Dr. Gerd K. Heinz
Dr. Helmut Jahne
Friedrich-Karl Staats
Uwe Schröder

Zentralinstitut für Kybernetik
und Informationsprozesse
Rudower Chaussee 5
Berlin, O-1199

Schnelle Datenpfad-Algorithmen für CMOS- ASIC

Schaltungsdynamisch sind **vollständige, generalisierte Baumhierarchien** das Medium, beliebig breite **Datenpfade mit theoretisch minimalen Verzögerungszeiten** zu gestalten. Mit der hier vorgestellten Arbeit wird der Versuch unternommen, diesen Gedanken stärker zu popularisieren.
Es sollen extrem schnelle (im Rahmen von ASIC-Entwurfsmethoden schnellste), in logarithmischer Zeit am Verzögerungszeitminimum arbeitende Datenpfadelemente beliebiger Busbreite durch **ausschließliche** Nutzung von Baumstrukturen erzeugt werden. Es wurden Algorithmen entwickelt, die in Orientierung auf Bitbreiten noch über 64 Bit bei Fanouts beliebiger, interner Leitungen oberhalb einer vorzugebenden Grenze abbrechen, und zu einer **Baumentwicklung des Knotens** übergehen. Die Methodik wurde mit dem Ziel entwickelt, extrem schnelle Datenpfade speziell für ASIC- Schaltkreise (Gatearrays, Standardzellen, PLDs) generieren zu können.
Erste Entwicklungsergebnisse machen es möglich, im Dialog aus dem Schematics- Editor eines modernen CAD- Systems heraus Datenpfad-Slices der Breite von 4 bis 32k Bit zu erzeugen, die ausschließlich aus extrem schnellen, linearen, binären Bäumen zusammengesetzt sind, dh. die auch **bei extrem hohen Busbreiten keine dynamischen Fehler verursachen.** So werden zB. Takt- und Reset- Leitungen in die Baumentwicklungen eingeschlossen. Schaltung und Symbol des Datenpfadelements werden generiert. Sie können, einem Bibliotheksgatter gleich, als Subblock benutzt werden. Vorzugeben sind Busbreite, Input-, Output-, und Control- Aktivitäten, Fanout im Schaltungsinnern sowie modulspezifische Parameter. Der Entwerfer erhält im Dialog Informationen zur maximalen Gatterlaufzeit sowie zur Transistorzahl des generierten Moduls. Der "Entwerfe- Verwerfe- Zyklus kann durch Einsatz der Generatoren beschleunigt werden. Innerhalb von Minuten ist es möglich, Baugruppen zu entwerfen, deren Entwicklung Wochen gekostet hätte.

Stand der Technik

Neben Lösungen boolscher Dekompositionsverfahren für die Logiksynthese in ASIC- Entwurfssystemen existiert bereits ein breites Spektrum technisch relevanter, auf unterschiedlichen Algorithmen beruhender Syntheseverfahren für spezielle Datenpfadelemente, so zB. für parallele Addierer (CLA, CSA), Multiplizierer (Booth[1], Luk/Vuillemine[2] etc.) oder Filter, die nicht in übliche Dekompositionsverfahren 'hineinpassen'. Datenpfadelemente mit baumartiger Verbindungsstruktur werden üblicherweise nur als Bibliothekselemente (fester Busbreite) angeboten.
Kern schneller Algorithmen für CMOS müssen aus schaltungsdynamischer Sicht baumartige Strukturen maximal homogener Aus- bzw. Einfächerung sein[3]. Zu optimieren sind Lastfaktoren der Gatter,

dh. das Verhältnis von angeschlossener Belastung zu eigener Treibfähigkeit[4] und damit Eingangskapazität; nicht aber die Stufenanzahl des Moduls. Dieser Gedanke findet in bislang gängigen Entwurfssystemen aber kaum Berücksichtigung. Er wurde zum Ausgangspunkt einer systematischen Suche nach einer passenden Methodik zur Entwicklung von Algorithmen für Datenpfadelemente auf Bäumen nach einheitlichem 'Strickmuster'.
Angestrebt wird eine universelle Eignung für CMOS- ASIC in asynchronen oder synchronen PLD-, Gatearray- und Standardzellanwendungen mit n- und p- Kanal- Transistoren von untereinander gleicher Kanalbreite. Dennoch werden keine Einzeltransistoren in die Generierung einbezogen. Es wird davon ausgegangen, daß fertige, etwa gleichgroße Bibliotheksgatter genutzt werden können, oder das solche mit geeigneter Software auf Basis von Hamilton- Minimierungen des Diffusion Sharing Graph[5] schnell generierbar sind. Es wurden Baumzerlegungsalgorithmen entwickelt, die die Bearbeitung von nahezu unbegrenzt hohen Busbreiten der Datenpfad- Scheiben gestatten. Es wird angenommen, daß CMOS- Komplementärgatter zur Verfügung stehen, bzw. daß der Datensatz auf vergleichbare Bibliothekselemente gemapt werden kann.
Anregung für die Entwicklung des vorgestellten Datenpfadgenerators war die Arbeit von Budach[6] u.a. zu rekursiven, geometrisch-starren Bildungsschemata für baumartige Layouts.

Höhere Design- Produktivität bei ASIC's

Beim Entwurf breiter Datenpfade hat der Entwerfer wiederkehrend sich ähnelnde Probleme zu lösen: bei jedem Datenpfad- Modul ist ausgehend von der maximal möglichen Signallaufzeit zu entscheiden, ob mit simplen, durch n-fache Wiederholung entwerfbaren, aber langsamen Wiederholstrukturen vom "Ripple"- Typ; mit Riesengattern oder mit baumartigen, schnellen, aber schwierig zu entwerfenden Strukturen vom Baumtyp (Wallace-Tree, Carry Lookahead etc.) gearbeitet wird. Es wäre wünschenswert, würde ein Schaltplan- oder Layouteditor nicht nur n- fache Aneinanderpappungen realisieren, sondern könnte er Zellen auch nach diffizieleren Algorithmen verdrahten. Dann wäre es Minutensache, eine Gleitkommaeinheit probehalber zu entwerfen und die Modulvarianten gegeneinander auszutauschen, um ihre Leistungsfähigkeit im Detail zu verbessern. Der Entwerfe- Verwerfe- Zyklus könnte neue Dimensionen erreichen, feiner abgestimmte, schnellere Entwürfe wären der Erfolg. Lokale Scheinoptimalität könnte durch globale Optimierung einer Schaltung ersetzt werden. Der Entwerfer - etwas verunsichert von der Masse an Literatur zu schnellen Algorithmen [7] - müsste sich nicht mit Irrungen höherer Art herumplagen (zB. DADDA- Multiplier [8]), um nach Wochen festzustellen, das er auf dem Holzweg ist. Es sollten möglichst schnelle, aber dennoch keine exotischen Algorithmen für jedes Detailproblem gefunden werden.
Es zeigt sich, daß eine kleine Klasse von Bäumen, sogenannte linearisierte oder lineare (L-) Bäume, geeignet ist, bei Busbreiten größer etwa 16 Bit ein geeignetes Gedankenmodell für die Entwicklung von schnellen Algorithmen für die vorgestellten Datenpfadelemente zu sein.

Kommerzielle Erfolgschance

Das VLSI/ULSI- Zeitalter bringt die Herausforderung großer Schaltungskomplexität mit sich. Verschiedene Hilfsmittel, wie Standard- und Makro- Zellkonzepte, Modulgeneratoren und Silicon-Compiler wurden entwickelt, um den Logik- und Layoutentwicklungsaufwand zu reduzieren. Je komplexer diese Hilfsmittel aber ausfallen, desto schwerer amortisieren sie sich als Softwarelösungen bei Verkaufszahlen, die den Dutzendbereich kaum übersteigen.
Gefragt sind Hilfsmittel, deren Grundarchitektur einfach und alterungsresistent ist, dh. die unabhängig von Skalierung, Technologie und Zellbibliotheken sind. Damit kann eine langjährige Weiterentwickelbarkeit ermöglicht werden, die mittelfristig Chancen für den Markteinstieg eröffnet.
Ein weiteres Kriterium gewinnt seit den siebziger Jahren zunehmend Einfluß auf Schaltkreis- System- und Softwarearchitekturen: die Irrtumswahrscheinlichkeit neuronaler Strukturen (Fehlerquote des Menschen) wächst in exponentieller Relation mit fehlendem Training. Das heißt: Die Lösung weniger komplexer Probleme erbringt drastisch fehlerärmere Ergebnisse.
Auf die Entwicklung von Software für den Schaltkreisentwurf angewandt heißt das: inhaltlich beschränkte Arbeitsmethoden (zB. nur auf Logik oder nur auf Layout, nur auf Datenpfade oder nur auf Steuerwerke) werden tendenziell immer wertvoller. Die Beschränkung eines Datenpfadgenerators auf algorithmische, nicht auf Layoutprobleme wird zum Kriterium für Fehlerarmut.
Damit kommt ein drittes Kriterium der Entwicklung von CAE Software zum Tragen: das Bemühen des Entwerfers vorausgestzt, von jedem Softwaretyp das beste Paket auszuwählen, und alle Pakete miteinander zu konglomerieren, werden Softwarelösungen der Zukunft vor Heterogenität strotzen. Flexible und einfache Anpassbarkeit der Software an verschiedenste Daten- Schnittstellen und Userinterfaces, Einbeziehbarkeit unterschiedlichster Bibliotheken wird zum Erfolgskriterium am Markt.

In der vorgestellten Arbeit sollte dies berücksichtigt werden. Sie begann Anfang 1989 in der DDR unter Maßgabe einer Orientierung auf den schnellen und effektiven (Nach-) Entwurf einer Vielzahl verschiedener VLSI- Prozessoren amerikanischer Bauart unter den Bedingungen des Wirtschaftsembargos. Mit dem Thema sollte ein algorithmischer Vorlauf geschaffen werden, um mit ASIC- Entwurfsmethoden in den Full- Custom Sektor einzudringen, und eine möglichst langjährige Weiterentwickelbarkeit und Nutzungsquote zu erzielen. Gefordert war eine Eignung für verschiedene Herstellungstechnologien sowie Skalierungsstufen, um den Entwicklungsaufwand minimal zu halten.

Datenpfad in Scheiben

Abbildung 1 zeigt einen für die Realisierung unterschiedlicher Datenpfade nötigen Grundumfang zu unterstützender Moduln.

Folgende Randwerte waren zu berücksichtigen:

1. Ziel ist eine Eignung für breite Busse (16...256 Bit), damit sind Prämissen zu setzen:
   - schnelle Datenpfadelemente verlangen komplexe Baum- Algorithmen (keine Ripple- Regularität mehr möglich)
   - die Layoutgenerierung beschränkt sich auf lokale Nachbarschaftsbeziehungen der Gatter, vermittelt über relative Gatter-

koordinaten; technologieorientierte Layoutprobleme werden ausgegrenzt
- die Busordnung zwischen den Moduln ist unter allen Umständen zu erhalten, da jede Permutation zwischen Datenpfad- Slices nahezu quadratische Layoutfläche kostet

2. Nur definiertes und niedriges Fanout an jedem Gatterausgang und Einbeziehung interner Buffer bei großen Bit- Distanzen und hohem Fanout eines Gatters sichert geringe Laufzeit der Moduln

3. Geringer Hardwareaufwand bei möglichst rechteckigen Shapes der Moduln (trotz nichtregulärer Netzgraphen) ist anzustreben

4. Pipelinefähigkeit ist zu sichern, das setzt Gleichzeitigkeit auf allen Bitpositionen im Modul voraus (was nutzt ein Modul, der 53 von 54 Bit in 8 Gatterlaufzeiten kompariert, wenn das letzte Bit erst nach 27τ fertig ist?

5. Eine Abbildung auf einfache, in einer ASIC- Bibliothek enthaltene, oder einfach nachrüstbare Grundgatter ist vorteilhafter, als die Nutzung von Spezialgattern, die theoretisch gut, praktisch aber nicht verfügbar sind. Eine typische CMOS-Kanalarchitektur (n- /p- Gleichverteilung) ist anzustreben

6. Layoutgenerierungen sind immer technologiegebunden, deshalb ist auf Netzlisten abzubilden. Maßgabe: ein ASIC- System mit Netzlistenschnittstelle und Autorouter steht zur Verfügung. Vorzug: Fehlerarmut der Algorithmen, auch wenn traditionelle Modulgeneratorhersteller (inclusive Layout) irgendwann 'nachziehen'

7. Extrem schnelle Sonder- Schaltungstechniken, wie CVSL[9], DSL etc. werden nicht unterstützt, da sie derzeit mit dem Entwicklungsziel nicht harmonieren

Linearisierte Bäume

Abb.2 zeigt einige technisch relevante Bäume mit zugehörigen Linearisierungen.
Die Bäume tasten mit ihren (so bezeichneten) Operatoren p,o,i verschiedene Bitpositionen eines oder mehrerer Busse (nicht dargestellt) ab. Schalter s,t,u,v stellen eine Verbindung beliebiger Bitbreite zwischen Operatoren p und Stamm z her. Der Ereignisfluß kann in Abhängigkeit von der Aufgabe sowohl von z nach p, als auch entgegengesetzt sein.
Gegenüber einem Datenpfad der Breite von n Operatoren in Ripple-Anordnung (Abb.2a) sind bei baumartiger Vernetzung der Elementaroperatoren auf Bitniveau $O\{\log_{exp}(n)\}$ Gatter zu passieren, dh. eine Baumrealisierung ist ab einer bestimmten Busbreite schneller, als eine Ripple- Anordnung. Basis exp des Logarithmus ist die Ein- oder Ausfächerung angenommen identisch bestückter Schalterebenen. Die Suche nach dem Optimum soll aber von den Algorithmen nicht unterstützt werden, das ist Sache von Elektrik- Simulationen.
Aus einem binären Baum (Abb.2b) ensteht zB. ein linearer L- Baum durch eine 2-fach Teilung des Rasters (Abb.2c,d). Die Schalter s,t,u,v werden anstelle eines jeden zweiten Operatorelementes p

eingeschoben. 'Zufällig' existiert dann gerade dort Freiraum, wo ein in der Hierarchie höher stehender Schalter angeordnet werden muß. Es ist zu erkennen, daß trotz baumartiger Verdrahtung eine rechteckige Zellarchitektur des Datenpfades entsteht. Zu beachten ist, daß die Verdrahtungsdichte zur Mitte hin mit dem Logarithmus der Busbreite wächst.
Eine Linearisierung trinärer, ternärer oder höher ausfächernder Bäume ist für Datenpfadanwendungen ebenfalls möglich. Die Wahl der Ausfächerung wird durch elektrische Technologie- Parameter sowie durch die angewandte Schaltungstechnik bestimmt. Allerdings füllt nur der binäre L- Baum einen Spaltenvektor lückenlos, wenn angenommen werden kann, daß jeweils nur ein Operator p,i,o oder ein Schalter an einer Position einer Matrix(spalte) stehen kann. Beim ternären und vierfach- Baum entstehen Lücken. Lücken deuten auf ein Verschenken aktiver Fläche bei gleichzeitiger Erhöhung der Anzahl kaskadierter Transistoren in den Gattern hin.
Man beachte die bei Bäumen gegenüber einer Ripple- Struktur verschiedenartige Einbeziehung des Initialisierungseingangs i bzw. des Übertragausgangs o in die Operatorebene.

Delay- Optimierung in Bäumen

Die Zweifach- Ausfächerung eines Binärbaumes trifft unter Berücksichtigung der angeschlossenen Leitbahnkapazität nahezu das Mead/Conwaysche Verzögerungszeit- Minimum e (e = 2,818...).

Unsymmetrien
Ein Binärbaum zeigt auf jeder Hierarchieebene ein homogenes Ausfächerungsverhalten, solange es gelingt, ihn nahezu symmetrisch in das Layout zu bringen (Abb.3).
Algorithmisch bedingte Unsymmetrien zwischen Astteilen von Bäumen führen zu Unsymmetrien im Timing der Astteile. Darunter leidet die Pipeline- Eignung des Moduls. Restlos können Unsymmetrien nicht verhindert werden (zB. beim Incrementer), da Layoutanordnung oder gewählter Algorithmus keine Alternativen offenlassen.

Richtungsabhängigkeit
Die Dynamik von aufwärts (in Zählrichtung der Hierarchieebenen) geführten Signalen ist geringfügig verschieden von der abwärts führender Signale. Während im Abwärtsfall zumeist zwei Eingänge von einem Schalter gespeist werden können, wird im Aufwärtsfall zumeist nur ein Eingang gespeist. Die Belastung ensteht beim Aufwärtsfall im Gatter selbst, siehe Abb.4, weil Potential X stets kapazitiv geerdet ist. Zwischen hoher Ein- und Ausfächerung existiert deshalb bei CMOS- Bäumen kein Wesensunterschied: dynamisch wirken sich beide ähnlich negativ aus.

Längenkompensation
Von Hierarchieebene zu Hierarchieebene aufwärts verdoppeln sich die Längen der Leitungen, die die Schalter verbinden. Demzufolge sind mit höherer Hierarchieebene auch höhere Signallaufzeiten zu erwarten. Oberhalb einer kritischen Signallaufzeit 'delay' werden in jede Stufe zusätzliche invertierende Buffer 2^i-fachen Treibvermögens einbezogen (i: Hierarchieebene). Die H/L- Aktivität der Schalterlogik wird dann vertauscht. Die Berechnung der kritischen Verzögerungszeit erfolgt ausgehend von üblichen, technologiespezifischen Näherungsformeln.

$$delay = tau\_chain + tau\_load \cdot (fanout - 1)$$

$$fanout = (cap\_of\_inputs + grid\_count \cdot cap\_per\_grid) / cap\_per\_unit$$

Einfügung eines Buffers für:

$$tau\_load \cdot (fanout - 1) > j \cdot tau\_chain, \quad j = const$$

Verallgemeinerbare Kriterien zur Verzögerungsoptimierung unterschiedlicher Bäume sind nicht angebbar, siehe dazu[10]. Der Entwerfer wird im Dialog nach dem maximal statthaften Fanout befragt. Damit kann der Entwerfer verschiedene Varianten erproben.
Die Ermittlung zusätzlicher Buffer wird baumspezifisch vorgenommen.

Algorithmen für Bäume

Schlüssel für Baum- Generierungen zu einer beliebigen Basis exp ist die höchste, in die aktuelle Busbreite n passende Potenz v(n) zur Basis exp, bei Binärbäumen also die höchste hineinpassende Zweierpotenz.
Wird (beim Binärbaum) die zu generierende Busbreite als Binärzahl dargestellt, repräsentiert der Wert der höchsten Bitposition diese höchste Potenz (zB. Busbreite 11 -> $v(11) = 2^3 = 8$), siehe Abb 4).
Die Lage der Schalter im Anordnungsvektor wird über feststehende Formeln bestimmt.
Unterhalb von v(n) entfaltet sich bis zur niederwertigsten Bitposition ein vollständiger Teilbaum. Oberhalb von v(n) ist bis zur höchsten Bitposition ein höherwertiger Restbaum zu berechnen, der durch besondere Schalter an abgeschnittenen Zweigteilen gekennzeichnet ist.
Bei binär verzweigenden Bäumen ist ein solcher Schalter ein Inverter, falls das Schalterelement vom AOI- Typ wäre, bzw. ein nichtinvertierender Buffer oder Kurzschluß, falls der gleichwertige Schalter aus einem nichtinvertierendem Doppelgatter oder einem Transfergate (Multiplexer) bestanden hätte.
Die Tiefe des Binärbaumes ist identisch der Anzahl Binärziffern der Busbreite (Beispiel: 1010_{bin} entspechend 4). Schalter unterschiedlicher Hierarchien erfüllen ia. eine adäquate Funktion. Wenn es gelingt, diese Funktion aus einem einzigen AOI- (And-Or- Invert) Gatter pro Schalter zu bilden, resultiert daraus eine ia. von Stufe zu Stufe der Hierarchie alternierend inverse Logikfunktion (zB. abwechselnd NAND/NOR).
Die Bestimmung der Schalter des Restbaumes, deren Logikfunktion nicht benötigt wird, ist einfach: man schreibe die zu realisierende Busbreite als Binärzahl neben den Baum und lese die Null-Stellen ab. Im Beispiel Abb.4 stehen die Nullen der Binärzahl auf den Positionen der Schalter u und s, diese Schalter werden je nach Fall durch Kurzschluß oder invertierenden/ nichtinvertierenden Buffer ersetzt.
Ein Problem stellt die Behandlung der abgeschnitten Schalter höherer Hierarchien in der Linearisierung des Baumes dar (vgl. den abgeschnittenen Schalter u in Abb.4. Hier sind in Abhängigkeit vom Modul Detailprobleme zu lösen, die im allgemeinen auf die Verschiebung des Schalters zum (n+1)- ten Matrix- bzw. Vekto-

relement hinauslaufen, um das Datenpfadelement nicht unnütz lang werden zu lassen. Die maximale Distanz zwischen einem abgeschnittenen Schalter und dem msb- Operator ist dabei nicht größer als ein Viertel der Busbreite.
Je nach algorithmischer Ausprägung entstehen in Abhängigkeit von der Gestaltung der Operatoren sowie der Schalter verschiedenartige Baumgebilde. In der Reihenfolge werden zur Berechnung **eines** Baumes ia. folgende Schritte abgearbeitet (Laufindex i); ein Datenpfad kann aber aus mehreren Bäumen bestehen, so zB. beim Incrementer (3 in sich verschachtelte Bäume); die Datenpfadscheiben stehen senkrecht:

1) Bestimmung der Busbreite als Binärzahl $b_{k-1}...b_o$

2) Ermittlung der Baumtiefe kmax als Stellenzahl der binären Busbreite

3) Netzkopfbehandlung
   - Berechnung der Randpin- Koordinaten und Namen
   - Rückgabe der Parameterliste als Kommentar

4) Bestimmung freier Matrixpositionen

5) Berechnung der Elemente

```
for (i = 0; i <= busbreite; i++)
 {
```

a) Bestimmung der Elemente der nullten Spalte (x = 0)

   Anordnung der Operatoren p(x,y) auf jedem j-tem Platz mit der Bitposition

$$p_i(x,y) = (0, ij) \quad \text{mit} \quad j \in \{1,2,3...\}$$

$$(0 =< ij =< \text{busbreite})$$

b) Bestimmung der Elemente der ersten Spalte (x = 1)

   Anordnung der Schalterpositionen s(x,y) von der höchsten Hierarchieebene absteigend; Bestimmung des Schaltertyps q entsprechend Hierarchieebene, Buffer und L/H- Aktivität

$$s_i(x,y,q) = (1, 2^{k-1} i + 2^{k-2}, q)$$

.
. (Bestimmung weiterer Spalten)
.

c) Benennung der abgehenden Leitung(en) des Gatters

d) Korrektur abgeschnittener Schalter

e) Nachbehandlung I/O- seitiger L/H- Aktivitäten

f) Kontrolle des statthaften Fanout

}

6) Konvertierung in gewähltes Netzlistenformat, Zeile in File schreiben

Ausgabe in verschiedene Netzlistenformate

An die Module wird die Liste der Parameter übergeben. Die Module reichen eine 'Struktur' weiter, die von einem einheitlichen Konverter in beliebige Netzsprachen übersetzt werden kann. Sämtliche Moduln können im menuegeführten Dialog, wie auch als Sprachelemente mit Parametrisierungen in einer Liste aufgerufen werden.
Verschiedene ASIC- Systeme besitzen eine verschiedene Syntax der Netzlisten. Mit der Übergabe der Generierungsergebnisse an eine relativ breit angelegte C-Struktur und einer anschließenden Konvertierung sind unterschiedliche Netztypen ohne Änderung an den Modulen generierbar. Eine Struktur lin entsteht zB. unter mehrfachem Aufruf der Unterstruktur pot. Die Struktur pot wurde dreidimensional angelegt, um Bäume, deren innerer Aufbau vorteilhaft mit einem dreidimensionalen Algorithmus behandelt werden kann (Formen von Cube Connected Cycles[11] etc.) optimal algorithmieren zu können. Der Parameter f gibt den jeweils genutzten Freiheitsgrad an.

```
struct pot    {char  *name;
               int   f,x,y,z;
               };

struct lin    {int     eltyp,inp_numb,out_numb,bidir_numb;
               char    *eltyp_char;
               struct pot inp[16];
               struct pot out[7];
               struct pot bid[6];
               struct pot ele_name;
               } *line;
```

Beispiel Zeroflag

Anhand des Beispiels Zeroflag soll eine algorithmische Methode zur Generierung von Binärbäumen demonstriert werden.
Das Zeroflag hat schnellstmöglich am Ausgang ein Signal (low- oder high- aktiv) abzugeben, wenn der Businhalt komplett null (low- oder high- aktiv) dh. null oder eins ist. Es stellt mithin den Oberbegriff über beliebig breite OR/NOR- und AND/NAND- Gatter dar, wenn vorausgesetzt wird, daß bei sämtlichen Datenpfadelementen low- und high- aktive In- und Outputs möglich sein müssen. (Eine schaltungstechnische Standardlösung des Zeroflags als einfaches NOR- Gatter möge für höhere Busbreiten absehbar als zu langsam gelten.)
Die Netzliste beginnt mit der Randbeschreibung ($NET, $INP, $OUT) des Moduls.

```
$NET art_zer12
$INP pa1x0 pa1x1 pa1x2 pa1x3 pa1x4 pa1x5 pa1x6 pa1x7 pa1x8 pa1x9
```

```
     pa1x10 pa1x11
$OUT pa2x0
```

Es folgen die Gatter- Aufrufe, jeweils bestehend aus $SUB, $INP, $OUT. Leitungsnamen beinhalten durch x getrennt die Gatterkoordinaten des Leitungsursprungs (treibendes Gatter). Die $SUB- Anweisung ist wie folgt zu lesen:

```
$SUB typ name x_koord y_koord
```

Typ und Name sind im Beispiel zufällig identisch.
Es beginnt mit der höchsten Hierarchie:

```
$SUB nand     nand    0 8
$INP pa2x4 pa2x12
$OUT pa2x8
```

Die Eingänge kommen von der darunterliegenden Ebene, diese wird als nächstes generiert

```
$SUB nor      nor     0 4
$INP pa2x2 pa2x6
$OUT pa2x4
```

Der Baum ist beim Element 12 abgeschnitten, der Restbaum ist zu behandeln, Element 12 wird durch einen Inverter ersetzt

```
$SUB inv      inv     0 12
$INP pa2x10
$OUT pa2x12
```

Diese Hierarchieebene ist fertig bearbeitet.
Vom Element 4 ausgehend wird die nächstniedere generiert

```
$SUB nand     nand    0 2
$INP pa2x1 pa2x3
$OUT pa2x2

$SUB nand     nand    0 6
$INP pa2x5 pa2x7
$OUT pa2x6
```

Vom Element 12 ausgehend gleichfalls eins tiefer

```
$SUB nand     nand    0 10
$INP pa2x9 pa2x11
$OUT pa2x10
```

Nun noch die tiefste Ebene

```
$SUB nor      nor     0 1
$INP pa1x0 pa1x1
$OUT pa2x1

$SUB nor      nor     0 3
$INP pa1x2 pa1x3
$OUT pa2x3

$SUB nor      nor     0 5
```

```
$INP pa1x4 pa1x5
$OUT pa2x5

$SUB nor      nor     0 7
$INP pa1x6 pa1x7
$OUT pa2x7

$SUB nor      nor     0 9
$INP pa1x8 pa1x9
$OUT pa2x9

$SUB nor      nor     0 11
$INP pa1x10 pa1x11
$OUT pa2x11
```

Zum Abschluß die Nachbehandlung: der Ausgang soll invertiert werden, dazu hielten wir bisher die Position (0, 0) frei:

```
$SUB inv      inv     0 0
$INP pa2x8
$OUT pa2x0
```

Da im Fall des Zeroflags an den Zweigen das Baumes keine Operatoren hä

```
$END
```

Beispiel Incrementer

Abbildung 6 zeigt eine Generierung eines arithmetischen Negators mit der ungewöhnlichen Busbreite von 13 Bit. Das kleine Beispiel läßt die Komplizierheit baumartiger Strukturen gegenüber Ripple-Schemata erahnen. Der Modul besteht aus drei Bäumen: Spalte 1 sucht nach mit einem einfächernden Baum nach Nullen, die Spalten 2 und 3 erzeugen mit einem ausfächernden Baum einen Vektor, der oberhalb der von LSB aus gesehen ersten Nullstelle des Eingangsvektors aus Nullen, und unterhalb aus Einsen besteht. Dementsprechend werden von der aus Antivalenzen bestehenden Spalte 4 alle Bitpositionen bis zur und einschließlich der ersten Null invertiert. Das Umsetzverfahren ist hierarchisch: eine Null breitet sich in allen darüberliegenden Baumzweigen aus, und blockiert alle MSB-seitig liegenden Äste in allen Hierarchiestufen.
In Spalte 2 ist das Novum der hier vorgestellten Methodik zu erkennen: Knoten mit einer Ausfächerung oberhalb einer im Dialog angebbaren Grenze sind **generell**, auch für Reset-, Control- oder Taktleitungen verboten. So ist statt einer einfachen INCR- Steuerleitung zusätzlich ein dritter, ausfächernder Baum zur Umschaltung des Moduls zwischen passieren/incrementieren (Steuersignal INCR) eingefügt. Nur dadurch ist gewährleistbar, das der Modul in einer abartig hohen Bitbreite ebenso solide funktioniert, wie bei geringen Bitbreiten.
Zu Anwendungen des Moduls ein Auszug aus den Help- Files, die unter VMS- Help installiert sind (Zahlen in der ersten Spalte zeigen in VMS-Help-Libraries die Hierarchieebene an):

```
3 INCREMENT
    Addition einer Eins zu jedem Datenwort innerhalb des Binaer-
    zahlbereiches 000...000 bis 111...111 inclusive Sprung von
    111...111 auf 000...000 (Ueberlauf).
```

4 APPLICATE

Anwendung zB. zur Zweierkomplementbildung. Der Incrementer ist bei Subtraktionen fuer die Bildung des Zweierkomplements geeignet. Das Datenwort kann ueber eine Komplementaerbelegung des Buseingangs negiert werden, vom Incrementer wird zum Datenwort binaer eine Eins auf der letzten Stelle addiert.

4 EXAMPLE

Das Zweierkomplement kann ueber eine Negation und eine anschliessende Inkrementierung gewonnen werden.
Die Negation uebernimmt der mit entgegengesetzter Aktivitaet betriebene Eingang.
Zweierkomplementbildung, Ein- und Ausgangsbus sind 1- aktiv:

```
INCR- Belegung = fix 1   (inkrementieren)
input_activity:      0   (Negation)
output_activity:     1
```

4 OVERFLOW

Ueber-/Unterlauf bei Zweierkomplementdarstellung:
Wird der Binaerwert 011...111 erreicht (groesste positive Zahl in Zweierkomplementdarstellung) springt der Inkrementer im Folgeschritt auf 100...000 (kleinste negative Zahl im Zweierkomplement). Der Uebergang von dez.-1 = 111...111 auf dez.0 = 000...000 erfolgt korrekt.

Ueber-/Unterlauf bei Vorzeichen/Betragsdarstellung:
In einer Vorzeichen/Betragsdarstellung arbeiten Incrementer und Decrementer nur innerhalb der positiven (00...000 bis 11...111) Zahlengruppe korrekt. Innerhalb der negativen Zahlengruppe (10...000 bis 11...111) arbeiten beide in inverser Richtung!
Der Uebergang von dez.+0 auf dez.-0 und zurueck sind falsch
(dez.0 = 00...000 = 11...111 <---> dez.-0 = 10...000).
Der Incrementer versucht, den absoluten Betrag der Zahl stetig zu vergroessern, der Decrementer den Betrag zu erniedrigen.

4 PINS

```
data_in:          in(0:(n-1))
data_out:         out(0:(n-1))
cntl_in:          incr
```

Der Eingangsbus ist generell 1-aktiv. Fuer den Ausgangsbus ist die Busaktivitaet (high/low) anzugeben.
Steuereingang INCR (1- aktiv):

```
0:  Durchgang      (data_out :== data_in)
1:  Inkrementieren  (data_out :== data_in + 1)
```

Applikation Digitalfilter für 300 MHz

Abbildung 7 zeigt eine Applikation zweier Moduln als digitales Tiefpaß- Filter für hohen Datendurchsatz. Dieses nichtrekursive, multipliziererfreie Filter wurde einem Beispiel von Vielhauer[12] entnommen. Zwei der in Kette geschalteten Moduln nach Abb. 7 ergeben einen Tiefpaß mit dem Übertragungsfaktor G >= 0,95 für 0 <= Ω <= 0,1 π und G <= 0,15 für 0,6 π <= Ω <= π.
Die Fertigung des Filters in einer 1μm- Technologie läßt einen

Datendurchsatz bis zu 300 MHz bei einer Gesamtverzögerung pro Datenpfadmodul kleiner als 10 ns erwarten. (Die Schaltung der CLA- Adder ist zu umfangreich für eine "entfaltete" Darstellung)

Zusammenfassung

Mit der vorgestellten Arbeit wurde ein erster Versuch unternommen, schnelle Datenpfade beliebiger Busbreite durch ausschließliche Nutzung binärer Baumstrukturen zu erzeugen. Es wurden Algorithmen für theoretisch maximale Geschwindigkeiten entwickelt, die in Orientierung auf Bitbreiten bis 256 Bit bei jedem, auch Modul- internen Signal oberhalb einer vom Entwerfer bestimmbaren Fanout- Grenze abbrechen, und zu einer Baumentwicklung übergehen. Je nach Fanout- Wahl entstehen binär-, drei- oder mehrfach gestaffelte Bäume.
Durch die Kombination der Generierung von Netzlisten mit relativen Matrixkoordinaten gelingt es, extrem schnelle Datenpfade auch für ASIC- Schaltkreise (Gatearrays, Standardzellen, PLDs) generieren zu können. Die bewußte Abkopplung vom Layout schafft den für die Bearbeitung komplizierter Baumalgorithmen nötigen Freiraum.
Es zeigt sich, daß ein wichtiges Werkzeug für die Entwicklung von Modulgeneratoren auf Bäumen ein solide funktionierender Schaltplan- Router ist. Leider war nur ein mit Mängeln behafteter verfügbar. Dadurch mußte viel manueller Arbeitsaufwand in die Kontrolle von Testgenerierungen investiert werden.
Die Modulgeneratoren wurden in der Programmiersprache C verfaßt, und in einen Schaltplaneditor moderner Bauart integriert. Dieser sollte virtuell unbegrenzte Seitenformate unterstützen, andernfalls ist vor dem Routing des Moduls eine Seitenteilung vorzusehen.
Möglichkeiten komfortabler Netzsprachen, wie zB. NBSF[13] konnten aus technischen Gründen nicht erprobt werden. Es wäre denkbar, daß die Verfügbarkeit von Netzsprachen mit zusätzlichen Möglichkeiten, wie mehrdimensionale Indizierungen, bedingte Sprünge, Verzweigungen und Marken (IF, THEN, ELSE, GOTO, label:) sowie Geometrieinformationen (relative Rasterkoordinaten der Gatter) die Modulentwicklung in der angegebenen Form beschleunigen könnte.
Mit dem vorgestellten Konzept wird es zB. ohne zusätzliche Technologieanforderungen möglich, auf einem 1μm Standardzell- ASIC in Pipeline arbeitende Datenpfade mit einem 200 MHz- Pipeline-Takt bis etwa zur Busbreite von 64 bit zu entwerfen.
Eine Förderung des Themas aus AZA- Mitteln wurde mit dem BMFT-Bescheid 415-5853-1-61/90 v.10.1.91 abgelehnt. Zum Zeitpunkt der Veröffentlichung ist die Arbeitsgruppe aufgelöst.

Literatur:

(1) M.Annaratone, W.-Z.Shen, "The Design of an LSI Booth Multiplier: nMOS vs. CMOS Technology", Carnegie Mellon University, Dept. of Comp. Science, CMU-CS-84-150, Pittsburgh PA, USA, 1984

(2) W.K.Luk, J.E.Vuillemin, "Recursive Implementation of Optimal Time VLSI Integer Multipliers", IFIP 1983, pp.155-174

(3) Heinz, G. "Ansätze zur analytischen Beschreibung der Dynamik digitaler CMOS-Gatter", Diss.A, Humboldt-Universität zu Berlin, 18.2.1988

(4) Carver Mead, Lynn Conway: Introduction to VLSI Systems. Addison- Wesley- Publ. Corp., Reading MA, USA, 1980

(5) C.C.Chen, S.-L. Chow, "The Layout Synthesizer: An Automatic Netlist-to-Layout System", 26th ACM/IEEE Design Automation Conference, Paper 16.1, pp.232-238

(6) L.Budach, E.G.Giessmann, H.Grasmann, B.Graw, Ch.Meinel, "Rekursives Layoutentwurfssystem RELACS", Handbuch, Humboldt-Universität zu Berlin, Sektion Mathematik, 1986

(7) Kai Hwang, "Computer Arithmetic", John Wiley & Sons, New York, 1979

(8) D.G.Crawley, G.A.J.Amaratunga, "8 x 8 pipelined Dadda multiplier in CMOS", IEE Proceedings, Vol.135, No.6, December 1988

(9) K.Yano, T.Yamanaka u.a."A 3.8-ns CMOS 16x16-b Multiplier Using Complementary Pass-Transistor Logic", IEEE JSC, Vol.25, No.2, April 1990

(10) Heinz, Gerd, "Untersuchung dynamischer Eigenschaften digitaler CMOS- Gatter", Nachrichtentechnik- Elektronik, Berlin, Vol.40, 1990, Heft 1

(11) G.M.Baudet, F.P.Preparata, J.E.Vuillemin, "Area-Time Optimal VLSI Circuits for Convolution", IEEE Transactions on Computers, Vol.C-32, No.7, 1983

(12) P. Vielhauer, "Einfache Methode zum Entwurf digitaler Filter ohne Multiplizierer", Nachrichtentechnik- Elektronik, Berlin, Heft 6, 1985, S. 218-220

(13) I.Teetz, T.Fischer, W.Issel, "Die Sprache NBSF für die strukturelle Beschreibung von Schaltkreisen", Akademie der Wissenschaften, Karl- Weierstraß- Institut für Mathematik, Report R-MATH-01/90, Berlin, 1990

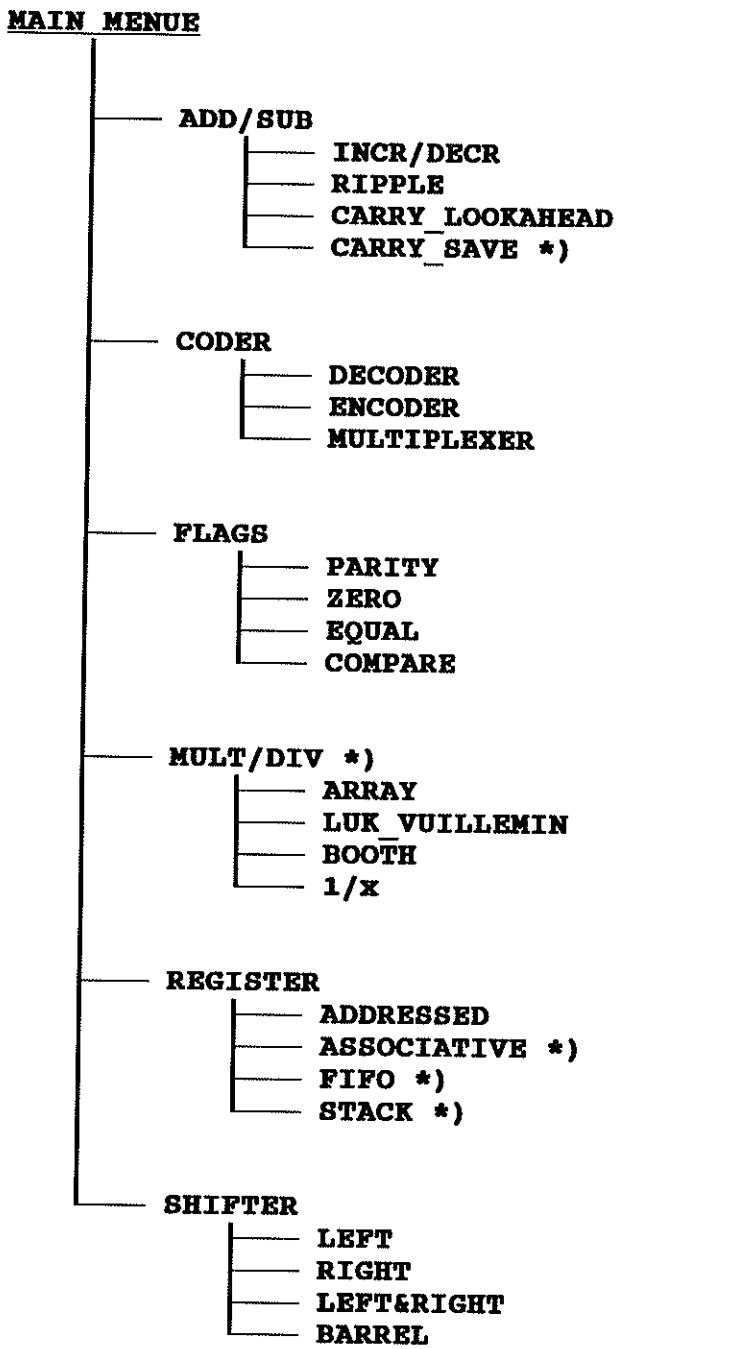


Abb.1 Modulsortiment des Datenpfadgenerators

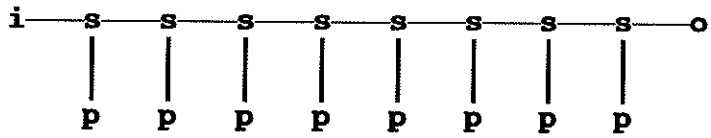


Abb.2a) Gewöhnliche Ripple– Anordnung

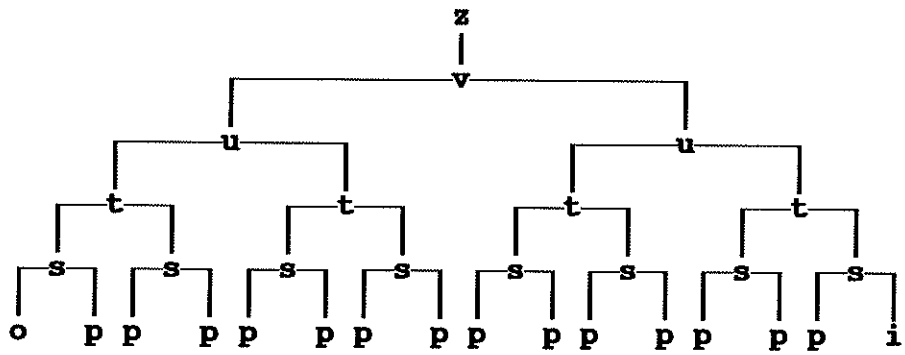


Abb.2b) Binärer Baum

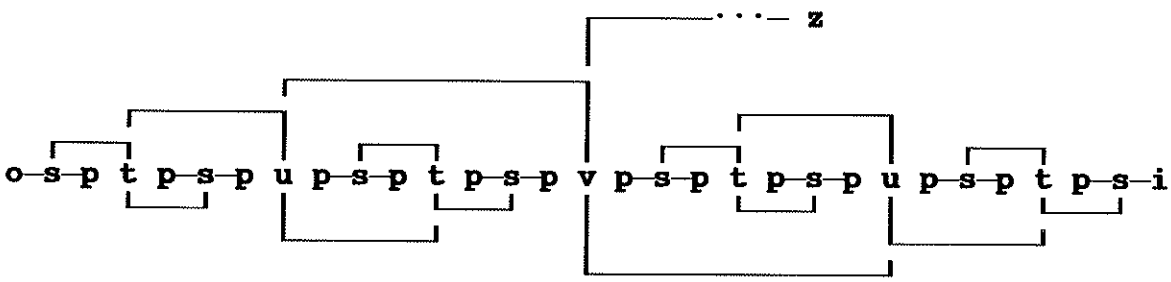


Abb.2c) Geometrische Linearisierung eines binären Baumes

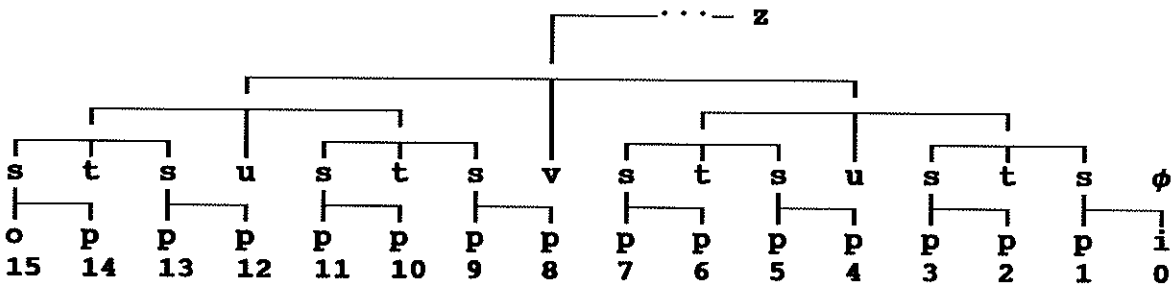


Abb.2d) Linearisierter, zweireihiger binärer Baum

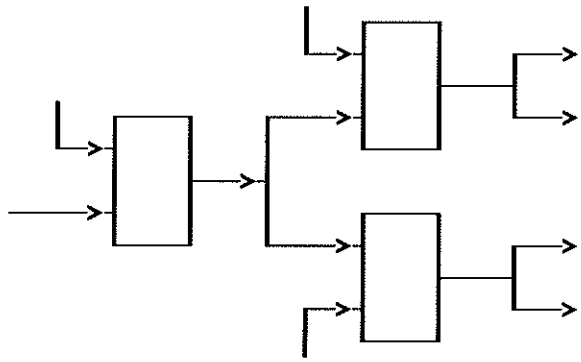


Abb.3a) Ausfächernder Baum

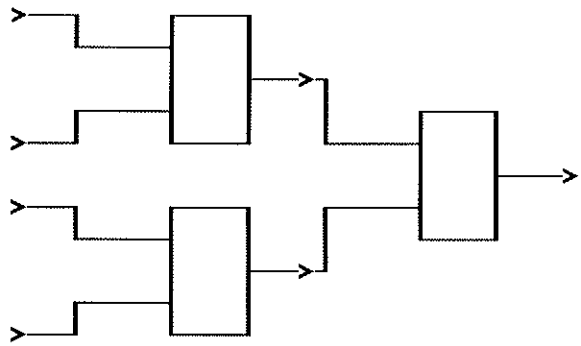


Abb.3b) Einfächernder Baum

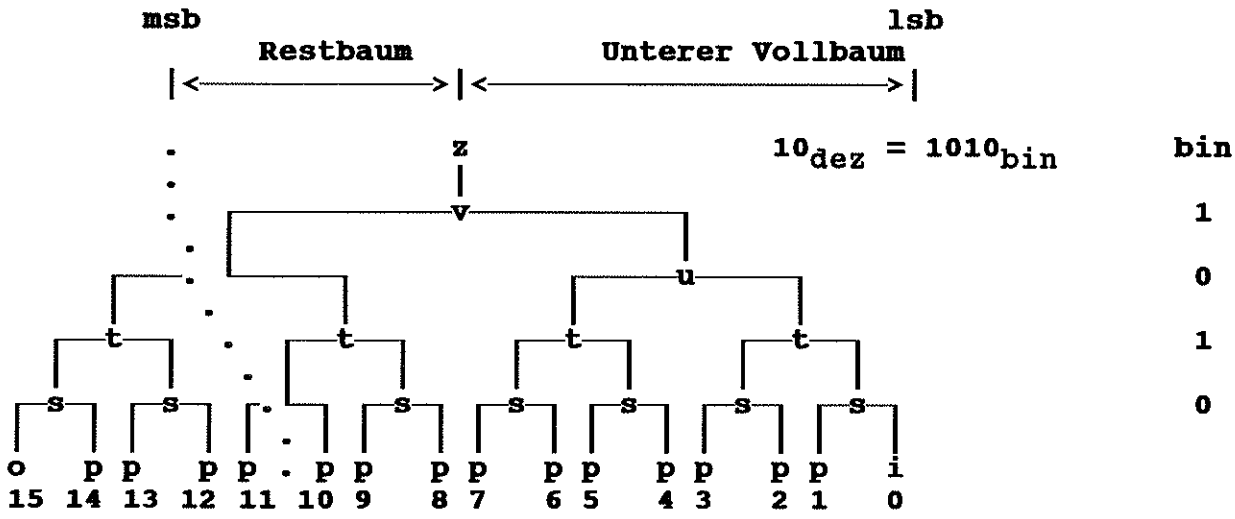


Abb.4) Geschnittener Binärbaum

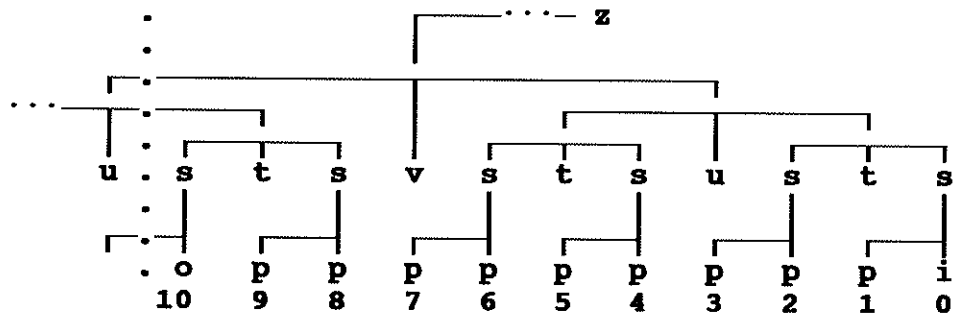


Abb.5 Abgeschnittener Schalter im linearisierten Baum (linkes u)

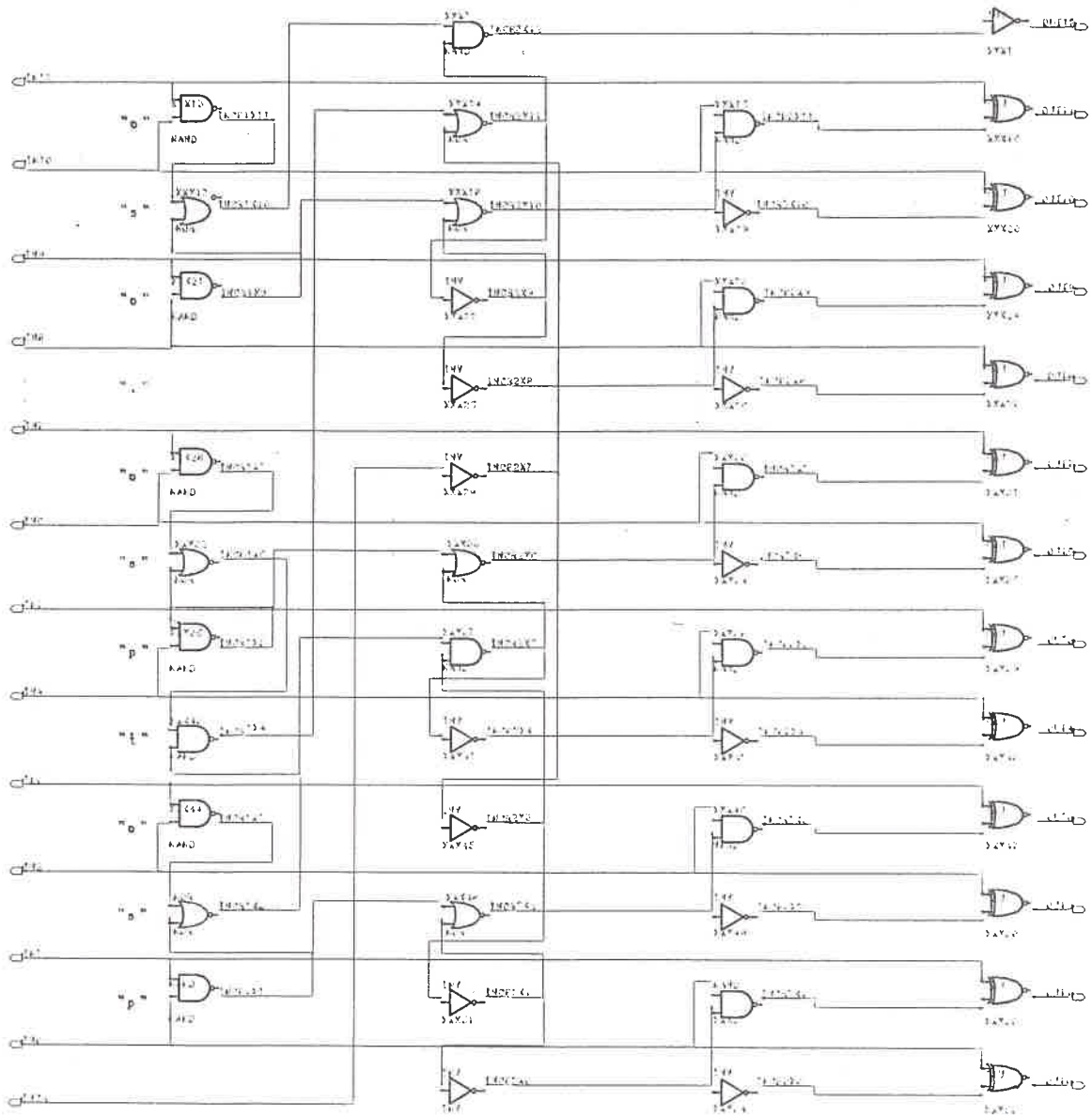
Abb.6 Beispiel: Incrementer 13bit als 2er Komplement- Bildner

Abb.7 Applikation: Elementartiefpaß 12bit für einen Datendurchsatz bis zu 300 MHz in einer 1µm- Technologie

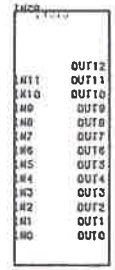
Inputs

to built two's complement, negate the inputs!

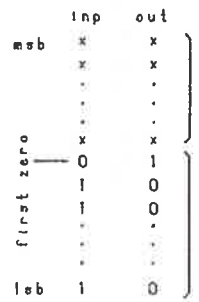
Outputs



Symbol



Function



input-activity: 1
output-activity: 1

Control { 0: pass / 1: incr

delay: 9 gates, 227 transistors

INCREMENT 13bit

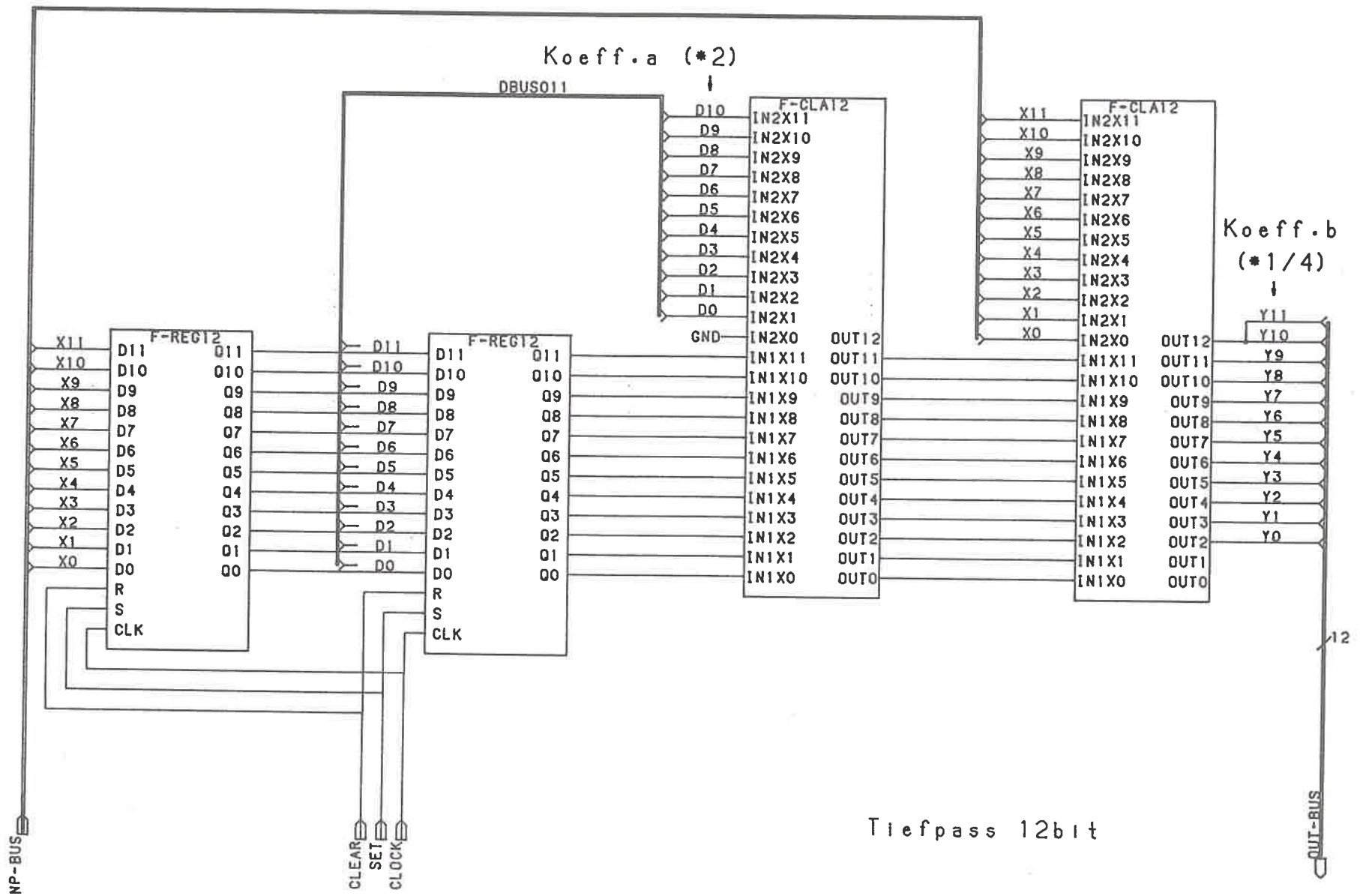
Column1 — zero-search-tree

Column2 — invert-stop-tree

Column3 — invert-stop-tree

Column4 — execute-neg

Abb.6


Koeff.a (*2)
DBUS011
F-REG12
F-REG12
F-CLA12
F-CLA12
Koeff.b
(*1/4)
INP-BUS
CLEAR
SET
CLOCK
OUT-BUS
12
Tiefpass 12bit

FILTER