SPACEAGE 2 DEBUGGER

DOKUMENTATION

Verwendete Begriffe:

In der folgenden Tabelle werden alle verwendeten Begriffe und ihre Beschreibung aufgelistet.

Begriff	Beschreibung
MicroCode Mikrobefehl	Ein MicroCode implementiert einen Assemblerbefehl. Er kann bis zu 32 Zustände enthalten, die vom Steuerwerk zum Setzen der Steuerleitungen benötigt werden. Ohne MicroCodes kann der SpaceAge2 keine MIPS Befehle durchführen.
Default State	Dies ist der Standardzustand, den ein MicroCode annehmen kann, um gefahrlos vom SpaceAge2 ausgeführt zu werden. Wird als Initialwert beim Parsen der MicroCodes verwendet.
ClockStep	Veranlasst den SpaceAge2 genau einen Takt weiter zu gehen. Dies umfasst das Laden und Ausführen des nächsten MicroCodeStates.
CommandStep	Veranlasst den SpaceAge2 den aktuellen Befehl zu überspringen und zum nächsten Befehl überzugehen. Dadurch wird der ProgramCounter inkrementiert.
DebugSession	Eine Debugsession ist eine Sitzung mit dem Debugger. Diese endet mit Beenden des Debuggers. Dabei kann ein eventuell gesetzter Breakpoint im Breakpointregister bestehen bleiben.

1. Beschreibung der Software:

Der Debugger ist der Vermittler zwischen geschriebener Software und der Space Age 2 Hardware. Mit Hilfe des Debuggers können Programme reingeladen, angezeigt und auf den Space Age 2 übertragen werden. Im Grunde orientiert sich die Software an den Eigenschaften der gängigen Debugging-Tools. Mit Hilfe dieser Software hat man zusätzlich noch die Möglichkeit, die MicroCodes aus einer Excel Tabelle zu parsen und auf den Space Age 2 zu übertragen. Es ist zudem möglich, den RAM, sowie Register auszulesen und zu manipulieren. Dabei wird das Setzen von Breakpoints ebenso unterstützt wie das Ausführen der DebugSteps ClockStep und CommandStep.

Die wesentlichen Features dieser Software sind nachfolgend zusammengefasst.

Mikrobefehle

Da sämtliche vom Space Age 2 unterstützten Befehle nur in Form von MicroCodes realisiert werden können, wurde eine Excel Tabelle erstellt, mit Hilfe derer schnell und vor allem übersichtlich die Mikrobefehle realisiert werden konnten. Diese Tabelle musste byte-weise auf den Rechner übertragen werden, weshalb sich daraus die folgenden Anforderungen an die Debug-Software ergaben:

- ✓ Parsen einer Excel-Datei, die alle MicroCodes enthält
- ✓ Setzen des DefaultStates für alle nicht verwendeten Adressen
- ✓ Aufteilen der geparsten MicroCodes in 9 Byte-breite Blöcke, für jeden ROM-Baustein ein Block
- ✓ Beschreiben der ROMs
- ✓ Rücklesen der ROMs inklusive Vergleich mit den geschriebenen MicroCodes

Achtung: Der MicroCodeParser liest den DefaultState **einmal** aus und setzt diesen global für alle zu parsenden MicroCodes!

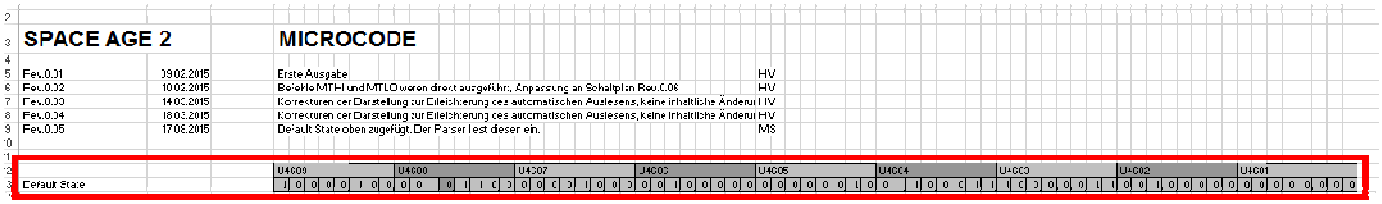


Abbildung 1: Header der MicroCode Exceltabelle. Rot vorgehoben: der DefaultState, der für alle nicht verwendeten MicroCodeAdressen gilt.

Programm

Das Programmieren des Space Age 2 ist eine wesentliche Anforderung an die Debug-Software.
Dazu gehört:

- ✓ Einlesen der Output-Datei des GCC-Assemblers
- ✓ Füllen aller nicht verwendeten Adressen mit NOPs
- ✓ Splitten in Byte-breite Dateien für jeden ROM-Baustein
- ✓ Übertragen des Anwendungsprogramms
- ✓ Beschreiben der ROMs
- ✓ Rücklesen der ROMs inklusive Vergleich mit dem geschriebenen Programm

Memory > RAM/ROM

Eine weitere Anforderung an den Debugger ist der Speicherzugriff.
Dazu gehört:

- ✓ Eingabe der Startadresse und Blockgröße durch den Anwender
- ✓ Anzeige des RAM-Inhalts auf dem Bildschirm (Hex-Dump)
- ✓ Export des RAM-Inhalts in eine Excel-kompatible Datei
- ✓ Möglichkeit des manuellen Modifizierens einzelner RAM-Inhalte
- ✓ Import eines Blocks (Angabe von Startadresse und Blockgröße) aus einer Excel-Datei

Register

Eine weitere Anforderung an den Debugger ist der Zugriff auf den Registersatz.
Dazu gehört:

- ✓ Register auslesen/beschreiben
- ✓ Anzeige des Register-Inhalts auf dem Bildschirm
- ✓ Export des Register-Inhalts in eine Excel-kompatible Datei
- ✓ Möglichkeit des manuellen Modifizierens einzelner Register-Inhalte
- ✓ Import eines Registersatzes aus einer Excel-Datei

Steuersignale und Zustand

Des Weiteren soll der Debugger den Zustand des SpaceAge2 anzeigen. Dies erfolgt durch Auslesen der verschiedenen Registerinhalte. Dafür wurde Folgendes gefordert:

- ✓ Zustände im Rechner anzeigen / modifizieren
- ✓ ALU Ergebnis
- ✓ ALU Status
- ✓ EPC Inhalt
- ✓ Befehlsregister (Instruction Register)
- ✓ Data Register / Data Register Input
- ✓ MicroCode Addresse
- ✓ Steuerwerk Status
- ✓ Steuerleitungen aus dem Steuerwerk (9 Bytes, siehe MicroCodeParser)
- ✓ Beschreiben des PC-Eingangs (PC Input Register)
- ✓ Beschreiben des Instruction Registers (IR)

Programmablauf steuern

Zum Steuern des Programmablaufs und zur Verifikation der Programme ist es von elementarer Wichtigkeit, dass der Debugger in der Lage ist, Breakpoints zu setzen, sowie den Rechner automatisch in den Haltemodus zu setzen. Es stellten sich die Anforderungen:

- ✓ Setzen von Breakpoints
- ✓ Run
- ✓ Halt
- ✓ Single Step auf Taktzyklenbasis
- ✓ Single Step auf Befehlszyklenbasis
- ▪ **Abschaltbare „Maskierung“ von Software-Emulationen (noch nicht implementiert)**

Die abschaltbare „Maskierung“ von Software-Emulationen für Multiplikation und Division (eine Emulation wird hierbei als ein Befehl behandelt, Erkennung durch PC-Adressausgabe unterhalb des Beginns des Bereich "„Text“, also im Bereich „Reserved“)

Tracen des Programmablaufs

Der Debugger führt automatisch Einzelschritte aus. Nach jedem Einzelschritt erfasst er die Registerinhalte und sonstige relevante Zustände. Diese Informationen können in eine TXT-Datei exportiert, werden.

2. Die GUI:

Bei Programmstart erfolgt zunächst eine automatisierte Abfrage ob der SpaceAge2 verbunden ist.

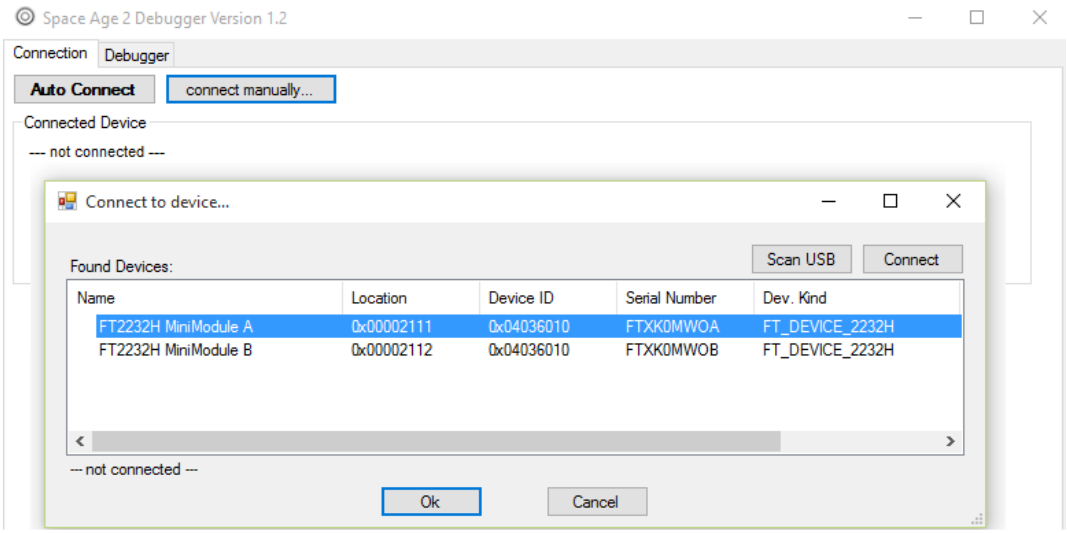


Abbildung 2 Connection Dialog

Mit Klick auf das Tab „Debugger“ wird dann der Debugger wie in Abbildung 3 dargestellt, geladen.

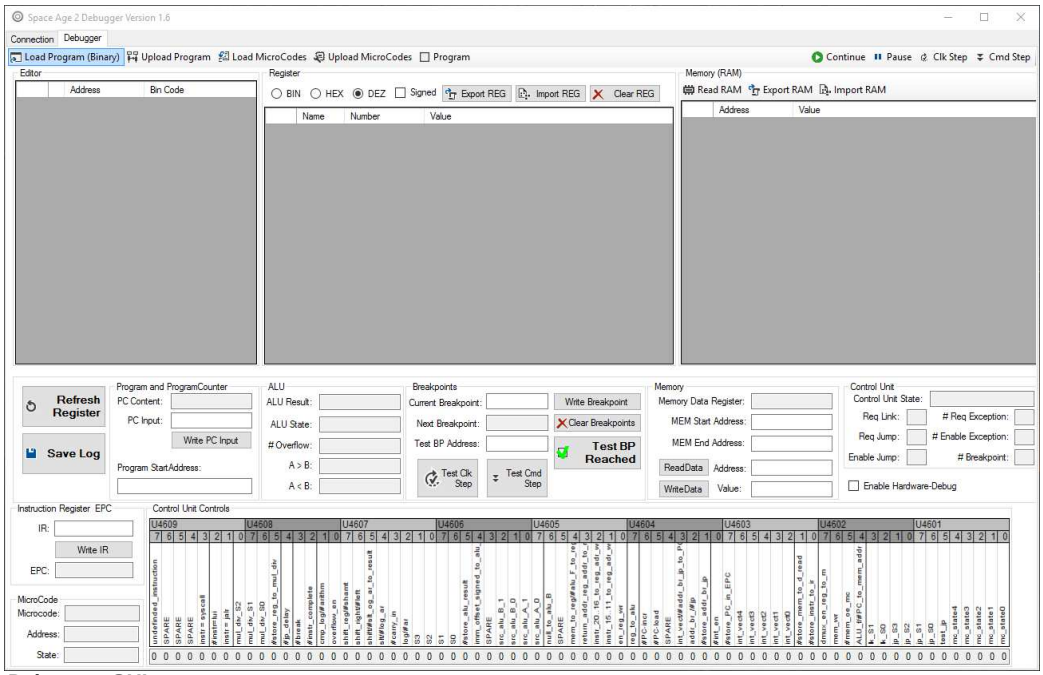


Abbildung 3 Die Debugger GUI

Die Elemente der Debugger GUI sind gruppenweise angeordnet. Oben befindet sich eine Leiste auf der sich Buttons befinden, die die wichtigsten Funktionen des Debuggers auslösen.

Aufbau der GUI:

Abbildung 3 zeigt die grafische Oberfläche des Debuggers. Diese ist in 4 Bereiche unterteilt: einen Editor zur Anzeige des geladenen Programms, sowie Setzen von Breakpoints, einer Registerliste, die den gesamten Registersatz des Space Age 2 abbildet, eine Liste, die den Speicherinhalt in einem bestimmten Adressbereich anzeigt, sowie Anzeigen für Steuer- und Kontrollsignale.

Die Toolbar:

Die Toolbar des Debuggers ist in 2 Teile unterteilt. Der linke Teil, in Abbildung 4 dargestellt, bietet die Funktionen: Programm laden, Programm hochladen, MicroCodes laden, MicroCodes hochladen, sowie eine Checkbox, mit Hilfe derer zwischen den Optionen: „Speicher vor der Installation komplett löschen“ Clear and Program, sowie „Programm einfach nur hochladen“ Program gewählt werden kann.



Abbildung 4 Steuerelemente für die Grundfunktionen

Mit einem Klick auf Load Program (Binary) wird ein fertig kompiliertes Programm im Binärformat in den Debugger geladen. Das geladene Programm wird anschließend im Editor angezeigt und ist bereit für die Übertragung auf den Space Age 2, die durch einen Klick auf Upload Program ausgelöst wird.

Ein Klick auf Load MicroCodes lädt die MicroCode Excel Tabelle in das Programm und parst die darin enthaltenen Mikrobefehle. Diese können im Anschluß mit einem Klick auf Upload MicroCodes auf den Space Age 2 übertragen werden. Zusätzlich wird die Adresse des MicroCodeAddress Registers mit den geparsten MicroCodes abgeglichen. Daraus

Abbildung 5 zeigt den zweiten Teil der Toolbar der für die Steuerung des Programmablaufs verantwortlich ist. Dieser beinhaltet die Funktionen Continue, Pause, ClockStep und CommandStep.



Abbildung 5 Steuerelemente für den Programmablauf

Standardmäßig wird beim Verbinden des Space Age 2 der Rechner durch Starten einer Debugsession in den Haltemodus gesetzt. Dies kann ebenfalls durch Klicken auf Pause erreicht werden. Im Haltemodus ist der Programcounter des Space Age 2 angehalten. Der Haltemodus wird durch einen Klick auf Continue beendet. Ein Single Step auf Taktzyklenbasis wird per Click auf Clk Step durchgeführt. Ein Single Step auf Befehlszyklenbasis wird per Click auf Cmd Step durchgeführt.

Der Editor:

Nach Laden der Binärdatei in den Debugger steht das Programm dem Nutzer im Editor zur Verfügung. Abbildung 6 zeigt den Editor mit einem geladenen Beispielprogramm. In diesem Fall wurden 3 Breakpoints gesetzt, von denen einer bei Adresse 0x30 erreicht wurde.

Das Setzen von Breakpoints erfolgt durch Doppelklick auf eine Zeile in der sich Binärcode befindet. Nach Setzen des Breakpoints wird die gesamte Zeile rot hinterlegt, sowie ein roter Breakpoint-Kreis angezeigt. Wird ein Breakpoint erreicht, so wird die gesamte Zeile gelb hinterlegt, und ein gelber Pfeil erscheint im Breakpoint-Kreis.

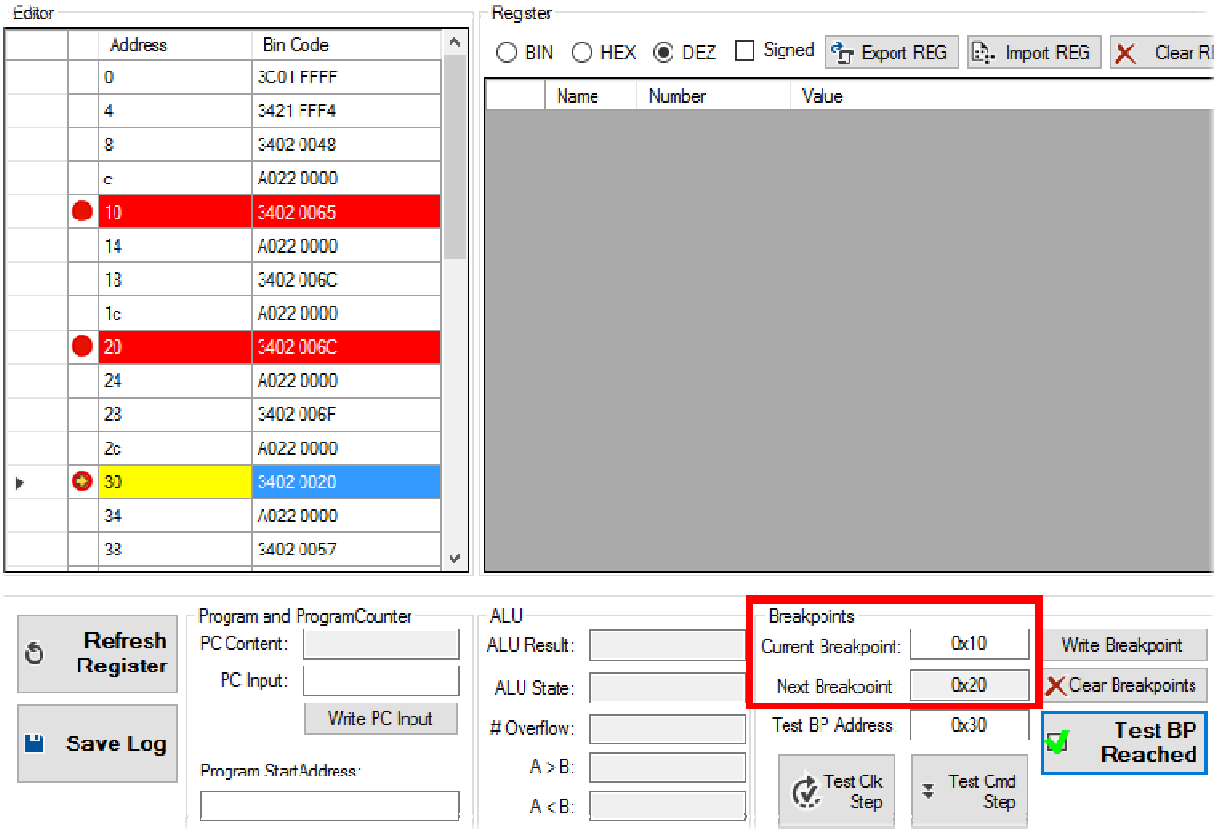
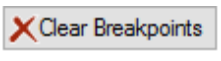


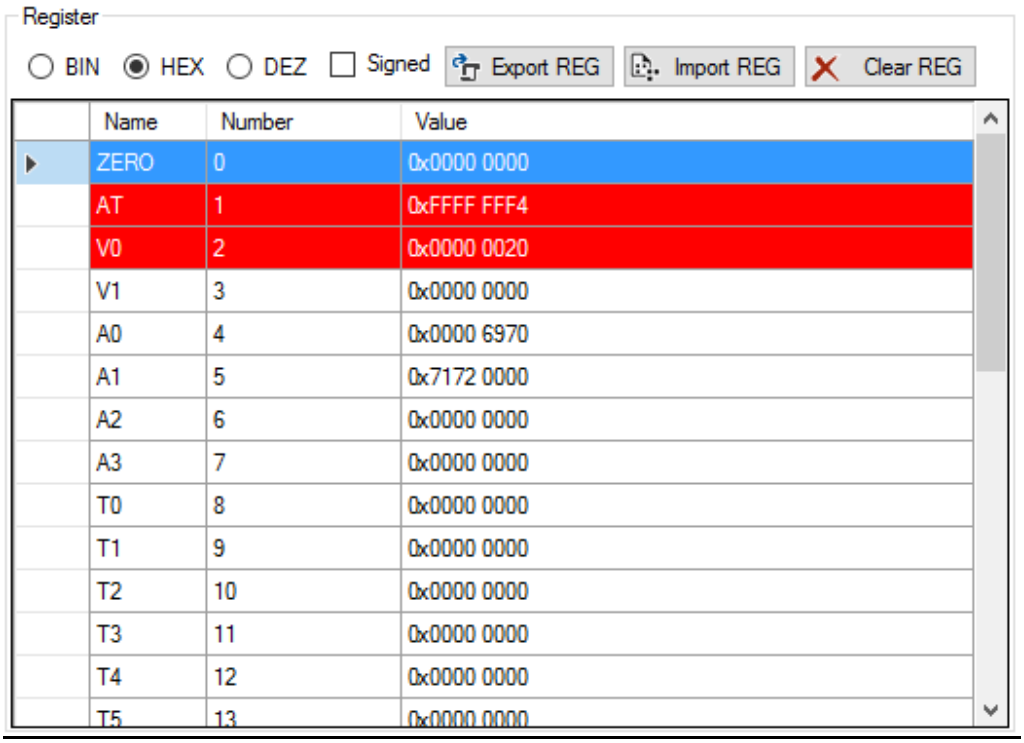
Abbildung 6 Der Editor mit einem geladenem Beispielprogramm, 3 gesetzten Breakpoints und ein erreichter Breakpoint

Es ist außerdem möglich, Breakpoints manuell zu setzen, sowie das Breakpoint Register mit Klick auf 

zu nullen. Dies ist insofern sinnvoll, falls sich der Space Age 2 aufgrund eines in einer früheren DebugSession gesetzten Breakpoints im Haltezustand befindet.

Die Registeransicht:

Die Registeransicht zeigt den vollständigen Registersatz des Space Age 2 an. Verändern sich Registereinträge, so werden die neuen Werte rot markiert dargestellt. Zusätzlich bietet die Registeransicht durch die 3 Radiobuttons und der Checkbox die Möglichkeit, die Darstellungsart der im Registersatz enthaltenen Werte zu ändern. Unterstützt werden hierbei das Binärformat, das Hexadezimalformat, sowie das Dezimalformat. Über die Checkbox kann die Vorzeichenbetrachtung der Zahlenwerte eingestellt werden. Durch Doppelklick auf eine Registerzeile kann der Wert des jeweiligen Registers verändert werden.



	Name	Number	Value
▶	ZERO	0	0x0000 0000
	AT	1	0xFFFF FFF4
	V0	2	0x0000 0020
	V1	3	0x0000 0000
	A0	4	0x0000 6970
	A1	5	0x7172 0000
	A2	6	0x0000 0000
	A3	7	0x0000 0000
	T0	8	0x0000 0000
	T1	9	0x0000 0000
	T2	10	0x0000 0000
	T3	11	0x0000 0000
	T4	12	0x0000 0000
	T5	13	0x0000 0000

Abbildung 7 Beispielansicht des Space Age 2 Registersatzes

Die 3 in Abbildung 7 dargestellten Buttons bieten die Möglichkeit den Registersatz in eine Excel Tabelle zu exportieren, den Import eines Registersatzes aus einer Excel Tabelle, sowie das Nullen des gesamten Registersatzes.

Die Speicheransicht:

Abbildung 8 zeigt einen Beispielauszug aus dem Speicher beginnend ab Adresse 0 bis zur Adresse 128.

Nach Eintragen des Adressbereichs wird der Datensatz durch Klicken auf Read RAM in die Speicheransicht geladen. Wie bei der Registeransicht kann der Speichersatz in eine Excel Tabelle (csv Format) exportiert und aus einer Excel Tabelle importiert werden. Mit einem Doppelklick auf eine Speicherzelle kann der Zellinhalt editiert und verändert werden.

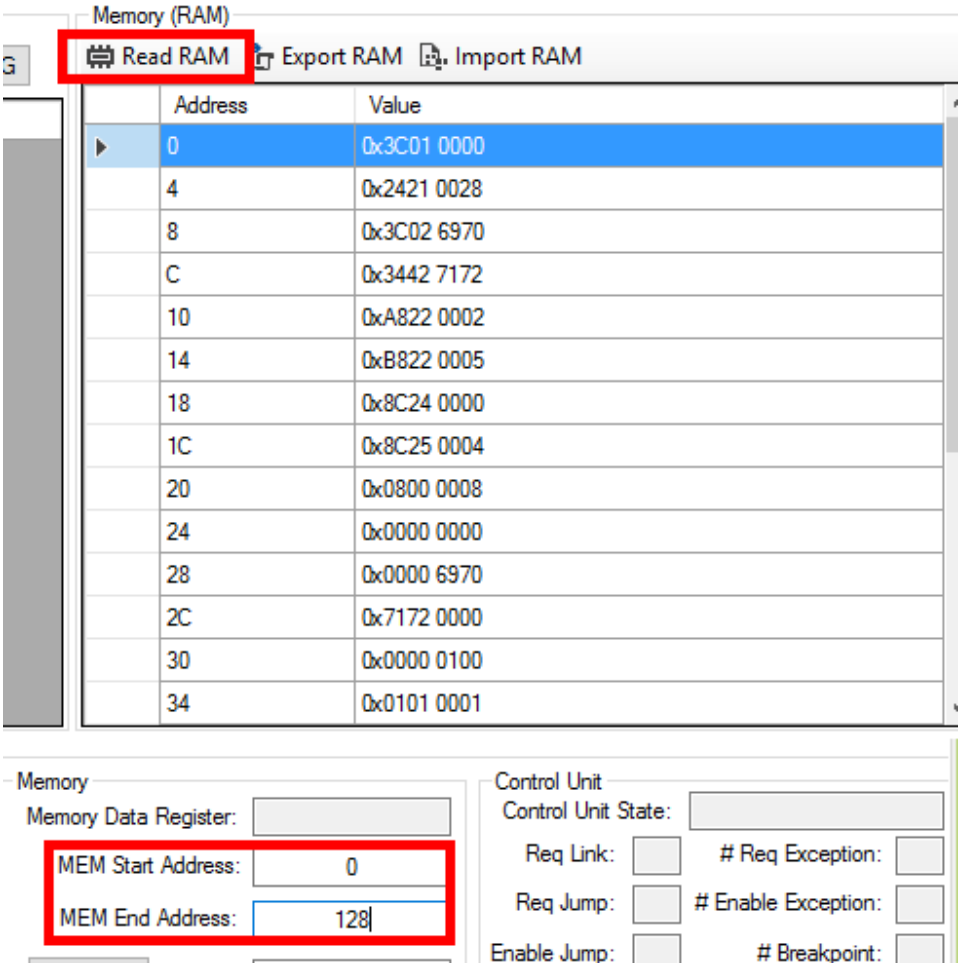


Abbildung 8 Beispielansicht des Space Age 2 Speichers

Die Anzeige der Steuersignale:

Abbildung 9 zeigt alle relevanten Steuersignale des Space Age 2 in Kategorien unterteilt.

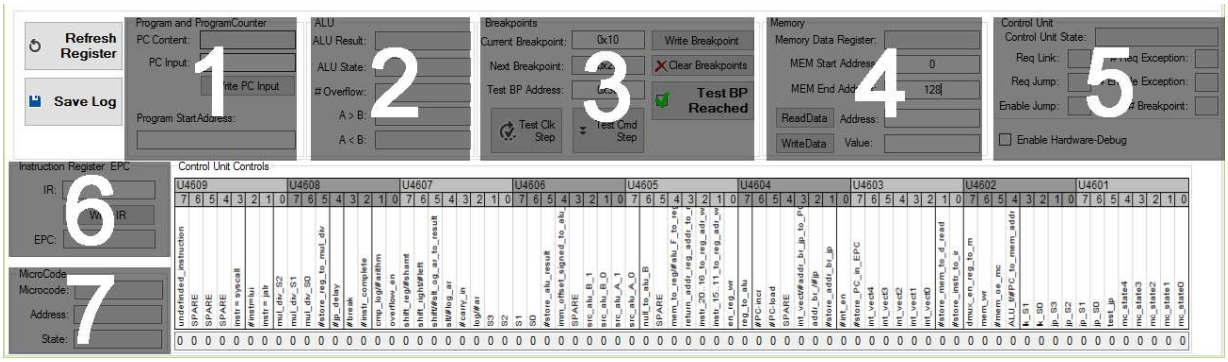
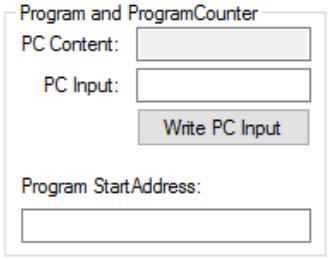


Abbildung 9 Anzeige der Steuersignale

Dies beinhaltet die folgenden Steuersignale:

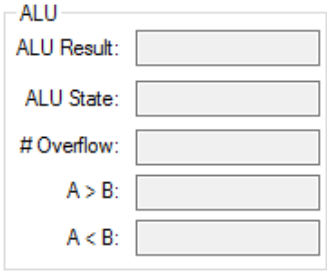
1

Zeigt den Inhalt des Programmzählers (PC) und des Eingangsregisters des Programmzählers (PC Input). Zudem ist es möglich den PC Input zu beschreiben.



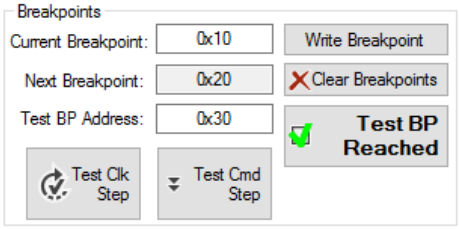
2

Zeigt das Ergebnis und den Status des Rechenwerks (ALU Result, ALU State)
Der ALU State ist in 3 Flags aufgeteilt, das invertierte #Overflow Signal, und die Vergleichssignale A > B und
A < B



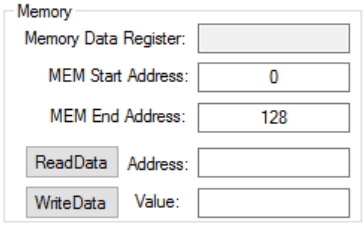
3

Zeigt die Breakpointgruppe – die Adresse des aktuellen Breakpoints und des nächsten Breakpoints. Dazu ist es möglich, eine Breakpoint-Adresse manuell zu setzen, sowie das Breakpoint register zu leeren.



4

Zeigt den Inhalt des Memory Data Registers. Über die Buttons ReadData und Write kann eine Speicherzelle testweise beschrieben/gelesen werden. Die Felder MEM Start Address und MEM End Address werden zum Auslesen des Speichers benötigt.



5

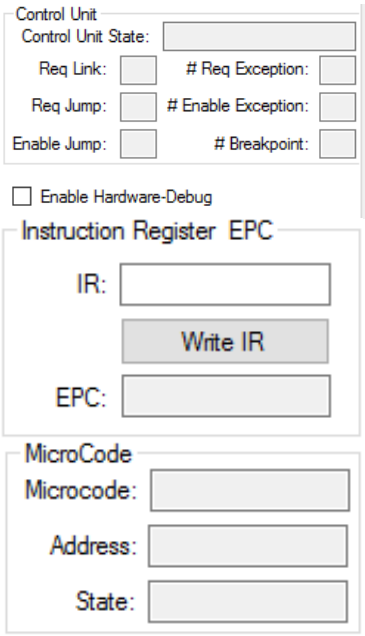
Zeigt den Zustand des Steuerwerks (Control Unit State).Der Control Unit State ist in die 6 Steuerleitung Req Link, Req Jump, Enable Jump, #Req Exception, #Enable Exception, #Breakpoint unterteilt

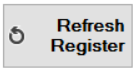
6

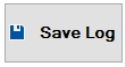
Zeigt den Inhalt des Befehlsregisters (IR), sowie des EPC. Dazu ist es möglich, den Wert des IR zu beschreiben.

7

Zeigt die MicroCode Adresse und den daraus berechneten MicroCode und dessen Zustand



Mit Hilfe des Buttons 

werden sämtliche Registerinhalte erneut abgefragt und angezeigt. Der Debugger loggt die wichtigsten Ereignisse, wie neue Registerinhalte in der Console sowie intern mit. Mit Hilfe des Buttons 

kann der Debug-Log in eine Datei gespeichert werden.

Abbildung 10 zeigt die Steuersignale des aktuellen MicroCodeStates.

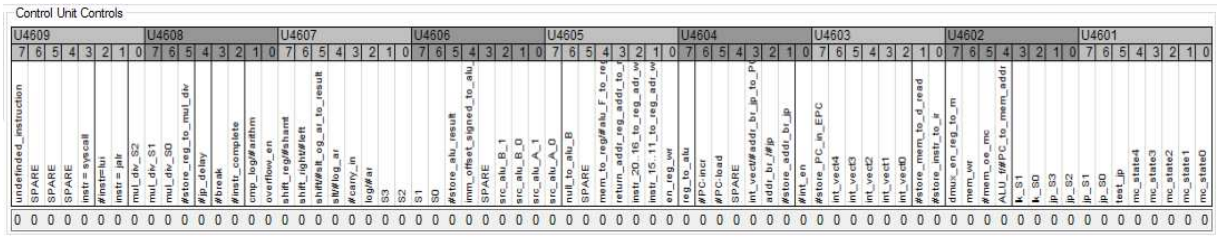


Abbildung 10 Steuersignale des aktuellen MicroCodeStates

3. Externe Programm Abläufe:

Dieser Abschnitt beschreibt **die für den Anwender sichtbaren Abläufe** beim Betreiben des Debuggers.

Übertragen der MicroCodes:

Um MicroCodes auf den SpaceAge2 zu übertragen, müssen diese zunächst in das Programm geladen werden. Klicke zunächst auf Load MicroCodes. Wenn der Parser die Excel Tabelle fertig geparst hat, erscheint eine MessageBox mit dem Hinweis „Parse Done!“. Klicke im Anschluß auf Upload MicroCodes. Es erscheint ein Fortschrittsbalken wie in Abbildung 10 zu sehen.



Abbildung 11 Fortschrittsbalken beim Schreiben der MicroCodes

Nachdem der Schreibvorgang abgeschlossen wurde, wird der MicroCode ROM ausgelesen und die übertragenen Daten mit den geschriebenen Daten verglichen, wie in Abbildung 11 zu sehen.



Abbildung 12 Fortschrittsbalken beim Verifizieren der geschriebenen MicroCodes

Sind beide Datensätze identisch, wurden die MicroCodes erfolgreich übertragen und es erscheint eine MessageBox wie in Abbildung 12 zu sehen.

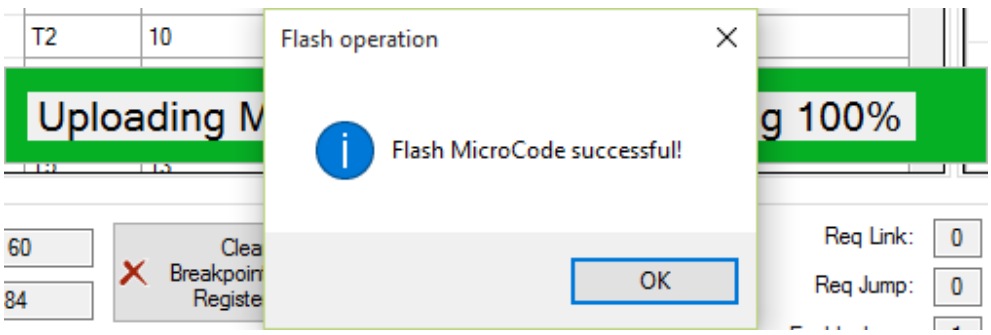


Abbildung 13 Erfolgreich durchgeführte Übertragung der MicroCodes

Übertragen des Programms:

Um ein Programm auf den Space Age 2 zu übertragen, klicke auf Load Program (Binary) und anschließend auf den Button Upload Program. Standardmäßig wird ein Programm **ohne** vorheriges Löschen des Speichers übertragen. Ist ein vorheriges Löschen des Speichers gewünscht, so klicke auf Program. Jetzt zeigt die Checkbox Clear and Program und löscht den Speicher vor dem Hochladen des Programms. Es wird ein Fortschrittsbalken gezeigt, sowie in der Console der aktuelle Fortschritt angezeigt.

Der Rest der Übertragung erfolgt analog zur Übertragung der MicroCodes (siehe Abbildung 14).

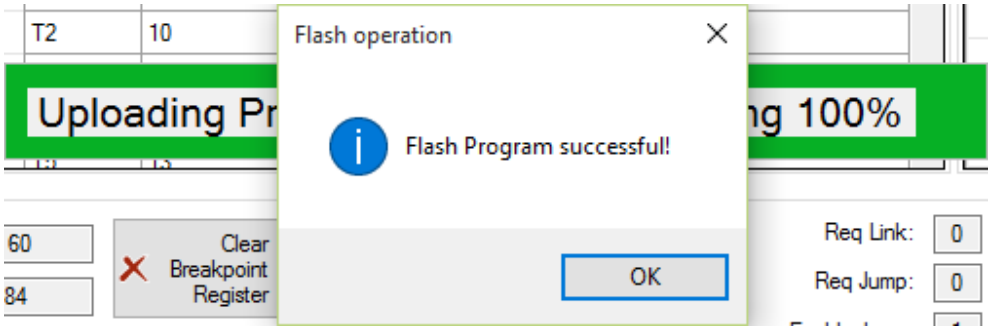


Abbildung 14 Abgeschlossene Übertragung eines Programms

ClockStep:

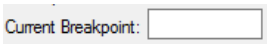
Ein Klick auf Clk Step bewirkt einen taktbasierten Einzelschritt. Dabei wird die MicroCode Adresse inkrementiert, was den aktuellen MicroCodeState auf den Nachfolgezustand setzt. Nach jedem ClockStep werden alle Registerinhalte neu abgefragt. Ein ClockStep kann auch ohne gesetzte Breakpoints durchgeführt werden.

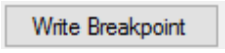
CommandStep:

Ein Klick auf Cmd Step bewirkt einen befehlsbasierten Einzelschritt. Dabei wird der aktuelle Befehl beendet und am Ende der Programcounter inkrementiert. Nach jedem CommandStep werden alle Registerinhalte neu abgefragt. Ein CommandStep kann auch ohne gesetzte Breakpoints durchgeführt werden.

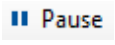
Breakpoints in Kombination mit Pause und Continue:

Nach dem der Debugger gestartet und mit dem Space Age 2 verbunden ist, wird dieser in den Haltezustand versetzt. Mit Klick auf 

wird der Haltezustand beendet und der Space Age 2 läuft normal weiter. **Wichtig ist hierbei, dass das Lesen und Beschreiben von Registern nur im Haltezustand fehlerfrei abläuft.** Ein Breakpoint wird über Doppelklick auf eine Editorzelle oder mit dem Eintragen einer Breakpoint-Adresse in 

und einem Klick auf 

gesetzt. Jeder gesetzte Breakpoint wird intern in einer Liste abgespeichert, damit nach Erreichen eines Breakpoints der nächste Breakpoint automatisch gesetzt werden kann.

Ein Klick auf 

setzt den Space Age 2 in den Haltezustand.

Der Debugger betreibt intern einen Readthread der den BreakpointStatus pollt und mit dem aktuell gesetzten Breakpoint vergleicht. Ist dies der Fall, so wird der Event BreakpointReached mit der Breakpoint-Adresse geworfen.

Dieser wird in der GUI entsprechend behandelt: die mit der Breakpoint-Adresse verknüpfte Editorzeile wird farbig verändert, es wird der nächste Breakpoint berechnet und gesetzt, sowie alle RegisterInhalte aktualisiert.

Ändern der Zahlendarstellung:

Mit einem Klick auf einen der Radiobuttons wird die Zahlendarstellung ausgewählter Registerinhalte verändert. Dabei wird die Funktion ShowRegisterValuesAccordingToNumberRepresentation.

```
private void OnRadioButtonBinClick(object sender, EventArgs e)
{
  if (_radioButtonBinary.Checked && (_mode != Mode.Binary))
  {
    _mode = Mode.Binary;
    ShowRegisterValuesAccordingToNumberRepresentation();
  }
}
```

Diese Methode verändert die Zahlendarstellung des geladenen Space Age 2 Registersatzes, der geladenen Speicherzellen, sowie die Inhalte des ProgramCounters, der aktuell angezeigten Breakpoint-Adresse, der nächsten Breakpoint-Adresse, des ALU-Ergebnisses, der MicroCode-Adresse, des InstructionRegisters, sowie des MemoryDataRegisters.

Nachfolgend ist der Code der Funktionen ShowRegisterValuesAccordingToNumberRepresentation und TranslateValueToRepresentation aufgelistet.

```
private void ShowRegisterValuesAccordingToNumberRepresentation()
{
  for (var i = 0; i < _registerGridView.Rows.Count; i++)
  {
    _registerGridView[_colRegValue.Name, i].Value = TranslateValueToRepresentation(_registers[i].Value);
    _registerGridView[_colRegValue.Name, i].Style = Equals(_registerGridView[_colRegValue.Name, i].Style, _breakpointSetStyle)
      ? _breakpointSetStyle : _breakpointUnsetStyle;
  }


  for (var i = 0; i < _memoryGridView.Rows.Count; i++)
  {
    var cell = _memoryGridView.Rows[i].Cells[_colMemValue.Name];
    var value = cell.Value;

    try
    {
      value = TranslateValueToRepresentation(_memory[i]);
    }
    catch (Exception ex)
    {
      ErrorDisplay.Show(ex);
      Logger.WriteLine(ex.Message);
    }

    _memoryGridView.Rows[i].Cells[_colMemValue.Name].Value = value;
  }

  UpdateTextBox(_pcContentTextBox.Name, TranslateValueToRepresentation(_pcValue));
  UpdateTextBox(_currentBreakPointTextBox.Name, TranslateValueToRepresentation(_currentBreakPointAddress));
  UpdateTextBox(_nextBreakPointTextBox.Name, TranslateValueToRepresentation(_nextBreakPointAddress));
  UpdateTextBox(_aluResultTextBox.Name, TranslateValueToRepresentation(_aluResult));
  UpdateTextBox(_microCodeAddressTextBox.Name, TranslateValueToRepresentation(_microCodeAddressContent));
  UpdateTextBox(_instructionRegisterContentTextBox.Name, TranslateValueToRepresentation(_instructionRegisterContent));
  UpdateTextBox(_memoryDataRegisterTextBox.Name, TranslateValueToRepresentation(_memoryDataRegisterContent));
}
```

```
private string TranslateValueToRepresentation(int value)
{
  var result = "";
  try
  {
    switch (_mode)
    {
      case Mode.Binary:
        result = Helper.Helper.InsertStringAtInterval(Convert.ToString(value, 2).PadLeft(32, '0'), " ", 4);
        break;

      case Mode.Dez:
        result = _representSigned ? value.ToString() : ((uint)value).ToString();
        break;

      case Mode.Hex:
        result = "0x" + Helper.Helper.InsertStringAtInterval(Convert.ToString(value, 16).PadLeft(8, '0').ToUpper(), " ", 4);
        break;
    }
  }
  catch (Exception ex)
  {
    ErrorDisplay.Show(ex);
    Logger.WriteLine(ex.Message);
  }
  return result;
}
```

4. Programm Struktur:

Dieser Abschnitt beschreibt die Struktur und die internen Abläufe der Debug-Software. Abbildung 15 zeigt die Struktur des Space Age 2 Debuggers. Die Software besteht aus 3 Teilen, der GUI, einer Mittelschicht und einem Treiber, der den FTDI ansteuert, der mit dem Space Age 2 verbunden ist.

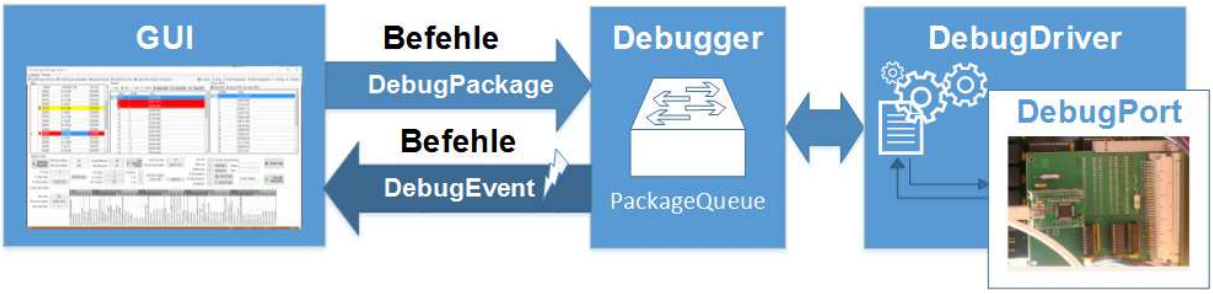


Abbildung 15 Struktur des Space Age 2 Debuggers

Die GUI:

Die GUI verarbeitet sämtliche Nutzereingaben und schickt Befehle an den Debugger, der diese asynchron verarbeitet. Jeder Befehl ist dabei in ein DebugPackage gekapselt, dessen Struktur in Abbildung 16 dargestellt ist.

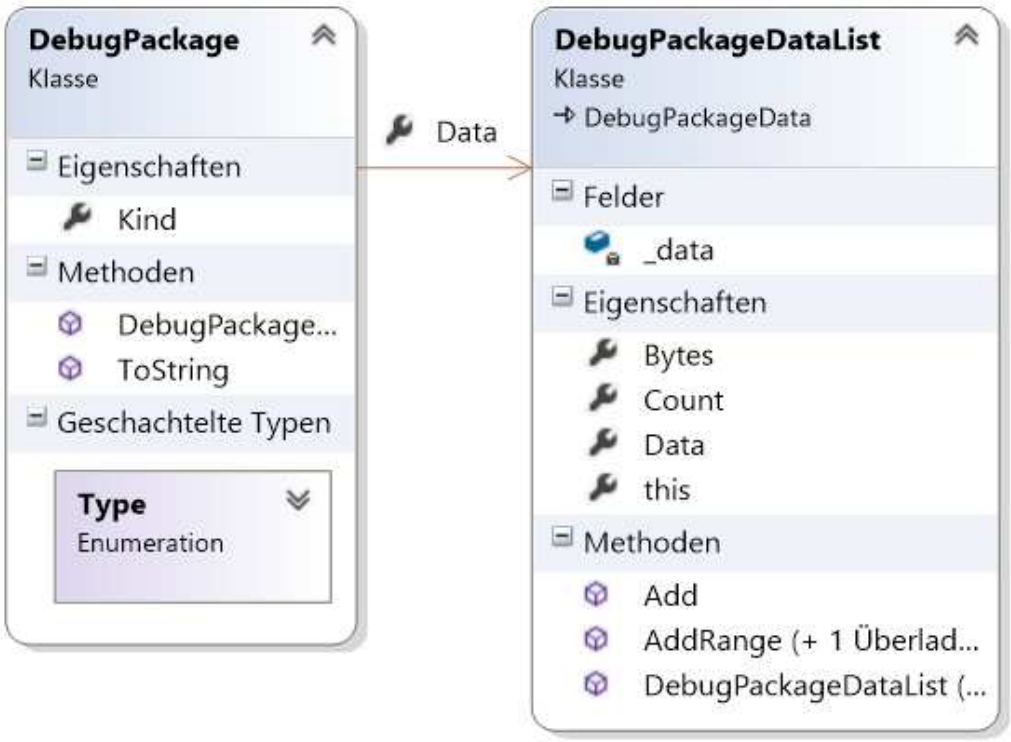


Abbildung 16 Struktur eines DebugPackages

Ein DebugPackage besteht aus einem Typ (Eigenschaft: Kind), der für eine bestimmte Operation steht, die der Debugger ausführen soll, sobald das Paket entpackt (geparst) wird. Jedes DebugPackage enthält über die Eigenschaft Data eine DebugPackageDataList, die eine Liste aus DebugPackageData-Objekten enthält. Abbildung 17 zeigt die abstrakte Basisklasse DebugPackageData. Diese besitzt die Eigenschaften Bytes und Data, welche von jeder ableitenden Klasse implementiert werden müssen. Insgesamt sind die Datentypen Bool, Byte, Int, Double, SpaceAgeRegister und Status implementiert.

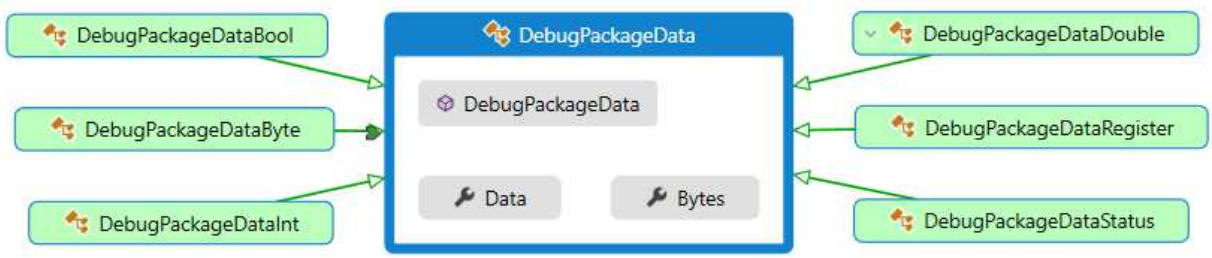


Abbildung 17 Basisklasse DebugPackageData und ihre Ableitungen

Abbildung 18 zeigt die Implementation der von der Basisklasse ableitenden Klasse DebugPackageDataInt. Diese hält einen Integer-Wert und überschreibt die Funktion Bytes und Data.

```
15 Verweise | Benjamin Bös, Vor 41 Tagen | 1 Änderung
public class DebugPackageDataInt : DebugPackageData
{
  private int _data;

  23 Verweise | Benjamin Bös, Vor 41 Tagen | 1 Änderung
  public override byte[] Bytes
  {
    get { return BitConverter.GetBytes(_data); }
  }

  39 Verweise | Benjamin Bös, Vor 41 Tagen | 1 Änderung
  public override object Data
  {
    get { return _data; }
    set { _data = (int)value; }
  }

  5 Verweise | Benjamin Bös, Vor 41 Tagen | 1 Änderung
  public DebugPackageDataInt(int data)
  {
    _data = data;
  }
}
```

Abbildung 18 Implementation der ableitenden Klasse DebugPackageDataInt

Ablauf beim Flashen der MicroCodes:

Durch einen Klick auf, wird intern der DebugPackageQueue ein neues DebugPackage zugefügt, wie in Abbildung 19 zu sehen.

```
_debugger.AddCommand(new DebugPackage(DebugPackage.Type.FlashMicroCode, _parser.MicroCodeByteList()));
```

Abbildung 19 Erzeugen eines DebugPackage mit dem Befehl, den MicroCode ROM zu beschreiben

Der Readthread des Debuggers pollt in der Zwischenzeit die PackageQueue. Ist ein Paket enthalten, so wird dieses der Methode ParseCommand übergeben wie in Abbildung 20 zu sehen. ParseCommand entscheidet anhand des PaketTyps, das Paket an die Funktion FlashMicroCode weiterzuleiten. Diese Methode wiederrum übergibt die Byteliste des DebugPackage der FlashMicroCode Funktion des DebugDrivers.

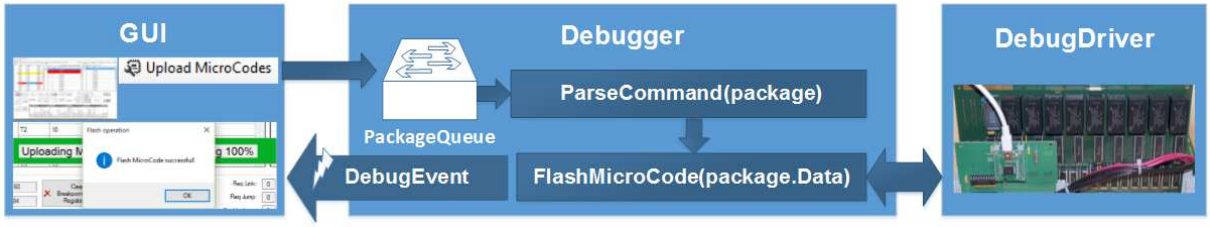


Abbildung 20 Ablauf bei der Durchführung des Befehls "FlashMicroCode"

Die Methode DebugDriver.FlashMicroCode überprüft zunächst ob das übergebene Byte-Array ganzzahlig durch 9 teilbar ist, da ein Byte-Array der Größe 72 KiloByte (9 * 8192 Bytes) erwartet wird. Anschließend wird eine Befehlssequenz initialisiert und abgeschickt, die die Steuerleitungen für den Datentransfer zum MicroCode-ROM setzt, wie in Abbildung 21 zu sehen.

```
//Zugriffsbedingungen herstellen
BeginCommandSequence();
WriteData(DebugAdressesForWrite.MicroCode_Control_Wr_Ctrl_Mc, 0x1);
WriteData(DebugAdressesForWrite.MicroCode_Control_Wr_Debug_To_D_Mc, 2);
WriteData(DebugAdressesForWrite.MicroCode_Control_Wr_Debug_To_Addr_Mc, 1);
CommitCommandSequence();
```

Abbildung 21 Setzen der Steuerleitungen für den Datentransfer zum MicroCode-ROM

Abbildung 22 zeigt die Funktion WriteData, mit Hilfe derer zunächst ein Adress-Byte und dann ein Daten-Byte geschrieben wird.

```
/// <summary>
/// Wrapper method for write operations.
/// Writes address first to data bus, then write command with data.
/// </summary>
2 Verweise | benjamin.boes + 1, Vor 15 Tagen | 4 Änderungen
private void WriteData(byte address, byte data)
{
  _debugPort.WriteAddress(address);
  _debugPort.WriteData(data);
}
```

Abbildung 22 Die Funktion WriteData schreibt zunächst ein Address-Byte und dann ein Daten-Byte

Die Funktionsweise des DebugPorts wird zu einem späteren Zeitpunkt erläutert.

Im Anschluß daran wird das 72 KB Array mittels einer For-Schleife byteweise übertragen. Dabei wird die Index-Variable als MicroCode-Adresse interpretiert. Es wird eine Befehlssequenz initialisiert, der zunächst die MicroCode-Adresse übergeben wird (Abbildung 23).

```
WriteData(DebugAdressesForWrite.MicroCodeAddr_B0, microcodeaddress[0]);
WriteData(DebugAdressesForWrite.MicroCodeAddr_B1, microcodeaddress[1]);
```

Abbildung 23 Setzen der MicroCode Adresse

Abbildung 24 zeigt die byte-weise Übertragung der MicroCode Bytes auf den Space Age 2. Dazu wird in jedem Schleifendurchlauf zunächst der ROM Block ausgewählt und dann das MicroCode Byte übertragen. Nach jedem Durchlauf bei dem ein 9-Byte Datensatz geschrieben wurde, wird der Fortschrittsbalken über die Methode UpdateFlashStatus aktualisiert.

```
//Schleife die die einzelnen (9) Speicherbausteine beschreibt
for (var j = 0; j < 9; j++)
{
    //Speicherbaustein auswählen
    WriteData(DebugAdressesForWrite.MicroCode_Control_Wr_Mc_Not_Wr, j);
    //Daten in den Speicherbaustein schreiben; darauf achten, dass die vorher geschrieben Adressen
    //mal 9 als Offset addiert werden müssen
    WriteData((int)DebugAdressesForWrite.MicroCodeRomData_B0 + j, program[(i * 9) + j]);

    //Schreibsequenz ausführen
    WriteData(DebugAdressesForWrite.MicroCode_Control_Wr_Ctrl_Mc, 0x0);
    WriteData(DebugAdressesForWrite.MicroCode_Control_Wr_Ctrl_Mc, 0x2);
    WriteData(DebugAdressesForWrite.MicroCode_Control_Wr_Ctrl_Mc, 0x0);
    WriteData(DebugAdressesForWrite.MicroCode_Control_Wr_Ctrl_Mc, 0x1);
    CommitCommandSequence();
}
//Fortschrittsbalken aktualisieren
var percentage = i / 8192.0 * 100;
UpdateFlashStatus("Write", percentage);
```

Abbildung 24 Schreiben der 9 Bytes je Adresse und anschließendes Updaten des prozentualen Fortschritts

Am Ende werden die Steuerleitungen wieder auf den Normalzustand zurückgesetzt (siehe Abbildung 25).

```
//Zugriff wieder auf Normalzustand zurücksetzen
BeginCommandSequence();
WriteData(DebugAdressesForWrite.MicroCode_Control_Wr_Ctrl_Mc, 0x0);
WriteData(DebugAdressesForWrite.MicroCode_Control_Wr_Debug_To_D_Mc, 0);
WriteData(DebugAdressesForWrite.MicroCode_Control_Wr_Debug_To_Addr_Mc, 0);
CommitCommandSequence();
```

Abbildung 25 Zurücksetzen der Steuerleitungen

Nach dem der Schreibvorgang abgeschlossen wurde, werden die geschriebenen MicroCodes verifiziert, in dem der 72 KB Inhalt des MicroCode ROMS zurückgelesen wird, wie in Abbildung 26 zu erkennen. Die Methode ReadMicroCode funktioniert analog zu FlashMicroCode, nur das diesmal Daten-Bytes gelesen werden. FlashMicroCode liefert true zurück, wenn die gelesenen MicroCode ROM Bytes mit den geschriebenen übereinstimmen. Auch bei dieser Methode wird der Fortschrittsbalken aktualisiert.

```
//Byte-Array für die zurückgelesenen Daten
byte[] writtenprogram;
//Anweisung für das Verifizieren. Dies wird immer ausgeführt
ReadMicroCode(out writtenprogram, program.Length);
//Überprüfung ob geschriebenes und gelesenes Array übereinstimmen
result = writtenprogram.SequenceEqual(program);
```

Abbildung 26 Rücklesen der MicroCode Bytes

Bei ReadMicroCode werden zunächst die Steuerleitungen wie in Abbildung 27 gesetzt.

```
BeginCommandSequence();
//Zugriffssignale anlegen
WriteData(DebugAdressesForWrite.MicroCode_Control_Wr_Ctrl_Mc, 1);
WriteData(DebugAdressesForWrite.MicroCode_Control_Wr_Debug_To_D_Mc, 0);
WriteData(DebugAdressesForWrite.MicroCode_Control_Wr_Debug_To_Addr_Mc, 1);
CommitCommandSequence();
```

Abbildung 27 Setzen der Steuerleitung für den Lesevorgang

Im Anschluß werden die 9 Byte-Adressen gesetzt.

```
//Zu lesende Adresse anlegen
WriteData(DebugAdressesForWrite.MicroCodeAddr_B0, microcodeaddress[0]);
WriteData(DebugAdressesForWrite.MicroCodeAddr_B1, microcodeaddress[1]);
//Die 9 Bytes lesen
for (var j = 0; j < 9; j++)
{
    ReadData((int)DebugAdressesForRead.MicroCodeRomData_B0 + j);
}
CommitCommandSequence();
```

Abbildung 28 Setzen der Rückleseadressen des MicroCode ROMs

Nachdem die Steuerleitungen wieder auf den Normalzustand zurückgesetzt werden, wird wie in Abbildung 29 zu sehen, gewartet, bis genügend Bytes zum Lesen bereit stehen. Danach werden diese zurückgelesen.

```
//Warten, dass die angeforderten Bytes zurückgelesen wurden
while (true)
{
    if (_debugPort.GetRxAvailable() >= (laufgroesse * 9))
    {
        break;
    }
    Thread.Sleep(10);
}
//Daten ins ZielArray zurücklesen; Darauf achten, dass durchlauf*900 als Offset
//addiert werden muss
for (var i = 0; i < (laufgroesse * 9); i++)
{
    _debugPort.ReadByte(out program[(durchlauf * 900) + i]);
}
```

Abbildung 29 Rücklesen des MicroCode ROMs