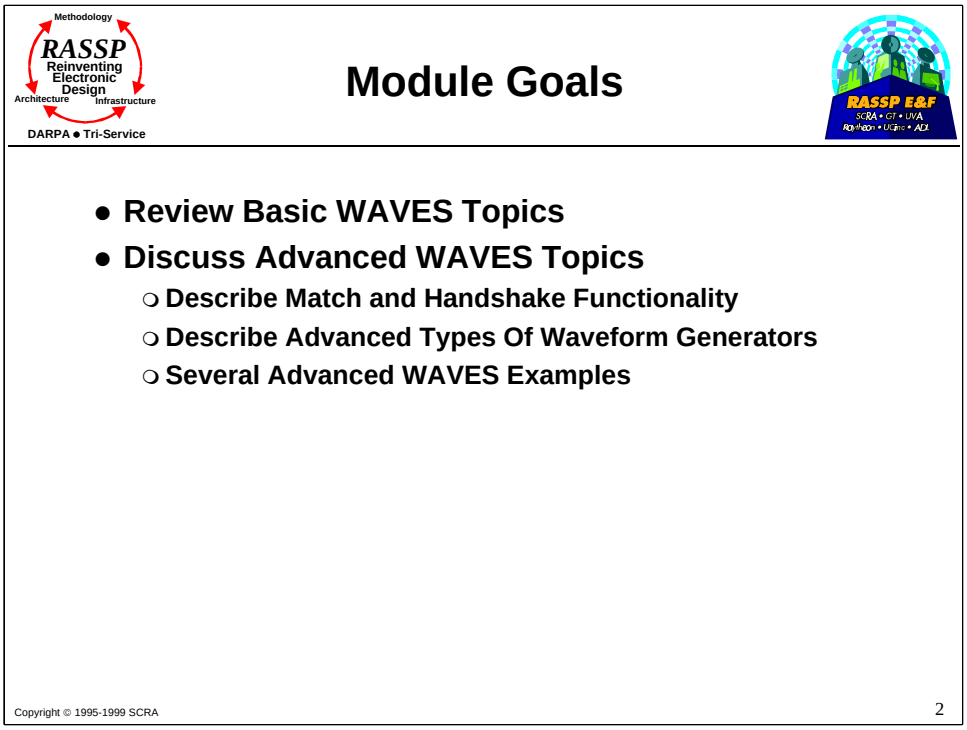
Advanced WAVES Topics

RASSP Education & Facilitation

Module 62

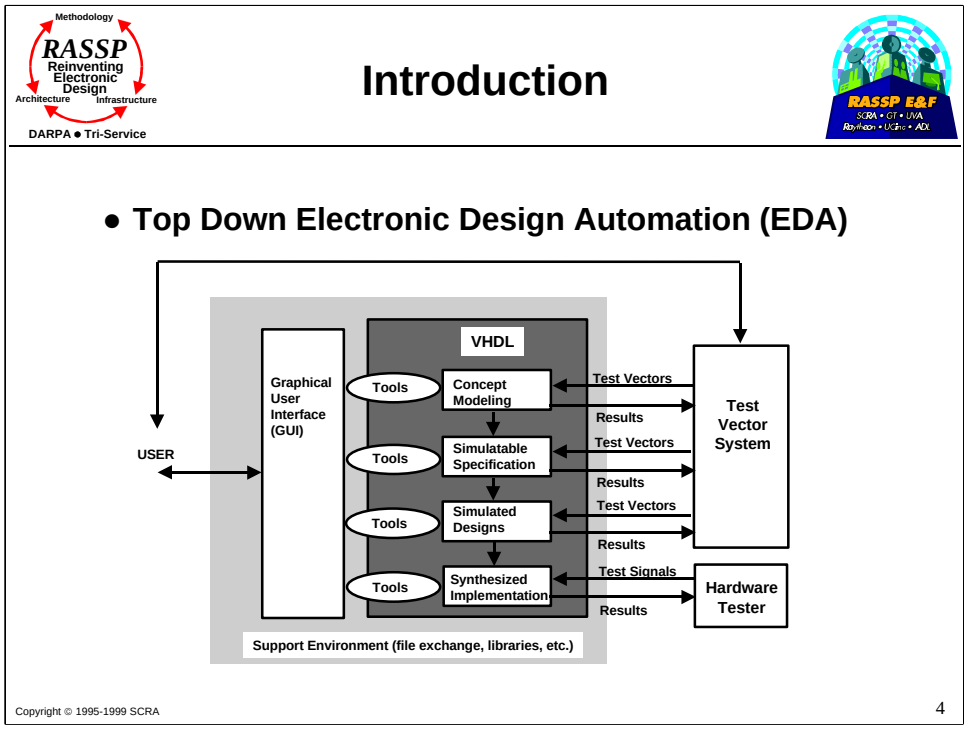
Version 3.00



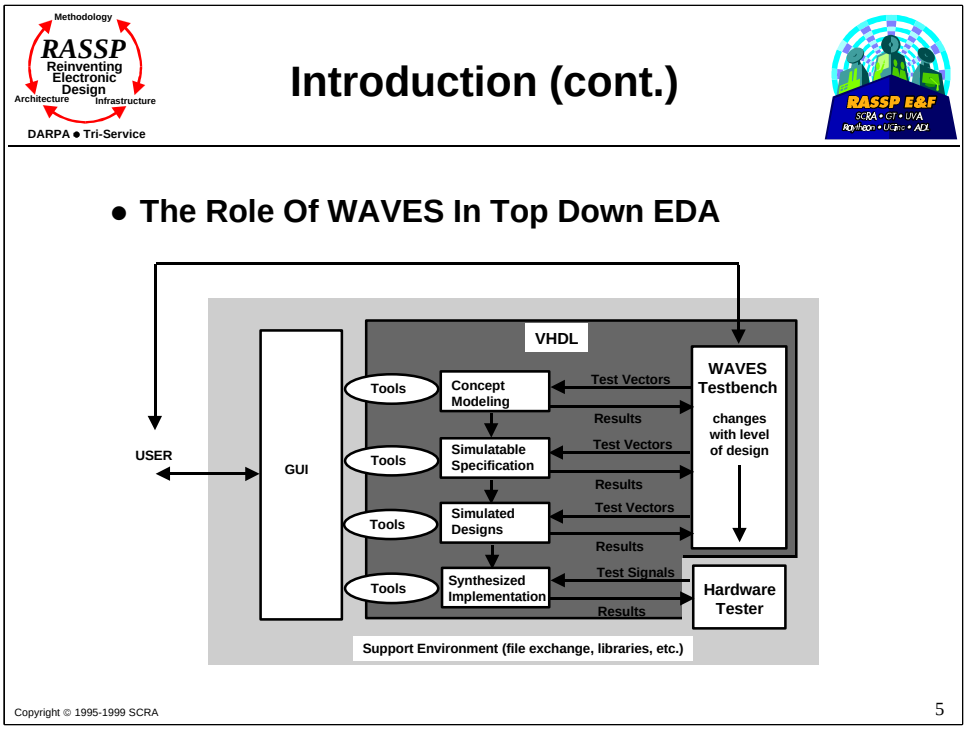
The goal of this module is to demonstrate some of the advanced features of WAVES. Many of the topics involve modifying the waveform generator to perform additional operations within the test set. After completing this module, the student should be able to create a WAVES test set using one or more of the advanced features described in this module. It is assumed that the student is familiar with the details of the VHDL language. Since WAVES is a subset of VHDL, the student should be able to follow the syntax and examples presented in this module.

Module Outline

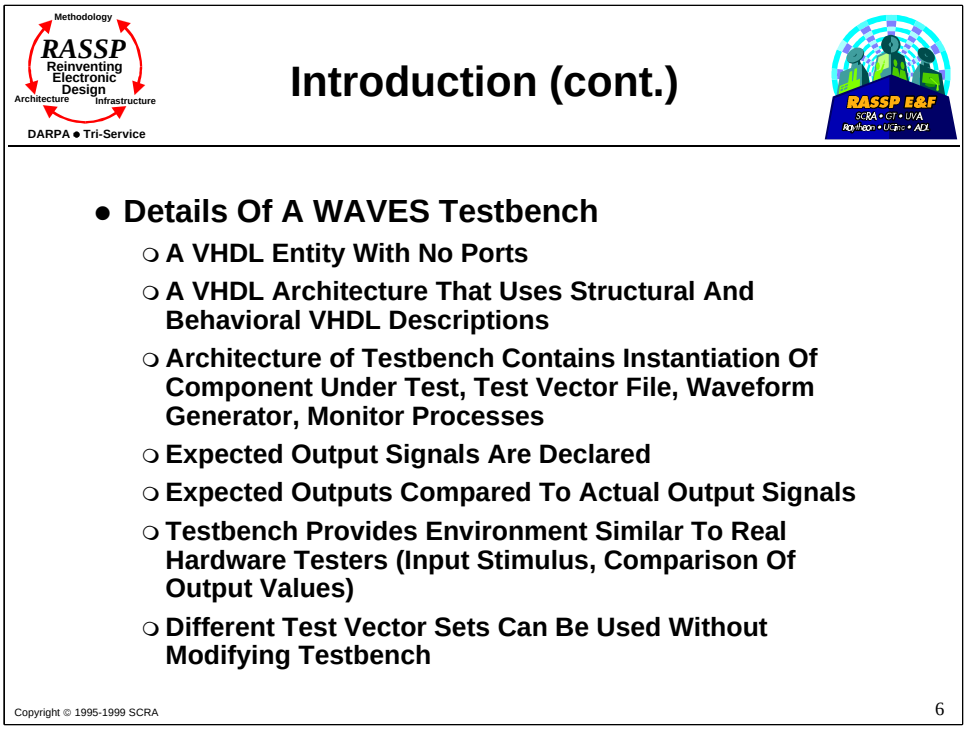
- **Introduction**
- **Match Functionality**
- **Handshake Functionality**
- **Multiple Waveform Generators**
- **Multiple Timing Sets And Bidirectional Test Pins**



Most EDA tool developers are using VHDL as the underlying engine beneath their tool suite. Using VHDL, a design can be simulated at any level, from concept to implementation. However, the unification of the EDA tools around VHDL has created a requirement. The designer needs to be able to stimulate the simulations at the various stages of development and to collect the results of these simulations. In other words, the designer needs test vector generation and results collection and comparison for the simulated development descriptions at all stages. WAVES was created to meet this need. [Hanna97]



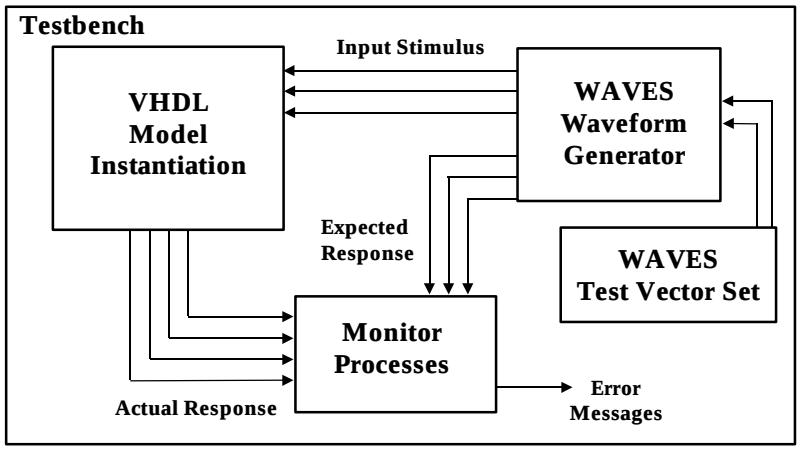
WAVES was designed to be the unified testing and results collection system to complement the unified development systems based on VHDL. Its purpose is to provide the means to define test stimuli, in the form of digital waveforms (or test vectors), to define the results to be collected, and to manage the insertion of the stimuli and the collection and comparison of the results as the VHDL description is simulated. It is also designed for compatibility with hardware testers, such that the same test stimuli and collection paradigm may be automatically communicated to hardware test systems. This ensures identical testing of the hardware and pre-implementation simulation. The testing and collection entity of WAVES, called the WAVES testbench, is written in VHDL and attached to the VHDL description. The testbench is analyzed, compiled and executed along with the rest of the VHDL under simulation. [Hanna97]



The WAVES testbench is described using the standard Entity and Architecture format for a VHDL description. The testbench entity has no ports, since all testing operations occur within the testbench. The component under test is instantiated using structural VHDL. The test vector file is referenced using a VHDL File statement. The waveform generator is referenced as a VHDL Procedure within the testbench architecture. The comparison signals are declared within the architecture of the testbench. These signals are used within the monitor processes contained in the testbench architecture. The WAVES testbench is similar to real hardware testers for the reasons shown. One advantage to the WAVES testbench is that different test vector sets can be used without any modification to the testbench.

Introduction (cont.)

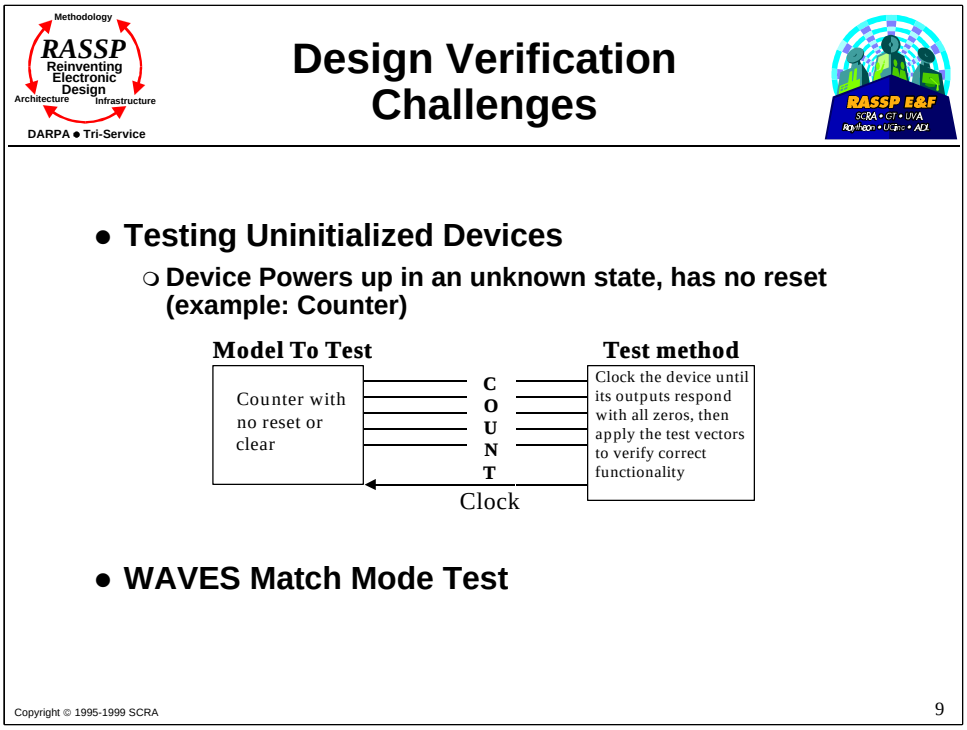
- **WAVES Testbench Diagram**



This diagram shows a complete WAVES test set. The waveform generator reads test vectors from a file and generates inputs for the component and the expected output values from that component. The component under test receives the input stimulus from the waveform generator and produces output values based on its VHDL description. These output values are then compared to the expected responses by monitor processes within the testbench. If the two responses do not match, then an error message is produced. The testbench creates a complete testing environment around the VHDL model. In some of the advanced applications described in this module, the waveform generator may interact with the testbench in order to control the application of each test vector.

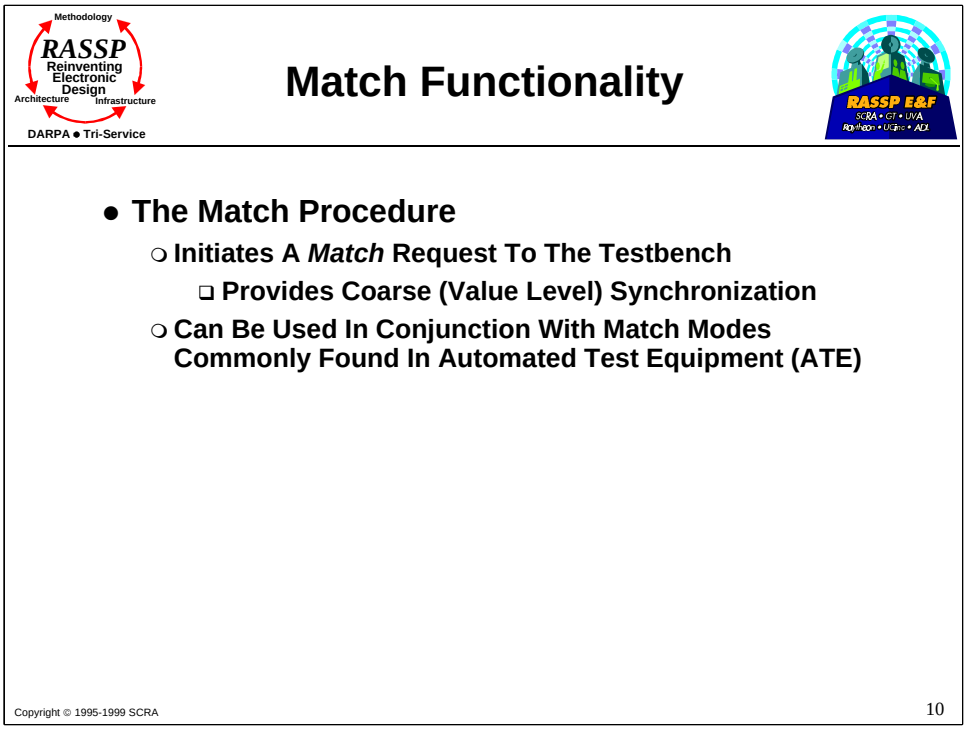
Module Outline

- Introduction
- **Match Functionality**
- Handshake Functionality
- Multiple Waveform Generators
- Multiple Timing Sets And Bidirectional Test Pins



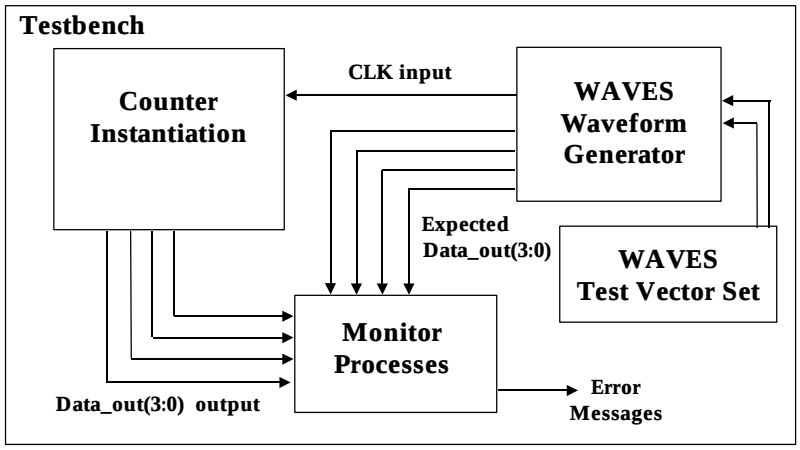
The WAVES Match function allows the designer to perform testing of uninitialized devices. If a hardware component powers up into an unknown state and has no reset capability, it is difficult to find a valid starting point for testing. It may be necessary to cycle the device through an unknown number of clock cycles to reach a known state. The Match functionality allows the designer to match a known value with the output of the model. This allows the designer to start testing after the behavior of the component is well established. The Match function allows the waveform to have an arbitrary number of clock pulses until the UUT reaches the desired state.

In the example shown above, a counter powers up into an unknown state and has no reset capability. Therefore, the testing approach uses the Match functionality to establish a valid starting point for applying test vectors. In this case, the counter is clocked until it responds will all zeros on its output. The Match function compares the counter output with the desired starting value of all zeros during every slice. If they match, then the test vectors will be applied to the counter. If they do not match, then the counter is clocked, and the output is compared again during the next slice. [Hanna97]

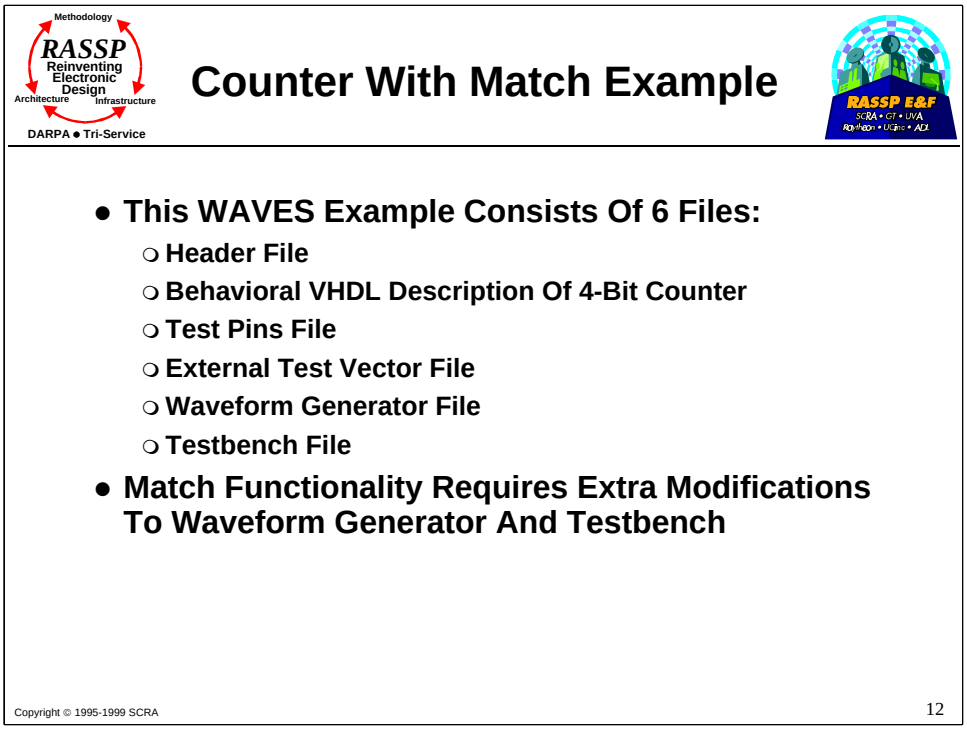


The Match procedure is used to initiate a matching operation in the WAVES test set. The Match procedure specifies which test pins are compared to a known initial value. The Match operation is terminated when the value on the test pins matches the initial value. The Match functionality is included in WAVES in order to facilitate interactions with Automatic Test Equipment (ATE). ATE commonly includes some kind of matching mode which could interface with a WAVES testbench. [Hanna97]

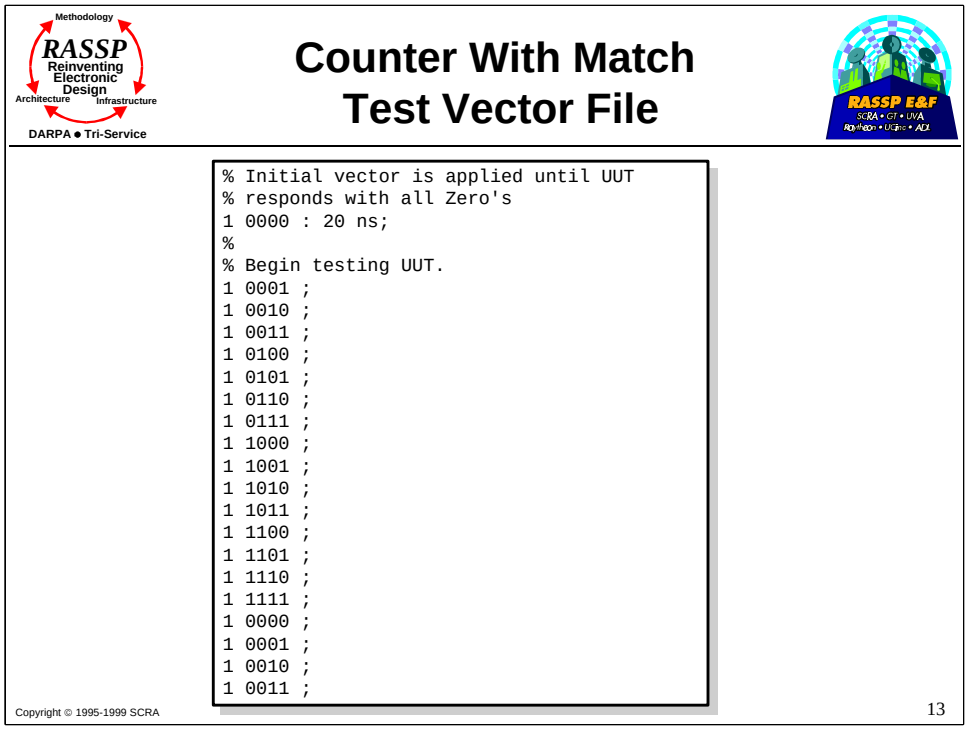
Counter With Match Example



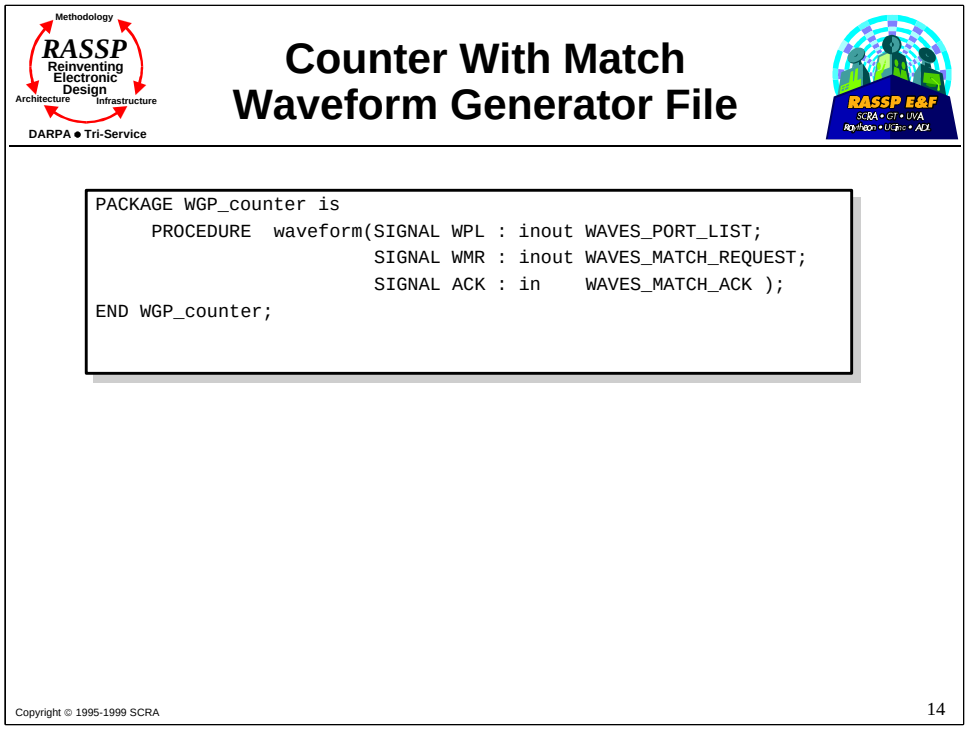
The first example in this module demonstrates the Match functionality with a 4-bit counter as the UUT.



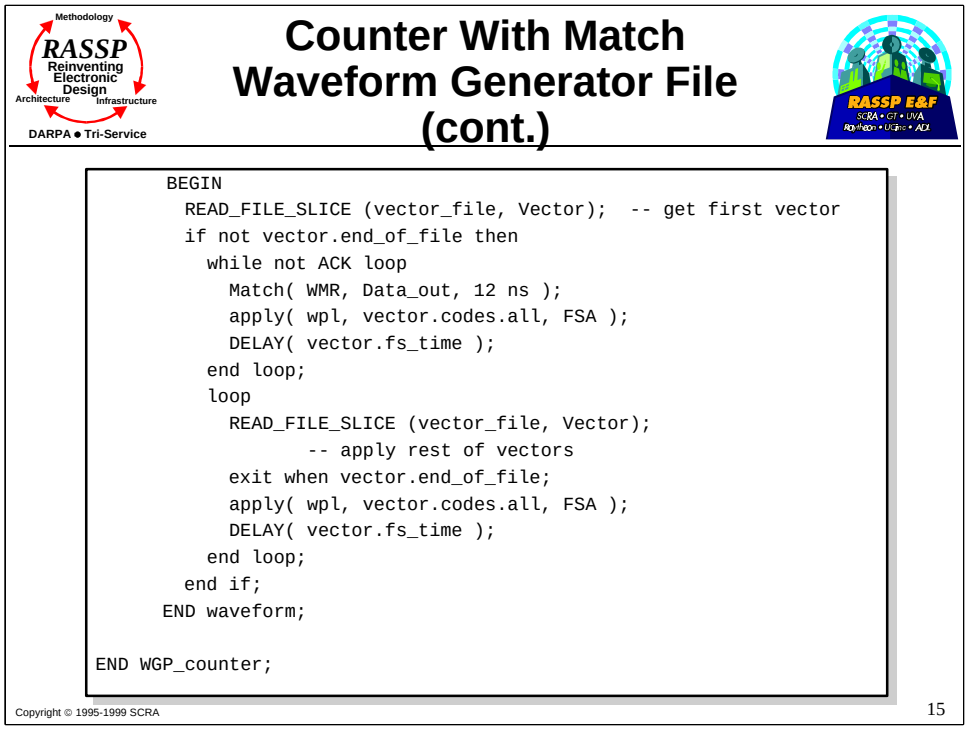
The Match functionality requires the designer to make extra modifications to the waveform generator and testbench. Since these changes are unique to the Match function, it is important to note the modifications made to the test set seen in this example. The required changes in the WAVES test set is summarized over the next several slides.



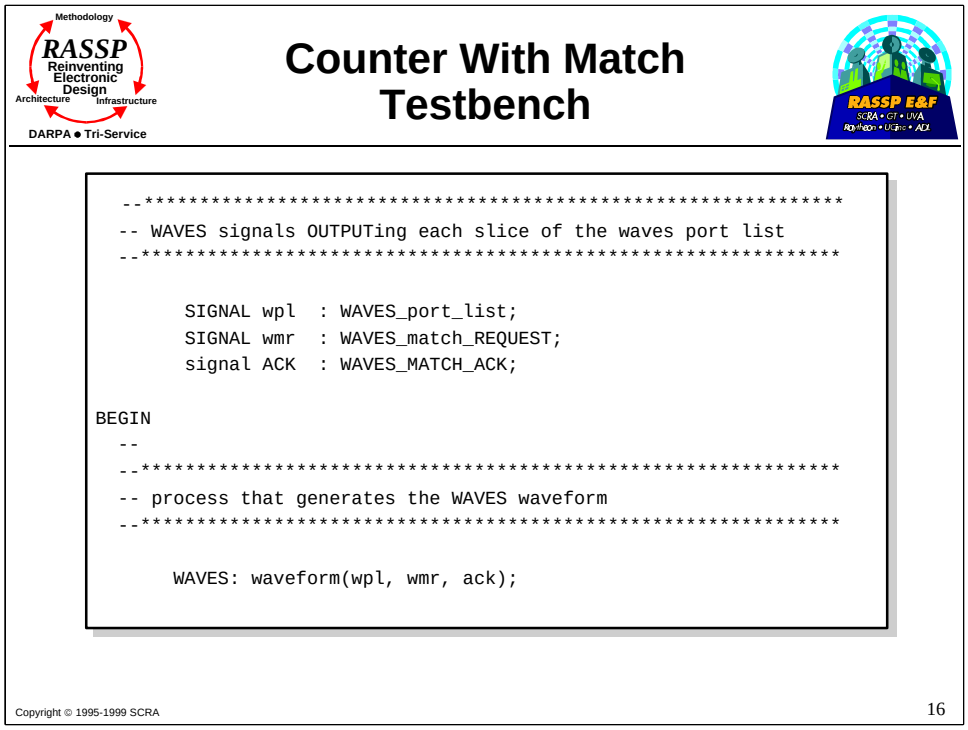
This is the external test vector file for the 4-bit counter. It cannot be created using the WAVES tool set. The first pin code is applied to the clock input pin (called CLK), while the other four pin codes are for the 4-bit output (called Data_out). The first test vector shows an output value of “0000”. Using the Match functionality, this initial test vector will be applied repeatedly until the UUT responds with “0000” as its output value. Once, the initial output values are matched, then each subsequent test vector in this file is applied to the UUT. The slice duration is 20ns for every vector in this file.



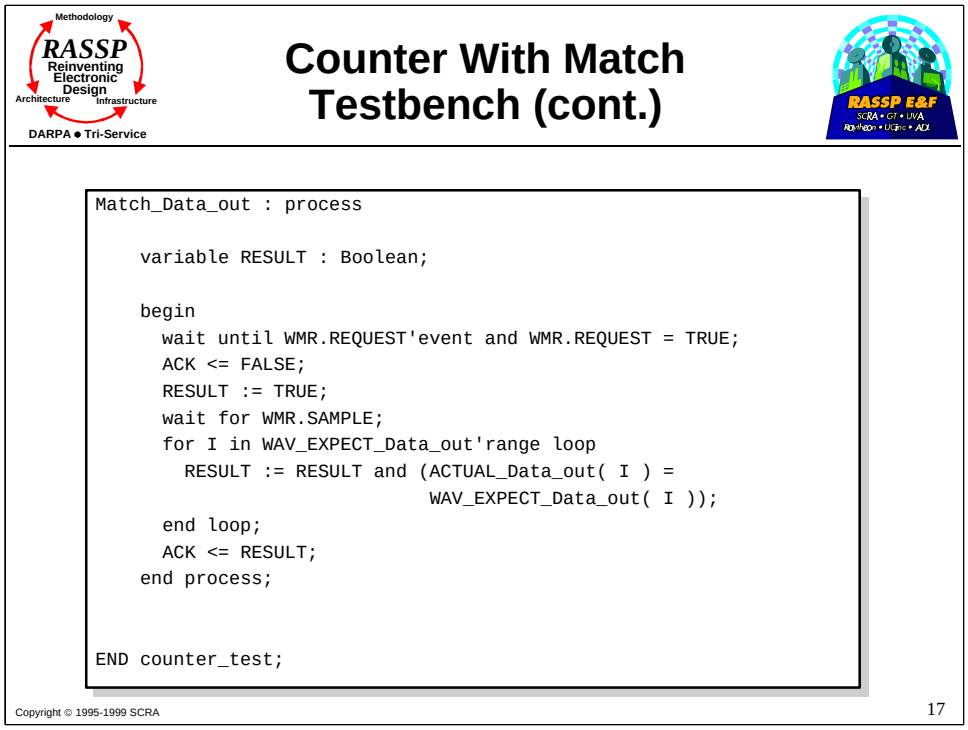
The waveform generator can be generated using the WAVES tool set. However, modifications to the waveform generator are necessary. In this example, two new signals are declared in the procedure Waveform: WMR and ACK. These signals are used to coordinate the Matching operation on the output of the counter. The signal WMR is used to initiate a Match request to the testing environment. The signal WMR is actually a record which stores the test pins involved in the Match, timing information, and a Boolean flag. The signal ACK is a signal set by the test environment when the previously issued Match request has been satisfied. [STD97]



Since this example uses the Match functionality, two LOOP statements must be included in the waveform generator. In the first loop, the initial test vector is read from the external file. Then, the Match function is called using the request signal WMR. The pinset Data_out is specified as the signals which are involved in the Match operation. The timing information in Match is used to specify at what time during the current slice the signals are to be compared. In this case, the Data_out signals from the counter are to be sampled 12ns from the start of each slice. Next, the Apply procedure generates events on the test pins using the pin code and timing information. If the UUT output matches the initial vector, then the ACK signal is set to '1' and the second loop is entered. Otherwise, ACK stays at '0' and the simulation remains in the first loop. In the second loop shown above, the test vectors from the external file are applied in the standard fashion. In summary, the use of the Match operation requires two loops in the waveform generator to control the various operations with the test vectors. [STD97]



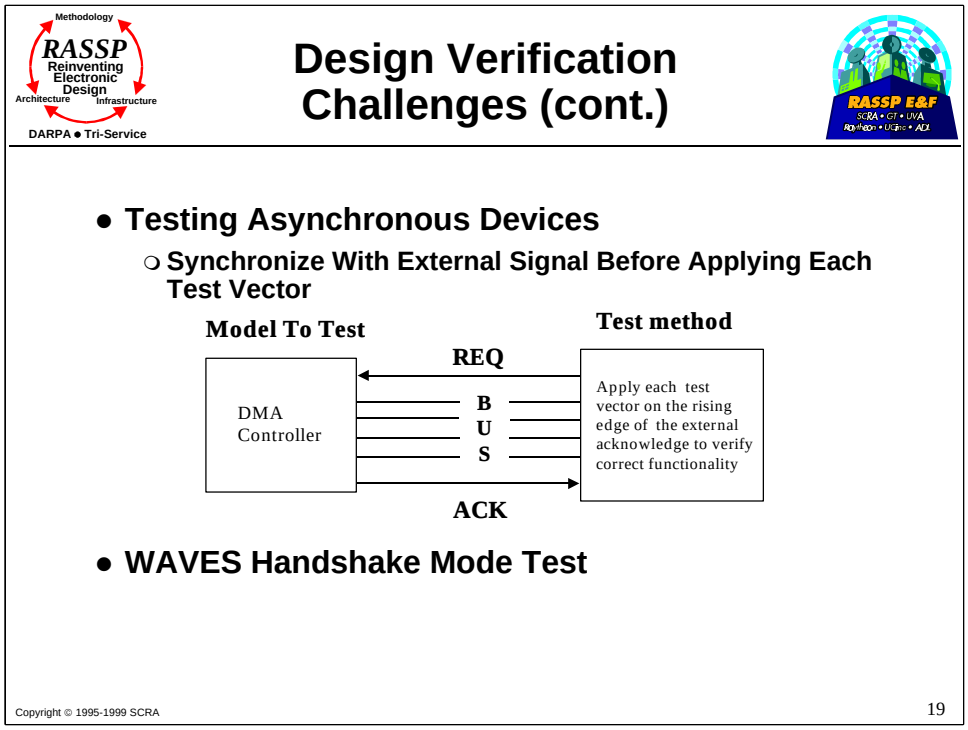
The testbench can be automatically generated using the WAVES tool set. In this slide, a signal WPL of type Waves_Port_List is declared. This signal is used to communicate signal and timing information to the UUT. The signal WMR is used within the Match function to request a Matching operation. The signal ACK is used to signify when the UUT output signals matches the desired value. The waveform generator procedure is called using the signals WPL, WMR, and ACK. [STD97]



Since this example is a VHDL simulation only, the user must add a process to control the Match ACK signal. Normally, if ATE were being used, the ACK signal would be controlled by the ATE interface. However, in this example, an extra process is required within the testbench. This process performs the actual comparison of signal values. When an event occurs on the signal WMR and the Boolean within WMR is true, then this process becomes active. The WAIT statement uses the timing information from WMR in order to sample the UUT output correctly. The FOR statement compares the UUT response with the expected response, which is the Match value. If the values agree, then ACK will be set to '1'. Otherwise, ACK will remain at logic '0'. Once the Match operation is completed, this process will not be used again unless another Match operation is requested. [Hanna97], [STD97]

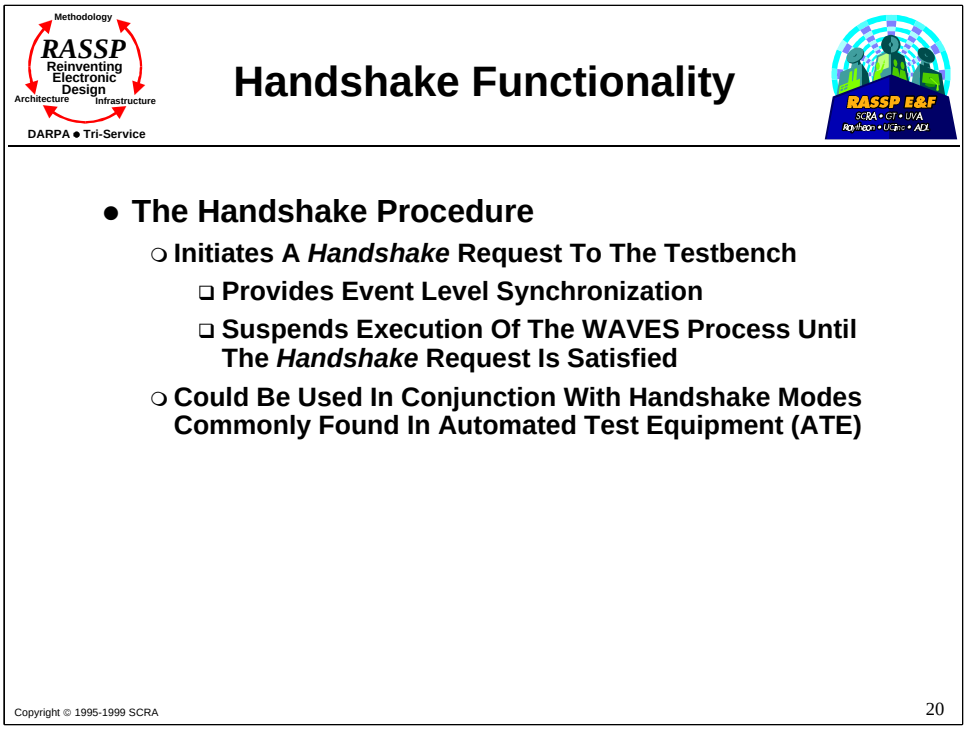
Module Outline

- Introduction
- Match Functionality
- **Handshake Functionality**
- Multiple Waveform Generators
- Multiple Timing Sets And Bidirectional Test Pins

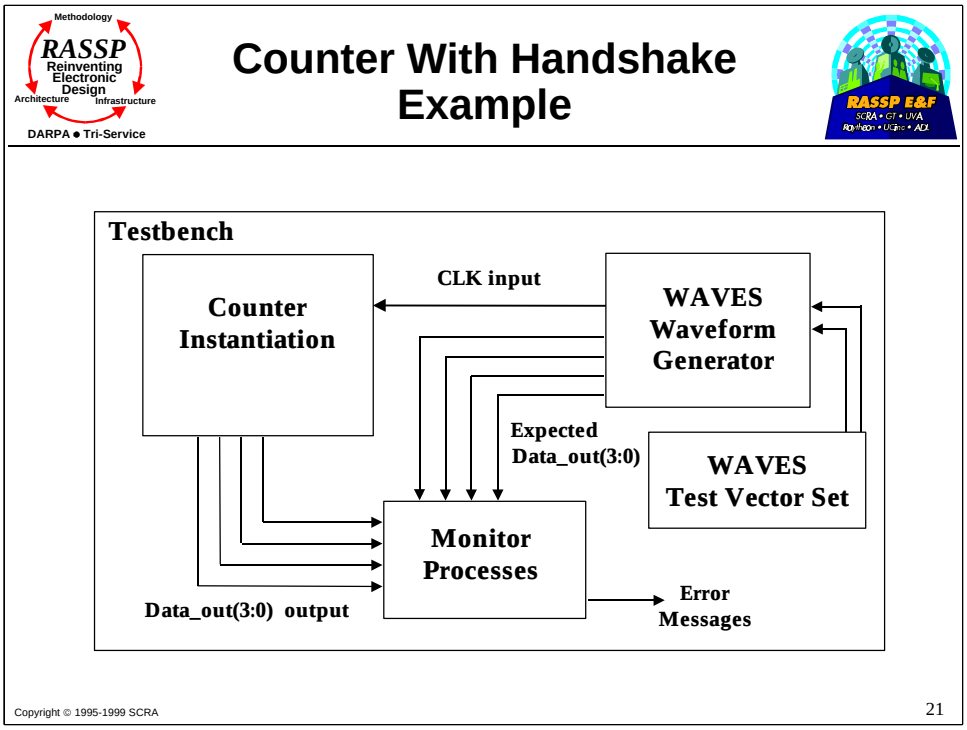


The WAVES Handshake function allows the designer to perform testing of asynchronous devices. The Handshake functionality is designed to control the application of each test vector with an asynchronous signal. In a system without a synchronous clock, the communications between component and tester is established using request and acknowledge signals. Both signals are asynchronous in nature. The tester makes a request to begin testing and the component acknowledges the request. However, the delay between a request and its acknowledgment may vary from one operation to the next. For consistency, each test vector cannot be applied until the tester has received the proper asynchronous acknowledgment. This insures that the test vector is applied at the proper time to perform its intended verification.

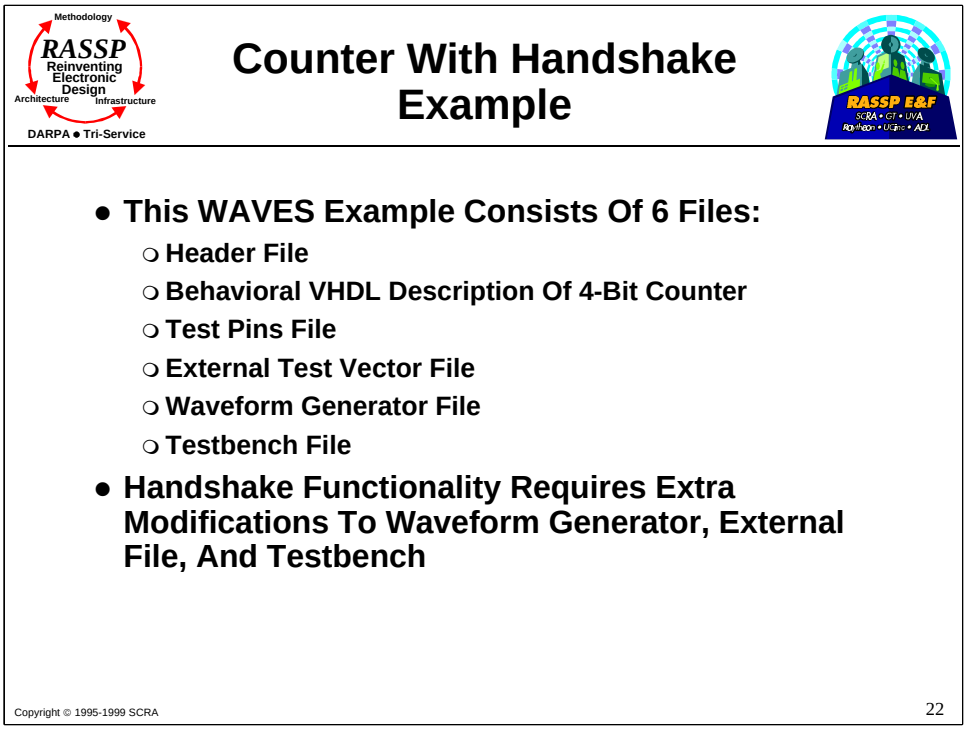
In the example shown above, the tester makes a request on the REQ line to the component. The component responds on a acknowledge (ACK) line to the tester. On the rising edge of the ACK line, the tester applies a test vector to the component. The application of the test vector has been synchronized to the asynchronous acknowledge signal. [Hanna97]



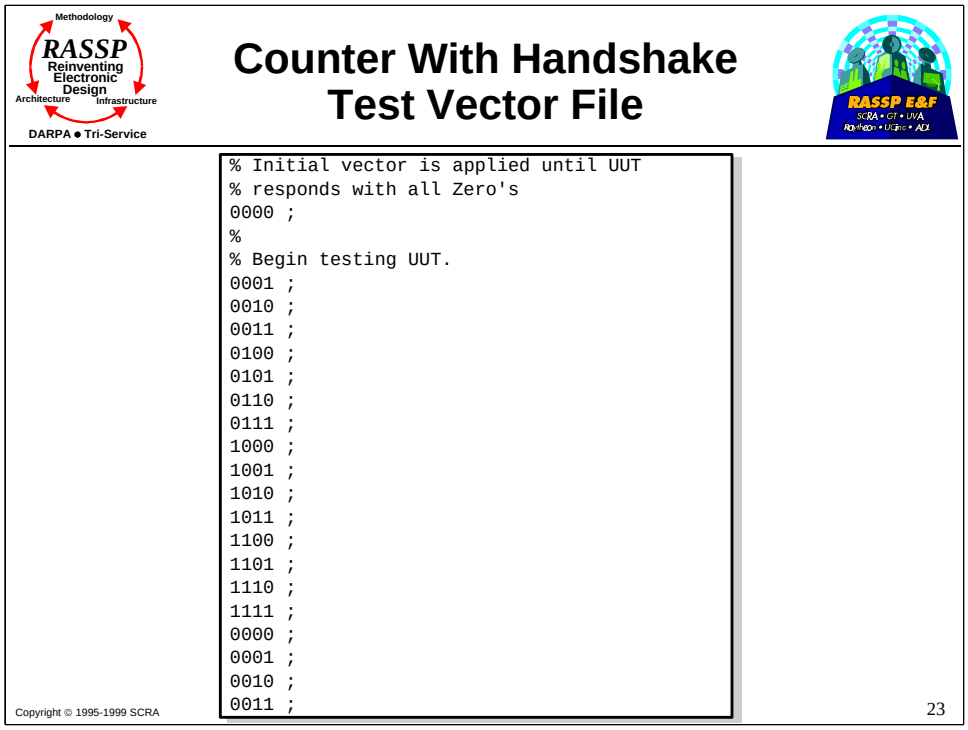
The Handshake procedure is used to achieve synchronization between asynchronous signals and the application of a test vector. This procedure specifies which signals the system will perform the Handshaking operation. This functionality will not apply a new test vector to the UUT until it is synchronized through the Handshaking process. WAVES includes Handshaking capability in order to facilitate communication with ATE. If a WAVES testbench were interfacing with ATE, then the ATE interface should generate the necessary signals for the Handshaking. [Hanna97]



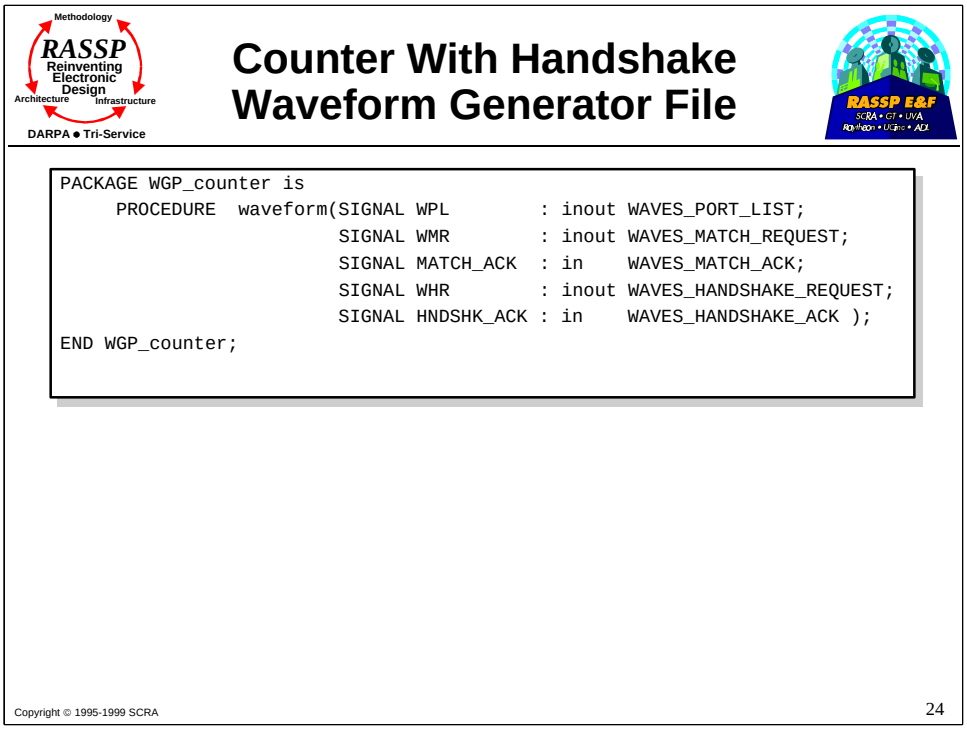
This example demonstrates the Handshake functionality with a 4-bit counter as the UUT. This example is actually an extension of the previous example since the Match functionality is also demonstrated.



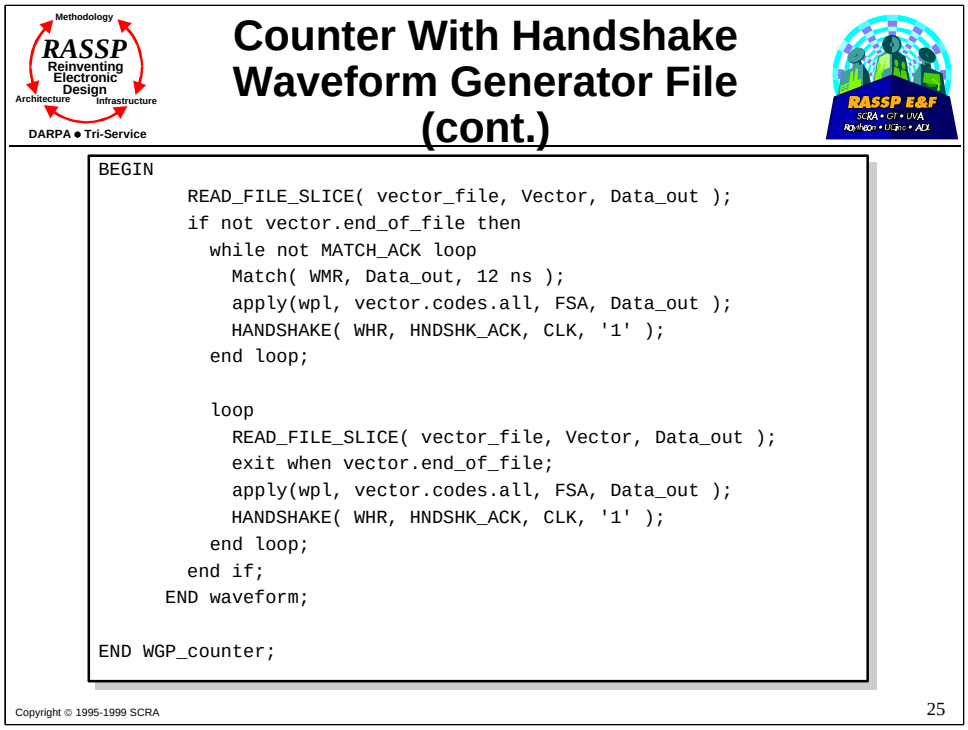
The Handshake functionality requires the designer to make extra modifications to the waveform generator and testbench. Since these changes are unique to the Handshake function, it is important to note the modifications made to the test set seen in this example.



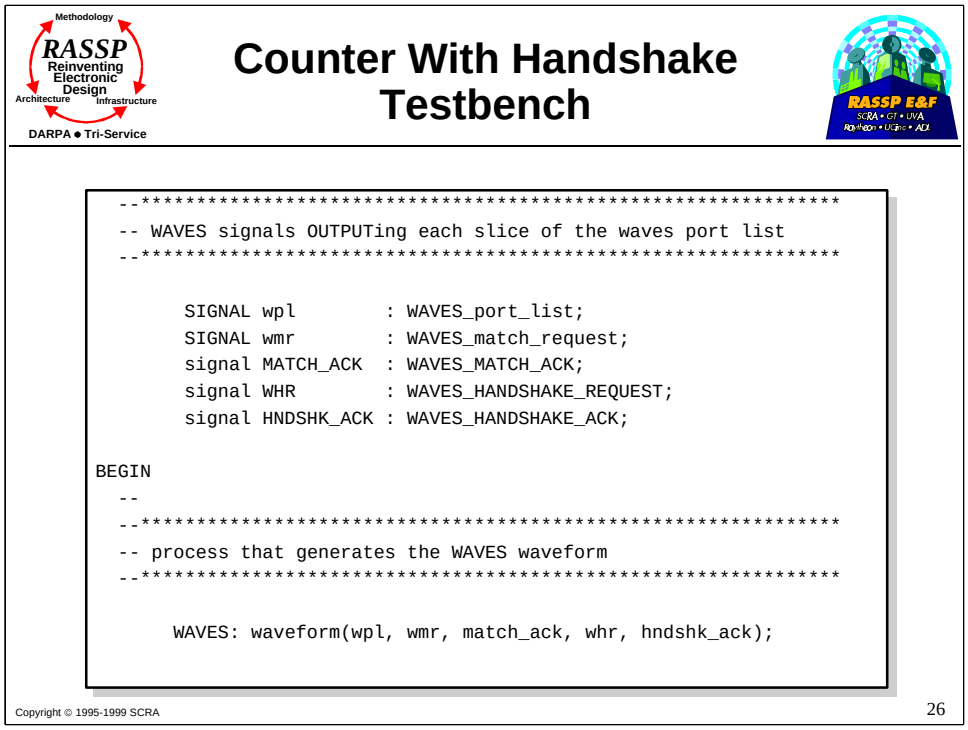
This test vector file is different than the file shown in the Counter With Match example. Since the clock input signal (called CLK) is involved in the asynchronous Handshaking, it does not require pin codes from the external file. The clock signal will be controlled by the testbench itself. Therefore, this external file only contains pin codes for the 4-bit output signal (called Data_out). No slice timing information is included because the slice timing is determined by the Handshaking operation within the testbench. The first file slice shown is actually the initial value for the state of the counter. This example is using the Match functionality to establish a valid starting point for applying test vectors.



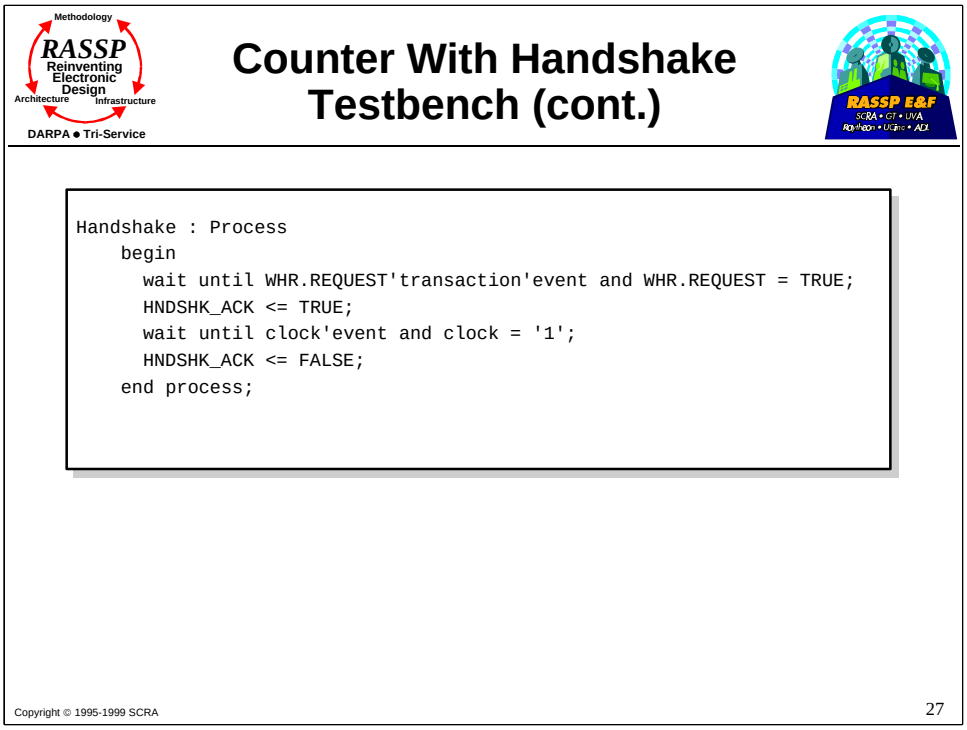
In the waveform generator, the procedure Waveform now includes five signals. The signal WPL, which has been seen before, communicates signal and event information to the UUT. The WMR and MATCH_ACK signals are used to handle the Match functionality described in the last section. The Handshake functionality is controlled by two new signals. The signal WHR is used as a request to begin a Handshaking operation. WHR is actually a record which will store a test pin and the logic value required to satisfy the request. The signal HNDSHK_ACK is used as the asynchronous acknowledgment by the test environment for a given Handshake request. [STD97]



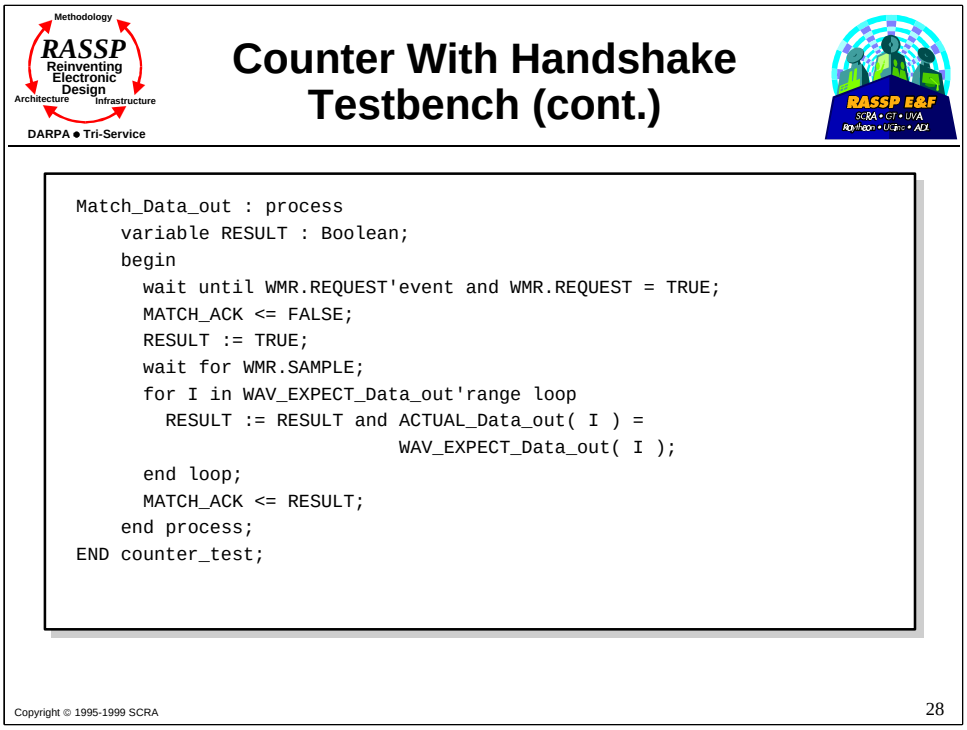
In a fashion similar to the Match example, this waveform generator has two LOOP statements. As stated before, this example is actually demonstrating both the Match and Handshake functionality. The two LOOP statements are almost identical the the waveform generator for the Counter With Match example. The first loop controls the Match operation, while the second controls the application of the remaining test vectors. However, the Handshake procedure appears where the DELAY procedure has normally appeared. The Handshake procedure requests a handshaking process from the testing environment by setting a Boolean flag in WHR. The WAVES test set is suspended until HNDSHK_ACK goes high and then low. Once the testing environment sees a Handshake request, it sets the acknowledge signal high and waits for the specified UUT signal to achieve the specified value. Once the desired value appears on the signal, the environment sets HNDSHK_ACK low which completes the Handshaking cycle. In this example, the environment waits until the signal CLK goes to a logic '1'. When it does, then the Handshaking cycle is complete. [Hanna97], [STD97]



The five signals controlling the testing of the counter are formally declared in the testbench. The Match and Handshake functions require their own request and acknowledge signals.



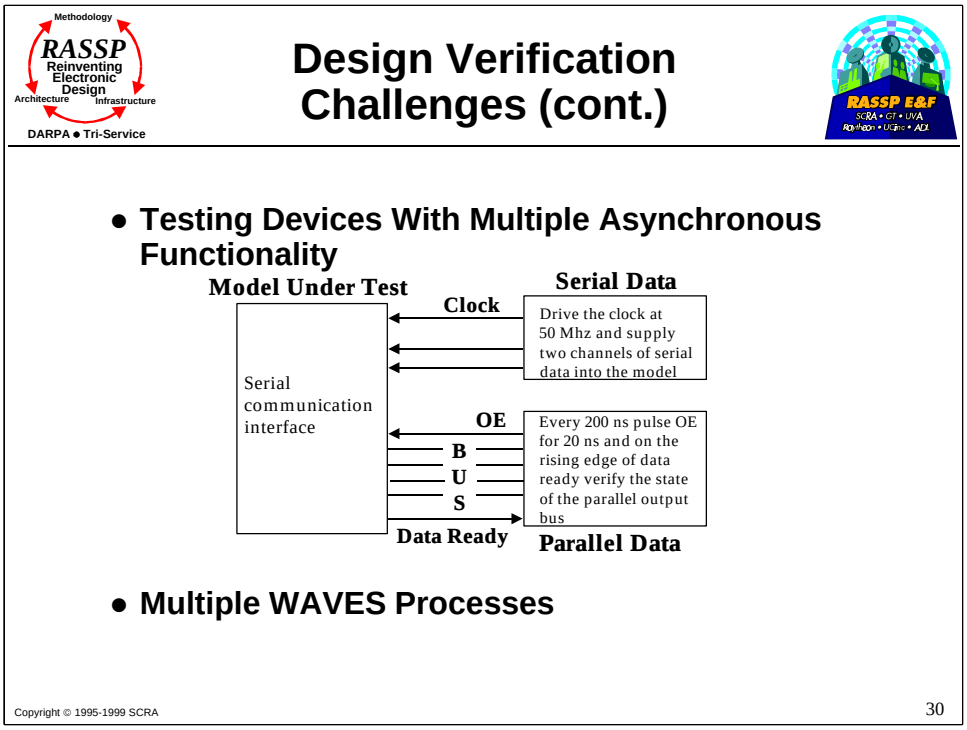
This process must be added to the testbench by the designer. Normally, if this WAVES testbench were interfacing with ATE, then the ATE interface would drive the Handshake acknowledge signal. However, since this example is only a VHDL simulation, an extra process must be included within the testbench. Once this process sees a valid Handshake request through WHR, then HNDSHK_ACK is set to TRUE. Next, the process waits until the clock signal becomes '1'. This is the signal and desired value that was specified in the waveform generator. Once the UUT has responded correctly, the acknowledge signal is set to FALSE which will terminate the Handshaking cycle. Therefore, this process mimics the behavior of ATE performing a Handshaking operation. [Hanna97], [STD97]



This process performs the Matching operation which was described in the previous example.

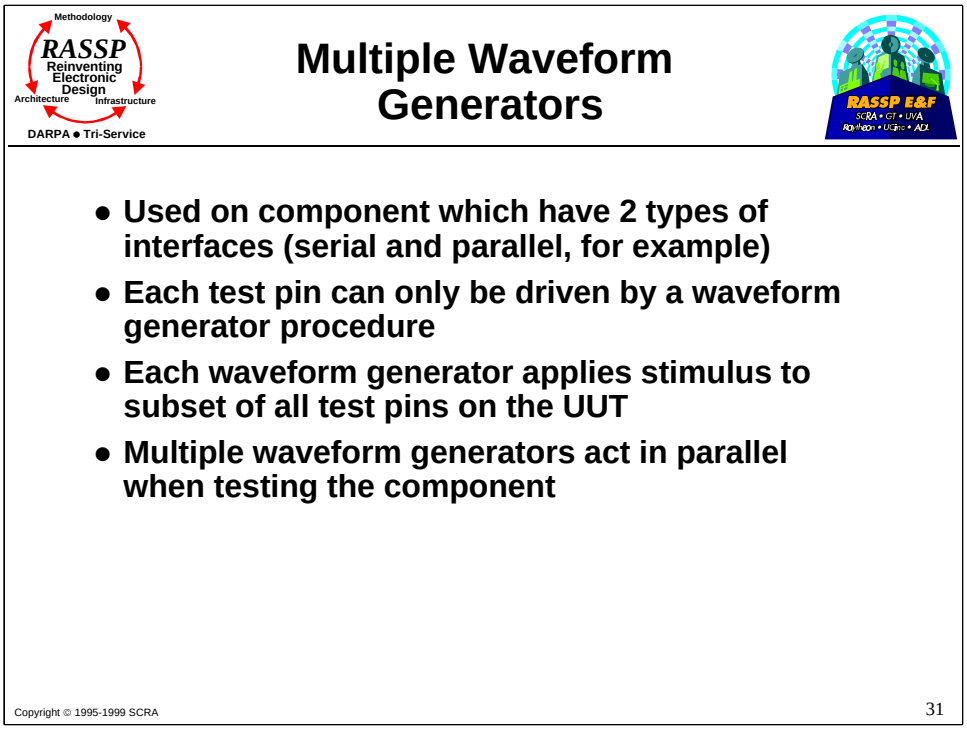
Module Outline

- Introduction
- Match Functionality
- Handshake Functionality
- **Multiple Waveform Generators**
- Multiple Timing Sets And Bidirectional Test Pins

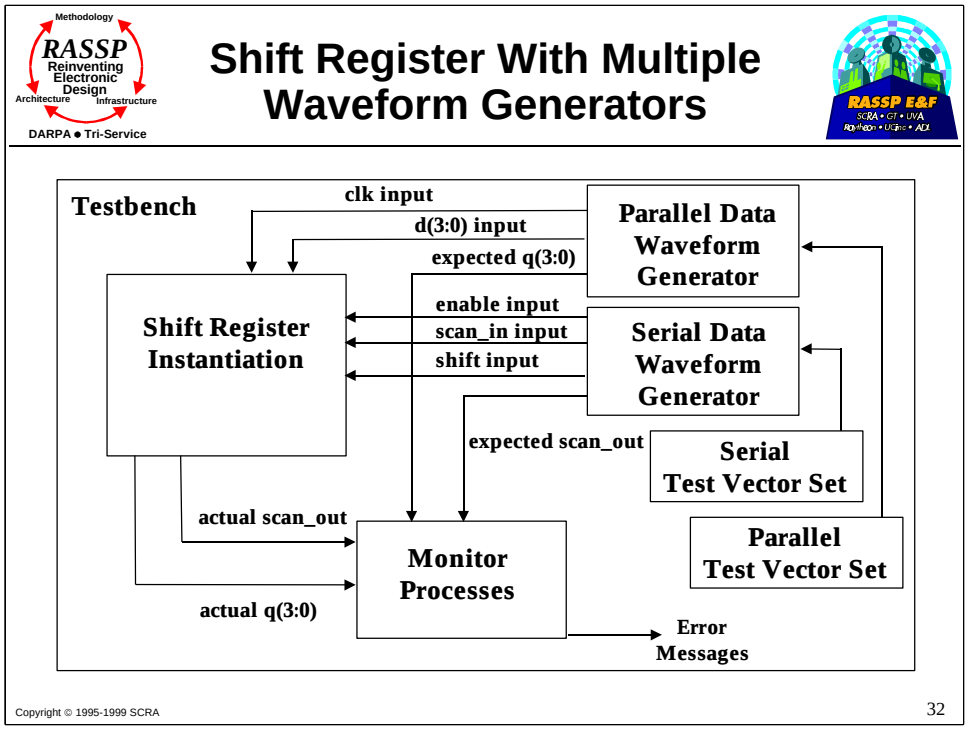


It is possible to create multiple waveform generators in WAVES which allows for multiple simultaneous tests of a component. Each waveform generator is designed to perform one type of testing for a particular component. The two waveform generators can perform their actions in parallel within the same testbench. However, each test pin can be driven only by one waveform generator. In other words, both waveform generators cannot drive the same test pins on the component under test.

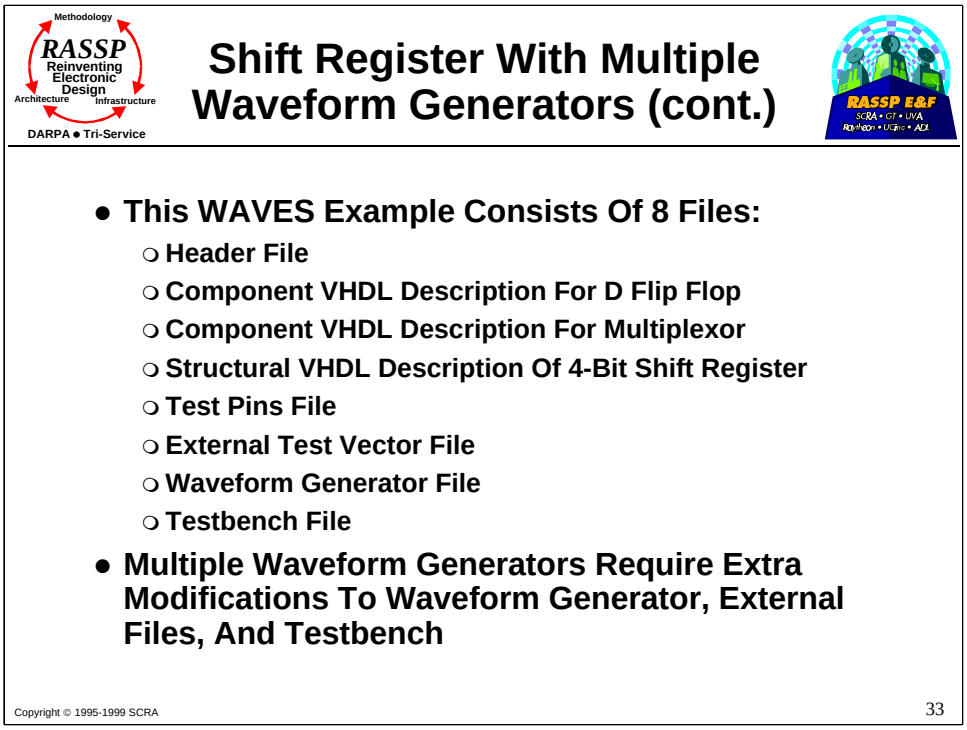
In the example shown above, the top waveform generator provides only serial data to the model under test. The bottom waveform generator drives the output enable (OE) line to allow the component to place data on the bus. When the component places the data on the bus and raises the Data Ready line, the waveform generator verifies the data on the bus. Therefore, two waveform generators are providing two separate testing operations on the same component under test.



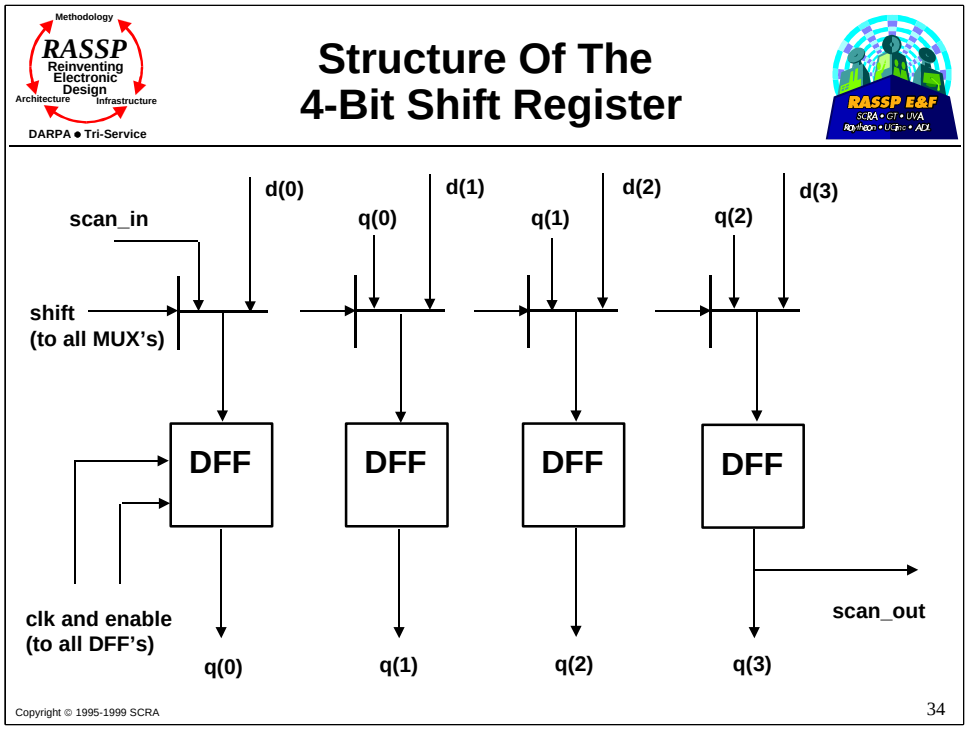
Multiple waveform generators can be used on a component which has two different types of interfaces. Each waveform generator would be designed to perform a unique testing operation on the component. However, each test pin can only be driven by one waveform generator.



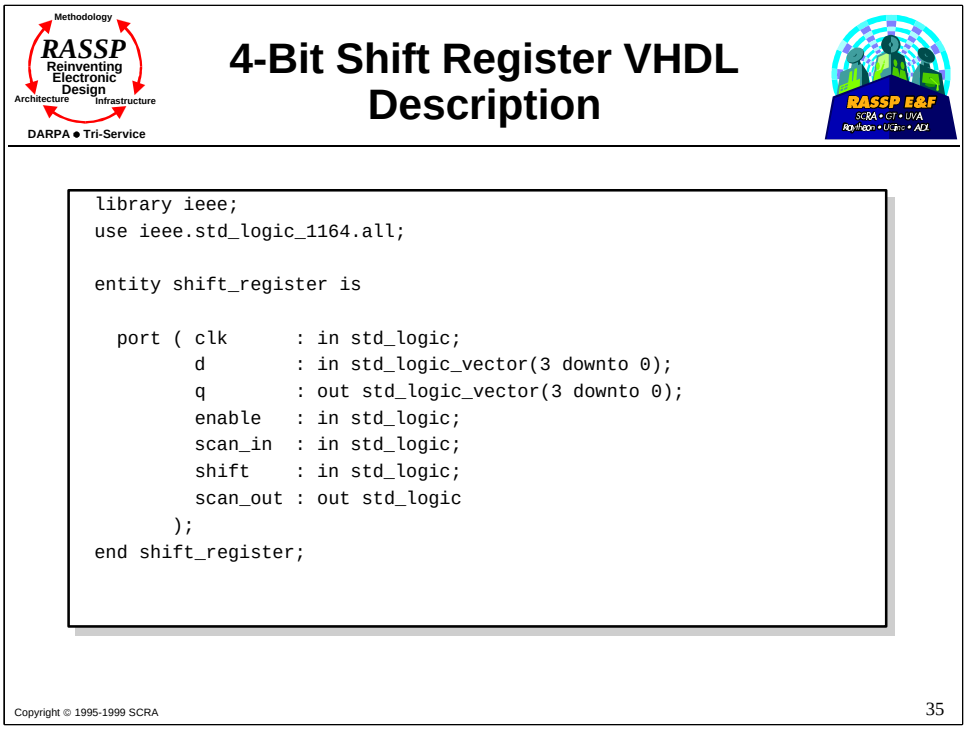
This example demonstrates the use of multiple waveform generators within the testbench. Each waveform generator has a unique test vector file. Also, each waveform generator interacts with a specific subset of the test pins. In this example, the component under test is a shift register. The test pins on the shift register are divided into two categories: serial data and parallel data. Two waveform generators are created in the test set. One generator will provide serial data to the shift register, while the other will provide parallel data.



The use of multiple waveform generators requires additional modifications to the various elements of the WAVES test set. The changes specific to the use of multiple waveform generators will be discussed throughout this example.



This diagram shows the structure of the shift register used in this example. The four multiplexors choose the input for each d flip flop every clock cycle. The common select line between the multiplexors is the shift line. The signal scan_in acts acts an input to the lowest bit position during shifting. The signal scan_out monitors the highest bit during shifting.

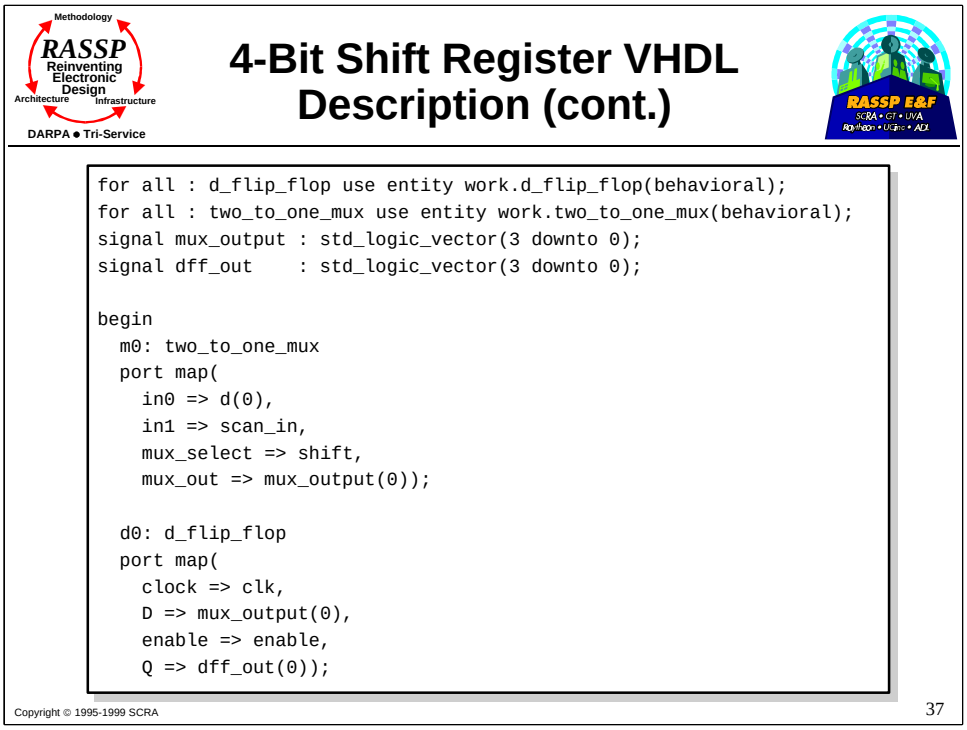


This is the entity description for the 4-bit shift register. The enable input controls when the input bus d is stored in the register. The shift input controls whether the register is shifting or not. The scan_in input is used to provide serial input data during shifting. The scan_out output is used to monitor the results of the shifting operation.

4-Bit Shift Register VHDL Description (cont.)

```
architecture structural of shift_register is

component d_flip_flop
  port ( clock  : in  std_logic ;
         D      : in  std_logic ;
         enable : in  std_logic ;
         Q      : out std_logic ;
         Q_bar  : out std_logic
       );
end component;

component two_to_one_mux
  port ( in0        : in  std_logic;
         in1        : in  std_logic;
         mux_select : in  std_logic;
         mux_out    : out std_logic
        );
end component;
```


The d flip flop and 2x1 multiplexor are formally declared as components in the shift register description.



The shift register consists of 4 d flip flops and 4 multiplexors. The next few slides show the port mapping of the various components.

4-Bit Shift Register VHDL Description (cont.)

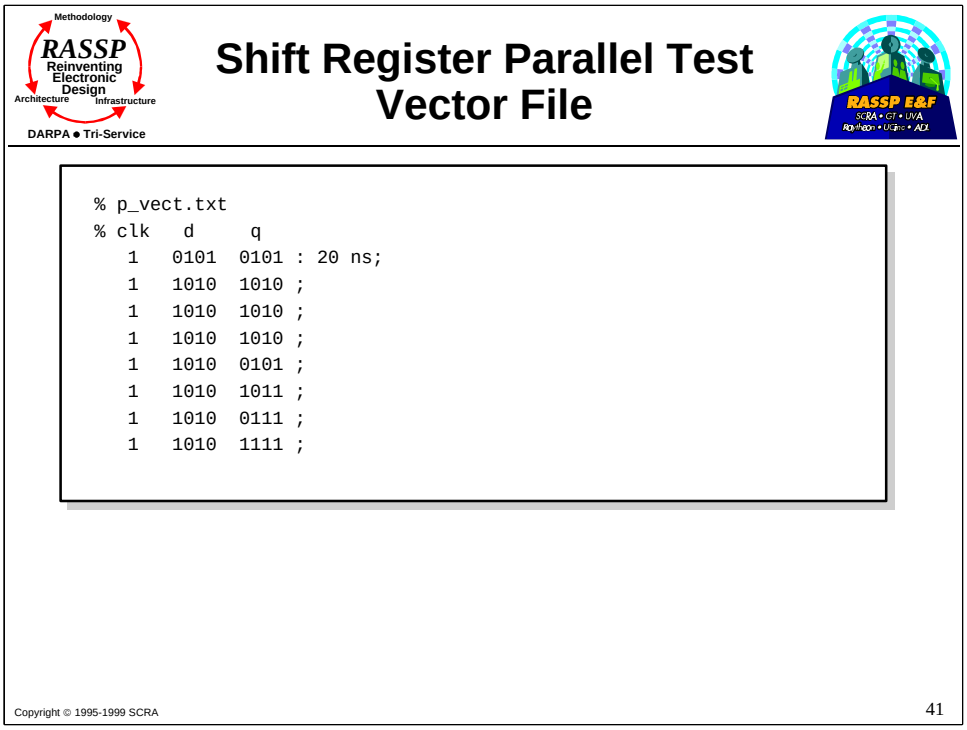

```
m1: two_to_one_mux
port map(
  in0 => d(1),
  in1 => dff_out(0),
  mux_select => shift,
  mux_out => mux_output(1));

d1: d_flip_flop
port map(
  clock => clk,
  D => mux_output(1),
  enable => enable,
  Q => dff_out(1));

m2: two_to_one_mux
port map(
  in0 => d(2),
  in1 => dff_out(1),
  mux_select => shift,
  mux_out => mux_output(2));
```

4-Bit Shift Register VHDL Description (cont.)

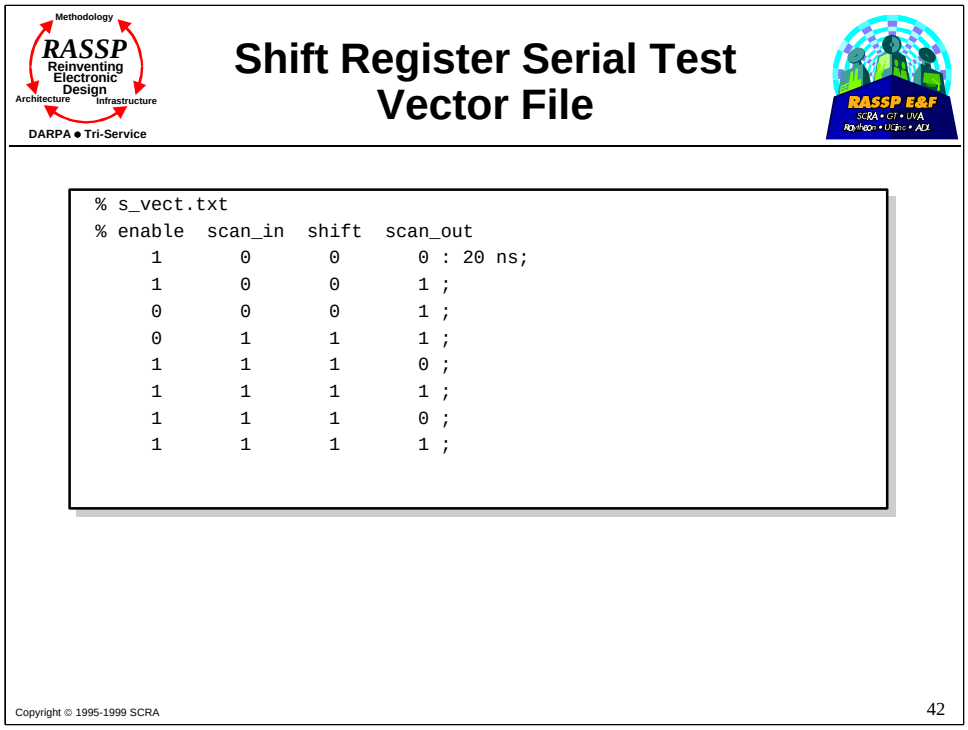

```
d2: d_flip_flop
port map(
  clock => clk,
  D => mux_output(2),
  enable => enable,
  Q => dff_out(2));

m3: two_to_one_mux
port map(
  in0 => d(3),
  in1 => dff_out(2),
  mux_select => shift,
  mux_out => mux_output(3));

d3: d_flip_flop
port map(
  clock => clk,
  D => mux_output(3),
  enable => enable,
  Q => dff_out(3));
```

4-Bit Shift Register VHDL Description (cont.)

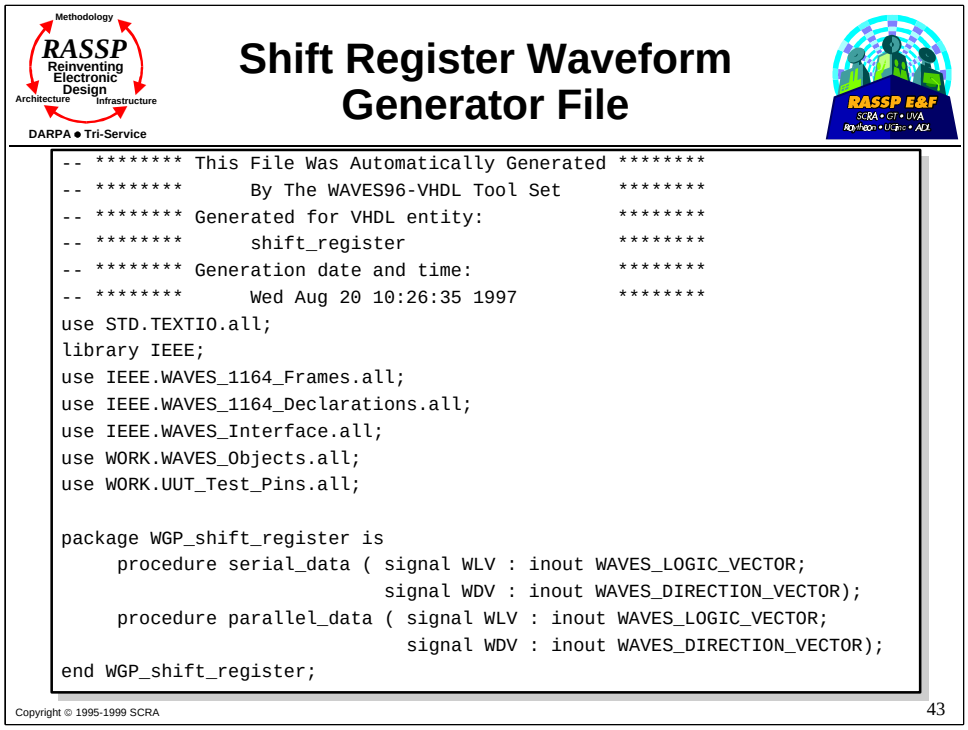

```
    scan_out <= dff_out(3);

    q(0) <= dff_out(0);
    q(1) <= dff_out(1);
    q(2) <= dff_out(2);
    q(3) <= dff_out(3);

  end structural;
```



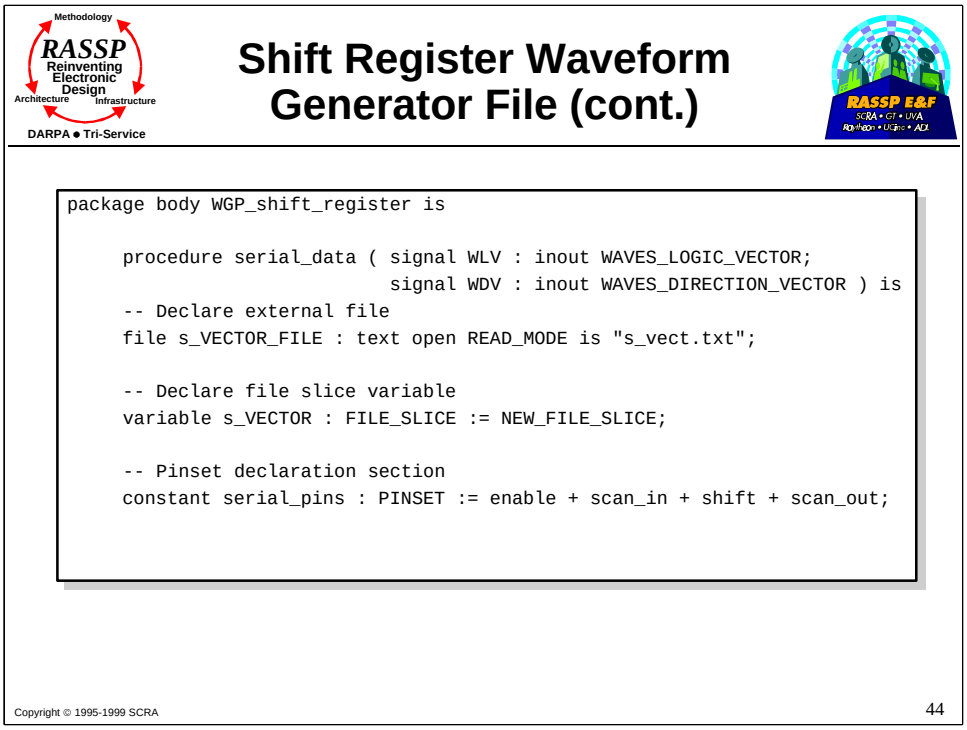
This is the test vector file for the parallel data waveform generator. Pin codes are included for clk, the d input bus, and the q output bus. The slice duration is defined as 20ns for each test vector.



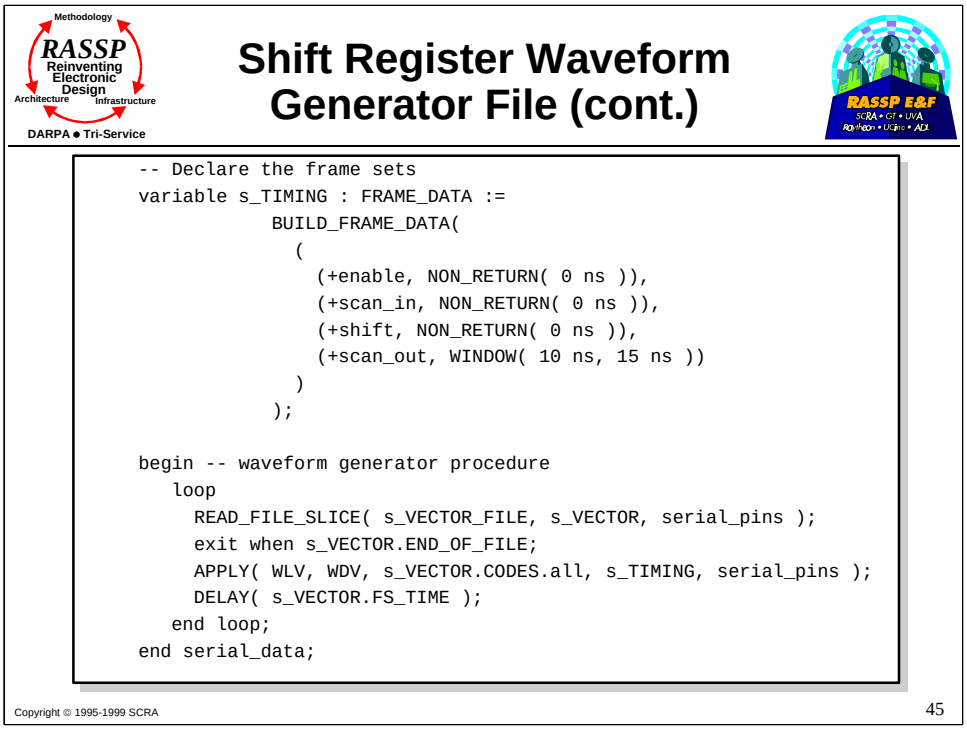
This is the test vector file for the serial data waveform generator. Pin codes are included for the serial input pins: enable, scan_in, shift, and scan_out. The slice duration is defined as 20ns in this case, which is consistent with the parallel test vector file. Therefore, in this example, the two waveform generators will be operating in synchronization. However, it is possible to have multiple waveform generators operating asynchronous from each other. [Hanna97]



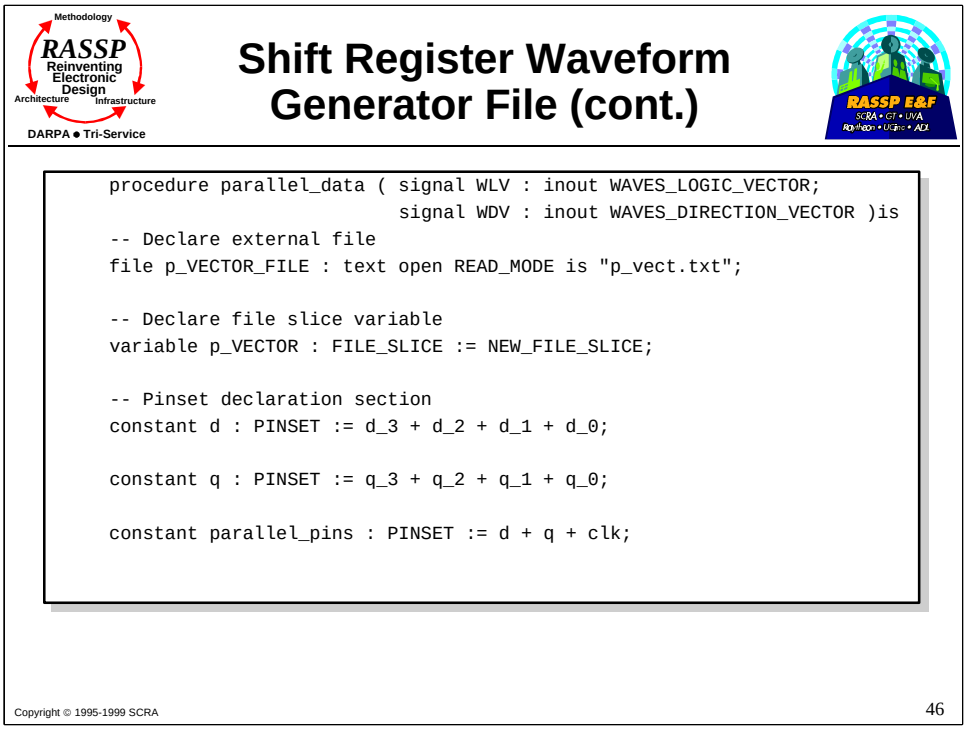
In the package declaration for WGP_shift_register, there are two procedures declared: serial_data and parallel_data. When using multiple waveform generators, the user must include the signals WLV and WDV in the procedure declarations. This is a change from all the previous examples described earlier. The signals WLV and WDV allow specific test pins to be mapped to each waveform generator. WLV is used to associate a logic value with each test pin on the shift register. WDV is used to associate a direction for each test pin. [Hanna97], [STD97]



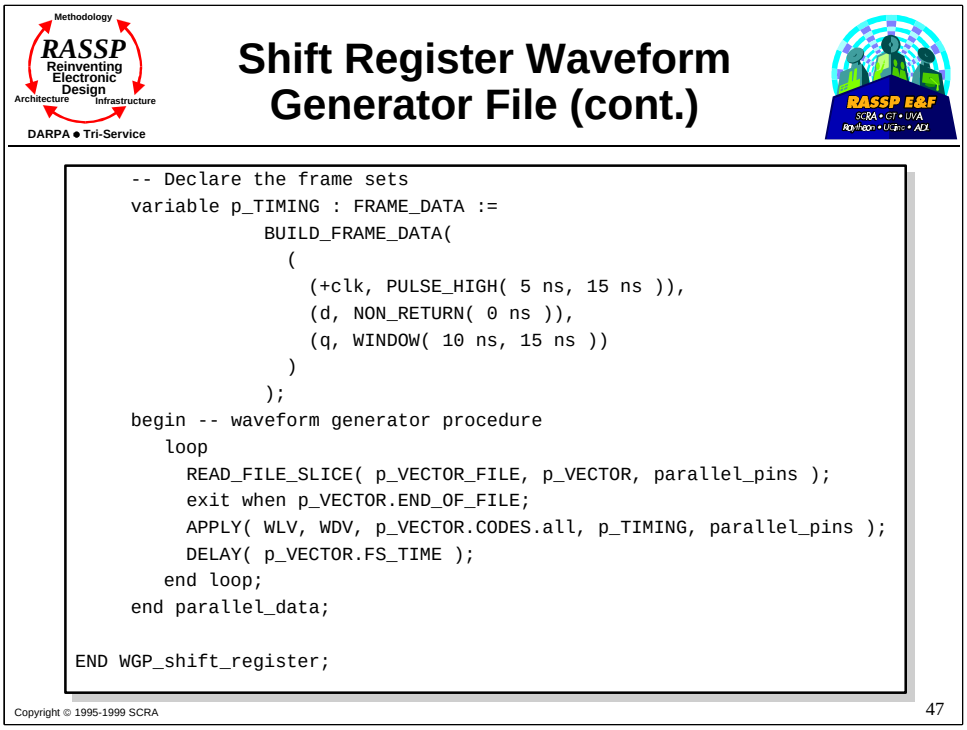
This is the procedure description for the serial data waveform generator. The serial test vector file is formally declared in the FILE statement seen above. The variable s_Vector will be used to store file slice information from the serial test vector file. A pinset is declared which includes all the serial test pins on the shift register.



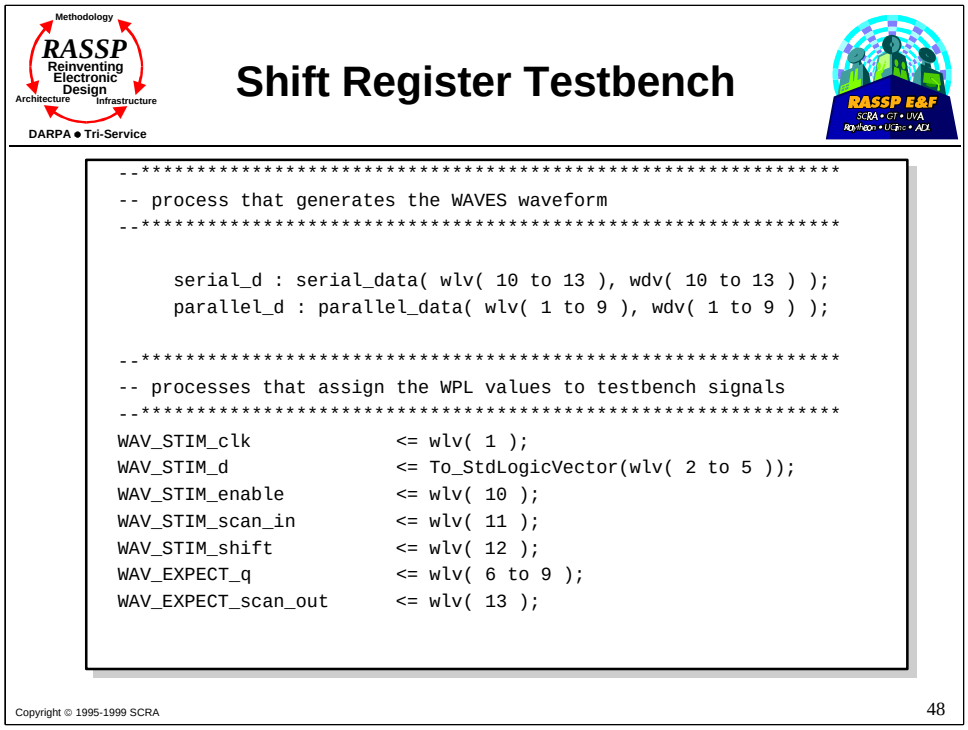
The frame sets for the serial waveform generator is declared. The Non_Return format is used on the serial input pins enable, scan_in, and shift. A 5ns Window format is used on the output pin scan_out. There are some changes required in the loop statement as well. In the call to the READ_FILE_SLICE procedure, the pinset serial_pins is included as a parameter. This insures that s_Vector will only store pin codes for the pinset serial_pins. In other words, s_Vector will only store 4 pin codes instead of a pin code for every test pin on the shift register. In the APPLY procedure, the signals WLV and WDV are included as parameters. The pinset serial_pins is included as well because this APPLY function will only schedule events on the pinset serial_pins. [STD97]



This is the procedure description for the parallel data waveform generator. The parallel test vector file is declared, along with the variable p_Vector. Two pinsets are declared: one for the d input bus and one for the q output bus. Another pinset parallel_pins is declared which includes all the test pins that this waveform generator will interface with.



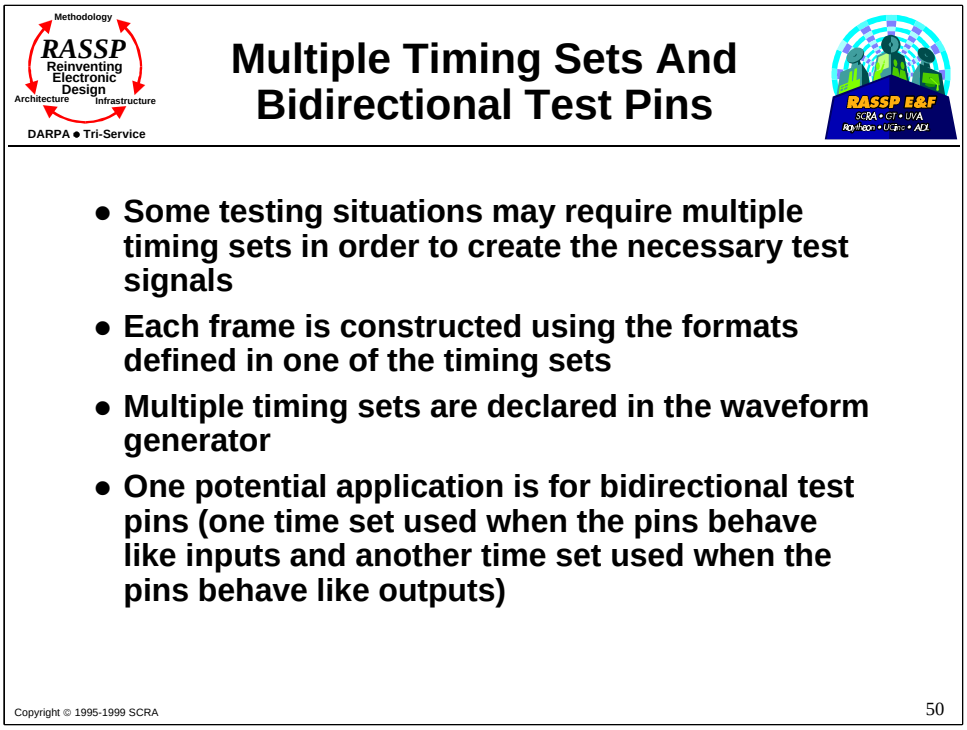
The frame sets are declared in the variable p_TIMING shown above. The PULSE_HIGH format is used on the input clk, establishing a 20ns clock period in this test set. The Non_Return format is used on the pinset d, while a 5ns Window format is used on the pinset q. In a manner similar to the serial waveform generator, the pinset parallel_pins is used in the Read_File_Slice and Apply procedures shown in the loop statement.



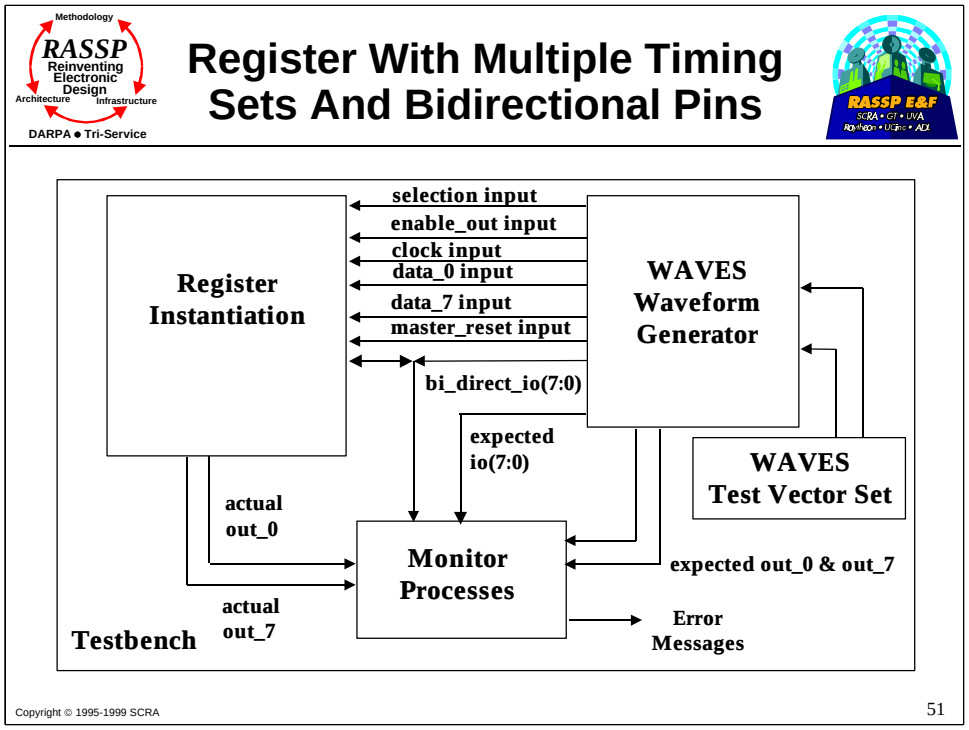
In the testbench, the processes serial_d and parallel_d are declared to generate the waveform. Each process uses the information stored in wlv and wdv for its intended test pins. The logic values in wlv are mapped to the various test signals in the testbench. The indexing into wlv and wdv obeys the order established in the test pin file.

Module Outline

- Introduction
- Match Functionality
- Handshake Functionality
- Multiple Waveform Generators
- **Multiple Timing Sets And Bidirectional Test Pins**



This section describes two advanced features of WAVES: multiple timing sets and bidirectional test pins. Multiple timing sets allow the frames to be constructed with different formats. During each slice, one frame set is selected to form the desired segment of the waveform. Bidirectional test pins can be included on the interface of the UUT. The WAVES test set must be modified to address bidirectional test pins. Bidirectional test pins are a good application of multiple timing sets since the test pins typically require one format for input, and another format for output. [Hanna97]

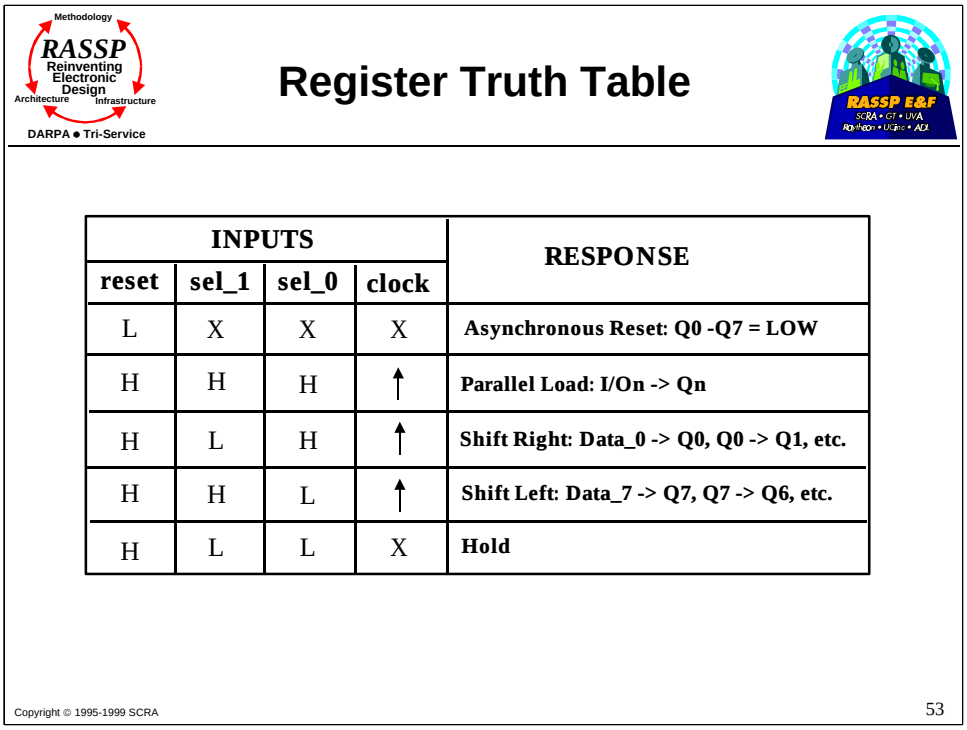


This example includes an 8-bit shift/storage register with bidirectional pins on the I/O bus. The diagram above shows the various signals produced by the waveform generator in this example. Note that the bi_direct_io bus can either send data from the waveform generator to the UUT or send data from the UUT to the monitor processes. This shows the bidirectional operations on this particular bus.

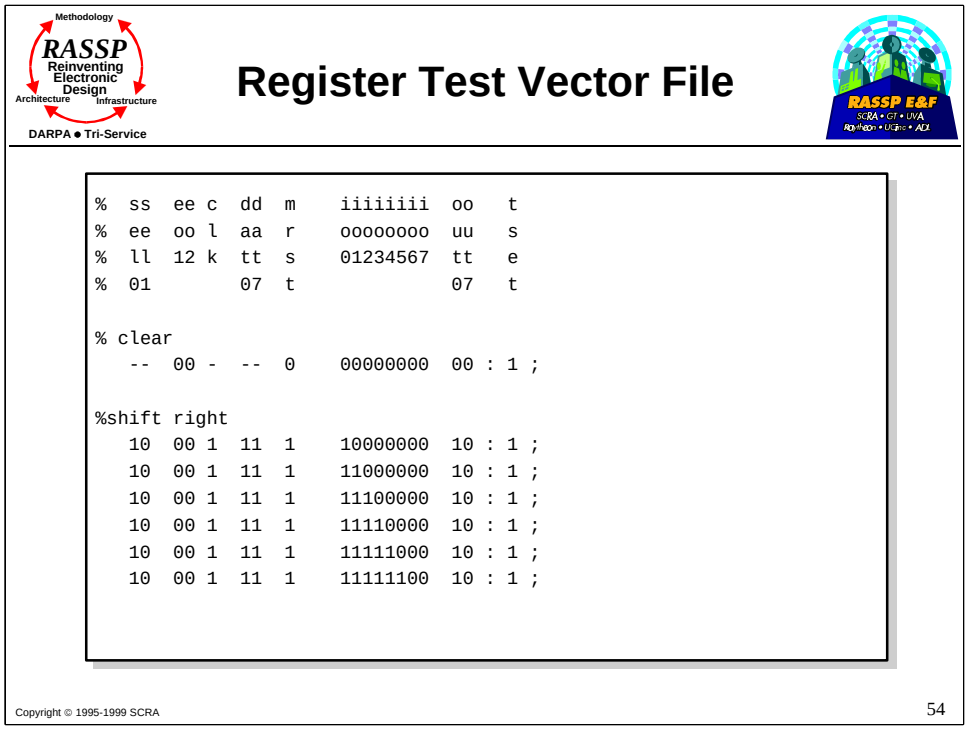
Register With Multiple Timing Sets (cont.)

- **This WAVES Example Consists Of 6 Files:**
  - **Header File**
  - **Component VHDL Description For Register**
  - **Test Pins File**
  - **External Test Vector File**
  - **Waveform Generator File**
  - **Testbench File**
- **Multiple Timing Sets And Bidirectional Test Pins Require Extra Modifications To Waveform Generator, External File, And Testbench**

The WAVES test set for this example consists of 6 files shown above. The use of multiple timing sets and bidirectional test pins require unique modifications to the waveform generator, external file, and testbench. The designer should take note of the changes in the WAVES test set that are specific to these operations.



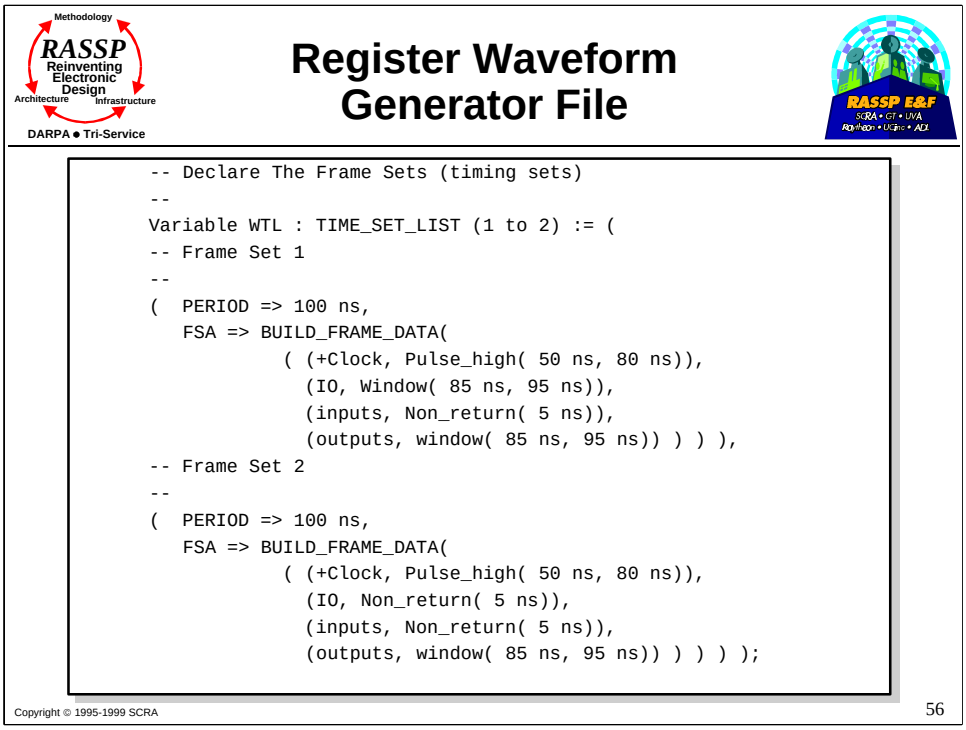
This table summarizes the behavior of the shift/storage register used in this example. This device can perform 5 unique operations as shown above. Except for reset, the 2-bit selection bus determines the operation that the register is to perform. [Hanna97]



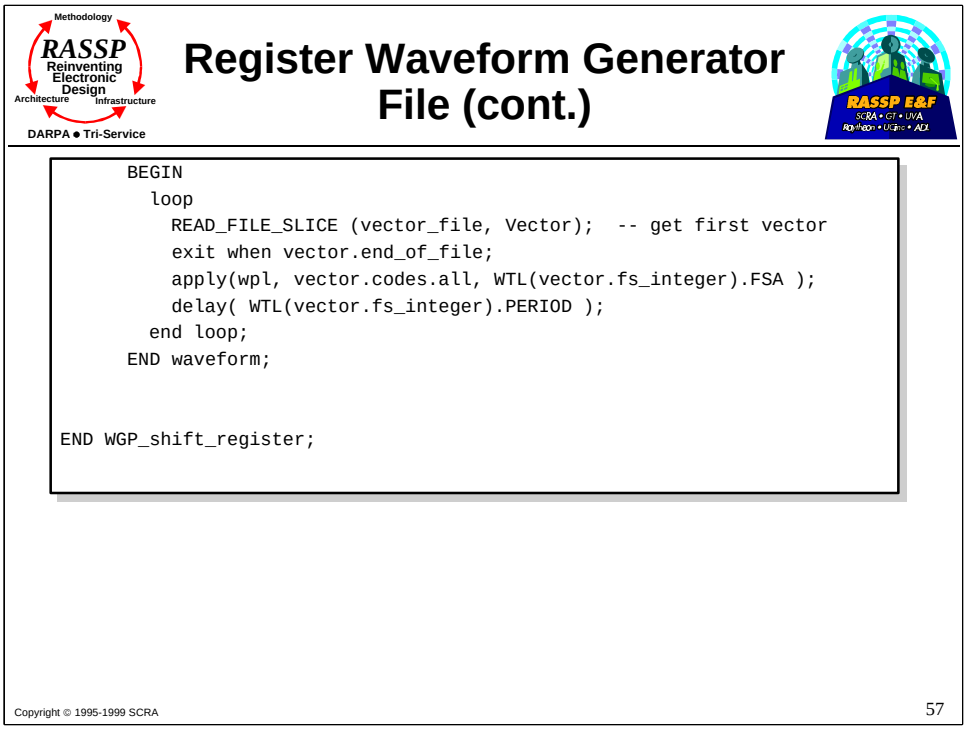
This is the test vector file for this example. The pin codes are organized according to the order shown at the top of the file. This file will perform several operations on the register during simulation. First, the register is reset. Then, six shift right operations are performed, followed by five shift left operations. The register then performs four hold operations. The register is then loaded with data and the output is disabled with the loaded information shifted left. The I/O pins are tested for a tri-state condition and the last operation enables the outputs and performs a final shift left operation. Note the integer at the end of each file slice. Instead of specifying a slice duration time, a timing set selection integer is specified. This integer identifies which timing set is used with the pin codes to construct the current slice. In this case, there are only two timing sets specified. [Hanna97]

Register Test Vector File (cont.)

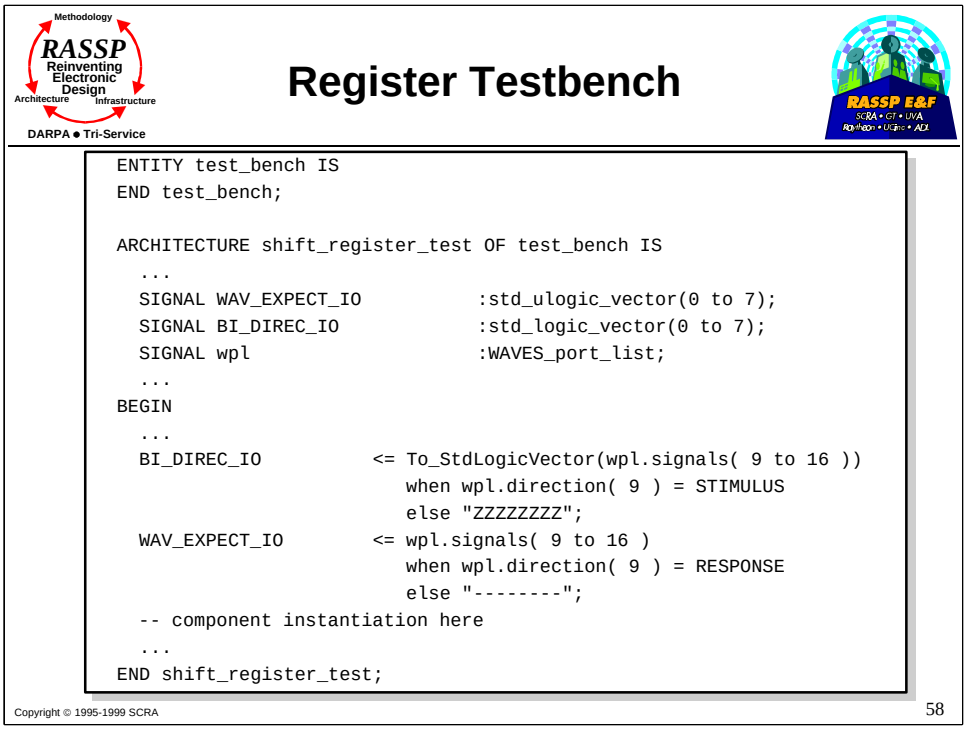
```
%shift left
   01  00 1  11  1    11111001  11 : 1 ;
   01  00 1  11  1    11110011  11 : 1 ;
   01  00 1  11  1    11100111  11 : 1 ;
   01  00 1  11  1    11001111  11 : 1 ;
   01  00 1  10  1    10011110  10 : 1 ;

% hold
   00  00 1  10  1    10011110  10 : 1 ;
   00  00 1  10  1    10011110  10 : 1 ;
   00  00 1  10  1    10011110  10 : 1 ;
   00  00 1  10  1    10011110  10 : 1 ;

% load
   11  10 1  10  1    01010101  01 : 2 ;

% enable & shift
   01  01 1  01  1    ZZZZZZZZ  11 : 1 ;
   01  00 1  01  1    01010111  01 : 1 ;
```



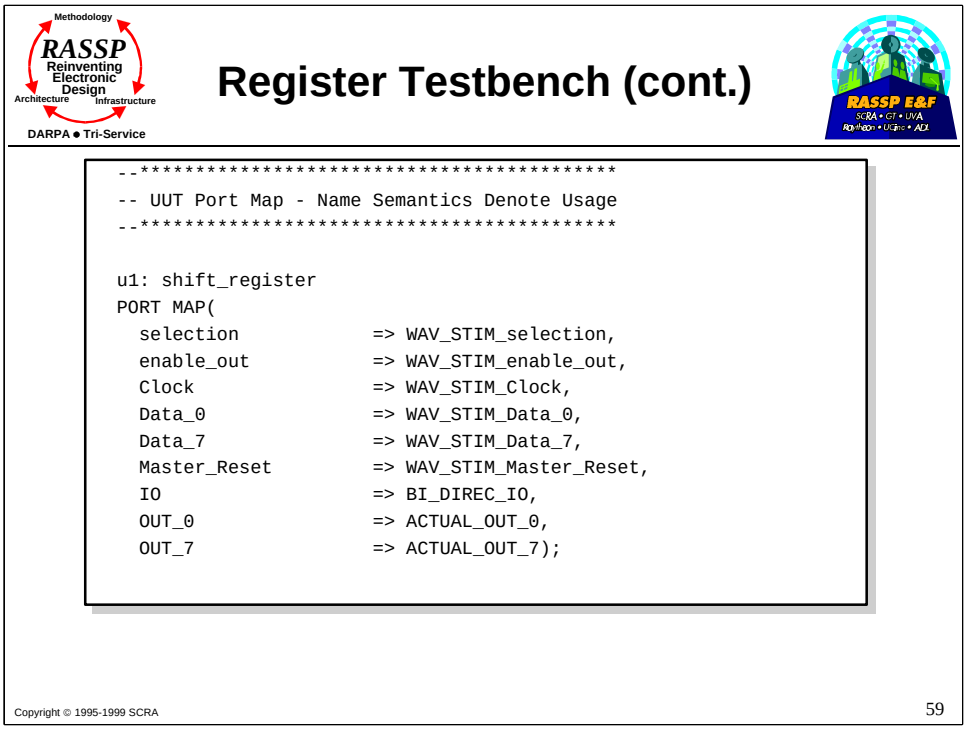
In this slide, the two timing sets are explicitly declared in the waveform generator. The variable WTL is of type TIME_SET_LIST, which is an array of time sets. Each element of TIME_SET_LIST will contain a timing specifier Period and frame format information in FSA. In this example, the only difference between the two timing sets is the format for the bidirectional IO bus. In the first timing set, the Window format is used on IO, which means that IO bus is acting like an output. This timing set is intended for the shift and hold operations. In the second timing set, the Non Return format is used on IO, which means that the bus is acting like an input. This timing set is intended for the load operation. [Hanna97], [STD97]



This slides shows the loop statement which controls the reading of test vectors and scheduling of events on the test pins. The variable Vector has a field fs_integer which will store the timing set selection integer found in the external file. In the Apply procedure, note that the variable WTL is included as a parameter. The events will be scheduled using the currently selected timing set specified by the fs_integer field of Vector. For example, if fs_integer is '1', then the first timing set specified in the previous slide will be used. The Window format will be used on the IO bus in this case. The Delay procedure also includes WTL as parameter in order to read the timing information stored in Period. In this example, both timing sets have a Period of 100ns, which defines the slice duration for this test set. In summary, the use of multiple timing sets requires an explicit TIME_SET_LIST to be declared in the waveform generator. This list is indexed using the timing set selection integer found in the external file. Then, the corresponding formats are used to create the current slice of the waveform. [Hanna97], [STD97]



The slide summarizes the specific parts of the testbench that address the bidirectional signals. In the testbench, an expected output value signal is declared for the IO bus (called Wav_Expect_IO). The testbench also includes the signals Bi_Direc_IO and WPL. A concurrent process is set up in the architecture of the testbench for the Bi_Direc_IO signal. When the direction specified in WPL is stimulus, then the IO bus is acting as an input. Therefore, the signal Bi_Direc_IO will carry input stimuli from the waveform generator to the UUT test pins. In the second process shown, the Wav_Expect_IO signal will take a value when the IO bus direction is response. This means that the IO bus is acting as an output, and an expected output signal must be created by the waveform generator. In this case, the signal Bi_Direc_IO is being driven by the UUT and not by the waveform generator. In summary, the direction information in WPL is used to drive the two IO signals to their desired values.



Note in the instantiation of the register that the signal Bi_Direc_IO is mapped to the bidirectional test pins on the UUT. A monitor process is included to monitor the IO bus when it acts as an output. The signals Wav_Expect_IO and Bi_Direc_IO are used in this comparison.

References

[Hanna97] Hanna, James P., Robert G. Hillman, Herb L. Hirsch, Tim H. Noh, Ranga R. Vemuri. Using WAVES And VHDL For Effective Design And Testing. Kluwer Academic Publishers, Boston, 1997.

[IEEE] All referenced IEEE material is used with permission.

[STD97] Draft IEEE Standard For VHDL Waveform And Vector Exchange (WAVES), IEEE Standard 1029.1-1996, IEEE Computer Society & IEEE Standards Coordinating Committee 20, May 1997.