



Application Note BD-01

Using SBC Format on Vistalogic and VistaVision

Synopsis: Application Note 14 in the Toolbox Application Note Manual is incomplete. Since many test engineers are interested in using the **Surround-By-Complement** (SBC) format, this brief description will provide theory and application examples. Also included is a brief discussion of *swav2sbc*, a vector translator that generates SBC ready vectors.

The Quartet/Duo line of test systems has no “native” SBC capability - that is, SBC format is not provided as a standard option for programming tester drive formats. Because of the extreme flexibility of the driver circuits, however, SBC can be accomplished at the expense of halving the number of available timesets.

Specifically, since the test hardware provides for the mapping of data bits to timeset selection bits, we can achieve SBC by using different timesets, *per pin*, for SBC data ‘0’ and SBC data ‘1’. Yes, you can program timeset selection per pin by using special programming modes in the Quartet. We can begin by reviewing the register-level programming of tester pin edges.

The ‘c’-based language of the Quartet uses **symbolic constants** to select format. Typically these are specified as RZ, R1, DNRZ, etc. However, these symbolic constants merely represent the actual hex data that is programmed into the driver circuit. The meaning of these bits is reproduced in Appendix I. For the purposes of generating SBC, we need only the following:

```
#define SBC 0xNNNN  
#define SBC_INH 0xMMMM
```

where M and N are hex data. These definitions are found in the include file **lt.h**, which is referenced in the “#include <box.h>” statement present in all Toolbox-based test programs. Unfortunately, these symbolic constants are defined with incorrect data and will not produce SBC data *under any circumstances*. (The program example in Application Note 14 uses its own #define statement, which will work, but no mention of it is made in the text).

We can add new #define statements to the test program via:

```
#define SBCDRIVE 0x0980F (other values may work. Refer to Appendix I.)  
#define SBCMUX 0x0ACF
```

By using these new constants we can now program SBC format on input pins. The steps used are shown below:

- ✳️ Use the special symbolic constants to provide data-driven timeset switching per pin (see Figure 1);
- ✳️ For input only pins, use the **set_mux_edges()** function to create four edges per cycle;
- ✳️ Program one of the mux edge times to suppress one edge (using -1), leaving three edges per cycle;
- ✳️ Program the two timesets to set up the correct edges for rise, fall, and mux. One timeset will have $t_{\text{rise}} < t_{\text{fall}}$ (RZ), and the other will be programmed with $t_{\text{rise}} > t_{\text{fall}}$ (R1). See Figure 2 for more information..

The following program code is excerpted from the file sbc245.c, available from Cre-
dence Training for testing a 74ACT245 device:

```
/*
*****
245 Octal Bus Buffer
sbc245.c
This program shows the use of SBC format for input pins AND bi-directional pins
Author: Peter Lindholm, Applications Engineer, Integrated Systems Test

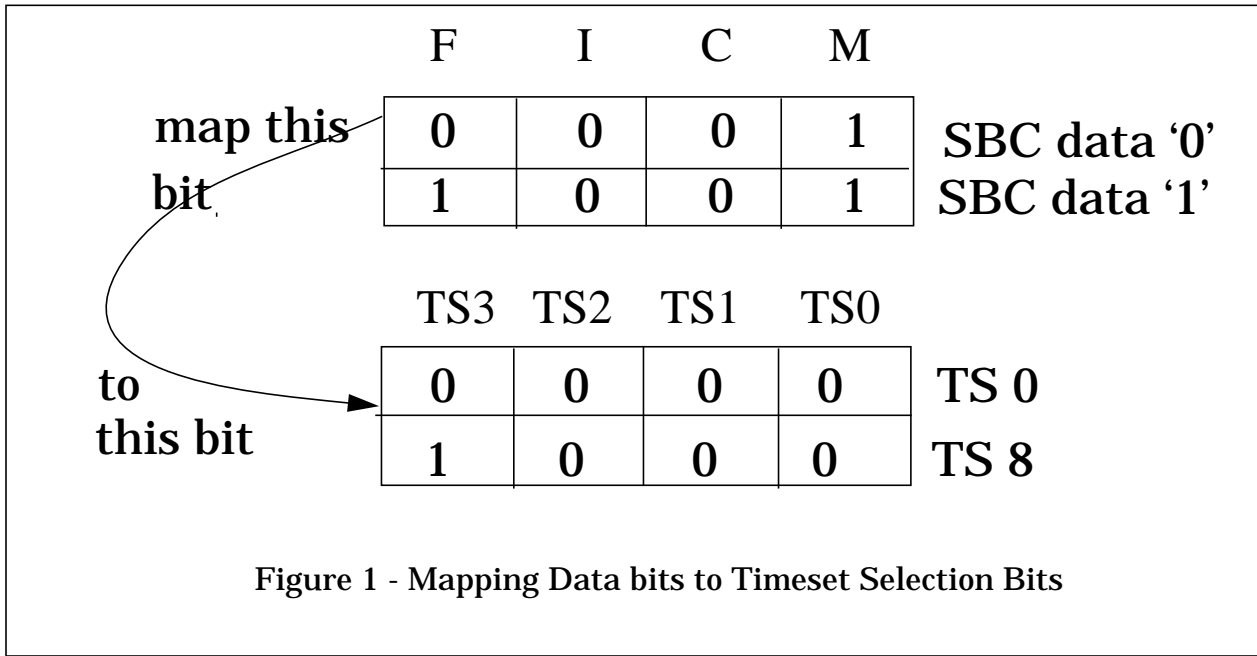
Version 1.0      Initial release 11/9/95
*****/

#include <box.h>
PINLIST *alla;
PINLIST *allio;

#define SBCDRIVE 0x80F
#define SBCMUX 0x0ACF

void initialize_tester()
{
...
}

void test_device()
{
...set_active_loads(allio, PULL_BOTH, 0.003, 0.5, 2.0, 0.007, 4.0, 2.0);
/*
***** Functional Test Showing SBC on A-Inputs Only *****
*/
*/
```



```

/* timeset 0 */
set_cycle_length(0, 100e-9, NULL);
set_force_edges(alla, SBCDRIVE, 0, 80.0*NANO, 20.0*NANO);
set_mux_edges(alla, SBCMUX, 0, 0.0, -1);
/* timeset 8 */
set_cycle_length(8, 100e-9, NULL);
set_force_edges(alla, SBCDRIVE, 8, 20.0*NANO, 80.0*NANO);
set_mux_edges(alla, SBCMUX, 8, -1, 0.0);

```

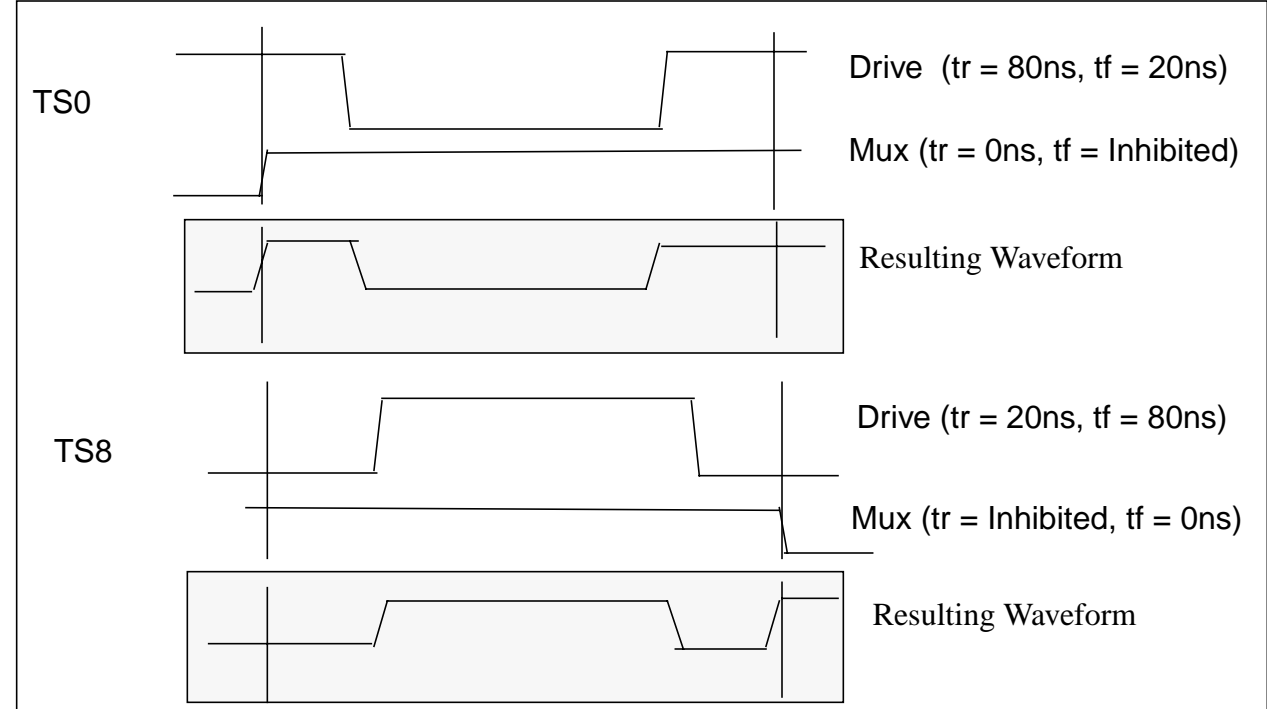


Figure 2 - Timesets 0 and 8 Drive and Mux signals for Input Only Pins

```

set_compare_strobe(allio, TRISTATE, 8, 60*NANO, NOCHANGE);
set_compare_strobe(allio, TRISTATE, 0, 60*NANO, NOCHANGE);

test_name("SBC_Inputs");
if (func_test(allpins,LABEL_START(sbc_on_alla),LABEL_STOP(sbc_on_alla)) == FAIL)
  {set_bin(bin_name("SBC_Fail"));
  goto finish;}

```

For I/O pins, there are additional complications and tradeoffs. First and foremost, the mux edge is no longer available for providing the “third edge”, because we need the inhibit bit to perform its normal I/O function of turning off the driver when the DUT is in an output state.

The third edge can be provided by using one of the force edges that is normally idle during a device output. We can program a rising or falling edge to occur during an output cycle, but since the driver is inhibited, the edge does not make it to the driver output. However, it has been **preconditioned**. As soon as the inhibit signal is removed from the driver, the driver output will swing to the level preconditioned during the DUT output cycle (See Figure 3). This has two significant implications:

- ✱SWAV patterns will become more complex, requiring the use of the Y, C, E and F characters.
- ✱SBC cannot be achieved when transitioning from an input ‘1’ to input ‘0’ and vice versa. The resulting format will be **Return-to-Complement (RC)**.

Why are the characters Y, C, E and F needed? Instead of having only the output states X, L, H and Z, we must define **state transitions** as follows:

X -> 0	X -> 1	L -> 0	L -> 1
H -> 0	H -> 1	Z -> 0	Z -> 1

The requirement for SBC on I/O pins is that we precondition the driver during the device output state to the level that will be driven in the subsequent cycle. Using the FICM bits, this leads to the following relationships:

<u>STATE</u>	<u>FICM</u>	<u>CHAR</u>	<u>STATE</u>	<u>FICM</u>	<u>CHAR</u>
X -> 0	0101	X	X -> 1	1101	Y
L -> 0	0100	L	L -> 1	1100	C
H -> 0	0110	H	H -> 1	1110	E
Z -> 0	0111	Z	Z -> 1	1111	F

This technique provides the third edge when transitioning from DUT output to DUT input cycles. DUT input cycles with same data require no third edge as only two edges per cycle are required, leaving only the 0->1 and 1->0 transitions. On these, SBC format cannot be realized. As the timeset change occurs, we are left with the RC format (See Figure 4).

```

/***** Functional Test Showing SBC on ALLIO Pins *****/
/***** Functional Test Showing SBC on ALLIO Pins *****/
/***** Functional Test Showing SBC on ALLIO Pins *****/
/* timeset 0 */
set_cycle_length(0, 100e-9, NULL);
set_force_edges(allio, SBCDRIVE, 0, 80.0*NANO, 20.0*NANO);
set_inhibit_edges(allio, DNRI, 0, 0.0, 0.0);
set_compare_strobe(allio, TRISTATE, 0, 60*NANO, NOCHANGE);

/* timeset 8 */
set_cycle_length(8, 100e-9, NULL);
set_force_edges(allio, SBCDRIVE, 8, 20.0*NANO, 80.0*NANO);
set_inhibit_edges(allio, DNRI, 8, 0.0, 0.0);
set_compare_strobe(allio, TRISTATE, 8, 60*NANO, NOCHANGE);

test_name("SBC_IO");
if (func_test(allpins, LABEL_START(sbc_on_bidir), LABEL_STOP(sbc_combos)) == FAIL)
    {set_bin(bin_name("SBC_Fail"));
    goto finish;}
...}

void shutdown_device()
{...
...}

```

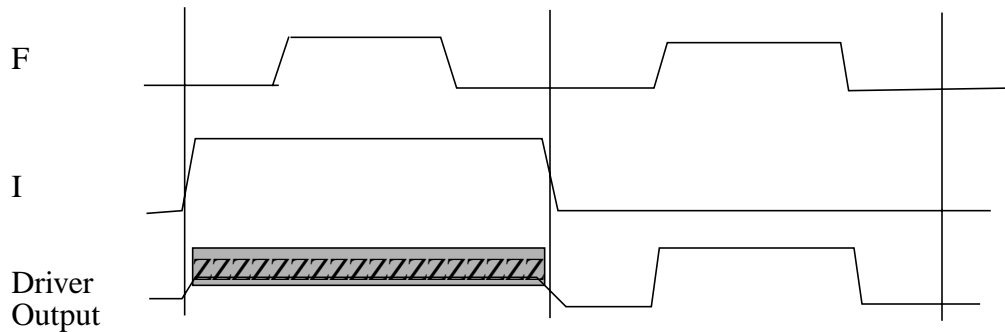


Figure 3 - Preconditioning the driver Output to Create SBC Format when Transitioning from DUT Output to DUT Input

Summary: SBC format can be created through the use of data-driven timeset switching and special characters in the swav file. There are several compromises to be made: reduction in the number of available timesets, more complex swav files, and use of RC format in certain cases. Many other types of complex waveforms may be created through bit level programming of the force, inhibit and mux edges.

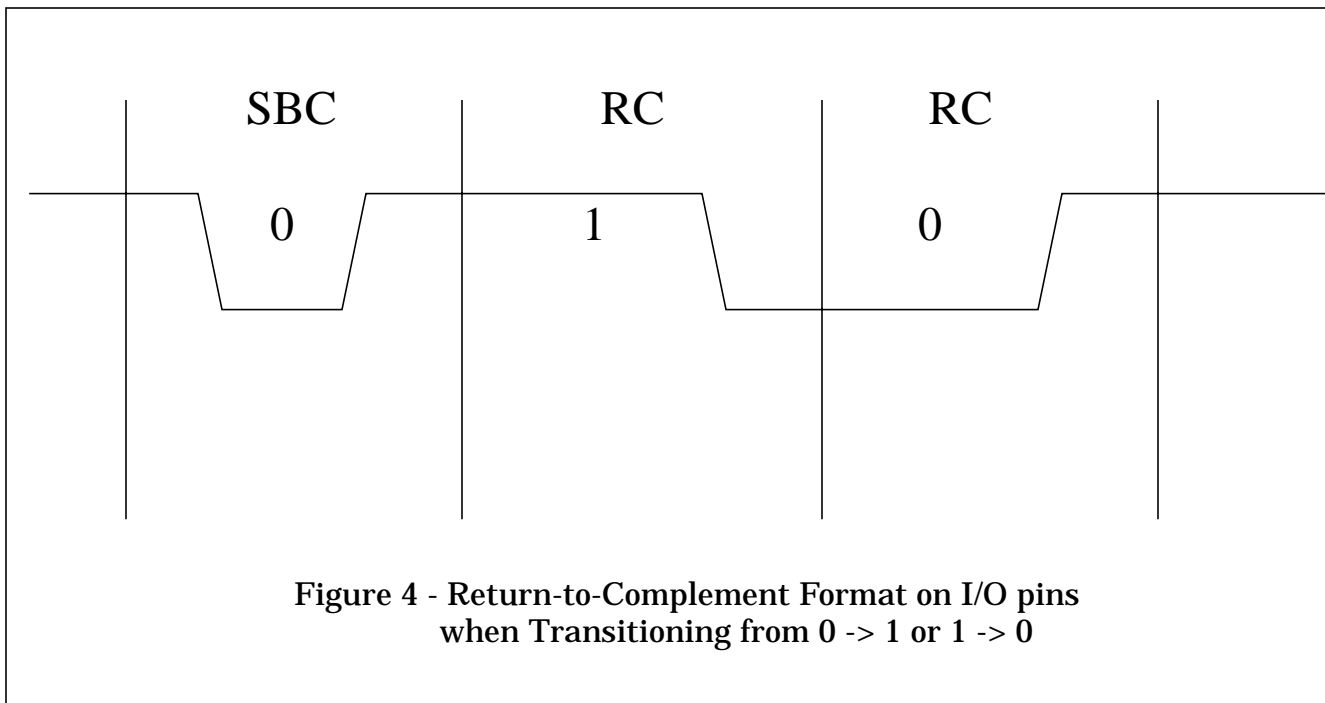


Figure 4 - Return-to-Complement Format on I/O pins when Transitioning from 0 -> 1 or 1 -> 0

APPENDIX I - BIT DEFINITIONS FOR FORCE/MUX REGISTER PROGRAMMING

The following is excerpted from the Credence Manual RCS Version 1.33. It shows how the various driver formats are derived, and can be used to derive valid bit combinations for SBC format or other complex driver waveforms.

FORCE - `set_force_edges(*pinlist, FORMAT, timeset, rise, fall);`

FORMAT defined as:

0x09cdefsrabyyzz

- | | | | |
|---------|---|-------|---|
| c = 0 | Normal operation of timeset select bit 3. | a = 0 | Normal operation of timeset select bit 1. |
| 1 | Force data bit replaces timeset select bit 3. | 1 | Force data bit replaces timeset select bit 1 |
| d = 0 | Normal operation of timeset select bit 2. | b = 0 | Normal operation of timeset select bit 0. |
| 1 | Force data bit replaces timeset select bit 2. | 1 | Force data bit replaces timeset select bit 0. |
| e = 0 | Normal operation of timeset select bit 1. | | |
| 1 | Force data bit replaces timeset select bit 1. | | |
| f = 0 | Normal operation of timeset select bit 0. | | |
| 1 | Force data bit replaces timeset select bit 0. | | |
| sr = 00 | Driver not set to an initial state. | | |
| 01 | Driver set to low state initially. | | |
| 10 | Driver set to high state initially. | | |
| 11 | Not valid. | | |
| yy = 00 | Never generate force start edge. | | |
| 01 | Generate force start edge when F = 0. | | |
| 10 | Generate force start edge when F = 1. | | |
| 11 | Always generate force start edge. | | |
| zz = 00 | Never generate force stop edge. | | |
| 01 | Generate force stop edge when F = 0. | | |
| 10 | Generate force stop edge when F = 1. | | |
| 11 | Always generate force stop edge. | | |

These bits can be overridden by bits e & f as shown.

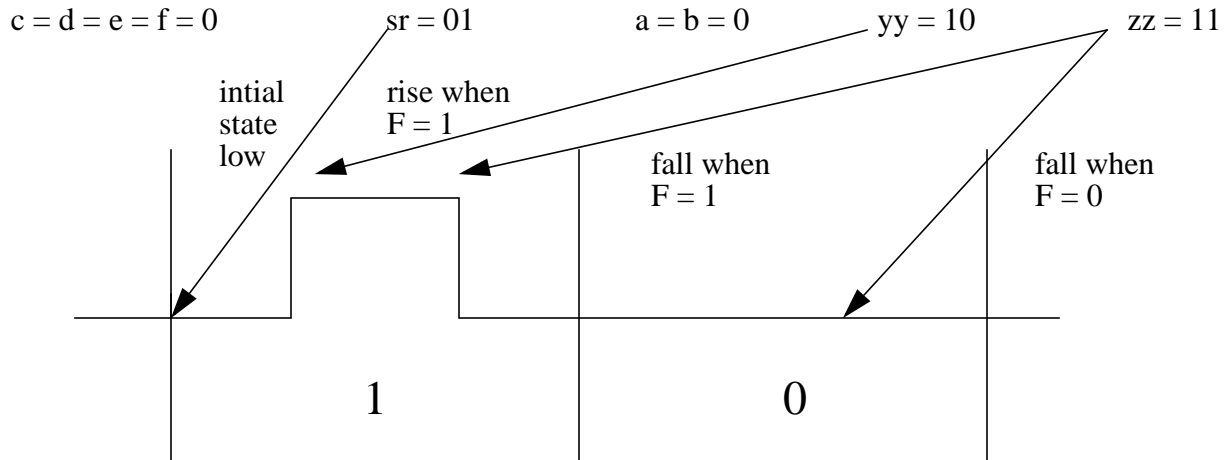
*MUX - set_mux_edges(*pinlist, FORMAT, timeset, rise, fall);*

FORMAT defined as:

0x0Asrabyzz

- sr = 00 Driver inhibit not altered..
- 01 Driver set to 'ON' state initially.
- 10 Driver set to 'OFF' state initially.
- 11 Driver set to 'ON' state initially, Multiplex Mode enabled..
- a = 0 Normal operation of timeset select bit 1.
- 1 Force data bit replaces timeset select bit 1
- b = 0 Normal operation of timeset select bit 0.
- 1 Force data bit replaces timeset select bit 0.
- yy = 00 Never generate inhibit start edge.
- 01 Generate inhibit start edge when F = 0.
- 10 Generate inhibit start edge when F = 1.
- 11 Always generate inhibit start edge.
- zz = 00 Never generate inhibit stop edge.
- 01 Generate inhibit stop edge when F = 0.
- 10 Generate inhibit stop edge when F = 1.
- 11 Always generate inhibit stop edge.

Example for RZ format:



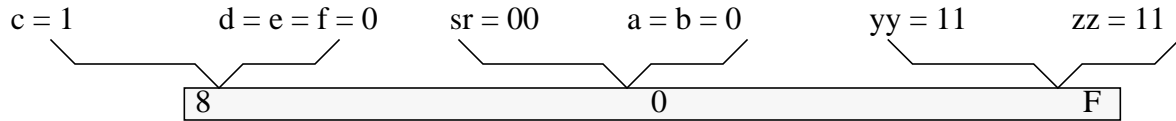
in lt.h, this is set with the statement:

```
#define RZ 0x4B (the '09' is automatically appended through the use of the  
set_force_edges statement, and the 0 for the cdef bits is implied)
```

Example for SBC format:

```
set_force_edges(allio, 0x0980F, timeset, fall, rise); /* for F = 0 */
set_force_edges(allio, 0x0980F, timeset + 8, rise, fall); /* for F = 1 */
set_mux_edges(allio, 0x0ACF, timeset, rise, -1); /* for F = 0 */
set_mux_edges(allio, 0x0ACF, timeset + 8, -1, fall); /* for F = 1 */;
```

FORCE:



use F-bit to replace
timeset select bit 3

always generate both
rising and falling edges

MUX:



Note: Programming $yyzz = 1100$ or $yyzz = 0011$ has the same effect as programming the `set_mux_edges` to -1 for fall and rise edges times, respectively.

APPENDIX II - swav2sbc, the SWAV to SBC Format Vector Converter

```

/*****
/* swav2sbc.c
/* The SWAV to Surround-by-Complement Vector Converter
/* Copyright 1995 Insyte, an ATE Services Company
/* Author: Peter Lindholm
/* Revision History
/* 12 Nov 1995 Initial Release
*/
#include <stdio.h>
#include <string.h>
#define MAXLINE 300

/* function declarations */
void parse_bidir_string();
long parse_test_vector();
/* global variable declarations */
int bidir_pins[512]; /* an array where we can track bidir pins */
int bidir_pins_index = 0; /* for indexing into that array */
FILE *swav, *sbcfile; /* file pointers for input/output */
main(argc, argv)
int argc;
char *argv[];
{
char ln[MAXLINE+1]; /* the line being read in from the file */
char *didit = 0; /* test for success */
int pincount = 0; /* */
int array_size = 0; /* */
int number; /* */
system("clear"); /* a clear screen is a tidy screen */
printf("\nSWAV2SBC, the Surround By Complement Vector Generator\n");
printf("Copyright 1995 Beacon Digital, an ATE Services Company\n\n");
printf("Converting Test Vectors...");
/* initialize the pattern column array */
for (bidir_pins_index = 0; bidir_pins_index < 512; bidir_pins_index++)
bidir_pins[bidir_pins_index] = 0;
bidir_pins_index = 0;
/* arg checking an usage errors */
if (argc < 3)
{
printf("\nError 100: swav2sbc requires two arguments.\n");
printf(" Usage: swav2sbc <input_filename> <output_filename>\n");
exit(1);
}
/* open all files */
if ((swav = fopen(argv[1], "r")) == NULL)
{
printf("\nError 101: Unable to open input file.\n Make sure file exists.\n");
exit(1);
}
if ((sbcfile = fopen(argv[2], "w")) == NULL)
{printf("\nError 102: Unable to open output file.\n Make sure file name is valid\n");
exit(1);
}
/* read the signal definition fields */
while (fgets(ln, MAXLINE, swav) != NULL)
{
didit = strstr(ln, "signal");
if (didit != NULL)
{
pincount++;
parse_bidir_string(ln, pincount);
}
}
}

```

```

/* verify array and figure out how many I/O pins there are */
for (bidir_pins_index = 0; bidir_pins_index < pincount; bidir_pins_index++)
{
    if (bidir_pins[bidir_pins_index] == 0)
    {
        array_size = bidir_pins_index;
        break;
    }
}
/* Now start reading vectors. Find out if the line is a vector. If it
is, replace make a note of the I/O states. Read the next vector to see if
it needs changing */
rewind(swav);
while (fgets(ln, MAXLINE, swav) != NULL)
{
    didit = strstr(ln, "pattern");
    if (didit != NULL)
    {
        fputs(ln, sbcfile);
        number = parse_test_vector(ln, array_size);
    }
    else fputs(ln, sbcfile);
}
fclose(swav);
fclose(sbcfile);
printf("\n..Done\n");
printf("%d Test Vectors written to file %s\n", number, argv[2]);
}
void parse_bidir_string(bidir_string, pattern_column)
char bidir_string[MAXLINE+1];
int pattern_column;
{
    char *token;
    int match;
    strtok(bidir_string, " ;{}");
    while ((token = (strtok(NULL, " ;{}")))!=NULL)
    {
        if ((match = strcmp(token, "bidir1")) == 0)
        {
            bidir_pins[bidir_pins_index] = pattern_column;
            bidir_pins_index++;
        }
    }
}
long parse_test_vector(vector_string, size)
char vector_string[MAXLINE + 1];
int size;
{
    int i;
    int index = 0;
    char compressed_vector[513];
    int maxpins;
    int j;
    char old_vector[MAXLINE+1];
    long first_vector = 0;
    char seq_string[50];
    int seq_flag = 0;
    int k;
    /* initialize array */
    for (j=0;j<=MAXLINE;j++)
        old_vector[j] = '0';

while (fgets(vector_string, MAXLINE, swav) != NULL)
    /* if it is a test vector and not a comment, marker, or pattern statement */
    if ((strstr(vector_string, "pattern") == NULL) && (strstr(vector_string, "marker") ==

```

```

NULL)&&(strstr(vector_string, "/*") == NULL) && (strstr(vector_string, "*/") == NULL))
    { /* until the semicolon ('59' ascii) */
    first_vector ++;
    while(vector_string[i] != 59)
        {
        if (vector_string[i] == 62)
            {
            seq_flag = 1;
            }
        if (vector_string[i] != ' ')
            {
            if (seq_flag == 0)
                {
                compressed_vector[index] = vector_string[i];
                index++;
                }
            }
        if (seq_flag == 1)
            {seq_string[k] = vector_string[i]; k++;}
        i++;
        }
    /* the width of the pattern */
    maxpins = index;
    /* set indeces back to 0 */
    i = 0;
    index = 0;
    seq_flag = 0;
    /* Now compare old_vector and compressed_vector */
    for (j = 0; j<=maxpins;j++)
        {
        if (compressed_vector[j] == '1')
            {
            switch(old_vector[j])
                {
                case ('H'):{
                old_vector[j] = 'E';break;}
                case ('L'):{
                old_vector[j] = 'C';break;}
                case ('Z'):{
                old_vector[j] = 'F';break;}
                case ('X'):{
                old_vector[j] = 'Y';break;}
                }
            }
        }
    if (first_vector > 1)
        {
        for (j=0;j<maxpins;j++)
            fputc(old_vector[j], sbcfile);
        for (j = 0;j<k;j++)
            fputc(seq_string[j], sbcfile);
        fputc(';', sbcfile);
        fputc(0x0a, sbcfile);
        }
    for (j=0;j<=maxpins;j++)
        old_vector[j] = compressed_vector[j];
    for (j=0;j<k;j++)
        seq_string[k]= 0;
    k = 0;
    }
    else fputs(vector_string, sbcfile);
    }
return first_vector;
}

```