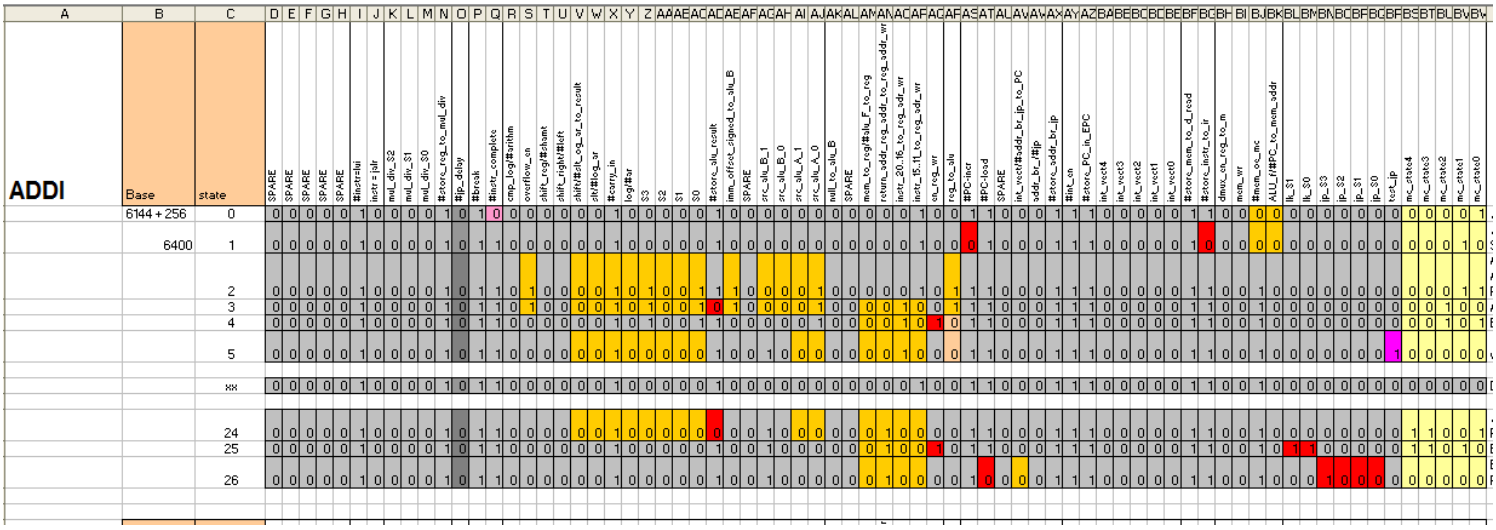
Doku Mikrocode Parser
(Quick and Dirty Edition)

Inhaltsverzeichnis

1. Aufbau der zu parsenden Exceltabelle 2
2. Programmbedienung 3
    2.1 Eingabeparameter 3
    2.2 Ausgabedateien für Debugzwecke 3
    2.3 Ausgabedateien für VHDL Simulation/Synthese 3
    2.4 Ausgabedateien für ROM ICs in Hardware 3
3. Programmaufbau 4
4. Fehlermeldungen 5
    CSVdump: Error opening file 5
    CSVdump: ERROR not enough columns at: <mnemonic>, only <Spaltenanzahl> 5
    CSVdump: ERROR at finding opcode redefinition, found at: <mnemonic> 5
    CSVdump: ERROR opcode redefinition found at: <mnemonic>, but no real opcode to redefine 5
    CSVdump: ERROR reading CSV Line at: <mnemonic> 5
    CSVdump: ERROR reading Baseadress: <mnemonic> 5
    CSVdump: ERROR no control bit definition at: <mnemonic> 5
    CSVdump: ERROR reading state address (empty), at: <mnemonic> 5
    CSVdump: ERROR reading state address (too big), at: <mnemonic> 5
    CSVdump: ERROR reading control bits, at: <mnemonic> 5
    CSVdump: ERROR no defaultpattern at: <mnemonic> 6
    CSVdump: ERROR wrong defaultpattern at: <mnemonic> 6
    CSVdump: ERROR copying controlbits at: <mnemonic> from: <mnemonic> 6
    CSVdump: ERROR ROM too small for: <mnemonic>, with base: <basisadresse> 6

1. Aufbau der zu parsenden Exceltabelle



Der Ankerpunkt ist das Mnemonik des Prozessorbefehls, von diesem aus werden über Offsets die Feldern wie Basisadresse, Redefinition, Zustandsadressen und die Kontrollbits selber erreicht.

Dies bedeutet:
- Das Mnemonik muss sich immer in Spalte A befinden.
- Die Basisadresse befindet sich 1 Spalte rechts und 2 Zeilen unterhalb vom Mnemonik.
- Die Zustandsadressen befinden sich 2 Spalten rechts und ab 1 Zeile unterhalb vom Mnemonik.
- Die Kontrollbits befinden sich 3 Spalten rechts und ab 1 Zeile unterhalb vom Mnemonik.
- Es müssen mindestens 2 Zeilen Zustände deklariert sein.
- Es müssen pro Zeile genau 72 Kontrollbits vorhanden sein, diese müssen 0 oder 1 sein.
- Das höherwertigste Bit steht links und es werden 8 Bit Gruppen geparsed.
- Die Zustandsadressen dürfen von 0 bis 31 reichen, wobei „xx“ als Füllzeile erkannt wird.
- Das Parsen eines Befehls wird beendet wenn 2 Leerzeilen erkannt wurden.
- Wenn Mikrocodes identisch sind so ist eine Redefinition möglich auf der selben Zeile wie das Mnemonik in Spalte E. Der Inhalt muss lauten: „Microcode identisch zu XXXXX“

2. Programmbedienung

2.1 Eingabeparameter

Die Exceltabelle muss zunächst in Excel geöffnet und als .csv Datei exportiert werden.
Als Trennzeichen wird dazu das standardmäßige Semikolon genutzt.
Das Programm wird aufgerufen mit „./CSVdump Dateiname.csv“

Nun wird diese Datei geparsed und eventuelle Fehlermeldungen werden über die Konsole ausgegeben.

2.2 Ausgabedateien für Debugzwecke

Nachdem sich das Programm beendet hat stehen 4 Debug Dateien zur Verfügung, diese beinhalten den jeweiligen Stand des geparsten Mikrocodes.

-> CSVdump_debug_fresh_parsed.txt
Beinhaltet die Rohdaten wie diese gefunden wurden, also ohne Auffüllung von Leerzeilen oder kopieren von Redefinitionen.

-> CSVdump_debug_default_filled.txt
Beinhaltet die Rohdaten mit den aufgefüllten Zeilen.

-> CSVdump_debug_finish.txt
Beinhaltet die fertigen Rohdaten inklusive kopierten Befehlen.

-> CSVdump_debug_ROM.txt
Beinhaltet die Daten wie diese ins ROM geschrieben werden. Hier sind bereits alle Befehle nach der Basisadresse sortiert hineinkopiert.

2.3 Ausgabedateien für VHDL Simulation/Synthese

Die Dateien U4601.mif bis U4609.mif beinhalten jeweils eine 8Bit Spalte des ROMs für das gleichnamige IC im Datenblatt.

2.4 Ausgabedateien für ROM ICs in Hardware

Folgt wenn das benötigte Format bekannt ist.
Update: Macht jetzt der Debugger!

3. Programmaufbau

Das Programm öffnet zuerst die Datei und sucht in der ersten Spalte nach Zeichenketten. Sobald eine Zeichenkette gefunden wurde, wird mithilfe der Funktion
check_mnemonic(field, mnemonics, mnemonics_s)
überprüft ob es sich um ein bekanntes Mnemonik handelt und wenn nicht, dann wird weiter Spalte A durchsucht.

Handelt es sich um ein bekanntes Mnemonik, so wird dessen Name in das struct opcode_tokenize struct kopiert und **parse_opcode()** aufgerufen.

Sobald alle Opcodes eingelesen sind, werden die Leerzeilen mit der gefundenen Füllzeile im struct des jeweiligen Opcodes überschrieben.

Wenn dies abgeschlossen ist, dann werden die Redefinierten Opcodes kopiert.

Nun ist alles so weit fertig, dass der ROM Inhalt erzeugt werden kann, ab jetzt stehen alle Befehle sortiert nach der Basisadresse zur Verfügung.

Schlussendlich werden die 8Bit Spalten aus dem ROM Inhalt ausgelesen und in die .mif Dateien kopiert.

parse_opcode() :
Diese Funktion prüft zuerst ob eine Redefinition vorliegt oder nicht.
Insofern eine vorliegt wird der Befehl zur Redefinition geparsed und die Basisadresse des Originalbefehls.

Liegt keine Redefinition vor so werden die ersten beiden Kontrolbitzeilen mit **get_opcodeline()** geparsed und bei der 2. Zeile auch die Basisasadresse. Anschließend werden so viele Befehlszeilen geparsed bis 2 Leerzeilen auftreten.

get_opcodeline():
Zuerst wird die Zustandsadresse eingelesen und „xx“ nach 32 übersetzt.
Nun werden die 72 Kontrollbits eingelesen und in dem struct gespeichert.

buf->controlbits[op_state_adress][8 - (i/8)] |= (field[0]-48)<<(7-(i%8));

[8 - (i/8)] -> passende Bytespalte, U4609 ist links, also höherwertigste Bytespalte auch
(field[0]-48) -> aus dem Character der CSV Datei einen int berechnen
<<(7-(i%8)) -> Schlussendlich das Bit an die richtige Stelle schieben.

4. Fehlermeldungen

CSVdump: Error opening file

Die beim Programmstart angegebene Datei konnte nicht geöffnet werden.

CSVdump: ERROR not enough columns at: <mnemonic>, only <Spaltenanzahl>

Es wurden nicht genügend Spalten gefunden um den Parser für diese Zeile starten zu können.

CSVdump: ERROR at finding opcode redefinition, found at: <mnemonic>

Es wurde das Stichwort „identisch“ gefunden aber kein Leerzeichen mehr hinter diesem Stwichwort an dem sich der redefinierende mnemonic befinden müsste.

CSVdump: ERROR opcode redefinition found at: <mnemonic>, but no real opcode to redefine

Es wurde eine redefinition gefunden, aber der erkannte mnemonic ist nicht in der mnemonic Tabelle des Parsingprogramms ausfgeführt.

CSVdump: ERROR reading CSV Line at: <mnemonic>

Beim einlesen einer neuen Zeile war die CSV Datei zu Ende.

CSVdump: ERROR reading Baseadress: <mnemonic>

Entweder ist das Feld leer oder enthält keine gültige Zahl.

CSVdump: ERROR no control bit definition at: <mnemonic>

Es wurde keine Zustandsadresse für die Controlbitdefinition gefunden.
Die ersten beiden Zeilen der Befehlsdefinition müssen Controlbits entahlten.

CSVdump: ERROR reading state address (empty), at: <mnemonic>

Diese Fehlermeldung tritt erst ab der dritten Kontrolbitzeile auf.
Es wurde keine Zustandsadresse für die Controlbitdefinition gefunden.

CSVdump: ERROR reading state address (too big), at: <mnemonic>

Die Zustandsadresse des Mikrocodes ist größer als 31.

CSVdump: ERROR reading control bits, at: <mnemonic>

Ein Kontrollbit war nicht 0 oder 1.

CSVdump: ERROR no defaultpattern at: <mnemonic>

Es wurde keine Zeile gefunden welche die Kontrolbits für nicht definierte Zustände enthält. (Die „xx“ Zeile)

CSVdump: ERROR wrong defaultpattern at: <mnemonic>

Der Parser lieferte einen Fehler beim lesen der Zeile für den Zustand nicht definierter Zeilen und wird daher nicht verwendet.

CSVdump: ERROR copying controlbits at: <mnemonic> from: <mnemonic>

Das mnemonic der Redefinition zum Kontrolbitkopieren ist fehlerhaft.

CSVdump: ERROR ROM too small for: <mnemonic>, with base: <basisadresse>

Fehler beim kopieren des Tokenstructs in ein Bitfield. Die Basisadresse + Mikrcode ist größer als die angegebene ROM Größe. (Adressmäßig)