Portrait:	
Gerd Evers	OStR an der BFS für Assistenten in Bremen
Studium:	Uni Hamburg: Physik, Elektrotechnik.
Lehrtätigkeit seit:	Fast drei Jahrzehnte an der FOS (SZU) Bremen
Lehrtätigkeit bei:	Physikalisch technische Assistenten
	Technische Assistenten für Informatik
	FOS Naturwissenschaft (Physik, Informatik)
Theorie:	Physik, Elektrotechnik / Elektronik, Technisches Englisch
Labor:	Elektrotechnik, Physik, Angewandte Mikrocomputertechnik

Projekt: GAL-Evaluationsboard [GDU32]

Der *GAL*-ante Weg zur programmierbaren Logik ...

ein Wiederbelebungsversuch für die PLD-Klassiker GAL 16V8 und 20V8

Gerd Evers

Es mag es schon sein, daß die programmierbaren Logikbausteine, allem voran die einfachen unter Ihnen ... die GALs ... heute wirklich nichts mehr darstellen, was die Taktfrequenz eines Profis merklich höher schlagen lassen.

Es ist jedoch zu hoffen, daß die GAL-Klassiker 16V8 und 20V8 aus diesem Grunde noch nicht vollständig in Vergessenheit geraten sind, denn man würde diesen logischen Multitalenten sicherlich unrecht tun und das hätten sie nun wirklich nicht verdient.
Ganz gewiß repräsentieren genannte GALs nicht den letzten Stand der Technik , aber sie bilden das Bindeglied zwischen den ersten PLDs, den PALs und der Spitze der heutigen Logiktechnologie, den hochintegrierten, komplexen ispLSI PLDs.

Nebenbei sollte man sich stets vergegenwärtigen, daß es nicht nur Profis gibt, sondern eine Vielzahl Interessierter, die nicht schon alles wissen .. können .. müssen, aber den Drang und das Recht haben, Neues zu erfahren.

Das hier vorgestellte Projekt ist wieder aus dem Tätigkeitsfeld des Autors heraus entstanden.

Wer sich durch die Geheimnisse der TTL-Technik hindurchgequält und das PAL-Konzept mit seinen nicht zu leugnenden Mängeln kennengelernt hat (u.a. die für den Einsteiger zu große Typenvielfalt, die nur einmalige Programmierbarkeit) konnte plötzlich mit Zuversicht in die *logische* Zukunft blicken, denn ... es begab sich die Zeit ... in der das Gal entwickelt wurde, der Wegbereiter für die ispLSI-PLDs.

Wer sich mit den GALs vertraut gemacht hat, für den ist der Weg frei zu den PICs, den Mikrocontrollern, den Mikroprozessoren usw.

Zusammenfassend läßt sich sagen,
Um die ispLSI-PLDs, in denen prinzipiell GALs implementiert sind, verstehen zu können, zahlt es sich mit Sicherheit aus, sich nicht nur theoretisch mit den GALs anzufreunden,
(mal die eine oder andere Simulation laufen zu lassen), sondern deren vielfältigen

Einsatzmöglichkeiten konkret im Hardwaretest zu erleben.
Das Ableben einer Hardwarekomponente als Folge eines fatalen Softwarefehler hinterläßt beim Anwender einen erheblich tieferen Eindruck, als es ein Simulationsprogramm jemals vermag.

Genau diese Erfahrungen waren der Anstoß dafür, im Rahmen der Aus- und Weiterbildung nach einem geeigneten Einsteiger-Kit für die Einarbeitung in die Geheimnisse der GALs zu suchen.

Konkret wurde eine Hardware gesucht,

- bei der sich alles, was zur Bewältigung der anstehenden Problematik relevant ist, On Board befindet
- bei der auf alle Input- und Outputpins der genannter Standard GALs zugegriffen werden kann.
- bei der eine externe stabilisierte Spannungsversorgung entfällt
- bei der ein externer Taktgenerator entfällt
- bei der alle Ausgangszustände durch LEDs angezeigt werden
- bei der eine Sieben-Segmentanzeige implementiert ist
- bei der sich mehrere Einzelkomponenten zu einer komplexen Anordnung erweitern lassen
- bei der es möglich ist, analoge Signalquellen in digitale Signalquellen zu konvertieren

Die Vorgabe war: Das *etwas andere* GAL-Entwicklungsboard - *nicht nur* für den Einsteiger
Das Ergebnis ist: Die **G**AL **D**evelopment **U**nit **[GDU32]**
[**3**. Entwurf mit **2** Nachbesserungen] - so entstehen Namen

Auf die möglicherweise erstaunte Frage des Lesers, warum erst jetzt nach einem entsprechenden Board gesucht wurde, ist prinzipiell leicht zu beantworten.
Leider ist es eine Tatsache, daß die mit der mit der praktischen Vermittlung technischer Sachverhalte verbundene Hard- und Softwarebeschaffung vielerorts dem aktuellen Stand der Technik weit hinterherhinkt ... aus welchen Gründen auch immer ...

Die Vermutung, es müsse ein reichhaltiges Angebot an GAL- Entwicklungsboards geben, die den gestellten Anforderungen entsprechen, wurde leider enttäuscht.
Das Suchergebnis bewegte sich nahe der Null-Linie.

Für die ehrwürdige TTL-Technik gibt es recht gute Experimentiersets
(ITT-Trainer und deren Nachfolger).

Auf den praktischen Einsatz von PALs im Rahmen der Aus- und Weiterbildung sollte man wegen der teuren Programmiergeräte für den unterrichtlichen Einsatz und der viel zu hohen Materialkosten wegen der nur einmaligen Programmiermöglichkeit ohnehin verzichten.

Für die Lattice ispPLDs-Bausteine (ispLSIxxx, ispGALxxx, ispGDSxxx) gibt es recht gute und preislich akzeptable Starterkits wie z.B. das PLD-Einsteigerset der Firma ELV oder den (der ? das ?) Volks-PLD von K. Engelhardt / ELRAD 94 Heft 10ff].

Für die besagten GAL-Oldies 16V8 und 20V8, die sich gerade für den Einstieg in die Welt der PLDs wegen der noch vorhandenen Überschaubarkeit hervorragend eignen, wurde die Suche nach einem Board, welches obigen Anforderungen entspricht, bald aufgegeben und die

Entscheidung getroffen, selber ein entsprechendes Board zu entwickeln.
Das Entwicklungsresultat , die GDU32 soll hier vorgestellte werden.

In mehreren Beiträgen soll der Versuch unternommen werden, den PLD-Interessierten soweit mit den GALs 16V8 und 20V8 vertraut zu machen, daß für anstehende Problemstellungen eigene Lösungen gefunden werden können.

In diesem ersten Beitrag werden die Hardware sowie die GAL-Entwicklungs- und Testmöglichkeiten der GDU vorgestellt.

In weiteren Beiträgen werden Möglichkeiten vorgestellt, mit denen sich elementare Gatter und Flip Flops mit einem GAL realisieren lassen.

Aufbauend hierauf wird der Leser an komplexere Anwendungen mit GALs herangeführt.
Es werden unterschiedliche Möglichkeiten zur Ansteuerung einer Siebensegmentanzeige, die Realisierung eines 8 Bit Binärzählers und ein 8 Bit Schieberegister mit Paralleleingabe vorgestellt.

Da es problemlos ist, mehrere GDU-Boards parallel zu schalten, wird als *Höhepunkt* ein mehrstufiger BCD Zähler mit 7 Segment Dekoder und der Aufbau für eine parallele und serielle Datenübertragung (Grundprinzip des I2 C-Bus) gezeigt.

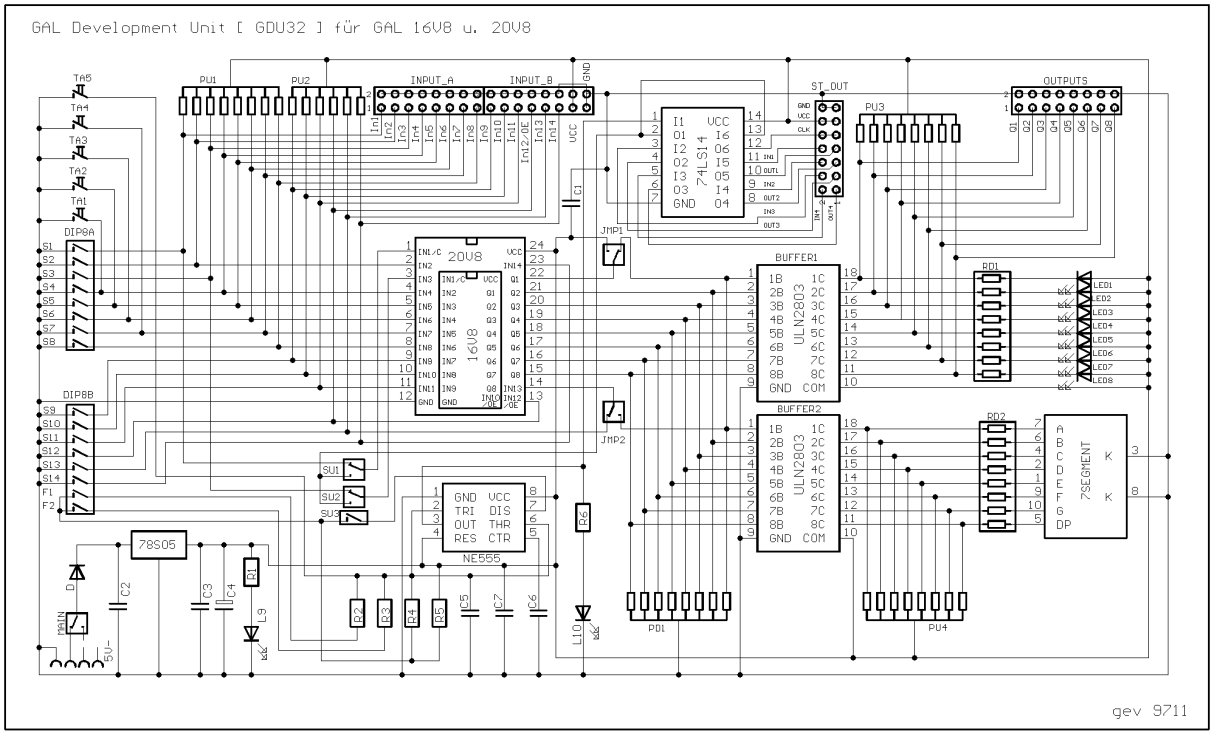
Um die Entwicklung der Logikgleichungen nachvollziehen zu können, ist in diesem Zusammenhang *an geeigneter Stelle* ein anwendungsbezogener Mini Logikkurs geplant. Hier sind die im Beitrag verwendeten Begriffe wie Wahrheitstabelle, disjunktive Normalform u.s.w. für den Neueinsteiger zu klären bzw. für den Nicht-Neueinsteiger zumindest in Erinnerung zu rufen.
In Verbindung mit dem Aufbau *galgerechter* Logikgleichungen wird es unumgänglich sein, sich mit dem de Morganschen Gesetz und dem Umgang mit dem KV-Diagramm vertraut zu machen.

Die wesentlichen Features der GDU auf einen Blick:

- konzipiert für die **GAL**s **16V8** und **20V8**
- leichtes Wechseln der Bausteine durch 24 poligen Textool Adapter
- konfigurierbar durch Jumper / Schalter
- **Sämtliche** *Eingänge* für die Ansteuerung durch *externe Systeme* herausgeführt
- **Sämtliche** *Eingänge* durch *DIP-Schalter* beschaltbar
- **4 Eingänge zusätzlich** mit *Tastern* belegt
- **Sämtliche** *Ausgänge* zur Ansteuerung einer 8stelligen LED-Reihe, eines 16-poligen Pfostensteckers (gepuffert) und einer Sieben-Segment-Anzeige (separat gepuffert) herausgeführt
- **Taktgenerator** *mit einstellbarem Taktmodus ON Board.*
  Betriebsmodus: Manuell und Automatik [4 Taktfrequenzen]
  Taktanzeige durch LED
- Gepuffertes, aufbereitetes Taktsignal über Schmitt-Trigger herausgeführt
- Bereitstellung von 4 Gattern zur Pufferung oder Signalanpassung mittels Schmitt-Trigger
- Spannungsversorgung in Verbindung mit einem AC-Adapter [> 7.5 V] *ON BOARD* [5V Regler, verpolungsgeschützt, LED-Anzeige, PWR-On Schalter]

<u>Schaltungsbeschreibung im einzelnen</u>:

<u>Stromlaufplan der GDU32:</u>



<u>Problemlosen Wechseln der GALs durch Textoolfassung:</u>

Da sich die GALs nicht auf der GDU programmieren lassen (Gruppe der nispSSI-PLDs) ist vor allem beim Einsatz in der Ausbildung mit einem häufigen Wechseln der Bausteine zwischen dem Testboard und dem Programmiergerät zu rechnen. Um die hieraus resultierende extreme Belastung sowohl für die Bausteine als auch für eine Standard IC-Fassung zu reduzieren, wurde das Board trotz der hohen Kosten mit einer 24poligen Textoolfassung ausgestattet.

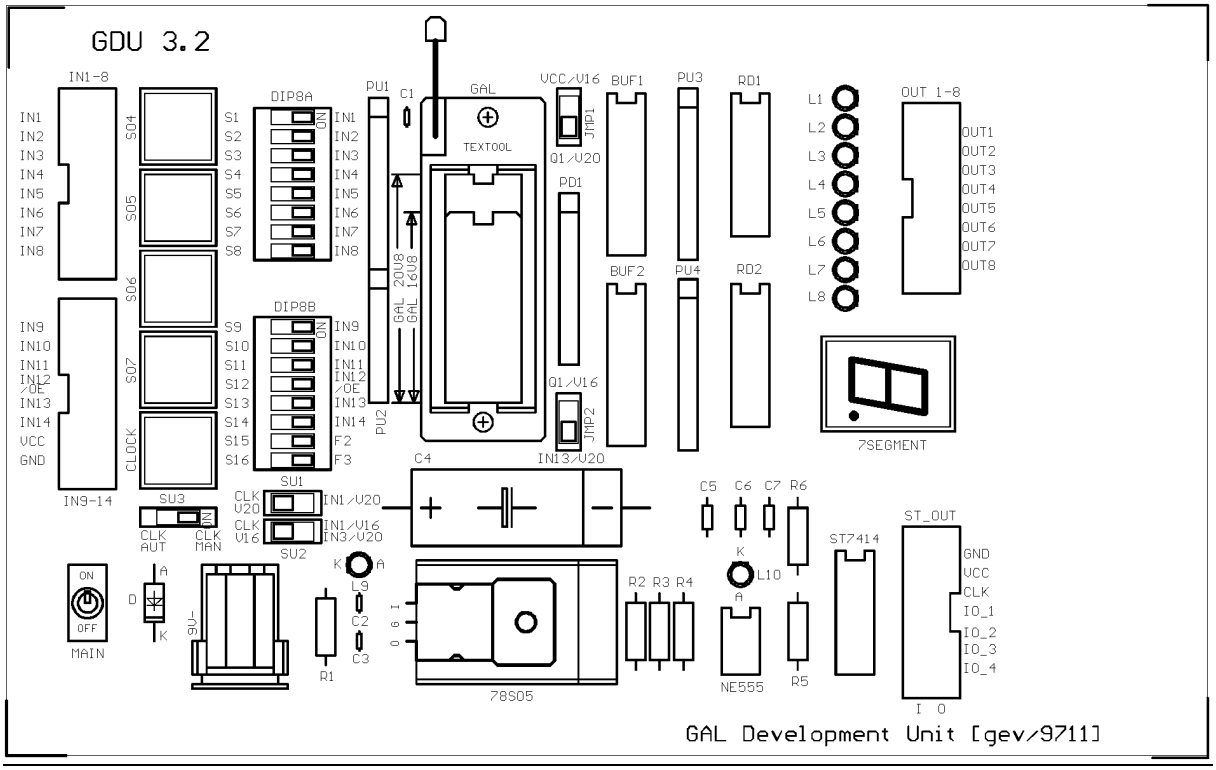
<u>Eine *gemeinsame* Fassung für beide GAL-Typen:</u>

Obwohl das Pinout des GALs 16V8 und 20V8 für die Boardkonzeption ausgesprochen ungünstig ist, erschien es trotzdem sinnvoll, sich schon aus Kostengründen auf eine gemeinsame Textoolfassung zu beschränken und wegen der versetzt angeordneten Ausgänge einen "Pinkompromiß" bei einem der GAL-Bausteine einzugehen.
Auf diese Besonderheit wird an gegebener Stelle näher eingegangen.

<u>Korrektes Einsetzen der GALs:</u>

Die korrekte Positionierung der GALs ist dem Bestückungsplan oder dem Bestückungsdruck des Boards zu entnehmen.

Bestückungsplan der GDU32:



Die Eingänge:

Alle Eingänge der GALs werden durch Pull-Up-Arrays im nichtbeschalteten Zustand (DIP-Schalter offen / TA1 - TA4 nicht gedrückt) auf den logischen Pegel 1 (HIGH) gezogen.

Die 10 Eingänge des GALs 16V8 bzw. 14 Eingänge des GALs 20V8 werden an eine zweireihige Pfostensteckerleiste geführt und stehen somit für die Ansteuerung durch externe Steuereinheiten zur Verfügung.

Wie der Schaltplan zeigt, nehmen die mit den Pins 1, 3 und 14 verbundenen Eingangsleitungen eine Sonderstellung ein. Während die übrigen Eingangsleitungen direkt zu den den DIP-Schaltern DIP8A und DIP8B geführt werden, laufen genannte Leitungen über die Umschalter SU1 und SU2 bzw. über den Jumper JMP2 zu den DIP-Schaltern S1, S3 und S13 bzw. zu den entsprechenden Eingangspins des Pfostensteckers.
Bei der im Schaltbild dargestellten Schalter- und Jumpereinstellung kann auf alle zur Verfügung stehenden Eingänge des GAL20V8 zugegriffen werden. Beim Einsetzen eines GAL16V8 sind die Eingänge S1 und S2 und damit der Umschalter SU1 ohne Bedeutung. Der Jumper JMP2 hingegen muß umgesteckt werden, da sich an der Pinposition 14 in diesem Falle der Ausgang Q8 des GAL16V8 befindet (Vgl. Titel: Die Ausgänge).

Die beschriebene Pinbelegung bezieht sich auf den Einsatz der Bausteine im Simple und Complex Mode ohne Registeroption.
Bei dem Einsatz der Bausteine im Complex Register Mode bekommen der Pin 1, 3 und 13 eine besondere Funktion zugeteilt. Pin 1 und 3 werden zu Clock Eingängen (CLK) und über den

Eingangspin 13 wird die Output Enable Funktion gesteuert. Die Schalterposition von SU1 (20V8) bzw. SU2 (16V8) sind entsprechend zu ändern, um das mit Hilfe des Schmitt-Triggers 74LS14 aufbereitete Taktsignal auf den entsprechenden Clockeingang aufzuschalten. Um die Ausgangsbuffer der Bausteine zu aktivieren, ist der DIP-Schalter S12 auf ON (LOW) zu schalten.

Parallel zu den DIP-Schaltern S4 - S7 bzw. den Eingängen IN4 - IN7 stehen Taster zur Verfügung, die das Austesten zahlreicher Anwendungen erleichtern (Testen der Funktion elementarer Gatter, Flip-Flops, Zähler u.a.m.)
Um die Taster sowohl beim 16V8 als auch beim 20V8 nutzen zu können, sind diese so gelegt, daß der Zugriff auf die "Nur-Eingänge " IN2 - IN5 des 16V8 bzw. IN4 - IN7 des 20V8 gewährleistet ist.

Anmerkung zu den GAL Betriebsmodi:
Die oben angesprochenen Betriebsmodi der GALs (Simple, Complex, Register) werden in einem der folgenden Beiträge soweit erläutert, wie es für das Verständnis der dort vorgestellten Anwendungsbeispiele erforderlich erscheint.

Die Ausgänge:

Wie bereits an anderer Stelle erwähnt, ist die versetzte Pinanordnung der GAL Ausgänge für die Konzeption eines Entwicklungsboards mit einer gemeinsamen IC-Fassung ein Handicap. Am günstigsten wäre eine Lösung, bei der sich alle 8 Ausgänge des 20V8 mit einem entsprechenden Umschalter um eine Position nach unten verschieben ließen, um auf dieselben Anzeigeelemente zugreifen zu können. Von einer derartigen Lösung wurde abgesehen und stattdessen die hier vorgestellte Kompromislösung realisiert.
Der Kompromis besteht darin, daß bei dem GAL16V8 der Ausgang Q8 über den entsprechend gesteckten Jumper JMP1 auf den Pin 1 der beiden Buffer1 und 2 und damit auf die LED1 und das Element A der 7 Segmentanzeige gelegt wird.

Die Erfahrung hat in der Zwischenzeit gezeigt, daß diese " Handicap-Lösung " für die Ausbildung methodisch ein recht wertvoller Kompromis ist. Bei der Anpassung einer vorhandenen 20V8 Applikation an ein GAL16V8 oder umgekehrt bedarf es lediglich einer simplen Softwareanpassung , um die ursprüngliche Wertigkeit vordefinierter Ausgänge der veränderten Hardwareumgebung anzupassen. Dieses ist beispielsweise bei der Ansteuerung einer 7 Segmentanzeige, bei der Realisierung eines Zählers usw. von Bedeutung, da die ursprüngliche geforderte Bitmustergenerierung ansonsten kräftig durcheinander gerät und die 7 Segmentanzeige recht erstaunliche Symbole generiert.
Gerade hier zeigt sich in sehr anschaulicher Weise die Stärke der programmierbaren Logikbausteine (PLD) schlechthin.
Die Ausgangssignale der GALs werden durch zwei parallel geschaltete 8fach Darlington Arrays (ULN2803) gebuffert .
Wegen der offenen Kollektortechnik der ULNs sind die Ausgänge mit den Pull-Up-Arrays PU3 bzw. PU4 zu versehen.
Durch die Bufferung wird den GALs eine echte Überlebenschance eingeräumt, wenn sie dem Härtetest der (lernenden) Anwender ausgesetzt sind.

Das Pull-Down-Array PD1 sorgt dafür, daß die Pins einer leeren IC-Fassung oder sich im TRI-State befindliche GAL-Ausgänge auf den logischen Pegel 0 gezogen werden.
Wegen der Invertierung der Signale durch die ULNs liegen die Bufferausgänge und somit die

für externe Anwendungen zur Verfügung stehenden Ausgänge Q1 bis Q8 (Pfostenstecker OUTPUTS) dann auf logisch 1.

Die Anzeigeelemente:

Um den aktuelle Stand des Entwicklungsergebnisses, d.h. die logischen Zustände aller Ausgänge optisch kontrollieren zu können, sind die Ausgänge mit LEDs bestückt. Hierdurch lassen sich alle Applikationen anschaulich überprüfen, in denen einzelne Bits, Bitmuster oder Bitmusterfolgen manipuliert werden.
Die auf dem Board zur Verfügung gestellte Siebensegmantanzeige nimmt dem Softwareentwickler die Mühe ab, mit nur zusätzlichem Hardwareaufwand eine Logik realisieren und testen zu können, bei denen eine dezimale Anzeige des Logikergebnisses gefordert ist.
Die Erfahrung zeigt, daß die angewandte Elektrotechnik, ja häufig schon der Gedanke an einen heißen Lötkolben für einen eingefleischten Softwareentwickler ein Greuel ist und es oftmals erheblicher Schubkräfte bedarf, den Schritt von der soften Simulation bis zur hardware(n) Realität zu wagen.

Sowohl aus methodischen als auch aus pragmatische Gründen wurde eine Siebensegmentanzeige mit gemeinsamer Katode gewählt. Zum einen wird die ohnehin recht geringe Stromaufnahme des Boards hierdurch noch weiter reduziert, da einer leuchtenden LED immer ein nichtleuchtendes Element der Siebensegmentanzeige zugeordnet ist. Zum anderen wird der Anwender aufgefordert bzw. gezwungen, die vielfältigen Konfigurationsmöglichkeiten der GALs für seine spezielle Anwendung zu nutzen.
(Pegelanpassung mit Hilfe der vielfältigen Invertierungsmöglichkeiten der Ein- und Ausgangssignale)
Die für die Ansteuerung der LEDs und der Siebensegmantanzeige erforderlichen Strombegrenzungswiderstände (RD1 und RD2 / Arrays im DIL-Format) sind auf dem Board gesockelt. Falls sich der Entwickler durch die meist kryptischen Symbole der Siebensegmentanzeige irritiert fühlt oder die LED-Anzeige aus welchen Gründen auch immer stört, läßt sich das nichtgewünschte Anzeigeelement durch Herausziehen des entsprechenden Arrays außer Funktion setzen.
Bei einer Sockelung der Siebensegmentanzeige bietet sich zusätzlich die Möglichkeit, nochmals acht gepufferte Bits für externe Anwendungen zur Verfügung zu stellen.

GND und VCC:

Der GND-Anschluß beider GAL-Typen ist fest verdrahtet.
Der VCC-Anschluß ist nur beim GAL20V8 fest verdrahtet.
Beim Einsatz des GAL16V8 ist der Jumper JMP1 so zu stecken, daß VCC an Pin 22 liegt.
Der Stützkondensator C1 sorgt für die Unterdrückung von Spannungseinbrüchen.

Der Taktgenerator:

Beim Taktgeneratur wurde auf eine bewährte Standardschaltung zurückgegriffen, wie sie prinzipiell bereits in anderen vorgestellten Projekten (UniStep, BiStep, ModuStep / ELRAD 92) eingesetzt wurde.
Im vorliegenden Fall wurde jedoch auf jedes Bauelement verzichtet, welches für die hier geforderte Anwendung unbedeutend ist.
Das Resultat ist ein wahlweise freischwingender oder manuell anzustoßender Taktgenerator,

dessen Taktfrequenz oder Taktverzögerung von der Kombination der Widerstände R2, R3 und R4 in Verbindung mit der Kapazität C5 abhängt.
Die für die Taktfrequenz f bzw. Taktverzögerung D (Delay) zuständige Zeitkonstante läßt sich mit den DIP-Schaltern F1 und F2 einstellen. (Siehe Tabelle)

F1	F2	f in Hz (ca.)	D in ms (ca.)
OFF	OFF	1	1000
ON	OFF	10	100
OFF	ON	100	10
ON	ON	110	9

Ungefähres Takt- bzw. Delayverhalten

Der freischwingende Modus (Auto Mode) oder manuelle Modus wird mit dem Schalter SU3 eingestellt.
Die Funktion des Taktgenerators läßt sich mit Hilfe einer LED (L10) kontrollieren.
Da das Taktsignal erst beim Erreichen einer fest vorgegeben Triggerschwelle ausgelöst wird, kommt es, was beim manuellen Betrieb beachtet werden sollte, zu einer Schaltverzögerung, die von der eingestellte Zeitkonstanten abhängt. (Schalterstellung von F1 und F2)
Eine niedrige Taktfrequenz bedeutet eine lange Taktverzögerung und umgekehrt.
Wird der Taster TA5 (CLK) betätigt, spricht die Takt-LED sofort an, der Taktimpuls wird hingegen erst ausgelöst, wenn die LED erlischt. Um im manuellen Betrieb eine quasi unmittelbare Reaktion zu erhalten, ist es folglich sinnvoll, die Zeitkonstante für diesen Betrieb so kurz wie möglich zu wählen, d.h. die beiden DIP-Schalter F1 und F2 zu schließen.

Die Signalaufbereitung:

Das beschriebene Generatorverhalten hat direkt mit der Minimierung des Beschaltungsaufwandes zu tun, aber es ist für die Arbeit mit dem Board grundsätzlich ohne Belang.
Für die Arbeit mit den GALs ist hingegen von Bedeutung, daß die Flankensteilheit und die Amplitude (etwa nur 3 Volt) des Taktsignals zwar bei einer rein kombinatorische Anwendungen ausreicht, für den Registermodus jedoch zu fatalen Fehlergebnissen führt.
Aus diesem Grunde ist die TTL-gerechte Aufbereitung des Taktsignals mit Hilfe eines Schmitt-Triggers unerläßlich. Das Taktsignal des NE555 (Pin 3 / OUT) wird zu diesem Zweck über den 74LS14 an die Umschalter SU1 und SU2 geführt und kann bei Bedarf (Registermode) auf die Clockeingänge der GALs aufgeschaltet werden.
Das invertierte, aufbereitete und gebufferte Taktsignal steht am Pfostenstecker (ST_OUT) für allgemeine Anwendungen zur Verfügung.
Die zunächst widerwillige Implementation eines zusätzlichen Bausteines hat sich im nachherein als ausgesprochen vorteilhaft erwiesen.
Bei der logischen Auswertung der von analog arbeitenden Sensoren generierten Signale oder der Realisierung komplexer Experimentieraufbauten durch Zusammenschaltung mehrerer GDUs zu beispielsweise mehrstufigen Zählern, paralleler oder serieller Übertragungssysteme u.a.m lassen sich ohne zusätzlichen Hardwareaufwand nicht-TTL-gerechte Steuersignale problemlos aufbereiten.
Die hierfür vorgesehenen Ein- und Ausgänge des 74LS14 stehen dem Anwender am Pfostenstecker (ST_OUT) zur Verfügung.

Die Spannungsversorgung:

Zur Spannungsversorgung bedarf es nur weniger Anmerkungen.
Für den Laboreinsatz ist es nützlich, die Arbeit mit einer Entwicklungsumgebung aufnehmen zu können, ohne gerade nicht benötigten Labornetzteilen, Anschlußkabel, einen passenden Schraubendreher u.ä. suchen zu müssen. Aus diesem Grunde wurde ein DC-Adaper in Printausführung gewählt (nicht überall leicht zu beschaffen), um jedes Universal Gleichspannungsnetzteil [9V < U- < 12V / > 150mA] (überall leicht zu beschaffen) einsetzen zu können.
Von einem Versorgungsstecker in Klinkenausführung wurde im Gegensatz zu einigen Herstellern ähnlicher Entwicklungsboards abgesehen. Hier soll nicht der klassische Kurzschlußfall bei der Inbetriebnahme einer Hardware demonstriert werden !

Es gibt die unterschiedlichsten Methoden, das Ableben einer Elektronik tatkräftig zu unterstützen. Beim Einsatz eines Netzteiles mit einer Umpolungsmöglichkeit kann es durchaus sein, daß der 5 Volt Spannungsregler hiervon überhaupt nicht begeistert ist und die GALs durch einen solchen Anschlag ebenfalls in Mitleidenschaft gezogen werden.
Durch den Einbau der Diode D vor dem Spannungsregler ist die Schaltung verpolungssicher.

Für eine stabile Spannungsversorgung von 5 Volt sorgen der Spannungsregler 78S05 in Verbindung mit den Stützkondensatoren C3, C4 sowie dem Elko C4.

Das Entwickeln und Testen einer Logik macht ein häufiges Wechseln der GALs und damit verbunden, ein ebenso häufiges Ein- und Ausschalten der Versorgungsspannung erforderlich. Durch den Einbau des PWR-On Schalters (MAIN) wird eine unnötige mechanische Belastung der Versorgungsadapter vermieden.
Diese an und für sich nicht erwähnenswerte Selbsverständlichkeit wird ebenso wie die LED zur Anzeige der Versorgungsspannung aus Kostengründen oftmals eingespart.

Konfiguration des Boards:

Mit den beiden Jumpern JMP1 und JMP2 und den Umschaltern SU1 und SU2 wird das Board für den gewünschten GAL-Typ konfiguriert.
Die in der Tabelle unterhalb der GAL-Typen angegebenen Konfigurationshinweise beziehen sich auf die Positionen der Jumper und der Schalter, wie sie dem Bestückungsplan bzw. dem Bestückungsdruck zu entnehmen sind.

Betriebs Modus	Schalter / Jumper	GAL16V8	GAL20V8
Simple Mode und Complex Mode	JMP1 JMP2 SU1 SU2 S12	VCC/V16 Q1/V16 don't care IN1/V16 OFF	Q1/V20 IN13/V20 IN1/V20 IN3/V20 OFF
Register Mode	JMP1 JMP2 SU1 SU2 S12	VCC/V16 Q1 / V16 don't care CLK/V16 ON	Q1/V20 IN13 / V20 CLK/V20 IN3/V20 ON

Das optionale Arbeitsfeld:

Das Experimentierfeld mit 9 * 32 Durchkontaktierungen bietet Platz für den Aufbau kleinerer Hardwareanwendungen.

Die Bauteilliste:

Widerstände:
R1 - 470 / 1/4 W
R2 - 10k / 1/4 W
R3 - 220k / 1/4 W
R4 - 1 M / 1/4 W
R5 - 10k / 1/4 W
R6 - 470 / 1/4 W

Widerstandsarrays:
PU1 / 8 x 4k7 + 1
PU2 / 6 x 4k7 + 1
PU3 / 8 x 3k3 + 1
PU4 / 8 x 3k3 + 1
PD1 / 8 x 10k + 1
RD1 / 8 x 470 (DIL)
RD2 / 8 x 470 (DIL)

Die GDU32 wird als
Leerplatine, Bausatz und Fertigboard
angeboten

Bezugsquelle: **PCS-Prozeßrechnertechnik & WebDesign**
Matthias-Claudius-Weg 26
D - 27753 Delmenhorst

Informationen unter: **http://www.pcs-he.com**
oder
http://www.phyta.aka.de

E-Mail-Adresse: **pcs@ae-web.com**
oder
gevers@ae-web.com

Kondensatoren:
C1 - 100 nF
C2 - 100 nF
C3 - 100 nF
C4 - 220 uF / 25V / Elko, axial
C5 - 470 nF
C6 - 100 nF
C7 - 100 nF

Dioden:
D Diode 1N4001 o.ä.
L1 - L10 LED [3mm]
7Segment LTS547 / T718

ICs:
1 * NE555
2 * ULN2803
1 * 74LS14
1 * 78S05 mit Kühlblech

Taster / Schalter / Stecker:
4 * Digi-Taster
3 * Subminiatur Umschalter
2 * 8 fach DIP-Schalter
1 * Kippschalter / 1 * Um
1 * DC Adapter
4 * Pfostenstecker 16 polig / 2 reihig
1 * Pfostenstecker 14 polig / 2 reihig
2 * Pfostenstecker 3 polig / 1 reihig mit Jumper
1 * Textool IC Fassung Tex 24-3 / schmale Ausführung

Inhalt des Bausatzes:

Widerstände:	2 * 470 Ohm / 0.25 W
	2 * 10 kOhm / 0.25 W
	1 * 220 kOhm / 0.25 W
	1 * 1 MOhm / 0.25 W
Widerstandsarrays:	2 * 4k7 / 8 + 1 / einreihig
	2 * 3k3 / 8 + 1 / einreihig
	1 * 10k / 8 + 1 / einreihig
	2 * 470 Ohm / DIL-Format
Kondensatoren:	5 * 100 nF
	1 * 470 nF
	1 * 220 uF / 25 V / Elko / axial
Dioden:	1 * 1N4001
	1 * LED [3 mm] / grün
	1 * LED [3 mm] / gelb
	1 * LTS547 / T718 / 7 Segmentanzeige
IC-Fassungen:	1 * 8 polig
	1 * 14 polig
	2 * 16 polig
	2 * 18 polig
	1 * Textool IC Fassung Tex 24-3 / schmale Ausführung
ICs:	1 * NE555
	2 * ULN2803
	1 * 74LS14
	1 * 78S05 mit Kühlblech
Taster:	4 * Digitaster [rot]
	1 * Digitaster [blau]
Schalter:	3 * Subminiatur Umschalter
	2 * 8 fach DIP-Schalter
	1 * Kippschalter / 1 * Um
Adapter:	1 * DC Adapter
Steckerleisten:	2 * Pfostenstecker 3 polig / 1 reihig mit Jumper
	1 * Pfostenstecker 14 polig / 2 reihig
	1 * Pfostenstecker 16 polig / 2 reihig
	1 * Pfostenstecker 32 polig / 2 reihig
Leiterplatte:	1 * Leiterplatte GDU32

Projekt: GAL-Evaluationsboard [GDU32]

Teil 2: Ohne Theorie geht es nicht

Motto: Mit dem GAL sollte nicht nach TTL-Manier geBool(e)t werden ...

> *... ein Versuch, sich von der TTL-Denkweise zu befreien, um den Weg für PLD-gerechte Lösungswege frei zu machen ...*
> *oder*
> *... wohl dem, der gar nicht weiß, daß es überhaupt ein TTL-Gatter gibt ... ?*

Von: **Gerd Evers**

Wie bereits im ersten Beitrag geäußert, gilt wiederum:
Auch in der Theorie gibt nicht nur Profis, sondern ebenfals solche, die nicht schon alles wissen, aber großes Interesse haben, Neues zu erfahren oder bereits Bekanntes eventuell aus einem anderen Blickwinkel zu betrachten.

Um den Umfang der Ausführungen in akzeptablen Grenzen halten zu können,, wird "logistisches" Basiswissen vorrausgesetzt. Die elementaren Logikbausteinen wie BUFFER, INVERTER, AND, NAND, OR und NOR sowie deren mathematische Beschreibung mit Hilfe der Booleschen Algebra sollten bekannt sein.

Um eine gemeinsame Ausgangsbasis für die anstehenden Erörterungen zu finden, erscheint es jedoch trotzdem sinnvoll, dem konkreten Umgang mit GALs einen Logik-Auffrischungskurs voranzustellen.
Hier werden die in den Ausführungen verwendeten Begriffe und Konventionen sowie die nicht zu umgehenden theoretischen Hintergründe zur Beschreibung komplexerer, logischer Zusammenhänge so weit angesprochen, wie es für das Verständnis der jeweils vorgestellten Problematik erforderlich erscheint.

Nicht nur in diesem "Logik-Crash-Kurs", sondern generell bei allen folgenden Überlegungen steht nicht die für die Problemlösung zu entwickelnde Hardware im Mittelpunkt (TTL-Denkweise), sondern vielmehr die grundsätzliche Überlegung, welche Kriterien von einer konfigurierbaren, im Rohzustand halbfertigen und damit zunächst "nutzlosen" Logikeinheit erfüllt werden müssen, um daraus eine konkrete Logikanwendungen wie beispielsweise eine XOR-Logik, unterschiedliche Flip-Flops, Binärzähler, Schieberegister u.a.m.zu realisieren
(PLD-Denkweise).

Zur Vorbereitung einer PLD- bzw- GALgerechte Denkweise sind Festlegungen zu treffen, die auf eine einheitliche, übergreifende Art der Problembeschreibung abzielen und somit generell auch auf den Umgang mit komplexeren Logikeinheiten wie PICs, Microcontrollern und schließlich Prozessoren zu übertragen sind.

Damit die Chance besteht, den Weg von der Entstehung einer Logik-Idee bis hin zur Problemlösung mit einem universellen Logik-Device mit PLD bzw. GAL-Struktur in dem hier gesteckten Rahmen beschreiben zu können, wird besonderer Wert darauf gelegt, sich von Anfang an von einer für PLD-Betrachtungen ‘blockierenden TTL-Denkstruktur ' zu lösen und stattdessen eine PLD-gerechte Denkweise zu übernehmen.

Begriffe, Darstellungsmöglichkeiten, Beschreibungswerkzeuge logischer Zusammenhänge:

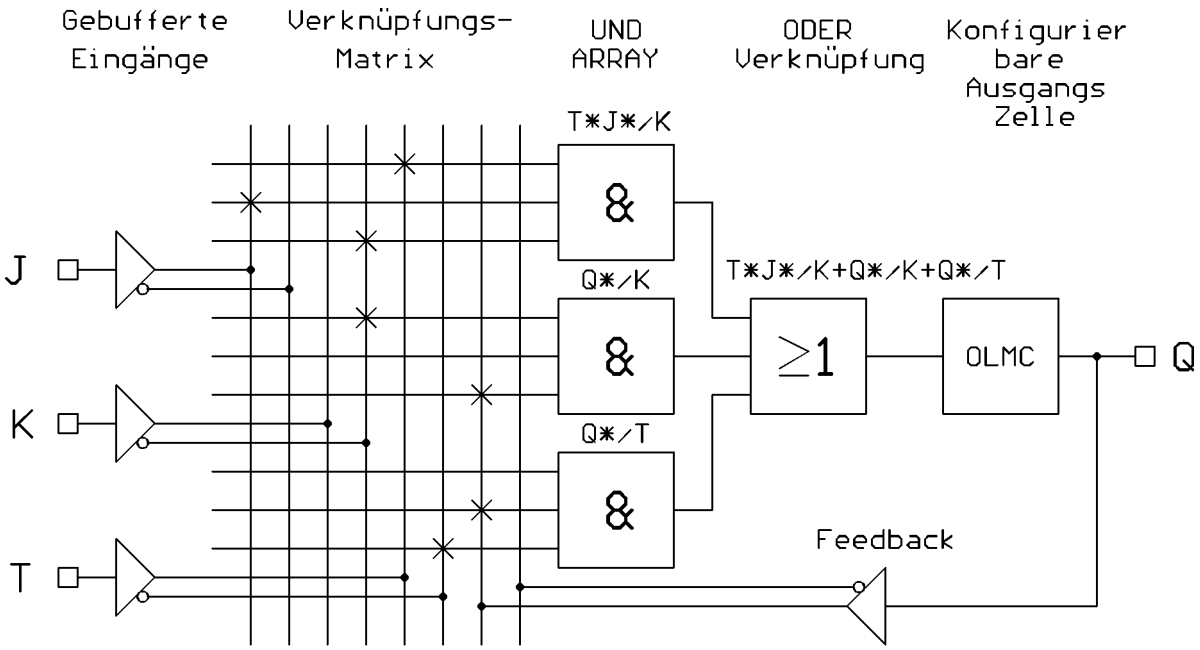
Mit Hilfe der im folgenden zugrunde gelegten Logikschaltungen werden Begriffe, Darstellungsmöglichkeiten und Werkzeuge vorgestellt, die für die Beschreibung auch komplexerer PLD-Anwendungen relevant erscheinen.

Die Umsetzung einer vorgegebenen Logikanforderung soll prinzipiell folgendermaßen realisiert werden:

- Formulierung der Problemstellung sowie zwingender Schlußfolgerungen und Vereinbarungen
- Minimierung des logischen Zusammenhanges mit Hilfe geeigneter Werkzeuge
- Mathematische Formulierung des Logikergebisses, d.h. der Logikgleichungen
- Eventuelle Umsetzung bzw. Anpassung der Logikgleichungen an die konkrete PLD-Entwicklungsumgebung

Um die aufgeführten Bedingungen umsetzen zu können, wird eine Standard-Hardware entsprechend der Abbildung 1 vorausgesetzt:

Abbildung 1:



Generelle Eigenschaften der Logikeinheit:

- Die Eingangsvariablen werden gepuffert und stehen in einer Verknüpfungsmatrix in entsprechend zugeordneten Spalten in invertierter und nicht invertierten Form zur Verfügung
- Auf die Eingänge der als Array angeordneten UND-Gatter kann in den entsprechenden zugeordneten Zeilen der Matrix zugegriffen werden
- Die Zeilen und Spalten der Matrix lassen sich an den Kreuzungspunkten verbinden oder trennen d.h. die Eingangsvariablen lassen sich beliebig miteinander verknüpfen
  Hierdurch steht eine programmierbare UND-Matrix zur Verfügung, mit der sich sog. Produkttherme bilden lassen. z.B. (E1 * E2) oder (/E1 * E2) usw. [lies: E1 UND E2 bzw. NICHT E1 UND E2]
- Offene Eingangsleitungen der UND-Gatter liegen vereinbarungsgemäß auf logisch 1
  Feste Verbindungen werden durch Punkte, frei konfigurierbare (programmierbare) Verbindungen durch Kreuze gekennzeichnet
- Die Ausgangssignale der UND-Gatter (die Logikergebnisse der Produkttherme) werden in einem fest verdrahteten ODER-Gatter miteinander verknüpft
- Das komplette Logikergebnis wird nicht direkt auf den Ausgang geschaltet, sondern läßt sich in einer konfigurierbaren Ausgangszelle den Peripherieanforderungen anpassen
  Durch diese konfigurierbare Ausgangszelle [bei realen GALs als Output Logic Macro Cell bzw. abgekürzt OLMC bezeichnet] stehen grundsätzlich folgende Ausgangsmodi zur Verfügung:

1. *Kombinatorischer Ausgang*
   - Ausgang einer aus reinen Verknüpfungsgliedern aufgebauten Logikschaltung
     Eine Änderung der Eingangsvariablen erscheint nur durch die Signallaufzeit verzögert am Ausgang
     Zwischen den Eingangszuständen (Eingangsvariablen) und dem Ausgangszustand (logisches Ergebnis) besteht ein fester funktioneller Zusammenhang, der sich mit den Gesetzmäßigkeiten der boolschen Algebra beschreiben läßt

- Kennt nur die beiden aktiven Zustände High (1) oder Low (0)
- Ein hochohmiger Zustand (Tristate) ist nicht möglich
- Keine Rückkopplungleitungen (Feedback) oder Speicherglieder (Latches / Register) vorhanden

2. *Invertierter Ausgang*
   - Der aktive Logikausgang kann wahlweise invertiert werden

3. *Tristate - Ausgang*
   - Der Logikausgang kann in die aktiven Zustände High und Low sowie in den hochohmigen Zustand geschaltet werden (Ausgang abschaltbar)

4. *Ausgang mit Feedback*
   - Das Logikergebnis wird am Ausgang zur Verfügung gestellt und zusätzlich vom Ausgang sowohl invertiert als auch nicht invertiert in die Logikmatrix zurückgeführt
     Durch diese Maßnahme steht das rückgeführte Logikergebnis als Eingangsvariable zur Verfügung und kann entsprechend der Anforderung mit anderen Variablen weiterverknüpft werden
     Das resultierende Logikergebnis (Folgezustand) ist somit eine Funktion des aktuellen Ausgangszustandes

5. *Register Ausgang [Ausgang mit transparentem D-Latch]*

   - Es läßt sich ein D-Latch zwischen das kombinatorische Netzwerk und den Logikausgang schalten.
     Das Logikergebnis wird auf den Eingang eines D-Latches geschaltet und steht nicht unmittelbar am Ausgang des D-Latches (Registers) zur Verfügung.
     Das Logikergebnis wird durch ein Clocksignals (CLK) in das Register eingeklinkt (ge-latched), zum Ausgang durchgeschaltet und schließlich am Ausgangspin sichtbar (transparent)

   Überall dort, wo ein Logikzustand " eingefroren " werden muß oder Schaltwerke wie Flip Flops, Zähler, Schieberegister u.a.m. zu realisieren sind, ist ein Registerausgang erforderlich

<u>Methoden und Werkzeuge zur Beschreibung logischer Zusammenhänge</u>

Im folgenden Beispiel sollen Methoden und Werkzeuge vorgestellt und erörtert werden, die zur Beschreibung und Lösung GAL-bezogener Problemstellungen erforderlich sind.

Die Problemstellung ist bewußt relativ komplex gewählt, damit möglichst viele der gebräuchlichen Lösungsstrategien vorgestellt werden können.

Problemstellung 1: Das in der Abbildung 1 dargestellte JK-Flip Flop in GAL Struktur ist zu entwickeln

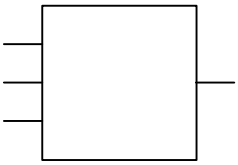
Konkrete Vorgaben:
Die Hardware ist so zu konfigurieren (Programmierung der Matrix), daß jeder der grundsätzlich möglichen Logikterme die in der Spalte Q der Wahrheitstabelle angegebenen Logikergebnisse ergibt.

Ergänzender Hinweis:

Da der geforderte Logikzustand des Ausganges Q nicht nur vom aktuellen Zustand der externen Variablen J,K und CLK abhängt, sondern ebenfalls vom aktuellen Zustand des Ausganges selber, ist Q in die Verknüpfungsmatrix zurückzuführen (Feedback) und entsprechend als Variable zu behandeln.

Der funktionelle Zusammenhang zwischen jeder Kombinationsmöglichkeit der Variablen (Logikterme) Q, K, J und CLK und dem geforderten Funktionsergebnis Q (resultierender Logikzustand des Ausganges) ist aus Gründen der Übersichtlichkeit und der Systematisierbarkeit in einer Wahrheitstabelle (Logiktabelle) zusammengestellt.

Blockschaltbild



Wahrheitstabelle des JK Flip Flops:

Nr.	CLK	J	K	Q	Q*	Funktion
1	0	0	0	0	0	Hold
2	0	0	0	1	1	Hold
3	0	0	1	0	0	Hold
4	0	0	1	1	1	Hold
5	0	1	0	0	0	Hold
6	0	1	0	1	1	Hold
7	0	1	1	0	0	Hold
8	0	1	1	1	1	Hold
9	1	0	0	0	0	Hold
10	1	0	0	1	1	Hold
11	1	0	1	0	0	Reset
12	1	0	1	1	0	Reset
13	1	1	0	0	1	Set
14	1	1	0	1	1	Set
15	1	1	1	0	1	Toggle
16	1	1	1	1	0	Toggle

Erläuterungen und Vereinbarungen zur Erstellung der Wahrheitstabelle:

- Die vier Variablen J, K, CLK und Q lassen sich 2 hoch 4 mal miteinander kombinieren (16 Logikterme)
- Die Kombinationsmöglichkeiten 0000, 0001, 0010, 0011 usw. werden konsequent so eingetragen, daß ihre binäre Wertigkeit von rechts nach links gelesen zunimmt.
  Die hier getroffene Vereinbarung ist nicht zwingend, erscheint jedoch überall dort sinnvoll, wo die Wertigkeit der Variablen eine Rolle spielt, wie z.B. bei Zählern, Schieberegistern usw.
- Die Einträge in die Spalte Q hängen *ausschließlich* davon ab, welcher logische Zusammenhang zwischen der jeweilig möglichen Kombination der Eingangsvariablen und dem Verknüpfungsergebnis vom Entwickler definiert wird.

In vorliegenden Fall ist gemäß der Problemstellung jeweils *die Logikbedingung* eingetragen worden, *die der eines JK-Flip Flops entspricht.*

Auswertung der Wertetabelle:

Das Ergebnis der Schaltfunktion für die *1-Auswertung* läßt sich anhand der Logiktabelle durch folgende Gleichung in der disjunktiven Normalform formulieren.

Q = (Q * /J * /K * /CLK)
+ (Q * /J * K * /CLK)
+ (Q * J * /K * /CLK)
+ (Q * J * K * /CLK)
+ (Q * /J * /K * CLK)
+ (/Q * J * /K * CLK)
+ (Q * J * /K * CLK)
+ (/Q * J * K * CLK)

Begriffsklärungen:

1-Auswertung: Auswertung *aller* Produktterme, bei denen der Ausgang Q auf 1 liegt
Disjunktive Normalform: ODER-Verknüpfung *aller* möglichen Produktterme
Disjunktiv: Nicht verbunden - hier: Isoliert nebeneinander, nicht Minimiert

Für die folgenden Ausführungen werden die angegebenen Vereinbarungen getroffen

- Es wird ein Forward-Slash [/] für die Kennzeichnung einer negierten Variablen verwendet
- Es wird ein Produktzeichen [*] für eine UND-Verknüpfung verwendet
- Eine UND-Verknüpfung wie beispielsweise (/D * MO) wird im folgenden als Produktterm bezeichnet
- Es wird ein Pluszeichen [+] für die ODER-Verknüpfung der Produktterme verwendet
- Die Logikgleichung hat immer die Struktur: Logikergebnis = (Produktterm1) + (Produktterm2) + ...

4. Werkzeuge zur Minimierung und Veranschaulichung logischer Zusammenhänge

In der disjunktiven Normalform (DNF) der Schaltfunktion werden alle Produktterme, die zu dem geforderten Ausgangspegel der Logik führen, *ODER-verknüpft*.
Ist die Anzahl der Produktterme, die zu einer 1 führen geringer als die Anzahl der Produktterme, die zu einer 0 führen, so sollte eine 1-Auswertung, ansonsten eine 0-Auswertung vorgenommen werden.

Trotz dieser Optimierungsmöglichkeit enthält die DNF-Formulierung häufig überflüssige Terme, die sich generell durch eine geeignete Maßnahme herausfiltern lassen.

4.1 Mathematische Minimierung der disjunktiven Normalform mit den Rechenregeln der booleschen Algebra

Beispiel: Eine Wahrheitstabelle möge zu der angegeben Logikgleichung [DNF] führen

Q = (E1 * E2 * E3) + (E1 * /E2 * E3)

Unter Anwendung der booleschen Algebra läßt sich ebenfalls schreiben:

Q = (E1 * E3) * (E2 + /E2)

Nach den Rechenregeln ergibt der Klammerausdruck (E2 + /E2) den Wert 1.
Somit ist es gleichgültig, ob die Variable E2 den Wert 1 ODER 0 führt.
E2 trägt nichts zu dem Logikergebnis bei, d.h. die Variable E2 ist "überflüssig ".

Folglich lautet die "minimierte" Funktionsgleichung : Q = (E1 * E3)

4.2 Theoreme von DE Morgan:

Auf eine Umformung und möglicherweise Minimierung der Logikgleichung mit den Theoremen von DE Morgan wird im Zusammenhang mit der GAL-gerechten Denkweise verzichtet.
Begründung:
Beim Umgang mit programmierbaren Logikeinheiten bis hin zum Prozessor ist eine Hardware-Minimierung, wie sie mit den DE Morganschen Theoremen meist angestrebt wird (Umformung von NANDs in NORs und umgekehrt / Zusammenfassung einzelnen Variablen zu komplexen Klammerausdrücken usw.) vollkommen fehl am Platze (TTL-Denkweise).
Die Umwandlung einzelner Variablen, wie z.B. Q = A * B in die äquivalente Form Q = /A + /B hat bei der Problemlösung mit PLDs eine untergeordnete Bedeutung.
Hier geht es ausschließlich darum, die Variablen mit Hilfe der vorgegeben frei programmierbaren Hardware in GAL Struktur so zu konfigurieren, daß der Anwender zu dem von ihm vordefinierten Logikergebnis gelangt.

5. Grafisches Verfahren zur Minimierung von Schaltfunktionen (KV-Diagramm)

Die angesprochene mathematische Minimierung ist bei Logikproblemen mit mehreren Variablen häufig recht mühsam und unhandlich.
Berücksichtigt man den mathematischen Hintergrund der Minimierungsstrategie und überträgt diese auf eine grafische Verfahrensweise, wie sie von Karnaugh und Veitch entwickelt wurde, so kommt man durchweg eleganter und sicherer zum Ziel.

Allgemeine Anmerkungen und Erläuterungen zum KV-Diagramm:

Im Rahmen dieses Beitrages wird darauf verzichtet, alle mit dem KV-Diagramm verbundenen Aspekte erschöpfend erörtern zu wollen. Aus der Sicht des Autors ist es nicht möglich, hier den mathematischen Hintergrund erschöpfend nachzuweisen, alle gebräuchlichen Darstellungsmöglichkeiten von KV-Diagrammen zu präsentieren, zu erörtern oder die unterschiedlichsten Auswertungsverfahren von KV-Diagrammen abzuhandeln.

Die für die hier anstehenden Problemstellungen wichtigsten Kriterien im Umgang mit einem KV-Diagramm sind im folgenden zusammengestellt.
Desweiteren wird auf entsprechende Fachliteratur verwiesen.

Grundsätzlicher Aufbau eines KV-Diagrammes

- Das KV-Diagramm ist prinzipiell nur eine andere Schreibweise der Wahrheitstabelle
- Jedem Logikterm der Wahrheitstabelle ist ein Feld im KV-Diagramm zugeordnet
- Die Anzahl der Felder entspricht der Anzahl der Logikterme
- n Variablen sind 2 hoch n Felder zuzuordnen
- Bei der Verarbeitung von mehr als zwei Variablen ist die Zuordnung der Variablen grundsätzlich so vorzunehmen, daß sich benachbarte Felder der Spalten oder der Zeilen nur in einer Variablen unterscheiden.

Auswertung eines KV-Diagrammes

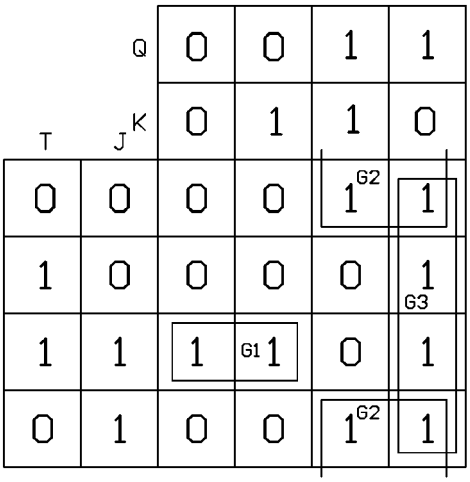
***Gruppenbildung**:*

- Die mit 1 ausgefüllten, benachbarten Felder sind in Gruppen zusammenzufassen.
- Eine Gruppe darf nur aus jeweils 2 hoch n Feldern (1, 2, 4, 8, 16 ...) bestehen.
- Die zu einer Gruppe zusammengefaßten Felder müssen eine geschlossene Einheit bilden
- Je größer die Anzahl der zu einer Gruppe zusammengeschlossenen Felder ist, desto kleiner wird der schaltalgebraische Ausdruck
- Gruppen können sich überlappen. Ein Feld kann auch Teil einer zweiten und dritten Gruppe sein
- Felder gegenüberliegender Ränder der KV-Tafel lassen sich zu Gruppen zusammenfassen, da diese als " logisch benachbart " einzustufen sind.

***Auswertung der Gruppen**:*

- Jede Gruppe repräsentiert einen Produktterm
- Bei der Verknüpfung der Variablen innerhalb einer Gruppe entfallen die Variablen, die sowohl mit ihren 1- als auch mit ihren 0-Werten an der Gruppenbildung beteiligt sind [Vgl. (E1 * /E1) = 1]
- Die Gruppen sind untereinander ODER-verknüpft

Nach diesen Vorgaben sieht das KV-Diagramm für das zu entwickelnde JK-Flip Flop wie dargestellt aus:



Gruppe 1 wird vollständig beschrieben durch:

/Q * J * T

Gruppe 2 (Randgruppe) wird vollständig beschrieben durch:

Q * /T

Gruppe 3 wird vollständig beschrieben durch:

Q * /K

Die Minimierung der disjunktive Normalform mit Hilfe des KV-Diagrammes führt zu eine Funktionsgleichung mit nur 3 statt 8 Logiktermen.

Das JK-Flip Flop wird folglich vollständig beschrieben durch: **Q = T * J * /K + Q * /K + Q * /T**

Die Abbildung 1 zu Beginn der Ausführungen zeigt die schaltungstechnische Umsetzung dieser Gleichung [JK-Flip Flop in GAL Technik]

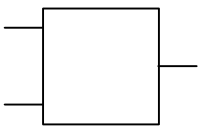
Der Autor **hofft**, daß, nachdem das bereits recht komplexe JK-Flip Flop erfolgreich ‘ erschlagen ‘ wurde, die Entwicklung einiger anderer Standardschaltungen zwecks Vertiefung keine unüberwindbaren Probleme mehr bereiten dürfte.
Anmerkung: **Hoffnung** ist das, was den Arbeitstag des Autors prägt !

Problemstellung 2: Es ist ein RS Flip Flop zu entwickeln

Wie bei allen Schaltwerken ist auch hier wieder zu berücksichtigen, daß der aktuelle Ausgangszustand den Folgezustand mitbestimmt (Prinzip der Selbsthaltung bei allen Schaltwerken)
Der Ausgang Q muß folglich wieder in die Verknüpfungsmatrix zurückgeführt und als Variable behandelt werden.

Lösungsstrategie:
- Es sind die Logikbedingungen, durch die das gewünschte RS-FF charakterisiert ist, wieder aus Gründen der Übersichtlichkeit und Systematisierbarkeit in eine Logiktabelle einzutragen
- Das Logikergebnis eines jeden möglichen Produktterms ist gemäß der Logikanforderung zu definieren (Vgl. Kommentarspalte)

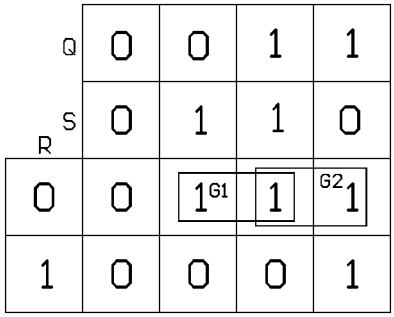
Blockschalbild



Wahrheitstabelle

R	S	Q	Q	Kommentar
0	0	0	0	bleibt 0
0	0	1	1	bleibt 1
0	1	0	1	wird 1
0	1	1	1	bleibt 1
1	0	0	0	bleibt 0
1	0	1	0	wird 0
1	1	0	0	bleibt 0
1	1	1	0	wird 0

KV-Diagramm



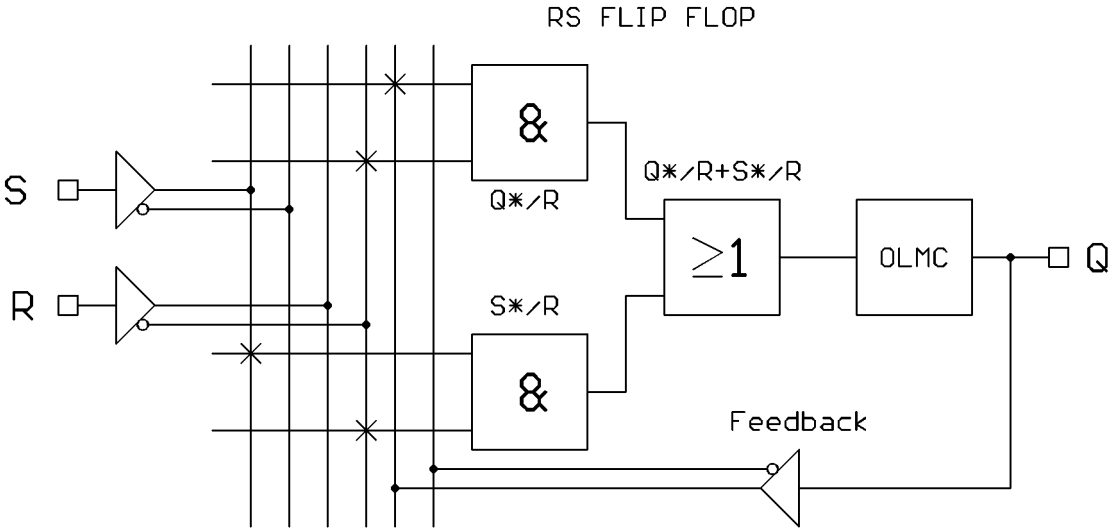
Erinnerung: Eine Gruppe muß sich aus 2 hoch n Variablen zusammensetzen.
Deshalb sind *minimal* 2 Gruppen zu bilden !

Gruppe 1 wird vollständig beschrieben durch: S * /R
Gruppe 2 wird vollständig beschrieben durch: Q * /R

Das RS Flip Flop läßt sich somit durch die Gleichung **Q = Q * /R + S * /R** beschreiben:

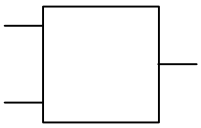
Die Abbildung 2 zeigt das RS Flip Flop in GAL Technik:

ABB. 2:



Problemstellung 3: Es ist ein transparentes D-Latch (1 Bit Register) zu entwickeln

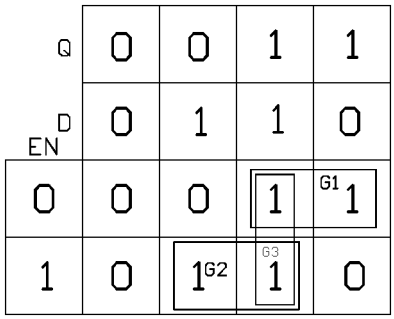
Blockschalbild



Wahrheitstabelle

EN	D	Q	Q	Kommentar
0	0	0	0	bleibt 0
0	0	1	1	bleibt 1
0	1	0	0	bleibt 0
0	1	1	1	bleibt 1
1	0	0	0	bleibt 0
1	0	1	0	wird 0
1	1	0	1	wird 1
1	1	1	1	bleibt 1

KV-Diagramm



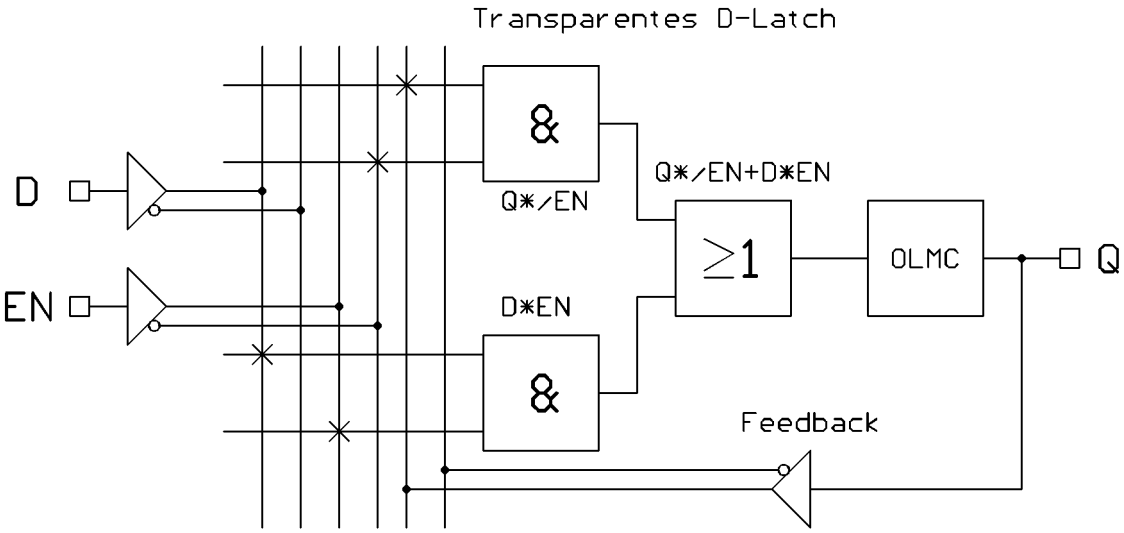
Gruppe 1 wird vollständig beschrieben durch: Q * /EN
Gruppe 2 wird vollständig beschrieben durch: D * EN

Die dritte Gruppenbildung ist zwar grundsätzlich möglich, aber sie liefert keine zusätzlichen Informationen zur Logikbeschreibeung. Sie ist also überflüssig.

Statt der 4 Logikterme der disjunktiven Normalform führt die Minimierung mit Hilfe der KV-Tafel zu einer Logikgleichung mit nur 2 Logiktermen !

Das D - Latch läßt sich folglich durch die Gleichung **Q = Q * /EN + D * EN** beschreiben:

Die Abbildung 3 zeigt das D-Latch in GAL Technik:

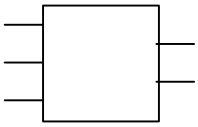


Problemstellung 4: Im letzten Beispiel ist eine Logik zu entwerfen, mit der die Temperatur einer Anlage durch drei unabhängig voneinander arbeitenden Sensoren überwacht wird.
Vorgaben:

- Die Sensoren sind über drei separate Meldeleitungen mit der Logik verbunden.
- Die Meldeleitungen haben so zusammenzuwirken, daß nur dann der Ausgang **TH** [**T**emperatur zu **H**och] umschaltet, *wenn* **alle 3** *oder* wenn **zwei Sensoren** ansprechen
- Spricht nur **einer** *oder* sprechen **zwei** Sensoren an, dann soll *zusätzlich* der Ausgang **TU** [**T**emperatur **U**ngleich] umschalten
- Die Sensoren werden mit S1, S2, S3 bezeichnet

Zur Realisierung dieser Logik werden *zwei* kombinatorische Ausgänge benötigt.
Auf die Option, mit Registerausgängen zu arbeiten, um den jeweiligen Meldezustand *einzufrieren* und explizit *rücksetzen* zu können, wird in diesem Beitrag verzichtet.

Blockschalbild



Wahrheitstabelle

S3	S2	S1	TH	TU
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	1
1	0	0	0	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	0

KV-Diagramm für TH

S3 \ S2 S1	00	01	11	10
0	0	0	G2 1	0
1	0	G1 1	1	G3 1

KV-Diagramm für TU

S3 \ S2 S1	00	01	11	10
0	G1 0	1	1	1
1	1	1	G2 0	1

Auswertungsentscheidung: Für den Ausgang TH wird eine 1- und für TU eine 0-Auswertung durchgeführt.

Die Auswertung der KV-Tafel für TH ergibt:

Gruppe 1 wird beschrieben durch: /S1*S2*S3
Gruppe 2 wird beschrieben durch: S1*S2
Gruppe 1 wird beschrieben durch: S1*S3

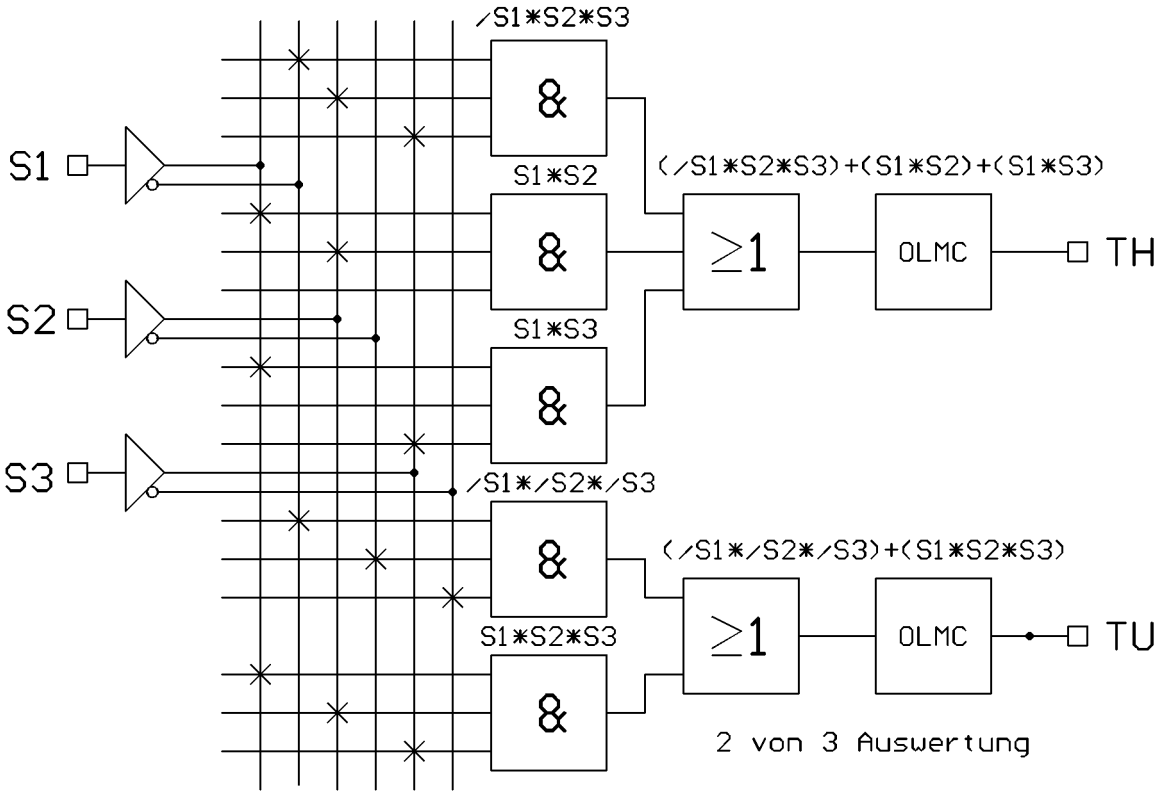
TH = (/S1 * S2 * S3) + (S1 * S2) + (S1 * S3)

Die Auswertung der KV-Tafel für TH ergibt:

Gruppe 1 wird beschrieben durch: /S1*/S2*/S3
Gruppe 2 wird beschrieben durch: S1*S2*S3
d.h.

/TU = (/S1 * /S2 * /S3) + (S1 * S2 * S3)

Die Abbildung 4 zeigt die ‘ 2 von 3 Auswertung in GAL Technik:



Im letzten Abschnitt dieses Beitrages soll der Sprung von der (frustrierenden) Theorie zur konkreten Umsetzung in die Praxis gewagt werden.

Die als bekannt vorausgesetzten Kriterien, durch die eine AND, OR, NAND, NOR und XOR-Logik charakterisiert sind, sollen als *BASIC_GATES* in einem GAL16V8 und die *in diesem Beitrag vorgestellten Flip Flops* und die *2 von 3 Auswertung* als *BFF_2V3* in einem GAL20V8 untergebracht werden.

Leider ... es ist nicht zu ändern ... geht es jedoch wieder nicht ganz ohne Theorie.
Die elementaren Informationen, die im Umgang mit den einzusetzenden Bausteinen GAL16V8 und GAL20V8, der Entwicklungssoftware GDS3.5 und der Testhardware GDU32 vorhanden sein müssen, sind nicht zu umgehen.

Für den tieferen Einstieg in die GAL Materie kommt man nicht an den vom GAL Hersteller LATTICE zur Verfügung gestellten Handbüchern und Datenblättern vorbei , ebensowenig wie an dem Handbuch für die hier verwendete Entwicklungssoftware GDS3.5 der Firma SH-Elektronik aus Kiel.
Die Entwicklungshardware GDU32 sollte mit dem ersten Beitrag ‘ abgehakt ‘ sein.

An dieser Stelle soll versucht werden, die für das weitere Verständnis der Betrachtungen erforderlichen Informationen in kompaktester Form ... ausnahmsweise einmal ohne den sonst so lebenswichtigen Ballast (stoff) ... vorzustellen.

Die GALs 16V8 und 20V8:

Kurzportrait:

- Die GALs 16V8 und 20V8 sind prinzipiell durch die Abbildungen 1 - 4 vorgestellte Struktur charakterisiert
- Das GAL16V8 verfuegt ueber ein Array aus 8 * 8 UND Gattern mit jeweils **32** Eingaengen
- Das GAL20V8 verfuegt ueber ein Array aus 8 * 8 UND Gattern mit jeweils **40** Eingaengen
  Mit der zusätzlich an die UND-Gatter geführte Leitung PTD [**P**roduct **T**erm **D**isable] werden alle nicht benoetigten Eingaenge auf einen definierten Pegel gelegt
  Die Anzahl dieser Eingänge (Anzahl der Spalten in der Verknüpfungsmatrix) bestimmt die maximale Anzahl aller möglichen UND-Verknüpfungen (UND-Terme)
- Beide GALs verfügen über 8 ODER-Gatter die jeweils die 8 UND-Gatter miteinander verknüpfen.
  Es lassen sich somit jeweils maximal 8 komplexe Logikterme zusammenfassen
- Jedem ODER-Ausgang ist eine konfigurierbare Ausgangszelle [OLMC] nachgeschaltet.
  Hierdurch lassen sich maximal 8 Logikergebnisse den Peripheriebedingungen anpassen
- Bei beiden GALs ist die Anzahl der verfügbaren Ein- und Ausgänge vom Betriebsmodus abhängig
  Die Tabellen GAL16V8 und GAL20V8 geben Auskunft über die zur Verfügung stehenden Ein- und Ausgänge
- GND, VCC, CLK und /OE sind *reservierte* Pinbezeichnungen !

Tabelle GAL16V8

Pin Nr.	Simple Mode	Complex Mode	Registered Mode
1	Input	Input	CLK
2-9	Input	Input	Input
11	Input	Input	/OE
12	In- oder Output	NUR Output	In- oder Output
13-14	In- oder Output	In- oder Output	In- oder Output
15-16	NUR Output	In- oder Output	In- oder Output
17-18	In- oder Output	In- oder Output	In- oder Output
19	In- oder Output	NUR Output	In- oder Output
10	GND	GND	GND
20	VCC	VCC	VCC

Tabelle GAL20V8

Pin Nr.	Simple Mode	Complex Mode	Registered Mode
1	Input	Input	CLK
2-11	Input	Input	Input
13	Input	Input	/OE
14	Input	Input	Input
15	In- oder Output	NUR Output	In- oder Output
16-17	In- oder Output	In- oder Output	In- oder Output
18-19	NUR Output	In- oder Output	In- oder Output
20-21	In- oder Output	In- oder Output	In- oder Output
22	In- oder Output	NUR Output	In- oder Output
23	Input	Input	Input
12	GND	GND	GND
24	VCC	VCC	VCC

Zusätzliche Informationen zum Block Diagramm, Pin Konfiguration und prinzipieller Innenstruktur:
Siehe Anhang: Auszug aus dem Lattice Data Book / 1994

Die Entwicklungssoftware GDS3.5

Auf eine Vorstellung und Bewertung des unterschiedlichen Softwareangebotes in Verbindung mit PLD-Entwicklungen wird hier verzichtet. Zu dieser Thematik stehen ausreichende Beiträge zur Verfügung.
Der Autor hat sich letztlich für die Entwicklungssoftware GDS3.5 der Firma SH-Elektronik entschieden.

Die bei der Einarbeitung in die Welt der GALs gewonnenen Erfahrungen - sowohl bei der eigenen Einarbeitung als auch im Rahmen der Ausbildung physikalisch technischer Assistenten, Informatikassistenten und FOS-Absolventen - haben die Entscheidung für die hier benutzte Software nicht schwer gemacht.

Trotz einiger Schwächen, die vor allem auf den ' DOS-Komfort ' zurückzuführen sind, hat sich das Entwicklungssystem aufgrund seiner Bedienerfreundlichkeit und Anschaulichkeit für die Ausbildung bestens bewährt.

Arbeiten mit der Entwicklungssoftware GDS35

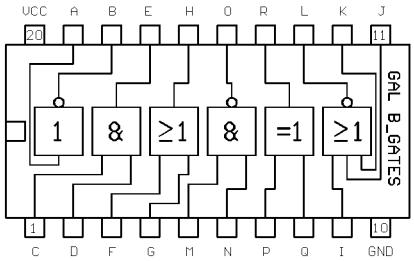
Zur Entwicklung eines konkreten GAL-Files und die Generierung des für die Brennersoftware erforderlichen JEDEC-Files bedarf es nur weniger Informationen.

Sofern man sich bemüht,
- der Sicherheit halber alles groß zu schreiben
- alle Kommentare nach dem Schlüsselwort [CHIP] konsequent durch ein Semikolon einzuleiten
- die Programm-Kopfzeile in der Form ' CHIP ... NAME ... GALTYP ... BETRIEBSMODE ' zu formulieren
- die Pinanordnung entsprechend der Syntaxvorgaben vorzunehmen und
- die Logikgleichungen wie in den Programmbeispielen aufzubauen,

dann hat man das GAL auf seiner Seite.

Die beiden folgenden Beispiele [BASIC_GATES] und [BFF_2V3] sollen den Programmaufbau und die Syntax verdeutlichen.

Programmbeispiel 1: Es sollen die elementaren Logikgatter AND, NAND, OR, NOR und XOR in einem GAL16V8 zur Verfügung gestellt werden
Die Pinbelegung ist durch die anzusteuernde Peripherie vorgegeben

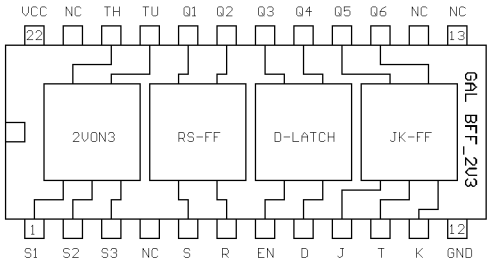


Das zu editierende Sourcefile ist jeweils zwischen den Trennlinien dargestellt.

```
;-------------------------------------------------------------------------------------------------------------------------
Titel: BASIC GATES
Entwickler: gev                 Version: 2.1      Letzte Änderung: 15.02.98

Geforderte GAL-Funktion:    INVERTER, AND, OR, NAND, NOR, XOR
Vorgeschriebener GAL-Typ:   GAL16V8
Pinzuweisung:               Entsprechend der Peripherievorgaben
Geforderter Betriebsmodus:  COMPLEX_MODE rein kombinatorisch


CHIP        B_GATES       GAL16V8       COMPLEX_MODE        ; Programm-Kopfzeile

; 1   2   3   4   5   6   7   8   9   10          Pinanordnung
  C   D   F   G   M   N   P   Q   I   GND

; 11  12  13  14  15  16  17  18  19  20          Pinanordnung
  J   K   L   R   O   H   E   B   A   VCC
```

```
; Logikgleichungen          ; Logikfunktionen

B = !A                      ; Inverter
E = C  * D                  ; AND    Gate mit zwei Eingängen
H = F + G                   ; OR     Gate mit zwei Eingängen
L = !I + !J + !K            ; NAND   Gate mit drei Eingängen
O = !M * !N                 ; NOR    Gate mit zwei Eingängen
R = P * /Q + /P * Q         ; XOR    Gate mit zwei Eingängen

USER_ID = gev1502           ; Anwenderspezifische GAL-Kennung
;-------------------------------------------------------------------------------------------------------------------------------
```

Programmbeispiel 2: Es sollen die in diesem Beitrag entwickelten Logikschaltungen in einem GAL20V8 zur Verfügung gestellt werden
Die Pinbelegung ist durch die anzusteuernde Peripherie vorgegeben




```
;-------------------------------------------------------------------------------------------------------------------------------
```

Titel: BASIC FLIP-FLOPS und 2 von 3 AUSWERTUNG

Entwickler: gev Version: 1.2 Letzte Änderung: 16.02.98

Geforderte GAL-Funktion: RS-FF, D-LATCH, JK-FF, 2von3-Auswertung
Vorgeschriebener GAL-Typ: GAL20V8
Pinzuweisung: Entsprechend der Peripherievorgaben
Geforderter Betriebsmodus: COMPLEX_MODE rein kombinatorisch

```
CHIP        BFF_2V3      GAL20V8      COMPLEX_MODE                ; Programm-Kopfzeile

; 1   2   3   4   5   6   7   8   9   10  11  12   Pinanordnung
  S1  S2  S3  NC  S   R   EN  D   J   T   K   GND

; 11  12  13  14  15  16  17  18  19  20  21  22   Pinanordnung
  NC  NC  Q6  Q5  Q4  Q3  Q2  Q1  TU  TH  NC  VCC

; Logikgleichungen                    ; Logikfunktionen

TH = /S1 * S2 * S3 + S1 * S2 + S1 * S3    ; Temperatur zu hoch
/TU =  /S1 * /S2 * /S3 + S1 * S2 * S3     ; Temperatur ungleich
Q1 = Q1 * /R + S * /R                     ; Ausgang RS-Flip-Flop
Q2 = /Q1                                  ; Invertierter Ausgang RS-FF
Q3 = Q3 * /EN + D * EN                    ; Ausgang D-Latch
Q4 = /Q3                                  ; Invertierter Ausgang D-Latch
```

```
Q5 = T * J * /K + Q5 * /K + Q5 * /T          ; Ausgang JK-Flip-Flop
Q6 = /Q5                                    ; Invertierter Ausgang JK-FF

USER_ID = gev1602                           ; Anwenderspezifische GAL-Kennung
;-------------------------------------------------------------------------------------------------------------------
```

Syntax-Check:

Unter der Option [COMPILIEREN] sollte vor dem Assemblieren des Quellfiles ein Syntaxcheck durchgeführt werden.

Wie die Abbildungen zeigen, erscheint das GAL mit den vom Anwender definierten Pinzuordnungen und Pinfunktionen in recht anschaulicher Weise.
Sysntaktische Fehler werden durch die Änderung der Bildschirm Hintergrundfarbe von grau auf rot unübersehbar angezeigt.

Die Interpretation der unterschiedlichen Fehlermeldungen sowie der detaillierte Umgang mit der GDS3.5 Software sind dem Handbuch des Herstellers zu entnehmen.

Programmieren der GALs mit dem Programmiergerät GALEP III:

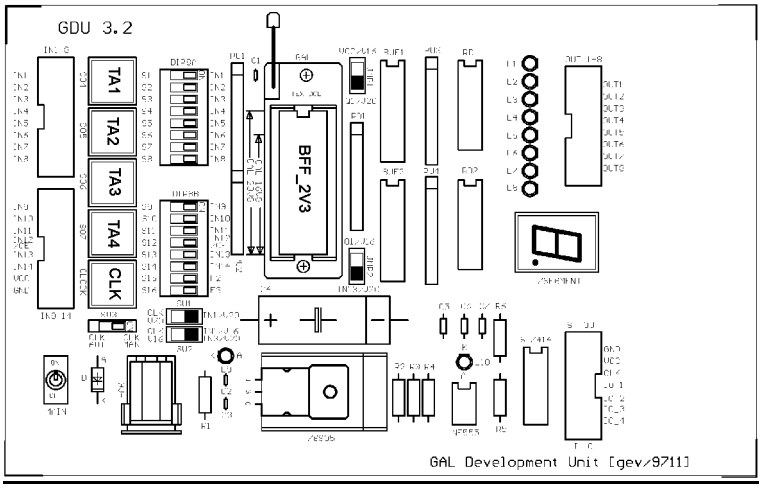
Für die Programmierung der GALs hat sich das von der Firma CONITEC vertriebene Gerät GALEP III hervorragend bewährt.
Die Software läuft unter Windows 3.11 bis hin zu Windows NT und ist so einfach zu handhaben, daß es keinerlei weiterer Informationen bedarf.

Testen der GALs mit der GDU32

Für das Testen der GALs steht die im Beitrag 1 vorgestellte GDU32 zur Verfügung.
Entsprechend der dort beschrieben Anleitung ist die Hardware für den zu testenden GAL-Typ zu konfigurieren.

Da beide hier entwickelten GALs im COMPLEX_MODE ohne Registerfunktion arbeiten, sind die Umschalter so einzustellen, daß auf alle Eingänge zugegriffen werden kann.
Der Taktgenerator ist sinnvollerweise auf manuellem Betrieb zu schalten, da er in den vorliegenden Fällen keine Funktion hat.
Für die Eingänge IN2 bis IN5 beim 16V8 und IN4 bis IN7 stehen neben den DIP-Schaltern die Taster TA1 bis TA4 für das Austesten der Logikbausteine zur Verfügung.

Die Darstellung zeigt die Konfiguration der GDU32 für den Test des GALs [BFF_2VON3]



Quellen / Anregungen:

Sämtliche im Rahmen des Projektes ' *Programmierbare Logik mit GALs in der naturwissenschaftlich - technischen Ausbildung am SZU Bremen* ' entwickelten Programme (GAL-Files, JEDEC-Files) stehen in der **Web-Site der Physikalisch Technischen Assistenten am SZU im Downloadverzeichnis als Zip Datei zur Verfügung.**

Das noch laufende Unterrichtsprojekt wird mit den Technischen Assistenten für Informatik, den Physikalisch Technischen Assistenten und der FOS Naturwissenschaft am SZU Bremen durchgeführt.
Die in diesem Beitrag gesammelten Erfahrungen beruhen auf der konkreten Arbeit der Absolventen mit der vorgestellten Soft- und Hardware.
In der Web-Site sind sämtliche Verweise auf die verwendete Hard- und Software zu finden.
[GALs der Firma Lattice, Evaluationsboard GDU32, GAL-Programmierer GALEP III, Entwicklungssoftware GDS3.5 von SH-Elektronik u.v.a.m.]

Die Internetadresse lautet: **http:/www.phyta.aka.de**

Bezugsquellen für die GDU32:

Die GDU32 wird in Kuerze von der Firma **PCS / Prozeßrechnertechnik &. Web Design**
und spaeter vom **Elektronikladen Detmold**
sowie der Firma **SH-Elektronik** vertrieben werden.

Letzte Anmerkung:
Ohne die zahlreichen Anregungen und die geduldige Mithilfe des Kollegen **S.O.Klapprott** vor allem in der schwierigen Anlaufphase wäre das Projekt nicht zustande gekommen. Danke.

Projekt: GAL-Evaluationsboard [GDU32]

Teil 3: Die GDU 32 voll im logischen Einsatz ... ganz leGAL ...

Von: **Gerd Evers**

Die Grundlagen sind gelegt.
Die GDU32 ist bereit.
Die theoretischen Grundlagen (hoffentlich) verstanden
Jetzt kann es richtig losgehen.
Ganz eGAL, wie die Problemstellung aussieht, jetzt zeigt es sich, ob und wie sich auch komplexe Logikprobleme GALant lösen und mit der GDU32 veranschaulichen lassen.

Die hier ausgewählten Anwendungsbeispiele resultieren wieder aus dem Tätigkeitsfeld des Autors und haben direkt mit den dort gestellten Anforderungen an die zukünftigen Logikprofis zu tun.

Es geht **einerseits** um *die Bewältigung konkret vorgegeben Problemstellungen* z.B.

- Es ist die Hardware einer Schnittstellenlogik für ein vorhandenes Prozeßkontrollsystem mit Hilfe neuerer Technik zu optimieren.
  Was ist konkret zu tun, um einen aus TTL-Gattern aufgebauten Adreßdekoder, der sowohl ein Chip-Select-Signal (aktiv Low oder aktiv High) als auch ein Card-Enable-Signal (Open Collector- bzw. Tristate-Anforderung) zu generieren hat, durch ein der Problemstellung angemessenes Logic-Device GAL zu ersetzen
  (Siehe ***Beispiel 1***).

oder

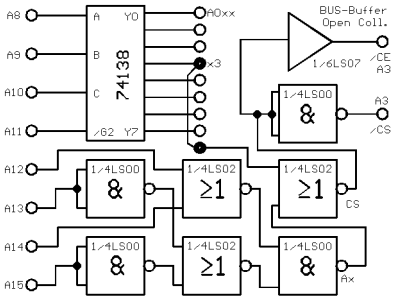
- Für ein bereits im Einsatz befindliches Prozeßkontrollsystem ist eine Speichererweiterung zu konzipieren, welche einen konkret vorgegeben Adreßraum abzudecken hat. (Siehe ***Beispiel 2***).
  Die Adreßlogik ist wiederum mit einer, der Problemstellung angemessenen Hardware zu lösen.

In beiden Fällen wäre ein GAL auch nach heutigen Gesichtspunkten sicherlich eine angemessene Lösung.

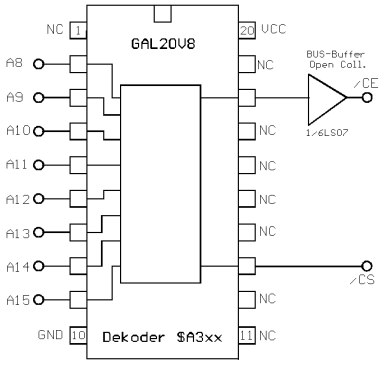
Es geht **andererseits** um *die Veranschaulichung der Funktion auch komplexerer Logikeinheiten*, wie beispielsweise die serielle Datenübertragung u. ä.

Beispiel 1:
Die dargestellte, real vorhandene Adreßlogik ist so ausgelegt, daß die Adreßbereiche $A0xx bis $A7xx durch Umstecken einer Brücke ausgewählt werden können.
Für die Default-Adresse der jeweilige Schnittstelle ist die Brücke (Jumper) fest eingelötet.
Mit einem GAL16V8 ist ein Adreßdekoder für den Adreßraum $A3xx zu entwerfen, der die dargestellte TTL-Logik ersetzt.

Vorhandene Logik:



Neue geforderte Logik:



Vorgaben:

- Verwendung der GDS35 - Syntax für die Erstellung des Source-Files.
- Einzuhaltende Pinbezeichnungen: Chip Select [CS]
  Card Enable [CE]
  Adreßleitungen [A8 - A15]

Lösungshinweis:
Da Chip-Select [CS] und Card-Enable [CE] den Pegel beim Ansprechen der Hexadezimal-Adresse $A3xx bzw. der Binär-Adresse %1010 0011 xxxx xxxx von high auf low ändern müssen, lassen sich die Logikgleichungen unmittelbar aus der Schaltbedingung herleiten.

Für die Adresse: %			*1*	*0*	*1*	*0*	*0*	*0*	*1*	*1*
ergibt sich:	**CS**	**=**	**A15**	**/A14**	**A13**	**/A12**	**/A11**	**/A10**	**A09**	**A08**
bzw. für CE:	**CE**	**=**	**CS**							

Card-Enable [/CE] wird an einem zweiten Ausgang herausgeführt und dem Bus über einen Treiber mit Open Collector (z.B.74LS07 / nicht invertierend) zur Verfügung gestellt.
Sollte ein invertierender Treiber zur Verfügung stehen, so ist das Ausgangssignal entsprechend anzupassen.

Das dargestellte Sourcefile für das GAL16V8 erfüllt die vorgegebene Bedingung:

```
;-------------------------------------------------------------------------------------------------------------------------------
Titel:                    Adreßdekoder für Serielle Schnittstelle / Hardwareadresse $A3xx
Zielsystem:               Prozeßkontrollsystem [ VPCS ]
Entwickler:               gev     Version: 2.1     Letzte Änderung: 28.02.98

Geforderte GAL-Funktion:  Adreßdekoder für $A3xx
Vorgeschriebener GAL-Typ: GAL16V8
Pinzuweisung:             Entsprechend der Peripherievorgaben
Geforderter Betriebsmodus: COMPLEX_MODE rein kombinatorisch
Ausgang CS:               Aktiv High oder aktiv Low
Ausgang CE:               Tristate / über Open-Collector-Buffer an Bus geführt / gesteuert durch CS

CHIP        DEC_A3XX    GAL16V8      COMPLEX_MODE    ; Programm-Kopfzeile

; 1    2    3    4    5    6    7    8    9    10      Pinanordnung
  NC   A15  A14  A13  A12  A11  A10  A9   A8   GND
; 11   12   13   14   15   16   17   18   19   20      Pinanordnung
  NC   NC   NC   NC   /CS  /CE  NC   NC   NC   VCC     ; CS und CE: Low-Aktiv
```

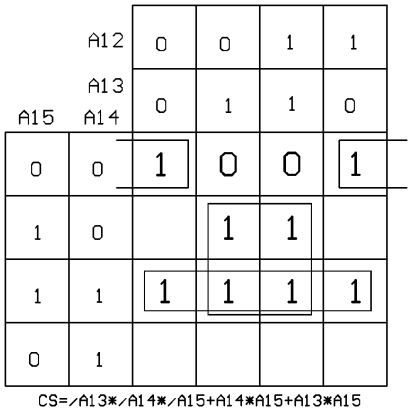
```
; Logikgleichung                                                    Logikfunktionen
CS  = A8 * A9 * /A10 * /A11 * /A12 * A13 * /A14 * A15             ; CS [ bei $A3xx activ Low ]
CE  = CS                                                          ; CE [ bei NICHT $A3xx / Tristate ]

USER_ID = gev2802                                                 ; User Identification
;-----------------------------------------------------------------------------------------------------------------------
```

Zum Austesten des Dekoders ist die GDU32 entsprechend der Tabelle 2 im ersten Beitrag zu konfigurieren, d.h.
- JMP1 in Position VCC/V16
- JMP2 in Position Q1/V16
- Mit den DIP-Schaltern S4 bis S11 läßt sich die obige Adresse (Bitkombination) einzustellen
- Der Taktgenerator wird nicht benötigt (sinnvollerweise auf CLK_MAN stellen)
- Das Logikergebnis wird durch die LEDs 4 und 5 angezeigt (/CS = LED5 und /CE = LED4)
  Beim Ansprechen der Adresse $A3xx müssen die LEDs dunkel sein

Beispiel 2:
Mit einem GAL16V8 soll eine 32k Speichererweiterung für den Adreßraum $2000 - $9FFF (32k) entworfen werden.
Vorgaben: Einzuhaltende Pinbezeichnungen: CS = Chip Select, CE = Card Enable
Ausdekodierung des Adreßraumes $2000-$9FFF über die Adreßleitungen A12 bis A15

Wahrheitstabelle: KV-Diagramm

ADR	A15	A14	A13	A12	CS	Funktion
0	0	0	0	0	1	Disable
1	0	0	0	1	1	Disable
2	0	0	1	0	0	Enable
3	0	0	1	1	0	Enable
4	0	1	0	0	0	Enable
5	0	1	0	1	0	Enable
6	0	1	1	0	0	Enable
7	0	1	1	1	0	Enable
8	1	0	0	0	0	Enable
9	1	0	0	1	0	Enable
A	1	0	1	0	1	Disable
B	1	0	1	1	1	Disable
C	1	1	0	0	1	Disable
D	1	1	0	1	1	Disable
E	1	1	1	0	1	Disable
F	1	1	1	1	1	Disable



Das Logikergebnis für die 1-Auswertung zeigt, daß für den Adreßdekoder lediglich die 3 oberen Bits des Adresse benötigt werden.

Logikbedingung für Chip Select: **CS = (/A13 * /A14 * /A15) + (A14 * A15) + (A13 * A15)**

Mit dem Card-Enable-Signal ist wie im Beispiel 1 zu verfahren, d.h. Card-Enable ist dem Steuerbus über einen Buffer mit Open-Collector zur Verfügung zu stellen.

Das dargestellte GAL Sourcefile erfüllt die geforderten Bedingungen.

```
;-------------------------------------------------------------------------------------------------------------------------
Titel:                          Adreßdekoder für 32k Speichererweiterung
Zielsystem:                     Prozeßkontrollsystem [ VPCS ]
Entwickler:                     gev     Version: 1.2 [     28.02.98 ]

Geforderte GAL-Funktion:    Adreßdekoder für Adreßraum $2000 - $9FFF
Vorgeschriebener GAL-Typ: GAL16V8
Pinzuweisung:               Entsprechend der Peripherievorgaben
Geforderter Betriebsmodus:  COMPLEX_MODE rein kombinatorisch
Ausgang CS:                 Aktiv High oder aktiv Low
Ausgang CE:                 Tristate / über Open-Collector-Buffer an Bus geführt / gesteuert durch CS

CHIP        DEC_32K     GAL16V8     COMPLEX_MODE        ; Programm-Kopfzeile
; 1   2     3     4     5     6     7     8     9     10      Pinanordnung
  NC  NC    NC    A15   A14   A13   NC    NC    NC    GND
; 11  12    13    14    15    16    17    18    19    20      Pinanordnung
  NC  NC    NC    NC    /CS   /CE   NC    NC    NC    VCC


; Logikgleichungen                                          Logikfunktionen
CS = ( /A13 * /A14 * /A15 ) +  ( A13 * A15 ) + ( A14 * A15 )  ; CS bei $2000 - $9FFF activ Low
CE = CS                                                     ; CE  nei NICHT $2000 - $9FFF / Tristate ]

USER_ID = gev2802                                           ; User Identification
;-------------------------------------------------------------------------------------------------------------------------
```

Zum Austesten des Dekoders ist die GDU32 wie im Beispiel 1 zu konfigurieren.
- Mit den DIP-Schaltern S4 - S6 bzw. den Tastern TA1 - TA3 läßt sich der definierte Adreßraum überprüfen
- Das Logikergebnis wird wieder durch die LEDs 4 und 5 angezeigt (CS = LED5 und /CE = LED4)
  Beim Ansprechen der Adresse $A3xx müssen die LEDs dunkel sein

Beispiel 3:
Zur Veranschaulichung der Funktion komplexerer Logikeinheiten werden im folgenden zunächst einmal ein 4 Bit Aufwärts- und Abwärtszähler mit Reset-Option und ein 4 Bit Aufwärtszähler mit Reset- und Pre-Load-Option vorgestellt.

Mit Hilfe der Wahrheitstabelle wird zunächst der 4 Bit Up-Down-Counter entwickelt.
Die Reset-Option läßt sich durch eine entsprechende Ergänzung durch die Variable [RES in jeden Logikterm einfügen.

Da bei diesem Zähler bereits alle ODER-Gatter des GALs benutzt werden, wird im zweiten Beispiel auf die Abwärts Zählfunktion verzichtet und stattdessen ein Pre-Load-Modus implementiert.
Hierdurch kann die Möglichkeit demonstriert werden, wie sich der Zähler mit einem voreingestellten Wert laden läßt.

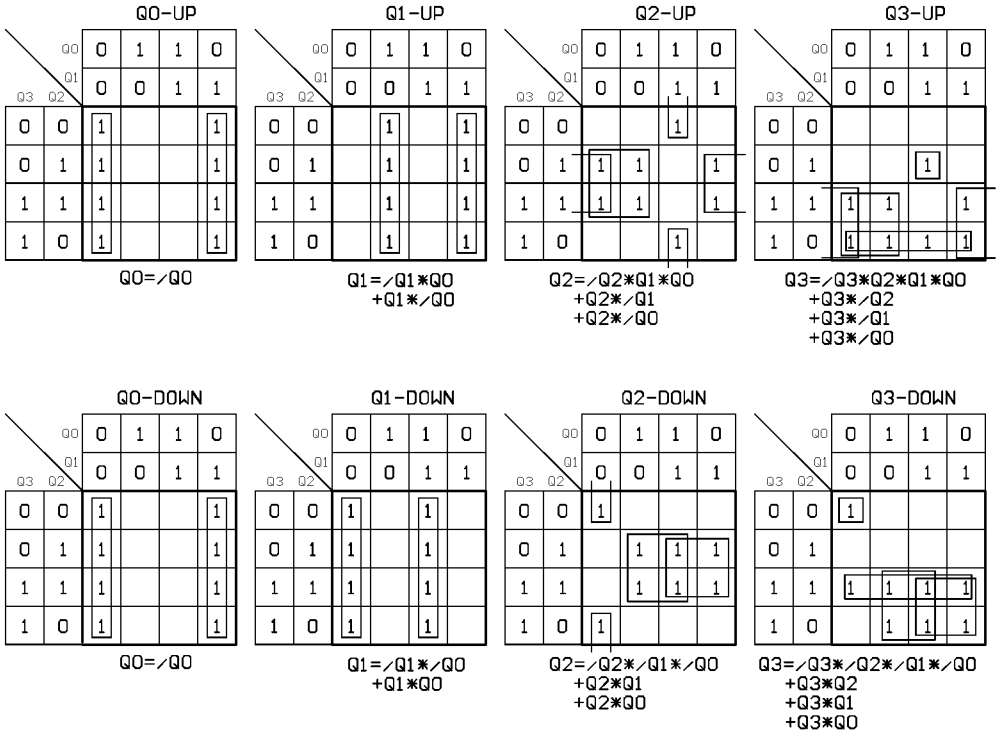
Hinweis / Erinnerung: Zum Aufbau sequentielle Schaltwerke wie z.B. Zähler oder Schieberegister sind Registerausgängen zu programmieren.
Die Syntax für die Verwendung von Registerausgängen lautet:

Ausgangsbedingung **:=** Logikterm1 + Logikterm2 + ... [Doppelpunkt - Gleich !]

Clk		Ausgangszustand Qn				Folgezustand Q n+1 [UP]				Folgezustand Q n+1 [DOWN]			

Nr.:	RES	Q3	Q2	Q1	Q0	Q3	Q2	Q1	Q0	Q3	Q2	Q1	Q0
0	1	0	0	0	0	0	0	0	1	1	1	1	1
1	1	0	0	0	1	0	0	1	0	0	0	0	0
2	1	0	0	1	0	0	0	1	1	0	0	0	1
3	1	0	0	1	1	0	1	0	0	0	0	1	0
4	1	0	1	0	0	0	1	0	1	0	0	1	1
5	1	0	1	0	1	0	1	1	0	0	1	0	0
6	1	0	1	1	0	0	1	1	1	0	1	0	1
7	1	0	1	1	1	1	0	0	0	0	1	1	0
8	1	1	0	0	0	1	0	0	1	0	1	1	1
9	1	1	0	0	1	1	0	1	0	1	0	0	0
10	1	1	0	1	0	1	0	1	1	1	0	0	1
11	1	1	0	1	1	1	1	0	0	1	0	1	0
12	1	1	1	0	0	1	1	0	1	1	0	1	1
13	1	1	1	0	1	1	1	1	0	1	1	0	0
14	1	1	1	1	0	1	1	1	1	1	1	0	1
15	1	1	1	1	1	0	0	0	0	1	1	1	0

Die Auswertung der Tabelle mit Hilfe der KV-Diagramme führt zu den angegebenen Ergebnissen:



Faßt man die Logikbedingungen für Q0 bis Q3 zusammen und erweitert die Gleichungen durch die Variablen Reset [RES], Count-Up-Down [UD] und / oder die Pre-Load-Option, so ergeben sich für die 4 Bit Zähler folgende Logikgleichungen:

```
;---------------------------------------------------------------
Titel: 4 Bit UP-DOWN-COUNTER mit RESET

CHIP  UPDWN4  GAL20V8A  COMPLEX_MODE

CLK NC NC RES  UD  NC  NC  NC  NC NC  NC  GND
GND
/OE  NC  Q0   Q1   Q2  Q3   NC  NC  NC NC  NC  VCC
```

```
;---------------------------------------------------------------
Titel : 4 Bit UP-COUNTER mit RESET und PRE-LOAD

CHIP  UP_PRE_LD4  GAL20V8A COMPLEX_MODE

CLK NC NC RES  LD  NC  NC  IN0  IN1 IN2  IN3
GND
/OE  NC  Q0   Q1   Q2  Q3   NC  NC  NC NC  NC  VCC
```

```
;Logikgleichungen
Q0 :=  /Q0 * RES

Q1 :=  /Q1 *  Q0  *  UD * RES
    +  Q1 * /Q0  *  UD * RES
    + /Q1 * /Q0  * /UD * RES
    +  Q1 *  Q0  * /UD * RES

Q2 :=  /Q2 *  Q1  *  Q0 *  UD * RES
    +  Q2 * /Q1  *  UD * RES
    +  Q2 * /Q0  *  UD * RES
    + /Q2 * /Q1  * /Q0 * /UD * RES
    +  Q2 *  Q1  * /UD * RES
    +  Q2 *  Q0  * /UD * RES

Q3 :=  /Q3 *  Q2  *  Q1 * Q0  * UD * RES
    +  Q3 * /Q2  *  UD * RES
    +  Q3 * /Q1  *  UD * RES
    +  Q3 * /Q0  *  UD * RES
    + /Q3 * /Q2  * /Q1 * /Q0  * /UD * RES
    +  Q3 *  Q2  * /UD * RES
    +  Q3 *  Q1  * /UD * RES
    +  Q3 *  Q0  * /UD * RES

USER_ID = gev030398
;---------------------------------------------------------------
```

```
;Logikgleichungen
Q0 :=  /Q0  * LD * RES
           +  IN0 * /LD

Q1 :=  /Q1 *   Q0  * LD * RES
           +    Q1 *  /Q0  * LD * RES
           +    IN1 * /LD

Q2 :=  /Q2 *  Q1  *  Q0 *  LD * RES
           +  Q2 * /Q1  *  LD * RES
           +  Q2 * /Q0  *  LD * RES
      +  IN2 * /LD

Q3 :=  /Q3 *  Q2  *  Q1 * Q0  * LD * RES
           +  Q3 * /Q2  *  LD * RES
           +  Q3 * /Q1  *  LD * RES
           +  Q3 * /Q0  *  LD * RES
           +  IN3 * /LD

USER_ID = gev030398
;---------------------------------------------------------------
```

Beispiel 4: Freilaufender 8-Bit Binär Aufwärstszähler mit Reset

Die Logikgleichungen lassen eine Systematik erkennen, mit der es möglich ist, einen freilaufenden 8 Bit Aufwärts-Zähler herleiten, ohne weitere 'Logikklimmzüge' machen zu müssen.

Die angegebenen Logikbedingungen beschreiben einen solchen 8 Bit Aufwärtszähler mit Reset-Option. Da für die Beschreibung des höchstwertigsten Ausganges (Q7) acht Logikterme miteinander verknüpft werden müssen, ist das GAL damit vollständig ausgeschöpft.

Pinbelegung:

/RES -> Reset (Counter auf Null setzen)
Q0 ... Q7 -> Counter Outputs

```
CHIP    CNT8BIT    GAL20V8A    COMPLEX_MODE

;S1   S2   S3   S4   S5   S6   S7   S8   S9   S10  S11  12
CLK  NC   NC   /RES NC   NC   NC   NC   NC   NC   NC   GND

;S12  S13  15   16   17   18   19   20   21   22   S14  24
/OE  NC   Q0   Q1   Q2   Q3   Q4   Q5   Q6   Q7   NC   VCC


Q0 :=          /Q0 * /RES

Q1 :=          /Q1 * Q0 * /RES
             +  Q1 * /Q0 * /RES

Q2 :=          /Q2 * Q1 * Q0 * /RES
             + Q2 * /Q1 * /RES
             +  Q2 * /Q0 * /RES

Q3 :=          /Q3 * Q2 * Q1 * Q0 * /RES
             +  Q3 * /Q2 * /RES
             +  Q3 * /Q1 * /RES
             +  Q3 * /Q0 * /RES

Q4 :=          /Q4 * Q3 * Q2 * Q1 * Q0 * /RES
             +  Q4 * /Q3 * /RES
             +  Q4 * /Q2 * /RES
             +  Q4 * /Q1 * /RES
             +  Q4 * /Q0 * /RES

Q5 :=          /Q5 * Q4 * Q3 * Q2 * Q1 * Q0 * /RES
             +  Q5 * /Q4 * /RES
             +  Q5 * /Q3 * /RES
```

Funktionsprüfung

Konfiguration der GDU32:

Die Standard Voreinstellungen für das GAL 20V8 sind
kursiv dargestellt !

Jumper JMP1 auf Stellung Q1/V20
Jumper JMP2 auf Stellung IN13/V20
Umschalter SU1 auf Stellung CLK/V20
Umschalter SU2 auf Stellung IN3/V20
Umschalter SU3 auf Stellung CLK_MAN oder CLK_AUT

DIP-Schalter S12 auf Stellung ON [LOW]: /OE

DIP-Schalter S15: Stellung OFF [CLK: Slow]
Stellung ON [CLK: Medium]

DIP-Schalter S16: Stellung OFF [CLK: Slow]
Stellung: ON [CLK: Fast]

Funktionstest:

- CLK automatisch (Umschalter S15 nach aussen !);
  CLK_STOP durch Drücken des Tasters TA5 (CLK)

- DIP-Schalter S15 (ON) und S16 auf OFF
  (CLK - Frequenz = Medium)

- Taster TA1 [Reset / S04] drücken (LOW):
  Reset wird beim folgenden Takt ausgelöst

```
        +  Q5 * /Q2 * /RES
        +  Q5 * /Q1 * /RES
        +  Q5 * /Q0 * /RES

Q6 :=       /Q6 * Q5 * Q4 * Q3 * Q2 * Q1 * Q0 * /RES
        +  Q6 * /Q5 * /RES
        +  Q6 * /Q4 * /RES
        +  Q6 * /Q3 * /RES
        +  Q6 * /Q2 * /RES
        +  Q6 * /Q1 * /RES
        +  Q6 * /Q0 * /RES

Q7 :=       /Q7 * Q6 * Q5 * Q5 * Q4 * Q3 * Q2 * Q1 * Q0 * /RES
        +  Q7 * /Q6 * /RES
        +  Q7 * /Q5 * /RES
        +  Q7 * /Q4 * /RES
        +  Q7 * /Q3 * /RES
        +  Q7 * /Q2 * /RES
        +  Q7 * /Q1 * /RES
        +  Q7 * /Q0 * /RES

USER_ID = gev3112
```

- Taster TA1 (S04) loslassen (HIGH):
  Counter startet mit dem folgenden Takt

- CLK-Frequenz variieren

- CLK manuell testen

- TRISTATE-Funktion testen: DIP-Schalter S12 auf OFF
  Sämtliche LEDs erlöschen; keine Funktion

Die Gleichungen zeigen, daß die GALs 16V8 bzw. 20V8 bereits vollständig ausgenutzt sind, da lediglich 8 ODER-Verknüpfungen pro Logikterm zur Verfügung stehen.
Da sowohl die Entwicklungssoftware GDS35 als auch das Evaluationsboard GDU32 die komplette GAL-Familie, die pinkompatibel zu genannten GALs ist bearbeiten kann, sollte der Anwender durch Auswahl eines anderen GAL-Typs, wie beispielsweise das GAL20RA10 oder das Kraftpaket 6001 bzw. 6002 in Erwägung ziehen, um den hier vorgestellten Counter entsprechend dem 4 Bit Beispiel auf einen 8 Bit UP-DWN Counter mit Reset und Pre-Load-Option zu erweitern.

Beispiel 5: Counter mit 7 Segment-Dekoder

Pinbezeichnungen: RES -> Reset [Counter auf Null setzen]
CO -> DezimalPunkt [**C**LK-**O**utput für Folgesegment]

Wenn man sich an die GAL-Denkweise gewöhnt hat, ist es kein Kunststück mehr, das GAL so zu programmieren, daß es von Null bis Neun zählen kann und zwar so, daß die 7 Segmentanzeige dieses auch entsprechend anzeigt.

Was muß man wissen ? Der aktueller Zustand des Counters entspricht der Logik-Eingangsbedingung
Worauf ist zu achten ? Nach einem Clock muß die in der Wahrheitstabelle angegebene Ausgangsbedingung erfüllt sein.
Dieses ist wiederum die Eingangsbedingung für den nächsten Folgezustand
Was ist zu tun ? Immer wieder dasselbe.
Die Logikbedingungen sind systematisch in eine Wahrheitstabelle einzutragen und daraus die Logikgleichungen nach bewährter Art zu formulieren.

Der aktueller Zustand entspricht der Eingangsbedingung										Der Zustand nach einem CLK entspricht der Ausgangsbedingung								
Ziffer	RES	g	f	e	d	c	b	a	CO	Ziffer	g	f	e	d	c	b	a	CO
0	1	1	0	0	0	0	0	0	1	1	1	1	1	1	0	0	1	1
1	1	1	1	1	1	0	0	1	1	2	0	1	0	0	1	0	0	1
2	1	0	1	0	0	1	0	0	1	3	0	1	1	0	0	0	0	1
3	1	0	1	1	0	0	0	0	1	4	0	0	1	1	0	0	1	1
4	1	0	0	1	1	0	0	1	1	5	0	0	1	0	0	1	0	1
5	1	0	0	1	0	0	1	0	1	6	0	0	0	0	0	1	0	1
6	1	0	0	0	0	0	1	0	1	7	1	1	1	1	0	0	0	1
7	1	1	1	1	1	0	0	0	1	8	0	0	0	0	0	0	0	1
8	1	0	0	0	0	0	0	0	1	9	0	0	1	0	0	0	0	0
9	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1
RES	0	x	x	x	x	x	x	x	x	RES	1	0	0	0	0	0	0	1

Die Logikgleichungen für die einzelnen Elemente der 7-Segmentanzeige werden anhand der Ausgangsbedingung aus der Wahrheitstabelle abgelesen.

Die Ausgangsbedingung für das Element a bei einer 1-Auswertung lautet beispielsweise:

Das Segment **a** muß *nach einem Clock* auf **1** gehen, wenn *vorher* die Segmente *a, b, c, d, e, f* auf **0** und g, *CO* sowie *RES* auf **1** gelegen haben **ODER**, wenn *vorher* die Segmente *a, b, c, d und g* auf **0** und e, f, *CO* sowie *RES* auf **1** gelegen haben.

Somit gilt für das Element a: a := /a * /b * /c * /d * /e * /f * g * CO * RES
\+ /a * /b * /c * /d * e * f * /g * CO * RES

Entsprechend lassen sich die Logikgleichungen für die anderen Segmente formulieren.

Es ist darauf zu achten, daß das GAL für den hier beschriebenen Anwendungsfall im Register-Modus zu arbeiten hat, d.h. Pin 1 muß als CLK und Pin 13 als /OE definiert werden !

Das folgende Source-File zeigt den Counter mit 7 Segment-Dekoder für das GAL20V8:

Counter mit 7 Segment-Dekoder

```
Vorgaben:  RES -> Reset
           Counter nach CLK auf Null setzen
           CO  -> ClockOut [ CLK für Folgesegment
           Dezimalpunkt der 7 Segment-Anzeige ]

CHIP    CNT7SEG     GAL20V8A      COMPLEX_MODE

;S1    S2   S3   S4    S5   S6   S7    S8   S9   S10   S11  12
CLK   NC   NC RES   NC   NC   NC   NC   NC   NC    NC    GND

;S12   S13   15   16   17   18   19    20    21   22   S14   24
/OE    NC    CO  g    f    e    d     c     b    a    NC    VCC

a      :=  /a * /b * /c * /d * /e * /f *  g *  CO *
       +  /a * /b * /c * /d *  e *  f * /g *  CO *
CLK_AUT

b      :=   a * /b * /c *  d *  e * /f * /g *  CO *
[LOW]: /OE
       +  /a *  b * /c * /d *  e * /f * /g *  CO *

c      :=   a * /b * /c *  d *  e *  f *  g *  CO *

d      :=  /a * /b * /c * /d * /e * /f *  g *  CO *
       +  /a * /b * /c * /d *  e *  f * /g *  CO *
       +  /a *  b * /c * /d * /e * /f * /g *  CO *

/e     :=   a * /b * /c *  d *  e *  f *  g *  CO *
       +  /a *   b * /c * /d *  e * /f * /g *  CO *
       +  /a * /b * /c *  d *  e *  f *  g *  CO *
       +  /a * /b * /c * /d *  e * /f * /g * /CO *
aussen )
       +                                      /RES

f      :=  /a * /b * /c * /d * /e * /f *  g *  CO *
CLK=Medium)
       +   a * /b * /c *  d *  e *  f *  g *   CO *
       +  /a * /b *  c * /d * /e *  f * /g *  CO *
       +  /a *  b * /c * /d * /e * /f * /g *  CO *
ausgelöst

g      :=  /a * /b * /c * /d * /e * /f *  g *  CO *
       +   /a *  b * /c * /d * /e * /f * /g *  CO *
       +  /a * /b * /c * /d *  e * /f * /g * /CO *
       +                                      /RES

/CO  := /a *  /b * /c * /d *  /e * /f * /g *  CO * RES

USER_ID = gev120398

;---------------------------------------------------------------
```

Funktionsprüfung:

Konfiguration der GDU32:

Kursiv gedruckt:
Standard Voreinstellung des GAL 20V8

- **Jumper JMP1 auf Stellung Q1/V20**
- **Jumper JMP2 auf Stellung IN13/V20**

- **Umschalter SU1 auf Stellung CLK/V20**
- **Umschalter SU2 auf Stellung IN3/V20**

```
RES
RES        - Umschalter SU3 auf Stellung CLK_MAN /


RES        - DIP-Schalter S12 auf Stellung ON

RES
- DIP-Schalter S15
RES                Stellung OFF -> CLK: Slow
 Stellung ON  -> CLK: Medium
RES
RES        - DIP-Schalter S16
RES                Stellung OFF -> CLK: Slow
  ON  -> CLK: Fast
RES
RES        Funktionstest:
RES
RES        - CLK automatisch ( Umschalter S15 nach

- CLK_STOP durch Drücken des Tasters TA5 ( CLK )

RES        - DIP-Schalter S15 (ON) / S16 auf OFF (

RES
RES        - Taster TA1 ( S04 ) drücken ( Low )
RES                Beim folgenden Takt wird RESET

 Counter arbeitet
RES
RES        - CLK-Frequenz variieren
RES
- CLK manuell testen
```

- TRISTATE-Funktion testen:
  DIP-Schalter S12 auf OFF
  Sämtliche LEDs erlöschen
  Keine Funktion

Beispiel 6: Aufbau eines n-stelligen Counters mit 7-Segmentanzeig

Erweiterte Konfiguration von Board 1:

- *Generelle Konfiguration wie oben angegeben*
- *Zusätzlich zur Aufbereitung des CLK-Signals für Board 2:*

a. Output 08 (CO bzw. DezimalPunkt) mit IN_01 des Schmitt-Triggers 74LS14 auf Board 1 verbinden
b. OUT_01 des 74LS14 mit IN1 (CLK) des Board 2 verbinden
c. GND und VCC von Board 1 entsprechend an Board 2 anlegen (gleiche Masse !)

Konfiguration von Board (n):

Fett gedruckt -> Abweichende Einstellungen zu Board 1:

- Jumper JMP1 auf Stellung Q1/V20
- Jumper JMP2 auf Stellung IN13/V20
- **Umschalter SU1 auf Stellung IN1/V20** [Clock-Signal von Board (n-1) aufschalten]
- Umschalter SU2 auf Stellung IN3/V20
- **Umschalter SU3 auf Stellung CLK_MAN**
- DIP-Schalter S12 auf Stellung ON [LOW]: /OE
- DIP-Schalter S15 / S16 auf Stellung ON oder OFF (kein Einfluß)

Funktionstest:

Board (n) zurücksetzen:

1. Möglichkeit: Taster TA1 (S04) drücken (LOW)
Beim folgenden Takt an Board (n) wird Reset ausgelöst / Counter arbeitet

2. Möglichkeit: Umschalter SU1 von Board (n) auf Stellung CLK/V20
Taster TA1 (S04 / Reset) drücken (Low)
Clock auslösen
Umschalter SU1 von Board (n) auf Stellung IN1_V20 / Counter arbeitet

Die vorgestellten Möglichkeiten gehen davon aus, daß der Anwender nicht unbedingt geneigt ist, den GDUs mit einem Lötkolben zu Leibe zu rücken und durch eine Hardware zu erweitern, die den mehrstelligen Counter durch eine entsprechend ausgelegte Resetschaltung auf Null setzt.

Beispiel 7: 8 Bit Rechts / Links-Schieberegister mit paralleler Eingabe

... für den TTL-Denker möglicherweise eine schreckeinflößende Aufgabe ...
... für den PLD-Denker eine erfrischen Übung ...
... das Ergebnis ist ‘ enttäuschend ‘ simpel ...
... nicht einmal ein KV-Diagramm muß für die ‘ Problem ?’-Lösung bemüht werden ...

Der Kern der Überlegungen ist wie immer die Frage:

‘ Welche Ausgangszustände müssen sich bei vorgegebenen Eingangsbedingungen nach einem Clock ergeben, damit das Logikverhalten des GALs die Funktion des geforderten Schieberegisters erfüllt ? ‘.

Sinnvollerweise sollten die Bedingungen wiederum in eine Logiktabelle eingetragen und daraus die Logikterme bzw. die Logikgleichungen formuliert werden.

Wahrheitstabelle des 8 Bit Schieberegisters mit paralleler Eingabe und Richtungswechsel

Pinvorgaben:

E0 ... E7	-> parallele Eingänge
Q0 ... Q7	-> parallele Ausgänge
CLK	-> Clock
/OE	-> Output Enable
LD	-> Laden [parallel]
MO	-> Mode:1[rechts] 0[links]
SI	-> Serial Input

E i n g ä n g e	Aktueller Eingangszustand	Ausgangszustand nach Clock
LD MO E7 E6 E5 E4 E3 E2 E1 E0	SI Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0	O7 O6 O5 O4 O3 O2 O1 O0

0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	SI	Q7	Q6	Q5	Q4	Q3	Q2	Q1
0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	Q6	Q5	Q4	Q3	Q2	Q1	Q0	SI
1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	E7	E6	E5	E4	E3	E2	E1	E0

Die Logikbedingungen lassen sich direkt aus der Tabelle ablesen.
So gelten beispielsweise für den Ausgang Q0 prinzipiell folgende Bedingungen:

Für jeden Ausgang sind jeweils 3 Ausgangszustände denkbar.

1. Wenn der Lademodus	NICHT aktiv ist [/LD] UND NICHT nach links geschoben wird [/MO], dann übernimmt Q0 nach dem Clock den Zustand von Q1	**Q0 = Q1 * /LD * /MO**
	ODER	
2. Wenn der Lademodus	NICHT aktiv ist [/LD] UND	+
	nach rechts geschoben wird [MO],	**Q0 = SI * /LD * MO**
	dann übernimmt Q0 den Zustand des seriellen Einganges SI	
	ODER	
3. Wenn der Lademodus	aktiv ist [LD],	+
	dann übernimmt Q0 den Zustand des Einganges E0	**Q0 = E0 * LD**

Für die vollständige Beschreibung aller Ausgangszustände ergibt sich folglich:

Ausgangsbedingung für Q0:	Q0 = Q1 * /LD * /MO + SI * /LD * MO + E0 * LD
Ausgangsbedingung für Q1:	Q1 = Q2 * /LD * /MO + Q0 * /LD * MO + E1 * LD
Ausgangsbedingung für Q2:	Q2 = Q3 * /LD * /MO + Q1 * /LD * MO + E2 * LD
Ausgangsbedingung für Q3:	Q3 = Q4 * /LD * /MO + Q2 * /LD * MO + E3 * LD
Ausgangsbedingung für Q4:	Q4 = Q5 * /LD * /MO + Q3 * /LD * MO + E4 * LD
Ausgangsbedingung für Q5:	Q5 = Q6 * /LD * /MO + Q4 * /LD * MO + E5 * LD
Ausgangsbedingung für Q6:	Q6 = Q7 * /LD * /MO + Q5 * /LD * MO + E6 * LD
Ausgangsbedingung für Q7:	Q7 = SI * /LD * /MO + Q6 * /LD * MO + E7* LD

Das folgende Source-File zeigt das GAL-File in der korrekten GDS35-Syntax:

```
8 Bit Rechts-Links-Schieberegister mit Pre-Load


CHIP   SHFTLR8  GAL20V8A  COMPLEX_MODE

;S1    S2   S3   S4   S5   S6   S7   S8   S9   S10   S11  12
CLK  E0   E1   E2   E3   E4   E5   E6   E7   SI    LD   GND

;S12  S13  15   16   17   18   19   20   21   22    S14  24
/OE   MO  Q7   Q6  Q5   Q4   Q3   Q2  Q1  Q0    NC   VCC

; Logikgleichungen

Q0     :=          Q1 * /LD * /MO
           +       SI *   /LD *  MO
           +       E0 * LD

Q1     :=          Q2 * /LD * /MO
           +       Q0 * /LD * MO
           +       E1 * LD

Q2     :=          Q3 * /LD * /MO
           +       Q1 * /LD * MO
           +       E2 * LD

Q3     :=          Q4 * /LD * /MO
           +       Q2 * /LD * MO
          +       E3 * LD

Q4     :=          Q5 * /LD * /MO
          +       Q3 * /LD * MO
          +       E4 * LD

Q5     :=          Q6 * /LD * /MO
           +       Q4 * /LD * MO
          +       E5 * LD

Q6     :=          Q7 * /LD * /MO

           +       Q5 * /LD * MO
          +       E6 * LD
geschoben

Q7     :=          SI * /LD * /MO
geschoben
           +       Q6 * /LD * MO
           +       E7 * LD

USER_ID = gev040398
```

Funktionsprüfung

Konfiguration der GDU32:

Kursiv gedruckt:
Standard Voreinstellungen des GAL 20V8

- **Jumper JMP1 auf Stellung Q1/V20**
- **Jumper JMP2 auf Stellung IN13/V20**

- **Umschalter SU1 auf Stellung CLK/V20**
- **Umschalter SU2 auf Stellung IN3/V20**

- Umschalter SU3 auf Stellung CLK_MAN
- **DIP-Schalter S12 auf Stellung ON [LOW]: /OE**

- DIP-Schalter S15 und S16 auf [ON]
  Clock-Delay [Short]

Funktionstest:

Mit Hilfe der DIP-Schalter S2 bis S9 Bitmuster einstellen.
Schalter OFF -> logisch 1
Schalter ON -> logisch 0
DIP-Schalter S11 [LD] und S13[MO] auf OFF (1)

Beispiel:

Mit DIP-Schalter:	S2	S3	S4	S5	S6	S7	S8	S9
Bitmuster einstellen.	0	0	0	0	1	1	0	1
Inputs:	/E0	/E1	/E2	/E3	E4	E5	/E6	E7

- DIP-Schalter S11 auf OFF [Load-Modus]
- Clock betätigen: Bitmuster wird übernommen
  LEDs zeigen obiges Bitmuster
  7 Segment- Anzeige zeigt die Ziffer [3]
- DIP-Schalter S11 [LD] auf ON [Schiebemodus]

- Clock betätigen:
  Mit jedem Clock wird das Bitmuster nach rechts

- DIP-Schalter S13 [MO] auf ON (0)
  Mit jedem Clock wird das Bitmuster nach links

- Im Schiebemodus [S11 = ON] wird mit jedem Clock der an S10 (Seriell Input SI) anstehende Pegel übernommen

Beispiel 8: Grundprinzip der synchronen seriellen Datenübertragung

Am folgenden Beispiel soll das Grundprinzip der synchronen seriellen Datenübertragung, wie sie auch der I^2C- oder der SPI-Schnittstelle zugrunde liegt, demonstriert werden.

Es werden drei GDU32 Boards entsprechend der Darstellung miteinander gekoppelt.
Für den einwandfreien Ablauf der seriellen Datenübertragung ist es erforderlich, einige Pegelanpassungen vorzunehmen.

1. Die Bitmuster werden von einer [GDU32] mit dem Programm [CNT7SEG] generiert und stehen an den Ausgängen wegen der Treiber ULN2803 als invertierte Signale zur Verfügung.
   Aus diesem Grunde müssen die Eingänge E0 - E7 der TRANSMITTER-GDU nochmals invertiert werden.

(Siehe hierzu: Pindefinition - TRANSMITTER)

2. Das niederwertigstes Datenbit des TRANSMITTER-Ausganges [SERIAL_OUT] wird durch den Schmitt-Trigger invertiert, aufbereitet und auf den seriellen Eingang [SERIAL_IN] der Receiver-GDU geschaltet.

Transmitter: SERIAL_OUT => Schmitt-Trigger => Receiver: SERIAL_IN

3. Das am Ausgang des Schmitt-Triggers anstehende Clocksignal [CLK] wird ebenfalls invertiert und auf den Clock-Eingang [SYNC_CLK] der Receiver-GDU geschaltet.

Transmitter: CLK => Schmitt-Trigger => Receiver: SYNC_CLK

Programmierhinweise:

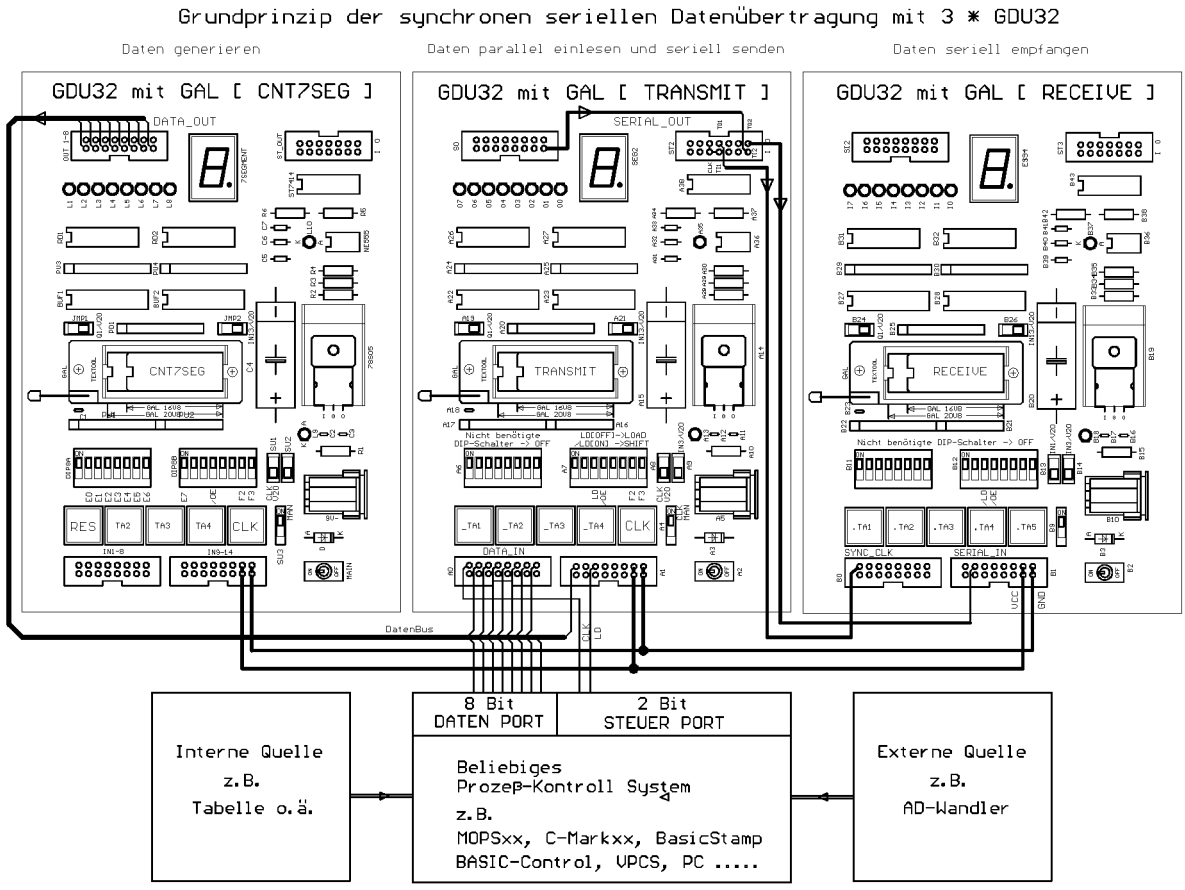
Bis auf eine Abweichung bei der Pindefinition des Transmitter-GALs sind die Source-Files für den seriellen Sender und Empfänger absolut identisch mit dem 8 Bit Schieberegisters mit paralleler Eingabe und Richtungswechsel [Beispiel 7]

TRANSMITTER-GAL: **REVEIVER-GAL:**

CHIP *TRANSMIT* GAL20V8A COMPLEX_MODE CHIP *RECEIVE* GAL20V8A COMPLEX_MODE

CLK **/E0 /E1 /E2 /E3 /E4 /E5 /E6 /E7** SI LD GND CLK *E0 E1 E2 E3 E4 E5 E6 E7* SI LD GND

Konfiguration der GDUs:



- Die Grundkonfiguration entspricht den *Standard Voreinstellungen des GAL 20V8* für Registerbetrieb, wie in den vorangegangenen Beispielen beschrieben.

- Spezielle Konfiguration der einzelnen GDUs:

Es sind nur die für die serielle Datenübertrageung relevanten Schalter ausgelistet

GDU mit GAL [CNT7SEG]:	SU3 auf CLK_MAN	-> Clock-Mode [manuell]
	SU1 auf CLK_V20	-> Clock [manuell] aufschalten
	DIP-Schalter S12 auf ON	-> [/OE] Output Enable
	DIP-Schalter S15 und S16 auf ON	-> [F2/F3] CLK-Delay minimal
GDU mit GAL [TRANSMIT]:	SU3 auf CLK_MAN	-> Clock-Mode [manuell]
	SU1 auf CLK_V20	-> Clock [manuell] aufschalten
	SU2 auf IN3/V20	-> E3 auf IN3 schalten
	DIP-Schalter S10 auf OFF	-> [SI] Seriell In (Fill with 1)
	DIP-Schalter S11 auf OFF	-> [LD] Load Data
	DIP-Schalter S11 auf ON	-> [/LD] Shift Data
	DIP-Schalter S12 auf ON	-> [/OE] Output Enable
	DIP-Schalter S13 auf OFF	-> [MO] Shift Right
	DIP-Schalter S15 und S16 auf ON	-> [F2/F3] CLK-Delay minimal
GDU mit GAL [RECEIVE]:	SU1 auf IN1/V20	-> SYNC_CLK aufschalten
	DIP-Schalter S10 auf OFF	-> [SI] Serial In
	DIP-Schalter S11 auf ON	-> [/LD] Shift Data
	DIP-Schalter S12 auf ON	-> [/OE] Output Enable
	DIP-Schalter S13 auf OFF	-> [MO] Shift Right

Funktionstest:

Zur seriellen Übertragung *eines Daten Bytes* sind manuell folgende Programmschritte auszuführen:

START:

INIT_GDUS: GDUs entsprechend der Abbildung konfigurieren

=> **Hardware konfigurieren**

LOOP:

CNT7SEG Mit Counter-GAL eine beliebige Ziffer (8 Bit) generieren.
Bitmuster (durch LEDs kontrollierbar) steht parallel an den Eingängen E0 - E7 des Sende-GALs an.

=> **DATUM generieren**
=> **DATUM an Senderegister anlegen**

TRANSMITTER in Load-Modus [LD] schalten und Clock auslösen.
Das an E0 - E7 anstehende Bitmuster (Datum) wird übernommen

=> **DATUM in Senderegister übernehmen**

TRANSMITTER in Shift-Modus [/LD] schalten und **8 CLOCKS** auslösen.
Mit jedem Clock werden die Datenbits um eine Stelle nach rechts aus dem Senderegister herausgeschoben

=> **DATUM senden**

RECEIVER mit Empfangs-GAL befindet sich im seriellen Einlesemodus [LD]
Da der RECEIVER synchrom mit dem TRANSMITTER taktet, werden die aus dem Senderegister herausgeschobene Datenbits vom Empfangsregister synchron eingelesen

=> **DATUM empfangen**

... Sendevorgang für ein Byte abgeschlossen ...

```
GOTO LOOP     ; Repeat until Christmas equates Easter
```

Allgemeine Formulierung des Programmablaufes:

```
START:          ; Here we go ...
INIT_HRDWRE:    HARDWARE konfigurieren

LOOP:

GET_DAT:        DATUM generieren
PUT_DAT:        DATUM an Senderegister anlegen
READ_DAT:       DATUM in Senderegister übernehmen
SEND_DAT:       DATUM senden
RECEIVE_DAT:    DATUM empfangen

GOTO LOOP       ; Please do it again ...
```

Da bei der GDU sämtliche Steuerpins herausgeführt sind, bietet es sich an, statt der *seriellen Handarbeit* ein wie auch immer gestaltetes Prozeßkontrollsystem zu bemühen, um die einzelnen Programmschritte automatisch abzuarbeiten.
Wegen der Systemunabhängigkeit der GDU kann die serielle Datenübertragung mit jedem System, welches über ein Port mit mindestens 3 Ausgängen [Counter-Clock, Transmitter / Receiver-Clock, Load / Shift-Steuerung] verfügt, gesteuert bzw. simuliert werden.
Soll auf den Counter als Datengenerator verzichtet werden, dann muß ein weiterer 8 Bit Port zur Verfügung stehen über den wie auch immer generierte Daten an den Transmitter ausgegeben werden können.

Im folgenden wird ein mögliches Steuerprogramm in einer *autorindividuellen Programmiersprache* vorgestellt, bei dem die Sendedaten nicht von einer CNT7SEG-GDU stammen.

```
START:                                  ; Here we go again ...

INIT_PORTS:  INIT    DATA_PORT          ; Daten Port vollständig als Ausgang definieren
             INIT    CTRL_PORT          ; 2 Bits des Steuerports als Ausgang definieren
LOOP:

GET_DAT:     LOAD    DATA               ; Datum z.B.aus Tabelle laden
PUT_DAT:     MOVE    DATA               ; Datum an GAL-Eingänge anlegen
LOAD_DAT:    SET     LOAD               ; Load-Modus einschalten [ LD-Bit auf 1 setzen ]
             MAKE    1_CLK              ; Mit folgendem Clock wird Datum übernommen
SHIFT_DAT:   SET     SHIFT              ; Shift-Modus einschalten  [ LD-Bit auf 0 setzen ]
             MAKE    8_CLKS             ; Mit 8 Clocks wird Datum gesendet

REPEAT:      GOTO    LOOP               ; Repeat until forever ...
```

Damit wenigstens diejenigender voll auf ihre Kosten kommen, die über den ehrwürdigen 65x02 ihren Weg in die Computertechnik gefunden haben, (es gibt *ihn* immer noch, *den 65x02* und es gibt *sie* immer noch, *die sich noch damit beschäftigen*) wird zum Abschluß ein vollständig funktionsfähiges und getestetes Assemblerprogramm vorgestellt, welches *in beeindruckender Weise* die oben aufgezeigte serielle Datenübertragung von der einen zur anderen GDU steuert.
Mit dem verwendeten Prozeßkontrollsystem (PROCOM-6C-65 / ELRAD 5 / 92) werder die Ziffern 0 bis 9 periodisch aus einer Tabelle gelesen und die serielle Datenübertragung von der einen zur anderen GDU gesteuert.

Wer dem Programm das gewisse I-Tüpfelchen aufsetzen möchte, sollte sich an eine Programmerweiterung heranwagen, mit der nicht nur ein unidirektionaler, sondern ebenfalls ein bidirektionaler Datenstrom kontrolliert wird. Dem Einfallsreichtum des Programmierers sind hier so gut wie keine Grenzen gesetzt..

Wer den 65x02 nicht mag, weil er mit dem Z80 oder dem 8085 groß geworden ist oder wer überhaupt kein Verhältnis zur Assemblerprogrammierung hat, sondern sich in C++ oder einer anderen Hochsprache Zuhause fühlt, ist in seiner Entscheidung bezüglich der Problemlösung absolut frei.

Nun zum konkreten Steuer-Programm:

Sowohl das Assembler Source-File (SEND_SER.A65) als auch die Beschaltung der GDUs werden ohne weitere Kommentierung durch den Autors vorgestellt.

```
;  ---------------------------------------------------------------------------------------------------------------
; Titel: SEND_SER.A65 / Synchrone serielle Datenübertragung [ Prinzip I²C ]
; Version 1.2 /  980310
; Zielsystem: PROCOM 6C-65 [ VPCS ]
; Assembler: UCASM [ fme ]
; Entwickler: gev
;  ---------------------------------------------------------------------------------------------------------------
;   Beschaltung der GDUs
;  ---------------------------------------------------------------------------------------------------------------
;   von Ports          an Transmitter          an Schmitt-Trigger          an  Receiver
;  ---------------------------------------------------------------------------------------------------------------
;    GND      ->       GND                                                 ->  GND
;    ------             VCC                                                 ->  VCC
;    PA5      ->       IN11 [ LD ]                                         ->  IN11 [ LD ]
;    PA7      ->       IN1 [ CLK ]
;    PA7                          -> IN_2 -> OUT_2 -> IN_1 -> OUT_1       ->  IN1 [ CLK ]
;    PB0      ->       IN2 [ D0 ]
;    PB1      ->       IN3 [ D1 ]
;    PB2      ->       IN4 [ D2 ]
;    PB3      ->       IN5 [ D3 ]
;    PB4      ->       IN6 [ D4 ]
;    PB5      ->       IN7 [ D5 ]
;    PB6      ->       IN8 [ D6 ]
;    -----              OUT8            -> IN_3 -> OUT3 -> IN4 -> OUT4      ->  IN10 [ SI ]
;  ---------------------------------------------------------------------------------------------------------------
;    DIP-Schalter    IN12 [ /OE ] auf [ ON ]                                    IN12 [ /OE ]
;  ---------------------------------------------------------------------------------------------------------------
;    Alle anderen DIP-Schalter auf [ OFF ]
;  ---------------------------------------------------------------------------------------------------------------

.NOLIST
.INCLUDE "PROC_SYS.ADR"                 ; Systenroutinen des PROCOM6C-65
.LIST

CODEBEGIN  .EQU   $2000                 ; Programm-Startadresse
PORTB      .EQU   PPI                   ; Port B -> Data-Port
DDRB       .EQU   PPI+2                 ; Direction Register Port B
PORTA      .EQU   PPI+1                 ; Port A -> Control-Port
DDRA       .EQU   PPI+3                 ; Direction Register Port A

                ;-----------------------------------------------------------------------------------------------
                ; Konstanten
                ;-----------------------------------------------------------------------------------------------
INIT_OUT   .EQU   % 1111 1111           ; PortB komplett als Output
INIT_CTRL  .EQU   % 1011 0000           ; PortA: CLK, /OE, LD = Output / Rest Input
SET_LOAD   .EQU   % 0010 0000           ; Steuersignal: Load=1, CLK=0
LOAD_CLK   .EQU   % 1010 0000           ; Steuersignal: Load=1, CLK=1
SET_SHIFT  .EQU   % 0000 0000           ; Steuersiganl: Load=0 -> schieben
```

```
SHIFT_CLK   .EQU   % 1000 0000      ; Clock=1 -> Schieben
TIME        .EQU   5               ; Delay = TIME * 0.1 sec

            ;-------------------------------------------------------------------------------------------------
            ; Definition der Bitmuster fuer Ausgabe-Ziffern 0 - 9
            ;-------------------------------------------------------------------------------------------------

DATE0       .EQU  % 0011 1111      ; Ziffer 0
DATE1       .EQU  % 0000 0110      ; Ziffer 1
DATE2       .EQU  % 0101 1011      ; Ziffer 2
DATE3       .EQU  % 0100 1111      ; Ziffer 3
DATE4       .EQU  % 0110 0110      ; Ziffer 4
DATE5       .EQU  % 0110 1101      ; Ziffer 5
DATE6       .EQU  % 0111 1101      ; Ziffer 6
DATE7       .EQU  % 0000 0111      ; Ziffer 7
DATE8       .EQU  % 0111 1111      ; Ziffer 8
DATE9       .EQU  % 0110 1111      ; Ziffer 9
            ;-------------------------------------------------------------------------------------------------

            .CODE CODEBEGIN        ; Programm-Startadresse

main:       jsr     init           ; Ports initialisieren
reset:      ldx     #9             ; Pointer auf zehn Ziffern in Tabelle
loop:       jsr     keycode1       ; Wait until Keypress
            jsr     ld_pat         ; Daten aus der Tabelle laden
            jsr     keycode1       ; Wait until keypress
            jsr     shift          ; Daten seriell senden
            dex                    ; x=x-1
            bpl     loop           ; Alle Ziffern ausgegeben? Nein, dann Loop
            bra     reset          ; Ja, dann von Vorne beginnen

            ;-------------------------------------------------------------------------------------------------
            ; Unterprogramm: init
            ;-------------------------------------------------------------------------------------------------
init:       lda     #INIT_OUT
            sta     DDRB           ;PortB vollständig Output
            lda     #INIT_CTRL
            sta     DDRA           ;PortA nur für Clk, LD, OE Output
            rts

            ;-------------------------------------------------------------------------------------------------
            ; Unterprogramm: ld_pat
            ;-------------------------------------------------------------------------------------------------
ld_pat:     lda     tabelle,x      ; Zeiger auf auszugebende Ziffer stellen
            sta     PORTB          ; Datum an Transmitter-GAL legen
            lda     #SET_LOAD
            sta     PORTA          ; Laden vorbereitet
            jsr     delay_01s      ; Warten, um Ablauf beobachten zu können
            lda     #LOAD_CLK
            sta     PORTA          ; Mit CLK wird Pattern vom GAL übernommen
            jsr     delay_01s      ; Warten, um Ablauf beobachten zu können
            rts

            ;-------------------------------------------------------------------------------------------------
            ; Unterprogramm: delay_01s
            ;-------------------------------------------------------------------------------------------------
delay_01s:  phx                    ; Delay-Routinen zum besseren Kontrollieren
            phy                    ; der Aktionen auf den GDUs
            pha                    ; Mit push und pull werden die Register
```

```
                lda     #time           ; vorsichtshalber gesichert
                 jsr    t01s
                pla
                ply
                plx
                rts

                ;-------------------------------------------------------------------------------------------------
                ; Unterprogramm: shift
                ;-------------------------------------------------------------------------------------------------
shift:          ldy     #8              ; 8x schieben
shiften:        lda     #SET_SHIFT      ; % 0000 0000
                sta     PORTA           ; Load-Signal wird deaktiviert
                jsr     delay_01s       ; Warten, um Ablauf beobachten zu können
                lda     #SHIFT_CLK      ; % 1000 0000
                sta     PORTA           ; Mit Clocksignal wird um ein Bit geschoben
                 jsr    delay_01s       ; Warten, um Ablauf beobachten zu können
                dey                     ; Alle Bits geschoben ? Nein,
                 bne    shiften         ; dann weiter schieben, sonst
                 jsr    delay_01s       ; Warten, um Ablauf beobachten zu können
                 rts

                ;-------------------------------------------------------------------------------------------------
                ; Datenbereich
                ;-------------------------------------------------------------------------------------------------
tabelle:        .byte DATE9, DATE8, DATE7, DATE6, DATE5
                .byte DATE4, DATE3, DATE2, DATE1, DATE0

                .ENDS
                .END
                ;-------------------------------------------------------------------------------------------------
```

Abschließende Anmerkungen:

Der Autor wäre zufrieden, sollte es ihm gelungen sein, mit seinen Beiträgen das Interesse des Lesers im Umgang mit programmierbaren Logikbausteinen im allgemeinen und den GALs im besonderen geweckt oder wiederbelebt zu haben.
Er wünscht allen inzwischen hoffentlich GAL-Begeisterten viel Freude und Erfolg beim experimentellen Lernen mit der GDU.
Für all diejenigen, die weitere Informationen und Übungsbeispiele suchen, wird auf zwei Quellen verwiesen, aus denen der Autor ebenfalls wertvolle Anregungen beziehen konnte.
Dieses ist zum einen
das GAL Entwicklungspaket GDS3.5 der Firma SH-Elektronik, in dem zahlreiche interessante GAL Anwendungen vorgestellt werden, die in Anlehnung an das empfehlenswerte GAL-Buch von D. Bitterle entstanden sind und
zum anderen
eben dieses Buch von Dieter Bitterle.
Titel: ' GALs - Programmierbare Logikbausteine in Theorie und Praxis '
3. Auflage / Franzis Verlag / [ISBN 3-7723-5904-3]

Bezugsquellen für das GAL Evaluationsboard:

Die GDU32 ist ab Mitte April bei folgenden Firmen zu beziehen:

PCS Prozessrechnertechnik & WebDesign
D-27753 Delmenhorst
Matthias Claudius Weg 26
E-Mail: pcs@ae-web.com
http://www.ae-web.com/pcs

oder

SH-Elektronik
D-24114 Kiel
Marthastraße 8
Tel.: 0431 - 8803838
E-Mail: hipp@physik.uni-kiel.de
http://www.sh-elektronik.de

Die GDU32 wird angeboten als:

1. Leerplatine / doppelseitig / Lötstopmaske / Siebdruck DM 56,-
2. Vertrieb als Bausatz (beide Firmen) DM 189,-
3. Vertrieb als Fertiggeraet (beide Firmen) DM 219,-

Die Preise verstehen sich als Endpreise zuzüglich der Versandkosten.