mantık devrelerinin tasarımında beliren sorunlar

yazan: DoUgldS LEWIN

çeviren: Kaya YAZGAN

UDK: 681.322.065.2

ÖZET

Mantık tasarımında önde gelen sorunlar özellikle karmaşık, geniş çapta tümleşik (large scale integration, LSI) devrelerin tasarımında karşılaşılan güçlükler tartışılmaktadır. Yerleşmiş kuramsal yöntemlerin, örneğin en aza indirgemenin (minimization) geçerliğini yitirdiği ve altdizge (subsystem) düzeyinde tasarımlama için yetersiz olduğu öne sürülmektedir. Salt oku bellekler (read only memory, ROM) ve çoklayıcı <multiplexer) devrelerin kullanılması ayrıntılarıyla gözden geçirilmekte ve dizge tasarımında günümüzdeki araştırmaların bir özeti verilmektedir.

SUMMARY

Current problems in logic design, in particular those concemed with the design and implementation of digital systems using complex LSI circui are discussed. it is argued that the established switching theoretic methods, .such as minimizatio are no longer relevant, and that existing theory is inadeguate for design at the subsystem level. The difficulties associated with the implementation of ROM and multiplexer circuits are reviewe in detail and finally a summary is given of present day research in systems specification and design.

Douglas Lewin, Prof., Brunel University.

Kaya Yazgan, TCDD Elektronik Araştırma Merkezi.

İL. GÎRÎŞ

VEDEĞİL, YADADEGÎL geçitleri, iki duraklılar *(flip-flop)* kullanarak mantık devrelerinin gerçekleştirilmesi üstünde uzun süredir çalışılıyor. Bu çalışmalar sonucunda en az sayıda geçit ve ikiduraklı kullanımına yönelik yöntemler geliştirildi. Bu yöntemler günümüzde de temel olarak (doğru olmakla birlikte tümleşik devre teknolojisindeki ilerlemeler nedeniyle hızla yararsız olma yolundadır. Modern geniş çapta tümleşik *(LSI)* devrelerde ve karmaşık metal oksit yarıiletken *jMOS)* devrelerinde bir geçit, tranzistor yada diyot tasarrufu hemen hemen önemsizdir. Bazı durumlarda başka amaçlarla bir en aza indirgeme *(minimization)* söz konusudur, örneğin geniş çapta tümleşik devre yapımında, devre alanı, geçitler ve modüller arasındaki iç bağlantılar önemlidir, îç bağlantıların sayısı geniş ölçüde modülün boyutunu belirlerken devre alanının küçülmesi de Üretimde artışa neden olmaktadır. Her iki durumda da en aza indirgeme kaydadeğer ucuzluk sağlamaktadır. Orta çapta tümleşik *(medium scale integration, MSI)* devreler ile sayıcılar *(counter)*, kayualı yazmaçlar *(shift register)*, kod çözücüler *(decoder)*, çoklayıcılar *(multiplexer)*, salt oku bellekler *(read only memozy, ROM)* gibi birçok sarmaşık mantık işlevleri ucuz, küçük devreler lalinde gerçekleştirildi. Bunlarla ikili *(binary)* fodlama ve kod çözme gibi işlemler bile yalnızca 3îr modül kullanarak yapılabilmektedir. Karmaşık tümleşik devrelerin bu gelişimi kaçınılmaz olarak nantık dizgelerinin tasarım yöntemlerini de etki-.emektedir. Birçok durumda standart orta çapta tüm-.eşiklik düzeyli devrelerin kullanılması -gereksiz geçitlere yol açsa bile- geçit sayısının en aza

indirgenmesinden daha iyi sonuç vermektedir. Denek ki mantık dizgelerinin ucuza gerçekleştirilnesinin koşulları günümüzde değişmiştir; şimdi nodüllerin sayısı, baskı devrelerin ve kablajın oasrafları göz önüne alınmalıdır.

Bundan başka ve belki de daha önemli bir değişiklik, tasarımcıların bu karmaşık elemanları bir altdizge *(subsystem)* devre elemanı olarak kullannaları; standart orta çapta tümleşimli yazmaçları, aritmetik ve mantık birimlerini kullanarak bir veri yapısı elde edip bütün işleyişi de programlaman bir salt oku bellek denetim birimi ile yönetneleridir. Böylelikle mantık tasarımı basit geçitlerden değil, karmaşık elemanlardan oluşan daha yüksek bir altdizge düzeyinde ele alınmaktadır. Genellikle bu durumun da ötesinde dizge tasarımcılarının hangi modülleri istediklerinden çok, devre elemanı yapımcılarının sağladığı modüllerin Kullanılmasına dayanan bir oluşum gözlenmekte -

lir.

Sonuç olarak tasarımcılar az bir kuramsal bilgiyle yada hiç kuramsal bilgi olmadan sezgisel çalışaaya zorlanmaktadırlar. Bunun nedeni açıktır; kuramın eyleme yetişememesi. Gerekli olan, şimdi elimizdeki gibi mantık geçiti düzeyinde değil, altdizge düzeyinde yeni bir kuramdır.

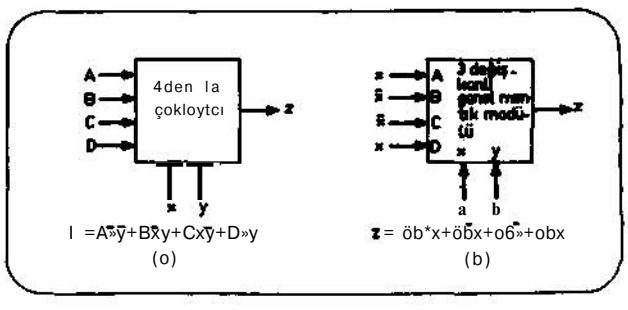
Bir başka büyük sorun mantık devrelerinin içerdiği karmaşıklıktır. Devre elemanlarındaki ve çeşitli araçlardaki teknolojik gelişimin sonucunda yapılan yeniden gözden geçirmelerin dışında, ilk kuşak bilgisayarlardan bu yana sayısal dizgelerin temel tasarım yöntemlerinde çok az değişiklik oldu. Birşeyler yapılmadığı takdirde salt düşüncel çabanın, ileri bir atılım için engel oluşturacağı noktaya hızla yaklaşıyoruz. Sayısal dizge tasarımında bilgisayar kullanılması ise bu soruna ancak kısmen çözüm bulabilmiştir.

Bilgisayarlar modüllerin yerleşimi, kablaj, deney, bakım gibi konularda çok verimli oldular [1,2]. Oysa gerçek sorunlar başlangıçtaki tasarım aşamalarında, kavramsal düzeyde ve sonraki gerçekleştirme aşamalarındadır. Tasarımcının dizgeden beklenenleri vermesi ve bilgisayar donanımı *(hardtrare)* yada yazılımı *(softtrare)* yardımıyla dizgenin tasarımlanması gerekmektedir [3]• Böyle bir çalışma sonucu bilgisayar bir mantık çizimi *(diagram)*, akış çizimi yada en iyisi yerleştirme ve kablaj çizimi vermelidir. Bu kadar karmaşık bir iş için, dizgenin hem davranış (bilgi akışı) hem de işlev (veri akışı) düzeylerinde belirlenmesi ön koşuldur. Bunun ötesinde tasarımcıya karmaşık mantık dizgelerinin oluşturulmasında yarduncı olmak amacıyla, dizgenin hem donanımını hem de yazılımını kapsayan bir yapısı oluşturulmalıdır. Ne yazık ki bu işlevi gerçekleştirecek bir gösterim *(notation)* henüz geliştirilememiştir.

2. KARMAŞIK TÜMLEŞİK DEVRELERLE MANTIK TASARIMI

Salt oku bellek, çoklayıcı gibi orta çapta tümleşik devreler uygulamadaki alışılagelmiş yöntemler üzerinde büyük etkiler yaptı. Birçok durumda geleneksel en aza indirgeme yöntemlerine hiç gerek kalmamakta, tasarımcı gerçeklik çizelgelerinden *(truth table)* elde edilen yasal *(canonical)* terimlerle çalışabilmektedir, özellikle salt oku belleklerin kullanılması hem kombinasyonel *(combinational)* hem de ardışık *(seguential)* devrelerin,

Şekil 1.



I =A$\bar{y}$+B$\bar{x}$y+Cx$\bar{y}$+D»y
(o)

z= öb*x+ö$\bar{b}$x+o$\bar{b}$»+obx
(b)

karmaşık tasarım tekniklerine gerek olmadan, ucuz modüllerle kurulmasını sağlamaktadır.

2.1. Çoklayıcılar

Çoklayıcı devreler Boole işlevlerini verebildiklerinden kombinasyonel devrelerin gerçekleştirilmesinde kullanılabilirler [4]. Örneğin Şekil la'da görülen 4 den 1 e çoklayıcı devresi, genel amaçlı, üç değişkenli bir mantık modülü olarak kullanılabilir [5]. Denetim girişleri (a, b) ile 0, 1, x ve $\bar{x}$, 4^4 = 256 farklı giriş kombinezonu verir. Bu sayı 3 değişkenli Boole işlevleri sayısına eşit olduğundan (n değişken sayısı ise 2^{2^n}), çoklayıcı devresi herhangi bir işlevi gerçekleştirmek amacıyla kullanılabilir. Örneğin a ve b nin denetim girişleri, x in ise çoklanan değişken olduğu Şekil lb'de

$$z = \bar{a}\bar{b}x + \bar{a}b\bar{x} + a\bar{b}\bar{x} + abx$$

doğrudan doğruya gerçekleştirilmektedir.

Genel olarak n değişkenli (n * 3) herhangi bir f(x1, X2,..., x_n) mantık işlevinin aşağıdaki gibi açılmasıyla, çoklayıcının işlevi daha biçimsel olarak gösterilebilir:

$$\begin{aligned} f(x_1,x_2,\ldots,x_n) &= \bar{x}_1\bar{x}_2 f(0,0,x_3,\ldots,x_n) \\ &+ \bar{x}_1 x_2 f(0,1,x_3,\ldots,x_n) \\ &+ x_1\bar{x}_2 f(1,0,x_3,\ldots,x_n) \\ &+ x_1 x_2 f(1,1,x_3,\ldots,x_n) \qquad (1) \end{aligned}$$

Örnek olarak 3 değişkenli bir f(x,y,z) işlevini ele alırsak açılım sonucunda:

$$\begin{aligned} f(x,y,z) &= \bar{x}\bar{y}f(0,0,z) + \bar{x}yf(0,1,z) \\ &+ x\bar{y}f(1,0,z) + xyf(1,1,z) \end{aligned}$$

buluruz. Burada f(0,0,z), f(0,1,z), f(1,O,z) ve f(1,1,z) artık *(residual)* işlevleri yalnızca z nin işlevidir ve her biri 0,1,z yada $\bar{z}$ olmak üzere 4 değerden birini alırlar. Dikkat edilirse bu denklem x ve y nin denetim girişleri olduğu yukarda değinilen 4 den 1 e veri seçicisini (data *selector)* tanımlamaktadır. Bundan başka Boole işlevleri her değişken sayısına göre açılabilirler; örneğin 3 değişkene göre açılma, S değişkenli tüm Boole işlevlerini veren 16 dan 1 e veri seçicisi ile sağlanabilir.

Yukardaki açıklamalardan görüldüğü gibi her sayıda değişkeni olan genel amaçlı mantık modülleri yapılabilirse de, devrelerin karmaşıklığının artışı,, maliyet artışları, bakım zorlukları gibi nedenler göz önüne alınarak, çok değişkenli mantık işlevlerinin gerçekleştirilmesinde, az değişkenli birkaç tane genel amaçlı mantık modülü kullanılmaktadır. Bu uygulama (1) denklemindeki artık işlevlerin yalnızca x_n in işlevi oluncaya dek yeniden açılmasına karşı gelen iki yada daha çok düzeyde diziler şeklinde bağlanmasıyla elde edilir.

Birkaç düzeyli uygulamalarda ilk düzeydeki denetim girişlerinin seçimi çok önemlidir; çünkü bu seçim ikinci ve daha sonraki düzeylerdeki çoklayıcı sayısını etkiler. Olanaklıysa 0, 1 ve ortak girişlerin (yani çoklayıcının veri girişlerinde paylaşılabilen girişlerin) sayısının en uygunu bulunmaktadır. Diğer bir yaklaşım daha ilerdeki düzeylerde yer alan değişkenleri birbirinin aynı, yada tersi olarak seçmektir. Bu ikinci yaklaşım, çoklayıcıların hem doğru hem de ters çıkışlarının olduğu durumda olanaklıdır. Bütün uygulamalarda amaç, daha ilerdeki düzeylerde gerekecek çoklayıcı sayısını azaltmaktır. İlerdeki düzeylerde denetim girişlerinin (birçok uygulamada aynı ise de) değişmemesi için bir neden yoktur ve bu durum da sorunu biraz daha karmaşıklaştırır. Modüllere özgü denetim girişleri çoğunlukla önde gelen düzeylerdeki gerekli modül sayısının azalmasına yol açar.

Çoklayıcı maliyetleri temel bazı geçit modüllerinin maliyetinin beş katı kadarsa da, bu gerçekleştirme yöntemi ile çok kazanç sağlanabilir. 16 dan 1 e çoklayıcı kullanımıyla sağlanan en görünür kazanç baskı devre ve kablaj kolaylığıdır. Yedek malzeme sayısının azalması ve bakım kolaylığı da gözönüne alınmalıdır. Birbiri ardına dizilmiş modüllerin bir sakıncalı yönü imlerin ileleme zamanlarının artmasıdır.

Çoklayıcılar, TTM *(transistor transistor logic„ TTL)* VEDEĞİL / YADADEGİL modüllerinden ortalama 2-3 kat daha yavaştır, fakat bu fark birçok mantık devresinin en az iki geçit düzeyi gerektirdiği gözönüne alınırsa dengelenmiş olur. Ne yazık ki ardarda bağlanmış çoklayıcıların kullanılması veya böyle dizgelerin en aza indirgenmeleri konusunda doğrudan doğruya uygulanabilecek bir kuram günümüze dek geliştirilememiştir. Bu arada burada yalnızca bir çıkışlı işlevlerin gerçekleştirilmesini gözönüne aldığımıza da dikkat edilmelidir, iki Birden çok çıkışı olan devreler bazı ek sorunlar neden olurlar.

2.2. Salt Oku Bellek Devreleri

Salt oku belleklerin kullanılmasıyla kombinasyon ve ardışık devrelerin gerçekleştirilmesi çok kolaylaşmaktadır [6,7]. Alışılagelmiş en aza indirgeme tekniklerinin kullanılmasına yada uygun durum belirlenmesine gerek yoktur. Tasarımcı dolaysız olarak doğruluk çizelgesini yada durum çizelgesini kullanabilir.

örneğin kombinasyonel bir devrenin gerçekleştirilmesinde, anahtarlama değişkenleri salt oku belleğin adres girişleri olarak kullanılır ve istenen çıkış değerleri de salt oku belleğe yerleştirilir. Böylece bir giriş kombinezonu salt oku belleğe uygulanınca, adreslenen sözcük (yani çıkış değeri), belleğin çıkışında bulunur. Salt oku belleklerin birden fazla çıkışlı devreler için özellikle çok uygun olduğu açıktır, çünkü belleğin sözcüklerinin her biti bir çıkış değerini verebilir.

Salt oku belleğin seçimi, değişken çıkış işlevi sayısı tarafından belirlenir, örneğin n-değişkenli bir devre 2^n sözcüklü bir salt oku bellek gerektirecektir. Sözcük uzunluğu ise doğrudan doğruyja istenen ^ıkış sayısına bağlıdır.

Değişken sayısı n olan bir işlevde küçük terim *(minterm)* sayısı $2^{11"1}$ den büyükse çoğunlukla işlevin tersinin programlanması daha uygundur. Bu durum büyük terim *(maxterm)lerin* kullanılmasına eşdeğerdir ve salt oku belleğin çıkışının evrib

siyle *(inversion)* istenen işlev elde edilir. Dikkat edilmesi gereken bir özellik, çıkışta belirli bir değer vermesi istenmeyen, tanımlanmamış giriş birleştirmelerinin (İngilizcede *don't çare* olarak adlandırılan durumların) salt oku bellek kullanılarak gerçekleştirilen tasarımlarda bir önem taşımamasıdır; çünkü bu devreler standart büyüklüktedir ve istenen sözcük sayısında bir indirim olanaksızdır.

Çok sayıda değişkeni olan uygulamalarda her değişkenin bellekte gereken söz.cük sayısını iki kat çoğaltması nedeniyle, tek bir salt oku bellek kullanılması olanaksızdır. Bu zorluk birçok durumda ardarda bağlanmış küçük salt oku bellekler kullanılarak yenilebilir. Daha büyük bellekler elde etmek için salt oku belleklerin birbirine bağlanması her zaman olanaklı olduğundan, örneğin iki tane 32 x 8 bit sözcüklü birim bağlanarak 64 x 8 bit sözcüklü bir bellek oluşturulabilir. Alt dizge (yani salt oku bellek) düzeyinde bir en aza indirgeme ardarda bağlantı tekniğiyle sağlanır.

Çok düzeyli bir gerçekleştirmeye örnek olarak 6 değişkenli

$$T = £(5,15,20,29,41,42,45,47,53,58,61,63)$$

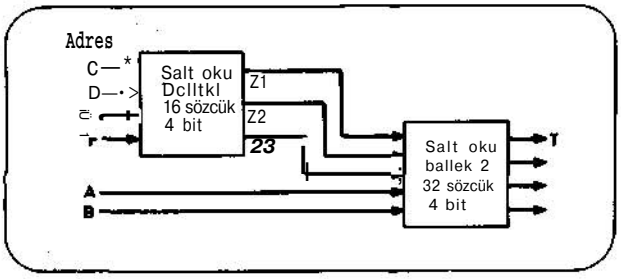
işlevini ele alalım. Devrenin ardarda bağlı salt oku belleklerle kurulması için değişkenler önce parçalanmalı, sonra istenen veri sıklığını elde etmek için yeniden kodlanmalıdır. Anahtarlama işlevi çoğunlukla ortak değişkenler yada küçük terimler taşıdığından bu işlem olanaklıdır, örneğin Çizelge 1'de değişkenler (AB) ve (CDEF) kümelerine ayrılabilirler. Şimdi (CDEF) içinde yalnızca

(0100, 0101, 1001, 1010, 1101 ve 1111)

bulunduğuna dikkat edilirse, bunları 3 bitli bir kod ile gerçekleştirmek olanaklıdır. Bu arada 0 ve çıkışta belirli bir değer vermesi istenmeyen giriş kombinezonlarının da kodlandığına dikkat edilmelidir. Şekil 2'de ardarda bağlanmış 2 salt oku bellek gösterilmektedir. CDEF birinci salt oku belleğe verilmekte, bunun verdiği Zı, Z2, Z_3 çıkışları A ve B ile birlikte, sonuçtaki işlevi gerçekleştiren ikinci salt oku belleğe uygulanmaktadır.

Bu uygulamada ilginç olan yön dört tane 16x4 bit bellek kullanılarak gerçekleştirilebilecek bir devrenin ardarda bağlantı yardımıyla üç adet 16 sözcüklü salt oku bellek ile gerçekleştirilebilmesidir.

şekil 2.



Özellikle daha çok sayıda değişkenli işlevlerde bu teknikle daha büyük kazançlar sağlanabilir. Ne yazık ki, kuramsal bir tasarım algoritması olmadığından, yöntem doğal olarak sezgiseldir. Uygulamada birden fazla çıkış gerektiğinde sezgisel tasarımın karşılaştığı sorunlar çok zor çözülmektedir. Özel salt oku bellek devreleri tasarımlayan yarıiletken yapımcıları tarafından gerçeklik çizelgelerini en uygun yöntemle parçalayan ve kodlayan bazı bilgisayar algoritmaları geliştirilmiştir.

Çizelge 1.

a. Yüksek terimler

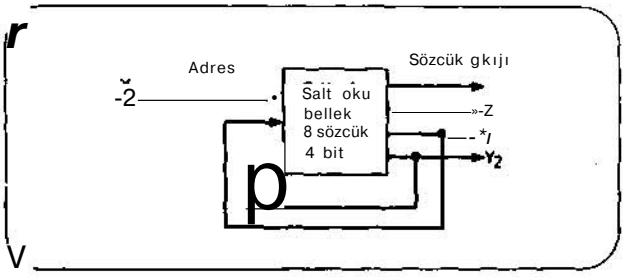
Ondalıklı yapı	Değişkenler M	CDEF
5	00	0101
15	00	1111
20	01	0100
29	01	1101
41	10	1001
42	10	1010
45	10	1101
47	10	1111
53	11	0101
58	11	1010
61	11	1101
63	11	1111

b.Payla$ılan terimler

Değişkenler CDEF	Kodlanmış yapı $Z_1Z_2Z_3$
0100	001
0101	010
1001	on
1010	100
1101	101
1111	110
diğer terimler	000

c. Birinci düzey salt oku bellek

Giriş değişkenleri AB	$Z_1Z_2Z_3$	Çıkışlar $T_1T_2T_3T_4$
00	010	1000
00	110	1000
01	001	1000
01	101	1000
10	011	1000
10	100	1000
10	101	1000
10	110	1000
11	010	1000
11	110	1000
11	101	1000
11	110	1000



Şekil 3.

Kolayca tahmin edilebileceği gibi salt oku bellekler eşzamanlı olan ve olmayan ardışık devreler için de çok uygundur. Salt oku belleklerin kullanılmasıyla gerçekleştirilen tasarımın ilk adımı yine akış çizelgesinin belirlenmesidir. Çizelge 2'de eşzamanlı olmayan tipik bir ikiye bölen sayıcının akış çizelgesi görülmektedir. Geçiş çizelgesi dışardan x girişi uygulandığında şimdiki durumdan gelecek duruma geçişleri göstermektedir: Şekil 3'de çizilen devrede yıy2 çıkışlarının doğrudan doğruya salt oku belleğin girişlerine uygulanarak, istenen ardışıklık özelliği, yani geribesleme işlemi gerçekleştirilmektedir. Devrenin bir Moore makinesi olduğuna, daha genel bir Mealy makinesinin de çıkış işlevlerinin dolaysız olarak salt oku belleğin uygun bitlerinden elde edilebileceğine dikkat edilmelidir.

Salt oku belleklerin etkin tümleşik devrelerden oluşması ve dolayısıyla geribesleme yolunda gereken kazancın sağlanabilmesi nedeniyle bu tür uygulamalar olanaklıdır. Eşzamanlı olmayan kararlı durum koşulu, salt oku belleklerde çıkış sözcüğünün kendi yerini adreslemesiyle, devingen bir çevre oluştutmasıyla sağlanır. Böylece salt oku bellek

Durum Değişkenleri	Giriş x		Çıkış
$y_1 y_2$	0	1	Z
00	00	01	0
10	11	01	0
11	11	10	1
10	00	10	1

a. Akış çizelgesi

Adres xy_1y_2	Çıkış sözcüğü y_1y_2Z
000	000
100	010
001	110
101	010
011	111
111	101
010	001
110	101

b. Geçiş çizelgesi

çizelge 2.

devresi alışılagelmiş VEDEĞİL / YADADEGİL devrelerine tamamiyle benzerdir. Ayrıca ek bir olumlu özellik, durgun tehlikeler *(static hazard)* için önlemlere gerek kalmamasıdır. En aza indirgeme sırasında oluşan bu tehlikeler, salt oku bellek gerçekleştirilmelerinde tüm terimler kullanıldığından karşımıza çıkmaz.

GÖzönüne alınması gereken bir zamanlama sorunu varsa da, salt oku bellek kullanarak, benzer bir şekilde eşzamanlı makineler de yapılabilir. Bazı durumlarda salt oku belleğin bir engelleme girişi vardır ve bu, dizge saati olarak kullanılabilir. Örneğin giriş uçlarındaki geçici durumlar sona erinceye dek çıkışlar engellenebilir. Çevredeki gerekli gecikme dışardan bağlanan orta çapta tümleşik D türü ikiduraklılar kullanılarak çıkışların adres girişlerine uygulanmasından önce gerçekleştirilebilir.

Mantık devresi tasarım kuramı üzerinde, salt oku bellekli eşzamanlı makinelerin etkisini incelemek ilginç olabilir. Ardışık olmayan devrelerin (örneğin iki duraklı giriş denklemlerinin) en aza indirgenmesinin gereksizliği açıktır. Benzer şekilde eşzamanlı makineler için en uygun durum belirlenmesi yöntemleri de gereksizdir. Bunlara karşılık durumların azaltılması ve satırların azaltılması yöntemleri, salt oku bellek sözcük sayısını azaltmakta kullanılabileceklerinden önemlerini biraz korumaktadır.-Bir salt oku belleğin eşzamanlı makinede kullanılması için boyutunu belirleyen tek koşul, giriş ve durum kodları için kullanılan bitlerin toplamının salt oku belleğin adres bitlerinden küçük yada ona eşit sayıda olması gereğidir. Salt oku belleğin sözcük uzunluğunu belirleyen tek koşul ise durum kodu ve çıkış işlevleri için gerekli bit sayısıdır.

Salt oku bellekler kullanarak ardışık makineler tasarımlayanlara yardımcı olacak kuram da çok azdır. Belleğin artırılması için salt oku belleklerin ardarda bağlandığı büyük dizgelerde bu yoksunluk özellikle sorun yaratır. Salt oku bellek dizgelerinde yararlı olabilecek bir kuramsal kavram büyük ardışık makinelerin daha küçükler tarafından gerçekleştirilmesidir. Bu küçükler daha sonra seri-paralel bağlanarak büyük makinenin özelliklerini taşıyan bir dizge oluşturabilir [8] . Bu çalışmalar verilen bir makine için gereken belleği azaltmak amacıyla kullanılabilir. Ashenhurst[1] [9]un eski çalışmaları ardarda bağlanmış salt oku bellek uygulamaları amacıyla yeniden değerlendirilebilir.

Yukardaki tartışmada salt oku belleklerin ekonomik üstünlüğü görülmemektedir. Örneğin, tartışılan ardışık devrelerin alışılagelmiş mantık birimleriyle gerçekleştirilmesi, salt oku bellek kullanımından çok daha ucuza sonuçlanırdı. Buna karşılık bir tek birimin kullanılmasındaki kolaylık, tasarım, deneme, bakım giderlerinin azlığı düşünülürse önemli üstünlükler ortaya çıkar. Kısacası günümüzde salt oku belleğin kullanılıp kullanılmaması konusunda özenle seçim yapılmalıdır. Küçük dizgeler için mantık paketleri hâlâ en ucuz yöntemdir ama büyük sayıda üretilen dizgeler için orta tümleşiklik düzeyli devreler çok daha üstündür. Yalnızca yeniden programlanmasıyla birçok

farklı işlevlerin gerçekleştirilebilmesi de salt oku bellek kullanımında olumlu yönlerden biridir. Böylelikle temel devre yapısı değişmeden, yeni bir salt oku bellek yerleştirilmesiyle, örneğin, ardışık bir kodlama devresinin ürettiği kod değiştirilebilir. Bu özelliğe "yüklenebilir kimlik" adı verilmektedir. Kavramın özellikle rasgele erişimli bellek kullanımında çok önemli olduğuna, "kimliğin" sürekli değişebileceğine dikkat edilmelidir. Bu düşünce çizgisinin doğal uzantısı Aleksander [10] tarafından tanımlanan uyumlu *(adaptive)* mantık dizgeleridir.

3. SAYISAL DİZGELERİN MODÜLLERLE GERÇEKLEŞTİRİLMELERİ

Sayısal dizgelerin (özellikle bilgisayarların) uygun mantık modüllerine parçalanması, dolayısıyla geniş çapta tümleşim teknolojisinin tüm üstünlüklerinden yararlanılması yönünde günümüze dek birçok çalışma yapıldı. Teknoloji açısından geniş çapta tümleşimli bir mantık modülünün istenen özellikleri;

a. Yüksek devre yoğunluğu, yani çok sayıda geçit,

b. Düzenli yapı ve iç bağlantı örgüsü,

c. Devreden çevreye bağlantı oranının yüksek olması,

d. Kullanımda esneklik,

e. Deneme kolaylığı

olarak sıralanabilir.

örneğin birçok birim, teknoloji için uygun olmakla birlikte, yeterli bir dizge kuramının yokluğu nedeniyle, böyle bir araç için gereken esneklikten yoksun oldu. Bu bölümde dizge gerçekleştirilmesinde kullanılmak üzere önerilen birkaç önemli altdizgeyi gözden geçireceğiz.

Sayısal altdizgeler üzerindeki çalışmaların çoğunluğu bilgisayar tasarımına yöneliktir. Fodraza, Gregg ve Slager [il] in tanıttığı bir dizge, genel olarak paralel veri işlemde *(data processing)* uygulanabilecek dört tane orta çapta tümleşimli yapı öbeği kullanır. Bu soruna bir başka yaklaşım gözesel *(cellular)* diziler kullanmaktır ve bu konuda önemli ölçüde çalışma yapılmıştır [12] . Gözesel diziler konusundaki ilk çalışmalar verili bir dizi içinde herhangi bir anahtarlama işlevi gerçekleştirecek bir algoritma geliştirmek, böylece dizinin evrensel bir anahtarlama öğesi olduğunu göstermek üzerine yoğunlaşmıştı. Daha sonraki çalışmalar, örneğin Kautz[13], Nicoud [14] , Dean [15] , Deverell ve diğerleri [16,17,18] ise özel amaçlı diziler kullan-ımını konu edinmektedir. Sayısal bilgisayarın denetim yapısı ve mikro-programında kullanılmak üzere Jump ve Fritsche [19] programlanan gözesel diziler de tanımladılar. IBM'den Gardner [20]in önerdiği işlevsel bellek de etkin (aktif) gözesel diziler kullanarak geniş çapta tümleşim uygulamasına getirilen benzer bir çözümdür. Dizgenin temeli, "0", "1" ve "tanımlanmamış" olmak üzere üç konumlu, yazılır-okunur gözelerden oluşan bir bellek dizisidir. Bu durumda salt oku bellek bir gözesel dizi olarak belirlenebilir. İşlevsel belleğin önde gelen üstünlüğü yasal *(canonical)* terimlerin saklanmamasıyla sağlanan azaltılmış bellek boyutudur.

Yukarda sözü edilen mantık modülleri tasarımcı tarafından piyasada bulunamaz, yalnızca dizge tasarımında günümüz eğilimlerini belirtmek amacıyla bunlara değinilmiştir. Ayrıca bu dizgelerin çoğunluğu bilgisayar endüstrisinden kaynaklanmış özel sorunların (örneğin denetim biriminin tasarımı) çözümüne yönelmiştir. Günümüzde bunlardan çok daha aşağı bir dizge düzeyinde olan salt oku bellek ve çoklayıcı dışında hiçbir genel amaçlı altdizge birimi bulunamaz.

Modüller ile tasarımda daha genelleştirilmiş tek pratik yaklaşım, özel "yazmaç aktarım" *(register. transfer)* birimleri kullanan PDP 16 kavramıdır [21] . Yazmaç aktarım dizgesi, temel yazmaç işlemlerini ve geçişlerini, verinin ve denetim imlerinin saklanmasını, çevre araçlarıyla ilişkileri sağlayan 20 kadar baskı devre biriminden oluşur. Kullanım sırasında tasarımcı uygun yazmaç aktarım birimlerinden oluşan bir veri yapısı kurar ve akış çizimi tekniği kullanarak gerekli denetim ve işlem algoritmalarını belirler. Böylelikle elde edilen akış çizgesi doğrudan doğruya denetim mantık devresinin kablajını verir.

Özet olarak, günümüzde, sayısal dizge tasarımında temel geçitlerin yerini alabilecek altdizge düzeyinde evrensel bir birim yoktur. Walker [22] birçok temel gereksinmeyi karşılayabilecek bir düğüm mantık *(node logic)* dizgesi önermişse de, hâlâ kuramsal bir tasarım yöntemi geliştirilememiştir.

Bu durumun nedeni nedir? Evrensel bir dizge biriminin tasaıunlanması gerçekten olanaklı mıdır? Bu soruların yanıtı,biçimlenmiş bir dizge kuramının olmadığı gerçeğindedir. özel bir birimin geliştirilmesi ve kullanılması, uygun bir tasarım yönteminin üretilmesi için kuramsal bir temel oluşturulmalıdır. Bu kuramın başlangıç noktası da değerlendirmeye ve tasarıma yönelecek, dizge belirlenmesi ve tanımlartmasıdır.

4. SAYISAL DİZGELERİN SUNULUŞU

"Bilgisayarlar yardımıyla tasarım" *(computer aided design)* çevresi içinde mantık yapılarının sunulması için en iyi yöntem aşağıda sıralanan nitelikleri taşımalıdır:

a. Hem donanım hem yazılım işlemlerini sunabilmeli,

b. Tasarımcı tarafından kolayca özümlenmeli,değiştirilmesi ve düzeltilmesi kolay olmalı,

c. Tasarımcılar, uygulayıcılar ve kullanıcılar arasında yanlış anlamaya yol açmayacak kesin tanımlar ile iletişim sağlanmalı,

d. Hem donanımda hem yazılımda tanıtmadan gerçekleştirmeye dek ilerlenebilmeli ve uygulamanın en uygun yöntemine açıklık getirmeli,

e. Eşzamanlı olan ve olmayan koşut (paralel) işlem yürütülebilmeli,

f. Mantık geçiti düzeyinde değil, çok değişkenli dizgelerde, yani makro yada altcizge düzeyinde bilgi akışını sunabilmeli,

g. Adım adım benzeşimden çok dizgenin şekilsel davranış çözümlemesine yönelmeli,

h. Önceden tanımlanan altdizgeleri de içine alan öbek yapılı, öncelikli bir yöntem olmalıdır.

Yazında sayısal dizgelerin tanıtılması ve tasarımı için kullanılan çeşitli teknikleri üç genel alanda toplayabiliriz:

a. işlevsel tanıtıcı programlama dillerij Yazmaç aktarım dilleri, benzeşim *(simulation)* dilleri (APL gibi);

b. Algoritma içeren teknikler; durum çizelgeleri, düzenli deyimler *(regular expressions)* akış çizimleri gibi sonlu durum *(finite state)* makine kuramına dayananlar;

c. Yönlü çizge *(graph)* teknikleri; durum çizimleri, Petri ağları kullanımı gibi yöntemler.

Dizgenin belirlenmesi ve tasarımı için gereken, yukarda sıralanan niteliklerden birçoğu bu tekniklerin birinde yada diğerinde vardır ama genellikle bugüne dek tanıtılan yöntemler kusursuz sayılamaz, örneğin işlemlerden kaynaklanan programlama dilleri, sağlam bir matematiksel yapıya sahip olamamanın sakıncalarını taşırlar. Bu nedenle de her tasarım yada değerlendirme işlemi, yorucu bir şekilde mantık koşullarının tüm olanaklarının gözden geçirilmesine dayanır. Daha da ötede bellek boyutunun büyümesini zorunlu kılar. Tanıtılan bütün yöntemler içinde iki tanesi, "yazmaç aktarım" ve "Petri ağları" sınırlı bir başarı sağlamıştır. Bu nedenle biraz daha ayrıntılı ele alınmaları gereklidir.

4.1. Yazmaç Aktarım Dilleri [23-25]

Bilgisayar dizgelerinin tasarımında kullanılan sezgisel tasarım yöntemleri genellikle önceden tanımlanmış bir yazmaç *(register)* yapısı çevresinde odaklaşır. Daha sonra makine dilindeki komutların uygulanması, mikro-program adı verilen, yazmaçlar arasında verilerin geçişini ve işlemleri denetleyen mikro düzeyinde bir dizi yardımıyla olur. Yazmaç yapısı ve bunlar arasındaki mantık işlemlerini belirleyen bir yazmaç aktarım dili kullanılarak bu sezgisel işlemler yarı özdevimsel hale getirildi. Bu tekniği kullanarak özel bir makine komutları topluluğu, yazmaç aktarım dili içinde işlemlere çevrilebilir, örneğin;

$$|C| : T \leftarrow R + S \qquad (2)$$

C = 1 mantık değerindeyken R ve S yazmaçlarının içindeki sayıların toplanıp sonucun T yazmacı içine yerleştirildiğini gösterir. Adı ve boyutu programın başında verilen yazmaçların içindekiler n bileşkeli bir yektör olarak işlem görür. İçindeki öğeler de altsimge *(subscript)* kullanılarak belirtilebilir:

$$T_{(5)} \leftarrow R(3) \; © \; S_{(2)}$$

Çeşitli mantık ve aritmetik işlemleri örneğin +, ", © vb. bu dil içinde yer alabilir. Temelde yazmaçlar arası koşut veri yolları ve işlemlerin tanımlanmasında yazmaç aktarım dilleri başarıyla kullanılır. Denetim işlemleri ise ya eşzamanlı *(synchronous)* dizgelerde program cümlelerinin yazılışındaki sıralamaya uygun olarak yada çeşitli koşul ilişkileri kullanılarak gerçekleştirilir. Örneğin yukardaki (2) denkleminde |c| yalnızca bir sjiat girişi ($|C_1|$, $|C_2 1$, $|C_3|$ vb.) yada $|\bar{A}_1 ' A_3 * \bar{A} 7 * C_5 = 1|$ gibi birleşik bir koşul olabilir.

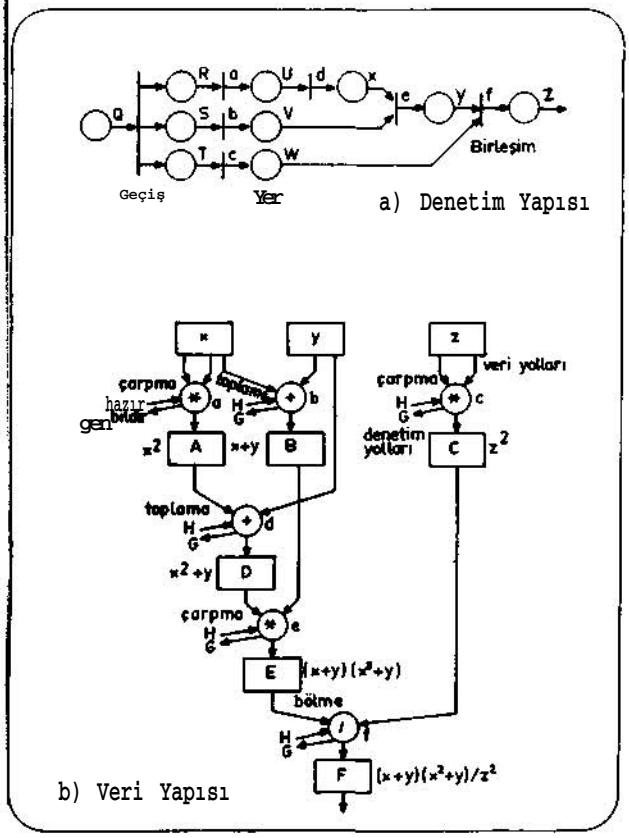
Mikro-programlar (yada genel olarak mantık dizgeleri) yazmaç aktarım dillerinin cümle yapısı içinde değerlendirildikten sonra, dizgenin mantık geçitleriyle gerçekleştirilebileceği Boole denklemlerinden oluşan kümelere dönüştürülebilirler. Ne yazık ki bu yolla elde edilen sonuç, geleneksel tasarım yöntemlerinden çok daha fazla sayıda geçit gerektiren verimsiz bir dizgedir. Yazmaç aktarım dilleri dizge benzeşiminde ve değerlendirilmesinde de kullanılabilir.

Özellikle IBM tarafından geniş ölçüde kullanılan bir dil Iverson gösterimine dayanan APL dilidir [26,27]. Iverson gösterimi temelde vektörler (yazmaç) ve diziler (bellek matrisleri) ile karmaşık ardışık mantık işlemlerini ve yazmaçlararası aktarımları mikro-program cümlelerine dönüştüren genel amaçlı algoritma yapılı bir dildir. Iverson tüm bir dizgenin tanımlanmasında [28] ve ALERT dizgesinde bir mantık tasarım dili olarak kullanılmıştır [25] . Bu tasarım dizgesi IBM 1800 bilgisayarının büyük kısmında kullanılmış ama gerekli olandan % 160 daha fazla sayıda geçit gerektirmistir.

Yazmaç aktarım dilinin bir büyük sakıncası tasarunlamayı eşzamanlı denetim ve önceden belirlenen yazmaç yapıları içinde düşünmeye zorlamasıdır. Daha da ötede, yazmaç aktarım dili yaklaşımı yalnızca donanım dizgelerine uygulanabilir. Bu dildeki tanımların, alışılagelmiş anahtar lama *(switching)* kuramı ile değerlendirilebilecek durum çizelgelerine nasıl dönüştürülebileceği Gerace'nin çalışmalarında [29] gösterilmiştir. Stabler de geliştirdiği [30] benzer bir yöntemde, denetim işlemlerini durum çizelgeleri cinsinden belirlemekte, mantık işlemleri yerine yazmaç aktarım dili cümlelerini kullanmakta ve bu ikisi arasındaki dönüşümü (hız ve donanım güçlükleriyle birlikte) göstermektedir.

4.2. Petri Dizgeleri [31-33]

Temelde durum çizimi (state *diagram)* yaklaşımına çok benzeyen Petri ağları dizge belirlenmesi ve gerçekleştirilmesine daha genel bir çözüm getirmektedir. Petri ağları bazı düğümlerden ve sayısal dizgedeki denetim yapısını yansıtan yönlendirilmiş çizgelerden oluşur. Şekil 4a'da gösterildiği gibi, Petri ağında iki tür düğüm vardır: Çember ile gösterilen "yerler", ve çubuklarla gösterilen "geçişler". Her yönlendirilmiş çizgi, bir yeri geçişe yada geçişi yere bağlar; ilk durumda yer, geçiş yeri olarak adlandırılır, ikincisinde ise yere, geçişin çıkış yeri adı verilir. Koşut yada aynı anda olan işlemlerin Petri ağında kolaylıkla gösterilebildiğine dikkat edilmelidir.



Şekil 4.

fetri ağı üzerindeki çeşitli yollar (ilk olarak |olt tarafından önerilen) işaret olarak belirti *ttoken)* kullanan bir oyuna benzetilebilir. Bir i-aretten diğerine ağ üzerindeki ilerleme aşağıda-ii kurallara uyarak sağlanır:

w Tüm giriş yerlerinde bir belirti varsa bir ge-çiş sağlanır.

'). Olanak verilen herhangi bir geçiş başlatıla-I bilir.

\. Geçişin başlatılmasıyla belirtiler giriş yerle-! rinden çıkış yerlerine iletilir.

j'etri ağlarının, sayısal dizgelerin denetim yapı-larına uygulanmasında, eşzamanlı olmayan bir ça-i.ışma varsayılır ve veri yapıları ile denetim yapı-ları arasında gidip gelen denetim imleri gönderi-ür. Veri yapısındaki (toplama, çarpma gibi) işlem-eri başlatmak için denetim yapısından bir "hazır" .mi gönderilir; işlemi gerçekleştiren birim ise şi bitince bunu bildirir ("El sıkma" ilkesi), (öylece Petri ağı için aktarma başlatılma süreci [e aşağıdaki temel adımları sağlayacak şekilde jeliştirilir:

ı. Giriş yerlerinden belirtileri al.
ı. İşlem birimine "hazır" imi gönder.
:. Gelecek yanıtı bekle.
1. Çıkış yerlerine belirti koy.

;ekil 4b'de $(x + y)(x^2 + y) / z^2$ gösterimini ulmak için gereken denetim ve veri yapıları gös-:erilmiştir. Q yerine bir belirti yerleştirmekle hazır imine eşdeğer) işlem başlar, x,y ve z yaz-maçları yüklenince veri yapısı bunu bildirir. Daha sonra yukarda belirtilen aktarma başlatma kuralları uygulanarak a, b, c aktarmaları sağlanır ve bu işlemler z de bir belirti olmasıyla belir-tilen, görevin tamamlanmasına dek sürer.

Bir denetim devresinin her bir işlemi yerine (ör-neğin belirti taşıyan bir yer, dallar, birleşim-ler) bir donanım devresi yerleştirilebilir. Bun-ların kullanımıyla bir Petri ağından denetim man-tığına dolaysız geçiş kolayca sağlanır.

Case Western Reserve Üniversitesinde geliştirilen LOGOS dizgesi [34] için yürütülen donanım/yazılım belirleme ve gerçekleştirme çalışmalarında bu kav-ramlar kullanılmaktadır. LOGOS dizgesi, sayısal dizgeleri (hem donanım, hem de yazılım işlemleri-ni) , biri veri akışı diğeri denetim olarak iki yönlendirilmiş çizge *(directed graph)* olarak ele alır. Veri çizgesi, veri yapısının algoritmasını ve üzerindeki dönüşümleri tanımlarken, ilgili denetim çizgesi de dönüşümlerin zamanlamasını ya-par ve denetim akışını tanımlar, öncelikli birim-ler halinde bir yaklaşım izlenir ve sonuçtaki dizge daha alt düzeylerde belirlenen öğelerden oluşur. Denetim çizgeleri donanım olarak kolayca gerçekleştirilebilir ama veri çizgesinin nasıl gerçekleştirileceği şimdilik kesin değildir. Çözüm-leme algoritmaları ve benzeşim işlemlerinin elde edilmesi konularında da önemli çalışmalar yürü-tülmüştür.

Kusursuz bir tasarım yolu geliştirilinceye dek çözülmesi gereken birçok sorun vardır. Yazmaç ak-tarım tekniği, yazmaç yapılı dizgelere uygundur, fakat özellikle donanıma yöneliktir ve çözümleme (benzeşim karşıtı olarak) algoritmaları henüz ge-liştirilmemiştir. Bu yazıda tanıtılan yöntemler içinde, Fetri ağlarına dayanan LOGOS dizgesi en umut verici yöntem olarak görülmektedir.

5. SONUÇLAR

Son on yıl içinde tümleşik devre teknolojisindeki hızlı gelişmeler, temel mantık geçitlerini kulla-nan tekniklerin hızla yetersiz kalmasına yol aç-tı. Sayısal dizgelerin orta çapta tümlesimli ve geniş çapta tümleşimli birimler kullanımı ile, karmaşık mantık işlevlerini gerçekleştirecek bi-çimde kurulması güncel bir konu durumuna geldi. Buna ek olarak mikro işlemci yongaları *(micro-processor chips)* gibi altdizge öğelerinin piyasa-ya sürülmesi de dizge tasarımının genel karmaşık-lığını artırmaktadır.

Ne yazık ki kuram, eylemin hızına ulaşamadı ve şim-di, altdizge düzeyinde uygun bir tasarım kuramı olmamasının neden olduğu büyük bir engelin önün-deyiz. Yarının mantık dizgeleri yalnızca sezgisel yöntemlerle tasarımlanmayacak denli karmaşık ola-caktır.

Geleceğin karmaşık dizgelerinin tasarımında bil-gisayar yardımlı tekniklerin önemli rol oynayaca-ğı kesindir. Ne var ki günümüzde mantık dizgeleri-nin tasarımı için düşüncel *(ideal)* bir belirleme, değerlendirme ve tasarım yöntemi yoktur. Sayısal dizgelerin gelecekteki gelişimi için uygun bir tasarım yönteminin elden geldiğince çabuk geliş-tirilmesi gereklidir.

KAYNAKLAR:

[1] *Breuer, M.A.;* "Recent Developments in the Design and Analysis of Digital Systems", Proc.IEEE, Cilt 60, s.12-27, 1972.

[2] Breuer, *M.A. (ed);* Design Automation of Digital Systems, Cilt 1: Theory and Technique, N.J.: Prentice-Hall, 1972.

[3] *Lewin, D.W., Purslovr, E.* ve *Bennetts, R.G.;* Computer Assisted Logic Design -the CALD System : Conf. on Computer Aided Design, IEE Conf. Pub.No.86, s.343-51, 1972.

[4] *Anderson, J.L.;* "Multiplexers Double as Logic Circuits", Electronics, Cilt 42, No 22, s.100-5, 27 Ekim 1969.

[5] *Yatı, S.S.* ve *Tang, C.K.;* "Universal Logic Modules and Their Applications", IEEE Trans. on Computers, Cilt C-19, S.141-9, 1970.

[6] *Fletcher, fi.I.* ve *Despain, A.M.;* "Simplify Combinational Logic. Circuits", Electronics Design, sayı 13, s.72-73, Haziran 1971, Sayı 14, s.70-72, Temmuz 1971.

[*İ] Kramae, F.;* "Standart Read Only Memories Simplify Complex Logic Design", Electronics, Cilt 43, Sayı 1, s.88-95, 1 Ocak 1970.

[8J *Hartmanis, J.* ve *Stearns, R.E.;* Algebraic Structure Theory of Seqüential Machines, N.J: Prentice Hail, 1966.

[9] *Ashenhurst, R.L.;* "The Decomposition of Svitching Functions", Annals of Computation Laboratory of Harvard University, Cilt 29, s.74-116, 1959.

[îo] *Aleksander, I., Stonham, T.J.* ve *tiilson M.T.D.',* "Adaptive Logic for Artificially Intelligent Systems", The Radio and Electronic Engineer, Cilt 44, s.39-44, Ocak 1974.

[il] *Podraza, G.V., Gregg, R.S.* ve *Slager, J.R.;* "Efficient m.s.i. Partitioning for a Digital Computer", IEEE Trans. on Computers, Cilt C-13, s.1020-8, 1970.

[12] *Hirmick, R.C.;* "A Survey of Microcellular Research", J.Ass.Comput.Mach., s.203-41, Nisan 1967.

[13] Jtautz, *W.H.;* "Cellular-Logic-in-Memory Arrays", IEEE Trans. on Computers, Cilt C-12, s.719-27, 1969.

[14] *Nicoud, J.D.;* "Iterative Arrays for Radix Conversion", IEEE Trans. on Computers, Cilt C-20, s.1479-89, 1971.

[15] Dean, *K.J.;* "Conversion Betveen Binary Code and Some Binary Decimal Codes", The Radio and Electronic Engineer, Cilt 35, s.49-53, 1958.

[16] Deverell, T.; "The Design of Cellular Arrays for Arithmetic", The Radio and Electronic Engineer, Cilt 44, s.21-26, Ocak 1974.

[17] *Spencer, A.J.* ve *Buckle, P.M.;* "Programmable Shift Registers for Numerical Multiplication", The Radio and Electronic Engineer, Cilt 44, s.33-8, Ocak 1974.

[18] *Jayashri, T.* ve *Basu, D.;* "Cellular Array for Multiplication of Signed Binary Numbers" The Radio and Electronic Engineer, Cilt 44, s.18-20, Ocak 1974.

[19j *Jump, J.R.* ve *Fritsche, D.R.* ; "Microprogramm Arrays", IEEE Trans. on Computers, Cilt C-21 s.974-84, 1972.

[20] *Gardner, P.L.;* "Functional Memory and its Microprogramming Implications", IEEE Trans. on Computers, Cilt C-20, s.764-75, 1971.

[21] *Bell, C.G. , Eggert, J.L. , Granson, J.* ve *trilliams, P.;* "The Description and Use of Register Transfer Modules (RTMS)", IEEE Tranj on Computers, Cilt C-22, s.495-500, 1972.

[22] *Halker, B.S. ;* "The Design of Sequential Logi| Circuits", The Radio and Electronic Engineerj Cilt 43, s.45-9, Ocak 1974.

[23] *Schoor, H.;* "Computer Aided Digital Design and Analysis Using a Register Transfer Langu age", IEEE Trans. on Electronic Computers, Cilt EC-13, s.730-57, 1964.

[24] *Duley, J.R.* ve *Dietmeger, D.L.* ; "A Digital System Design Language (DDL)", IEEE Trans. on Computers, Cilt C-17, s.850-61, 1968.

[25] *Friedman, T.D.* ve *Yang, S.C.;* "Methods üsed an Automatic Logic Design Generator (ALERT)" IEEE Trans. on Computers, Cilt C-17, s.1044-61, 1968.

[26] *iversem, K.E.;* A Programming Language, New York:Wiley, 1962.

[27] *Mili, F.J.* ve *Peterson, G.R.;* Digital Systeı^ Hardvare Organization and Design, Nev York: Wiley, 1973.

[28] *Falkoff, A.D., iver son, K.E.* ve *Sussenguth, E.H.;* "Formal Description of System / 360", IBM Systems J., Cilt 3, s.198-262, 1964.

[29] *Gerace, G.B.;* "Digital System Design Automation - a Method for Designing a Digital System as a Sequential Netvork System", IEE Trans. on Computers, Cilt C-17, s.1044-61, 196Ö.

[30] *Stabler, F.P.;* "Micro-program Transformatiori IEEE Trans. on Computers, Cilt C-19, s.908-16, 1970.

[3İ] *Holt, A.W.;* "Information System Theory Project", Applied Data Research, Tech.Report No RADC-TR-68-305, 1968.

[32] *Patel, S.* ve *Dennis, J.B.;* Tke Description and Realization of Digital Systems, Proc.Coi Conf. 72, Sixth Annual IEEE Computer Societ^ Int.Conf., s.223-6, 1972.

[33] *Holt, A.W. ;* "Introduction to Occurance Systems", in Associative Information Techniques, (Jacks, E.J. ed.), Nev YorktElseviei 1971.

[34] *Heath, F.G.;* "The LOGOS System", Conf. on Computer Aided Design, IEE Conf. Pub.No 86, s.225-30, 1972.