Departamento de Electrónica y Computadores
Grupo de Diseño y Verificación de Circuitos Integrados (D.Y.V.C.I.).

Manual de herramientas de diseño lógico en SIS.

1. Introducción.

Las herramientas que se describen a continuación están integradas dentro del paquete SIS (Sequential Circuits Synthesis), aunque también pueden usarse desde fuera de él. SIS es un paquete para el diseño de circuitos digitales que contiene herramientas para la síntesis de circuitos digitales. SIS ha sido realizado para el sistema operativo UNIX (disponible en ftp://ic.eecs.berkeley.edu/pub/Sis/) pero existe una versión para el sistema operativo MS-DOS, disponible en http://www.cs.bris.ac.uk/~paul/dos_sis/, con pequeñas limitaciones, generada a partir del programa de conversión djgpp. Esta versión funciona bien para el MS-DOS de los sistemas operativos Windows 95 y 98, pero no funciona bien cuando el sistema operativo del ordenador es el Windows XP. Para Windows XP hay que utilizar un parche disponible en http://ee.unipr.it/ciampolini/sis/sis.html.

Entre las herramientas incluidas en SIS se encuentran manipuladores de expresiones lógicas que se utilizan en el diseño de circuitos combinacionales, tales como Espresso que es un minimizador de expresiones lógicas en dos niveles, o como operaciones para síntesis multinivel como factorización (factor), descomposición (decomp), colapsado (collapse), substitución (resub), extractor de cubos comunes y de múltiples cubos comunes (gcx y gkx), obtención de divisores comunes (fx), etc. También existen opciones típicas del diseño de circuitos secuenciales, como son la minimización de estados (state_minimize, stamina por defecto o sred) y la asignación secundaria de estados (state_assign, nova por defecto ó jedi, ó one_hot). SIS también realiza la tarea de mapear las expresiones lógicas, dado un catálogo concreto de dispositivos (puertas lógicas y flip-flops), modificándolas en expresiones directamente implementables por los dispositivos del catálogo (read_library para seleccionar el catálogo y map para realizar la implementación). Además, SIS contiene utilidades que permiten simular el comportamiento del circuito (simulate), comprobar si dos descripciones son equivalentes en circuitos combinacionales (verify) y secuenciales (verify_fsm), hacer una estimación de la potencia disipada por el circuito (power_estimate) o encontrar una secuencia de prueba para fallos en el circuito (atpg).

SIS permite varios formatos de descripción de expresiones lógicas y circuitos. El formato propio de SIS es el formato blif (Berkeley Logic Interchange Format), que es relativamente complejo y permite hacer relación a alguna característica eléctrica (cargabilidad) o temporal. También se permiten otros formatos como el pla, típico de Espresso, para definir de forma sencilla expresiones lógicas combinacionales en dos niveles, o el formato kiss, que permite definir máquinas de estados finitos. Dentro de SIS, las expresiones lógicas se muestran en forma SOP, donde los nudos utilizados se muestran con el nombre fijado en la descripción de entrada, o como IN_X para las entradas sin nombre (X representa el índice de la entrada),

{OUT_X} para las salidas sin nombre y LatchOut_vX para las variables de estado sin nombre, y [XXX] para los nudos intermedios (XXX es el índice del nudo); se utiliza el carácter **‘** para indicar la complementación de un literal y el carácter **+** para la operación OR.

2. Minimización en dos niveles de funciones de conmutación: Espresso.

2. 1. Descripción de las funciones de conmutación.

La descripción de las funciones de conmutación en Espresso se realiza mediante cubos o términos productos. En la descripción hay una serie de comandos básicos (nº de entradas, nº de salidas y términos productos), a los que se pueden añadir otros comandos optativos. La descripción básica tiene el siguiente formato (los comandos optativos se muestran entre corchetes:

.i N (nº de entradas)
.o M (nº de salidas)
[**.p** P]
Relación de términos producto
[**.e**]

Los términos producto se describen como una serie de líneas con un formato:

XXX… YYY…. (**01- 1-0**, por ejemplo)

XXX… representa un término producto, siendo cada X el valor que toma cada variable de entrada en el término producto. X puede tomar valor 1 (la entrada correspondiente está en el término producto sin complementar), 0 (la entrada correspondiente está en el término producto complementada) ó - (para indicar que el término producto no depende de la variable de entrada, la entrada no está en el término producto). Las N entradas se referencian con nombres v0 (la entrada más a la izquierda), v1, v2, …,v_{N-1}, (la entrada más a la derecha).

Los valores en las salidas son **YYY…**, donde Y representa lo que produce un término producto en cada salida del circuito. Y puede tomar el valor 1 (el término producto produce 1s en la salida), 0 (el término producto produce 0s en la salida), - (el término producto produce "don't cares" en la salida) y ~ (el término producto no produce nada en la salida, es decir no se tiene en cuenta). Estos valores pueden ser modificados por el comando **.type** que se comenta en el siguiente apartado, y que permite activar el valor (el valor se comporta normalmente) o desactivarlo (el valor se comporta como si fuese ~); por defecto los valores 1 y – están activados, y el valor 0 está desactivado. Las M salidas se referencian con nombres $v_N.0$ (la salida más a la izquierda, N es el número de entradas), $v_N.1$, $v_N.2$, …,$v_{N.M-1}$, (la salida más a la derecha.

Opcionalmente, los términos productos se indican entre los comandos **.p** P (donde P es el número de términos productos) y **.e**.

El formato por defecto en el que se muestra el resultado de la minimización es similar al formato de entrada, (nº de entradas, nº de salidas y términos productos en formato XXX… YYY…), donde los términos producto corresponden al resultado de la minimización y se muestran entre los comandos **.p** y **.e**. Los XXX… de los términos productos se comportan como en el formato de descripción, pero los términos YYY… sólo toman los valores 1 (el término producto pertenece a la salida correspondiente del circuito) y 0 (el término producto no pertenece a la salida correspondiente del circuito). El orden y el nombre de referencia de las entradas (v0, v1, …) y de las salidas (vN.0, vN.1, …) en los términos producto son idénticos a los utilizados en el fichero de entrada.

2.2. Comandos de caracterización de las funciones de conmutación.

Existen otros comandos que permiten definir características de las funciones de conmutación o de su descripción. Estos comandos deben indicarse antes de la definición de los términos producto de las funciones. A continuación se citan dos comandos frecuentemente usados: **.type** y **.phase**.

El comando **.type** se utiliza para definir los tipos de datos que producen los términos productos. Para definir un circuito digital los términos productos se incluyen en tres tipos de conjuntos: el ON (los 1s), el OFF (los 0s) y el DC ("don't cares"). Dentro del fichero de entrada de Espresso puede determinarse los conjuntos que se indican en la descripción mediante la orden:

.type [f | r | fd | fr | rd | fdr] (entre corchetes se muestran las opciones posibles *f*, *r*, *fr*, etc)

donde las opciones indican los conjuntos que están activos al leer la descripción (*f* el ON, *d* el DC y *r* el OFF). Por defecto, está activa la opción *fd*, de forma que al leer el fichero de entrada sólo se reconocen los términos productos que producen 1 (ON) y - (DC) en las salidas, y no se toman en cuenta los que producen 0 (OFF). La orden **.type fdr** permite reconocer todos los conjuntos 1 (ON), 0 (OFF) y - (DC), mientras que, por ejemplo, la opción **.type r** permite reconocer sólo a los términos que generan 0 (OFF). Cuando el valor ~ aparece en un término producto indica que el término producto en la salida correspondiente no es ni ON ni OFF ni DC, cuando un valor no aparece en el comando **.type**, todos los valores de ese tipo se comportan como ~ .
Al aplicarse Espresso, el programa lee de la descripción los conjuntos ON, OFF y DC que estén activos, donde al menos uno de los conjuntos ON ú OFF deben estar activos. El programa genera un conjunto no activo (DC sólo si ON y OFF están activos; ON ú OFF, el no activo, en los demás casos) en función de los conjuntos activos leídos.

El comando **.phase** permite definir el nivel de aserción de cada una de las salidas en la minimización, mediante la orden:

.phase PPP…

donde cada P toma valor 1 si se desea que la señal de salida es asertada alta (se minimiza el conjunto ON), y 0 si se desea que la señal de salida sea asertada baja (se minimiza el conjunto OFF). Se deben incluir M valores P (uno por cada salida). Cuando no se usa este comando se presupone que todas las salidas están asertadas altas.

2.3. Opciones de ejecución de Espresso.

La ejecución básica del programa se realiza desde el sistema operativo MS-DOS mediante la orden básica: **espresso *mi_fichero***, donde *mi_fichero* contiene la descripción de entrada. Esta ejecución realiza un minimización conjunta de todas las funciones de conmutación (o salidas) definidas. Los resultados de la ejecución de Espresso aparecen en el terminal gráfico, para almacenarlos en un fichero, se debe utilizar la opción de redireccionamiento del sistema operativo UNIX: **espresso *mi_fichero* > *fich_salida***. Por defecto, este comando genera una minimización de tipo heurístico con un método iterativo de aproximación a la función óptima, pero que no garantiza que en todos los casos se obtenga la función mínima. El criterio de coste que se utiliza es el de conseguir el menor número de términos producto en la solución final, aunque se incluye un procedimiento final para reducir el número de literales.

Espresso tiene algunas opciones de ejecución. Para obtener una ayuda en pantalla de estas opciones de Espresso hay que introducir la orden **espresso -h**. Las opciones de ejecución más interesantes en cuanto a la ejecución tienen un formato del tipo:

espresso -D*opción* -o*formato* *mi_fichero*

donde *opción* representa las diferentes opciones de ejecución de Espresso y *formato* representa el formato esperado en la salida.

Algunas de las opciones de ejecución de Espresso que se utilizan son:

- **Dcheck**: comprueba si las funciones de conmutación están mal definidas. No realiza la minimización. Conviene situar el comando **.type** al valor ***fdr*** con esta opción.
- **Dexact**: realiza una minimización exacta, que garantiza encontrar la solución óptima. En problemas grandes el tiempo de cómputo requerido es alto.
- **Dopo**: realiza una minimización conjunta en la que se permite asignar la fase o la aserción de las señales de cada salida para mejorar la minimización.
- **Dso**: realiza una minimización para cada salida por separado.
- **Dso_both**: realiza una minimización para cada salida por separado, utilizando la fase o la aserción de cada señal que minimice la optimización de cada salida.

El formato de salida de Espresso tiene varias opciones posibles. Al igual que en la definición de **.type** se pueden indicar los valores **-o**[f | d | r | fd | fr | dr | fdr], que indica los conjuntos que se van a mostrar: f el ON, r el OFF y d el DC. Por defecto, se tiene la opción **-of** que muestra el conjunto ON resultante de la minimización. Otra opción que se puede utilizar es **-oeqntott**,

que muestra la salida en un formato algebraico, en el que la operación AND se representa por **&**, OR por **|**, y NOT por **!**.

2.4. Ejemplos.

Ejemplo A. Un circuito sumador completo de un bit (dos operandos de entrada de un bit y el carry-in) puede ser definido por el siguiente fichero de entrada para Espresso en el que se indica claramente la tabla de verdad de cada salida (de suma S y de carry-out Cout).

```
.i 3
.o 2
000 00
001 01
010 01
011 10
100 01
101 10
110 10
111 11
```

En cada fila de la definición de la tabla de verdad del problema se muestran por columnas primero las tres entradas A, B, Ci, que son funcionalmente equivalentes, y que en los resultados de Espresso, se llamarían v0, v1 y v2, respectivamente. Después se muestra el valor de las salidas Cout y S (en este orden), que en los resultados de Espresso, se llamarían v3.0 y v3.1. La ejecución de Espresso da como resultado:

```
.i 3
.o 2
.p 7
100 01
010 01
001 01
111 01
-11 10
1-1 10
11- 10
.e
```

Esta salida lo que indica que los primeros cuatro términos productos pertenecen a la segunda salida, y los tres últimos a la primera. Así, los resultados son:

- Para S. v3.1 = v0v1'v2' + v0'v1v2' + v0' v1'v2 + v0v1v2
- Para Cout. v3.0 = v1v2 + v0v2 + v0v1.

Ejemplo B. El formato de entrada requerido para minimizar la función

$F(A, B, C) = \sum(1, 5, 6, 7) + \sum\phi(0, 3)$,

tomando como A la entrada v0 (izquierda), como B la entrada v1 (centro) y como C la entrada v2 (derecha) es:

```
.i 3
.o 1
001 1
101 1
110 1
111 1
000 -
011 -
```

En esta definición se tiene en cuenta que la opción de .type por defecto es *fd*, y sólo se reconocen los 1s y los - ("don't cares"), a partir de los cuales el programa obtiene los 0s. El resultado de la ejecución de Espresso es:

```
.i 3
.o 1
.p 2
11- 1
--1 1
.e
```

Por tanto, F = v0v1 + v2 (F = A B + C).

Ejemplo C. También puede definirse una función directamente mediante cubos en vez de mediante minterms. El conjunto de funciones:

F1 = A'B'C + ABD + A'BD + B'C'
F2 = ABD + BC' + B'C'
F3 = A'B'C + A'BD + BC'

se describe utilizando el formato v0v1v2v3 v4.0v4.1v4.2 (ABCD F1F2F3) como:

```
.i 3
.o 3
.p 5
001- 101
11-1 110
01-1 101
-00- 110
-10- 011
.e
```

Como por defecto el comando **.type** es *fd* sólo se leen los 1s y los – (aunque no hay -). Los 0s no se leen (equivalen al valor ~) y los genera automáticamente el programa Espresso.

3. Diseño lógico de circuitos secuenciales.

3.1. Introducción.

Existen varias tareas en el diseño lógico de circuitos secuenciales. El diseño lógico de un problema secuencial se plantea mediante una máquina de estados finitos (FSM) que puede representarse mediante una tabla de estados, donde los estados se definen mediante un mnemónico. Las transiciones dentro de la tabla se representan en un formato en el que en el estado presente (PS) y con unas entradas determinadas, se produce una salida y se llega a un estado siguiente (NS), normalmente cuando el sistema es activado por un flanco de reloj.

La definición de la tabla, debido al planteamiento del problema que puede ser muy complejo o no estar perfectamente definido, y debido a la existencia de salidas o de estados indiferentes ("don't cares"), puede contener más estados de los realmente necesarios, ya que pueden existir dentro de la tabla los que se llaman estados compatibles. Una primera tarea consiste en minimizar la tabla de estados, sustituyéndola por otra equivalente con el menor número de estados. Esta tarea la realiza en SIS los programas STAMINA ó SRED.

Otra tarea de diseño lógico consiste en codificar los mnemónicos de los estados por valores binarios correspondientes a las variables de estado necesarias. Dependiendo de la codificación será necesaria una lógica combinacional distinta para realizar los decodificadores del siguiente estado y de salida. Existen herramientas que realizan la asignación secundaria, con el objetivo de reducir en lo posible el tamaño de dichos decodificadores. Se disponen de dos herramientas de asignación secundaria: NOVA, orientada a un realización de los decodificadores mediante lógica de dos niveles, y JEDI orientada a la realización de los decodificadores mediante lógica multinivel.

Una vez hecha la asignación secundaria, el resto del problema es puramente combinacional, en especial si se suponen flip-flops de tipo D, y se pueden utilizar las utilidades de diseño en dos niveles o multinivel para calcular las expresiones lógicas que definen los decodificadores del siguiente estado y de salida.

3.2. Definición de máquinas de estados finitos (FSM).

Las máquinas de estados finitos se definen en un formato del tipo KISS. Kiss es una herramienta para la asignación secundaria de estados. El formato es similar al de Espresso, algunos de los comandos que se citan a continuación son optativos (entre corchetes) pero para evitar problemas de compatibilidad con SIS se suponen aquí obligatorios. El formato es:

.i N (nº de entradas)
.o M (nº de salidas)
.p P (nº de términos en la descripción de la FSM).
.s S (nº de estados)
[**.r** R (nombre del estado de reset o reinicio)]

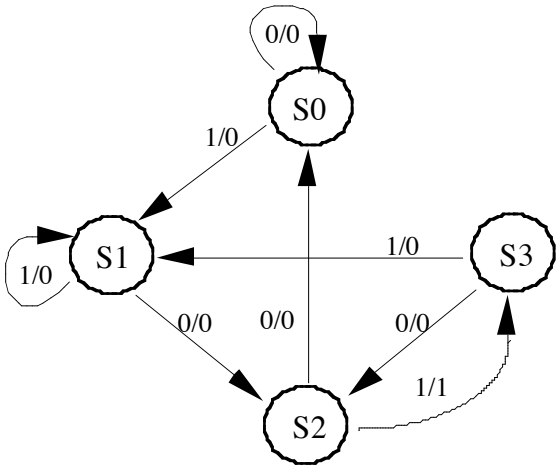
INP PS NS OUT (líneas con los términos de descripción de las transiciones en la FSM)
.e

Las líneas de descripción de la FSM representan las transiciones entre estados asociadas su tabla de estado. Con un formato típico de una máquina de Mealy, dichas líneas están descritas por cuatro campos:

• INP: valores lógicos aplicados a las entradas de la FSM: XXX…, donde X puede tomar el valor 0, 1 o - (don't care) como en Espresso.
• PS: nombre (string alfanumérico) del estado presente (o de partida) en la transición.
• NS: nombre (string alfanumérico) del siguiente estado (o de llegada) en la transición.
• OUT: valores que toman las salidas de la FSM para el estado actual (PS) y las entradas (INP): YYY…, donde Y puede tomar los valores 0, 1 ó -, como en Espresso.

Ejemplo. La FSM representada en el diagrama de estados se corresponde con la descripción de tipo kiss.

```
.i 1
.o 1
.s 4
.p 8
.r S0
0 S0 S0 0
1 S0 S1 0
0 S1 S2 0
1 S1 S1 0
0 S2 S0 0
1 S2 S3 1
0 S3 S2 0
1 S3 S1 0
.e
```



3.3. Minimización de estados: Stamina.

Stamina es una herramienta que minimiza el número de estados de una FSM incompletamente especificada atendiendo a sus clases de compatibilidad. El formato básico de aplicación del minimizador de estados es **stamina *mi_fichero***, donde *mi_fichero* contiene la descripción de la FSM. Además la herramienta permite otras opciones que se pueden observar en pantalla mediante la orden **stamina -h**.

Las opciones de Stamina son:

• **-s n**: Indica el tipo de algoritmo (n toma valores de 0 a 3) utilizado para encontrar los máximos compatibles de la tabla de estado. Por defecto n toma el valor 0 que indica un método exacto. Bajo los valores 1, 2, y 3 se utilizan métodos heurísticos.

• **-m n**: indica el tipo de algoritmo para generar (o mapear) la tabla de estados reducida a partir de los máximos compatibles de forma que se reduzca el coste de la implementación de la FSM. Stamina no toma como único objetivo la reducción del número de estados sino también la reducción del coste de la implementación de la FSM. El valor 0 indica un valor aleatorio, y los valores 1 y 4 (por defecto) métodos heurísticos (según el manual).
• **-v n**: indica el nivel de información que aparece en pantalla al ejecutar Stamina. Los números posibles son 0 (sólo tabla reducida), 1 (tabla reducida, estados compatibles utilizados en la reducción y estadísticas) y 2 (tabla reducida, estados compatibles utilizados en la reducción, estadísticas y máximos compatibles de la tabla inicial).
• **-o *salida***. Genera un fichero de salida donde se describe la tabla de estados reducida en formato Kiss.
• **-S**. Reduce, si es posible, el número de estados de la tabla inicial asociados a cada estado de la tabla reducida manteniendo una cobertura cerrada.

La aplicación de Stamina a la FSM del ejemplo anterior encuentra que hay dos estados compatibles (S1 y S3) que puede reducir en un único estado. La solución que genera en tres estados también se da en formato Kiss así:

```
.i 1
.o 1
.p 6
.s 3
.r S1
0 S0 S2 0
1 S0 S0 0
0 S1 S1 0
1 S1 S0 0
0 S2 S1 0
1 S2 S0 1
.e
```

3.4. Asignación secundaria de estados: Nova.

Nova es una herramienta que se puede utilizar para la codificación de los estados de una FSM en valores binarios de las variables de estado. El criterio de coste utilizado es la reducción de la lógica en dos niveles para la implementación del circuito. En realidad Nova es una herramienta más completa que permite describir en forma simbólica (mediante un nombre) no sólo los estados, sino también las entradas del circuito y, por ello, realizar asignaciones binarias a los estados y a las entradas.

En un formato de descripción de tipo Kiss de FSMs para indicar que la entradas se referencian mediante símbolos y no mediante valores binarios hay que indicar el comando *.symbolic input* antes de la descripción de la FSM.

La orden básica de ejecución de la herramienta es **nova *mi_fichero***, donde *mi_fichero* contiene la descripción de la FSM. Además, la herramienta permite otras opciones que se pueden observar en pantalla mediante la orden **nova -h**. Para guardar los resultados

producidos en un fichero de salida hay que utilizar la opción de redireccionamiento de pantalla típica del sistema operativo UNIX: **nova *mi_fichero* > *salida***, donde *salida* es el nombre del fichero donde se guardan los resultados.

Además, Nova tiene una serie de opciones que controlan su ejecución. Algunas de las opciones de Nova son:

• **-e xxx**. Selecciona el tipo de restricciones que se consideran y el tipo de algoritmo que se aplica, que se indican en *xxx*. Las restricciones que se consideran para la codificación óptima pueden ser de varios tipos: de las entradas, de las salidas, o de las entradas y de las salidas. El algoritmo puede ser exacto, o heurístico de dos tipos: greedy ó híbrido.
Las posibilidades son:
-e ig (por defecto). Restricciones de entradas y algoritmo greedy.
-e ih. Restricciones de entradas y algoritmo híbrido.
-e ie. Restricciones de entradas y algoritmo exacto.
-e ia. Restricciones de entradas y algoritmo basado en "simulated annealing".
-e ioh. Restricciones de entradas y salidas, y algoritmo híbrido. Generalmente da la mejor solución posible.
-e iov. Restricciones de entradas y salidas, y algoritmo híbrido.
-e ioh -y. Restricciones de salidas, y algoritmo híbrido.
-e h. Codificación de tipo one-hot.
-e r -n = x. Codificaciones aleatorias haciendo hasta x pruebas.
-e u. Toma las codificaciones de un fichero de nombre file.codes.

• **-i n**. Fuerza que la codificación de las entradas simbólicas se realice en n variables.
• **-s n**. Fuerza que la codificación de los estados se realice en n variables.
• **-r**. Intenta todas las rotaciones sobre los códigos.
• **-t**. Muestra durante la ejecución la descripción de los decodificadores en formato Espresso. Permite una implementación directa en dos niveles suponiendo flip-flops de tipo D.
• **-z xxx**. Indica que se debe asignar un código formado por 0s al estado de nombre xxx. No funciona cuando se utilizan las restricciones impuestas en las salidas.

La ejecución de Nova produce un resultado en pantalla (o redireccionado a un fichero), en el que se muestra bajo el comando **.code** el código binario asociado a cada estado. A continuación se da nombre a las entradas (**.inputs**), a las salidas (**.outputs**) y a las variables de estado mediante el comando **.latch *inp out valor***, donde *inp* es el nombre del nudo de entrada del flip-flop de tipo D correspondiente a la variable de estado, *out* es el nombre del nudo de salida y valor es el valor inicial del flip-flop. En el comando **.latch_order** se indica el orden de los flip-flops (por sus salidas) o, lo que es lo mismo, de las variables de estado para la codificación de los estados. También se muestra la definición de los decodificadores dada en formato blif. Para obtenerla en formato Espresso hay que usar la opción **-t** del programa nova, al hacerlo aparece una descripción tipo Espresso en la que se muestra la descripción de los decodificadores según las entradas y salidas utilizadas en los comandos .**inputs** y .**outputs**.

A continuación se muestra la solución de la aplicación de Nova sobre el ejemplo anterior. En esta solución se muestran solamente las líneas más relevantes.

```
.code S0 00          (codificaciones de los estados S0, S1 y S2 en dos variables de estado)
.code S1 10
.code S2 11
.inputs v0           (la entrada se llama v0, como en Espresso)
.outputs v3.2        (la salida se llama v3.2, como en Espresso)
.latch v3.0 v1 0     (flip-flop de entrada D v3.0 y de salida Q v0)
.latch v3.1 v2 0     (flip-flop de entrada D v3.1 y de salida Q v2)
.latch_order v1 v2 (en .code el valor de la izquierda se carga en el flip-flop v1, el de la derecha en el flip-flop v2)
(Resultados de los decodificadores en formato Espresso, en las columnas de izquierda a derecha: las entrada v0,
la salida del primer flip-flop v1, la salida del segundo flip-flop v2, la entrada del primer flip-flop v3.0; la entrada
del segundo flip-flop v3.1, la salida v3.2)
.i 3
.o 3
.p 3
1-1  001
00- 010
0--  100
.e
```