1. Grundlagen der GALs

1.1 Entwicklungsgeschichte der GALs

Programmierbare Logikbausteine (PLDs = Programmable Logic Devices) sind heute wichtige Komponenten beim Design elektronischer Systeme. Die flexible Realisierung der benötigten Logikfunktionen führt zu einer Verringerung der erforderlichen Bauelemente und damit zu einer Platzeinsparung auf der Leiterplatte. Auch die Forderung nach höherer Integration und nach kürzeren Entwicklungszeiten wird durch die PLDs unterstützt. Gegenüber der ähnlich eingesetzten anwendungsspezifischen integrierten Schaltung (ASIC) hat der PLD den Vorteil, dass er als Standardprodukt preiswerter hergestellt werden kann.

Ein PLD besteht im Wesentlichen aus digitalen logischen Funktionsblöcken, die in einer regelmässigen Struktur (Array) angeordnet sind, und einem Netzwerk von Verbindungen zwischen diesen Blöcken. Die speziellen Funktionen der einzelnen Blöcke und die Auswahl der benötigten Verbindungen sind vom Anwender frei programmierbar. Damit kann eine bestimmte logische Schaltung implementiert werden, man spricht dabei von einer Personalisierung des Bausteins.

Die ersten programmierbaren Logikbausteine waren die in bipolarer Technik hergestellten PALs (Programmable Array Logic). Ein wesentlicher Nachteil dieser Bauelemente ist jedoch die Tatsache, dass sie nur einmal programmierbar sind.

Im Jahre 1985 entwickelte die Fa. LATTICE eine neue Generation programmierbarer Logikbausteine, die GALs (Generic Array Logic). Die für diese Bauteile entwickelte EECMOS-Technik kombiniert den CMOS-Prozess mit einer elektrisch löschbaren Speichertechnologie. Diese Technologie lässt mittlerweile eine bis zu 10 000-fache (Re-) Programmierung zu.

Die bisher beschriebenen PLDs werden auch als SPLDs (Simple PLDs) bezeichnet. Ihre Komplexität liegt im Bereich bis zu ca. 10 000 Gatteräquivalenten. Alle zu verknüpfenden Logiksignale werden auf eine gemeinsame UND-Matrix geführt. Es gibt in einem SPLD also nur globale Verbindungen. Auf diese Weise werden Bausteine bis maximal 32 Ausgangsmakrozellen realisiert.

Da noch größere Bausteine zu einer großen UND-Matrix führen würden, wurde mit den CPLD-Bausteinen (Complex PLD) eine neue Architektur eingeführt. Dabei werden mehrere PAL-ähnliche Blöcke (LAB = Logic Array Block) auf einem Chip mittels einer Schaltmatrix miteinander verbunden. Hier sind die verknüpften Logiksignale nicht mehr fest an eine bestimmte Makrozelle gebunden, sondern können nach Bedarf verschaltet werden. Charakteristisch für die CPLDs sind der hierarchische Aufbau aus einzelnen LABs und dass es in den einzelnen LABs lokale Verbindungen gibt.

Nach den (S)PLDs und CPLDs stellen die FPGAs (Field Programmable Gate Array) die dritte Variante programmierbarer Logik dar. Her sind viele kleine Logikzellen auf einem Array angeordnet und können über ein Neztwerk von Verbindungsleitungen miteinander verschaltet werden. Die speziellen Funktionen der einzelnen Zellen und die Auswahl der benötigten Verbindungen sind programmierbar.

1.2 Vor- und Nachteile der GALs

Die Vorteile der GALs sind:

Kurze Entwicklungszeit für Aufbau und Test der Schaltung, das Layout kann parallel zur Schaltungsentwicklung erfolgen,

Platzeinsparung auf der Leiterplatte,

Reduzierung der Bauteile-Typenvielfalt,

Hohe Flexibilität: Leichte Durchführung von Änderungen auch in späten Entwicklungsphasen.

EECMOS-Technologie: Bietet die Geschwindigkeit bipolarer Bausteine mit verminderter Verlustleistung,

Kopierschutz gegen unerlaubtes Vervielfältigen.

Nachteil der GALs:

Es wird eine Entwicklungs-Software und ein entsprechendes Programmiergerät benötigt.

1.3 GAL-Entwicklungs-Software

Die Schaltungsbeschreibung wird in Form von Booleschen Gleichungen, Wertetabellen oder von Zustandstabellen über einen Texteditor zur Weiterverarbeitung mit dem GAL-Assembler eingegeben. Diese Hardwarebeschreibungssprache wird generell mit HDL (Hardware Description Language) bezeichnet. Als Standards haben sich dabei ABEL-HDL (Advanced Boolean Expression Language) und für schnelle Schaltungen VHDL (Very High Speed Integrated Circuit - HDL) herausgebildet.
Für komplexe Aufgaben stehen auch Editoren mit grafischer Eingabe des Schaltplans zur Verfügung.
Mit dem GAL-Compiler wird die Schaltungsbeschreibung dann in eine Information umgesetzt, die vom Programmiergerät gelesen werden kann. Dies ist eine so genannte JEDEC-Datei, die als genormte Schnittstelle von allen Compilern unterstützt wird.
Viele Softwarepakete bieten zusätzlich noch die Möglichkeit der Simulation der im GAL abgelegten Funktion.
Gängige Entwicklungs-Softwarepakete sind z.B.:
easyABEL, ABEL, Synario (Data I/O), Log/iC (ISDATA), PLAN, OPTIMAL (NS), OrCAD PLD (OrCAD), Schema-PLD (Omation), Tango PLD (Accel), PALASM (AMD) u.v.a.

1.4 GAL-Programmiergerät

Mit dem Programmiergerät werden die Informationen der JEDEC-Datei in den GAL-Baustein übertragen. Dabei werden die programmierbaren Verbindungen ("Sicherungen") im GAL entsprechend den Vorgaben gelöscht.
Manche Programmiergeräte bieten zusätzlich die Möglichkeit den GAL mit Hilfe vorgegebener Testvektoren auf seine Funktion zu testen.

2. Interne Struktur der GALs

2.1 Die Logikmatrix

Die interne Struktur der GALs wird durch die Logikmatrix bestimmt. Sie ist eine übersichtliche Darstellungsform der Verknüpfungen, die im GAL programmiert werden. Sie besteht aus einer Vielzahl von Zeilen und Spalten, an die sich die Ausgangszellen (OLMC = Output Logic Macrocell) anschließen.
Von den Eingängen werden jeweils das nicht invertierte und das invertierte Signal auf die Spalten der Matrix geführt. Auch die Signale der Ausgangszellen werden (teilweise) in die Matrix zurückgekoppelt und stehen so auch für Verknüpfungen zur Verfügung.
Die Zeilen der Matrix sind über UND-Verknüpfungsglieder mit einer Ausgangszelle verbunden, jede Zeile entspricht hierbei einem Produktterm. In der Ausgangszelle werden die Produktterme ODER-verknüpft. Das Ergebnis dieser Verknüpfung gelangt dann auf einen (Register-)Ausgang.
Das GAL entspricht damit der Regel der Booleschen Algebra, dass jede Logikfunktion durch die ODER-Verknüpfung von Produkttermen verwirklicht werden kann.
In der Logikmatrix sind die UND-Verknüpfungen der Übersichtlichkeit wegen nur durch Kreuze angedeutet. Diese Kreuze geben an, welche Eingangssignale in dieser Zeile UND-verknüpft sind, sie entsprechen also einer Verbindung zwischen Spalte und Zeile.
Beim unprogrammierten (gelöschten) GAL sind alle Verbindungen geschlossen, erst beim Programmiervorgang werden einzelne Verbindungen entfernt und damit die gewünschte Logikfunktion erzielt. Eine Zeile, in der noch alle Verbindungen intakt sind, hat also keinen Einfluss auf die Logikfunktion, da hier alle Eingangssignale mit ihren negierten Werten UND-verknüpft sind und sich damit für die nachfolgende ODER-Verknüpfung immer der Wert 0 ergibt.

Die Anzahl der für eine Ausgangsvariable verknüpfbaren Produktterme ist vom GAL-Typ abhängig, beim GAL 16V8 sind es acht und beim GAL 22V10 sind es bis zu sechzehn Produktterme.

2.2 Die Betriebsmodi der GALs

Neben den Eingängen verfügen dieGALs über konfigurierbare Ausgänge, deren Struktur vom Anwender selbst festgelegt werden kann.

Betriebsmodus 1: Kombinatorischer Ausgang

In diesem Betriebsmodus sind die konfigurierbaren Ausgangszellen als kombinatorische Ausgänge definiert. Die Ausgänge sind ständig aktiviert und hängen direkt von den logischen Werten der Eingangssignale ab. Die Logikschaltung wird mit Verknüpfungsgliedern aufgebaut und enthält keine Speicherglieder. Die Zuordnung der Eingangsvariablen zu den Ausgangsvariablen erfolgt hier häufig über eine Wahrheitstabelle, die in entsprechende Produktterme umgesetzt wird. Die Ausgänge können wahlweise als aktiv High oder aktiv Low definiert werden.

Betriebsmodus 2: Tristate-Ausgang

Hier sind bei den Ausgangszellen Tristate-Ausgänge vorgesehen, die durch ein Steuersignal hochohmig geschaltet werden können.
Ein Tristate-Ausgang mit Rückführung kann (im hochohmigen Zustand) auch als zusätzlicher Eingang verwendet werden.

Betriebsmodus 3: Register-Ausgang

Beim Register-Ausgang wird ein D-Flipflop zwischen das Verknüpfungsergebnis und den eigentlichen Ausgang geschaltet. Diese Flipflop bewirkt, dass das Ergebnis einer Verknüpfung nicht sofort am Ausgang erscheint, sondern erst mit der Flanke eines extern zugeführten Taktsignals dorthin gelangt. Man erhält dadurch einen synchronen Betrieb, da das Taktsignal allen Flipflops gleichzeitig zugeführt wird.
Das Ausgangssignal der Register-Ausgänge wird zurückgeführt. Ein Tristate-Ausgang ist auch hier möglich.

Aufbau des Quellprogramms

Header: Der Header enthält den Namen des Moduls.

Kommentare: Sie beginnen mit einem doppelten Anführungszeichen und enden entweder mit dem Zeilenende oder einem weiteren Anführungszeichen.

Declarations: Hier kann bereits ein spezieller GAL-Typ angegeben werden, die Auswahl kann aber auch später durch das Entwicklungsprogramm erfolgen.
Die Anschlüsse des GALs werden den Signalnamen zugeordnet, eine falsche Zuordnung von Ein- und Ausgängen wird vom Compiler entdeckt und als Fehlermeldung ausgegeben. Jede Anweisung wird mit einem ";" beendet, auch wenn sie sich über mehrere Zeilen erstreckt.

Mit „ISTYPE“ legt man die Signalattribute von PIN-Deklarationen fest, folgende Attribute sind möglich:

“REG“	Register	“INVERT“	Ausgangsinverter
“REG_D“	D-Register	“BUFFER“	kein Ausgangsinverter
“REG_T“	T-Register	“COM“	ungetaktet, kein Registerausgang

“REG_SR“	SR-Register	“POS“	positive Ausgangspolarität (Standard)
“REG_JK“	JK-Register	“NEG“	negative Ausgangspolarität
“REG_G“	D-Register mit „gated clock“	„d“XOR“	erzwingt XOR-Ausgang

Es sind spezielle Konstanten mit folgender Bedeutung bereits vordefiniert:

.C.	Taktimpuls (low **à** high)	.U.	steigende Flanke (low **à** high)
.D.	fallende Flanke (high **à** low)	.X.	don’t care-Bedingung
.F.	Signal weder high noch low	.Z.	Tristate-Ausgang
.K.	Taktimpuls (high **à** low)	H	high (logisch „1“)
.P.	Register vorladen	L	low (logisch „0“)

Logic Description: Hier wird das Design in beliebiger Zusammenstellung mit Logikgleichungen, Wertetabellen und Zustandsdiagrammen beschrieben. Im abschnitt 3.1.2 sind diese genauer beschrieben.

Test Vectors: Dieser Abschnitt ist optional. Mit den Testvektoren überprüft der Entwickler die Funktion des Designs, indem er alle Eingangs- und Ausgangssignal-Zustände angibt.

End: Mit dem Schlüsselwort „END“ wird das Modul beendet. Der Modulname kann hierbei auch entfallen.

3.1.2 Logikgleichungen

In den Logikgleichungen ist auf die Reihenfolge der Prioritäten zu achten. Folgende Operatoren werden im Allgemeinen verwendet:

Tabelle 2.2 a – Logische Operatoren

Verknüpfung	**Standard**	**Alternativ**	Priorität
NICHT	!	/	1 (höchste)
UND	&	*	2
ODER	#	+	3
XOR	$	:+:	3
XNOR	!$	:*:	3

Tabelle 2.2 b – Arithmetische Operatoren

		Priorität
Zweierkomplement	-A	1 (höchste)
Multiplikation	A*B	2
Division	A/B	2
Modulus	A%B	2
A um B Bits nach links schieben	A<<B	2
A um B Bits nach rechts schieben	A>>B	2
Addition	A+B	3
Subtraktion	A-B	3

Tabelle 2.2 c – Relationale Operatoren

		Priorität
gleich	==	4
ungleich	!=	4
kleiner	<	4
kleiner oder gleich	<=	4
größer	>	4
größer oder gleich	>=	4

Tabelle 2.2 d – Zuweisungs-Operatoren

kombinatorisch	=
Register	:=