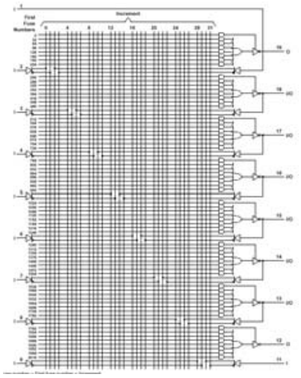
Modeling Example: A PAL

- Goal is to model a standard PAL
  - Both functionality and timing
- Will look at PAL16l8 model, but approach is valid for any standard PAL
- Functionality is defined via a JEDEC file
- Timing is defined via datasheets
- This model was written by Vince Sanders, MSU.

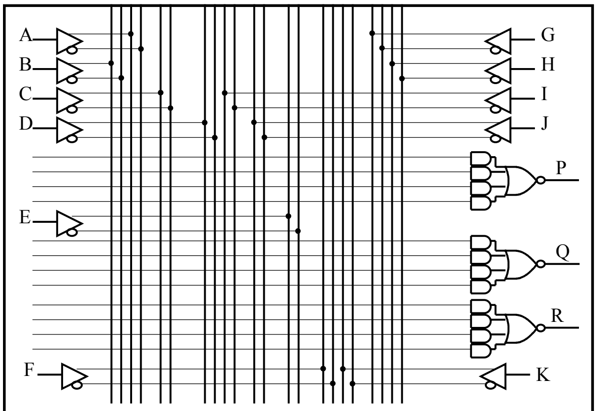


PAL 16L8

10 input only

1 output only

6 input/outputs

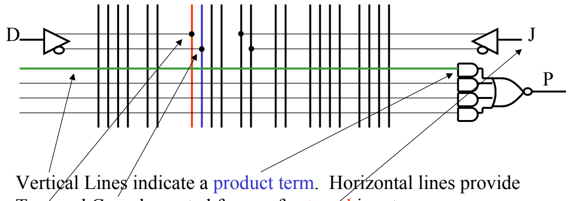


A Sample PAL

Comments on Sample PAL

- 11 inputs, 3 outputs
  - Can implement three functions - functions can share inputs or not share inputs
- Each output implements a SOP equation with Four product terms.
  - Each product term can include complemented or uncomplemented form of an input.

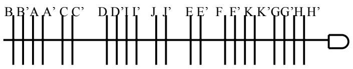
Understanding the Diagram



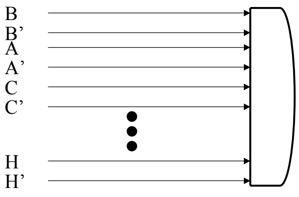
Vertical Lines indicate a product term. Horizontal lines provide True and Complemented forms of external inputs.

Even though a product term looks like it has only one input, it actually has 2 * N inputs, where N is the number of external inputs.

Product Term

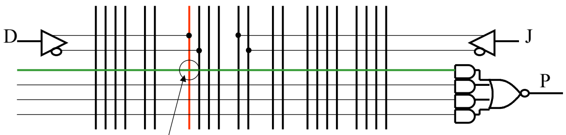


This looks like an AND gate with one input. Is actually:



Only drawn with a single line to save space.

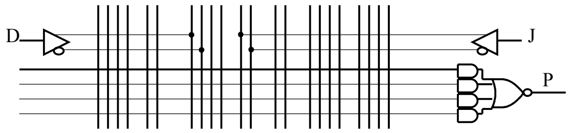
Fuse Points



A cross over of a Vertical input line and a horizontal product term line is a FUSE LOCATION. When the PAL is in its blank or erased state, all FUSES are connected. This means that each product term implements the equation:

(A A' B B' C C'....... KK') will be '0'! This means that the output will be high!

Programming

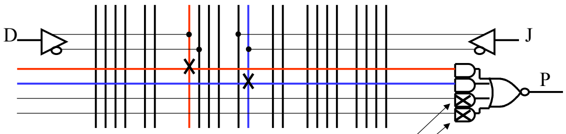


To program, will want to BLOW most of the fuses (break the vertical/horizontal crossover connection). To indicate a logic function, will use a ' X ' over a fuse that I want to KEEP INTACT.

X ← Will mark Intact fuse location.

When a fuse is blown, that product term input acts as a '1' so that the input no longer effects the product term.

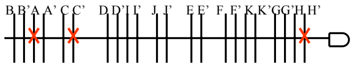
P' = D + J'



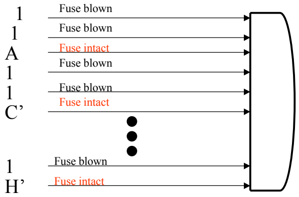
When implementing an equation, sometimes will not want to use all available product terms. If ALL fuses along product term are left intact, then product term value will be '0' and will not affect equation. Mark unused PT's by placing an X over them -- all fuses in that PT row are assumed intact.

Note that P' must be implemented!

Example Product Term AC'H'

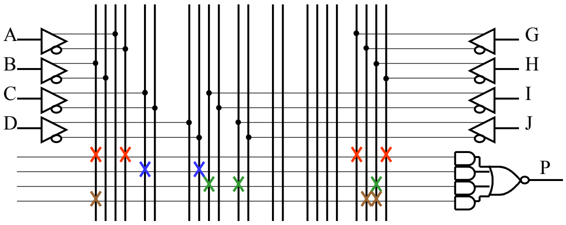


The connections will be:



Actually, fuses are not 'blown' in eraseable PLDs - the connection is broken in a non-destructive way for eraseable PLDs.

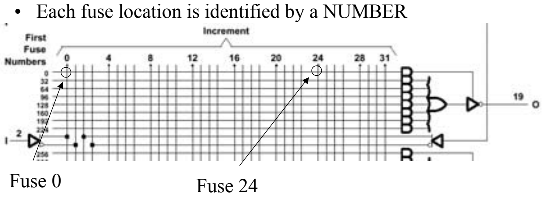
Another Example



P' = A'BGH' + CD' + HIJ + BG'H

JEDEC Files

- Model must read a JEDEC file that defines the programming
- Each fuse location is identified by a NUMBER



Just need to know if the fuse is intact or not. All fuses are intact in unprogrammed state.

Sample Portion of a JEDEC File

```
L00440
11111111111111111111111111111111111111111111
11110111111111111111111111111111111011101110
11111011111111111111111111111111111011101101
11111111111111111111011111111111111111011110
11111111111111111111101111111111111111011101
11111111111111111111111111111111110111011111
00000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
* Node y_3[22] => OE : 1 ,LOGIC : 10 *
L00924
11111111111111111111111111111111111111111111
11111111101111111111111111111111111011101101
11111111011111111111111111111111111011101110
```

Fuse Number

‘1’ fuse blown
‘0’ fuse intact

Comment line

JEDEC File Header, End of File

```
C22V10*
QP24*            Number of Pins*
QF5828*           Number of Fuses*
F0*              Note: Default fuse setting 0*
G0*              Note: Security bit Unprogrammed*
NOTE DEVICE C22V10*
NOTE PINS  a_3:2 a_2:3 a_1:4 a_0:5 b_3:6 b_2:7 b_1:8 b_0:9 s_0:13 s_1:14 *
NOTE PINS  s_2:15 y_0:19 y_1:20 y_2:21 y_3:22 *
NOTE PINS *
NOTE NODES *
L00000
00000000000000000000000000000000000000000000
........

C79CF*            Note: Fuse Checksum*
5FE9
```

# of fuses

default Fuse state

Checksum at end of file

JEDEC File Reader

- *utilities* directory has a JEDEC file reader package (*jedec_reader*)
  - Independent of PAL type being modeled
- Approach is to read the JEDEC file and return a bit_vector of fuse values
  - ‘0’ means connected (fuse intact), ‘1’ is disconnected (fuse blown)
- Handles the following record types
  - ‘L’ – fuse list
  - ‘Q’ – number of total fuses in device – needed to allocate fuse array
  - ‘F’ – default fuse state, used to fill fuse array with default state
  - ‘C’ – checksum record, reads this but does nothing with it.
- Other records ignored

Modeling a Programmable Part: Approaches

- Approach #1: have internal data structure that represents the entire programmable substrate
  - Read programming bits from external data file and “program” data structure to have needed routing and logic functionality
  - Model simply exercises programmable substrate with the presence of programming data
  - Perhaps most accurate simulation since it is closest to the hardware
  - execution time, memory requirements may be steep
- Approach #2: have an external model ‘generator’ (i.e. a Perl script) that reads the programming bits and generates only the functionality needed
  - Memory, execution time resources will be proportional to the percentage of the programmable device actually used

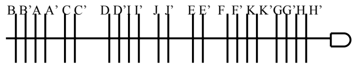
Modeling Functionality

- JEDEC reader returns a bit vector whose size is equal to the number of fuse locations
- This model uses the GENERATE statement in VHDL to create a model whose memory and runtime complexity is proportional to the number of fuses that are programmed
  - Model will take less memory space and run faster if less of the device is actually programmed
- GENERATE statement allows processes/signals to be generated at model elaboration time
  - somewhat similar to a macro capability in other languages

Compilation/Elaboration/Execution

- *Compilation* converts VHDL text to simulator dependent object code
- *Elaboration* is what happens when the model is loaded into memory
  - Initial processes/signal structures are created in memory
- GENERATE statements can be used to create signals and processes based upon parameters during elaboration.
  - This is a very powerful language feature.
- *Execution* happens after elaboration, and is the simulation loop of scheduling events and executing processes.

Product Term Modeling Approach



Use a *resolved* data type for product term (column signals are simply multiple drivers on the product term)

```
CONSTANT rows     : Natural := 64;
CONSTANT columns   : Natural := 32;
CONSTANT outputs   : Natural := 8;

SUBTYPE  ResolvedAndSL  IS Resolve_AND Std_Logic;
TYPE     ResolvedAndSLV IS ARRAY (Natural RANGE <>) OF ResolvedAndSL;


SIGNAL   AndTermsResolved : ResolvedAndSLV(0 TO rows - 1)      := (OTHERS => '1'); --And Terms Resolved
SIGNAL   AndTerms        : Std_Logic_Vector(0 TO rows - 1);                   --And Terms
SIGNAL   ci             : Std_Logic_Vector(0 TO columns - 1);                --Column Inputs
SIGNAL   OrTerms         : Std_Logic_Vector(0 TO outputs - 1);                 --Or Terms
SIGNAL   fb             : Std_Logic_Vector(1 TO 6);                      --fb(1 TO 6) <==> io(7 DOWNTO 2)
```

pal16L8 dependent

Model is for pal16L8 – fuse locations, # inputs, etc are all dependent on this PAL type.

Resolution Function for Product Term Types

```
-------------------------------------------------------------------------------
--Resolve_AND (Internal)
-------------------------------------------------------------------------------

 FUNCTION Resolve_AND (v : Std_Logic_Vector) RETURN Std_Logic IS
   VARIABLE result : Std_Logic := '1';
 BEGIN
   FOR ii IN v'RANGE LOOP
     result := result AND v(ii);
     EXIT WHEN result = '0';
   END LOOP;
   RETURN result;
 END Resolve_AND;
-------------------------------------------------------------------------------
SUBTYPE  ResolvedAndSL  IS Resolve_AND Std_Logic;
 TYPE     ResolvedAndSLV IS ARRAY (Natural RANGE <>) OF ResolvedAndSL;


 SIGNAL   AndTermsResolved : ResolvedAndSLV(0 TO rows - 1)      := (OTHERS => '1'); --And Terms
Resolved
```

Note early exit when function is zero.

Note that this is a subtype of Std_logic, which is itself a resolved type!!!!!

Connecting Inputs to Column Signals

```
ColumnConnect_i1to8Gen: --i(1 TO 8)
  FOR ii IN 1 TO 8 GENERATE
    ci((ii-1)*4)     <= TRANSPORT To_UX01(i(ii)) AFTER WD_i(ii);
    ci((ii-1)*4 + 1) <= TRANSPORT     NOT(i(ii)) AFTER WD_i(ii);
  END GENERATE ColumnConnect_i1to8Gen;
```

inverter

- Input signals are array *i(1 to 8)*
- column signals are *ci(0 to num_columns –1)*
- Each input connected to a pair of column signals (2nd connection is a complemented version of the input
- *wd_i* are wire delay generics that are defined on the entity
- The GENERATE statement causes these signal assignments to be expanded at elaboration time

Product Term Connections

```
ColumnConnect_i1to8Gen: --i(1 TO 8)
-------------------------------------------------------------------------------
--And Plane
-------------------------------------------------------------------------------
  AndPlaneGen:
  FOR row IN AndTermsResolved'RANGE GENERATE
    RowAllConnectedGen:
    IF (RowAllConnected(row)) GENERATE
      AndTermsResolved(row) <= '0';
    END GENERATE RowAllConnectedGen;
    RowNotAllConnectedGen:
    IF (NOT RowAllConnected(row)) GENERATE
      ColumnGen:
      FOR col IN ci'RANGE GENERATE
        ConnectGen:
        IF (FuseMap(row * ci'LENGTH + col) = connected) GENERATE
          AndTermsResolved(row) <= ci(col);
        END GENERATE ConnectGen;
      END GENERATE ColumnGen;
    END GENERATE RowNotAllConnectedGen;
  END GENERATE AndPlaneGen;
```

Reduces # of signal assignments, improves performance

Only generated if fuse map location = ‘0’!!

RowAllConnected Function

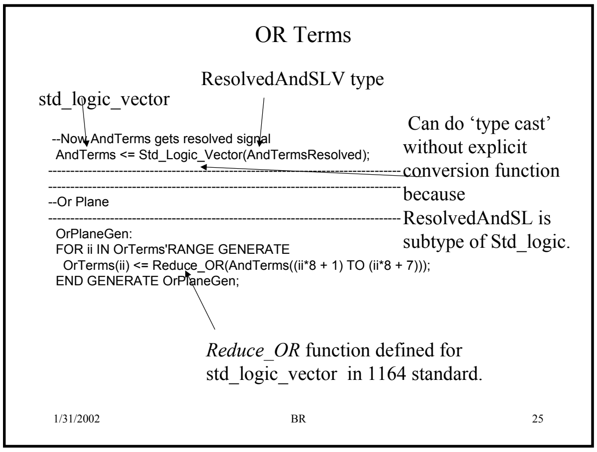
Recall that if all column inputs are connected to a product term, then product term output is ‘0’. Can reduce model complexity (memory and execution time) if detect this case.

```
-------------------------------------------------------------------------------
--RowAllConnected (Internal)
-------------------------------------------------------------------------------
 FUNCTION RowAllConnected (row : Natural) RETURN Boolean IS
 BEGIN
  RETURN NOT To_Boolean(Reduce_OR(
                                   FuseMap(row*columns TO (row+1)*columns - 1)));
 END RowAllConnected;
-------------------------------------------------------------------------------
```

Reduce_OR function defined in VHDL for bit_vectors – returns a ‘1’ if any bit in bit vector is a ‘1’, else returns ‘0’.

Model Complexity

- Model complexity is proportional to memory required and execution time
- The number of signals, signal assignments, and processes in a model impacts complexity
  - More signals and signal assignments means more events means more execution effort required
  - More signals means more memory needed to track waveform history
- Use of GENERATE statements only creates the required signal assignments for the product terms based upon the fuse map contents



OR Terms

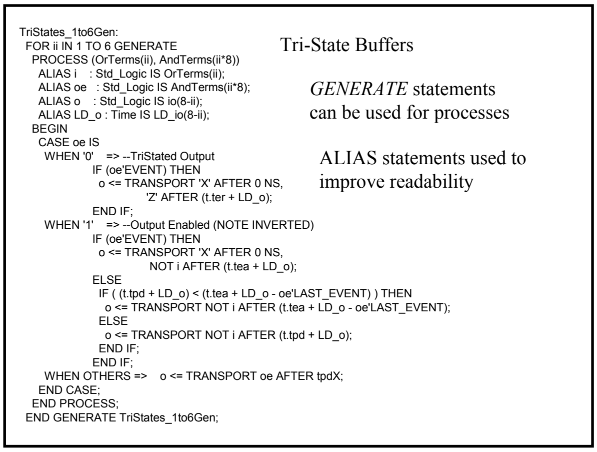
std_logic_vector

ResolvedAndSLV type

```
--Now AndTerms gets resolved signal
 AndTerms <= Std_Logic_Vector(AndTermsResolved);
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
--Or Plane
-------------------------------------------------------------------------------
 OrPlaneGen:
 FOR ii IN OrTerms'RANGE GENERATE
   OrTerms(ii) <= Reduce_OR(AndTerms((ii*8 + 1) TO (ii*8 + 7)));
 END GENERATE OrPlaneGen;
```

Can do ‘type cast’ without explicit conversion function because ResolvedAndSL is subtype of Std_logic.

Reduce_OR function defined for std_logic_vector in 1164 standard.



Tri-State Buffers

```
TriStates_1to6Gen:
 FOR ii IN 1 TO 6 GENERATE
   PROCESS (OrTerms(ii), AndTerms(ii*8))
     ALIAS i    : Std_Logic IS OrTerms(ii);
     ALIAS oe   : Std_Logic IS AndTerms(ii*8);
     ALIAS o    : Std_Logic IS io(8-ii);
     ALIAS LD_o : Time IS LD_io(8-ii);
   BEGIN
     CASE oe IS
       WHEN '0'    => --TriStated Output
                 IF (oe'EVENT) THEN
                   o <= TRANSPORT 'X' AFTER 0 NS,
                                 'Z' AFTER (t.ter + LD_o);
                 END IF;
       WHEN '1'    => --Output Enabled (NOTE INVERTED)
                 IF (oe'EVENT) THEN
                   o <= TRANSPORT 'X' AFTER 0 NS,
                                 NOT i AFTER (t.tea + LD_o);
                 ELSE
                   IF ( (t.tpd + LD_o) < (t.tea + LD_o - oe'LAST_EVENT) ) THEN
                     o <= TRANSPORT NOT i AFTER (t.tea + LD_o - oe'LAST_EVENT);
                   ELSE
                     o <= TRANSPORT NOT i AFTER (t.tpd + LD_o);
                   END IF;
                 END IF;
       WHEN OTHERS =>    o <= TRANSPORT oe AFTER tpdX;
     END CASE;
   END PROCESS;
 END GENERATE TriStates_1to6Gen;
```

GENERATE statements can be used for processes

ALIAS statements used to improve readability