DENEY 4

Lojik Devre Laboratuvarı

4.Deney Ek Dokümanları

EK BİLGİ

PLPL (Programlanabilir Lojik Programlama Dili)

PLPL, kullanıcının lojik fonksiyonları ve durum makinelerini yüksek düzeyde tanımlamasına olanak veren bir programlanabilir geliştirme paketidir. Bu paket geçekte 5 ayrı programdan oluşur.

-- Programlanabilir lojik derleyici(PLC), tasarım dosyasında tanımlanan fonksiyonları lojik denklemlere dönüştürür ve bir ara dosyada saklar.

-- Lojik optimize edici(OPTIMIZE), ara dosyadaki denklemler üzerinde indirgeme işlemlerini yapar.

-- JEDEC-standardı sigorta haritası üreticisi(JM), ara/optimize dosyadaki lojik denklemlerden sigorta haritasını oluşturur ve bir dosyaya yazar. Bu dosya daha sonra PLD elemanını programlamada kullanılacaktır.

-- Test vektör üreticisi(TESTV), tasarım dosyasında kullanıcı tarafından belirlenen fonksiyon tablosundan, JEDEC-standardı test vektörlerini üretir. Bu vektörler, simülasyon programı tarafından elemanı modellemede kullanılır.

-- Fonksiyon simülasyon programı(SIM) ara/optimize dosyadaki lojik denklemleri, kullanıcının tanımladığı test vektörlerini kullanarak, sınar.

PLPL paketinde desteklenen her eleman için bir veri-tabanı bulunur. Her veri-tabanı dosyasının adı, kullanılan elemanın isminin başına P harfinin eklenmesiyle oluşmuştur. örneğin, AmPAL22V10 için kullanılan dosyanın adı P22V10 dur.

PLPL Kullanarak PLD Tasarımı Yapmak:

önce PLD'yi programlayacak olan lojik fonksiyonları içeren bir tasarım dosyası yaratılır. Daha sonra PLPL.EXE programı çağrılır. Bu programla ana menüden istenen işlem seçilir.

Tasarım aşamasında kullanılan seçenekler:

C (Lojik Derleyici): Giriş dosyası <dosya adı>.pld
Çıkış dosyası <dosya adı>.int

O (Optimize Edici): Giriş dosyası <dosya adı>.int
Çıkış dosyası <dosya adı>.opt

J (JEDEC çıkışı/Denklemlerin Listesi): Giriş dosyası <dosya adı>.int
veya <dosya adı>.opt
Çıkış dosyası <dosya adı>.jed

T (Test Vektörü Üreticisi): Giriş dosyası <dosya adı>.pld
Çıkış dosyası <dosya adı>.tst

S (Simülasyon Programı): Giriş dosyaları #1 <dosya adı>.jed
#2 <dosya adı>.tst
Çıkış dosyası <dosya adı>.sim

Sistem desteği sağlayan komutlar :

R : Menüden çıkmadan işletim sistemine ait komutları yürütme olanağı sağlar.

H : Seçenekler hakkında kullanıcıya bilgi verir.

E : Menüden çıkış komutu.

PLPL Tasarım Dosyası:

PLPL tasarım dosyasının, kontrol karakterleri içermeyen bir ASCII dosya olması gerekmektedir. Bu dosya herhangi bir editör kullanılarak oluşturulabilir. Tasarım dosyası şu bölümleri içerir: tasarım ismi, başlık, ve lojik fonksiyon tanımlamaları.

Tasarım İsmi: Bu bölüm DEVICE kelimesini izleyen tasarım ismi ve parantez içinde yazılmış eleman isminden oluşur.

Başlık: Başlık bölümü iki alt bölümden oluşur: Bacak tanımlamaları ve tanımlama bölümü.

Bacak tanımlama bölümünde tasarımcı her bacağa isim verir, ilgili bacağı giriş ya da çıkış olarak tanımlar(yalnız iki yönlü kullanılabilen bacaklar için), sıfır etkin olup olmadığını belirtir. Bacak tanımlama bölümünün bittiği bir noktalı virgül ile belirtilmelidir.

Tanımlama bölümünde kullanıcı isterse değişkenler tanımlayabilir. Bir değişkene herhangi bir tamsayı değer ya da sık kullanılan bir lojik denklem atanabilir. Vektör kullanım imkanı yoktur. Bu bölüm DEFINE kelimesiyle başlar, noktalı virgül ile sonlandırılır.

Lojik Fonksiyon Tanımlamaları: Bu tanımlamalar BEGIN-END bloğu içinde yer almalıdır. Girişlerin lojik fonksiyonlara tabi tutulmuş ifadeleri çıkışlara atanabileceği gibi(lojik denklemler), doğrudan çıkışların koşul altında almaları gereken değerler de belirtilebilir(yüksek düzeyde tanımlama).

Lojik denklemlerde kullanılan lojik fonksiyonlar sadece VE(*), VEYA(+), DEGIL(/), ve DAR VEYA(%) olabilir. Her denklem bir noktalı virgül ile diğerinden ayrılır. Vektör halinde atamalar mümkündür.

Yüksek düzeyde tanımlamada ise sadece iki yapı vardır: IF-THEN-ELSE ve CASE. Bu yapılar herhengi bir programlama dilindeki gibi kullanılırlar. IF kelimesini izleyen koşul basit bir lojik test işlemi olabileceği gibi karmaşık lojik ifadelerden de oluşabilir. CASE komutu sayesinde

ise bir vektörün farklı değerleri için farklı atamalar yapılabilir. CASE kelimesinin ardından BEGIN-END bloğu içinde vektörün alabileceği değerlere göre gerekli atamalar yapılır. Bu atamalar da BEGIN-END blokları içinde bulunmalıdırlar. Cümleler birbirlerinden noktalı virgülle ayrılırlar.

özel Fonksiyonlar: Çıkış bacakları için bazı özel fonksiyonlar tanımlanmıştır. Bulardan üçü RESET, PRESET ve ENABLE komutlarıdır. RESET fonksiyonu ilgili çıkışı sıfırlarken, PRESET birleme işlemini yapar. ENABLE fonksiyonu ise kendisine parametre olarak verilen çıkış ya da vektör elemanlarının etkin olup olmamasını sağlar. Bu fonksiyona sabit bir değer yerine herhangi bir giriş bacağı da atanabilir.

örneğin:

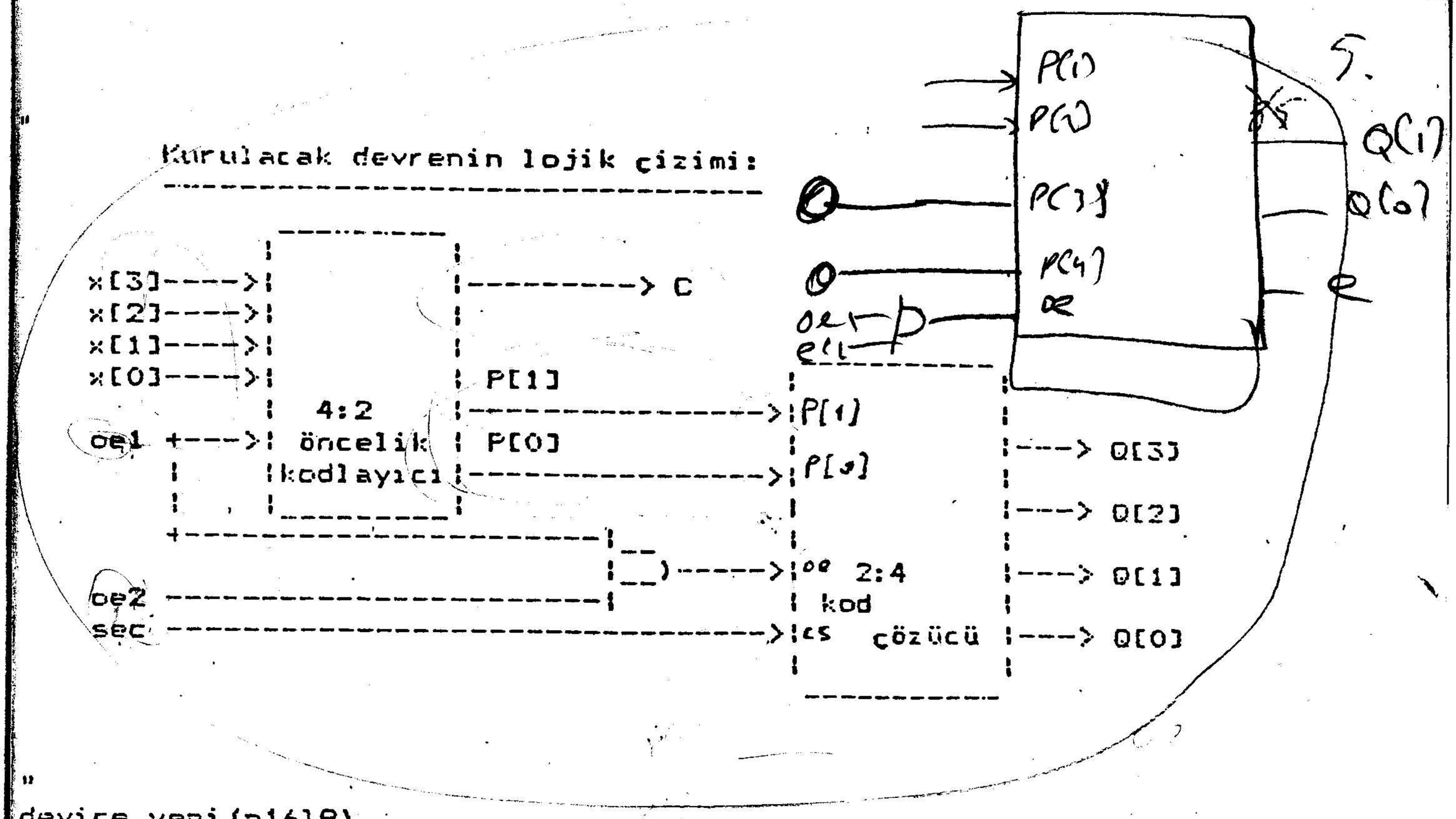
```
ENABLE(a[3:0]) = #b1101;    "a çıkış vektörünün birinci bacağı
                             . hariç tüm bacakları  etkin      "

 ENABLE(b[2:0]) = c;         "b çıkış vektörü, c girişi bir
                              .ise etkin                       "
```

Test Vektörlerinin Oluşturulması: Tasarım aşamasında verilen fonksiyonların doğruluğunu sınamak amacıyla test vektörlerinden yararlanılır. Bu vektörler çeşitli girişler için çıkış bacaklarının alması gereken değerleri gösterir. Tasarım dosyasının sonunda TEST-VECTORS terimi bu bölümün başını işaret eder. Bunu bacak sınıflamaları izler. Dört tip bacak tanımlanmıştır: IN, OUT, I_O ve BREG (sırasıyla giriş, çıkış, giriş/çıkış ve iç saklayıcılı). Daha sonra BEGIN-END bloğu içinde bacakların alması gereken değerler sınıflamadaki sırayla verilir. Kullanılan bazı bacak değerleri şunlardır:

```
0  - giriş lojik sıfır
1  - giriş lojik bir
L  - test çıkışı sıfır
H  - test çıkışı bir
Z  - test giriş/çıkışı yüksek empedans
C  - pozitif saat darbesi (yükselen kenar)
K  - negatif saat darbesi (düşen kenar)
P  - saklayıcılara ön yükleme
B  - iç saklayıcılara ön yükleme
N  - besleme bacakları veya test edilmeyen çıkışlar
X  - test edilmeyen çıkışlar
```

Kurulacak devrenin lojik çizimi:




```
"
device yeni(p1618)
Pin
oe1= 1 (combinatorial input)                    " öncelik Kodlayıcısına ait
                                                  çıkış izni "

x[3]= 18 (combinatorial input)                  " iki yönlü kullanılabilen bir
                                                  hat giriş olarak kullanılacak

x[2:0]= 3:5 (combinatorial input)               " Saf giriş uçları "

C= 19 (output active_low combinatorial)          " Saf çıkış uçları "
P[1:0]= 17:16 (IO active_low combinatorial);" öncelik Kodlayıcısının
                                                  çıkışları daha sonra kod
                                                  çözücüye giriş olarak
                                                  verilmeli "

Begin
     enable(C) = #b1;                           " C ucu her zaman çıkış olarak
                                                  kullanılacak "
     enable(P[1:0]) = /oe1;"                    " öncelik Kodlayıcısı çıkışları
                                                  izin girişinden kontrol
                                                  ediliyor "
     enable(x[3]) =#b0;                         " x[3] hattı her zaman giriş
                                                  olarak kullanılacak "


     /C = /x[0]*/x[1]*/x[2]*/x[3];              " Girişlerden en az bir tanesi
                                                  lojik bir seviyesinde ise C
                                                  çıkışı etkin olur(etkin sıfır)

     /P[1] = x[3] +
```

6.

```
            x[2];

    /P[0] = x[3] +
            x[1]*/x[2];


End.

test_vectors
   in oe1;
   in x[3:0];
   out C;
   IO P[1:0];


begin
"
 o
 e  x  x  x  x     P  P
 1  3  2  1  0  C  1  0
 ------------------------"
 1  X  X  X  X  X  Z  Z    ; "1 : Cıkıs izni yok "
 0  0  0  0  0  H  L  L    ; "2 : Hiç bir giris etkin degil "
 0  0  0  0  1  L  L  L    ; "3 : Sadece x[0] etkin "
 0  0  0  1  X  L  L  H    ; "4 : x[3] ve x[2] etkin degil; x[1] etkin "
 0  0  1  X  X  L  H  L    ; "5 : x[3] etkin degil; x[2] etkin "
 0  0  1  X  X  L  H  L    ; "6 : x[3] etkin "
 0  1  X  X  X  L  H  H    ; "7 : Cıkıs izni yok; herhangi bir giris etkin
 1  0  0  1  0  L  Z  Z    ; "8 : Cıkıs izni yok; hic bir giris etkin degi
 1  0  0  0  0  H  Z  Z    ;
end.
```

```
PLD Programmer Codes for [p1618]
- [  data_io] = [amd_9717]
- [stag_zl_30] = [amd_9029]
          PLD [p1618] Device Map for
          Design [yeni]*
QF2048*
QP20*

F0*
L0000 1111 1111 1111 1111 1111 1111 1111 1111 *
L0032 1111 1111 1111 0111 1111 1111 1111 1111 *
L0064 1111 1111 0111 1111 1111 1111 1111 1111 *
L0096 1111 0111 1111 1111 1111 1111 1111 1111 *
L0128 1111 1101 1111 1111 1111 1111 1111 1111 *
L0256 0000 0000 0000 0000 0000 0000 0000 0000 *
L0512 1110 1111 1111 1111 1111 1111 1111 1111 *
L0544 1111 1010 1111 1111 1111 1111 1111 1111 *
L0768 1110 1111 1111 1111 1111 1111 1111 1111 *
L0800 1111 1110 1011 1111 1111 1111 1111 1111 *
L0832 1111 0110 1111 1111 1111 1111 1111 1111 *
C25B5*
8C42

Simulating [1_deney.jed] (device file [c:\pldbase\s161B])
        with vectors in file [1_deney.tst]

==> Device Pin #: [0000 0000 0111 1111 1112]
- - - - - - - - - - [1234 5678 9012 3456 7890]
- Applying [  1]: [1XXX XXXX XNXX XXXZ ZXXN] -
  Calculated   => [1XXX XXXX XNXZ ZZZZ ZZHN]
- Applying [  2]: [0X00 0XXX XNXX XXXL LOHN] -
  Calculated   => [0X00 0XXX XNXZ ZZZL LOHN]
- Applying [  3]: [0X00 1XXX XNXX XXXL LOLN] -
  Calculated   => [0X00 1XXX XNXZ ZZZL LOLN]
- Applying [  4]: [0X01 XXXX XNXX XXXH LOLN] -
  Calculated   => [0X01 XXXX XNXZ ZZZH LOLN]
- Applying [  5]: [0X1X XXXX XNXX XXXL HOLN] -
  Calculated   => [0X1X XXXX XNXZ ZZZL HOLN]
- Applying [  6]: [0XXX XXXX XNXX XXXH H1LN] -
  Calculated   => [0XXX XXXX XNXZ ZZZH H1LN]
- Applying [  7]: [1X01 0XXX XNXX XXXZ ZOLN] -
  Calculated   => [1X01 0XXX XNXZ ZZZZ ZOLN]
- Applying [  8]: [1X00 0XXX XNXX XXXZ ZOHN] -
  Calculated   => [1X00 0XXX XNXZ ZZZZ ZOHN]
Simulation Completed: Errors [  0]
```

PLPL: Programmable Logic Programming Language
Software Version V2.1

PLPL supports all AmPAL devices, including the following:

16L8, 16R8/6/4 *
16P8 *
23S8
22V10 *
20L10 *
22P10, 20RP10/8/6/4
22XP10, 20XRP10/8/6/4
29M16
29MA16

Devices marked with an asterisk are also supported by PALASM 2 software. PLPL will automatically be shipped with the PC version of PALASM 2 software.

PLPL is a programmable logic development package which lets the designer describe logic functions and state machines in a high-level syntax. Various programs in the PLPL package are used to process this design or source file before programming a device.

PLPL is composed of 6 separate programs:

- A programmable logic compiler (PLC) converts the design file into logic equations and stores these in an intermediate file.
- A logic optimizer (OPTIMIZE) logically reduces the Boolean equations in the intermediate file.
- A JEDEC-standard fuse map generator (JM) converts the equations in the intermediate (or optimized) file and writes these into a fuse map file. This fuse map is used to program the device.
- A manual test vector generator (TESTV) generates JEDEC-standard test vectors from a user-specified function table in the design file. These vectors are used by the simulator when modeling the part.
- A functional simulator (SIM) tests the logic equations in the intermediate/optimized file using the user-defined test vectors.
- A PLD program which helps the user define the architecture features on a device (available for PCs).

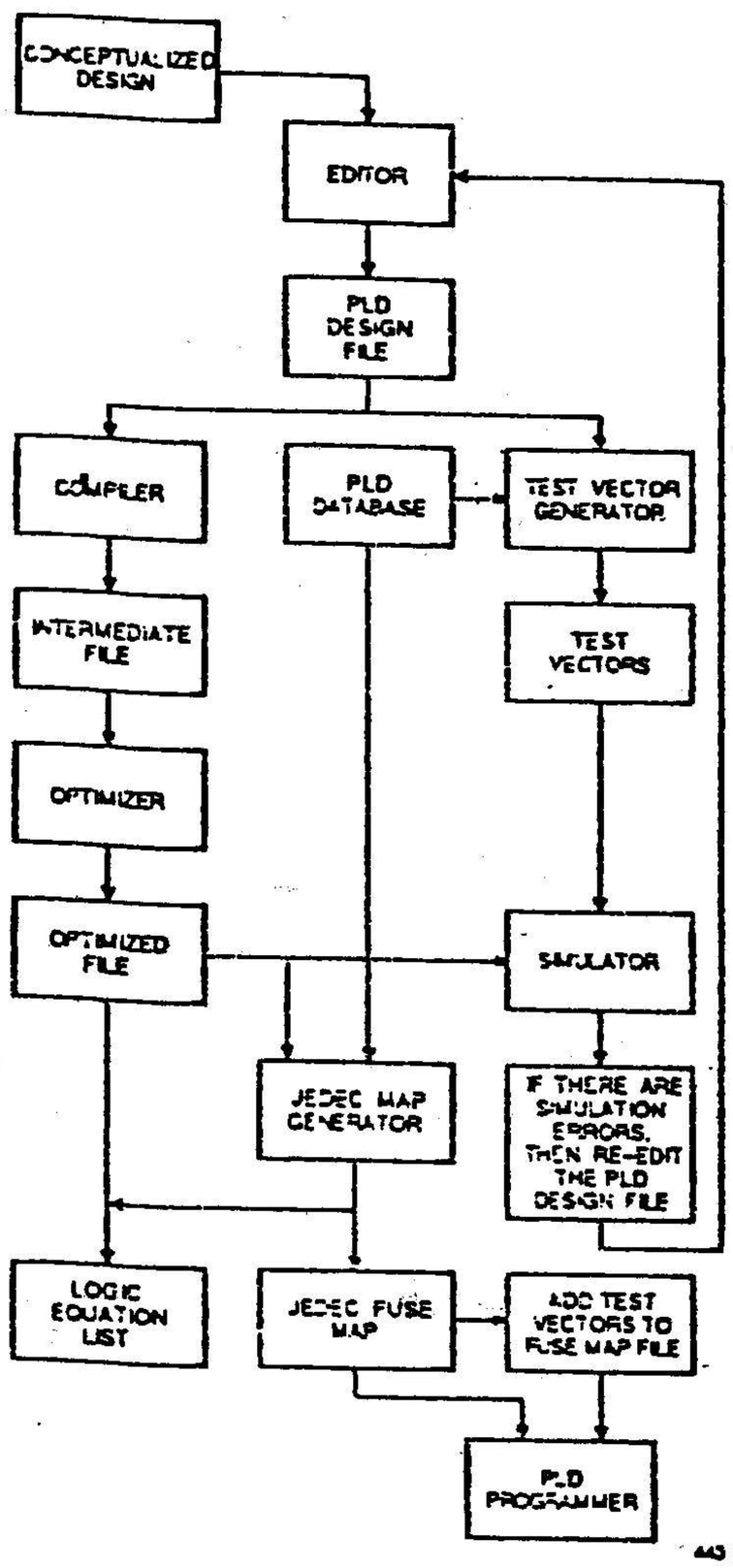
PLPL has a database file for every supported part. Each database file name is composed of the letter P and the numeric designation of the part. For example, the AmPAL22V10 database file is called P22V10.

PLD Design Methodology: Using PLPL

A typical PLPL design cycle contains the following steps:

1. Write a design file specifying the logic functions to be programmed into a PLD using the PLPL language.
2. Use PLC to compile the design file; the output of PLC is called an intermediate file.
3. If required, use the optimizer to reduce the logic equations in the intermediate file produced by PLC.
4. Specify a function table in the PLD design file. Use TESTV to generate JEDEC-format test vectors from the function table.
5. Use JM to produce a JEDEC-standard fuse map from the equations in the intermediate file.
6. Use SIM to simulate the logic model represented by the intermediate file with the test vectors generated by TESTV.
7. If there are any errors, repeat steps (1) to (6).
8. Load the fuse map into a PLD programmer to program the PLD.

PLPL PLD Design Environment



443 01

PLPL: Programmable Logic Programming Language
Software Version V2.1

PLPL supports all AmPAL devices, including the following:

16L8, 16R8/6/4 *
16P8 *
23S8
22V10 *
20L10 *
22P10, 20RP10/8/6/4
22XP10, 20XRP10/8/6/4
29M16
29MA16

Devices marked with an asterisk are also supported by PALASM 2 software. PLPL will automatically be shipped with the PC version of PALASM 2 software.

PLPL is a programmable logic development package which lets the designer describe logic functions and state machines in a high-level syntax. Various programs in the PLPL package are used to process this design or source file before programming a device.

PLPL is composed of 6 separate programs:

- A programmable logic compiler (PLC) converts the design file into logic equations and stores these in an intermediate file.
- A logic optimizer (OPTIMIZE) logically reduces the Boolean equations in the intermediate file.
- A JEDEC-standard fuse map generator (JM) converts the equations in the intermediate (or optimized) file and writes these into a fuse map file. This fuse map is used to program the device.
- A manual test vector generator (TESTV) generates JEDEC-standard test vectors from a user-specified function table in the design file. These vectors are used by the simulator when modeling the part.
- A functional simulator (SIM) tests the logic equations in the intermediate/optimized file using the user-defined test vectors.
- A PLD program which helps the user define the architecture features on a device (available for PCs).

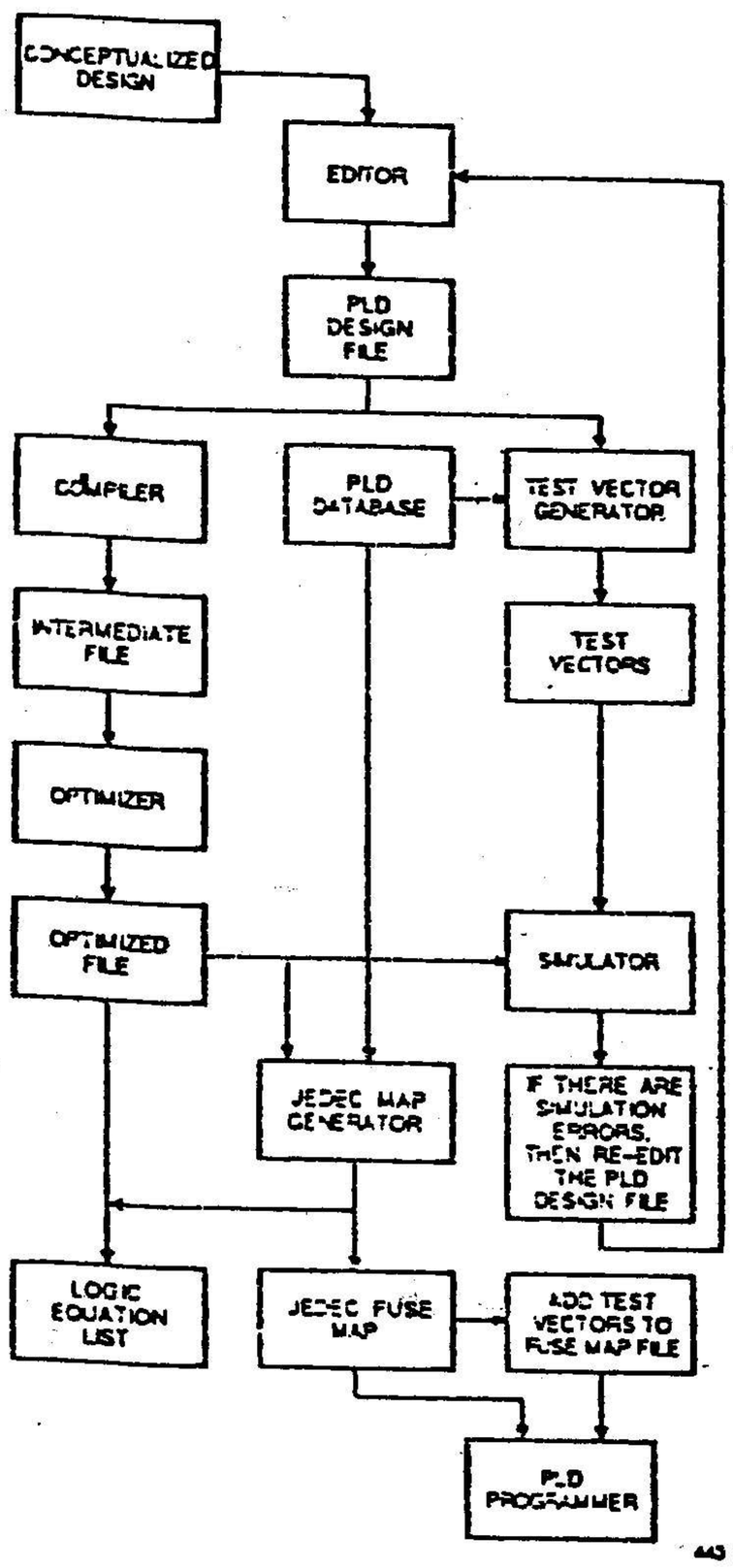
PLPL has a database file for every supported part. Each database file name is composed of the letter P and the numeric designation of the part. For example, the AmPAL22V10 database file is called P22V10.

PLD Design Methodology: Using PLPL

A typical PLPL design cycle contains the following steps:

1. Write a design file specifying the logic functions to be programmed into a PLD using the PLPL language.
2. Use PLC to compile the design file; the output of PLC is called an intermediate file.
3. If required, use the optimizer to reduce the logic equations in the intermediate file produced by PLC.
4. Specify a function table in the PLD design file. Use TESTV to generate JEDEC-format test vectors from the function table.
5. Use JM to produce a JEDEC-standard fuse map from the equations in the intermediate file.
6. Use SIM to simulate the logic model represented by the intermediate file with the test vectors generated by TESTV.
7. If there are any errors, repeat steps (1) to (6).
8. Load the fuse map into a PLD programmer to program the PLD.

PLPL PLD Design Environment



443 01