

NACHRICHTENTECHNISCHES PRAKTIKUM

PROGRAMMIERUNG EINES KOMPLEXEN LOGIKBAUSTEINES

1. Einführung	2
2. Programmierbare Logikbausteine (PLDs) – Überblick	4
2.1 Produkttermbasierte PLDs.....	4
2.2 Programmierbare Gate Arrays	7
3. Die Entwicklungsumgebung MAX + plus II	9
3.1 Eingabe der Schaltung	9
3.2 Aufteilung des Projektes in einzelne Module.....	10
3.3 Kompilation des Projektes.....	11
3.4 Simulation.....	11
4. Das Projekt: Synchronisation eines DVB-Empfängers	11
5. Die Realisierung	13
5.1 Der Sync-Zähler.....	13
5.2 Der Prü fzähler.....	13
5.3 Die Steuerung der Zähler.....	14
5.4 Der Sync-Byte-Detektor	16
6. Die Versuchsdurchführung	17
6.1 Vorbereitungen	17
6.2 Der Umgang mit dem Graphik Editor	18
6.3 Die Eingabe mit dem Text-Editor	20
6.4 Die Top-Level-Datei.....	20
6.5 Die Kompilation	21
6.6 Die Simulation.....	21
6.6.1 Überprüfung der Schaltung auf ihre prinzipielle Funktionstüchtigkeit.....	21
6.6.2 Ermittlung der Leistungsfähigkeit der Schaltung.....	23
6.7 Programmierung des PLDs.....	25
6.8 Funktionstest und Messung	25
7. Zusammenfassung	26
8. Abkürzungen	27
9. Literatur	28

1. Einführung

Programmierbare Logikbausteine, kurz PLDs (Programmable Logic Device), bieten die Möglichkeit, logische Schaltungen in einen Baustein zu integrieren. Die Integration der Schaltungen erfolgt im Allgemeinen durch den Anwender selbst; er kann funktionspezifische und auf die jeweilige Aufgabe optimal zugeschnittene ICs erstellen. Durch die Nutzung von PLDs ergeben sich gegenüber einer Schaltungsrealisierung mit diskreten Standardbauelementen verschiedene Vorteile:

1. Die Entwicklung der Schaltung erfolgt über einen Rechner – beispielsweise einen PC – mit Hilfe spezieller Programmpakete, sogenannter Entwicklungsumgebungen. Das Design ist daher schnell zu entwickeln und leicht veränderbar.
2. Die Entwicklungsumgebungen bieten die Option, die logische Funktion der Schaltung zu verifizieren.
3. Durch Simulatoren können die internen Laufzeiten der einzelnen Signale überprüft und dadurch beispielsweise Timing-Fehler aufgedeckt werden.
4. Die fertige Schaltung wird kompakter.

Aus diesen Gründen bietet sich der Einsatz von PLDs zunächst im Entwicklungsstadium einer Schaltung an. Darüber hinaus können sie auch bei der anschließenden Produktion in kleinen bis mittleren Stückzahlen effektiv eingesetzt werden. Insbesondere für die Fertigung kleiner Stückzahlen sind als „Low-Cost-Versionen“ nichtlöschbare Varianten verfügbar. Bei einem mittleren Stückzahlenbedarf werden im Allgemeinen maskenprogrammierte PLDs (MPLDs) verwendet. Ihre Programmierung erfolgt mit dem vom Anwender entwickelten und getesteten Design durch den Hersteller. Die Eigenschaften des entstehenden Bausteines sind dadurch bekannt. Für die Massenfertigung empfiehlt sich aus Kostengründen die Verwendung von kundenspezifischen ICs (ASIC – Custom Application-Specific Integrated Circuit). Der Aufwand für die Entwicklung eines ASICs ist allerdings sehr viel höher als der Entwicklungsaufwand eines PLDs. Außerdem ist die Zeitdauer bis zur Fertigstellung der ICs größer.

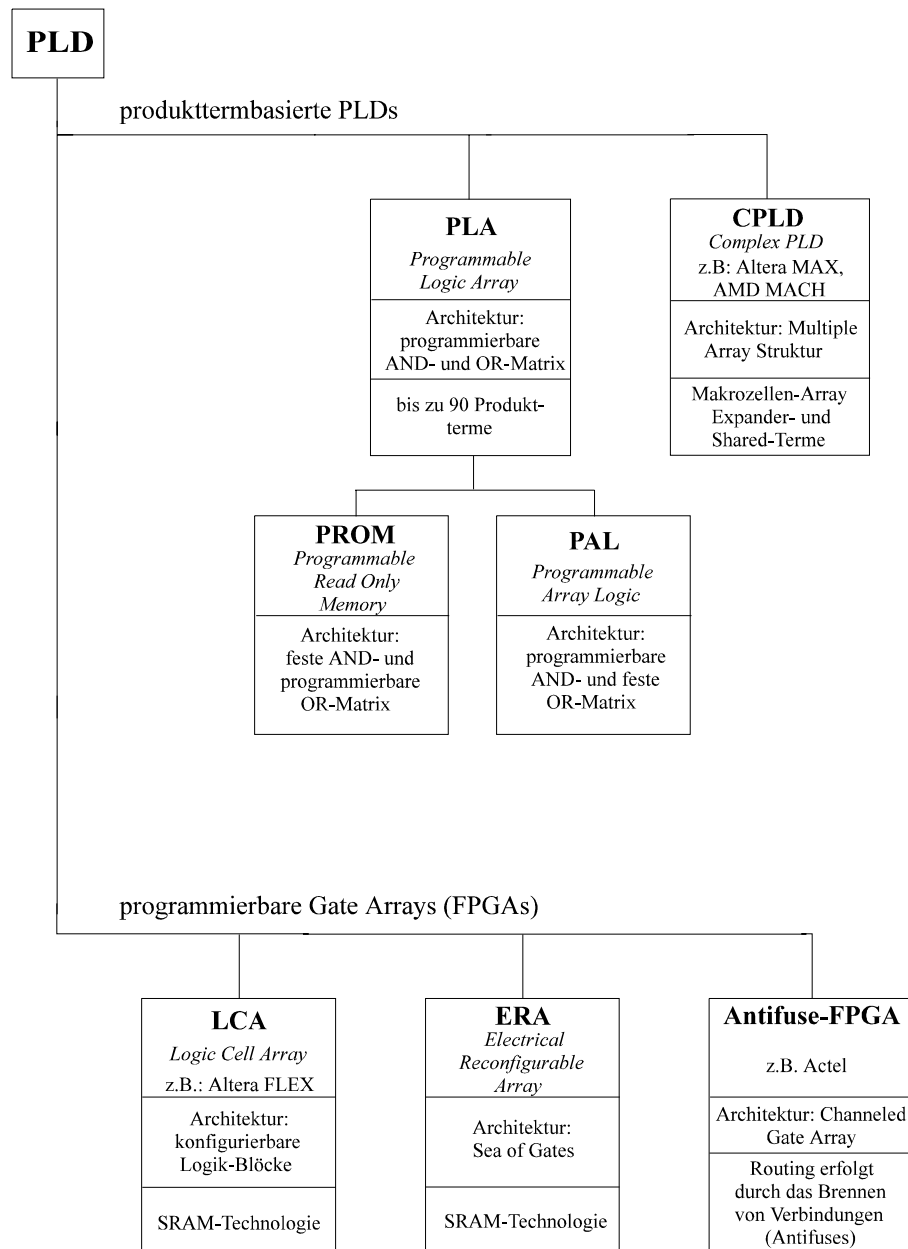


Bild 1.1: Übersicht programmierbarer Logikbausteine

In diesem Versuch soll ein komplexer PLD programmiert werden. Zur Vorbereitung wird zunächst ein allgemeiner Überblick über das derzeitige Angebot an PLDs gegeben und der im Versuch eingesetzte Baustein mit der Bezeichnung EPM 7032 des Herstellers Altera vorgestellt. Anschließend erfolgt eine Beschreibung der verwendeten Entwicklungsumgebung MAX+plus II von Altera. Danach wird die zu bearbeitende Aufgabe dargestellt. Nach den Anweisungen zur Versuchsdurchführung folgt eine Zusammenfassung der Lernziele dieses Versuches. Als weitere Hilfsmittel zur Versuchsvorbereitung dienen ein Literatur- und ein Abkürzungsverzeichnis.

2. Programmierbare Logikbausteine (PLDs) – Überblick

PLDs lassen sich zunächst in zwei Gruppen einteilen (Bild 1.1):

1. Produkttermbasierte PLDs und
2. Programmierbare Gate Arrays.

2.1 Produkttermbasierte PLDs

Bei einem Produktterm handelt es sich um die konjunktive Verknüpfung von n Eingangsvariablen (z.B. $a = e_1 \wedge e_2 \wedge \dots \wedge e_n$). Diese Produktterme können bei produkttermbasierten PLDs disjunktiv miteinander verknüpft werden (vgl. Bild 2.1). Bezüglich der Programmierbarkeit der die Produktterme repräsentierenden AND-Matrix bzw. der ausgangsseitigen OR-Matrix lässt sich eine weitere Unterteilung vornehmen (vgl. Bild 1.1, 2.1). Komplexere PLDs (CPLDs) dieser Art (Altera MAX, AMD MACH u.a.) besitzen mehrere solcher Matrizen, welche untereinander verbunden werden können. Dadurch lassen sich auch kompliziertere logische Funktionen in einem Baustein realisieren. Als Beispiel wird im Folgenden die Altera-CPLD-Familie MAX 7000 (Multiple Array Matrix) vorgestellt.

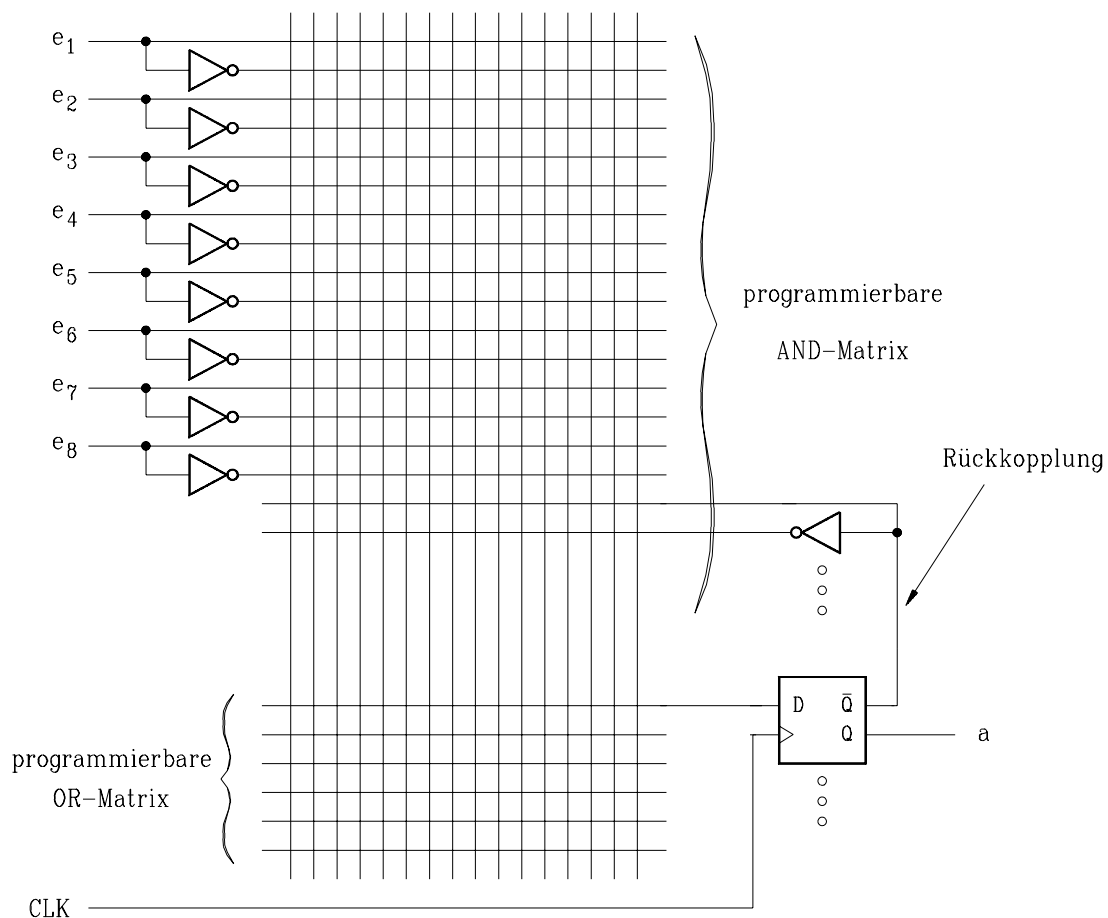


Bild 2.1: Struktur eines PLAs

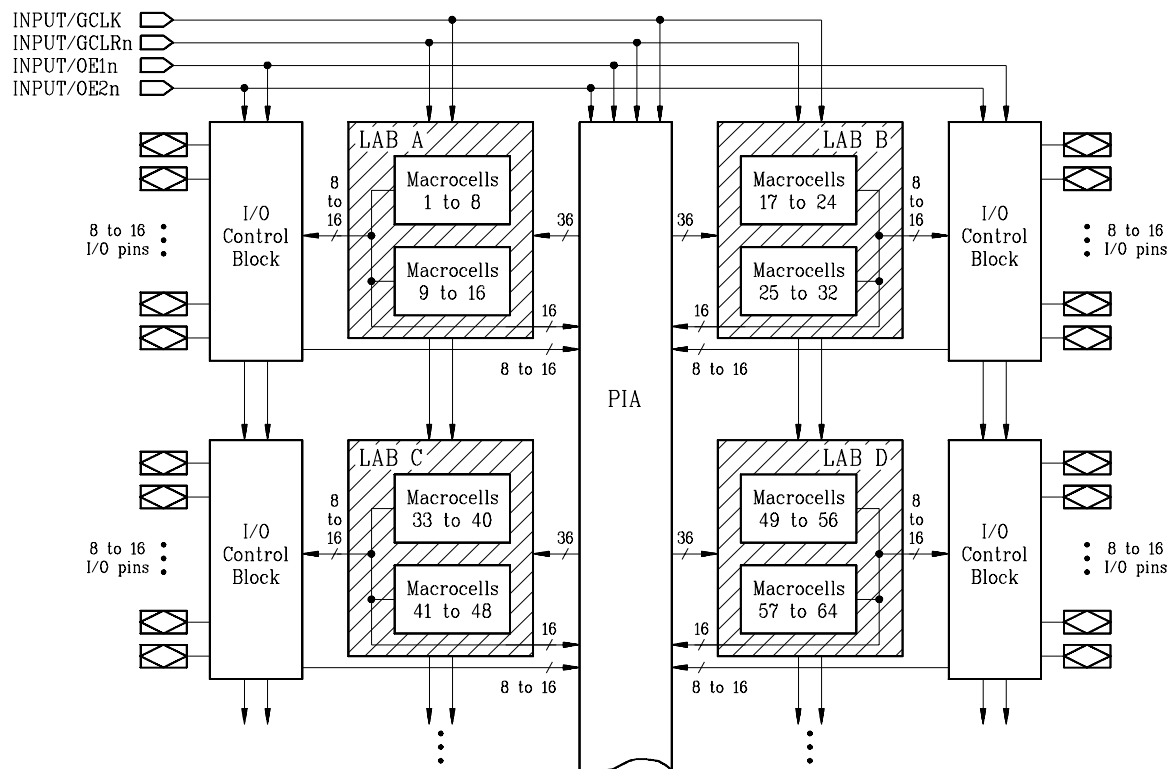


Bild 2.2: Architektur der Bausteine EPM7032, EPM 7064 und EPM7096

Das Basiselement der Architektur der 7000er PLDs ist die sogenannte Makrozelle. Die auf einem IC verfügbaren Makrozellen werden in Gruppen zu 16 Stück, den LABs (Logic Array Block), zusammengefasst. Die Kommunikation der einzelnen LABs untereinander übernimmt das PIA (Programmable Interconnect Array), während die Verbindungen nach außen mit Hilfe spezieller Input/Output Control-Blöcke hergestellt werden (siehe Bild 2.2).

Der Aufbau der Makrozelle des EPM 7032 ist in Bild 2.3 dargestellt. Hauptbestandteil ist das für die Bildung der Produktterme zuständige Logic Array. Es erhält zwei Gruppen von Eingangssignalen: Einerseits werden die Programmable Interconnect Signals zugeführt, welche dem PIA entstammen (vgl. Bild 2.2). Bei der zweiten Gruppe handelt es sich um sogenannte Expander Product Terms, die die Makrozellen innerhalb eines LABs verbinden können und somit erweiterte Verknüpfungsmöglichkeiten schaffen. Man unterscheidet zwischen shareable und parallelen Expandern. Jede Makrozelle stellt einen shareable Expander zur Verfügung, dessen invertierter Ausgang nicht nur in die eigene Matrix zurückgekoppelt, sondern auch in jeder der 15 anderen Makrozellen dieses LABs verwendet werden kann. Außerdem besteht die Möglichkeit, Produktterme, die innerhalb einer Makrozelle nicht genutzt werden, einer benachbarten Makrozelle zuzuteilen. Derartige Produktterme werden als parallele Expander bezeichnet. Insgesamt kann eine Makrozelle maximal 20 Produktterme verwalten, die sich aus den fünf eigenen und 15 von den Nachbarzellen geliehen zusammensetzen.

Unter Hinzunahme der Product-Term Select Matrix mit nachfolgendem OR-Gatter lässt sich eine solche Makrozelle bis zu dieser Stelle als PLA betrachten. Das daran anschließende XOR-Gatter kann beispielsweise zur Invertierung verwendet werden.

Besondere Beachtung verdient jedoch das programmierbare Register. Es kann als D, W (engl.: T, für Toggle), JK oder SR-Flip-Flop geschaltet und wahlweise global getaktet oder mit einem durch logische Verknüpfungen in der Logic Array Matrix generierten Takt versorgt werden. Die weiteren Konfigurationsmöglichkeiten (Register Bypass, Enable Select, Clear Select) sind aus Bild 2.3 direkt ersichtlich.

Der Aufbau des PIAs basiert ebenfalls auf einer Matrix, innerhalb der die Signale aller I/O-Pins sowie die Ausgänge der Makrozellen zur Verfügung stehen. Durch ein programmierbares AND- und ein festes OR-Gatter wird festgelegt, ob jeweils eines dieser Signale an einen Eingang eines LABs gelangt. Durch dieses Verbindungssystem entsteht der Vorteil einer konstanten Durchlaufzeit. Jedes Signal besitzt die gleiche Verzögerung (vgl. Abschnitt 2.2).

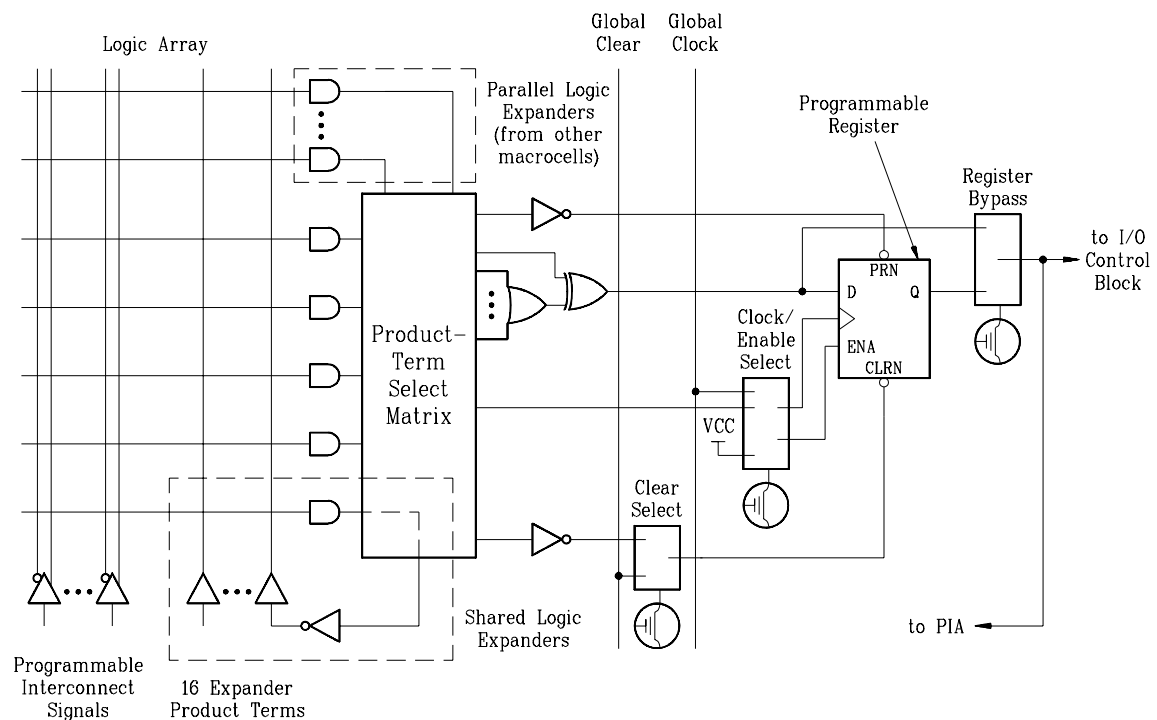


Bild 2.3: Aufbau der Makrozelle des EPM7032

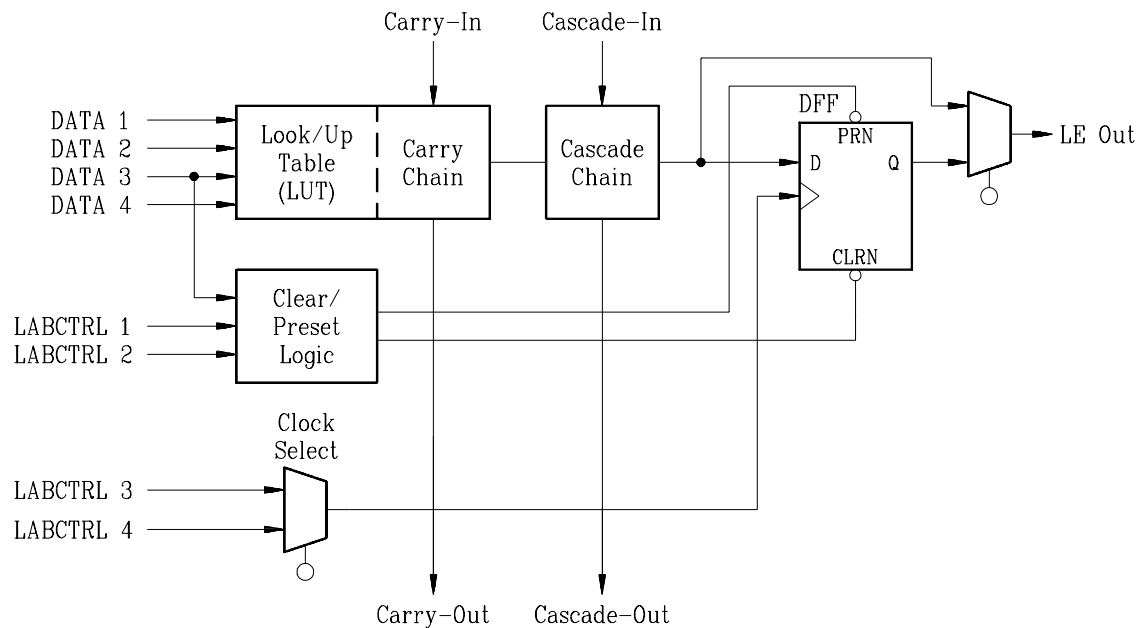


Bild 2.4: Aufbau des Logic Elements der FLEX-8000-Reihe

Die MAX-7000-PLDs werden in EEPROM-basierter CMOS-Technologie hergestellt, folglich sind sie elektrisch löschtbar. Zurzeit sind Bausteine mit 32 (EPM 7032) bis 256 (EPM 7256E) Makrozellen bzw. 600 bis 5000 nutzbaren Gattern erhältlich. Der größte Baustein bietet 164 für den Anwender verwendbare I/O-Pins. Ein in den schnellsten PLD der MAX-7000-Reihe, dem EPM 7032, integrierter 16-Bit-Zähler kann mit einer maximalen Taktfrequenz von 178,6 MHz betrieben werden [4].

2.2 Programmierbare Gate Arrays

Ein programmierbares Gate Array (FPGA – Field Programmable Gate Array) besteht aus elementaren Logikblöcken, welche untereinander durch sogenannte Routing-Channels verbunden sind. Die Unterschiede der einzelnen Typen liegen in dem Aufbau bzw. der Komplexität der Logikblöcke sowie in der Art der Verbindungssysteme. Einige Beispiele befinden sich im Bild 1.1. Die Bezeichnung „programmierbares Gate Array“ wird im Allgemeinen für Bausteine verwendet, die durch den Anwender programmierbar sind.

Als aktuelles Beispiel für programmierbare Gate Arrays mit relativ komplexen Logikblöcken dient die nachfolgend beschriebene FLEX-8000-Familie von Altera. An die Stelle der Makrozelle als kleinster Einheit tritt bei den FLEX-Bausteinen das Logic Element (LE). Sein Aufbau, der in Bild 2.4 dargestellt ist, unterscheidet sich wesentlich von dem der Makrozelle. Insbesondere sind das Array, das die Produktterme repräsentiert und die nachgeschalteten disjunktiven Verknüpfungen durch ein Look-Up Table (LUT) ersetzt worden, welches vier Eingänge besitzt. Mit dem LUT lässt sich jede Funktion aus vier Eingangsvariablen realisieren.

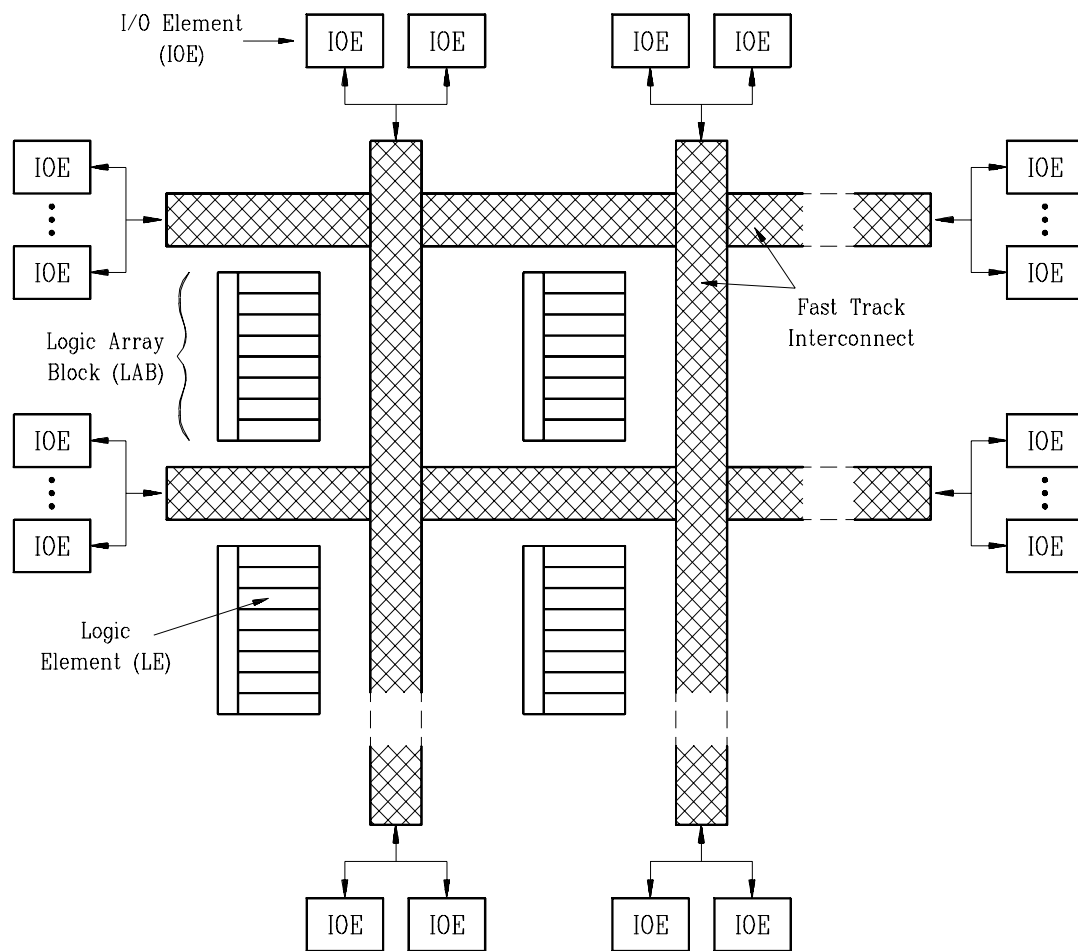


Bild 2.5: Architektur der FLEX-8000-Bausteine

Eine direkte und damit schnelle Verbindung zu benachbarten LEs lässt sich mit Hilfe der Carry Chain und der Cascade Chain herstellen. Diese Verbindungsmöglichkeiten finden insbesondere bei der Implementierung von Zählern und Addierern Verwendung. Sie können als Äquivalent zu den Expander-Product-Termen der MAX-7000-Makrozellen aufgefasst werden. Eine intensive Nutzung der beiden Chain-Leitungen schränkt die Implementierungsmöglichkeiten der weiteren Logik stark ein, wodurch die zur Verfügung stehenden Ressourcen schlecht ausgenutzt werden können. Auch die LEs werden wieder zu übergeordneten LABs zusammengefasst. Es bilden jeweils acht LEs einen LAB (vgl. Bild 2.5). Auffällig ist allerdings die Anordnung der LABs in Zeilen und Spalten, worauf auch die Verbindung der LABs untereinander bzw. zu den Ein- und Ausgängen mittels des Fast Track Interconnect basiert. Hieraus entsteht der entscheidende Nachteil der FPGAs gegenüber den produkttermbasierten PLDs. Die in dem Baustein entstehenden Verzögerungen der verarbeiteten Signale sind von der jeweiligen Lage der miteinander verknüpften LABs bzw. dem gewählten Verbindungsweg abhängig und damit schwer vorhersagbar. (Zum Vergleich sei noch einmal an die Architektur der MAX-PLDs erinnert, bei der jedem LAB ein eigener I/O-Control Block zur Verfügung steht!). Die Vorteile der FPGAs liegen in der hohen Integration. Die FLEX-8000-Bausteine stellen z.Zt. maximal 16.000 nutzbare Gatter, 1.500 Register und 208 I/O-Pins zur

Verfügung. Die maximale Taktfrequenz für einen integrierten 16-Bit-Auf- bzw. Abwärtszähler beträgt bei dem schnellsten Baustein dieser Reihe 125 MHz.

Die PLDs der FLEX-Reihe sind ebenfalls elektrisch löschbar. Allerdings werden sie – wie auch die Bausteine der dritten Generation der MAX-PLDs (MAX 9000) – direkt in ihrer Schaltungsumgebung programmiert, was in der Entwicklungsphase eine zusätzliche Zeitersparnis ermöglicht. Da die FLEX-8000-PLDs auf SRAM-Technologie basieren, müssen sie nach jedem Einschalten neu konfiguriert werden.

Der Vollständigkeit halber soll auch die von dem Hersteller Actel eingesetzte Antifuse-Technik kurz vorgestellt werden. Die bei der Programmierung eines auf Antifuses basierenden PLDs zu erzeugenden Verbindungen werden „gebrannt“. Dieser Vorgang ist irreversibel – die PLDs können nicht mehr gelöscht werden, was in der Entwicklungsphase einer Schaltung nachteilig ist. Allerdings weisen sie in einer durch starke Störstrahlung beeinträchtigten Umgebung eine höhere Betriebssicherheit auf. Deshalb finden sie beispielsweise bei Raumfahrt-Projekten bevorzugt Verwendung [1; 2; 3; 4].

3. Die Entwicklungsumgebung MAX + plus II

Die von Altera für die MAX- und FLEX-Reihen angebotene Entwicklungsumgebung MAX+plus II läuft in der PC-Version unter Microsoft Windows (3.1, 3.11, Win95, Win98 bzw. NT) [5]. Die Hardware muss folgende Mindestanforderungen erfüllen:

- Prozessor: 486 oder Pentium
- RAM-Speicher: 16 MByte
- Freier Platz auf der Festplatte: 100 MByte.

3.1 Eingabe der Schaltung

MAX+plus II bietet verschiedene Möglichkeiten der Schaltungseingabe (Design Entries):

1. Die direkte Eingabe des Schaltplanes mit dem Graphic-Editor

MAX+plus II ermöglicht mit dem integrierten Graphic-Editor die direkte Eingabe des Schaltplanes der entworfenen Schaltung. Dazu stehen zunächst logische Symbole wie Gatter und Inverter zur Verfügung. Zusätzlich beinhaltet MAX+plus II vorgefertigte Makrofunktionen, beispielsweise für die Implementierung von Zählern und Addierern. Außerdem existiert eine Reihe von Makrofunktionen für die Funktionsbeschreibung von Bausteinen diskreter Logik. Somit können z.B. aus TTL-Bausteinen bestehende Schaltungen ohne Änderungen eingegeben werden.

Die mit dem Graphic-Editor erstellte Schaltung lässt sich in zwei Formaten speichern: Für die interne Verwendung der Datei empfiehlt sich die Speicherung als Graphic Design File (Dateiendung: „.gdf“). MAX+plus II bietet aber auch die Möglichkeit, die Datei in dem von dem CAD-Programm Or-

CAD verwendeten Format zu speichern, also als OrCAD Schematic File (Endung: „.sch“). Ebenso können mit OrCAD erstellte Dateien gelesen werden, so dass sowohl ein Import als auch ein Export möglich ist.

2. Die Beschreibung der Funktion der Schaltung mit Hilfe spezieller Hardware-Programmiersprachen

Bei der Implementierung von bestimmten Schaltungen, beispielsweise reinen Schaltnetzen, kann die Eingabe der Schaltung auf Gatterebene (mit dem Graphic Editor) recht zeitaufwendig werden. Hier bietet sich eine direkte Eingabe der logischen Funktion, also die Abhängigkeit der Ausgangssignale von den Eingangssignalen an.

Zur Hardware-Beschreibung hat sich die Sprache VHDL (Very High Speed Integrated Circuit Hardware Description Language) etabliert, welche auch von MAX+plus II unterstützt wird. VHDL wurde 1987 in der IEEE-Norm 1076-1987 festgelegt. Diese technologieunabhängige Sprache dient aber nicht nur zur Beschreibung, sondern auch zur Simulation von Schaltungen, wodurch eine spezielle Simulationssprache überflüssig wird. Darüber hinaus kann in MAX+plus II auch die Sprache AHDL (Altera Hardware Description Language) verwendet werden. Für die Programmierung mit Hilfe dieser beiden Sprachen stehen ebenfalls umfangreiche Makrofunktionen zur Verfügung.

Zur Eingabe der Programmtexte beinhaltet MAX+plus II einen Text-Editor. Es kann jedoch auch jeder andere ASCII-Editor genutzt werden. Die erzeugten Dateien haben bei der Verwendung von AHDL die Endung „.tdf“ (Text Design File). Auf VHDL basierende Dateien sind an der Endung „.vhd“ zu erkennen.

3. Die Eingabe der gewünschten Signalverläufe mit dem Waveform-Editor

Die dritte Möglichkeit, eine Schaltung zu programmieren besteht darin, die gewünschten Verläufe der Ausgangssignale mit dem Waveform-Editor einzugeben. Hierauf wird im Rahmen dieses Versuches jedoch nicht näher eingegangen.

3.2 Aufteilung des Projektes in einzelne Module

Um den Schaltungsentwurf übersichtlich zu gestalten sowie die eventuelle Fehlersuche zu vereinfachen, kann das gesamte Projekt hierarchisch in einzelne Module, sogenannte „Subdesigns“, unterteilt werden, wobei jedes Modul wiederum aus mehreren Teilen bestehen kann. Für jedes dieser Module kann der Entwickler die Form der Schaltungseingabe aus den eben vorgestellten Möglichkeiten separat wählen.

Die Verknüpfung der einzelnen Subdesigns sowie die Festlegung der Ein- und Ausgänge der Schaltung erfolgen in einer speziellen Top-Level-Datei, die die Spitze der Hierarchie bildet. Die Top-Level-Datei kann sowohl mit dem Graphic Editor in Form eines Schaltbildes als auch mit dem Text-Editor in AHDL oder VHDL erzeugt werden.

3.3 Kompilation des Projektes

Die Umsetzung der Schaltungseingaben in die für den Programmiervorgang erforderlichen Befehle erfolgt mit dem in MAX+plus II integrierten Compiler. In manchen Fällen mag es erforderlich sein, der Entwicklungsumgebung bestimmte Auflagen für die Kompilation mitzuteilen. Insbesondere die Pinbelegung des fertigen PLDs ist hier als Beispiel zu nennen. Des Weiteren kann auch der PLD-Typ vorgegeben werden; zwingend ist eine solche Festlegung jedoch nicht. Vorgaben des Entwicklers werden mit dem Floorplan Editor erstellt, welcher sie in einer Datei mit der Endung „.acf“ (Assignment & Configuration File) speichert. Nach diesen Vorbereitungen kann das Projekt kompiliert und in den ausgewählten Baustein „eingepasst“ werden. Während der Kompilation werden u.a. die folgenden Dateien erzeugt:

- **Report File („.rpt“):** Es handelt sich hierbei um eine ASCII-Datei. Ihr ist zu entnehmen, wie das Projekt in den PLD eingepasst wurde (Pinbelegung, Nutzung und Verbindung der einzelnen LABs usw.).
- **Simulator Netlist File („.snf“):** Diese Datei ist für die spätere Simulation erforderlich.
- **Programmer Object File („.pof“):** Hier werden die Daten für die Programmierung des PLDs gespeichert.

3.4 Simulation

Die Bearbeitung des Projektes für den eigentlichen Programmiervorgang des PLDs ist mit der Kompilation prinzipiell abgeschlossen. Um jedoch eventuell vorhandene Fehler schon vor der Programmierung aufzuspüren, sollte die Schaltung zuerst simuliert werden. Die Simulation kann nur nach der Kompilation erfolgen, da erst dann die in dem speziellen Baustein entstehenden Verzögerungen der einzelnen Signale bekannt sind. Für die Simulation der Schaltung ist zunächst eine adäquate Testsequenz – bestehend aus charakteristischen Eingangs- und gewünschten Ausgangssignalen – zu entwerfen. Der zeitliche Verlauf der Eingangssignale wird MAX+plus II vorgegeben. Die Eingabe erfolgt mit dem Waveform Editor und wird in der Simulator-Channel-Datei (Endung: „.scf“) gespeichert. Daraufhin kann der Simulator gestartet werden, der die zu diesen Eingangssignalen gehörenden Ausgangssignale des PLDs berechnet. Die Simulationsergebnisse werden ebenfalls in der Simulator-Channel-Datei abgelegt und können anschließend parallel zu den Eingangssignalen mit dem Waveform Editor betrachtet werden. Dem Entwickler bietet sich nun die Möglichkeit, die Simulationsergebnisse mit seinen erwarteten Signalverläufen zu vergleichen. Abschließend wird der PLD mit dem an den PC angeschlossenen Programmiergerät programmiert.

4. Das Projekt: Synchronisation eines DVB-Empfängers

Das europäische DVB-Projekt (Digital Video Broadcasting Project) [7] wurde gegründet, um Standards zu erarbeiten, die eine einheitliche Übertragung von digitalen Fernseh- und Datensignalen über verschiedene Übertragungsstrecken

wie Satellit, Kabel oder den terrestrischen Funkkanal garantieren. Die Normung der verschiedenen Standards erfolgt durch ETSI (European Telecommunications Standards Institute). Während die Signalverarbeitung bei jedem Standard verschieden und auf das jeweilige Übertragungsmedium angepasst ist, ist die Struktur der senderseitigen Eingangs- bzw. der empfängerseitigen Ausgangsschnittstelle bei allen Übertragungsstandards identisch.

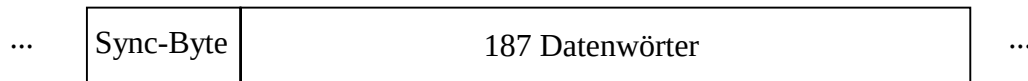


Bild 4.1: Die Paketstruktur des DVB-Datenstromes

Die DVB-Datenströme bestehen an dieser Stelle aus Paketen zu jeweils 188 Bytes (Bild 4.1). Der Beginn eines solchen Paketes wird durch ein Sync-Byte gekennzeichnet, dem 187 Daten-Bytes folgen. Zur Decodierung des Datenstroms müssen die einzelnen Pakete erkannt und gekennzeichnet werden. Hierzu dient neben der Kenntnis des Synchronwortes die redundante Information der konstanten Paketlänge, die einen festen zeitlichen Abstand der einzelnen Sync-Bytes nach sich zieht.

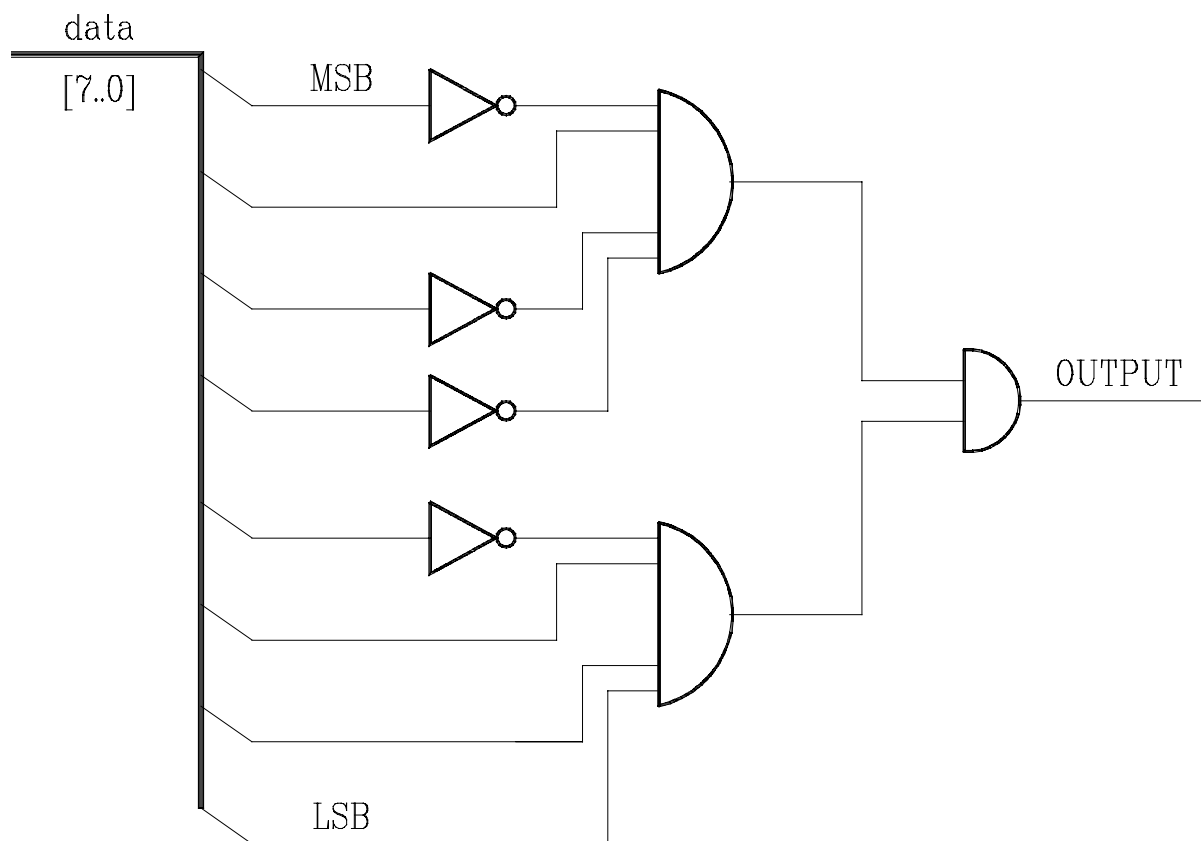


Bild 4.2: Schaltnetz zur Detektion der Sync-Bytes

In diesem Versuch soll in einem CPLD eine Schaltung entwickelt werden, die die wichtige Sync-Byte-Erkennung und die anschließende Kennzeichnung der einzelnen Pakete durchführt. Zur Detektion eines Datenwortes, welches den Wert eines Sync-Bytes besitzt, kann beispielsweise das in Bild 4.2 gezeigte Schaltnetz dienen. Das parallel eingehende Daten-Byte wird mit dem bekannt-

ten Synchronwort verglichen. Sind beide Wörter identisch, so ist der Ausgang der Schaltung für eine Taktperiode auf logisch HIGH, anderenfalls auf logisch LOW. Dieses Ausgangssignal wird künftig mit RSYNC (received sync) bezeichnet.

Hausaufgabe 1: Aus der in Bild 4.2 dargestellten Schaltung ist das Synchronwort zu ermitteln und in binärer, dezimaler sowie hexadezimaler Schreibweise anzugeben.

Die auf diese Art detektierten Datenwörter entsprechen nicht unbedingt den tatsächlichen Sync-Bytes und kennzeichnen damit nicht die wirklichen Paketanfänge. Einerseits kann ein Datenwort, das sich innerhalb des Paketes befindet, durchaus den gleichen Wert wie das Sync-Byte haben. Der Detektor erkennt ein derartiges Datenwort fälschlicherweise als Paketbeginn. Andererseits kann ein Sync-Byte durch Störungen auf der Übertragungsstrecke verfälscht werden. In diesem Fall wird der Paketanfang durch den Detektor nicht mehr identifiziert. Die bloße Detektion der Synchronwörter ist also notwendig, aber nicht hinreichend. Vielmehr muss der bekannte konstante Abstand der tatsächlichen Sync-Bytes ebenfalls zur Ermittlung der Paketanfänge herangezogen werden.

5. Die Realisierung

5.1 Der Sync-Zähler

Eine Lösung zur Nutzung der gegebenen Redundanz besteht darin, zunächst einen Zähler, im Folgenden als Sync-Zähler bezeichnet, so zu beschalten, dass er Impulse der Länge einer Taktperiode im erforderlichen Abstand von 188 Takten erzeugt. Dieses Ausgangssignal des Sync-Zählers muss anschließend auf den eingangsseitigen Datenstrom synchronisiert werden. Hierzu ist der Sync-Zähler mit dem Eintreffen eines Sync-Bytes zu starten.

5.2 Der Prü fzähler

Da von keinem detektierten Datenwort bekannt ist, ob es sich tatsächlich um einen Paketanfang handelt, wird der Sync-Zähler zunächst willkürlich durch ein eintreffendes Datenwort gestartet, welches den Wert des Sync-Bytes besitzt. Es ist daraufhin noch zu überprüfen, ob dieses Datenwort wirklich einen Paketbeginn repräsentiert. Dieses geschieht durch ständigen Vergleich der im Sync-Zähler erzeugten Sync-Impulse, künftig auch als SYNC bezeichnet, mit den Ausgangssignalen des Detektors (RSYNC), der – wie bereits erwähnt – alle Datenwörter detektiert, die mit dem Synchronwort identisch sind. Fällt die Mehrzahl der Vergleiche positiv aus, so ist die Wahrscheinlichkeit groß, dass der Sync-Zähler tatsächlich zu Beginn eines Paketes gestartet wurde. Die Synchronisation des Sync-Zählers übernimmt ein zweiter Zähler, der sogenannte Prü fzähler. Es handelt sich dabei um einen Auf- und Abwärtszähler, der bei Übereinstimmung von SYNC und RSYNC aufwärts zählt. Bei falsch erzeugten Synchronimpulsen (SYNC = HIGH, aber RSYNC = LOW) zählt er hingegen abwärts. Erreicht der Zähler seinen Maximalwert von 15_{DEZ}, so ist die Synchronbedingung erfüllt und die Schaltung synchronisiert.

Entsprechend wird die Ausgabe seines Minimalwertes als Signal dafür verwendet, dass der Sync-Zähler nicht bei einem Paketanfang, sondern bei einem Datenwort, dessen Wert gleich dem des Sync-Bytes war, zu zählen begann; in diesem Falle wird ein neuer Synchronisationsversuch eingeleitet.

5.3 Die Steuerung der Zähler

Zur Implementierung der beiden Zähler werden die von MAX+plus II angebotenen Makrofunktionen 4COUNT (binärer 4-Bit-Auf-/Abwärtszähler) und 8COUNT eingesetzt. Die Funktionsbeschreibungen der verschiedenen Baugruppen können über das „Help“-Menü aufgerufen werden. Es erscheinen z.B. bei den beiden verwendeten Makrofunktionen die in den Bildern 5.1a und 5.1b dargestellten Fenster.

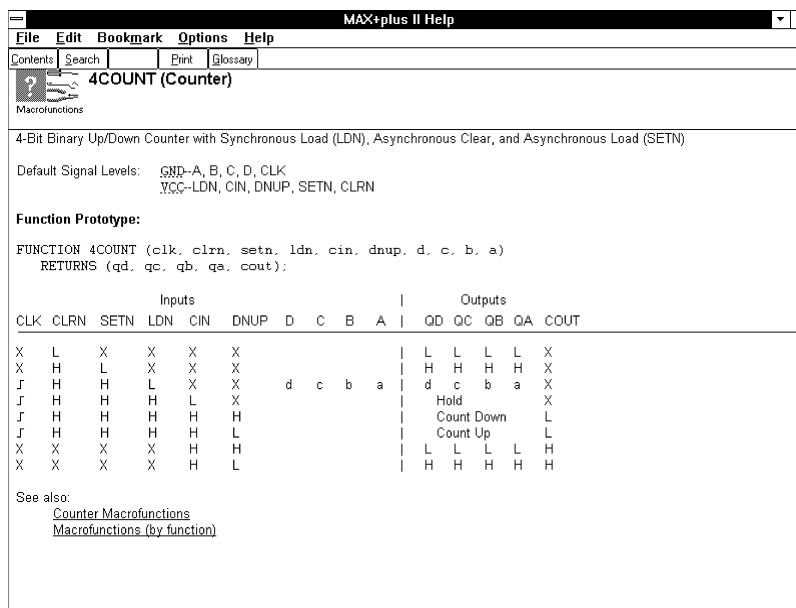


Bild 5.1a: Die Makrofunktion 4COUNT

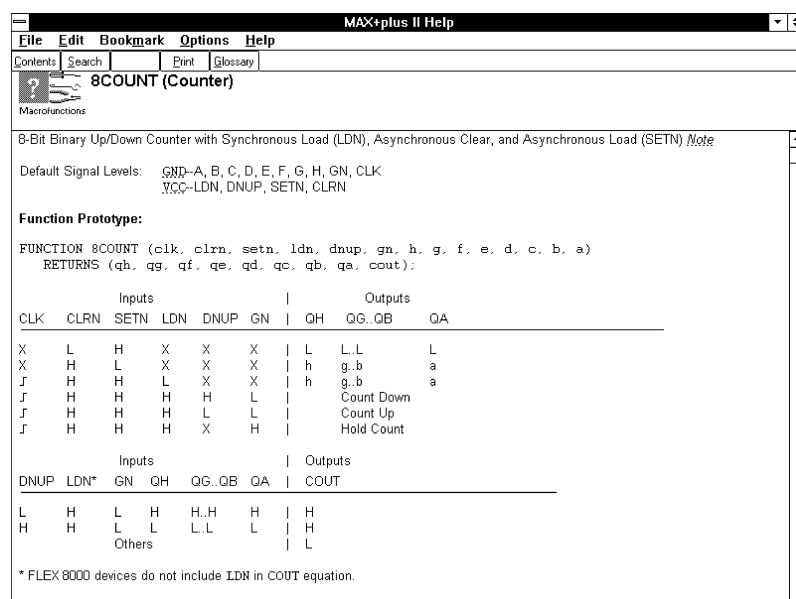


Bild 5.1b: Die Makrofunktion 8COUNT

Zunächst ist insbesondere die positive Flankensteuerung beider Zähler zu beachten, was bedeutet, dass Steuersignale wie DNUP, GN und CIN während der steigenden Taktflanken ihre jeweiligen Sollwerte besitzen müssen. Die Eingänge CIN (4COUNT) und GN (8COUNT) lassen sich zur Auswertung von Enable-Signalen verwenden.

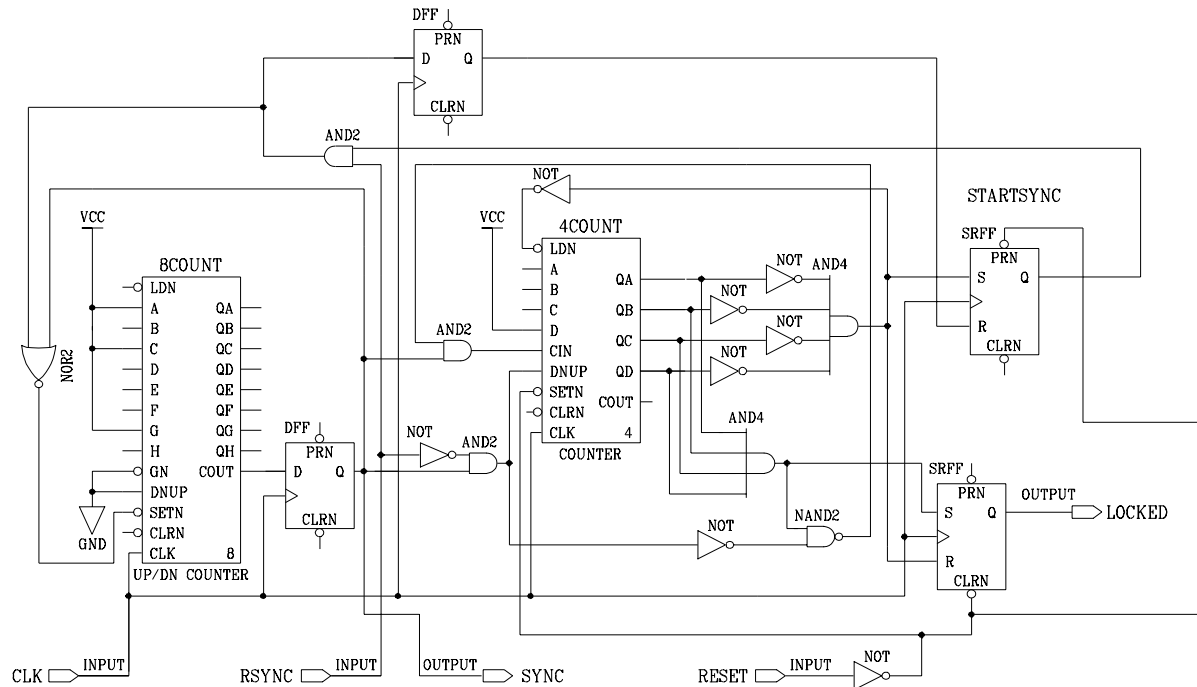


Bild 5.2: Das Modul „MAIN“ des Projektes

In Bild 5.2 ist das Schaltbild des Hauptmoduls („MAIN“) des Projektes dargestellt, es zeigt die Beschaltung der beiden Zähler. Als Takt dient das vom DVB-Testsignalgenerator mitgelieferte CLK-Signal. Der als Sync-Zähler eingesetzte 8-Bit-Zähler zählt ständig aufwärts (also DNUP = GN = LOW). Zur Erzeugung der Synchronimpulse wird der Carry-Bit-Ausgang verwendet. Das diesem Ausgang folgende D-Flip-Flop verzögert das Ausgangssignal des Zählers um eine Taktperiode. Gleichzeitig wird der Zähler auf den Wert 69_{DEZ} geladen – die Eingänge H-A sind entsprechend beschaltet – und eine neue Zählenschleife beginnt.

Der Prü fzähler wird so gesteuert, dass er nach dem Erreichen seines Maximalwertes nicht mehr aufwärts, sondern – im Fall von fehlerhaft erzeugten Synchronimpulsen – nur noch abwärts zählen kann. Ein weiteres Aufwärtszählen hätte das Überlaufen des Zählers zur Folge. Ist der Ausgang des Zählers gleich Null, wird zunächst das zur Erzeugung des LOCKED-Signales dienende Flip-Flop gelöscht. Die Schaltung ist nun nicht mehr synchronisiert. Zum Starten eines neuen Synchronisationsvorganges wird das STARTSYNC-Flip-Flop gesetzt. Ein Neustart des Sync-Zählers ist nur möglich, wenn der Ausgang dieses Flip-Flops auf logisch HIGH liegt. Außerdem wird der Prü fzähler auf den Wert 8_{DEZ} geladen. Beim nächsten eintreffenden RSYNC-Impuls wird der Sync-Zähler gestartet und das STARTSYNC-Flip-Flop zur Verhinderung weiterer Neustarts – um eine Taktperiode verzögert – gelöscht. Die nun von dem Sync-Zähler erzeugten Sync-Impulse veranlassen den Prü fzähler in Abhängigkeit

vom logischen Pegel des RSYNC-Signales entweder zum Auf- oder Abwärtszählen. Folglich erreicht er praktisch in einem endlichen Zeitraum seinen Maximal- oder Minimalwert.

5.4 Der Sync-Byte-Detektor

Wie bereits in dem Abschnitt 3.1 angedeutet wurde, bietet sich zur Realisierung des Sync-Byte-Detektors die Programmierung mit Hilfe einer Hardware-Programmiersprache an. Zum Einsatz kommt in diesem Versuch die Sprache AHDL. Das für den Sync-Byte-Detektor zu verwendende Modul ist im Folgenden wiedergegeben:

```
subdesign syncdet
(
    data[7..0] :INPUT;
    rsync      :OUTPUT;
)
BEGIN
    rsync = (Verknüpfung);
END;
```

Nach der Festlegung der Ein- und Ausgänge des Moduls (vgl. Bild 3.2) folgt die Beschreibung der eigentlichen Funktion – beispielsweise wie hier mittels einer Bool'schen Gleichung.

***Hausaufgabe 2:** Die Bool'sche Gleichung für das den Sync-Byte-Detektor repräsentierende Modul ist unter Verwendung des Ergebnisses der ersten Hausaufgabe sowie der gegebenen Ein- und Ausgänge zu vervollständigen. Der Operator für den Vergleich zweier gleich großer Datenwörter auf Äquivalenz lautet „=“.*

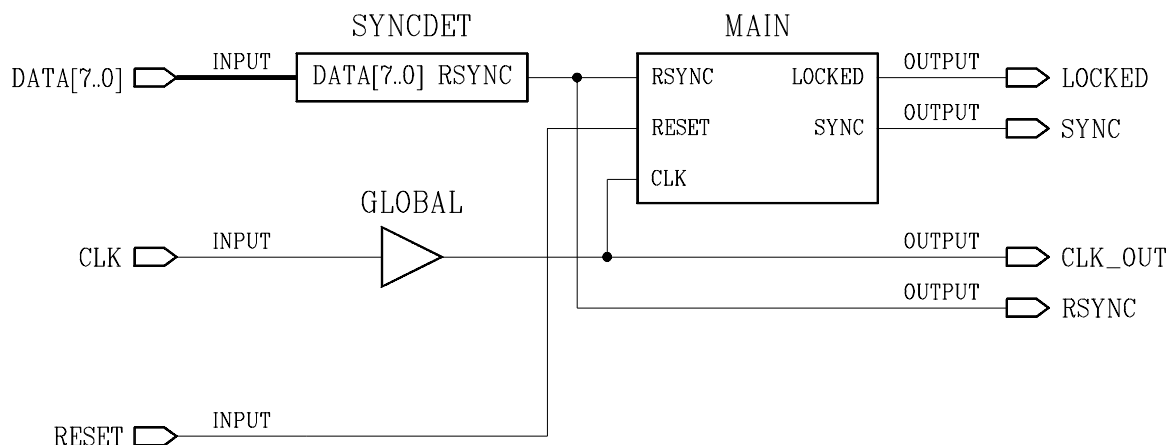


Bild 5.3: Das Schaltbild der Top-Level-Datei

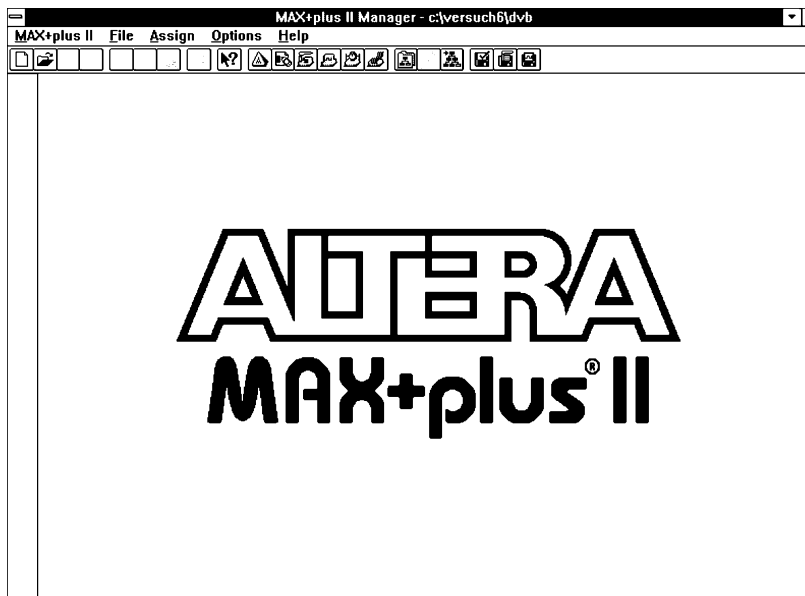


Bild 5.4: MAX+plus II

In Bild 5.3 sind schließlich die Verschaltungen der beiden Module untereinander sowie mit den Ein- und Ausgängen des gesamten Projektes dargestellt. Dieses Schaltbild ist in der Top-Level-Datei „dvb.gdf“ gespeichert.

6. Die Versuchsdurchführung

6.1 Vorbereitungen

Nach dem Einschalten des Rechners ist zunächst Microsoft Windows zu starten. Im erscheinenden Programm-Manager wird das Programm MAX+plus II durch einen doppelten Klick mit der linken Maustaste (LMT) auf dem entsprechenden Icon aufgerufen. Das nun geöffnete Fenster ist im Bild 5.4 dargestellt. Es ist zunächst zu überprüfen, ob in der obersten Leiste des Fensters, der Titelleiste, der Projektname „versuch6\dvb“ erscheint. Ist dieses nicht der Fall, so sind folgende Schritte durchzuführen:

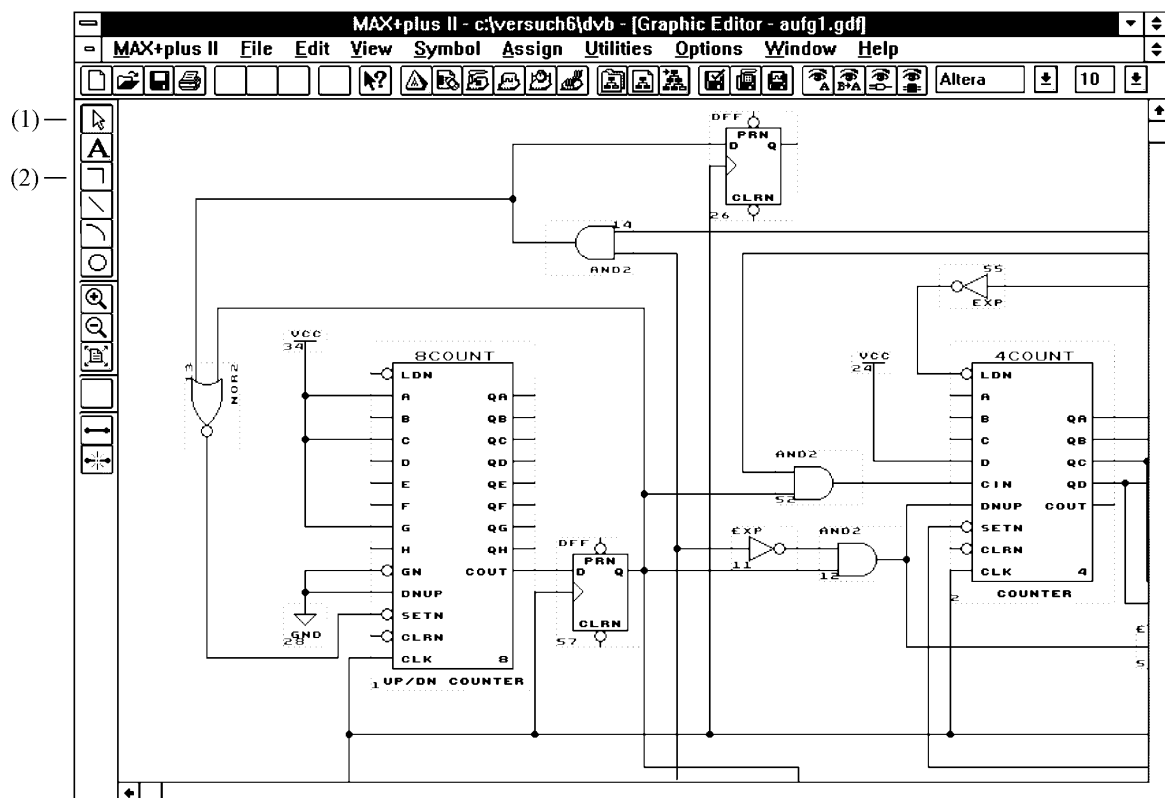
1. Auswählen der Box „**Project Name**“ aus dem „**File**“-Menü
2. Anklicken des Verzeichnisses C:\„**versuch6**“ (Spalte „**Directories**“)
3. Das Projekt „**DVB**“ sollte nun in der entsprechenden Spalte erscheinen. Es ist durch einen Klick mit der LMT auszuwählen. Die Box wird durch Anklicken des „**OK**“-Buttons geschlossen.
4. Der Projektname in der Titelleiste wird sich entsprechend ändern.

6.2 Der Umgang mit dem Graphik Editor

Die Datei „aufg1.gdf“ ist zu laden:

- „File“-Menü
- „Open“
- „aufg1.gdf“ auswählen.

Die Endung „gdf“ veranlasst MAX+plus II dazu, automatisch den Graphic-Editor aufzurufen: Das gewünschte Schaltbild erscheint in einem dem Bild 6.1 ähnlichen Fenster.



(1): Selection tool

(2): Orthogonal Line tool

Bild 6.1: Das Fenster des Graphik Editors

Zunächst ist die Datei unter dem Namen „main.gdf“ zu speichern:

- „File“-Menü
- „Save As“
- „main.gdf“ in Zeile „File Name“ eingeben, durch Drücken der „Enter“-Taste abschließen
- Die Frage „File 'c:\versuch6\main.gdf' already exists. OK to overwrite?“ ist mit „Ja“ zu beantworten.

Das hierdurch geladene Schaltbild entspricht dem in Bild 5.2 gezeigten bis auf zwei fehlende SR-Flip-Flops mit den angeschlossenen Verbindungen. Es handelt sich um das STARTSYNC-Flip-Flop und das für die Erzeugung des LOCKED-Signals verwendete Flip-Flop. Die Flip-Flops und die dazugehörigen Verbindungen sind nun einzufügen; es empfiehlt sich folgende Vorgehensweise:

1. **Festlegen des Ortes**, an dem das Flip-Flop erscheinen soll (vgl. Bild 5.2) durch **einmaligen Klick** auf die **LMT** an der entsprechenden Stelle. Kontrolle: Ein kleines schwarzes Kästchen erscheint blinkend an dem ausgewählten Ort.
2. Öffnen der „Enter Symbol“-Box:
 - „Symbol“-Menü
 - „Enter“ Symbol.
3. In der Zeile „**Symbol Name**“ ist „**srff**“ (für SR-Flip-Flop) einzugeben und die Box durch Anklicken des „**OK**“-Buttons zu schließen.
4. Das Flip-Flop erscheint nun an der zuvor bestimmten Stelle. Zum Verschieben eines Symbols ist zunächst der Button „**Selection tool**“ (siehe Bild 6.1) zu betätigen. Anschließend kann das Symbol durch Bewegen der Maus bei **gedrückter LMT plaziert** werden.
5. Für die herzustellenden Verbindungen sind rechtwinklige Linien zu verwenden; diese Linienform muss mit dem Button „**Orthogonal Line tool**“ (vgl. Bild 6.1) ausgewählt werden.
6. Die **Linien** werden mit **gedrückt gehaltener LMT gezeichnet**. Jede Linie darf einen rechten Winkel enthalten. Sollten mehrere Winkel erforderlich sein, ist die Verbindung durch Loslassen der LMT auf mehrere Linien aufzuteilen. Die für die fünf benutzten Anschlüsse des Flip-Flops erforderlichen Verbindungen sind zu erstellen. Sollte das **Löschen einer Linie** notwendig werden, ist die zu entfernende Linie mit der **rechten Maustaste (RMT)** anzuklicken. Die Linie erscheint danach in **roter Farbe**, und es öffnet sich eine Box. Dort ist „**Delete**“ auszuwählen.
7. Entsprechendes Wiederholen der Schritte 1 bis 6 für das zweite Flip-Flop.
8. **Speichern** der Datei:
 - „File“-Menü
 - „Save“.
9. **Schließen** des Graphic-Editors.

6.3 Die Eingabe mit dem Text-Editor

Wie bereits in Abschnitt 5.4 erwähnt, wird der Sync-Byte-Detektor durch die Beschreibung mit AHDL programmiert. Das Eingeben des AHDL-Subdesigns erfolgt mit dem in MAX+plus II enthaltenen Text-Editor.

1. Aufruf des Text-Editors:
 - „File“-Menü
 - „New“
 - als Filetyp wird **„Text Editor file“** gewählt und die Box mit **„OK“** geschlossen - das Fenster des Text-Editors erscheint.
2. Das in Abschnitt 5.4 wiedergegebene Subdesign ist mit der in der zweiten Hausaufgabe ermittelten Bool'schen Gleichung einzugeben.
3. Die Datei ist unter dem Namen „syncdet.tdf“ zu **speichern**:
 - „File“-Menü
 - **„Save As“**
 - **„syncdet.tdf“** in Zeile „File Name“ eingeben, durch Drücken der „Enter“-Taste abschließen
 - Die Frage „File 'c:\versuch6\syncdet.tdf' already exists. OK to overwrite?“ ist mit **„Ja“** zu beantworten.
4. Das Fenster des Text-Editors wird **geschlossen**.

6.4 Die Top-Level-Datei

Die Top-Level-Datei „dvb.gdf“ ist zu öffnen:

- „File“-Menü
- „Open“
- Auswahl **„dvb.gdf“**
- „OK“.

In dem Fenster des Graphic Editors erscheint nun das aus Kapitel 5 bekannte Schaltbild. Da die zwei Subdesigns des Projektes während der Versuchsdurchführung verändert wurden, müssen auch deren Symbole in der Top-Level-Datei aktualisiert werden:

- „Symbol“-Menü
- „Update Symbol...“
- „All Symbols in the File“
- „OK“.

Schließlich ist die Datei wieder zu speichern:

- „File“-Menü
- Save.

6.5 Die Kompilation

Das gesamte Projekt ist nun eingegeben und kann kompiliert werden. Dazu ist der Software zunächst der PLD-Typ mitzuteilen, der später mit dem Projekt programmiert werden soll:

- „Assign“-Menü
- „Device“
- „Device Family“: „**MAX 7000**“
- „Show All Packages & Speed Grades“ aktivieren
- „**EPM7032LC44-12**“ wählen
- „OK“.

Die Kompilation kann nun gestartet werden:

- „MAX+plus II“-Menü
- „Compiler“
- das Fenster des Compilers erscheint
- dort „**Start**“-Button anklicken.

Während der Kompilation wird die jeweils aktuelle Phase in dem Compiler-Fenster angezeigt. Sofern die Eingaben während der Versuchsdurchführung korrekt waren, wird der Compiler das Ende der Kompilation mit der Meldung „**Project Compilation was successful, 0 errors, 0 warnings**“ signalisieren. (Bei Fehlermeldungen ist der Betreuer zu informieren!)

6.6 Die Simulation

6.6.1 Überprüfung der Schaltung auf ihre prinzipielle Funktionstüchtigkeit

Der für die Simulation erforderliche zeitliche Verlauf der Test-Signale ist in der Datei „dvb.scf“ festgelegt. Mit der Simulation soll sowohl der Synchronisationsvorgang der Schaltung als auch das Erkennen eines neuen Synchronisationsrasters überprüft werden, dazu ist die Datei folgendermaßen aufgebaut:

Die Periodendauer eines Taktes beträgt 100 ns, entsprechend einer Taktfrequenz von 10 MHz. Das erste Sync-Byte erscheint bei $t = 18,95 \mu\text{s}$; es folgen neun weitere im Abstand von 188 Bytes bzw. $18,8 \mu\text{s}$.

Danach verschiebt sich das Raster der Sync-Bytes, wie es z.B. bei einem Wechsel auf einen anderen Übertragungskanal passieren kann, wenn beide Sender keine synchronen DVB-Signalströme liefern. Das erste Sync-Byte des neuen Abschnittes liegt bei $t = 218,75 \mu\text{s}$. Die Schaltung muss diesen Rasterwechsel erkennen und daraufhin korrekt neu synchronisieren.

Für die Simulation ist der Simulator zu starten:

- „MAX+plus II“-Menü
- „Simulator“
- „Start“.

Das Fortschreiten des Simulationsvorganges wird durch einen roten Balken veranschaulicht. Nach Beenden der Simulation wird die Datei „dvb.scf“ durch Anklicken des „**Open SCF**“-Buttons geöffnet. Der zum Betrachten und Bearbeiten der SCF-Datei erforderliche Waveform Editor wird hierbei automatisch gestartet. In dem nun geöffneten Fenster ist der zeitliche Verlauf der Eingangssignale **DATA [7..0]**, **RESET** und **CLK**, der Ausgangssignale **SYNC**, **LOCKED** und **RSYNC** sowie der internen Signale („buried nodes“) **COUT** (Sync-Zähler) und **QD ... QA** (Prüf-Zähler) dargestellt. Während die Eingangssignale Simulationsvorgaben sind, handelt es sich bei den übrigen Signalen um Ergebnisse der eben durchgeführten Simulation. Der jeweilige Vergrößerungsmaßstab der Darstellung lässt sich mit den beiden links im Fenster befindlichen Lupen verändern. Zunächst ist der Maßstab so zu wählen, dass die gesamte Simulationssequenz von 0 bis $770 \mu\text{s}$ zu erkennen ist. Dieses wird durch entsprechend häufiges Betätigen der mit „-“ bezeichneten Lupe erreicht. Ist die Schaltung funktionstüchtig, so ist die erste Synchronisation am Wechsel des **LOCKED**-Signales von logisch LOW auf logisch HIGH zu erkennen. Ebenso sind das vorübergehende Ausrasten und die Neusynchronisation auf das zweite Raster der Testsequenz ersichtlich. Folgende Fragen sind zu beantworten:

1. *Nach wie vielen empfangenen Sync-Bytes ist die Schaltung erstmalig synchronisiert?*

Antwort: Nach _____ Sync-Bytes.

2. *Wie viele Sync-Bytes werden von dem Sync-Zähler nach dem Programmwechsel noch in dem alten Raster erzeugt, bevor **LOCKED** auf logisch LOW wechselt?*

Antwort: _____ Impulse.

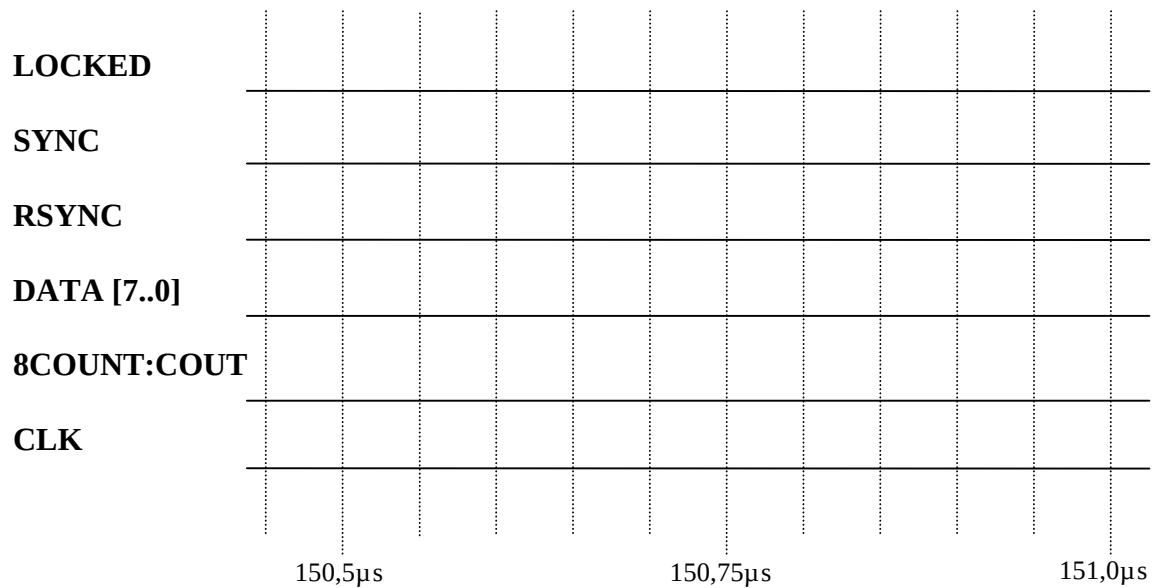


Bild 6.2: Diagramm zur Versuchsdurchführung (Abschnitt 6.6.2)

	RSYNC	COUT	SYNC
Zeitpunkt der jeweils vorhandenen CLK- Flanke [μ s]			
Zeitpunkt der Signalflanke [μ s]			
Verzögerung [ns]			

Tabelle 6.1: Ermittlung der Verzögerungszeiten

6.6.2 Ermittlung der Leistungsfähigkeit der Schaltung

Es sollen nun die Vorgänge beim Einrasten der Schaltung näher untersucht werden. Dazu ist der entsprechende Ausschnitt folgendermaßen zu vergrößern: Mit der **LMT** wird auf die erste steigende Flanke des LOCKED-Signales geklickt. Daraufhin erscheint an dieser Stelle eine **vertikale blaue Linie**, der Reference cursor. Anschließend wird die mit „+“ gekennzeichnete Lupe einige Male angeklickt, bis der Ausschnitt von ca. **150,5 μ s** bis **151,0 μ s** formatfüllend dargestellt wird. Die Verläufe der in Bild 6.2 aufgeführten Signale sind dort einzutragen.

Im Folgenden sollen die genauen Verzögerungen der Signale RSYNC, COUT und SYNC, bezogen auf die jeweils vorhergehende steigende CLK-Flanke, bestimmt werden. Dazu kann der **Reference cursor** mit den beiden **Pfeil-Buttons** neben dem **Ref-Feld** bewegt und auf die Flanken aller Signale gesetzt werden. Am oberen Ende des Cursors ist der aktuelle Zeitpunkt abzulesen, auf dem der Cursor plazierte ist. Somit können die Zeitpunkte der steigenden Flanken der

o.a. Signale sowie des Taktes ausgemessen und die entsprechenden Verzögerungen berechnet werden. Die ermittelten Verzögerungen sind in Tabelle 6.1 einzutragen.

Schließlich ist noch die maximale Taktfrequenz, mit der die Schaltung betrieben werden darf, von Interesse. Zu ihrer Bestimmung beinhaltet MAX+plus II einen Timing Analyser, der zunächst aufzurufen ist:

- „MAX+plus II“-Menü
- „Timing Analyser“
- In dem „**Analysis**“-Menü ist die Analyseart „**Registered Performance**“ auszuwählen und die Analyse zu starten.

Nach Fertigstellung der Analyse meldet sich MAX+plus II mit der Meldung „**Timing Analysis completed**“, welche mit „**OK**“ zu bestätigen ist.

Die maximale Taktfrequenz beträgt _____ MHz.

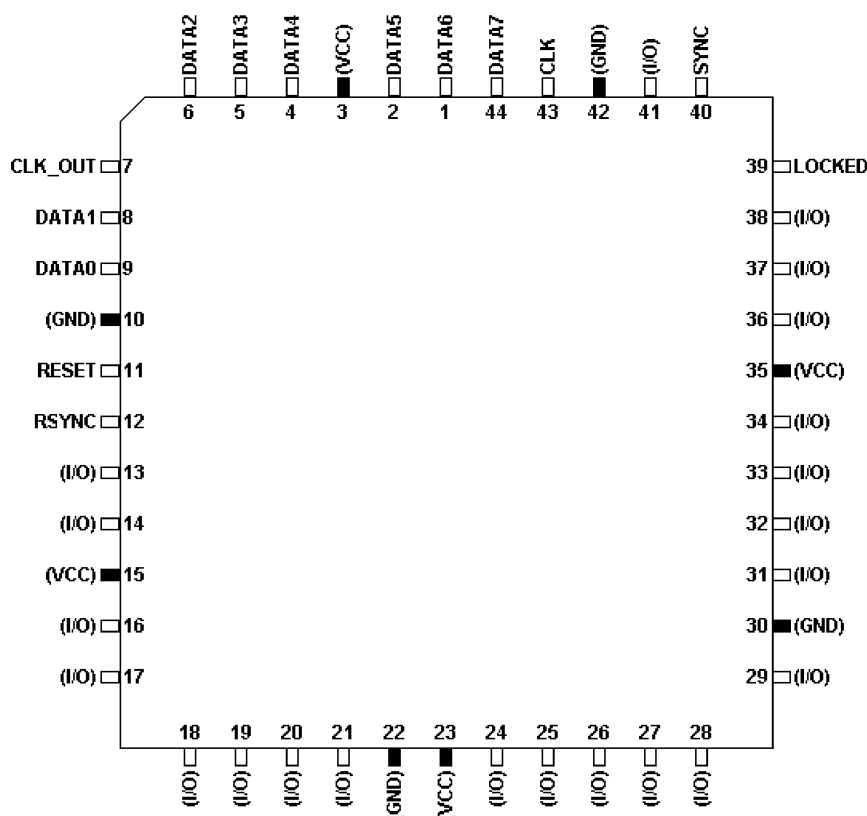


Bild 6.3: Pinbelegung des programmierten ICs

Abschließend wird der Floorplan-Editor aufgerufen:

- „MAX+plus II“-Menü
- „Floorplan-Editor“
- „Layout“-Menü: „Device View“ und „Last Compilation Floorplan“ auswählen
- Die Pinbelegung des PLDs ist mit der in Bild 6.3 wiedergegebenen zu **vergleichen**.

Anmerkung: Die korrekte Pinbelegung ist zwar in der entsprechenden Datei (vgl. Abschnitt 3.3) vorgegeben worden, diese Überprüfung dient aber der zusätzlichen Sicherheit.

6.7 Programmierung des PLDs

Zur Programmierung des Bausteines ist zunächst das Fenster des **Programmers** zu öffnen:

- „MAX+plus II“-Menü
- „Programmer“.

Unter „**File**“ muss der Name „**dvb.pof**“ erscheinen. Der **PLD** ist anschließend in den Sockel des Programmiergerätes **einzusetzen**. Der **Programmiervorgang** wird durch Anklicken des „**Program**“-Buttons **gestartet**. Nach Beendigung des Programmiervorganges ist das **Programmer-Fenster** zu **schließen** und der **PLD** dem Programmiergerät zu **entnehmen**.

Zur Durchführung eines Funktionstests des programmierten PLDs stehen ein DVB-Testsignalgenerator und ein auf den PLD abgestimmtes Test-Board zur Verfügung. Dieses Test-Board stellt die erforderlichen Verbindungen zu den einzelnen Pins des ICs her. Außerdem ist auf ihm ein Taster angebracht, mit dessen Hilfe der Reset-Eingang des PLDs (Pin 11) aktiviert werden kann. Eine LED, die über einen Vorwiderstand an Pin 39 des ICs angeschlossen ist, zeigt den aktuellen Zustand des LOCKED-Signales an. Ferner stehen die Signale CLK, CLK_OUT (das durch den PLD geleitete CLK-Signal), RSYNC und SYNC an separaten Anschlussbuchsen für weiterführende Messungen (s.u.) zur Verfügung.

6.8 Funktionstest und Messung

Zur Durchführung des Funktionstests ist das IC in die Fassung des Test-Boards einzusetzen. Anschließend wird die Versorgungsspannung eingeschaltet. Um sowohl den DVB-Generator als auch den in den PLD integrierten Sync-Byte-Empfänger in einen definierten Ausgangszustand zu versetzen, sind bei beiden Geräten die Reset-Taster zu betätigen. Anschließend ist am Generator der Schalter „DATA“ auf die Position „ON“ zu schalten:

Erst jetzt werden die Datenausgänge des Generators aktiv, während das CLK-Signal bereits nach dem Anlegen der Versorgungsspannung erzeugt wird. Bei einer ordnungsgemäß funktionierenden Schaltung sollte nun die mit „Locked“ bezeichnete LED leuchten. Entsprechend sollte sie nach dem Abschalten des Datenstromes („DATA“: „OFF“) erlöschen. Der Funktionstest ist damit beendet.

Die Ergebnisse der Schaltungssimulation (Abschnitt 6.6.2) sollen nun mit dem programmierten PLD verifiziert werden. Dazu steht ein digitales Speicher-Oszilloskop zur Verfügung. Die Verläufe der Signale LOCKED, SYNC und RSYNC sind zu messen und mit den in Bild 6.2 eingetragenen zu vergleichen. Dabei ist der bereits eingezeichnete Verlauf des CLK-Signales als Bezugsgröße zu verwenden. Zu beachten ist, dass die absoluten Zeitpunkte der gemessenen Signalfanken nicht mit denen der Simulation übereinstimmen. Der Grund hierfür ist, dass das von dem DVB-Testsignalgenerator erzeugte Signal quasi Zufallscharakter hat. So erfolgt beispielsweise das Einrasten der Schaltung, also der Wechsel des LOCKED-Signals von logisch LOW auf logisch HIGH, nicht unbedingt zum selben Zeitpunkt wie in der Simulation. Vielmehr ist die relative Lage der einzelnen Signale zum CLK-Signal von Interesse. Des Weiteren sind die Verzögerungen der Signale RSYNC und SYNC in Tabelle 6.1 einzutragen. Die Abweichungen zwischen Simulation und Messung sind zu protokollieren. Schließlich soll die Verzögerung des CLK_OUT-Signales gegenüber dem CLK-Signal gemessen werden.

7. Zusammenfassung

Dieser Versuch ist eine Einführung in die Programmierung von komplexen PLDs. Nach einem Überblick über die derzeit erhältlichen Bausteingruppen folgt eine Erläuterung der prinzipiellen Unterschiede zwischen produkttermbasierten PLDs und programmierbaren Gate Arrays. Als Beispiel für diese beiden Bausteinfamilien wird auf die Architektur der MAX-7000 und FLEX-8000-Serie von Altera eingegangen. Eine Einführung in MAX+plus II verdeutlicht die Leistungsfähigkeit aktueller Entwicklungsumgebungen sowie die Vor- und Nachteile der beiden verwendeten Design Entries anhand von einfachen Beispielen. Darüber hinaus wurden die umfangreichen Simulationsmöglichkeiten vorgestellt und im Rahmen der Versuchsdurchführung die Aussagekraft der Simulationsergebnisse durch Messungen am realen Baustein überprüft.

8. Abkürzungen

Abkürzungen aus dem Bereich der PLDs:

ASIC	Custom Application-Specific Integrated Circuit
CPLD	Complex PLD
ERA	Electrical Reconfigurable Array
FPGA	Field Programmable Gate Array
GAL	Generic Array Logic
LAB	Logic Array Block (Altera)
LCA	Logic Cell Array
LE	Logic Element
MAX	Multiple Array Matrix (Altera)
PAL	Programmable Array Logic
PGA	Pin-Grid Array (Bezeichnung einer Gehäuseform)
PIA	Programmable Interconnect Array (Altera)
PLA	Programmable Logic Array
PLD	Programmable Logic Device
PROM	Programmable Read Only Memory
SRAM	Static Random Access Memory

Anmerkung:

Steht vor einer PLD-Abkürzung der Buchstabe „E“, so ist der entsprechende Baustein löschbar (z.B. mit UV-Strahlung); fängt die Abkürzung mit „EE“ oder „E²“ an, so deutet dieses auf die elektrische Lösbarkeit hin („electrically erasable“).

Weitere in diesem Versuch verwandte Abkürzungen:

AHDL	Altera Hardware Description Language
CIN	Enable-Eingang, siehe Bild 9a
CLK	clock
COUT	Carry Out
DNUP	Down/Up
DVB	Digital Video Broadcasting
ETSI	European Telecommunications Standards Institute
GDF	Graphic Design File
GN	Enable-Eingang, siehe Bild 9b
LDN	Load-Eingang, siehe Bild 9
LMT	linke Maustaste
MAIN	Hauptmodul des Projektes
POF	Programmer Object File
RPT	Report File
RMT	rechte Maustaste
RSYNC	Signal für empfangenes Synchronwort
SCF	Simulator Channel File
SNF	Simulator Netlist File
LOCKED	eingerstet (synchronisiert)
TDF	Text Design File
VHDL	<u>V</u> ery <u>H</u> igh Speed Integrated Circuit <u>H</u> ardware <u>D</u> escription <u>L</u> anguage

9. Literatur

- [1] Heusinger/ Ronge/ Stock: *Handbuch der PLDs und FPGAs.*
Franzis'-Verlag, 1994.

- [2] Kuhlmann, Ulrike: *Völlig logisch.*
Elrad, H. 5/1994, S. 56-62.

- [3] Diverse Verfasser: *Abwägen – Komplexe, programmierbare Logik beurteilt.*
Elektronik Praxis Nr. 17 (September 1994), S. 40 – 45.

- [4] Altera: *1995 Data Book.*

- [5] Altera: *MAX+plus II - Getting Started.*

- [6] Altera: *MAX+plus II – AHDL.*

- [7] Reimers, U. (Hrsg.): *Digitale Fernsehtechnik - Datenkompression und Übertragungstechnik für DVB.*
Springer-Verlag, Berlin, Heidelberg, New York, ..., 1995.