ИНСТИТУТ ЯДЕРНОЙ ФИЗИКИ
им. Г.И. Будкера СО РАН

БИБЛИОТЕКА РАДИОИНЖЕНЕРА
ВВЕДЕНИЕ В АБЕЛЬ

(Информационный материал)

НОВОСИБИРСК

ИНСТИТУТ ЯДЕРНОЙ ФИЗИКИ
им. Г.И. БУДКЕРА СО РАН

БИБЛИОТЕКА РАДИОИНЖЕНЕРА

ВВЕДЕНИЕ В АБЕЛЬ

(Информационный материал)

Новосибирск
1993

БИБЛИОТЕКА РАДИОИНЖЕНЕРА

ВВЕДЕНИЕ В АБЕЛЬ

(Информационный материал)

Институт ядерной физики им. Г.И. Будкера
630090, Новосибирск 90, Россия

АННОТАЦИЯ

Издание предназначено для разработчиков радиоэлектронной аппаратуры. Пособие представляет из себя краткое введение в АБЕЛЬ-систему проектирования программируемых логических устройств.

Институт ядерной физики им. Г.И. Будкера СО РАН

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В разработках радиоинженеров Института ядерной физики широко используются программируемая логика- ППЗУ, ПЛМ и ПМЛ. Применение микросхем программируемой логики позволяет уменьшить количество микросхем в устройстве, увеличить схемную гибкость, увеличить надежность устройства и сократить время проектирования. Однако, если для первых представителей семейства программируемой логики удавалось спроектировать таблицу программирования "вручную" и за относительно короткое время, то для современных микросхем процесс составления такой таблицы занимает значительное время и сопровождается длительным процессом отладки. Ситуацию может изменить современная система проектирования.

В институте начала использоваться современная система проектирования устройств с программируемой логикой АБЕЛЬ. Однако, отсутствие руководств по использованию АБЕЛЯ сдерживает распространение этой системы.

Совет по автоматизации Института принял решение об издании справочной библиотечки радиоинженера. Предполагается издать руководства по наиболее распространенным и перспективным системам автоматизированного проектирования.

По системе проектирования АБЕЛЬ предполагается издать следующие пособия:

Введение в АБЕЛЬ.

Справочник по языку АБЕЛЬ.

Руководство пользователя языка АБЕЛЬ.

Руководство по применению языка АБЕЛЬ.

Первое руководство является незначительно сокращенным переводом системы обучения системы АБЕЛЬ. Остальные руководства предполагается значительно сократить и, возможно, совместить друг с другом в целях экономии объема.

ПОЧЕМУ ВЫ ДОЛЖНЫ ИСПОЛЬЗОВАТЬ АБЕЛЬ?

Используя АБЕЛЬ чтобы сделать вашу разработку в программируемой логике, вы можете получить экономию времени, денег, объема и мощности:

* С программируемой логикой вы можете комбинировать многие логические функции в одну программируемую микросхему, сберегая таким образом место на печатной плате, рассеиваемую мощность, задержки распространения и т.п.

* ABEL является средством проектирования, который делает разработки с программируемой логикой быстрыми и легкими. ABEL ограничивает ошибки из-за работы на битовом уровне, давая вам высокоуровневый символический язык проектирования, который автоматически транслирует описание в данные для программирования и тестирование микросхемы.

* ABEL ускоряет цикл разработки, отладки, перепроектирования и оптимизации. Вспомогательные средства АБЕЛя включают в себя отладочный SIMULATOR, символические имена выводов и сигналов для легких глобальных изменений, LOGIC REDUCTION чтобы уменьшить количество термов, диагностические сообщения об ошибках и дополнительную вспомогательную документацию.

* Выражать вашу разработку с помощью АБЕЛя легко, благодаря использованию техники описания проекта уже знакомой разработчикам логических устройств.

ЧТО МОЖЕТ ДЕЛАТЬ АБЕЛЬ?

Как вам нравится инструмент проектирования, который позволяет выразить описание декодера следующим образом?

```
truth_table (bcd -> led)
" bcd    a    b    c    d    e    f    g
0  ->  [ on,  on,  on,  on,  on,  on,  off];    "      a
1  ->  [off,  on,  on, off, off, off, off];    "     ---
2  ->  [ on,  on, off,  on,  on, off,  on];    "  f | g | b
3  ->  [ on,  on,  on,  on, off, off,  on];    "     ---
4  ->  [off,  on,  on, off, off,  on,  on];    "  e | d | c
5  ->  [ on, off,  on,  on, off,  on,  on];    "     ---
6  ->  [ on, off,  on,  on,  on,  on,  on];    "
7  ->  [ on,  on,  on, off, off, off, off];
8  ->  [ on,  on,  on,  on,  on,  on,  on];
9  ->  [ on,  on,  on,  on, off,  on,  on];
```

АБЕЛЬ сокращает разработку этого декодера двоично-десятичного кода в семисегментный. Если ваш проект выражается наиболее естественно таблицей истинности, вы можете позволить АБЕЛю обработать проект в этом виде.

АБЕЛЬ позволяет определить автомат следующим образом?

```
state_diagram LOOK_FOR_1_0
    state WAIT_FOR_1:  FLAG_1_0 = FALSE;
                       if INPUT == 1 then FOUND_1
                                     else WAIT_FOR_1;
    state FOUND_1:     FLAG_1_0 = FALSE;
                       if INPUT == 0 then FOUND_1_0
                                     else FOUND_1;
    state FOUND_1_0:   FLAG_1_0 = TRUE;
                       if INPUT == 1 then FOUND_1;
                                     else WAIT_FOR_1;
    state NULL;        FLAG_1_0 = FALSE;
                       goto WAIT_FOR_1;
```

АБЕЛЬ позволит вам спроектировать автомат прямо из вашей диаграммы состояний- никаких дополнительный трансляций в уравнения Буля, никаких карт Карно.

Разработчик логики всегда нуждается в уравнениях Буля. С АБЕЛем вы можете выразить ваши уравнения в той форме, которая вам требуется, не зависимо от типа устройства, которое вы выбрали.

```
XOR = IN1 $ IN2;                        " direct XOR OPERATOR
PROD_SUMS = (IN1#IN2) & !(IN3#IN4); " PRODUCT-OF-SUMS NOTATION
Q4 := IN1 & !IN2 & !IN3          " REGISTERED, SUM-OF-PRODUCTS,
   # !RESET;                          " COMPLEMENT NOTATION
```

В дополнение к стандартным логическим операциям, АБЕЛЬ также поддерживает арифметические и логические операции над группами (SETS). Группой является группа двоичных сигналов, которые могут обрабатываться как целое. Например, если ADDR является группой из восьми старших бит адресной шины, декодирование сигнала выборки микросхемы ППЗУ 2732 для шестнадцатеричных адресов F000-FFFF будет выглядеть очень просто:

```
!CHIP_ENABLE = (ADDR >= ^hF0); " low enable when ADDR >=hex F0
```

РАБОТА С АБЕЛЕМ

Создание полного логического проекта, используя АБЕЛЬ, осуществляется пятиступенчатым процессом:

1. Концептуализировать проект.
2. Написать логический проект с помощью языка АБЕЛЬ на вашем компьютере (IBM PC, VAX и т.п.) вашим любимым текстовым редактором. Это станет вашим ИСХОДНЫМ ФАЙЛОМ.
3. Обработать ваш исходный файл, используя языковой процессор АБЕЛЬ.
4. Загрузить результирующий ЗАГРУЗОЧНЫЙ ФАЙЛ ПРОГРАММАТОРА, содержащий карту перемычек, и ТЕСТОВЫЕ ВЕКТОРА в ваш программатор логики.
5. Программировать и проверить целевое устройство.

Хотя существует много вариантов подходящих для обработки проекта, одна простая команда выполнит стандартные функции:

* Преобразование всех ЛОГИЧЕСКИХ ОПИСАНИЙ в уравнения Буля.
* Выполнение логической редукции уравнений.
* Вычисление состояний перемычек, требуемых для определенного логического устройства.
* Моделирование операций устройства, чтобы проверить разработку перед программированием.
* Преобразование ваших тестовых векторов в формат загрузочного файла программатора.
* Генерирование соответствующего пакета документации.

СВОДКА ОСОБЕННОСТЕЙ АБЕЛЯ

АБЕЛЬ делает логическую разработку быстрой и легкой. Вот некоторые причины, почему:

* Логическое описание уравнениями Буля, таблицами истинности или диаграммой состояний в любой комбинации.
* Непосредственное использование любого оператора в булевских уравнениях.
* Автоматическое преобразование де-Моргана позволяет определить любую полярность сигналов.
* Логическая редукция может втиснуть вашу разработку в меньшую матрицу перемычек.
* Средства отладки разработки включают отладочный симулятор с трассированием и точками останова, диагностическими сообщениями об ошибках и файлами листинга отладки.
* Высокоуровневая логическая разработка обеспечивается использованием групп, макросами, включением файлов, директивами и другими конструкциями языка высокого уровня.
* Дополнительная проектная документация включает распечатки уравнений, имен переменных, помеченных рисунков микросхем, карт перемычек и тестовых векторов.

ОСНОВЫ ЛОГИЧЕСКОЙ РАЗРАБОТКИ

ВВЕДЕНИЕ

В этом уроке проектирования мы выполним некоторые основные логические элементы в устройстве PAL. Поскольку это будет наша первая логическая разработка в АБЕЛе, мы уделим особое внимание деталям абелевского исходного файла и полному процессу проектирования.

Наш проект будет выполнять трехвходовой вентиль И-НЕ, четырехвходовой блок И-ИЛИ-НЕ и двухвходовой вентиль ИСКЛЮЧАЮЩЕЕ ИЛИ в микросхеме 10L8. Мы продемонстрируем основные разделы и характеристики абелевского исходного файла в процессе и увидим как проект выражается в АБЕЛе. Мы напишем также тестовые вектора, чтобы протестировать один из наших вентилей и увидим как работает SIMULATOR.

Результатом обработки АБЕЛем нашего проекта должен быть файл документации, файл моделирования, файл листинга предварительного анализа и загрузочный файл программатора. Мы изучим файлы документации и моделирования и подготовим файл для программатора.

ИСХОДНЫЙ ФАЙЛ ПРОЕКТА

Исходный файл описывает логический проект в АБЕЛе. Написание проекта является последовательным процессом декларирования целевого устройства, поименования входных и выходных выводов, написания логических уравнений и написания тестовых векторов. Прочитайте проект ниже, чтобы почувствовать выражение проекта АБЕЛем, а детали будут раскрыты на нескольких следующих страницах.

Номер строки справа не является частью реального абелевского исходного файла; они используются для дальнейших ссылок в этом уроке.

```
                                                        (Line pos.)
module  BASIC_LOGIC;                                        (1)

title                                                       (2)
'ABEL basic logic design example        11 Nov. 1983        (3)
Data I/O Corp.   Redmond, WA                                (4)
" You can place comments anywhere, after a quotation mark.
   IC4          device  'P10L8';        " declare device    (5)
"
" Declare our input pins:
"
   NA1, NA2, NA3        pin  1,2,3;     " NAND inputs       (6)
   AOI1, AOI2,                                              (7)
   AOI3, AOI4           pin  4,5,6,7;   " AND_OR_INV inputs (8)
   X1, X2               pin  8,9;       " XOR  inputs       (9)
"
" Declare output pins:
"
   NAND,                                                    (10)
   AND_OR_INV, XOR,     pin 19,18,17;                       (11)
```

```
"
" Logic description bye Boolean equations:
equations                                                    (12)
    NAND        = !(NA1 & NA2 & NA3);                        (13)
    AND_OR_INV  = !((AOI1 & AOI2) # (AOI3 & AOI4));          (14)
    XOR         = X1 $ X2;                                   (15)
"
" Test vectors for NAND gate only:
test_vectors    ( [NA1, NA2, NA3] -> NAND)                   (16)
    [0, 0, 0]  ->  1;                                        (17)
    [0, 0, 1]  ->  1;                                        (18)
    [0, 1, 0]  ->  1;                                        (19)
    [0, 1, 1]  ->  1;    " test vectors are shown as         (20)
    [1, 0, 0]  ->  1;    " inputs -> outputs                 (21)
    [1, 0, 1]  ->  1;                                        (22)
    [1, 1, 0]  ->  1;                                        (23)
    [1, 1, 1]  ->  0;                                        (24)
end     BASIC_LOGIC;                                         (25)
```

ПРОВЕРКА ИСХОДНОГО ФАЙЛА

Мы будем сейчас использовать исходный файл BASIC_LOGIC, приведенный на предыдущих страницах, чтобы продемонстрировать несколько характеристик абелевского исходного файла.

Абелевский исходный файл составлен последовательностью одного или более модулей. BASIC_LOGIC является единственным модулем. Модуль начинается и заканчивается ключевыми словами "module" и "end" (1 и 25 строки файла BASIC_LOGIC).

Модуль состоит из четырех секций:

```
* Title.....................BASIC_LOGIC строки 2 по 4
* Declarations..............BASIC_LOGIC строки 5 по 11
* LOGIC DESCRIPTION........BASIC_LOGIC строки 12 по 15
* Test vectors.............BASIC_LOGIC строки 16 по 24
```

Соответствующие ключевые слова вводят секцию. Эти ключевые слова включают:

```
* title (BASIC_LOGIC строка 2)        * truth_table
* equations (BASIC_LOGIC строка 12)   * state_diagram
* test_vectors (BASIC_LOGIC строка 16)
```

Некоторые из деклараций, которые могут быть сделаны в секции декларации, включают:

= чтобы присвоить постоянное значение

device чтобы определить программируемое устройство (BASIC_LOGIC строка 5)

pin чтобы присвоить имена номерам выводов (BASIC_LOGIC строки 6 по 11)

NODE чтобы присвоить имена внутренним псевдовыводам (узлам)

MACRO чтобы определить макросы текстовой подстановки

АБЕЛЬ поддерживает широкое множество логических и арифметических операторов в своих логических уравнениях. BASIC_LOGIC демонстрирует несколько из них:

= для присвоения

для логического ИЛИ

! для логического НЕ

$ для логического ИСКЛЮЧАЮЩЕЕ ИЛИ

& для логического И

Тестовые вектора использованы чтобы проверить логический проект и функционально протестировать запрограммированное устройство. Мы изучим их использование позже.

Формат исходного абелевского файла спроектирован легко составляемым и читаемым. Имена устройств и выводов определяются вверху, затем следует описание проекта в той форме, которая вам требуется, затем тестовые вектора в четкой и графической форме. Пустые строки, пробелы и табуляторы могут быть помещены везде где вы пожелаете для улучшения читаемости.

ЯЗЫКОВОЙ ПРОЦЕССОР АБЕЛЬ

Обработаем пример BASIC_LOGIC. АБЕЛЬ определит набор состояний перемычек, который будет выполнять определенных нами функций И-НЕ, И-ИЛИ-НЕ и ИСКЛЮЧАЮЩЕЕ ИЛИ, когда эти перемычки будут запрограммированы в PAL 10L8. Наши тестовые вектора будут преобразованы в JEDEC формат для тестирования запрограммированного устройства.

Стандартная команда для обработки абелевского исходного файла, названного "filename.ABL" является простой:

ABEL filename

Эта команда автоматически запускает все программы требуемые чтобы обработать и промоделировать абелевский исходный файл и произвести пакет документов. Многие режимы могут быть выбраны включением параметров после имени файла, позволяя вам передать аргументы входному файлу (такие как тип устройства), определить уровень логической редукции, установить параметры загрузочного файла программатора, определить форматирование документов и так далее.

ОБРАБОТКА BASIC_LOGIC

Полностью операции обработки заставляют работать каждую из программ, которые составляют языковой процессор АБЕЛЬ. Мы будем использовать единственную команду "ABEL" чтобы делать это автоматически, как описано на предыдущей странице. Будем считать, что наш исходный файл BASIC_LOGIC хранится на диске как файл с именем "LOGIC.ABL". Мы будем использовать все параметры умолчания, так что никакой из специальных параметров, упомянутых на предыдущих страницах, не потребуется.

A>ABEL LOGIC (нажмите Enter для запуска)

На экране последовательно распечатается следующее:

```
A>echo off

.
ABEL Version 1.00
.
PARSE Version 1.00  Copiright (c) 1983,1984 Data I/O, Redmond, WA
module BASIC_LOGIC
PARSE complete. Time: 7 seconds
```

```
.
TRANSFOR Version 1.00 Copiright (c) 1983,1984 Data I/O,Redmond,WA
module BASIC_LOGIC
TRANSFOR complete. Time: 4 seconds
.
REDUCE Version 1.00  Copiright (c) 1983,1984 Data I/O,  Redmond,WA
module BASIC_LOGIC
_device IC4
Simple Reductions
REDUCE complete. Time: 5 seconds
.
FUSEMAP Version 1.00 Copiright (c) 1983,1984 Data I/O,Redmond,WA
module BASIC_LOGIC
_device IC4
5 of 16 terms used
FUSEMAP complete. Time: 7 seconds
.
SIMULATE Version 1.00 Copiright (c) 1983,1984 Data I/O,Redmond,WA
module BASIC_LOGIC
8 of 8 vectors in IC4 passed
SIMULATE complete. Time: 4 seconds
.
DOCUMENT Version 1.00 Copiright (c) 1983,1984 Data I/O,Redmond,WA
module BASIC_LOGIC
_device IC4
DOCUMENT complete. Time: 4 seconds
A>
```

ВЫХОД ЯЗЫКОВОГО ПРОЦЕССОРА

В примере мы увидели как АБЕЛЬ обрабатывает наш исходный файл с примером, запуская каждую из шести программ, составляющие языковой процессор АБЕЛЬ. Некоторые факты о программах процессора могут быть отмечены здесь:

* Каждая программа перечисляет каждый модуль внутри исходного файла, когда этот модуль обрабатывается. Наш пример состоит из единственного модуля, BASIC_LOGIC.
* Программы FUSEMAP и DOCUMENT перечисляют также имена устройств, когда они встречаются. АБЕЛЬ поддерживает множество устройств внутри модуля.
* Распечатывается истекшее время для каждой программы.

АБЕЛЬ создает ряд файлов как результат обработки исходного файла. Обеспечиваются многие режимы, но по умолчанию файловая структура из файла LOGIC.ABL будет следующая:

```
    LOGIC.ABL----------> IC4.JED (загрузочный файл
(исходный файл)  |                 программатора)
                 |----> LOGIC.DOC (файл документации)
                 |
                 |----> LOGIC.SIM (файл симуляции)
                 |
                 |----> LOGIC.LST (файл входного разбора)
```


ФАЙЛ ДОКУМЕНТАЦИИ

На нескольких следующих страницах мы будем изучать файл документации LOGIC.DOC. Этот файл является стандартным ASCII файлом, пригодным для печати. Возможны многие вариации, но по умолчанию файл состоит из двух секций:

* Финальные логические уравнения.
* Изображение микросхемы и ее выводов.

Первая секция LOGIC.DOC показана ниже. Заметьте, что наверху расположено стандартное начало, включающее **дату, номер страницы и заголовок**, который копируется из исходного файла. **Уравнения** здесь показаны в конечной форме, после преобразования к логической пригодности для целевого устройства. Отметьте, что уравнение ИСКЛЮЧАЮЩЕЕ ИЛИ преобразовано в сумму произведений, требуемых для 10L8.

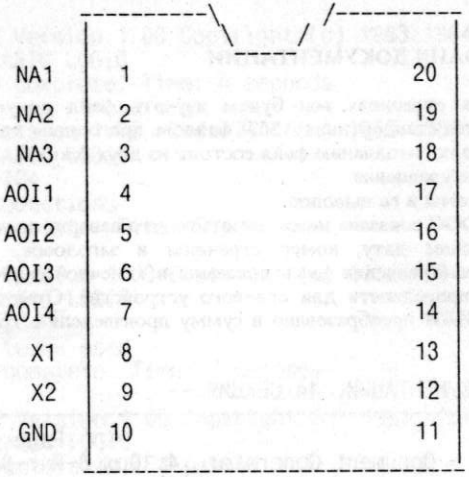
-- ФАЙЛ ДОКУМЕНТАЦИИ, 1я СЕКЦИЯ --

```
                                                              Page 1
ABEL(tm) Version  1.00   - Document Generator  4:10pm 8-Mar-84
ABEL basic logic design example       11 Nov. 1983
DATA I/O Corp.    Redmond, WA

Equations for Module BASIC_LOGIC

Device IC4:

  Reduced Equations:

    XOR = !((!X2 & !X1 # X2 & X1));

    AND_OR_INV = !((AOI4 & AOI3 # AOI2 & AOI1));

    NAND = !(NA3 & NA2 & NA1 );
```

Вторая секция файла документации состоит из разводки выводов на изображении микросхемы для всех устройств в этом проекте. Эти изображения полезны, когда отлаживается аппаратная часть, и могут быть включены в сервисную документацию.

-- ФАЙЛ ДОКУМЕНТАЦИИ, 2я СЕКЦИЯ --

```
                                                              Page 2
ABEL(tm) Version  1.00   - Document Generator  4:10pm 8-Mar-84
ABEL basic logic design example       11 Nov. 1983
DATA I/O Corp.    Redmond, WA

Chip diagram for Module BASIC_LOGIC

Device IC4:
```




```
end of module BASIC_LOGIC
```

ФАЙЛЫ ПРЕДВАРИТЕЛЬНОГО РАЗБОРА И СИМУЛЯЦИИ

Два других файла генерируемых языковым процессором это файл предварительного разбора и файл симуляции.

Файл предварительного разбора показывает эффекты расширения исходного файла использованием макросов, включений файлов или директив, а также отображает где найдены синтаксические ошибки. Поскольку мы не использовали каких-либо способов расширения исходного файла и не сделали никаких синтаксических ошибок, наш файл предварительного разбора LOGIC.LST выглядит аналогично оригинальному исходному файлу и здесь мы его исследовать не будем.

Файл симуляции генерируется программой SIMULATE, сравнивающей наши тестовые вектора с внутренней моделью PAL 10L8, запрограммированной состоянием перемычек, результирующих из наших логических уравнений. Существует несколько уровней детализации возможных в файле симуляции и вы можете изменить уровень детализации для различных векторов, используя точки останова.

ИЗУЧЕНИЕ ФАЙЛА СИМУЛЯЦИИ

Стандартный формат файла симуляции, называемый уровнем слежения 1, показан для нашего примера ниже. Это простейший формат; в процессе обработки может быть выбран больший уровень детальности. Уровень слежения 3 обеспечивает полное описание состояний для каждого вектора, включая номера и имена выводов и информацию о термах и перемычках.

На уровне слежения 1 каждый вектор трактуется отдельно тремя строками. Номер вектора появляется в первой строке, на следующей строке печатается выходной вектор, как это определяется симулятором. Третья строка показывает вектор как определено в исходном файле, в порядке последовательности выводов и

использует символы JEDEC для состояний выводов. Если выходные вектора во второй и третьей строках соответствующие, то симуляция происходит правильно. Любые ошибки будут отмечены диагностическими сообщениями. Заметим, что "N" означает "output not tested" (выход не проверяется) в векторах исходного файла.

```
                -- ФАЙЛ СИМУЛЯЦИИ, УРОВЕНЬ СЛЕЖЕНИЯ 1
-------------------------------------------------------------------
Simulate device IC4, type 'P10L8'

Vector 1
Output          [                    ] -> [..........HHHHHLHH.]
V 0001          [00000000000000000000] -> [..........NNNNNNNH.]

Vector 2
Output          [                    ] -> [..........HHHHHLHH.]
V 0002          [00100000000000000000] -> [..........NNNNNNNH.]

Vector 3
Output          [                    ] -> [..........HHHHHLHH.]
V 0003          [01000000000000000000] -> [..........NNNNNNNH.]

Vector 4
Output          [                    ] -> [..........HHHHHLHH.]
V 0004          [01100000000000000000] -> [..........NNNNNNNH.]

Vector 5
Output          [                    ] -> [..........HHHHHLHH.]
V 0005          [10100000000000000000] -> [..........NNNNNNNH.]

Vector 6
Output          [                    ] -> [..........HHHHHLHH.]
V 0006          [11000000000000000000] -> [..........NNNNNNNH.]

Vector 7
Output          [                    ] -> [..........HHHHHLHL.]
V 0007          [11100000000000000000] -> [..........NNNNNNNL.]
```

ЗАГРУЗКА ПРОГРАММИРУЮЩИХ ДАННЫХ

Последний шаг в разработке программируемой логики заключается в программировании и тестировании микросхем. Когда АБЕЛЬ обработал исходный файл и сформировал файлы загрузки программатора, содержащие данные для программирования и тестирования микросхем, эти загрузочные файлы могут быть загружены в программатор логики и использованы чтобы запрограммировать и проверить микросхемы.

Загрузочным файлом программатора для единственного устройства в нашей разработке является файл, названный IC4.JED. Это ASCII файл в формате, поддерживаемом JEDEC, готовый для копирования в порт для загрузки в программатор.

РАЗРАБОТКА АВТОМОБИЛЬНОГО СТОРОЖА

ПОДХОД К РАЗРАБОТКЕ

В этом примере разработки, мы будем программировать PAL чтобы выполнить сигнализатор вторжения в автомобиль. Мы спроектируем сторож как автомат, описывая логику, используя абелевскую STATE DIAGRAM (диаграмму состояний).

Мы будем следовать разработочному подходу, описанному во введении:

1. Концептуализировать проект.
2. Написать логический проект на языке АБЕЛЬ.
3. Обработать ваш исходный файл, используя языковой процессор АБЕЛЬ.
4. Загрузить результирующий ЗАГРУЗОЧНЫЙ ФАЙЛ ПРОГРАММАТОРА, содержащий карту перемычек, и ТЕСТОВЫЕ ВЕКТОРА в ваш программатор логики.
5. Программировать и проверить целевое устройство.

КОНЦЕПТУАЛИЗАЦИЯ ПРОЕКТА

Наш сигнализатор о вторжении должен запускаться выключателями открытия или закрытия, когда двери открыты. Активирование системы (состояние READY) должно выполняться коротким поворотом ключа зажигания в позицию "accessory". Когда дверь открыта, чтобы водитель вышел, должен прозвучать предупреждающий сигнал активной системы (состояние WARNING). Закрывание двери в этот первый раз должно остановить предупреждающий сигнал и полностью переключиться в сторожевое состояние (состояние ARMED). Теперь если дверь откроется, на несколько секунд (состояние ALERT) должен зазвучать предостерегающий сигнал, а затем включиться сигнал тревоги (состояние ALARM), независимо от закрытия двери. Поворачивание ключа зажигания в положение "ignition" в любое время, отключает систему.

Переходы между состояниями могут быть нарисованы следующим образом:

```
-->DISARMED-a->READY-b->WARNING-c->ARMED-b->ALERT-d->ALARM
|            |e     |e         |e       |e       |e
|            |      |          |        |        |
-----------------------------------------------------------

a = ключ в "accessory"              d = таймаут
b = выключатель двери защелкнут     e = ключ на зажигание
c = все выключатели выключены           (ignition)
```

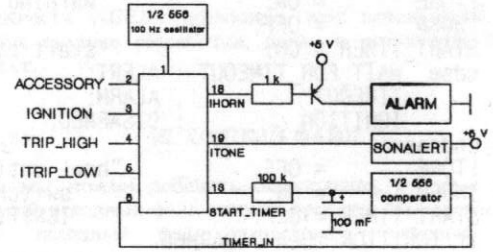
STATE MACHINES (автомат) выполнен на регистровом устройстве. Шесть состояний сторожевой логики (DISARMED, READY, WARNING, ARMED, ALERT, and ALARM) будут требовать трех триггеров, так как число возможных состояний автомата равно два в степени числа триггеров. PAL типа 16R4 (1556ХП4) должна прекрасно работать, предоставляя четыре триггера, восемь входных выводов и четыре вывода входов/выводов.

Диаграмма PAL выглядит следующим образом:


```
                                  -----------
                  Такты --->|         |---> Старт таймаута
      Ключ = Accessory --->|         |
       Ключ = Ignition --->|  16R4   |---> Сигнал предуп-
     Таймаут завершен --->| 1556ХП4 |          реждения
Дверной выкл. (актив. HIGH)--->|    |---> Сигнал тревоги
Дверной выкл. (актив. LOW) --->|    |
```

Упрощенная схема показана ниже. Заметим, что определены активными как низкий, так и высокий потенциалы сигналов, в зависимости от потребностей схемы.

УПРОЩЕННАЯ СХЕМА СТОРОЖА



ОПИСАНИЕ ПРОЕКТА В АБЕЛЕ

Теперь, когда концепция сформулирована, она может быть написана на АБЕЛе. Если ваши состояния и переходы определены, вы можете непосредственно на АБЕЛе писать диаграмму состояний, не делая ручного преобразования в булевские уравнения и карты Карно. Используя имена и полярности сигналов из схемы на предыдущей странице и диаграмму переходов, описанную выше, мы можем написать диаграмму состояний, как показано ниже.

```
state_diagram  STATES
state DISARMED: !TONE       = OFF;
                !HORN       = OFF;
                START_TIMER = OFF;
                if ACCESSORY then READY
                     else DISARMED;   " wait for arming
state READY:    !TONE       = OFF;
                !HORN       = OFF;
                START_TIMER = OFF;
                case  TRIP      :  WARNING; "go to WARNING when
                      UNTRIPPED:  READY;    "   switch tripped
                      IGNITION :  DISARMED; "   (driver exits)
                endcase;
```

```
state WARNING:  !TONE        = ON;            " warning tone ON
                !HORN        = OFF;
                START_TIMER = OFF;
                case  TRIP       :  WARNING;
                      UNTRIPPED:  ARMED;      " arm when all
                      IGNITION :  DISARMED; "switches untripped
                endcase;
state ARMED:    !TONE        = OFF;
                !HORN        = OFF;
                START_TIMER = OFF;
                case  TRIP       :  ALERT;      " ALERT on tripped
                      UNTRIPPED:  ARMED;      "   switch
                      IGNITION :  DISARMED;
                endcase;
state ALERT:    !TONE        = ON;            " warning tone ON
                !HORN        = OFF;
                START_TIMER = ON;            " start timeout
                case  WAIT_FOR_TIMEOUT : ALERT;
                      TIMEOUT          : ALARM;
                      IGNITION         : DISARMED;
                endcase;
state ALARM:    !TONE        = OFF;          "howl until key is
                !HORN        = ON;           "   switched to
                START_TIMER = OFF;           "   IGNITION
                if IGNITION then DISARMED
                            else ALARM;
```

ИЗУЧЕНИЕ ДИАГРАММЫ СОСТОЯНИЙ

Диаграмма состояний состоит из серии определения состояний. Каждое определение состояний описывает какие уровни выходных сигналов должны считываться (здесь !TONE, !HORN и START_TIMER), и какие состояния должны быть по следующему тактовому импульсу. Определение переходов между состояниями делаются одним из трех решающих конструкций, как показано ниже. Примеры из нашего сторожевого устройства показывают как они используют:

IF/THEN/ELSE В состоянии DISARMED, IF ACCESSORY истинно, мы идем в состояние READY, ELSE мы остаемся в состоянии DISARMED.

CASE В состоянии READY, в CASE of TRIP истинно, мы идем в состояние WARNING, UNTRIPPED истинно означает, что мы остаемся в состоянии READY, и IGNITION истинно- значит перейти в состояние DISARMED.

GOTO просто безусловно определяет следующее состояние.

Физически, состояния автомата определены двоичным кодом, который содержится в регистре устройства. Мы не используем регистр состояний в нашем сторожевом устройстве для чего-либо кроме счета состояний, поэтому текущие двоичные значения состояний являются произвольными. Во многих разработках, регистр состояний сам может действовать как выход, поэтому двоичное значение состояний может быть важным.

МАКРОСЫ В ДИАГРАММЕ СОСТОЯНИЙ

TRIP, UNTRIPPED, WAIT_FOR_TIMEOUT, и TIMEOUT представляют собой условия на входных выводах, TRIP означает, что входные выводы проверяют скрытый выключатель и так далее. Определяя состояния входных выводов соответственно условиям, мы можем просто обращаться к условиям по их именам во всем проекте и определять эти состояния в терминах наборов состояний входных выводов. Эта техника называется макроопределениями и TRIP, UNTRIPPED, WAIT_FOR_TIMEOUT, и TIMEOUT являются макросами. Макросы определяются в секции деклараций абелевского исходного файла.

Когда бы макрос не появлялся в исходном абелевском файле, он замещается в процессе обработки текстовым блоком, который вы определили. Эти блоки текста для наших макросов будут состоять из булевских выражений, определяющих входные выводы.

Макро возможности АБЕЛя позволяют вам использовать высокоуровневые описания на ранних стадиях разработки, давая в результате более надежный и естественный проект.

ИСХОДНЫЙ ФАЙЛ

В этом месте мы можем добавить определения, которые нам нужны чтобы сделать завершенный исходный файл нашего проекта. Мы сделаем это в деталях, а затем завершим исходный файл написанием тестовых векторов, чтобы протестировать проект.

Составление завершенного исходного файла из диаграммы состояний влечет добавление имени MODULE, заголовка, устройства, секции декларации и каких-нибудь комментариев к диаграмме состояний, которые мы уже описали. Все это показано ниже.

```
module     AUTO_ALARM;
title
'Automobile intrusion alarm          29 Sep. 1983
ABEL design example, Data I/O Corp.'
    U1     device       'P16R4';     " Declare our target device
" Constants:
    ON          =      1;
    OFF         =      0;
    H           =      1;
    L           =      0;
    X           =      .X.;         " ABEL 'don't care' symbol
    C           =      .C.;         " ABEL 'clocking input' symbol

" State definitions(i.e.binary value of output register STATES):
    DISARMED    =      ^b1111;      " These are arbitrary binary
    READY       =      ^b1110;      " state numbers.  See text.
    WARNING     =      ^b1100;
    ARMED       =      ^b1101;
    ALERT       =      ^b1001;
    ALARM       =      ^b1011;
```

```
" Pin names:
    ACCESSORY   pin   2;          " Input pins
    IGNITION    pin   3;
    TRIP_HIGH   pin   4;
    TRIP_LOW    pin   5;
    TIMER_IN    pin   6;
    CLK         pin   1;          " Clock pin
    TONE        pin   19;         " Output pins
    HORN        pin   18;
    START_TIMER pin   13;
    STATES3,                      " STATES, set of output pins
    STATES2,
    STATES1,
    STATES0     pin   17, 16, 15, 14;
    STATES      =     [STATES3, STATES2, STATES1, STATES0];

" Macros:
    TRIP        macro
      { (TRIP_HIGH # !TRIP_LOW) & !IGNITION };"true if switch
                                              "tripped
    UNTRIPPED   macro
      { !TRIP_HIGH & TRIP_LOW & !IGNITION };"true when all
                                           "switches untripped

    WAIT_FOR_TIMEOUT  macro
      { TIMER_IN & !IGNITION };     " true waiting for timeout

    TIMEOUT     macro
          { !TIMER_IN & !IGNITION };      " true on timeout
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

state_diagram  STATES
state DISARMED: !TONE       = OFF;
                !HORN       = OFF;
                START_TIMER = OFF;
                if ACCESSORY then READY
                             else DISARMED;  " wait for arming
state READY:    !TONE       = OFF;
                !HORN       = OFF;
                START_TIMER = OFF;
                case  TRIP      :  WARNING; "go to WARNING when
                      UNTRIPPED:   READY;   "    switch tripped
                      IGNITION :   DISARMED; "   (driver exits)
                endcase;
state WARNING:  !TONE       = ON;           " warning tone ON
                !HORN       = OFF;
                START_TIMER = OFF;
                case  TRIP      :  WARNING;
                      UNTRIPPED:   ARMED;   " arm when all
                      IGNITION :   DISARMED; "switches untripped
                endcase;
```



```
state ARMED:    !TONE       = OFF;
                !HORN       = OFF;
                START_TIMER = OFF;
                case  TRIP        :  ALERT; " ALERT on tripped
                      UNTRIPPED   :  ARMED; "   switch
                      IGNITION    :  DISARMED;
                endcase;
state ALERT:    !TONE       = ON;           " warning tone ON
                !HORN       = OFF;
                START_TIMER = ON;           " start timeout
                case  WAIT_FOR_TIMEOUT : ALERT;
                      TIMEOUT          : ALARM;
                      IGNITION         : DISARMED;
                endcase;
state ALARM:    !TONE       = OFF;     "howl until key is
                !HORN       = ON;      "   switched to
                START_TIMER = OFF;     "   IGNITION
                if IGNITION then DISARMED
                            else ALARM;
```

ИЗУЧЕНИЕ ИСХОДНОГО ФАЙЛА

Ключ к трансформации вашего оригинального абелевского проекта в полный исходный файл лежит в использовании способностей АБЕЛя к декларациям и определениям. Важной особенностью является способность присваивать осмысленные имена выводам и состояниям. Это делает вашу диаграмму состояний, уравнения и таблицы истинности легко читаемыми и упрощает изменение присвоенных значениях при экспериментах с проектом.

Цифровые значения присваиваемые состояниям автомата являются хорошим кандидатом для экспериментирования с проектом. Во многих случаях одна группа номеров состояний будет генерировать уравнения, для которых трудно выполнить логическую редукцию, в то время как другая быстро уменьшится к компактному результату. Номера состояний, которые мы выбрали для нашего проекта представляют собой хороший первоначальный выбор по двум причинам:

1. Состояния, которые отличаются только одним битом имеют тенденцию легко редуцироваться.

2. Способ генерации уравнений из диаграммы состояний заставляет неопределенные состояния автоматически переходить в двоичное состояние 1111 в PAL 16R4 (1556ХП4). Таким образом все наши неопределенные состояния переходят в состояние DISARMED без написания дополнительных уравнений.

Группы являются еще одной особенностью АБЕЛя. Определением набора сигналов как группы можно получить обработку группы как целого. Таким образом, написание STATES=5 вынудит АБЕЛЬ автоматически генерировать уравнения, чтобы перевести четыре вывода состояния (STATES) в двоичное 5 (0101) следующим образом:

STATES3=0; STATES2=1; STATES1=0; STATES0=1;

Группы могут также складываться, вычитаться, сравниваться и т.д. и АБЕЛЬ будет автоматически управлять генерацией всех уравнений.

Мы сделали также необходимые макроопределения для наших макросов TRIP, UNTRIPPED, WAIT_FOR_TIMEOUT, и TIMEOUT. Блок текста между фигурными

скобками макроопределений вставляется в исходный файл когда используется макро. Таким образом, когда мы пишем:

```
case WAIT_FOR_TIMEOUT : ALERT;
```

ABEL на самом деле обработает:

```
case TIMER_IN & !IGNITION : ALERT;
```

ТЕСТОВЫЕ ВЕКТОРА

После составления логического описания проекта, следующим шагом нужно написать тестовые вектора, чтобы проверить правильность проекта. Тестовые вектора являются методом переописания того же проекта но другим способом для проверки проекта. Обработка тестовых векторов включенных в проект симулятором АБЕЛя является прекрасным способом удвоить проверку вашего проекта и обеспечить ценной отладочной информацией.

Тестовые вектора для большинства автоматов не могут проверить всех возможных состояний, вследствие огромного числа возможных комбинаций. Здесь мы будем использовать "x" в наших векторах, чтобы сказать симулятору "не используй этот вход в своих логических вычислениях" и этим ограничить тысячи возможных входных состояний. Нормальные переходы рабочих состояний, однако, будут полностью проверены.

```
test_vectors ([ACCESSORY,IGNITION,TRIP_HIGH,TRIP_LOW,TIMER_IN,CLK]
->
                [STATES, TONE, HORN, START_TIMER] )
                  " specify [inputs] -> [outputs]
"      A  I  T  T  T  C                T  H  S   <-- Column labels
"      C  G  R  R  I  L                O  O  T
"      C  N  P  P  M  K                N  R  R
"         H  L  E      STATES          E  N  T

[x, H, x, x, x, c] -> [DISARMED, 1, 1, 0];"IGNITION initializes
[H, L, x, x, x, c] -> [READY,    1, 1, 0];
[x, L, H, H, x, c] -> [WARNING,  0, 1, 0];"WARNING via TRIP_HIGH

[x, H, x, x, x, c] -> [DISARMED, 1, 1, 0];
[H, L, x, x, x, c] -> [READY,    1, 1, 0];
[x, L, L, L, x, c] -> [WARNING,  0, 1, 0];"WARNING via TRIP_LOW
[x, L, L, L, x, c] -> [WARNING,  0, 1, 0];
[x, L, L, H, x, c] -> [ARMED,    1, 1, 0];"ARMED when untripped
[x, L, H, H, x, c] -> [ALERT,    0, 1, 1];"ALERT via TRIP_HIGH
[x, H, x, x, x, c] -> [DISARMED, 1, 1, 0];
[H, L, x, x, x, c] -> [READY,    1, 1, 0];
[x, L, L, L, x, c] -> [WARNING,  0, 1, 0];
[x, L, L, H, x, c] -> [ARMED,    1, 1, 0];
[x, L, L, H, x, c] -> [ARMED,    1, 1, 0]; "wait till tripped
[x, L, L, L, x, c] -> [ALERT,    0, 1, 1]; "ALERT via TRIP_LOW
[x, L, x, x, H, c] -> [ALERT,    0, 1, 1]; "wait for timeout
[x, L, x, x, L, c] -> [ALARM,    1, 0, 0]; "ALARM when timeout
[x, L, x, x, x, c] -> [ALARM,    1, 0, 0]; "Honk till IGNITION
[x, H, x, x, x, c] -> [DISARMED, 1, 1, 0];
```


ИЗУЧЕНИЕ ТЕСТОВЫХ ВЕКТОРОВ

О тестовых векторах нужно думать как о перепроекте ваших логических определений с точки зрения устройства в работе. Когда вы написали логические определения с функциональной точки зрения, очень полезно сдвинуть вашу точку зрения и определить все входы так как устройство их видит и изучить нужные выходы.

Тестовые вектора и симулятор АБЕЛя помогают разработчику в этом отношении. Формат тестовых векторов следующий:

[входной вектор] -> [выходной вектор]

где каждый вектор составлен из списка логических уровней, соответствующих определенным входным и выходным выводам. Логические уровни могут быть также выражены в виде группы, как мы делали с STATES.

Тестовые вектора для автоматов определяют также тактовый вывод во входном векторе. В нашем примере вы можете увидеть состояние входов слева, затем следует тактовый вывод и выходной вектор справа.

ПОЛНЫЙ ИСХОДНЫЙ ФАЙЛ

Теперь мы можем включить тестовые вектора в наш исходный файл с диаграммой состояний и иметь полный абелевский проект.

```
module      AUTO_ALARM;
title
'Automobile intrusion alarm          29 Sep. 1983
ABEL design example, Data I/O Corp.'

    U1    device      'P16R4';   " Declare our target device
" Constants:
    ON          =     1;
    OFF         =   - 0;
    H           =     1;
    L           =     0;
    x           =     .X.;  " ABEL 'don't care' symbol
    c           =     .C.;  " ABEL 'clocking input' symbol

" State definitions (i.e.binary value of output register STATES):
    DISARMED    =     ^b1111; "These are arbitrary binary state
    READY       =     ^b1110; " numbers.  See text.
    WARNING     =     ^b1100;
    ARMED       =     ^b1101;
    ALERT       =     ^b1001;
    ALARM       =     ^b1011;

" Pin names:
    ACCESSORY   pin   2;          " Input pins
    IGNITION    pin   3;
    TRIP_HIGH   pin   4;
    TRIP_LOW    pin   5;
    TIMER_IN    pin   6;
```

```
    CLK          pin   1;          " Clock pin
    TONE         pin   19;         " Output pins
    HORN         pin   18;
    START_TIMER  pin   13;
    STATES3,                       " STATES, set of output pins
    STATES2,
    STATES1,
    STATES0      pin   17, 16, 15, 14;
    STATES       =     [STATES3, STATES2, STATES1, STATES0];

" Macros:
TRIP         macro
   { (TRIP_HIGH # !TRIP_LOW) & !IGNITION };"true if switch
                                           "tripped

UNTRIPPED    macro
   { !TRIP_HIGH & TRIP_LOW & !IGNITION };"true when all
                                         "switches untripped
WAIT_FOR_TIMEOUT  macro
   { TIMER_IN & !IGNITION };          "true waiting for timeout
TIMEOUT      macro
     { !TIMER_IN & !IGNITION };       " true on timeout
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

state_diagram  STATES
state DISARMED: !TONE       = OFF;
                !HORN       = OFF;
                START_TIMER = OFF;
                if ACCESSORY then READY
                             else DISARMED; " wait for arming
state READY:    !TONE       = OFF;
                !HORN       = OFF;
                START_TIMER = OFF;
                case  TRIP      : WARNING; "go to WARNING when
                      UNTRIPPED: READY;     "   switch tripped
                      IGNITION : DISARMED;  "   (driver exits)
                endcase;
state WARNING:  !TONE       = ON;          " warning tone ON
                !HORN       = OFF;
                START_TIMER = OFF;
                case  TRIP    :  WARNING;
                    UNTRIPPED:  ARMED;     " arm when all
                    IGNITION :  DISARMED;  "switches untripped
                endcase;
state ARMED:    !TONE       = OFF;
                !HORN       = OFF;
                START_TIMER = OFF;
                case  TRIP      :  ALERT;     " ALERT on tripped
                      UNTRIPPED:  ARMED;     "   switch
                      IGNITION :  DISARMED;
                endcase;
```



```
state ALERT:    !TONE       = ON;          " warning tone ON
                !HORN       = OFF;
                START_TIMER = ON;          " start timeout
                case  WAIT_FOR_TIMEOUT : ALERT;
                      TIMEOUT          : ALARM;
                      IGNITION         : DISARMED;
                endcase;
state ALARM:    !TONE       = OFF;         "howl until key is
                !HORN       = ON;          "   switched to
                START_TIMER = OFF;         "   IGNITION
                if IGNITION then DISARMED
                            else ALARM;
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

test_vectors ([ACCESSORY,IGNITION,TRIP_HIGH,TRIP_LOW,TIMER_IN,CLK]-
>
              [STATES, TONE, HORN, START_TIMER] )
                " specify [inputs] -> [outputs]

"     A  I  T  T  T  C                T  H  S   <-- Column labels
"     C  G  R  R  I  L                O  O  T
"     C  N  P  P  M  K                N  R  R
"           H  L  E        STATES     E  N  T

[x, H, x, x, x, c] -> [DISARMED, 1, 1, 0];"IGNITION initializes
[H, L, x, x, x, c] -> [READY,    1, 1, 0];
[x, L, H, H, x, c] -> [WARNING,  0, 1, 0];"WARNING via TRIP_HIGH

[x, H, x, x, x, c] -> [DISARMED, 1, 1, 0];
[H, L, x, x, x, c] -> [READY,    1, 1, 0];
[x, L, L, L, x, c] -> [WARNING,  0, 1, 0];"WARNING via TRIP_LOW
[x, L, L, L, x, c] -> [WARNING,  0, 1, 0];
[x, L, L, H, x, c] -> [ARMED,    1, 1, 0];"ARMED when untripped
[x, L, H, H, x, c] -> [ALERT,    0, 1, 1];"ALERT via TRIP_HIGH

[x, H, x, x, x, c] -> [DISARMED, 1, 1, 0];
[H, L, x, x, x, c] -> [READY,    1, 1, 0];
[x, L, L, L, x, c] -> [WARNING,  0, 1, 0];
[x, L, L, H, x, c] -> [ARMED,    1, 1, 0];
[x, L, L, H, x, c] -> [ARMED,    1, 1, 0];"wait till tripped
[x, L, L, L, x, c] -> [ALERT,    0, 1, 1];"ALERT via TRIP_LOW

[x, L, x, x, H, c] -> [ALERT,    0, 1, 1];"wait for timeout
[x, L, x, x, L, c] -> [ALARM,    1, 0, 0];"ALARM when timeout
[x, L, x, x, x, c] -> [ALARM,    1, 0, 0];"Honk till IGNITION
[x, H, x, x, x, c] -> [DISARMED, 1, 1, 0];
end  AUTO_ALARM;
```

ОБРАБОТКА ПОЛНОГО ПРОЕКТА

Теперь мы готовы обработать наш проект. Автомат почти всегда требует логической редукции, поэтому мы используем параметр "r2" чтобы определить полную логическую редукцию. Мы будем использовать также параметры "-f" и "-v" чтобы включить в наш файл документации тестовые вектора и карту перемычек, только для демонстрации. Когда проект будет завершен, эти параметры могут быть включены в сам исходный файл, чтобы избежать ввода (и запоминания) их каждый раз когда проект обрабатывается.

Считая, что наш исходный файл записан на диске под именем "ALARM.ABL", мы вводим команды, написанные ниже. Для этого примера, АБЕЛЬ создает загрузочный файл программатора под именем U1.JED, состоящий из данных программирования и тестирования в формате JEDEC, файл документации под названием ALARM.DOC, файл предварительного разбора, названный ALARM.LST, и файл симуляции с именем ALARM.SIM.

```
A>ABEL ALARM -r2 -f -v  (press Enter (Return) to begin)
A>echo off
ABEL Version 1.00
PARSE Version 1.00 Copiright (c) 1983,1984 Data I/O,Redmond,WA
module AUTO_ALARM
PARSE complete. Time: 19 seconds
TRANSFOR Version 1.00 Copiright (c) 1983,1984 Data I/O,Redmond,WA
module AUTO_ALARM .......
TRANSFOR complete. Time: 18 seconds
REDUCE Version 1.00 Copiright (c) 1983,1984 Data I/O,Redmond,WA
module AUTO_ALARM
_device U1
Simple Reductions
...
PRESTO Algorithm
.......
REDUCE complete. Time: 28 seconds
.
FUSEMAP Version 1.00 Copiright (c) 1983,1984 Data I/O,Redmond,WA
module AUTO_ALARM
_device U1
20 of 64 terms used
FUSEMAP complete. Time: 15 seconds
SIMULATE Version 1.00 Copiright (c) 1983,1984 Data I/O,Redmond,WA
module AUTO_ALARM
19 of 19 vectors in U1 passed
SIMULATE complete. Time: 17 seconds
DOCUMENT Version 1.00 Copiright (c) 1983,1984 Data I/O,Redmond,WA
module AUTO_ALARM
_device U1
DOCUMENT complete. Time: 8 seconds
A>
```

АБЕЛЬ полностью обрабатывает проект автомата, включая логическую редукцию, приблизительно три минуты. При использовании жесткого диска или диска в памяти обработка будет быстрее.

ИЗУЧЕНИЕ ФАЙЛА ДОКУМЕНТАЦИИ

Файл документации ALARM.DOC состоит по умолчанию из двух секций документационного файла, плюс еще две, которые включены нашими ключами -f и -v:

* Конечные логические уравнения
* Изображение и расположение выводов микросхемы
* Карта перемычек
* Тестовые вектора

Первая секция файла ALARM.DOC содержит логические уравнения, реализующие проект, в их конечной форме после преобразования и логической редукции. Заметьте, что полярность логики определяется автоматически для выбранного устройства (все выходы активны низким потенциалом для нашей 16R4) (1556ХП4):

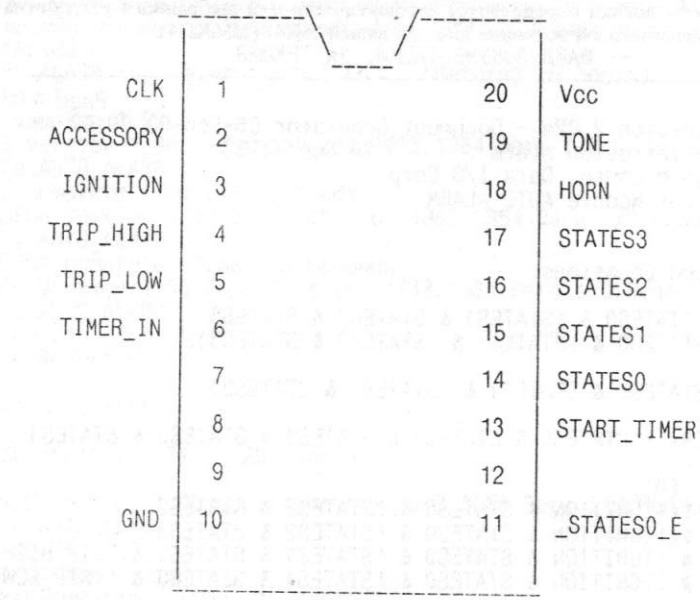
-- ФАЙЛ ДОКУМЕНТАЦИИ, 1я СЕКЦИЯ

```
                                                              Page 1
ABEL(tm) Version 2.02a - Document Generator 06-Feb-90 09:07 AM
Automobile intrusion alarm          29 Sep. 1983
ABEL design example, Data I/O Corp.
Equations for Module AUTO_ALARM

Device U1
     Reduced Equations:

TONE = !(!STATES0 & !STATES1 & STATES2 & STATES3
       # STATES0 & !STATES1 & !STATES2 & STATES3);

HORN = !(STATES0 & STATES1 & !STATES2 & STATES3);

START_TIMER= !(STATES0 & STATES1 & STATES3 # STATES2 & STATES3);

STATES3:= !(0);
STATES2:= !(!IGNITION & STATES0 & !STATES2 & STATES3
STATES2:= !(!IGNITION & STATES0 & !STATES2 & STATES3
          # !IGNITION & STATES0 & !STATES1 & STATES3 & TRIP_HIGH
          # !IGNITION & STATES0 & !STATES1 & STATES3 & !TRIP_LOW);

STATES1:= !(!IGNITION & !STATES0 & STATES2 & STATES3 & TRIP_HIGH
          # !IGNITION & !STATES0 & STATES2 & STATES3 & !TRIP_LOW
          # !IGNITION & STATES0 & !STATES1 & STATES3 & TIMER_IN
          # !IGNITION & !STATES1 & STATES2 & STATES3);

STATES0:= !(ACCESSORY & STATES0 & STATES1 & STATES2 & STATES3
          # !IGNITION & !STATES0 & STATES2 & STATES3 & TRIP_HIGH
          # !IGNITION & !STATES0 & STATES2 & STATES3 & !TRIP_LOW
          # !IGNITION & !STATES0 & STATES1 & STATES2 & STATES3);
          # !IGNITION & !STATES0 & STATES1 & STATES2 & STATES3);

Следующая секция ALARM.DOC это изображение нашей микросхемы U1 с
идентифицированными именами выводов:
```

ABEL(tm) Version 2.02a - Document Generator 06-Feb-90 09:07 AM
Automobile intrusion alarm 29 Sep. 1983
ABEL design example, Data I/O Corp.
Chip diagram for Module AUTO_ALARM

Device U1

P16R4



Секция карты перемычек файла документации содержит карты перемычек для каждого устройства в проекте. Карта перемычек является графическим представлением матрицы программируемых перемычек для устройства, адаптированным к логическим диаграммам, поставляемым производителем устройства. Номера первых перемычек и разрывы через каждые 10 перемычек показаны для вычисления номера определенной перемычки. Секция карты перемычек файла ALARM.DOC требует нескольких страниц и они показаны ниже.

ABEL(tm) Version 2.02a - Document Generator 06-Feb-90 09:07 AM
Automobile intrusion alarm 29 Sep. 1983
ABEL design example, Data I/O Corp.
Fuse Map for Module AUTO_ALARM
Device U1

```
      0          10         20         30
    0: ---------- ---------- ---------- --
   32: ---------- X---X----X ---X------ --
   64: ---------- X----X---X --X------- --
   96: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
  128: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
  160: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
  192: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
  224: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
  256: ---------- ---------- ---------- --
  288: ---------- X----X--X- --X------- --
  320: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
  352: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
  384: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
  416: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
  448: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
  480: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
  512: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
  544: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
  576: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
  608: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
  640: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
  672: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
  704: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
  736: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
  768: -----X---- X----X---- --X------- --
  800: -----X--X- X--------X --X------- --
  832: -----X---- X--X-----X --X------- --
  864: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
  896: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
  928: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
  960: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
  992: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
 1024: -----X--X- X---X----- ---X------ --
 1056: -----X---- X--XX----- ---X------ --
 1088: -----X---- X-----X--X --X------- --
 1120: -----X---- X---X----X ---------- --
 1152: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
 1184: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
 1216: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
 1248: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
 1280: X--------- X---X---X- --X------- --
 1312: -----X--X- X---X----- ---X------ --
```

```
1344: -----X---- X--XX----- ---X------ --
1376: -----X---- X---X---X- ---X------ --
1408: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
1440: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
1472: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
1504: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
1536: ---------- ---------- ---------- --
```



```
ABEL(tm) Version 2.02a - Document Generator 06-Feb-90 09:07 AM
Automobile intrusion alarm          29 Sep. 1983
ABEL design example, Data I/O Corp.
Fuse Map for Module AUTO_ALARM
Device U1
1568: ---------- X-------X- --X------- --
1600: ---------- X---X----- ---------- --
1632: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
1664: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
1696: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
1728: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
1760: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
1792: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
1824: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
1856: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
1888: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
1920: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
1952: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
1984: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
2016: XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XX
```

Секция тестовых векторов файла документации является форматированным списком тестовых векторов из исходного файла. Вектора были адаптированы для большей схожести с форматом JEDEC для передачи тестовых векторов.

Тестовые вектора печатаются как [входной вектор] -> [выходной вектор], со всеми выводами устройства показанными в последовательном порядке для обоих векторов. Состояния, определенные входным вектором, результируют в выходы, перечисленные в выходном векторе, когда приложены к устройству запрограммированному картой перемычек, полученной из обработки логических описаний.

-- ФАЙЛ ДОКУМЕНТАЦИИ, 4я СЕКЦИЯ


```
ABEL(tm) Version 2.02a - Document Generator 06-Feb-90 09:07 AM
Automobile intrusion alarm          29 Sep. 1983
ABEL design example, Data I/O Corp.
Test Vectors for Module AUTO_ALARM
Device U1
   1 [CX1X XX-- ---- ---- ----] -> [---- ---- ---- LHHH HHH-];
   2 [C10X XX-- ---- ---- ----] -> [---- ---- ---- LLHH HHH-];
   3 [CX01 1X-- ---- ---- ----] -> [---- ---- ---- LLLH HHL-];
   4 [CX1X XX-- ---- ---- ----] -> [---- ---- ---- LHHH HHH-];
   5 [C10X XX-- ---- ---- ----] -> [---- ---- ---- LLHH HHH-];
   6 [CX00 0X-- ---- ---- ----] -> [---- ---- ---- LLLH HHL-];
```



```
   7 [CX00 0X-- ---- ---- ----] -> [---- ---- ---- LLLH HHL-];
   8 [CX00 1X-- ---- ---- ----] -> [---- ---- ---- LHLH HHH-];
   9 [CX01 1X-- ---- ---- ----] -> [---- ---- ---- HHLL HHL-];
  10 [CX1X XX-- ---- ---- ----] -> [---- ---- ---- LHHH HHH-];
  11 [C10X XX-- ---- ---- ----] -> [---- ---- ---- LLHH HHH-];
  12 [CX00 0X-- ---- ---- ----] -> [---- ---- ---- LLLH HHL-];
  13 [CX00 1X-- ---- ---- ----] -> [---- ---- ---- LHLH HHH-];
  14 [CX00 1X-- ---- ---- ----] -> [---- ---- ---- LHLH HHH-];
  15 [CX00 0X-- ---- ---- ----] -> [---- ---- ---- HHLL HHL-];
  16 [CX0X X1-- ---- ---- ----] -> [---- ---- ---- HHLL HHL-];
  17 [CX0X X0-- ---- ---- ----] -> [---- ---- ---- LHHL HLH-];
  18 [CX0X XX-- ---- ---- ----] -> [---- ---- ---- LHHL HLH-];
  19 [CX1X XX-- ---- ---- ----] -> [---- ---- ---- LHHH HHH-];

end of module AUTO_ALARM
```

ИЗУЧЕНИЕ ФАЙЛА СИМУЛЯЦИИ

Частичное тестирование файла симуляции ALARM.SIM показано ниже. формат аналогичен формату для примера Basic Logic, только теперь присутствует строка "## Clock ##" чтобы индицировать переход из текущего в следующее состояние.

-- ФАЙЛ СИМУЛЯЦИИ, УРОВЕНЬ СЛЕЖЕНИЯ 1

```
Simulate device U1, type 'P16R4'

Vector 1
Output      [                   ] -> [..........ZLLLLLHH.]
## Clock ##
Output      [                   ] -> [..........ZLHHHHHH.]
V 0001      [C0100000000000000000] -> [..........NLHHHHHH.]

Vector 2
Output      [                   ] -> [..........ZLHHHHHH.]
## Clock ##
Output      [                   ] -> [..........ZLLHHHHH.]
V 0002      [C1000000000000000000] -> [..........NLLHHHHH.]

Vector 3
Output      [                   ] -> [..........ZLLHHHHH.]
## Clock ##
Output      [                   ] -> [..........ZLLLHHHL.]
V 0003      [C0011000000000000000] -> [..........NLLLHHHL.]

Vector 4
Output      [                   ] -> [..........ZLLLHHHL.]
## Clock ##
Output      [                   ] -> [..........ZLHHHHHH.]
V 0004      [C0100000000000000000] -> [..........NLHHHHHH.]
```

ИЗУЧЕНИЕ ФАЙЛА ПРЕДВАРИТЕЛЬНОГО РАЗБОРА

Часть файла предварительного разбора показана ниже. Этот файл является представлением того, что АБЕЛЬ на самом деле обрабатывает. Он показывает номер строки исходного файла и может содержать все текстовые подстановки, вызванные макросами, директивами и т.п. если вы пожелаете. Файл предварительного разбора очень полезен, когда отлаживаются синтаксические ошибки, так как сообщения об ошибках и индикаторы их положения могут быть включены в файл, но поскольку наш проект не содержит ошибок, мы просматриваем этот файл только для ознакомления.

-- ФАЙЛ ПРЕДВАРИТЕЛЬНОГО РАЗБОРА

```
0001 module     AUTO_ALARM;
0002
0003 title
0004 'Automobile intrusion alarm          29 Sep. 1983
0005 ABEL design example, Data I/O Corp.'
0006
0007      U1    device      'P16R4';"Declare our target device
0008
0009 " Constants:
0010
0011      ON          =     1;
0012      OFF         =     0;
0013      H           =     1;
0014      L           =     0;
0015      x           =     .X.;" ABEL 'don't care' symbol
0016      c           =     .C.;" ABEL 'clocking input' symbol
0017
0018 "State definitions (i.e.binary value of register STATES):
0019
0020      DISARMED    =     ^b1111;"These are arbitrary binary
0021      READY       =     ^b1110;"state numbers.  See text.
0022      WARNING     =     ^b1100;
0023      ARMED       =     ^b1101;
0024      ALERT       =     ^b1001;
0025      ALARM       =     ^b1011;
0026
0027 " Pin names:
0028
0029      ACCESSORY   pin   2;          " Input pins
0030      IGNITION    pin   3;
0031      TRIP_HIGH   pin   4;
0032      TRIP_LOW    pin   5;
0033      TIMER_IN    pin   6;
0034
```


ЗАГРУЗКА ПРОГРАММИРУЮЩИХ ДАННЫХ

Когда АБЕЛЬ определил состояния перемычек и тестовые вектора, требуемые для того чтобы выполнить и проверит наше сторожевое устройство, мы можем загрузить эту информацию в логический программатор.

ЗАКЛЮЧЕНИЕ

В этом примере разработки мы рассмотрели простой проект от его первоначальной концепции до программирования и тестирования PAL. Особенности АБЕЛя, которые мы раскрыли, включают в себя:

* Логический проект с диаграммой состояний
* Константы поименование выводов
* Определение и использование групп
* Определение и использование макросов
* Проверка проекта тестовыми векторами
* Обработка полного проекта
* Изучение файлов документации и симуляции
* Загрузка программирующих и тестовых данных

СЛОВАРЬ АБЕЛЯ

АВТОМАТ

Тактированный (регистровый) логический проект, который изменяет состояние в зависимости от текущего состояния и/или состояния избранных входных сигналов. Состояние автомата определяется как двоичное число сформированное регистровыми выходами.

ASCII ФАЙЛ

Файл, который содержит печатные символы ASCII (от 20 до 1E шестнадцатеричные) и три управляющих кода Carriage Return, Line Feed, and Form Feed.

ГРУППА

Набор переменных (имен выводов, узлов и пр.) которые обрабатываются как целое. Например, если SET определен как группа [A,B,C,D], написание !SET будет означать дополнение каждого члена группы, т.е. !A, !B, !C, !D.

ДИАГРАММА СОСТОЯНИЙ

Метод определения тактированного (регистрового) логического проекта описанием переходов из одного состояния в следующее. Каждое состояние дается числом и операторы описывают в которое состояние нужно перейти и при каких условиях.

ЗАГРУЗОЧНЫЙ ФАЙЛ ПРОГРАММАТОРА

Выходной файл АБЕЛя содержащий информацию о программировании и тестировании в формате JEDEC для загрузки в программатор.

ИСХОДНЫЙ ФАЙЛ

Текстовый файл созданный разработчиком, который описывает какие функции должен выполнить языковой процессор АБЕЛЬ. Языковой процессор читает исходный файл и преобразует его в состояния перемычек и тестовые вектора так, как требуется чтобы запрограммировать и протестировать логический проект.

КАРТА ПЕРЕМЫЧЕК

Графическое представление матрицы перемычек программируемого устройства, адаптированное к логическим диаграммам производителя. Типично используется "X" чтобы представить нетронутую перемычку и "-" для запрограммированной перемычки.

КЛЮЧЕВЫЕ СЛОВА

Последовательность символов, которые языковой процессор интерпретирует как единый символ с фиксированным значением. В качестве примера можно привести "title", "truth_table", и "end". Пользователь не может использовать ключевые слова как имена переменных.

ЛОГИЧЕСКАЯ РЕДУКЦИЯ

Процесс применения булевских алгебраических манипуляций к логическим выражениям для того чтобы ограничить избыточные термы. В качестве примера, A & B # !A & B редуцируются к просто B. Существует несколько возможных уровней редукции, от тривиальных (ограничение A&0 и пр.) до требующих времени математически полных решений. АБЕЛЬ использует быстрый алгоритм, который приближается к полной редукции.

ЛОГИЧЕСКОЕ ОПИСАНИЕ

Это секция внутри абелевского модуля, которая описывает требуемые логические функции которые должны быть выполнены запрограммированным устройством. Логическое описание может состоять из булевских уравнений, диаграмм состояний или таблиц истинности.

МАКРО

Определенный пользователем символ в исходном файле, который заставляет АБЕЛЬ подставлять блок текста вместо этого символа в процессе обработки.

МОДУЛЬ

Главная единица абелевского исходного файла. Обычно модуль содержит абелевский исходный код для полного логического проекта, который может включать несколько устройств. Несколько модулей могут быть объединены в единый исходный файл.

ОПЕРАТОР

Это символ для логических или арифметических операций, такой как "&" для логического И, "#" для логического ИЛИ, и т.п.

СИМУЛЯТОР

Один из программных компонентов языкового процессора АБЕЛЬ. Симулятор строит внутреннюю модель выбранного устройства, запрограммированного чтобы выполнить определенное логическое описание. Он затем прилагает логические уровни, определенные тестовыми векторами к модели устройства и проверяет, что получились желаемые выходные вектора.

ТАБЛИЦА ИСТИННОСТИ

Метод определения логического проекта перечислением состояний входных выводов, за которыми следуют состояния выходных выводов.

ТЕСТОВЫЕ ВЕКТОРА

Список условий зависящих от выводов устройства, которое должно быть протестировано. К входным выводам прикладываются определенные условия и выходные выводы проверяются на соответствие ожидаемым условиям. Тестовые вектора определяются разработчиком. АБЕЛЬ использует тестовые вектора и логическое описание чтобы симулировать программируемое устройство для проверки проекта, а также переформирует тестовые вектора в формат JEDEC и включает их в загрузочный файл программатора.

УЗЕЛ

Это абелевская терминология для псевдовывода, который на самом деле находится внутри устройства. Узлы управляются уровнями сигналов тем же образом, что и выводы, но они недоступны снаружи устройства. Логические устройства обеспечивающие программируемые типы триггеров (D, J-K, S-R, и т.д.) являются примером использования узлов; определенные перемычки, определяющие тип триггера, доступны только при определении сопутствующих узлов.

ФОРМАТ JEDEC

Формат передачи данных для программируемой логики, стандарт JEDEC JC-42.1-81-62. Стандарт полностью описан в руководстве "Data I/O's ABEL and PLDS".