

Creating DTS Models for onTAP

The DTS test program format as used in onTAP for cluster tests is a subset of GenRad's in-circuit test DTS format. A difference is that onTAP allows the comments to be delimited with double slash characters. The test programs consist of a header section where all pins, pin names, and pin I/O are declared and a main section consisting of test instructions. Diagnostics are managed by onTAP.

This example is based on a FIFO device, the 72V3690, which is included in the onTAP/cluster_test folder. An additional example for the IDT71416 RAM is included in the cluster_test folder.

Highlights: DTS Models

- Flexible & reusable
- Non-project specific
- Change on-the-fly
- Familiar C-like language
- Simple edits enable use for a variety of device types

A Reference DTS Test Program

Statements in the following sample DTS will be used to explain DTS test programs.

```

////////////////////////////////////
/*
    Device Type: 72v3690
*/

.HEAD;

.SIZE 128;

.INPUT(112=BM,114=BE,51=D0,50=D1,49=D2,47=D3,46=D4,45=D5,43=D6,42=D7,41=D8,40=D9,
      39=D10,38=D11,36=D12,34=D13,33=D14,32=D15,31=D16,30=D17,28=D18,27=D19,26=D20,
      25=D21,23=D22,20=D24,19=D25,18=D26,17=D27,16=D28,15=D29,13=D30,12=D31,10=D32,
      9=D33,8=D34,7=D35,118=FS0,115=FS1,124=FWFT,113=IP,125=LD,102=OE,119=OW,126=MR
S,
      109=PFM,127=PRS,105=RCLK,104=REN,107=RM,103=RT,2=SEN,128=WCLK,1=WEN);

.OUTPUT_TRI(53=Q0,54=Q1,55=Q2,56=Q3,57=Q4,58=Q5,60=Q6,62=Q7,63=Q8,64=Q9,65=Q10,66=
Q11,
      68=Q12,69=Q13,70=Q14,71=Q15,74=Q16,75=Q17,77=Q18,78=Q19,79=Q20,80=Q21,82=Q23,
      85=Q24,86=Q25,88=Q26,89=Q27,90=Q28,91=Q29,92=Q30,93=Q31,96=Q32,97=Q33,98=Q34,
      99=Q35);

.OUTPUT(108=EF,123=FF,117=HF,110=PAE,121=PAF);

.POWER(4=VCC,11=VCC,24=VCC,35=VCC,48=VCC,61=VCC,72=VCC,73=VCC,
      87=VCC,100=VCC,101=VCC,111=VCC,122=VCC);

.GROUND(14=GND,22=GND,29=GND,37=GND,44=GND,52=GND,59=GND,67=GND,
      76=GND,83=GND,84=GND,94=GND,95=GND,106=GND,116=GND,120=GND);
.END HEAD;

```

```

.DECLARE GROUP DATAH(D35,D34,D33,D32,D31,D30,D29,D28,D27,D26,D25,D24,D23,D22,D21,D20,D19);
.DECLARE GROUP DATAL(D18,D17,D16,D15,D14,D13,D12,D11,D10,D9,D8,D7,D6,D5,D4,D3,D2,D1,D0);
.DECLARE GROUP OUTH(Q35,Q34,Q33,Q32,Q31,Q30,Q29,Q28,Q27,Q26,Q25,Q24,Q23,Q22,Q21,Q20,Q19);
.DECLARE GROUP OUTL(Q18,Q17,Q16,Q15,Q14,Q13,Q12,Q11,Q10,Q9,Q8,Q7,Q6,Q5,Q4,Q3,Q2,Q1,Q0);

.MAIN;

IH(MRS,PRS); // set inputs MRS and PRS high
IL(MRS); // input low.. Fifo Reset - all registers cleared, output flags reset */
IH(MRS);
OS(EF,FF) OH(FF) OL(EF); /* empty flag low indicating no data stored */
OI(FF,EF);
/* begin write operation */
IH(WEN,WCLK,REN,RCLK) IG(DATAH=X'5A5A5',DATAL=X'5A5A5');
/* WRITE FIRST DATA INTO MEMORY */
IL(WEN);
IL(WCLK);
IH(WCLK);
OS(EF) OH(EF); // enable output sense and test pin EF for a logic high
OI(EF);
IL(WCLK)IG(DATAH=X'5A5A5',DATAL=X'5A5A5'); /* first word write staggers data */
IH(WCLK);
IL(WCLK)IG(DATAH=X'FFFFF',DATAL=X'FFFFF'); /* 2nd word loads all 1's */
IH(WCLK);
IL(WCLK)IG(DATAH=X'A5A5A',DATAL=X'A5A5A'); /* 3rd word write staggers data */
IH(WCLK);
IL(WCLK)IG(DATAH=X'00000',DATAL=X'00000'); /* 4th word loads all 0's */
IH(WCLK);
IL(WCLK)IG(DATAH=X'5A5A5',DATAL=X'5A5A5'); /* 5th word write staggers data */
IH(WCLK);
IL(WCLK)IG(DATAH=X'FFFFF',DATAL=X'FFFFF'); /* 6th word loads all 1's */
IH(WCLK);
IL(WCLK)IG(DATAH=X'A5A5A',DATAL=X'A5A5A'); /* 7th word write staggers data */
IH(WCLK);
IL(WCLK)IG(DATAH=X'00000',DATAL=X'00000'); /* 8th word loads all 0's */
IH(WCLK);
IL(WCLK)IG(DATAH=X'5A5A5',DATAL=X'5A5A5'); /* 9th word write staggers data */
IH(WCLK);
IL(WCLK)IG(DATAH=X'FFFFF',DATAL=X'FFFFF'); /* 10th word loads all 1's */
IH(WCLK);
IL(WCLK)IG(DATAH=X'A5A5A',DATAL=X'A5A5A'); /* 11th word write staggers data */
IH(WCLK);
IL(WCLK)IG(DATAH=X'00000',DATAL=X'00000'); /* 12th word loads all 0's */
IH(WCLK);
IH(WEN);
/* begin read operation */
IL(REN);
IL(RCLK)OG(OUTH=X'5A5A5',OUTL=X'5A5A5'); /* first word write staggers OUT */
IH(RCLK);
////////////////////

```

```

//// Test Truncated Here
////////////////////////////////////
.END MAIN;
////////////////////////////////////

```

Let's review the statements in the header and main sections, using the above reference code as an example.
The Header Section (.HEAD)

The header section begins with a .HEAD declaration followed on the next line by a statement of the device's pin count, .SIZE 128. Next, the pins are listed by I/O type, which are inputs shown as .INPUT(pins), outputs shown as .OUTPUT(pins), output tri states shown as .OUTPUT_TRI(pins), bidirectionals shown as .BIDIR_TRI(pins), power pins shown as .POWER(pins), and grounds shown as .GROUND(pins).

Each "pins" word in parentheses consists of a comma delimited list of expressions equating a package pin name to a pins functional name.

Unused pins do not need to be present in the header section. It is important that semicolons terminate each statement. The .END HEAD; expression terminates the header section.

The .MAIN Section.

The main section begins with a .MAIN; statement and ends with a .END MAIN; statement. Instructions are available to drive inputs high and low, sense outputs high and low, connect and disconnect drivers and to sense or ignore outputs. Constructions such as .REPEAT are available to handle program loops.

The basic test statements are shown immediately below. The pinlist expression is a comma delimited list of pin function names from the header section, e.g., (EF,FF,HF,PAE,PAF).

- IH(pinlist);

IH means input high and sets each pin in the pinlist to a logic high value. A pin will remain high until an IL or ID statement changes its value.

- IL(pinlist);

IL means input low and sets each pin in the pinlist to a logic low. A pin will remain low until an IH or ID statement changes its value.

- OS(pinlist);

OS means output sense and enables sensing for a pin. If an OH or OL statement has not been issued for the pin, OL is assumed.

- OH(pinlist);

OH means sense or test a pin for a logic high. The OH condition will remain effective until changed by an OL or OI instruction.

- OL(pinlist); The OL condition will remain effective until changed by an OH or OI instruction.

- OI(pinlist);

OI means output ignore and will stop the testing of a pin. If the # character is substituted for pinlist, then all pins will be ignored.

- Declare Group group_name(pinlist);

Represents a group of pins with a name, such as:

```
.DECLARE GROUP ADDR(A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11);
```

Pin groups may be used in input and output group expressions, IG() and OG(), and may also be manipulated as bit strings.

- `IG(group_control_expression);`
IG means input group and `group_control_expression` is a comma delimited list of expressions which equate pin groups to hexadecimal values. Each pin will be driven as an input to the indicated value.
- `OG(group_control_expression);`
OG means output group and `group_control_expression` is a comma delimited list of expressions which equate pin groups to hexadecimal values. Each pin will be sensed as an output at the indicated value.
- `.DECLARE GROUP();`
Represents a group of pins with a name, such as:
`.DECLARE GROUP ADDR(A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11);`

Pin groups may be used in input and output group expressions, `IG()` and `OG()`, and may also be manipulated as bit strings. For example:

```
IG(ADDR=B'000000000000000000',DATA=X'00000000');
IG(ADDR);
```

- `.DECLARE BSTRING(SIZE);`
Declares a variable with a bit string format. An example is:
`.DECLARE BSTRING NEXT(20);`
Simple arithmetic operations may be performed on bit strings and their values may be assigned to pin groups as in
`LET NEXT=D'1'; // SET VARIABLE NEXT TO DECIMAL 1`
`IG(ADDR=NEXT);`
`LET NEXT = NEXT * 2;`

- `LET`
Keyword flags the beginning of assignment and arithmetic statements such as
`LET NEXT = NEXT * 2;`

- `.REPEAT SIZE;`
Keyword identifies the beginning of a loop such as

```
.REPEAT D'19'; // 19 ADDRESS BITS - BEGIN LOOP
IG(ADDR=NEXT,DATA=X'AAAAAAAA');
LET NEXT = NEXT + 1;
.END REPEAT;
```

- `.END REPEAT;`
Keyword flags the end of a repeat loop.

Note that the above pin-related instructions are “sticky” so that once a pin value is set it does not have to be repeated and remains effective until changed in a later statement. The basic statements may be grouped for one test vector as shown in the following examples:

```
IL(pinlist_A) IH(pinlist_A) OS(pinlist_B) OH(pinlist_G) OL(pinlist_C);
IH(pinlist_H);
IH(pinlist_T);
```

Again, each instruction must be terminated by a semicolon.