset.ToString("X"); }
00125         set
00126         {
00127             if (string.IsNullOrEmpty(value))
00128                 return;
00129             try
00130             {
00131                 _memoryPageOffset = Convert.ToInt32(value, 16);
00132             }
00133             catch { }
00134         }
00135     }
```

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](#).

```
00054 { 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](#).

```
00117 { get; private set; }
```

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

The output log

Definition at line 79 of file [MainViewModel.cs](#).

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

6.17.5.18 RemoveBreakPointCommand RelayCommand Emulator.ViewModel.MainViewModel.RemoveBreakPointCommand [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 program 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.UpdateMemoryMapCommand [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; } }
```

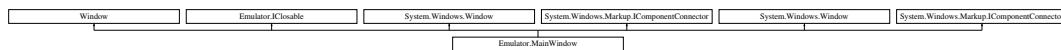
The documentation for this class was generated from the following file:

- [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](#).

```
00019     {
00020         InitializeComponent();
00021         Messenger.Default.Register<NotificationMessage>(this, NotificationMessageReceived);
00022         Messenger.Default.Register<NotificationMessage<StateFileModel>>(this,
NotificationMessageReceived);
00023         Messenger.Default.Register<NotificationMessage<SettingsModel>>(this,
NotificationMessageReceived);
00024     }
```

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](#).

```
00042     {
00043         Messenger.Default.Send(new NotificationMessage("CloseFile"));
00044     }
```

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](#).

```
00373                                                                 {
00374         switch (connectionId)
00375         {
00376             case 1:
00377                 this.EmulatorWindow = ((Emulator.MainWindow)(target));
00378                 return;
00379             case 2:
00380
00381 #line 72 "..\..\..\MainWindow.xaml"
00382                 ((System.Windows.Controls.MenuItem)(target)).Click += new
System.Windows.RoutedEventHandler(this.LoadFile);
00383
00384 #line default
00385 #line hidden
00386                 return;
00387             case 3:
00388
00389 #line 73 "..\..\..\MainWindow.xaml"
00390                 ((System.Windows.Controls.MenuItem)(target)).Click += new
System.Windows.RoutedEventHandler(this.SaveFile);
00391
00392 #line default
00393 #line hidden
00394                 return;
00395             case 4:
00396
00397 #line 74 "..\..\..\MainWindow.xaml"
00398                 ((System.Windows.Controls.MenuItem)(target)).Click += new
System.Windows.RoutedEventHandler(this.CloseFile);
```

```
00399
00400 #line default
00401 #line hidden
00402         return;
00403         case 5:
00404
00405 #line 76 "..\..\..\MainWindow.xaml"
00406         ((System.Windows.Controls.MenuItem)(target)).Click += new
System.Windows.RoutedEventHandler(this.ToClose);
00407
00408 #line default
00409 #line hidden
00410         return;
00411         case 6:
00412             this.OutputLog = ((System.Windows.Controls.DataGrid)(target));
00413             return;
00414         case 7:
00415             this.Run = ((System.Windows.Controls.Button)(target));
00416             return;
00417         case 8:
00418             this.Step = ((System.Windows.Controls.Button)(target));
00419             return;
00420         case 9:
00421             this.Reset = ((System.Windows.Controls.Button)(target));
00422             return;
00423         case 10:
00424             this.RomFileNameText = ((System.Windows.Controls.TextBlock)(target));
00425             return;
00426         case 11:
00427             this.ComPortNameText = ((System.Windows.Controls.TextBlock)(target));
00428             return;
00429         case 12:
00430             this.Breakpoints = ((System.Windows.Controls.DataGrid)(target));
00431             return;
00432         case 13:
00433             this.YRegister = ((System.Windows.Controls.TextBox)(target));
00434             return;
00435         case 14:
00436             this.XRegister = ((System.Windows.Controls.TextBox)(target));
00437             return;
00438         case 15:
00439             this.Accumulator = ((System.Windows.Controls.TextBox)(target));
00440             return;
00441         case 16:
00442             this.StackPointer = ((System.Windows.Controls.TextBox)(target));
00443             return;
00444         case 17:
00445             this.ProgramCounter = ((System.Windows.Controls.TextBox)(target));
00446             return;
00447         case 18:
00448             this.Dissambly = ((System.Windows.Controls.TextBox)(target));
00449             return;
00450         case 19:
00451             this.CycleCount = ((System.Windows.Controls.TextBox)(target));
00452             return;
00453         case 20:
00454             this.XRegisterText = ((System.Windows.Controls.TextBlock)(target));
00455             return;
00456         case 21:
00457             this.YRegisterText = ((System.Windows.Controls.TextBlock)(target));
00458             return;
00459         case 22:
00460             this.StackPointerRegisterText = ((System.Windows.Controls.TextBlock)(target));
00461             return;
00462         case 23:
00463             this.AText = ((System.Windows.Controls.TextBlock)(target));
00464             return;
00465         case 24:
00466             this.CurrentInstructionText = ((System.Windows.Controls.TextBlock)(target));
00467             return;
00468         case 25:
00469             this.ProgramCounterText = ((System.Windows.Controls.TextBlock)(target));
00470             return;
00471         case 26:
00472             this.CycleCountText = ((System.Windows.Controls.TextBlock)(target));
00473             return;
00474         case 27:
00475             this.CarryFlag = ((System.Windows.Controls.CheckBox)(target));
00476             return;
00477         case 28:
00478             this.CarryFlagText = ((System.Windows.Controls.TextBlock)(target));
00479             return;
00480         case 29:
00481             this.ZeroFlag = ((System.Windows.Controls.CheckBox)(target));
00482             return;
00483         case 30:
00484             this.ZeroFlagText = ((System.Windows.Controls.TextBlock)(target));
```

```

00485         return;
00486     case 31:
00487         this.InterruptFlag = ((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 }

```

6.18.3.3 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](#).

```

00373                                     {
00374         switch (connectionId)
00375         {
00376         case 1:
00377             this.EmulatorWindow = ((Emulator.MainWindow) (target));
00378             return;
00379         case 2:
00380             #line 72 "..\..\..\MainWindow.xaml"
00381             ((System.Windows.Controls.MenuItem) (target)).Click += new
00382                 System.Windows.RoutedEventHandler(this.LoadFile);
00383         #line default
00384         #line hidden
00385             return;
00386         case 3:
00387             #line 73 "..\..\..\MainWindow.xaml"
00388             ((System.Windows.Controls.MenuItem) (target)).Click += new
00389                 System.Windows.RoutedEventHandler(this.SaveFile);
00390         #line default
00391         #line hidden
00392             return;
00393         case 4:
00394             #line 74 "..\..\..\MainWindow.xaml"
00395             ((System.Windows.Controls.MenuItem) (target)).Click += new
00396                 System.Windows.RoutedEventHandler(this.CloseFile);
00397         #line default
00398         #line hidden
00399             return;
00400         }
00401     }

```

```
00402         return;
00403         case 5:
00404
00405 #line 76 "..\..\..\MainWindow.xaml"
00406         ((System.Windows.Controls.MenuItem)(target)).Click += new
System.Windows.RoutedEventHandler(this.ToClose);
00407
00408 #line default
00409 #line hidden
00410         return;
00411         case 6:
00412             this.OutputLog = ((System.Windows.Controls.DataGrid)(target));
00413             return;
00414         case 7:
00415             this.Run = ((System.Windows.Controls.Button)(target));
00416             return;
00417         case 8:
00418             this.Step = ((System.Windows.Controls.Button)(target));
00419             return;
00420         case 9:
00421             this.Reset = ((System.Windows.Controls.Button)(target));
00422             return;
00423         case 10:
00424             this.RomFileNameText = ((System.Windows.Controls.TextBlock)(target));
00425             return;
00426         case 11:
00427             this.ComPortNameText = ((System.Windows.Controls.TextBlock)(target));
00428             return;
00429         case 12:
00430             this.Breakpoints = ((System.Windows.Controls.DataGrid)(target));
00431             return;
00432         case 13:
00433             this.YRegister = ((System.Windows.Controls.TextBox)(target));
00434             return;
00435         case 14:
00436             this.XRegister = ((System.Windows.Controls.TextBox)(target));
00437             return;
00438         case 15:
00439             this.Accumulator = ((System.Windows.Controls.TextBox)(target));
00440             return;
00441         case 16:
00442             this.StackPointer = ((System.Windows.Controls.TextBox)(target));
00443             return;
00444         case 17:
00445             this.ProgramCounter = ((System.Windows.Controls.TextBox)(target));
00446             return;
00447         case 18:
00448             this.Dissambly = ((System.Windows.Controls.TextBox)(target));
00449             return;
00450         case 19:
00451             this.CycleCount = ((System.Windows.Controls.TextBox)(target));
00452             return;
00453         case 20:
00454             this.XRegisterText = ((System.Windows.Controls.TextBlock)(target));
00455             return;
00456         case 21:
00457             this.YRegisterText = ((System.Windows.Controls.TextBlock)(target));
00458             return;
00459         case 22:
00460             this.StackPointerRegisterText = ((System.Windows.Controls.TextBlock)(target));
00461             return;
00462         case 23:
00463             this.AText = ((System.Windows.Controls.TextBlock)(target));
00464             return;
00465         case 24:
00466             this.CurrentInstructionText = ((System.Windows.Controls.TextBlock)(target));
00467             return;
00468         case 25:
00469             this.ProgramCounterText = ((System.Windows.Controls.TextBlock)(target));
00470             return;
00471         case 26:
00472             this.CycleCountText = ((System.Windows.Controls.TextBlock)(target));
00473             return;
00474         case 27:
00475             this.CarryFlag = ((System.Windows.Controls.CheckBox)(target));
00476             return;
00477         case 28:
00478             this.CarryFlagText = ((System.Windows.Controls.TextBlock)(target));
00479             return;
00480         case 29:
00481             this.ZeroFlag = ((System.Windows.Controls.CheckBox)(target));
00482             return;
00483         case 30:
00484             this.ZeroFlagText = ((System.Windows.Controls.TextBlock)(target));
00485             return;
00486         case 31:
00487             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",
00359             System.UriKind.Relative);
00360 #line 1 "..\..\..\MainWindow.xaml"
00361         System.Windows.Application.LoadComponent(this, resourceLocater);
00362 #line default
00363 #line hidden
00364         }
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",
00359             System.UriKind.Relative);
00360 #line 1 "..\..\..\MainWindow.xaml"
00361         System.Windows.Application.LoadComponent(this, resourceLocater);
00362 #line default
00363 #line hidden
00364         }
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](#).

```
00032     {
00033         Messenger.Default.Send(new NotificationMessage("LoadFile"));
00034     }
```

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](#).

```
00047     {
00048         if (notificationMessage.Notification == "CloseWindow")
00049         {
00050             Close();
00051         }
00052     }
```

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](#).

```
00064     {
00065         if (notificationMessage.Notification == "SettingsWindow")
00066         {
00067             var settingsFile = new Settings { DataContext = new
SettingsViewModel(notificationMessage.Content) };
00068             settingsFile.ShowDialog();
00069         }
00070     }
```

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](#).

```
00055     {
00056         if (notificationMessage.Notification == "SaveFileWindow")
00057         {
00058             var saveFile = new SaveFile { DataContext = new
SaveFileViewModel(notificationMessage.Content) };
00059             saveFile.ShowDialog();
00060         }
00061     }
```

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

Definition at line 36 of file [MainWindow.xaml.cs](#).

```
00037     {
00038         Messenger.Default.Send(new NotificationMessage("SaveState"));
00039     }
```

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

Definition at line 26 of file [MainWindow.xaml.cs](#).

```
00027     {  
00028         Close();  
00029     }
```

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](#).

```
00088     {
00089         Processor = processor;
00090         GPIO = gpio;
00091         MM65SIB = mm65sib;
00092         ACIA = acia;
00093         SharedROM = sharedRom;
00094         BankedROM = bankedRom;
00095         BankedRAM = bankedRam;
00096     }
```

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

Returns the byte at the given address.

Parameters

<i>address</i>	The address to return
----------------	-----------------------

Returns

the byte being returned

Definition at line 103 of file [MemoryMap.cs](#).

```
00104     {
00105         var value = ReadWithoutCycle(address);
00106         Processor.IncrementCycleCount();
00107         return value;
00108     }
```

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

<i>address</i>	The address to return
----------------	-----------------------

Returns

the byte being returned

Definition at line 115 of file [MemoryMap.cs](#).

```
00116     {
00117         int _address = address;
00118         if ((ACIA.Offset <= _address) && (_address <= (ACIA.Offset + ACIA.Length)))
00119         {
00120             return ACIA.Read(address);
00121         }
00122         else if ((GPIO.Offset <= _address) && (_address <= (GPIO.Offset + GPIO.Length)))
00123         {
00124             return GPIO.Read(_address);
00125         }
00126         else if ((DeviceArea.Offset <= _address) && (_address <= DeviceArea.End))
00127         {
00128             throw new ArgumentOutOfRangeException("Device area accessed where there is no
device!");
00129         }
00130         else if ((SharedROM.Offset <= _address) && (_address <= SharedROM.End))
00131         {
00132             return SharedROM.Read(_address);
00133         }
00134         else if ((BankedROM.Offset <= _address) && (_address <= BankedROM.End))
00135         {
00136             return BankedROM.Read(_address);
00137         }
00138         else if ((BankedRAM.Offset <= _address) && (_address <= BankedRAM.End))
00139         {
00140             return BankedRAM.Read(_address);
00141         }
00142         else
00143         {
00144             return 0x00;
00145         }
00146     }
```

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

Writes data to the given address.

Parameters

<i>address</i>	The address to write data to.
<i>data</i>	The data to write.

Definition at line 153 of file [MemoryMap.cs](#).

```
00154     {
00155         Processor.IncrementCycleCount();
00156         WriteWithoutCycle(address, data);
00157     }
```

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

<i>address</i>	The address to write data to.
<i>data</i>	The data to write.

Definition at line 164 of file [MemoryMap.cs](#).

```
00165     {
00166         if ((ACIA.Offset <= address) && (address <= (ACIA.Offset + ACIA.Length)))
00167         {
00168             ACIA.Write(address, data);
00169         }
00170         else if ((GPIO.Offset <= address) && (address <= (GPIO.Offset + GPIO.Length)))
00171         {
00172             GPIO.Write(address, data);
00173         }
00174         else if ((SharedROM.Offset <= address) && (address <= (SharedROM.Offset +
SharedROM.Length)))
00175         {
00176             SharedROM.Write(address, data);
00177         }
00178         else if ((BankedROM.Offset <= address) && (address <= (BankedROM.Offset +
BankedROM.Length)))
00179         {
00180             BankedROM.Write(address, data);
00181         }
00182         else if ((BankedRAM.Offset <= address) && (address <= (BankedRAM.Offset +
BankedRAM.Length)))
00183         {
00184             BankedRAM.Write(address, data);
00185         }
00186         else
00187         {
00188             throw new ApplicationException(String.Format("Cannot write to address: {0}",
address));
00189         }
00190     }
```

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](#).

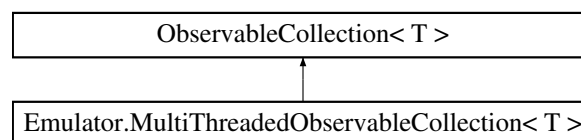
The documentation for this class was generated from the following file:

- 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 >:



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

Template Parameters

<i>T</i>	
----------	--

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](#).

```
00020     {
00021
00022     }
```

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

Instantiates a new instance of the [MultiThreadedObservableCollection](#)

Parameters

<i>collection</i>	The initial collection to be loaded
-------------------	-------------------------------------

Definition at line 28 of file [MultiThreadedCollection.cs](#).

```

00029         : base(collection)
00030     {
00031
00032     }

```

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

Instantiates a new instance of the [MultiThreadedObservableCollection](#)

Parameters

<i>list</i>	The initial list to be loaded
-------------	-------------------------------

Definition at line 38 of file [MultiThreadedCollection.cs](#).

```

00039         : base(list)
00040     {
00041
00042     }

```

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](#).

```

00054     {
00055         var collectionChanged = CollectionChanged;
00056         if (collectionChanged != null)
00057             foreach (NotifyCollectionChangedEventHandler nh in
collectionChanged.GetInvocationList())
00058             {
00059                 var dispObj = nh.Target as DispatcherObject;
00060                 if (dispObj != null)
00061                 {
00062                     var dispatcher = dispObj.Dispatcher;
00063                     if (dispatcher != null && !dispatcher.CheckAccess())
00064                     {
00065                         var nh1 = nh;
00066                         dispatcher.BeginInvoke(
00067                             (Action) (() => nh1.Invoke(this,
new
00068                                 NotifyCollectionChangedEventArgs(NotifyCollectionChangedAction.Reset))),
00069                             DispatcherPriority.DataBind);
00070                         continue;
00071                     }
00072                 }
00073                 nh.Invoke(this, e);
00074             }
00075     }

```

6.22.4 Event Documentation

6.22.4.1 CollectionChanged `override NotifyCollectionChangedEventHandler Emulator.MultiThreadedObservableCollection<T>.CollectionChanged`

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

Definition at line 47 of file [MultiThreadedCollection.cs](#).

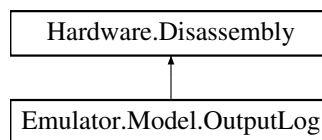
The documentation for this class was generated from the following file:

- [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 Accumulator
- 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](#).

```
00013     {  
00014         DisassemblyOutput = disassembly.DisassemblyOutput;  
00015         HighAddress = disassembly.HighAddress;  
00016         LowAddress = disassembly.LowAddress;  
00017         OpCodeString = disassembly.OpCodeString;  
00018     }
```

6.23.3 Property Documentation

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

The Accumulator

Definition at line 39 of file [OutputLog.cs](#).

```
00039 { 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](#).

```
00027 { 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](#).

```
00047 { 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](#).

```
00023 { 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 I 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 1' 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 I 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; }
```

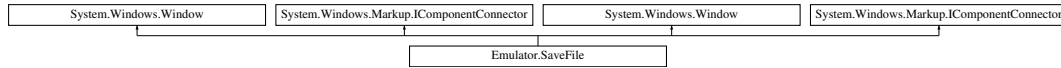
The documentation for this class was generated from the following file:

- [Emulator/Model/RomFileModel.cs](#)

6.27 Emulator.SaveFile Class Reference

SaveFile

Inheritance diagram for Emulator.SaveFile:



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](#).

```

00011     {
00012         InitializeComponent ();
00013         Messenger.Default.Register<NotificationMessage>(this, NotificationMessageReceived);
00014     }
  
```

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](#).

```

00089                                     {
00090         if (_contentLoaded) {
00091             return;
00092         }
00093         _contentLoaded = true;
00094         System.Uri resourceLocator = new System.Uri("/Emulator;component/savefile.xaml",
System.UriKind.Relative);
00095
00096 #line 1 "..\..\..\SaveFile.xaml"
00097         System.Windows.Application.LoadComponent(this, resourceLocator);
00098
00099 #line default
00100 #line hidden
00101     }

```

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

InitializeComponent

Definition at line 89 of file [SaveFile.g.i.cs](#).

```

00089                                     {
00090         if (_contentLoaded) {
00091             return;
00092         }
00093         _contentLoaded = true;
00094         System.Uri resourceLocator = new System.Uri("/Emulator;component/savefile.xaml",
System.UriKind.Relative);
00095
00096 #line 1 "..\..\..\SaveFile.xaml"
00097         System.Windows.Application.LoadComponent(this, resourceLocator);
00098
00099 #line default
00100 #line hidden
00101     }

```

6.27.3.5 NotificationMessageReceived() void Emulator.SaveFile.NotificationMessageReceived (NotificationMessage notificationMessage) [inline], [private]Definition at line 16 of file [SaveFile.xaml.cs](#).

```

00017     {
00018         if (notificationMessage.Notification == "CloseSaveFileWindow")
00019             Close();
00020     }

```

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](#).

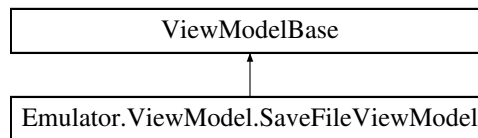
The documentation for this class was generated from the following files:

- [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](#).

```
00052     {
00053
00054     }
```

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

Instantiates a new instance of the [SaveFileViewModel](#)

Parameters

<i>stateFileModel</i>	The StateFileModel to be serialized to a file
-----------------------	---

Definition at line 60 of file [SaveFileViewModel.cs](#).

```
00061     {
00062         SaveFileCommand = new RelayCommand(Save);
00063         CloseCommand = new RelayCommand(Close);
00064         SelectFileCommand = new RelayCommand(Select);
00065         _stateFileModel = stateFileModel;
00066     }
```

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](#).

```
00081     {
00082         Messenger.Default.Send(new NotificationMessage("CloseSaveFileWindow"));
00083     }
```

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

Definition at line 70 of file [SaveFileViewModel.cs](#).

```
00071     {
00072         var formatter = new BinaryFormatter();
00073         Stream stream = new FileStream(Filename, FileMode.Create, FileAccess.Write,
        FileShare.None);
00074         formatter.Serialize(stream, _stateFileModel);
00075         stream.Close();
00076
00077         Close();
00078     }
```

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

Definition at line 85 of file [SaveFileViewModel.cs](#).

```
00086     {
00087         var dialog = new SaveFileDialog { DefaultExt = ".6502", Filter = "WolfNet W65C02 Emulator
        Save State (*.6502)|*.6502" };
00088
00089         var result = dialog.ShowDialog();
00090
00091         if (result != true)
00092             return;
00093
00094         Filename = dialog.FileName;
00095         RaisePropertyChanged("Filename");
00096         RaisePropertyChanged("SaveEnabled");
00097
00098     }
```

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; }
```

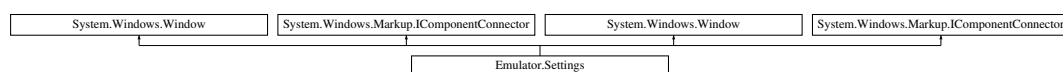
The documentation for this class was generated from the following file:

- [Emulator/ViewModel/SaveFileViewModel.cs](#)

6.29 Emulator.Settings Class Reference

Settings

Inheritance diagram for Emulator.Settings:



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](#).

```
00015     {  
00016         InitializeComponent ();  
00017         Messenger.Default.Register<NotificationMessage>(this, NotificationMessageReceived);  
00018         Messenger.Default.Register<NotificationMessage<SettingsModel>>(this,  
    NotificationMessageReceived);  
00019     }
```

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](#).

```
00101 {
00102     switch (connectionId)
00103     {
00104         case 1:
00105             this.ComPortCombo = ((System.Windows.Controls.ComboBox) (target));
00106
00107 #line 7 "..\..\..\Settings.xaml"
00108             this.ComPortCombo.DropDownClosed += new
System.EventHandler(this.PortSelectionDropDownClosed);
00109
00110 #line default
00111 #line hidden
00112             return;
00113         case 2:
00114             this.PortText = ((System.Windows.Controls.TextBlock) (target));
00115             return;
00116         case 3:
00117             this.ApplyButton = ((System.Windows.Controls.Button) (target));
00118             return;
00119         case 4:
00120             this.CloseButton = ((System.Windows.Controls.Button) (target));
00121             return;
00122     }
00123     this._contentLoaded = true;
00124 }
```

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](#).

```
00101 {
00102     switch (connectionId)
00103     {
00104         case 1:
00105             this.ComPortCombo = ((System.Windows.Controls.ComboBox) (target));
00106
00107 #line 7 "..\..\..\Settings.xaml"
00108             this.ComPortCombo.DropDownClosed += new
System.EventHandler(this.PortSelectionDropDownClosed);
00109
00110 #line default
00111 #line hidden
00112             return;
00113         case 2:
00114             this.PortText = ((System.Windows.Controls.TextBlock) (target));
00115             return;
00116         case 3:
00117             this.ApplyButton = ((System.Windows.Controls.Button) (target));
00118             return;
00119         case 4:
00120             this.CloseButton = ((System.Windows.Controls.Button) (target));
00121             return;
00122     }
00123     this._contentLoaded = true;
00124 }
```

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

InitializeComponent

Definition at line 81 of file [Settings.g.cs](#).

```

00081                                     {
00082         if (_contentLoaded) {
00083             return;
00084         }
00085         _contentLoaded = true;
00086         System.Uri resourceLocator = new System.Uri("/Emulator;component/settings.xaml",
System.UriKind.Relative);
00087
00088 #line 1 "..\..\..\Settings.xaml"
00089         System.Windows.Application.LoadComponent(this, resourceLocator);
00090
00091 #line default
00092 #line hidden
00093     }

```

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

InitializeComponent

Definition at line 81 of file [Settings.g.i.cs](#).

```

00081                                     {
00082         if (_contentLoaded) {
00083             return;
00084         }
00085         _contentLoaded = true;
00086         System.Uri resourceLocator = new System.Uri("/Emulator;component/settings.xaml",
System.UriKind.Relative);
00087
00088 #line 1 "..\..\..\Settings.xaml"
00089         System.Windows.Application.LoadComponent(this, resourceLocator);
00090
00091 #line default
00092 #line hidden
00093     }

```

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

NotificationMessage notificationMessage) [inline], [private]

Definition at line 21 of file [Settings.xaml.cs](#).

```

00022     {
00023         if (notificationMessage.Notification == "CloseSettingsWindow")
00024         {
00025             Close();
00026         }
00027     }

```

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](#).

```

00030     {
00031         if (notificationMessage.Notification == "SettingsWindow")
00032         {
00033             SettingsViewModel.SettingsModel = notificationMessage.Content;
00034             ComPortCombo.SelectedItem = notificationMessage.Content.ComPortName;
00035         }
00036     }

```

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

Definition at line 38 of file [Settings.xaml.cs](#).

```
00039     {
00040         if (!(ComPortCombo.SelectedValue == null))
00041         {
00042             string port = ComPortCombo.SelectedValue.ToString();
00043             SettingsViewModel.ComPortSelection = port;
00044         }
00045     }
```

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](#).

```
00009     {
00010         // Create new settings file.
00011         SettingsModel _settings = new SettingsModel
00012         {
00013             SettingsVersionMajor = Versioning.SettingsFile.Major,
00014             SettingsVersionMinor = Versioning.SettingsFile.Minor,
00015             SettingsVersionBuild = Versioning.SettingsFile.Build,
00016             SettingsVersionRevision = Versioning.SettingsFile.Revision,
00017 #if DEBUG
00018             ComPortName = "COM9",
00019 #else
00020             ComPortName = "COM1",
00021 #endif
00022         };
00023         return _settings;
00024     }
```

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; }
```

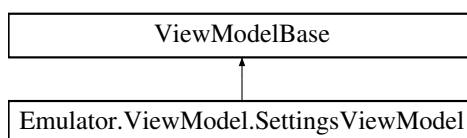
The documentation for this class was generated from the following file:

- [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](#).

```
00052     {
00053
00054     }
```

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

Instantiates a new instance of the [SettingsViewModel](#)

Parameters

<i>settingsModel</i>	The SettingsFileModel to be serialized to a file
----------------------	--

Definition at line 60 of file [SettingsViewModel.cs](#).

```
00061     {
00062         ApplyCommand = new RelayCommand(Apply);
00063         CloseCommand = new RelayCommand(Close);
00064         ComPortSelection = settingsModel.ComPortName;
00065
00066         UpdatePortList();
00067     }
```


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](#).

```
00085     {
00086         Messenger.Default.Send(new NotificationMessage<SettingsModel>(new SettingsModel
00087         {
00088             SettingsVersionMajor = Versioning.SettingsFile.Major,
00089             SettingsVersionMinor = Versioning.SettingsFile.Minor,
00090             SettingsVersionBuild = Versioning.SettingsFile.Build,
00091             SettingsVersionRevision = Versioning.SettingsFile.Revision,
00092             ComPortName = ComPortSelection,
00093         }, "SettingsApplied"));
00094         Messenger.Default.Send(new NotificationMessage("CloseSettingsWindow"));
00095     }
```

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

Definition at line 97 of file [SettingsViewModel.cs](#).

```
00098     {
00099         Messenger.Default.Send(new NotificationMessage("CloseSettingsWindow"));
00100     }
```

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](#).

```
00073     {
00074         PortList.Clear();
00075         foreach (string s in SerialPort.GetPortNames())
00076         {
00077             PortList.Add(s);
00078         }
00079         RaisePropertyChanged("PortList");
00080     }
```

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](#).

```

00008     {
00009         switch (i)
00010         {
00011             case 0x69: //ãADCãImmediate
00012             case 0x65: //ãADCãZeroãPage
00013             case 0x75: //ãADCãZeroãPageãX
00014             case 0x6D: //ãADCãAbsolute
00015             case 0x7D: //ãADCãAbsoluteãX
00016             case 0x79: //ãADCãAbsoluteãY
00017             case 0x61: //ãADCãIndirectãX
00018             case 0x71: //ãADCãIndirectãY
00019             {
00020                 return "ADC";
00021             }
00022             case 0x29: //ãANDãImmediate
00023             case 0x25: //ãANDãZeroãPage
00024             case 0x35: //ãANDãZeroãPageãX
00025             case 0x2D: //ãANDãAbsolute
00026             case 0x3D: //ãANDãAbsoluteãX
00027             case 0x39: //ãANDãAbsoluteãY
00028             case 0x21: //ãANDãIndirectãX
00029             case 0x31: //ãANDãIndirectãY
00030             {
00031                 return "AND";
00032             }
00033             case 0x0A: //ãASLãAccumulator
00034             case 0x06: //ãASLãZeroãPage
00035             case 0x16: //ãASLãZeroãPageãX
00036             case 0x0E: //ãASLãAbsolute
00037             case 0x1E: //ãASLãAbsoluteãX
00038             {
00039                 return "ASL";
00040             }
00041             case 0x90: //ãBCCãRelative
00042             {
00043                 return "BCC";
00044             }
00045             case 0xB0: //ãBCSãRelative
00046             {
00047                 return "BCS";
00048             }
00049             case 0xF0: //ãBEQãRelative
00050             {
00051                 return "BEQ";
00052             }
00053             case 0x24: //ãBITãZeroãPage
00054             case 0x2C: //ãBITãAbsolute
00055             {
00056                 return "BIT";
00057             }
00058             case 0x30: //ãBMIãRelative
00059             {
00060                 return "BMI";
00061             }
00062             case 0xD0: //ãBNEãRelative
00063             {
00064                 return "BNE";
00065             }
00066             case 0x10: //ãBPLãRelative
00067             {
00068                 return "BPL";
00069             }
00070             case 0x00: //ãBRKãImplied
00071             {
00072                 return "BRK";
00073             }
00074             case 0x50: // BVC Relative
00075             {
00076                 return "BVC";
00077             }
00078             case 0x70: //BVS Relative
00079             {
00080                 return "BVS";
00081             }
00082             case 0x18: //ãCLCãImplied
00083             {
00084                 return "CLC";
00085             }
00086             case 0xD8: //ãCLDãImplied
00087             {
00088                 return "CLD";
00089             }

```

```

00090         case 0x58:    //äCLIäImplied
00091         {
00092             return "CLI";
00093         }
00094         case 0xB8:    //äCLVäImplied
00095         {
00096             return "CLV";
00097         }
00098         case 0xC9:    //äCMPäImmediate
00099         case 0xC5:    //äCMPäZeroPage
00100         case 0xD5:    //äCMPäZeroäPageäX
00101         case 0xCD:    //äCMPäAbsolute
00102         case 0xDD:    //äCMPäAbsoluteäX
00103         case 0xD9:    //äCMPäAbsoluteäY
00104         case 0xC1:    //äCMPäIndirectäX
00105         case 0xD1:    //äCMPäIndirectäY
00106         {
00107             return "CMP";
00108         }
00109         case 0xE0:    //äCPXäImmediate
00110         case 0xE4:    //äCPXäZeroPage
00111         case 0xEC:    //äCPXäAbsolute
00112         {
00113             return "CPX";
00114         }
00115         case 0xC0:    //äCPYäImmediate
00116         case 0xC4:    //äCPYäZeroPage
00117         case 0xCC:    //äCPYäAbsolute
00118         {
00119             return "CPY";
00120         }
00121         case 0xC6:    //äDECäZeroäPage
00122         case 0xD6:    //äDECäZeroäPageäX
00123         case 0xCE:    //äDECäAbsolute
00124         case 0xDE:    //äDECäAbsoluteäX
00125         {
00126             return "DEC";
00127         }
00128         case 0xCA:    //äDEXäImplied
00129         {
00130             return "DEX";
00131         }
00132         case 0x88:    //äDEYäImplied
00133         {
00134             return "DEY";
00135         }
00136         case 0x49:    //äEORäImmediate
00137         case 0x45:    //äEORäZeroäPage
00138         case 0x55:    //äEORäZeroäPageäX
00139         case 0x4D:    //äEORäAbsolute
00140         case 0x5D:    //äEORäAbsoluteäX
00141         case 0x59:    //äEORäAbsoluteäY
00142         case 0x41:    //äEORäIndirectäX
00143         case 0x51:    //äEORäIndirectäY
00144         {
00145             return "EOR";
00146         }
00147         case 0xE6:    //äINCäZeroäPage
00148         case 0xF6:    //äINCäZeroäPageäX
00149         {
00150             return "INC";
00151         }
00152         case 0xE8:    //äINXäImplied
00153         {
00154             return "INX";
00155         }
00156         case 0xC8:    //äINYäImplied
00157         {
00158             return "INY";
00159         }
00160         case 0xEE:    //äINCäAbsolute
00161         case 0xFE:    //äINCäAbsoluteäX
00162         {
00163             return "INC";
00164         }
00165         case 0x4C:    //äJMPäAbsolute
00166         case 0x6C:    //äJMPäIndirect
00167         {
00168             return "JMP";
00169         }
00170         case 0x20:    //äJSRäAbsolute
00171         {
00172             return "JSR";
00173         }
00174         case 0xA9:    //äLDAäImmediate
00175         case 0xA5:    //äLDAäZeroäPage
00176         case 0xB5:    //äLDAäZeroäPageäX

```



```

00177         case 0xAD: //äLDAäAbsolute
00178         case 0xBD: //äLDAäAbsoluteäX
00179         case 0xB9: //äLDAäAbsoluteäY
00180         case 0xA1: //äLDAäIndirectäX
00181         case 0xB1: //äLDAäIndirectäY
00182         {
00183             return "LDA";
00184         }
00185         case 0xA2: //äLDXäImmediate
00186         case 0xA6: //äLDXäZeroäPage
00187         case 0xB6: //äLDXäZeroäPageäY
00188         case 0xAE: //äLDXäAbsolute
00189         case 0xBE: //äLDXäAbsoluteäY
00190         {
00191             return "LDX";
00192         }
00193         case 0xA0: //äLDYäImmediate
00194         case 0xA4: //äLDYäZeroäPage
00195         case 0xB4: //äLDYäZeroäPageäY
00196         case 0xAC: //äLDYäAbsolute
00197         case 0xBC: //äLDYäAbsoluteäY
00198         {
00199             return "LDY";
00200         }
00201         case 0x4A: //äLSRäAccumulator
00202         case 0x46: //äLSRäZeroäPage
00203         case 0x56: //äLSRäZeroäPageäX
00204         case 0x4E: //äLSRäAbsolute
00205         case 0x5E: //äLSRäAbsoluteäX
00206         {
00207             return "LSR";
00208         }
00209         case 0xEA: //äNOPäImplied
00210         {
00211             return "NOP";
00212         }
00213         case 0x09: //äORAäImmediate
00214         case 0x05: //äORAäZeroäPage
00215         case 0x15: //äORAäZeroäPageäX
00216         case 0x0D: //äORAäAbsolute
00217         case 0x1D: //äORAäAbsoluteäX
00218         case 0x19: //äORAäAbsoluteäY
00219         case 0x01: //äORAäIndirectäX
00220         case 0x11: //äORAäIndirectäY
00221         {
00222             return "ORA";
00223         }
00224         case 0x48: //äPHAäImplied
00225         {
00226             return "PHA";
00227         }
00228         case 0x08: //äPHPäImplied
00229         {
00230             return "PHP";
00231         }
00232         case 0x68: //äPLAäImplied
00233         {
00234             return "PLA";
00235         }
00236         case 0x28: //äPLPäImplied
00237         {
00238             return "PLP";
00239         }
00240         case 0x2A: //äROLäAccumulator
00241         case 0x26: //äROLäZeroäPage
00242         case 0x36: //äROLäZeroäPageäX
00243         case 0x2E: //äROLäAbsolute
00244         case 0x3E: //äROLäAbsoluteäX
00245         {
00246             return "ROL";
00247         }
00248         case 0x6A: //äRORäAccumulator
00249         case 0x66: //äRORäZeroäPage
00250         case 0x76: //äRORäZeroäPageäX
00251         case 0x6E: //äRORäAbsolute
00252         case 0x7E: //äRORäAbsoluteäX
00253         {
00254             return "ROR";
00255         }
00256         case 0x40: //äRTIäImplied
00257         {
00258             return "RTI";
00259         }
00260         case 0x60: //äRTSäImplied
00261         {
00262             return "RTS";
00263         }

```

```

00264         case 0xE9: //äSBCäImmediate
00265         case 0xE5: //äSBCäZeroäPage
00266         case 0xF5: //äSBCäZeroäPageäX
00267         case 0xED: //äSBCäAbsolute
00268         case 0xFD: //äSBCäAbsoluteäX
00269         case 0xF9: //äSBCäAbsoluteäY
00270         case 0xE1: //äSBCäIndirectäX
00271         case 0xF1: //äSBCäIndirectäY
00272         {
00273             return "SBC";
00274         }
00275         case 0x38: //äSECäImplied
00276         {
00277             return "SEC";
00278         }
00279         case 0xF8: //äSEDäImplied
00280         {
00281             return "SED";
00282         }
00283         case 0x78: //äSEIäImplied
00284         {
00285             return "SEI";
00286         }
00287         case 0x85: //äSTAäZeroäPage
00288         case 0x95: //äSTAäZeroäPageäX
00289         case 0x8D: //äSTAäAbsolute
00290         case 0x9D: //äSTAäAbsoluteäX
00291         case 0x99: //äSTAäAbsoluteäY
00292         case 0x81: //äSTAäIndirectäX
00293         case 0x91: //äSTAäIndirectäY
00294         {
00295             return "STA";
00296         }
00297         case 0x86: //äSTXäZeroäPage
00298         case 0x96: //äSTXäZeroäPageäY
00299         case 0x8E: //äSTXäAbsolute
00300         {
00301             return "STX";
00302         }
00303         case 0x84: //äSTYäZeroäPage
00304         case 0x94: //äSTYäZeroäPageäX
00305         case 0x8C: //äSTYäAbsolute
00306         {
00307             return "STY";
00308         }
00309         case 0xAA: //äTAXäImplied
00310         {
00311             return "TAX";
00312         }
00313         case 0xA8: //äTAYäImplied
00314         {
00315             return "TAY";
00316         }
00317         case 0xBA: //äTSXäImplied
00318         {
00319             return "TSX";
00320         }
00321         case 0x8A: //äTXAäImplied
00322         {
00323             return "TXA";
00324         }
00325         case 0x9A: //äTXSäImplied
00326         {
00327             return "TXS";
00328         }
00329         case 0x98: //äTYAäImplied
00330         {
00331             return "TYA";
00332         }
00333         default:
00334             throw new InvalidEnumArgumentException(string.Format("A Valid Conversion does not
exist for OpCode {0}", i.ToString("X")));
00335     }
00336 }
00337

```

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](#).

```
00030     {
00031         ServiceLocator.SetLocatorProvider(() => SimpleIoc.Default);
00032
00033         SimpleIoc.Default.Register<MainViewModel>();
00034         SimpleIoc.Default.Register<SaveFileViewModel>();
00035         SimpleIoc.Default.Register<SettingsViewModel>();
00036     }
```

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](#).

```
00066     {
00067     /// <todo>
00068     /// Clear the ViewModels
00069     /// </todo>
00070 }
```

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](#).

```
00042     {
00043         get { return ServiceLocator.Current.GetInstance<MainViewModel>(); }
00044     }
```

6.38.4.2 SaveFile [SaveFileViewModel](#) Emulator.ViewModel.ViewModelLocator.SaveFile [get]

The [SaveFileViewModel](#) Instance

Definition at line 49 of file [ViewModelLocator.cs](#).

```
00050     {  
00051         get { return ServiceLocator.Current.GetInstance<SaveFileViewModel>(); }  
00052     }
```

6.38.4.3 Settings [SettingsViewModel](#) Emulator.ViewModel.ViewModelLocator.Settings [get]

The [SaveFileViewModel](#) Instance

Definition at line 57 of file [ViewModelLocator.cs](#).

```
00058     {  
00059         get { return ServiceLocator.Current.GetInstance<SettingsViewModel>(); }  
00060     }
```

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 instead 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 occurring 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 operation. 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 occurs*
- void [ProcessIRQ](#) ()
- This is ran anytime an IRQ occurs*

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](#).

```
00144     {
00145         StackPointer = 0x100;
00146         CycleCountIncrementedAction = () => { };
00147     }
```

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

<i>addressingMode</i>	The addressing mode used to perform this operation.
-----------------------	---

Definition at line 1888 of file [W65C02.cs](#).

```

01889     {
01890         //Accumulator, Carry = Accumulator + ValueInMemoryLocation + Carry
01891         var memoryValue = MemoryMap.Read(GetAddressByAddressingMode(addressingMode));
01892         var newValue = memoryValue + Accumulator + (CarryFlag ? 1 : 0);
01893
01894
01895         OverflowFlag = (((Accumulator ^ newValue) & 0x80) != 0) && (((Accumulator ^ memoryValue) &
0x80) == 0);
01896
01897         if (DecimalFlag)
01898         {
01899             newValue = int.Parse(memoryValue.ToString("x")) + int.Parse(Accumulator.ToString("x"))
+ (CarryFlag ? 1 : 0);
01900
01901             if (newValue > 99)
01902             {
01903                 CarryFlag = true;
01904                 newValue -= 100;
01905             }
01906             else
01907             {
01908                 CarryFlag = false;
01909             }
01910
01911             newValue = (int)Convert.ToInt64(string.Concat("0x", newValue), 16);
01912         }
01913         else
01914         {
01915             if (newValue > 255)
01916             {
01917                 CarryFlag = true;
01918                 newValue -= 256;
01919             }
01920             else
01921             {
01922                 CarryFlag = false;
01923             }
01924         }
01925
01926         SetZeroFlag(newValue);
01927         SetNegativeFlag(newValue);
01928
01929         Accumulator = newValue;
01930     }

```

6.39.3.2 AndOperation() void Hardware.W65C02.AndOperation (
AddressingMode *addressingMode*) [inline], [private]

The AND - Compare Memory with Accumulator operation

Parameters

<i>addressingMode</i>	The addressing mode being used
-----------------------	--------------------------------

Definition at line 1936 of file [W65C02.cs](#).

```

01937     {
01938         Accumulator = MemoryMap.Read(GetAddressByAddressingMode(addressingMode)) & Accumulator;
01939
01940         SetZeroFlag(Accumulator);
01941         SetNegativeFlag(Accumulator);
01942     }

```

6.39.3.3 AslOperation() void Hardware.W65C02.AslOperation (
 AddressingMode addressingMode) [inline]

The ASL - Shift Left One Bit (Memory or Accumulator)

Parameters

<i>addressingMode</i>	The addressing Mode being used
-----------------------	--------------------------------

Definition at line 1948 of file W65C02.cs.

```

01949     {
01950         int value;
01951         var memoryAddress = 0;
01952         if (addressingMode == AddressingMode.Accumulator)
01953         {
01954             MemoryMap.Read(ProgramCounter + 1);
01955             value = Accumulator;
01956         }
01957         else
01958         {
01959             memoryAddress = GetAddressByAddressingMode(addressingMode);
01960             value = MemoryMap.Read(memoryAddress);
01961         }
01962
01963         //Dummy Write
01964         if (addressingMode != AddressingMode.Accumulator)
01965         {
01966             MemoryMap.Write(memoryAddress, (byte)value);
01967         }
01968
01969         //If the 7th bit is set, then we have a carry
01970         CarryFlag = ((value & 0x80) != 0);
01971
01972         //The And here ensures that if the value is greater than 255 it wraps properly.
01973         value = (value << 1) & 0xFE;
01974
01975         SetNegativeFlag(value);
01976         SetZeroFlag(value);
01977
01978
01979         if (addressingMode == AddressingMode.Accumulator)
01980             Accumulator = value;
01981         else
01982         {
01983             MemoryMap.Write(memoryAddress, (byte)value);
01984         }
01985     }

```

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

<i>addressingMode</i>	
-----------------------	--

Definition at line 2008 of file W65C02.cs.

```

02009     {
02010
02011         var memoryValue = MemoryMap.Read(GetAddressByAddressingMode(addressingMode));
02012         var valueToCompare = memoryValue & Accumulator;
02013
02014         OverflowFlag = (memoryValue & 0x40) != 0;
02015
02016         SetNegativeFlag(memoryValue);
02017         SetZeroFlag(valueToCompare);
02018     }

```

6.39.3.5 BranchOperation() void Hardware.W65C02.BranchOperation (
 bool *performBranch*) [inline], [private]

Performs the different branch operations.

Parameters

<i>performBranch</i>	Is a branch required
----------------------	----------------------

Definition at line 1991 of file [W65C02.cs](#).

```

01992     {
01993         var value = MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.Relative));
01994
01995         if (!performBranch)
01996         {
01997             ProgramCounter++;
01998             return;
01999         }
02000
02001         MoveProgramCounterByRelativeValue(value);
02002     }

```

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](#).

```

02355     {
02356         MemoryMap.Read(++ProgramCounter);
02357
02358         //Put the high value on the stack
02359         //When we RTI the address will be incremented by one, and the address after a break will
not be used.
02360         PokeStack((byte)((ProgramCounter) >> 8) & 0xFF);
02361         StackPointer--;
02362         IncrementCycleCount();
02363
02364         //Put the low value on the stack
02365         PokeStack((byte)((ProgramCounter) & 0xFF));
02366         StackPointer--;
02367         IncrementCycleCount();
02368
02369         //We only set the Break Flag is a Break Occurs
02370         if (isBrk)
02371             PokeStack((byte)(ConvertFlagsToByte(true) | 0x10));
02372         else
02373             PokeStack(ConvertFlagsToByte(false));
02374
02375         StackPointer--;
02376         IncrementCycleCount();
02377
02378         DisableInterruptFlag = true;
02379
02380         ProgramCounter = (MemoryMap.Read(vector + 1) << 8) | MemoryMap.Read(vector);
02381
02382         _previousInterrupt = false;
02383     }

```

6.39.3.7 ChangeMemoryByOne() void Hardware.W65C02.ChangeMemoryByOne (
 AddressingMode *addressingMode*,
 bool *decrement*) [inline], [private]

Changes a value in memory by 1

Parameters

<i>addressingMode</i>	The addressing mode to use
<i>decrement</i>	If the operation is decrementing or incrementing the vaulue by 1

Definition at line 2044 of file [W65C02.cs](#).

```

02045     {
02046         var memoryLocation = GetAddressByAddressingMode(addressingMode);
02047         var memory = MemoryMap.Read(memoryLocation);
02048
02049         MemoryMap.Write(memoryLocation, memory);
02050
02051         if (decrement)
02052             memory -= 1;
02053         else
02054             memory += 1;
02055
02056         SetZeroFlag(memory);
02057         SetNegativeFlag(memory);
02058
02059         MemoryMap.Write(memoryLocation, memory);
02060     }
02061 
```

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

<i>useXRegister</i>	If the operation is using the X or Y register
<i>decrement</i>	If the operation is decrementing or incrementing the vaulue by 1

Definition at line 2068 of file [W65C02.cs](#).

```

02069     {
02070         var value = useXRegister ? XRegister : YRegister;
02071
02072         if (decrement)
02073             value -= 1;
02074         else
02075             value += 1;
02076
02077         if (value < 0x00)
02078             value += 0x100;
02079         else if (value > 0xFF)
02080             value -= 0x100;
02081
02082         SetZeroFlag(value);
02083         SetNegativeFlag(value);
02084         IncrementCycleCount();
02085
02086         if (useXRegister)
02087             XRegister = value;
02088         else
02089             YRegister = value;
02090     }

```

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

<i>addressingMode</i>	The addressing mode to use
<i>comparisonValue</i>	The value to compare against memory

Definition at line 2025 of file [W65C02.cs](#).

```

02026     {
02027         var memoryValue = MemoryMap.Read(GetAddressByAddressingMode(addressingMode));
02028         var comparedValue = comparisonValue - memoryValue;
02029
02030         if (comparedValue < 0)
02031             comparedValue += 0x10000;
02032
02033         SetZeroFlag(comparedValue);
02034
02035         CarryFlag = memoryValue <= comparisonValue;
02036         SetNegativeFlag(comparedValue);
02037     }

```

6.39.3.10 ConvertFlagsToByte() byte Hardware.W65C02.ConvertFlagsToByte (bool *setBreak*) [inline], [private]

Converts the Flags into its byte representation.

Parameters

<i>setBreak</i>	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](#).

```

01523     {
01524         return (byte)((CarryFlag ? 0x01 : 0) + (ZeroFlag ? 0x02 : 0) + (DisableInterruptFlag ?
0x04 : 0) +
01525             (DecimalFlag ? 8 : 0) + (setBreak ? 0x10 : 0) + 0x20 + (OverflowFlag ? 0x40 : 0)
+ (NegativeFlag ? 0x80 : 0));
01526     }

```

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

<i>addressingMode</i>	The addressing mode to use
-----------------------	----------------------------

Definition at line 2096 of file [W65C02.cs](#).

```

02097     {
02098         Accumulator = Accumulator ^ MemoryMap.Read(GetAddressByAddressingMode(addressingMode));

```

```

02099
02100         SetNegativeFlag(Accumulator);
02101         SetZeroFlag(Accumulator);
02102     }

```

6.39.3.12 ExecuteOpCode() void Hardware.W65C02.ExecuteOpCode () [inline], [private]

Executes an Opcode

Definition at line 239 of file W65C02.cs.

```

00240     {
00241         //The x+ cycles denotes that if a page wrap occurs, then an additional cycle is consumed.
00242         //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./
00243         //This is handled inside the GetValueFromMemory Method
00244         switch (CurrentOpCode)
00245         {
00246             #region Add / Subtract Operations
00247                 //ADC Add With Carry, Immediate, 2 Bytes, 2 Cycles
00248                 case 0x69:
00249                 {
00250                     AddWithCarryOperation(AddressingMode.Immediate);
00251                     break;
00252                 }
00253                 //ADC Add With Carry, Zero Page, 2 Bytes, 3 Cycles
00254                 case 0x65:
00255                 {
00256                     AddWithCarryOperation(AddressingMode.ZeroPage);
00257                     break;
00258                 }
00259                 //ADC Add With Carry, Zero Page X, 2 Bytes, 4 Cycles
00260                 case 0x75:
00261                 {
00262                     AddWithCarryOperation(AddressingMode.ZeroPageX);
00263                     break;
00264                 }
00265                 //ADC Add With Carry, Absolute, 3 Bytes, 4 Cycles
00266                 case 0x6D:
00267                 {
00268                     AddWithCarryOperation(AddressingMode.Absolute);
00269                     break;
00270                 }
00271                 //ADC Add With Carry, Absolute X, 3 Bytes, 4+ Cycles
00272                 case 0x7D:
00273                 {
00274                     AddWithCarryOperation(AddressingMode.AbsoluteX);
00275                     break;
00276                 }
00277                 //ADC Add With Carry, Absolute Y, 3 Bytes, 4+ Cycles
00278                 case 0x79:
00279                 {
00280                     AddWithCarryOperation(AddressingMode.AbsoluteY);
00281                     break;
00282                 }
00283                 //ADC Add With Carry, Indexed Indirect, 2 Bytes, 6 Cycles
00284                 case 0x61:
00285                 {
00286                     AddWithCarryOperation(AddressingMode.IndirectX);
00287                     break;
00288                 }
00289                 //ADC Add With Carry, Indexed Indirect, 2 Bytes, 5+ Cycles
00290                 case 0x71:
00291                 {
00292                     AddWithCarryOperation(AddressingMode.IndirectY);
00293                     break;
00294                 }
00295                 //SBC Subtract with Borrow, Immediate, 2 Bytes, 2 Cycles
00296                 case 0xE9:
00297                 {
00298                     SubtractWithBorrowOperation(AddressingMode.Immediate);
00299                     break;
00300                 }
00301                 //SBC Subtract with Borrow, Zero Page, 2 Bytes, 3 Cycles
00302                 case 0xE5:
00303                 {
00304                     SubtractWithBorrowOperation(AddressingMode.ZeroPage);
00305                     break;
00306                 }
00307                 //SBC Subtract with Borrow, Zero Page X, 2 Bytes, 4 Cycles

```

```

00308         case 0xF5:
00309             {
00310                 SubtractWithBorrowOperation(AddressingMode.ZeroPageX);
00311                 break;
00312             }
00313         //SBC Subtract with Borrow, Absolute, 3 Bytes, 4 Cycles
00314         case 0xED:
00315             {
00316                 SubtractWithBorrowOperation(AddressingMode.Absolute);
00317                 break;
00318             }
00319         //SBC Subtract with Borrow, Absolute X, 3 Bytes, 4+ Cycles
00320         case 0xFD:
00321             {
00322                 SubtractWithBorrowOperation(AddressingMode.AbsoluteX);
00323                 break;
00324             }
00325         //SBC Subtract with Borrow, Absolute Y, 3 Bytes, 4+ Cycles
00326         case 0xF9:
00327             {
00328                 SubtractWithBorrowOperation(AddressingMode.AbsoluteY);
00329                 break;
00330             }
00331         //SBC Subtract with Borrow, Indexed Indirect, 2 Bytes, 6 Cycles
00332         case 0xE1:
00333             {
00334                 SubtractWithBorrowOperation(AddressingMode.IndirectX);
00335                 break;
00336             }
00337         //SBC Subtract with Borrow, Indexed Indirect, 2 Bytes, 5+ Cycles
00338         case 0xF1:
00339             {
00340                 SubtractWithBorrowOperation(AddressingMode.IndirectY);
00341                 break;
00342             }
00343     #endregion
00344
00345     #region Branch Operations
00346         //BCC Branch if Carry is Clear, Relative, 2 Bytes, 2++ Cycles
00347         case 0x90:
00348             {
00349                 BranchOperation(!CarryFlag);
00350                 break;
00351             }
00352         //BCS Branch if Carry is Set, Relative, 2 Bytes, 2++ Cycles
00353         case 0xB0:
00354             {
00355                 BranchOperation(CarryFlag);
00356                 break;
00357             }
00358         //BEQ Branch if Zero is Set, Relative, 2 Bytes, 2++ Cycles
00359         case 0xF0:
00360             {
00361                 BranchOperation(ZeroFlag);
00362                 break;
00363             }
00364         // BMI Branch if Negative Set
00365         case 0x30:
00366             {
00367                 BranchOperation(NegativeFlag);
00368                 break;
00369             }
00370         //BNE Branch if Zero is Not Set, Relative, 2 Bytes, 2++ Cycles
00371         case 0xD0:
00372             {
00373                 BranchOperation(!ZeroFlag);
00374                 break;
00375             }
00376         // BPL Branch if Negative Clear, 2 Bytes, 2++ Cycles
00377         case 0x10:
00378             {
00379                 BranchOperation(!NegativeFlag);
00380                 break;
00381             }
00382         // BVC Branch if Overflow Clear, 2 Bytes, 2++ Cycles
00383         case 0x50:
00384             {
00385                 BranchOperation(!OverflowFlag);
00386                 break;
00387             }
00388         // BVS Branch if Overflow Set, 2 Bytes, 2++ Cycles
00389         case 0x70:
00390             {
00391                 BranchOperation(OverflowFlag);
00392                 break;
00393             }
00394

```



```

00395     }
00396 #endregion
00397
00398 #region BitWise Comparison Operations
00399 //AND Compare Memory with Accumulator, Immediate, 2 Bytes, 2 Cycles
00400 case 0x29:
00401 {
00402     AndOperation(AddressingMode.Immediate);
00403     break;
00404 }
00405 //AND Compare Memory with Accumulator, Zero Page, 2 Bytes, 3 Cycles
00406 case 0x25:
00407 {
00408     AndOperation(AddressingMode.ZeroPage);
00409     break;
00410 }
00411 //AND Compare Memory with Accumulator, Zero PageX, 2 Bytes, 3 Cycles
00412 case 0x35:
00413 {
00414     AndOperation(AddressingMode.ZeroPageX);
00415     break;
00416 }
00417 //AND Compare Memory with Accumulator, Absolute, 3 Bytes, 4 Cycles
00418 case 0x2D:
00419 {
00420     AndOperation(AddressingMode.Absolute);
00421     break;
00422 }
00423 //AND Compare Memory with Accumulator, AbsoluteX 3 Bytes, 4+ Cycles
00424 case 0x3D:
00425 {
00426     AndOperation(AddressingMode.AbsoluteX);
00427     break;
00428 }
00429 //AND Compare Memory with Accumulator, AbsoluteY, 3 Bytes, 4+ Cycles
00430 case 0x39:
00431 {
00432     AndOperation(AddressingMode.AbsoluteY);
00433     break;
00434 }
00435 //AND Compare Memory with Accumulator, IndexedIndirect, 2 Bytes, 6 Cycles
00436 case 0x21:
00437 {
00438     AndOperation(AddressingMode.IndirectX);
00439     break;
00440 }
00441 //AND Compare Memory with Accumulator, IndirectIndexed, 2 Bytes, 5 Cycles
00442 case 0x31:
00443 {
00444     AndOperation(AddressingMode.IndirectY);
00445     break;
00446 }
00447 //BIT Compare Memory with Accumulator, Zero Page, 2 Bytes, 3 Cycles
00448 case 0x24:
00449 {
00450     BitOperation(AddressingMode.ZeroPage);
00451     break;
00452 }
00453 //BIT Compare Memory with Accumulator, Absolute, 2 Bytes, 4 Cycles
00454 case 0x2C:
00455 {
00456     BitOperation(AddressingMode.Absolute);
00457     break;
00458 }
00459 //EOR Exclusive OR Memory with Accumulator, Immediate, 2 Bytes, 2 Cycles
00460 case 0x49:
00461 {
00462     EorOperation(AddressingMode.Immediate);
00463     break;
00464 }
00465 //EOR Exclusive OR Memory with Accumulator, Zero Page, 2 Bytes, 3 Cycles
00466 case 0x45:
00467 {
00468     EorOperation(AddressingMode.ZeroPage);
00469     break;
00470 }
00471 //EOR Exclusive OR Memory with Accumulator, Zero Page X, 2 Bytes, 4 Cycles
00472 case 0x55:
00473 {
00474     EorOperation(AddressingMode.ZeroPageX);
00475     break;
00476 }
00477 //EOR Exclusive OR Memory with Accumulator, Absolute, 3 Bytes, 4 Cycles
00478 case 0x4D:
00479 {
00480     EorOperation(AddressingMode.Absolute);
00481     break;

```

```

00482     }
00483     //EOR Exclusive OR Memory with Accumulator, Absolute X, 3 Bytes, 4+ Cycles
00484     case 0x5D:
00485     {
00486         EorOperation(AddressingMode.AbsoluteX);
00487         break;
00488     }
00489     //EOR Exclusive OR Memory with Accumulator, Absolute Y, 3 Bytes, 4+ Cycles
00490     case 0x59:
00491     {
00492         EorOperation(AddressingMode.AbsoluteY);
00493         break;
00494     }
00495     //EOR Exclusive OR Memory with Accumulator, IndexedIndirect, 2 Bytes 6 Cycles
00496     case 0x41:
00497     {
00498         EorOperation(AddressingMode.IndirectX);
00499         break;
00500     }
00501     //EOR Exclusive OR Memory with Accumulator, IndirectIndexed, 2 Bytes 5 Cycles
00502     case 0x51:
00503     {
00504         EorOperation(AddressingMode.IndirectY);
00505         break;
00506     }
00507     //ORA Compare Memory with Accumulator, Immediate, 2 Bytes, 2 Cycles
00508     case 0x09:
00509     {
00510         OrOperation(AddressingMode.Immediate);
00511         break;
00512     }
00513     //ORA Compare Memory with Accumulator, Zero Page, 2 Bytes, 2 Cycles
00514     case 0x05:
00515     {
00516         OrOperation(AddressingMode.ZeroPage);
00517         break;
00518     }
00519     //ORA Compare Memory with Accumulator, Zero PageX, 2 Bytes, 4 Cycles
00520     case 0x15:
00521     {
00522         OrOperation(AddressingMode.ZeroPageX);
00523         break;
00524     }
00525     //ORA Compare Memory with Accumulator, Absolute, 3 Bytes, 4 Cycles
00526     case 0x0D:
00527     {
00528         OrOperation(AddressingMode.Absolute);
00529         break;
00530     }
00531     //ORA Compare Memory with Accumulator, AbsoluteX 3 Bytes, 4+ Cycles
00532     case 0x1D:
00533     {
00534         OrOperation(AddressingMode.AbsoluteX);
00535         break;
00536     }
00537     //ORA Compare Memory with Accumulator, AbsoluteY, 3 Bytes, 4+ Cycles
00538     case 0x19:
00539     {
00540         OrOperation(AddressingMode.AbsoluteY);
00541         break;
00542     }
00543     //ORA Compare Memory with Accumulator, IndexedIndirect, 2 Bytes, 6 Cycles
00544     case 0x01:
00545     {
00546         OrOperation(AddressingMode.IndirectX);
00547         break;
00548     }
00549     //ORA Compare Memory with Accumulator, IndirectIndexed, 2 Bytes, 5 Cycles
00550     case 0x11:
00551     {
00552         OrOperation(AddressingMode.IndirectY);
00553         break;
00554     }
00555 #endregion
00556
00557 #region Clear Flag Operations
00558     //CLC Clear Carry Flag, Implied, 1 Byte, 2 Cycles
00559     case 0x18:
00560     {
00561         CarryFlag = false;
00562         IncrementCycleCount();
00563         break;
00564     }
00565     //CLD Clear Decimal Flag, Implied, 1 Byte, 2 Cycles
00566     case 0xD8:
00567     {
00568         DecimalFlag = false;

```

```

00569             IncrementCycleCount();
00570             break;
00571         }
00572     }
00573     //CLI Clear Interrupt Flag, Implied, 1 Byte, 2 Cycles
00574     case 0x58:
00575     {
00576         DisableInterruptFlag = false;
00577         IncrementCycleCount();
00578         break;
00579     }
00580 }
00581 //CLV Clear Overflow Flag, Implied, 1 Byte, 2 Cycles
00582 case 0xB8:
00583 {
00584     OverflowFlag = false;
00585     IncrementCycleCount();
00586     break;
00587 }
00588 }
00589 #endregion
00590
00591 #region Compare Operations
00592 //CMP Compare Accumulator with Memory, Immediate, 2 Bytes, 2 Cycles
00593 case 0xC9:
00594 {
00595     CompareOperation(AddressingMode.Immediate, Accumulator);
00596     break;
00597 }
00598 //CMP Compare Accumulator with Memory, Zero Page, 2 Bytes, 3 Cycles
00599 case 0xC5:
00600 {
00601     CompareOperation(AddressingMode.ZeroPage, Accumulator);
00602     break;
00603 }
00604 //CMP Compare Accumulator with Memory, Zero Page x, 2 Bytes, 4 Cycles
00605 case 0xD5:
00606 {
00607     CompareOperation(AddressingMode.ZeroPageX, Accumulator);
00608     break;
00609 }
00610 //CMP Compare Accumulator with Memory, Absolute, 3 Bytes, 4 Cycles
00611 case 0xCD:
00612 {
00613     CompareOperation(AddressingMode.Absolute, Accumulator);
00614     break;
00615 }
00616 //CMP Compare Accumulator with Memory, Absolute X, 2 Bytes, 4 Cycles
00617 case 0xDD:
00618 {
00619     CompareOperation(AddressingMode.AbsoluteX, Accumulator);
00620     break;
00621 }
00622 //CMP Compare Accumulator with Memory, Absolute Y, 2 Bytes, 4 Cycles
00623 case 0xD9:
00624 {
00625     CompareOperation(AddressingMode.AbsoluteY, Accumulator);
00626     break;
00627 }
00628 //CMP Compare Accumulator with Memory, Indirect X, 2 Bytes, 6 Cycles
00629 case 0xC1:
00630 {
00631     CompareOperation(AddressingMode.IndirectX, Accumulator);
00632     break;
00633 }
00634 //CMP Compare Accumulator with Memory, Indirect Y, 2 Bytes, 5 Cycles
00635 case 0xD1:
00636 {
00637     CompareOperation(AddressingMode.IndirectY, Accumulator);
00638     break;
00639 }
00640 //CPX Compare Accumulator with X Register, Immediate, 2 Bytes, 2 Cycles
00641 case 0xE0:
00642 {
00643     CompareOperation(AddressingMode.Immediate, XRegister);
00644     break;
00645 }
00646 //CPX Compare Accumulator with X Register, Zero Page, 2 Bytes, 3 Cycles
00647 case 0xE4:
00648 {
00649     CompareOperation(AddressingMode.ZeroPage, XRegister);
00650     break;
00651 }
00652 //CPX Compare Accumulator with X Register, Absolute, 3 Bytes, 4 Cycles
00653 case 0xEC:
00654 {
00655     CompareOperation(AddressingMode.Absolute, XRegister);

```

```

00656         break;
00657     }
00658     //CPY Compare Accumulator with Y Register, Immediate, 2 Bytes, 2 Cycles
00659     case 0xC0:
00660     {
00661         CompareOperation(AddressingMode.Immediate, YRegister);
00662         break;
00663     }
00664     //CPY Compare Accumulator with Y Register, Zero Page, 2 Bytes, 3 Cycles
00665     case 0xC4:
00666     {
00667         CompareOperation(AddressingMode.ZeroPage, YRegister);
00668         break;
00669     }
00670     //CPY Compare Accumulator with Y Register, Absolute, 3 Bytes, 4 Cycles
00671     case 0xCC:
00672     {
00673         CompareOperation(AddressingMode.Absolute, YRegister);
00674         break;
00675     }
00676 #endregion
00677
00678 #region Increment/Decrement Operations
00679     //DEC Decrement Memory by One, Zero Page, 2 Bytes, 5 Cycles
00680     case 0xC6:
00681     {
00682         ChangeMemoryByOne(AddressingMode.ZeroPage, true);
00683         break;
00684     }
00685     //DEC Decrement Memory by One, Zero Page X, 2 Bytes, 6 Cycles
00686     case 0xD6:
00687     {
00688         ChangeMemoryByOne(AddressingMode.ZeroPageX, true);
00689         break;
00690     }
00691     //DEC Decrement Memory by One, Absolute, 3 Bytes, 6 Cycles
00692     case 0xCE:
00693     {
00694         ChangeMemoryByOne(AddressingMode.Absolute, true);
00695         break;
00696     }
00697     //DEC Decrement Memory by One, Absolute X, 3 Bytes, 7 Cycles
00698     case 0xDE:
00699     {
00700         ChangeMemoryByOne(AddressingMode.AbsoluteX, true);
00701         IncrementCycleCount();
00702         break;
00703     }
00704     //DEX Decrement X Register by One, Implied, 1 Bytes, 2 Cycles
00705     case 0xCA:
00706     {
00707         ChangeRegisterByOne(true, true);
00708         break;
00709     }
00710     //DEY Decrement Y Register by One, Implied, 1 Bytes, 2 Cycles
00711     case 0x88:
00712     {
00713         ChangeRegisterByOne(false, true);
00714         break;
00715     }
00716     //INC Increment Memory by One, Zero Page, 2 Bytes, 5 Cycles
00717     case 0xE6:
00718     {
00719         ChangeMemoryByOne(AddressingMode.ZeroPage, false);
00720         break;
00721     }
00722     //INC Increment Memory by One, Zero Page X, 2 Bytes, 6 Cycles
00723     case 0xF6:
00724     {
00725         ChangeMemoryByOne(AddressingMode.ZeroPageX, false);
00726         break;
00727     }
00728     //INC Increment Memory by One, Absolute, 3 Bytes, 6 Cycles
00729     case 0xEE:
00730     {
00731         ChangeMemoryByOne(AddressingMode.Absolute, false);
00732         break;
00733     }
00734     //INC Increment Memory by One, Absolute X, 3 Bytes, 7 Cycles
00735     case 0xFE:
00736     {
00737         ChangeMemoryByOne(AddressingMode.AbsoluteX, false);
00738         IncrementCycleCount();
00739         break;
00740     }
00741     //INX Increment X Register by One, Implied, 1 Bytes, 2 Cycles
00742     case 0xE8:

```

```

00743         {
00744             ChangeRegisterByOne(true, false);
00745             break;
00746         }
00747         //INY Increment Y Register by One, Implied, 1 Bytes, 2 Cycles
00748         case 0xC8:
00749             {
00750                 ChangeRegisterByOne(false, false);
00751                 break;
00752             }
00753     #endregion
00754
00755     #region GOTO and GOSUB Operations
00756     //JMP Jump to New Location, Absolute 3 Bytes, 3 Cycles
00757     case 0x4C:
00758         {
00759             ProgramCounter = GetAddressByAddressingMode(AddressingMode.Absolute);
00760             break;
00761         }
00762     //JMP Jump to New Location, Indirect 3 Bytes, 5 Cycles
00763     case 0x6C:
00764         {
00765             ProgramCounter = GetAddressByAddressingMode(AddressingMode.Absolute);
00766
00767             if ((ProgramCounter & 0xFF) == 0xFF)
00768             {
00769                 //Get the first half of the address
00770                 int address = MemoryMap.Read(ProgramCounter);
00771
00772                 //Get the second half of the address, due to the issue with page boundary
00773                 //it reads from the wrong location!
00774                 address += 256 * MemoryMap.Read(ProgramCounter - 255);
00775                 ProgramCounter = address;
00776             }
00777             else
00778             {
00779                 ProgramCounter = GetAddressByAddressingMode(AddressingMode.Absolute);
00780             }
00781             break;
00782         }
00783     //JSR Jump to SubRoutine, Absolute, 3 Bytes, 6 Cycles
00784     case 0x20:
00785         {
00786             JumpToSubRoutineOperation();
00787             break;
00788         }
00789     //BRK Simulate IRQ, Implied, 1 Byte, 7 Cycles
00790     case 0x00:
00791         {
00792             BreakOperation(true, 0xFFFE);
00793             break;
00794         }
00795     //RTI Return From Interrupt, Implied, 1 Byte, 6 Cycles
00796     case 0x40:
00797         {
00798             ReturnFromInterruptOperation();
00799             break;
00800         }
00801     //RTS Return From Subroutine, Implied, 1 Byte, 6 Cycles
00802     case 0x60:
00803         {
00804             ReturnFromSubRoutineOperation();
00805             break;
00806         }
00807     #endregion
00808
00809     #region Load Value From Memory Operations
00810     //LDA Load Accumulator with Memory, Immediate, 2 Bytes, 2 Cycles
00811     case 0xA9:
00812         {
00813             Accumulator =
00814             MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.Immediate));
00815             SetZeroFlag(Accumulator);
00816             SetNegativeFlag(Accumulator);
00817             break;
00818         }
00819     //LDA Load Accumulator with Memory, Zero Page, 2 Bytes, 3 Cycles
00820     case 0xA5:
00821         {
00822             Accumulator =
00823             MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.ZeroPage));
00824             SetZeroFlag(Accumulator);
00825             SetNegativeFlag(Accumulator);
00826             break;
00827         }
00828     //LDA Load Accumulator with Memory, Zero Page X, 2 Bytes, 4 Cycles

```

```

00827         case 0xB5:
00828         {
00829             Accumulator =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.ZeroPageX));
00830             SetZeroFlag(Accumulator);
00831             SetNegativeFlag(Accumulator);
00832             break;
00833         }
00834         //LDA Load Accumulator with Memory, Absolute, 3 Bytes, 4 Cycles
00835         case 0xAD:
00836         {
00837             Accumulator =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.Absolute));
00838             SetZeroFlag(Accumulator);
00839             SetNegativeFlag(Accumulator);
00840             break;
00841         }
00842         //LDA Load Accumulator with Memory, Absolute X, 3 Bytes, 4+ Cycles
00843         case 0xBD:
00844         {
00845             Accumulator =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.AbsoluteX));
00846             SetZeroFlag(Accumulator);
00847             SetNegativeFlag(Accumulator);
00848             break;
00849         }
00850         //LDA Load Accumulator with Memory, Absolute Y, 3 Bytes, 4+ Cycles
00851         case 0xB9:
00852         {
00853             Accumulator =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.AbsoluteY));
00854             SetZeroFlag(Accumulator);
00855             SetNegativeFlag(Accumulator);
00856             break;
00857         }
00858         //LDA Load Accumulator with Memory, Index Indirect, 2 Bytes, 6 Cycles
00859         case 0xA1:
00860         {
00861             Accumulator =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.IndirectX));
00862             SetZeroFlag(Accumulator);
00863             SetNegativeFlag(Accumulator);
00864             break;
00865         }
00866         //LDA Load Accumulator with Memory, Indirect Index, 2 Bytes, 5+ Cycles
00867         case 0xB1:
00868         {
00869             Accumulator =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.IndirectY));
00870             SetZeroFlag(Accumulator);
00871             SetNegativeFlag(Accumulator);
00872             break;
00873         }
00874         //LDX Load X with memory, Immediate, 2 Bytes, 2 Cycles
00875         case 0xA2:
00876         {
00877             XRegister =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.Immediate));
00878             SetZeroFlag(XRegister);
00879             SetNegativeFlag(XRegister);
00880             break;
00881         }
00882         //LDX Load X with memory, Zero Page, 2 Bytes, 3 Cycles
00883         case 0xA6:
00884         {
00885             XRegister =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.ZeroPage));
00886             SetZeroFlag(XRegister);
00887             SetNegativeFlag(XRegister);
00888             break;
00889         }
00890         //LDX Load X with memory, Zero Page Y, 2 Bytes, 4 Cycles
00891         case 0xB6:
00892         {
00893             XRegister =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.ZeroPageY));
00894             SetZeroFlag(XRegister);
00895             SetNegativeFlag(XRegister);
00896             break;
00897         }
00898         //LDX Load X with memory, Absolute, 3 Bytes, 4 Cycles
00899         case 0xAE:
00900         {
00901             XRegister =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.Absolute));
00902             SetZeroFlag(XRegister);
00903             SetNegativeFlag(XRegister);

```

```

00904             break;
00905         }
00906         //LDX Load X with memory, Absolute Y, 3 Bytes, 4+ Cycles
00907         case 0xBE:
00908         {
00909             XRegister =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.AbsoluteY));
00910             SetZeroFlag(XRegister);
00911             SetNegativeFlag(XRegister);
00912             break;
00913         }
00914         //LDY Load Y with memory, Immediate, 2 Bytes, 2 Cycles
00915         case 0xA0:
00916         {
00917             YRegister =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.Immediate));
00918             SetZeroFlag(YRegister);
00919             SetNegativeFlag(YRegister);
00920             break;
00921         }
00922         //LDY Load Y with memory, Zero Page, 2 Bytes, 3 Cycles
00923         case 0xA4:
00924         {
00925             YRegister =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.ZeroPage));
00926             SetZeroFlag(YRegister);
00927             SetNegativeFlag(YRegister);
00928             break;
00929         }
00930         //LDY Load Y with memory, Zero Page X, 2 Bytes, 4 Cycles
00931         case 0xB4:
00932         {
00933             YRegister =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.ZeroPageX));
00934             SetZeroFlag(YRegister);
00935             SetNegativeFlag(YRegister);
00936             break;
00937         }
00938         //LDY Load Y with memory, Absolute, 3 Bytes, 4 Cycles
00939         case 0xAC:
00940         {
00941             YRegister =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.Absolute));
00942             SetZeroFlag(YRegister);
00943             SetNegativeFlag(YRegister);
00944             break;
00945         }
00946         //LDY Load Y with memory, Absolute X, 3 Bytes, 4+ Cycles
00947         case 0xBC:
00948         {
00949             YRegister =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.AbsoluteX));
00950             SetZeroFlag(YRegister);
00951             SetNegativeFlag(YRegister);
00952             break;
00953         }
00954         #endregion
00955
00956         #region Push/Pull Stack
00957         //PHA Push Accumulator onto Stack, Implied, 1 Byte, 3 Cycles
00958         case 0x48:
00959         {
00960             MemoryMap.Read(ProgramCounter + 1);
00961
00962             PokeStack((byte)Accumulator);
00963             StackPointer--;
00964             IncrementCycleCount();
00965             break;
00966         }
00967
00968         //PHP Push Flags onto Stack, Implied, 1 Byte, 3 Cycles
00969         case 0x08:
00970         {
00971             MemoryMap.Read(ProgramCounter + 1);
00972
00973             PushFlagsOperation();
00974             StackPointer--;
00975             IncrementCycleCount();
00976             break;
00977         }
00978         //PLA Pull Accumulator from Stack, Implied, 1 Byte, 4 Cycles
00979         case 0x68:
00980         {
00981             MemoryMap.Read(ProgramCounter + 1);
00982             StackPointer++;
00983             IncrementCycleCount();
00984

```

```

00985         Accumulator = PeekStack();
00986         SetNegativeFlag(Accumulator);
00987         SetZeroFlag(Accumulator);
00988
00989         IncrementCycleCount();
00990         break;
00991     }
00992     //PLP Pull Flags from Stack, Implied, 1 Byte, 4 Cycles
00993     case 0x28:
00994     {
00995         MemoryMap.Read(ProgramCounter + 1);
00996
00997         StackPointer++;
00998         IncrementCycleCount();
00999
01000         PullFlagsOperation();
01001
01002         IncrementCycleCount();
01003         break;
01004     }
01005     //TSX Transfer Stack Pointer to X Register, 1 Bytes, 2 Cycles
01006     case 0xBA:
01007     {
01008         XRegister = StackPointer;
01009
01010         SetNegativeFlag(XRegister);
01011         SetZeroFlag(XRegister);
01012         IncrementCycleCount();
01013         break;
01014     }
01015     //TXS Transfer X Register to Stack Pointer, 1 Bytes, 2 Cycles
01016     case 0x9A:
01017     {
01018         StackPointer = (byte)XRegister;
01019         IncrementCycleCount();
01020         break;
01021     }
01022 #endregion
01023
01024 #region Set Flag Operations
01025     //SEC Set Carry, Implied, 1 Bytes, 2 Cycles
01026     case 0x38:
01027     {
01028         CarryFlag = true;
01029         IncrementCycleCount();
01030         break;
01031     }
01032     //SED Set Interrupt, Implied, 1 Bytes, 2 Cycles
01033     case 0xF8:
01034     {
01035         DecimalFlag = true;
01036         IncrementCycleCount();
01037         break;
01038     }
01039     //SEI Set Interrupt, Implied, 1 Bytes, 2 Cycles
01040     case 0x78:
01041     {
01042         DisableInterruptFlag = true;
01043         IncrementCycleCount();
01044         break;
01045     }
01046 #endregion
01047
01048 #region Shift/Rotate Operations
01049     //ASL Shift Left 1 Bit Memory or Accumulator, Accumulator, 1 Bytes, 2 Cycles
01050     case 0x0A:
01051     {
01052         AslOperation(AddressingMode.Accumulator);
01053         break;
01054     }
01055     //ASL Shift Left 1 Bit Memory or Accumulator, Zero Page, 2 Bytes, 5 Cycles
01056     case 0x06:
01057     {
01058         AslOperation(AddressingMode.ZeroPage);
01059         break;
01060     }
01061     //ASL Shift Left 1 Bit Memory or Accumulator, Zero PageX, 2 Bytes, 6 Cycles
01062     case 0x16:
01063     {
01064         AslOperation(AddressingMode.ZeroPageX);
01065         break;
01066     }
01067     //ASL Shift Left 1 Bit Memory or Accumulator, Absolute, 3 Bytes, 6 Cycles
01068     case 0x0E:
01069     {
01070         AslOperation(AddressingMode.Absolute);
01071         break;

```



```

01072     }
01073     //ASL Shift Left 1 Bit Memory or Accumulator, AbsoluteX, 3 Bytes, 7 Cycles
01074     case 0x1E:
01075     {
01076         AslOperation(AddressingMode.AbsoluteX);
01077         IncrementCycleCount();
01078         break;
01079     }
01080     //LSR Shift Left 1 Bit Memory or Accumulator, Accumulator, 1 Bytes, 2 Cycles
01081     case 0x4A:
01082     {
01083         LsrOperation(AddressingMode.Accumulator);
01084         break;
01085     }
01086     //LSR Shift Left 1 Bit Memory or Accumulator, Zero Page, 2 Bytes, 5 Cycles
01087     case 0x46:
01088     {
01089         LsrOperation(AddressingMode.ZeroPage);
01090         break;
01091     }
01092     //LSR Shift Left 1 Bit Memory or Accumulator, Zero PageX, 2 Bytes, 6 Cycles
01093     case 0x56:
01094     {
01095         LsrOperation(AddressingMode.ZeroPageX);
01096         break;
01097     }
01098     //LSR Shift Left 1 Bit Memory or Accumulator, Absolute, 3 Bytes, 6 Cycles
01099     case 0x4E:
01100     {
01101         LsrOperation(AddressingMode.Absolute);
01102         break;
01103     }
01104     //LSR Shift Left 1 Bit Memory or Accumulator, AbsoluteX, 3 Bytes, 7 Cycles
01105     case 0x5E:
01106     {
01107         LsrOperation(AddressingMode.AbsoluteX);
01108         IncrementCycleCount();
01109         break;
01110     }
01111     //ROL Rotate Left 1 Bit Memory or Accumulator, Accumulator, 1 Bytes, 2 Cycles
01112     case 0x2A:
01113     {
01114         RolOperation(AddressingMode.Accumulator);
01115         break;
01116     }
01117     //ROL Rotate Left 1 Bit Memory or Accumulator, Zero Page, 2 Bytes, 5 Cycles
01118     case 0x26:
01119     {
01120         RolOperation(AddressingMode.ZeroPage);
01121         break;
01122     }
01123     //ROL Rotate Left 1 Bit Memory or Accumulator, Zero PageX, 2 Bytes, 6 Cycles
01124     case 0x36:
01125     {
01126         RolOperation(AddressingMode.ZeroPageX);
01127         break;
01128     }
01129     //ROL Rotate Left 1 Bit Memory or Accumulator, Absolute, 3 Bytes, 6 Cycles
01130     case 0x2E:
01131     {
01132         RolOperation(AddressingMode.Absolute);
01133         break;
01134     }
01135     //ROL Rotate Left 1 Bit Memory or Accumulator, AbsoluteX, 3 Bytes, 7 Cycles
01136     case 0x3E:
01137     {
01138         RolOperation(AddressingMode.AbsoluteX);
01139         IncrementCycleCount();
01140         break;
01141     }
01142     //ROR Rotate Right 1 Bit Memory or Accumulator, Accumulator, 1 Bytes, 2 Cycles
01143     case 0x6A:
01144     {
01145         RorOperation(AddressingMode.Accumulator);
01146         break;
01147     }
01148     //ROR Rotate Right 1 Bit Memory or Accumulator, Zero Page, 2 Bytes, 5 Cycles
01149     case 0x66:
01150     {
01151         RorOperation(AddressingMode.ZeroPage);
01152         break;
01153     }
01154     //ROR Rotate Right 1 Bit Memory or Accumulator, Zero PageX, 2 Bytes, 6 Cycles
01155     case 0x76:
01156     {
01157         RorOperation(AddressingMode.ZeroPageX);
01158         break;

```

Generated by WolfNet Computing using Doxygen 1.9.5

```

01237         (byte)XRegister);
01238             break;
01239         }
01239         //STY Store Index Y, Zero Page, 2 Bytes, 3 Cycles
01240         case 0x84:
01241         {
01242             MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.ZeroPage),
01243         (byte)YRegister);
01243             break;
01244         }
01245         //STY Store Index Y, Zero Page X, 2 Bytes, 4 Cycles
01246         case 0x94:
01247         {
01248             MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.ZeroPageX),
01249         (byte)YRegister);
01249             break;
01250         }
01251         //STY Store Index Y, Absolute, 2 Bytes, 4 Cycles
01252         case 0x8C:
01253         {
01254             MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.Absolute),
01255         (byte)YRegister);
01255             break;
01256         }
01257     #endregion
01258
01259     #region Transfer Operations
01260         //TAX Transfer Accumulator to X Register, Implied, 1 Bytes, 2 Cycles
01261         case 0xAA:
01262         {
01263             IncrementCycleCount();
01264             XRegister = Accumulator;
01265
01266             SetNegativeFlag(XRegister);
01267             SetZeroFlag(XRegister);
01268             break;
01269         }
01270         //TAY Transfer Accumulator to Y Register, 1 Bytes, 2 Cycles
01271         case 0xA8:
01272         {
01273             IncrementCycleCount();
01274             YRegister = Accumulator;
01275
01276             SetNegativeFlag(YRegister);
01277             SetZeroFlag(YRegister);
01278             break;
01279         }
01280         //TXA Transfer X Register to Accumulator, Implied, 1 Bytes, 2 Cycles
01281         case 0x8A:
01282         {
01283             IncrementCycleCount();
01284             Accumulator = XRegister;
01285
01286             SetNegativeFlag(Accumulator);
01287             SetZeroFlag(Accumulator);
01288             break;
01289         }
01290         //TYA Transfer Y Register to Accumulator, Implied, 1 Bytes, 2 Cycles
01291         case 0x98:
01292         {
01293             IncrementCycleCount();
01294             Accumulator = YRegister;
01295
01296             SetNegativeFlag(Accumulator);
01297             SetZeroFlag(Accumulator);
01298             break;
01299         }
01300     #endregion
01301
01302         //NOP Operation, Implied, 1 Byte, 2 Cycles
01303         case 0xEA:
01304         {
01305             IncrementCycleCount();
01306             break;
01307         }
01308
01309         default:
01310             throw new NotSupportedException(string.Format("The OpCode {0} is not supported",
01311         CurrentOpCode));
01311     }
01312 }

```

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

<i>addressingMode</i>	The addressing Mode to use
-----------------------	----------------------------

Returns

The memory Location

Definition at line 1340 of file W65C02.cs.

```
01341     {
01342         int address;
01343         int highByte;
01344         switch (addressingMode)
01345         {
01346             case (AddressingMode.Absolute):
01347             {
01348                 return (MemoryMap.Read(ProgramCounter++) | (MemoryMap.Read(ProgramCounter++) <<
01349 8));
01350             }
01351             case AddressingMode.AbsoluteX:
01352             {
01353                 //Get the low half of the address
01354                 address = MemoryMap.Read(ProgramCounter++);
01355                 //Get the high byte
01356                 highByte = MemoryMap.Read(ProgramCounter++);
01357                 //We crossed a page boundary, so an extra read has occurred.
01358                 //However, if this is an ASL, LSR, DEC, INC, ROR, ROL or STA operation, we do
01359                 not decrease it by 1.
01360                 if (address + XRegister > 0xFF)
01361                 {
01362                     switch (CurrentOpCode)
01363                     {
01364                         case 0x1E:
01365                         case 0xDE:
01366                         case 0xFE:
01367                         case 0x5E:
01368                         case 0x3E:
01369                         case 0x7E:
01370                         case 0x9D:
01371                         {
01372                             //This is a MemoryMap.Read Fetch Write Operation, so we don't
01373                             make the extra read.
01374                             return ((highByte << 8 | address) + XRegister) & 0xFFFF;
01375                         }
01376                         default:
01377                         {
01378                             MemoryMap.Read((((highByte << 8 | address) + XRegister) - 0xFF)
01379                             & 0xFFFF);
01380                             break;
01381                         }
01382                     }
01383                 }
01384                 return ((highByte << 8 | address) + XRegister) & 0xFFFF;
01385             }
01386             case AddressingMode.AbsoluteY:
01387             {
01388                 //Get the low half of the address
01389                 address = MemoryMap.Read(ProgramCounter++);
01390                 //Get the high byte
01391                 highByte = MemoryMap.Read(ProgramCounter++);
01392                 //We crossed a page boundary, so decrease the number of cycles by 1 if the
01393                 operation is not STA
01394                 if (address + YRegister > 0xFF && CurrentOpCode != 0x99)
01395                 {
01396                     MemoryMap.Read((((highByte << 8 | address) + YRegister) - 0xFF) & 0xFFFF);
01397                 }
```

```

01398
01399 //Bitshift the high byte into place, AND with FFFF to handle wrapping.
01400 return ((highByte << 8 | address) + YRegister) & 0xFFFF;
01401 }
01402 case AddressingMode.Immediate:
01403 {
01404     return ProgramCounter++;
01405 }
01406 case AddressingMode.IndirectX:
01407 {
01408     //Get the location of the address to retrieve
01409     address = MemoryMap.Read(ProgramCounter++);
01410     MemoryMap.Read(address);
01411
01412     address += XRegister;
01413
01414     //Now get the final Address. The is not a zero page address either.
01415     var finalAddress = MemoryMap.Read((address & 0xFF) | (MemoryMap.Read((address
+ 1) & 0xFF) << 8));
01416     return finalAddress;
01417 }
01418 case AddressingMode.IndirectY:
01419 {
01420     address = MemoryMap.Read(ProgramCounter++);
01421
01422     var finalAddress = MemoryMap.Read(address) + (MemoryMap.Read((address + 1) &
0xFF) << 8);
01423
01424     if ((finalAddress & 0xFF) + YRegister > 0xFF && CurrentOpCode != 0x91)
01425     {
01426         MemoryMap.Read((finalAddress + YRegister - 0xFF) & 0xFFFF);
01427     }
01428
01429     return (finalAddress + YRegister) & 0xFFFF;
01430 }
01431 case AddressingMode.Relative:
01432 {
01433     return ProgramCounter;
01434 }
01435 case (AddressingMode.ZeroPage):
01436 {
01437     address = MemoryMap.Read(ProgramCounter++);
01438     return address;
01439 }
01440 case (AddressingMode.ZeroPageX):
01441 {
01442     address = MemoryMap.Read(ProgramCounter++);
01443     MemoryMap.Read(address);
01444
01445     address += XRegister;
01446     address &= 0xFF;
01447
01448     //This address wraps if its greater than 0xFF
01449     if (address > 0xFF)
01450     {
01451         address -= 0x100;
01452         return address;
01453     }
01454
01455     return address;
01456 }
01457 case (AddressingMode.ZeroPageY):
01458 {
01459     address = MemoryMap.Read(ProgramCounter++);
01460     MemoryMap.Read(address);
01461
01462     address += YRegister;
01463     address &= 0xFF;
01464
01465     return address;
01466 }
01467 default:
01468     throw new InvalidOperationException(string.Format("The Address Mode '{0}' does not
require an address", addressingMode));
01469 }
01470 }

```

6.39.3.14 GetAddressingMode() `AddressingMode` Hardware.W65C02.GetAddressingMode () [inline], [private]

Definition at line 1685 of file [W65C02.cs](#).

```

01686     {
01687         switch (CurrentOpCode)
01688         {
01689             case 0x0D: //ORA
01690             case 0x2D: //AND
01691             case 0x4D: //EOR
01692             case 0x6D: //ADC
01693             case 0x8D: //STA
01694             case 0xAD: //LDA
01695             case 0xCD: //CMP
01696             case 0xED: //SBC
01697             case 0x0E: //ASL
01698             case 0x2E: //ROL
01699             case 0x4E: //LSR
01700             case 0x6E: //ROR
01701             case 0x8E: //SDX
01702             case 0xAE: //LDX
01703             case 0xCE: //DEC
01704             case 0xEE: //INC
01705             case 0x2C: //Bit
01706             case 0x4C: //JMP
01707             case 0x8C: //STY
01708             case 0xAC: //LDY
01709             case 0xCC: //CPY
01710             case 0xEC: //CPX
01711             case 0x20: //JSR
01712             {
01713                 return AddressingMode.Absolute;
01714             }
01715             case 0x1D: //ORA
01716             case 0x3D: //AND
01717             case 0x5D: //EOR
01718             case 0x7D: //ADC
01719             case 0x9D: //STA
01720             case 0xBD: //LDA
01721             case 0xDD: //CMP
01722             case 0xFD: //SBC
01723             case 0xBC: //LDY
01724             case 0xFE: //INC
01725             case 0x1E: //ASL
01726             case 0x3E: //ROL
01727             case 0x5E: //LSR
01728             case 0x7E: //ROR
01729             {
01730                 return AddressingMode.AbsoluteX;
01731             }
01732             case 0x19: //ORA
01733             case 0x39: //AND
01734             case 0x59: //EOR
01735             case 0x79: //ADC
01736             case 0x99: //STA
01737             case 0xB9: //LDA
01738             case 0xD9: //CMP
01739             case 0xF9: //SBC
01740             case 0xBE: //LDX
01741             {
01742                 return AddressingMode.AbsoluteY;
01743             }
01744             case 0x0A: //ASL
01745             case 0x4A: //LSR
01746             case 0x2A: //ROL
01747             case 0x6A: //ROR
01748             {
01749                 return AddressingMode.Accumulator;
01750             }
01751
01752             case 0x09: //ORA
01753             case 0x29: //AND
01754             case 0x49: //EOR
01755             case 0x69: //ADC
01756             case 0xA0: //LDY
01757             case 0xC0: //CPY
01758             case 0xE0: //CMP
01759             case 0xA2: //LDX
01760             case 0xA9: //LDA
01761             case 0xC9: //CMP
01762             case 0xE9: //SBC
01763             {
01764                 return AddressingMode.Immediate;
01765             }
01766             case 0x00: //BRK
01767             case 0x18: //CLC
01768             case 0xD8: //CLD
01769             case 0x58: //CLI
01770             case 0xB8: //CLV
01771             case 0xDE: //DEC
01772             case 0xCA: //DEX

```

```

01773         case 0x88: //DEY
01774         case 0xE8: //INX
01775         case 0xC8: //INY
01776         case 0xEA: //NOP
01777         case 0x48: //PHA
01778         case 0x08: //PHP
01779         case 0x68: //PLA
01780         case 0x28: //PLP
01781         case 0x40: //RTI
01782         case 0x60: //RTS
01783         case 0x38: //SEC
01784         case 0xF8: //SED
01785         case 0x78: //SEI
01786         case 0xAA: //TAX
01787         case 0xA8: //TAY
01788         case 0xBA: //TSX
01789         case 0x8A: //TXA
01790         case 0x9A: //TXS
01791         case 0x98: //TYA
01792         {
01793             return AddressingMode.Implied;
01794         }
01795         case 0x6C:
01796         {
01797             return AddressingMode.Indirect;
01798         }
01799
01800         case 0x61: //ADC
01801         case 0x21: //AND
01802         case 0xC1: //CMP
01803         case 0x41: //EOR
01804         case 0xA1: //LDA
01805         case 0x01: //ORA
01806         case 0xE1: //SBC
01807         case 0x81: //STA
01808         {
01809             return AddressingMode.IndirectX;
01810         }
01811         case 0x71: //ADC
01812         case 0x31: //AND
01813         case 0xD1: //CMP
01814         case 0x51: //EOR
01815         case 0xB1: //LDA
01816         case 0x11: //ORA
01817         case 0xF1: //SBC
01818         case 0x91: //STA
01819         {
01820             return AddressingMode.IndirectY;
01821         }
01822         case 0x90: //BCC
01823         case 0xB0: //BCS
01824         case 0xF0: //BEQ
01825         case 0x30: //BMI
01826         case 0xD0: //BNE
01827         case 0x10: //BPL
01828         case 0x50: //BVC
01829         case 0x70: //BVS
01830         {
01831             return AddressingMode.Relative;
01832         }
01833         case 0x65: //ADC
01834         case 0x25: //AND
01835         case 0x06: //ASL
01836         case 0x24: //BIT
01837         case 0xC5: //CMP
01838         case 0xE4: //CPX
01839         case 0xC4: //CPY
01840         case 0xC6: //DEC
01841         case 0x45: //EOR
01842         case 0xE6: //INC
01843         case 0xA5: //LDA
01844         case 0xA6: //LDX
01845         case 0xA4: //LDY
01846         case 0x46: //LSR
01847         case 0x05: //ORA
01848         case 0x26: //ROL
01849         case 0x66: //ROR
01850         case 0xE5: //SBC
01851         case 0x85: //STA
01852         case 0x86: //STX
01853         case 0x84: //STY
01854         {
01855             return AddressingMode.ZeroPage;
01856         }
01857         case 0x75: //ADC
01858         case 0x35: //AND
01859         case 0x16: //ASL

```

```

01860         case 0xD5: //CMP
01861         case 0xD6: //DEC
01862         case 0x55: //EOR
01863         case 0xF6: //INC
01864         case 0xB5: //LDA
01865         case 0xB6: //LDX
01866         case 0xB4: //LDY
01867         case 0x56: //LSR
01868         case 0x15: //ORA
01869         case 0x36: //ROL
01870         case 0x76: //ROR
01871         case 0xF5: //SBC
01872         case 0x95: //STA
01873         case 0x96: //STX
01874         case 0x94: //STY
01875         {
01876             return AddressingMode.ZeroPageX;
01877         }
01878         default:
01879             throw new NotSupportedException(string.Format("Opcode {0} is not supported",
CurrentOpCode));
01880     }
01881 }

```

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](#).

```

00210     {
00211         return _cycleCount;
00212     }

```

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](#).

```

00218     {
00219         _cycleCount++;
00220         CycleCountIncrementedAction();
00221
00222         _previousInterrupt = _interrupt;
00223         _interrupt = TriggerNmi || (TriggerIRQ && !DisableInterruptFlag);
00224     }

```

6.39.3.17 InterruptRequest() void Hardware.W65C02.InterruptRequest () [inline]

The InterruptRequest or IRQ

Definition at line 200 of file [W65C02.cs](#).

```

00201     {
00202         TriggerIRQ = true;
00203     }

```


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.

```
02314     {
02315         IncrementCycleCount();
02316
02317         //Put the high value on the stack, this should be the address after our operation -1
02318         //The RTS operation increments the PC by 1 which is why we don't move 2
02319         PokeStack((byte)((ProgramCounter + 1) >> 8) & 0xFF);
02320         StackPointer--;
02321         IncrementCycleCount();
02322
02323         PokeStack((byte)((ProgramCounter + 1) & 0xFF));
02324         StackPointer--;
02325         IncrementCycleCount();
02326
02327         ProgramCounter = GetAddressByAddressingMode(AddressingMode.Absolute);
02328     }
```

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

<i>addressingMode</i>	The addressing mode to use
-----------------------	----------------------------

Definition at line 2108 of file W65C02.cs.

```
02109     {
02110         int value;
02111         var memoryAddress = 0;
02112         if (addressingMode == AddressingMode.Accumulator)
02113         {
02114             MemoryMap.Read(ProgramCounter + 1);
02115             value = Accumulator;
02116         }
02117         else
02118         {
02119             memoryAddress = GetAddressByAddressingMode(addressingMode);
02120             value = MemoryMap.Read(memoryAddress);
02121         }
02122
02123         //Dummy Write
02124         if (addressingMode != AddressingMode.Accumulator)
02125         {
02126             MemoryMap.Write(memoryAddress, (byte)value);
02127         }
02128
02129         NegativeFlag = false;
02130
02131         //If the Zero bit is set, we have a carry
02132         CarryFlag = (value & 0x01) != 0;
02133
02134         value = (value >> 1);
02135
02136         SetZeroFlag(value);
02137         if (addressingMode == AddressingMode.Accumulator)
02138             Accumulator = value;
02139         else
02140         {
02141             MemoryMap.Write(memoryAddress, (byte)value);
02142         }
02143     }
```

6.39.3.20 MoveProgramCounterByRelativeValue() void Hardware.W65C02.MoveProgramCounterByRelativeValue (byte valueToMove) [inline], [private]

Moves the ProgramCounter in a given direction based on the value inputted

Definition at line 1476 of file W65C02.cs.

```
01477     {
01478         var movement = valueToMove > 127 ? (valueToMove - 255) : valueToMove;
01479
01480         var newProgramCounter = ProgramCounter + movement;
01481
01482         //This makes sure that we always land on the correct spot for a positive number
01483         if (movement >= 0)
01484             newProgramCounter++;
01485
01486         //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
01487         if (((ProgramCounter + 1 ^ newProgramCounter) & 0xff00) != 0x0000)
01488         {
01489             IncrementCycleCount();
01490         }
01491
01492         ProgramCounter = newProgramCounter;
01493         MemoryMap.Read(ProgramCounter);
01494     }
```

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.

```
00172     {
00173         SetDisassembly();
00174
00175         //Have to read this first otherwise it causes tests to fail on a NES
00176         CurrentOpCode = MemoryMap.Read(ProgramCounter);
00177
00178         ProgramCounter++;
00179
00180         ExecuteOpCode();
00181
00182         if (_previousInterrupt)
00183         {
00184             if (TriggerNmi)
00185             {
00186                 ProcessNMI();
00187                 TriggerNmi = false;
00188             }
00189             else if (TriggerIRQ)
00190             {
00191                 ProcessIRQ();
00192                 TriggerIRQ = false;
00193             }
00194         }
00195     }
```

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

<i>addressingMode</i>	The addressing mode to use
-----------------------	----------------------------

Definition at line 2149 of file W65C02.cs.

```
02150     {
02151         Accumulator = Accumulator | MemoryMap.Read(GetAddressByAddressingMode(addressingMode));
02152     }
02153     SetNegativeFlag(Accumulator);
02154     SetZeroFlag(Accumulator);
02155 }
```

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.

```
01501     {
01502         //The stack lives at 0x100-0x1FF, but the value is only a byte so it needs to be
01503         translated return MemoryMap.Read(StackPointer + 0x100);
01504     }
```

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

<i>value</i>	The value to be written to the stack
--------------	--------------------------------------

Definition at line 1511 of file W65C02.cs.

```
01512     {
01513         //The stack lives at 0x100-0x1FF, but the value is only a byte so it needs to be
01514         translated MemoryMap.Write(StackPointer + 0x100, value);
01515     }
```

6.39.3.25 ProcessIRQ() void Hardware.W65C02.ProcessIRQ () [inline], [private]

This is ran anytime an IRQ occurs

Definition at line 2425 of file W65C02.cs.

```
02426     {
02427         if (DisableInterruptFlag)
02428             return;
02429     }
02430     ProgramCounter--;
02431     BreakOperation(false, 0xFFFE);
02432     CurrentOpCode = MemoryMap.Read(ProgramCounter);
02433     SetDisassembly();
02434 }
02435 }
```

6.39.3.26 ProcessNMI() void Hardware.W65C02.ProcessNMI () [inline], [private]

This is ran anytime an NMI occurs

Definition at line 2413 of file [W65C02.cs](#).

```
02414     {
02415         ProgramCounter--;
02416         BreakOperation(false, 0xFFFA);
02417         CurrentOpCode = MemoryMap.Read(ProgramCounter);
02418
02419         SetDisassembly();
02420     }
```

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](#).

```
02298     {
02299         var flags = PeekStack();
02300         CarryFlag = (flags & 0x01) != 0;
02301         ZeroFlag = (flags & 0x02) != 0;
02302         DisableInterruptFlag = (flags & 0x04) != 0;
02303         DecimalFlag = (flags & 0x08) != 0;
02304         OverflowFlag = (flags & 0x40) != 0;
02305         NegativeFlag = (flags & 0x80) != 0;
02306
02307
02308     }
```

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](#).

```
02290     {
02291         PokeStack(ConvertFlagsToByte(true));
02292     }
```

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](#).

```
00153     {
00154         ResetCycleCount();
00155         StackPointer = 0x1FD;
00156         //Set the Program Counter to the Reset Vector Address.
00157         ProgramCounter = 0xFFFC;
00158         //Reset the Program Counter to the Address contained in the Reset Vector
00159         ProgramCounter = (MemoryMap.Read(ProgramCounter) | (MemoryMap.Read(ProgramCounter + 1) <<
00160         8));
00160         CurrentOpCode = MemoryMap.Read(ProgramCounter);
00161         //SetDisassembly();
00162         DisableInterruptFlag = true;
00163         _previousInterrupt = false;
00164         TriggerNmi = false;
00165         TriggerIRQ = false;
00166     }
```

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.

```
00230     {
00231         _cycleCount = 0;
00232     }
```

6.39.3.31 ReturnFromInterruptOperation() void Hardware.W65C02.ReturnFromInterruptOperation () [inline], [private]

The RTI routine. Called when returning from a BRK operation. 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.

```
02391     {
02392         MemoryMap.Read(++ProgramCounter);
02393         StackPointer++;
02394         IncrementCycleCount();
02395
02396         PullFlagsOperation();
02397         StackPointer++;
02398         IncrementCycleCount();
02399
02400         var lowBit = PeekStack();
02401         StackPointer++;
02402         IncrementCycleCount();
02403
02404         var highBit = PeekStack() << 8;
02405         IncrementCycleCount();
02406
02407         ProgramCounter = (highBit | lowBit);
02408     }
```

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.

```
02334     {
02335         MemoryMap.Read(++ProgramCounter);
02336         StackPointer++;
02337         IncrementCycleCount();
02338
02339         var lowBit = PeekStack();
02340         StackPointer++;
02341         IncrementCycleCount();
02342
02343         var highBit = PeekStack() << 8;
02344         IncrementCycleCount();
02345
02346         ProgramCounter = (highBit | lowBit) + 1;
02347         IncrementCycleCount();
02348     }
```

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

<i>addressingMode</i>	The addressing mode to use
-----------------------	----------------------------

Definition at line 2161 of file W65C02.cs.

```

02162     {
02163         int value;
02164         var memoryAddress = 0;
02165         if (addressingMode == AddressingMode.Accumulator)
02166         {
02167             //Dummy MemoryMap.Read
02168             MemoryMap.Read(ProgramCounter + 1);
02169             value = Accumulator;
02170         }
02171         else
02172         {
02173             memoryAddress = GetAddressByAddressingMode(addressingMode);
02174             value = MemoryMap.Read(memoryAddress);
02175         }
02176
02177         //Dummy Write
02178         if (addressingMode != AddressingMode.Accumulator)
02179         {
02180             MemoryMap.Write(memoryAddress, (byte)value);
02181         }
02182
02183         //Store the carry flag before shifting it
02184         var newCarry = (0x80 & value) != 0;
02185
02186         //The And here ensures that if the value is greater than 255 it wraps properly.
02187         value = (value << 1) & 0xFE;
02188
02189         if (CarryFlag)
02190             value = value | 0x01;
02191
02192         CarryFlag = newCarry;
02193
02194         SetZeroFlag(value);
02195         SetNegativeFlag(value);
02196
02197
02198         if (addressingMode == AddressingMode.Accumulator)
02199             Accumulator = value;
02200         else
02201         {
02202             MemoryMap.Write(memoryAddress, (byte)value);
02203         }
02204     }

```

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

<i>addressingMode</i>	The addressing mode to use
-----------------------	----------------------------

Definition at line 2210 of file W65C02.cs.

```

02211     {
02212         int value;
02213         var memoryAddress = 0;
02214         if (addressingMode == AddressingMode.Accumulator)
02215         {
02216             //Dummy MemoryMap.Read
02217             MemoryMap.Read(ProgramCounter + 1);
02218             value = Accumulator;
02219         }
02220         else
02221         {
02222             memoryAddress = GetAddressByAddressingMode(addressingMode);

```

```

02223         value = MemoryMap.Read(memoryAddress);
02224     }
02225
02226     //Dummy Write
02227     if (addressingMode != AddressingMode.Accumulator)
02228     {
02229         MemoryMap.Write(memoryAddress, (byte)value);
02230     }
02231
02232     //Store the carry flag before shifting it
02233     var newCarry = (0x01 & value) != 0;
02234
02235     value = (value » 1);
02236
02237     //If the carry flag is set then 0x
02238     if (CarryFlag)
02239         value = value | 0x80;
02240
02241     CarryFlag = newCarry;
02242
02243     SetZeroFlag(value);
02244     SetNegativeFlag(value);
02245
02246     if (addressingMode == AddressingMode.Accumulator)
02247         Accumulator = value;
02248     else
02249     {
02250         MemoryMap.Write(memoryAddress, (byte)value);
02251     }
02252 }

```

6.39.3.35 SetDisassembly() void Hardware.W65C02.SetDisassembly () [inline], [private]

Definition at line 1529 of file W65C02.cs.

```

01530     {
01531         if (!_logger.IsDebugEnabled)
01532             return;
01533
01534         var addressMode = GetAddressingMode();
01535
01536         var currentProgramCounter = ProgramCounter;
01537
01538         currentProgramCounter = WrapProgramCounter(++currentProgramCounter);
01539         int? address1 = MemoryMap.Read(currentProgramCounter);
01540
01541         currentProgramCounter = WrapProgramCounter(++currentProgramCounter);
01542         int? address2 = MemoryMap.Read(currentProgramCounter);
01543
01544         string disassembledStep = string.Empty;
01545
01546         switch (addressMode)
01547         {
01548             case AddressingMode.Absolute:
01549             {
01550                 disassembledStep = string.Format("${0}{1}",
01551 address2.Value.ToString("X").PadLeft(2, '0'), address1.Value.ToString("X").PadLeft(2, '0'));
01552                 break;
01553             }
01554             case AddressingMode.AbsoluteX:
01555             {
01556                 disassembledStep = string.Format("${0}{1},X",
01557 address2.Value.ToString("X").PadLeft(2, '0'), address1.Value.ToString("X").PadLeft(2, '0'));
01558                 break;
01559             }
01560             case AddressingMode.AbsoluteY:
01561             {
01562                 disassembledStep = string.Format("${0}{1},Y",
01563 address2.Value.ToString("X").PadLeft(2, '0'), address1.Value.ToString("X").PadLeft(2, '0'));
01564                 break;
01565             }
01566             case AddressingMode.Accumulator:
01567             {
01568                 address1 = null;
01569                 address2 = null;
01570
01571                 disassembledStep = "A";
01572                 break;
01573             }
01574             case AddressingMode.Immediate:
01575             {

```

```

01573             disassembledStep = string.Format("#{0}",
address1.Value.ToString("X").PadLeft(4, '0'));
01574             address2 = null;
01575             break;
01576         }
01577         case AddressingMode.Implied:
01578         {
01579             address1 = null;
01580             address2 = null;
01581             break;
01582         }
01583         case AddressingMode.Indirect:
01584         {
01585             disassembledStep = string.Format("#{0}{1}",
address2.Value.ToString("X").PadLeft(2, '0'), address1.Value.ToString("X").PadLeft(2, '0'));
01586             break;
01587         }
01588         case AddressingMode.IndirectX:
01589         {
01590             address2 = null;
01591
01592             disassembledStep = string.Format("#{0},X",
address1.Value.ToString("X").PadLeft(2, '0'));
01593             break;
01594         }
01595         case AddressingMode.IndirectY:
01596         {
01597             address2 = null;
01598
01599             disassembledStep = string.Format("#{0}),Y",
address1.Value.ToString("X").PadLeft(2, '0'));
01600             break;
01601         }
01602         case AddressingMode.Relative:
01603         {
01604             var valueToMove = (byte)address1.Value;
01605
01606             var movement = valueToMove > 127 ? (valueToMove - 255) : valueToMove;
01607
01608             var newProgramCounter = ProgramCounter + movement;
01609
01610             //This makes sure that we always land on the correct spot for a positive
number
01611             if (movement >= 0)
01612                 newProgramCounter++;
01613
01614             var stringAddress = ProgramCounter.ToString("X").PadLeft(4, '0');
01615
01616             address1 = int.Parse(stringAddress.Substring(0, 2),
NumberStyles.AllowHexSpecifier);
01617             address2 = int.Parse(stringAddress.Substring(2, 2),
NumberStyles.AllowHexSpecifier);
01618
01619             disassembledStep = string.Format("#{0}",
newProgramCounter.ToString("X").PadLeft(4, '0'));
01620
01621             break;
01622         }
01623         case AddressingMode.ZeroPage:
01624         {
01625             address2 = null;
01626
01627             disassembledStep = string.Format("#{0}",
address1.Value.ToString("X").PadLeft(2, '0'));
01628             break;
01629         }
01630         case AddressingMode.ZeroPageX:
01631         {
01632             address2 = null;
01633
01634             disassembledStep = string.Format("#{0},X",
address1.Value.ToString("X").PadLeft(2, '0'));
01635             break;
01636         }
01637         case AddressingMode.ZeroPageY:
01638         {
01639             address2 = null;
01640
01641             disassembledStep = string.Format("#{0},Y",
address1.Value.ToString("X").PadLeft(4, '0'));
01642             break;
01643         }
01644         default:
01645             throw new InvalidEnumArgumentException("Invalid Addressing Mode");
01646     }
01647 }
01648

```



```

01649
01650         CurrentDisassembly = new Disassembly
01651     {
01652         HighAddress = address2.HasValue ? address2.Value.ToString("X").PadLeft(2, '0') :
string.Empty,
01653         LowAddress = address1.HasValue ? address1.Value.ToString("X").PadLeft(2, '0') :
string.Empty,
01654         OpCodeString = CurrentOpCode.ConvertOpCodeIntoString(),
01655         DisassemblyOutput = disassembledStep
01656     };
01657
01658     _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}",
01659         ProgramCounter.ToString("X").PadLeft(4, '0'),
01660         CurrentOpCode.ToString("X").PadLeft(2, '0'),
01661         CurrentDisassembly.LowAddress,
01662         CurrentDisassembly.HighAddress,
01663
01664         CurrentDisassembly.OpCodeString,
01665         CurrentDisassembly.DisassemblyOutput.PadRight(10, ' '),
01666
01667         Accumulator.ToString("X").PadLeft(3, '0'),
01668         XRegister.ToString("X").PadLeft(3, '0'),
01669         YRegister.ToString("X").PadLeft(3, '0'),
01670         StackPointer.ToString("X").PadLeft(3, '0'),
01671         Convert.ToInt16(NegativeFlag),
01672         Convert.ToInt16(OverflowFlag),
01673         0,
01674         Convert.ToInt16(DecimalFlag),
01675         Convert.ToInt16(DisableInterruptFlag),
01676         Convert.ToInt16(ZeroFlag),
01677         Convert.ToInt16(CarryFlag));
01678     }

```

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.

```

01319     {
01320         //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.
01321         NegativeFlag = value > 127;
01322     }

```

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.

```

01329     {
01330         ZeroFlag = value == 0;

```

```
01331     }
```

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

<i>addressingMode</i>	The addressing mode to use
-----------------------	----------------------------

Definition at line 2258 of file [W65C02.cs](#).

```
02259     {
02260         var memoryValue = MemoryMap.Read(GetAddressByAddressingMode(addressingMode));
02261         var newValue = DecimalFlag ? int.Parse(Accumulator.ToString("x")) -
int.Parse(memoryValue.ToString("x")) - (CarryFlag ? 0 : 1) : Accumulator - memoryValue - (CarryFlag
? 0 : 1);
02262
02263         CarryFlag = newValue >= 0;
02264
02265         if (DecimalFlag)
02266         {
02267             if (newValue < 0)
02268                 newValue += 100;
02269
02270             newValue = (int)Convert.ToInt64(string.Concat("0x", newValue), 16);
02271         }
02272         else
02273         {
02274             OverflowFlag = (((Accumulator ^ newValue) & 0x80) != 0) && (((Accumulator ^
memoryValue) & 0x80) != 0);
02275
02276             if (newValue < 0)
02277                 newValue += 256;
02278         }
02279
02280         SetNegativeFlag(newValue);
02281         SetZeroFlag(newValue);
02282
02283         Accumulator = newValue;
02284     }
```

6.39.3.39 WrapProgramCounter() `int Hardware.W65C02.WrapProgramCounter (int value) [inline], [private]`

Definition at line 1680 of file [W65C02.cs](#).

```
01681     {
01682         return value & 0xFFFF;
01683     }
```

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 instead 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 occurring 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](#).

```
00128 { 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](#).

```
00121 { 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](#).

```
00061 {
00062     get { return _programCounter; }
00063     private set { _programCounter = WrapProgramCounter(value); }
00064 }
```

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](#).

```
00071 {
00072     get { return _stackPointer; }
00073     private set
00074     {
00075         if (value > 0xFF)
00076             _stackPointer = value - 0x100;
00077         else if (value < 0x00)
00078             _stackPointer = value + 0x100;
00079         else
00080             _stackPointer = value;
00081     }
00082 }
```

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](#).

```
00136 { 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](#).

```
00124     {
00125         if (offset > MemoryMap.DeviceArea.Length)
00126             throw new ArgumentException(String.Format("The offset: {0} is greater than the device
area: {1}", offset, MemoryMap.DeviceArea.Length));
00127         T1Init(1000);
00128         T2Init(1000);
00129
00130         Offset = MemoryMap.DeviceArea.Offset | offset;
00131         Memory = new byte[length + 1];
00132         Length = length;
00133         Processor = processor;
00134     }
```

6.40.3 Member Function Documentation

6.40.3.1 Init() `void Hardware.W65C22.Init (`
`double timer) [inline]`

Initialization routine for the VIA.

Parameters

<i>timer</i>	Amount of time to set timers for.
--------------	-----------------------------------

Definition at line 151 of file [W65C22.cs](#).

```
00152     {
00153         T1Init(timer);
00154         T2Init(timer);
```



```
00155     }
```

6.40.3.2 OnT1Timeout() `void Hardware.W65C22.OnT1Timeout (`
 `object sender,`
 `ElapsedEventArgs e) [inline], [private]`

Called whenever System.Timers.Timer event elapses.

Parameters

<i>sender</i>	
<i>e</i>	

Definition at line 248 of file [W65C22.cs](#).

```
00249     {
00250         if (Processor.isRunning)
00251         {
00252             if (T1IsEnabled)
00253             {
00254                 Write(IFR, (byte)(IFR_T1 & IFR_INT));
00255                 if (T1IsIRQ)
00256                 {
00257                     Processor.InterruptRequest();
00258                 }
00259                 else
00260                 {
00261                     Processor.TriggerNmi = true;
00262                 }
00263             }
00264         }
00265     }
```

6.40.3.3 OnT2Timeout() `void Hardware.W65C22.OnT2Timeout (`
 `object sender,`
 `ElapsedEventArgs e) [inline], [private]`

Called whenever System.Timers.Timer event elapses

Parameters

<i>sender</i>	
<i>e</i>	

Definition at line 273 of file [W65C22.cs](#).

```
00274     {
00275         if (Processor.isRunning)
00276         {
00277             if (T2IsEnabled)
00278             {
00279                 Write(IFR, (byte)(IFR_T2 & IFR_INT));
00280                 if (T2IsIRQ)
00281                 {
00282                     Processor.InterruptRequest();
00283                 }
00284                 else
00285                 {
00286                     Processor.TriggerNmi = true;
00287                 }
00288             }
00289         }
```

```
00290     }
```

6.40.3.4 Read() `byte Hardware.W65C22.Read (int address) [inline]`

Routine to read from local memory.

Parameters

<i>address</i>	Address to read from.
----------------	-----------------------

Returns

Byte value stored in the local memory.

Definition at line 192 of file [W65C22.cs](#).

```
00193     {
00194         if ((Offset <= address) && (address <= End))
00195         {
00196             byte data = 0x00;
00197             if (T1TimerControl)
00198             {
00199                 data = (byte)(data | ACR_T1TC);
00200             }
00201             else if (T2TimerControl)
00202             {
00203                 data = (byte)(data | ACR_T2TC);
00204             }
00205             return data;
00206         }
00207         else
00208         {
00209             return Memory[address - Offset];
00210         }
00211     }
```

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](#).

```
00140     {
00141         T1TimerControl = false;
00142         T1IsEnabled = false;
00143         T2TimerControl = false;
00144         T2IsEnabled = false;
00145     }
```

6.40.3.6 T1Init() `void Hardware.W65C22.T1Init (double value) [inline]`

T1 counter initialization routine.

Parameters

<i>value</i>	Timer initialization value in milliseconds.
--------------	---

Definition at line 162 of file [W65C22.cs](#).

```

00163     {
00164         T1Object = new Timer(value);
00165         T1Object.Start();
00166         T1Object.Elapsed += OnT1Timeout;
00167         T1TimerControl = true;
00168         T1IsEnabled = false;
00169     }

```

6.40.3.7 T2Init() void Hardware.W65C22.T2Init (
 double value) [inline]

T2 counter initialization routine.

Parameters

<i>value</i>	Timer initialization value in milliseconds.
--------------	---

Definition at line 176 of file [W65C22.cs](#).

```

00177     {
00178         T2Object = new Timer(value);
00179         T2Object.Start();
00180         T2Object.Elapsed += OnT2Timeout;
00181         T2TimerControl = true;
00182         T2IsEnabled = false;
00183     }

```

6.40.3.8 Write() void Hardware.W65C22.Write (
 int address,
 byte data) [inline]

Writes data to the specified address in local memory.

Parameters

<i>address</i>	The address to write data to.
<i>data</i>	The data to be written.

Definition at line 219 of file [W65C22.cs](#).

```

00220     {
00221         if ((address == Offset + ACR) && ((data | ACR_T1TC) == ACR_T1TC))
00222         {
00223             T1TimerControl = true;
00224         }
00225         else if ((address == Offset + ACR) && ((data | ACR_T2TC) == ACR_T2TC))
00226         {
00227             T2TimerControl = true;
00228         }
00229         else if ((address == Offset + IER) && ((data | IER_T1) == IER_T1) && ((data | IER_EN) ==
IER_EN))
00230         {
00231             T1Init(T1Interval);
00232         }

```

```
00233         else if ((address == Offset + IER) && ((data | IER_T2) == IER_T2) && ((data | IER_EN) ==  
00234             IER_EN))  
00235         {  
00236             T2Init(T2Interval);  
00237         }  
00238         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 reference 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](#).

```
00042     {
00043         if (offset > MemoryMap.DeviceArea.Length)
00044             throw new ArgumentException(String.Format("The offset: {0} is greater than the device
area: {1}", offset, MemoryMap.DeviceArea.Length));
00045
00046         Processor = processor;
00047
00048         Offset = MemoryMap.DeviceArea.Offset | offset;
00049         Length = 0x04;
00050         Memory = new byte[Length + 1];
00051
00052         _backgroundWorker = new BackgroundWorker
00053         {
00054             WorkerSupportsCancellation = true
00055         };
00056         _backgroundWorker.DoWork += BackgroundWorkerDoWork;
00057         _backgroundWorker.RunWorkerAsync();
00058     }
```

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](#).

```
00679     {
00680         var worker = sender as BackgroundWorker;
00681
00682         while (true)
00683         {
00684             if (worker != null && worker.CancellationPending)
00685             {
00686                 e.Cancel = true;
00687                 return;
00688             }
00689
00690             if (Processor.isRunning)
00691             {
00692                 if (ReceiverFull || Overrun)
00693                 {
00694                     Memory[Offset + 1] = (byte)(Memory[Offset + 1] | 0x80);
00695                     Interrupted = true;
00696                     Processor.InterruptRequest();
00697                 }
00698
00699                 if (DataRead)
00700                 {
00701                     ReceiverFull = false;
00702                     Interrupted = false;
00703                     Overrun = false;
00704                     DataRead = false;
00705                 }
00706             }
00707         }
00708     }
```

6.41.3.2 ComFini() void Hardware.W65C51.ComFini (
 SerialPort serialPort) [inline], [private]

Called when the window is closed.

Parameters

<i>serialPort</i>	SerialPort Object to close
-------------------	----------------------------

Definition at line 196 of file W65C51.cs.

```
00197     {
00198         if (serialPort != null)
00199         {
00200             serialPort.Close();
00201         }
00202
00203         _backgroundWorker.CancelAsync();
00204         _backgroundWorker.DoWork -= BackgroundWorkerDoWork;
00205     }
```

6.41.3.3 ComInit() void Hardware.W65C51.ComInit (
 SerialPort serialPort) [inline], [private]

Called whenever the ACIA is initialized.

Parameters

<i>serialPort</i>	SerialPort object to initialize.
-------------------	----------------------------------

Definition at line 148 of file W65C51.cs.

```
00149     {
00150         try
00151         {
00152             serialPort.Open();
00153         }
00154         catch (UnauthorizedAccessException w)
00155         {
00156             FileStream file = new FileStream(FileLocations.ErrorFile, FileMode.OpenOrCreate,
FileAccess.ReadWrite);
00157             StreamWriter stream = new StreamWriter(file);
00158             stream.WriteLine(w.Message);
00159             stream.WriteLine(w.Source);
00160             stream.Flush();
00161             file.Flush();
00162             stream.Close();
00163             file.Close();
00164             return;
00165         }
00166         serialPort.ReadTimeout = 50;
00167         serialPort.WriteTimeout = 50;
00168         serialPort.DataReceived += new SerialDataReceivedEventHandler(SerialDataReceived);
00169         try
00170         {
00171             serialPort.Write("-----\r\n");
00172             serialPort.Write(" WolfNet 6502 WBC Emulator\r\n");
00173             serialPort.Write("-----\r\n");
00174             serialPort.Write("\r\n");
00175         }
00176         catch (TimeoutException t)
00177         {
00178             _ = t;
00179             FileStream file = new FileStream(FileLocations.ErrorFile, FileMode.OpenOrCreate,
FileAccess.ReadWrite);
00180             StreamWriter stream = new StreamWriter(file);
00181             stream.WriteLine("Read/Write error: Port timed out!");
00182             stream.WriteLine("Please ensure all cables are connected properly!");
00183             stream.Flush();
00184             file.Flush();
00185         }
```

```

00185         stream.Close();
00186         file.Close();
00187         return;
00188     }
00189 }

```

6.41.3.4 CommandRegister() void Hardware.W65C51.CommandRegister (byte data) [inline], [private]

Definition at line 297 of file [W65C51.cs](#).

```

00298 {
00299     byte test = (byte)(data & 0x20);
00300     if (test == 0x20)
00301     {
00302         throw new ArgumentException("Parity must NEVER be enabled!");
00303     }
00304
00305     test = (byte)(data & 0x10);
00306     if (test == 0x10)
00307     {
00308         EchoMode = true;
00309     }
00310     else
00311     {
00312         EchoMode = false;
00313     }
00314
00315     test = (byte)(data & 0x0C);
00316     if (test == 0x00)
00317     {
00318         Object.Handshake = Handshake.None;
00319         Object.RtsEnable = true;
00320         Object.Handshake = Handshake.RequestToSend;
00321     }
00322     else if (test == 0x04)
00323     {
00324         Object.Handshake = Handshake.None;
00325         Object.RtsEnable = false;
00326     }
00327     else if ((test == 0x08) || (test == 0x0C))
00328     {
00329         throw new NotImplementedException("This cannot be emulated on windows!");
00330     }
00331     else
00332     {
00333         throw new ArgumentOutOfRangeException("RtsControl is invalid!");
00334     }
00335
00336     test = (byte)(data & 0x02);
00337     if (test == 0x02)
00338     {
00339         InterruptDisabled = true;
00340     }
00341     else
00342     {
00343         InterruptDisabled = false;
00344     }
00345
00346     test = (byte)(data & 0x01);
00347     if (test == 0x01)
00348     {
00349         Object.DtrEnable = true;
00350     }
00351     else
00352     {
00353         Object.DtrEnable = false;
00354     }
00355 }

```

6.41.3.5 CommandRegisterUpdate() void Hardware.W65C51.CommandRegisterUpdate () [inline], [private]

Definition at line 357 of file [W65C51.cs](#).

```

00358     {
00359         byte data = Memory[Offset + 2];
00360
00361         if (ParityEnabled)
00362         {
00363             data |= 0x20;
00364         }
00365         else
00366         {
00367             data &= 0xD0;
00368         }
00369
00370         if (EchoMode)
00371         {
00372             data |= 0x10;
00373         }
00374         else
00375         {
00376             data &= 0xE0;
00377         }
00378
00379         data &= RtsControl;
00380
00381         if (InterruptDisabled)
00382         {
00383             data |= 0x02;
00384         }
00385         else
00386         {
00387             data &= 0x0D;
00388         }
00389         if (Object.DtrEnable)
00390         {
00391             data |= 0x01;
00392         }
00393         else
00394         {
00395             data &= 0x0E;
00396         }
00397
00398         Memory[Offset + 2] = data;
00399     }

```

6.41.3.6 ControlRegister() void Hardware.W65C51.ControlRegister (byte data) [inline], [private]

Definition at line 401 of file [W65C51.cs](#).

```

00402     {
00403         byte test = (byte)(data & 0x80);
00404         if (test == 0x80)
00405         {
00406             test = (byte)(data & 0x60);
00407             if (test == 0x60)
00408             {
00409                 Object.StopBits = StopBits.OnePointFive;
00410             }
00411             else
00412             {
00413                 Object.StopBits = StopBits.Two;
00414             }
00415         }
00416         else
00417         {
00418             Object.StopBits = StopBits.One;
00419         }
00420
00421         test = (byte)(data & 0x60);
00422         if (test == 0x20)
00423         {
00424             Object.DataBits = 7;
00425         }
00426         else if (test == 0x40)
00427         {
00428             Object.DataBits = 6;
00429         }
00430         else if (test == 0x60)
00431         {
00432             Object.DataBits = 5;
00433         }
00434         else

```

```

00435         {
00436             Object.DataBits = 8;
00437         }
00438
00439         test = (byte)(data & 0x10);
00440         if (!(test == 0x10))
00441         {
00442             throw new ArgumentException("External clock rate not available on the WolfNet 65C02
WBC!");
00443         }
00444
00445         test = (byte)(data & 0x0F);
00446         if (test == 0x00)
00447         {
00448             Object.BaudRate = 115200;
00449         }
00450         else if (test == 0x01)
00451         {
00452             Object.BaudRate = 50;
00453         }
00454         else if (test == 0x02)
00455         {
00456             Object.BaudRate = 75;
00457         }
00458         else if (test == 0x03)
00459         {
00460             Object.BaudRate = 110;
00461         }
00462         else if (test == 0x04)
00463         {
00464             Object.BaudRate = 135;
00465         }
00466         else if (test == 0x05)
00467         {
00468             Object.BaudRate = 150;
00469         }
00470         else if (test == 0x06)
00471         {
00472             Object.BaudRate = 300;
00473         }
00474         else if (test == 0x07)
00475         {
00476             Object.BaudRate = 600;
00477         }
00478         else if (test == 0x08)
00479         {
00480             Object.BaudRate = 1200;
00481         }
00482         else if (test == 0x09)
00483         {
00484             Object.BaudRate = 1800;
00485         }
00486         else if (test == 0x0A)
00487         {
00488             Object.BaudRate = 2400;
00489         }
00490         else if (test == 0x0B)
00491         {
00492             Object.BaudRate = 3600;
00493         }
00494         else if (test == 0x0C)
00495         {
00496             Object.BaudRate = 4800;
00497         }
00498         else if (test == 0x0D)
00499         {
00500             Object.BaudRate = 7200;
00501         }
00502         else if (test == 0x0E)
00503         {
00504             Object.BaudRate = 9600;
00505         }
00506         else
00507         {
00508             Object.BaudRate = 19200;
00509         }
00510     }

```

6.41.3.7 ControlRegisterUpdate() void Hardware.W65C51.ControlRegisterUpdate () [inline],
[private]

Definition at line 512 of file W65C51.cs.

```

00513     {
00514         byte controlRegister = Memory[Offset + 3];
00515
00516         if (Object.StopBits == StopBits.Two)
00517         {
00518             controlRegister |= 0x80;
00519         }
00520         else if ((Object.StopBits == StopBits.OnePointFive) && (Object.DataBits == 5) ||
00521 (Object.StopBits == StopBits.One))
00522         {
00523             controlRegister &= 0x7F;
00524         }
00525         else
00526         {
00527             throw new ArgumentOutOfRangeException("StopBits or combination of StopBits and
00528 DataBits is invalid!");
00529         }
00530
00531         if (Object.DataBits == 8)
00532         {
00533             controlRegister &= 0x9F;
00534         }
00535         else if (Object.DataBits == 7)
00536         {
00537             controlRegister |= 0x20;
00538         }
00539         else if (Object.DataBits == 6)
00540         {
00541             controlRegister |= 0x40;
00542         }
00543         else if (Object.DataBits == 5)
00544         {
00545             controlRegister |= 0x60;
00546         }
00547         else
00548         {
00549             throw new ArgumentOutOfRangeException("DataBits is out of range!");
00550         }
00551
00552         if (Object.BaudRate == 115200)
00553         {
00554             controlRegister &= 0xF0;
00555         }
00556         else if (Object.BaudRate == 50)
00557         {
00558             controlRegister |= 0x01;
00559         }
00560         else if (Object.BaudRate == 75)
00561         {
00562             controlRegister |= 0x02;
00563         }
00564         else if (Object.BaudRate == 110)
00565         {
00566             controlRegister |= 0x03;
00567         }
00568         else if (Object.BaudRate == 135)
00569         {
00570             controlRegister |= 0x04;
00571         }
00572         else if (Object.BaudRate == 150)
00573         {
00574             controlRegister |= 0x05;
00575         }
00576         else if (Object.BaudRate == 300)
00577         {
00578             controlRegister |= 0x06;
00579         }
00580         else if (Object.BaudRate == 600)
00581         {
00582             controlRegister |= 0x07;
00583         }
00584         else if (Object.BaudRate == 1200)
00585         {
00586             controlRegister |= 0x08;
00587         }
00588         else if (Object.BaudRate == 1800)
00589         {
00590             controlRegister |= 0x09;
00591         }
00592         else if (Object.BaudRate == 2400)
00593         {
00594             controlRegister |= 0x0A;
00595         }
00596         else if (Object.BaudRate == 3600)
00597         {
00598             controlRegister |= 0x0B;
00599         }
00600         else
00601         {
00602             throw new ArgumentOutOfRangeException("BaudRate is out of range!");
00603         }
00604     }

```

```

00597         }
00598         else if (Object.BaudRate == 4800)
00599         {
00600             controlRegister |= 0x0C;
00601         }
00602         else if (Object.BaudRate == 7200)
00603         {
00604             controlRegister |= 0x0D;
00605         }
00606         else if (Object.BaudRate == 9600)
00607         {
00608             controlRegister |= 0x0E;
00609         }
00610         else if (Object.BaudRate == 19200)
00611         {
00612             controlRegister |= 0x0F;
00613         }
00614         else
00615         {
00616             throw new ArgumentOutOfRangeException("BaudRate is outside the range of Baud Rates
supported by the W65C51!");
00617         }
00618
00619         Memory[Offset + 3] = controlRegister;
00620     }

```

6.41.3.8 Fini() void Hardware.W65C51.Fini () [inline]

Called when the window is closed.

Definition at line 95 of file [W65C51.cs](#).

```

00096     {
00097         ComFini(Object);
00098     }

```

6.41.3.9 HardwarePreRead() void Hardware.W65C51.HardwarePreRead (int address) [inline], [private]

Definition at line 274 of file [W65C51.cs](#).

```

00275     {
00276         if (address == Offset)
00277         {
00278             Interrupted = false;
00279             Overrun = false;
00280             ReceiverFull = false;
00281         }
00282         else if (address == Offset + 1)
00283         {
00284             StatusRegisterUpdate();
00285         }
00286         else if (address == Offset + 2)
00287         {
00288             CommandRegisterUpdate();
00289         }
00290         else if (address == Offset + 3)
00291         {
00292             ControlRegisterUpdate();
00293         }
00294     }
00295 }

```


6.41.3.10 HardwarePreWrite() `void Hardware.W65C51.HardwarePreWrite (`
`int address,`
`byte data) [inline], [private]`

Definition at line 254 of file [W65C51.cs](#).

```
00255     {
00256         if (address == Offset)
00257         {
00258             WriteCOM(data);
00259         }
00260         else if (address == Offset + 1)
00261         {
00262             Reset();
00263         }
00264         else if (address == Offset + 2)
00265         {
00266             CommandRegister(data);
00267         }
00268         else if (address == Offset + 3)
00269         {
00270             ControlRegister(data);
00271         }
00272     }
```

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

<i>port</i>	COM Port to use for I/O
-------------	-------------------------

Definition at line 70 of file [W65C51.cs](#).

```
00071     {
00072         Object = new SerialPort(port, defaultBaudRate, Parity.None, 8, StopBits.One);
00073         ObjectName = port;
00074
00075         ComInit(Object);
00076     }
```

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

<i>port</i>	COM Port to use for I/O
<i>baudRate</i>	Baud Rate to use for I/O

Definition at line 84 of file [W65C51.cs](#).

```
00085     {
00086         Object = new SerialPort(port, baudRate, Parity.None, 8, StopBits.One);
00087         ObjectName = port;
00088
00089         ComInit(Object);
00090     }
```

6.41.3.13 Read() `byte Hardware.W65C51.Read (int address) [inline]`

Returns the byte at a given address.

Parameters

<i>address</i>	
----------------	--

Returns

the byte being returned

Definition at line 107 of file [W65C51.cs](#).

```
00108     {
00109         HardwarePreRead(address);
00110         byte data = Memory[address - Offset];
00111         DataRead = true;
00112         return data;
00113     }
```

6.41.3.14 Reset() `void Hardware.W65C51.Reset () [inline]`

Definition at line 60 of file [W65C51.cs](#).

```
00061     {
00062         IsEnabled = false;
00063     }
```

6.41.3.15 SerialDataReceived() `void Hardware.W65C51.SerialDataReceived (object sender, SerialDataReceivedEventArgs e) [inline], [private]`

Called whenever SerialDataReceivedEventHandler event occurs.

Parameters

<i>sender</i>	
<i>e</i>	

Definition at line 213 of file [W65C51.cs](#).

```
00214     {
00215         try
00216         {
00217             if (EchoMode)
00218             {
00219                 WriteCOM(Convert.ToByte(Object.ReadByte()));
00220             }
00221             else
00222             {
00223                 if (!ReceiverFull)
00224                 {
00225                     ReceiverFull = true;
```

```

00226         }
00227         else
00228         {
00229             Overrun = true;
00230         }
00231         Memory[0] = Convert.ToByte(Object.ReadByte());
00232     }
00233
00234     if (!InterruptDisabled)
00235     {
00236         Interrupted = true;
00237         Processor.InterruptRequest();
00238     }
00239 }
00240 catch (Win32Exception w)
00241 {
00242     FileStream file = new FileStream(FileLocations.ErrorFile, FileMode.OpenOrCreate,
FileAccess.ReadWrite);
00243     StreamWriter stream = new StreamWriter(file);
00244     stream.WriteLine(w.Message);
00245     stream.WriteLine(w.ErrorCode.ToString());
00246     stream.WriteLine(w.Source);
00247     stream.Flush();
00248     stream.Close();
00249     file.Flush();
00250     file.Close();
00251 }
00252 }

```

6.41.3.16 StatusRegisterUpdate() void Hardware.W65C51.StatusRegisterUpdate () [inline], [private]

Definition at line 622 of file W65C51.cs.

```

00623 {
00624     byte statusRegister = Memory[Offset + 1];
00625
00626     if (Interrupted)
00627     {
00628         statusRegister |= 0x80;
00629     }
00630     else
00631     {
00632         statusRegister &= 0x7F;
00633     }
00634
00635     if (Object.DsrHolding == false)
00636     {
00637         statusRegister |= 0x40;
00638     }
00639     else
00640     {
00641         statusRegister &= 0xBF;
00642     }
00643
00644     if (Object.CDHolding)
00645     {
00646         statusRegister |= 0x20;
00647     }
00648     else
00649     {
00650         statusRegister &= 0xDF;
00651     }
00652
00653     statusRegister |= 0x10;
00654
00655     if (ReceiverFull)
00656     {
00657         statusRegister |= 0x08;
00658     }
00659     else
00660     {
00661         statusRegister &= 0xF7;
00662     }
00663
00664     if (Overrun)
00665     {
00666         statusRegister |= 0x04;
00667     }
00668     else
00669     {
00670         statusRegister &= 0xFB;

```

```

00671         }
00672
00673         statusRegister &= 0xFC;
00674
00675         Memory[Offset + 1] = statusRegister;
00676     }

```

6.41.3.17 Write() void Hardware.W65C51.Write (

```

    int address,
    byte data ) [inline]

```

Writes data to the given address.

Parameters

<i>address</i>	The address to write data to
<i>data</i>	The data to write

Definition at line 121 of file [W65C51.cs](#).

```

00122     {
00123         HardwarePreWrite(address, data);
00124         if (!(address == Offset) || (address == Offset + 1))
00125         {
00126             Memory[address - Offset] = data;
00127         }
00128     }

```

6.41.3.18 WriteCOM() void Hardware.W65C51.WriteCOM (

```

    byte data ) [inline]

```

Called in order to write to the serial port.

Parameters

<i>data</i>	Byte of data to send
-------------	----------------------

Definition at line 135 of file [W65C51.cs](#).

```

00136     {
00137         byte[] writeByte = new byte[] { data };
00138         Object.Write(writeByte, 0, 1);
00139     }

```

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](#).

```
00025 { get; set; }
```

6.41.5.15 ReceiverFull bool Hardware.W65C51.ReceiverFull [get], [set], [private]

Definition at line [36](#) of file [W65C51.cs](#).

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

6.41.5.16 RtsControl byte Hardware.W65C51.RtsControl [get], [set], [private]

Definition at line [37](#) of file [W65C51.cs](#).

```
00037 { 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.](#)

```
00001 namespace Emulator
00002 {
00003     /// <summary>
00004     /// Interaction logic for App.xaml
00005     /// </summary>
00006     public partial class App
00007     {
00008     }
00009 }
```

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.](#)

```
00001 using System;
00002 using System.Collections.Generic;
00003 using System.Linq;
00004 using System.Text;
00005 using System.Threading.Tasks;
00006
00007 namespace Emulator
00008 {
00009     public class ExitCodes
00010     {
00011         public static readonly int NO_ERROR = 0x00;
00012
00013         public static readonly int USER_ERROR = 0x01;
00014
00015         public static readonly int NO_BIOS = 0x02;
00016         public static readonly int LOAD_BIOS_FILE_ERROR = 0x03;
00017         public static readonly int BIOS_LOADPROGRAM_ERROR = 0x04;
00018         public static readonly int LOAD_ROM_FILE_ERROR = 0x05;
00019         public static readonly int ROM_LOADPROGRAM_ERROR = 0x06;
00020         public static readonly int LOAD_STATE_ERROR = 0x07;
00021     }
00022 }
```

7.5 Emulator/Classes/FileLocations.cs File Reference

Namespaces

- namespace [Emulator](#)

7.6 FileLocations.cs

[Go to the documentation of this file.](#)

```
00001 namespace Emulator
00002 {
00003     internal class FileLocations
00004     {
00005         #region Fields
00006         public static string SettingsFile = "./Settings.xml";
00007         public static string ErrorFile = "./Errors.log";
00008         #if DEBUG
00009         public static string BiosFile = "../../bios.bin";
00010         #else
00011         public static string BiosFile = "./bios.bin";
00012         #endif
00013     #endregion
00014     }
00015 }
```


7.7 Hardware/Classes/FileLocations.cs File Reference

Namespaces

- namespace [Hardware](#)

7.8 FileLocations.cs

[Go to the documentation of this file.](#)

```
00001 using System;
00002 using System.Collections.Generic;
00003 using System.Linq;
00004 using System.Text;
00005 using System.Threading.Tasks;
00006
00007 namespace Hardware
00008 {
00009     internal class FileLocations
00010     {
00011 #region Fields
00012         public static string ErrorFile = "./Hardware_Library_Errors.log";
00013         public static string MemoryDump = "./Hardware_Library_Memory_Dump.log";
00014 #endregion
00015     }
00016 }
```

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.](#)

```
00001 using Emulator.Model;
00002 using GalaSoft.MvvmLight.Messaging;
00003
00004 namespace Emulator
00005 {
00006     public static class SettingsFile
00007     {
00008         public static SettingsModel CreateNew()
00009         {
00010             // Create new settings file.
00011             SettingsModel _settings = new SettingsModel
00012             {
00013                 SettingsVersionMajor = Versioning.SettingsFile.Major,
00014                 SettingsVersionMinor = Versioning.SettingsFile.Minor,
00015                 SettingsVersionBuild = Versioning.SettingsFile.Build,
00016                 SettingsVersionRevision = Versioning.SettingsFile.Revision,
00017 #if DEBUG
00018                 ComPortName = "COM9",
00019 #else
00020                 ComPortName = "COM1",
00021 #endif
00022             };
00023             return _settings;
00024         }
00025     }
00026 }
```

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.](#)

```
00001 using System.Deployment;
00002 using System.Reflection;
00003 using System;
00004
00005 namespace Emulator
00006 {
00007     public static class Versioning
00008     {
00009         public class Product
00010         {
00011             public const int Major = 0;
00012             public const int Minor = 1;
00013             public const int Build = 3;
00014             public const int Revision = 1;
00015             public const string Title = Name;
00016             public const string Name = "WolfNet 65C02 WorkBench Computer Emulator";
00017             public const string Company = "WolfNet Computing";
00018             public const string Copyright = "Copyright 1' WolfNet Computing 2022";
00019             public const string VersionString = "0.2.3.1";
00020             public const string Description = "Emulator for the WolfNet 65C02 WorkBench Computer coded
in C# using the .NET Framework";
00021         }
00022         public class SettingsFile
00023         {
00024             public const byte Major = 1;
00025             public const byte Minor = 0;
00026             public const byte Build = 0;
00027             public const byte Revision = 0;
00028         }
00029     }
00030 }
00031 }
```

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.](#)

```
00001 namespace Hardware
00002 {
00003     internal class Versioning
00004     {
00005         public class Product
00006         {
00007             public const string Title = Name;
00008             public const string Name = "WolfNet 65C02 Hardware Library";
00009             public const string Company = "WolfNet Computing";
00010             public const string Copyright = "Copyright 1' WolfNet Computing 2022";
00011             public const string Version = "1.3.0.0";
00012             public const string Description = "65C02 Hardware Library, coded in C# using the .NET
Framework";
00013         }
00014     }
00015 }
```

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.](#)

```
00001 using System;
00002 using System.Collections.Generic;
00003 using System.Linq;
00004 using System.Text;
00005 using System.Threading.Tasks;
00006
00007 namespace Emulator
00008 {
00009     public interface IClosable
00010     {
00011         void Close();
00012     }
00013 }
```

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.](#)

```

00001 using GalaSoft.MvvmLight.Messaging;
00002 using Emulator.Model;
00003 using Emulator.ViewModel;
00004 using System;
00005 using System.ComponentModel;
00006 using System.Windows;
00007 using Hardware;
00008 using System.IO;
00009 using System.Xml.Serialization;
00010
00011 namespace Emulator
00012 {
00013     /// <summary>
00014     /// Interaction logic for MainWindow.xaml
00015     /// </summary>
00016     public partial class MainWindow : Window, IClosable
00017     {
00018         public MainWindow()
00019         {
00020             InitializeComponent();
00021             Messenger.Default.Register<NotificationMessage>(this, NotificationMessageReceived);
00022             Messenger.Default.Register<NotificationMessage<StateFileModel>>(this,
NotificationMessageReceived);
00023             Messenger.Default.Register<NotificationMessage<SettingsModel>>(this,
NotificationMessageReceived);
00024         }
00025
00026         private void ToClose(Object sender, EventArgs e)
00027         {
00028             Close();
00029         }
00030
00031         private void LoadFile(Object sender, EventArgs e)
00032         {
00033             Messenger.Default.Send(new NotificationMessage("LoadFile"));
00034         }
00035
00036         private void SaveFile(Object sender, EventArgs e)
00037         {
00038             Messenger.Default.Send(new NotificationMessage("SaveState"));
00039         }
00040
00041         private void CloseFile(Object sender, EventArgs e)
00042         {
00043             Messenger.Default.Send(new NotificationMessage("CloseFile"));
00044         }
00045
00046         private void NotificationMessageReceived(NotificationMessage notificationMessage)
00047         {
00048             if (notificationMessage.Notification == "CloseWindow")
00049             {
00050                 Close();
00051             }
00052         }
00053
00054         private void NotificationMessageReceived(NotificationMessage<StateFileModel>
notificationMessage)
00055         {
00056             if (notificationMessage.Notification == "SaveFileWindow")
00057             {
00058                 var saveFile = new SaveFile { DataContext = new
SaveFileViewModel(notificationMessage.Content) };
00059                 saveFile.ShowDialog();
00060             }
00061         }
00062
00063         private void NotificationMessageReceived(NotificationMessage<SettingsModel>
notificationMessage)
00064         {
00065             if (notificationMessage.Notification == "SettingsWindow")
00066             {
00067                 var settingsFile = new Settings { DataContext = new
SettingsViewModel(notificationMessage.Content) };
00068                 settingsFile.ShowDialog();
00069             }
00070         }
00071     }
00072 }

```

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.](#)

```
00001 using System.Collections.Generic;
00002
00003 namespace Emulator.Model
00004 {
00005     /// <summary>
00006     /// A Representation of a Breakpoint
00007     /// </summary>
00008     public class Breakpoint
00009     {
00010         /// <summary>
00011         /// Is the Breakpoint enabled or disabled
00012         /// </summary>
00013         public bool IsEnabled { get; set; }
00014
00015         /// <summary>
00016         /// The Value of the Breakpoint
00017         /// </summary>
00018         public string Value { get; set; }
00019
00020         /// <summary>
00021         /// The Type of breakpoint being set
00022         /// </summary>
00023         public string Type { get; set; }
00024
00025         public List<string> AllTypes
00026         {
00027             get { return BreakpointType.AllTypes; }
00028         }
00029     }
00030 }
```

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.](#)

```
00001 using System.Collections.Generic;
00002
00003 namespace Emulator.Model
00004 {
00005     /// <summary>
00006     /// The Type of Breakpoint
00007     /// </summary>
00008     public class BreakpointType
00009     {
00010         /// <summary>
00011         /// A Listing of all of the Current Types
00012         /// </summary>
00013         public static List<string> AllTypes = new List<string>
00014         {
00015             ProgramCounterType,
00016             NumberOfCycleType
00017         };
00018
00019         /// <summary>
00020         /// The ProgramCounter Breakpoint Type
00021         /// </summary>
00022         public const string ProgramCounterType = "Program Counter";
00023
00024         /// <summary>
00025         /// The CycleCount Breakpoint Type
00026         /// </summary>
00027         public const string NumberOfCycleType = "Number of Cycles";
00028     }
00029 }
00030 }
```

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.](#)

```
00001 namespace Emulator.Model
00002 {
00003     /// <summary>
00004     /// A Model of a Single Page of memory
00005     /// </summary>
00006     public class MemoryRowModel
00007     {
00008         /// <summary>
00009         /// The offset of this row. Expressed in hex
00010         /// </summary>
00011         public string Offset { get; set; }
00012         /// <summary>
00013         /// The memory at the location offset + 00
00014         /// </summary>
00015         public string Location00 { get; set; }
00016         /// <summary>
00017         /// The memory at the location offset + 01
00018         /// </summary>
00019         public string Location01 { get; set; }
00020     }
00021 }
```

```

00021 /// The memory at the location offset + 02
00022 /// </summary>
00023     public string Location02 { get; set; }
00024 /// <summary>
00025 /// The memory at the location offset + 03
00026 /// </summary>
00027     public string Location03 { get; set; }
00028 /// <summary>
00029 /// The memory at the location offset + 04
00030 /// </summary>
00031     public string Location04 { get; set; }
00032 /// <summary>
00033 /// The memory at the location offset + 05
00034 /// </summary>
00035     public string Location05 { get; set; }
00036 /// <summary>
00037 /// The memory at the location offset + 06
00038 /// </summary>
00039     public string Location06 { get; set; }
00040 /// <summary>
00041 /// The memory at the location offset + 07
00042 /// </summary>
00043     public string Location07 { get; set; }
00044 /// <summary>
00045 /// The memory at the location offset + 08
00046 /// </summary>
00047     public string Location08 { get; set; }
00048 /// <summary>
00049 /// The memory at the location offset + 09
00050 /// </summary>
00051     public string Location09 { get; set; }
00052 /// <summary>
00053 /// The memory at the location offset + 0A
00054 /// </summary>
00055     public string Location0A { get; set; }
00056 /// <summary>
00057 /// The memory at the location offset + 0B
00058 /// </summary>
00059     public string Location0B { get; set; }
00060 /// <summary>
00061 /// The memory at the location offset + 0C
00062 /// </summary>
00063     public string Location0C { get; set; }
00064 /// <summary>
00065 /// The memory at the location offset + 0D
00066 /// </summary>
00067     public string Location0D { get; set; }
00068 /// <summary>
00069 /// The memory at the location offset + 0E
00070 /// </summary>
00071     public string Location0E { get; set; }
00072 /// <summary>
00073 /// The memory at the location offset + 0F
00074 /// </summary>
00075     public string Location0F { get; set; }
00076     }
00077 }

```

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.

```

00001 using System;
00002 using Hardware;
00003
00004 namespace Emulator.Model
00005 {
00006     /// <summary>
00007     /// The OutputLog Model. Used by the outputlog grid to show a history of operations performed by the
    CPU
00008     /// </summary>
00009     [Serializable]
00010     public class OutputLog : Disassembly
00011     {
00012         public OutputLog(Disassembly disassembly)
00013         {
00014             DisassemblyOutput = disassembly.DisassemblyOutput;
00015             HighAddress = disassembly.HighAddress;
00016             LowAddress = disassembly.LowAddress;
00017             OpCodeString = disassembly.OpCodeString;
00018         }
00019
00020     /// <summary>
00021     /// The Program Counter Value
00022     /// </summary>
00023     public string ProgramCounter { get; set; }
00024     /// <summary>
00025     /// The Current Ope Code
00026     /// </summary>
00027     public string CurrentOpCode { get; set; }
00028     /// <summary>
00029     /// The X Register
00030     /// </summary>
00031     public string XRegister { get; set; }
00032     /// <summary>
00033     /// The Y Register
00034     /// </summary>
00035     public string YRegister { get; set; }
00036     /// <summary>
00037     /// The Accumulator
00038     /// </summary>
00039     public string Accumulator { get; set; }
00040     /// <summary>
00041     /// The Stack Pointer
00042     /// </summary>
00043     public string StackPointer { get; set; }
00044     /// <summary>
00045     /// The number of cycles executed since the last load or reset
00046     /// </summary>
00047     public int NumberOfCycles { get; set; }
00048     }
00049 }

```

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.](#)

```
00001 namespace Emulator.Model
00002 {
00003     /// <summary>
00004     /// The Model used when Loading a Program.
00005     /// </summary>
00006     public class RomFileModel
00007     {
00008         /// <summary>
00009         /// The Program Converted into Hex.
00010         /// </summary>
00011         public byte[][] Rom { get; set; }
00012
00013         /// <summary>
00014         /// The path of the Program that was loaded.
00015         /// </summary>
00016         public byte RomBanks { get; set; }
00017
00018         /// <summary>
00019         /// The name of the Program that was loaded.
00020         /// </summary>
00021         public int RomBankSize { get; set; }
00022
00023         /// <summary>
00024         /// The name of the Program that was loaded.
00025         /// </summary>
00026         public string RomFileName { get; set; }
00027
00028         /// <summary>
00029         /// The path of the Program that was loaded.
00030         /// </summary>
00031         public string RomFilePath { get; set; }
00032     }
00033 }
```

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.](#)

```
00001 using System;
00002 using System.Xml.Serialization;
00003
00004 namespace Emulator.Model
00005 {
00006     /// <summary>
00007     /// Model that contains the required information needed to save the current settings to disk
00008     /// </summary>
00009     [Serializable]
00010     [XmlRootAttribute("SettingsFileModel", Namespace="Emulator.Model", IsNullable = false)]
00011     public class SettingsModel
00012     {
00013         /// <summary>
00014         /// The version of the file that is being saved
00015         /// </summary>
00016         public byte SettingsVersionMajor { get; set; }
00017     }
```

```

00018 /// <summary>
00019 /// The version of the file that is being saved
00020 /// </summary>
00021     public byte SettingsVersionMinor { get; set; }
00022
00023 /// <summary>
00024 /// The version of the file that is being saved
00025 /// </summary>
00026     public byte SettingsVersionBuild { get; set; }
00027
00028 /// <summary>
00029 /// The version of the file that is being saved
00030 /// </summary>
00031     public byte SettingsVersionRevision { get; set; }
00032
00033 /// <summary>
00034 /// The PC port that is being saved
00035 /// </summary>
00036     public string ComPortName { get; set; }
00037 }
00038 }

```

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.](#)

```

00001 using System;
00002 using System.Collections.Generic;
00003
00004 namespace Emulator.Model
00005 {
00006     /// <summary>
00007     /// Model that contains the required information needed to save the current state of the processor to
00008     /// disk
00009     /// </summary>
00010     [Serializable]
00011     public class StateFileModel
00012     {
00013         /// <summary>
00014         /// The Number of Cycles the Program has Ran so Far
00015         /// </summary>
00016         public int NumberOfCycles { get; set; }
00017
00018         /// <summary>
00019         /// The output of the program
00020         /// </summary>
00021         public IList<OutputLog> OutputLog { get; set; }
00022
00023         /// <summary>
00024         /// The Processor Object that is being saved
00025         /// </summary>
00026         public Hardware.W65C02 W65C02 { get; set; }
00027
00028         /// <summary>
00029         /// The first VIA Object that is being saved
00030         /// </summary>
00031         public Hardware.W65C22 W65C22 { get; set; }
00032
00033         /// <summary>
00034         /// The second VIA Object that is being saved

```

```

00034 /// </summary>
00035     public Hardware.W65C22 MM65SIB { get; set; }
00036
00037 /// <summary>
00038 /// The ACIA Object that is being saved
00039 /// </summary>
00040     public Hardware.W65C51 W65C51 { get; set; }
00041
00042 /// <summary>
00043 /// The Shared ROM Object that is being saved
00044 /// </summary>
00045     public Hardware.AT28CXX AT28C010 { get; set; }
00046
00047 /// <summary>
00048 /// The Banked ROM Object that is being saved
00049 /// </summary>
00050     public Hardware.AT28CXX AT28C64 { get; set; }
00051     }
00052 }

```

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.](#)

```

00001 using System;
00002 using System.Collections.Generic;
00003 using System.Collections.ObjectModel;
00004 using System.Collections.Specialized;
00005 using System.Windows.Threading;
00006
00007 namespace Emulator
00008 {
00009     /// <summary>
00010     /// A MultiThreaedObservableCollection.
00011     /// This allows multiple threads to access the same observable collection in a safe manner.
00012     /// </summary>
00013     /// <typeparam name="T"></typeparam>
00014     public class MultiThreadedObservableCollection<T> : ObservableCollection<T>
00015     {
00016         /// <summary>
00017         /// Instantiates a new instance of the MultiThreadedObservableCollection
00018         /// </summary>
00019         public MultiThreadedObservableCollection()
00020         {
00021         }
00022     }
00023
00024     /// <summary>
00025     /// Instantiates a new instance of the MultiThreadedObservableCollection
00026     /// </summary>
00027     /// <param name="collection">The initial collection to be loaded</param>
00028     public MultiThreadedObservableCollection(IEnumerable<T> collection)
00029         : base(collection)
00030     {
00031     }
00032 }
00033
00034 /// <summary>
00035 /// Instantiates a new instance of the MultiThreadedObservableCollection
00036 /// </summary>

```

```

00037 /// <param name="list">The initial list to be loaded</param>
00038     public MultiThreadedObservableCollection(List<T> list)
00039     {
00040         : base(list)
00041     }
00042 }
00043
00044 /// <summary>
00045 /// The NotifyCollectionChangedEventHandler, Sends a notification anytime the collection has been
00046 modified.
00047 /// </summary>
00048     public override event NotifyCollectionChangedEventHandler CollectionChanged;
00049
00050 /// <summary>
00051 /// The NotifyCollectionChangedEventHandler, Notifies the listeners in a thread safe manner
00052 /// </summary>
00053     protected override void OnCollectionChanged(NotifyCollectionChangedEventArgs e)
00054     {
00055         var collectionChanged = CollectionChanged;
00056         if (collectionChanged != null)
00057             foreach (NotifyCollectionChangedEventHandler nh in
00058                 collectionChanged.GetInvocationList())
00059             {
00060                 var dispObj = nh.Target as DispatcherObject;
00061                 if (dispObj != null)
00062                 {
00063                     var dispatcher = dispObj.Dispatcher;
00064                     if (dispatcher != null && !dispatcher.CheckAccess())
00065                     {
00066                         var nh1 = nh;
00067                         dispatcher.BeginInvoke(
00068                             (Action)() => nh1.Invoke(this,
00069                                 new
00070                                     NotifyCollectionChangedEventArgs(NotifyCollectionChangedAction.Reset)),
00071                                 DispatcherPriority.DataBind);
00072                         continue;
00073                     }
00074                 }
00075                 nh.Invoke(this, e);
00076             }
00077 }

```

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.](#)

```

00001 // <autogenerated />
00002 using System;
00003 using System.Reflection;
00004 [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.](#)

```

00001 // <autogenerated />
00002 using System;
00003 using System.Reflection;
00004 [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.

```
00001 #pragma checksum "..\..\..\App.xaml" "{8829d00f-11b8-4213-878b-770e8597ac16}"
      "3C3B83350F313F767CDD9CA458D577D426BB4EF0F6F94CE9866749BCB08F1D0F"
00002 //-----
00003 // <auto-generated>
00004 //     This code was generated by a tool.
00005 //     Runtime Version:4.0.30319.42000
00006 //
00007 //     Changes to this file may cause incorrect behavior and will be lost if
00008 //     the code is regenerated.
00009 // </auto-generated>
00010 //-----
00011
00012 using Emulator.ViewModel;
00013 using System;
00014 using System.Diagnostics;
00015 using System.Windows;
00016 using System.Windows.Automation;
00017 using System.Windows.Controls;
00018 using System.Windows.Controls.Primitives;
00019 using System.Windows.Data;
00020 using System.Windows.Documents;
00021 using System.Windows.Ink;
00022 using System.Windows.Input;
00023 using System.Windows.Markup;
00024 using System.Windows.Media;
00025 using System.Windows.Media.Animation;
00026 using System.Windows.Media.Effects;
00027 using System.Windows.Media.Imaging;
00028 using System.Windows.Media.Media3D;
00029 using System.Windows.Media.TextFormatting;
00030 using System.Windows.Navigation;
00031 using System.Windows.Shapes;
00032 using System.Windows.Shell;
00033
00034
00035 namespace XamlGeneratedNamespace {
00036
00037
00038     /// <summary>
00039     /// GeneratedApplication
00040     /// </summary>
00041     public partial class GeneratedApplication : System.Windows.Application {
00042
00043         private bool _contentLoaded;
00044
00045         /// <summary>
00046         /// InitializeComponent
00047         /// </summary>
00048         [System.Diagnostics.DebuggerNonUserCodeAttribute()]
00049         [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
00050         public void InitializeComponent() {
00051             if (_contentLoaded) {
00052                 return;
00053             }
00054             _contentLoaded = true;
00055
00056 #line 2 "..\..\..\App.xaml"
00057             this.StartupUri = new System.Uri("MainWindow.xaml", System.UriKind.Relative);
00058
00059 #line default
00060 #line hidden
```

```

00061         System.Uri resourceLocator = new System.Uri("/Emulator;component/app.xaml",
System.UriKind.Relative);
00062
00063 #line 1 "..\..\..\App.xaml"
00064         System.Windows.Application.LoadComponent(this, resourceLocator);
00065
00066 #line default
00067 #line hidden
00068     }
00069
00070 /// <summary>
00071 /// Application Entry Point.
00072 /// </summary>
00073 [System.STAThreadAttribute()]
00074 [System.Diagnostics.DebuggerNonUserCodeAttribute()]
00075 [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
00076 public static void Main() {
00077     SplashScreen splashScreen = new SplashScreen("splashscreen.png");
00078     splashScreen.Show(true);
00079     XamlGeneratedNamespace.GeneratedApplication app = new
XamlGeneratedNamespace.GeneratedApplication();
00080     app.InitializeComponent();
00081     app.Run();
00082 }
00083 }
00084 }
00085

```

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.](#)

```

00001 #pragma checksum "..\..\..\App.xaml" "{8829d00f-11b8-4213-878b-770e8597ac16}"
"3C3B83350F313F767CDD9CA458D577D426BB4EF0F6F94CE9866749BCB08F1D0F"
00002 //-----
00003 // <auto-generated>
00004 //     This code was generated by a tool.
00005 //     Runtime Version:4.0.30319.42000
00006 //
00007 //     Changes to this file may cause incorrect behavior and will be lost if
00008 //     the code is regenerated.
00009 // </auto-generated>
00010 //-----
00011
00012 using Emulator.ViewModel;
00013 using System;
00014 using System.Diagnostics;
00015 using System.Windows;
00016 using System.Windows.Automation;
00017 using System.Windows.Controls;
00018 using System.Windows.Controls.Primitives;
00019 using System.Windows.Data;
00020 using System.Windows.Documents;
00021 using System.Windows.Ink;
00022 using System.Windows.Input;
00023 using System.Windows.Markup;
00024 using System.Windows.Media;
00025 using System.Windows.Media.Animation;
00026 using System.Windows.Media.Effects;
00027 using System.Windows.Media.Imaging;
00028 using System.Windows.Media.Media3D;
00029 using System.Windows.Media.TextFormatting;

```

```

00030 using System.Windows.Navigation;
00031 using System.Windows.Shapes;
00032 using System.Windows.Shell;
00033
00034
00035 namespace XamlGeneratedNamespace {
00036
00037
00038     /// <summary>
00039     /// GeneratedApplication
00040     /// </summary>
00041     public partial class GeneratedApplication : System.Windows.Application {
00042
00043         private bool _contentLoaded;
00044
00045         /// <summary>
00046         /// InitializeComponent
00047         /// </summary>
00048         [System.Diagnostics.DebuggerNonUserCodeAttribute()]
00049         [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
00050         public void InitializeComponent() {
00051             if (_contentLoaded) {
00052                 return;
00053             }
00054             _contentLoaded = true;
00055
00056             #line 2 "..\..\..\App.xaml"
00057             this.StartupUri = new System.Uri("MainWindow.xaml", System.UriKind.Relative);
00058
00059             #line default
00060             #line hidden
00061             System.Uri resourceLocator = new System.Uri("/Emulator;component/app.xaml",
00062                 System.UriKind.Relative);
00063
00064             #line 1 "..\..\..\App.xaml"
00065             System.Windows.Application.LoadComponent(this, resourceLocator);
00066
00067             #line default
00068             #line hidden
00069         }
00070
00071         /// <summary>
00072         /// Application Entry Point.
00073         /// </summary>
00074         [System.STAThreadAttribute()]
00075         [System.Diagnostics.DebuggerNonUserCodeAttribute()]
00076         [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
00077         public static void Main() {
00078             SplashScreen splashScreen = new SplashScreen("splashscreen.png");
00079             splashScreen.Show(true);
00080             XamlGeneratedNamespace.GeneratedApplication app = new
00081             XamlGeneratedNamespace.GeneratedApplication();
00082             app.InitializeComponent();
00083             app.Run();
00084         }
00085     }

```

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.](#)

```

00001 //-----
00002 // <auto-generated>
00003 //     This code was generated by a tool.
00004 //     Runtime Version:4.0.30319.42000
00005 //
00006 //     Changes to this file may cause incorrect behavior and will be lost if
00007 //     the code is regenerated.
00008 // </auto-generated>
00009 //-----
00010
00011 [assembly: System.Windows.Resources.AssemblyAssociatedContentFileAttribute("nlog.config")]
00012
00013

```

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.](#)

```
00001 //-----
00002 // <auto-generated>
00003 //      This code was generated by a tool.
00004 //      Runtime Version:4.0.30319.42000
00005 //
00006 //      Changes to this file may cause incorrect behavior and will be lost if
00007 //      the code is regenerated.
00008 // </auto-generated>
00009 //-----
00010
00011 [assembly: System.Windows.Resources.AssemblyAssociatedContentFileAttribute("nlog.config")]
00012
00013
```

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.](#)

```
00001 //-----
00002 // <auto-generated>
00003 //      This code was generated by a tool.
00004 //      Runtime Version:4.0.30319.42000
00005 //
00006 //      Changes to this file may cause incorrect behavior and will be lost if
00007 //      the code is regenerated.
00008 // </auto-generated>
00009 //-----
00010
00011 namespace XamlGeneratedNamespace {
00012
00013
00014     /// <summary>
00015     /// GeneratedInternalTypeHelper
00016     /// </summary>
00017     [System.Diagnostics.DebuggerNonUserCodeAttribute()]
00018     [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
00019     [System.ComponentModel.EditorBrowsableAttribute(System.ComponentModel.EditorBrowsableState.Never)]
00020     public sealed class GeneratedInternalTypeHelper : System.Windows.Markup.InternalTypeHelper {
00021
00022     /// <summary>
00023     /// CreateInstance
00024     /// </summary>
00025     protected override object CreateInstance(System.Type type, System.Globalization.CultureInfo
culture) {
00026         return System.Activator.CreateInstance(type, ((System.Reflection.BindingFlags.Public |
System.Reflection.BindingFlags.NonPublic)
00027         | (System.Reflection.BindingFlags.Instance |
System.Reflection.BindingFlags.CreateInstance)), null, null, culture);
00028     }
00029
00030     /// <summary>
00031     /// GetPropertyValue
00032     /// </summary>
00033
```



```

00033         protected override object GetPropertyValue(System.Reflection.PropertyInfo propertyInfo, object
target, System.Globalization.CultureInfo culture) {
00034             return propertyInfo.GetValue(target, System.Reflection.BindingFlags.Default, null, null,
culture);
00035         }
00036
00037     /// <summary>
00038     /// SetPropertyValue
00039     /// </summary>
00040     protected override void SetPropertyValue(System.Reflection.PropertyInfo propertyInfo, object
target, object value, System.Globalization.CultureInfo culture) {
00041         propertyInfo.SetValue(target, value, System.Reflection.BindingFlags.Default, null, null,
culture);
00042     }
00043
00044     /// <summary>
00045     /// CreateDelegate
00046     /// </summary>
00047     protected override System.Delegate CreateDelegate(System.Type delegateType, object target,
string handler) {
00048         return ((System.Delegate) (target.GetType().InvokeMember("_CreateDelegate",
(System.Reflection.BindingFlags.InvokeMethod
| (System.Reflection.BindingFlags.NonPublic |
System.Reflection.BindingFlags.Instance)), null, target, new object[] {
00050             delegateType,
00051             handler}, null)));
00052     }
00053
00054     /// <summary>
00055     /// AddEventHandler
00056     /// </summary>
00057     protected override void AddEventHandler(System.Reflection.EventInfo eventInfo, object target,
System.Delegate handler) {
00058         eventInfo.AddEventHandler(target, handler);
00059     }
00060 }
00061 }
00062

```

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.](#)

```

00001 //-----
00002 // <auto-generated>
00003 //     This code was generated by a tool.
00004 //     Runtime Version:4.0.30319.42000
00005 //
00006 //     Changes to this file may cause incorrect behavior and will be lost if
00007 //     the code is regenerated.
00008 // </auto-generated>
00009 //-----
00010
00011 namespace XamlGeneratedNamespace {
00012
00013
00014     /// <summary>
00015     /// GeneratedInternalTypeHelper
00016     /// </summary>
00017     [System.Diagnostics.DebuggerNonUserCodeAttribute()]
00018     [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
00019     [System.ComponentModel.EditorBrowsableAttribute(System.ComponentModel.EditorBrowsableState.Never)]

```

```

00020     public sealed class GeneratedInternalTypeHelper : System.Windows.Markup.InternalTypeHelper {
00021
00022     /// <summary>
00023     /// CreateInstance
00024     /// </summary>
00025     protected override object CreateInstance(System.Type type, System.Globalization.CultureInfo
culture) {
00026         return System.Activator.CreateInstance(type, ((System.Reflection.BindingFlags.Public |
System.Reflection.BindingFlags.NonPublic)
00027         | (System.Reflection.BindingFlags.Instance |
System.Reflection.BindingFlags.CreateInstance)), null, null, culture);
00028     }
00029
00030     /// <summary>
00031     /// GetPropertyValue
00032     /// </summary>
00033     protected override object GetPropertyValue(System.Reflection.PropertyInfo propertyInfo, object
target, System.Globalization.CultureInfo culture) {
00034         return propertyInfo.GetValue(target, System.Reflection.BindingFlags.Default, null, null,
culture);
00035     }
00036
00037     /// <summary>
00038     /// SetPropertyValue
00039     /// </summary>
00040     protected override void SetPropertyValue(System.Reflection.PropertyInfo propertyInfo, object
target, object value, System.Globalization.CultureInfo culture) {
00041         propertyInfo.SetValue(target, value, System.Reflection.BindingFlags.Default, null, null,
culture);
00042     }
00043
00044     /// <summary>
00045     /// CreateDelegate
00046     /// </summary>
00047     protected override System.Delegate CreateDelegate(System.Type delegateType, object target,
string handler) {
00048         return ((System.Delegate) (target.GetType().InvokeMember("_CreateDelegate",
(System.Reflection.BindingFlags.InvokeMethod
00049         | (System.Reflection.BindingFlags.NonPublic |
System.Reflection.BindingFlags.Instance)), null, target, new object[] {
00050             delegateType,
00051             handler}, null)));
00052     }
00053
00054     /// <summary>
00055     /// AddEventHandler
00056     /// </summary>
00057     protected override void AddEventHandler(System.Reflection.EventInfo eventInfo, object target,
System.Delegate handler) {
00058         eventInfo.AddEventHandler(target, handler);
00059     }
00060 }
00061 }
00062

```

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.](#)

```

00001 #pragma checksum "..\..\..\MainWindows.xaml" "{8829d00f-11b8-4213-878b-770e8597ac16}"
        "661FDF568BC60BE24BAB53613C08D1DB3A4A19717F25E8058F980E67E6302D8D"
00002 //-----

```

```

00003 // <auto-generated>
00004 //     This code was generated by a tool.
00005 //     Runtime Version:4.0.30319.42000
00006 //
00007 //     Changes to this file may cause incorrect behavior and will be lost if
00008 //     the code is regenerated.
00009 // </auto-generated>
00010 //-----
00011
00012 using System;
00013 using System.Diagnostics;
00014 using System.Windows;
00015 using System.Windows.Automation;
00016 using System.Windows.Controls;
00017 using System.Windows.Controls.Primitives;
00018 using System.Windows.Data;
00019 using System.Windows.Documents;
00020 using System.Windows.Ink;
00021 using System.Windows.Input;
00022 using System.Windows.Markup;
00023 using System.Windows.Media;
00024 using System.Windows.Media.Animation;
00025 using System.Windows.Media.Effects;
00026 using System.Windows.Media.Imaging;
00027 using System.Windows.Media.Media3D;
00028 using System.Windows.Media.TextFormatting;
00029 using System.Windows.Navigation;
00030 using System.Windows.Shapes;
00031 using System.Windows.Shell;
00032
00033
00034 namespace Emulator {
00035
00036
00037     /// <summary>
00038     ///     MainWindow
00039     /// </summary>
00040     public partial class MainWindow : System.Windows.Window,
00041         System.Windows.Markup.IComponentConnector {
00042
00043         #line 2 "..\..\..\MainWindow.xaml"
00044         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00045             "CA1823:AvoidUnusedPrivateFields")]
00046         internal Emulator.MainWindow EmulatorWindow;
00047
00048         #line default
00049         #line hidden
00050
00051         #line 89 "..\..\..\MainWindow.xaml"
00052         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00053             "CA1823:AvoidUnusedPrivateFields")]
00054         internal System.Windows.Controls.DataGrid OutputLog;
00055
00056         #line default
00057         #line hidden
00058
00059         #line 106 "..\..\..\MainWindow.xaml"
00060         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00061             "CA1823:AvoidUnusedPrivateFields")]
00062         internal System.Windows.Controls.Button Run;
00063
00064         #line default
00065         #line hidden
00066
00067         #line 107 "..\..\..\MainWindow.xaml"
00068         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00069             "CA1823:AvoidUnusedPrivateFields")]
00070         internal System.Windows.Controls.Button Step;
00071
00072         #line default
00073         #line hidden
00074
00075         #line 108 "..\..\..\MainWindow.xaml"
00076         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00077             "CA1823:AvoidUnusedPrivateFields")]
00078         internal System.Windows.Controls.Button Reset;
00079
00080         #line default
00081         #line hidden
00082
00083         #line 110 "..\..\..\MainWindow.xaml"

```

```
00084         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00085         "CA1823:AvoidUnusedPrivateFields")]
00086         internal System.Windows.Controls.TextBlock RomFileNameText;
00087 #line default
00088 #line hidden
00089
00090
00091 #line 111 "..\..\..\MainWindows\MainWindow.xaml"
00092         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00093         "CA1823:AvoidUnusedPrivateFields")]
00094         internal System.Windows.Controls.TextBlock ComPortNameText;
00095 #line default
00096 #line hidden
00097
00098
00099 #line 112 "..\..\..\MainWindows\MainWindow.xaml"
00100         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00101         "CA1823:AvoidUnusedPrivateFields")]
00102         internal System.Windows.Controls.DataGrid Breakpoints;
00103 #line default
00104 #line hidden
00105
00106
00107 #line 137 "..\..\..\MainWindows\MainWindow.xaml"
00108         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00109         "CA1823:AvoidUnusedPrivateFields")]
00110         internal System.Windows.Controls.TextBox YRegister;
00111 #line default
00112 #line hidden
00113
00114
00115 #line 138 "..\..\..\MainWindows\MainWindow.xaml"
00116         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00117         "CA1823:AvoidUnusedPrivateFields")]
00118         internal System.Windows.Controls.TextBox XRegister;
00119 #line default
00120 #line hidden
00121
00122
00123 #line 139 "..\..\..\MainWindows\MainWindow.xaml"
00124         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00125         "CA1823:AvoidUnusedPrivateFields")]
00126         internal System.Windows.Controls.TextBox Accumulator;
00127 #line default
00128 #line hidden
00129
00130
00131 #line 140 "..\..\..\MainWindows\MainWindow.xaml"
00132         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00133         "CA1823:AvoidUnusedPrivateFields")]
00134         internal System.Windows.Controls.TextBox StackPointer;
00135 #line default
00136 #line hidden
00137
00138
00139 #line 141 "..\..\..\MainWindows\MainWindow.xaml"
00140         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00141         "CA1823:AvoidUnusedPrivateFields")]
00142         internal System.Windows.Controls.TextBox ProgramCounter;
00143 #line default
00144 #line hidden
00145
00146
00147 #line 142 "..\..\..\MainWindows\MainWindow.xaml"
00148         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00149         "CA1823:AvoidUnusedPrivateFields")]
00150         internal System.Windows.Controls.TextBox Dissassembly;
00151 #line default
00152 #line hidden
00153
00154
00155 #line 143 "..\..\..\MainWindows\MainWindow.xaml"
00156         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00157         "CA1823:AvoidUnusedPrivateFields")]
00158         internal System.Windows.Controls.TextBox CycleCount;
00159 #line default
00160 #line hidden
```

```
00161
00162
00163 #line 144 "..\..\..\MainWindow.xaml"
00164     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00165         "CA1823:AvoidUnusedPrivateFields")]
00166         internal System.Windows.Controls.TextBlock XRegisterText;
00166
00167 #line default
00168 #line hidden
00169
00170
00171 #line 145 "..\..\..\MainWindow.xaml"
00172     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00173         "CA1823:AvoidUnusedPrivateFields")]
00174         internal System.Windows.Controls.TextBlock YRegisterText;
00174
00175 #line default
00176 #line hidden
00177
00178
00179 #line 146 "..\..\..\MainWindow.xaml"
00180     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00181         "CA1823:AvoidUnusedPrivateFields")]
00182         internal System.Windows.Controls.TextBlock StackPointerRegisterText;
00182
00183 #line default
00184 #line hidden
00185
00186
00187 #line 147 "..\..\..\MainWindow.xaml"
00188     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00189         "CA1823:AvoidUnusedPrivateFields")]
00190         internal System.Windows.Controls.TextBlock AText;
00190
00191 #line default
00192 #line hidden
00193
00194
00195 #line 148 "..\..\..\MainWindow.xaml"
00196     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00197         "CA1823:AvoidUnusedPrivateFields")]
00198         internal System.Windows.Controls.TextBlock CurrentInstructionText;
00198
00199 #line default
00200 #line hidden
00201
00202
00203 #line 149 "..\..\..\MainWindow.xaml"
00204     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00205         "CA1823:AvoidUnusedPrivateFields")]
00206         internal System.Windows.Controls.TextBlock ProgramCounterText;
00206
00207 #line default
00208 #line hidden
00209
00210
00211 #line 150 "..\..\..\MainWindow.xaml"
00212     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00213         "CA1823:AvoidUnusedPrivateFields")]
00214         internal System.Windows.Controls.TextBlock CycleCountText;
00214
00215 #line default
00216 #line hidden
00217
00218
00219 #line 151 "..\..\..\MainWindow.xaml"
00220     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00221         "CA1823:AvoidUnusedPrivateFields")]
00222         internal System.Windows.Controls.CheckBox CarryFlag;
00222
00223 #line default
00224 #line hidden
00225
00226
00227 #line 152 "..\..\..\MainWindow.xaml"
00228     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00229         "CA1823:AvoidUnusedPrivateFields")]
00230         internal System.Windows.Controls.TextBlock CarryFlagText;
00230
00231 #line default
00232 #line hidden
00233
00234
00235 #line 153 "..\..\..\MainWindow.xaml"
00236     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00237         "CA1823:AvoidUnusedPrivateFields")]
00237         internal System.Windows.Controls.CheckBox ZeroFlag;
```

```
00238
00239 #line default
00240 #line hidden
00241
00242
00243 #line 154 "..\..\..\MainWindow.xaml"
00244 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00245 "CA1823:AvoidUnusedPrivateFields")]
00246     internal System.Windows.Controls.TextBlock ZeroFlagText;
00247 #line default
00248 #line hidden
00249
00250
00251 #line 155 "..\..\..\MainWindow.xaml"
00252 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00253 "CA1823:AvoidUnusedPrivateFields")]
00254     internal System.Windows.Controls.CheckBox InterrupFlag;
00255 #line default
00256 #line hidden
00257
00258
00259 #line 156 "..\..\..\MainWindow.xaml"
00260 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00261 "CA1823:AvoidUnusedPrivateFields")]
00262     internal System.Windows.Controls.TextBlock InterruptFlagText;
00263 #line default
00264 #line hidden
00265
00266
00267 #line 157 "..\..\..\MainWindow.xaml"
00268 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00269 "CA1823:AvoidUnusedPrivateFields")]
00270     internal System.Windows.Controls.CheckBox BcdFlag;
00271 #line default
00272 #line hidden
00273
00274
00275 #line 158 "..\..\..\MainWindow.xaml"
00276 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00277 "CA1823:AvoidUnusedPrivateFields")]
00278     internal System.Windows.Controls.TextBlock BcdFlagText;
00279 #line default
00280 #line hidden
00281
00282
00283 #line 159 "..\..\..\MainWindow.xaml"
00284 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00285 "CA1823:AvoidUnusedPrivateFields")]
00286     internal System.Windows.Controls.CheckBox BreakFlag;
00287 #line default
00288 #line hidden
00289
00290
00291 #line 160 "..\..\..\MainWindow.xaml"
00292 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00293 "CA1823:AvoidUnusedPrivateFields")]
00294     internal System.Windows.Controls.TextBlock BreakFlagText;
00295 #line default
00296 #line hidden
00297
00298
00299 #line 161 "..\..\..\MainWindow.xaml"
00300 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00301 "CA1823:AvoidUnusedPrivateFields")]
00302     internal System.Windows.Controls.CheckBox OverflowFlag;
00303 #line default
00304 #line hidden
00305
00306
00307 #line 162 "..\..\..\MainWindow.xaml"
00308 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00309 "CA1823:AvoidUnusedPrivateFields")]
00310     internal System.Windows.Controls.TextBlock OverflowFlagText;
00311 #line default
00312 #line hidden
00313
00314
00315 #line 163 "..\..\..\MainWindow.xaml"
```

```

00316         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00317         "CA1823:AvoidUnusedPrivateFields")]
00318         internal System.Windows.Controls.CheckBox NegativeFlag;
00319 #line default
00320 #line hidden
00321
00322
00323 #line 164 "..\..\..\MainWindow.xaml"
00324         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00325         "CA1823:AvoidUnusedPrivateFields")]
00326         internal System.Windows.Controls.TextBlock NegativeFlagText;
00327 #line default
00328 #line hidden
00329
00330
00331 #line 165 "..\..\..\MainWindow.xaml"
00332         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00333         "CA1823:AvoidUnusedPrivateFields")]
00334         internal System.Windows.Controls.Slider CpuSpeed;
00335 #line default
00336 #line hidden
00337
00338
00339 #line 166 "..\..\..\MainWindow.xaml"
00340         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00341         "CA1823:AvoidUnusedPrivateFields")]
00342         internal System.Windows.Controls.TextBlock SpeedText;
00343 #line default
00344 #line hidden
00345
00346         private bool _contentLoaded;
00347
00348         /// <summary>
00349         /// InitializeComponent
00350         /// </summary>
00351         [System.Diagnostics.DebuggerNonUserCodeAttribute()]
00352         [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
00353         public void InitializeComponent() {
00354             if (_contentLoaded) {
00355                 return;
00356             }
00357             _contentLoaded = true;
00358             System.Uri resourceLocater = new System.Uri("/Emulator;component/mainwindow.xaml",
00359             System.UriKind.Relative);
00360 #line 1 "..\..\..\MainWindow.xaml"
00361             System.Windows.Application.LoadComponent(this, resourceLocater);
00362 #line default
00363 #line hidden
00364         }
00365
00366         [System.Diagnostics.DebuggerNonUserCodeAttribute()]
00367         [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
00368
00369         [System.ComponentModel.EditorBrowsableAttribute(System.ComponentModel.EditorBrowsableState.Never)]
00370         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Design",
00371         "CA1033:InterfaceMethodsShouldBeCallableByChildTypes")]
00372         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Maintainability",
00373         "CA1502:AvoidExcessiveComplexity")]
00374         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00375         "CA1800:DoNotCastUnnecessarily")]
00376         void System.Windows.Markup.IComponentConnector.Connect(int connectionId, object target) {
00377             switch (connectionId)
00378             {
00379                 case 1:
00380                     this.EmulatorWindow = ((Emulator.MainWindow)(target));
00381                     return;
00382                 case 2:
00383                     ((System.Windows.Controls.MenuItem)(target)).Click += new
00384                     System.Windows.RoutedEventHandler(this.LoadFile);
00385 #line default
00386 #line hidden
00387                 case 3:
00388                     return;
00389 #line 73 "..\..\..\MainWindow.xaml"
00390                 ((System.Windows.Controls.MenuItem)(target)).Click += new
00391                 System.Windows.RoutedEventHandler(this.SaveFile);
00392

```

```
00392 #line default
00393 #line hidden
00394         return;
00395     case 4:
00396
00397 #line 74 "..\..\..\MainWindow.xaml"
00398         ((System.Windows.Controls.MenuItem)(target)).Click += new
            System.Windows.RoutedEventHandler(this.CloseFile);
00399
00400 #line default
00401 #line hidden
00402         return;
00403     case 5:
00404
00405 #line 76 "..\..\..\MainWindow.xaml"
00406         ((System.Windows.Controls.MenuItem)(target)).Click += new
            System.Windows.RoutedEventHandler(this.ToClose);
00407
00408 #line default
00409 #line hidden
00410         return;
00411     case 6:
00412         this.OutputLog = ((System.Windows.Controls.DataGrid)(target));
00413         return;
00414     case 7:
00415         this.Run = ((System.Windows.Controls.Button)(target));
00416         return;
00417     case 8:
00418         this.Step = ((System.Windows.Controls.Button)(target));
00419         return;
00420     case 9:
00421         this.Reset = ((System.Windows.Controls.Button)(target));
00422         return;
00423     case 10:
00424         this.RomFileNameText = ((System.Windows.Controls.TextBlock)(target));
00425         return;
00426     case 11:
00427         this.ComPortNameText = ((System.Windows.Controls.TextBlock)(target));
00428         return;
00429     case 12:
00430         this.Breakpoints = ((System.Windows.Controls.DataGrid)(target));
00431         return;
00432     case 13:
00433         this.YRegister = ((System.Windows.Controls.TextBox)(target));
00434         return;
00435     case 14:
00436         this.XRegister = ((System.Windows.Controls.TextBox)(target));
00437         return;
00438     case 15:
00439         this.Accumulator = ((System.Windows.Controls.TextBox)(target));
00440         return;
00441     case 16:
00442         this.StackPointer = ((System.Windows.Controls.TextBox)(target));
00443         return;
00444     case 17:
00445         this.ProgramCounter = ((System.Windows.Controls.TextBox)(target));
00446         return;
00447     case 18:
00448         this.Dissassembly = ((System.Windows.Controls.TextBox)(target));
00449         return;
00450     case 19:
00451         this.CycleCount = ((System.Windows.Controls.TextBox)(target));
00452         return;
00453     case 20:
00454         this.XRegisterText = ((System.Windows.Controls.TextBlock)(target));
00455         return;
00456     case 21:
00457         this.YRegisterText = ((System.Windows.Controls.TextBlock)(target));
00458         return;
00459     case 22:
00460         this.StackPointerRegisterText = ((System.Windows.Controls.TextBlock)(target));
00461         return;
00462     case 23:
00463         this.AText = ((System.Windows.Controls.TextBlock)(target));
00464         return;
00465     case 24:
00466         this.CurrentInstructionText = ((System.Windows.Controls.TextBlock)(target));
00467         return;
00468     case 25:
00469         this.ProgramCounterText = ((System.Windows.Controls.TextBlock)(target));
00470         return;
00471     case 26:
00472         this.CycleCountText = ((System.Windows.Controls.TextBlock)(target));
00473         return;
00474     case 27:
00475         this.CarryFlag = ((System.Windows.Controls.CheckBox)(target));
00476         return;
```



```

00477         case 28:
00478             this.CarryFlagText = ((System.Windows.Controls.TextBlock) (target));
00479             return;
00480         case 29:
00481             this.ZeroFlag = ((System.Windows.Controls.CheckBox) (target));
00482             return;
00483         case 30:
00484             this.ZeroFlagText = ((System.Windows.Controls.TextBlock) (target));
00485             return;
00486         case 31:
00487             this.InterruptFlag = ((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 }
00525 }
00526 }
00527

```

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.](#)

```

00001 #pragma checksum "..\..\..\MainWindows\MainWindow.xaml" "{8829d00f-11b8-4213-878b-770e8597ac16}"
00002 "661FDF568BC60BE24BAB53613C08D1DB3A4A19717F25E8058F980E67E6302D8D"
00002 //-----
00003 // <auto-generated>
00004 //     This code was generated by a tool.
00005 //     Runtime Version:4.0.30319.42000

```

```

00006 //
00007 //     Changes to this file may cause incorrect behavior and will be lost if
00008 //     the code is regenerated.
00009 // </auto-generated>
00010 //-----
00011
00012 using System;
00013 using System.Diagnostics;
00014 using System.Windows;
00015 using System.Windows.Automation;
00016 using System.Windows.Controls;
00017 using System.Windows.Controls.Primitives;
00018 using System.Windows.Data;
00019 using System.Windows.Documents;
00020 using System.Windows.Ink;
00021 using System.Windows.Input;
00022 using System.Windows.Markup;
00023 using System.Windows.Media;
00024 using System.Windows.Media.Animation;
00025 using System.Windows.Media.Effects;
00026 using System.Windows.Media.Imaging;
00027 using System.Windows.Media.Media3D;
00028 using System.Windows.Media.TextFormatting;
00029 using System.Windows.Navigation;
00030 using System.Windows.Shapes;
00031 using System.Windows.Shell;
00032
00033
00034 namespace Emulator {
00035
00036
00037     /// <summary>
00038     /// MainWindow
00039     /// </summary>
00040     public partial class MainWindow : System.Windows.Window,
        System.Windows.Markup.IComponentConnector {
00041
00042
00043         #line 2 "..\..\..\Main\MainWindow.xaml"
00044         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
            "CA1823:AvoidUnusedPrivateFields")]
00045         internal Emulator.MainWindow EmulatorWindow;
00046
00047         #line default
00048         #line hidden
00049
00050
00051         #line 89 "..\..\..\Main\MainWindow.xaml"
00052         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
            "CA1823:AvoidUnusedPrivateFields")]
00053         internal System.Windows.Controls.DataGrid OutputLog;
00054
00055         #line default
00056         #line hidden
00057
00058
00059         #line 106 "..\..\..\Main\MainWindow.xaml"
00060         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
            "CA1823:AvoidUnusedPrivateFields")]
00061         internal System.Windows.Controls.Button Run;
00062
00063         #line default
00064         #line hidden
00065
00066
00067         #line 107 "..\..\..\Main\MainWindow.xaml"
00068         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
            "CA1823:AvoidUnusedPrivateFields")]
00069         internal System.Windows.Controls.Button Step;
00070
00071         #line default
00072         #line hidden
00073
00074
00075         #line 108 "..\..\..\Main\MainWindow.xaml"
00076         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
            "CA1823:AvoidUnusedPrivateFields")]
00077         internal System.Windows.Controls.Button Reset;
00078
00079         #line default
00080         #line hidden
00081
00082
00083         #line 110 "..\..\..\Main\MainWindow.xaml"
00084         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
            "CA1823:AvoidUnusedPrivateFields")]
00085         internal System.Windows.Controls.TextBlock RomFileNameText;

```

```
00086
00087 #line default
00088 #line hidden
00089
00090
00091 #line 111 "..\..\..\MainWindow.xaml"
00092 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00093 "CA1823:AvoidUnusedPrivateFields")]
00093     internal System.Windows.Controls.TextBlock ComPortNameText;
00094
00095 #line default
00096 #line hidden
00097
00098
00099 #line 112 "..\..\..\MainWindow.xaml"
00100 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00101 "CA1823:AvoidUnusedPrivateFields")]
00101     internal System.Windows.Controls.DataGrid Breakpoints;
00102
00103 #line default
00104 #line hidden
00105
00106
00107 #line 137 "..\..\..\MainWindow.xaml"
00108 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00109 "CA1823:AvoidUnusedPrivateFields")]
00109     internal System.Windows.Controls.TextBox YRegister;
00110
00111 #line default
00112 #line hidden
00113
00114
00115 #line 138 "..\..\..\MainWindow.xaml"
00116 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00117 "CA1823:AvoidUnusedPrivateFields")]
00117     internal System.Windows.Controls.TextBox XRegister;
00118
00119 #line default
00120 #line hidden
00121
00122
00123 #line 139 "..\..\..\MainWindow.xaml"
00124 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00125 "CA1823:AvoidUnusedPrivateFields")]
00125     internal System.Windows.Controls.TextBox Accumulator;
00126
00127 #line default
00128 #line hidden
00129
00130
00131 #line 140 "..\..\..\MainWindow.xaml"
00132 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00133 "CA1823:AvoidUnusedPrivateFields")]
00133     internal System.Windows.Controls.TextBox StackPointer;
00134
00135 #line default
00136 #line hidden
00137
00138
00139 #line 141 "..\..\..\MainWindow.xaml"
00140 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00141 "CA1823:AvoidUnusedPrivateFields")]
00141     internal System.Windows.Controls.TextBox ProgramCounter;
00142
00143 #line default
00144 #line hidden
00145
00146
00147 #line 142 "..\..\..\MainWindow.xaml"
00148 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00149 "CA1823:AvoidUnusedPrivateFields")]
00149     internal System.Windows.Controls.TextBox Dissassembly;
00150
00151 #line default
00152 #line hidden
00153
00154
00155 #line 143 "..\..\..\MainWindow.xaml"
00156 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00157 "CA1823:AvoidUnusedPrivateFields")]
00157     internal System.Windows.Controls.TextBox CycleCount;
00158
00159 #line default
00160 #line hidden
00161
00162
00163 #line 144 "..\..\..\MainWindow.xaml"
```

```
00164         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00165         "CA1823:AvoidUnusedPrivateFields")]
00166         internal System.Windows.Controls.TextBlock XRegisterText;
00167 #line default
00168 #line hidden
00169
00170
00171 #line 145 "..\..\..\MainWindows.Xaml"
00172         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00173         "CA1823:AvoidUnusedPrivateFields")]
00174         internal System.Windows.Controls.TextBlock YRegisterText;
00175 #line default
00176 #line hidden
00177
00178
00179 #line 146 "..\..\..\MainWindows.Xaml"
00180         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00181         "CA1823:AvoidUnusedPrivateFields")]
00182         internal System.Windows.Controls.TextBlock StackPointerRegisterText;
00183 #line default
00184 #line hidden
00185
00186
00187 #line 147 "..\..\..\MainWindows.Xaml"
00188         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00189         "CA1823:AvoidUnusedPrivateFields")]
00190         internal System.Windows.Controls.TextBlock AText;
00191 #line default
00192 #line hidden
00193
00194
00195 #line 148 "..\..\..\MainWindows.Xaml"
00196         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00197         "CA1823:AvoidUnusedPrivateFields")]
00198         internal System.Windows.Controls.TextBlock CurrentInstructionText;
00199 #line default
00200 #line hidden
00201
00202
00203 #line 149 "..\..\..\MainWindows.Xaml"
00204         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00205         "CA1823:AvoidUnusedPrivateFields")]
00206         internal System.Windows.Controls.TextBlock ProgramCounterText;
00207 #line default
00208 #line hidden
00209
00210
00211 #line 150 "..\..\..\MainWindows.Xaml"
00212         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00213         "CA1823:AvoidUnusedPrivateFields")]
00214         internal System.Windows.Controls.TextBlock CycleCountText;
00215 #line default
00216 #line hidden
00217
00218
00219 #line 151 "..\..\..\MainWindows.Xaml"
00220         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00221         "CA1823:AvoidUnusedPrivateFields")]
00222         internal System.Windows.Controls.CheckBox CarryFlag;
00223 #line default
00224 #line hidden
00225
00226
00227 #line 152 "..\..\..\MainWindows.Xaml"
00228         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00229         "CA1823:AvoidUnusedPrivateFields")]
00230         internal System.Windows.Controls.TextBlock CarryFlagText;
00231 #line default
00232 #line hidden
00233
00234
00235 #line 153 "..\..\..\MainWindows.Xaml"
00236         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00237         "CA1823:AvoidUnusedPrivateFields")]
00238         internal System.Windows.Controls.CheckBox ZeroFlag;
00239 #line default
00240 #line hidden
```

```
00241
00242
00243 #line 154 "..\..\..\MainWindow.xaml"
00244     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00245         "CA1823:AvoidUnusedPrivateFields")]
00246         internal System.Windows.Controls.TextBlock ZeroFlagText;
00247 #line default
00248 #line hidden
00249
00250
00251 #line 155 "..\..\..\MainWindow.xaml"
00252     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00253         "CA1823:AvoidUnusedPrivateFields")]
00254         internal System.Windows.Controls.CheckBox InterrupFlag;
00255 #line default
00256 #line hidden
00257
00258
00259 #line 156 "..\..\..\MainWindow.xaml"
00260     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00261         "CA1823:AvoidUnusedPrivateFields")]
00262         internal System.Windows.Controls.TextBlock InterruptFlagText;
00263 #line default
00264 #line hidden
00265
00266
00267 #line 157 "..\..\..\MainWindow.xaml"
00268     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00269         "CA1823:AvoidUnusedPrivateFields")]
00270         internal System.Windows.Controls.CheckBox BcdFlag;
00271 #line default
00272 #line hidden
00273
00274
00275 #line 158 "..\..\..\MainWindow.xaml"
00276     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00277         "CA1823:AvoidUnusedPrivateFields")]
00278         internal System.Windows.Controls.TextBlock BcdFlagText;
00279 #line default
00280 #line hidden
00281
00282
00283 #line 159 "..\..\..\MainWindow.xaml"
00284     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00285         "CA1823:AvoidUnusedPrivateFields")]
00286         internal System.Windows.Controls.CheckBox BreakFlag;
00287 #line default
00288 #line hidden
00289
00290
00291 #line 160 "..\..\..\MainWindow.xaml"
00292     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00293         "CA1823:AvoidUnusedPrivateFields")]
00294         internal System.Windows.Controls.TextBlock BreakFlagText;
00295 #line default
00296 #line hidden
00297
00298
00299 #line 161 "..\..\..\MainWindow.xaml"
00300     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00301         "CA1823:AvoidUnusedPrivateFields")]
00302         internal System.Windows.Controls.CheckBox OverflowFlag;
00303 #line default
00304 #line hidden
00305
00306
00307 #line 162 "..\..\..\MainWindow.xaml"
00308     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00309         "CA1823:AvoidUnusedPrivateFields")]
00310         internal System.Windows.Controls.TextBlock OverflowFlagText;
00311 #line default
00312 #line hidden
00313
00314
00315 #line 163 "..\..\..\MainWindow.xaml"
00316     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00317         "CA1823:AvoidUnusedPrivateFields")]
00318         internal System.Windows.Controls.CheckBox NegativeFlag;
```

```

00318
00319 #line default
00320 #line hidden
00321
00322
00323 #line 164 "..\..\..\MainWindows.Xaml"
00324 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
    "CA1823:AvoidUnusedPrivateFields")]
00325     internal System.Windows.Controls.TextBlock NegativeFlagText;
00326
00327 #line default
00328 #line hidden
00329
00330
00331 #line 165 "..\..\..\MainWindows.Xaml"
00332 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
    "CA1823:AvoidUnusedPrivateFields")]
00333     internal System.Windows.Controls.Slider CpuSpeed;
00334
00335 #line default
00336 #line hidden
00337
00338
00339 #line 166 "..\..\..\MainWindows.Xaml"
00340 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
    "CA1823:AvoidUnusedPrivateFields")]
00341     internal System.Windows.Controls.TextBlock SpeedText;
00342
00343 #line default
00344 #line hidden
00345
00346     private bool _contentLoaded;
00347
00348 /// <summary>
00349 /// InitializeComponent
00350 /// </summary>
00351 [System.Diagnostics.DebuggerNonUserCodeAttribute()]
00352 [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
00353 public void InitializeComponent() {
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 "..\..\..\MainWindows.Xaml"
00361     System.Windows.Application.LoadComponent(this, resourceLocater);
00362
00363 #line default
00364 #line hidden
00365 }
00366
00367 [System.Diagnostics.DebuggerNonUserCodeAttribute()]
00368 [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
00369 [System.ComponentModel.EditorBrowsableAttribute(System.ComponentModel.EditorBrowsableState.Never)]
00370 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Design",
    "CA1033:InterfaceMethodsShouldBeCallableByChildTypes")]
00371 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Maintainability",
    "CA1502:AvoidExcessiveComplexity")]
00372 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
    "CA1800:DoNotCastUnnecessarily")]
00373 void System.Windows.Markup.IComponentConnector.Connect(int connectionId, object target) {
00374     switch (connectionId)
00375     {
00376     case 1:
00377         this.EmulatorWindow = ((Emulator.MainWindow)(target));
00378         return;
00379     case 2:
00380
00381 #line 72 "..\..\..\MainWindows.Xaml"
00382         ((System.Windows.Controls.MenuItem)(target)).Click += new
            System.Windows.RoutedEventHandler(this.LoadFile);
00383
00384 #line default
00385 #line hidden
00386         return;
00387     case 3:
00388
00389 #line 73 "..\..\..\MainWindows.Xaml"
00390         ((System.Windows.Controls.MenuItem)(target)).Click += new
            System.Windows.RoutedEventHandler(this.SaveFile);
00391
00392 #line default
00393 #line hidden
00394         return;

```

```
00395         case 4:
00396
00397 #line 74 "..\..\MainWindow.xaml"
00398     ((System.Windows.Controls.MenuItem)(target)).Click += new
        System.Windows.RoutedEventHandler(this.CloseFile);
00399
00400 #line default
00401 #line hidden
00402         return;
00403     case 5:
00404
00405 #line 76 "..\..\MainWindow.xaml"
00406     ((System.Windows.Controls.MenuItem)(target)).Click += new
        System.Windows.RoutedEventHandler(this.ToClose);
00407
00408 #line default
00409 #line hidden
00410         return;
00411     case 6:
00412         this.OutputLog = ((System.Windows.Controls.DataGrid)(target));
00413         return;
00414     case 7:
00415         this.Run = ((System.Windows.Controls.Button)(target));
00416         return;
00417     case 8:
00418         this.Step = ((System.Windows.Controls.Button)(target));
00419         return;
00420     case 9:
00421         this.Reset = ((System.Windows.Controls.Button)(target));
00422         return;
00423     case 10:
00424         this.RomFileNameText = ((System.Windows.Controls.TextBlock)(target));
00425         return;
00426     case 11:
00427         this.ComPortNameText = ((System.Windows.Controls.TextBlock)(target));
00428         return;
00429     case 12:
00430         this.Breakpoints = ((System.Windows.Controls.DataGrid)(target));
00431         return;
00432     case 13:
00433         this.YRegister = ((System.Windows.Controls.TextBox)(target));
00434         return;
00435     case 14:
00436         this.XRegister = ((System.Windows.Controls.TextBox)(target));
00437         return;
00438     case 15:
00439         this.Accumulator = ((System.Windows.Controls.TextBox)(target));
00440         return;
00441     case 16:
00442         this.StackPointer = ((System.Windows.Controls.TextBox)(target));
00443         return;
00444     case 17:
00445         this.ProgramCounter = ((System.Windows.Controls.TextBox)(target));
00446         return;
00447     case 18:
00448         this.Dissassembly = ((System.Windows.Controls.TextBox)(target));
00449         return;
00450     case 19:
00451         this.CycleCount = ((System.Windows.Controls.TextBox)(target));
00452         return;
00453     case 20:
00454         this.XRegisterText = ((System.Windows.Controls.TextBlock)(target));
00455         return;
00456     case 21:
00457         this.YRegisterText = ((System.Windows.Controls.TextBlock)(target));
00458         return;
00459     case 22:
00460         this.StackPointerRegisterText = ((System.Windows.Controls.TextBlock)(target));
00461         return;
00462     case 23:
00463         this.AText = ((System.Windows.Controls.TextBlock)(target));
00464         return;
00465     case 24:
00466         this.CurrentInstructionText = ((System.Windows.Controls.TextBlock)(target));
00467         return;
00468     case 25:
00469         this.ProgramCounterText = ((System.Windows.Controls.TextBlock)(target));
00470         return;
00471     case 26:
00472         this.CycleCountText = ((System.Windows.Controls.TextBlock)(target));
00473         return;
00474     case 27:
00475         this.CarryFlag = ((System.Windows.Controls.CheckBox)(target));
00476         return;
00477     case 28:
00478         this.CarryFlagText = ((System.Windows.Controls.TextBlock)(target));
00479         return;
```

```

00480         case 29:
00481             this.ZeroFlag = ((System.Windows.Controls.CheckBox) (target));
00482             return;
00483         case 30:
00484             this.ZeroFlagText = ((System.Windows.Controls.TextBlock) (target));
00485             return;
00486         case 31:
00487             this.InterruptFlag = ((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 }
00525 }
00526 }
00527

```

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.](#)

```

00001 #pragma checksum "..\..\..\SaveFile.xaml" "{8829d00f-11b8-4213-878b-770e8597ac16}"
00002 "34689CE75633CB3BE5E4FDF3C6E7ECDD6274F88E3F05662C41A2D31C677175A9"
00002 //-----
00003 // <auto-generated>
00004 //     This code was generated by a tool.
00005 //     Runtime Version:4.0.30319.42000
00006 //
00007 //     Changes to this file may cause incorrect behavior and will be lost if
00008 //     the code is regenerated.

```



```

00009 // </auto-generated>
00010 //-----
00011
00012 using System;
00013 using System.Diagnostics;
00014 using System.Windows;
00015 using System.Windows.Automation;
00016 using System.Windows.Controls;
00017 using System.Windows.Controls.Primitives;
00018 using System.Windows.Data;
00019 using System.Windows.Documents;
00020 using System.Windows.Ink;
00021 using System.Windows.Input;
00022 using System.Windows.Markup;
00023 using System.Windows.Media;
00024 using System.Windows.Media.Animation;
00025 using System.Windows.Media.Effects;
00026 using System.Windows.Media.Imaging;
00027 using System.Windows.Media.Media3D;
00028 using System.Windows.Media.TextFormatting;
00029 using System.Windows.Navigation;
00030 using System.Windows.Shapes;
00031 using System.Windows.Shell;
00032
00033
00034 namespace Emulator {
00035
00036
00037 /// <summary>
00038 /// SaveFile
00039 /// </summary>
00040 public partial class SaveFile : System.Windows.Window, System.Windows.Markup.IComponentConnector
00041 {
00042
00043 #line 7 "..\..\..\SaveFile.xaml"
00044 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00045 "CA1823:AvoidUnusedPrivateFields")]
00046 internal System.Windows.Controls.Button SelectFile;
00047 #line default
00048 #line hidden
00049
00050
00051 #line 8 "..\..\..\SaveFile.xaml"
00052 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00053 "CA1823:AvoidUnusedPrivateFields")]
00054 internal System.Windows.Controls.TextBox FilePath;
00055 #line default
00056 #line hidden
00057
00058
00059 #line 9 "..\..\..\SaveFile.xaml"
00060 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00061 "CA1823:AvoidUnusedPrivateFields")]
00062 internal System.Windows.Controls.TextBlock PathText;
00063 #line default
00064 #line hidden
00065
00066
00067 #line 10 "..\..\..\SaveFile.xaml"
00068 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00069 "CA1823:AvoidUnusedPrivateFields")]
00070 internal System.Windows.Controls.Button CancelButton;
00071 #line default
00072 #line hidden
00073
00074
00075 #line 11 "..\..\..\SaveFile.xaml"
00076 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00077 "CA1823:AvoidUnusedPrivateFields")]
00078 internal System.Windows.Controls.Button LoadButton;
00079 #line default
00080 #line hidden
00081
00082 private bool _contentLoaded;
00083
00084 /// <summary>
00085 /// InitializeComponent
00086 /// </summary>
00087 [System.Diagnostics.DebuggerNonUserCodeAttribute()]
00088 [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
00089 public void InitializeComponent() {

```

```

00090         if (_contentLoaded) {
00091             return;
00092         }
00093         _contentLoaded = true;
00094         System.Uri resourceLocator = new System.Uri("/Emulator;component/savefile.xaml",
System.UriKind.Relative);
00095
00096 #line 1 "..\..\..\SaveFile.xaml"
00097         System.Windows.Application.LoadComponent(this, resourceLocator);
00098
00099 #line default
00100 #line hidden
00101     }
00102
00103     [System.Diagnostics.DebuggerNonUserCodeAttribute()]
00104     [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
00105
00106     [System.ComponentModel.EditorBrowsableAttribute(System.ComponentModel.EditorBrowsableState.Never)]
00107     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Design",
"CA1033:InterfaceMethodsShouldBeCallableByChildTypes")]
00108     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Maintainability",
"CA1502:AvoidExcessiveComplexity")]
00109     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1800:DoNotCastUnnecessarily")]
00110     void System.Windows.Markup.IComponentConnector.Connect(int connectionId, object target) {
00111         switch (connectionId)
00112         {
00113             case 1:
00114                 this.SelectFile = ((System.Windows.Controls.Button) (target));
00115                 return;
00116             case 2:
00117                 this.FilePath = ((System.Windows.Controls.TextBox) (target));
00118                 return;
00119             case 3:
00120                 this.PathText = ((System.Windows.Controls.TextBlock) (target));
00121                 return;
00122             case 4:
00123                 this.CancelButton = ((System.Windows.Controls.Button) (target));
00124                 return;
00125             case 5:
00126                 this.LoadButton = ((System.Windows.Controls.Button) (target));
00127                 return;
00128             }
00129         this._contentLoaded = true;
00130     }
00131 }
00132

```

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.](#)

```

00001 #pragma checksum "..\..\..\SaveFile.xaml" "{8829d00f-11b8-4213-878b-770e8597ac16}"
"34689CE75633CB3BE5E4FDF3C6E7ECDD6274F88E3F05662C41A2D31C677175A9"
00002 //-----
00003 // <auto-generated>
00004 //     This code was generated by a tool.
00005 //     Runtime Version:4.0.30319.42000
00006 //
00007 //     Changes to this file may cause incorrect behavior and will be lost if
00008 //     the code is regenerated.

```

```

00009 // </auto-generated>
00010 //-----
00011
00012 using System;
00013 using System.Diagnostics;
00014 using System.Windows;
00015 using System.Windows.Automation;
00016 using System.Windows.Controls;
00017 using System.Windows.Controls.Primitives;
00018 using System.Windows.Data;
00019 using System.Windows.Documents;
00020 using System.Windows.Ink;
00021 using System.Windows.Input;
00022 using System.Windows.Markup;
00023 using System.Windows.Media;
00024 using System.Windows.Media.Animation;
00025 using System.Windows.Media.Effects;
00026 using System.Windows.Media.Imaging;
00027 using System.Windows.Media.Media3D;
00028 using System.Windows.Media.TextFormatting;
00029 using System.Windows.Navigation;
00030 using System.Windows.Shapes;
00031 using System.Windows.Shell;
00032
00033
00034 namespace Emulator {
00035
00036
00037 /// <summary>
00038 /// SaveFile
00039 /// </summary>
00040 public partial class SaveFile : System.Windows.Window, System.Windows.Markup.IComponentConnector
00041 {
00042
00043 #line 7 "..\..\..\SaveFile.xaml"
00044 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00045 "CA1823:AvoidUnusedPrivateFields")]
00046 internal System.Windows.Controls.Button SelectFile;
00047 #line default
00048 #line hidden
00049
00050
00051 #line 8 "..\..\..\SaveFile.xaml"
00052 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00053 "CA1823:AvoidUnusedPrivateFields")]
00054 internal System.Windows.Controls.TextBox FilePath;
00055 #line default
00056 #line hidden
00057
00058
00059 #line 9 "..\..\..\SaveFile.xaml"
00060 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00061 "CA1823:AvoidUnusedPrivateFields")]
00062 internal System.Windows.Controls.TextBlock PathText;
00063 #line default
00064 #line hidden
00065
00066
00067 #line 10 "..\..\..\SaveFile.xaml"
00068 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00069 "CA1823:AvoidUnusedPrivateFields")]
00070 internal System.Windows.Controls.Button CancelButton;
00071 #line default
00072 #line hidden
00073
00074
00075 #line 11 "..\..\..\SaveFile.xaml"
00076 [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00077 "CA1823:AvoidUnusedPrivateFields")]
00078 internal System.Windows.Controls.Button LoadButton;
00079 #line default
00080 #line hidden
00081
00082 private bool _contentLoaded;
00083
00084 /// <summary>
00085 /// InitializeComponent
00086 /// </summary>
00087 [System.Diagnostics.DebuggerNonUserCodeAttribute()]
00088 [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
00089 public void InitializeComponent() {

```

```

00090         if (_contentLoaded) {
00091             return;
00092         }
00093         _contentLoaded = true;
00094         System.Uri resourceLocator = new System.Uri("/Emulator;component/savefile.xaml",
System.UriKind.Relative);
00095
00096 #line 1 "..\\..\\..\\SaveFile.xaml"
00097         System.Windows.Application.LoadComponent(this, resourceLocator);
00098
00099 #line default
00100 #line hidden
00101     }
00102
00103     [System.Diagnostics.DebuggerNonUserCodeAttribute()]
00104     [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
00105
00106     [System.ComponentModel.EditorBrowsableAttribute(System.ComponentModel.EditorBrowsableState.Never)]
00107     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Design",
"CA1033:InterfaceMethodsShouldBeCallableByChildTypes")]
00108     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Maintainability",
"CA1502:AvoidExcessiveComplexity")]
00109     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1800:DoNotCastUnnecessarily")]
00110     void System.Windows.Markup.IComponentConnector.Connect(int connectionId, object target) {
00111         switch (connectionId)
00112         {
00113             case 1:
00114                 this.SelectFile = ((System.Windows.Controls.Button) (target));
00115                 return;
00116             case 2:
00117                 this.FilePath = ((System.Windows.Controls.TextBox) (target));
00118                 return;
00119             case 3:
00120                 this.PathText = ((System.Windows.Controls.TextBlock) (target));
00121                 return;
00122             case 4:
00123                 this.CancelButton = ((System.Windows.Controls.Button) (target));
00124                 return;
00125             case 5:
00126                 this.LoadButton = ((System.Windows.Controls.Button) (target));
00127                 return;
00128             }
00129         this._contentLoaded = true;
00130     }
00131 }
00132

```

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.](#)

```

00001 #pragma checksum "..\\..\\..\\Settings.xaml" "{8829d00f-11b8-4213-878b-770e8597ac16}"
"5C331E215A507ACA3F7FF07CFD574A81287117C06061A7F3A96858A63F0BA78B"
00002 //-----
00003 // <auto-generated>
00004 //     This code was generated by a tool.
00005 //     Runtime Version:4.0.30319.42000
00006 //
00007 //     Changes to this file may cause incorrect behavior and will be lost if
00008 //     the code is regenerated.

```

```

00009 // </auto-generated>
00010 //-----
00011
00012 using System;
00013 using System.Diagnostics;
00014 using System.Windows;
00015 using System.Windows.Automation;
00016 using System.Windows.Controls;
00017 using System.Windows.Controls.Primitives;
00018 using System.Windows.Data;
00019 using System.Windows.Documents;
00020 using System.Windows.Ink;
00021 using System.Windows.Input;
00022 using System.Windows.Markup;
00023 using System.Windows.Media;
00024 using System.Windows.Media.Animation;
00025 using System.Windows.Media.Effects;
00026 using System.Windows.Media.Imaging;
00027 using System.Windows.Media.Media3D;
00028 using System.Windows.Media.TextFormatting;
00029 using System.Windows.Navigation;
00030 using System.Windows.Shapes;
00031 using System.Windows.Shell;
00032
00033
00034 namespace Emulator {
00035
00036
00037 /// <summary>
00038 /// Settings
00039 /// </summary>
00040 public partial class Settings : System.Windows.Window, System.Windows.Markup.IComponentConnector
00041 {
00042
00043 #line 7 "..\..\..\Settings.xaml"
00044     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00045         "CA1823:AvoidUnusedPrivateFields")]
00046     internal System.Windows.Controls.ComboBox ComPortCombo;
00047
00048 #line default
00049 #line hidden
00050
00051 #line 8 "..\..\..\Settings.xaml"
00052     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00053         "CA1823:AvoidUnusedPrivateFields")]
00054     internal System.Windows.Controls.TextBlock PortText;
00055
00056 #line default
00057 #line hidden
00058
00059 #line 9 "..\..\..\Settings.xaml"
00060     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00061         "CA1823:AvoidUnusedPrivateFields")]
00062     internal System.Windows.Controls.Button ApplyButton;
00063
00064 #line default
00065 #line hidden
00066
00067 #line 10 "..\..\..\Settings.xaml"
00068     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00069         "CA1823:AvoidUnusedPrivateFields")]
00070     internal System.Windows.Controls.Button CloseButton;
00071
00072 #line default
00073 #line hidden
00074     private bool _contentLoaded;
00075
00076 /// <summary>
00077 /// InitializeComponent
00078 /// </summary>
00079     [System.Diagnostics.DebuggerNonUserCodeAttribute()]
00080     [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
00081     public void InitializeComponent() {
00082         if (_contentLoaded) {
00083             return;
00084         }
00085         _contentLoaded = true;
00086         System.Uri resourceLocater = new System.Uri("/Emulator;component/settings.xaml",
00087             System.UriKind.Relative);
00088
00089 #line 1 "..\..\..\Settings.xaml"
00090         System.Windows.Application.LoadComponent(this, resourceLocater);

```

```

00090
00091 #line default
00092 #line hidden
00093     }
00094
00095     [System.Diagnostics.DebuggerNonUserCodeAttribute()]
00096     [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
00097
00098     [System.ComponentModel.EditorBrowsableAttribute(System.ComponentModel.EditorBrowsableState.Never)]
00099     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Design",
00100 "CA1033:InterfaceMethodsShouldBeCallableByChildTypes")]
00101     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Maintainability",
00102 "CA1502:AvoidExcessiveComplexity")]
00103     [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00104 "CA1800:DoNotCastUnnecessarily")]
00105     void System.Windows.Markup.IComponentConnector.Connect(int connectionId, object target) {
00106         switch (connectionId)
00107         {
00108             case 1:
00109                 this.ComPortCombo = ((System.Windows.Controls.ComboBox)(target));
00110
00111 #line 7 "..\\..\\..\\Settings.xaml"
00112                 this.ComPortCombo.DropDownClosed += new
00113 System.EventHandler(this.PortSelectionDropDownClosed);
00114
00115 #line default
00116 #line hidden
00117                 return;
00118             case 2:
00119                 this.PortText = ((System.Windows.Controls.TextBlock)(target));
00120                 return;
00121             case 3:
00122                 this.ApplyButton = ((System.Windows.Controls.Button)(target));
00123                 return;
00124             case 4:
00125                 this.CloseButton = ((System.Windows.Controls.Button)(target));
00126                 return;
00127         }
00128         this._contentLoaded = true;
00129     }
00130 }
00131 }
00132 }
00133 }
00134 }
00135 }
00136 }
00137 }

```

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.](#)

```

00001 #pragma checksum "..\\..\\..\\Settings.xaml" "{8829d00f-11b8-4213-878b-770e8597ac16}"
00002 "5C331E215A507ACA3F7FF07CFD574A81287117C06061A7F3A96858A63F0BA78B"
00003 //-----
00004 // <auto-generated>
00005 //     This code was generated by a tool.
00006 //     Runtime Version:4.0.30319.42000
00007 //
00008 //     Changes to this file may cause incorrect behavior and will be lost if
00009 //     the code is regenerated.
00010 // </auto-generated>
00011 //-----
00012 using System;
00013 using System.Diagnostics;

```

```

00014 using System.Windows;
00015 using System.Windows.Automation;
00016 using System.Windows.Controls;
00017 using System.Windows.Controls.Primitives;
00018 using System.Windows.Data;
00019 using System.Windows.Documents;
00020 using System.Windows.Ink;
00021 using System.Windows.Input;
00022 using System.Windows.Markup;
00023 using System.Windows.Media;
00024 using System.Windows.Media.Animation;
00025 using System.Windows.Media.Effects;
00026 using System.Windows.Media.Imaging;
00027 using System.Windows.Media.Media3D;
00028 using System.Windows.Media.TextFormatting;
00029 using System.Windows.Navigation;
00030 using System.Windows.Shapes;
00031 using System.Windows.Shell;
00032
00033
00034 namespace Emulator {
00035
00036
00037     /// <summary>
00038     /// Settings
00039     /// </summary>
00040     public partial class Settings : System.Windows.Window, System.Windows.Markup.IComponentConnector
00041     {
00042
00043         #line 7 "..\..\..\Settings.xaml"
00044         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00045             "CA1823:AvoidUnusedPrivateFields")]
00046         internal System.Windows.Controls.ComboBox ComPortCombo;
00047
00048         #line default
00049         #line hidden
00050
00051         #line 8 "..\..\..\Settings.xaml"
00052         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00053             "CA1823:AvoidUnusedPrivateFields")]
00054         internal System.Windows.Controls.TextBlock PortText;
00055
00056         #line default
00057         #line hidden
00058
00059         #line 9 "..\..\..\Settings.xaml"
00060         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00061             "CA1823:AvoidUnusedPrivateFields")]
00062         internal System.Windows.Controls.Button ApplyButton;
00063
00064         #line default
00065         #line hidden
00066
00067         #line 10 "..\..\..\Settings.xaml"
00068         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00069             "CA1823:AvoidUnusedPrivateFields")]
00070         internal System.Windows.Controls.Button CloseButton;
00071
00072         #line default
00073         #line hidden
00074
00075         private bool _contentLoaded;
00076
00077         /// <summary>
00078         /// InitializeComponent
00079         /// </summary>
00080         [System.Diagnostics.DebuggerNonUserCodeAttribute()]
00081         [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
00082         public void InitializeComponent() {
00083             if (_contentLoaded) {
00084                 return;
00085             }
00086             _contentLoaded = true;
00087             System.Uri resourceLocater = new System.Uri("/Emulator;component/settings.xaml",
00088                 System.UriKind.Relative);
00089
00090         #line 11 "..\..\..\Settings.xaml"
00091         System.Windows.Application.LoadComponent(this, resourceLocater);
00092
00093         #line default
00094         #line hidden
00095     }
00096
00097 }
00098
00099
00100
00101
00102
00103
00104
00105
00106
00107
00108
00109
00110
00111
00112
00113
00114
00115
00116
00117
00118
00119
00120
00121
00122
00123
00124
00125
00126
00127
00128
00129
00130
00131
00132
00133
00134
00135
00136
00137
00138
00139
00140
00141
00142
00143
00144
00145
00146
00147
00148
00149
00150
00151
00152
00153
00154
00155
00156
00157
00158
00159
00160
00161
00162
00163
00164
00165
00166
00167
00168
00169
00170
00171
00172
00173
00174
00175
00176
00177
00178
00179
00180
00181
00182
00183
00184
00185
00186
00187
00188
00189
00190
00191
00192
00193
00194
00195
00196
00197
00198
00199
00200
00201
00202
00203
00204
00205
00206
00207
00208
00209
00210
00211
00212
00213
00214
00215
00216
00217
00218
00219
00220
00221
00222
00223
00224
00225
00226
00227
00228
00229
00230
00231
00232
00233
00234
00235
00236
00237
00238
00239
00240
00241
00242
00243
00244
00245
00246
00247
00248
00249
00250
00251
00252
00253
00254
00255
00256
00257
00258
00259
00260
00261
00262
00263
00264
00265
00266
00267
00268
00269
00270
00271
00272
00273
00274
00275
00276
00277
00278
00279
00280
00281
00282
00283
00284
00285
00286
00287
00288
00289
00290
00291
00292
00293
00294
00295
00296
00297
00298
00299
00300
00301
00302
00303
00304
00305
00306
00307
00308
00309
00310
00311
00312
00313
00314
00315
00316
00317
00318
00319
00320
00321
00322
00323
00324
00325
00326
00327
00328
00329
00330
00331
00332
00333
00334
00335
00336
00337
00338
00339
00340
00341
00342
00343
00344
00345
00346
00347
00348
00349
00350
00351
00352
00353
00354
00355
00356
00357
00358
00359
00360
00361
00362
00363
00364
00365
00366
00367
00368
00369
00370
00371
00372
00373
00374
00375
00376
00377
00378
00379
00380
00381
00382
00383
00384
00385
00386
00387
00388
00389
00390
00391
00392
00393
00394
00395
00396
00397
00398
00399
00400
00401
00402
00403
00404
00405
00406
00407
00408
00409
00410
00411
00412
00413
00414
00415
00416
00417
00418
00419
00420
00421
00422
00423
00424
00425
00426
00427
00428
00429
00430
00431
00432
00433
00434
00435
00436
00437
00438
00439
00440
00441
00442
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999

```

```

00095         [System.Diagnostics.DebuggerNonUserCodeAttribute()]
00096         [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
00097
00098         [System.ComponentModel.EditorBrowsableAttribute(System.ComponentModel.EditorBrowsableState.Never)]
00099         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Design",
00100             "CA1033:InterfaceMethodsShouldBeCallableByChildTypes")]
00101         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Maintainability",
00102             "CA1502:AvoidExcessiveComplexity")]
00103         [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
00104             "CA1800:DoNotCastUnnecessarily")]
00105         void System.Windows.Markup.IComponentConnector.Connect(int connectionId, object target) {
00106             switch (connectionId)
00107             {
00108                 case 1:
00109                     this.ComPortCombo = ((System.Windows.Controls.ComboBox) (target));
00110
00111                     #line 7 "..\..\..\Settings.xaml"
00112                     this.ComPortCombo.DropDownClosed += new
00113                         System.EventHandler(this.PortSelectionDropDownClosed);
00114
00115                     #line default
00116                     #line hidden
00117                     return;
00118                 case 2:
00119                     this.PortText = ((System.Windows.Controls.TextBlock) (target));
00120                     return;
00121                 case 3:
00122                     this.ApplyButton = ((System.Windows.Controls.Button) (target));
00123                     return;
00124                 case 4:
00125                     this.CloseButton = ((System.Windows.Controls.Button) (target));
00126                     return;
00127             }
00128             this._contentLoaded = true;
00129         }
00130     }
00131 }
00132
00133 }
00134
00135 }
00136
00137 }

```

7.63 Emulator/Properties/AssemblyInfo.cs File Reference

7.64 AssemblyInfo.cs

[Go to the documentation of this file.](#)

```

00001 using System.Resources;
00002 using System.Reflection;
00003 using System.Runtime.InteropServices;
00004 using System.Windows;
00005 using Emulator;
00006
00007 // General Information about an assembly is controlled through the following
00008 // set of attributes. Change these attribute values to modify the information
00009 // associated with an assembly.
00010 [assembly: AssemblyTitle(Versioning.Product.Title)]
00011 [assembly: AssemblyDescription(Versioning.Product.Description)]
00012 [assembly: AssemblyConfiguration("")]
00013 [assembly: AssemblyCompany(Versioning.Product.Company)]
00014 [assembly: AssemblyProduct(Versioning.Product.Name)]
00015 [assembly: AssemblyCopyright(Versioning.Product.Copyright)]
00016 [assembly: AssemblyTrademark("")]
00017 [assembly: AssemblyCulture("")]
00018
00019 // Setting ComVisible to false makes the types in this assembly not visible
00020 // to COM components. If you need to access a type in this assembly from
00021 // COM, set the ComVisible attribute to true on that type.
00022 [assembly: ComVisible(false)]
00023
00024 //In order to begin building localizable applications, set
00025 // <UICulture>CultureYouAreCodingWith</UICulture> in your .csproj file
00026 // inside a <PropertyGroup>. For example, if you are using US english
00027 // in your source files, set the <UICulture> to en-US. Then uncomment
00028 // the NeutralResourceLanguage attribute below. Update the "en-US" in
00029 // the line below to match the UICulture setting in the project file.
00030
00031 //[assembly: NeutralResourceLanguage("en-US", UltimateResourceFallbackLocation.Satellite)]
00032
00033
00034 [assembly: ThemeInfo(
00035     ResourceDictionaryLocation.None, //where theme specific resource dictionaries are located
00036     //(used if a resource is not found in the page,

```



```
00037 // or application resource dictionaries)
00038 ResourceDictionaryLocation.SourceAssembly //where the generic resource dictionary is located
00039 //(used if a resource is not found in the page,
00040 // app, or any theme specific resource dictionaries)
00041 }]
00042
00043
00044 // Version information for an assembly consists of the following four values:
00045 //
00046 //     Major Version
00047 //     Minor Version
00048 //     Build Number
00049 //     Revision
00050 //
00051 // You can specify all the values or you can default the Build and Revision Numbers
00052 // by using the '*' as shown below:
00053 // [assembly: AssemblyVersion("1.0.*")]
00054 [assembly: AssemblyVersion(Versioning.Product.VersionString)]
00055 [assembly: AssemblyFileVersion(Versioning.Product.VersionString)]
00056 [assembly: NeutralResourcesLanguage("en-GB")]
```

7.65 Hardware/Properties/AssemblyInfo.cs File Reference

7.66 AssemblyInfo.cs

[Go to the documentation of this file.](#)

```
00001 using System.Resources;
00002 using System.Reflection;
00003 using System.Runtime.InteropServices;
00004 using Hardware;
00005
00006 // General Information about an assembly is controlled through the following
00007 // set of attributes. Change these attribute values to modify the information
00008 // associated with an assembly.
00009 [assembly: AssemblyTitle(Versioning.Product.Title)]
00010 [assembly: AssemblyDescription(Versioning.Product.Description)]
00011 [assembly: AssemblyConfiguration("")]
00012 [assembly: AssemblyCompany(Versioning.Product.Company)]
00013 [assembly: AssemblyProduct("")]
00014 [assembly: AssemblyCopyright(Versioning.Product.Copyright)]
00015 [assembly: AssemblyTrademark("")]
00016 [assembly: AssemblyCulture("")]
00017
00018 // Setting ComVisible to false makes the types in this assembly not visible
00019 // to COM components. If you need to access a type in this assembly from
00020 // COM, set the ComVisible attribute to true on that type.
00021 [assembly: ComVisible(false)]
00022
00023 // The following GUID is for the ID of the typelib if this project is exposed to COM
00024 [assembly: Guid("f4afef76-2e8f-4497-86c6-c903aa70eebd")]
00025
00026 // Version information for an assembly consists of the following four values:
00027 //
00028 //     Major Version
00029 //     Minor Version
00030 //     Build Number
00031 //     Revision
00032 //
00033 // You can specify all the values or you can default the Build and Revision Numbers
00034 // by using the '*' as shown below:
00035 // [assembly: AssemblyVersion("1.0.*")]
00036 [assembly: AssemblyVersion(Versioning.Product.Version)]
00037 [assembly: AssemblyFileVersion(Versioning.Product.Version)]
00038 [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.](#)

```
00001 using GalaSoft.MvvmLight.Messaging;
00002
00003 namespace Emulator
00004 {
00005     /// <summary>
00006     /// Interaction logic for SaveState.xaml
00007     /// </summary>
00008     public partial class SaveFile
00009     {
00010         public SaveFile()
00011         {
00012             InitializeComponent();
00013             Messenger.Default.Register<NotificationMessage>(this, NotificationMessageReceived);
00014         }
00015
00016         private void NotificationMessageReceived(NotificationMessage notificationMessage)
00017         {
00018             if (notificationMessage.Notification == "CloseSaveFileWindow")
00019                 Close();
00020         }
00021     }
00022 }
```

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.](#)

```
00001 using GalaSoft.MvvmLight.Messaging;
00002 using Emulator.Model;
00003 using Emulator.ViewModel;
00004 using System;
00005 using System.Windows;
00006
00007 namespace Emulator
00008 {
00009     /// <summary>
00010     /// Interaction logic for Settings.xaml
00011     /// </summary>
00012     public partial class Settings
00013     {
00014         public Settings()
00015         {
00016             InitializeComponent();
00017             Messenger.Default.Register<NotificationMessage>(this, NotificationMessageReceived);
00018             Messenger.Default.Register<NotificationMessage<SettingsModel>>(this,
NotificationMessageReceived);
00019         }
00020
00021         private void NotificationMessageReceived(NotificationMessage notificationMessage)
```

```
00022         {
00023             if (notificationMessage.Notification == "CloseSettingsWindow")
00024             {
00025                 Close();
00026             }
00027         }
00028
00029         private void NotificationMessageReceived(NotificationMessage<SettingsModel>
notificationMessage)
00030         {
00031             if (notificationMessage.Notification == "SettingsWindow")
00032             {
00033                 SettingsViewModel.SettingsModel = notificationMessage.Content;
00034                 ComPortCombo.SelectedItem = notificationMessage.Content.ComPortName;
00035             }
00036         }
00037
00038         private void PortSelectionDropDownClosed(object sender, EventArgs e)
00039         {
00040             if (!(ComPortCombo.SelectedValue == null))
00041             {
00042                 string port = ComPortCombo.SelectedValue.ToString();
00043                 SettingsViewModel.ComPortSelection = port;
00044             }
00045         }
00046     }
00047 }
```

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; }

```

```

00065
00066 /// <summary>
00067 /// The AT28C010 ROM.
00068 /// </summary>
00069 public AT28CXX AT28C010 { get; private set; }
00070
00071 /// <summary>
00072 /// The Current Memory Page
00073 /// </summary>
00074 public MultiThreadedObservableCollection<MemoryRowModel> MemoryPage { get; set; }
00075
00076 /// <summary>
00077 /// The output log
00078 /// </summary>
00079 public MultiThreadedObservableCollection<OutputLog> OutputLog { get; private set; }
00080
00081 /// <summary>
00082 /// The Breakpoints
00083 /// </summary>
00084 public MultiThreadedObservableCollection<Breakpoint> Breakpoints { get; set; }
00085
00086 /// <summary>
00087 /// The Currently Selected Breakpoint
00088 /// </summary>
00089 public Breakpoint SelectedBreakpoint { get; set; }
00090
00091 /// <summary>
00092 /// The currently loaded binary file. (If it is indeed loaded, that is.)
00093 /// </summary>
00094 public RomFileModel RomFile { get; set; }
00095
00096 /// <summary>
00097 /// The Current Disassembly
00098 /// </summary>
00099 public string CurrentDisassembly
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 }
00113
00114 /// <summary>
00115 /// The number of cycles.
00116 /// </summary>
00117 public int NumberOfCycles { get; private set; }
00118
00119 /// <summary>
00120 /// The Memory Page number.
00121 /// </summary>
00122 public string MemoryPageOffset
00123 {
00124     get { return _memoryPageOffset.ToString("X"); }
00125     set
00126     {
00127         if (string.IsNullOrEmpty(value))
00128             return;
00129         try
00130         {
00131             _memoryPageOffset = Convert.ToInt32(value, 16);
00132         }
00133         catch { }
00134     }
00135 }
00136
00137 /// <summary>
00138 /// Is the Program Running
00139 /// </summary>
00140 public bool IsRunning
00141 {
00142     get { return W65C02.IsRunning; }
00143     set
00144     {
00145         W65C02.IsRunning = value;
00146         RaisePropertyChanged("IsRunning");
00147     }
00148 }
00149
00150 /// <summary>

```

```

00151 /// Is the banked ROM Loaded.
00152 /// </summary>
00153     public bool IsRomLoaded { get; set; }
00154
00155     /// <summary>
00156     /// The Slider CPU Speed
00157     /// </summary>
00158     public int CpuSpeed { get; set; }
00159
00160     /// <summary>
00161     /// The Model used for saving, loading and using data from Settings.xml
00162     /// </summary>
00163     public static SettingsModel SettingsModel { get; set; }
00164
00165     /// <summary>
00166     /// RelayCommand for Stepping through the program one instruction at a time.
00167     /// </summary>
00168     public RelayCommand StepCommand { get; set; }
00169
00170     /// <summary>
00171     /// RelayCommand to Reset the Program back to its initial state.
00172     /// </summary>
00173     public RelayCommand ResetCommand { get; set; }
00174
00175     /// <summary>
00176     /// RelayCommand that Run/Pauses Execution
00177     /// </summary>
00178     public RelayCommand RunPauseCommand { get; set; }
00179
00180     /// <summary>
00181     /// RelayCommand that updates the Memory Map when the Page changes
00182     /// </summary>
00183     public RelayCommand UpdateMemoryMapCommand { get; set; }
00184
00185     /// <summary>
00186     /// The RelayCommand that adds a new breakpoint
00187     /// </summary>
00188     public RelayCommand AddBreakPointCommand { get; set; }
00189
00190     /// <summary>
00191     /// The RelayCommand that opens the About window.
00192     /// </summary>
00193     public RelayCommand AboutCommand { get; set; }
00194
00195     /// <summary>
00196     /// The RelayCommand that Removes an existing breakpoint.
00197     /// </summary>
00198     public RelayCommand RemoveBreakPointCommand { get; set; }
00199
00200     /// <summary>
00201     /// The Command that loads or saves the settings.
00202     /// </summary>
00203     public RelayCommand SettingsCommand { get; set; }
00204
00205     /// <summary>
00206     /// The Command that loads or saves the settings.
00207     /// </summary>
00208     public RelayCommand<IClosable> CloseCommand { get; private set; }
00209
00210     /// <summary>
00211     /// The current serial port object name.
00212     /// </summary>
00213     public string CurrentSerialPort
00214     {
00215         get
00216         {
00217             return W65C51.ObjectName;
00218         }
00219     }
00220
00221     /// <summary>
00222     /// The title for the main window.
00223     /// </summary>
00224     public string WindowTitle { get { return Versioning.Product.Title; } }
00225 #endregion
00226
00227 #region public Methods
00228     /// <summary>
00229     /// Creates a new Instance of the MainViewModel.
00230     /// </summary>
00231     public MainViewModel()
00232     {
00233         var _formatter = new XmlSerializer(typeof(SettingsModel));
00234         Stream _stream = new FileStream(FileLocations.SettingsFile, FileMode.OpenOrCreate);
00235         if (!(_stream == null) || (0 >= _stream.Length))
00236         {
00237             SettingsModel = (SettingsModel)_formatter.Deserialize(_stream);

```

```

00238         if ((SettingsModel.SettingsVersionMajor < Versioning.SettingsFile.Major) ||
00239             (SettingsModel.SettingsVersionMinor < Versioning.SettingsFile.Minor) ||
00240             (SettingsModel.SettingsVersionBuild < Versioning.SettingsFile.Build) ||
00241             (SettingsModel.SettingsVersionRevision < Versioning.SettingsFile.Revision))
00242         {
00243             #if !DEBUG
00244                 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));
00245             #else
00246                 MessageBox.Show("Settings file contains old information...\nDeleting old settings
file...",
00247                                 "Settings file stale!", MessageBoxButton.OKCancel,
MessageBoxImage.Warning,
00248                                 MessageBoxResult.OK);
00249                 // Close the file, then delete it.
00250                 _stream.Close();
00251                 File.Delete(FileLocations.SettingsFile);
00252                 SettingsModel = SettingsFile.CreateNew();
00253             #endif
00254         }
00255     }
00256     else
00257     {
00258         MessageBox.Show("Creating new settings file...");
00259         SettingsModel = SettingsFile.CreateNew();
00260     }
00261     _stream.Close();
00262
00263     HM62256 = new HM62256(MemoryMap.BankedRam.TotalBanks, MemoryMap.BankedRam.Offset,
MemoryMap.BankedRam.Length);
00264     AT28C64 = new AT28CXX(MemoryMap.SharedRom.Offset, MemoryMap.SharedRom.Length, 1);
00265     AT28C010 = new AT28CXX(MemoryMap.BankedRom.Offset, MemoryMap.BankedRom.Length,
MemoryMap.BankedRom.TotalBanks);
00266     W65C02 = new W65C02();
00267     W65C51 = new W65C51(W65C02, MemoryMap.Devices.ACIA.Offset);
00268     W65C51.Init(SettingsModel.ComPortName.ToString());
00269     W65C22 = new W65C22(W65C02, MemoryMap.Devices.GPIO.Offset, MemoryMap.Devices.GPIO.Length);
00270     W65C22.Init(1000);
00271     MM65SIB = new W65C22(W65C02, MemoryMap.Devices.MM65SIB.Offset,
MemoryMap.Devices.MM65SIB.Length);
00272     MM65SIB.Init(1000);
00273
00274     MemoryMap.Init(W65C02, W65C22, MM65SIB, W65C51, HM62256, AT28C010, AT28C64);
00275
00276     // Now we can load the BIOS.
00277     byte[][] _bios = AT28C64.ReadFile(FileLocations.BiosFile);
00278     if (_bios == null)
00279     {
00280         Environment.Exit(ExitCodes.NO_BIOS);
00281     }
00282     AT28C64.Load(_bios);
00283
00284     AboutCommand = new RelayCommand(About);
00285     AddBreakPointCommand = new RelayCommand(AddBreakPoint);
00286     CloseCommand = new RelayCommand<IClosable>(Close);
00287     RemoveBreakPointCommand = new RelayCommand(RemoveBreakPoint);
00288     ResetCommand = new RelayCommand(Reset);
00289     RunPauseCommand = new RelayCommand(RunPause);
00290     SettingsCommand = new RelayCommand(Settings);
00291     StepCommand = new RelayCommand(Step);
00292     UpdateMemoryMapCommand = new RelayCommand(UpdateMemoryPage);
00293
00294     Messenger.Default.Register<NotificationMessage>(this, GenericNotification);
00295     Messenger.Default.Register<NotificationMessage<RomFileModel>>(this,
BinaryLoadedNotification);
00296     Messenger.Default.Register<NotificationMessage<SettingsModel>>(this,
SettingsAppliedNotification);
00297     Messenger.Default.Register<NotificationMessage<StateFileModel>>(this,
StateLoadedNotification);
00298
00299     MemoryPage = new MultiThreadedObservableCollection<MemoryRowModel>();
00300     OutputLog = new MultiThreadedObservableCollection<OutputLog>();
00301     Breakpoints = new MultiThreadedObservableCollection<Breakpoint>();
00302
00303     UpdateMemoryPage();
00304
00305     _backgroundWorker = new BackgroundWorker { WorkerSupportsCancellation = true,
WorkerReportsProgress = false };
00306     _backgroundWorker.DoWork += BackgroundWorkerDoWork;
00307     Application.Current.MainWindow.Closing += new CancelEventHandler(OnClose);
00308     Application.Current.MainWindow.Loaded += new RoutedEventHandler(OnLoad);
00309
00310     Reset();
00311 }
00312
00313 public void OnLoad(Object sender, RoutedEventArgs e)

```

```

00314     {
00315     #if !DEBUG
00316         if (Versioning.Product.Major < 1)
00317         {
00318             var result = MessageBox.Show(String.Format("Thank you for using {0}\n" +
00319                                                         "Be warned that this is a beta build.\n" +
00320                                                         "It may break or have bugs.",
00321                                                         Versioning.Product.Name),
00322                                                         Versioning.Product.Title,
00323                                                         MessageBoxButton.OKCancel,
00324                                                         MessageBoxImage.Warning,
00325                                                         MessageBoxResult.None);
00326             if (result == MessageBoxResult.Cancel)
00327             {
00328                 // Exit without making any changes.
00329                 Environment.Exit(ExitCodes.NO_ERROR);
00330             }
00331         }
00332     #endif
00333     }
00334     public void OnClose(Object sender, CancelEventArgs e)
00335     {
00336         e.Cancel = false;
00337         if (IsRunning)
00338         {
00339             MessageBox.Show("You can't quit the emulator while it is actively running!",
00340                             "You can't do that!", MessageBoxButton.OK, MessageBoxImage.Stop);
00341             e.Cancel = true;
00342             return;
00343         }
00344         #if !DEBUG
00345         else
00346         {
00347             var result = MessageBox.Show("Are you sure you want to quit the emulator?",
00348                                         "To quit, or not to quit -- that is the question.",
00349                                         MessageBoxButton.YesNo, MessageBoxImage.Question,
00350                                         MessageBoxResult.No);
00351             if (result == MessageBoxResult.No)
00352             {
00353                 e.Cancel = true;
00354                 return;
00355             }
00356         }
00357     #endif
00358     Stream stream = new FileStream(FileLocations.SettingsFile, FileMode.Create,
00359                                   FileAccess.Write, FileShare.None);
00360     XmlSerializer XmlFormatter = new XmlSerializer(typeof(SettingsModel));
00361     XmlFormatter.Serialize(stream, MainViewModel.SettingsModel);
00362     stream.Flush();
00363     stream.Close();
00364     W65C51.Fini();
00365     }
00366     #endregion
00367     #region Private Methods
00368     private void Close(IClosable window)
00369     {
00370         if ((window != null) && (!IsRunning))
00371         {
00372             Environment.Exit(ExitCodes.NO_ERROR);
00373         }
00374     }
00375     private void BinaryLoadedNotification(NotificationMessage<RomFileModel> notificationMessage)
00376     {
00377         if (notificationMessage.Notification != "FileLoaded")
00378         {
00379             return;
00380         }
00381         // Load Banked ROM
00382         AT28C010.Load(notificationMessage.Content.Rom);
00383         IsRomLoaded = true;
00384         RaisePropertyChanged("IsRomLoaded");
00385         Reset();
00386     }
00387     private void StateLoadedNotification(NotificationMessage<StateFileModel> notificationMessage)
00388     {
00389         if (notificationMessage.Notification != "StateLoaded")
00390         {
00391             return;
00392         }
00393         Reset();
00394     }
00395 
```



```

00397
00398         OutputLog = new
MultiThreadedObservableCollection<OutputLog>(notificationMessage.Content.OutputLog);
00399         RaisePropertyChanged("OutputLog");
00400
00401         NumberOfCycles = notificationMessage.Content.NumberOfCycles;
00402
00403         W65C02 = notificationMessage.Content.W65C02;
00404         W65C22 = notificationMessage.Content.W65C22;
00405         MM65SIB = notificationMessage.Content.MM65SIB;
00406         W65C51 = notificationMessage.Content.W65C51;
00407         AT28C010 = notificationMessage.Content.AT28C010;
00408         AT28C64 = notificationMessage.Content.AT28C64;
00409         UpdateMemoryPage();
00410         UpdateUi();
00411
00412         IsRomLoaded = true;
00413         RaisePropertyChanged("IsRomLoaded");
00414     }
00415
00416     private void GenericNotification(NotificationMessage notificationMessage)
00417     {
00418         if (notificationMessage.Notification == "CloseFile")
00419         {
00420             AT28C010.Clear();
00421             if (IsRunning) { RunPause(); }
00422             IsRomLoaded = false;
00423             RaisePropertyChanged("IsRomLoaded");
00424             return;
00425         }
00426         else if (notificationMessage.Notification == "LoadFile")
00427         {
00428             var dialog = new OpenFileDialog { DefaultExt = ".bin", Filter =
00429                 "All Files (*.bin, *.65C02)|*.bin;*.65C02|Binary
Assembly (*.bin)|" +
00430                 "*.bin|WolfNet 65C02 Emulator Save State
(*.65C02)|*.65C02" };
00431             var result = dialog.ShowDialog();
00432             if (result != true)
00433             {
00434                 return;
00435             }
00436
00437             if (Path.GetExtension(dialog.FileName.ToUpper()) == ".BIN")
00438             {
00439                 byte[][] _rom = AT28C010.ReadFile(dialog.FileName);
00440
00441                 Messenger.Default.Send(new NotificationMessage<RomFileModel>(new RomFileModel
00442                 {
00443                     Rom = _rom,
00444                     RomBanks = AT28C010.Banks,
00445                     RomBankSize = AT28C010.Length,
00446                     RomFilePath = dialog.FileName,
00447                     RomFileName = Path.GetFileName(dialog.FileName),
00448                     }, "FileLoaded"));
00449             }
00450             else if (Path.GetExtension(dialog.FileName.ToUpper()) == ".6502")
00451             {
00452                 var formatter = new BinaryFormatter();
00453                 Stream stream = new FileStream(dialog.FileName, FileMode.Open);
00454                 var fileModel = (StateFileModel)formatter.Deserialize(stream);
00455
00456                 stream.Close();
00457
00458                 Messenger.Default.Send(new NotificationMessage<StateFileModel>(fileModel,
"StateLoaded"));
00459             }
00460         }
00461         else if (notificationMessage.Notification == "SaveState")
00462         {
00463             var dialog = new SaveFileDialog { DefaultExt = ".65C02", Filter =
00464                 "WolfNet W65C02 Emulator Save State
(*.65C02)|*.65C02" };
00465             var result = dialog.ShowDialog();
00466             if (result != true)
00467             {
00468                 return;
00469             }
00470
00471             var formatter = new BinaryFormatter();
00472             Stream stream = new FileStream(dialog.FileName, FileMode.Create, FileAccess.Write,
FileShare.None);
00473             formatter.Serialize(stream, new StateFileModel
00474             {
00475                 NumberOfCycles = NumberOfCycles,

```

```

00478         OutputLog = OutputLog,
00479         W65C02 = W65C02,
00480         W65C22 = W65C22,
00481         MM65SIB = MM65SIB,
00482         W65C51 = W65C51,
00483         AT28C010 = AT28C010,
00484         AT28C64 = AT28C64,
00485     });
00486     stream.Close();
00487 }
00488 else
00489 {
00490     return;
00491 }
00492 }
00493
00494 private void SettingsAppliedNotification(NotificationMessage<SettingsModel>
notificationMessage)
00495 {
00496     if (notificationMessage.Notification != "SettingsApplied")
00497     {
00498         return;
00499     }
00500
00501     SettingsModel = notificationMessage.Content;
00502     W65C51.Init(notificationMessage.Content.ComPortName);
00503     RaisePropertyChanged("SettingsModel");
00504     UpdateUi();
00505 }
00506
00507 private void UpdateMemoryPage()
00508 {
00509     MemoryPage.Clear();
00510     var offset = _memoryPageOffset * 256;
00511
00512     var multiplier = 0;
00513     for (ushort i = (ushort)offset; i < 256 * (_memoryPageOffset + 1); i++)
00514     {
00515
00516         MemoryPage.Add(new MemoryRowModel
00517         {
00518             Offset = ((16 * multiplier) + offset).ToString("X").PadLeft(4, '0'),
00519             Location00 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00520             Location01 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00521             Location02 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00522             Location03 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00523             Location04 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00524             Location05 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00525             Location06 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00526             Location07 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00527             Location08 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00528             Location09 = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00529             Location0A = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00530             Location0B = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00531             Location0C = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00532             Location0D = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00533             Location0E = MemoryMap.ReadWithoutCycle(i++).ToString("X").PadLeft(2, '0'),
00534             Location0F = MemoryMap.ReadWithoutCycle(i).ToString("X").PadLeft(2, '0'),
00535         });
00536         multiplier++;
00537     }
00538 }
00539
00540 private void Reset()
00541 {
00542     IsRunning = false;
00543
00544     if (_backgroundWorker.IsBusy)
00545         _backgroundWorker.CancelAsync();
00546
00547     // "Reset" the Hardware...
00548     W65C02.Reset();
00549     RaisePropertyChanged("W65C02");
00550     W65C22.Reset();
00551     RaisePropertyChanged("W65C22");
00552     MM65SIB.Reset();
00553     RaisePropertyChanged("MM65SIB");
00554     W65C51.Reset();
00555     RaisePropertyChanged("W65C51");
00556     HM62256.Reset();
00557     RaisePropertyChanged("HM62256");
00558
00559     IsRunning = false;
00560     NumberOfCycles = 0;
00561     RaisePropertyChanged("NumberOfCycles");
00562
00563     UpdateMemoryPage();

```

```

00564         RaisePropertyChanged("MemoryPage");
00565
00566         OutputLog.Clear();
00567         RaisePropertyChanged("CurrentDisassembly");
00568
00569         OutputLog.Insert(0, GetOutputLog());
00570         UpdateUi();
00571     }
00572
00573     private void Step()
00574     {
00575         IsRunning = false;
00576
00577         if (_backgroundWorker.IsBusy)
00578             _backgroundWorker.CancelAsync();
00579
00580         StepProcessor();
00581         UpdateMemoryPage();
00582
00583         OutputLog.Insert(0, GetOutputLog());
00584         UpdateUi();
00585     }
00586
00587     private void UpdateUi()
00588     {
00589         RaisePropertyChanged("W65C02");
00590         RaisePropertyChanged("NumberOfCycles");
00591         RaisePropertyChanged("CurrentDisassembly");
00592         RaisePropertyChanged("MemoryPage");
00593     }
00594
00595     private void StepProcessor()
00596     {
00597         W65C02.NextStep();
00598         NumberOfCycles = W65C02.GetCycleCount();
00599     }
00600
00601     private OutputLog GetOutputLog()
00602     {
00603         if (W65C02.CurrentDisassembly == null)
00604         {
00605             return new OutputLog(new Disassembly());
00606         }
00607
00608         return new OutputLog(W65C02.CurrentDisassembly)
00609         {
00610             XRegister = W65C02.XRegister.ToString("X").PadLeft(2, '0'),
00611             YRegister = W65C02.YRegister.ToString("X").PadLeft(2, '0'),
00612             Accumulator = W65C02.Accumulator.ToString("X").PadLeft(2, '0'),
00613             NumberOfCycles = NumberOfCycles,
00614             StackPointer = W65C02.StackPointer.ToString("X").PadLeft(2, '0'),
00615             ProgramCounter = W65C02.ProgramCounter.ToString("X").PadLeft(4, '0'),
00616             CurrentOpCode = W65C02.CurrentOpCode.ToString("X").PadLeft(2, '0')
00617         };
00618     }
00619
00620     private void RunPause()
00621     {
00622         var isRunning = !IsRunning;
00623
00624         if (isRunning)
00625             _backgroundWorker.RunWorkerAsync();
00626         else
00627             _backgroundWorker.CancelAsync();
00628
00629         IsRunning = !IsRunning;
00630     }
00631
00632     private void BackgroundWorkerDoWork(object sender, DoWorkEventArgs e)
00633     {
00634         var worker = sender as BackgroundWorker;
00635         var outputLogs = new List<OutputLog>();
00636
00637         while (true)
00638         {
00639             if (worker != null && worker.CancellationPending || IsBreakPointTriggered())
00640             {
00641                 e.Cancel = true;
00642
00643                 RaisePropertyChanged("W65C02");
00644
00645                 foreach (var log in outputLogs)
00646                     OutputLog.Insert(0, log);
00647
00648                 UpdateMemoryPage();
00649                 return;
00650             }

```

```
00651
00652         StepProcessor();
00653         outputLogs.Add(GetOutputLog());
00654
00655         if (NumberOfCycles % GetLogModValue() == 0)
00656         {
00657             foreach (var log in outputLogs)
00658                 OutputLog.Insert(0, log);
00659
00660             outputLogs.Clear();
00661             UpdateUi();
00662         }
00663         Thread.Sleep(GetSleepValue());
00664     }
00665 }
00666
00667 private bool IsBreakPointTriggered()
00668 {
00669     //This prevents the Run Command from getting stuck after reaching a breakpoint
00670     if (_breakpointTriggered)
00671     {
00672         _breakpointTriggered = false;
00673         return false;
00674     }
00675
00676     foreach (var breakpoint in Breakpoints.Where(x => x.IsEnabled))
00677     {
00678         if (!int.TryParse(breakpoint.Value, NumberStyles.AllowHexSpecifier,
00679             CultureInfo.InvariantCulture, out int value))
00680             continue;
00681
00682         if (breakpoint.Type == BreakpointType.NumberOfCycleType && value == NumberOfCycles)
00683         {
00684             _breakpointTriggered = true;
00685             RunPause();
00686             return true;
00687         }
00688
00689         if (breakpoint.Type == BreakpointType.ProgramCounterType && value ==
00690             W65C02.ProgramCounter)
00691         {
00692             _breakpointTriggered = true;
00693             RunPause();
00694             return true;
00695         }
00696     }
00697     return false;
00698 }
00699 private int GetLogModValue()
00700 {
00701     switch (CpuSpeed)
00702     {
00703         case 0:
00704         case 1:
00705         case 2:
00706         case 3:
00707         case 4:
00708         case 5:
00709             return 1;
00710         case 6:
00711             return 5;
00712         case 7:
00713             return 20;
00714         case 8:
00715             return 30;
00716         case 9:
00717             return 40;
00718         case 10:
00719             return 50;
00720         default:
00721             return 5;
00722     }
00723 }
00724
00725 private int GetSleepValue()
00726 {
00727     switch (CpuSpeed)
00728     {
00729         case 0:
00730             return 550;
00731         case 1:
00732             return 550;
00733         case 2:
00734             return 440;
00735         case 3:
```

```

00736         return 330;
00737     case 4:
00738         return 220;
00739     case 5:
00740         return 160;
00741     case 6:
00742         return 80;
00743     case 7:
00744         return 40;
00745     case 8:
00746         return 20;
00747     case 9:
00748         return 10;
00749     case 10:
00750         return 5;
00751     default:
00752         return 5;
00753     }
00754 }
00755
00756 private void About()
00757 {
00758     IsRunning = false;
00759
00760     if (_backgroundWorker.IsBusy)
00761         _backgroundWorker.CancelAsync();
00762
00763     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);
00764 }
00765
00766 private void Settings()
00767 {
00768     IsRunning = false;
00769
00770     if (_backgroundWorker.IsBusy)
00771         _backgroundWorker.CancelAsync();
00772
00773     Messenger.Default.Send(new NotificationMessage<SettingsModel>(SettingsModel,
"SettingsWindow"));
00774 }
00775
00776 private void AddBreakPoint()
00777 {
00778     Breakpoints.Add(new Breakpoint());
00779     RaisePropertyChanged("Breakpoints");
00780 }
00781
00782 private void RemoveBreakPoint()
00783 {
00784     if (SelectedBreakpoint == null)
00785         return;
00786
00787     Breakpoints.Remove(SelectedBreakpoint);
00788     SelectedBreakpoint = null;
00789     RaisePropertyChanged("SelectedBreakpoint");
00790 }
00791 #endregion
00792 }
00793 }

```

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.](#)

```

00001 using System.IO;
00002 using System.Runtime.Serialization.Formatters.Binary;
00003 using GalaSoft.MvvmLight;
00004 using GalaSoft.MvvmLight.Command;
00005 using GalaSoft.MvvmLight.Ioc;
00006 using GalaSoft.MvvmLight.Messaging;
00007 using Microsoft.Win32;
00008 using Emulator.Model;
00009
00010 namespace Emulator.ViewModel
00011 {
00012     /// <summary>
00013     /// The ViewModel Used by the SaveFileView
00014     /// </summary>
00015     public class SaveFileViewModel : ViewModelBase
00016     {
00017         private readonly StateFileModel _stateFileModel;
00018
00019 #region Properties
00020     /// <summary>
00021     /// The Relay Command called when saving a file
00022     /// </summary>
00023     public RelayCommand SaveFileCommand { get; set; }
00024
00025     /// <summary>
00026     /// The Relay Command called when closing a file
00027     /// </summary>
00028     public RelayCommand CloseCommand { get; set; }
00029
00030     /// <summary>
00031     /// The Relay Command called when Selecting a file
00032     /// </summary>
00033     public RelayCommand SelectFileCommand { get; set; }
00034
00035     /// <summary>
00036     /// The file to be saved
00037     /// </summary>
00038     public string Filename { get; set; }
00039
00040     /// <summary>
00041     /// Tells the UI that that a file has been selected and can be saved.
00042     /// </summary>
00043     public bool SaveEnabled { get { return !string.IsNullOrEmpty(Filename); }}
00044 #endregion
00045
00046 #region Public Methods
00047     /// <summary>
00048     /// Instantiates a new instance of the SaveFileViewModel. This is used by the IOC to create the
    default instance.
00049     /// </summary>
00050     [PreferredConstructor]
00051     public SaveFileViewModel()
00052     {
00053
00054     }
00055
00056     /// <summary>
00057     /// Instantiates a new instance of the SaveFileViewModel
00058     /// </summary>
00059     /// <param name="stateFileModel">The StateFileModel to be serialized to a file</param>
00060     public SaveFileViewModel(StateFileModel stateFileModel)
00061     {
00062         SaveFileCommand = new RelayCommand(Save);
00063         CloseCommand = new RelayCommand(Close);
00064         SelectFileCommand = new RelayCommand(Select);
00065         _stateFileModel = stateFileModel;
00066     }
00067 #endregion
00068
00069 #region Private Methods
00070     private void Save()
00071     {
00072         var formatter = new BinaryFormatter();
00073         Stream stream = new FileStream(Filename, FileMode.Create, FileAccess.Write,
FileShare.None);
00074         formatter.Serialize(stream, _stateFileModel);
00075         stream.Close();
00076
00077         Close();
00078     }
00079
00080     private static void Close()
00081     {

```

```

00082         Messenger.Default.Send(new NotificationMessage("CloseSaveFileDialog"));
00083     }
00084
00085     private void Select()
00086     {
00087         var dialog = new SaveFileDialog { DefaultExt = ".6502", Filter = "WolfNet W65C02 Emulator
Save State (*.6502)|*.6502" };
00088
00089         var result = dialog.ShowDialog();
00090
00091         if (result != true)
00092             return;
00093
00094         Filename = dialog.FileName;
00095         RaisePropertyChanged("Filename");
00096         RaisePropertyChanged("SaveEnabled");
00097     }
00098 }
00099 #endregion
00100 }
00101 }

```

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.](#)

```

00001 using System;
00002 using System.Collections.ObjectModel;
00003 using System.IO;
00004 using System.IO.Ports;
00005 using System.Xml.Serialization;
00006 using GalaSoft.MvvmLight;
00007 using GalaSoft.MvvmLight.Command;
00008 using GalaSoft.MvvmLight.Ioc;
00009 using GalaSoft.MvvmLight.Messaging;
00010 using Emulator.Model;
00011
00012 namespace Emulator.ViewModel
00013 {
00014     /// <summary>
00015     /// The ViewModel Used by the SaveFileView
00016     /// </summary>
00017     public class SettingsViewModel : ViewModelBase
00018     {
00019         #region Properties
00020         /// <summary>
00021         /// The RelayCommand called when saving a file
00022         /// </summary>
00023         public RelayCommand ApplyCommand { get; set; }
00024
00025         /// <summary>
00026         /// The RelayCommand called when closing a file
00027         /// </summary>
00028         public RelayCommand CloseCommand { get; set; }
00029
00030         /// <summary>
00031         /// Tells the UI that that a file has been selected and can be saved.
00032         /// </summary>
00033         public bool ApplyEnabled { get { return
!string.IsNullOrEmpty(Emulator.FileLocations.SettingsFile); } }

```

```

00034
00035 /// <summary>
00036 /// Creates a new instance of PortList, the list of all COM ports available to the computer
00037 /// </summary>
00038 ///
00039     public ObservableCollection<string> PortList { get { return _PortList; } }
00040     private readonly ObservableCollection<string> _PortList = new ObservableCollection<string>();
00041
00042     public static string ComPortSelection { get; set; }
00043     public static SettingsModel SettingsModel { get; set; }
00044 #endregion
00045
00046 #region Public Methods
00047 /// <summary>
00048 /// Instantiates a new instance of the SettingsViewModel. This is used by the IOC to create the
    default instance.
00049 /// </summary>
00050     [PreferredConstructor]
00051     public SettingsViewModel()
00052     {
00053     }
00054 }
00055
00056 /// <summary>
00057 /// Instantiates a new instance of the SettingsViewModel
00058 /// </summary>
00059 /// <param name="settingsModel">The SettingsFileModel to be serialized to a file</param>
00060     public SettingsViewModel(SettingsModel settingsModel)
00061     {
00062         ApplyCommand = new RelayCommand(Apply);
00063         CloseCommand = new RelayCommand(Close);
00064         ComPortSelection = settingsModel.ComPortName;
00065
00066         UpdatePortList();
00067     }
00068
00069 /// <summary>
00070 /// Updates PortList with the COM ports available to the computer
00071 /// </summary>
00072     public void UpdatePortList()
00073     {
00074         PortList.Clear();
00075         foreach (string s in SerialPort.GetPortNames())
00076         {
00077             PortList.Add(s);
00078         }
00079         RaisePropertyChanged("PortList");
00080     }
00081 #endregion
00082
00083 #region Private Methods
00084     private void Apply()
00085     {
00086         Messenger.Default.Send(new NotificationMessage<SettingsModel>(new SettingsModel
00087         {
00088             SettingsVersionMajor = Versioning.SettingsFile.Major,
00089             SettingsVersionMinor = Versioning.SettingsFile.Minor,
00090             SettingsVersionBuild = Versioning.SettingsFile.Build,
00091             SettingsVersionRevision = Versioning.SettingsFile.Revision,
00092             ComPortName = ComPortSelection,
00093         }, "SettingsApplied"));
00094         Messenger.Default.Send(new NotificationMessage("CloseSettingsWindow"));
00095     }
00096
00097     private static void Close()
00098     {
00099         Messenger.Default.Send(new NotificationMessage("CloseSettingsWindow"));
00100     }
00101 #endregion
00102 }
00103 }

```

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.](#)

```

00001 /*
00002 In App.xaml:
00003 <Application.Resources>
00004 <vm:ViewModelLocator xmlns:vm="clr-namespace:Emulator"
00005 x:Key="Locator" />
00006 </Application.Resources>
00007
00008 In the View:
00009 DataContext="{Binding Source={StaticResource Locator}, Path=ViewModelName}"
00010
00011 You can also use Blend to do all this with the tool's support.
00012 See http://www.galasoft.ch/mvvm
00013 */
00014
00015 using GalaSoft.MvvmLight.Ioc;
00016 using Microsoft.Practices.ServiceLocation;
00017
00018 namespace Emulator.ViewModel
00019 {
00020     /// <summary>
00021     /// This class contains static references to all the view models in the
00022     /// application and provides an entry point for the bindings.
00023     /// </summary>
00024     public class ViewModelLocator
00025     {
00026         /// <summary>
00027         /// Initializes a new instance of the ViewModelLocator class.
00028         /// </summary>
00029         public ViewModelLocator()
00030         {
00031             ServiceLocator.SetLocatorProvider(() => SimpleIoc.Default);
00032
00033             SimpleIoc.Default.Register<MainViewModel>();
00034             SimpleIoc.Default.Register<SaveFileViewModel>();
00035             SimpleIoc.Default.Register<SettingsViewModel>();
00036         }
00037
00038         /// <summary>
00039         /// The MainViewModel Instance
00040         /// </summary>
00041         public MainViewModel Main
00042         {
00043             get { return ServiceLocator.Current.GetInstance<MainViewModel>(); }
00044         }
00045
00046         /// <summary>
00047         /// The SaveFileViewModel Instance
00048         /// </summary>
00049         public SaveFileViewModel SaveFile
00050         {
00051             get { return ServiceLocator.Current.GetInstance<SaveFileViewModel>(); }
00052         }
00053
00054         /// <summary>
00055         /// The SaveFileViewModel Instance
00056         /// </summary>
00057         public SettingsViewModel Settings
00058         {
00059             get { return ServiceLocator.Current.GetInstance<SettingsViewModel>(); }
00060         }
00061
00062         /// <summary>
00063         /// The Cleanup Method
00064         /// </summary>
00065         public static void Cleanup()
00066         {
00067             /// <todo>
00068             /// Clear the ViewModels
00069             /// </todo>
00070         }
00071     }
00072 }

```

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.](#)

```
00001 using System;
00002 using System.IO;
00003
00004 namespace Hardware
00005 {
00006     /// <summary>
00007     /// An implementation of a W65C02 Processor.
00008     /// </summary>
00009     [Serializable]
00010     public class AT28CXX
00011     {
00012         ///All of the properties here are public and read only to facilitate ease of debugging and
00013         testing.
00014         #region Properties
00015         /// <summary>
00016         /// The ROM.
00017         /// </summary>
00018         public byte[][] Memory { get; private set; }
00019         /// <summary>
00020         /// The total number of banks on the ROM.
00021         /// </summary>
00022         public byte Banks { get; private set; }
00023         /// <summary>
00024         /// The bank the ROM is currently using.
00025         /// </summary>
00026         public byte CurrentBank { get; private set; }
00027         /// <summary>
00028         /// The memory offset
00029         /// </summary>
00030         public int Offset { get; private set; }
00031         /// <summary>
00032         /// The end of memory
00033         /// </summary>
00034         public int End { get { return Offset + Length; } }
00035         /// <summary>
00036         /// The memory length
00037         /// </summary>
00038         public int Length { get; private set; }
00039         /// <summary>
00040         /// The processor reference
00041         /// </summary>
00042         public W65C02 Processor { get; private set; }
00043         #endregion
00044         #region Public Methods
00045         /// <summary>
00046         /// Default Constructor, Instantiates a new instance of the processor.
00047         /// </summary>
00048         public AT28CXX(int offset, int length, byte banks)
00049         {
00050             Memory = new byte[banks][];
00051             for (int i = 0; i < banks; i++)
00052             {
00053                 Memory[i] = new byte[length + 1];
00054             }
00055         }
00056     }
00057 }
```

```

00061         Offset = offset;
00062         Length = length;
00063         Banks = banks;
00064         CurrentBank = 0;
00065     }
00066
00067     /// <summary>
00068     /// Loads a program into ROM.
00069     /// </summary>
00070     /// <param name="data">The program to be loaded</param>
00071     public void Load(byte[][] data)
00072     {
00073         for (byte i = 0; i < Banks; i++)
00074         {
00075             Load(i, data[i]);
00076         }
00077     }
00078
00079     /// <summary>
00080     /// Loads a program into ROM.
00081     /// </summary>
00082     /// <param name="bank">The bank to load data to.</param>
00083     /// <param name="data">The data to be loaded to ROM.</param>
00084     public void Load(byte bank, byte[] data)
00085     {
00086         for (int i = 0; i <= Length; i++)
00087         {
00088             Memory[bank][i] = data[i];
00089         }
00090     }
00091
00092     public byte[][] ReadFile(string filename)
00093     {
00094         byte[][] bios = new byte[Banks][];
00095         try
00096         {
00097             FileStream file = new FileStream(filename, FileMode.Open, FileAccess.Read);
00098             for (int i = 0; i < Banks; i++)
00099             {
00100                 bios[i] = new byte[Length + 1];
00101                 for (int j = 0; j <= Length; j++)
00102                 {
00103                     bios[i][j] = new byte();
00104                     bios[i][j] = (byte)file.ReadByte();
00105                 }
00106             }
00107         }
00108         catch (Exception)
00109         {
00110             return null;
00111         }
00112         return bios;
00113     }
00114
00115     /// <summary>
00116     /// Returns the byte at a given address without incrementing the cycle. Useful for test harness.
00117     /// </summary>
00118     /// <param name="bank">The bank to read data from.</param>
00119     /// <param name="address"></param>
00120     /// <returns>the byte being returned</returns>
00121     public byte Read(int address)
00122     {
00123         return Memory[CurrentBank][address - Offset];
00124     }
00125
00126     /// <summary>
00127     /// Writes data to the given address without incrementing the cycle.
00128     /// </summary>
00129     /// <param name="bank">The bank to load data to.</param>
00130     /// <param name="address">The address to write data to</param>
00131     /// <param name="data">The data to write</param>
00132     public void Write(int address, byte data)
00133     {
00134         _ = address;
00135         _ = data;
00136         return;
00137     }
00138
00139     /// <summary>
00140     /// Dumps the entire memory object. Used when saving the memory state
00141     /// </summary>
00142     /// <returns>2 dimensional array of data analogous to the ROM of the computer.</returns>
00143     public byte[][] DumpMemory()
00144     {
00145         return Memory;
00146     }
00147

```

```

00148 /// <summary>
00149 /// Dumps the selected ROM bank.
00150 /// </summary>
00151 /// <param name="bank">The bank to dump data from.</param>
00152 /// <returns>Array that represents the selected ROM bank.</returns>
00153 public byte[] DumpMemory(byte bank)
00154 {
00155     byte[] _tempMemory = new byte[MemoryMap.BankedRom.Length + 1];
00156     for (var i = 0; i < MemoryMap.BankedRom.Length; i++) {
00157         _tempMemory[i] = Memory[bank][i];
00158     }
00159     return _tempMemory;
00160 }
00161
00162 /// <summary>
00163 /// Clears the ROM.
00164 /// </summary>
00165 public void Clear()
00166 {
00167     for (byte i = 0; i < Banks; i++)
00168     {
00169         for (int j = 0; j < Length; j++)
00170         {
00171             Memory[i][j] = 0x00;
00172         }
00173     }
00174 }
00175 #endregion
00176 }
00177 }

```

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.](#)

```

00001 namespace Hardware
00002 {
00003     /// <summary>
00004     /// The addressing modes used by the 6502 Processor
00005     /// </summary>
00006     public enum AddressingMode
00007     {
00008         /// <summary>
00009         /// In this mode a full address is given to operation on IE: Memory byte[] { 0x60, 0x00, 0xFF }
00010         /// would perform an ADC operation and Add the value at ADDRESS 0xFF00 to the accumulator.
00011         /// The address is always LSB first
00012         /// </summary>
00013         Absolute = 1,
00014         /// <summary>
00015         /// In this mode a full address is given to operation on IE: Memory byte[] { 0x7D, 0x00, 0xFF } The
00016         /// full value would then be added to the X Register.
00017         /// If the X register was 0x01 then the address would be 0xFF01. and the value stored there would
00018         /// have an ADC operation performed on it and the value would
00019         /// be added to the accumulator.
00020         /// </summary>
00021         AbsoluteX = 2,
00022         /// <summary>
00023         /// In this mode a full address is given to operation on IE: Memory byte[] { 0x79, 0x00, 0xFF } The
00024         /// full value would then be added to the Y Register.
00025         /// If the Y register was 0x01 then the address would be 0xFF01. and the value stored there would
00026         /// have an ADC operation performed on it and the value would
00027         /// be added to the accumulator
00028     }
00029 }

```

```

00024 /// </summary>
00025     AbsoluteY = 3,
00026 /// <summary>
00027 /// In this mode the instruction operates on the accumulator. No operands are needed.
00028 /// </summary>
00029     Accumulator = 4,
00030 /// <summary>
00031 /// In this mode, the value to operate on immediately follows the instruction. IE: Memory byte[] {
00032     0x69, 0x01 }
00032 /// would perform an ADC operation and Add 0x01 directly to the accumulator
00033 /// </summary>
00034     Immediate = 5,
00035 /// <summary>
00036 /// No address is needed for this mode. EX: BRK (Break), CLC (Clear Carry Flag) etc
00037 /// </summary>
00038     Implied = 6,
00039 /// <summary>
00040 /// In this mode assume the following
00041 /// Memory = { 0x61, 0x02, 0x04, 0x00, 0x03 }
00042 /// RegisterX = 0x01
00043 /// 1. Take the sum of the X Register and the value after the opcode 0x01 + 0x01 = 0x02.
00044 /// 2. Starting at position 0x02 get an address (0x04,0x00) = 0x0004
00045 /// 3. Perform the ADC operation and Add the value at 0x0005 to the accumulator
00046 /// Note: if the Zero Page address is greater than 0xff then roll over the value. IE 0x101 rolls
00047     over to 0x01
00047 /// </summary>
00048     IndirectX = 7,
00049 /// <summary>
00050 /// In this mode assume the following
00051 /// Memory = { 0x61, 0x02, 0x04, 0x00, 0x03 }
00052 /// RegisterY = 0x01
00053 /// 1. Starting at position 0x02 get an address (0x04,0x00) = 0x0004
00054 /// 2. Take the sum of the Y Register and the absolute address 0x01+0x0004 = 0x0005
00055 /// 3. Perform the ADC operation and Add the value at 0x0005 to the accumulator
00056 /// Note: if the address is great that 0xffff then roll over IE: 0x10001 rolls over to 0x01
00057 /// </summary>
00058     IndirectY = 8,
00059 /// <summary>
00060 /// JMP is the only operation that uses this mode. In this mode an absolute address is specified that
00061     points to the location of the absolute address we want to jump to.
00061 /// </summary>
00062     Indirect = 9,
00063 /// <summary>
00064 /// This Mode Changes the PC. It allows the program to change the location of the PC by 127 in either
00065     direction.
00065 /// </summary>
00066     Relative = 10,
00067 /// <summary>
00068 /// In this mode, a zero page address of the value to operate on is specified. This mode can only
00069     operation on values between 0x0 and 0xFF, or those that sit on the zero page of memory. IE: Memory
00070     byte[] { 0x69, 0x02, 0x01 }
00069 /// would perform an ADC operation and Add 0x01 directly to the Accumulator
00070 /// </summary>
00071     ZeroPage = 11,
00072 /// <summary>
00073 /// In this mode, a zero page address of the value to operate on is specified, however the value of
00074     the X register is added to the address IE: Memory byte[] { 0x86, 0x02, 0x01, 0x67, 0x04, 0x01 }
00074 /// In this example we store a value of 0x01 into the X register, then we would perform an ADC
00075     operation using the address of 0x04+0x01=0x05 and Add the result of 0x01 directly to the Accumulator
00075 /// </summary>
00076     ZeroPageX = 12,
00077 /// <summary>
00078 /// This works the same as ZeroPageX except it uses the Y register instead of the X register.
00079 /// </summary>
00080     ZeroPageY = 13,
00081     }
00082 }

```

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.](#)

```

00001 using System;
00002
00003 namespace Hardware
00004 {
00005     /// <summary>
00006     /// Used to help simulating. This class contains the disassembly properties.
00007     /// </summary>
00008     [Serializable]
00009     public class Disassembly
00010     {
00011         /// <summary>
00012         /// The low Address
00013         /// </summary>
00014         public string LowAddress { get; set; }
00015
00016         /// <summary>
00017         /// The High Address
00018         /// </summary>
00019         public string HighAddress { get; set; }
00020
00021         /// <summary>
00022         /// The string representation of the OpCode
00023         /// </summary>
00024         public string OpCodeString { get; set; }
00025
00026         /// <summary>
00027         /// The disassembly of the current step
00028         /// </summary>
00029         public string DisassemblyOutput { get; set; }
00030     }
00031 }

```

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.](#)

```

00001 using System;
00002 using System.IO;
00003
00004 namespace Hardware
00005 {
00006     public class MemoryMap
00007     {
00008         public static class BankedRam
00009         {
00010             private static int _Offset = 0x0000;

```

```

00011         private static int _Length = 0x7FFF;
00012
00013         public static int TotalLength = (BankSize * TotalBanks) - 1;
00014         public static int BankSize = (int)(Length + 1);
00015         public static byte TotalBanks = 16;
00016
00017         public static int Offset { get { return _Offset; } }
00018         public static int Length { get { return _Length; } }
00019     }
00020
00021     public static class DeviceArea
00022     {
00023         private static int _Offset = 0xD000;
00024         private static int _Length = 0x00FF;
00025
00026         /// <summary>
00027         /// The end of memory
00028         /// </summary>
00029         public static int End { get { return Offset + Length; } }
00030         public static int Offset { get { return _Offset; } }
00031         public static int Length { get { return _Length; } }
00032     }
00033
00034     public static class BankedRom
00035     {
00036         private static int _Offset = 0x8000;
00037         private static int _Length = 0x3FFF;
00038
00039         public static byte TotalBanks = 16;
00040
00041         public static int Offset { get { return _Offset; } }
00042         public static int Length { get { return _Length; } }
00043     }
00044
00045     public static class SharedRom
00046     {
00047         private static int _Offset = 0xE000;
00048         private static int _Length = 0x1FFF;
00049
00050         public static byte TotalBanks = 1;
00051
00052         public static int Offset { get { return _Offset; } }
00053         public static int Length { get { return _Length; } }
00054     }
00055
00056     public static class Devices
00057     {
00058         public static class ACIA
00059         {
00060             public static int Length = 0x03;
00061             public static byte Offset = 0x10;
00062         }
00063
00064         public static class GPIO
00065         {
00066             public static int Length = 0x0F;
00067             public static byte Offset = 0x20;
00068         }
00069
00070         public static class MM65SIB
00071         {
00072             public static int Length = 0x0F;
00073             public static byte Offset = 0x30;
00074         }
00075     }
00076
00077     public static readonly int Length = 0xFFFF;
00078
00079     private static W65C02 Processor { get; set; }
00080     private static W65C22 GPIO { get; set; }
00081     private static W65C22 MM65SIB { get; set; }
00082     private static W65C51 ACIA { get; set; }
00083     private static AT28CXX SharedROM { get; set; }
00084     private static AT28CXX BankedROM { get; set; }
00085     private static HM62256 BankedRAM { get; set; }
00086
00087     public static void Init(W65C02 processor, W65C22 gpio, W65C22 mm65sib, W65C51 acia, HM62256
bankedRam, AT28CXX bankedRom, AT28CXX sharedRom)
00088     {
00089         Processor = processor;
00090         GPIO = gpio;
00091         MM65SIB = mm65sib;
00092         ACIA = acia;
00093         SharedROM = sharedRom;
00094         BankedROM = bankedRom;
00095         BankedRAM = bankedRam;
00096     }

```

```

00097
00098 /// <summary>
00099 /// Returns the byte at the given address.
00100 /// </summary>
00101 /// <param name="address">The address to return</param>
00102 /// <returns>the byte being returned</returns>
00103 public static byte Read(int address)
00104 {
00105     var value = ReadWithoutCycle(address);
00106     Processor.IncrementCycleCount();
00107     return value;
00108 }
00109
00110 /// <summary>
00111 /// Returns the byte at the given address without incrementing the cycle count.
00112 /// </summary>
00113 /// <param name="address">The address to return</param>
00114 /// <returns>the byte being returned</returns>
00115 public static byte ReadWithoutCycle(int address)
00116 {
00117     int _address = address;
00118     if ((ACIA.Offset <= _address) && (_address <= (ACIA.Offset + ACIA.Length)))
00119     {
00120         return ACIA.Read(address);
00121     }
00122     else if ((GPIO.Offset <= _address) && (_address <= (GPIO.Offset + GPIO.Length)))
00123     {
00124         return GPIO.Read(_address);
00125     }
00126     else if ((DeviceArea.Offset <= _address) && (_address <= DeviceArea.End))
00127     {
00128         throw new ArgumentOutOfRangeException("Device area accessed where there is no
device!");
00129     }
00130     else if ((SharedROM.Offset <= _address) && (_address <= SharedROM.End))
00131     {
00132         return SharedROM.Read(_address);
00133     }
00134     else if ((BankedROM.Offset <= _address) && (_address <= BankedROM.End))
00135     {
00136         return BankedROM.Read(_address);
00137     }
00138     else if ((BankedRAM.Offset <= _address) && (_address <= BankedRAM.End))
00139     {
00140         return BankedRAM.Read(_address);
00141     }
00142     else
00143     {
00144         return 0x00;
00145     }
00146 }
00147
00148 /// <summary>
00149 /// Writes data to the given address.
00150 /// </summary>
00151 /// <param name="address">The address to write data to.</param>
00152 /// <param name="data">The data to write.</param>
00153 public static void Write(int address, byte data)
00154 {
00155     Processor.IncrementCycleCount();
00156     WriteWithoutCycle(address, data);
00157 }
00158
00159 /// <summary>
00160 /// Writes data to the given address without incrementing the cycle count.
00161 /// </summary>
00162 /// <param name="address">The address to write data to.</param>
00163 /// <param name="data">The data to write.</param>
00164 public static void WriteWithoutCycle(int address, byte data)
00165 {
00166     if ((ACIA.Offset <= address) && (address <= (ACIA.Offset + ACIA.Length)))
00167     {
00168         ACIA.Write(address, data);
00169     }
00170     else if ((GPIO.Offset <= address) && (address <= (GPIO.Offset + GPIO.Length)))
00171     {
00172         GPIO.Write(address, data);
00173     }
00174     else if ((SharedROM.Offset <= address) && (address <= (SharedROM.Offset +
SharedROM.Length)))
00175     {
00176         SharedROM.Write(address, data);
00177     }
00178     else if ((BankedROM.Offset <= address) && (address <= (BankedROM.Offset +
BankedROM.Length)))
00179     {
00180         BankedROM.Write(address, data);

```



```

00181         }
00182         else if ((BankedRAM.Offset <= address) && (address <= (BankedRAM.Offset +
BankedRAM.Length)))
00183         {
00184             BankedRAM.Write(address, data);
00185         }
00186         else
00187         {
00188             throw new ApplicationException(String.Format("Cannot write to address: {0}",
address));
00189         }
00190     }
00191 }
00192 }

```

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.](#)

```

00001 using System.ComponentModel;
00002
00003 namespace Hardware
00004 {
00005     public static class Utility
00006     {
00007         public static string ConvertOpCodeIntoString(this int i)
00008         {
00009             switch (i)
00010             {
00011                 case 0x69: //ãADCãImmediate
00012                 case 0x65: //ãADCãZeroãPage
00013                 case 0x75: //ãADCãZeroãPageãX
00014                 case 0x6D: //ãADCãAbsolute
00015                 case 0x7D: //ãADCãAbsoluteãX
00016                 case 0x79: //ãADCãAbsoluteãY
00017                 case 0x61: //ãADCãIndirectãX
00018                 case 0x71: //ãADCãIndirectãY
00019                 {
00020                     return "ADC";
00021                 }
00022                 case 0x29: //ãANDãImmediate
00023                 case 0x25: //ãANDãZeroãPage
00024                 case 0x35: //ãANDãZeroãPageãX
00025                 case 0x2D: //ãANDãAbsolute
00026                 case 0x3D: //ãANDãAbsoluteãX
00027                 case 0x39: //ãANDãAbsoluteãY
00028                 case 0x21: //ãANDãIndirectãX
00029                 case 0x31: //ãANDãIndirectãY
00030                 {
00031                     return "AND";
00032                 }
00033                 case 0x0A: //ãASLãAccumulator
00034                 case 0x06: //ãASLãZeroãPage
00035                 case 0x16: //ãASLãZeroãPageãX
00036                 case 0x0E: //ãASLãAbsolute
00037                 case 0x1E: //ãASLãAbsoluteãX
00038                 {
00039                     return "ASL";
00040                 }
00041                 case 0x90: //ãBCCãRelative
00042                 {
00043                     return "BCC";
00044                 }
00045                 case 0xB0: //ãBCSãRelative

```

```

00046         {
00047             return "BCS";
00048         }
00049     case 0xF0: // ãBEQãRelative
00050     {
00051         return "BEQ";
00052     }
00053     case 0x24: // ãBITãZeroãPage
00054     case 0x2C: // ãBITãAbsolute
00055     {
00056         return "BIT";
00057     }
00058     case 0x30: // ãBMIãRelative
00059     {
00060         return "BMI";
00061     }
00062     case 0xD0: // ãBNEãRelative
00063     {
00064         return "BNE";
00065     }
00066     case 0x10: // ãBPLãRelative
00067     {
00068         return "BPL";
00069     }
00070     case 0x00: // ãBRKãImplied
00071     {
00072         return "BRK";
00073     }
00074     case 0x50: // BVC Relative
00075     {
00076         return "BCV";
00077     }
00078     case 0x70: // BVS Relative
00079     {
00080         return "BVS";
00081     }
00082     case 0x18: // ãCLCãImplied
00083     {
00084         return "CLC";
00085     }
00086     case 0xD8: // ãCLDãImplied
00087     {
00088         return "CLD";
00089     }
00090     case 0x58: // ãCLIãImplied
00091     {
00092         return "CLI";
00093     }
00094     case 0xB8: // ãCLVãImplied
00095     {
00096         return "CLV";
00097     }
00098     case 0xC9: // ãCMPãImmediate
00099     case 0xC5: // ãCMPãZeroPage
00100     case 0xD5: // ãCMPãZeroãPageãX
00101     case 0xCD: // ãCMPãAbsolute
00102     case 0xDD: // ãCMPãAbsoluteãX
00103     case 0xD9: // ãCMPãAbsoluteãY
00104     case 0xC1: // ãCMPãIndirectãX
00105     case 0xD1: // ãCMPãIndirectãY
00106     {
00107         return "CMP";
00108     }
00109     case 0xE0: // ãCPXãImmediate
00110     case 0xE4: // ãCPXãZeroPage
00111     case 0xEC: // ãCPXãAbsolute
00112     {
00113         return "CPX";
00114     }
00115     case 0xC0: // ãCPYãImmediate
00116     case 0xC4: // ãCPYãZeroPage
00117     case 0xCC: // ãCPYãAbsolute
00118     {
00119         return "CPY";
00120     }
00121     case 0xC6: // ãDECãZeroãPage
00122     case 0xD6: // ãDECãZeroãPageãX
00123     case 0xCE: // ãDECãAbsolute
00124     case 0xDE: // ãDECãAbsoluteãX
00125     {
00126         return "DEC";
00127     }
00128     case 0xCA: // ãDEXãImplied
00129     {
00130         return "DEX";
00131     }
00132     case 0x88: // ãDEYãImplied

```

```

00133         {
00134             return "DEY";
00135         }
00136     case 0x49: //äEORäImmediate
00137     case 0x45: //äEORäZeroäPage
00138     case 0x55: //äEORäZeroäPageäX
00139     case 0x4D: //äEORäAbsolute
00140     case 0x5D: //äEORäAbsoluteäX
00141     case 0x59: //äEORäAbsoluteäY
00142     case 0x41: //äEORäIndirectäX
00143     case 0x51: //äEORäIndirectäY
00144         {
00145             return "EOR";
00146         }
00147     case 0xE6: //äINCäZeroäPage
00148     case 0xF6: //äINCäZeroäPageäX
00149         {
00150             return "INC";
00151         }
00152     case 0xE8: //äINXäImplied
00153         {
00154             return "INX";
00155         }
00156     case 0xC8: //äINYäImplied
00157         {
00158             return "INY";
00159         }
00160     case 0xEE: //äINCäAbsolute
00161     case 0xFE: //äINCäAbsoluteäX
00162         {
00163             return "INC";
00164         }
00165     case 0x4C: //äJMPäAbsolute
00166     case 0x6C: //äJMPäIndirect
00167         {
00168             return "JMP";
00169         }
00170     case 0x20: //äJSRäAbsolute
00171         {
00172             return "JSR";
00173         }
00174     case 0xA9: //äLDAäImmediate
00175     case 0xA5: //äLDAäZeroäPage
00176     case 0xB5: //äLDAäZeroäPageäX
00177     case 0xAD: //äLDAäAbsolute
00178     case 0xBD: //äLDAäAbsoluteäX
00179     case 0xB9: //äLDAäAbsoluteäY
00180     case 0xA1: //äLDAäIndirectäX
00181     case 0xB1: //äLDAäIndirectäY
00182         {
00183             return "LDA";
00184         }
00185     case 0xA2: //äLDXäImmediate
00186     case 0xA6: //äLDXäZeroäPage
00187     case 0xB6: //äLDXäZeroäPageäY
00188     case 0xAE: //äLDXäAbsolute
00189     case 0xBE: //äLDXäAbsoluteäY
00190         {
00191             return "LDX";
00192         }
00193     case 0xA0: //äLDYäImmediate
00194     case 0xA4: //äLDYäZeroäPage
00195     case 0xB4: //äLDYäZeroäPageäY
00196     case 0xAC: //äLDYäAbsolute
00197     case 0xBC: //äLDYäAbsoluteäY
00198         {
00199             return "LDY";
00200         }
00201     case 0x4A: //äLSRäAccumulator
00202     case 0x46: //äLSRäZeroäPage
00203     case 0x56: //äLSRäZeroäPageäX
00204     case 0x4E: //äLSRäAbsolute
00205     case 0x5E: //äLSRäAbsoluteäX
00206         {
00207             return "LSR";
00208         }
00209     case 0xEA: //äNOPäImplied
00210         {
00211             return "NOP";
00212         }
00213     case 0x09: //äORAäImmediate
00214     case 0x05: //äORAäZeroäPage
00215     case 0x15: //äORAäZeroäPageäX
00216     case 0x0D: //äORAäAbsolute
00217     case 0x1D: //äORAäAbsoluteäX
00218     case 0x19: //äORAäAbsoluteäY
00219     case 0x01: //äORAäIndirectäX

```

```

00220         case 0x11: //äORAäIndirectäY
00221         {
00222             return "ORA";
00223         }
00224         case 0x48: //äPHAäImplied
00225         {
00226             return "PHA";
00227         }
00228         case 0x08: //äPHPäImplied
00229         {
00230             return "PHP";
00231         }
00232         case 0x68: //äPLAäImplied
00233         {
00234             return "PLA";
00235         }
00236         case 0x28: //äPLPäImplied
00237         {
00238             return "PLP";
00239         }
00240         case 0x2A: //äROLäAccumulator
00241         case 0x26: //äROLäZeroäPage
00242         case 0x36: //äROLäZeroäPageäX
00243         case 0x2E: //äROLäAbsolute
00244         case 0x3E: //äROLäAbsoluteäX
00245         {
00246             return "ROL";
00247         }
00248         case 0x6A: //äRORäAccumulator
00249         case 0x66: //äRORäZeroäPage
00250         case 0x76: //äRORäZeroäPageäX
00251         case 0x6E: //äRORäAbsolute
00252         case 0x7E: //äRORäAbsoluteäX
00253         {
00254             return "ROR";
00255         }
00256         case 0x40: //äRTIäImplied
00257         {
00258             return "RTI";
00259         }
00260         case 0x60: //äRTSäImplied
00261         {
00262             return "RTS";
00263         }
00264         case 0xE9: //äSBCäImmediate
00265         case 0xE5: //äSBCäZeroäPage
00266         case 0xF5: //äSBCäZeroäPageäX
00267         case 0xED: //äSBCäAbsolute
00268         case 0xFD: //äSBCäAbsoluteäX
00269         case 0xF9: //äSBCäAbsoluteäY
00270         case 0xE1: //äSBCäIndirectäX
00271         case 0xF1: //äSBCäIndirectäY
00272         {
00273             return "SBC";
00274         }
00275         case 0x38: //äSECäImplied
00276         {
00277             return "SEC";
00278         }
00279         case 0xF8: //äSEDäImplied
00280         {
00281             return "SED";
00282         }
00283         case 0x78: //äSEIäImplied
00284         {
00285             return "SEI";
00286         }
00287         case 0x85: //äSTAäZeroPage
00288         case 0x95: //äSTAäZeroäPageäX
00289         case 0x8D: //äSTAäAbsolute
00290         case 0x9D: //äSTAäAbsoluteäX
00291         case 0x99: //äSTAäAbsoluteäY
00292         case 0x81: //äSTAäIndirectäX
00293         case 0x91: //äSTAäIndirectäY
00294         {
00295             return "STA";
00296         }
00297         case 0x86: //äSTXäZeroäPage
00298         case 0x96: //äSTXäZeroäPageäY
00299         case 0x8E: //äSTXäAbsolute
00300         {
00301             return "STX";
00302         }
00303         case 0x84: //äSTYäZeroäPage
00304         case 0x94: //äSTYäZeroäPageäX
00305         case 0x8C: //äSTYäAbsolute
00306         {

```

```

00307         return "STY";
00308     }
00309     case 0xAA: //äTAXäImplied
00310     {
00311         return "TAX";
00312     }
00313     case 0xA8: //äTAYäImplied
00314     {
00315         return "TAY";
00316     }
00317     case 0xBA: //äTSXäImplied
00318     {
00319         return "TSX";
00320     }
00321     case 0x8A: //äTXAäImplied
00322     {
00323         return "TXA";
00324     }
00325     case 0x9A: //äTXSäImplied
00326     {
00327         return "TXS";
00328     }
00329     case 0x98: //äTYAäImplied
00330     {
00331         return "TYA";
00332     }
00333     default:
00334         throw new InvalidEnumArgumentException(string.Format("A Valid Conversion does not
exist for OpCode {0}", i.ToString("X")));
00335
00336     }
00337 }
00338 }
00339 }

```

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.](#)

```

00001 using System;
00002
00003 namespace Hardware
00004 {
00005     public class HM62256
00006     {
00007         /// <summary>
00008         /// The memory area.
00009         /// </summary>
00010         public byte[][] Memory { get; set; }
00011
00012         /// <summary>
00013         /// The memory offset.
00014         /// </summary>
00015         public int Offset { get; set; }
00016
00017         /// <summary>
00018         /// The memory length.
00019         /// </summary>
00020         public int Length { get; set; }
00021
00022         /// <summary>
00023         /// The location of the end of memory.
00024         /// </summary>
00025         public int End { get { return Offset + Length; } }

```

```

00026
00027 /// <summary>
00028 /// The number of banks the memory has.
00029 /// </summary>
00030 public byte Banks { get; set; }
00031
00032 /// <summary>
00033 /// The currently selected bank.
00034 /// </summary>
00035 public byte CurrentBank { get; set; }
00036
00037 /// <summary>
00038 /// Called whenever a new 62256 object is required.
00039 /// </summary>
00040 /// <param name="banks">Number of banks the new memory will have.</param>
00041 /// <param name="offset">Offset of the new memory in the address space.</param>
00042 /// <param name="length">Length of each bank of memory.</param>
00043 public HM62256(byte banks, int offset, int length)
00044 {
00045     Memory = new byte[banks][];
00046     for (int i = 0; i < banks; i++)
00047     {
00048         Memory[i] = new byte[length + 1];
00049     }
00050     Length = length;
00051     Banks = banks;
00052     Offset = offset;
00053     CurrentBank = 0;
00054 }
00055
00056 /// <summary>
00057 /// Called whenever the emulated computer is reset.
00058 /// </summary>
00059 public void Reset()
00060 {
00061     Clear();
00062 }
00063
00064 /// <summary>
00065 /// Clears the memory.
00066 /// </summary>
00067 public void Clear()
00068 {
00069     for (var i = 0; i < Banks; i++)
00070     {
00071         for (var j = 0; j < Memory.Length; j++)
00072         {
00073             Memory[i][j] = 0x00;
00074         }
00075     }
00076 }
00077
00078 /// <summary>
00079 /// Returns the byte at a given address without incrementing the cycle. Useful for test harness.
00080 /// </summary>
00081 /// <param name="bank">The bank to read data from.</param>
00082 /// <param name="address"></param>
00083 /// <returns>The byte being read.</returns>
00084 public byte Read(int address)
00085 {
00086     return Memory[CurrentBank][address - Offset];
00087 }
00088
00089 /// <summary>
00090 /// Writes data to the given address without incrementing the cycle.
00091 /// </summary>
00092 /// <param name="bank">The bank to load data to.</param>
00093 /// <param name="address">The address to write data to</param>
00094 /// <param name="data">The data to write</param>
00095 public void Write(int address, byte data)
00096 {
00097     Memory[CurrentBank][address - Offset] = data;
00098 }
00099
00100 /// <summary>
00101 /// Dumps the entire memory object. Used when saving the memory state
00102 /// </summary>
00103 /// <returns>Jagged array representing the banked memory.</returns>
00104 public byte[][] DumpMemory()
00105 {
00106     return Memory;
00107 }
00108 }
00109 }

```

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.](#)

```
00001 using NLog;
00002 using System;
00003 using System.ComponentModel;
00004 using System.Diagnostics;
00005 using System.Globalization;
00006
00007 namespace Hardware
00008 {
00009     /// <summary>
00010     /// An implementation of a W65C02 Processor.
00011     /// </summary>
00012     [Serializable]
00013     public class W65C02
00014     {
00015         #region Fields
00016         private readonly ILogger _logger = LogManager.GetLogger("Processor");
00017         private int _programCounter;
00018         private int _stackPointer;
00019         private int _cycleCount;
00020         private bool _previousInterrupt;
00021         private bool _interrupt;
00022
00023         /// <summary>
00024         /// Checks shether the emulated computer is running or not.
00025         /// </summary>
00026         public bool isRunning;
00027     #endregion
00028
00029     #region Properties
00030     /// <summary>
00031     /// The Accumulator. This value is implemented as an integer intead of a byte.
00032     /// This is done so we can detect wrapping of the value and set the correct number of cycles.
00033     /// </summary>
00034     public int Accumulator { get; protected set; }
00035
00036     /// <summary>
00037     /// The X Index Register
00038     /// </summary>
00039     public int XRegister { get; private set; }
00040
00041     /// <summary>
00042     /// The Y Index Register
00043     /// </summary>
00044     public int YRegister { get; private set; }
00045
00046     /// <summary>
00047     /// The Current Op Code being executed by the system
00048     /// </summary>
00049     public int CurrentOpCode { get; private set; }
00050
00051     /// <summary>
00052     /// The disassembly of the current operation. This value is only set when the CPU is built in debug
00053     /// mode.
00054     /// </summary>
00054     public Disassembly CurrentDisassembly { get; private set; }
00055
00056     /// <summary>
00057     /// Points to the Current Address of the instruction being executed by the system.
00058     /// The PC wraps when the value is greater than 65535, or less than 0.
00059     /// </summary>
00060     public int ProgramCounter
```

```

00061     {
00062         get { return _programCounter; }
00063         private set { _programCounter = WrapProgramCounter(value); }
00064     }
00065
00066     /// <summary>
00067     /// Points to the Current Position of the Stack.
00068     /// This value is a 00-FF value but is offset to point to the location in memory where the stack
00069     /// resides.
00070     /// </summary>
00071     public int StackPointer
00072     {
00073         get { return _stackPointer; }
00074         private set
00075         {
00076             if (value > 0xFF)
00077                 _stackPointer = value - 0x100;
00078             else if (value < 0x00)
00079                 _stackPointer = value + 0x100;
00080             else
00081                 _stackPointer = value;
00082         }
00083     }
00084     /// <summary>
00085     /// An external action that occurs when the cycle count is incremented
00086     /// </summary>
00087     public Action CycleCountIncrementedAction { get; set; }
00088
00089     //Status Registers
00090     /// <summary>
00091     /// This is the carry flag. when adding, if the result is greater than 255 or 99 in BCD Mode, then
00092     /// this bit is enabled.
00093     /// In subtraction this is reversed and set to false if a borrow is required IE the result is less
00094     /// than 0
00095     /// </summary>
00096     public bool CarryFlag { get; protected set; }
00097
00098     /// <summary>
00099     /// Is true if one of the registers is set to zero.
00100     /// </summary>
00101     public bool ZeroFlag { get; private set; }
00102
00103     /// <summary>
00104     /// This determines if Interrupts are currently disabled.
00105     /// This flag is turned on during a reset to prevent an interrupt from occurring during
00106     /// startup/initialization.
00107     /// If this flag is true, then the IRQ pin is ignored.
00108     /// </summary>
00109     public bool DisableInterruptFlag { get; private set; }
00110
00111     /// <summary>
00112     /// Binary Coded Decimal Mode is set/cleared via this flag.
00113     /// when this mode is in effect, a byte represents a number from 0-99.
00114     /// </summary>
00115     public bool DecimalFlag { get; private set; }
00116
00117     /// <summary>
00118     /// This property is set when an overflow occurs. An overflow happens if the high bit(7) changes
00119     /// during the operation. Remember that values from 128-256 are negative values
00120     /// as the high bit is set to 1.
00121     /// Examples:
00122     /// 64 + 64 = -128
00123     /// -128 + -128 = 0
00124     /// </summary>
00125     public bool OverflowFlag { get; protected set; }
00126
00127     /// <summary>
00128     /// Set to true if the result of an operation is negative in ADC and SBC operations.
00129     /// Remember that 128-256 represent negative numbers when doing signed math.
00130     /// In shift operations the sign holds the carry.
00131     /// </summary>
00132     public bool NegativeFlag { get; private set; }
00133
00134     /// <summary>
00135     /// Set to true when an NMI should occur
00136     /// </summary>
00137     public bool TriggerNmi { get; set; }
00138
00139     /// <summary>
00140     /// Set to true when an IRQ has occurred and is being processed by the CPU
00141     /// </summary>
00142     public bool TriggerIRQ { get; private set; }
00143 #endregion
00144 #region Public Methods
00145     /// <summary>
00146     /// Default Constructor, Instantiates a new instance of the processor.
00147     /// </summary>

```



```

00143     public W65C02()
00144     {
00145         StackPointer = 0x100;
00146         CycleCountIncrementedAction = () => { };
00147     }
00148
00149     /// <summary>
00150     /// Initializes the processor to its default state.
00151     /// </summary>
00152     public void Reset()
00153     {
00154         ResetCycleCount();
00155         StackPointer = 0x1FD;
00156         //Set the Program Counter to the Reset Vector Address.
00157         ProgramCounter = 0xFFFC;
00158         //Reset the Program Counter to the Address contained in the Reset Vector
00159         ProgramCounter = (MemoryMap.Read(ProgramCounter) | (MemoryMap.Read(ProgramCounter + 1) <
8));
00160         CurrentOpCode = MemoryMap.Read(ProgramCounter);
00161         //SetDisassembly();
00162         DisableInterruptFlag = true;
00163         _previousInterrupt = false;
00164         TriggerNmi = false;
00165         TriggerIRQ = false;
00166     }
00167
00168     /// <summary>
00169     /// Performs the next step on the processor
00170     /// </summary>
00171     public void NextStep()
00172     {
00173         SetDisassembly();
00174
00175         //Have to read this first otherwise it causes tests to fail on a NES
00176         CurrentOpCode = MemoryMap.Read(ProgramCounter);
00177
00178         ProgramCounter++;
00179
00180         ExecuteOpCode();
00181
00182         if (_previousInterrupt)
00183         {
00184             if (TriggerNmi)
00185             {
00186                 ProcessNMI();
00187                 TriggerNmi = false;
00188             }
00189             else if (TriggerIRQ)
00190             {
00191                 ProcessIRQ();
00192                 TriggerIRQ = false;
00193             }
00194         }
00195     }
00196
00197     /// <summary>
00198     /// The InterruptRequest or IRQ
00199     /// </summary>
00200     public void InterruptRequest()
00201     {
00202         TriggerIRQ = true;
00203     }
00204
00205     /// <summary>
00206     /// Gets the Number of Cycles that have elapsed
00207     /// </summary>
00208     /// <returns>The number of elapsed cycles</returns>
00209     public int GetCycleCount()
00210     {
00211         return _cycleCount;
00212     }
00213
00214     /// <summary>
00215     /// Increments the Cycle Count, causes a CycleCountIncrementedAction to fire.
00216     /// </summary>
00217     public void IncrementCycleCount()
00218     {
00219         _cycleCount++;
00220         CycleCountIncrementedAction();
00221
00222         _previousInterrupt = _interrupt;
00223         _interrupt = TriggerNmi || (TriggerIRQ && !DisableInterruptFlag);
00224     }
00225
00226     /// <summary>
00227     /// Resets the Cycle Count back to 0
00228     /// </summary>

```

```

00229         public void ResetCycleCount ()
00230         {
00231             _cycleCount = 0;
00232         }
00233 #endregion
00234
00235 #region Private Methods
00236 /// <summary>
00237 /// Executes an Opcode
00238 /// </summary>
00239 private void ExecuteOpCode ()
00240 {
00241     //The x+ cycles denotes that if a page wrap occurs, then an additional cycle is consumed.
00242     //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./
00243     //This is handled inside the GetValueFromMemory Method
00244     switch (CurrentOpCode)
00245     {
00246 #region Add / Subtract Operations
00247         //ADC Add With Carry, Immediate, 2 Bytes, 2 Cycles
00248         case 0x69:
00249         {
00250             AddWithCarryOperation(AddressingMode.Immediate);
00251             break;
00252         }
00253         //ADC Add With Carry, Zero Page, 2 Bytes, 3 Cycles
00254         case 0x65:
00255         {
00256             AddWithCarryOperation(AddressingMode.ZeroPage);
00257             break;
00258         }
00259         //ADC Add With Carry, Zero Page X, 2 Bytes, 4 Cycles
00260         case 0x75:
00261         {
00262             AddWithCarryOperation(AddressingMode.ZeroPageX);
00263             break;
00264         }
00265         //ADC Add With Carry, Absolute, 3 Bytes, 4 Cycles
00266         case 0x6D:
00267         {
00268             AddWithCarryOperation(AddressingMode.Absolute);
00269             break;
00270         }
00271         //ADC Add With Carry, Absolute X, 3 Bytes, 4+ Cycles
00272         case 0x7D:
00273         {
00274             AddWithCarryOperation(AddressingMode.AbsoluteX);
00275             break;
00276         }
00277         //ADC Add With Carry, Absolute Y, 3 Bytes, 4+ Cycles
00278         case 0x79:
00279         {
00280             AddWithCarryOperation(AddressingMode.AbsoluteY);
00281             break;
00282         }
00283         //ADC Add With Carry, Indexed Indirect, 2 Bytes, 6 Cycles
00284         case 0x61:
00285         {
00286             AddWithCarryOperation(AddressingMode.IndirectX);
00287             break;
00288         }
00289         //ADC Add With Carry, Indexed Indirect, 2 Bytes, 5+ Cycles
00290         case 0x71:
00291         {
00292             AddWithCarryOperation(AddressingMode.IndirectY);
00293             break;
00294         }
00295         //SBC Subtract with Borrow, Immediate, 2 Bytes, 2 Cycles
00296         case 0xE9:
00297         {
00298             SubtractWithBorrowOperation(AddressingMode.Immediate);
00299             break;
00300         }
00301         //SBC Subtract with Borrow, Zero Page, 2 Bytes, 3 Cycles
00302         case 0xE5:
00303         {
00304             SubtractWithBorrowOperation(AddressingMode.ZeroPage);
00305             break;
00306         }
00307         //SBC Subtract with Borrow, Zero Page X, 2 Bytes, 4 Cycles
00308         case 0xF5:
00309         {
00310             SubtractWithBorrowOperation(AddressingMode.ZeroPageX);
00311             break;
00312         }
00313         //SBC Subtract with Borrow, Absolute, 3 Bytes, 4 Cycles
00314         case 0xED:

```

```

00315         {
00316             SubtractWithBorrowOperation(AddressingMode.Absolute);
00317             break;
00318         }
00319         //SBC Subtract with Borrow, Absolute X, 3 Bytes, 4+ Cycles
00320         case 0xFD:
00321         {
00322             SubtractWithBorrowOperation(AddressingMode.AbsoluteX);
00323             break;
00324         }
00325         //SBC Subtract with Borrow, Absolute Y, 3 Bytes, 4+ Cycles
00326         case 0xF9:
00327         {
00328             SubtractWithBorrowOperation(AddressingMode.AbsoluteY);
00329             break;
00330         }
00331         //SBC Subtract with Borrow, Indexed Indirect, 2 Bytes, 6 Cycles
00332         case 0xE1:
00333         {
00334             SubtractWithBorrowOperation(AddressingMode.IndirectX);
00335             break;
00336         }
00337         //SBC Subtract with Borrow, Indexed Indirect, 2 Bytes, 5+ Cycles
00338         case 0xF1:
00339         {
00340             SubtractWithBorrowOperation(AddressingMode.IndirectY);
00341             break;
00342         }
00343     #endregion
00344
00345     #region Branch Operations
00346         //BCC Branch if Carry is Clear, Relative, 2 Bytes, 2++ Cycles
00347         case 0x90:
00348         {
00349             BranchOperation(!CarryFlag);
00350             break;
00351         }
00352     }
00353     //BCS Branch if Carry is Set, Relative, 2 Bytes, 2++ Cycles
00354     case 0xB0:
00355     {
00356         BranchOperation(CarryFlag);
00357         break;
00358     }
00359     //BEQ Branch if Zero is Set, Relative, 2 Bytes, 2++ Cycles
00360     case 0xF0:
00361     {
00362         BranchOperation(ZeroFlag);
00363         break;
00364     }
00365
00366     // BMI Branch if Negative Set
00367     case 0x30:
00368     {
00369         BranchOperation(NegativeFlag);
00370         break;
00371     }
00372     //BNE Branch if Zero is Not Set, Relative, 2 Bytes, 2++ Cycles
00373     case 0xD0:
00374     {
00375         BranchOperation(!ZeroFlag);
00376         break;
00377     }
00378     // BPL Branch if Negative Clear, 2 Bytes, 2++ Cycles
00379     case 0x10:
00380     {
00381         BranchOperation(!NegativeFlag);
00382         break;
00383     }
00384     // BVC Branch if Overflow Clear, 2 Bytes, 2++ Cycles
00385     case 0x50:
00386     {
00387         BranchOperation(!OverflowFlag);
00388         break;
00389     }
00390     // BVS Branch if Overflow Set, 2 Bytes, 2++ Cycles
00391     case 0x70:
00392     {
00393         BranchOperation(OverflowFlag);
00394         break;
00395     }
00396 #endregion
00397
00398 #region BitWise Comparison Operations
00399     //AND Compare Memory with Accumulator, Immediate, 2 Bytes, 2 Cycles
00400     case 0x29:
00401     {

```

```

00402         AndOperation(AddressingMode.Immediate);
00403         break;
00404     }
00405     //AND Compare Memory with Accumulator, Zero Page, 2 Bytes, 3 Cycles
00406     case 0x25:
00407     {
00408         AndOperation(AddressingMode.ZeroPage);
00409         break;
00410     }
00411     //AND Compare Memory with Accumulator, Zero PageX, 2 Bytes, 3 Cycles
00412     case 0x35:
00413     {
00414         AndOperation(AddressingMode.ZeroPageX);
00415         break;
00416     }
00417     //AND Compare Memory with Accumulator, Absolute, 3 Bytes, 4 Cycles
00418     case 0x2D:
00419     {
00420         AndOperation(AddressingMode.Absolute);
00421         break;
00422     }
00423     //AND Compare Memory with Accumulator, AbsoluteX 3 Bytes, 4+ Cycles
00424     case 0x3D:
00425     {
00426         AndOperation(AddressingMode.AbsoluteX);
00427         break;
00428     }
00429     //AND Compare Memory with Accumulator, AbsoluteY, 3 Bytes, 4+ Cycles
00430     case 0x39:
00431     {
00432         AndOperation(AddressingMode.AbsoluteY);
00433         break;
00434     }
00435     //AND Compare Memory with Accumulator, IndexedIndirect, 2 Bytes, 6 Cycles
00436     case 0x21:
00437     {
00438         AndOperation(AddressingMode.IndirectX);
00439         break;
00440     }
00441     //AND Compare Memory with Accumulator, IndirectIndexed, 2 Bytes, 5 Cycles
00442     case 0x31:
00443     {
00444         AndOperation(AddressingMode.IndirectY);
00445         break;
00446     }
00447     //BIT Compare Memory with Accumulator, Zero Page, 2 Bytes, 3 Cycles
00448     case 0x24:
00449     {
00450         BitOperation(AddressingMode.ZeroPage);
00451         break;
00452     }
00453     //BIT Compare Memory with Accumulator, Absolute, 2 Bytes, 4 Cycles
00454     case 0x2C:
00455     {
00456         BitOperation(AddressingMode.Absolute);
00457         break;
00458     }
00459     //EOR Exclusive OR Memory with Accumulator, Immediate, 2 Bytes, 2 Cycles
00460     case 0x49:
00461     {
00462         EorOperation(AddressingMode.Immediate);
00463         break;
00464     }
00465     //EOR Exclusive OR Memory with Accumulator, Zero Page, 2 Bytes, 3 Cycles
00466     case 0x45:
00467     {
00468         EorOperation(AddressingMode.ZeroPage);
00469         break;
00470     }
00471     //EOR Exclusive OR Memory with Accumulator, Zero Page X, 2 Bytes, 4 Cycles
00472     case 0x55:
00473     {
00474         EorOperation(AddressingMode.ZeroPageX);
00475         break;
00476     }
00477     //EOR Exclusive OR Memory with Accumulator, Absolute, 3 Bytes, 4 Cycles
00478     case 0x4D:
00479     {
00480         EorOperation(AddressingMode.Absolute);
00481         break;
00482     }
00483     //EOR Exclusive OR Memory with Accumulator, Absolute X, 3 Bytes, 4+ Cycles
00484     case 0x5D:
00485     {
00486         EorOperation(AddressingMode.AbsoluteX);
00487         break;
00488     }

```

```

00489 //EOR Exclusive OR Memory with Accumulator, Absolute Y, 3 Bytes, 4+ Cycles
00490 case 0x59:
00491 {
00492     EorOperation(AddressingMode.AbsoluteY);
00493     break;
00494 }
00495 //EOR Exclusive OR Memory with Accumulator, IndexedIndirect, 2 Bytes 6 Cycles
00496 case 0x41:
00497 {
00498     EorOperation(AddressingMode.IndirectX);
00499     break;
00500 }
00501 //EOR Exclusive OR Memory with Accumulator, IndirectIndexed, 2 Bytes 5 Cycles
00502 case 0x51:
00503 {
00504     EorOperation(AddressingMode.IndirectY);
00505     break;
00506 }
00507 //ORA Compare Memory with Accumulator, Immediate, 2 Bytes, 2 Cycles
00508 case 0x09:
00509 {
00510     OrOperation(AddressingMode.Immediate);
00511     break;
00512 }
00513 //ORA Compare Memory with Accumulator, Zero Page, 2 Bytes, 2 Cycles
00514 case 0x05:
00515 {
00516     OrOperation(AddressingMode.ZeroPage);
00517     break;
00518 }
00519 //ORA Compare Memory with Accumulator, Zero PageX, 2 Bytes, 4 Cycles
00520 case 0x15:
00521 {
00522     OrOperation(AddressingMode.ZeroPageX);
00523     break;
00524 }
00525 //ORA Compare Memory with Accumulator, Absolute, 3 Bytes, 4 Cycles
00526 case 0x0D:
00527 {
00528     OrOperation(AddressingMode.Absolute);
00529     break;
00530 }
00531 //ORA Compare Memory with Accumulator, AbsoluteX 3 Bytes, 4+ Cycles
00532 case 0x1D:
00533 {
00534     OrOperation(AddressingMode.AbsoluteX);
00535     break;
00536 }
00537 //ORA Compare Memory with Accumulator, AbsoluteY, 3 Bytes, 4+ Cycles
00538 case 0x19:
00539 {
00540     OrOperation(AddressingMode.AbsoluteY);
00541     break;
00542 }
00543 //ORA Compare Memory with Accumulator, IndexedIndirect, 2 Bytes, 6 Cycles
00544 case 0x01:
00545 {
00546     OrOperation(AddressingMode.IndirectX);
00547     break;
00548 }
00549 //ORA Compare Memory with Accumulator, IndirectIndexed, 2 Bytes, 5 Cycles
00550 case 0x11:
00551 {
00552     OrOperation(AddressingMode.IndirectY);
00553     break;
00554 }
00555 #endregion
00556
00557 #region Clear Flag Operations
00558 //CLC Clear Carry Flag, Implied, 1 Byte, 2 Cycles
00559 case 0x18:
00560 {
00561     CarryFlag = false;
00562     IncrementCycleCount();
00563     break;
00564 }
00565 //CLD Clear Decimal Flag, Implied, 1 Byte, 2 Cycles
00566 case 0xD8:
00567 {
00568     DecimalFlag = false;
00569     IncrementCycleCount();
00570     break;
00571 }
00572
00573 //CLI Clear Interrupt Flag, Implied, 1 Byte, 2 Cycles
00574 case 0x58:
00575 {

```

```

00576             DisableInterruptFlag = false;
00577             IncrementCycleCount();
00578             break;
00579         }
00580     }
00581     //CLV Clear Overflow Flag, Implied, 1 Byte, 2 Cycles
00582     case 0xB8:
00583     {
00584         OverflowFlag = false;
00585         IncrementCycleCount();
00586         break;
00587     }
00588
00589 #endregion
00590
00591 #region Compare Operations
00592 //CMP Compare Accumulator with Memory, Immediate, 2 Bytes, 2 Cycles
00593 case 0xC9:
00594 {
00595     CompareOperation(AddressingMode.Immediate, Accumulator);
00596     break;
00597 }
00598 //CMP Compare Accumulator with Memory, Zero Page, 2 Bytes, 3 Cycles
00599 case 0xC5:
00600 {
00601     CompareOperation(AddressingMode.ZeroPage, Accumulator);
00602     break;
00603 }
00604 //CMP Compare Accumulator with Memory, Zero Page x, 2 Bytes, 4 Cycles
00605 case 0xD5:
00606 {
00607     CompareOperation(AddressingMode.ZeroPageX, Accumulator);
00608     break;
00609 }
00610 //CMP Compare Accumulator with Memory, Absolute, 3 Bytes, 4 Cycles
00611 case 0xCD:
00612 {
00613     CompareOperation(AddressingMode.Absolute, Accumulator);
00614     break;
00615 }
00616 //CMP Compare Accumulator with Memory, Absolute X, 2 Bytes, 4 Cycles
00617 case 0xDD:
00618 {
00619     CompareOperation(AddressingMode.AbsoluteX, Accumulator);
00620     break;
00621 }
00622 //CMP Compare Accumulator with Memory, Absolute Y, 2 Bytes, 4 Cycles
00623 case 0xD9:
00624 {
00625     CompareOperation(AddressingMode.AbsoluteY, Accumulator);
00626     break;
00627 }
00628 //CMP Compare Accumulator with Memory, Indirect X, 2 Bytes, 6 Cycles
00629 case 0xC1:
00630 {
00631     CompareOperation(AddressingMode.IndirectX, Accumulator);
00632     break;
00633 }
00634 //CMP Compare Accumulator with Memory, Indirect Y, 2 Bytes, 5 Cycles
00635 case 0xD1:
00636 {
00637     CompareOperation(AddressingMode.IndirectY, Accumulator);
00638     break;
00639 }
00640 //CPX Compare Accumulator with X Register, Immediate, 2 Bytes, 2 Cycles
00641 case 0xE0:
00642 {
00643     CompareOperation(AddressingMode.Immediate, XRegister);
00644     break;
00645 }
00646 //CPX Compare Accumulator with X Register, Zero Page, 2 Bytes, 3 Cycles
00647 case 0xE4:
00648 {
00649     CompareOperation(AddressingMode.ZeroPage, XRegister);
00650     break;
00651 }
00652 //CPX Compare Accumulator with X Register, Absolute, 3 Bytes, 4 Cycles
00653 case 0xEC:
00654 {
00655     CompareOperation(AddressingMode.Absolute, XRegister);
00656     break;
00657 }
00658 //CPY Compare Accumulator with Y Register, Immediate, 2 Bytes, 2 Cycles
00659 case 0xC0:
00660 {
00661     CompareOperation(AddressingMode.Immediate, YRegister);
00662     break;

```

```

00663     }
00664     //CPY Compare Accumulator with Y Register, Zero Page, 2 Bytes, 3 Cycles
00665     case 0xC4:
00666     {
00667         CompareOperation(AddressingMode.ZeroPage, YRegister);
00668         break;
00669     }
00670     //CPY Compare Accumulator with Y Register, Absolute, 3 Bytes, 4 Cycles
00671     case 0xCC:
00672     {
00673         CompareOperation(AddressingMode.Absolute, YRegister);
00674         break;
00675     }
00676 #endregion
00677
00678 #region Increment/Decrement Operations
00679     //DEC Decrement Memory by One, Zero Page, 2 Bytes, 5 Cycles
00680     case 0xC6:
00681     {
00682         ChangeMemoryByOne(AddressingMode.ZeroPage, true);
00683         break;
00684     }
00685     //DEC Decrement Memory by One, Zero Page X, 2 Bytes, 6 Cycles
00686     case 0xD6:
00687     {
00688         ChangeMemoryByOne(AddressingMode.ZeroPageX, true);
00689         break;
00690     }
00691     //DEC Decrement Memory by One, Absolute, 3 Bytes, 6 Cycles
00692     case 0xCE:
00693     {
00694         ChangeMemoryByOne(AddressingMode.Absolute, true);
00695         break;
00696     }
00697     //DEC Decrement Memory by One, Absolute X, 3 Bytes, 7 Cycles
00698     case 0xDE:
00699     {
00700         ChangeMemoryByOne(AddressingMode.AbsoluteX, true);
00701         IncrementCycleCount();
00702         break;
00703     }
00704     //DEX Decrement X Register by One, Implied, 1 Bytes, 2 Cycles
00705     case 0xCA:
00706     {
00707         ChangeRegisterByOne(true, true);
00708         break;
00709     }
00710     //DEY Decrement Y Register by One, Implied, 1 Bytes, 2 Cycles
00711     case 0x88:
00712     {
00713         ChangeRegisterByOne(false, true);
00714         break;
00715     }
00716     //INC Increment Memory by One, Zero Page, 2 Bytes, 5 Cycles
00717     case 0xE6:
00718     {
00719         ChangeMemoryByOne(AddressingMode.ZeroPage, false);
00720         break;
00721     }
00722     //INC Increment Memory by One, Zero Page X, 2 Bytes, 6 Cycles
00723     case 0xF6:
00724     {
00725         ChangeMemoryByOne(AddressingMode.ZeroPageX, false);
00726         break;
00727     }
00728     //INC Increment Memory by One, Absolute, 3 Bytes, 6 Cycles
00729     case 0xEE:
00730     {
00731         ChangeMemoryByOne(AddressingMode.Absolute, false);
00732         break;
00733     }
00734     //INC Increment Memory by One, Absolute X, 3 Bytes, 7 Cycles
00735     case 0xFE:
00736     {
00737         ChangeMemoryByOne(AddressingMode.AbsoluteX, false);
00738         IncrementCycleCount();
00739         break;
00740     }
00741     //INX Increment X Register by One, Implied, 1 Bytes, 2 Cycles
00742     case 0xE8:
00743     {
00744         ChangeRegisterByOne(true, false);
00745         break;
00746     }
00747     //INY Increment Y Register by One, Implied, 1 Bytes, 2 Cycles
00748     case 0xC8:
00749     {

```

```

00750             ChangeRegisterByOne(false, false);
00751             break;
00752         }
00753 #endregion
00754
00755 #region GOTO and GOSUB Operations
00756 //JMP Jump to New Location, Absolute 3 Bytes, 3 Cycles
00757 case 0x4C:
00758     {
00759         ProgramCounter = GetAddressByAddressingMode(AddressingMode.Absolute);
00760         break;
00761     }
00762 //JMP Jump to New Location, Indirect 3 Bytes, 5 Cycles
00763 case 0x6C:
00764     {
00765         ProgramCounter = GetAddressByAddressingMode(AddressingMode.Absolute);
00766
00767         if ((ProgramCounter & 0xFF) == 0xFF)
00768         {
00769             //Get the first half of the address
00770             int address = MemoryMap.Read(ProgramCounter);
00771
00772             //Get the second half of the address, due to the issue with page boundary
00773             it reads from the wrong location!
00774             address += 256 * MemoryMap.Read(ProgramCounter - 255);
00775             ProgramCounter = address;
00776         }
00777         else
00778         {
00779             ProgramCounter = GetAddressByAddressingMode(AddressingMode.Absolute);
00780         }
00781         break;
00782     }
00783 //JSR Jump to SubRoutine, Absolute, 3 Bytes, 6 Cycles
00784 case 0x20:
00785     {
00786         JumpToSubRoutineOperation();
00787         break;
00788     }
00789 //BRK Simulate IRQ, Implied, 1 Byte, 7 Cycles
00790 case 0x00:
00791     {
00792         BreakOperation(true, 0xFFFE);
00793         break;
00794     }
00795 //RTI Return From Interrupt, Implied, 1 Byte, 6 Cycles
00796 case 0x40:
00797     {
00798         ReturnFromInterruptOperation();
00799         break;
00800     }
00801 //RTS Return From Subroutine, Implied, 1 Byte, 6 Cycles
00802 case 0x60:
00803     {
00804         ReturnFromSubRoutineOperation();
00805         break;
00806     }
00807 #endregion
00808
00809 #region Load Value From Memory Operations
00810 //LDA Load Accumulator with Memory, Immediate, 2 Bytes, 2 Cycles
00811 case 0xA9:
00812     {
00813         Accumulator =
00814         MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.Immediate));
00815         SetZeroFlag(Accumulator);
00816         SetNegativeFlag(Accumulator);
00817         break;
00818     }
00819 //LDA Load Accumulator with Memory, Zero Page, 2 Bytes, 3 Cycles
00820 case 0xA5:
00821     {
00822         Accumulator =
00823         MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.ZeroPage));
00824         SetZeroFlag(Accumulator);
00825         SetNegativeFlag(Accumulator);
00826         break;
00827     }
00828 //LDA Load Accumulator with Memory, Zero Page X, 2 Bytes, 4 Cycles
00829 case 0xB5:
00830     {
00831         Accumulator =
00832         MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.ZeroPageX));
00833         SetZeroFlag(Accumulator);
00834         SetNegativeFlag(Accumulator);
00835         break;
00836     }

```



```

00833         }
00834         //LDA Load Accumulator with Memory, Absolute, 3 Bytes, 4 Cycles
00835         case 0xAD:
00836         {
00837             Accumulator =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.Absolute));
00838             SetZeroFlag(Accumulator);
00839             SetNegativeFlag(Accumulator);
00840             break;
00841         }
00842         //LDA Load Accumulator with Memory, Absolute X, 3 Bytes, 4+ Cycles
00843         case 0xBD:
00844         {
00845             Accumulator =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.AbsoluteX));
00846             SetZeroFlag(Accumulator);
00847             SetNegativeFlag(Accumulator);
00848             break;
00849         }
00850         //LDA Load Accumulator with Memory, Absolute Y, 3 Bytes, 4+ Cycles
00851         case 0xB9:
00852         {
00853             Accumulator =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.AbsoluteY));
00854             SetZeroFlag(Accumulator);
00855             SetNegativeFlag(Accumulator);
00856             break;
00857         }
00858         //LDA Load Accumulator with Memory, Index Indirect, 2 Bytes, 6 Cycles
00859         case 0xA1:
00860         {
00861             Accumulator =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.IndirectX));
00862             SetZeroFlag(Accumulator);
00863             SetNegativeFlag(Accumulator);
00864             break;
00865         }
00866         //LDA Load Accumulator with Memory, Indirect Index, 2 Bytes, 5+ Cycles
00867         case 0xB1:
00868         {
00869             Accumulator =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.IndirectY));
00870             SetZeroFlag(Accumulator);
00871             SetNegativeFlag(Accumulator);
00872             break;
00873         }
00874         //LDX Load X with memory, Immediate, 2 Bytes, 2 Cycles
00875         case 0xA2:
00876         {
00877             XRegister =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.Immediate));
00878             SetZeroFlag(XRegister);
00879             SetNegativeFlag(XRegister);
00880             break;
00881         }
00882         //LDX Load X with memory, Zero Page, 2 Bytes, 3 Cycles
00883         case 0xA6:
00884         {
00885             XRegister =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.ZeroPage));
00886             SetZeroFlag(XRegister);
00887             SetNegativeFlag(XRegister);
00888             break;
00889         }
00890         //LDX Load X with memory, Zero Page Y, 2 Bytes, 4 Cycles
00891         case 0xB6:
00892         {
00893             XRegister =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.ZeroPageY));
00894             SetZeroFlag(XRegister);
00895             SetNegativeFlag(XRegister);
00896             break;
00897         }
00898         //LDX Load X with memory, Absolute, 3 Bytes, 4 Cycles
00899         case 0xAE:
00900         {
00901             XRegister =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.Absolute));
00902             SetZeroFlag(XRegister);
00903             SetNegativeFlag(XRegister);
00904             break;
00905         }
00906         //LDX Load X with memory, Absolute Y, 3 Bytes, 4+ Cycles
00907         case 0xBE:
00908         {
00909             XRegister =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.AbsoluteY));

```

```

00910             SetZeroFlag(XRegister);
00911             SetNegativeFlag(XRegister);
00912             break;
00913         }
00914         //LDY Load Y with memory, Immediate, 2 Bytes, 2 Cycles
00915         case 0xA0:
00916         {
00917             YRegister =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.Immediate));
00918             SetZeroFlag(YRegister);
00919             SetNegativeFlag(YRegister);
00920             break;
00921         }
00922         //LDY Load Y with memory, Zero Page, 2 Bytes, 3 Cycles
00923         case 0xA4:
00924         {
00925             YRegister =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.ZeroPage));
00926             SetZeroFlag(YRegister);
00927             SetNegativeFlag(YRegister);
00928             break;
00929         }
00930         //LDY Load Y with memory, Zero Page X, 2 Bytes, 4 Cycles
00931         case 0xB4:
00932         {
00933             YRegister =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.ZeroPageX));
00934             SetZeroFlag(YRegister);
00935             SetNegativeFlag(YRegister);
00936             break;
00937         }
00938         //LDY Load Y with memory, Absolute, 3 Bytes, 4 Cycles
00939         case 0xAC:
00940         {
00941             YRegister =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.Absolute));
00942             SetZeroFlag(YRegister);
00943             SetNegativeFlag(YRegister);
00944             break;
00945         }
00946         //LDY Load Y with memory, Absolute X, 3 Bytes, 4+ Cycles
00947         case 0xBC:
00948         {
00949             YRegister =
MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.AbsoluteX));
00950             SetZeroFlag(YRegister);
00951             SetNegativeFlag(YRegister);
00952             break;
00953         }
00954         #endregion
00955     #region Push/Pull Stack
00956         //PHA Push Accumulator onto Stack, Implied, 1 Byte, 3 Cycles
00957         case 0x48:
00958         {
00959             MemoryMap.Read(ProgramCounter + 1);
00960
00961             PokeStack((byte)Accumulator);
00962             StackPointer--;
00963             IncrementCycleCount();
00964             break;
00965         }
00966     }
00967     //PHP Push Flags onto Stack, Implied, 1 Byte, 3 Cycles
00968     case 0x08:
00969     {
00970         MemoryMap.Read(ProgramCounter + 1);
00971
00972         PushFlagsOperation();
00973         StackPointer--;
00974         IncrementCycleCount();
00975         break;
00976     }
00977 }
00978 //PLA Pull Accumulator from Stack, Implied, 1 Byte, 4 Cycles
00979 case 0x68:
00980 {
00981     MemoryMap.Read(ProgramCounter + 1);
00982     StackPointer++;
00983     IncrementCycleCount();
00984
00985     Accumulator = PeekStack();
00986     SetNegativeFlag(Accumulator);
00987     SetZeroFlag(Accumulator);
00988
00989     IncrementCycleCount();
00990     break;
00991 }

```

```

00992         //PLP Pull Flags from Stack, Implied, 1 Byte, 4 Cycles
00993         case 0x28:
00994         {
00995             MemoryMap.Read(ProgramCounter + 1);
00996
00997             StackPointer++;
00998             IncrementCycleCount();
00999
01000             PullFlagsOperation();
01001
01002             IncrementCycleCount();
01003             break;
01004         }
01005         //TSX Transfer Stack Pointer to X Register, 1 Bytes, 2 Cycles
01006         case 0xBA:
01007         {
01008             XRegister = StackPointer;
01009
01010             SetNegativeFlag(XRegister);
01011             SetZeroFlag(XRegister);
01012             IncrementCycleCount();
01013             break;
01014         }
01015         //TXS Transfer X Register to Stack Pointer, 1 Bytes, 2 Cycles
01016         case 0x9A:
01017         {
01018             StackPointer = (byte)XRegister;
01019             IncrementCycleCount();
01020             break;
01021         }
01022     #endregion
01023
01024     #region Set Flag Operations
01025         //SEC Set Carry, Implied, 1 Bytes, 2 Cycles
01026         case 0x38:
01027         {
01028             CarryFlag = true;
01029             IncrementCycleCount();
01030             break;
01031         }
01032         //SED Set Interrupt, Implied, 1 Bytes, 2 Cycles
01033         case 0xF8:
01034         {
01035             DecimalFlag = true;
01036             IncrementCycleCount();
01037             break;
01038         }
01039         //SEI Set Interrupt, Implied, 1 Bytes, 2 Cycles
01040         case 0x78:
01041         {
01042             DisableInterruptFlag = true;
01043             IncrementCycleCount();
01044             break;
01045         }
01046     #endregion
01047
01048     #region Shift/Rotate Operations
01049         //ASL Shift Left 1 Bit Memory or Accumulator, Accumulator, 1 Bytes, 2 Cycles
01050         case 0x0A:
01051         {
01052             AslOperation(AddressingMode.Accumulator);
01053             break;
01054         }
01055         //ASL Shift Left 1 Bit Memory or Accumulator, Zero Page, 2 Bytes, 5 Cycles
01056         case 0x06:
01057         {
01058             AslOperation(AddressingMode.ZeroPage);
01059             break;
01060         }
01061         //ASL Shift Left 1 Bit Memory or Accumulator, Zero PageX, 2 Bytes, 6 Cycles
01062         case 0x16:
01063         {
01064             AslOperation(AddressingMode.ZeroPageX);
01065             break;
01066         }
01067         //ASL Shift Left 1 Bit Memory or Accumulator, Absolute, 3 Bytes, 6 Cycles
01068         case 0x0E:
01069         {
01070             AslOperation(AddressingMode.Absolute);
01071             break;
01072         }
01073         //ASL Shift Left 1 Bit Memory or Accumulator, AbsoluteX, 3 Bytes, 7 Cycles
01074         case 0x1E:
01075         {
01076             AslOperation(AddressingMode.AbsoluteX);
01077             IncrementCycleCount();
01078             break;

```

```

01079     }
01080     //LSR Shift Left 1 Bit Memory or Accumulator, Accumulator, 1 Bytes, 2 Cycles
01081     case 0x4A:
01082     {
01083         LsrOperation(AddressingMode.Accumulator);
01084         break;
01085     }
01086     //LSR Shift Left 1 Bit Memory or Accumulator, Zero Page, 2 Bytes, 5 Cycles
01087     case 0x46:
01088     {
01089         LsrOperation(AddressingMode.ZeroPage);
01090         break;
01091     }
01092     //LSR Shift Left 1 Bit Memory or Accumulator, Zero PageX, 2 Bytes, 6 Cycles
01093     case 0x56:
01094     {
01095         LsrOperation(AddressingMode.ZeroPageX);
01096         break;
01097     }
01098     //LSR Shift Left 1 Bit Memory or Accumulator, Absolute, 3 Bytes, 6 Cycles
01099     case 0x4E:
01100     {
01101         LsrOperation(AddressingMode.Absolute);
01102         break;
01103     }
01104     //LSR Shift Left 1 Bit Memory or Accumulator, AbsoluteX, 3 Bytes, 7 Cycles
01105     case 0x5E:
01106     {
01107         LsrOperation(AddressingMode.AbsoluteX);
01108         IncrementCycleCount();
01109         break;
01110     }
01111     //ROL Rotate Left 1 Bit Memory or Accumulator, Accumulator, 1 Bytes, 2 Cycles
01112     case 0x2A:
01113     {
01114         RolOperation(AddressingMode.Accumulator);
01115         break;
01116     }
01117     //ROL Rotate Left 1 Bit Memory or Accumulator, Zero Page, 2 Bytes, 5 Cycles
01118     case 0x26:
01119     {
01120         RolOperation(AddressingMode.ZeroPage);
01121         break;
01122     }
01123     //ROL Rotate Left 1 Bit Memory or Accumulator, Zero PageX, 2 Bytes, 6 Cycles
01124     case 0x36:
01125     {
01126         RolOperation(AddressingMode.ZeroPageX);
01127         break;
01128     }
01129     //ROL Rotate Left 1 Bit Memory or Accumulator, Absolute, 3 Bytes, 6 Cycles
01130     case 0x2E:
01131     {
01132         RolOperation(AddressingMode.Absolute);
01133         break;
01134     }
01135     //ROL Rotate Left 1 Bit Memory or Accumulator, AbsoluteX, 3 Bytes, 7 Cycles
01136     case 0x3E:
01137     {
01138         RolOperation(AddressingMode.AbsoluteX);
01139         IncrementCycleCount();
01140         break;
01141     }
01142     //ROR Rotate Right 1 Bit Memory or Accumulator, Accumulator, 1 Bytes, 2 Cycles
01143     case 0x6A:
01144     {
01145         RorOperation(AddressingMode.Accumulator);
01146         break;
01147     }
01148     //ROR Rotate Right 1 Bit Memory or Accumulator, Zero Page, 2 Bytes, 5 Cycles
01149     case 0x66:
01150     {
01151         RorOperation(AddressingMode.ZeroPage);
01152         break;
01153     }
01154     //ROR Rotate Right 1 Bit Memory or Accumulator, Zero PageX, 2 Bytes, 6 Cycles
01155     case 0x76:
01156     {
01157         RorOperation(AddressingMode.ZeroPageX);
01158         break;
01159     }
01160     //ROR Rotate Right 1 Bit Memory or Accumulator, Absolute, 3 Bytes, 6 Cycles
01161     case 0x6E:
01162     {
01163         RorOperation(AddressingMode.Absolute);
01164         break;
01165     }

```

```

01166          //ROR Rotate Right 1 Bit Memory or Accumulator, AbsoluteX, 3 Bytes, 7 Cycles
01167          case 0x7E:
01168              {
01169                  RorOperation(AddressingMode.AbsoluteX);
01170                  IncrementCycleCount();
01171                  break;
01172              }
01173      #endregion
01174
01175      #region Store Value In Memory Operations
01176          //STA Store Accumulator In Memory, Zero Page, 2 Bytes, 3 Cycles
01177          case 0x85:
01178              {
01179                  MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.ZeroPage),
01180 (byte)Accumulator);
01181                  break;
01182              }
01183          //STA Store Accumulator In Memory, Zero Page X, 2 Bytes, 4 Cycles
01184          case 0x95:
01185              {
01186                  MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.ZeroPageX),
01187 (byte)Accumulator);
01188                  break;
01189              }
01190          //STA Store Accumulator In Memory, Absolute, 3 Bytes, 4 Cycles
01191          case 0x8D:
01192              {
01193                  MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.Absolute),
01194 (byte)Accumulator);
01195                  break;
01196              }
01197          //STA Store Accumulator In Memory, Absolute X, 3 Bytes, 5 Cycles
01198          case 0x9D:
01199              {
01200                  MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.AbsoluteX),
01201 (byte)Accumulator);
01202                  IncrementCycleCount();
01203                  break;
01204              }
01205          //STA Store Accumulator In Memory, Absolute Y, 3 Bytes, 5 Cycles
01206          case 0x99:
01207              {
01208                  MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.AbsoluteY),
01209 (byte)Accumulator);
01210                  IncrementCycleCount();
01211                  break;
01212              }
01213          //STA Store Accumulator In Memory, Indexed Indirect, 2 Bytes, 6 Cycles
01214          case 0x81:
01215              {
01216                  MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.IndirectX),
01217 (byte)Accumulator);
01218                  break;
01219              }
01220          //STA Store Accumulator In Memory, Indirect Indexed, 2 Bytes, 6 Cycles
01221          case 0x91:
01222              {
01223                  MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.IndirectY),
01224 (byte)Accumulator);
01225                  IncrementCycleCount();
01226                  break;
01227              }
01228          //STX Store Index X, Zero Page, 2 Bytes, 3 Cycles
01229          case 0x86:
01230              {
01231                  MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.ZeroPage),
01232 (byte)XRegister);
01233                  break;
01234              }
01235          //STX Store Index X, Zero Page Y, 2 Bytes, 4 Cycles
01236          case 0x96:
01237              {
01238                  MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.ZeroPageY),
01239 (byte)XRegister);
01240                  break;
01241              }
01242          //STX Store Index X, Absolute, 3 Bytes, 4 Cycles
01243          case 0x8E:
01244              {
01245                  MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.Absolute),
01246 (byte)XRegister);
01247                  break;
01248              }
01249          //STY Store Index Y, Zero Page, 2 Bytes, 3 Cycles
01250          case 0x84:
01251              {
01252                  MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.ZeroPage),
01253 (byte)YRegister);
01254                  break;
01255              }
01256          //STY Store Index Y, Zero Page X, 2 Bytes, 4 Cycles
01257          case 0x94:
01258              {
01259                  MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.ZeroPageX),
01260 (byte)YRegister);
01261                  break;
01262              }
01263          //STY Store Index Y, Absolute, 3 Bytes, 4 Cycles
01264          case 0x8C:
01265              {
01266                  MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.Absolute),
01267 (byte)YRegister);
01268                  break;
01269              }
01270          //STY Store Index Y, Absolute X, 3 Bytes, 5 Cycles
01271          case 0x9C:
01272              {
01273                  MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.AbsoluteX),
01274 (byte)YRegister);
01275                  IncrementCycleCount();
01276                  break;
01277              }
01278          //STY Store Index Y, Absolute Y, 3 Bytes, 5 Cycles
01279          case 0x94:
01280              {
01281                  MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.AbsoluteY),
01282 (byte)YRegister);
01283                  IncrementCycleCount();
01284                  break;
01285              }
01286          //STX Store Index X, Indexed Indirect, 2 Bytes, 6 Cycles
01287          case 0x81:
01288              {
01289                  MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.IndirectX),
01290 (byte)XRegister);
01291                  break;
01292              }
01293          //STX Store Index X, Indirect Indexed, 2 Bytes, 6 Cycles
01294          case 0x91:
01295              {
01296                  MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.IndirectY),
01297 (byte)XRegister);
01298                  IncrementCycleCount();
01299                  break;
01300              }
01301          //STX Store Index X, Absolute, 3 Bytes, 4 Cycles
01302          case 0x8E:
01303              {
01304                  MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.Absolute),
01305 (byte)XRegister);
01306                  break;
01307              }
01308          //STY Store Index Y, Absolute, 3 Bytes, 4 Cycles
01309          case 0x8C:
01310              {
01311                  MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.Absolute),
01312 (byte)YRegister);
01313                  break;
01314              }
01315          //STY Store Index Y, Absolute X, 3 Bytes, 5 Cycles
01316          case 0x9C:
01317              {
01318                  MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.AbsoluteX),
01319 (byte)YRegister);
01320                  IncrementCycleCount();
01321                  break;
01322              }
01323          //STY Store Index Y, Absolute Y, 3 Bytes, 5 Cycles
01324          case 0x94:
01325              {
01326                  MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.AbsoluteY),
01327 (byte)YRegister);
01328                  IncrementCycleCount();
01329                  break;
01330              }
01331      #endregion
01332  }
01333  }
01334  }
01335  }
01336  }
01337  }
01338  }
01339  }
01340  }
01341  }
01342  }
01343  }
01344  }
01345  }
01346  }
01347  }
01348  }
01349  }
01350  }
01351  }
01352  }
01353  }
01354  }
01355  }
01356  }
01357  }
01358  }
01359  }
01360  }
01361  }
01362  }
01363  }
01364  }
01365  }
01366  }
01367  }
01368  }
01369  }
01370  }
01371  }
01372  }
01373  }
01374  }
01375  }
01376  }
01377  }
01378  }
01379  }
01380  }
01381  }
01382  }
01383  }
01384  }
01385  }
01386  }
01387  }
01388  }
01389  }
01390  }
01391  }
01392  }
01393  }
01394  }
01395  }
01396  }
01397  }
01398  }
01399  }
01400  }
01401  }
01402  }
01403  }
01404  }
01405  }
01406  }
01407  }
01408  }
01409  }
01410  }
01411  }
01412  }
01413  }
01414  }
01415  }
01416  }
01417  }
01418  }
01419  }
01420  }
01421  }
01422  }
01423  }
01424  }
01425  }
01426  }
01427  }
01428  }
01429  }
01430  }
01431  }
01432  }
01433  }
01434  }
01435  }
01436  }
01437  }
01438  }
01439  }
01440  }
01441  }
01442  }
01443  }
01444  }
01445  }
01446  }
01447  }
01448  }
01449  }
01450  }
01451  }
01452  }
01453  }
01454  }
01455  }
01456  }
01457  }
01458  }
01459  }
01460  }
01461  }
01462  }
01463  }
01464  }
01465  }
01466  }
01467  }
01468  }
01469  }
01470  }
01471  }
01472  }
01473  }
01474  }
01475  }
01476  }
01477  }
01478  }
01479  }
01480  }
01481  }
01482  }
01483  }
01484  }
01485  }
01486  }
01487  }
01488  }
01489  }
01490  }
01491  }
01492  }
01493  }
01494  }
01495  }
01496  }
01497  }
01498  }
01499  }
01500  }
01501  }
01502  }
01503  }
01504  }
01505  }
01506  }
01507  }
01508  }
01509  }
01510  }
01511  }
01512  }
01513  }
01514  }
01515  }
01516  }
01517  }
01518  }
01519  }
01520  }
01521  }
01522  }
01523  }
01524  }
01525  }
01526  }
01527  }
01528  }
01529  }
01530  }
01531  }
01532  }
01533  }
01534  }
01535  }
01536  }
01537  }
01538  }
01539  }
01540  }
01541  }
01542  }
01543  }
01544  }
01545  }
01546  }
01547  }
01548  }
01549  }
01550  }
01551  }
01552  }
01553  }
01554  }
01555  }
01556  }
01557  }
01558  }
01559  }
01560  }
01561  }
01562  }
01563  }
01564  }
01565  }
01566  }
01567  }
01568  }
01569  }
01570  }
01571  }
01572  }
01573  }
01574  }
01575  }
01576  }
01577  }
01578  }
01579  }
01580  }
01581  }
01582  }
01583  }
01584  }
01585  }
01586  }
01587  }
01588  }
01589  }
01590  }
01591  }
01592  }
01593  }
01594  }
01595  }
01596  }
01597  }
01598  }
01599  }
01600  }
01601  }
01602  }
01603  }
01604  }
01605  }
01606  }
01607  }
01608  }
01609  }
01610  }
01611  }
01612  }
01613  }
01614  }
01615  }
01616  }
01617  }
01618  }
01619  }
01620  }
01621  }
01622  }
01623  }
01624  }
01625  }
01626  }
01627  }
01628  }
01629  }
01630  }
01631  }
01632  }
01633  }
01634  }
01635  }
01636  }
01637  }
01638  }
01639  }
01640  }
01641  }
01642  }
01643  }
01644  }
01645  }
01646  }
01647  }
01648  }
01649  }
01650  }
01651  }
01652  }
01653  }
01654  }
01655  }
01656  }
01657  }
01658  }
01659  }
01660  }
01661  }
01662  }
01663  }
01664  }
01665  }
01666  }
01667  }
01668  }
01669  }
01670  }
01671  }
01672  }
01673  }
01674  }
01675  }
01676  }
01677  }
01678  }
01679  }
01680  }
01681  }
01682  }
01683  }
01684  }
01685  }
01686  }
01687  }
01688  }
01689  }
01690  }
01691  }
01692  }
01693  }
01694  }
01695  }
01696  }
01697  }
01698  }
01699  }
01700  }
01701  }
01702  }
01703  }
01704  }
01705  }
01706  }
01707  }
01708  }
01709  }
01710  }
01711  }
01712  }
01713  }
01714  }
01715  }
01716  }
01717  }
01718  }
01719  }
01720  }
01721  }
01722  }
01723  }
01724  }
01725  }
01726  }
01727  }
01728  }
01729  }
01730  }
01731  }
01732  }
01733  }
01734  }
01735  }
01736  }
01737  }
01738  }
01739  }
01740  }
01741  }
01742  }
01743  }
01744  }
01745  }
01746  }
01747  }
01748  }
01749  }
01750  }
01751  }
01752  }
01753  }
01754  }
01755  }
01756  }
01757  }
01758  }
01759  }
01760  }
01761  }
01762  }
01763  }
01764  }
01765  }
01766  }
01767  }
01768  }
01769  }
01770  }
01771  }
01772  }
01773  }
01774  }
01775  }
01776  }
01777  }
01778  }
01779  }
01780  }
01781  }
01782  }
01783  }
01784  }
01785  }
01786  }
01787  }
01788  }
01789  }
01790  }
01791  }
01792  }
01793  }
01794  }
01795  }
01796  }
01797  }
01798  }
01799  }
01800  }
01801  }
01802  }
01803  }
01804  }
01805  }
01806  }
01807  }
01808  }
01809  }
01810  }
01811  }
01812  }
01813  }
01814  }
01815  }
01816  }
01817  }
01818  }
01819  }
01820  }
01821  }
01822  }
01823  }
01824  }
01825  }
01826  }
01827  }
01828  }
01829  }
01830  }
01831  }
01832  }
01833  }
01834  }
01835  }
01836  }
01837  }
01838  }
01839  }
01840  }
01841  }
01842  }
01843  }
01844  }
01845  }
01846  }
01847  }
01848  }
01849  }
01850  }
01851  }
01852  }
01853  }
01854  }
01855  }
01856  }
01857  }
01858  }
01859  }
01860  }
01861  }
01862  }
01863  }
01864  }
01865  }
01866  }
01867  }
01868  }
01869  }
01870  }
01871  }
01872  }
01873  }
01874  }
01875  }
01876  }
01877  }
01878  }
01879  }
01880  }
01881  }
01882  }
01883  }
01884  }
01885  }
01886  }
01887  }
01888  }
01889  }
01890  }
01891  }
01892  }
01893  }
01894  }
01895  }
01896  }
01897  }
01898  }
01899  }
01900  }
01901  }
01902  }
01903  }
01904  }
01905  }
01906  }
01907  }
01908  }
01909  }
01910  }
01911  }
01912  }
01913  }
01914  }
01915  }
01916  }
01917  }
01918  }
01919  }
01920  }
01921  }
01922  }
01923  }
01924  }
01925  }
01926  }
01927  }
01928  }
01929  }
01930  }
01931  }
01932  }
01933  }
01934  }
01935  }
01936  }
01937  }
01938  }
01939  }
01940  }
01941  }
01942  }
01943  }
01944  }
01945  }
01946  }
01947  }
01948  }
01949  }
01950  }
01951  }
01952  }
01953  }
01954  }
01955  }
01956  }
01957  }
01958  }
01959  }
01960  }
01961  }
01962  }
01963  }
01964  }
01965  }
01966  }
01967  }
01968  }
01969  }
01970  }
01971  }
01972  }
01973  }
01974  }
01975  }
01976  }
01977  }
01978  }
01979  }
01980  }
01981  }
01982  }
01983  }
01984  }
01985  }
01986  }
01987  }
01988  }
01989  }
01990  }
01991  }
01992  }
01993  }
01994  }
01995  }
01996  }
01997  }
01998  }
01999  }
02000  }

```

```

    (byte)YRegister);
01243         break;
01244     }
01245     //STY Store Index Y, Zero Page X, 2 Bytes, 4 Cycles
01246     case 0x94:
01247     {
01248         MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.ZeroPageX),
    (byte)YRegister);
01249         break;
01250     }
01251     //STY Store Index Y, Absolute, 2 Bytes, 4 Cycles
01252     case 0x8C:
01253     {
01254         MemoryMap.Write(GetAddressByAddressingMode(AddressingMode.Absolute),
    (byte)YRegister);
01255         break;
01256     }
01257 #endregion
01258
01259 #region Transfer Operations
01260     //TAX Transfer Accumulator to X Register, Implied, 1 Bytes, 2 Cycles
01261     case 0xAA:
01262     {
01263         IncrementCycleCount();
01264         XRegister = Accumulator;
01265
01266         SetNegativeFlag(XRegister);
01267         SetZeroFlag(XRegister);
01268         break;
01269     }
01270     //TAY Transfer Accumulator to Y Register, 1 Bytes, 2 Cycles
01271     case 0xA8:
01272     {
01273         IncrementCycleCount();
01274         YRegister = Accumulator;
01275
01276         SetNegativeFlag(YRegister);
01277         SetZeroFlag(YRegister);
01278         break;
01279     }
01280     //TXA Transfer X Register to Accumulator, Implied, 1 Bytes, 2 Cycles
01281     case 0x8A:
01282     {
01283         IncrementCycleCount();
01284         Accumulator = XRegister;
01285
01286         SetNegativeFlag(Accumulator);
01287         SetZeroFlag(Accumulator);
01288         break;
01289     }
01290     //TYA Transfer Y Register to Accumulator, Implied, 1 Bytes, 2 Cycles
01291     case 0x98:
01292     {
01293         IncrementCycleCount();
01294         Accumulator = YRegister;
01295
01296         SetNegativeFlag(Accumulator);
01297         SetZeroFlag(Accumulator);
01298         break;
01299     }
01300 #endregion
01301
01302     //NOP Operation, Implied, 1 Byte, 2 Cycles
01303     case 0xEA:
01304     {
01305         IncrementCycleCount();
01306         break;
01307     }
01308
01309     default:
01310         throw new NotSupportedException(string.Format("The OpCode {0} is not supported",
    CurrentOpCode));
01311     }
01312 }
01313
01314 /// <summary>
01315 /// Sets the IsSignNegative register
01316 /// </summary>
01317 /// <param name="value"></param>
01318 protected void SetNegativeFlag(int value)
01319 {
01320     //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.
01321     NegativeFlag = value > 127;
01322 }
01323
01324 /// <summary>

```

```

01325 /// Sets the IsResultZero register
01326 /// </summary>
01327 /// <param name="value"></param>
01328 protected void SetZeroFlag(int value)
01329 {
01330     ZeroFlag = value == 0;
01331 }
01332
01333 /// <summary>
01334 /// Uses the AddressingMode to return the correct address based on the mode.
01335 /// Note: This method will not increment the program counter for any mode.
01336 /// Note: This method will return an error if called for either the immediate or accumulator modes.
01337 /// </summary>
01338 /// <param name="addressingMode">The addressing Mode to use</param>
01339 /// <returns>The memory Location</returns>
01340 protected int GetAddressByAddressingMode(AddressingMode addressingMode)
01341 {
01342     int address;
01343     int highByte;
01344     switch (addressingMode)
01345     {
01346         case AddressingMode.Absolute):
01347             {
01348                 return (MemoryMap.Read(ProgramCounter++) | (MemoryMap.Read(ProgramCounter++) <
01349 8));
01350             }
01351         case AddressingMode.AbsoluteX:
01352             {
01353                 //Get the low half of the address
01354                 address = MemoryMap.Read(ProgramCounter++);
01355                 //Get the high byte
01356                 highByte = MemoryMap.Read(ProgramCounter++);
01357                 //We crossed a page boundry, so an extra read has occurred.
01358                 //However, if this is an ASL, LSR, DEC, INC, ROR, ROL or STA operation, we do
01359                 not decrease it by 1.
01360                 if (address + XRegister > 0xFF)
01361                 {
01362                     switch (CurrentOpCode)
01363                     {
01364                         case 0x1E:
01365                         case 0xDE:
01366                         case 0xFE:
01367                         case 0x5E:
01368                         case 0x3E:
01369                         case 0x7E:
01370                         case 0x9D:
01371                         {
01372                             //This is a MemoryMap.Read Fetch Write Operation, so we don't
01373                             make the extra read.
01374                             return ((highByte < 8 | address) + XRegister) & 0xFFFF;
01375                         }
01376                         default:
01377                         {
01378                             MemoryMap.Read((((highByte < 8 | address) + XRegister) - 0xFF)
01379 & 0xFFFF);
01380                             break;
01381                         }
01382                     }
01383                 }
01384                 return ((highByte < 8 | address) + XRegister) & 0xFFFF;
01385             }
01386         case AddressingMode.AbsoluteY:
01387             {
01388                 //Get the low half of the address
01389                 address = MemoryMap.Read(ProgramCounter++);
01390                 //Get the high byte
01391                 highByte = MemoryMap.Read(ProgramCounter++);
01392                 //We crossed a page boundry, so decrease the number of cycles by 1 if the
01393                 operation is not STA
01394                 if (address + YRegister > 0xFF && CurrentOpCode != 0x99)
01395                 {
01396                     MemoryMap.Read((((highByte < 8 | address) + YRegister) - 0xFF) & 0xFFFF);
01397                 }
01398                 //Bitshift the high byte into place, AND with FFFF to handle wrapping.
01399                 return ((highByte < 8 | address) + YRegister) & 0xFFFF;
01400             }
01401         case AddressingMode.Immediate:
01402             {
01403                 return ProgramCounter++;
01404             }
01405         case AddressingMode.IndirectX:

```

```

01407         {
01408             //Get the location of the address to retrieve
01409             address = MemoryMap.Read(ProgramCounter++);
01410             MemoryMap.Read(address);
01411
01412             address += XRegister;
01413
01414             //Now get the final Address. The is not a zero page address either.
01415             var finalAddress = MemoryMap.Read((address & 0xFF)) | (MemoryMap.Read((address
+ 1) & 0xFF) << 8);
01416             return finalAddress;
01417         }
01418         case AddressingMode.IndirectY:
01419         {
01420             address = MemoryMap.Read(ProgramCounter++);
01421
01422             var finalAddress = MemoryMap.Read(address) + (MemoryMap.Read((address + 1) &
0xFF) << 8);
01423
01424             if ((finalAddress & 0xFF) + YRegister > 0xFF && CurrentOpCode != 0x91)
01425             {
01426                 MemoryMap.Read((finalAddress + YRegister - 0xFF) & 0xFFFF);
01427             }
01428
01429             return (finalAddress + YRegister) & 0xFFFF;
01430         }
01431         case AddressingMode.Relative:
01432         {
01433             return ProgramCounter;
01434         }
01435         case (AddressingMode.ZeroPage):
01436         {
01437             address = MemoryMap.Read(ProgramCounter++);
01438             return address;
01439         }
01440         case (AddressingMode.ZeroPageX):
01441         {
01442             address = MemoryMap.Read(ProgramCounter++);
01443             MemoryMap.Read(address);
01444
01445             address += XRegister;
01446             address &= 0xFF;
01447
01448             //This address wraps if its greater than 0xFF
01449             if (address > 0xFF)
01450             {
01451                 address -= 0x100;
01452                 return address;
01453             }
01454
01455             return address;
01456         }
01457         case (AddressingMode.ZeroPageY):
01458         {
01459             address = MemoryMap.Read(ProgramCounter++);
01460             MemoryMap.Read(address);
01461
01462             address += YRegister;
01463             address &= 0xFF;
01464
01465             return address;
01466         }
01467         default:
01468             throw new InvalidOperationException(string.Format("The Address Mode '{0}' does not
require an address", addressingMode));
01469     }
01470 }
01471
01472 /// <summary>
01473 /// Moves the ProgramCounter in a given direction based on the value inputted
01474 ///
01475 /// </summary>
01476 private void MoveProgramCounterByRelativeValue(byte valueToMove)
01477 {
01478     var movement = valueToMove > 127 ? (valueToMove - 255) : valueToMove;
01479
01480     var newProgramCounter = ProgramCounter + movement;
01481
01482     //This makes sure that we always land on the correct spot for a positive number
01483     if (movement >= 0)
01484         newProgramCounter++;
01485
01486     //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
01487     if (((ProgramCounter + 1 ^ newProgramCounter) & 0xff00) != 0x0000)
01488     {
01489         IncrementCycleCount();

```



```

01490     }
01491
01492     ProgramCounter = newProgramCounter;
01493     MemoryMap.Read(ProgramCounter);
01494 }
01495
01496 /// <summary>
01497 /// Returns a the value from the stack without changing the position of the stack pointer
01498 /// </summary>
01499 /// <returns>The value at the current Stack Pointer</returns>
01500 private byte PeekStack()
01501 {
01502     //The stack lives at 0x100-0x1FF, but the value is only a byte so it needs to be
01503     translated return MemoryMap.Read(StackPointer + 0x100);
01504 }
01505
01506 /// <summary>
01507 /// Write a value directly to the stack without modifying the Stack Pointer
01508 /// </summary>
01509 ///
01510 /// <param name="value">The value to be written to the stack</param>
01511 private void PokeStack(byte value)
01512 {
01513     //The stack lives at 0x100-0x1FF, but the value is only a byte so it needs to be
01514     translated MemoryMap.Write(StackPointer + 0x100, value);
01515 }
01516
01517 /// <summary>
01518 /// Coverts the Flags into its byte representation.
01519 /// </summary>
01520 /// <param name="setBreak">Determines if the break flag should be set during conversion.  IRQ does not
01521 set the flag on the stack, but PHP and BRK do</param>
01522 /// <returns></returns>
01522 private byte ConvertFlagsToByte(bool setBreak)
01523 {
01524     return (byte)((CarryFlag ? 0x01 : 0) + (ZeroFlag ? 0x02 : 0) + (DisableInterruptFlag ?
01525 0x04 : 0) +
01526 (DecimalFlag ? 8 : 0) + (setBreak ? 0x10 : 0) + 0x20 + (OverflowFlag ? 0x40 : 0)
01527 + (NegativeFlag ? 0x80 : 0));
01528 }
01529 [Conditional("DEBUG")]
01529 private void SetDisassembly()
01530 {
01531     if (!_logger.IsDebugEnabled)
01532         return;
01533
01534     var addressMode = GetAddressingMode();
01535
01536     var currentProgramCounter = ProgramCounter;
01537
01538     currentProgramCounter = WrapProgramCounter(++currentProgramCounter);
01539     int? address1 = MemoryMap.Read(currentProgramCounter);
01540
01541     currentProgramCounter = WrapProgramCounter(++currentProgramCounter);
01542     int? address2 = MemoryMap.Read(currentProgramCounter);
01543
01544     string disassembledStep = string.Empty;
01545
01546     switch (addressMode)
01547     {
01548         case AddressingMode.Absolute:
01549         {
01550             disassembledStep = string.Format("${0}{1}",
01551 address2.Value.ToString("X").PadLeft(2, '0'), address1.Value.ToString("X").PadLeft(2, '0'));
01552             break;
01553         }
01554         case AddressingMode.AbsoluteX:
01555         {
01556             disassembledStep = string.Format("${0}{1},X",
01557 address2.Value.ToString("X").PadLeft(2, '0'), address1.Value.ToString("X").PadLeft(2, '0'));
01558             break;
01559         }
01560         case AddressingMode.AbsoluteY:
01561         {
01562             disassembledStep = string.Format("${0}{1},Y",
01563 address2.Value.ToString("X").PadLeft(2, '0'), address1.Value.ToString("X").PadLeft(2, '0'));
01564             break;
01565         }
01566         case AddressingMode.Accumulator:
01567         {
01568             address1 = null;
01569             address2 = null;
01570
01571             disassembledStep = "A";
01572         }
01573     }

```

```

01569             break;
01570         }
01571         case AddressingMode.Immediate:
01572         {
01573             disassembledStep = string.Format("#{0}",
01574 address1.Value.ToString("X").PadLeft(4, '0'));
01575             address2 = null;
01576             break;
01577         }
01578         case AddressingMode.Implied:
01579         {
01580             address1 = null;
01581             address2 = null;
01582             break;
01583         }
01584         case AddressingMode.Indirect:
01585         {
01586             disassembledStep = string.Format("{0}{1}",
01587 address2.Value.ToString("X").PadLeft(2, '0'), address1.Value.ToString("X").PadLeft(2, '0'));
01588             break;
01589         }
01590         case AddressingMode.IndirectX:
01591         {
01592             address2 = null;
01593             disassembledStep = string.Format("{0},X",
01594 address1.Value.ToString("X").PadLeft(2, '0'));
01595             break;
01596         }
01597         case AddressingMode.IndirectY:
01598         {
01599             address2 = null;
01600             disassembledStep = string.Format("{0},Y",
01601 address1.Value.ToString("X").PadLeft(2, '0'));
01602             break;
01603         }
01604         case AddressingMode.Relative:
01605         {
01606             var valueToMove = (byte)address1.Value;
01607             var movement = valueToMove > 127 ? (valueToMove - 255) : valueToMove;
01608             var newProgramCounter = ProgramCounter + movement;
01609             //This makes sure that we always land on the correct spot for a positive
01610             number
01611             if (movement >= 0)
01612                 newProgramCounter++;
01613             var stringAddress = ProgramCounter.ToString("X").PadLeft(4, '0');
01614             address1 = int.Parse(stringAddress.Substring(0, 2),
01615 NumberStyles.AllowHexSpecifier);
01616             address2 = int.Parse(stringAddress.Substring(2, 2),
01617 NumberStyles.AllowHexSpecifier);
01618             disassembledStep = string.Format("#{0}",
01619 newProgramCounter.ToString("X").PadLeft(4, '0'));
01620             break;
01621         }
01622         case AddressingMode.ZeroPage:
01623         {
01624             address2 = null;
01625             disassembledStep = string.Format("#{0}",
01626 address1.Value.ToString("X").PadLeft(2, '0'));
01627             break;
01628         }
01629         case AddressingMode.ZeroPageX:
01630         {
01631             address2 = null;
01632             disassembledStep = string.Format("#{0},X",
01633 address1.Value.ToString("X").PadLeft(2, '0'));
01634             break;
01635         }
01636         case AddressingMode.ZeroPageY:
01637         {
01638             address2 = null;
01639             disassembledStep = string.Format("#{0},Y",
01640 address1.Value.ToString("X").PadLeft(4, '0'));
01641             break;
01642         }
01643         default:
01644

```

```

01645             throw new InvalidEnumArgumentException("Invalid Addressing Mode");
01646         }
01647     }
01648
01649     CurrentDisassembly = new Disassembly
01650     {
01651         HighAddress = address2.HasValue ? address2.Value.ToString("X").PadLeft(2, '0') :
01652         string.Empty,
01653         LowAddress = address1.HasValue ? address1.Value.ToString("X").PadLeft(2, '0') :
01654         string.Empty,
01655         OpCodeString = CurrentOpCode.ConvertOpCodeIntoString(),
01656         DisassemblyOutput = disassembledStep
01657     };
01658
01659     _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}",
01660         ProgramCounter.ToString("X").PadLeft(4, '0'),
01661         CurrentOpCode.ToString("X").PadLeft(2, '0'),
01662         CurrentDisassembly.LowAddress,
01663         CurrentDisassembly.HighAddress,
01664         CurrentDisassembly.OpCodeString,
01665         CurrentDisassembly.DisassemblyOutput.PadRight(10, ' '),
01666         Accumulator.ToString("X").PadLeft(3, '0'),
01667         XRegister.ToString("X").PadLeft(3, '0'),
01668         YRegister.ToString("X").PadLeft(3, '0'),
01669         StackPointer.ToString("X").PadLeft(3, '0'),
01670         Convert.ToInt16(NegativeFlag),
01671         Convert.ToInt16(OverflowFlag),
01672         0,
01673         Convert.ToInt16(DecimalFlag),
01674         Convert.ToInt16(DisableInterruptFlag),
01675         Convert.ToInt16(ZeroFlag),
01676         Convert.ToInt16(CarryFlag));
01677     }
01678
01679     private int WrapProgramCounter(int value)
01680     {
01681         return value & 0xFFFF;
01682     }
01683
01684     private AddressingMode GetAddressingMode()
01685     {
01686         switch (CurrentOpCode)
01687         {
01688             case 0x0D: //ORA
01689             case 0x2D: //AND
01690             case 0x4D: //EOR
01691             case 0x6D: //ADC
01692             case 0x8D: //STA
01693             case 0xAD: //LDA
01694             case 0xCD: //CMP
01695             case 0xED: //SBC
01696             case 0x0E: //ASL
01697             case 0x2E: //ROL
01698             case 0x4E: //LSR
01699             case 0x6E: //ROR
01700             case 0x8E: //SDX
01701             case 0xAE: //LDX
01702             case 0xCE: //DEC
01703             case 0xEE: //INC
01704             case 0x2C: //Bit
01705             case 0x4C: //JMP
01706             case 0x8C: //STY
01707             case 0xAC: //LDY
01708             case 0xCC: //CPY
01709             case 0xEC: //CPX
01710             case 0x20: //JSR
01711             {
01712                 return AddressingMode.Absolute;
01713             }
01714             case 0x1D: //ORA
01715             case 0x3D: //AND
01716             case 0x5D: //EOR
01717             case 0x7D: //ADC
01718             case 0x9D: //STA
01719             case 0xBD: //LDA
01720             case 0xDD: //CMP
01721             case 0xFD: //SBC
01722             case 0xBC: //LDY
01723             case 0xFE: //INC
01724             case 0x1E: //ASL
01725             case 0x3E: //ROL
01726             case 0x5E: //LSR
01727             case 0x7E: //ROR

```

```

01729         {
01730             return AddressingMode.AbsoluteX;
01731         }
01732         case 0x19: //ORA
01733         case 0x39: //AND
01734         case 0x59: //EOR
01735         case 0x79: //ADC
01736         case 0x99: //STA
01737         case 0xB9: //LDA
01738         case 0xD9: //CMP
01739         case 0xF9: //SBC
01740         case 0xBE: //LDX
01741         {
01742             return AddressingMode.AbsoluteY;
01743         }
01744         case 0x0A: //ASL
01745         case 0x4A: //LSR
01746         case 0x2A: //ROL
01747         case 0x6A: //ROR
01748         {
01749             return AddressingMode.Accumulator;
01750         }
01751
01752         case 0x09: //ORA
01753         case 0x29: //AND
01754         case 0x49: //EOR
01755         case 0x69: //ADC
01756         case 0xA0: //LDY
01757         case 0xC0: //CPY
01758         case 0xE0: //CMP
01759         case 0xA2: //LDX
01760         case 0xA9: //LDA
01761         case 0xC9: //CMP
01762         case 0xE9: //SBC
01763         {
01764             return AddressingMode.Immediate;
01765         }
01766         case 0x00: //BRK
01767         case 0x18: //CLC
01768         case 0xD8: //CLD
01769         case 0x58: //CLI
01770         case 0xB8: //CLV
01771         case 0xDE: //DEC
01772         case 0xCA: //DEX
01773         case 0x88: //DEY
01774         case 0xE8: //INX
01775         case 0xC8: //INY
01776         case 0xEA: //NOP
01777         case 0x48: //PHA
01778         case 0x08: //PHP
01779         case 0x68: //PLA
01780         case 0x28: //PLP
01781         case 0x40: //RTI
01782         case 0x60: //RTS
01783         case 0x38: //SEC
01784         case 0xF8: //SED
01785         case 0x78: //SEI
01786         case 0xAA: //TAX
01787         case 0xA8: //TAY
01788         case 0xBA: //TSX
01789         case 0x8A: //TXA
01790         case 0x9A: //TXS
01791         case 0x98: //TYA
01792         {
01793             return AddressingMode.Implied;
01794         }
01795         case 0x6C:
01796         {
01797             return AddressingMode.Indirect;
01798         }
01799
01800         case 0x61: //ADC
01801         case 0x21: //AND
01802         case 0xC1: //CMP
01803         case 0x41: //EOR
01804         case 0xA1: //LDA
01805         case 0x01: //ORA
01806         case 0xE1: //SBC
01807         case 0x81: //STA
01808         {
01809             return AddressingMode.IndirectX;
01810         }
01811         case 0x71: //ADC
01812         case 0x31: //AND
01813         case 0xD1: //CMP
01814         case 0x51: //EOR
01815         case 0xB1: //LDA

```

```

01816         case 0x11: //ORA
01817         case 0xF1: //SBC
01818         case 0x91: //STA
01819         {
01820             return AddressingMode.IndirectY;
01821         }
01822         case 0x90: //BCC
01823         case 0xB0: //BCS
01824         case 0xF0: //BEQ
01825         case 0x30: //BMI
01826         case 0xD0: //BNE
01827         case 0x10: //BPL
01828         case 0x50: //BVC
01829         case 0x70: //BVS
01830         {
01831             return AddressingMode.Relative;
01832         }
01833         case 0x65: //ADC
01834         case 0x25: //AND
01835         case 0x06: //ASL
01836         case 0x24: //BIT
01837         case 0xC5: //CMP
01838         case 0xE4: //CPX
01839         case 0xC4: //CPY
01840         case 0xC6: //DEC
01841         case 0x45: //EOR
01842         case 0xE6: //INC
01843         case 0xA5: //LDA
01844         case 0xA6: //LDX
01845         case 0xA4: //LDY
01846         case 0x46: //LSR
01847         case 0x05: //ORA
01848         case 0x26: //ROL
01849         case 0x66: //ROR
01850         case 0xE5: //SBC
01851         case 0x85: //STA
01852         case 0x86: //STX
01853         case 0x84: //STY
01854         {
01855             return AddressingMode.ZeroPage;
01856         }
01857         case 0x75: //ADC
01858         case 0x35: //AND
01859         case 0x16: //ASL
01860         case 0xD5: //CMP
01861         case 0xD6: //DEC
01862         case 0x55: //EOR
01863         case 0xF6: //INC
01864         case 0xB5: //LDA
01865         case 0xB6: //LDX
01866         case 0xB4: //LDY
01867         case 0x56: //LSR
01868         case 0x15: //ORA
01869         case 0x36: //ROL
01870         case 0x76: //ROR
01871         case 0xF5: //SBC
01872         case 0x95: //STA
01873         case 0x96: //STX
01874         case 0x94: //STY
01875         {
01876             return AddressingMode.ZeroPageX;
01877         }
01878         default:
01879             throw new NotSupportedException(string.Format("Opcode {0} is not supported",
01880                 CurrentOpCode));
01881     }
01882 }
01883 #region Op Code Operations
01884 /// <summary>
01885 /// The ADC - Add Memory to Accumulator with Carry Operation
01886 /// </summary>
01887 /// <param name="addressingMode">The addressing mode used to perform this operation.</param>
01888 protected void AddWithCarryOperation(AddressingMode addressingMode)
01889 {
01890     //Accumulator, Carry = Accumulator + ValueInMemoryLocation + Carry
01891     var memoryValue = MemoryMap.Read(GetAddressByAddressingMode(addressingMode));
01892     var newValue = memoryValue + Accumulator + (CarryFlag ? 1 : 0);
01893
01894     OverflowFlag = (((Accumulator ^ newValue) & 0x80) != 0) && (((Accumulator ^ memoryValue) &
01895         0x80) == 0);
01896
01897     if (DecimalFlag)
01898     {
01899         newValue = int.Parse(memoryValue.ToString("x")) + int.Parse(Accumulator.ToString("x"))
01900             + (CarryFlag ? 1 : 0);

```

```

01900
01901         if (newValue > 99)
01902         {
01903             CarryFlag = true;
01904             newValue -= 100;
01905         }
01906         else
01907         {
01908             CarryFlag = false;
01909         }
01910
01911         newValue = (int)Convert.ToInt64(string.Concat("0x", newValue), 16);
01912     }
01913     else
01914     {
01915         if (newValue > 255)
01916         {
01917             CarryFlag = true;
01918             newValue -= 256;
01919         }
01920         else
01921         {
01922             CarryFlag = false;
01923         }
01924     }
01925
01926     SetZeroFlag(newValue);
01927     SetNegativeFlag(newValue);
01928
01929     Accumulator = newValue;
01930 }
01931
01932 /// <summary>
01933 /// The AND - Compare Memory with Accumulator operation
01934 /// </summary>
01935 /// <param name="addressingMode">The addressing mode being used</param>
01936 private void AndOperation(AddressingMode addressingMode)
01937 {
01938     Accumulator = MemoryMap.Read(GetAddressByAddressingMode(addressingMode)) & Accumulator;
01939
01940     SetZeroFlag(Accumulator);
01941     SetNegativeFlag(Accumulator);
01942 }
01943
01944 /// <summary>
01945 /// The ASL - Shift Left One Bit (Memory or Accumulator)
01946 /// </summary>
01947 /// <param name="addressingMode">The addressing Mode being used</param>
01948 public void AslOperation(AddressingMode addressingMode)
01949 {
01950     int value;
01951     var memoryAddress = 0;
01952     if (addressingMode == AddressingMode.Accumulator)
01953     {
01954         MemoryMap.Read(ProgramCounter + 1);
01955         value = Accumulator;
01956     }
01957     else
01958     {
01959         memoryAddress = GetAddressByAddressingMode(addressingMode);
01960         value = MemoryMap.Read(memoryAddress);
01961     }
01962
01963     //Dummy Write
01964     if (addressingMode != AddressingMode.Accumulator)
01965     {
01966         MemoryMap.Write(memoryAddress, (byte)value);
01967     }
01968
01969     //If the 7th bit is set, then we have a carry
01970     CarryFlag = ((value & 0x80) != 0);
01971
01972     //The And here ensures that if the value is greater than 255 it wraps properly.
01973     value = (value << 1) & 0xFE;
01974
01975     SetNegativeFlag(value);
01976     SetZeroFlag(value);
01977
01978
01979     if (addressingMode == AddressingMode.Accumulator)
01980         Accumulator = value;
01981     else
01982     {
01983         MemoryMap.Write(memoryAddress, (byte)value);
01984     }
01985 }
01986

```

```

01987 /// <summary>
01988 /// Performs the different branch operations.
01989 /// </summary>
01990 /// <param name="performBranch">Is a branch required</param>
01991 private void BranchOperation(bool performBranch)
01992 {
01993     var value = MemoryMap.Read(GetAddressByAddressingMode(AddressingMode.Relative));
01994
01995     if (!performBranch)
01996     {
01997         ProgramCounter++;
01998         return;
01999     }
02000
02001     MoveProgramCounterByRelativeValue(value);
02002 }
02003
02004 /// <summary>
02005 /// The bit operation, does an & comparison between a value in memory and the accumulator
02006 /// </summary>
02007 /// <param name="addressingMode"></param>
02008 private void BitOperation(AddressingMode addressingMode)
02009 {
02010
02011     var memoryValue = MemoryMap.Read(GetAddressByAddressingMode(addressingMode));
02012     var valueToCompare = memoryValue & Accumulator;
02013
02014     OverflowFlag = (memoryValue & 0x40) != 0;
02015
02016     SetNegativeFlag(memoryValue);
02017     SetZeroFlag(valueToCompare);
02018 }
02019
02020 /// <summary>
02021 /// The compare operation. This operation compares a value in memory with a value passed into it.
02022 /// </summary>
02023 /// <param name="addressingMode">The addressing mode to use</param>
02024 /// <param name="comparisonValue">The value to compare against memory</param>
02025 private void CompareOperation(AddressingMode addressingMode, int comparisonValue)
02026 {
02027     var memoryValue = MemoryMap.Read(GetAddressByAddressingMode(addressingMode));
02028     var comparedValue = comparisonValue - memoryValue;
02029
02030     if (comparedValue < 0)
02031         comparedValue += 0x10000;
02032
02033     SetZeroFlag(comparedValue);
02034
02035     CarryFlag = memoryValue <= comparisonValue;
02036     SetNegativeFlag(comparedValue);
02037 }
02038
02039 /// <summary>
02040 /// Changes a value in memory by 1
02041 /// </summary>
02042 /// <param name="addressingMode">The addressing mode to use</param>
02043 /// <param name="decrement">If the operation is decrementing or incrementing the vaulue by 1</param>
02044 private void ChangeMemoryByOne(AddressingMode addressingMode, bool decrement)
02045 {
02046     var memoryLocation = GetAddressByAddressingMode(addressingMode);
02047     var memory = MemoryMap.Read(memoryLocation);
02048
02049     MemoryMap.Write(memoryLocation, memory);
02050
02051     if (decrement)
02052         memory -= 1;
02053     else
02054         memory += 1;
02055
02056     SetZeroFlag(memory);
02057     SetNegativeFlag(memory);
02058
02059     MemoryMap.Write(memoryLocation, memory);
02060 }
02061
02062
02063 /// <summary>
02064 /// Changes a value in either the X or Y register by 1
02065 /// </summary>
02066 /// <param name="useXRegister">If the operation is using the X or Y register</param>
02067 /// <param name="decrement">If the operation is decrementing or incrementing the vaulue by 1</param>
02068 private void ChangeRegisterByOne(bool useXRegister, bool decrement)
02069 {
02070     var value = useXRegister ? XRegister : YRegister;
02071
02072     if (decrement)
02073         value -= 1;

```

```

02074         else
02075             value += 1;
02076
02077         if (value < 0x00)
02078             value += 0x100;
02079         else if (value > 0xFF)
02080             value -= 0x100;
02081
02082         SetZeroFlag(value);
02083         SetNegativeFlag(value);
02084         IncrementCycleCount();
02085
02086         if (useXRegister)
02087             XRegister = value;
02088         else
02089             YRegister = value;
02090     }
02091
02092     /// <summary>
02093     /// The EOR Operation, Performs an Exclusive OR Operation against the Accumulator and a value in
02094     /// memory
02095     /// </summary>
02096     /// <param name="addressingMode">The addressing mode to use</param>
02097     private void EorOperation(AddressingMode addressingMode)
02098     {
02099         Accumulator = Accumulator ^ MemoryMap.Read(GetAddressByAddressingMode(addressingMode));
02100
02101         SetNegativeFlag(Accumulator);
02102         SetZeroFlag(Accumulator);
02103     }
02104
02105     /// <summary>
02106     /// The LSR Operation. Performs a Left shift operation on a value in memory
02107     /// </summary>
02108     /// <param name="addressingMode">The addressing mode to use</param>
02109     private void LsrOperation(AddressingMode addressingMode)
02110     {
02111         int value;
02112         var memoryAddress = 0;
02113         if (addressingMode == AddressingMode.Accumulator)
02114         {
02115             MemoryMap.Read(ProgramCounter + 1);
02116             value = Accumulator;
02117         }
02118         else
02119         {
02120             memoryAddress = GetAddressByAddressingMode(addressingMode);
02121             value = MemoryMap.Read(memoryAddress);
02122         }
02123
02124         //Dummy Write
02125         if (addressingMode != AddressingMode.Accumulator)
02126         {
02127             MemoryMap.Write(memoryAddress, (byte)value);
02128         }
02129
02130         NegativeFlag = false;
02131
02132         //If the Zero bit is set, we have a carry
02133         CarryFlag = (value & 0x01) != 0;
02134
02135         value = (value » 1);
02136
02137         SetZeroFlag(value);
02138         if (addressingMode == AddressingMode.Accumulator)
02139             Accumulator = value;
02140         else
02141         {
02142             MemoryMap.Write(memoryAddress, (byte)value);
02143         }
02144     }
02145
02146     /// <summary>
02147     /// The Or Operation. Performs an Or Operation with the accumulator and a value in memory
02148     /// </summary>
02149     /// <param name="addressingMode">The addressing mode to use</param>
02150     private void OrOperation(AddressingMode addressingMode)
02151     {
02152         Accumulator = Accumulator | MemoryMap.Read(GetAddressByAddressingMode(addressingMode));
02153
02154         SetNegativeFlag(Accumulator);
02155         SetZeroFlag(Accumulator);
02156     }
02157
02158     /// <summary>
02159     /// The ROL operation. Performs a rotate left operation on a value in memory.
02160     /// </summary>

```



```

02160 /// <param name="addressingMode">The addressing mode to use</param>
02161 private void RolOperation(AddressingMode addressingMode)
02162 {
02163     int value;
02164     var memoryAddress = 0;
02165     if (addressingMode == AddressingMode.Accumulator)
02166     {
02167         //Dummy MemoryMap.Read
02168         MemoryMap.Read(ProgramCounter + 1);
02169         value = Accumulator;
02170     }
02171     else
02172     {
02173         memoryAddress = GetAddressByAddressingMode(addressingMode);
02174         value = MemoryMap.Read(memoryAddress);
02175     }
02176
02177     //Dummy Write
02178     if (addressingMode != AddressingMode.Accumulator)
02179     {
02180         MemoryMap.Write(memoryAddress, (byte)value);
02181     }
02182
02183     //Store the carry flag before shifting it
02184     var newCarry = (0x80 & value) != 0;
02185
02186     //The And here ensures that if the value is greater than 255 it wraps properly.
02187     value = (value << 1) & 0xFE;
02188
02189     if (CarryFlag)
02190         value = value | 0x01;
02191
02192     CarryFlag = newCarry;
02193
02194     SetZeroFlag(value);
02195     SetNegativeFlag(value);
02196
02197     if (addressingMode == AddressingMode.Accumulator)
02198         Accumulator = value;
02199     else
02200     {
02201         MemoryMap.Write(memoryAddress, (byte)value);
02202     }
02203 }
02204
02205
02206 /// <summary>
02207 /// The ROR operation. Performs a rotate right operation on a value in memory.
02208 /// </summary>
02209 /// <param name="addressingMode">The addressing mode to use</param>
02210 private void RorOperation(AddressingMode addressingMode)
02211 {
02212     int value;
02213     var memoryAddress = 0;
02214     if (addressingMode == AddressingMode.Accumulator)
02215     {
02216         //Dummy MemoryMap.Read
02217         MemoryMap.Read(ProgramCounter + 1);
02218         value = Accumulator;
02219     }
02220     else
02221     {
02222         memoryAddress = GetAddressByAddressingMode(addressingMode);
02223         value = MemoryMap.Read(memoryAddress);
02224     }
02225
02226     //Dummy Write
02227     if (addressingMode != AddressingMode.Accumulator)
02228     {
02229         MemoryMap.Write(memoryAddress, (byte)value);
02230     }
02231
02232     //Store the carry flag before shifting it
02233     var newCarry = (0x01 & value) != 0;
02234
02235     value = (value >> 1);
02236
02237     //If the carry flag is set then 0x
02238     if (CarryFlag)
02239         value = value | 0x80;
02240
02241     CarryFlag = newCarry;
02242
02243     SetZeroFlag(value);
02244     SetNegativeFlag(value);
02245
02246     if (addressingMode == AddressingMode.Accumulator)

```

```

02247         Accumulator = value;
02248     else
02249     {
02250         MemoryMap.Write(memoryAddress, (byte)value);
02251     }
02252 }
02253
02254 /// <summary>
02255 /// The SBC operation. Performs a subtract with carry operation on the accumulator and a value in
memory.
02256 /// </summary>
02257 /// <param name="addressingMode">The addressing mode to use</param>
02258 protected void SubtractWithBorrowOperation(AddressingMode addressingMode)
02259 {
02260     var memoryValue = MemoryMap.Read(GetAddressByAddressingMode(addressingMode));
02261     var newValue = DecimalFlag ? int.Parse(Accumulator.ToString("x")) -
int.Parse(memoryValue.ToString("x")) - (CarryFlag ? 0 : 1) : Accumulator - memoryValue - (CarryFlag
? 0 : 1);
02262
02263     CarryFlag = newValue >= 0;
02264
02265     if (DecimalFlag)
02266     {
02267         if (newValue < 0)
02268             newValue += 100;
02269
02270         newValue = (int)Convert.ToInt64(string.Concat("0x", newValue), 16);
02271     }
02272     else
02273     {
02274         OverflowFlag = (((Accumulator ^ newValue) & 0x80) != 0) && (((Accumulator ^
memoryValue) & 0x80) != 0);
02275
02276         if (newValue < 0)
02277             newValue += 256;
02278     }
02279
02280     SetNegativeFlag(newValue);
02281     SetZeroFlag(newValue);
02282
02283     Accumulator = newValue;
02284 }
02285
02286 /// <summary>
02287 /// The PSP Operation. Pushes the Status Flags to the stack
02288 /// </summary>
02289 private void PushFlagsOperation()
02290 {
02291     PokeStack(ConvertFlagsToByte(true));
02292 }
02293
02294 /// <summary>
02295 /// The PLP Operation. Pull the status flags off the stack on sets the flags accordingly.
02296 /// </summary>
02297 private void PullFlagsOperation()
02298 {
02299     var flags = PeekStack();
02300     CarryFlag = (flags & 0x01) != 0;
02301     ZeroFlag = (flags & 0x02) != 0;
02302     DisableInterruptFlag = (flags & 0x04) != 0;
02303     DecimalFlag = (flags & 0x08) != 0;
02304     OverflowFlag = (flags & 0x40) != 0;
02305     NegativeFlag = (flags & 0x80) != 0;
02306
02307 }
02308
02309
02310 /// <summary>
02311 /// The JSR routine. Jumps to a subroutine.
02312 /// </summary>
02313 private void JumpToSubRoutineOperation()
02314 {
02315     IncrementCycleCount();
02316
02317     //Put the high value on the stack, this should be the address after our operation -1
02318     //The RTS operation increments the PC by 1 which is why we don't move 2
02319     PokeStack((byte)((ProgramCounter + 1) >> 8) & 0xFF));
02320     StackPointer--;
02321     IncrementCycleCount();
02322
02323     PokeStack((byte)((ProgramCounter + 1) & 0xFF));
02324     StackPointer--;
02325     IncrementCycleCount();
02326
02327     ProgramCounter = GetAddressByAddressingMode(AddressingMode.Absolute);
02328 }
02329

```

```

02330 /// <summary>
02331 /// The RTS routine. Called when returning from a subroutine.
02332 /// </summary>
02333 private void ReturnFromSubRoutineOperation()
02334 {
02335     MemoryMap.Read(++ProgramCounter);
02336     StackPointer++;
02337     IncrementCycleCount();
02338
02339     var lowBit = PeekStack();
02340     StackPointer++;
02341     IncrementCycleCount();
02342
02343     var highBit = PeekStack() << 8;
02344     IncrementCycleCount();
02345
02346     ProgramCounter = (highBit | lowBit) + 1;
02347     IncrementCycleCount();
02348 }
02349
02350
02351 /// <summary>
02352 /// The BRK routine. Called when a BRK occurs.
02353 /// </summary>
02354 private void BreakOperation(bool isBrk, int vector)
02355 {
02356     MemoryMap.Read(++ProgramCounter);
02357
02358     //Put the high value on the stack
02359     //When we RTI the address will be incremented by one, and the address after a break will
not be used.
02360     PokeStack((byte)((ProgramCounter) >> 8) & 0xFF);
02361     StackPointer--;
02362     IncrementCycleCount();
02363
02364     //Put the low value on the stack
02365     PokeStack((byte)((ProgramCounter) & 0xFF));
02366     StackPointer--;
02367     IncrementCycleCount();
02368
02369     //We only set the Break Flag is a Break Occurs
02370     if (isBrk)
02371         PokeStack((byte)(ConvertFlagsToByte(true) | 0x10));
02372     else
02373         PokeStack(ConvertFlagsToByte(false));
02374
02375     StackPointer--;
02376     IncrementCycleCount();
02377
02378     DisableInterruptFlag = true;
02379
02380     ProgramCounter = (MemoryMap.Read(vector + 1) << 8) | MemoryMap.Read(vector);
02381
02382     _previousInterrupt = false;
02383 }
02384
02385 /// <summary>
02386 /// The RTI routine. Called when returning from a BRK operation.
02387 /// Note: when called after a BRK operation the Program Counter is not set to the location after the
BRK,
02388 /// it is set +1
02389 /// </summary>
02390 private void ReturnFromInterruptOperation()
02391 {
02392     MemoryMap.Read(++ProgramCounter);
02393     StackPointer++;
02394     IncrementCycleCount();
02395
02396     PullFlagsOperation();
02397     StackPointer++;
02398     IncrementCycleCount();
02399
02400     var lowBit = PeekStack();
02401     StackPointer++;
02402     IncrementCycleCount();
02403
02404     var highBit = PeekStack() << 8;
02405     IncrementCycleCount();
02406
02407     ProgramCounter = (highBit | lowBit);
02408 }
02409
02410 /// <summary>
02411 /// This is ran anytime an NMI occurs
02412 /// </summary>
02413 private void ProcessNMI()
02414 {

```

```

02415         ProgramCounter--;
02416         BreakOperation(false, 0xFFFA);
02417         CurrentOpCode = MemoryMap.Read(ProgramCounter);
02418
02419         SetDisassembly();
02420     }
02421
02422     /// <summary>
02423     /// This is ran anytime an IRQ occurs
02424     /// </summary>
02425     private void ProcessIRQ()
02426     {
02427         if (DisableInterruptFlag)
02428             return;
02429
02430         ProgramCounter--;
02431         BreakOperation(false, 0xFFFE);
02432         CurrentOpCode = MemoryMap.Read(ProgramCounter);
02433
02434         SetDisassembly();
02435     }
02436 #endregion
02437
02438 #endregion
02439     }
02440 }

```

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.](#)

```

00001 using System;
00002 using System.IO;
00003 using System.Timers;
00004
00005 namespace Hardware
00006 {
00007     /// <summary>
00008     /// An implementation of a W65C22 VIA.
00009     /// </summary>
00010     [Serializable]
00011     public class W65C22
00012     {
00013         #region Fields
00014         public readonly bool T1IsIRQ = false;
00015         public readonly bool T2IsIRQ = true;
00016         public int T1CL = 0x04;
00017         public int T1CH = 0x05;
00018         public int T2CL = 0x08;
00019         public int T2CH = 0x09;
00020         public int ACR = 0x0B;
00021         public int IFR = 0x0D;
00022         public int IER = 0x0E;
00023
00024         public byte ACR_T1TC = (byte) (1 << 7);
00025         public byte ACR_T2TC = (byte) (1 << 6);
00026
00027         public byte IFR_T2 = (byte) (1 << 5);
00028         public byte IFR_T1 = (byte) (1 << 6);
00029         public byte IFR_INT = (byte) (1 << 7);
00030
00031         public byte IER_T2 = (byte) (1 << 5);

```

```

00032         public byte IER_T1 = (byte)(1 << 6);
00033         public byte IER_EN = (byte)(1 << 7);
00034 #endregion
00035
00036 #region Properties
00037 /// <summary>
00038 /// The memory area.
00039 /// </summary>
00040         public byte[] Memory { get; set; }
00041
00042 /// <summary>
00043 /// The memory offset of the device.
00044 /// </summary>
00045         public int Offset { get; set; }
00046
00047 /// <summary>
00048 /// The length of the device memory.
00049 /// </summary>
00050         public int Length { get; set; }
00051
00052 /// <summary>
00053 /// The end of memory
00054 /// </summary>
00055         public int End { get { return Offset + Length; } }
00056
00057 /// <summary>
00058 /// T1 timer control
00059 /// </summary>
00060         public bool T1TimerControl
00061         {
00062             get { return T1Object.AutoReset; }
00063             set { T1Object.AutoReset = value; }
00064         }
00065
00066 /// <summary>
00067 /// T2 timer control.
00068 /// </summary>
00069         public bool T2TimerControl
00070         {
00071             get { return T2Object.AutoReset; }
00072             set { T2Object.AutoReset = value; }
00073         }
00074
00075 /// <summary>
00076 /// Enable or check whether timer 1 is enabled or not.
00077 /// </summary>
00078         public bool T1IsEnabled
00079         {
00080             get { return T1Object.Enabled; }
00081             set { T1Object.Enabled = value; }
00082         }
00083
00084 /// <summary>
00085 /// Enable or check whether timer 2 is enabled or not.
00086 /// </summary>
00087         public bool T2IsEnabled
00088         {
00089             get { return T2Object.Enabled; }
00090             set { T2Object.Enabled = value; }
00091         }
00092
00093 /// <summary>
00094 /// Set or check the timer 1 interval.
00095 /// </summary>
00096         public double T1Interval { get { return (int)(Read(T1CL) | (Read(T1CH) << 8)); } }
00097
00098 /// <summary>
00099 /// Set or check the timer 2 interval.
00100 /// </summary>
00101         public double T2Interval
00102         {
00103             get { return (int)(Read(T2CL) | (Read(T2CH) << 8)); }
00104         }
00105
00106 /// <summary>
00107 /// Set or get the timer 1 object.
00108 /// </summary>
00109         public Timer T1Object { get; set; }
00110
00111 /// <summary>
00112 /// Set or get the timer 2 object.
00113 /// </summary>
00114         public Timer T2Object { get; set; }
00115
00116 /// <summary>
00117 /// Local referemce to the processor object.
00118 /// </summary>

```

```

00119         private W65C02 Processor { get; set; }
00120 #endregion
00121
00122 #region Public Methods
00123     public W65C22(W65C02 processor, byte offset, int length)
00124     {
00125         if (offset > MemoryMap.DeviceArea.Length)
00126             throw new ArgumentException(String.Format("The offset: {0} is greater than the device
area: {1}", offset, MemoryMap.DeviceArea.Length));
00127         T1Init(1000);
00128         T2Init(1000);
00129
00130         Offset = MemoryMap.DeviceArea.Offset | offset;
00131         Memory = new byte[length + 1];
00132         Length = length;
00133         Processor = processor;
00134     }
00135
00136 /// <summary>
00137 /// Reset routine called whenever the emulated computer is reset.
00138 /// </summary>
00139     public void Reset()
00140     {
00141         T1TimerControl = false;
00142         T1IsEnabled = false;
00143         T2TimerControl = false;
00144         T2IsEnabled = false;
00145     }
00146
00147 /// <summary>
00148 /// Initialization routine for the VIA.
00149 /// </summary>
00150 /// <param name="timer">Amount of time to set timers for.</param>
00151     public void Init(double timer)
00152     {
00153         T1Init(timer);
00154         T2Init(timer);
00155     }
00156
00157 /// <summary>
00158 /// T1 counter initialization routine.
00159 /// </summary>
00160 ///
00161 /// <param name="value">Timer initialization value in milliseconds.</param>
00162     public void T1Init(double value)
00163     {
00164         T1Object = new Timer(value);
00165         T1Object.Start();
00166         T1Object.Elapsed += OnT1Timeout;
00167         T1TimerControl = true;
00168         T1IsEnabled = false;
00169     }
00170
00171 /// <summary>
00172 /// T2 counter initialization routine.
00173 /// </summary>
00174 ///
00175 /// <param name="value">Timer initialization value in milliseconds.</param>
00176     public void T2Init(double value)
00177     {
00178         T2Object = new Timer(value);
00179         T2Object.Start();
00180         T2Object.Elapsed += OnT2Timeout;
00181         T2TimerControl = true;
00182         T2IsEnabled = false;
00183     }
00184
00185 /// <summary>
00186 /// Routine to read from local memory.
00187 /// </summary>
00188 ///
00189 /// <param name="address">Address to read from.</param>
00190 ///
00191 /// <returns>Byte value stored in the local memory.</returns>
00192     public byte Read(int address)
00193     {
00194         if ((Offset <= address) && (address <= End))
00195         {
00196             byte data = 0x00;
00197             if (T1TimerControl)
00198             {
00199                 data = (byte)(data | ACR_T1TC);
00200             }
00201             else if (T2TimerControl)
00202             {
00203                 data = (byte)(data | ACR_T2TC);
00204             }
00205         }
00206     }

```

```

00205         return data;
00206     }
00207     else
00208     {
00209         return Memory[address - Offset];
00210     }
00211 }
00212
00213 /// <summary>
00214 /// Writes data to the specified address in local memory.
00215 /// </summary>
00216 ///
00217 /// <param name="address">The address to write data to.</param>
00218 /// <param name="data">The data to be written.</param>
00219 public void Write(int address, byte data)
00220 {
00221     if ((address == Offset + ACR) && ((data | ACR_T1TC) == ACR_T1TC))
00222     {
00223         T1TimerControl = true;
00224     }
00225     else if ((address == Offset + ACR) && ((data | ACR_T2TC) == ACR_T2TC))
00226     {
00227         T2TimerControl = true;
00228     }
00229     else if ((address == Offset + IER) && ((data | IER_T1) == IER_T1) && ((data | IER_EN) ==
IER_EN))
00230     {
00231         T1Init(T1Interval);
00232     }
00233     else if ((address == Offset + IER) && ((data | IER_T2) == IER_T2) && ((data | IER_EN) ==
IER_EN))
00234     {
00235         T2Init(T2Interval);
00236     }
00237     Memory[address - Offset] = data;
00238 }
00239 #endregion
00240
00241 #region Private Methods
00242 /// <summary>
00243 /// Called whenever System.Timers.Timer event elapses.
00244 /// </summary>
00245 ///
00246 /// <param name="sender"></param>
00247 /// <param name="e"></param>
00248 private void OnT1Timeout(object sender, ElapsedEventArgs e)
00249 {
00250     if (Processor.isRunning)
00251     {
00252         if (T1IsEnabled)
00253         {
00254             Write(IFR, (byte)(IFR_T1 & IFR_INT));
00255             if (T1IsIRQ)
00256             {
00257                 Processor.InterruptRequest();
00258             }
00259             else
00260             {
00261                 Processor.TriggerNmi = true;
00262             }
00263         }
00264     }
00265 }
00266
00267 /// <summary>
00268 /// Called whenever System.Timers.Timer event elapses
00269 /// </summary>
00270 ///
00271 /// <param name="sender"></param>
00272 /// <param name="e"></param>
00273 private void OnT2Timeout(object sender, ElapsedEventArgs e)
00274 {
00275     if (Processor.isRunning)
00276     {
00277         if (T2IsEnabled)
00278         {
00279             Write(IFR, (byte)(IFR_T2 & IFR_INT));
00280             if (T2IsIRQ)
00281             {
00282                 Processor.InterruptRequest();
00283             }
00284             else
00285             {
00286                 Processor.TriggerNmi = true;
00287             }
00288         }
00289     }

```

```

00290     }
00291 #endregion
00292     }
00293 }

```

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.](#)

```

00001 using System;
00002 using System.Collections.Generic;
00003 using System.ComponentModel;
00004 using System.IO;
00005 using System.IO.Ports;
00006
00007 namespace Hardware
00008 {
00009     /// <summary>
00010     /// An implementation of a W65C51 ACIA.
00011     /// </summary>
00012     [Serializable]
00013     public class W65C51
00014     {
00015         #region Fields
00016         public readonly int defaultBaudRate = 115200;
00017         public byte byteIn;
00018 #endregion
00019
00020 #region Properties
00021         public byte[] Memory { get; set; }
00022         public bool IsEnabled { get; set; }
00023         public SerialPort Object { get; set; }
00024         public string ObjectName { get; set; }
00025         private W65C02 Processor { get; set; }
00026         private BackgroundWorker _backgroundWorker { get; set; }
00027         public int Offset { get; set; }
00028         public int Length { get; set; }
00029
00030         private bool DataRead { get; set; }
00031         private bool EchoMode { get; set; }
00032         private bool InterruptDisabled { get; set; }
00033         private bool Interrupted { get; set; }
00034         private bool Overrun { get; set; }
00035         private bool ParityEnabled { get; set; }
00036         private bool ReceiverFull { get; set; }
00037         private byte RtsControl { get; set; }
00038 #endregion
00039
00040 #region Public Methods
00041         public W65C51(W65C02 processor, byte offset)
00042         {
00043             if (offset > MemoryMap.DeviceArea.Length)
00044                 throw new ArgumentException(String.Format("The offset: {0} is greater than the device
area: {1}", offset, MemoryMap.DeviceArea.Length));
00045
00046             Processor = processor;
00047
00048             Offset = MemoryMap.DeviceArea.Offset | offset;
00049             Length = 0x04;
00050             Memory = new byte[Length + 1];
00051
00052             _backgroundWorker = new BackgroundWorker

```



```

00053         {
00054             WorkerSupportsCancellation = true
00055         };
00056         _backgroundWorker.DoWork += BackgroundWorkerDoWork;
00057         _backgroundWorker.RunWorkerAsync();
00058     }
00059
00060     public void Reset()
00061     {
00062         IsEnabled = false;
00063     }
00064
00065     /// <summary>
00066     /// Default Constructor, Instantiates a new instance of COM Port I/O.
00067     /// </summary>
00068     ///
00069     /// <param name="port"> COM Port to use for I/O</param>
00070     public void Init(string port)
00071     {
00072         Object = new SerialPort(port, defaultBaudRate, Parity.None, 8, StopBits.One);
00073         ObjectName = port;
00074
00075         ComInit(Object);
00076     }
00077
00078     /// <summary>
00079     /// Default Constructor, Instantiates a new instance of COM Port I/O.
00080     /// </summary>
00081     ///
00082     /// <param name="port">COM Port to use for I/O</param>
00083     /// <param name="baudRate">Baud Rate to use for I/O</param>
00084     public void Init(string port, int baudRate)
00085     {
00086         Object = new SerialPort(port, baudRate, Parity.None, 8, StopBits.One);
00087         ObjectName = port;
00088
00089         ComInit(Object);
00090     }
00091
00092     /// <summary>
00093     /// Called when the window is closed.
00094     /// </summary>
00095     public void Fini()
00096     {
00097         ComFini(Object);
00098     }
00099
00100     /// <summary>
00101     /// Returns the byte at a given address.
00102     /// </summary>
00103     ///
00104     /// <param name="address"></param>
00105     ///
00106     /// <returns>the byte being returned</returns>
00107     public byte Read(int address)
00108     {
00109         HardwarePreRead(address);
00110         byte data = Memory[address - Offset];
00111         DataRead = true;
00112         return data;
00113     }
00114
00115     /// <summary>
00116     /// Writes data to the given address.
00117     /// </summary>
00118     ///
00119     /// <param name="address">The address to write data to</param>
00120     /// <param name="data">The data to write</param>
00121     public void Write(int address, byte data)
00122     {
00123         HardwarePreWrite(address, data);
00124         if (!(address == Offset) || (address == Offset + 1))
00125         {
00126             Memory[address - Offset] = data;
00127         }
00128     }
00129
00130     /// <summary>
00131     /// Called in order to write to the serial port.
00132     /// </summary>
00133     ///
00134     /// <param name="data">Byte of data to send</param>
00135     public void WriteCOM(byte data)
00136     {
00137         byte[] writeByte = new byte[] { data };
00138         Object.Write(writeByte, 0, 1);
00139     }

```

```

00140 #endregion
00141
00142 #region Private Methods
00143 /// <summary>
00144 /// Called whenever the ACIA is initialized.
00145 /// </summary>
00146 ///
00147 /// <param name="serialPort">SerialPort object to initialize.</param>
00148 private void ComInit(SerialPort serialPort)
00149 {
00150     try
00151     {
00152         serialPort.Open();
00153     }
00154     catch (UnauthorizedAccessException w)
00155     {
00156         FileStream file = new FileStream(FileLocations.ErrorFile, FileMode.OpenOrCreate,
FileAccess.ReadWrite);
00157         StreamWriter stream = new StreamWriter(file);
00158         stream.WriteLine(w.Message);
00159         stream.WriteLine(w.Source);
00160         stream.Flush();
00161         file.Flush();
00162         stream.Close();
00163         file.Close();
00164         return;
00165     }
00166     serialPort.ReadTimeout = 50;
00167     serialPort.WriteTimeout = 50;
00168     serialPort.DataReceived += new SerialDataReceivedEventHandler(SerialDataReceived);
00169     try
00170     {
00171         serialPort.Write("-----\r\n");
00172         serialPort.Write(" WolfNet 6502 WBC Emulator\r\n");
00173         serialPort.Write("-----\r\n");
00174         serialPort.Write("\r\n");
00175     }
00176     catch (TimeoutException t)
00177     {
00178         _ = t;
00179         FileStream file = new FileStream(FileLocations.ErrorFile, FileMode.OpenOrCreate,
FileAccess.ReadWrite);
00180         StreamWriter stream = new StreamWriter(file);
00181         stream.WriteLine("Read/Write error: Port timed out!");
00182         stream.WriteLine("Please ensure all cables are connected properly!");
00183         stream.Flush();
00184         file.Flush();
00185         stream.Close();
00186         file.Close();
00187         return;
00188     }
00189 }
00190
00191 /// <summary>
00192 /// Called when the window is closed.
00193 /// </summary>
00194 ///
00195 /// <param name="serialPort">SerialPort Object to close</param>
00196 private void ComFini(SerialPort serialPort)
00197 {
00198     if (serialPort != null)
00199     {
00200         serialPort.Close();
00201     }
00202
00203     _backgroundWorker.CancelAsync();
00204     _backgroundWorker.DoWork -= BackgroundWorkerDoWork;
00205 }
00206
00207 /// <summary>
00208 /// Called whenever SerialDataReceivedEventHandler event occurs.
00209 /// </summary>
00210 ///
00211 /// <param name="sender"></param>
00212 /// <param name="e"></param>
00213 private void SerialDataReceived(object sender, SerialDataReceivedEventArgs e)
00214 {
00215     try
00216     {
00217         if (EchoMode)
00218         {
00219             WriteCOM(Convert.ToByte(Object.ReadByte()));
00220         }
00221         else
00222         {
00223             if (!ReceiverFull)
00224             {

```

```

00225             ReceiverFull = true;
00226         }
00227         else
00228         {
00229             Overrun = true;
00230         }
00231         Memory[0] = Convert.ToByte(Object.ReadByte());
00232     }
00233
00234     if (!InterruptDisabled)
00235     {
00236         Interrupted = true;
00237         Processor.InterruptRequest();
00238     }
00239 }
00240 catch (Win32Exception w)
00241 {
00242     FileStream file = new FileStream(FileLocations.ErrorFile, FileMode.OpenOrCreate,
FileAccess.ReadWrite);
00243     StreamWriter stream = new StreamWriter(file);
00244     stream.WriteLine(w.Message);
00245     stream.WriteLine(w.ErrorCode.ToString());
00246     stream.WriteLine(w.Source);
00247     stream.Flush();
00248     stream.Close();
00249     file.Flush();
00250     file.Close();
00251 }
00252 }
00253
00254 private void HardwarePreWrite(int address, byte data)
00255 {
00256     if (address == Offset)
00257     {
00258         WriteCOM(data);
00259     }
00260     else if (address == Offset + 1)
00261     {
00262         Reset();
00263     }
00264     else if (address == Offset + 2)
00265     {
00266         CommandRegister(data);
00267     }
00268     else if (address == Offset + 3)
00269     {
00270         ControlRegister(data);
00271     }
00272 }
00273
00274 private void HardwarePreRead(int address)
00275 {
00276     if (address == Offset)
00277     {
00278         Interrupted = false;
00279         Overrun = false;
00280         ReceiverFull = false;
00281     }
00282     else if (address == Offset + 1)
00283     {
00284         StatusRegisterUpdate();
00285     }
00286     else if (address == Offset + 2)
00287     {
00288         CommandRegisterUpdate();
00289     }
00290     else if (address == Offset + 3)
00291     {
00292         ControlRegisterUpdate();
00293     }
00294 }
00295 }
00296
00297 private void CommandRegister(byte data)
00298 {
00299     byte test = (byte)(data & 0x20);
00300     if (test == 0x20)
00301     {
00302         throw new ArgumentException("Parity must NEVER be enabled!");
00303     }
00304
00305     test = (byte)(data & 0x10);
00306     if (test == 0x10)
00307     {
00308         EchoMode = true;
00309     }
00310     else

```

```
00311         {
00312             EchoMode = false;
00313         }
00314
00315         test = (byte)(data & 0x0C);
00316         if (test == 0x00)
00317         {
00318             Object.Handshake = Handshake.None;
00319             Object.RtsEnable = true;
00320             Object.Handshake = Handshake.RequestToSend;
00321         }
00322         else if (test == 0x04)
00323         {
00324             Object.Handshake = Handshake.None;
00325             Object.RtsEnable = false;
00326         }
00327         else if ((test == 0x08) || (test == 0x0C))
00328         {
00329             throw new NotImplementedException("This cannot be emulated on windows!");
00330         }
00331         else
00332         {
00333             throw new ArgumentOutOfRangeException("RtsControl is invalid!");
00334         }
00335
00336         test = (byte)(data & 0x02);
00337         if (test == 0x02)
00338         {
00339             InterruptDisabled = true;
00340         }
00341         else
00342         {
00343             InterruptDisabled = false;
00344         }
00345
00346         test = (byte)(data & 0x01);
00347         if (test == 0x01)
00348         {
00349             Object.DtrEnable = true;
00350         }
00351         else
00352         {
00353             Object.DtrEnable = false;
00354         }
00355     }
00356
00357     private void CommandRegisterUpdate()
00358     {
00359         byte data = Memory[Offset + 2];
00360
00361         if (ParityEnabled)
00362         {
00363             data |= 0x20;
00364         }
00365         else
00366         {
00367             data &= 0xD0;
00368         }
00369
00370         if (EchoMode)
00371         {
00372             data |= 0x10;
00373         }
00374         else
00375         {
00376             data &= 0xE0;
00377         }
00378
00379         data &= RtsControl;
00380
00381         if (InterruptDisabled)
00382         {
00383             data |= 0x02;
00384         }
00385         else
00386         {
00387             data &= 0x0D;
00388         }
00389         if (Object.DtrEnable)
00390         {
00391             data |= 0x01;
00392         }
00393         else
00394         {
00395             data &= 0x0E;
00396         }
00397     }
```

```

00398         Memory[Offset + 2] = data;
00399     }
00400
00401     private void ControlRegister(byte data)
00402     {
00403         byte test = (byte)(data & 0x80);
00404         if (test == 0x80)
00405         {
00406             test = (byte)(data & 0x60);
00407             if (test == 0x60)
00408             {
00409                 Object.StopBits = StopBits.OnePointFive;
00410             }
00411             else
00412             {
00413                 Object.StopBits = StopBits.Two;
00414             }
00415         }
00416         else
00417         {
00418             Object.StopBits = StopBits.One;
00419         }
00420
00421         test = (byte)(data & 0x60);
00422         if (test == 0x20)
00423         {
00424             Object.DataBits = 7;
00425         }
00426         else if (test == 0x40)
00427         {
00428             Object.DataBits = 6;
00429         }
00430         else if (test == 0x60)
00431         {
00432             Object.DataBits = 5;
00433         }
00434         else
00435         {
00436             Object.DataBits = 8;
00437         }
00438
00439         test = (byte)(data & 0x10);
00440         if (!(test == 0x10))
00441         {
00442             throw new ArgumentException("External clock rate not available on the WolfNet 65C02
WBC!");
00443         }
00444
00445         test = (byte)(data & 0x0F);
00446         if (test == 0x00)
00447         {
00448             Object.BaudRate = 115200;
00449         }
00450         else if (test == 0x01)
00451         {
00452             Object.BaudRate = 50;
00453         }
00454         else if (test == 0x02)
00455         {
00456             Object.BaudRate = 75;
00457         }
00458         else if (test == 0x03)
00459         {
00460             Object.BaudRate = 110;
00461         }
00462         else if (test == 0x04)
00463         {
00464             Object.BaudRate = 135;
00465         }
00466         else if (test == 0x05)
00467         {
00468             Object.BaudRate = 150;
00469         }
00470         else if (test == 0x06)
00471         {
00472             Object.BaudRate = 300;
00473         }
00474         else if (test == 0x07)
00475         {
00476             Object.BaudRate = 600;
00477         }
00478         else if (test == 0x08)
00479         {
00480             Object.BaudRate = 1200;
00481         }
00482         else if (test == 0x09)
00483         {

```

```
00484         Object.BaudRate = 1800;
00485     }
00486     else if (test == 0x0A)
00487     {
00488         Object.BaudRate = 2400;
00489     }
00490     else if (test == 0x0B)
00491     {
00492         Object.BaudRate = 3600;
00493     }
00494     else if (test == 0x0C)
00495     {
00496         Object.BaudRate = 4800;
00497     }
00498     else if (test == 0x0D)
00499     {
00500         Object.BaudRate = 7200;
00501     }
00502     else if (test == 0x0E)
00503     {
00504         Object.BaudRate = 9600;
00505     }
00506     else
00507     {
00508         Object.BaudRate = 19200;
00509     }
00510 }
00511
00512 private void ControlRegisterUpdate()
00513 {
00514     byte controlRegister = Memory[Offset + 3];
00515
00516     if (Object.StopBits == StopBits.Two)
00517     {
00518         controlRegister |= 0x80;
00519     }
00520     else if ((Object.StopBits == StopBits.OnePointFive) && (Object.DataBits == 5) ||
00521 (Object.StopBits == StopBits.One))
00522     {
00523         controlRegister &= 0x7F;
00524     }
00525     else
00526     {
00527         throw new ArgumentOutOfRangeException("StopBits or combination of StopBits and
DataBits is invalid!");
00528     }
00529
00530     if (Object.DataBits == 8)
00531     {
00532         controlRegister &= 0x9F;
00533     }
00534     else if (Object.DataBits == 7)
00535     {
00536         controlRegister |= 0x20;
00537     }
00538     else if (Object.DataBits == 6)
00539     {
00540         controlRegister |= 0x40;
00541     }
00542     else if (Object.DataBits == 5)
00543     {
00544         controlRegister |= 0x60;
00545     }
00546     else
00547     {
00548         throw new ArgumentOutOfRangeException("DataBits is out of range!");
00549     }
00550
00551     if (Object.BaudRate == 115200)
00552     {
00553         controlRegister &= 0xF0;
00554     }
00555     else if (Object.BaudRate == 50)
00556     {
00557         controlRegister |= 0x01;
00558     }
00559     else if (Object.BaudRate == 75)
00560     {
00561         controlRegister |= 0x02;
00562     }
00563     else if (Object.BaudRate == 110)
00564     {
00565         controlRegister |= 0x03;
00566     }
00567     else if (Object.BaudRate == 135)
00568     {
00569         controlRegister |= 0x04;
```

```

00569         }
00570         else if (Object.BaudRate == 150)
00571         {
00572             controlRegister |= 0x05;
00573         }
00574         else if (Object.BaudRate == 300)
00575         {
00576             controlRegister |= 0x06;
00577         }
00578         else if (Object.BaudRate == 600)
00579         {
00580             controlRegister |= 0x07;
00581         }
00582         else if (Object.BaudRate == 1200)
00583         {
00584             controlRegister |= 0x08;
00585         }
00586         else if (Object.BaudRate == 1800)
00587         {
00588             controlRegister |= 0x09;
00589         }
00590         else if (Object.BaudRate == 2400)
00591         {
00592             controlRegister |= 0x0A;
00593         }
00594         else if (Object.BaudRate == 3600)
00595         {
00596             controlRegister |= 0x0B;
00597         }
00598         else if (Object.BaudRate == 4800)
00599         {
00600             controlRegister |= 0x0C;
00601         }
00602         else if (Object.BaudRate == 7200)
00603         {
00604             controlRegister |= 0x0D;
00605         }
00606         else if (Object.BaudRate == 9600)
00607         {
00608             controlRegister |= 0x0E;
00609         }
00610         else if (Object.BaudRate == 19200)
00611         {
00612             controlRegister |= 0x0F;
00613         }
00614         else
00615         {
00616             throw new ArgumentOutOfRangeException("BaudRate is outside the range of Baud Rates
supported by the W65C51!");
00617         }
00618
00619         Memory[Offset + 3] = controlRegister;
00620     }
00621
00622     private void StatusRegisterUpdate()
00623     {
00624         byte statusRegister = Memory[Offset + 1];
00625
00626         if (Interrupted)
00627         {
00628             statusRegister |= 0x80;
00629         }
00630         else
00631         {
00632             statusRegister &= 0x7F;
00633         }
00634
00635         if (Object.DsrHolding == false)
00636         {
00637             statusRegister |= 0x40;
00638         }
00639         else
00640         {
00641             statusRegister &= 0xBF;
00642         }
00643
00644         if (Object.CDHolding)
00645         {
00646             statusRegister |= 0x20;
00647         }
00648         else
00649         {
00650             statusRegister &= 0xDF;
00651         }
00652
00653         statusRegister |= 0x10;
00654

```

```
00655         if (ReceiverFull)
00656         {
00657             statusRegister |= 0x08;
00658         }
00659         else
00660         {
00661             statusRegister &= 0xF7;
00662         }
00663
00664         if (Overflow)
00665         {
00666             statusRegister |= 0x04;
00667         }
00668         else
00669         {
00670             statusRegister &= 0xFB;
00671         }
00672
00673         statusRegister &= 0xFC;
00674
00675         Memory[Offset + 1] = statusRegister;
00676     }
00677
00678 private void BackgroundWorkerDoWork(object sender, DoWorkEventArgs e)
00679 {
00680     var worker = sender as BackgroundWorker;
00681
00682     while (true)
00683     {
00684         if (worker != null && worker.CancellationPending)
00685         {
00686             e.Cancel = true;
00687             return;
00688         }
00689
00690         if (Processor.IsRunning)
00691         {
00692             if (ReceiverFull || Overflow)
00693             {
00694                 Memory[Offset + 1] = (byte)(Memory[Offset + 1] | 0x80);
00695                 Interrupted = true;
00696                 Processor.InterruptRequest();
00697             }
00698
00699             if (DataRead)
00700             {
00701                 ReceiverFull = false;
00702                 Interrupted = false;
00703                 Overflow = false;
00704                 DataRead = false;
00705             }
00706         }
00707     }
00708 }
00709 #endregion
00710 }
00711 }
```


Index

- [_Length](#)
 - [Hardware.MemoryMap.BankedRam, 18](#)
 - [Hardware.MemoryMap.BankedRom, 20](#)
 - [Hardware.MemoryMap.DeviceArea, 23](#)
 - [Hardware.MemoryMap.SharedRom, 100](#)
 - [_Offset](#)
 - [Hardware.MemoryMap.BankedRam, 18](#)
 - [Hardware.MemoryMap.BankedRom, 20](#)
 - [Hardware.MemoryMap.DeviceArea, 23](#)
 - [Hardware.MemoryMap.SharedRom, 100](#)
 - [_PortList](#)
 - [Emulator.ViewModel.SettingsViewModel, 98](#)
 - [_backgroundWorker](#)
 - [Emulator.ViewModel.MainViewModel, 51](#)
 - [Hardware.W65C51, 174](#)
 - [_breakpointTriggered](#)
 - [Emulator.ViewModel.MainViewModel, 51](#)
 - [_contentLoaded](#)
 - [Emulator.MainWindow, 64](#)
 - [Emulator.SaveFile, 85](#)
 - [Emulator.Settings, 93](#)
 - [XamlGeneratedNamespace.GeneratedApplication, 30](#)
 - [_cycleCount](#)
 - [Hardware.W65C02, 147](#)
 - [_interrupt](#)
 - [Hardware.W65C02, 147](#)
 - [_logger](#)
 - [Hardware.W65C02, 148](#)
 - [_memoryPageOffset](#)
 - [Emulator.ViewModel.MainViewModel, 51](#)
 - [_previousInterrupt](#)
 - [Hardware.W65C02, 148](#)
 - [_programCounter](#)
 - [Hardware.W65C02, 148](#)
 - [_stackPointer](#)
 - [Hardware.W65C02, 148](#)
 - [_stateFileModel](#)
 - [Emulator.ViewModel.SaveFileViewModel, 88](#)
- [About](#)
 - [Emulator.ViewModel.MainViewModel, 43](#)
- [AboutCommand](#)
 - [Emulator.ViewModel.MainViewModel, 51](#)
- [Accumulator](#)
 - [Emulator.Model.OutputLog, 77](#)
 - [Hardware.W65C02, 148](#)
- [ACIA](#)
 - [Hardware.MemoryMap, 68](#)
- [ACR](#)
 - [Hardware.W65C22, 157](#)
- [ACR_T1TC](#)
 - [Hardware.W65C22, 157](#)
- [ACR_T2TC](#)
 - [Hardware.W65C22, 157](#)
- [AddBreakPoint](#)
 - [Emulator.ViewModel.MainViewModel, 43](#)
- [AddBreakPointCommand](#)
 - [Emulator.ViewModel.MainViewModel, 51](#)
- [AddEventHandler](#)
 - [XamlGeneratedNamespace.GeneratedInternalTypeHelper, 31](#)
- [AddressingMode](#)
 - [Hardware, 9](#)
- [AddWithCarryOperation](#)
 - [Hardware.W65C02, 113](#)
- [AllTypes](#)
 - [Emulator.Model.Breakpoint, 21](#)
 - [Emulator.Model.BreakpointType, 22](#)
- [AndOperation](#)
 - [Hardware.W65C02, 114](#)
- [Apply](#)
 - [Emulator.ViewModel.SettingsViewModel, 98](#)
- [ApplyCommand](#)
 - [Emulator.ViewModel.SettingsViewModel, 98](#)
- [ApplyEnabled](#)
 - [Emulator.ViewModel.SettingsViewModel, 99](#)
- [AslOperation](#)
 - [Hardware.W65C02, 114](#)
- [AT28C010](#)
 - [Emulator.Model.StateFileModel, 101](#)
 - [Emulator.ViewModel.MainViewModel, 51](#)
- [AT28C64](#)
 - [Emulator.Model.StateFileModel, 102](#)
 - [Emulator.ViewModel.MainViewModel, 52](#)
- [AT28CXX](#)
 - [Hardware.AT28CXX, 12](#)
- [BackgroundWorkerDoWork](#)
 - [Emulator.ViewModel.MainViewModel, 43](#)
 - [Hardware.W65C51, 163](#)
- [BankedRAM](#)
 - [Hardware.MemoryMap, 68](#)
- [BankedROM](#)
 - [Hardware.MemoryMap, 68](#)
- [Banks](#)
 - [Hardware.AT28CXX, 16](#)
 - [Hardware.HM62256, 37](#)
- [BankSize](#)
 - [Hardware.MemoryMap.BankedRam, 18](#)
- [BinaryLoadedNotification](#)
 - [Emulator.ViewModel.MainViewModel, 44](#)
- [BIOS_LOADPROGRAM_ERROR](#)
 - [Emulator.ExitCodes, 26](#)
- [BitOperation](#)
 - [Hardware.W65C02, 115](#)
- [BranchOperation](#)
 - [Hardware.W65C02, 116](#)
- [BreakOperation](#)
 - [Hardware.W65C02, 116](#)

- Breakpoints
 - Emulator.ViewModel.MainViewModel, [52](#)
- Build
 - Emulator.Versioning.Product, [79](#)
 - Emulator.Versioning.SettingsFile, [94](#)
- byteIn
 - Hardware.W65C51, [173](#)
- CarryFlag
 - Hardware.W65C02, [148](#)
- ChangeMemoryByOne
 - Hardware.W65C02, [116](#)
- ChangeRegisterByOne
 - Hardware.W65C02, [117](#)
- Cleanup
 - Emulator.ViewModel.ViewModelLocator, [109](#)
- Clear
 - Hardware.AT28CXX, [12](#)
 - Hardware.HM62256, [36](#)
- Close
 - Emulator.IClosable, [39](#)
 - Emulator.ViewModel.MainViewModel, [44](#)
 - Emulator.ViewModel.SaveFileViewModel, [87](#)
 - Emulator.ViewModel.SettingsViewModel, [98](#)
- CloseCommand
 - Emulator.ViewModel.MainViewModel, [52](#)
 - Emulator.ViewModel.SaveFileViewModel, [88](#)
 - Emulator.ViewModel.SettingsViewModel, [99](#)
- CloseFile
 - Emulator.MainWindow, [58](#)
- CollectionChanged
 - Emulator.MultiThreadedObservableCollection< T>, [75](#)
- ComFini
 - Hardware.W65C51, [163](#)
- ComInit
 - Hardware.W65C51, [164](#)
- CommandRegister
 - Hardware.W65C51, [165](#)
- CommandRegisterUpdate
 - Hardware.W65C51, [165](#)
- Company
 - Emulator.Versioning.Product, [79](#)
 - Hardware.Versioning.Product, [80](#)
- CompareOperation
 - Hardware.W65C02, [117](#)
- ComPortName
 - Emulator.Model.SettingsModel, [95](#)
- ComPortSelection
 - Emulator.ViewModel.SettingsViewModel, [99](#)
- Connect
 - Emulator.MainWindow, [58](#), [60](#)
 - Emulator.SaveFile, [84](#)
 - Emulator.Settings, [90](#), [91](#)
- ControlRegister
 - Hardware.W65C51, [166](#)
- ControlRegisterUpdate
 - Hardware.W65C51, [167](#)
- ConvertFlagsToByte
 - Hardware.W65C02, [119](#)
- ConvertOpCodeIntoString
 - Hardware.Utility, [103](#)
- Copyright
 - Emulator.Versioning.Product, [79](#)
 - Hardware.Versioning.Product, [80](#)
- CpuSpeed
 - Emulator.ViewModel.MainViewModel, [52](#)
- CreateDelegate
 - XamlGeneratedNamespace.GeneratedInternalTypeHelper, [31](#), [32](#)
- CreateInstance
 - XamlGeneratedNamespace.GeneratedInternalTypeHelper, [32](#)
- CreateNew
 - Emulator.SettingsFile, [93](#)
- CurrentBank
 - Hardware.AT28CXX, [16](#)
 - Hardware.HM62256, [37](#)
- CurrentDisassembly
 - Emulator.ViewModel.MainViewModel, [52](#)
 - Hardware.W65C02, [149](#)
- CurrentOpCode
 - Emulator.Model.OutputLog, [77](#)
 - Hardware.W65C02, [149](#)
- CurrentSerialPort
 - Emulator.ViewModel.MainViewModel, [53](#)
- CycleCountIncrementedAction
 - Hardware.W65C02, [149](#)
- DataRead
 - Hardware.W65C51, [174](#)
- DecimalFlag
 - Hardware.W65C02, [149](#)
- defaultBaudRate
 - Hardware.W65C51, [173](#)
- Description
 - Emulator.Versioning.Product, [79](#)
 - Hardware.Versioning.Product, [81](#)
- DisableInterruptFlag
 - Hardware.W65C02, [149](#)
- DisassemblyOutput
 - Hardware.Disassembly, [25](#)
- DumpMemory
 - Hardware.AT28CXX, [13](#)
 - Hardware.HM62256, [36](#)
- EchoMode
 - Hardware.W65C51, [174](#)
- Emulator, [7](#)
- Emulator.App, [11](#)
- Emulator.ExitCodes, [26](#)
 - BIOS_LOADPROGRAM_ERROR, [26](#)
 - LOAD_BIOS_FILE_ERROR, [26](#)
 - LOAD_ROM_FILE_ERROR, [27](#)
 - LOAD_STATE_ERROR, [27](#)
 - NO_BIOS, [27](#)
 - NO_ERROR, [27](#)
 - ROM_LOADPROGRAM_ERROR, [27](#)

- USER_ERROR, 27
- Emulator.IClosable, 38
 - Close, 39
- Emulator.MainWindow, 57
 - _contentLoaded, 64
 - CloseFile, 58
 - Connect, 58, 60
 - InitializeComponent, 62
 - LoadFile, 62
 - MainWindow, 58
 - NotificationMessageReceived, 63
 - SaveFile, 63
 - ToClose, 63
- Emulator.Model, 8
- Emulator.Model.Breakpoint, 20
 - AllTypes, 21
 - IsEnabled, 21
 - Type, 21
 - Value, 21
- Emulator.Model.BreakpointType, 22
 - AllTypes, 22
 - NumberOfCycleType, 22
 - ProgramCounterType, 23
- Emulator.Model.MemoryRowModel, 69
 - Location00, 70
 - Location01, 70
 - Location02, 70
 - Location03, 70
 - Location04, 70
 - Location05, 70
 - Location06, 71
 - Location07, 71
 - Location08, 71
 - Location09, 71
 - Location0A, 71
 - Location0B, 71
 - Location0C, 72
 - Location0D, 72
 - Location0E, 72
 - Location0F, 72
 - Offset, 72
- Emulator.Model.OutputLog, 76
 - Accumulator, 77
 - CurrentOpCode, 77
 - NumberOfCycles, 77
 - OutputLog, 77
 - ProgramCounter, 77
 - StackPointer, 77
 - XRegister, 78
 - YRegister, 78
- Emulator.Model.RomFileModel, 81
 - Rom, 82
 - RomBanks, 82
 - RomBankSize, 82
 - RomFileName, 82
 - RomFilePath, 82
- Emulator.Model.SettingsModel, 94
 - ComPortName, 95
 - SettingsVersionBuild, 95
 - SettingsVersionMajor, 95
 - SettingsVersionMinor, 95
 - SettingsVersionRevision, 96
- Emulator.Model.StateFileModel, 101
 - AT28C010, 101
 - AT28C64, 102
 - MM65SIB, 102
 - NumberOfCycles, 102
 - OutputLog, 102
 - W65C02, 102
 - W65C22, 102
 - W65C51, 103
- Emulator.MultiThreadedObservableCollection< T >, 73
 - CollectionChanged, 75
 - MultiThreadedObservableCollection, 74, 75
 - OnCollectionChanged, 75
- Emulator.SaveFile, 83
 - _contentLoaded, 85
 - Connect, 84
 - InitializeComponent, 84, 85
 - NotificationMessageReceived, 85
 - SaveFile, 83
- Emulator.Settings, 89
 - _contentLoaded, 93
 - Connect, 90, 91
 - InitializeComponent, 91, 92
 - NotificationMessageReceived, 92
 - PortSelectionDropDownClosed, 92
 - Settings, 90
- Emulator.SettingsFile, 93
 - CreateNew, 93
- Emulator.Versioning, 108
- Emulator.Versioning.Product, 78
 - Build, 79
 - Company, 79
 - Copyright, 79
 - Description, 79
 - Major, 79
 - Minor, 79
 - Name, 79
 - Revision, 79
 - Title, 80
 - VersionString, 80
- Emulator.Versioning.SettingsFile, 94
 - Build, 94
 - Major, 94
 - Minor, 94
 - Revision, 94
- Emulator.ViewModel, 8
- Emulator.ViewModel.MainViewModel, 39
 - _backgroundWorker, 51
 - _breakpointTriggered, 51
 - _memoryPageOffset, 51
 - About, 43
 - AboutCommand, 51
 - AddBreakPoint, 43
 - AddBreakPointCommand, 51

- AT28C010, 51
- AT28C64, 52
- BackgroundWorkerDoWork, 43
- BinaryLoadedNotification, 44
- Breakpoints, 52
- Close, 44
- CloseCommand, 52
- CpuSpeed, 52
- CurrentDisassembly, 52
- CurrentSerialPort, 53
- GenericNotification, 44
- GetLogModValue, 45
- GetOutputLog, 46
- GetSleepValue, 46
- HM62256, 53
- IsBreakPointTriggered, 46
- IsRomLoaded, 53
- IsRunning, 53
- MainViewModel, 41
- MemoryPage, 53
- MemoryPageOffset, 54
- MM65SIB, 54
- NumberOfCycles, 54
- OnClose, 47
- OnLoad, 47
- OutputLog, 54
- RemoveBreakPoint, 48
- RemoveBreakPointCommand, 54
- Reset, 48
- ResetCommand, 55
- RomFile, 55
- RunPause, 48
- RunPauseCommand, 55
- SelectedBreakpoint, 55
- Settings, 49
- SettingsAppliedNotification, 49
- SettingsCommand, 55
- SettingsModel, 55
- StateLoadedNotification, 49
- Step, 50
- StepCommand, 56
- StepProcessor, 50
- UpdateMemoryMapCommand, 56
- UpdateMemoryPage, 50
- UpdateUi, 50
- W65C02, 56
- W65C22, 56
- W65C51, 56
- WindowTitle, 56
- Emulator.ViewModel.SaveFileViewModel, 86
 - _stateFileModel, 88
 - Close, 87
 - CloseCommand, 88
 - Filename, 88
 - Save, 87
 - SaveEnabled, 89
 - SaveFileCommand, 89
 - SaveFileViewModel, 87
 - Select, 88
 - SelectFileCommand, 89
- Emulator.ViewModel.SettingsViewModel, 96
 - _PortList, 98
 - Apply, 98
 - ApplyCommand, 98
 - ApplyEnabled, 99
 - Close, 98
 - CloseCommand, 99
 - ComPortSelection, 99
 - PortList, 99
 - SettingsModel, 99
 - SettingsViewModel, 97
 - UpdatePortList, 98
- Emulator.ViewModel.ViewModelLocator, 108
 - Cleanup, 109
 - Main, 109
 - SaveFile, 109
 - Settings, 110
 - ViewModelLocator, 109
- Emulator/App.xaml.cs, 176
- Emulator/Classes/ExitCodes.cs, 177
- Emulator/Classes/FileLocations.cs, 177
- Emulator/Classes/SettingsFile.cs, 178
- Emulator/Classes/Versioning.cs, 179
- Emulator/Interfaces/IClosable.cs, 180
- Emulator/MainWindow.xaml.cs, 180, 181
- Emulator/Model/Breakpoint.cs, 182
- Emulator/Model/BreakpointType.cs, 182, 183
- Emulator/Model/MemoryRowModel.cs, 183
- Emulator/Model/OutputLog.cs, 184, 185
- Emulator/Model/RomFileModel.cs, 185, 186
- Emulator/Model/SettingsModel.cs, 186
- Emulator/Model/StateFileModel.cs, 187
- Emulator/MultiThreadedCollection.cs, 188
- Emulator/obj/x86/Debug/.NETFramework,Version=v4.8.AssemblyAttributes.cs, 189
- Emulator/obj/x86/Debug/App.g.cs, 190
- Emulator/obj/x86/Debug/App.g.i.cs, 191
- Emulator/obj/x86/Debug/Emulator_Content.g.cs, 192
- Emulator/obj/x86/Debug/Emulator_Content.g.i.cs, 193
- Emulator/obj/x86/Debug/GeneratedInternalTypeHelper.g.cs, 193
- Emulator/obj/x86/Debug/GeneratedInternalTypeHelper.g.i.cs, 194
- Emulator/obj/x86/Debug/MainWindow.g.cs, 195
- Emulator/obj/x86/Debug/MainWindow.g.i.cs, 202
- Emulator/obj/x86/Debug/SaveFile.g.cs, 209
- Emulator/obj/x86/Debug/SaveFile.g.i.cs, 211
- Emulator/obj/x86/Debug/Settings.g.cs, 213
- Emulator/obj/x86/Debug/Settings.g.i.cs, 215
- Emulator/Properties/AssemblyInfo.cs, 217
- Emulator/SaveFile.xaml.cs, 218, 219
- Emulator/Settings.xaml.cs, 219
- Emulator/ViewModel/MainViewModel.cs, 220, 221
- Emulator/ViewModel/SaveFileViewModel.cs, 230, 231
- Emulator/ViewModel/SettingsViewModel.cs, 232
- Emulator/ViewModel/ViewModelLocator.cs, 233, 234

- End
 - Hardware.AT28CXX, 17
 - Hardware.HM62256, 38
 - Hardware.MemoryMap.DeviceArea, 24
 - Hardware.W65C22, 159
- EorOperation
 - Hardware.W65C02, 119
- ExecuteOpCode
 - Hardware.W65C02, 120
- Filename
 - Emulator.ViewModel.SaveFileViewModel, 88
- Fini
 - Hardware.W65C51, 169
- GenericNotification
 - Emulator.ViewModel.MainViewModel, 44
- GetAddressByAddressingMode
 - Hardware.W65C02, 132
- GetAddressingMode
 - Hardware.W65C02, 134
- GetCycleCount
 - Hardware.W65C02, 137
- GetLogModValue
 - Emulator.ViewModel.MainViewModel, 45
- GetOutputLog
 - Emulator.ViewModel.MainViewModel, 46
- GetPropertyValue
 - XamlGeneratedNamespace.GeneratedInternalTypeHelper, 33
- GetSleepValue
 - Emulator.ViewModel.MainViewModel, 46
- GPIO
 - Hardware.MemoryMap, 68
- Hardware, 8
 - AddressingMode, 9
- Hardware.AT28CXX, 11
 - AT28CXX, 12
 - Banks, 16
 - Clear, 12
 - CurrentBank, 16
 - DumpMemory, 13
 - End, 17
 - Length, 17
 - Load, 13, 15
 - Memory, 17
 - Offset, 17
 - Processor, 17
 - Read, 15
 - ReadFile, 15
 - Write, 16
- Hardware.Disassembly, 25
 - DisassemblyOutput, 25
 - HighAddress, 25
 - LowAddress, 25
 - OpCodeString, 26
- Hardware.HM62256, 35
 - Banks, 37
 - Clear, 36
 - CurrentBank, 37
 - DumpMemory, 36
 - End, 38
 - HM62256, 35
 - Length, 38
 - Memory, 38
 - Offset, 38
 - Read, 36
 - Reset, 37
 - Write, 37
- Hardware.MemoryMap, 64
 - ACIA, 68
 - BankedRAM, 68
 - BankedROM, 68
 - GPIO, 68
 - Init, 65
 - Length, 67
 - MM65SIB, 68
 - Processor, 68
 - Read, 65
 - ReadWithoutCycle, 66
 - SharedROM, 68
 - Write, 66
 - WriteWithoutCycle, 67
- Hardware.MemoryMap.BankedRam, 18
 - _Length, 18
 - _Offset, 18
 - BankSize, 18
 - Length, 19
 - Offset, 19
 - TotalBanks, 18
 - TotalLength, 19
- Hardware.MemoryMap.BankedRom, 19
 - _Length, 20
 - _Offset, 20
 - Length, 20
 - Offset, 20
 - TotalBanks, 20
- Hardware.MemoryMap.DeviceArea, 23
 - _Length, 23
 - _Offset, 23
 - End, 24
 - Length, 24
 - Offset, 24
- Hardware.MemoryMap.Devices, 24
- Hardware.MemoryMap.Devices.ACIA, 10
 - Length, 10
 - Offset, 10
- Hardware.MemoryMap.Devices.GPIO, 34
 - Length, 34
 - Offset, 34
- Hardware.MemoryMap.Devices.MM65SIB, 73
 - Length, 73
 - Offset, 73
- Hardware.MemoryMap.SharedRom, 100
 - _Length, 100
 - _Offset, 100

- Length, [100](#)
- Offset, [101](#)
- TotalBanks, [100](#)
- Hardware.Utility, [103](#)
 - ConvertOpCodeIntoString, [103](#)
- Hardware.Versioning.Product, [80](#)
 - Company, [80](#)
 - Copyright, [80](#)
 - Description, [81](#)
 - Name, [81](#)
 - Title, [81](#)
 - Version, [81](#)
- Hardware.W65C02, [110](#)
 - _cycleCount, [147](#)
 - _interrupt, [147](#)
 - _logger, [148](#)
 - _previousInterrupt, [148](#)
 - _programCounter, [148](#)
 - _stackPointer, [148](#)
 - Accumulator, [148](#)
 - AddWithCarryOperation, [113](#)
 - AndOperation, [114](#)
 - AslOperation, [114](#)
 - BitOperation, [115](#)
 - BranchOperation, [116](#)
 - BreakOperation, [116](#)
 - CarryFlag, [148](#)
 - ChangeMemoryByOne, [116](#)
 - ChangeRegisterByOne, [117](#)
 - CompareOperation, [117](#)
 - ConvertFlagsToByte, [119](#)
 - CurrentDisassembly, [149](#)
 - CurrentOpCode, [149](#)
 - CycleCountIncrementedAction, [149](#)
 - DecimalFlag, [149](#)
 - DisableInterruptFlag, [149](#)
 - EorOperation, [119](#)
 - ExecuteOpCode, [120](#)
 - GetAddressByAddressingMode, [132](#)
 - GetAddressingMode, [134](#)
 - GetCycleCount, [137](#)
 - IncrementCycleCount, [137](#)
 - InterruptRequest, [137](#)
 - isRunning, [148](#)
 - JumpToSubRoutineOperation, [137](#)
 - LsrOperation, [138](#)
 - MoveProgramCounterByRelativeValue, [138](#)
 - NegativeFlag, [149](#)
 - NextStep, [139](#)
 - OrOperation, [139](#)
 - OverflowFlag, [150](#)
 - PeekStack, [140](#)
 - PokeStack, [140](#)
 - ProcessIRQ, [140](#)
 - ProcessNMI, [140](#)
 - ProgramCounter, [150](#)
 - PullFlagsOperation, [141](#)
 - PushFlagsOperation, [141](#)
 - Reset, [141](#)
 - ResetCycleCount, [141](#)
 - ReturnFromInterruptOperation, [142](#)
 - ReturnFromSubRoutineOperation, [142](#)
 - RolOperation, [142](#)
 - RorOperation, [143](#)
 - SetDisassembly, [144](#)
 - SetNegativeFlag, [146](#)
 - SetZeroFlag, [146](#)
 - StackPointer, [150](#)
 - SubtractWithBorrowOperation, [147](#)
 - TriggerIRQ, [150](#)
 - TriggerNmi, [150](#)
 - W65C02, [113](#)
 - WrapProgramCounter, [147](#)
 - XRegister, [151](#)
 - YRegister, [151](#)
 - ZeroFlag, [151](#)
- Hardware.W65C22, [151](#)
 - ACR, [157](#)
 - ACR_T1TC, [157](#)
 - ACR_T2TC, [157](#)
 - End, [159](#)
 - IER, [157](#)
 - IER_EN, [157](#)
 - IER_T1, [157](#)
 - IER_T2, [157](#)
 - IFR, [158](#)
 - IFR_INT, [158](#)
 - IFR_T1, [158](#)
 - IFR_T2, [158](#)
 - Init, [153](#)
 - Length, [159](#)
 - Memory, [159](#)
 - Offset, [159](#)
 - OnT1Timeout, [154](#)
 - OnT2Timeout, [154](#)
 - Processor, [160](#)
 - Read, [155](#)
 - Reset, [155](#)
 - T1CH, [158](#)
 - T1CL, [158](#)
 - T1Init, [155](#)
 - T1Interval, [160](#)
 - T1IsEnabled, [160](#)
 - T1IsIRQ, [158](#)
 - T1Object, [160](#)
 - T1TimerControl, [160](#)
 - T2CH, [158](#)
 - T2CL, [159](#)
 - T2Init, [156](#)
 - T2Interval, [160](#)
 - T2IsEnabled, [161](#)
 - T2IsIRQ, [159](#)
 - T2Object, [161](#)
 - T2TimerControl, [161](#)
 - W65C22, [153](#)
 - Write, [156](#)

- Hardware.W65C51, 161
 - _backgroundWorker, 174
 - BackgroundWorkerDoWork, 163
 - byteIn, 173
 - ComFini, 163
 - ComInit, 164
 - CommandRegister, 165
 - CommandRegisterUpdate, 165
 - ControlRegister, 166
 - ControlRegisterUpdate, 167
 - DataRead, 174
 - defaultBaudRate, 173
 - EchoMode, 174
 - Fini, 169
 - HardwarePreRead, 169
 - HardwarePreWrite, 169
 - Init, 170
 - InterruptDisabled, 174
 - Interrupted, 174
 - IsEnabled, 174
 - Length, 174
 - Memory, 175
 - Object, 175
 - ObjectName, 175
 - Offset, 175
 - Overrun, 175
 - ParityEnabled, 175
 - Processor, 175
 - Read, 171
 - ReceiverFull, 176
 - Reset, 171
 - RtsControl, 176
 - SerialDataReceived, 171
 - StatusRegisterUpdate, 172
 - W65C51, 163
 - Write, 173
 - WriteCOM, 173
- Hardware/AT28CXX.cs, 235
- Hardware/Classes/AddressingMode.cs, 237
- Hardware/Classes/Disassembly.cs, 238, 239
- Hardware/Classes/FileLocations.cs, 178
- Hardware/Classes/MemoryMap.cs, 239
- Hardware/Classes/Utility.cs, 242
- Hardware/Classes/Versioning.cs, 179, 180
- Hardware/HM62256.cs, 246
- Hardware/obj/Debug/.NETFramework,Version=v4.8.AssemblyAttributes.cs, 189
- Hardware/Properties/AssemblyInfo.cs, 218
- Hardware/W65C02.cs, 248
- Hardware/W65C22.cs, 277
- Hardware/W65C51.cs, 281
- HardwarePreRead
 - Hardware.W65C51, 169
- HardwarePreWrite
 - Hardware.W65C51, 169
- HighAddress
 - Hardware.Disassembly, 25
- HM62256
 - Emulator.ViewModel.MainViewModel, 53
 - Hardware.HM62256, 35
- IER
 - Hardware.W65C22, 157
- IER_EN
 - Hardware.W65C22, 157
- IER_T1
 - Hardware.W65C22, 157
- IER_T2
 - Hardware.W65C22, 157
- IFR
 - Hardware.W65C22, 158
- IFR_INT
 - Hardware.W65C22, 158
- IFR_T1
 - Hardware.W65C22, 158
- IFR_T2
 - Hardware.W65C22, 158
- IncrementCycleCount
 - Hardware.W65C02, 137
- Init
 - Hardware.MemoryMap, 65
 - Hardware.W65C22, 153
 - Hardware.W65C51, 170
- InitializeComponent
 - Emulator.MainWindow, 62
 - Emulator.SaveFile, 84, 85
 - Emulator.Settings, 91, 92
 - XamlGeneratedNamespace.GeneratedApplication, 28, 29
- InterruptDisabled
 - Hardware.W65C51, 174
- Interrupted
 - Hardware.W65C51, 174
- InterruptRequest
 - Hardware.W65C02, 137
- IsBreakPointTriggered
 - Emulator.ViewModel.MainViewModel, 46
- IsEnabled
 - Emulator.Model.Breakpoint, 21
 - Hardware.W65C51, 174
- IsRomLoaded
 - Emulator.ViewModel.MainViewModel, 53
- IsRunning
 - Emulator.ViewModel.MainViewModel, 53
- isRunning
 - Hardware.W65C02, 148
- JumpToSubRoutineOperation
 - Hardware.W65C02, 137
- Length
 - Hardware.AT28CXX, 17
 - Hardware.HM62256, 38
 - Hardware.MemoryMap, 67
 - Hardware.MemoryMap.BankedRam, 19
 - Hardware.MemoryMap.BankedRom, 20
 - Hardware.MemoryMap.DeviceArea, 24

- Hardware.MemoryMap.Devices.ACIA, [10](#)
- Hardware.MemoryMap.Devices.GPIO, [34](#)
- Hardware.MemoryMap.Devices.MM65SIB, [73](#)
- Hardware.MemoryMap.SharedRom, [100](#)
- Hardware.W65C22, [159](#)
- Hardware.W65C51, [174](#)
- Load
 - Hardware.AT28CXX, [13](#), [15](#)
- LOAD_BIOS_FILE_ERROR
 - Emulator.ExitCodes, [26](#)
- LOAD_ROM_FILE_ERROR
 - Emulator.ExitCodes, [27](#)
- LOAD_STATE_ERROR
 - Emulator.ExitCodes, [27](#)
- LoadFile
 - Emulator.MainWindow, [62](#)
- Location00
 - Emulator.Model.MemoryRowModel, [70](#)
- Location01
 - Emulator.Model.MemoryRowModel, [70](#)
- Location02
 - Emulator.Model.MemoryRowModel, [70](#)
- Location03
 - Emulator.Model.MemoryRowModel, [70](#)
- Location04
 - Emulator.Model.MemoryRowModel, [70](#)
- Location05
 - Emulator.Model.MemoryRowModel, [70](#)
- Location06
 - Emulator.Model.MemoryRowModel, [71](#)
- Location07
 - Emulator.Model.MemoryRowModel, [71](#)
- Location08
 - Emulator.Model.MemoryRowModel, [71](#)
- Location09
 - Emulator.Model.MemoryRowModel, [71](#)
- Location0A
 - Emulator.Model.MemoryRowModel, [71](#)
- Location0B
 - Emulator.Model.MemoryRowModel, [71](#)
- Location0C
 - Emulator.Model.MemoryRowModel, [72](#)
- Location0D
 - Emulator.Model.MemoryRowModel, [72](#)
- Location0E
 - Emulator.Model.MemoryRowModel, [72](#)
- Location0F
 - Emulator.Model.MemoryRowModel, [72](#)
- LowAddress
 - Hardware.Disassembly, [25](#)
- LsrOperation
 - Hardware.W65C02, [138](#)
- Main
 - Emulator.ViewModel.ViewModelLocator, [109](#)
 - XamlGeneratedNamespace.GeneratedApplication, [29](#)
- MainViewModel
 - Emulator.ViewModel.MainViewModel, [41](#)
- MainViewModel.cs
 - W65C02, [220](#)
 - W65C22, [220](#)
 - W65C51, [221](#)
- MainWindow
 - Emulator.MainWindow, [58](#)
- Major
 - Emulator.Versioning.Product, [79](#)
 - Emulator.Versioning.SettingsFile, [94](#)
- Memory
 - Hardware.AT28CXX, [17](#)
 - Hardware.HM62256, [38](#)
 - Hardware.W65C22, [159](#)
 - Hardware.W65C51, [175](#)
- MemoryPage
 - Emulator.ViewModel.MainViewModel, [53](#)
- MemoryPageOffset
 - Emulator.ViewModel.MainViewModel, [54](#)
- Minor
 - Emulator.Versioning.Product, [79](#)
 - Emulator.Versioning.SettingsFile, [94](#)
- MM65SIB
 - Emulator.Model.StateFileModel, [102](#)
 - Emulator.ViewModel.MainViewModel, [54](#)
 - Hardware.MemoryMap, [68](#)
- MoveProgramCounterByRelativeValue
 - Hardware.W65C02, [138](#)
- MultiThreadedObservableCollection
 - Emulator.MultiThreadedObservableCollection< T >, [74](#), [75](#)
- Name
 - Emulator.Versioning.Product, [79](#)
 - Hardware.Versioning.Product, [81](#)
- NegativeFlag
 - Hardware.W65C02, [149](#)
- NextStep
 - Hardware.W65C02, [139](#)
- NO_BIOS
 - Emulator.ExitCodes, [27](#)
- NO_ERROR
 - Emulator.ExitCodes, [27](#)
- NotificationMessageReceived
 - Emulator.MainWindow, [63](#)
 - Emulator.SaveFile, [85](#)
 - Emulator.Settings, [92](#)
- NumberOfCycles
 - Emulator.Model.OutputLog, [77](#)
 - Emulator.Model.StateFileModel, [102](#)
 - Emulator.ViewModel.MainViewModel, [54](#)
- NumberOfCycleType
 - Emulator.Model.BreakpointType, [22](#)
- Object
 - Hardware.W65C51, [175](#)
- ObjectName
 - Hardware.W65C51, [175](#)
- Offset
 - Emulator.Model.MemoryRowModel, [72](#)

- Hardware.AT28CXX, [17](#)
- Hardware.HM62256, [38](#)
- Hardware.MemoryMap.BankedRam, [19](#)
- Hardware.MemoryMap.BankedRom, [20](#)
- Hardware.MemoryMap.DeviceArea, [24](#)
- Hardware.MemoryMap.Devices.ACIA, [10](#)
- Hardware.MemoryMap.Devices.GPIO, [34](#)
- Hardware.MemoryMap.Devices.MM65SIB, [73](#)
- Hardware.MemoryMap.SharedRom, [101](#)
- Hardware.W65C22, [159](#)
- Hardware.W65C51, [175](#)
- OnClose
 - Emulator.ViewModel.MainViewModel, [47](#)
- OnCollectionChanged
 - Emulator.MultiThreadedObservableCollection< T>, [75](#)
- OnLoad
 - Emulator.ViewModel.MainViewModel, [47](#)
- OnT1Timeout
 - Hardware.W65C22, [154](#)
- OnT2Timeout
 - Hardware.W65C22, [154](#)
- OpCodeString
 - Hardware.Disassembly, [26](#)
- OrOperation
 - Hardware.W65C02, [139](#)
- OutputLog
 - Emulator.Model.OutputLog, [77](#)
 - Emulator.Model.StateFileModel, [102](#)
 - Emulator.ViewModel.MainViewModel, [54](#)
- OverflowFlag
 - Hardware.W65C02, [150](#)
- Overrun
 - Hardware.W65C51, [175](#)
- ParityEnabled
 - Hardware.W65C51, [175](#)
- PeekStack
 - Hardware.W65C02, [140](#)
- PokeStack
 - Hardware.W65C02, [140](#)
- PortList
 - Emulator.ViewModel.SettingsViewModel, [99](#)
- PortSelectionDropDownClosed
 - Emulator.Settings, [92](#)
- ProcessIRQ
 - Hardware.W65C02, [140](#)
- ProcessNMI
 - Hardware.W65C02, [140](#)
- Processor
 - Hardware.AT28CXX, [17](#)
 - Hardware.MemoryMap, [68](#)
 - Hardware.W65C22, [160](#)
 - Hardware.W65C51, [175](#)
- ProgramCounter
 - Emulator.Model.OutputLog, [77](#)
 - Hardware.W65C02, [150](#)
- ProgramCounterType
 - Emulator.Model.BreakpointType, [23](#)
- PullFlagsOperation
 - Hardware.W65C02, [141](#)
- PushFlagsOperation
 - Hardware.W65C02, [141](#)
- Read
 - Hardware.AT28CXX, [15](#)
 - Hardware.HM62256, [36](#)
 - Hardware.MemoryMap, [65](#)
 - Hardware.W65C22, [155](#)
 - Hardware.W65C51, [171](#)
- ReadFile
 - Hardware.AT28CXX, [15](#)
- ReadWithoutCycle
 - Hardware.MemoryMap, [66](#)
- ReceiverFull
 - Hardware.W65C51, [176](#)
- RemoveBreakPoint
 - Emulator.ViewModel.MainViewModel, [48](#)
- RemoveBreakPointCommand
 - Emulator.ViewModel.MainViewModel, [54](#)
- Reset
 - Emulator.ViewModel.MainViewModel, [48](#)
 - Hardware.HM62256, [37](#)
 - Hardware.W65C02, [141](#)
 - Hardware.W65C22, [155](#)
 - Hardware.W65C51, [171](#)
- ResetCommand
 - Emulator.ViewModel.MainViewModel, [55](#)
- ResetCycleCount
 - Hardware.W65C02, [141](#)
- ReturnFromInterruptOperation
 - Hardware.W65C02, [142](#)
- ReturnFromSubRoutineOperation
 - Hardware.W65C02, [142](#)
- Revision
 - Emulator.Versioning.Product, [79](#)
 - Emulator.Versioning.SettingsFile, [94](#)
- RolOperation
 - Hardware.W65C02, [142](#)
- Rom
 - Emulator.Model.RomFileModel, [82](#)
- ROM_LOADPROGRAM_ERROR
 - Emulator.ExitCodes, [27](#)
- RomBanks
 - Emulator.Model.RomFileModel, [82](#)
- RomBankSize
 - Emulator.Model.RomFileModel, [82](#)
- RomFile
 - Emulator.ViewModel.MainViewModel, [55](#)
- RomFileName
 - Emulator.Model.RomFileModel, [82](#)
- RomFilePath
 - Emulator.Model.RomFileModel, [82](#)
- RorOperation
 - Hardware.W65C02, [143](#)
- RtsControl
 - Hardware.W65C51, [176](#)
- RunPause

- Emulator.ViewModel.MainViewModel, [48](#)
- RunPauseCommand
 - Emulator.ViewModel.MainViewModel, [55](#)
- Save
 - Emulator.ViewModel.SaveFileViewModel, [87](#)
- SaveEnabled
 - Emulator.ViewModel.SaveFileViewModel, [89](#)
- SaveFile
 - Emulator.MainWindow, [63](#)
 - Emulator.SaveFile, [83](#)
 - Emulator.ViewModel.ViewModelLocator, [109](#)
- SaveFileCommand
 - Emulator.ViewModel.SaveFileViewModel, [89](#)
- SaveFileViewModel
 - Emulator.ViewModel.SaveFileViewModel, [87](#)
- Select
 - Emulator.ViewModel.SaveFileViewModel, [88](#)
- SelectedBreakpoint
 - Emulator.ViewModel.MainViewModel, [55](#)
- SelectFileCommand
 - Emulator.ViewModel.SaveFileViewModel, [89](#)
- SerialDataReceived
 - Hardware.W65C51, [171](#)
- SetDisassembly
 - Hardware.W65C02, [144](#)
- SetNegativeFlag
 - Hardware.W65C02, [146](#)
- SetPropertyValue
 - XamlGeneratedNamespace.GeneratedInternalTypeHelper, [33](#)
- Settings
 - Emulator.Settings, [90](#)
 - Emulator.ViewModel.MainViewModel, [49](#)
 - Emulator.ViewModel.ViewModelLocator, [110](#)
- SettingsAppliedNotification
 - Emulator.ViewModel.MainViewModel, [49](#)
- SettingsCommand
 - Emulator.ViewModel.MainViewModel, [55](#)
- SettingsModel
 - Emulator.ViewModel.MainViewModel, [55](#)
 - Emulator.ViewModel.SettingsViewModel, [99](#)
- SettingsVersionBuild
 - Emulator.Model.SettingsModel, [95](#)
- SettingsVersionMajor
 - Emulator.Model.SettingsModel, [95](#)
- SettingsVersionMinor
 - Emulator.Model.SettingsModel, [95](#)
- SettingsVersionRevision
 - Emulator.Model.SettingsModel, [96](#)
- SettingsViewModel
 - Emulator.ViewModel.SettingsViewModel, [97](#)
- SetZeroFlag
 - Hardware.W65C02, [146](#)
- SharedROM
 - Hardware.MemoryMap, [68](#)
- StackPointer
 - Emulator.Model.OutputLog, [77](#)
 - Hardware.W65C02, [150](#)
- StateLoadedNotification
 - Emulator.ViewModel.MainViewModel, [49](#)
- StatusRegisterUpdate
 - Hardware.W65C51, [172](#)
- Step
 - Emulator.ViewModel.MainViewModel, [50](#)
- StepCommand
 - Emulator.ViewModel.MainViewModel, [56](#)
- StepProcessor
 - Emulator.ViewModel.MainViewModel, [50](#)
- SubtractWithBorrowOperation
 - Hardware.W65C02, [147](#)
- T1CH
 - Hardware.W65C22, [158](#)
- T1CL
 - Hardware.W65C22, [158](#)
- T1Init
 - Hardware.W65C22, [155](#)
- T1Interval
 - Hardware.W65C22, [160](#)
- T1IsEnabled
 - Hardware.W65C22, [160](#)
- T1IsIRQ
 - Hardware.W65C22, [158](#)
- T1Object
 - Hardware.W65C22, [160](#)
- T1TimerControl
 - Hardware.W65C22, [160](#)
- T2CH
 - Hardware.W65C22, [158](#)
- T2CL
 - Hardware.W65C22, [159](#)
- T2Init
 - Hardware.W65C22, [156](#)
- T2Interval
 - Hardware.W65C22, [160](#)
- T2IsEnabled
 - Hardware.W65C22, [161](#)
- T2IsIRQ
 - Hardware.W65C22, [159](#)
- T2Object
 - Hardware.W65C22, [161](#)
- T2TimerControl
 - Hardware.W65C22, [161](#)
- Title
 - Emulator.Versioning.Product, [80](#)
 - Hardware.Versioning.Product, [81](#)
- ToClose
 - Emulator.MainWindow, [63](#)
- TotalBanks
 - Hardware.MemoryMap.BankedRam, [18](#)
 - Hardware.MemoryMap.BankedRom, [20](#)
 - Hardware.MemoryMap.SharedRom, [100](#)
- TotalLength
 - Hardware.MemoryMap.BankedRam, [19](#)
- TriggerIRQ
 - Hardware.W65C02, [150](#)
- TriggerNmi

- Hardware.W65C02, [150](#)
- Type
 - Emulator.Model.Breakpoint, [21](#)
- UpdateMemoryMapCommand
 - Emulator.ViewModel.MainViewModel, [56](#)
- UpdateMemoryPage
 - Emulator.ViewModel.MainViewModel, [50](#)
- UpdatePortList
 - Emulator.ViewModel.SettingsViewModel, [98](#)
- UpdateUi
 - Emulator.ViewModel.MainViewModel, [50](#)
- USER_ERROR
 - Emulator.ExitCodes, [27](#)
- Value
 - Emulator.Model.Breakpoint, [21](#)
- Version
 - Hardware.Versioning.Product, [81](#)
- VersionString
 - Emulator.Versioning.Product, [80](#)
- ViewModelLocator
 - Emulator.ViewModel.ViewModelLocator, [109](#)
- W65C02
 - Emulator.Model.StateFileModel, [102](#)
 - Emulator.ViewModel.MainViewModel, [56](#)
 - Hardware.W65C02, [113](#)
 - MainViewModel.cs, [220](#)
- W65C22
 - Emulator.Model.StateFileModel, [102](#)
 - Emulator.ViewModel.MainViewModel, [56](#)
 - Hardware.W65C22, [153](#)
 - MainViewModel.cs, [220](#)
- W65C51
 - Emulator.Model.StateFileModel, [103](#)
 - Emulator.ViewModel.MainViewModel, [56](#)
 - Hardware.W65C51, [163](#)
 - MainViewModel.cs, [221](#)
- WindowTitle
 - Emulator.ViewModel.MainViewModel, [56](#)
- WrapProgramCounter
 - Hardware.W65C02, [147](#)
- Write
 - Hardware.AT28CXX, [16](#)
 - Hardware.HM62256, [37](#)
 - Hardware.MemoryMap, [66](#)
 - Hardware.W65C22, [156](#)
 - Hardware.W65C51, [173](#)
- WriteCOM
 - Hardware.W65C51, [173](#)
- WriteWithoutCycle
 - Hardware.MemoryMap, [67](#)
- XamlGeneratedNamespace, [10](#)
- XamlGeneratedNamespace.GeneratedApplication, [28](#)
 - _contentLoaded, [30](#)
 - InitializeComponent, [28](#), [29](#)
 - Main, [29](#)
- XamlGeneratedNamespace.GeneratedInternalType-Helper, [30](#)
 - AddEventHandler, [31](#)
 - CreateDelegate, [31](#), [32](#)
 - CreateInstance, [32](#)
 - GetPropertyValue, [33](#)
 - SetPropertyValue, [33](#)
- XRegister
 - Emulator.Model.OutputLog, [78](#)
 - Hardware.W65C02, [151](#)
- YRegister
 - Emulator.Model.OutputLog, [78](#)
 - Hardware.W65C02, [151](#)
- ZeroFlag
 - Hardware.W65C02, [151](#)