Acceptance Tests
for the
Washington University Gigabit Switch

Presented by:

John DeHart
jdd@arl.wustl.edu
http://www.arl.wustl.edu/~jdd
http://www.arl.wustl.edu/arl

Applied Research Laboratory

WUGS Kits Program
Washington University
July 13-24, 1998
August 3-14, 1998

WUGS Software Overview

---

Interactive Script Files

Jammer

NCCP
TCP

NCCP
TCP

NCCP
TCP
PRIQ

NCMO API

TCP
PRIQ

Session Management
Connection Management

GBNSC

ATMCard
TCP
NATM

ENI

Cells in Packets

TCP
ATMCard

NodeSim

Cfg File

WUGS

Raw ATM Cell Interface to Switch

Control Port
Acceptance Tests: Purpose

• Test as much of the switch as possible

• Give us confidence that the switch is truly operational

• Provide a standard procedure for detecting known problems

With our previous switch, these tests became quite extensive as we found problems with a particular switch and added tests to look for that problem in later assembled switches. This has not been the case with this switch. This suite of tests mainly tests basic functionality. Our experience with this set of switches has been that if there is something wrong it is obvious and easy to detect with simple tests.

Acceptance Tests: Component Tests

• ping ports: use pings (test cell[012]) to test paths to/from each combination of ports using each copy, both copies and range copy.

• TGI/UD: Test upstream discard and trunk group identifier.

• BI: Test busy/idle bit

• VCXT: test that we can write all the bits in each VCXT entry in the switch.

• VCXT Integrity: test all the VCXT entries by writing them all and then going back to test that they have not changed.

• Data Integrity: test data cells through each port.
int return_value
int number_of_errors
int CONTROL_PORT
int PORTS[8]
CONTROL_PORT = 2

# Mark which ports are present. Except for Control port
# each port present should also be looped back with
# an external fiber loopback.
# The control Port will be connected to the control processor
# but it should still be marked as present with a 1 below.
PORTS[0] = 1
PORTS[1] = 1
PORTS[2] = 1
PORTS[3] = 1
PORTS[4] = 1
PORTS[5] = 1
PORTS[6] = 1
PORTS[7] = 1
return_value = 0
number_of_errors = 0

set print quiet
include do_ping_ports.js
set print quiet

if ($return_value != 0)
    echo "ping tests failed, no need to continue..."
    quit
fi
include do_tgi_ud_test.js
set print quiet
include do_bi_test.js
set print quiet
include do_vcxt_test.js
set print quiet
include do_vcxt_integrity_test.js
set print quiet
include do_data_test.js

echo "DONE with all tests: $number_of_errors Errors Reported"
quit
Data Test Example (data_test.js)

```javascript
set print quiet

proc loadswitch(int inputPort, int inputVCI, int outputPort, int outputVCI, int startPort, int endPort, int startVCI, int endVCI, int cs_1, int cyc1, int outCyc1)

set print quiet
int port
int nextPort
int tmpVCI
int tempPort
int foundit

tmpVCI = $startVCI

write mr $inputPort 2 0 128 32 0 255 1 1 1 1 100000 0
write vpxt $inputPort 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
write vpxt $inputPort 128 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
write mr $outputPort 2 0 128 32 0 255 1 1 1 1 100000 0
write vpxt $outputPort 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

write vcxt $inputPort $inputVCI 0 2 1 $cyc1 0 $cs_1 0000000 $startVCI 0000 $startPort 0
while ($port <= $endPort)
if ($PORTS[$port] == 1)
    #echo "Doing port = $port"
    nextport = $port+1
if ($port != $endPort)
    if ($PORTS[$nextport] == 0)
        tempPort = $nextport
        foundit = 0
        while ($tempPort <= 7 )
            if ($PORTS[$tempPort] == 1)
                if ($foundit == 0)
                    foundit = 1
                nextport = $tempPort
            fi
        fi
        tempPort++
```

Data Test (con’t)

```javascript
port = $startPort

# C C V V B V B A A
# Y Y U U V R P C D D D
# B R C C C D D S P C B I I II II I R R
# I C D 12 S12 U R 1 1 2 2 1 1 2
write vcxt $inputPort $inputVCI 0 2 1 $cyc1 0 $cs_1 0 0 0 0 0 0 0 0 $startVCI 0 0 0 0 0 0 $startPort 0

while ($port <= $endPort)
    if ($SPORTS[$port] == 1)
        #echo "Doing port = $port"
        nextport = $port+1
    if ($port != $endPort)
        if ($SPORTS[$nextPort] == 0)
            tempPort = $nextPort
            foundit = 0
            while ($tempPort <= 7 )
                if ($SPORTS[$tempPort] == 1)
                    if ($foundit == 0)
                        foundit = 1
                    nextPort = $tempPort
                fi
            fi
            tempPort++
```
Data Test (con’t)

done
fi
fi
#echo "nextport = $nextport"
write mr $port 2 0 128 32 0 255 1 1 1 1 100000 0
write vpxt $port 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0

tmpVCI = $startVCI
while ($tmpVCI < $endVCI)
  # CC V V BVVB AA
  # Y Y U U V R P C D P C D D D
  # BR C C C D D S P C B I I I I I R R
  # I C D 1 2 S 1 2 C T O R 1 1 1 2 2 2 1 2
  write vcxt $port $tmpVCI 1 2 1 $cyc1 0 $cs_1 0000000 ($tmpVCI+1) 0 0 0 0 $port 0
  tmpVCI++
  done

if ($port != $endPort)
  # CC V V BVVB A A
  # Y Y U U V R P C D P C D D D
  # BR C C C D D S P C B I I I I I R R
  # I C D 1 2 S 1 2 C T O R 1 1 1 2 2 2 1 2
  write vcxt $port $endVCI 1 2 1 $cyc1 0 $cs_1 0000000 $startVCI 0000 $nextport 0
fi

port = $nextport
else
  #echo "Skipping port = $port"
  port++
fi
wait
done

#now out
# CC V V BVVB A A
# Y Y U U V R P C D P C D D D
# BR C C C D D S P C B I I I I I R R
# I C D 1 2 S 1 2 C T O R 1 1 1 2 2 2 1 2
write vcxt SendPort $endVCI 1 2 1 $outCyc1 0 $cs_1 0 0 0 0 0 0 $outputVCI 0 0 0 0 $outputPort 0
write vcxt $inputPort $inputVCI 1 2 1 $cyc1 0 $cs_1 0000000 $startVCI 0000 $startPort 0

wait
#set print normal
end
### Data Test (con’t)

**proc putload**

```plaintext
proc putload(int inputPort, int inputVCI, int numCells, int byte0, int byte1, int byte2, int byte3, int byte4, int byte5, int byte6, int byte7)
    set print quiet
    put cells 0 $inputVCI $numCells 6 0 $byte0, $byte1, $byte2, $byte3, $byte4, $byte5, $byte6, $byte7
end
```

**proc getload**

```plaintext
proc getload(int goQuiet, int lastPort, int outputPort, int lastVCI, int outputVCI, int timeout, int numCells, int byte0, int byte1, int byte2, int byte3, int byte4, int byte5, int byte6, int byte7)
    set print quiet
    clear failed
    get cells 0 $outputVCI $numCells $timeout 6 0 $byte0, $byte1, $byte2, $byte3, $byte4, $byte5, $byte6, $byte7
    if ($goQuiet == 1)
        set print quiet
    else
        set print normal
    fi
    # CC V V BVVB A A
    # Y YU UV RP C D P C DD D
    # B R C C C D D S P C B I I IIII R R
    # ------------------------------------------------------------------------------------------------------
    write vcxt $lastPort $lastVCI 1210010000000 $outputVCI 0000 $outputPort 0
    set print quiet
    if (failed)
        echo "get cells failed with bytes: $byte0, $byte1, $byte2, $byte3, $byte4, $byte5, $byte6, $byte7"
    fi
end
```
proc data_test()
  set print quiet
  int patternBytes[8]
  int patternByte
  int pattern1
  int pattern2

  patternByte = 0
  patternBytes[0] = 0
  patternBytes[1] = 0
  patternBytes[2] = 0
  patternBytes[3] = 0
  patternBytes[4] = 0
  patternBytes[5] = 0
  patternBytes[6] = 0
  patternBytes[7] = 0

  while ($patternByte <= 0xff)
    patternBytes[0] = $patternByte
    patternBytes[1] = $patternByte
    patternBytes[2] = $patternByte
    patternBytes[3] = $patternByte
    patternBytes[4] = $patternByte
    patternBytes[5] = $patternByte
    patternBytes[6] = $patternByte
    patternBytes[7] = $patternByte

    loadswitch($CONTROL_PORT,33, $CONTROL_PORT,33, 0,7, 150, 150, 1, 1, 1)
    putload($CONTROL_PORT, 33, 20, $patternBytes[0], $patternBytes[1], $patternBytes[2], $patternBytes[3], $patternBytes[4], $patternBytes[5], $patternBytes[6], $patternBytes[7])
    getload(1, 7,$CONTROL_PORT,150,33, 5, 20, $patternBytes[0], $patternBytes[1], $patternBytes[2], $patternBytes[3], $patternBytes[4], $patternBytes[5], $patternBytes[6], $patternBytes[7])
    if (failed)
      patternByte = 0x100
    fi
    patternByte++
  done
if (failed)
    echo "Something failed with all bytes = $patternBytes[0]"
    clear failed
    return_value = 1
    number_of_errors++
else
    echo "All uniform payload data tests passed"
fi

# this set will test a rank of 0x00 followed by a rank of 0xFF
#                a rank of 0x11 followed by a rank of 0xEE
#                a rank of 0x22 followed by a rank of 0xDD
# and so on...

clear failed
patternByte = 0x00

while ($patternByte <= 0xff)
    #
    # loadswitch($CONTROL_PORT,33, $CONTROL_PORT,33, 0,7, 150, 150, 1, 1, 1)
    #
    # putload($CONTROL_PORT, 33, 20, $patternBytes[0], $patternBytes[1], $patternBytes[2], $patternBytes[3], $patternBytes[4], $patternBytes[5], $patternBytes[6], $patternBytes[7])
    #
    # getload(1, 7,$CONTROL_PORT,150,33, 5, 20, $patternBytes[0], $patternBytes[1], $patternBytes[2], $patternBytes[3], $patternBytes[4], $patternBytes[5], $patternBytes[6], $patternBytes[7])
    #
    if (failed)
        patternByte = 0x100
    fi
    patternByte = $patternByte + 0x11
done
Data Test (con’t)

if (failed)
    echo "Something failed with ranks of $patternBytes[0] and ranks of $patternBytes[4]"
    clear failed
    return_value = 1
    number_of_errors++
else
    echo "All alternating ranks data tests passed"
fi

set print normal
end
set print normal