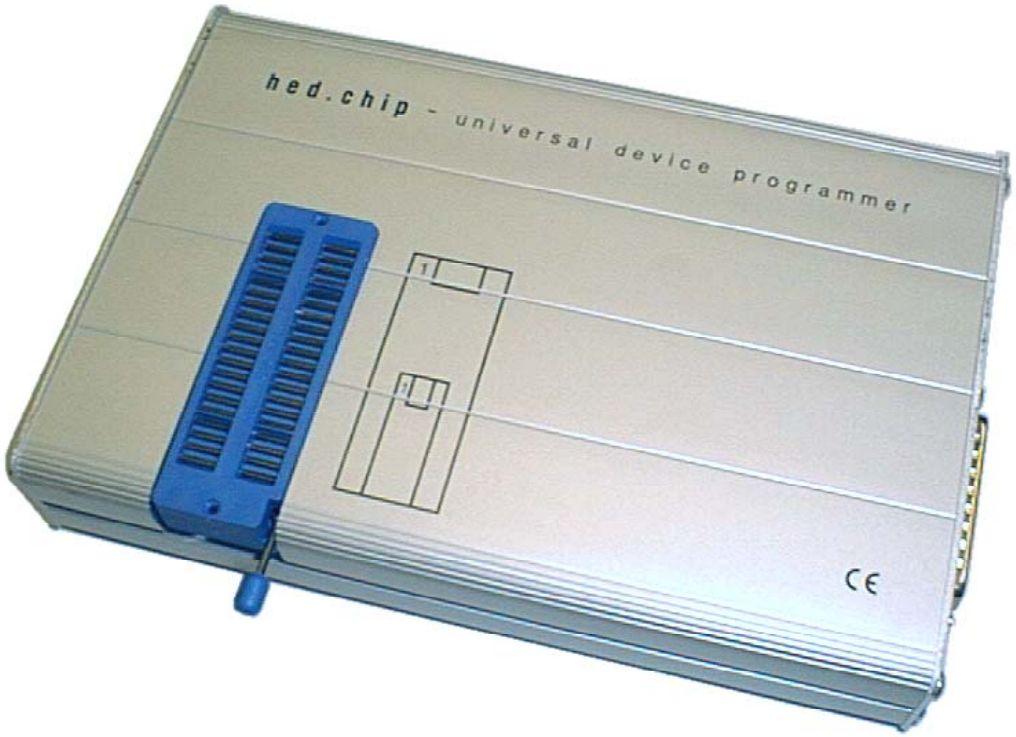

hed.chip - universal device programmer
CE

1 Allgemeine Hinweise zur Bedienung

1.1 Wichtige Hinweise

Dieser Programmer ist ausschließlich für die in der Bausteinliste erwähnten Bausteine geeignet. Andere Typen werden von der Software zurückgewiesen. Nicht geeignete Bausteine können jedoch bereits bei der Initialisierung beschädigt werden.

Bausteine müssen entsprechend dem Piktogramm neben dem Testsockel richtig orientiert eingesetzt werden. Kleine Bausteine müssen möglichst nahe beim Arretierungshebel eingesetzt werden.

Die Schnittstelle, an die der Programmer angeschlossen wird, muß absolut IBM-kompatibel sein.

hed.chip und diese Anleitung sind für Benutzer mit elektrotechnischen Grundkenntnissen gedacht. Die grundsätzliche Handhabung von elektrischen Geräten und elektronischen Bausteinen wird als bekannt vorausgesetzt. In der Elektrotechnik übliche Sicherheitsmaßnahmen müssen beachtet werden.

Die Hard- und Software des Benutzers ist hier nicht bekannt. Für Schäden an kundeneigener Ware kann nicht gehaftet werden.

1.2 Einleitung

hed.chip ist ein universales Programmiersystem. Die Auswahl der programmierbaren Bausteine orientiert sich an den Erfordernissen des praktisch arbeitenden Entwicklers und wird laufend erweitert. Die Programmieralgorithmen sind in der auf dem PC ablaufenden Software enthalten und können bei Bedarf durch ein einfaches Update auf den neusten Stand gebracht werden. Die Bedienung kann wahlweise über die DOS-Kommandozeile oder über eine Windows-Oberfläche mit leistungsfähiger Baustein-Datenbank erfolgen.

Durch handelsübliche, preiswerte Adapter kann die Unterstützung auf SMD-Bausteine in den Gehäusen PLCC und SOIC ausgedehnt werden. Spezialadapter ermöglichen die Programmierung weiterer Bausteinfamilien.

Diese Anleitung besteht aus mehreren Teilen. In diesem Kapitel 1 wird der Programmer und seine grundsätzliche Bedienung beschrieben. Kapitel 2 geht auf die speziellen Eigenschaften der verschiedenen Bausteine ein. Eine Liesmich-Datei gibt aktuelle Informationen zum jeweiligen Stand der Software. Da die graphische Oberfläche kaum einer Erklärung bedarf, konzentriert sich diese Anleitung auf die Bedienung des DOS-Programms HEDCHIP.EXE. Diese Anleitung steht auch als Windows Hilfe-Datei zur Verfügung. Über ein Inhaltsverzeichnis, einen Index und die Suche nach Begriffen im Text kann direkt auf jede beliebige Information zugegriffen werden.

1.3 hed.chip Hardware-Versionen

Von **hed.chip** existieren zwei leicht unterschiedliche Hardware-Versionen. Seit dem 01.11.99 wird die Version 2 vertrieben. Alle älteren Geräte sind Hardware-Version 1.

Die Software unterstützt beide Versionen. Die Benutzeroberfläche HC95 bietet bei der Bausteinauswahl automatisch die vom jeweiligen Gerät programmierbaren Bausteine zur Auswahl an.

In dieser Anleitung wird **hed.chip** in der Hardware-Version 2 beschrieben. In der Windows Hilfe gibt es eine Seite mit speziellen Hinweisen für Benutzer der Hardware-Version 1.

Das Programm HEDCHIP.EXE meldet die Hardware-Version des Programmiergeräts. Siehe: Kaptil „Anschluß und Installation der Software“.

1.4 Hard- und Softwarevoraussetzungen

Kommandozeilen-Betrieb

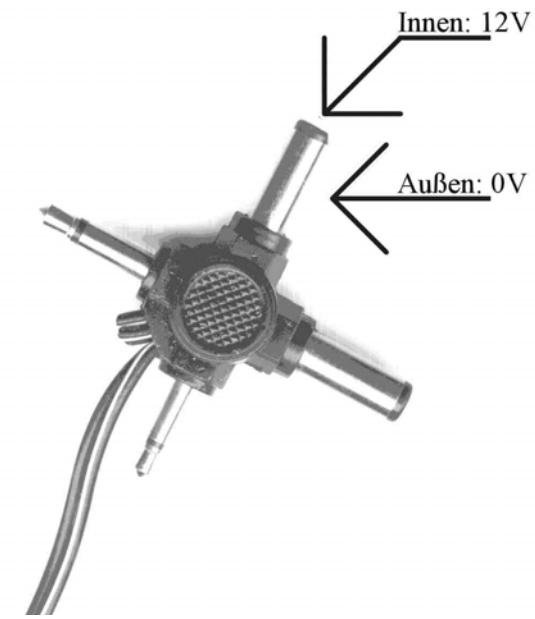
IBM kompatibler PC, 80386 oder höher empfohlen, DOS5.0 bzw. DOS-Task von Windows 95 oder Windows NT 4.0 (P166MMX notwendig), vollständig IBM kompatible Drucker-Schnittstelle

graphische Benutzer-Oberfläche HC95

IBM kompatibler PC, Pentium P100 mit 32 MB RAM, Windows 95 oder Windows NT 4.0 (P166MMX notwendig), vollständig IBM kompatible Drucker-Schnittstelle.

1.5 Anschluß und Installation der Software

hed.chip wird an eine parallele (Drucker-) Schnittstelle des Rechners angeschlossen. Die Schnittstelle muß im BIOS des Rechners ordnungsgemäß installiert sein. Es werden die Schnittstellen LPT1 bis 4 unterstützt. Bei modernen, bidirektionalen Schnittstellen sollte per Jumper oder BIOS-Setup der Standardmodus eingestellt werden. Es wird ein 25poliges, 1:1-Datenkabel mit Stecker/Buchse SubminD benötigt.



Zur Stromversorgung wird ein ungeregeltes Steckernetzteil 12V / 800mA benötigt, das bei Strömen zwischen 50mA und 500mA eine Ausgangsspannung zwischen 12V und 15V aufweist. **hed.chip** hat für den Anschluß des Steckernetzteils eine Niedervolt-Koaxialbuchse für Stecker mit einem Innendurchmesser von 1.95mm und 2.1mm. Übliche Steckernetzteile haben einen solchen Stecker in der Form eines Stecker-Kreuzes. Siehe Bild links.

Bei diesen Netzteilen kann die Polarität eingestellt werden. **hed.chip** nimmt in keinem Fall Schaden, funktioniert jedoch nur bei richtig eingestellter Polarität. Die Polarität ist richtig eingestellt, wenn die Außenoberfläche 0V und das innere eine dazu positive Spannung von mehr als 12V aufweist. Die Spannung des Netzteils muß auf 12V eingestellt sein.

Ein geeignetes Steckernetzteil und ein entsprechendes Datenkabel können von uns bezogen werden.

Die DOS-Software kann auf die Festplatte in ein beliebiges Verzeichnis kopiert werden.

Die Windows Software wird durch Aufruf von SETUP installiert. Unter Windows NT sind dazu Administratorrechte erforderlich.

Internationales Steckernetzteil

Normalerweise wird **hed.chip** mit einem Steckernetzteil für Deutschland, Schweiz und Österreich geliefert. Es hat einen Stecker passend für diese Länder und benötigt eine Spannung von 220 bis 240VAC, 50 Hz.

Für die USA und Großbritannien liefern wir ein Internationales Steckernetzteil. Diese Netzteil hat Adapter für die Netzsteckdosen von den USA, Großbritannien und Deutschland. Es arbeitet mit einem Spannungsbereich von 100 bis 240VAC, 50 oder 60 Hz.

Plug and Prog

hed.chip unterstützt den Anwender beim Anschluß des Netzteils. Dazu muß nur das Programm HEDCHIP.EXE ohne irgendwelche Parameter gestartet werden. HEDCHIP.EXE kann feststellen, ob und an welcher Schnittstelle der Programmer angeschlossen ist. Weiterhin kann HEDCHIP.EXE feststellen, ob das Netzteil richtig angeschlossen ist.

```
hedchip<CR>                                          ; ← Eingabe des Benutzers
hed.chip - universal device programmer, Version 2.62

Test LPT1 - hed.chip gefunden - nicht betriebsbereit

Schliessen Sie die Stromversorgung jetzt an.
Stellen Sie den Polaritaets-Wahlschalter des Netzteils ein.
Stellen Sie den Spannungs-Wahlschalter des Netzteils auf 12V ein.

Wenn die Stromversorgung angeschlossen ist, geht es automatisch weiter
Abbrechen : Escape
```

Das Netzteil kann nun angeschlossen werden. Die Software wiederholt den Test, ob die Stromversorgung angeschlossen ist, ständig. Sobald alle Anschlüsse und Einstellungen richtig sind, wird folgende Meldung ausgegeben:

```
Test LPT1 - hed.chip gefunden - betriebsbereit
Test wiederholen (J/N)
Hardware-Version: 0002
```

Es wird die Hardware-Version des Programmiergeräts (hier: 2) angezeigt.

Unter Windows NT wird **hed.chip** meist an LPT2 gefunden, auch wenn der Rechner nur eine LPT-Schnittstelle hat. Das ist normal und kein Grund zur Sorge.

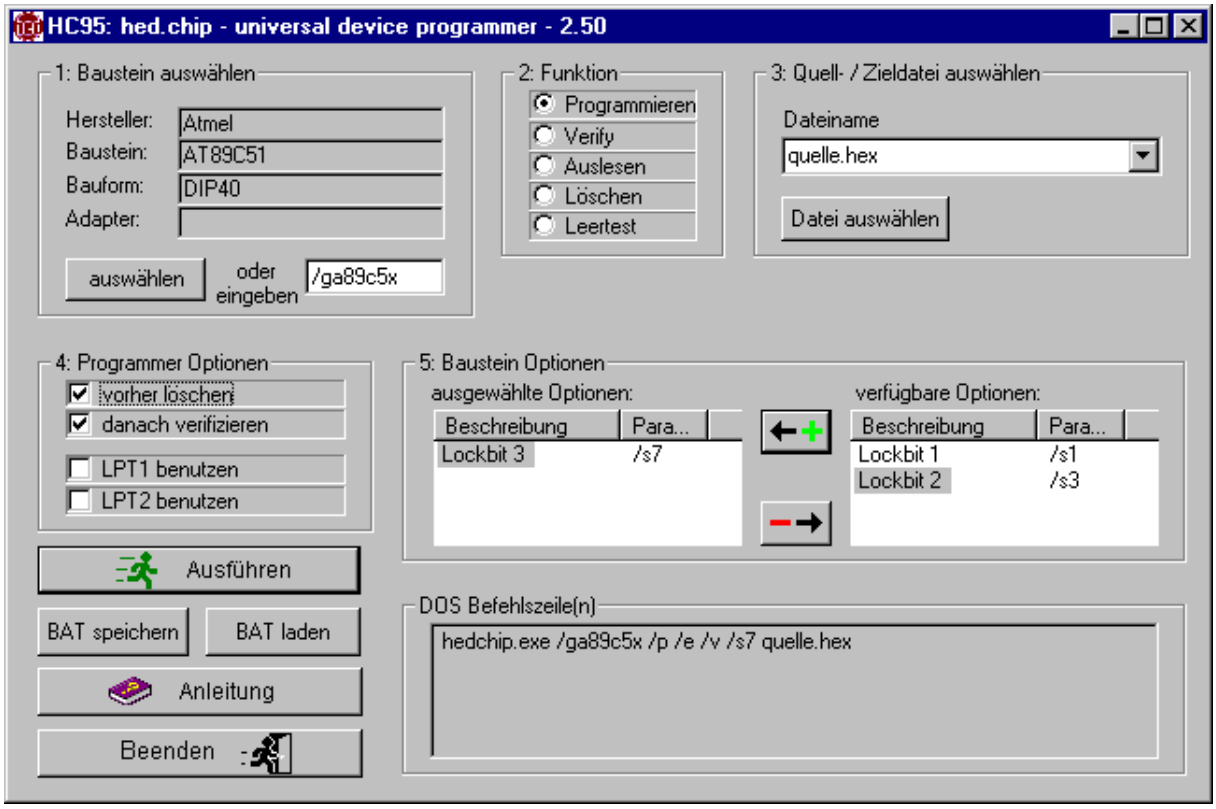
1.6 Bedienung

Letztlich führt immer das DOS-Programm HEDCHIP.EXE die Programmierung durch. Es setzt die Befehle des Anwenders und die Quelldatei in Befehle für den Programmer um. Trotzdem braucht niemand, der es nicht will, sich mit den Feinheiten der DOS-Kommandozeile zu beschäftigen.

hed.chip kann auf drei verschiedene Arten bedient werden:

1. Mit dem Windows Programm HC95. Eine grafische Benutzeroberfläche mit angeschlossener Datenbank gestattet die Auswahl von Bausteinen, Dateien und Parametern. Aus den Eingaben des Benutzers erzeugt HC95 eine DOS-Kommandozeile, die dann voll automatisch ausgeführt wird. Es stehen eine deutschsprachige Hilfedatei und deutsche Erklärungen zu Fehlern des Programmiergeräts zur Verfügung. Abhängig von der eingestellten Sprache des Rechners, wird automatisch die englische oder deutsche Version der Meldungen angezeigt.
2. Die mit HC95 erzeugte DOS-Kommadozeile kann auch in eine Batchdatei gespeichert werden. Dies bietet sich an, wenn **hed.chip** an einem anderen Rechner ohne Windows angeschlossen ist. Mit HC95 erzeugte Batchdateien können auch wieder eingelesen werden. Auf diese Weise können einmal erarbeitete Programmer-Aufrufe jederzeit exakt wiederholt werden.
3. Direkt mit HEDCHIP.EXE an der DOS- oder Windows NT-Eingabeaufforderung.

Windows: HC95



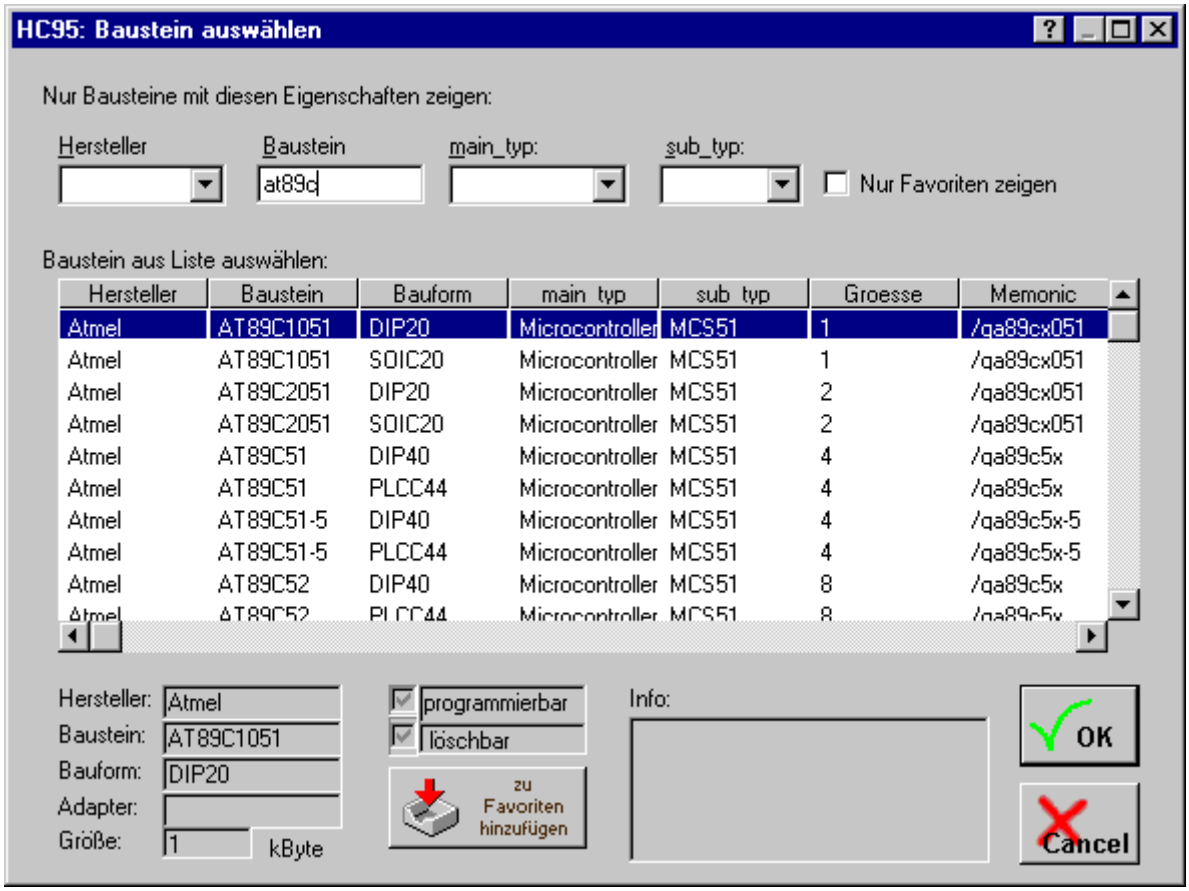
Das Hauptfenster von HC95

Das obige Bild zeigt die englische Variante des Fensters. Die Bedienelemente sind in Gruppen in der Reihenfolge angeordnet, in der man sinnvollerweise die Eingaben vornimmt:

1. Auswahl des Bausteins. Durch Click auf den Button "auswählen" gelangt man zu einem Bildschirm, der die Liste der verfügbaren Bausteine anzeigt. Mehr dazu unten. Benutzer, die mit der Kommandozeile vertraut sind, können das Bausteinmnemonic auch direkt im Editierfeld eingeben. Das setzt allerdings jegliche Überprüfung der Eingaben durch HC95 außer Kraft.
2. Auswahl der Funktion. Hier wird bestimmt, ob der Baustein programmiert, mit einer Datei verglichen, ausgelesen, gelöscht oder auf leer getestet werden soll.
3. Auswahl der Quell- oder Zieldatei: Hier wird ausgewählt, welche Datei in den Baustein programmiert werden soll, mit welcher Datei der Baustein verglichen werden soll oder in welche Datei der Baustein ausgelesen werden soll.
4. Programmer-Optionen:
   4.1. vorher löschen: HEDCHIP.EXE führt vor der Programmierung automatisch immer einen Leertest aus. Wenn diese Option aktiviert ist, wird ein nicht-leerer Baustein automatisch gelöscht.
   4.2. danach verifizieren: Diese Option veranlaßt den Programmer, nach der Programmierung automatisch ein Verify (Vergleich mit der Quelldatei) auszuführen.
   4.3. LPT1 benutzen, LPT2 benutzen: Mit diesen Optionen kann die automatische Druckerporterkennung (Plug and Prog) umgangen werden. Das kann sinnvoll sein, wenn dadurch Konflikte entstehen oder mehr als ein Programmiergerät **hed.chip** angeschlossen ist.
   4.4. Zusätzliche Optionen können über das Menü „Extras\Zusätzliche Optionen“ ausgewählt werden. Diese Optionen beeinflussen, wenn Leertests ausgeführt werden.

5. Baustein-Optionen: Manche Bausteine verfügen über zusätzliche Features, wie z.B. Schreib- und/oder Leseschutz. In der Listbox werden die jeweils verfügbaren Optionen zur Auswahl angeboten. Wer mit der Kommandozeile von HEDCHIP.EXE vertraut ist, kann den Optionsparameter auch in der Textbox eingeben.

Bausteine auswählen:



Hinter HC95 steht eine Datenbank mit allen programmierbaren Bausteinen und deren Eigenschaften. Der zu programmierende Baustein muß aus der großen Liste in der Fenstermitte ausgewählt werden. In den Feldern unter der Liste werden die Eigenschaften des gewählten Bausteins angezeigt. Die Liste ist recht lang. Von "A" wie AMD bis zu "X" wie Xicor in der Liste zu wandern, wäre ziemlich mühselig. Deshalb kann mit den vier Feldern über der Liste die Auswahl eingeschränkt werden. Es können Hersteller, Bausteinbezeichnung und die Einteilung der Bausteine in Haupt- und Untergruppen (main_typ, sub_typ) als Kriterium benutzt werden. Jede beliebige Kombination und die unbegrenzte Verwendung von Wildcards '?' und '*' sind möglich. Zum Beispiel: "*28F*" im Feld Baustein führt zur Anzeige aller FLASH-Bausteine, die in ihrer Bezeichnung an beliebiger Stelle die Buchstaben "28F" enthalten. Das '?' ersetzt dabei genau ein Zeichen, das '*' ersetzt 0 bis beliebig viele Zeichen in der Zeichenkette.

Favoriten: häufig benötigte Bausteintypen

Für Bausteine, die immer wieder mit **hed.chip** bearbeitet werden sollen, gibt es eine bessere Auswahlmöglichkeit als die Suche in der Datenbank. Man kann diese Bausteine zu „Favoriten“ erklären.

Je nach dem, ob ein Baustein bereits Favorit ist, bietet der Button die Möglichkeit, den Baustein zu den Favoriten hinzuzufügen oder ihn aus den Favoriten wieder zu entfernen.

Wenn man dann die Anzeige auf „nur Favoriten zeigen“ umstellt, bekommt man seine ganz persönliche Auswahl an Bausteinen angezeigt:

Mit HC95 programmieren

Jetzt sind alle notwendigen Eingaben gemacht. In der Box "DOS Befehlszeile(n)" unten rechts, zeigt HC95 die erzeugte Kommandozeile an. Bei Bausteinen, die besondere Behandlung vor der Programmierung erfordern, können es auch mehrere Zeilen sein.

Mit "Ausführen" kann die erzeugte Kommandozeile ausgeführt werden. HC95 öffnet ein DOS-Fenster und führt darin die erzeugten Kommandozeilen aus. Nach Beendigung wird das Fenster automatisch geschlossen und eine Erfolgsmeldung oder eine Fehlermeldung angezeigt.

Mit "BAT speichern" kann die Kommandozeile als Batch-Datei abgespeichert werden.

Windows: Mit HC95 Batch-Dateien arbeiten

Die mit "BAT speichern" erzeugte Batch-Datei kann z.B. auf einem anderen Rechner benutzt werden. Dieser Weg bietet sich an, wenn der Rechner nicht über Windows verfügt. Die Batch-Datei kann von HC95 mit "BAT laden" auch wieder eingelesen werden. Auf diese Weise können Programmierungen immer wieder auf exakt die gleiche Weise wiederholt werden.

Windows: HC95 Default-Einstellungen

Default-Einstellungen dienen der Arbeitserleichterung. Wer z.B. **hed.chip** an LPT2 angeschlossen hat, möchte vielleicht die Einstellung der Schnittstelle als Default-Einstellung beim Start von HC95 automatisch laden.

Beim Starten von HC95 kann der Name einer von HC95 erzeugten Batch-Datei angegeben werden. Wenn kein Dateiname angegeben wird, werden die Einstellungen aus der Datei DEFAULT.BAT geladen.

Einstellungen als Default abspeichern:

Wenn die aktuellen Einstellungen automatisch beim nächsten Start von HC95 wiederhergestellt werden sollen, speichern Sie diese mit der Funktion „BAT speichern“ unter dem Namen DEFAULT.BAT im Verzeichnis von HC95 ab.

Sinnlose oder fehlerhafte Default-Einstellungen beseitigen:

Löschen Sie im Verzeichnis von HC95 die Datei DEFAULT.BAT. Beim nächsten Start von HC95 wird automatisch eine neue Datei DEFAULT.BAT erzeugt.

Verschiedene Einstellungen verwalten:

Wenn Sie bestimmte Programmieraufgaben immer wieder ausführen müssen, können Sie die dafür notwendigen Einstellungen in einigen Batch-Dateien abspeichern. Auf dem Desktop oder im Startmenü können Sie Verknüpfungen auf HC95 unter Angabe einer solchen Batch-Datei erzeugen.

Das geht so: Erstellen Sie auf dem Desktop eine Verknüpfung auf HC95.EXE. Editieren Sie dann die Eigenschaften. Im Feld „Ziel“ fügen Sie hinter „<Pfad>\HC95.EXE“ ein Leerzeichen und dann den Namen Ihrer Batch-Datei ein.

Windows: Prüfsummen

Nach der Programmierung oder nach dem Auslesen zeigt die Software für den Baustein eine Prüfsumme an. Diese Prüfsumme kann benutzt werden, um schnell und einfach den Versionsstand einer Software in einem Baustein zu überprüfen. Dazu müssen Sie lediglich die angezeigte Prüfsumme in Ihre Projektdokumentation aufnehmen.

Das Verfahren zur Berechnung der Prüfsumme ist folgendes:

```
for (int i = 0; i < len; i++)
    sum += (unsigned char)(*(pData +i) + 1);
```

Zu jedem Byte wird zunächst 1 hinzuaddiert. Da es sich um Byte-Werte handelt, wird dadurch aus 255 der Wert 0. Dann werden alle Bytes der Datei zu einer 32-Bit Summe aufaddiert. Weil sich der Wert 255 neutral zur Prüfsumme verhält, hat ein leerer Baustein immer die Prüfsumme 0. Bei einem teilweise programmierten Baustein ist die Prüfsumme der Quelldatei und die Prüfsumme des ausgelesenen Bausteins identisch.

Dieses Verfahren zur Berechnung von Prüfsumme versagt bei Bausteinen mit zusätzlichen Konfigurationsinformationen. In diese Kategorie von Bausteinen fallen z.B. Microchip PIC Microcontroller und Lattice GAL. Es ist bei diesen Bausteinen normal, dass sich die Prüfsummen einer Quelldatei und des ausgelesenen Bausteins unterscheiden.

Beim Hexeditor **hed.HexEd** kann man einstellen, nach welchem Verfahren Prüfsummen berechnet werden sollen. Damit Sie dort die gleichen Prüfsummen angezeigt bekommen, müssen Sie in die Checkbox „hed.chip kompatible Prüfsumme“ ein Häkchen machen.

DOS: HEDCHIP.EXE

Wer ohnehin mit DOS arbeitet oder den Programmeraufruf in eigene Entwicklungsumgebungen einbinden möchte, kann direkt mit HEDCHIP.EXE arbeiten. Für den Einbau des Aufrufs in eigene Entwicklungsumgebungen sind die DOS-Returncodes in Kapitel 3 aufgelistet.

Die Bedienung von HEDCHIP.EXE erfolgt ausschließlich über Parameter in der Kommandozeile. Alle Parameter beginnen mit einem Schrägstrich ‘/’ (Shift-7). Nur der Dateiname der Quelldatei wird ohne ‘/’ als letzter Parameter aufgeführt. Alle anderen Parameter können in beliebiger Reihenfolge aufgeführt werden. HEDCHIP.EXE interpretiert den ersten Parameter ohne Schrägstrich ‘/’ als Dateinamen und beendet die Bearbeitung der Kommandozeile.

Die Kommandozeile muß immer folgende Elemente enthalten:

Bausteinmnemonic	/gMNEMONIC; z.B. /ga16v8, /gi87c5x, etc.	
Befehlsparameter	/?	Kommandoübersicht ausgeben
	/b	Bausteinliste ausgeben
	/d	Direkt Modus (unterdrückt Tastaturabfragen)
	/e	Baustein löschen
	/l	Leertest
	/p	Baustein programmieren
	/r	Baustein in Datei auslesen
	/v	Baustein mit Datei vergleichen

Zu den Befehlsparametern /p (Programmieren), /r (Auslesen) und /v (Vergleichen) ist die Angabe des Dateinamens der Quell- bzw. im Falle von /r der Zieldatei erforderlich.

HEDCHIP.EXE verarbeitet als Quelldatei folgende Formate: JEDEC (*.JED), Intel Hex (*.HEX), Motorola S-Record (*.MOT) und Binärdateien (alle anderen).

Geeignete JEDEC-Dateien können mit CUPL, Gal Development System GDS3.5 oder easyABEL erzeugt werden. Die in diesen Dateien enthaltenen Prüfsummen werden nicht beachtet. Es muß immer der vollständige Dateiname inklusive Extension, zum Beispiel .JED, .HEX oder .BIN, angegeben werden. Dateien mit den Extensions .JED, .HEX und .MOT werden automatisch in ein binäres Datenformat umgewandelt. Alle andere Extensions werden als Binärdaten interpretiert und direkt ohne weitere Umwandlung in den Baustein programmiert.

Zu dem Befehlsparameter /p (Programmieren) können folgende, optionale Parameter zusätzlich angegeben werden:

/e	Baustein automatisch löschen, wenn nicht leer.
/sn	Security-Bits programmieren. Für 'n' muß die Nr. des zu programmierenden Bits eingesetzt werden. Beispiele: /s1, /s2 oder /s3
/v	Programmierung bzw. Löschvorgang verifizieren
/n	Kein Leertest vor dem Programmieren
/m	In Verbindung mit /p/e/v: Wenn gelöscht wird kein Leertest nach dem Löschen.

HEDCHIP.EXE findet die Schnittstelle, an der das Programmiergerät angeschlossen ist, automatisch. Diese Automatik kann durch explizite Angabe der Schnittstelle übergangen werden:

/lpt1	hed.chip an LPT1
/lpt2	hed.chip an LPT2

Mit dem Befehlsparameter /r wird der Baustein in eine Datei, die Zieldatei, ausgelesen. Einfache PLDs, 16V8, 20V8, 18V10, 22V10 und 20RA10, werden in JEDEC-Dateien ausgelesen. Damit eine solche Datei wieder in einen anderen Baustein programmiert werden kann, *muß* ein Dateiname mit der Extension .JED angegeben werden. Alle anderen Bausteine, komplexe PLDs, Microcontroller und Speicher werden in Binärdateien ausgelesen. Dabei darf *kein* Dateiname mit den Extensions .HEX , .JED oder .MOT angegeben werden.

Falls bereits eine Datei mit dem als Zieldatei angegebenen Dateinamen existiert, wird diese ohne Warnung überschrieben.

Bei den Befehlen /p (Programmieren) und /v (Vergleichen) verwendet **hed.chip** immer das Minimum von Bausteingröße und Quelldatei. Es gibt keine Fehlermeldung, wenn die Quelldatei für den verwendeten Baustein zu groß ist.

Der Direkt Modus:wird mit dem Parameter /d aktiviert. Dieser Parameter ist für die Verwendung in Batch-Dateien gedacht. Im Direkt Modus werden vom Benutzer keinerlei Tastatureingaben abgefragt. Die Software übergeht Abfragen „weiter mit Taste“ ohne anzuhalten. Abfragen, bei denen die Auswahl „Ja / Nein / Abbruch“ geboten wird, werden automatisch mit „Nein“ beantwortet. Bei der Erstellung von Batch-Dateien sollte man das Batch zunächst ohne den Parameter /d testen. Wenn alle Möglichkeiten (mit und ohne eingesetzten Baustein, leerer oder programmierter Baustein) getestet sind, kann man störende Tastaturabfragen mit dem Parameter /d unterdrücken.

1.7 Hinweise zum Betrieb unter Windows NT4.0

Wie bei Windows 95 kann die Programmierung über die Windows Benutzeroberfläche oder von der Eingabeaufforderung gestartet werden. HEDCHIP.EXE erkennt automatisch das Betriebssystem. Für Funktionen, die den direkten Hardwarezugriff erfordern, werden automatisch Treiber geladen und nach Beendigung des Programms wieder entladen.

Diese Treiber sind speziell für Windows NT entwickelt worden. **hed.chip** verhält sich unter Windows NT voll systemkonform. Details, welche Datei sich wo befinden muß und welche Einträge bei der Installation in die Registry geschrieben werden, sind in der Windows Hilfe zu HC95 enthalten.

Angabe der Schnittstelle:

Unter Windows NT sollten Sie für das Programmiergerät keine Schnittstelle angeben. Sie werden feststellen, daß die Software das Programmiergerät bei den meisten Rechnern an LPT2 finden wird, auch wenn Ihr Rechner nur eine LPT-Schnittstelle hat. Windows NT gibt DOS-Programmen den Anschein, als hätte der Rechner 3 oder 4 LPT-Schnittstellen.

1.8 Programmieren von Schreib- und/oder Leseschutz

Leseschutz bietet bei Microcontrollern und PLD-Bausteinen Schutz vor nicht autorisierten Kopien der Software. Bei MCS51-Microcontrollern wird dieser mehrstufige Schutz „Lockbits“ genannt. PLD-Bausteine haben zu diesem Zweck eine Securityfuse.

Viele elektrisch löschbare Speicherbausteine (FLASH, EEPROM) haben einen Schreibschutz. Damit kann verhindert werden, daß ein abgestürzter Prozessor unbeabsichtigt den Inhalt des Speicherbausteins verändert. Je nach Typ können Teile des Bausteins oder der gesamte Baustein geschützt werden. Einige Formen des Schreibschutzes sind irreversibel, andere können wieder aufgehoben bzw. deaktiviert werden.

Die Schutzmöglichkeiten sind bei verschiedenen Bausteinen sehr unterschiedlich. Beschreibungen finden sich bei den jeweiligen Bausteinen.

Grundsätzlich wird für die Programmierung des Schreib-/Leseschutzes der Parameter /s verwendet. Dieser Parameter kann bei der Programmierung mit angegeben werden. Dabei wird der gewünschte Schutzlevel als Zahl mit angegeben. Einige Beispiele:

```
hedchip /gl22v10 /p /v /e /s1 myapp.jed      ; Lattice GAL22V10 löschen, programmieren,
                                             ; verifizieren und schützen.
hedchip /ga89c5x /p /v /e /s7 myapp.hex      ; Atmel Controller löschen, programmieren,
                                             ; verifizieren und alle 3 Lockbits  programmieren.
```

Lesegeschützte Bausteine können natürlich auch vom **hed.chip** weder ausgelesen noch kopiert werden. Je nach Typ passiert folgendes:

- Der Baustein wird erkannt und auch der aktivierte Schutz wird festgestellt. **hed.chip** erzeugt dann entsprechende Meldungen.
- Der Baustein wird erkannt und ist scheinbar leer. Vor der Programmierung müssen solche Bausteine durch einen separaten Aufruf von HEDCHIP.EXE nur mit dem Parameter /e gelöscht werden.
- Der Baustein kann nicht identifiziert werden. Bei solchen Bausteinen fragt HEDCHIP.EXE den Benutzer, ob trotzdem fortgefahren werden soll. Dies gilt bisher nur für Atmel AT89C5x Controller.

Einige Speicherbausteine verfügen über mehrere Schreibschutzmöglichkeiten. Bei einem Teil dieser Bausteine bauen die Schreibschutzstufen aufeinander auf, bei anderen Typen können sie in beliebiger Kombination aktiviert werden. Welche Schutzmöglichkeiten zur Verfügung stehen und wie diese aktiviert werden, ist bei den jeweiligen Bausteinen beschrieben.

Schreibgeschützte Speicherbausteine können von **hed.chip** ausgelesen und kopiert werden. Um einen schreibgeschützten Baustein neu zu programmieren, muß der Baustein vorher gelöscht werden. Das gilt auch, wenn nur ein Teil des Bausteins geschützt ist.

Der Parameter /s wird auch benutzt, um andere, besondere Eigenschaften bestimmter Bausteine zu programmieren. Bei Atmel Serie AT17C FPGA-Configuration Memories wird damit die ‘Polarity Option’ eingestellt.

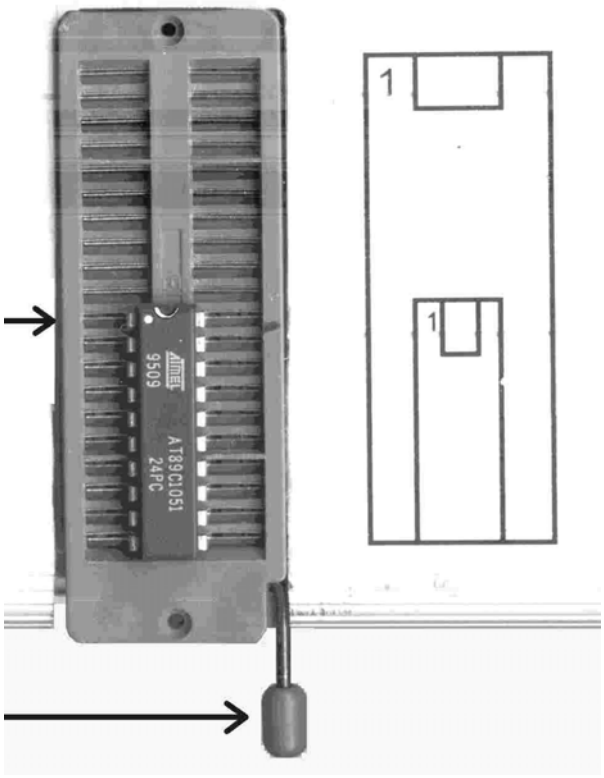
1.9 Einsetzen des Bausteins

Der zu programmierende Baustein kann jederzeit eingesetzt werden. Die Markierungskerbe des Bausteins muß in entgegengesetzte Richtung vom Arretierungshebel zeigen. Der Baustein muß bündig am Arretierungshebel eingesetzt werden. Die Kontakte am Arretierungshebel dürfen auch bei kleinen Bausteinen nie frei bleiben.

Pin 1

Foto:
Testsockel des Programmers mit eingesetztem Baustein Atmel AT89C1051-24PC

Arretierungshebel



1.10 Adapter

Für Bausteine in SMD-Gehäusen, zum Beispiel PLCC oder SOIC, werden Adapter benötigt. Im Handel sind Adapter für Microcontroller in der Bauform PLCC44 oder Speicherbausteine in der Bauform PLCC32 erhältlich. Grundsätzlich sind Adapter zu wählen, die die Signale vom DIP40 Testsockel an die entsprechenden Anschlüsse des PLCC-Bausteins übertragen. Im Falle der parallelen Speicherbausteine (EPROM, EEPROM und FLASH) wurde die Software so angepaßt, daß alle diese Bausteine mit einem Adapter DIP32 auf PLCC32 programmiert werden können. Bei der Programmierung muß dazu das der Bauform entsprechende Bausteinmnemonic aus der Bausteinliste verwendet werden. Solche allgemeinen Adapter können gegebenenfalls auch selbst hergestellt werden. Diese Adapter werden von uns nach der Bauform benannt, etwa Adapter PLCC32 oder PLCC44. Für die beiden meist verwendeten PLCC-Typen, Speicherbausteine in PLCC32 und MCS51-Microcontroller in PLCC44, gibt es einen Kombi-Adapter mit zwei Testsockeln. Da es sich in jedem Fall um die gleiche Platine handelt, kann der zweite Testsockel gegebenenfalls auch nachträglich selbst bestückt werden.

Nicht selbst hergestellt werden können Adapter, die das Programmiergerät an spezielle Erfordernisse bestimmter Bausteine anpassen. Für diese Adapter ergibt sich die Bezeichnung aus der Bauform und der Typbezeichung. Ein Atmel ATV750 in der Bauform DIP24 wird daher im Adapter DIP750 programmiert, für die PLCC-Version wird der Adapter PLCC750 verwendet.

Einsetzen der Adapter in das Programmiergerät

Auf den meisten Adaptern ist ein Aufdruck angebracht, der zeigt, wie der Adapter in den Testsockel des Programmiergeräts eingesetzt werden muss. Grundsätzlich gilt für alle Adapter mit DIP Testsockel gilt folgende Regel beim Einsetzen: Der Arretierungshebel des Testsockels des Adapters muss in die gleiche Richtung zeigen wie der des Testsockels des Programmiergeräts.

1.11 Selbstbau von Adaptern

Adapter für die PLCC-Versionen von Bausteinen, zu denen es auch entsprechende Versionen in der Bauform DIP gibt, können selbst hergestellt werden. Eine Funktionsgarantie für solche Adapter oder von anderen Firmen bezogene Adapter gibt es nicht.

1.12 Typbestimmung und Angabe des richtigen Bausteinmnemonics

Begriffsbestimmung: Im Englischen wird der Wort „Mnemonic“ in der Bedeutung „einprägsamer Platzhalter“ benutzt. Fremdwörter-Duden: *Mnemotechnik: Kunst, das Einprägen von Gedächtnisstoff durch besondere Lernhilfen zu erleichtern.* In der **hed.chip** Software werden Bausteine intern durch einen Zahlenkode identifiziert. Ein „Bausteinmnemonic“ ist ein Platzhalter, der genau für einen bestimmten Baustein steht.

hed.chip bemüht sich, den Baustein im Programmiersockel zu identifizieren. Er wird die Programmierung eines Bausteins, der nicht mit dem in der Kommandozeile angegebenen Mnemonic übereinstimmt, ablehnen. Dies dient dem Schutz der wertvollen Bauteile. Der Schutz ist allerdings nicht absolut, da bereits für den Test die Versorgungsspannung und je nach Mnemonic in einigen Fällen auch die Programmierspannung angelegt werden muß. Es ist also sehr wichtig, daß das richtige Mnemonic angegeben wird. Wenn Bausteine trotz Angabe des richtigen Mnemonic nicht erkannt oder nicht programmiert werden, kann es sich um einen geschützten Baustein handeln. Die meisten geschützten Bausteine sind entweder nicht mehr identifizierbar oder scheinbar leer. Details zu dem Verhalten bestimmter Bausteine sind im zweiten Kapitel näher beschrieben.

hed.chip gibt eine Liste der verfügbaren Mnemonics aus, wenn das Programm nur mit dem Schnittstellenparameter aufgerufen wird. Beispiel:

```
hedchip /b                                  ; gibt Liste der Mnemonics aus
```

2 Details der Bausteine

2.1 Programmierbare Logik - von PLDs und GALs

Der Begriff PLD (= Programmable Logic Device) ist ein sehr weit gefaßter Begriff. Er umfaßt Bausteine von einfachen TTL-PROMs bis hin zu Gate Arrays, in die ganze Prozessoren programmiert werden können.

Der Begriff GAL ist ein geschütztes Warenzeichen der Firma Lattice. Es handelt sich um einfache PLDs, teilweise auch als SPLD abgekürzt. Bausteine mit mehr Funktionalität werden als komplexe PLDs, abgekürzt CPLD, bezeichnet.

hed.chip programmiert eine Auswahl beliebter und leistungsfähiger Bausteine dieser Logikfamilien. Hier eine Auswahl der bei Drucklegung dieser Anleitung programmierbaren Bausteine:

ATF16V8	ATF20V8	ATF22V10	
GAL16V8	GAL20V8	GAL18V10	GAL22V10
GAL6001	GAL6001B	GAL6002B	GAL20RA10
PALCE16V8	PALCE20V8	PALCE22V10	

Die komplexen PLDs von Atmel (ATV750, ATV2500) stellen besondere Anforderungen an die Programmierhardware. Für diese Typen sind die Adapter DIP750, PLCC750, DIP2500, PLCC2500 bzw. PLCC1500 erforderlich.

Um eine Anwendung für einen PLD zu entwickeln, muß zunächst eine JEDEC-Datei erstellt werden. Das PLD-Entwicklungssystem, z.B. CUPL, Gal Development System GDS 3.5 oder easyABEL Version 4.3, setzt die logischen Gleichungen in eine solche JEDEC-Datei um. Es kann außerdem das zu erwartende Verhalten des PLD simulieren.

Prüfsummen und Testvektoren in der JEDEC-Datei werden ignoriert. Dies gestattet es, die JEDEC-Datei bei Bedarf mit einem normalen Texteditor zu verändern.

Wenn die logischen Gleichungen in der Datei MYAPP.PLD stehen, erzeugt CUPL daraus die Datei MYAPP.JED. Diese kann dann mit **hed.chip** in den Baustein programmiert werden. Im Falle einer Anwendung für einen ATF22V10 müßte dazu folgende Befehlszeile verwendet werden:

```
hedchip /ga22v10 /p myapp.jed
```

Die zusätzliche Angabe des Parameters /v veranlaßt den Programmer, die Programmierung anschließend zu verifizieren. Bei Angabe von /e wird der Baustein gelöscht, wenn der vor der Programmierung automatisch durchgeführte Leertest ergibt, daß dies nötig ist.

```
hedchip /ga22v10 /p /v /e myapp.jed
```

Securityfuse

Wenn der Baustein gegen Auslesen und Kopieren geschützt werden soll, kann die Securityfuse programmiert werden. Bei dem Entwicklungssystem CUPL kann die Anweisung dazu bereits bei der Erstellung der JEDEC-Datei gegeben werden.

CUPL fügt dann eine Anweisung *G1 in die JEDEC-Datei ein und veranlaßt dadurch **hed.chip**, die Securityfuse des Bausteins zu programmieren. Falls das nicht erwünscht ist, kann entweder diese Anweisung wieder aus der JEDEC-Datei entfernt werden oder in der Kommandozeile zusätzlich der Parameter /s0 angegeben werden. Der Parameter /s1 in der Kommandozeile hingegen veranlaßt **hed.chip,** die Securityfuse auch dann zu programmieren, wenn die JEDEC-Datei die Anweisung

*G0 enthält. Kommandozeilenparameter, ob die Securityfuse programmiert werden soll oder nicht, haben Vorrang vor Anweisungen in der JEDEC-Datei.

hed.chip kann bei einigen PLDs die Securityfuse testen und wird bei dem Versuch, einen solchen Baustein auszulesen, eine Fehlermeldung erzeugen.

Andere geschützte PLDs werden als leer verkannt. Solche Bausteine können auch nicht durch die zusätzliche Angabe von /e bei der Programmierung gelöscht werden. In einem solchen Fall muß der Baustein in einem seperaten Arbeitsschritt gelöscht werden:

```
hedchip /gl22v10 /e                    ; Beispiel für ein GAL22V10
```

Bausteine kopieren

Mit **hed.chip** können PLDs auch ausgelesen und kopiert werden. Beim Auslesen erzeugt **hed.chip** bei den einfachen PLDs 16V8, 20V8, 18V10, 22V10 und 20RA10 eine JEDEC-Datei mit den gleichen Anweisungen wie es auch CUPL macht. Zum Auslesen dieser Bausteine *muß* ein Dateiname mit der Extension .JED angegeben werden. Komplexe PLDs (GAL6001/2, Atmel ATV-Serie) werden in ein binäres Datenformat ausgelesen. Für diese Bausteine *darf kein* Dateiname mit der Extension .JED verwendet werden. **hed.chip** kann diese Binärdateien in andere Bausteine gleichen Typs programmieren. Beispiel:

```
hedchip /gatv750 /r myapp.bin          ;Atmel ATV750 in Datei MYAPP.BIN auslesen
hedchip /gatv750 /p/v myapp.bin        ; anderen Baustein gleichen Typs programmieren
```

ATF16V8, ATF20V8, ATF22V10

Atmel schreibt für diese PLD-Bausteine auf FLASH-Speicherbasis vor, daß sie vor der ersten Programmierung konditioniert werden. Das heißt, der ganze Baustein wird zweimal vollständig mit 0 programmiert und anschließend wieder gelöscht. Während der Konditionierung können Verify-Fehler ignoriert werden. Dem **hed.chip** ist eine JEDEC-Datei beigefügt, die zur Konditionierung aller Atmel-PLD benutzt werden kann. Für einen ATF20V8 geht das so:

```
hedchip /ga20v8 /p/e conditio.jed      ; 1. Mal programmieren
hedchip /ga20v8 /p/e conditio.jed      ; 2. Mal programmieren
hedchip /ga20v8 /e/v                   ; Löschen, Leertest
```

Die Befehle zur Konditionierung können auch in die ohnehin empfohlene Batchdatei aufgenommen werden.

ATV750(B) und ATV2500(B):

Für die PLDs ATV750(B) und ATV2500(B) werden spezielle Adapter DIP750, DIP2500, PLCC750 bzw. PLCC2500 benötigt. Auf diesen Adaptern befinden sich je zwei Jumper. Zur Programmierung des ATV750 und ATV2500 müssen beide Jumper gesteckt und zur Programmierung des ATV750B und ATV2500B beide entfernt werden.

Der Adapter DIP750 wird auch für die Bausteine Atmel AT22V10/L und AT22V10B/L verwendet. Dabei handelt es sich um eine ältere Version des 22V10 auf EPROM-Speicherbasis.

AMD PALCE-Serie

hed.chip unterstützt aus dieser PLD-Serie die Typen PALCE16V8H/Q, PALCE20V8H/Q. Der Typ PALCE22V10H/Q wird in der Revision 4 und 5 unterstützt. Diese werden in Katalogen z.B. als PALCE22V10H-25PC4 angeboten. Alle PALCE-Bausteine, auch neue und leere Bausteine, müssen vor der Programmierung gelöscht werden. Eine Programmierung mit der Datei CONDITIO.JED, wie z.B. für Atmel ATF-PLD vorgesehen, ist nicht erforderlich.

Die Bausteine AMD PALCE16V8 und PALCE20V8 sind vollständig funktionskompatibel zu den entsprechenden Bausteinen von Atmel und Lattice. Zwischen diesen Bausteinen gibt es lediglich

einen kleinen Unterschied bei der Rückführung der Registerausgänge in die UND-Matrix. In den meisten Fällen können für GAL16V8 und GAL20V8 erstellte Quelldateien ohne Änderung in einen PALCE-Baustein programmiert werden.

Um Inkompatibilitäten vollkommen auszuschließen, sollte bei der Erstellung der JEDEC-Datei der richtige Zielbaustein, also entweder GAL16V8, GAL20V8 oder PALCE16V8, PALCE20V8 angegeben werden. Die Unterschiede zwischen diesen Bausteinen wurden in der Ausgabe 1/94, Seite 52ff der Zeitschrift Elektor ausführlich beschrieben.

Der Typ PALCE22V10 kann direkt mit für GAL22V10 entwickelten und compilierten JEDEC-Dateien programmiert werden.

2.2 MCS51 Microcontroller

hed.chip programmiert fast alle CMOS-MCS51-Varianten der Hersteller AMD, Atmel, Dallas, Intel, Philips, Siemens, SST, Temic und Winbond. Für Controller in den Bauformen PLCC44 und SOIC20 sind Adapter verfügbar.

Wenn ein solcher Controller programmiert oder ausgelesen werden soll, muß zunächst in der Befehlszeile das richtige Mnemonic angegeben werden. **hed.chip** vergleicht das Mnemonic mit der Hersteller-ID und der Baustein-ID und wählt dann automatisch den richtigen Programmieralgorithmus.

Wenn eine Anwendung z.B. für einen Philips 87C52 entwickelt werden soll, muß mit Hilfe eines Crossassemblers bzw. Crosscompilers eine Intel-Hex-Datei oder eine Binärdatei erzeugt werden. Diese kann **hed.chip** dann in den Controller programmieren:

```
hedchip /gp87c5x /p/v myapp.hex
```

Es ist notwendig, daß in dem obigen Beispiel das Mnemonic /gp87c5x und nicht etwa /gi87c5x für Intel Controller verwendet wird. Wenn die Hersteller-ID nicht mit dem Mnemonic übereinstimmt, wird **hed.chip** jede Aktion mit dem Baustein ablehnen. Das gleiche gilt für den Fall, daß die Baustein-ID **hed.chip** nicht bekannt ist. Wir werden uns bemühen, für neue Bausteine so schnell wie möglich Updates der **hed.chip**-Software zu erstellen.

Achtung: Es wird keine Fehlermeldung und auch keine Warnung erzeugt, wenn die Quelldatei für den Speicher des Controllers zu groß ist.

Lockbits

MCS51 Controller haben zwei oder drei sogenannte Lockbits, die den Baustein schützen:

Parameter	Lockbits	Funktion
/S1	1	schützt gegen weitere Programmierung
/S3	1 + 2	schützt gegen Auslesen des Programmspeichers. Da nach wie vor externe Programme ablaufen können, ist der Schutz nicht sehr sicher. S3 beinhaltet automatisch S1
/S7	1 + 2 + 3	verhindert, daß externe Programme ablaufen. Der Zustand des PINs EA/ ist bei gesetztem 3. Lockbit ohne Bedeutung. S7 beinhaltet automatisch S3 und S1. Nicht alle Bausteine verfügen über dieses Lockbit.

Mit S2 oder höher geschützte Bausteine werden vom Programmer entweder nicht erkannt, weil auch die Hersteller-ID nicht ausgelesen werden kann, oder sie werden als leer verkannt. Der Versuch, einen solchen, vermeintlich leeren Baustein zu programmieren, führt zu der Fehlermeldung ‘Baustein nicht programmierbar’.

Falls ein Baustein also von **hed.chip** nicht akzeptiert wird oder nicht programmiert werden kann, sollte dieser Baustein zunächst gelöscht werden. Zur Programmierung der Lockbits muß in der Kommandozeile zusätzlich zu /p der Parameter /s1, /s3 oder /s7 angegeben werden. Die höheren Lockbits schließen automatisch die niedrigeren mit ein; /s7 programmiert also alle Lockbits bei jedem beliebigen MCS51 Controller:

```
hedchip /gp87c5x /p /v /s7 myapp.hex
```

Außerdem verfügen einige MCS51 Controller über ein 'Encryption Array'. Die Programmierung dieser 'Schutzmaßnahme' wird von **hed.chip** nicht unterstützt, da uns dafür keine sinnvolle Anwendung bekannt ist.

Atmel Controller der Serie AT89C**

können im **hed.chip** elektrisch gelöscht werden. Auch die kleinen Varianten im DIP20 Gehäuse können direkt in den Programmiersockel des hed.chip eingesetzt werden. Diese Bausteine können auch in einem Arbeitsgang gelöscht und neu programmiert werden:

```
hedchip /ga89c5x /p /v /e myapp.hex          ; für AT89C51/2, LV51/2
hedchip /ga89cx051 /p /v /e myapp.hex        ; für AT89C1051/2051
```

Lediglich wenn ein Baustein mit gesetzten Lockbits neu programmiert werden soll, muß zuvor ein separater Löschvorgang durchgeführt werden.

Für die AT89C**-Varianten mit 5V Programmierspannung wird das Mnemonic /ga89c5x-5 verwendet. Die Typen AT89LV** können mit den gleichen Einstellungen programmiert werden.

AT89C51RC und AT89C55WD

Für diese Bausteine wird das Bausteinmnemonik /ga89c5x2 verwendet. Lesegeschützte Bausteine können von leeren Bausteinen nicht unterschieden werden. Um programmierte und lesegeschützte Bausteine neu zu programmieren, muß der Baustein vorher in einem separaten Aufruf des Programmiergerätes gelöscht werden:

```
hedchip /ga89c5x2 /e                         ; Löschen (für geschützte Bausteine)
hedchip /ga89c5x2 /p /v /s7 myapp.hex        ; Programmieren, Verifizieren, Leseschützen.
```

Atmel Controller der Serie AT89S**

können im **hed.chip** gelöscht und programmiert werden. **hed.chip** kann auch die SPI Security Fuse aktivieren und den EEPROM-Datenspeicher des AT89S8252 programmieren. Zum Programmieren des FLASH-Programmspeichers und des EEPROM-Datenspeichers werden zwei verschiedene Mnemonics benutzt. Die Schutzfunktionen können nur in Verbindung mit dem Mnemonic a89sxxxx benutzt werden. Der Löschvorgang wirkt sich immer auf den ganzen Baustein, also FLASH-Programmspeicher, EEPROM-Datenspeicher und alle Schutz-Funktionen aus.

```
hedchip /ga89sxxxx /p/v/e/s7 myapp.hex       ; AT89S8252 oder AT89S53 FLASH löschen, pro-
                                             ; grammieren, verifizieren und schreib-/leseschützen
hedchip /ga89sxxxx /p/v/e/s15 myapp.hex      ; dito, auch SPI Security Fuse setzen
hedchip /ga89sxxxx /p/v/e/s7/s8 myapp.hex    ; dito, /s7/s8 entspricht /s15
hedchip /ga89seeprom /p/v myapp.hex          ; AT89S8252 EEPROM programmieren und
                                             ; verifizieren.
```

Wenn Programm- und Datenspeicher programmiert und geschützt werden sollen, muß folgende Reihenfolge eingehalten werden: Baustein löschen, EEPROM-Datenspeicher programmieren, FLASH-Programmspeicher programmieren und schützen. Der Datenspeicher ist dann auch geschützt.

Die Typen AT89LS** können mit den gleichen Einstellungen programmiert werden.

Atmel/Temic (A)T89C51R*2

Diese Controller verfügen über einige besondere Optionen. Nach dem Löschen befindet sich der Controller im Urzustand wie nach der Auslieferung durch den Hersteller. Der Hersteller Temic wurde von Atmel übernommen und die Temic-Produkte werden nach und nach in das Atmel-

Programm integriert. Bei einigen Typen gibt es noch Unterschiede, z.B. bei Atmel AT89C51RD2 und Temic T89C51RD2. Andere Typen sind ohne Einschränkung austauschbar.

Option	Nicht programmiert	Programmiert
Lockbit 1	MOVC kann in externem Code verwendet werden. (Default)	MOVC-Befehl im externen Speicher ist deaktiviert
Lockbit 2	Programmspeicher kann mit Programmer ausgelesen werden. (Default)	Auslesen des Bausteins mit Programmer ist deaktiviert
Lockbit 3	Code kann in externem Speicher ausgeführt werden. (Default)	Ausführen von Code im externen Speicher ist deaktiviert.
XRAM	XRAM ist aktiviert. (Default)	XRAM ist deaktiviert
OSC	OSCA ist aktiviert (Default) (nur AT89C51IC2)	OSCB ist aktiviert (nur AT89C51IC2)
BLJB	Programmausführung startet an Adresse 0x0000 (Default)	Programmausführung startet an Adresse 0xFC00 (Boot Loader)
BLLB	Programmierung des Boot Loader Segment erlaubt (Default) (nur T89C51RD2)	Programmierung des Boot Loader Segment deaktviert. (nur T89C51RD2)
X2	Standard mode (12 Taktzyklen/Befehl) (Default).	X2-Mode (6 Taktzyklen/Befehl)

In der Benutzeroberfläche HC95 werden zu jedem Baustein die jeweils relevanten Optionen zur Programmierung angeboten.

Bei der Programmierung wird der Inhalt des HSB auch an Adresse 0x0004 im XAF kopiert. Dort kann die Controllersoftware also den Inhalt des HSB abfragen.

Philips 87C7**-Controller

Für die Philips Controller 87C749 und 87C752 ist der Adapter DIP752 notwendig. Zu beachten ist, daß die Controller 87C748, ‘749, ‘751 und ‘752 mit dem Mnemonic /gp87c7xx programmiert werden, der 87C750 jedoch mit dem Mnemonic /gp87c750. Controller von Signetics werden wie Philips Bausteine behandelt.

Philips P89C**-Controller

hed.chip unterstützt die Typen P89C51Uxxx, P89C52Uxxx, P89C54Uxxx, P89C58Uxxx, P89C51RC+, -RD+, P89C51RB2, -RC2, -RD2. Alle diese Typen haben die üblichen 3 Lockbits wie andere MCS51 Microcontroller auch.

P89C51RC+ und –RD+ verfügt zusätzlich über ein Status-Byte. Dieses kann programmiert werden, wenn bei der Programmierung /S8 zusätzlich angegeben wird.

P89C51RB2, -RC2, -RD2 arbeiten per Voreinstellung im 6x clock mode. In diesem Mode wird ein Maschinenzyklus in 6 Taktzyklen ausgeführt. Der Baustein arbeitet also mit doppelter Geschwindigkeit im Vergleich zu anderen, normalen MCS51 Microcontrollern. Wenn die Option /s16 (= 12x clock mode) programmiert wird, arbeitet der Microcontroller mit der normalen Geschwindigkeit. Wenn diese Option einmal programmiert ist, dann ist es nicht möglich, den Microcontroller wieder in den 6x clock mode zu versetzen. Der 12x clock mode sollte programmiert werden, wenn man diese Controller als Ersatz für andere MCS51 Microcontroller einsetzen will und der Controller das gleiche Zeitverhalten wie der ursprünglich Controller haben soll.

Philips P89C51M*2-Controller

hed.chip unterstützt die Typen P87C51MA2, P87C51MB2 und P87C51MC2. Bei einigen (Vorserien-) Mustern kann es sein, dass das Programmiergerät den genauen Typ nicht bestimmen kann. Das kann dzu führen, dass aus einem P87C51MB2 beim Auslesen 96kByte gelesen werden, auch wenn der Controller nur 64kByte EPROM Programmspeicher hat.

Dallas High Speed Controller DS87C520/530

Die Bausteine DS87C520 und ‘530 haben einen Watchdog Timer. Dieser kann im Falle eines Programmabsturzes ein RESET des Controllers auslösen. Der Watchdog Timer läuft immer, löst aber nur dann ein RESET aus, wenn die entsprechende Funktion aktiviert wurde. Damit der Watchdog von Anfang an im Falle eines Programmabsturzes ein RESET auslösen kann, müssen die Bausteine mit dem Mnemonic d87c5x0-w programmiert werden.

Dallas High Speed Controller DS89C420

17.03.2002: Bei den ersten Entwicklungsmustern des DS89C420 scheint die Programmierung der Lock Bits nicht zu funktionieren.

Siemens SAB-C513A

hed.chip unterstützt auch den Baustein Siemens SAB-C513A-H. Dieser Baustein ist von Siemens nur zum Zwecke der Entwicklung gedacht. Er verfügt über keine Lockbits und beim Einsatz sollte das Siemens Errata Sheet, Release 1.2 vom 20.09.1995 für Bausteine mit der Markierung „ES-BA“, beachtet werden.

Siemens C505A-4E, C505CA-4E

Diese Microcontroller von Siemens verfügen über 32kByte OTP EPROM Speicher. (OTP = One Time Programmable = einmalig programmierbar). Der Controller steht nur in der Bauform PQFP44 zur Verfügung.

Das Pinout unterscheidet sich etwas von anderen MCS51 Microcontrollern in dieser Bauform. An Pins 38/39 liegt die Versorgung für den internen AD-Wandler, an Pin 17 liegt die digitale Versorgung. Für die Programmierung wird nur die digitale Versorgung angeschlossen.

Wegen dieser Besonderheit muss zur Programmierung der Adapter PQFP44_C505 verwendet werden. Dieser Adapter wurde freundlicherweise von der Firma Kriwan entworfen und hergestellt.

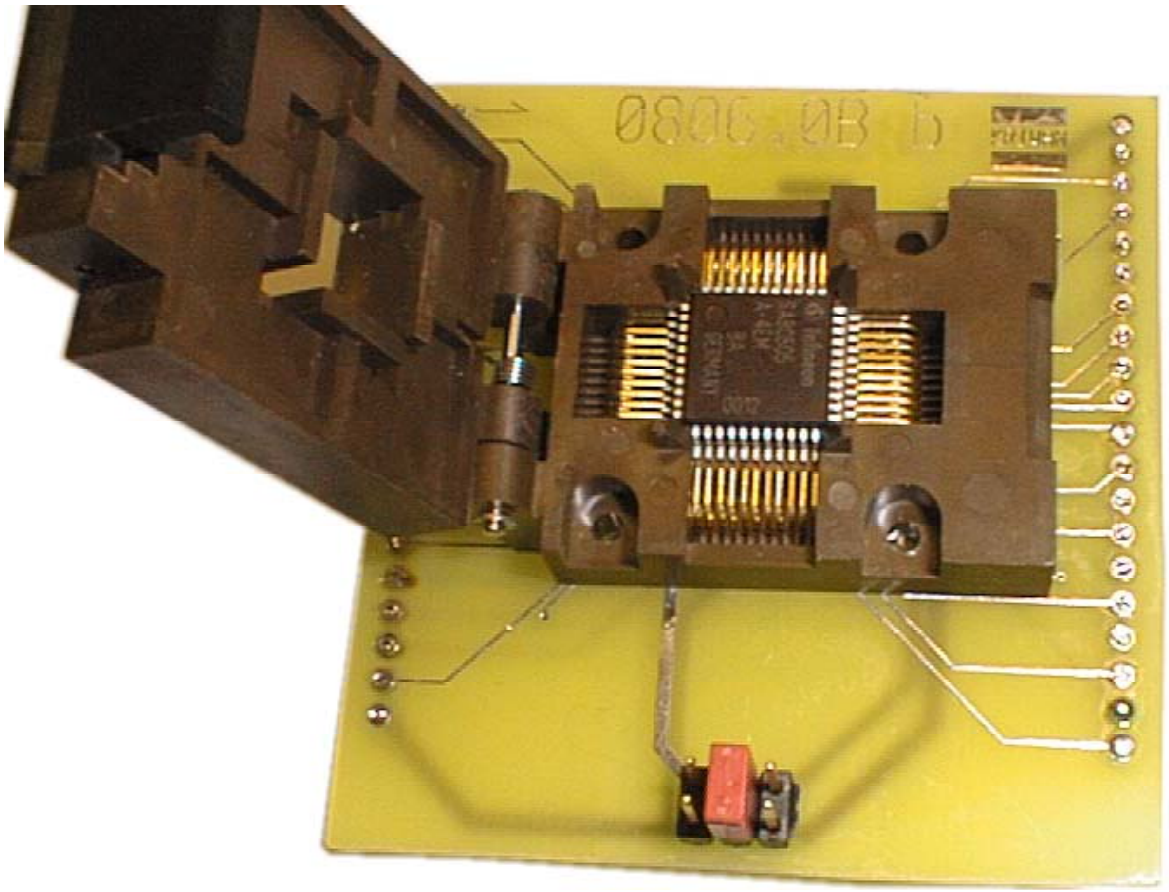


Bild: Bitte beachten Sie die Orientierung des Siemens C505A im PQFP44-Sockel und die Einstellung des Jumpers

SST89F5*

Die MCS51 Micocontroller von SST zeichnen sich durch eine Besonderheit aus. Sie haben zwei getrennte Blöcke FLASH Programmspeicher. Block 0 ist der primäre Speicher und hat eine Größe von 16 bzw. 32 kByte. Block 1 hat eine Größe von 4 kByte und liegt an Adresse 0xF000. Eine Besonderheit ist, daß diese Microcontroller den eigenen FLASH Programmspeicher beschreiben können. Im Datenblatt wird dieses Feature als „In-Application Programming“ beschrieben.

Für die Programmierung des Block 0 wird das Bausteinmnemonic /gsst89f5x_0 verwendet.

Für die Programmierung des Block 1 wird das Bausteinmnemonic /gsst89f5x_1 verwendet. Bei Quelldateien im Intel Hex Format muß darauf geachtet werden, daß die Quelldatei keinen Offset enthält. Dazu muß im Assembler die Anweisung „.phase 0xf000“ statt „.org 0xf000“ benutzt werden.

Die Löschfunktion löscht immer beide Speicherblöcke.

Diese Microcontroller haben keine Lockbits. Statt dessen wird der Lese- und Schreibschutz des internen Speichers durch den Inhalt der letzten Speicherzelle im Block 1 bestimmt. Dieses Byte wird Security Byte genannt. Der Schreibschutz ist sinnvoll, damit der Programmspeicher nicht unbeabsichtigt verändert werden kann.

Parameter	Sec. Byte	Funktion
/S0	0xFF	Kein Schutz.
/S85	0x55	Beide FLASH Speicherblöcke sind geschützt (hard lock)
/S245	0xF5	Nur der obere 4 kByte Block ist geschützt (hard lock)
/S5	0x05	Beide Speicherblöcke sind geschützt, können aber durch In-Application Programming programmiert werden. (soft lock).

Der Schutz kann entweder durch den Inhalt der Quelldatei für Block 1 oder durch Parameter in der Kommadozeile aktiviert werden. Parameter in der Kommandozeile haben Vorrang vor einen Wert für das Security Byte in der Quelldatei. Wenn beide Speicherblöcke programmiert werden sollen, darf der Schutz erst bei der Programmierung des zweiten Speicherblocks aktiviert werden.

Temic TSC87C51

Dieser Microcontroller hat keine Lockbits. Da **hed.chip** das Encryption-Array nicht programmieren kann, gibt es keinen Schutz gegen Auslesen. Das Encryption-Array ist als Schutz gegen Auslesen ohnehin fast wirkungslos.

Temic TS87C52X2

Dieser Microcontroller ist bei gleicher Taktfrequenz doppelt so schnell, wie normale MCS51 Microcontroller. Er verfügt über die üblichen 3 Lockbits zum Schutz gegen Auslesen

2.3 Atmel AVR-RISC Microcontroller

Diese neue Microcontroller Familie basiert auf einer Weiterentwicklung der Peripherie der MCS51 Familie und einem neu entwickelten Prozessorkern. Der Prozessor wurde auf optimale Unterstützung durch die Programmiersprache C optimiert, wird aber auch von Assembler Programmierern als komfortabel empfunden.

Die AVR-RISC-Controller verfügen über FLASH Programmspeicher und EEPROM Datenspeicher. Sie können sowohl parallel mit einem Programmiergerät als auch seriell IN CIRCUIT programmiert werden. Beide Speichertypen und die Bausteinoptionen können mit **hed.chip** programmiert werden. Die Löschfunktion haben beide Speichertypen gemeinsam. Das heißt, Löschen des FLASH-Programmspeicher löscht automatisch auch den EEPROM-Datenspeicher und umgekehrt.

Wenn der FLASH-Programmspeicher nicht leer ist, muss er vor der Programmierung gelöscht werden. Der EEPROM-Datenspeicher ohne vorheriges Löschen neu programmiert werden.

Zur Zeit sind folgende Mitglieder dieser Familie voll verfügbar: AT90S1200, AT90S2313, AT90S4414 und AT90S8515. Weitere Typen sind angekündigt. Zur Programmierung des FLASH-Programmspeichers werden die Mnemonics /gavr20 (Bauform DIP20) bzw. /gavr40_2 (Bauform DIP40), für den EEPROM-Datenspeicher die Mnemonics /gavr20e bzw. /gavr40e benutzt.

AT90S Lockbits und Fuses

Der FLASH-Programmspeicher kann mit Lockbits gegen Veränderung und Auslesen geschützt werden. Mit zwei Fuses können weitere Optionen eingestellt werden. Diese Fuses können nur mit einem Programmiergerät, nicht IN CIRCUIT via SPI programmiert werden. Der Parameter /s kann benutzt werden, um im Anschluß an die Programmierung des FLASH-Programmspeichers diese Optionen zu aktivieren.

Parameter	Lockbits	Funktion
S1	1	Schützt FLASH Programmspeicher gegen weitere Programmierung
S3	1 + 2	Schützt gegen Auslesen des Programmspeichers. S3 beinhaltet automatisch S1
S4	RCEN	AT90S1200/2313: Aktiviert den internen Oszillator des Watchdog Timer als Taktquelle für den Prozessor. Dann arbeitet der Controller ohne externen Quarz mit einer Taktfrequenz von ca. 1MHz.
S4	SPI disable	AT90S4414/8515: Schaltet die INCIRCUIT Programmierbarkeit ab.
S8	SPI disable	AT90S1200/2313: Schaltet die INCIRCUIT Programmierbarkeit ab.
S8	FSTRT	AT90S4414/8515: verkürzt die RESET-Wartezeit nach einem PowerOn. Sinnvoll bei schnell startenden Taktquellen, z.B. Keramik-Resonatoren.

Diese Optionen können in beliebiger Kombination benutzt werden. Dabei kann man entweder die Werte selbst addieren oder den Parameter mehrfach in der Befehlszeile benutzen.

```
hedchip /gavr20 /p/v/e /s15 yourapp.bin          ; Lockbits, SPI disable und RCEN programmieren
hedchip /gavr20 /p/v/e /s3/s4/s8 yourapp.bin     ; entspricht Programmierung mit /s15
```

Wenn der FLASH Programmspeicher ausgelesen wird, speichert **hed.chip** den Zustand der Fuses (SPI disable und RCEN) als letztes Byte in die Zieldatei. Bei der Programmierung eines Bausteins mit einer solchen Datei werden also auch die Fuses entsprechend programmiert.

ATtiny2313

Der ATtiny2313 ist der Nachfolger des AT90S2313. Allerdings ist beim Ersatz einiges zu beachten. Der ATtiny2313 hat andere und mehr Fuses als der AT90S2313. Beim ATtiny2313 können die Fuses nur über die Quelldatei programmiert werden. Die Quelldatei muss dazu an der Adresse 2048 (= 0x800 in Hexadezimal) ein Datenwort für die Fuses enthalten. Beim Auslesen eines Bausteins wird der Inhalt der Fuses an das Ende des FLASH-Programmspeichers (Adresse 2048) angehängt.

Stand 21.10.06: Das High-Byte der Fuses wird immer mit 0xFF ausgelesen. Es ist unklar, ob der Versuch, ein Byte zu programmieren scheitert oder ob lediglich das anschließende Zurücklesen des Bytes scheitert. Bei einem neuen Baustein wird der Inhalt der Fuses mit 0xFF64 ausgelesen.

ATmega

Die Bausteine der ATmega-Serie sind den AT90S-Typen sehr ähnlich. Allerdings haben diese Bausteine zu viele Fuses und Lockbits, um diese Eigenschaften über die Befehlszeile programmierbar zu machen.

Die Fuses können daher nur über die Quelldatei programmiert werden. In der Quelldatei muss dazu ein Byte (z.B. ATMEGA161) bzw. ein Word (z.B. ATMEGA163) unmittelbar nach den Daten für den FLASH Programmspeicher folgen. Die dritte Zeile in der folgenden Tabelle gibt den Default-Zustand an, in dem der Baustein vom Hersteller ausgeliefert wird. Dieser Wert wird von **hed.chip** beim Löschen eines Bausteins ebenfalls wieder hergestellt

Beispiel: ATMEGA161
Adresse: 0x4000
Fuse-Byte (8 Bit):

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	X	BOOTRST	SPIEN	BODLEVEL	BODEN	CKSEL(2)	CKSEL(1)	CKSEL(0)
Default:	0	1	0	1	1	0	1	0

Die Quelldatei für einen ATMEGA161 hat inkl. Programmierung der Fuses eine Größe von 16385 Bytes.

ATmega Lockbits

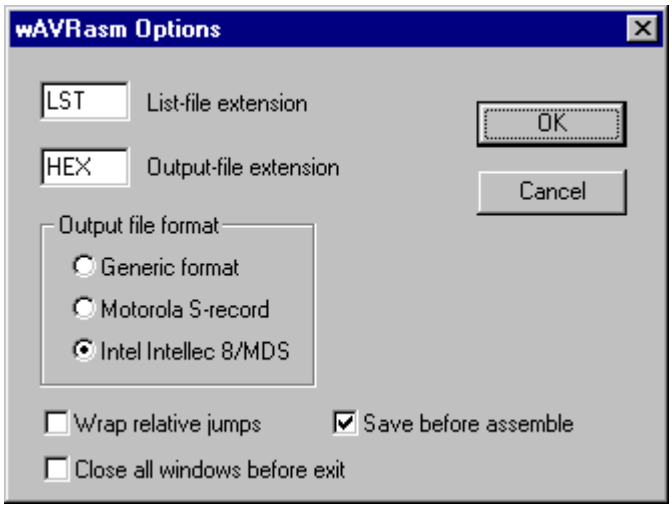
ATmega Bausteine können den eigenen Programmspeicher selbst beschreiben. Alle nicht benötigten Schreibvorgänge sollten mit diesen Lock Bits verhindert werden, damit im Falle eines Absturzes der FLASH Programmspeicher nicht überschrieben wird oder wenigstens der Inhalt eines Boot-Blocks erhalten bleibt.

Parameter	Lockbits	Funktion
S1	LB1	Schützt FLASH Programmspeicher gegen weitere Programmierung mit einem Programmiergerät
S2	LB2	Schützt gegen Auslesen des Programmspeichers mit einem Pro-

Parameter	Lockbits	Funktion
		grammiergerät..
S4	BLB01	Schützt gegen Programmierung des Application Speicher Blocks
S8	BLB02	Schützt gegen Auslesen des Application Speicher Blocks
S16	BLB11	Schützt gegen Programmierung des Boot Loader Speicher Blocks
S32	BLB12	Schützt gegen Auslesen des Boot Loader Speicher Blocks

Atmel AVR Assembler 1.30

Dieser Assembler für die AT90S-Serie kann kostenlos von Atmel aus dem WWW bezogen werden. Für die Programmierung der Bausteine sind folgende Einstellungen zu wählen:



Das Format „Intel Intellec 8/MDS“ entspricht dem normalen Intel HEX Format. Wenn im Assembler ein „.eseg“ Bereich erzeugt wird, gibt der Assembler auch für das EEPROM des Controllers Daten aus. Diese Datei bekommt automatisch und ohne Einstellmöglichkeit die Namenserweiterung „.EEP“. Das Datenformat entspricht dem für das Programm eingestellten Format. Bei der obigen Einstellung wird also eine Datei im Intel Hex Format erzeugt, deren Namen nicht auf „.HEX“ endet. Damit diese Datei bei der Programmierung korrekt konvertiert wird, muß sie umbenannt werden. Beispiel:

Sie assemblieren die Datei yourapp.asm. Daraus werden unter anderem folgende Dateien erzeugt:

```
yourapp.hex                    ; in FLASH-Programmspeicher programmieren
yourapp.eep                    ; umbenennen in EEPROM.HEX und in
                               ; EEPROM-Datenspeicher programmieren
```

Mit yourapp.hex wird der FLASH-Programmspeicher des Controllers programmiert. Die Datei yourapp.epp wird umbenannt in eeprom.hex. Die Änderungen des Namens ist notwendig, damit die Datei als Intel Hex Datei erkannt wird. Dann wird die Datei in den EEPROM-Datenspeicher des Controllers programmiert. Dabei sollte die Option „vorher löschen“ nicht aktiviert werden. Achtung: Die Löschfunktion löscht immer beide Speicher (FLASH und EEPROM) des Controllers.

2.4 Microchip PIC Microcontroller

Mit Hilfe des Adapters UNIPIC unterstützt **hed.chip** auch eine große Zahl von PIC Microcontrollern. **hed.chip** erfüllt alle Anforderungen, die Microchip für einen „production quality programmer“ spezifiziert hat. Für die verschiedenen Bauformen DIP8, DIP18, DIP28 und DIP40 hat der Adapter die jeweils passenden Testsockel bzw. einen Präzisionssockel für die Bauform DIP8. Der Adapter wird in zwei Versionen angeboten:

1. UNIPIC18: bestückt mit Präzisionssockel DIP8 und Testsockel DIP18. Sockel DIP28 und DIP40 können gegebenenfalls selbst nachträglich bestückt werden.
2. UNIPIC: komplett bestückt mit Präzisionssockel DIP8 und drei Testsockeln DIP18, DIP28 und DIP40.

PIC Microcontroller verfügen über EPROM- oder FLASH-Programmspeicher. Die Wortbreite des Speichers beträgt je nach Typ 12, 14 oder 16 Bits. Zusätzlich zu dem normalen Programmspeicher haben diese Bausteine 16 Bits Speicher für eine User-ID und ein Speicherwort zur Konfiguration des Prozessors (Microchip: „configuration word“). Die 16 Bits der User-ID verteilen sich auf 4 Speicherworte. Beim Auslesen werden User-ID und configuration word mit ausgelesen. Die Zieldatei ist daher immer 10 Byte größer als der Programmspeicher des Bausteins.

User-ID

PIC Microcontroller haben eine 16 Bit User-ID. Diese User-ID wird in einem besonderen Adressbereich in 4 Speicherstellen, die jeweils mit 4 Bit der User-ID programmiert werden können, abgelegt. **hed.chip** kann eine User-ID aus der Quelldatei in die dafür vorgesehenen Speicherstellen programmieren. Die User-ID wird in der Quelldatei hinter den Daten für den Programmspeicher erwartet. Dabei spielt die genaue Position keine Rolle. Wenn die Quelldatei mindestens 10 Bytes größer ist als der Programmspeicher des Bausteins, werden die letzten 10 Bytes der Quelldatei in die User-ID und das configuration word programmiert.

Beispiel für PIC16C84: Der Baustein hat 1024 Worte Programmspeicher. Das entspricht einer Größe der Quelldatei von 2048 Byte. Die nun folgenden 4 Wörter (= 8 Bytes) werden als User-ID interpretiert. Von jedem Wort werden die unteren 4 Bits als User-ID in den Baustein programmiert. Um Fehlermeldungen beim Verify zu vermeiden, müssen die oberen 12 Bits des Worts 0 sein. Eine Quelldatei mit User-ID und configuration word hat beim PIC16C84 also eine Größe von 2058 Bytes.

Configuration Word

PIC Microcontroller haben in einem besonderen Adressbereich eine Speicherstelle, mit deren Hilfe der Microcontroller konfiguriert werden kann. Die Bits des configuration word stellen den Taktgenerator auf bestimmte Taktquellen ein und beeinflussen das Verhalten des Timers und des Watchdog. Mit weiteren Bits kann der Prozessor gegen Auslesen des Programmspeichers geschützt werden. Nach dem Löschen des Bausteins sind alle Bits des configuration word auf 1 gesetzt. Bei der Programmierung können Bausteinoptionen angegeben werden, um eines oder mehrere Bits des configuration word auf 0 zu programmieren. Die folgende Tabelle gibt eine Übersicht über die Optionen und die zugehörigen Kommandozeilenparameter, die zur Programmierung angegeben werden können. Die graphische Benutzeroberfläche HC95 bietet bei der Programmierung die für den gewählten Baustein verfügbaren Bausteinoptionen zur Auswahl an. Es können beliebige Kombinationen der Optionen ausgewählt werden.

Parameter	Option	Funktion
/S1 /S2	FOSC0 FOSC1	Oscillator Selection Bit. In Verbindung mit dem Parameter FOSC1 können verschiedene Typen für den Taktgenerator des Microcontrollers einstellt werden. RC Oscillator : default, keine Optionsangabe. Controller wird mit RC-Glied beschaltet. HS Oscillator: FOSC0 programmieren. Controller mit Quarz beschaltet, hohe Taktfrequenz. XT Oscillator: FOSC1 programmieren. Controller mit Quarz beschaltet, mittlere Taktfrequenz LP Oscillator: FOSC0 und FOSC1 programmieren. Controller mit Quarz beschaltet, niedrige Taktfrequenz.
/S4	WDTE WDTEN	Watch Dog Timer Enable. Per Default ist der Watchdog Timer nach einem Reset aktiviert. Die Angabe der Option WDTE bei der Programmierung deaktiviert den Watchdog Timer.
/S8	PWRTE PWRTE# PWRTEN PWRTEN#	PWRTE: Power Up Timer Disable Bit. Bei einigen Bausteinen ist der Timer nach einem Reset per Default aktiviert und kann durch Angabe dieser Option deaktiviert werden. PWRTE#: Power Up Timer Enable Bit. Bei einigen Bausteinen ist der Timer nach einem Reset per Default deaktiviert und kann durch Angabe dieser Option aktiviert werden. Die Benutzeroberfläche bietet jeweils die zu dem Baustein passende Option zur Programmierung an.
/S16	BODEN BOREN	Brown Out Enable Bit. Bei Bausteinen mit dieser Option gibt es einen Schutz gegen langsamen Ausfall der Betriebsspannung.
/S16	FOSC2	Nur PIC12C67x: Oscillator Selection Bit. Diese Option darf nur in Verbindung mit FOSC0 und/oder FOSC1 angegeben werden.
/S32	CP CP0 CP#	Leseschutz. Je nach Baustein wird der gesamte Speicher oder die obere Hälfte gegen Auslesen geschützt.
/S32	LVP	Low Voltage Programming Enable: Wenn gesetzt, dann hat Pin RB3 die Funktion PGM. Wenn nicht gesetzt, dann Pin RB3 normale IO-Funktion. Zur Programmierung muss Programmierspannung an Pin MCLR# angelegt werden.
/S64	CP1	Leseschutz. Bei Bausteinen mit geteilt aktivierbarem Leseschutz schützt diese Option die untere Hälfte des Speichers gegen Auslesen.
/S64	DP CPD	PIC16CR83/84 und andere: Data EEPROM Leseschutz.
/S64	BORV0	PIC16C773/4: Brown Out Reset Voltage. Einstellung der Spannung für die Erkennung des langsamen Abfalls der Versorgungsspannung.

Parameter	Option	Funktion
/S128	MCLRE	Master Clear pin Enable Bit. Wenn diese Option programmiert wird, wird der MCLR-Pin des Bausteins deaktiviert und MCLR im Controller mit Vdd verbunden. Nur bei PIC12C508/9
/S128	WRT	FLASH Memory Write Enable. Wenn diese Option programmiert wird, wird verhindert, dass der Microcontroller seinen eigenen FLASH-Speicher beschreiben kann.
/S128	BORV1	PIC16C773/4: Brown Out Reset Voltage. Einstellung der Spannung für die Erkennung des langsamen Abfalls der Versorgungsspannung.

Das configuration word kann auch aus der Quelldatei heraus programmiert werden. Dazu muß das configuration word in der Quelldatei unmittelbar hinter der User-ID liegen. Alternativ können di-diese Informationen auch an den von Microchip vorgesehen stellen liegen. Sie: Kaptel „Entwicklungssystem MPLAB“. Beim Auslesen werden User-ID und configuration word im Anschluß an den Inhalt des Programmspeichers in der Zieldatei abgelegt. Man kann also Bausteine inklusive User-ID und configuration word kopieren. Der genaue Aufbau des configuration word ist bei fast allen Typen unterschiedlich. Das gilt auch, wenn diese Typen über die gleichen Optionen verfügen.

Leseschutz bei UV-löschbaren PIC Microcontrollern

Microchip empfiehlt, daß Microcontroller im Keramik-Gehäuse mit Fenster nicht gegen Auslesen geschützt werden. Das heißt, die Bausteinoptionen /s32 und /s64 sollten bei der Programmierung dieser Typen *nicht* angegeben werden. Wir haben die Erfahrung gemacht, daß auch intensive UV-Bestrahlung diese Bausteine nicht mehr vollständig löschen kann, wenn die configuration bits CP0 (/s32) und/oder CP1 (/s64) programmiert wurden.

Der Leseschutz kann auch durch Programmierung mit einem entsprechende Wert für das configuration word in der Quelldatei aktiviert werden. Dabei wird aber eine Fehlermeldung erzeugt, daß der Baustein nicht programmierbar sei. Die Aktivierung des Leseschutzes kann auch unbeabsichtigt passieren, wenn der Baustein mit einer Datei programmiert wird, die nicht für diesen Typ geeignet ist.

Daten EEPROM

Einige Typen, z.B. PIC16F84, verfügen über EEPROM Datenspeicher. Dieser Speicher erscheint in der Bausteinliste je nach Größe als PICDATA64, PICDATA128, bzw. PICDATA256.

Um also die 128 Bytes EEPROM Datenspeicher eines PIC16F873 zu programmieren, wählen Sie aus der Bausteinliste den Baustein PICDATA128 aus. In der Kommandozeile verwenden Sie das Baustein-Mnemonik /gpicdata128.

Der EEPROM Datenspeicher hat eine Wortbreite von 8 Bits. Trotzdem werden die Daten wortweise (16 Bits) aus der Quelldatei gelesen. Von jedem 16 Bit Wort werden nur die unteren 8 Bits verwendet. Eine Quelldatei für 128 Bytes EEPROM Datenspeicher hat also eine Größe von 256 Bytes.

Der Datenspeicher muss vor der Programmierung des Programmspeichers programmiert werden. Folgende Reihenfolge ist einzuhalten:

1. Den jeweiligen Baustein, z.B. PIC16F873, löschen. Dadurch wird auch das Daten EEPROM gelöscht.
2. Daten EEPROM programmieren. Für PIC16F873 muss dazu aus der Bausteinliste PICDATA128 ausgewählt werden.
3. EPROM oder FLASH Programmspeicher programmieren und gegebenenfalls gegen Auslesen schützen. Dabei kann auch der Datenspeicher gegen Auslesen geschützt werden.

Entwicklungssystem MPLAB

hed.chip unterstützt das Entwicklungssystem MPLAB von Microchip.

Damit ist insbesondere gemeint, dass User-ID und Configuration-Word aus den mit MPLAB erstellten Dateien an die richtigen Stellen im PIC Microcontroller programmiert werden.

Damit das funktioniert, werden die Dateiformate von dem Programm HEXBIN.EXE konvertiert. Der Aufruf von HEXBIN.EXE erfolgt automatisch. HEXBIN.EXE konvertiert die Formate MPLAB-14Bit und MPLAB-12Bit in das HEDCHIP-Format.

Dateiformate:

1. HEDCHIP-Format
   HEDCHIP erwartet User-ID und Configuration-Word unmittelbar im Anschluss an den Programmspeicher des Controllers. Die genaue Adresse ist von der Speichergröße des Controllers abhängig. Beispiel für PIC16C84:
   Speichergröße: 400h Worte = 800h Bytes = 2kByte
   User-ID an Adresse: 0800h (Byte-Adresse)
   Config-Word an Adresse: 0808h (Byte-Adresse)
2. MPLAB-14Bit:
   User-ID an Adresse: 4000h (Byte-Adresse)
   Config-Word an Adresse: 400eh (Byte-Adresse)
   Im Falle eines PIC16C84 verschiebt HEXBIN.EXE die User-ID von Adresse 4000h nach Adresse 0800h und das Config-Word von Adresse 400eh nach Adresse 0808h.
3. MPLAB-12Bit:
   User-ID an Adresse: Unmittelbar im Anschluß an den Programmspeicher, z.B. bei PIC12C508 an Adresse 0400h (Byte-Adresse)
   Config-Word an Adresse: 1ffeh
   Im Falle eine PIC12C508 verschiebt HEXBIN.EXE das Config-Word von Adresse 1ffeh nach Adresse 0408h.

Empfohlene Arbeitsweise:

Geben Sie in MPLAB alle Configurations-Optionen mit Ausnahme des Leseschutzes an. Wenn Sie mit UV-Löschbaren Controller, die eine Oscillator-Calibration brauchen, arbeiten, können Sie dafür einen geeigneten Wert in die Quelldatei schreiben. Der Oscillator-Calibration wert wird nur dann programmiert, wenn der Microcontroller an dieser Stelle noch keinen programmierten Wert enthält.

Mit der so entstandenen Quelldatei können Sie Bausteine programmieren. Während der Entwicklung und wenn Sie UV-Löschbare Controller verwenden, sollten Sie keinen Leseschutz benutzen. Für die Serienfertigung können Sie in HC95 die Bausteinoptionen CP bzw. CP0 + CP1 zusätzlich aktivieren. Optionen, die durch entsprechende Daten in der Quelldatei programmiert werden und in HC95 ausgewählte Optionen addieren sich.

Übersicht über die unterstützten PIC Microcontroller Typen

Die folgende Tabelle listet die unterstützten Microcontroller auf und nennt das bei der Programmierung anzugebende Bausteinmnemonic. Typen, die sich für das Programmiergerät nicht unterscheiden, wurden zu einem Typ zusammengefaßt. Deshalb wird z.B. der Baustein PIC16C71 durch Angabe des BausteinMnemonics /gpic16c61 programmiert. In der Datenbank von HC95 sind alle Typen aufgeführt und es wird automatisch mit dem richtigen Bausteinmnemonic programmiert.

Der Adapter UNIPIC hat für jeden Microcontroller den jeweils passenden Sockel. Der Baustein muß in den der Bauform entsprechenden Sockel eingesetzt werden. Beispiel: Ein Baustein PIC16C84 in DIP18 darf nur in den Testsockel DIP18 eingesetzt werden. PIC Microcontroller können vom Programmiergerät nicht erkannt werden. Es ist sehr wichtig, daß der richtige Baustein ausgewählt wird. Das gilt insbesondere für Typen mit und ohne ´A´ als letzten Buchstaben in der Typbezeichnung. Beispiel: PIC16C62 und PIC16C62A sind nicht identisch.

Baustein	Bauform	Mnemonic	Programmierbare Optionen
PIC12C508/A	DIP8	/gpic12c508	FOSC0, FOSC1, WDTE, CP, MCLRE
PIC12C509/A	DIP8	/gpic12c509	FOSC0, FOSC1, WDTE, CP, MCLRE
PIC12C671	DIP8	/gpic12c671	FOSC0, FOSC1, FOSC2, WDTE, PWRTE, CP0, CP1, MCLRE
PIC12C672	DIP8	/gpic12c672	FOSC0, FOSC1, FOSC2, WDTE, PWRTE, CP0, CP1, MCLRE
PIC12CE518	DIP8	/gpic12c508	FOSC0, FOSC1, WDTE, CP, MCLRE
PIC12CE519	DIP8	/gpic12c509	FOSC0, FOSC1, WDTE, CP, MCLRE
PIC12CE673	DIP8	/gpic12c671	FOSC0, FOSC1, FOSC2, WDTE, PWRTE, CP0, CP1, MCLRE
PIC12CE674	DIP8	/gpic12c672	FOSC0, FOSC1, FOSC2, WDTE, PWRTE, CP0, CP1, MCLRE
PIC12F629	DIP8	/gpic12f629	FOSC0, FOSC1, FOSC2, WDTE, PWRTE#, MCLRE, BODEN, CP, CPD
PIC16C61	DIP18	/gpic16c61	FOSC0, FOSC1, WDTE, PWRTE, CP
PIC16C62	DIP28	/gpic16c62	FOSC0, FOSC1, WDTE, PWRTE, CP0, CP1
PIC16C620	DIP18	/gpic16c620	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16C620A	DIP18	/gpic16c620	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16C621	DIP18	/gpic16c621	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16C621A	DIP18	/gpic16c621	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16C622	DIP18	/gpic16c62a	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16C622A	DIP18	/gpic16c62a	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16C62A	DIP28	/gpic16c62a	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16C62B	DIP28	/gpic16c62a	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16C62C	DIP28	/gpic16c62a	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16C63	DIP28	/gpic16c63	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16C63A	DIP28	/gpic16c63	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16C64	DIP40	/gpic16c62	FOSC0, FOSC1, WDTE, PWRTE, CP0, CP1
PIC16C64A	DIP40	/gpic16c62a	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16C65	DIP40	/gpic16c65	FOSC0, FOSC1, WDTE, PWRTE, CP0, CP1
PIC16C65A	DIP40	/gpic16c63	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16C65B	DIP40	/gpic16c63	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16C66	DIP28	/gpic16c66	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16C67	DIP40	/gpic16c66	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16C71	DIP18	/gpic16c61	FOSC0, FOSC1, WDTE, PWRTE, CP
PIC16C710	DIP18	/gpic16c710	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP
PIC16C711	DIP18	/gpic16c711	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP
PIC16C712	DIP18	/gpic16c621	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1

Baustein	Bauform	Mnemonic	Programmierbare Optionen
PIC16C716	DIP28	/gpic16c62a	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16C717	DIP18	/gpic16c717	FOSC0, FOSC1, FOSC2, WDTE, PWRTE, MCLRE, BORV0, BORV1, CP
PIC16C72	DIP28	/gpic16c62a	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16C72A	DIP28	/gpic16c62a	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16C73	DIP28	/gpic16c65	FOSC0, FOSC1, WDTE, PWRTE, CP0, CP1
PIC16C73A	DIP28	/gpic16c63	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16C73B	DIP28	/gpic16c63	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16C74	DIP40	/gpic16c65	FOSC0, FOSC1, WDTE, PWRTE, CP0, CP1
PIC16C745	DIP28	/gpic16c745	FOSC0, FOSC1, WDTEN, PWRTEN#, CP0, CP1
PIC16C74A	DIP40	/gpic16c63	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16C74B	DIP40	/gpic16c63	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16C76	DIP28	/gpic16c66	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16C765	DIP40	/gpic16c745	FOSC0, FOSC1, WDTEN, PWRTEN#, CP0, CP1
PIC16C77	DIP40	/gpic16c66	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16C770	DIP20	/gpic16c770	FOSC0, FOSC1, FOSC2, WDTE, PWRTE, MCLRE, BORV0, BORV1, CP
PIC16C771	DIP20	/gpic16c771	FOSC0, FOSC1, FOSC2, WDTE, PWRTE, MCLRE, BORV0, BORV1, CP
PIC16C773	DIP28	/gpic16c773	FOSC0, FOSC1, WDTEN, PWRTEN#, BOREN, BORV0, BORV1, CP0, CP1
PIC16C774	DIP28	/gpic16c773	FOSC0, FOSC1, WDTEN, PWRTEN#, BOREN, BORV0, BORV1, CP0, CP1
PIC16C781	DIP20	/gpic16c781	FOSC0, FOSC1, FOSC2, WDTE, PWRTE, MCLRE, BORV0, BORV1, CP
PIC16C782	DIP20	/gpic16c770	FOSC0, FOSC1, FOSC2, WDTE, PWRTE, MCLRE, BORV0, BORV1, CP
PIC16C84	DIP18	/gpic16c84	FOSC0, FOSC1, WDTE, PWRTE, CP
PIC16C923	PLCC68	/gpic16c923	FOSC0, FOSC1, WDTE, PWRTE#, CP0, CP1
PIC16C924	PLCC68	/gpic16c924	FOSC0, FOSC1, WDTE, PWRTE#, CP0, CP1
PIC16CE623	DIP18	/gpic16c620	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16CE624	DIP18	/gpic16c621	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16CE625	DIP28	/gpic16c62a	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16CR62	DIP28	/gpic16c62a	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16CR64	DIP40	/gpic16c62a	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1
PIC16CR83	DIP18	/gpic16cr83	FOSC0, FOSC1, WDTE, PWRTE#, CP, DP
PIC16CR84	DIP18	/gpic16cr84	FOSC0, FOSC1, WDTE, PWRTE#, CP, DP
PIC16F627	DIP18	/gpic16f627	FOSC0, FOSC1, FOSC2, WDTE, PWRTE#+BODEN, CP0+CP1+CPD, LVP, MCLRE
PIC16F628	DIP18	/gpic16f628	FOSC0, FOSC1, FOSC2, WDTE, PWRTE#+BODEN, CP0+CP1+CPD, LVP, MCLRE

Baustein	Bauform	Mnemonic	Programmierbare Optionen
PIC16F630	DIP14	/gpic12f629	FOSC0, FOSC1, FOSC2, WDTE, PWRTE#, MCLRE, BODEN, CP, CPD
PIC16F676	DIP14	/gpic12f629	FOSC0, FOSC1, FOSC2, WDTE, PWRTE#, MCLRE, BODEN, CP, CPD
PIC16F83	DIP18	/gpic16f83	FOSC0, FOSC1, WDTE, PWRTE#, CP
PIC16F84	DIP18	/gpic16f84	FOSC0, FOSC1, WDTE, PWRTE#, CP
PIC16F84A	DIP18	/gpic16f84	FOSC0, FOSC1, WDTE, PWRTE#, CP
PIC16F870	DIP28	/gpic16f870	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP, DP, WRT
PIC16F871	DIP40	/gpic16f870	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP, DP, WRT
PIC16F872	DIP28	/gpic16f870	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP, DP, WRT
PIC16F873	DIP28	/gpic16f873	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1, WRT
PIC16F873A	DIP28	/gpic16f873a	FOSC0, FOSC1, WDTEN, PWRTEN#, BOREN, LVP, CP, WRT
PIC16F874	DIP40	/gpic16f873	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1, WRT
PIC16F874A	DIP40	/gpic16f873a	FOSC0, FOSC1, WDTEN, PWRTEN#, BOREN, LVP, CP, WRT
PIC16F876	DIP28	/gpic16f876	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1, WRT
PIC16F876A	DIP28	/gpic16f876a	FOSC0, FOSC1, WDTEN, PWRTEN#, BOREN, LVP, CP, WRT
PIC16F877	DIP40	/gpic16f876	FOSC0, FOSC1, WDTE, PWRTE#, BODEN, CP0, CP1, WRT
PIC16F877A	DIP40	/gpic16f876a	FOSC0, FOSC1, WDTEN, PWRTEN#, BOREN, LVP, CP, WRT

PIC16CR83/84, PIC16F83/84

Die Bausteine PIC16CR83 und PIC16CR84 verfügen EPROM-Programmspeicher, die Bausteine PIC16F83, PIC16F84(A) verfügen über FLASH-Programmspeicher.

Zusätzlich zum Programmspeicher haben diese Bausteine 64 Bytes EEPROM-Datenspeicher. Zur Programmierung des Datenspeichers wird das Bausteinmnemonic /gpicdata64 benutzt. Die Daten, die in das Daten EEPROM programmiert werden sollen, müssen in einer separaten Datei beginnend bei Adresse 0 vorliegen. In der Quelldatei müssen die Daten wortweise vorliegen. Bei der Programmierung wird von jedem Wort das Low-Byte in den Baustein programmiert.

Beispiel: Bei einem PIC16F84 soll sowohl der Daten- wie der Programmspeicher programmiert werden. Der Programmspeicher soll gegen Auslesen geschützt werden:

```
hedchip /gpicdata64 /p /e /v datafile.hex      ; löscht, programmiert und verifiziert EEPROM-
                                                 Datenspeicher.
hed.chip /gpic16f84 /p /v /e /32 codefile.hex  ; löscht, programmiert, verfiziert und schützt
                                                 FLASH-Programmspeicher.
```

Die Bausteine PIC16CR83 und PIC16CR84 haben eine Bausteinoption, mit der sich der EEPROM-Datenspeicher gegen Auslesen schützen lässt. Die Benutzeroberfläche bietet dazu die Bausteinoption DP an. In der Kommandozeile wird zur Programmierung dieser Option der Parameter /s64 verwendet. Die Bausteinoption CP (entspricht Kommandozeilenparameter /s32) schützt den Programmspeicher gegen Auslesen.

PIC12C5XX, RC-Oszillator Kalibrierung

Bei diesen Bausteinen dient die letzte Speicherstelle des EPROM Programmspeichers der Kalibrierung des RC-Oszillators. Microchip sieht vor, daß diese Speicherstelle mit einem MOVLW-Befehl zum Laden eines Kalibrierungswerts programmiert wird. Bei neuen Bausteinen ist diese Speicherstelle bereits entsprechend programmiert. Der Wert 0c80h entspricht dem Assembler Befehl

MOVLW 080h. Dieser Befehl wird nach einem Reset als erster Befehl ausgeführt und der program counter springt danach auf 0000h um. In der **hed.chip** Software ist diese Besonderheit berücksichtigt. Trotzdem müssen noch einige Details beachtet werden:

1. Bei UV-löschbaren Bausteinen wird diese Speicherstelle ebenfalls gelöscht und muß anschließend mit einem sinnvollen Wert, z.B. 0c80h = MOVLW 080h, programmiert werden. Falls gewünscht kann der Oszillator auch mit einem anderen Wert kalibriert werden.
2. Wenn in der Quelldatei ein Wert für die Kalibrierung des Oszillators enthalten ist, dann wird dieser Wert nur dann programmiert, wenn der Baustein an dieser Stelle vorher keinen Wert enthält. Sie können also in jedem Fall einen Wert in der Quelldatei vorsehen. Wenn Sie einen Microcontroller programmieren, der bereits ab Werk kalibriert ist, dann wird der von Ihnen vorgesehene Wert ignoriert.
3. Wenn Sie einen Vergleich mit einer Datei ausführen, dann wird auch der Wert für die Kalibrierung verglichen. Wenn die Datei an dieser Stelle einen anderen Wert enthält, wird ein Verify-Fehler ausgegeben. Dies gilt nur für den Fall eines expliziten Vergleichs mit einer Datei. Im Falle der Programmierung mit anschließendem, automatischen Verify, wird kein Verify-Fehler erzeugt.
4. Der Befehl MOVLW XX lädt den entsprechende Wert (z.B. 080h) in das W-Register. Es bleibt dem Programm des Controllers überlassen, diese Wert in das OSCCAL-Register, Adresse 05ch zu schreiben.

PIC12C67X, RC-Oszillator Kalibrierung

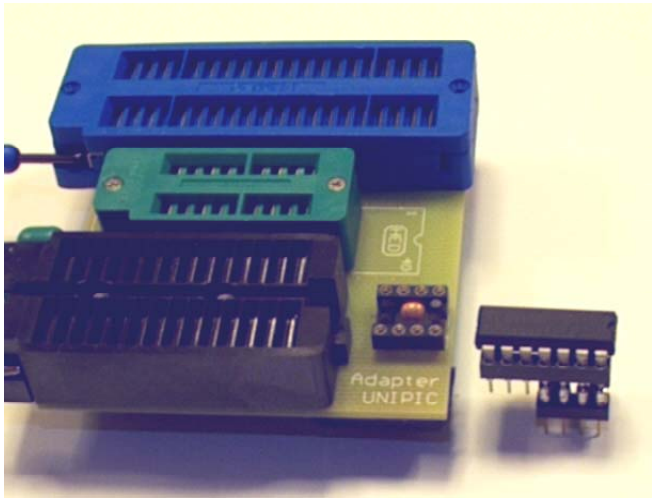
Bei diesen Bausteinen dient die letzte Speicherstelle des EPROM Programmspeichers der Kalibrierung des Oszillators für den Mode INTRC (Baustein Option FOSC1 oder FOSC0+FOSC1). Das Prinzip ist das Gleiche wie für die Bausteine PIC12C5XX. Statt des Befehls MOVLW wird diese Speicherstelle jedoch mit dem Befehl RETLW programmiert.

PIC16F87xA

Für die Bausteine PIC16F873A und PIC16F874A wird das Mnemonic /g16f873a verwendet, für die Bausteine PIC16F876A und PIC16F877A das Mnemonic /g16f876a.

Für den EEPROM Datenspeicher werden die Mnemonics /gpicdata128 bzw /gpicdata256 verwendet. Mit diesen Mnemonics lässt sich der EEPROM Datenspeicher programmieren aber nicht löschen. Löschen ist für die Programmierung nicht erforderlich. Der Leertest lässt sich mit dem Kommandozeilenparameter /n umgehen. In der Benutzeroberfläche HC95 gibt es dafür im Menü „Extras“ den Punkt „Zusätzliche Optionen“. Der EEPROM Datenspeicher wird automatisch gelöscht, wenn der FLASH Programmspeicher gelöscht wird.

PIC16F630/676 in Adapter einsetzen



Diese Baustein in der Bauform DIP14 müssen in den DIP8-Sockel des Adapters UNIPIC bzw. UNIPIC18 eingesetzt werden.

Im Falle des Adapters UNIPIC würde der Baustein dabei gegen den Testsockel DIP28 stoßen. Das kann man vermieden werden, in dem man aus 2 Sockel DIP8 und DIP14 einen Zwischenadapter baut.

Pin 1 des Bausteins muss in Pin 1 des Sockel DIP8 eingesetzt werden.

Siehe Bild.

PIC12F629, PIC16F630 Config Word

Bei diesen Bausteinen sollte das Config-Word nicht über die Quelldatei programmiert werden. Im Config Word dieser Bausteine ist in den Bits 12 und 13 eine vom Hersteller eingestellte „Band Gap Referenz“ enthalten.

Diese Bits würden durch eine Programmierung mit Daten aus der Quelldatei überschrieben. Je nach Inhalt des Config-Word und der Quelldatei kann es auch passieren, dass die Programmierung mit der Meldung „Baustein nicht programmierbar“ abgebrochen wird.

2.5 Toshiba Microcontroller

Mit Hilfe von speziellen Adaptern, die von Toshiba zur Verfügung gestellt werden, lassen sich diese Microcontroller mit Programmieralgorithmen wie sie für EPROMs verwendet werden, programmieren. Zur Zeit werden folgende Typen unterstützt:

Baustein	Bauform	Mnemonic	Adapter, Speicher, Bemerkungen
TC571000AD	DIP32	/gam27c010	Kein Adapter erforderlich, 128kByte
TC571001AD	DIP32	/gam27c010	Kein Adapter erforderlich, 128kByte
TMP88PH40N	DIP28	/gtmp88ph40	Toshiba BM11196, 16kByte
TMP88PH40M	SOIC288	/gtmp88ph40	Toshiba BM11195, 16kByte
TMP88PS43F	P-LQFP80	/gtmp88ps43	Toshiba BM11180A, 64kByte

2.6 Serielle Speicherbausteine

EEPROMs. serielles 2-Draht Interface, I²C

hed.chip programmiert I²C-EEPROMs, angefangen beim 128 Byte 24C01A bis hin zum 32 kByte 24C256. Der Programmieralgorithmus orientiert sich an den Bausteinen von Atmel, Serie AT24C__. Es werden aber keine speziellen Features dieser Bausteine benutzt, so daß auch Bausteine anderer Hersteller programmierbar sein sollten. Die Bausteine vieler Hersteller wurden schon entsprechend getestet und in die Bausteinliste aufgenommen.

Einige I2C-EEPROM verfügen über einen programmierbaren Schreibschutz. Im Falle der SGS Thomson Bausteine der Serien ST24C** und ST25C** wird dieser Schutz durch den Inhalt der zwei letzten Speicherstellen des Bausteins und dem Pegel am Eingang PRE# bestimmt. Um diesen Schutzmechanismus zu benutzen, muß die Quelldatei für diese Speicherstellen die entsprechenden Werte enthalten.

Moderne Bausteine gibt es auch in Versionen für niedrige Versorgungsspannungen. Alle mir bekannten Bausteine können jedoch auch bei 5V programmiert werden. Manche Kunden benutzen **hed.chip** um I2C-EEPROMs in einer Schaltung zu programmieren. Für diese Kunden wurden für einen Teil der I2C-Bausteine Low Voltage Routinen implementiert. In der Bausteinliste sind diese Bausteine als Typen „AT24LV***“ enthalten.

Philips PCF85C-2**

hed.chip unterstützt die Bausteine PCF8582, PCF8594 und PCF8594. Diese I2C-EEPROM sind den entsprechenden Typen der Serie 24C sehr ähnlich. Der einzige Unterschied ist, dass beim Lesen der interne Adress-Zeiger nicht über die 256-Byte Seitengrenze hinaus inkrementiert wird.

EEPROMs, serielles 3-Draht Interface, SPI

hed.chip programmiert die Serien AT25*** von Atmel und X25*** sowie X25F0** von Xicor. Bei diesen Bausteinen kann ¼, ½ oder der ganze Baustein per Software gegen weitere Programmierung geschützt werden. Dazu muß bei der Programmierung der Parameter /s angegeben werden:

/s0	Baustein ungeschützt
/s1	erstes ¼ des Bausteins schreibgeschützt
/s2	erste ½ des Bausteins schreibgeschützt
/s3	ganzer Baustein schreibgeschützt

Um auf diese Weise geschützte Bausteine mit **hed.chip** neu zu programmieren, muß der Baustein zunächst gelöscht werden. Dazu reicht die zusätzliche Angabe des Paramters /e in der Befehlszeile. SPI-EEPROMs anderer Hersteller sind in Vorbereitung.

EEPROMs, Microwire-Interface

hed.chip unterstützt die EEPROMs 93C06, 93C46, 93C56 und 93C66 aus dieser Serie. Es wird der Adapter SERMEM benötigt.

Weitere Typen sind in Vorbereitung. In der Bausteinliste sind zunächst nur die bereits getesteten Typen aufgelistet. Es müssten sich aber alle Typen von allen Herstellern programmieren lassen.

FPGA-Configuration Memories der Serie AT17C***

werden mit dem Adapter SERMEM programmiert. In diesen Adapter können Bausteine in den Bauformen DIP8 eingesetzt werden. Für andere Bauformen müssen lediglich die Pins der Bauform PLCC20 bzw. SOIC20 mit den entsprechenden Pins der Bauform DIP8 verbunden werden. Die Atmel Application Note zur Programmierung dieser Bausteine liefert alle erforderlichen Informationen. **hed.chip** erkennt automatisch den eingesetzten Baustein. Die Polarity Option wird abhängig von vier Byte in der Quelldatei programmiert. „active LOW“ bedeutet dabei, daß OE bei LOW active ist.

Baustein	Adresse	Inhalt	Polarität	Option
AT17C65	02000h	FF FF FF FF	active LOW	(RESET/oe#)
AT17C65	02000h	00 00 00 00	active HIGH	(reset#/OE)
AT17C128	04000h	FF FF FF FF	active LOW	(RESET/oe#)
AT17C128	04000h	00 00 00 00	active HIGH	(reset#/OE)
AT17C256	08000h	FF FF FF FF	active LOW	(RESET/oe#)
AT17C256	08000h	00 00 00 00	active HIGH	(reset#/OE)

Wenn die Quelldatei an diesen Adressen andere Werte enthält, bleibt die Polarity Option unverändert. **hed.chip** meldet dann beim Verify einen Fehler ab Adresse 2000h, 4000h, bzw. 8000h. Die gleiche Fehlermeldung erscheint, wenn man bei der Programmierung über die Kommandozeile eine andere Polarität erzwingt als in der Quelldatei vorgegeben und anschließend mit dem Befehlsparameter /v einen Vergleich durchführt.

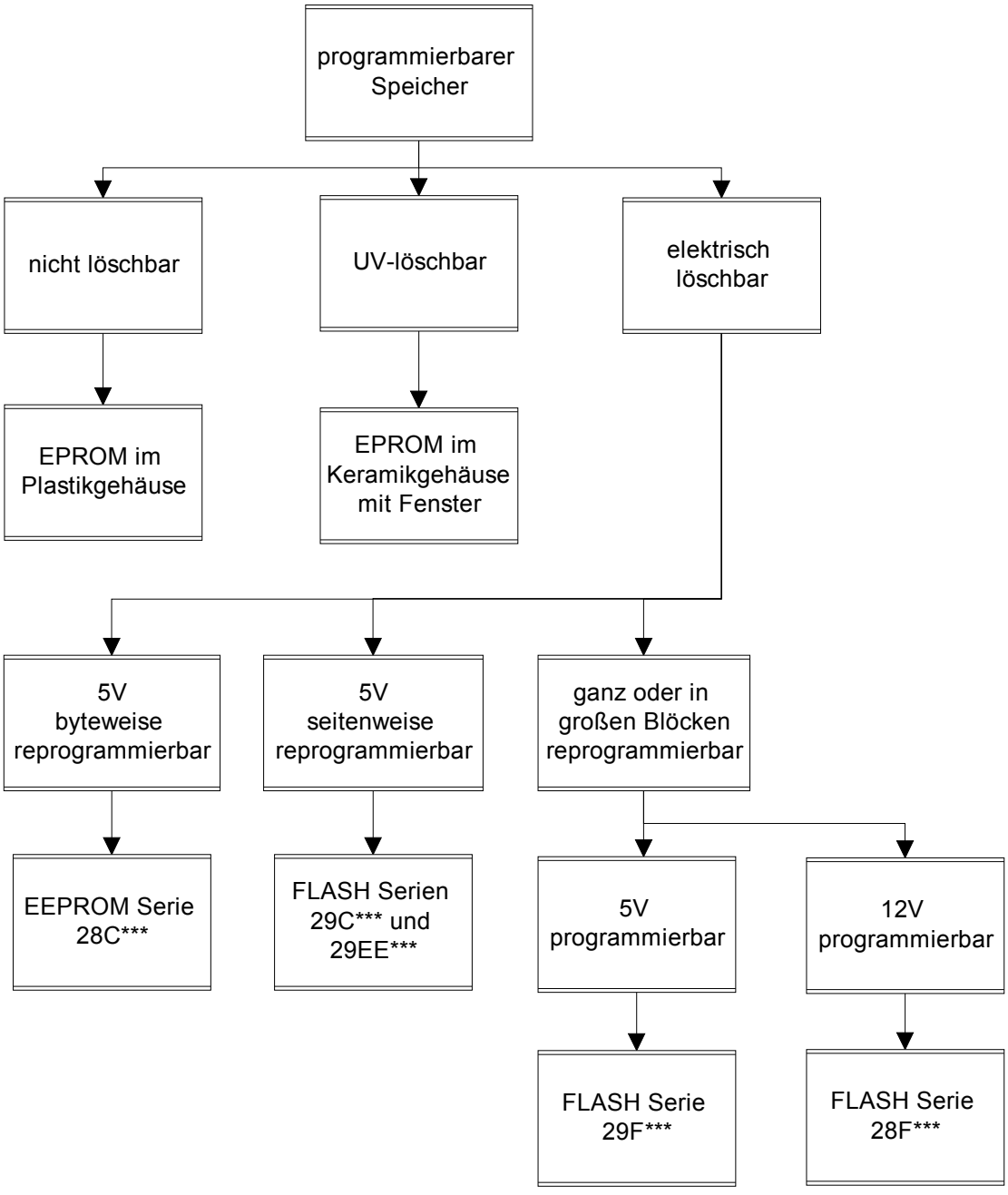
Die Polarity Option kann auch durch zusätzliche Angabe von /s1 bzw. /s2 in der Kommandozeile bestimmt werden. Dabei haben Angaben in der Kommandozeile Vorrang vor Angaben in der Quelldatei.

S0:	Polarität wird durch Quelldatei bestimmt (Default)
S1:	Polarität active LOW (RESET/oe#)
S2:	Polarität active HIGH (reset#/OE)

Werden serielle Speicherbausteine gelöscht, wird der gesamte Baustein mit 0FFh beschrieben und im Falle der AT17C-Serie die Polarity Option auf active LOW eingestellt. Ein Löschvorgang ist vor der Programmierung nicht erforderlich.

2.7 Parallele Speicherbausteine

hed.chip programmiert EPROMs von 8 kByte bis 512 kByte sowie EEPROMs und FLASH-PEROM von 0.5kByte bis 512kByte.



hed.chip benutzt das Programm HEXBIN.EXE zur Umwandlung einer HEX-Datei in eine Binärdatei. Die Angabe eines Offsets wird nicht unterstützt. Wenn zum Beispiel ein EPROM programmiert werden soll, das später von einem Prozessor ab Adresse 08000h gesehen wird, dann muß die Umwandlung Intel-HEX --> Binär vom Anwender selbst durchgeführt werden, oder es muß mit geeigneten Assembleranweisungen dafür gesorgt werden, daß die HEX-Datei keinen Offset enthält. Andernfalls würde eine Binärdatei erzeugt, die zunächst bis zur Adresse 08000h den Wert 0FFh enthält und erst danach die eigentlich zu programmierenden Daten. Die aktuelle Version des Pro-

gramms HEXBIN.EXE unterstützt keine segmentierte Adressierung. In diesem Fall muß die Konvertierung HEX → Binär vom Anwender mit geeigneten Werkzeugen selbst durchgeführt werden.

EPROMs

Die Hersteller von EPROMs weisen ausdrücklich darauf hin, daß die genaue Einhaltung der Spezifikationen für eine optimale Programmierbarkeit und den langfristigen Erhalt der Daten unbedingt notwendig ist. Obwohl sich die Programmieralgorithmen gleichen, hat doch jeder Hersteller eigene Vorstellungen, wie EPROMs programmiert werden sollten. Der erste Schritt vor der Programmierung sollte also immer ein Blick in die Bausteinliste sein, um das jeweils richtige Bausteinmnemonic zu ermitteln.Dann wird **hed.chip** die Programmierung genau gemäß den Spezifikationen des Herstellers durchführen.

Es wurden von allen aufgeführten Herstellern die neuesten Algorithmen verwendet. Dabei fiel auf, daß sich in einigen Fällen die Typbezeichnung in den letzten 10 Jahren nicht geändert hat, obwohl alte Datenbücher andere Algorithmen mit meist längeren Pulszeiten verwenden. Den damals üblichen Algorithmen entsprechen am ehesten die Algorithmen für die Bausteine M2764A, M27128A und M27256 von SGS Thomson. Die zugehörigen BausteinMnemonics sind: s2764, s27128 und s27256. Diese Mnemonics sollten verwendet werden, wenn sehr alte Bausteine programmiert werden sollen.

Mit dem Adapter PLCC32 können einige, aber nicht alle EPROMs in der Bauform PLCC32 programmiert werden. Welche Typen in der Bauform PLCC32 programmiert werden können, ist der Bausteinliste zu entnehmen.

EPROMs 2708, 2716 und 2732

Diese Typen werden von dem Programmiergerät hed.chip nicht unterstützt. Für die Programmierung werden Spannungen über 20V und teils mehrere, unterschiedliche Versorgungsspannungen benötigt. Als Ersatz für 2716 und 2732 können in den meisten Fällen die CMOS-Versionen 27C16 und 27C32 eingesetzt werden. Die Pinbelegung muß im Einzelfall an Hand der Hersteller Datenblätter überprüft werden. Vom Standard abweichend war z.B. der Typ TMS2716 von Texas Instruments.

EPROMs mit UV-Licht löschen

EPROMs werden mit UV-Licht gelöscht. Theoretisch kann man ein EPROM dazu einfach in die Sonne legen. Das dauert dann ca. 2 Wochen und das Ergebnis entspricht nicht den Spezifikationen des Herstellers. Richtig löscht man ein EPROM mit einem speziellen Löschgerät. Die Löschzeit beträgt dann ca. 15 Minuten. Wichtig ist dabei, dass man die Löschzeit nicht unterschreitet. Typisch für unzureichend gelöschte EPROMs ist, wenn der Leertest manchmal versagt oder bei einem mehrfachen Vergleich an unterschiedlichen Stellen Abweichungen festgestellt werden.

EEPROMs 28C-Serie

EEPROMs sind grundsätzlich byteweise programmierbar. Die größeren Typen gestatten meist auch die Programmierung mehrerer Bytes einer Seite in einem Schreibvorgang. Soweit vorhanden wird diese Möglichkeit der seitenweisen Programmierung zur Steigerung der Geschwindigkeit benutzt. Außerdem programmiert **hed.chip** - soweit vorhanden - den Schreibschutz dieser Bausteine, wenn in der Kommandozeile bei der Programmierung der Parameter /s1 zusätzlich angegeben wird.

Bei der seitenweisen Programmierung und zur Aktivierung des Schreibschutzes muß das jeweils folgende Byte in einer bestimmten max. Zeitspanne geschrieben werden. In einer DOS-Task eines Multitasking Systems wie z.B. Windows 95 oder Windows NT 4.0 ist das in der Regel möglich, kann aber nicht unter allen Umständen gewährleistet werden. Die DOS-Task sollte dazu im Vollbild Modus ausgeführt werden. Alle Einstellmöglichkeiten sind auf maximale Geschwindigkeit hin zu optimieren. Druckertreiber, die den gleichen Druckerport bedienen, sollten entfernt werden. Die

Programmierung sollte unbedingt mit der Option /v verifiziert werden. Falls dabei wiederholt Fehler festgestellt werden, muß zur Programmierung dieser Bausteine der Rechner mit DOS gebootet werden.

Beispiel: Ein AT28C256 soll programmiert, verifiziert und der Schreibschutz aktiviert werden:

```
hedchip lpt2 /g28c256 /p /v /s1 myapp.hex          ; 28C256 in DIP28
```

Der Schreibschutz dieser Bausteine ist eine sinnvolle Eigenschaft. In der Anwenderschaltung schützt er zuverlässig vor unbeabsichtigten Schreibvorgängen. Solche Schreibvorgänge können zum Beispiel während des Einschaltens der Versorgungsspannung entstehen.

Normalerweise ist es nicht nötig, ein EEPROM vor der Programmierung zu löschen. Der Schreibschutz aber wird nur aufgehoben, wenn der Baustein gelöscht wird:

```
hedchip /g28c256 /e
```

Nicht-flüchtiger RAM-Speicher

Hierbei handelt es sich um RAM in einem Modul mit Stützbatterie. Der Vorteil solcher Bausteine ist, dass der Speicher unbegrenzt oft wiederbeschreibbar ist. Es gibt diese Baustein mit und ohne eingebauter Uhr.

Bei Bausteinen mit eingebauter Uhr sollte unbedingt folgendes beachtet werden:

- Diese Bausteine sollte nicht gelöscht werden. Das ist bei diesen Bausteinen für eine Wiederprogrammierung ohnehin nicht erforderlich.
- Der Speicherbereich mit den Registern für die Uhr sollte nicht überschrieben werden. Bei einem SGS Thomson M48T18 sollte die Quelldatei also nicht größer als 01FF8h Bytes sein.

Die Register, die vom Baustein für die Uhr benutzt werden, können mit **hed.chip** wie normale Speicherzellen beschrieben werden. Dabei wird allerdings keine Rücksicht genommen, welche Auswirkungen das auf die Uhr hat.

FLASH-ROM 29C- und 29EE-Serie

Diese mit 5V programmierbaren FLASH sind EEPROMs sehr ähnlich. Der Unterschied besteht darin, daß immer alle Bytes einer Seite geschrieben werden müssen. Nicht beschriebene Bytes einer Seite werden beim internen Schreibvorgang des Bausteins gelöscht. Eine Seite besteht bei den meisten Typen aus 128 Bytes.

hed.chip programmiert diese Bausteine von einer Reihe von Herstellern. Weitere werden zur Zeit getestet bzw. in die Programmiersoftware implementiert. **hed.chip** wertet die Baustein-ID dieser Bausteine aus und stellt daraufhin automatisch die Größe des Bausteins und die Programmierparameter ein. Nicht bekannte Bausteine, oder wenn der Hersteller-Code nicht mit dem angegebenen Mnemonic übereinstimmt, werden abgewiesen. Die Liste der dem **hed.chip** bekannten Bausteine wird laufend erweitert.

Der allgemeine Schreibschutz dieser Bausteine kann durch Angabe des Parameters /s1 bei der Programmierung aktiviert werden. Es gelten die gleichen Hinweise wie bei der seitenweisen Programmierung von EEPROMs.

Beispiel: Programmierung Atmel AT29C010, Schreibschutz aktivieren

```
hedchip /ga29cxxx /p /v /e /s1 myapp.bin
```

Dabei haben die zusätzlichen Parameter folgende Wirkung:

/v	Die Programmierung und der Löschvorgang werden verifiziert.
/e	Der Baustein wird gegebenenfalls vor der Programmierung gelöscht. Ohne diese Angabe könnte ein Baustein mit aktiviertem Schreibschutz nicht programmiert werden.
/s1	Der Schreibschutz des Bausteins wird nach der Programmierung aktiviert.

FLASH-Rom mit Bootblock Schreibschutz

Betrifft zur Zeit: Atmel AT29C010A, AT29C020, AT29C040/A, Winbond W29C020, W29C040

Durch Angabe des Parameters /sn bei der Programmierung können bei diesen Bausteinen je ein Speicherbereich am Speicheranfang und am Speicherende gegen weitere Programmierung geschützt werden. Dies ist sinnvoll, wenn der Baustein später im System neu programmiert werden kann und sichergestellt werden soll, daß ein Boot Block mit unverzichtbaren Routinen auf jeden Fall erhalten bleibt.

Über den Parameter /s kann jede Kombination von Softwareschutz (SDP) und Boot Block Lock eingestellt werden:

/s1	SDP (allgemeiner, reversibler Schreibschutz für den ganzen Baustein)
/s2	Lower Address Boot Block Lock (LABBL)
/s3	SDP + LABBL
/s4	Higher Address Boot Block Lock (HABBL)
/s5	SDP + HABBL
/s6	LABBL + HABBL
/s7	SDP + LABBL + HABBL

Die Angabe der verschiedenen Schreibschutzmechnismen kann auch einzeln erfolgen. Die beiden folgenden Befehlszeilen sind in ihrer Wirkung identisch und aktivieren beide Boot Block Locks und den reversiblen Schreibschutz SDP:

```
hedchip /ga29cxxx /p /v /e /s7 myapp.bin
hedchip /ga29cxxx /p /v /e /s1 /s2 /s4 myapp.bin
```

Der Schreibschutz der Boot Blöcke, LABBL und HABBL, kann nicht wieder aufgehoben werden. Die weitere Programmierung eines auf diese Weise geschützten Bausteins ist möglich, erfordert jedoch einige Umstände:

1. Wenn der Schreibschutz des ganzen Bausteins, SDP, aktiviert ist, muß der Baustein mit dem Löschbefehl gelöscht werden. Dabei wird nur der Schreibschutz SDP aufgehoben. Der Baustein selbst kann nicht gelöscht werden.
2. Falls das Lower Address Boot Block Lock (LABBL) aktiviert ist, muß der Baustein zunächst gelesen werden. Bei der anschließenden Programmierung muß der Speicherbereich des Lower Address Boot Block mit den zuvor gelesenen Daten wieder beschrieben werden.

Erklärung: **hed.chip** kann Bausteine nur kontinuierlich beginnend ab Adresse 00000h programmieren. Da der gesperrte Speicherbereich nicht verändert werden kann, muß vermieden werden, daß **hed.chip** bei dem Versuch feststellt, daß der Baustein in diesem Bereich nicht mehr programmierbar ist.

Winbond Serie 29EE und 29C

Einige Typen aus dieser Serie werden mit aktiviertem Schreibschutz geliefert. Um den Schreibschutz zu deaktivieren, müssen diese Bausteine vor der ersten Programmierung gelöscht werden.

FLASH, Serie 29F

Diese Bausteine können wie die zuvor beschriebenen Bausteine der Serien 29C und 29EE ohne erhöhte Programmierspannung programmiert werden. Sie müssen jedoch vor jeder Programmierung gelöscht werden. Der Sektor-Schreibschutz dieser Bausteine kann mit **hed.chip** weder aktiviert noch deaktiviert werden.

hed.chip wertet die Baustein-ID dieser Bausteine aus und stellt automatisch die Größe des Bausteins ein und benutzt die richtigen Parameter zur Programmierung.

Zur Programmierung aller Bausteine dieser Serie wird das Bausteinmnemonic 29fxxx benutzt:

```
hedchip /g29fxxx /p/v/e your_app.bin          ; löscht, programmiert und verifiziert 29F-FLASH
```

FLASH, 49F-Serie

Diese FLASH entsprechen weitgehend den Bausteinen der Serie 29F. Sie verfügen über einen Boot-Block, der gegen Überschreiben geschützt werden kann. Der Boot-Block kann durch Angabe des Kommandozeilenparameters /s2 geschützt werden.

Einige Typen verfügen über einen Eingang RESET. RESET hat folgende Funktionen:

- RESET = low: Alle Ausgänge des Bausteins sind hochohmig.
- RESET = high: Normaler Lese- und Schreibzugriff möglich.
- RESET = 12V: Boot-Block kann trotz aktiviertem Schreibschutz gelöscht und beschrieben werden. Ein einmal aktivierter Boot-Block Schreibschutz wird dadurch nicht aufgehoben sondern nur zeitweilig umgangen.

Für die Bausteine ohne RESET-Pin werden die Bausteinmnemonics 29fxxx bzw. 29lvxxx benutzt. Bausteine mit RESET-Pin werden mit dem Mnemonics 49f00x bzw. 49lv00x programmiert.

FLASH 28F-Serie

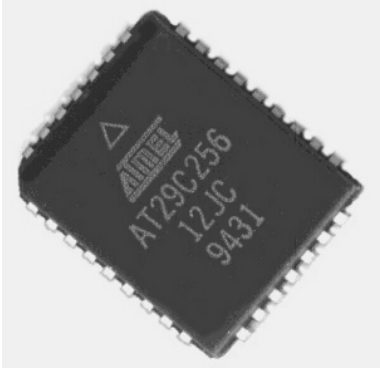
Diese FLASH benötigen zur Programmierung eine Programmierspannung von 12V. Sie müssen vor der Programmierung gelöscht werden. Das Bausteinmnemonic 28fxxx wird für alle Bausteine dieser Serie verwendet. **hed.chip** wertet die Baustein-ID des Bausteins aus und stellt automatisch die Größe des Bausteins ein und wählt den richtigen Programmieralgorithmus. Beispiel:

```
hedchip /g28fxxx /p/v/e your_app.bin          ; löscht, programmiert und verifiziert 28F-FLASH
```

FLASH Intel 28F001B

Dieser Baustein hat einen durch Hardware geschützten 8kByte BootBlock. Beim 28F001BX-T liegt dieser Block an der Adresse 01E000h, beim 28F001BX-B an der Adresse 0. Der BootBlock kann nur beschrieben bzw. gelöscht werden, wenn an RP# (Pin 30) 12V anliegt. Dazu muß folgendes gemacht werden: Mit einer Drahtbrücke muß Pin 1 des Bausteins mit Pin 30 verbunden werden. Leertest und Verify sollten ohne diese Drahtbrücke durchgeführt werden.

Speicherbausteine in der Bauform PLCC32



Grundsätzlich wird dafür ein Adapter 1:1 von DIP32 auf PLCC32 verwendet. Die Software des **hed.chip** wurde so ausgelegt, daß damit die PLCC32-Versionen von 24, 28 und 32poligen DIP-Bausteinen programmiert werden können. Eine Ausnahme gibt es allerdings: Für EPROM 27C512 in PLCC32 wird der Adapter PLCC32_28 benötigt.

Die Bausteinliste gibt detaillierte Auskunft, welche Bausteine in der Bauform PLCC32 unterstützt werden. Für die Programmierung dieser Bausteine sind je nach Typ andere Mnemonics als für die DIP-Versionen notwendig.

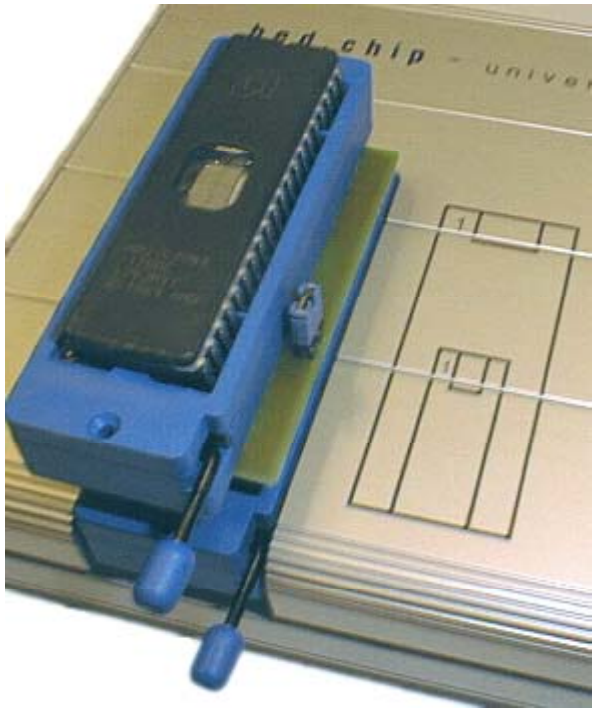
Beispiel für den Baustein 28C256

Ein 28C256 in der Bauform PLCC32 soll gelöscht, programmiert, verifiziert und anschließend schreibgeschützt werden. Weil die Pinbelegung der 32poligen PLCC-Bauform von der 28poligen DIP-Bauform abweicht, muß das Bausteinmnemonic /g28c256plcc verwendet werden.

```
hedchip /g28c256plcc /p/v/e/s1 myapp.hex          ; nicht: /g28c256
```

16-Bit Speicherbausteine in der Bauform DIP40

Für die Programmierung dieser Baustein ist der Adapter MEM16_DIP40 erforderlich.



Für Speicherbausteine muss der Jumper auf dem Adapter gesetzt werden. Das heißt, auf die 2-polige Stiftliste wird ein Steckschuh gesteckt.

Bitte beachten Sie:

- Orientierung des Bausteins im Adapter. (Hier: Am27C2048)
- Gesetzter Steckschuh auf Jumper
- Orientierung Adapter MEM16_DIP40 im Programmiergerät.

Low Voltage

hed.chip ist auch in der Lage Low Voltage Bausteine zu programmieren. Low Voltage Bausteine, die auch mit 5V betrieben werden können, sind bereits in der Bausteinliste aufgeführt. Bausteine, die nur bei 3.3V betrieben werden können, werden nach und nach implementiert und in die Bausteinliste aufgenommen. Kundenwünsche nach der Programmierung bestimmter Bausteine werden bevorzugt behandelt.

3 DOS-Returncodes

Diese Werte werden von dem Programm HEDCHIP.EXE beim Beenden an das Betriebssystem zurückgegeben. Sie können in Batchprogrammen für Verzweigungen benutzt werden. Siehe dazu Kapitel 1.11.

Code	Bedeutung
000	Funktion fehlerfrei ausgeführt. Im Falle von Leertest oder Vergleich: „leer“ bzw. „gleich“.
001	vorzeitiges Ende der Befehlszeile, der Hilfstext wurde angezeigt.
002	in der Befehlszeile fehlt die Angabe des BausteinMnemonics. Es wurde eine der Bau-steinmnemonic-Listen angezeigt.
003	in der Befehlszeile fehlt die Angabe der Schnittstelle.
004	in der Befehlszeile fehlt die Angabe des Befehlsparameters (z.B. /p für Programmieren).
005	in der Befehlszeile fehlt die Angabe des Dateinamens der Quelldatei für die Befehle /v oder /p bzw. der Zieldatei für den Befehl /r.
006	in der Befehlszeile ist ein ungültiger Parameter.
007	in der Befehlszeile wurde eine nicht vorhandene Schnittstelle angegeben.
008	das Unterprogramm JEDECASM.COM oder HEXBIN.EXE konnte nicht gestartet werden. Mögliche Ursache: nicht genügend Speicher oder das Unterprogramm wurde nicht gefunden.
009	das Unterprogramm zur Umwandlung der Quelldatei in eine Binärdatei, JEDECASM.COM oder HEXBIN.EXE, hat einen Fehler gemeldet.
010	HEDCHIP.EXE hat vom Benutzer eine Entscheidung verlangt und es wurde die Wahl „Abbrechen“ bzw. „Nicht fortfahren“ getroffen.
011	Dateifehler. Eine Datei konnte nicht geöffnet, nicht gelesen oder nicht geschrieben werden. Mögliche Ursachen: Die Datei wurde nicht gefunden, die Datei ist schreibgeschützt oder sie wird von einem anderen Programm benutzt. Der Datenträger ist voll.
012	falsches Betriebssystem. HEDCHIP.EXE hat festgestellt, daß der Rechner mit einem nicht geeigneten Betriebssystem gefahren wird.
013	Initialisierung Betriebssytemerweiterung fehlgeschlagen. Unter Windows NT konnte diese Erweiterung entweder nicht geladen werden oder bei der Initialisierung wurde ein Fehler festgestellt.

Code	Bedeutung
129	Es wurde ein Leertest oder ein Vergleich ausgeführt. Das Ergebnis ist „nicht leer“ bzw. „nicht gleich“. Dieser Code wird auch geliefert, wenn der Baustein gegen Auslesen geschützt ist oder der Programmer den Baustein nicht erkennt und daher vermutet, daß es sich um einen gegen Auslesen geschützten Baustein handeln könnte. (Bei „leer“ oder „gleich“ wird der Code 000 geliefert)
130	Der Baustein sollte gelöscht werden. Dabei wurde ein Fehler festgestellt, oder ein anschließend durchgeführter Leertest hat „nicht leer“ ergeben.
131	Der Baustein sollte programmiert werden. Dabei wurde ein Fehler festgestellt oder ein anschließender Vergleich hat „nicht gleich“ ergeben.
132	Securityfuse, Lockbits bzw. Schreibschutz konnten nicht programmiert werden. Dieser Fehler tritt teilweise auch auf, wenn der Baustein nicht über eine solche Funktion oder den angegebenen Schutzlevel verfügt.
133	Der Baustein sollte in eine Datei ausgelesen werden. Der Baustein konnte jedoch nicht ausgelesen werden.
134	Der Baustein konnte nicht identifiziert werden. In einem solchen Fall wird der Benutzer aufgefordert, den Baustein einzusetzen und eine Taste zu drücken. Wenn der dann nochmal durchgeführte Identifizierungsversuch scheitert, wird dieser Code geliefert. Bei einigen Bausteinen (z.B. Atmel AT89Cxx-Controller) wird statt dessen die Möglichkeit geboten, ohne Identifizierung fortzufahren.
135	Der Baustein sollte in eine Datei ausgelesen werden. Das war nicht möglich, weil der Baustein gegen Auslesen geschützt ist. Lesegeschützte Bausteine können weder ausgelesen noch kopiert werden.
255	Der Programmer reagiert nicht auf Befehle. Ähnlich wie ein Drucker muß der Programmer jeden Befehl mit Acknowledge quittieren. Dieser Fehlercode tritt auf, wenn der Programmer nicht angeschlossen ist oder nicht mit Strom versorgt wird.

4 Zubehör

Für SMD-Bausteine oder Bausteine mit besonderen Anforderungen an den Programmer werden Adapter benötigt. Diese Liste ist eine Auswahl zur Zeit verfügbarer Adapter, jeweils mit einer kurzen Beschreibung. Weitere werden kurzfristig erstellt, wenn sie von den Benutzern nachgefragt werden.

Adapterbezeichnung	Beschreibung
Adapter DIP40_MEM16	Spezialadapter für 16Bit Speicherbausteine in der Bauform DIP40, z.B. M27C4002 und für einige AT90S-Microcontroller mit AD-Wandler, z.B. AT90S8535-8PC
Adapter DIPMEM	Spezialadapter für diverse Bausteine. Mit DIP40 Testsockel. **hed.chip** in der Hardware-Version 2 benötigt diesen Adapter nur für EPROMs 27C16. Ältere **hed.chip** benötigen diesen Adapter für EPROMs 27C16, 27C32, 27(C)512 und 27C080 sowie für Atmel AT90S in der Bauform DIP40.
Adapter MEM16_DIP40	Spezialadapter für Speicherbausteine in der Bauform DIP40 und einige Atmel Microcontroller mit AD-Wandler, z.B. AT90S8535. Mit Testsockel DIP40.
Adapter PLCC20	für PLD-Bausteine in der Bauform PLCC20. Mit PLCC20 Testsockel.
Adapter PLCC2500	Spezialadapter für die Bausteine Atmel ATV2500H und ATV2500B. Mit PLCC44 Testsockel.
Adapter PLCC28	für Bausteine in der Bauform PLCC28. Mit PLCC28 Testsockel.
Adapter PLCC28_24	für PLD-Bausteine in der Bauform PLCC28, die sonst als DIP24 zu bekommen sind.. Mit PLCC28 Testsockel.
Adapter PLCC32	für parallele Speicherbausteine (EPROMs, FLASH, etc.). Dieser Adapter überträgt die Pins für DIP32-Bausteine 1:1 auf die Bauform PLCC. Die Software wurde so gestaltet, dass auch Bausteine, deren DIP-Bauform 28 Pins hat, mit diesem Adapter programmiert werden können. Mit PLCC32 Testsockel.
Adapter PLCC32/44	Für parallele Speicherbausteine (EPROMs, FLASH) und MCS51 Microcontroller. Mit PLCC32 und PLCC44 Testsockeln.
Adapter PLCC32_28	für parallele Speicherbausteine EPROM 27C512 in der Bauform PLCC32. Dieser Adapter überträgt die Pins für DIP28-Bausteine auf die Bauform PLCC32. Die meisten PLCC32-Bausteine können mit dem 1:1 PLCC32 Adapter programmiert werden (siehe oben). 27C512 ist da eine Ausnahme. Mit PLCC32 Testsockel.
Adapter PLCC44	für MCS51 Microcontroller in der Bauform PLCC44. Mit PLCC44 Testsockel.
Adapter PLCC52	für Dallas DS87C530. Mit normalem Lowcost PLCC52 Sockel.

Adapterbezeichnung	Beschreibung
Adapter PLCC68_40	für MCS51 Microcontroller in der Bauform PLCC68. Mit PLCC68 Testsockel.
Adapter PLCC750	Spezialadapter für die Bausteine Atmel AT22V10, ATV750/L und ATV750B in der Bauform PLCC28. Mit PLCC28 Testsockel.
Adapter PQPF44_C505	Spezialadapter für Siemens/Infineon C505A und C505CA.
Adapter SERMEM	Adapter mit Sockel DIP8 für bestimmte serielle EEPROM. Gleiche Platine wie Adapter SOIC20. Der SOIC20 Testsockel des Adapters SOIC20 kann nachträglich bestückt werden.
Adapter SOIC20	für PLD-Bausteine und MCS51 Microcontroller. Dieser Adapter überträgt die Pins für DIP20 Bausteine 1:1 auf die Bauform SOIC20. Mit SOIC20 Testsockel. Gleiche Platine wie Adapter SERMEM. Der DIP8 Sockel des Adapters SERMEM kann nachträglich bestückt werden.
Adapter TQFP44	für MCS51 Microcontroller in der Bauform TQFP44, z.B. Atmel AT89C51-20AC. Mit TQFP44 Testsockel.
Adapter TSOP32	für parallele Speicherbausteine (EPROMs, FLASH, etc.). Dieser Adapter überträgt die Pins für DIP32-Bausteine 1:1 auf die Bauform TSOP32. Mit TSOP32 Testsockel.
Adapter UNIPIC	Spezialadapter für PIC Microcontroller von Microchip. Mit Testsockeln DIP18, DIP28 und DIP40, Präzisionssockel DIP8.
Adapter UNIPIC18	Spezialadapter für PIC Microcontroller von Microchip. Mit Testsockel DIP18 und Präzisionssockel DIP8. Testsockel DIP28 und DIP40 nicht im Lieferumfang enthalten. Diese Testsockel können bei Bedarf selbst nachträglich bestückt werden.
Adapter VSOP32	für parallele Speicherbausteine (EPROMs, FLASH, etc.). Dieser Adapter überträgt die Pins für DIP32-Bausteine 1:1 auf die Bauform VSOP32. Mit VSOP32 Testsockel. VSOP ähnelt TSOP, ist aber etwas kleiner.
Internationales Steckernetzteil	Geeignet für die Netzsteckdosen in den USA, Großbritannien und Deutschland. Großer Eingangsspannungsbereich: 100 bis 240VAC, 50 oder 60 Hz.

5 Bausteinliste

Version 3.26 vom 06.03.2008

Diese Liste gibt an:

- Welche Bausteine mit **hed.chip** programmiert werden können.
- Gegebenenfalls: welcher Adapter verwendet werden muß. Hinweis: Die Bemerkung „auf Anfrage“ bezieht sich immer auf aktuelle Zeile. Beispiel: AMD Am27C020 in der Bauform DIP32 kann programmiert werden, ein passender Adapter für die Bauform PLCC32 kann auf Anfrage erstellt werden.
- HEDCHIP.EXE: Welches Bausteinmnemonic in der Befehlszeile angegeben werden muß.

Hersteller:	AMD			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
Am27C010	DIP32	/gam27c010		
	PLCC32	/gam27c010	PLCC32	
Am27C020	DIP32	/gam27c020		
	PLCC32			auf Anfrage
Am27C040	DIP32	/gam27c040		
	PLCC32	/gam27c040	PLCC32	
Am27C1024	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
Am27C128	DIP28	/gam27c128		
Am27C2048	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
Am27C256	DIP28	/gam27c256		
	PLCC32	/gam27c256	PLCC32_28	
Am27C4096	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
Am27C512	DIP28	/gam27c512_2		
	PLCC32	/gam27c512_2	PLCC32_28	
Am27C64	DIP28	/gam27c64		
Am27LV010/B	DIP32	/gam27c010		
	PLCC32	/gam27c010	PLCC32	
Am27LV020/B	DIP32	/gam27c020		
	PLCC32			auf Anfrage
Am28F010	DIP32	/g28fxxx		
	PLCC32	/g28fxxx	PLCC32	
Am28F010A	DIP32	/g28fxxx		
	PLCC32	/g28fxxx	PLCC32	
Am28F020	DIP32	/g28fxxx		
	PLCC32	/g28fxxx	PLCC32	
Am28F020A	DIP32	/g28fxxx		
	PLCC32	/g28fxxx	PLCC32	
Am28F256	DIP32	/g28fxxx		
	PLCC32	/g28fxxx	PLCC32	
Am28F256A	DIP32	/g28fxxx		
	PLCC32	/g28fxxx	PLCC32	
Am28F512	DIP32	/g28fxxx		
	PLCC32	/g28fxxx	PLCC32	
Am28F512A	DIP32	/g28fxxx		
	PLCC32	/g28fxxx	PLCC32	
Am29F002B(B/T)	DIP32	/g29f00x		Sector Protection nicht programmierbar
	PLCC32	/g29f00x	PLCC32	
	TSOP32	/g29fxxx	TSOP32	

Hersteller:	AMD			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
Am29F002N(B/T)	DIP32	/g29fxxx		Sector Protection
	PLCC32	/g29fxxx	PLCC32	nicht programmierbar
	TSOP32	/g29fxxx	TSOP32	
Am29F010	DIP32	/g29fxxx		Sector Protection
	PLCC32	/g29fxxx	PLCC32	nicht programmierbar
	TSOP32	/g29fxxx	TSOP32	
Am29F040	DIP32	/g29fxxx		Sector Protection
	PLCC32	/g29fxxx	PLCC32	nicht programmierbar
	TSOP32	/g29fxxx	TSOP32	
Am29LV010B	PLCC32	/g29lvxxx	PLCC32	Sector Protection nicht programmierbar
Am29LV040B	PLCC32	/g29lvxxx	PLCC32	Sector Protection nicht programmierbar
N87C52T2	DIP40	/gam87c5x		
	PLCC44	/gam87c5x	PLCC44	
PALCE16V8H/Q	DIP20	/gam16v8		
PALCE20V8H/Q	DIP24	/gam20v8		
	PLCC28	/gam20v8	PLCC28_24	
PALCE22V10H/Q	DIP24	/gam22v10		nur -PC4 und -PC5
	PLCC28	/gam22v10	PLCC28_24	

Hersteller:	Amic			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
A29001	DIP32	/g29fxxx		
	PLCC32	/g29fxxx	PLCC32	
	TSOP32	/g29fxxx	TSOP32	
A290011	DIP32	/g29fxxx		
	PLCC32	/g29fxxx	PLCC32	
	TSOP32	/g29fxxx	TSOP32	
A29002	DIP32	/g29fxxx		
	PLCC32	/g29fxxx	PLCC32	
	TSOP32	/g29fxxx	TSOP32	
A290021	DIP32	/g29fxxx		
	PLCC32	/g29fxxx	PLCC32	
	TSOP32	/g29fxxx	TSOP32	
A29010	DIP32	/g29fxxx		
	PLCC32	/g29fxxx	PLCC32	
	TSOP32	/g29fxxx	TSOP32	
A29040A	DIP32	/g29fxxx		
	PLCC32	/g29fxxx	PLCC32	
	TSOP32	/g29fxxx	TSOP32	

Hersteller:	ASD			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
AE29F1008	DIP32	/ga29cxxx		
	PLCC32	/ga29cxxx	PLCC32	
AE29F2008	DIP32	/ga29cxxx		
	PLCC32	/ga29cxxx	PLCC32	

Hersteller	Atmel			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
AT17C128	DIP8	/ga17cxxx	SERMEM	
	PLCC20	/ga17cxxx	auf Anfrage	
	SOIC20	/ga17cxxx	auf Anfrage	
AT17C256	DIP8	/ga17cxxx	SERMEM	
	PLCC20	/ga117cxx	auf Anfrage	
	SOIC20	/ga17cxxx	auf Anfrage	
AT17C65	DIP8	/ga17cxxx	SERMEM	
	PLCC20	/ga17cxxx	auf Anfrage	
	SOIC20	/ga17cxxx	auf Anfrage	
AT17C65B	DIP8	/ga17c65b	SERMEM	B-Version hat B in
	PLCC20		auf Anfrage	Datumsangabe
	SOIC20		auf Anfrage	
AT22LV10/L	DIP24	/gat22v10	DIP750	
	PLCC28	/gat22v10	PLCC750	
AT22V10/L	DIP24	/gat22v10	DIP750	
	PLCC28	/gat22v10	PLCC750	
AT22V10B	DIP24	/gat22v10	DIP750	
	PLCC28	/gat22v10	PLCC750	
AT24C01	DIP8	/g24c01		
AT24C01A	DIP8	/g24c01a		
AT24C02	DIP8	/g24c02		
AT24C04	DIP8	/g24c04		
AT24C08	DIP8	/g24c08		
AT24C128	DIP8	/g24xc128		
AT24C16	DIP8	/g24c16		
AT24C164	DIP8	/g24c16		
AT24C256	DIP8	/g24xc256		
AT24C32	DIP8	/g24xc32		
AT24C64	DIP8	/g24xc64		
AT24LV02	DIP8	/g24lv02		Low Voltage, 3.3V
AT24LV128	DIP8	/g24lv128		Low Voltage, 3.3V
AT24LV256	DIP8	/g24lv256		Low Voltage, 3.3V
AT25010	DIP8	/ga25010		
AT25020	DIP8	/ga25020		
AT25040	DIP8	/ga25040		
AT25080	DIP8	/ga25080		
AT25128	DIP8	/g25128		
AT25160	DIP8	/ga25160		
AT25320	DIP8	/ga25320		
AT25640	DIP8	/ga25640		
AT27BV010	PLCC32	/ga27c010	PLCC32	
AT27BV020	PLCC32	/ga27c020	PLCC32	
AT27BV040	PLCC32	/ga27c040	PLCC32	
AT27BV256	DIP28	/ga27c256		
	PLCC32	/ga27c256	PLCC32_28	
AT27BV512	DIP28	/ga27c512_2		
	PLCC32	/ga27c512_2	PLCC32_28	
AT27C010/L	DIP32	/ga27c010		
	PLCC32	/ga27c010	PLCC32	
AT27C020	DIP32	/ga27c020		
	PLCC32	/ga27c020	PLCC32	
AT27C040	DIP32	/ga27c040		
	PLCC32	/ga27c040	PLCC32	

Hersteller	Atmel			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
AT27C080	DIP32	/ga27c080_2		
AT27C1024	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
AT27C2048	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
AT27C256R	DIP28	/ga27c256		
	PLCC32	/ga27c256	PLCC32_28	
AT27C4096	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
AT27C512R	DIP28	/ga27c512_2		
	PLCC32	/ga27c512_2	PLCC32_28	
AT27LV010A	PLCC32	/ga27c010	PLCC32	
AT27LV020A	PLCC32	/ga27c020	PLCC32	
AT27LV040A	PLCC32	/ga27c040	PLCC32	
AT27LV256A	DIP28	/ga27c256		
	PLCC32	/ga27c256	PLCC32_28	
AT27LV512A	DIP28	/ga27c512_2		
	PLCC32	/ga27c512_2	PLCC32_28	
AT28C04	DIP24	/g28c04		
	PLCC32	/g28c04plcc	PLCC32	
AT28C16/E	DIP24	/g28c16		
	PLCC32	/g28c16plcc	PLCC32	
	SOIC24		auf Anfrage	
AT28C17	DIP28	/g28c17		
	PLCC32	/g28c16plcc	PLCC32	
	SOIC28		auf Anfrage	
AT28C256	DIP28	/g28c256		
	PLCC32	/g28c256plcc	PLCC32	
	PGA28		auf Anfrage	
	SOIC28		auf Anfrage	
AT28C64/X	DIP28	/g28c64		
	PLCC32	/g28c64plcc	PLCC32	
	SOIC28		auf Anfrage	
AT28C64B	DIP28	/g28c64b		
	PLCC32	/g28c64bplcc	PLCC32	
	SOIC28			
AT28HC256	DIP28	/g28c256		
	PLCC32	/g28c256plcc	PLCC32	
	PGA28		auf Anfrage	
	SOIC28		auf Anfrage	
AT28HC64B	DIP28	/g28c64b		
	PLCC32	/g28c64bplcc	PLCC32	
	SOIC28		auf Anfrage	
AT29BV010A	DIP32	/ga29lvxxx		
	PLCC32	/ga29lvxxx	PLCC32	
AT29BV020	PLCC32	/ga29lvxxx	PLCC32	
AT29BV040A	TSOP32	/ga29lvxxx	TSOP32	
AT29C010A	DIP32	/ga29cxxx		
	PLCC32	/ga29cxxx	PLCC32	
AT29C020	DIP32	/ga29cxxx		
	PLCC32	/ga29cxxx	PLCC32	
AT29C040	DIP32	/ga29cxxx		
AT29C040A	DIP32	/ga29cxxx		
AT29C256	DIP28	/ga29c256		
	PLCC32	/ga29c256plcc	PLCC32	
AT29C257	PLCC32	/ga29cxxx	PLCC32	

Hersteller	Atmel			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
AT29C512	DIP32	/ga29cxxx		
	PLCC32	/ga29cxxx	PLCC32	
AT29LV010A	DIP32	/ga29lvxxx		
	PLCC32	/ga29lvxxx	PLCC32	
AT29LV020	PLCC32	/ga29lvxxx	PLCC32	
AT29LV040A	PLCC32	/ga29lvxxx	PLCC32	
	TSOP32	/ga29lvxxx	TSOP32	
AT29LV256	DIP28	/ga29lv256		
	PLCC32	/ga29lv256plcc	PLCC32	
AT29LV512	DIP32	/ga29lvxxx		
	PLCC32	/ga29lvxxx	PLCC32	
AT34Cxxx				in Vorbereitung
AT49BV001/T	DIP32	/g49lv00x		mit RESET pin, Low
	PLCC32	/g49lv00x	PLCC32	Voltage
AT49BV001N/T	DIP32	/g29lvxxx		ohne RESET pin,
	PLCC32	/g29lvxxx	PLCC32	Low Voltage
AT49BV002/T	DIP32	/g49lv00x		mit RESET pin, Low
	PLCC32	/g49lv00x	PLCC32	Voltage
AT49BV002N/T	DIP32	/g29lvxxx		ohne RESET pin,
	PLCC32	/g29lvxxx	PLCC32	Low Voltage
AT49BV010	DIP32	/g29lvxxx		Low Voltage
	PLCC32	/g29lvxxx	PLCC32	
AT49BV020	DIP32	/g29lvxxx		Low Voltage
	PLCC32	/g29lvxxx	PLCC32	
AT49BV040/T	DIP32	/g29lvxxx		Low Voltage
	PLCC32	/g29lvxxx	PLCC32	
AT49BV512	DIP32	/g29lvxxx		Low Voltage
	PLCC32	/g29lvxxx	PLCC32	
AT49F001/T	DIP32	/g49f00x		mit RESET pin
	PLCC32	/g49f00x	PLCC32	
AT49F001N/T	DIP32	/g29fxxx		ohne RESET pin
	PLCC32	/g29fxxx	PLCC32	
AT49F002/T	DIP32	/g49f00x		mit RESET pin
	PLCC32	/g49f00x	PLCC32	
AT49F002N/T	DIP32	/g29fxxx		ohne RESET pin
	PLCC32	/g29fxxx	PLCC32	
AT49F010	DIP32	/g29fxxx		
	PLCC32	/g29fxxx	PLCC32	
AT49F020	DIP32	/g29fxxx		
	PLCC32	/g29fxxx	PLCC32	
AT49F040/T	DIP32	/g29fxxx		
	PLCC32	/g29fxxx	PLCC32	
AT49F512	DIP32	/g29fxxx		
	PLCC32	/g29fxxx	PLCC32	
AT49LV001/T	DIP32	/g49lv00x		mit RESET pin, Low
	PLCC32	/g49lv00x	PLCC32	Voltage
AT49LV001N/T	DIP32	/g29lvxxx		ohne RESET pin,
	PLCC32	/g29lvxxx	PLCC32	Low Voltage
AT49LV002/T	DIP32	/g49lv00x		mit RESET pin
	PLCC32	/g49lv00x	PLCC32	
AT49LV002N/T	DIP32	/g29lvxxx		ohne RESET pin,
	PLCC32	/g29lvxxx	PLCC32	Low Voltage
AT49LV010	DIP32	/g29lvxxx		Low Voltage

Hersteller	Atmel			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
	PLCC32	/g29lvxxx	PLCC32	
AT49LV020	DIP32	/g29lvxxx		Low Voltage
	PLCC32	/g29lvxxx	PLCC32	
AT49LV040/T	DIP32	/g29lvxxx		Low Voltage
	PLCC32	/g29lvxxx	PLCC32	
AT89C1051	DIP20	/ga89cx051		
	SOIC20	/ga89cx051	SOIC20	
AT89C1051U	DIP20	/ga89cx051		1051 mit UART
	SOIC20	/ga89cx051	SOIC20	
AT89C2051	DIP20	/ga89cx051		
	SOIC20	/ga89cx051	SOIC20	
AT89C4051	DIP20	/ga89cx051		
	SOIC20	/ga89cx051	SOIC20	
AT89C51	DIP40	/ga89c5x		
	PLCC44	/ga89c5x	PLCC44	
	TQFP44	/ga89c5x	TQFP44	
AT89C51-5	DIP40	/ga89c5x-5		Vpp = 5V
	PLCC44	/ga89c5x-5	PLCC44	Vpp = 5V
	TQFP44	/ga89c5x-5	TQFP44	
AT89C51IC2	DIP40	/gt89c51rc2-5		
	PLCC44	/gt89c51rc2-5	PLCC44	
	TQFP44	/gt89c51rc2-5	TQFP44	
AT89C51RB2	DIP40	/gt89c51rc2-5		
	PLCC44	/gt89c51rc2-5	PLCC44	
	TQFP44	/gt89c51rc2-5	TQFP44	
AT89C51RC	DIP40	/ga89c5x2		
	PLCC44	/ga89c5x2	PLCC44	
	TQFP44	/ga89c5x2	TQFP44	
AT89C51RC2	DIP40	/gt89c51rc2-5		
	PLCC44	/gt89c51rc2-5	PLCC44	
	TQFP44	/gt89c51rc2-5	TQFP44	
AT89C52	DIP40	/ga89c5x		
	PLCC44	/ga89c5x	PLCC44	
	TQFP44	/ga89c5x	TQFP44	
AT89C52-5	DIP40	/ga89c5x-5		Vpp = 5V
	PLCC44	/ga89c5x-5	PLCC44	Vpp = 5V
	TQFP44	/ga89c5x-5	TQFP44	
AT89C55	DIP40	/ga89c5x		
	PLCC44	/ga89c5x	PLCC44	
	TQFP44	/ga89c5x	TQFP44	
AT89C55WD	DIP40	/ga89c5x2		
	PLCC44	/ga89c5x2	PLCC44	
	TQFP44	/ga89c5x2	TQFP44	
AT89LS51	DIP40	/ga89sxx2		
	PLCC44	/ga89sxx2	PLCC44	
	TQFP44	/ga89sxx2	TQFP44	
AT89LS52	DIP40	/ga89sxx2		
	PLCC44	/ga89sxx2	PLCC44	
	TQFP44	/ga89sxx2	TQFP44	
AT89LS53	DIP40	/ga89sxxxx		
	PLCC44	/ga89sxxxx	PLCC44	
	TQFP44	/ga89sxxxx	TQFP44	
AT89LS8252	DIP40	/ga89sxxxx		FLASH Programm-

Hersteller	Atmel			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
	PLCC44	/ga89sxxxx	PLCC44	speicher
	TQFP44	/ga89sxxxx	TQFP44	
AT89LS8252	DIP40	/ga89seeprom		EEPROM Datenspei-
	PLCC44	/ga89seeprom	PLCC44	cher
	TQFP44	/ga89seeprom	TQFP44	
AT89LV51	DIP40	/ga89c5x		
	PLCC44	/ga89c5x	PLCC44	
	TQFP44	/ga89c5x	TQFP44	
AT89LV51-5	DIP40	/ga89c5x-5		Vpp = 5V
	PLCC44	/ga89c5x-5	PLCC44	Vpp = 5V
	TQFP44	/ga89c5x-5	TQFP44	
AT89LV52	DIP40	/ga89c5x		
	PLCC44	/ga89c5x	PLCC44	
	TQFP44	/ga89c5x	TQFP44	
AT89LV52-5	DIP40	/ga89c5x-5		Vpp = 5V
	PLCC44	/ga89c5x-5	PLCC44	Vpp = 5V
	TQFP44	/ga89c5x-5	TQFP44	
AT89LV55	DIP40	/ga89c5x		
	PLCC44	/ga89c5x	PLCC44	
	TQFP44	/ga89c5x	TQFP44	
AT89S51	DIP40	/ga89sxx2		
	PLCC44	/ga89sxx2	PLCC44	
	TQFP44	/ga89sxx2	TQFP44	
AT89S52	DIP40	/ga89sxx2		
	PLCC44	/ga89sxx2	PLCC44	
	TQFP44	/ga89sxx2	TQFP44	
AT89S53	DIP40	/ga89sxxxx		
	PLCC44	/ga89sxxxx	PLCC44	
	TQFP44	/ga89sxxxx	TQFP44	
AT89S8252	DIP40	/ga89sxxxx		FLASH Programm-
	PLCC44	/ga89sxxxx	PLCC44	speicher
	TQFP44	/ga89sxxxx	TQFP44	
AT89S8252E	DIP40	/ga89seeprom		EEPROM Datenspei-
	PLCC44	/ga89seeprom	PLCC44	cher
	TQFP44	/ga89seeprom	TQFP44	
AT89S8253	DIP40	/ga89sxx3		FLASH Programm-
	PLCC44	/ga89sxx3	PLCC44	speicher
	TQFP44	/ga89sxx3	TQFP44	
AT89S8253E	DIP40	/ga89seeprom3		EEPROM Datenspei-
	PLCC44	/ga89seeprom3	PLCC44	cher
	TQFP44	/ga89seeprom3	TQFP44	
AT90S1200	DIP20	/gavr20		FLASH Programm-
	SOIC20	/gavr20	SOIC20	speicher
AT90S1200E	DIP20	/gavr20e		EEPROM Datenspei-
	SOIC20	/gavr20e	SOIC20	cher
AT90S2313	DIP20	/gavr20		FLASH Programm-
	SOIC20	/gavr20	SOIC20	speicher
Attiny2313	DIP20	/gavrtiny20		Siehe Hinweise !!!
	SOIC20	/gavrtiny20	SOIC20	
AT90S2313E	DIP20	/gavr20e		EEPROM Datenspei-
	SOIC20	/gavr20e	SOIC20	cher
AT90S4414	DIP40	/gavr40_2		FLASH
AT90S4414E	DIP40	/gavr40e_2		EEPROM

Hersteller	Atmel			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
AT90S8515	DIP40	/gavr40_2		FLASH
AT90S8515E	DIP40	/gavr40e_2		EEPROM
ATF16V8B/BQ/BQL	DIP20	/ga16v8		
	PLCC20	/ga16v8	PLCC20	
	SOIC20	/ga16v8	SOIC20	
ATF20V8B/BQ/BQL	DIP24	/ga20v8		
	PLCC28	/ga20v8	PLCC28_24	
	SOIC24		auf Anfrage	
ATF22V10B/BQ/BQL	DIP24	/ga22v10		
	PLCC28	/ga22v10	PLCC28_24	
	SOIC28		auf Anfrage	
ATMEGA161	DIP40	/gatmega_2		FLASH Programm-
	TQFP44	/gatmega_2	TQFP44	speicher
ATMEGA161	DIP40	/gatmega_e_2		EEPROM Datenspei-
EEPROM	TQFP44	/gatmega_e_2	TQFP44	cher
T89C51RB2	DIP40	/gt89c51rc2-5		
	PLCC44	/gt89c51rc2-5	PLCC44	
	TQFP44	/gt89c51rc2-5	TQFP44	
T89C51RC2	DIP40	/gt89c51rc2-5		
	PLCC44	/gt89c51rc2-5	PLCC44	
	TQFP44	/gt89c51rc2-5	TQFP44	
T89C51RD2	DIP40	/gt89c51rx2-5		
	PLCC44	/gt89c51rx2-5	PLCC44	
TS87C52X2	DIP40	/gt87c5x		baugleich Temic
	PLCC44	/gt87c5x	PLCC44	TS87C52X2

Hersteller	Bright			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
Bm29F040	DIP32	/g29fxxx		Sector Protection
	PLCC32	/g29fxxx	PLCC32	nicht programmierbar

Hersteller	Catalyst			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
25C128	DIP8	/g25128		
28F001	DIP32	/gi28f00x		
	PLCC32	/gi28f00x	PLCC32	
28F010	DIP32	/g28fxxx		
	PLCC32	/g28fxxx	PLCC32	
28F020	DIP32	/g28fxxx		
	PLCC32	/g28fxxx	PLCC32	
28F512	DIP32	/g28fxxx		
	PLCC32	/g28fxxx	PLCC32	
93C66	DIP8	/g93c66	SERMEM	
CAT27C210/I	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
CAT28C16A	DIP24	/g28c16		
	PLCC32	/g28c16plcc	PLCC32	
CAT28C17A	DIP28	/g28c17		
	PLCC32	/g28c16plcc	PLCC32	
CAT28C256	DIP28	/gcat28c256		
	PLCC32	/gcat28c256plcc	PLCC32	
CAT28C64B	DIP28	/gcat28c64b		

Hersteller	Catalyst			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
	PLCC32	/gcat28c64bplcc	PLCC32	
CAT28C65B	DIP28	/gcat28c64b		
	PLCC32	/gcat28c64bplcc	PLCC32	

Hersteller	Dallas			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
DS1220AB/AD	DIP24	/g28c16		
DS1225AB/AD	DIP28	/g28c64		
DS1230AB/AD	DIP28	/gsram256		
DS1245AB/AD	DIP32	/gsram1024		
DS87C520	DIP40	/gd87c5x0		Watchdog aus
	DIP40	/gd87c5x0-w		Watchdog an
	PLCC44	/gd87c5x0	PLCC44	Watchdog aus
	PLCC44	/gd87c5x0-w	PLCC44	Watchdog an
DS89C420	DIP40	/gd89cxxx		
	PLCC44	/gd89cxxx	PLCC44	
DS87C530	PLCC52	/gd87c5x0	PLCC52	Watchdog aus
	PLCC52	/gd87c5x0-w	PLCC52	Watchdog an
DS87C550	PLCC68	/gd87c5x0	PLCC68_40	Watchdog aus
	PLCC68	/gd87c5x0-w	PLCC68_40	Watchdog an

Hersteller	Eon Silicon Devices			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
EN29F002B/T	DIP32	/g29f00x		
	PLCC32	/g29f00x	PLCC32	
	TSOP32	/g29f00x	TSOP32	
EN29F002NB/T	DIP32	/g29fxxx		
	PLCC32	/g29fxxx	PLCC32	
	TSOP32	/g29fxxx	TSOP32	

Hersteller	Fairchild			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
FM27C256	DIP28	/gn27c256		
	PLCC32	/gn27c256	PLCC32_28	
FM27C512	DIP28	/gf27c512_2		
	PLCC32	/gf27c512_2	PLCC32_28	

Hersteller	Fujitsu			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
MBM27C1024	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
MBM27C4096	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen

Hersteller	Hitachi			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
HN27C101AG/AP	DIP32	/gh27c101		
HN27C1024H	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
HN27C256AG/AP	DIP28	/gh27c256		
	PLCC32	/gh27c256	PLCC32_28	

Hersteller	Hitachi			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
HN27C301AG/AP	DIP32	/gh27c301		
HN27C4001G	DIP32	/gh27c4001		
HN27C4096	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
HN28F101	DIP32	/g28fxxx		
	PLCC32	/g28fxxx	PLCC32	

Hersteller	Holtek			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
93LC46	DIP8	/g93c46	SERMEM	
93LC56	DIP8	/g93c56	SERMEM	
93LC66	DIP8	/g93c66	SERMEM	
HT24LC02	DIP8	/g24c02		
HT24LC04	DIP8	/g24c04		
HT24LC08	DIP8	/g24c08		
HT24LC16	DIP8	/g24c16		
HT27C010	DIP32	/ght27c010		Auch: HT27LC010
	PLCC32	/ght27c010	PLCC32	
HT27C020	DIP32	/ght27c020		Auch: HT27LC020
	PLCC32	/ght27c020	PLCC32	
HT27C040	DIP32	/ght27c040		Auch: HT27LC040
	PLCC32	/ght27c040	PLCC32	
HT27C4096	DIP40	/geprom16_t2	MEM16_DIP	Auch: HT27LC4096
HT27C512	DIP32	/ght27c512_2		Auch: HT27LC512
	PLCC32	/ght27c512_2	PLCC32_28	

Hersteller	Hynix			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
Hy29F002	PLCC32	/g29f00x	PLCC32	
	TSOP32	/g29f00x	TSOP32	
Hy29F040A	PLCC32	/g29fxxx	PLCC32	
	TSOP32	/g29fxxx	TSOP32	

Hersteller	Integrated Silicon Solution Inc. (ISSI)			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
IS24C02	dip8	/g24c02		
IS24C04	dip8	/g24c04		
IS27C010	dip32	/gis27c010		
	plcc32	/gis27c010	PLCC32	
IS27C020	dip32	/gis27c020		
	plcc32	/gis27c020	PLCC32	
IS27C2048	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
IS27C256	dip28	/gis27c256		
	PLCC32	/gis27c256	PLCC32_28	
IS27HC010	dip32	/gis27c010		
	plcc32	/gis27c010	PLCC32	
IS27HC256	dip28	/gis27c256		
	PLCC32	/gis27c256	PLCC32_28	
IS27LV010	dip32	/gis27c010		
	plcc32	/gis27c010	PLCC32	
IS27LV020	plcc32	/gis27c020	PLCC32	

Hersteller	Integrated Silicon Solution Inc. (ISSI)			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
IS28F010	dip32	/g28fxxx		
	plcc32	/g28fxxx	PLCC32	
IS28F020	dip32	/g28fxxx		
	plcc32	/g28fxxx	PLCC32	
IS93C46-3	DIP8	/g93c46	SERMEM	
IS93C56-3	DIP8	/g93c56	SERMEM	
IS93C66-3	DIP8	/g93c66	SERMEM	

Hersteller	Intel			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
27C210	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
27C220	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
27C240	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
28F001	DIP32	/g28fxxx		siehe Hinweis Anlei-
	PLCC32	/g28fxxx	PLCC32	tung
28F010	DIP32	/g28fxxx		
	PLCC32	/g28fxxx	PLCC32	
28F020	DIP32	/g28fxxx		
	PLCC32	/g28fxxx	PLCC32	
28F256A	DIP32	/g28fxxx		
	PLCC32	/g28fxxx	PLCC32	
28F512	DIP32	/g28fxxx		
	PLCC32	/g28fxxx	PLCC32	
82802AB	PLCC32	/gi82802	PLCC32	
82802AC	PLCC32	/gi82802	PLCC32	
87C51/FA/FB/FC	DIP40	/gi87c5x		alte u. neue Version
	PLCC44	/gi87c5x	PLCC44	
87C52	DIP40	/gi87c5x		
	PLCC44	/gi87c5x	PLCC44	
87C54	DIP40	/gi87c5x		
	PLCC44	/gi87c5x	PLCC44	
87C58	DIP40	/gi87c5x		
	PLCC44	/gi87c5x	PLCC44	

Hersteller	Lattice, SGS Thomson			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
GAL16LV8C/D	DIP20	/gL16Lv8		Low Voltage
	PLCC20	/gL16Lv8	PLCC20	
GAL16LV8Z/ZD	DIP20	/gL16Lv8		Low Voltage
	PLCC20	/gL16Lv8	PLCC20	
GAL16V8A/B/C/D	DIP20	/gL16v8		
	PLCC20	/gL16v8	PLCC20	
GAL16V8Z/ZD	DIP20	/gL16v8		
	PLCC20	/gL16v8	PLCC20	
GAL18V10/B	DIP20	/gL18v10		
	PLCC20	/gL18v10	PLCC20	
GAL20LV8C/D	DIP24	/gL20Lv8		Low Voltage
	PLCC28	/gL20Lv8	PLCC28_24	
GAL20LV8Z/ZD	DIP24	/gL20Lv8		Low Voltage
	PLCC28	/gL20Lv8	PLCC28_24	
GAL20RA10	DIP24	/gL20ra10		

Hersteller	Lattice, SGS Thomson			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
	PLCC28	/gL20ra10	PLCC28_24	
GAL20V8A/B/C/D	DIP24	/gL20v8		
	PLCC28	/gL20v8	PLCC28_24	
GAL20V8Z/ZD	DIP24	/gL20v8		
	PLCC28	/gL20v8	PLCC28_24	
GAL22LV10C/D	DIP24	/gL22Lv10		Low Voltage
	PLCC28	/gL22Lv10	PLCC28	
GAL22LV10Z/ZD	DIP24	/gL22Lv10		Low Voltage
	PLCC28	/gL22Lv10	PLCC28	
GAL22V10/B/C/D	DIP24	/gL22v10		
	PLCC28	/gL22v10	PLCC28	
GAL22V10Z/ZD	DIP24	/gL22v10		
	PLCC28	/gL22v10	PLCC28	
GAL6001/B	DIP24	/gL6001		
	PLCC28	/gL6001	PLCC28	
GAL6002B	DIP24	/gL6002		
	PLCC28	/gL6002	PLCC28	

Hersteller	Macronix			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
MX26C512A	DIP28	/gam27c512_2		Löschen mit
	PLCC32	/gam27c512_2	PLCC32_28	hed.eraser
MX27C1000	DIP32	/gmx27c1000		auch: MX27L1000
	PLCC32	/gmx27c1000	PLCC32	
MX27C1000A	DIP32	/gmx27c1000a		
	PLCC32	/gmx27c1000a	PLCC32	
MX27C2000	DIP32	/gmx27c2000		auch: MX27L2000
	PLCC32	/gmx27c2000	PLCC32	
MX27C2000A	DIP32	/gmx27c2000a		
	PLCC32	/gmx27c2000a	PLCC32	
MX27C256	DIP28	/gis27c256		
	PLCC32	/gis27c256	PLCC32_28	
MX27C4000	DIP32	/gmx27c4000		auch: MX27L4000
	PLCC32	/gmx27c4000	PLCC32	
MX27C4000A	DIP32	/gmx27c4000a		
	PLCC32	/gmx27c4000a	PLCC32	
MX27C512	DIP28	/gam27c512_2		
	PLCC32	/gam27c512_2	PLCC32_28	
MX27C8000	DIP32	/gmx27c8000		
	PLCC32	/gmx27c8000	PLCC32	
MX27C8000A	DIP32	/gmx27c8000a		
	PLCC32	/gmx27c8000a	PLCC32	
MX28F1000P	DIP32	/gmx28fxxxx		
	PLCC32	/gmx28fxxxx	PLCC32	
MX29F001T/B	DIP32	/g29fxxx		Sektor Protection
	PLCC32	/g29fxxx	PLCC32	nicht programmierbar
MX29F002T/B	DIP32	/g29fxxx		Sektor Protection
	PLCC32	/g29fxxx	PLCC32	nicht programmierbar
MX29F022	DIP32	/g29fxxx		Sektor Protection
	PLCC32	/g29fxxx	PLCC32	nicht programmierbar
MX29F040	DIP32	/g29fxxx		Sektor Protection
	PLCC32	/g29fxxx	PLCC32	nicht programmierbar

Hersteller	Macronix			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
MX29LV040C	PLCC32	/g29lvxxx	PLCC32	Sektor Protection nicht programmierbar

Hersteller	Microchip			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
24AA16	DIP8	/g24c16		
24AA32A	DIP8	/g24xc32		
24AA64	DIP8	/g24xc64		
24C00	DIP8	/g24c00		
24C01A	DIP8	/g24c01a		
24C02A	DIP8	/g24c02		
24C04A	DIP8	/g24c04		
24C08	DIP8	/g24c08		
24C164	DIP8	/g24c16		
24C16B	DIP8	/g24c16		
24C32A	DIP8	/g24xc32		
24C65	DIP8	/g24xc64		Security nicht programmierbar
24LC128	DIP8	/g24xc128		auch 24AA128, 24FC128
24LC16B	DIP8	/g24c16		
24LC256	DIP8	/g24xc256		auch 24FC256
24LC32A	DIP8	/g24xc32		
24LC64	DIP8	/g24xc64		
24LCS52	DIP8	/g24lcs52		Schreibschutz ist irreversibel
27C128	DIP28	/gm27c128		
27C256	DIP28	/gm27c256		
	PLCC32	/gm27c256	PLCC32_28	
27C512A	DIP28	/gm27c512a_2		
	PLCC32	/gm27c512a_2	PLCC32_28	
27C64	DIP28	/gm27c64		
28C04A	DIP24	/g28c04		
	PLCC32	/g28c04plcc	PLCC32	
28C16A	DIP24	/g28c16b		
	PLCC32	/g28c16bplcc	PLCC32	
28C17A	DIP28	/g28c17		
	PLCC32	/g28c16plcc	PLCC32	
28C64A	DIP28	/g28c64		
	PLCC32	/g28c64plcc	PLCC32	
93C56A/B	DIP8	/g93c56	SERMEM	
93C66A/B	DIP8	/g93c66	SERMEM	
93C76	DIP8	/g93c76	SERMEM	
93C86	DIP8	/g93c86	SERMEM	
93LC46	DIP8	/g93c46	SERMEM	
93LC46A/B	DIP8	/g93c46	SERMEM	
93LC56	DIP8	/g93c56	SERMEM	
93LC66A/B	DIP8	/g93c66	SERMEM	
93LC76B	DIP8	/g93c76	SERMEM	
93LC86B	DIP8	/g93c86	SERMEM	
PIC12C508/A	DIP8	/gpic12c508	UNIPIC18	
PIC12C509/A	DIP8	/gpic12c509	UNIPIC18	
PIC12C671	DIP8	/gpic12C671	UNIPIC18	

Hersteller	Microchip			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
PIC12C672	DIP8	/gpic12C672	UNIPIC18	
PIC12CE518	DIP8	/gpic12c508	UNIPIC18	
PIC12CE519	DIP8	/gpic12c509	UNIPIC18	
PIC12CE673	DIP8	/gpic12C671	UNIPIC18	
PIC12CE674	DIP8	/gpic12C672	UNIPIC18	
PIC12F629	DIP8	/gpic12f629	UNIPIC18	Siehe Anleitung !
PIC12F675	DIP8	/gpic12f629	UNIPIC18	Siehe Anleitung !
PIC16C61	DIP18	/gpic16c61	UNIPIC18	
PIC16C62	DIP28	/gpic16c62	UNIPIC	
PIC16C620	DIP18	/gpic16c620	UNIPIC18	
PIC16C620A	DIP18	/gpic16c620	UNIPIC18	
PIC16C621	DIP18	/gpic16c621	UNIPIC18	
PIC16C621A	DIP18	/gpic16c621	UNIPIC18	
PIC16C622	DIP18	/gpic16c62a	UNIPIC18	
PIC16C622A	DIP18	/gpic16c62a	UNIPIC18	
PIC16C62A	DIP28	/gpic16c62a	UNIPIC	
PIC16C62B	DIP28	/gpic16c62a	UNIPIC	
PIC16C62C	DIP28	/gpic16c62a	UNIPIC	
PIC16C63	DIP28	/gpic16c63	UNIPIC	
PIC16C64	DIP40	/gpic16c62	UNIPIC	
PIC16C64A	DIP40	/gpic16c62a	UNIPIC	
PIC16C65	DIP40	/gpic16c65	UNIPIC	
PIC16C65A	DIP40	/gpic16c63	UNIPIC	
PIC16C65B	DIP40	/gpic16c63	UNIPIC	
PIC16C66	DIP28	/gpic16c66	UNIPIC	
PIC16C67	DIP40	/gpic16c66	UNIPIC	
PIC16C71	DIP18	/gpic16c61	UNIPIC18	
PIC16C710	DIP18	/gpic16c710	UNIPIC18	
PIC16C711	DIP18	/gpic16c711	UNIPIC18	
PIC16C712	DIP18	/gpic16c621	UNIPIC18	
PIC16C716	DIP28	/gpic16c62a	UNIPIC	
PIC16C717	DIP18	/gpic16c717	UNIPIC18	
PIC16C72	DIP28	/gpic16c62a	UNIPIC	
PIC16C72A	DIP28	/gpic16c62a	UNIPIC	
PIC16C73	DIP28	/gpic16c65	UNIPIC	
PIC16C73A	DIP28	/gpic16c63	UNIPIC	
PIC16C73B	DIP28	/gpic16c63	UNIPIC	
PIC16C74	DIP40	/gpic16c65	UNIPIC	
PIC16C745	DIP28	/gpic16c745	UNIPIC	
PIC16C74A	DIP40	/gpic16c63	UNIPIC	
PIC16C74B	DIP40	/gpic16c63	UNIPIC	
PIC16C76	DIP28	/gpic16c66	UNIPIC	
PIC16C765	DIP40	/gpic16c745	UNIPIC	
PIC16C77	DIP40	/gpic16c66	UNIPIC	
PIC16C770	DIP20	/gpic16c770	UNIPIC18	siehe Windows Hilfe wegen DIP20
PIC16C771	DIP20	/gpic16c771	UNIPIC18	siehe Windows Hilfe wegen DIP20
PIC16C773	DIP28	/gpic16c773	UNIPIC	
PIC16C774	DIP40	/gpic16c773	UNIPIC	
PIC16C781	DIP20	/gpic16c781	UNIPIC18	siehe Windows Hilfe wegen DIP20
PIC16C782	DIP20	/gpic16c770	UNIPIC18	siehe Windows Hilfe

Hersteller	Microchip			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
				wegen DIP20
PIC16C84	DIP18	/gpic16c84	UNIPIC18	
PIC16C923	PLCC68	/gpic16c923		Adapter auf Anfrage
PIC16C924	PLCC68	/gpic16c924		
PIC16CE623	DIP18	/gpic16c620	UNIPIC18	
PIC16CE624	DIP18	/gpic16c621	UNIPIC18	
PIC16CE625	DIP28	/gpic16c62a	UNIPIC	
PIC16CR62	DIP28	/gpic16c62a	UNIPIC	
PIC16CR64	DIP40	/gpic16c62a	UNIPIC	
PIC16CR83	DIP18	/gpic16cr83	UNIPIC18	auch PIC16LCR83
PIC16CR84	DIP18	/gpic16cr84	UNIPIC18	auch PIC16LCR84
PIC16F627	DIP18	/gpic16f627	UNIPIC18	auch PIC16LF627
PIC16F628	DIP18	/gpic16f628	UNIPIC18	auch PIC16LF628
PIC16F630	DIP14	/gpic12f629	UNIPIC18	Siehe Anleitung !
PIC16F676	DIP14	/gpic12f629	UNIPIC18	Siehe Anleitung !
PIC16F83	DIP18	/gpic16f83	UNIPIC18	auch PIC16LF83
PIC16F84	DIP18	/gpic16f84	UNIPIC18	auch PIC16LF84
PIC16F84A	DIP18	/gpic16f84	UNIPIC18	auch PIC16LF84A
PIC16F870	DIP28	/gpic16f870	UNIPIC	
PIC16F871	DIP40	/gpic16f870	UNIPIC	
PIC16F872	DIP28	/gpic16f870	UNIPIC	
PIC16F873	DIP28	/gpic16f873	UNIPIC	
PIC16F873A	DIP28	/gpic16f873a	UNIPIC	
PIC16F874	DIP40	/gpic16f873	UNIPIC	
PIC16F874A	DIP40	/gpic16f873a	UNIPIC	
PIC16F876	DIP28	/gpic16f876	UNIPIC	
PIC16F876A	DIP28	/gpic16f876a	UNIPIC	
PIC16F877	DIP40	/gpic16f876	UNIPIC	
PIC16F877A	DIP40	/gpic16f876a	UNIPIC	
PICDATA128		/gpicdata128	UNIPIC	
PICDATA256		/gpicdata256	UNIPIC	
PICDATA64		/gpicdata64	UNIPIC	

Hersteller	Mitsubishi			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
M5M28F101A	DIP32	/g28fxxx		
	PLCC32	/g28fxxx	PLCC32	

Hersteller	National Semiconductor (NSC)			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
GAL16V8	DIP20	/gL16v8		baugleich Lattice
	PLCC20	/gL16v8	PLCC20	GAL16V8
GAL20V8	DIP24	/gL20v8		baugleich Lattice
	PLCC28	/gL20v8	PLCC28	GAL20V8
NM24C02	DIP8	/g24c02		
NM24C03	DIP8	/g24c02		
NM24C04	DIP8	/g24c04		
NM24C05	DIP8	/g24c04		
NM24C08	DIP8	/g24c08		
NM24C09	DIP8	/g24c08		
NM24C16	DIP8	/g24c16		

Hersteller	National Semiconductor (NSC)			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
NM24C17	DIP8	/g24c16		
NM27C010	DIP32	/gn27c010		
	PLCC32	/gn27c010	PLCC32	
NM27C020	DIP32	/gn27c020		
	PLCC32	/gn27c020	PLCC32	
NM27C040	DIP32	/gn27c040		
	PLCC32	/gn27c040	PLCC32	
NM27C128	DIP28	/gn27c128		
NM27C16B	DIP24	/gn27c16	DIPMEM	
NM27C210	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
NM27C220	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
NM27C240	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
NM27C256	DIP28	/gn27c256		
	PLCC32	/gn27c256	PLCC32_28	
NM27C32	DIP24	/gn27c32_2		
NM27C512	DIP28	/gn27c512_2		
	PLCC32	/gn27c512_2	PLCC32_28	
NM27C64	DIP28	/gn27c64		
NM27LC256	DIP28	/gn27lc256		
NM27LC64	DIP28	/gn27c64		
NM27LV010	DIP32	/gn27c010		
NM27P040	DIP32	/gn27c040		
NM27P210	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
NM27P220	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
NM27P240	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
NM93C06	DIP8	/g93c06	SERMEM	
NM93C46	DIP8	/g93c46	SERMEM	
NM93C56	DIP8	/g93c56	SERMEM	
NM93C66	DIP8	/g93c66	SERMEM	
NM93C86A	DIP8	/g93c86	SERMEM	
NMC27C2048	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
NMC27C4096	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
NMC87C257	DIP28	/gn27c256		
	PLCC32	/gn27c256	PLCC32_28	
NMX27C1024	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen

Hersteller	Philips			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
27C210	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
27C240	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
87C504	DIP40	/gp87c5x		
	PLCC44	/gp87c5x	PLCC44	
87C504	DIP40	/gp87c5x		
	PLCC44	/gp87c5x	PLCC44	
87C51/FA/FB/FC	DIP40	/gp87c5x		
	PLCC44	/gp87c5x	PLCC44	
87C51/FA/FB/FC	DIP40	/gp87c5x		
	PLCC44	/gp87c5x	PLCC44	
87C51/RA/RB/RC/RD+	DIP40	/gp87c5x		
	PLCC44	/gp87c5x	PLCC44	
87C52	DIP40	/gp87c5x		
	PLCC44	/gp87c5x	PLCC44	

Hersteller	Philips			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
87C524	DIP40	/gp87c5x		
	PLCC44	/gp87c5x	PLCC44	
87C528	DIP40	/gp87c5x		
	PLCC44	/gp87c5x	PLCC44	
87C54	DIP40	/gp87c5x		
	PLCC44	/gp87c5x	PLCC44	
87C550	DIP40	/gp87c5x		
	PLCC44	/gp87c5x	PLCC44	
87C552	PLCC68	/gp87c5x	PLCC68_40	
87C575	DIP40	/gp87c5x		
	PLCC44	/gp87c5x	PLCC44	
87C576	DIP40	/gp87c5x		
	PLCC44	/gp87c5x	PLCC44	
87C58	DIP40	/gp87c5x		
	PLCC44	/gp87c5x	PLCC44	
87C652	PLCC68	/gp87c5x	PLCC68_40	
87C654	DIP40	/gp87c5x		
	PLCC44	/gp87c5x	PLCC44	
87C748	DIP24	/gp87c7xx		
87C749	DIP28	/gp87c7xx	DIP752	
87C750	DIP24	/gp87c750		
87C751	DIP24	/gp87c7xx		
87C752	DIP28	/gp87c7xx	DIP752	
P87C51MA2	PLCC44	/gp87c51mx2	PLCC44	
P87C51MB2	PLCC44	/gp87c51mx2	PLCC44	
P87C51MC2	PLCC44	/gp87c51mx2	PLCC44	
P89C51RB2	DIP40	/gp89c5x		
	PLCC44	/gp89c5x	PLCC44	
P89C51RC+	DIP40	/gp89c5x		
	PLCC44	/gp89c5x	PLCC44	
P89C51RC2	DIP40	/gp89c5x		
	PLCC44	/gp89c5x	PLCC44	
P89C51RD+	DIP40	/gp89c5x		
	PLCC44	/gp89c5x	PLCC44	
P89C51RD2	DIP40	/gp89c5x		
	PLCC44	/gp89c5x	PLCC44	
P89C51Uxxx	DIP40	/gp89c5x		
	PLCC44	/gp89c5x	PLCC44	
P89C52Uxxx	DIP40	/gp89c5x		
	PLCC44	/gp89c5x	PLCC44	
P89C54Uxxx	DIP40	/gp89c5x		
	PLCC44	/gp89c5x	PLCC44	
P89C58Uxxx	DIP40	/gp89c5x		
	PLCC44	/gp89c5x	PLCC44	
P89C660	PLCC44	/gp89c5x	PLCC44	
P89C662	PLCC44	/gp89c5x	PLCC44	
P89C664	PLCC44	/gp89c5x	PLCC44	
P89C668	PLCC44	/gp89c5x	PLCC44	
PCF8582C-2	DIP8	/gpcf8582		
PCF8594C-2	DIP8	/gpcf8594		
PCF8598C-2	DIP8	/gpcf8598		

Hersteller	PMC Flash			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
Pm29F002B/T	DIP32	/g29fxxx		
	PLCC32	/g29fxxx	PLCC32	
Pm39F010	DIP32	/g29fxxx		
	PLCC32	/g29fxxx	PLCC32	
Pm39LV010	PLCC32	/g29fxxx	PLCC32	
	VSOP32	/g29fxxx	VSOP32	
Pm39LV512	PLCC32	/g29fxxx	PLCC32	
	VSOP32	/g29fxxx	VSOP32	
Pm49FL002	PLCC32	/g49lf00xa	PLCC32	
	VSOP32	/g49lf00xa	VSOP32	
Pm49FL004	PLCC32	/g49lf00xa	PLCC32	
	VSOP32	/g49lf00xa	VSOP32	
Pm49FL008	PLCC32	/g49lf00xa	PLCC32	
	VSOP32	/g49lf00xa	VSOP32	

Hersteller	SGS Thomson			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
M24C01	DIP8	/g24c01a		
M24C02	DIP8	/g24c02		
M24C04	DIP8	/g24c04		
M24C08	DIP8	/g24c08		
M24C128	DIP8	/g24xc128		
M24C16	DIP8	/g24c16		
M24C256	DIP8	/g24xc256		
M24C32	DIP8	/g24xc32		
M24C64	DIP8	/g24xc64		
M27128A	DIP28	/gs27128		
M27256	DIP28	/gs27256		
M27512	DIP28	/gs27512_2		
M2764A	DIP28	/gs2764		
M27C1000	DIP32	/gs27c1000		
M27C1001	DIP32	/gs27c1001		
	PLCC32	/gs27c1001	PLCC32	
M27C1024	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
M27C128A	DIP28	/gs27c128		
M27C2001	DIP32	/gs27c2001		
	PLCC32	/gs27c2001	PLCC32	
M27C202	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
M27C210	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
M27C256B	DIP28	/gs27c256		
	PLCC32	/gs27c256	PLCC32_28	
M27C4001	DIP32	/gs27c4001		
	PLCC32	/gs27c4001	PLCC32	
M27C4002	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
M27C512	DIP28	/gs27c512_2		
	PLCC32	/gs27c512_2	PLCC32_28	
M27C64A	DIP28	/gs27c64		
M27C801	DIP32	/gs27c801_2		
M27V101	DIP32	/gs27c1001		
	PLCC32	/gs27c1001	PLCC32	
M27V201	DIP32	/gs27c2001		
	PLCC32	/gs27c2001	PLCC32	

Hersteller	SGS Thomson			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
M27V401	DIP32	/gs27c4001		
	PLCC32	/gs27c4001	PLCC32	
M27W101	DIP32	/gs27c1001		
	PLCC32	/gs27c1001	PLCC32	
M27W102	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
M27W201	DIP32	/gs27c2001		
	PLCC32	/gs27c2001	PLCC32	
M27W401	DIP32	/gs27c4001		
	PLCC32	/gs27c4001	PLCC32	
M27W402	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
M28F101	DIP32	/g28fxxx		
	PLCC32	/g28fxxx	PLCC32	
M28F201	DIP32	/g28fxxx		
	PLCC32	/g28fxxx	PLCC32	
M28F256	DIP32	/g28fxxx		
	PLCC32	/g28fxxx	PLCC32	
M28F512	DIP32	/g28fxxx		
	PLCC32	/g28fxxx	PLCC32	
M29F002B(B/T)	DIP32	/g29f00x		Sector Protection nicht
	PLCC32	/g29f00x	PLCC32	programmierbar
	TSOP32	/g29fxxx	TSOP32	
M29F002BN(B/T)	DIP32	/g29fxxx		Sector Protection nicht
	PLCC32	/g29fxxx	PLCC32	programmierbar
	TSOP32	/g29fxxx	TSOP32	
M29F010B	DIP32	/g29fxxx		Sector Protection nicht
	PLCC32	/g29fxxx	PLCC32	programmierbar
M29F040	DIP32	/g29fxxx		Sector Protection nicht
	PLCC32	/g29fxxx	PLCC32	programmierbar
M29F512B	PLCC32	/g29fxxx	PLCC32	Sector Protection nicht programmierbar
M29W010B	PLCC32	/g29lvxxx	PLCC32	Sector Protection nicht programmierbar
M29W022B	PLCC32	/g29lvxxx	PLCC32	Sector Protection nicht programmierbar
M29W040B	PLCC32	/g29lvxxx	PLCC32	Sector Protection nicht programmierbar
M48T02	DIP24	/g28c16		
M48T08	DIP28	/gsram64		
M48T12	DIP24	/g28c16		
M48T18	DIP28	/gsram64		
M87C257	DIP28	/gs27c256		
	PLCC32	/gs27c256	PLCC32_28	
M93C06	DIP8	/g93c06	SERMEM	
M93C46	DIP8	/g93c56	SERMEM	
M93C56	DIP8	/g93c56	SERMEM	
M93C66	DIP8	/g93c66	SERMEM	
M93C76	DIP8	/g93c76	SERMEM	
M93C86	DIP8	/g93c86	SERMEM	
M95128	DIP8	/g25128		
M95256	DIP8	/g25256		
ST24E32	DIP8	/g24xc32		
ST24E64	DIP8	/g24xc64		
ST25E32	DIP8	/g24xc32		

Hersteller	SGS Thomson			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
ST25E64	DIP8	/g24xc64		

Hersteller	Siemens			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
SAB-C513A-H	PLCC44	/gc513a	PLCC44	
SAB-C501-1E	DIP40	/gc501		
	PLCC44	/gc501	PLCC44	
SAB-C505A-4E	PQFP44	/gc505a	PQFP44_C505	
SAB-C505CA-4E	PQFP44	/gc505a	PQFP44_C505	

Hersteller	Silicon Storage Technology SST			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
27SF010	DIP28	/gss27sf010		Nur Programmieren,
	PLCC32	/gss27sf010	PLCC32	nicht Löschen
27SF020	DIP28	/gss27sf020		Nur Programmieren,
	PLCC32	/gss27sf020	PLCC32	nicht Löschen
27SF256	DIP28	/gss27sf256		Nur Programmieren,
	PLCC32	/gss27sf256	PLCC32_28	nicht Löschen
27SF512	DIP28	/gss27sf512_2		Nur Programmieren,
	PLCC32	/gss27sf512_2	PLCC32_28	nicht Löschen
28VF040A	DIP32	/gss28vf040		Low-Voltage
	PLCC32	/gss28vf040	PLCC32	
28xF040(A)	DIP32	/gss28xf040		Nicht für 28VF040 !
	PLCC32	/gss28xf040	PLCC32	
29EE010	DIP32	/gss29eexxx		
	PLCC32	/gss29eexxx	PLCC32	
29EE020	DIP32	/gss29eexxx		
	PLCC32	/gss29eexxx	PLCC32	
29EE512	DIP32	/gss29eexxx		
	PLCC32	/gss29eexxx	PLCC32	
39LF010	DIP32	/g29lvxxx		Low Voltage
	PLCC32	/g29lvxxx	PLCC32	
39LF020	DIP32	/g29lvxxx		Low Voltage
	PLCC32	/g29lvxxx	PLCC32	
39LF040	DIP32	/g29lvxxx		Low Voltage
	PLCC32	/g29lvxxx	PLCC32	
39LF512	DIP32	/g29lvxxx		Low Voltage
	PLCC32	/g29lvxxx	PLCC32	
39SF010A	DIP32	/g29fxxx		
	PLCC32	/g29fxxx	PLCC32	
39SF020A	DIP32	/g29fxxx		
	PLCC32	/g29fxxx	PLCC32	
39SF040	DIP32	/g29fxxx		
	PLCC32	/g29fxxx	PLCC32	
39SF512	DIP32	/g29fxxx		
	PLCC32	/g29fxxx	PLCC32	
39VF010	DIP32	/g29lvxxx		Low Voltage
	PLCC32	/g29lvxxx	PLCC32	
39VF020	DIP32	/g29lvxxx		Low Voltage
	PLCC32	/g29lvxxx	PLCC32	
39VF040	DIP32	/g29lvxxx		Low Voltage

Hersteller	Silicon Storage Technology SST			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
	PLCC32	/g29lvxxx	PLCC32	
39VF512	DIP32	/g29lvxxx		Low Voltage
	PLCC32	/g29lvxxx	PLCC32	
49LF002A	PLCC32	/g49lf00xa	PLCC32	
	VSOP32	/g49lf00xa	VSOP32	
49LF003A	PLCC32	/g49lf00xa	PLCC32	
	VSOP32	/g49lf00xa	VSOP32	
49LF004A	PLCC32	/g49lf00xa	PLCC32	
	VSOP32	/g49lf00xa	VSOP32	
49LF004B	PLCC32	/g49lf00xa	PLCC32	
	VSOP32	/g49lf00xa	VSOP32	
49LF008A	PLCC32	/g49lf00xa	PLCC32	
	VSOP32	/g49lf00xa	VSOP32	
49LF020	PLCC32	/g49lf00xa	PLCC32	
	TSOP32	/g49lf00xa	TSOP32	
49LF040	PLCC32	/g49lf00xa	PLCC32	
	TSOP32	/g49lf00xa	TSOP32	
89F54	DIP40	/gss89f5x_0		FLASH Block 0
	PLCC44	/gss89f5x_0	PLCC44	
89F54	DIP40	/gss89f5x_1		FLASH Block 1
	PLCC44	/gss89f5x_1	PLCC44	
89F58	DIP40	/gss89f5x_0		FLASH Block 0
	PLCC44	/gss89f5x_0	PLCC44	
89F58	DIP40	/gss89f5x_1		FLASH Block 1
	PLCC44	/gss89f5x_1	PLCC44	

Hersteller	Temic Semiconductors			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
TSC87C51	DIP40	/gt87c5x		hat keine Lockbits
	PLCC44	/gt87c5x	PLCC44	

Weitere MCS51 Microcontroller: siehe Atmel

Hersteller	Texas Instruments			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
TMS27C010A	DIP32	/gt27c010		
	PLCC32	/gt27c010	PLCC32	
TMS27C020	DIP32	/gt27c020		
	PLCC32			auf Anfrage
TMS27C040	DIP32	/gt27c040		
	PLCC32	/gt27c040	PLCC32	
TMS27C128	DIP28	/gt27c128		
TMS27C210	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
TMS27C240	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
TMS27C256	DIP28	/gt27c256		
	PLCC32	/gt27c256	PLCC32_28	
TMS27C510	DIP32	/gt27c510		
	PLCC32	/gt27c510	PLCC32	
TMS27C512	DIP28	/gt27c512_2		
	PLCC32	/gt27c512_2	PLCC32_28	

Hersteller	Texas Instruments			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
TMS27PC210	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
TMS27PC240	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen

Hersteller	Toshiba			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
TC571000AD	DIP32	/gam27c010		
TC571001AD	DIP32	/gam27c010		
TMP88PH40M/N	DIP28	/gtmp88ph40	Toshiba BM11196	Spezialadapter von
	SOIC28	/gtmp88ph40	Toshiba BM11195	Toshiba verwenden
TMP88PS43F	QFP80	/gtmp88ps43		Spezialadapter von Toshiba verwenden

Hersteller	Winbond			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
W27C020	DIP32	/gw27c020		Nur Programmieren, Löschen nicht möglich.
	PLCC32	/gw27c020	PLCC32	
W27C040	DIP32	/gw27c040		Nur Programmieren, Löschen nicht möglich.
	PLCC32	/gw27c040	PLCC32	
W27C4096	DIP40	/geprom16_t1	MEM16_DIP40	Jumper setzen
W27C512	DIP28	/gw27e512_2		Nur Programmieren, Löschen nicht möglich.
	PLCC32	/gw27e512_2	PLCC32_28	
W27E010	DIP32	/gw27e010		Nur Programmieren, Löschen nicht möglich.
W27C010	PLCC32	/gw27e010	PLCC32	
W27E257	DIP28	/gw27e257		Nur Programmieren, Löschen nicht möglich.
	PLCC32	/gw27e257	PLCC32_28	
W27E512	DIP28	/gw27e512_2		Nur Programmieren, Löschen nicht möglich.
	PLCC32	/gw27e512_2	PLCC32_28	
W29C011A	DIP32	/gw29eexxx		
	PLCC32	/gw29eexxx	PLCC32	
W29C020/C	DIP32	/gw29eexxx		
	PLCC32	/gw29eexxx	PLCC32	
W29C040	DIP32	/gw29eexxx		
	PLCC32	/gw29eexxx	PLCC32	
W29EE011	DIP32	/gw29eexxx		
	PLCC32	/gw29eexxx	PLCC32	
W29EE512	DIP32	/gw29eexxx		
	PLCC32	/gw29eexxx	PLCC32	
W39L040	PLCC32	/g29lvxxx	PLCC32	
W39L040A	PLCC32	/g29lvxxx	PLCC32	
W39V040AP	PLCC32	/g49lf00xa	PLCC32	
W39V040FAP	PLCC32	/g49lf00xa	PLCC32	
W49F002	DIP32	/g49f00x		Bottom Boot Block Protection und Reset-Pin
	PLCC32	/g49f00x	PLCC32	
W49F002B	DIP32	/g29fxxx		Bottom Boot Block Protection, ohne Reset-Pin
	PLCC32	/g29fxxx	PLCC32	
W49F002N	DIP32	/g29fxxx		Top Boot Block Protection, ohne Reset-Pin
	PLCC32	/g29fxxx	PLCC32	
W49F002U	DIP32	/g49f00x		Top Boot Block Protection und Reset-Pin
	PLCC32	/g49f00x	PLCC32	
W49F020	DIP32	/g29fxxx		Boot Block Protection
	PLCC32	/g29fxxx	PLCC32	

Hersteller	Winbond			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
W49V002AP	PLCC32	/g49lf00xa	PLCC32	
W49V002FAP	PLCC32	/g49lf00xa	PLCC32	
W77E58/P	DIP40	/gw78e58		
	PLCC44	/gw78e58	PLCC44	
W78E516B/P EPROM	DIP40	/gw78e516eprom		4 kB EPROM Loader Memory (LDROM)
	PLCC44	/gw78e516eprom	PLCC44	
W78E516B/P FLASH	DIP40	/gw78e516flash		64 kB FLASH Program Memory (APROM)
	PLCC44	/gw78e516flash	PLCC44	
W78E51B/P	DIP40	/gw78e51b		
	PLCC44	/gw78e51b	PLCC44	
W78E52B/P	DIP40	/gw78e52b		
	PLCC44	/gw78e52b	PLCC44	
W78E54/B/P/M	DIP40	/gw78e54		
	PLCC44	/gw78e54	PLCC44	
	TQFP44	/gw78e54	TQFP44	
W78E58/P	DIP40	/gw78e58		
	PLCC44	/gw78e58	PLCC44	
W78LE812/P	DIP40	/gw78e52b		
	PLCC44	/gw78e52b	PLCC44	

Hersteller	Xicor			
Baustein	**Bauform**	**Mnemonic**	**Adapter**	**Bemerkung**
X25020	DIP8	/gx25020		
X25040	DIP8	/gx25040		
X25080	DIP8	/g25080		
X25128	DIP8	/g25128		
X25160	DIP8	/g25160		
X25320	DIP8	/g25320		
X25640	DIP8	/g25640		
X25642	DIP8	/g25640		
X25F008	DIP8	/g25080		
X25F016	DIP8	/g25160		
X25F032	DIP8	/g25320		
X25F064	DIP8	/g25640		
X25F128	DIP8	/g25128		
X28HC64	DIP28	/g28c64b		
	PLCC32	/g28c64bplcc	PLCC32	