E-LAB

Production Programmer System

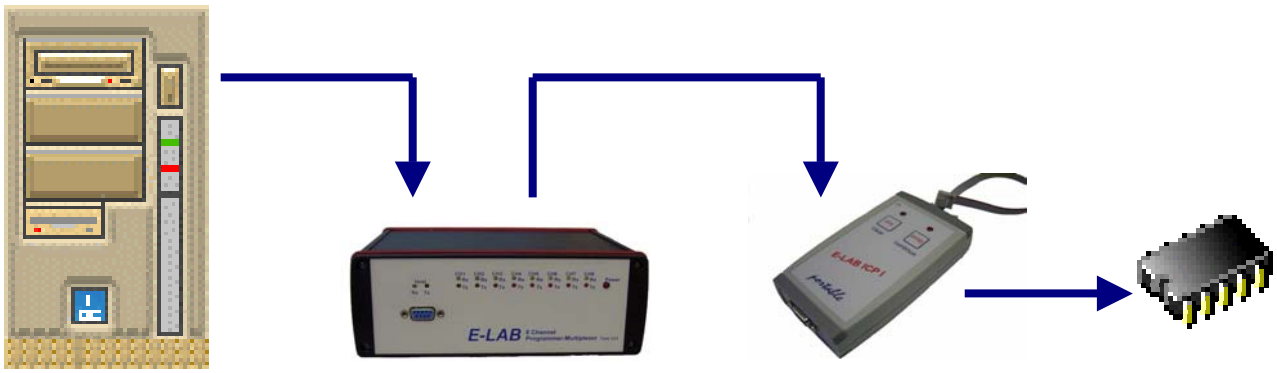
AVR *Multiplexer*

- **Produktions Programmiersystem für Atmel AVR CPUs**
- **Bis zu 8 simultane Programmierungen**
- **Steuerung über WIN32 DLL**
- **Steuerungs Programm vom Anwender selbst erstellbar**
- **Nur eine serielle Schnittstelle notwendig**
- **Optionaler Anschluss über USB oder Ethernet**
- **Durch Einsatz der E-LAB ICP Programmer flexible und schnelle InCircuit Programmierung**
- **Optionale Stückzahlen Limitierung und Passwortschutz**
- **Durch verschlüsselte Dateien ist ein re-Ingeneering ausgeschlossen**

Second edition März 2003

E-LAB Computers D74906 Bad Rappenau www.e-lab.de

In-Circuit Programmierung von Single Chips in der Produktion



In der Serien Produktion von elektronischen Baugruppen die mit Single-Chips bestückt sind, stellt sich immer wieder das Problem der In-Circuit Programmierung der CPUs innerhalb des Board Test Vorgangs. Herkömmliche ISP Programmiergeräte sind zwar preiswert und problemlos in der Handhabung, eignen sich jedoch nicht besonders gut für automatisierte Vorgänge.

Lässt sich der Programmierstart in den meisten Fällen durch einen Start Impuls noch relativ einfach steuern, so kommt es spätestens bei der Ergebnisauswertung zu einem Problem. Die zum Programmer zugehörige Steuersoftware ist in erster Line für manuellen Betrieb gedacht. Auch wenn das Protokoll für die Anbindung des Programmers an einen PC offengelegt ist, bedeutet das erheblichen Programmieraufwand bei der Erstellung eigener Treibersoftware.

Moderne Single-Chips wie der AVR benötigen z.B. eine grosse Anzahl von Fuse- und Lockbits, die das System alle handhaben muss. Auch ist es i.A. nicht Aufgabe der Produktions Verantwortlichen umfangreiche und komplexe Software zu erstellen, vor allen Dingen wenn der Produktionsstandort räumlich von der Entwicklung getrennt ist.

Fast unlösbar wird die Aufgabe, wenn im Nutzen programmiert werden soll, oder wenn mehrere CPUs sich auf dem selben Board befinden. Hier reichen zumindest die vorhandenen seriellen Schnittstellen nicht mehr aus, ganz zu schweigen von dem ebenfalls vervielfachten Software Aufwand.

Wenn das Programmiersystem aber komplett durch eine sogenannte DLL (dynamic link library) gesteuert werden kann, reduziert sich der Programmaufwand in dem Testsystem im wesentlichen auf allgemeine Kommandos wie Start programming und Resultat Auswertung.

Mit dem E-LAB Programmer-Multiplexer und seiner DLL lässt sich relativ einfach die In-Circuit Programmierung in ein vorhandenes Testsystem integrieren. Die Software kommuniziert mit ein paar wenigen DLL-Calls über die DLL mit dem Multiplexer. Die DLL und der Multiplexer leisten dabei die komplette Arbeit, ohne das Testsystem über Gebühr zu belasten.

Ein Produktions Programmiersystem besteht aus dem E-LAB Programmer-Multiplexer, 1 bis 8 E-LAB ICP Programmern und der WIN32 DLL.

E-LAB Computers **D74906 Bad Rappenau Germany**
Tel. 07268/9124-0 **Fax. 07268/9124-24**
WEB: www.e-lab.de **mail: info@e-lab.de**

Produktions Programmierer/Multiplexer

Der E-LAB Multiplexer dient als Bindeglied zwischen Test-Automat (oder PC) und ICP-Programmern in der Produktion von Boards, die mit Atmel AVRs bestückt sind. Der Multiplexer kann bis zu 8 ICP-Programmer gleichzeitig steuern und ist daher sehr gut für die Nutzen Fertigung bzw. Programmierung geeignet.

Das System besteht aus einer Windows DLL (WIN98..WIN2000) und dem Multiplexer Gerät mit daran angeschlossenen ICP Programmern.

Der Test Automat kommuniziert über die DLL mit dem Multiplexer. Die DLL muss in dem Test Automat installiert sein. Durch die Verwendung einer DLL kann das Steuerprogramm im Test Automat in fast jeder beliebigen Programmier-Sprache erstellt werden. Auch DLL-fähige Script Tools sind denkbar.

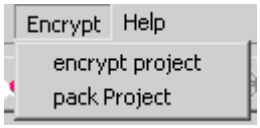
Der Multiplexer besitzt einen seriellen Eingang V24/RS232 (optional USB oder Ethernet) und 8 serielle Ausgänge. Die Netzversorgung ist umschaltbar zwischen 120V~ und 240V~

Die ACCUs der angeschlossen ICP-Programmer werden durch den Multiplexer kontinuierlich geladen, so dass ein Ladegerät an die ICPs normalerweise nicht angeschlossen werden muss. Sollte der Multiplexer länger als 1..2 Wochen abgeschaltet werden, so sollten die ICPs mit ihren Ladegeräten verbunden werden.

Das System DLL-Multiplexer benötigt gepackte oder verschlüsselte Arbeitsdateien, die mit dem E-LAB Programm AVRprog.exe erstellt werden müssen. Diese Dateien enthalten alle notwendigen Informationen (Flash-HEX file, EEprom –HEX file, CPU-Typ, Lockbits, Fusebits etc). Eine Manipulation bzw. fehlerhafte Einstellung der Programmer durch die Fertigung ist damit ausgeschlossen. Optional (encrypted files) ist hier auch eine Stückzahlen Begrenzung und sogar ein Zielsystem abhängiger Password Schutz möglich. (ausführliche Beschreibung im ICP Handbuch).

Gepackte oder verschlüsselte Dateien

Die DLL kann zwei Datei Typen handhaben. Beide Datei Versionen müssen durch das E-LAB PC-Programm ***AVRprog.exe*** erstellt werden.



Gepackte Datei ist eine binäre Datei, die alle notwendigen Informationen enthält, jedoch *keine* Verschlüsselung, Passwörter oder Stückzahlen Begrenzung. Sie darf deshalb im Gegensatz zu der verschlüsselten Version (encrypted) *nicht* durch „AddNewFile“ im Datei Pool angemeldet werden.
Die Datei Endung ist immer **.encr**

Verschlüsselte Datei: ist eine binäre Datei, die alle notwendigen Informationen enthält, *inklusive* Verschlüsselung, Passwörter und Stückzahlen Begrenzung. Sie muss deshalb *immer* durch „AddNewFile“ im Datei Pool angemeldet werden. Die Datei Endung ist immer **.pack**

DLL

Die DLL übernimmt im wesentlichen alle anfallenden Aufgaben und ist so konzipiert, dass das Steuerprogramm im Test Automat (oder PC) sich auf das wesentliche reduzieren lässt : Download von Programmen/Firmware in die einzelnen ICP-Programmer, Programmieren starten, Ergebnis auswerten. Alle Funktionen liefern als Funktions Ergebnis einen Text (Pchar) und ein Integer (res) als Aufzählungstyp (Enumeration) zurück.

Aufzählungstyp (Enumeration) der Funktions Ergebnisse:

resNone, noMux, MuxFound, noProg, progFound, MemError, progBusy, progIdle, progProtected, eraChip, eraEEp, eraFlash, prgEEp, prgFlash, verifyEEp, verifyFlash, errPwrDown, errSignature, errProtected, errNotEmpty, errVerify, progDone, dwnLoading, dwnLoadErrP, dwnLoadErrE, dwnLoadErrF, dwnLoadErr, invFile, invFName, FileExist, notFound, invPassword, limitExc, errProgType, resOk

Hierbei hat z.B. „resNone“ den Wert 0 und „MuxFound“ den Wert 2. ProgFound = 4

Die **Interface Funktionen** der DLL sind folgende:

```
procedure AddNewFile(FileName: PChar; var res : integer; resStr: PChar); stdcall; external 'MuxDLLN.dll'
// res = progBusy, invFile, invFName, invPassword, FileExist, resOk

procedure DeleteFile(FileName: PChar; var res : integer; resStr: PChar); stdcall;external 'MuxDLLN.dll';
// res = progBusy, invFName, notFound, errProtected, resOk

procedure GetFile(index : integer; var res : integer; resStr: PChar); stdcall; external 'MuxDLLN.dll';
// res = progBusy, notFound, resOk

procedure GetPasswd(var res : integer; resStr: PChar); stdcall; external 'MuxDLLN.dll'
// res = progBusy, resOk

procedure CheckMultiplexer(var ComPort: integer; var res : integer; resStr: PChar); stdcall; external
'MuxDLLN.dll';
// res = progBusy, noMux, MuxFound

procedure CheckProgrammer(MuxPort: integer; var res : integer; resStr: PChar); stdcall; external
'MuxDLLN.dll'
// res = progBusy, noMux, noProg, errProgType, progFound, MemError

procedure DownloadFile(ProjName: PChar; BurnCycles: Integer; MuxPort : Integer; var res : integer;
                       resStr: PChar); stdcall; external 'MuxDLLN.dll';
// res = progBusy, limitExc, errProgType, progProtected, dwnLoadErrP, dwnLoadErrF,
//       dwnLoadErrE, dwnLoadErr, noMux, resOk

procedure GetProjName(MuxPort : integer; var Age, res : integer; resStr: PChar); stdcall;
                      external 'MuxDLLN.dll';
// res = progBusy, noMux, noProg, errProgType, resOk

procedure GetACCUstate(MuxPort: integer; var ACCU, res : integer; resStr: PChar); stdcall;
                       external 'MuxDLLN.dll';
// res = progBusy, noMux, noProg, errProgType, resOk

procedure GetTargVolt(MuxPort: integer; var Volt, res : integer; resStr: PChar); stdcall;
                      external 'MuxDLLN.dll';
// res = progBusy, noMux, noProg, errProgType, resOk

procedure CheckDevice(MuxPort: Integer; var res : integer; resStr: PChar); stdcall; external 'MuxDLLN.dll';
// res = resNone, progBusy, noMux, noProg, errProgType, errPwrDown, errSignature,
//       errNotEmpty, errProtected

procedure ProgDevice(MuxPort: Integer; var res : integer; resStr: PChar); stdcall; external 'MuxDLLN.dll'
// res = progBusy, noMux, noProg, resOk  rest of states must be done with "GetProgStatus"
```

```
procedure GetProgStatus(MuxPort: integer; var res : integer; resStr: PChar); stdcall; external 'MuxDLLN.dll';
// res = resNone, limitExc, eraChip, prgEEp, prgFlash, progDone, errPwrDown,
//       errSignature, errProtected, errNotEmpty, errVerify

procedure DownLoadBlockF(Block : Pointer; BlockSize, dest, MuxPort : Integer; var res : integer;
                                    resStr: PChar); stdcall;  external 'MuxDLLN.dll';
// res = progBusy, noMux, noProg, dwnLoadErrF, resOk

procedure DownLoadBlockE(Block : Pointer; BlockSize, dest, MuxPort : Integer; var res : integer;
                                    resStr: PChar); stdcall;  external 'MuxDLLN.dll';
// res = progBusy, noMux, noProg, dwnLoadErrE, resOk

procedure AbortAll; stdcall; external 'MuxDLLN.dll'
```

Funktions Details

Im folgenden wird der Begriff „COMport“ für die serielle Schnittstelle des Steuerrechners benutzt (1..8).
Der Begriff „MuxPort“ steht für einen bestimmten seriellen Port des Multiplexers (0..7) und damit indirekt für einen speziellen am Multiplexer angeschlossenen ICP-Programmer. „Steuer System“ und „Host“ stehen hierbei für das Testsystem, auf dem die DLL und das zugehörige Programm installiert ist.

Verwaltungs Funktionen

Diese Funktionen können immer aufgerufen werden, ohne dass ein Multiplexer oder Programmer am System angeschlossen sind.

procedure **AddNewFile**(FileName: PChar; var res : integer; resStr: PChar);
// res = progBusy, invFile, invFName, invPassword, FileExist, resOk
Eine gepackte Datei (Projekt) wird dem Datei Pool hinzugefügt. Ist die Datei z.B. Password geschützt, und der Rechner entspricht nicht dem internen Datei-Password, wird die Datei mit dem Fehlercode „invPassword“ zurückgewiesen.
Achtung: Diese Funktion darf **nicht** für gepackte Dateien angewendet werden.

procedure **DeleteFile**(FileName: PChar; var res : integer; resStr: PChar);
// res = progBusy, invFName, notFound, errProtected, resOk
Eine Datei/Projekt wird aus dem Pool entfernt, wobei die Datei selbst nicht gelöscht wird. Hat dieses Projekt eine Stückzahlen Limitierung, wird eine Löschung mit „errProtected“ zurückgewiesen.

procedure **GetFile**(index : integer; var res : integer; resStr: PChar);
// res = progBusy, notFound, resOk
Diese Funktion dient zum Auslesen bzw. Auflisten der im Pool vorhandenen Projekte. Die Applikation inkrementiert solange den Übergabe Wert „index“, bis das Funktions-Ergebnis ungleich „resOk“ ist. Bei resOk zeigt „PChar“ auf einen String, der den Projekt-Namen enthält, der mit dem „index“ im Pool gelistet ist.

procedure **GetPasswd**(var res : integer; resStr: PChar);
// res = progBusy, resOk
Diese Funktion liefert als Ergebnis das Passwort, das Packungs Programm „AVRprog.exe“ benötigt, um Passwort geschützte Dateien speziell für diesen Ziel-Rechner erstellen zu können. In den meisten Fällen (innerbetriebliche Anwendung) ist weder eine Stückzahlen Limitierung und/oder ein Passwort sinnvoll oder notwendig. Bei Fernost Produktionen ist dieses Feature u.U. sehr segensreich. ☺

Initialisierung

Diese Funktionen werden vor dem Programmier Start einmal aufgerufen. Die ersten beiden sind absolut notwendig. Der Download erfolgt nur im Bedarfsfall.

procedure **CheckMultiplexer**(var ComPort: integer; var res : integer; resStr: PChar);
// res = progBusy, noMux, MuxFound
Diese Funktion sucht innerhalb des Systems auf den seriellen Schnittstellen nach dem Multiplexer. Hierbei kann mit dem Parameter „COMport“ eine ganz bestimmtes Port (1..8) vorgegeben werden. Alternativ kann mit dem Parameter 0 sämtliche Ports 1..8 durchsucht werden. Das kann jedoch zu Problemen führen, wenn serielle Treiber installiert sind (z.B. manche Mäuse), die nicht ganz Windows konform sind. Da der Multiplexer normalerweise immer am gleichen Port angeschlossen ist, ist eine feste Einstellung des COMports ohne Scanfunktion die bessere Wahl.
Im Parameter „COMport“ wird im Erfolgsfall (res = MuxFound) der gefundene COMport zurückgegeben. Die Funktion erzeugt ein Hardware Reset auf dem Multiplexer.

procedure **CheckProgrammer**(MuxPort: integer; var res : integer; resStr: PChar);
// res = progBusy, noMux, noProg, errProgType, progFound, MemError
Wurde der Multiplexer gefunden, muss über den Multiplexer die angeschlossen Programmer gesucht werden. Wird ein ICP-Programmer gefunden, wird dieser auch gleichzeitig initialisiert. Ein Download erfolgt jedoch nicht. Beim Funktionsaufruf bezeichnet der Parameter „MuxPort“ den zu suchenden bzw. zu prüfenden Programmer. Die Funktion erzeugt ein Hardware Reset auf dem ausgewählten Programmer. Wenn „MemError“ angezeigt wird muss ein neuer Download erfolgen.

procedure **GetProjName**(MuxPort : integer; var Age, res : integer; resStr: PChar);
// res = progBusy, noMux, noProg, errProgType, resOk
Diese Funktion dient zum Feststellen des aktuell in einem ICP-Programmer gespeicherten Projekts. Im Parameter wird die Nummer des gewünschten Programmers übergeben. Das Ergebnis der Funktion ist der im ICP eingespeicherte Projektnamen. Im Parameter „Age“ wird das Erstellungsdatum der gepackten Original Datei Datei zur weiteren Information enthalten.

procedure **GetACCUstate**(MuxPort: integer; var ACCU, res : integer; resStr: PChar);
// res = progBusy, noMux, noProg, errProgType, resOk
Wenn erfolgreich, gibt die Funktion als Resultat und im Parameter „ACCU“ den ACCU Ladezustand in Prozent zurück.

procedure **GetTargVolt**(MuxPort: integer; var Volt, res : integer; resStr: PChar);
// res = progBusy, noMux, noProg, errProgType, resOk
Wenn erfolgreich, gibt die Funktion als Resultat und im Parameter „Volt“ die Board Spannung (CPU) in 10mV Schritten zurück.

procedure **DownloadFile**(ProjName: PChar; BurnCycles: Integer; MuxPort : Integer; var res : integer;
resStr: PChar);
// res = progBusy, limitExc, errProgType, progProtected, dwnLoadErrP, dwnLoadErrF,
// dwnLoadErrE, dwnLoadErr, noMux, resOk
Diese Funktion ist normalerweise nur aufzurufen, wenn ein neues Projekt in einen bestimmten, am Multiplexer angeschlossenen Programmer geladen werden soll. Da die E-LAB ICP Programmer ACCU-gepuffert sind, muss nur bei einem Projekt Wechsel fur diesen Programmer neu herunter geladen werden.

Die Fehlermeldungen dwnLoadErrP, dwnLoadErrF und dwnLoadErrE dienen im Fehlerfall zur Information, wo das Problem aufgetreten ist (Download Parameter, Download Flash oder Download EEprom). Der Fehler „limitExc“ tritt auf, wenn versucht wird, eine Stückzahlen begrenzte Datei herunterzuladen, derer Begrenzung inzwischen erreicht wurde. (Encrypted Files)

Es ist selbsverständlich möglich, dass jeder angeschlossene Programmer ein eigenes Projekt erhält, so dass auch grosse Boards mit mehreren unterschiedlichen AVRs mit unterschiedlicher Firmware programmiert werden können.

Encrypted Files
Der Parameter BurnCycles ist nur von Bedeutung, wenn das Projekt Stückzahlen limitiert ist. Wenn die Begrenzung insgesamt bei 10000 liegt und es sind 4 ICP angeschlossen, die jeweils die gleiche Stückzahl programmieren sollen, so muss der Parameter „BurnCycles“ auf 10000/4 = 2500 gesetzt werden. Jeder Programmer kann jetzt 2500 Chips programmieren.
Ohne eine Limitierungs Vorgabe innerhalb des Projekts sollte der Parameter auf 0 gesetzt werden.

Packed files
Der Parameter **ProjName** muss auch den Pfad und File Extension einschliessen(.pack). Der Parameter **BurnCycles** ist ohne Bedeutung und sollte auf 0 gesetzt werden. Das File wird direkt in den Programmierer geladen ohne Beteiligung des File Pools.

Produktion

procedure **ProgDevice**(MuxPort: Integer; var res : integer; resStr: PChar);
// res = progBusy, noMux, noProg, resOk
Mit dieser Funktion wird ein Programmiervorgang gestartet. Ist das Ergebnis „res“ = resOk, war der Start erfolgreich. Sind alle relevanten ICP-Programmer gestartet, muss die Applikation jetzt kontinuierlich mit der Funktion „GetProgStatus“ den Multiplexer pollen, um für alle gestarteten ICPs den Status abzuholen.

procedure **GetProgStatus**(MuxPort: integer; var res : integer; resStr: PChar);
// res = resNone, limitExc, eraChip, prgEEp, prgFlash, progDone, errPwrDown,
// errSignature, errProtected, errNotEmpty, errVerify, MemError
Diese Funktion ist die eigentliche Arbeitsfunktion. Nach dem Programmier-Start Befehl „ProgDevice“ durch das Steuer System muss mit der Funktion „GetProgStatus“ kontinuierlich alle gestarteten Programmer gepollt werden. Der Multiplexer selbst erhält fortwährend Status Informationen von den gestarteten Programmern. Diese Informationen werden im Multiplexer temporär gespeichert. Das bedeutet, dass der Multiplexer immer die neueste Information jedes einzelnen Programmers bereit hält.

Werden diese Informationen vom Host nicht rechtzeitig abgeholt, so werden diese durch die neueste Status Information der Programmer ersetzt. Das bedeutet jedoch keinen wichtigen Informations Verlust, da in der Regel nur der letzte, abschliessende Status von Interesse ist.

Die Infos kommen in folgender Reihenfolge:
1. eraChip : die CPU wird gelöscht.
2. prgFlash : das Flash wird programmiert
3. prgEEp : das EEprom wird programmiert, falls so vorgegeben.
4. progDone : der Vorgang inkl. Verify ist komplett abgeschlossen.

Da an jeder Stelle ein Fehler auftreten kann, ist in diesem Fall der Fehlercode der letzte Status, der empfangen wird. Anderst ausgedrückt heisst das dass jeder Programmer so lange gepollt werden muss, bis entweder ein „progDone“ oder ein Fehlercode auftritt. Erst dann ist der Programmer fertig und bereit eine neue Aktion auszuführen.

procedure **CheckDevice**(MuxPort: Integer; var res : integer; resStr: PChar);
// res = resNone, progBusy, noMux, noProg, errProgType, errPwrDown, errSignature,
// errNotEmpty, errProtected
Diese Funktion kann nach z.B. nach dem erfolgreichen Programmiervorgang aufgerufen werden, um festzustellen, ob das Chip auslesegeschützt ist. Im Normalfall ist das jedoch nicht notwendig.

procedure **DownLoadBlockF**(Block : Pointer; BlockSize, dest, MuxPort : Integer; var res : integer;
resStr: PChar); stdcall; external 'MuxDLLN.dll';
// res = progBusy, noMux, noProg, dwnLoadErrF, resOk (* Block ins Flash *)

procedure **DownLoadBlockE**(Block : Pointer; BlockSize, dest, MuxPort : Integer; var res : integer;
resStr: PChar); stdcall; external 'MuxDLLN.dll';
// res = progBusy, noMux, noProg, dwnLoadErrE, resOk (* Block ins EEprom *)
Diese beiden Funktionen ermöglichen eine Manipulation des aktuellen Flash und EEprom Inhaltes im Programmer. Damit ist es möglich während der Produktion kontinuierlich z.B. Seriennummern fortzuschreiben. Jeder Download wird für den nächsten Programmiervorgang gültig. Die Blockgrösse sollte 256 Bytes nicht überschreiten. Für die korrekte Blockgrösse und die Zieladresse „dest“ ist der Programmierer selbst verantwortlich. Eine Überprüfung findet nicht statt.

procedure **AbortAll**;
Diese Prozedur ist notwendig, wenn das System zum Beispiel „hängt“ oder der Multiplexer mit dem Host oder mit einem oder mehreren Programmern ausser Synchronisation gekommen ist. Nach dem Kommando AbortAll befindet sich der Multiplexer im Grundzustand und kann neue Befehle empfangen.

Der sicherste Weg um aus einem instabilen Zustand herauszukommen ist jedoch eine komplette Neu-Initialisierung mit „CheckMultiplexer“ und „CheckProgrammer“. Die erste Funktion führt einen Hardware Reset auf dem Multiplexer durch, die weiteren jeweils ein Hardware Reset auf den einzelnen Programmern.

Beispiele und Sourcen

Zum Lieferumfang des Multiplexer Systems gehören die WIN32 DLL ***MuxDLLN.DLL***, ein ausführliches Delphi/Pascal Programm, das als EXE mitgeliefert wird sowie ein Visual Basic und ein C++ Programm. Die Quelltexte aller Programme sind ebenfalls enthalten.

Dem geübten Windows Programmierer dürfte es deshalb nicht allzu schwer fallen, ein eigenes, den Verhältnissen angepasstes Programm zu erstellen.

Import der DLL in Delphi

```
type
  tMxResult   =  (resNone, noMux, MuxFound, noProg, progFound, MemError, progBusy,
                  progIdle, progProtected, eraChip, eraEEp, eraFlash, prgEEp,
                  prgFlash, verifyEEp, verifyFlash, errPwrDown, errSignature,
                  errProtected, errNotEmpty, errVerify, progDone, dwnLoading,
                  dwnLoadErrP, dwnLoadErrE, dwnLoadErrF, dwnLoadErr, invFile,
                  invFName, FileExist, notFound, invPassword, limitExc, errProgType, resOk);

procedure AddNewFile(FileName: PChar; var res : integer; resStr: PChar); stdcall; external 'MuxDLLN.dll'
procedure DeleteFile(FileName: PChar; var res : integer; resStr: PChar); stdcall;external 'MuxDLLN.dll';
procedure GetFile(index : integer; var res : integer; resStr: PChar); stdcall; external 'MuxDLLN.dll';
procedure GetPasswd(var res : integer; resStr: PChar); stdcall; external 'MuxDLLN.dll'
procedure CheckMultiplexer(var ComPort: integer; var res : integer; resStr: PChar); stdcall;
                                external 'MuxDLLN.dll';
procedure CheckProgrammer(MuxPort: integer; var res : integer; resStr: PChar); stdcall;
                                 external 'MuxDLLN.dll';
procedure GetProjName(MuxPort : integer; var Age, res : integer; resStr: PChar); stdcall;
                               external 'MuxDLLN.dll';
procedure DownloadFile(ProjName: PChar; BurnCycles: Integer; MuxPort : Integer; var res : integer;
                               resStr: PChar); stdcall; external 'MuxDLLN.dll';
procedure GetACCUstate(MuxPort: integer; var ACCU, res : integer; resStr: PChar);
procedure GetTargVolt(MuxPort: integer; var Volt, res : integer; resStr: PChar);
procedure ProgDevice(MuxPort: Integer; var res : integer; resStr: PChar); stdcall; external 'MuxDLLN.dll';
procedure GetProgStatus(MuxPort: integer; var res : integer; resStr: PChar); stdcall; external 'MuxDLLN.dll';
procedure CheckDevice(MuxPort: Integer; var res : integer; resStr: PChar); stdcall; external 'MuxDLLN.dll';
procedure DownLoadBlockF(Block : Pointer; BlockSize, dest, MuxPort : Integer; var res : integer;
                                resStr: PChar); stdcall;  external 'MuxDLLN.dll';
procedure DownLoadBlockE(Block : Pointer; BlockSize, dest, MuxPort : Integer; var res : integer;
                                resStr: PChar); stdcall;  external 'MuxDLLN.dll';
procedure AbortAll; stdcall; external 'MuxDLLN.dll';
```

Import der DLL in Visual Basic

VB Interface

```
Const resNone          = 0
Const noMux            = 1
Const MuxFound         = 2
Const noProg           = 3
Const progFound        = 4
Const MemError         = 5
Const progBusy         = 6
Const progIdle         = 7
Const progProtected    = 8
Const eraChip          = 9
Const eraEEp           = 10
Const eraFlash         = 11
Const prgEEp           = 12
Const prgFlash         = 13
Const verifyEEp        = 14
Const verifyFlash      = 15
Const errPwrDown       = 16
Const errSignature     = 17
Const errProtected     = 18
Const errNotEmpty      = 19
Const errVerify        = 20
Const progDone         = 21
Const dwnLoading       = 22
Const dwnLoadErrP      = 23
Const dwnLoadErrE      = 24
Const dwnLoadErrF      = 25
Const dwnLoadErr       = 26
Const invFile          = 27
Const invFName         = 28
Const FileExist        = 29
Const notFound         = 30
Const invPassword      = 31
Const limitExc         = 32
Const errProgType      = 33
Const resOk            = 34
```

Private Declare Procedure **AbortAll** Lib "MuxDLLN.dll" ()

T.B.D.

Import der DLL in C++

```
enum _tMxResult
{
	resNone = 0,
	noMux = 1,
	MuxFound = 2,
	noProg = 3,
	progFound = 4,
	MemError = 5,
	progBusy = 6,
	progIdle = 7,
	progProtected = 8,
	eraChip = 9,
	eraEEp = 10,
	eraFlash = 11,
	prgEEp = 12,
	prgFlash = 13,
	verifyEEp = 14,
	verifyFlash = 15,
	errPwrDown = 16,
	errSignature = 17,
	errProtected = 18,
	errNotEmpty = 19,
	errVerify = 20,
	progDone = 21,
	dwnLoading = 22,
	dwnLoadErrP = 23,
	dwnLoadErrE = 24,
	dwnLoadErrF = 25,
	dwnLoadErr = 26,
	invFile = 27,
	invFName = 28,
	FileExist = 29,
	notFound = 30,
	invPassword = 31,
	limitExc = 32,
	errProgType = 33
};

typedef   enum _tMxResult  tMxResult;

void AbortAll();
```

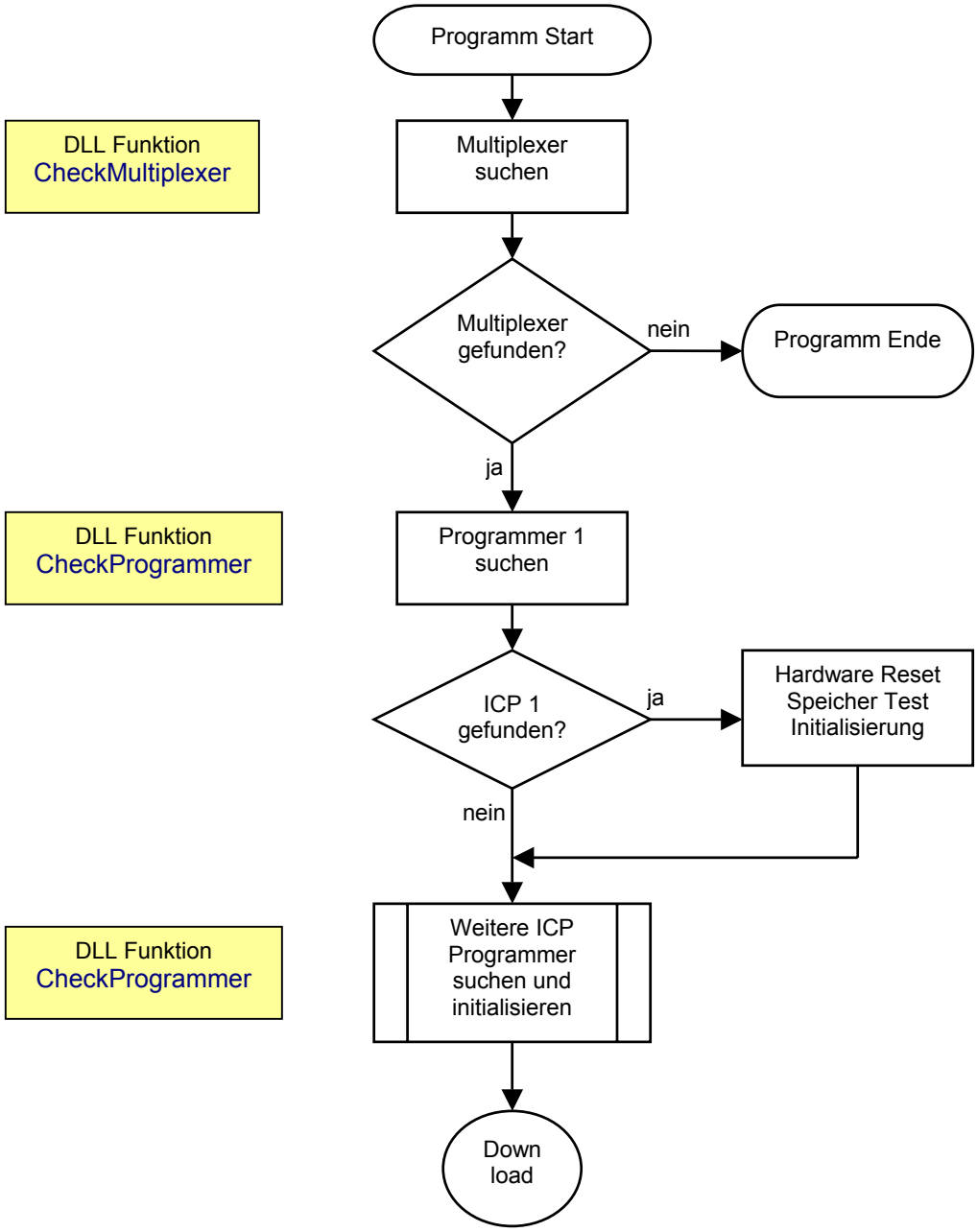
T.B.D.

Fluss Diagramme

Zum besseren Verständniss der Vorgehensweise beim Einsatz des AVR Multiplexers in der Produktion sind drei Diagramme vorhanden: 1. Initialisierung 2. Download 3. Produktion.

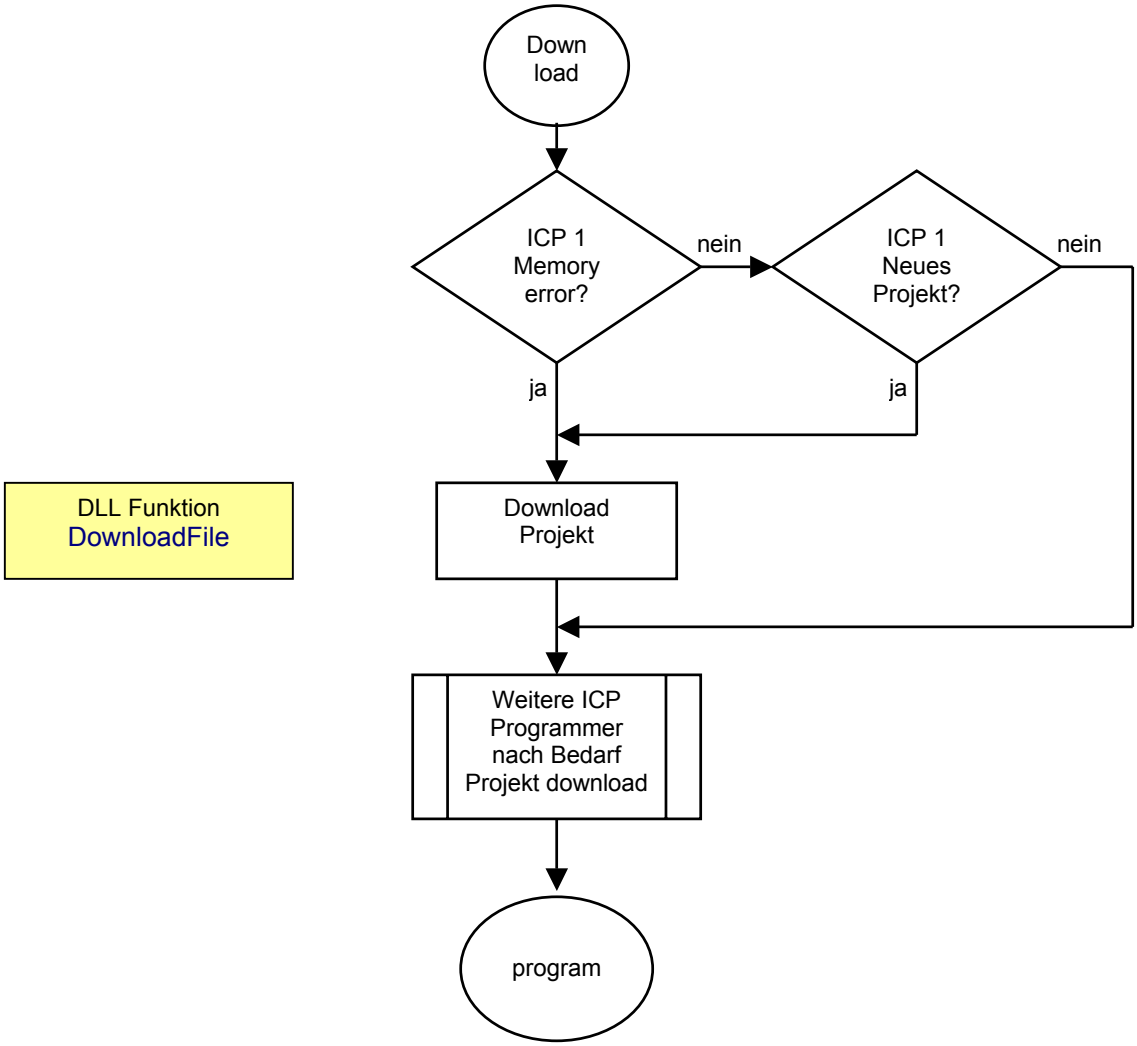
Initialisierung

Der Multiplexer muss an dem Testrechner angeschlossen sein (COM1..COM8). Auf dem Testrechner muss ein 32bit Windows laufen (WIN98/NT/WIN200). Die DLL „MuxDLLN.DLL“ muss sich in dem gleichen Verzeichnis befinden, indem sich auch das Anwender Steuerprogramm für den Multiplexer befindet.
Beim Starten des Steuerprogramms muss zuerst nach dem Multiplexer und dann nach daran angeschlossenen Programmern gesucht werden. Ist der Multiplexer gefunden, erfolgt ein Hardware Reset auf den Multiplexer. Wird ein ICP-Programmer durch den Multiplexer gefunden, erfolgt ebenfalls ein Hardware Reset auf diesen ICP. Der ICP meldet einen eventuellen Speicherfehler über den Multiplexer an das Steuerprogramm. Damit ist die Initialisierung abgeschlossen. Optional kann jetzt auch noch der Ladezustand des ICP-ACCUs abgefragt werden.



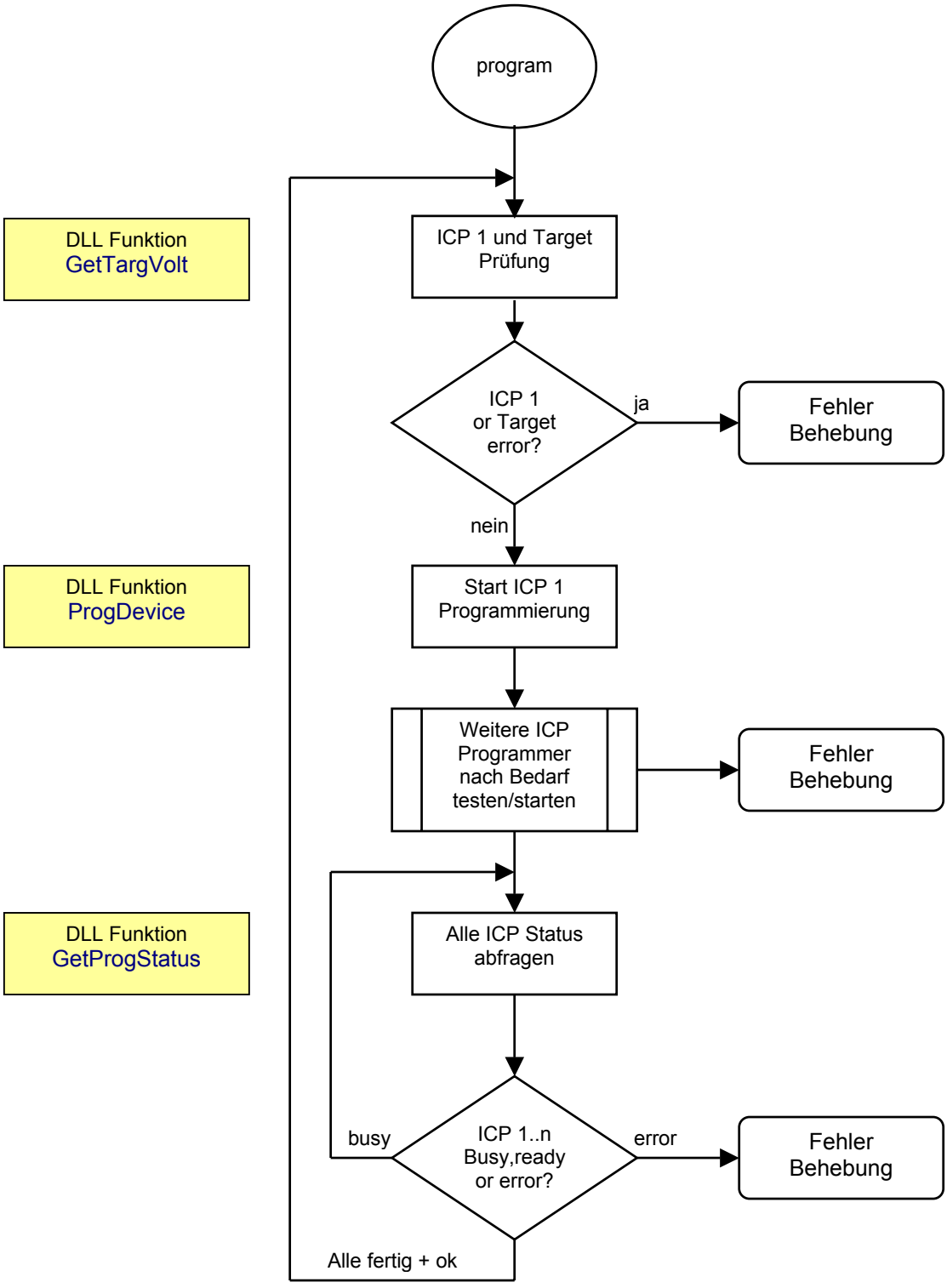
DownLoad

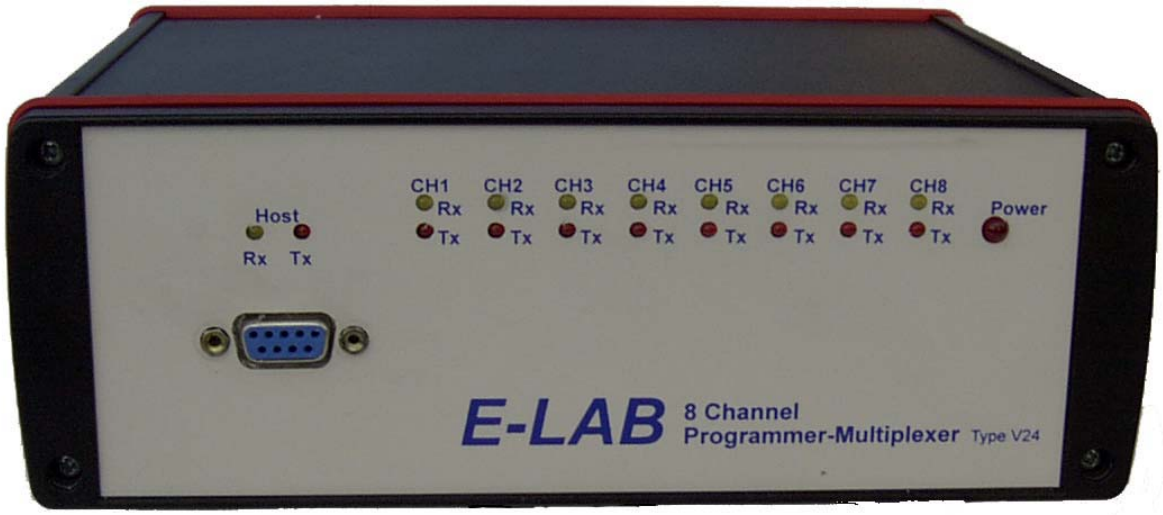
Ein Project Download in einen angeschlossenen Programmer ist nur notwendig, wenn entweder der Programmer während der Initialisierung einen Speichertest Fehler gemeldet hat, oder das geladene Projekt durch ein neueres ersetzt werden soll, oder ein anderes Projekt ab sofort vom Programmer verarbeitet werden soll.
Die ICP Programmer sind ACCU gepuffert und behalten ein einmal geladenes Projekt auch nach Abschaltung in ihrem Speicher. Dieser Speicher wird nach jedem PowerOn bzw. Reset mittels Checksummen geprüft.



Produktion

Der Programmiervorgang muss für jeden einzelnen angeschlossenen ICP-Programmer auch einzeln gestartet werden. Dabei gibt die Startfunktion als Ergebnis schon ein generelles OK oder Ausfall zurück. Ein Ausfall kann hierbei ein nicht vorhandener Multiplexer oder Programmer sein. Unmittelbar vor dem Programmierzyklus sollten deshalb alle relevanten Programmer mit der Funktion „GetTargVolt“ auf Funktion und die Ziel-CPU auf vorhandene Betriebsspannung geprüft werden.
Der eigentliche Programmierstatus wird nach dem Start aller Programmer durch eine generelle Poll Funktion abgeholt und analysiert. Wenn alle aktiven Programmer entweder ein „progDone“ oder eine Fehlermeldung abgegeben haben, ist der Programmierzyklus zu Ende und es kann ein neuer gestartet werden.





E-LAB Computers D74906 Bad Rappenau Germany
Tel. 07268/9124-0 Fax. 07268/9124-24
WEB: www.e-lab.de mail: info@e-lab.de