

Exit port schemes

OTPL: Test result selects exit port. Test result can be any integer number. Each exit port can handle either a single numerical result, or a list or range of positive or negative numbers.

```
Flow FlowTest1
{
  FlowItem FlowTest1_Min MyFunctionalTest1Min
  {
    Result 0
    {
      Property PassFail = "Pass";
      IncrementCounters PassCount;
      GoTo FlowTest1_Typ;
    }

    Result 1
    {
      Property PassFail = "Fail";
      IncrementCounters FailCount;
      Return 1;
    }

    # This result block will be executed if
    # MyFunctionalTest1Min returns any of
    # 2, 5, 6, 7, -20, -19, -18
    Result 2, 5:7, -20:-18
    {
      Property PassFail = "Fail";
      IncrementCounters FailCount;
      Return 1;
    }
  }

  FlowItem FlowTest1_Typ { ... }
  FlowItem FlowTest1_Max { ... }
}
```

Stylus: Sequence of boolean expressions select exit port. First boolean expression satisfied determines port (even if subsequent expressions are also true). This implies an order dependency to the list of exit expressions – and it is the user’s responsibility to insure that only one expression is true, or that, if more than one is true, the first one true is the desired one. Can handle complex expressions. The boolean expression is normally based on the test result ('Result==Fail'), but can be any legal boolean expression.

```
ProgSeq "Main" {
  Segment 11 {
    Type TEST;
    Test ContactTest;
    Action FailOpens { 'Result==Fail'; Bin=11; }
    Action FailShorts { 'Result==Fail'; Bin=12; }
    Action { 'Result==Pass'; Next=41; }
  }
  Segment 41 {
    Test FunctionalTests;
    Action { 'Result==Fail'; Bin=41; }
    Action { 'Result==Pass'; Next=15; }
  }
}
```

Envision: Each test contains a sequence of boolean expressions. The test object interpreter evaluates each expression starting at port 0 and stepping through the ports sequentially until it finds a port that returns a TRUE condition. When this port is located, the port number is returned to the Flow object interpreter where the Flow port table can be indexed (using the returned port number as specified by the **PortSelect** statement) to determine the next Flow object (FlowNode) to be executed.

```

Test Continuity {
    Result = Expr { String = "#"; Mode = Output; }
    PortExpr[0] = Expr { String = "Continuity.Result = tm_rslt:PASS"; }
    PortExpr[1] = Expr { String = "Continuity.Test_result.Shortcs = tm_rslt:FAIL"; }
    PortExpr[2] = Expr { String = "Continuity.Test_result.Open_to_Ground =
tm_rslt:FAIL"; }
    PortExpr[3] = Expr { String = "Continuity.Test_result.Open_to_Power =
tm_rslt:FAIL"; }
    Title[0] = Shorts;
    Title[1] = Open_to_Ground;
    Title[2] = Open_to_Power;
    TestMethod = OSpins;
    Test_result[0] = Expr { String = "#"; Mode = Output; }
    Test_result[1] = Expr { String = "#"; Mode = Output; }
    Test_result[2] = Expr { String = "#"; Mode = Output; }
}

SubFlow MainFlow_OnStart {
    Node[4] = FlowNode_ MainFlow_OnStart_0_4 {
        XCoord = (13,25);
        Port[0] {
            To = 5;
            UIFFPort = 113;
        }
        UIFInfo = 0;
        TestId = "17";
        PortSelect = "0";
    }
    Node[5] = FlowNode_ MainFlow_OnStart_0_5 {
        Port[0] {
            To = 6;
        }
        Port[1] {
            To = 14;
        }
        Port[2] {
            To = 15;
        }
        Port[3] {
            To = 16;
        }
        PortSelect = "Continuity.ExecResult";
        Exec = Continuity;
    }
    Node[6] = FlowNode_ MainFlow_OnStart_0_6 {
        Port[0] {
            To = 17;
        }
        Port[1] {
            To = 13;
        }
        PortSelect = "Input_Leakage.ExecResult";
        Exec = Input_Leakage;
    }
    Node[14] = FlowNode_ MainFlow_OnStart_0_14 {
        PortSelect = "FAIL_Shortcs.ExecResult";
    }
}

```

```

        Exec = FAIL_Shorts;
    }
    Node[15] =      FlowNode_ MainFlow_OnStart_0_15 {
        PortSelect = "FAIL_Open_to_Gnd.ExecResult";
        Exec = FAIL_Open_to_Gnd;
    }
    Node[16] =      FlowNode_ MainFlow_OnStart_0_16 {
        PortSelect = "FAIL_Open_to_Pwr.ExecResult";
        Exec = FAIL_Open_to_Pwr;
    }
}

```

ASAP: Test returns an integer value. This value selects one of N exit ports in the RETURN { } subblock. The port numbers are implied by the ordinal sequence of the port statements (starting at 0) in the RETURN { } subblock. Each port has a type attribute (PASS, FAIL, or OTHERWISE). Only 1 fail port is allowed, and it MUST be port 0. However, it is possible to have no fail ports – in that case, all ports MUST be of type PASS. It is also to have OTHERWISE ports. Only the first or last port can be OTHERWISE. If the first port is OTHERWISE, any remaining ports are of type PASS. If the last port is otherwise, then the first port (port 0) is FAIL.

```

segment DC_1 {
    <08:17:1993 09:50:37>,
    X_POS = 250,
    Y_POS = 62,
    ENTRY = W52,
    ICON = "dcseg_c",
    TOOL = "dctool",
    TEST = vdd_opens_cont,
    RETURN = {
        { POS = S38, FAIL, End_1, BINS = fo_signal_opens },          PORT 0
        { POS = E41, PASS, DC_18 }                                  PORT 1
    }
}; /* end of SEGMENT DC_1 */

```

SmarTest: Only one or two exit ports allowed (pass, or pass and fail). Test result (a boolean flag with values PASS or FAIL) selects exit port. Generally, the “else” statement (which correlates to a failing test) is the branch that contains the bin statement, which terminates test execution. If the first part of the expression (run_and_branch(funcnt_first)) is true, then execution continues at the next flow node.

```

test_flow
    run_and_branch(funcnt_first)      then
    {
    }
    else
    {
        stop_bin "YZ", "fp_func1_fail", , bad,noreprobe,red, 7, not_over_on;
    }
    run_and_branch(funcnt_second)      then
    {
    }
    else
    {
        stop_bin "YY", "fp_func2_fail", , bad,noreprobe,red, 7, not_over_on;
    }
    run(simulate_test_time__2__); // No branching, always passes.
    stop_bin "AA", "p_good_part", , good,noreprobe,green, 1, not_over_on;
end

```

Credence XTOS: XTOS uses the return value from a test or flow (a quoted or unquoted string) in a sequence of boolean expressions to determine the exit port taken. The boolean expressions take the form `<GoTo result="FAIL">` (which really means (GoTo if result == "Fail"). The expressions are evaluated in the order in which they occur in the file, and the first expression satisfied determines the path taken. The Decision can be Pass, Fail, or Otherwise. If the boolean expression following the GoTo is missing, then when that GoTo block is reached in the evaluation process, that path is unconditionally taken.

```

<Flow name="Init">
  <Node name="START">
    <Type>Entry</Type>
    <GoTo result="PASS">
      <NodeRef>TPValidateNode</NodeRef>
      <Decision>Pass</Decision>
    </GoTo>
  </Node>
  <Node name="TPValidateNode">
    <Type>Test</Type>
    <Icon>FTest.gif</Icon>
    <TestRef>TestProgramCheck</TestRef>
    <GoTo result="FAIL">
      <NodeRef>BadExit</NodeRef>
      <Decision>Fail</Decision>
    </GoTo>
    <GoTo result="PASS">
      <NodeRef>GoodExit</NodeRef>
      <Decision>Pass</Decision>
    </GoTo>
  </Node>
  <Node name="GoodExit">
    <Type>Exit</Type>
    <Decision>Pass</Decision>
  </Node>
  <Node name="BadExit">
    <Type>Exit</Type>
    <Decision>Fail</Decision>
  </Node>
</Flow>

```

Credence D10: In the Diamond (D10) syntax an exit port is a string. It can contain any characters including punctuation and whitespace. If the exit port contains only or lower case letters, numbers, or underscore ([a-zA-Z0-9_]), it can be quoted or unquoted. If the exit port contains any characters other than upper or lower case letters, numbers, or underscore ([a-zA-Z0-9_]) then the string **MUST** be quoted. The reason for using strings is an integer can be converted to a string so using a string as the port id covers both cases (integer return value and arbitrary string return value).

The test method description declares the exit port names. The test method function uses an API function to set an exit port for each active site based on the test result. The test flow processes each site after the test taking the action specified in the exit port in the flow for each site.

```

TestFlow myflow
{
  Parameters {
    key String;
  };

  FlowNode Opens {

```

```

    PreActions {
        testnum=100;
        testname="Opens";
    }
    TestObjectExec OpensTest;
    ExitPorts {
        Pass { Next "Shorts"; }
        Fail { Bin bin8; Stop; }
    }
}
FlowNode Shorts {
    PreActions {
        testnum=200;
        // testname defaults to FlowNode name
    }
    TestObjectExec ShortsTest;
    ExitPorts {
        Pass { Next "sort or class"; }
        Fail { Bin bin8; Stop; }
    }
}
FlowNode "sort or class" {
    PreActions {
        if key="sort" GoTo sort;
        if key="class" GoTo class;
    }
    ExitPorts {
        sort { Next "Substrate Bias"; }
        class { Next "Speed Tests"; }
    }
}
FlowNode "Gross Functional Tests" {
    TestFlowExec "Gross Functional Flow";
    ExitPorts {
        Pass { Bin bin1; Next "Input Leakage"; }
        Fail { Bin bin5; Stop; }
    }
}
Flow "Gross Functional Flow"
{
    FlowNode "10mhz Pattern 1" {
        PreActions { testnum=500; }
        TestObjectExec 10Mhz_Pattern_1;
        ExitPorts {
            Pass { Next "10mhz Pattern 2"; }
            Fail { Return Fail; }
        }
    }
    FlowNode "10mhz Pattern 2" {
        PreActions { testnum=501; }
        TestObjectExec ="10Mhz_Pattern_2";
        ExitPorts {
            Pass { Next "10mhz Pattern 3"; }
            Fail { Return Fail; }
        }
    }
    FlowNode "Input Leakage" {
        PreActions { testnum=700; }
        TestObjectExec InputLeakage;
        ExitPorts {
            Pass { Next "Output Leakage"; }
            Fail { Bin bin7; Stop }
        }
    }
}

```

```

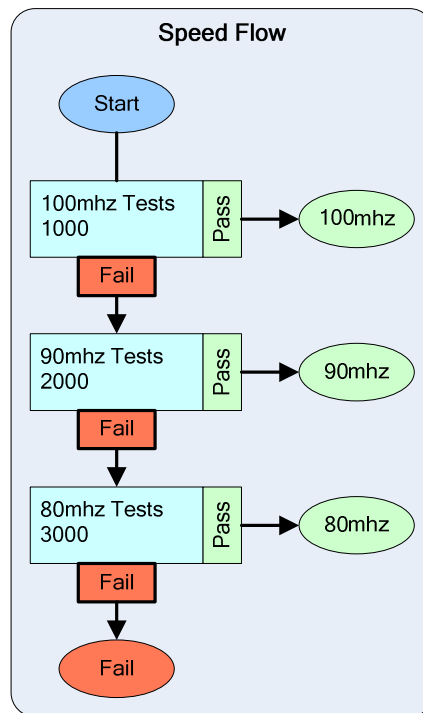
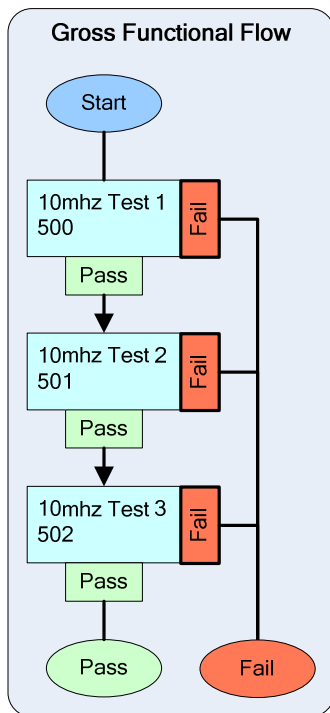
}
}

Flow "Speed Flow"
{
  FlowNode "100mhz Tests" {
    PreActions { testnum=1000; }
    TestObjectExec "100mhz_Tests";
    ExitPorts {
      Pass { Return 100mhz; }
      Fail { Next "90mhz Tests"; }
    }
  }

  FlowNode "90mhz Tests" {
    PreActions { testnum=2000; }
    TestObjectExec "90mhz_Tests";
    ExitPorts {
      Pass { Return 90mhz; }
      Fail { Next "80mhz Tests"; }
    }
  }

  FlowNode "80mhz Tests" {
    PreActions { testnum=3000; }
    TestObjectExec "80mhz_Tests";
    ExitPorts {
      Pass { Return 80mhz; }
      Fail { Return Fail; }
    }
  }
}
}

```



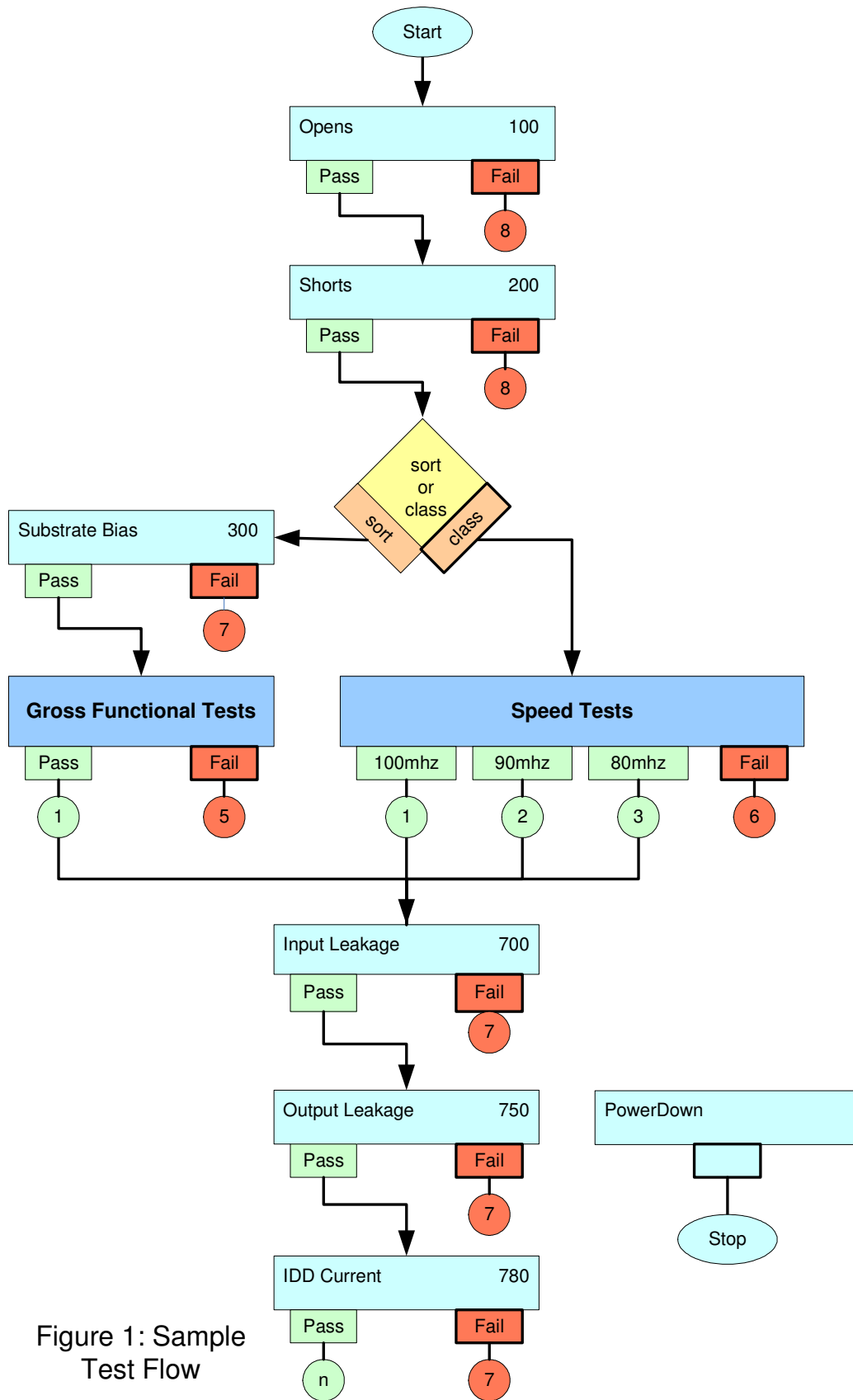


Figure 1: Sample Test Flow