SFU SIMON FRASER UNIVERSITY

**Department of Engineering Science**
Faculty of Applied Science

| | |
|---|---|
| Course Code: | ENSC 252: 1207 |
| Course Title: | Fundamentals of Digital Logic and Design |
| Semester Code: | Fall 2020 |

| | |
|---|---|
| Instructor: | Anita Tino |

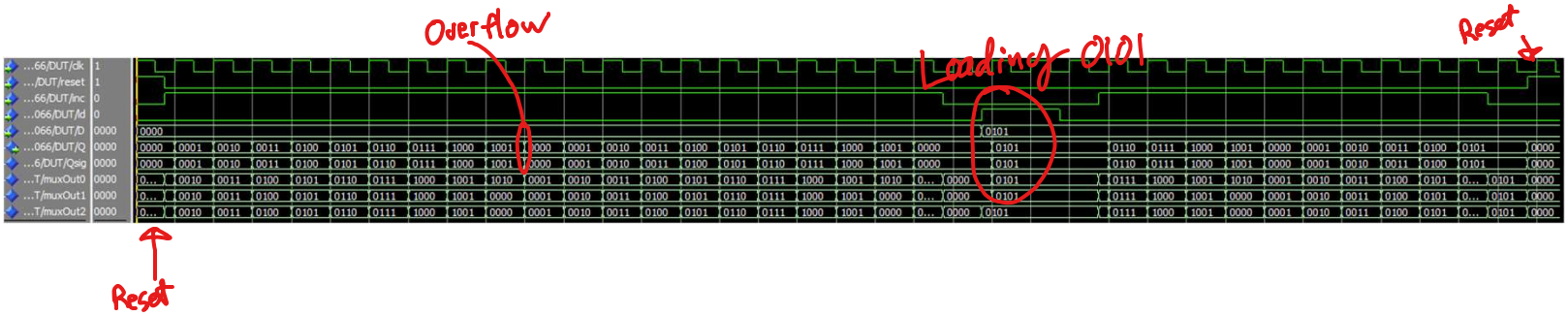
| | |
|---|---|
| Asst/Lab No. : | 4 |
| Asst/Lab Title: | Take-A-Number System |

| | |
|---|---|
| Submission Date: | Nov 8 2020 |

| | |
|---|---|
| Student Name: | Devon Burnham |
| SFU ID: | 301394066 |
| Signature*: | (Devon Burnham) |

***By signing above, I certify that the contribution made to this lab/assignment is solely mine and confirm that all work completed is my own. Any suspicion of plagiarism and/or copying will result in an investigation in the ENSC department. A mark of zero or F may be assigned, and depending on the level of severity, an FD grade on my transcript and/or expulsion from the school, according SFU's Code of Academic Integrity found online at** http://www.sfu.ca/policies/gazette/student/s10-01.html

Waveforms for reg_4066




A single register doesn't have that many input combinations, so the complete test can be fit into a single screenshot. The initial reset initializes everything to 0, then on every rising edge of the clock, Q is incremented by one until it reaches 9, at which point it overflows to 0. This process is repeated to ensure no weirdness occurs. At that point, the value 0101 is loaded into the register to test load functionality, then more incremental testing until the final segment, testing the reset once again.

VHDL code for reg_4066

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY reg_4066 IS
    GENERIC (
        data_width : INTEGER := 4;
        N : INTEGER := 9);
    PORT (
        clk, reset, inc, ld : IN STD_LOGIC;
        D : IN UNSIGNED(data_width - 1 DOWNTO 0);
        Q : OUT UNSIGNED(data_width - 1 DOWNTO 0));
END reg_4066;

ARCHITECTURE behaviour OF reg_4066 IS

    --signals
    SIGNAL Qsig : unsigned(data_width - 1 DOWNTO 0);
    SIGNAL muxOut0 : unsigned(data_width - 1 DOWNTO 0);
    SIGNAL muxOut1 : unsigned(data_width - 1 DOWNTO 0);
    SIGNAL muxOut2 : unsigned(data_width - 1 DOWNTO 0);

BEGIN
    --first muxer
    muxOut0 <= (Qsig + to_unsigned(1, data_width)) WHEN (inc = '1') ELSE
        Qsig;

    --second muxer, equality checker, and and gate
    muxOut1 <= to_unsigned(0, data_width) WHEN (to_unsigned(N, data_width) = Qsig AND inc = '1')
ELSE
        muxOut0;

    --third muxer
    muxOut2 <= (D) WHEN (ld = '1') ELSE
        muxOut1;

    --flipflops
    PROCESS (reset, clk) IS
    BEGIN
        IF (reset = '1') THEN
            Qsig <= to_unsigned(0, data_width);
        ELSIF (rising_edge(clk)) THEN
            Qsig <= muxOut2;
        ELSE
            Qsig <= Qsig;
        END IF;
    END PROCESS;

    Q <= Qsig;
END behaviour;
```

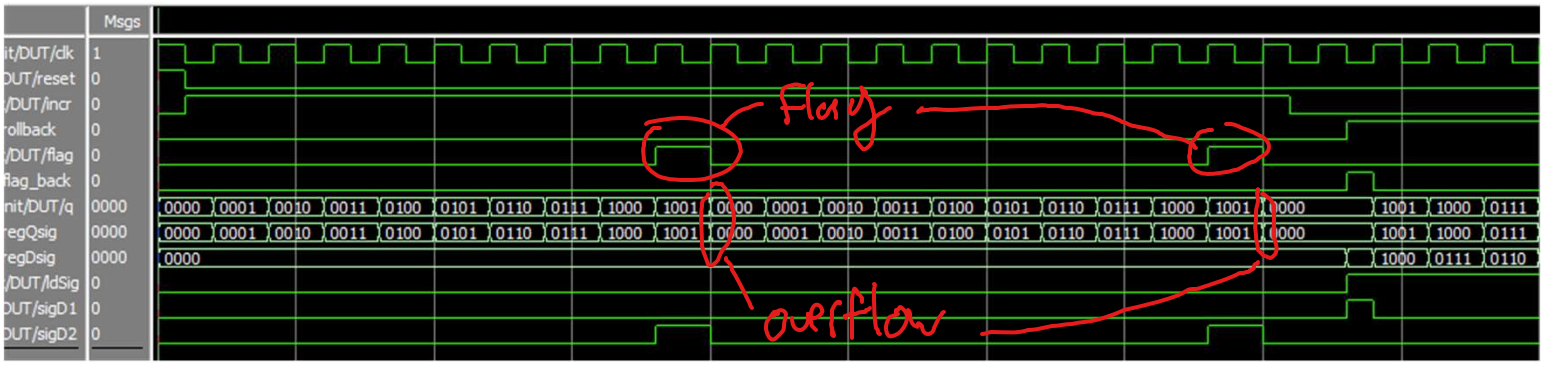
Testbench for reg_4066

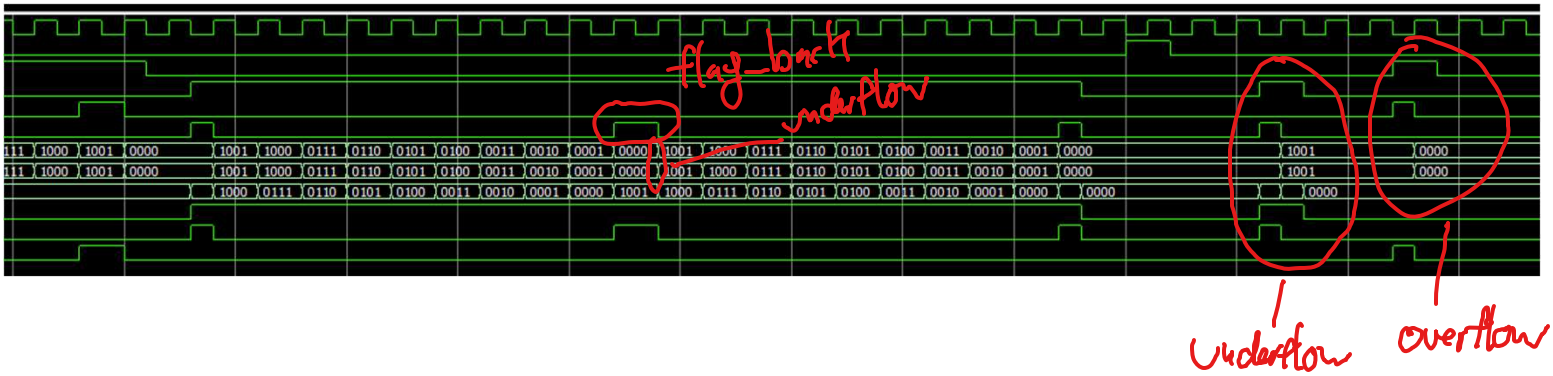
Whitespace between lines removed to fit on one page

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
--testbenches have empty entity sections
ENTITY tb_reg_4066 IS
END tb_reg_4066;
ARCHITECTURE test OF tb_reg_4066 IS
    COMPONENT reg_4066 IS
        GENERIC (
            data_width : INTEGER := 4;
            N : INTEGER := 9);
        PORT (
            clk, reset, inc, ld : IN STD_LOGIC;
            D : IN UNSIGNED(data_width - 1 DOWNTO 0);
            Q : OUT UNSIGNED(data_width - 1 DOWNTO 0));
    END COMPONENT;
    --signals and constants
    CONSTANT data_width : INTEGER := 4;
    CONSTANT N : INTEGER := 9;
    CONSTANT HALF_PERIOD : TIME := 10 ns;
    CONSTANT PERIOD : TIME := 20 ns;
    SIGNAL sigQ, sigD : unsigned(data_width - 1 DOWNTO 0);
    SIGNAL sigclk : STD_LOGIC := '1';
    SIGNAL sigreset, siginc, sigld : STD_LOGIC;
BEGIN
    DUT : reg_4066 GENERIC MAP(4, 9)
    PORT MAP(sigclk, sigreset, siginc, sigld, sigD, sigQ);
    --to cycle clk for the duration of the test
    sigclk <= NOT sigclk AFTER HALF_PERIOD;
    PROCESS IS
    BEGIN
        --reset
        sigreset <= '1';
        siginc <= '0';
        sigld <= '0';
        sigD <= to_unsigned(0, data_width);
        WAIT FOR HALF_PERIOD;
        sigreset <= '0';

        --test incrementing all the way up and overflowing, twice
        siginc <= '1';
        WAIT FOR 20 * PERIOD;
        siginc <= '0';
        WAIT FOR PERIOD;

        --test loading
        sigld <= '1';
        sigD <= to_unsigned(5, data_width);
        WAIT FOR 2 * PERIOD;
        sigld <= '0';
        WAIT FOR PERIOD;

        --test incrementing all the way from a loaded starting place
        siginc <= '1';
        WAIT FOR 10 * PERIOD;
        siginc <= '0';
        WAIT FOR PERIOD;

        --test resetting again, with set values
        sigreset <= '1';
        WAIT FOR PERIOD;
        WAIT;
    END PROCESS;
END test;
```

Waveforms for the increment control unit

These waveforms were too long to fit into a single screenshot, but these two screenshots overlap in the middle to cover the testbench. It increments to 9, then overflow twice, throwing flag = 1 each time it is at 9 with intention to increment.




Next, we decrement down from zero, underflowing to 9, and throwing flag_back whenever we are at 0 with desire to rollback. The value of 0 is held briefly, a superfluous reset is called, then single decrements and increments are tested, with desired results and flags.

VHDL code for the increment control unit

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY increment_control_unit IS
    GENERIC ( N : INTEGER := 9; data_width : INTEGER := 4);
    PORT ( clk, reset : IN STD_LOGIC;
        incr, rollback : IN STD_LOGIC; --generates next number, rollback decrements
        flag, flag_back : OUT STD_LOGIC;
        q : OUT UNSIGNED(data_width - 1 DOWNTO 0)); --output value
END increment_control_unit;

ARCHITECTURE structure OF increment_control_unit IS

    COMPONENT reg_4066 IS
        GENERIC ( data_width : INTEGER := 4; N : INTEGER := 9);
        PORT ( clk, reset, inc, ld : IN STD_LOGIC;
            D : IN UNSIGNED(data_width - 1 DOWNTO 0);
            Q : OUT UNSIGNED(data_width - 1 DOWNTO 0));
    END COMPONENT;

    SIGNAL regQsig : unsigned(data_width - 1 DOWNTO 0);
    SIGNAL regDsig : unsigned(data_width - 1 DOWNTO 0);
    SIGNAL ldSig : STD_LOGIC;
    SIGNAL sigD1 : STD_LOGIC;
    SIGNAL sigD2 : STD_LOGIC;
    SIGNAL sigFlag : STD_LOGIC;
    SIGNAL sigFlagBack : STD_LOGIC;

BEGIN

    regObj : reg_4066 GENERIC MAP(data_width, N)
    PORT MAP(clk => clk, reset => reset, inc => incr, ld => ldSig, D => regDsig, Q => regQsig);

    --IFL
    ldSig <= (rollback AND (NOT incr));


    sigD1 <= '1' when ((rollback = '1') AND (incr = '0') and (regQsig = to_unsigned(0, data_width)))
else '0';
    sigD2 <= '1' when ((rollback = '0') AND (incr = '1') AND (N = (regQsig))) else '0';

    PROCESS (rollback, incr, regQsig) IS --only needs clk because nothing that it points to is async
    BEGIN
        IF ((rollback = '1') AND (incr = '0')) THEN
            IF (regQsig = to_unsigned(0, data_width)) THEN
                regDsig <= to_unsigned(N, data_width);
            ELSE
                regDsig <= (regQsig - 1);
            END IF;
        ELSE
            regDsig <= to_unsigned(0, data_width);
        END IF;
    END PROCESS;


    q <= regQsig;
    flag_back <= sigD1;
    flag <= sigD2;
END structure;
```

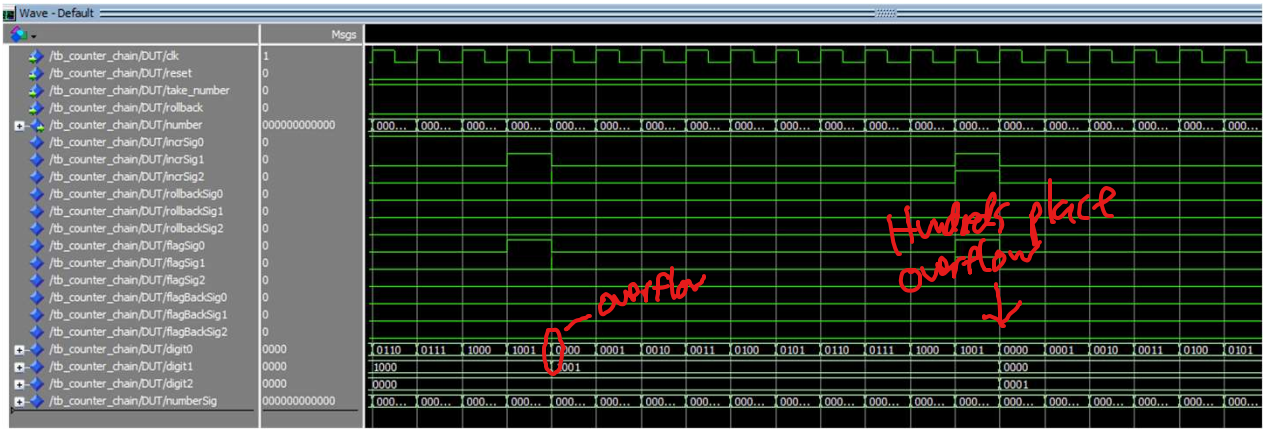
Testbench code for the increment control unit

Whitespace between lines removed to fit on one page

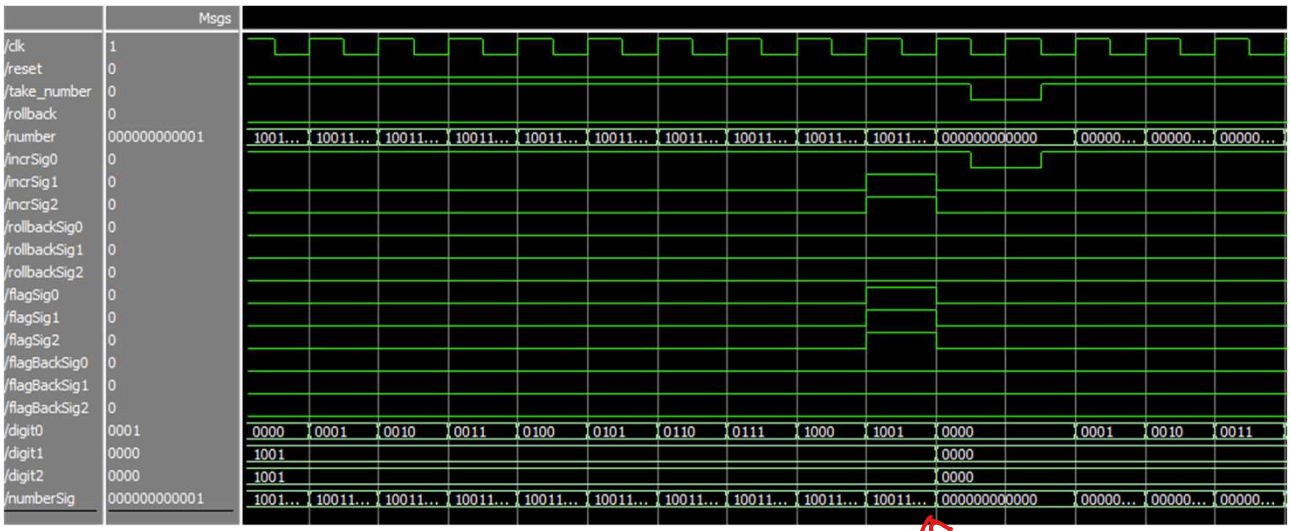
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
--testbenches have empty entity sections
ENTITY tb_increment_control_unit IS
END tb_increment_control_unit;
ARCHITECTURE test OF tb_increment_control_unit IS
    COMPONENT increment_control_unit IS
        GENERIC ( N : INTEGER := 9; data_width : INTEGER := 4);
        PORT ( clk, reset : IN STD_LOGIC;
            incr, rollback : IN STD_LOGIC; --generates next number, rollback decrements
            flag, flag_back : OUT STD_LOGIC;
            q : OUT UNSIGNED(data_width - 1 DOWNTO 0)); --output value
    END COMPONENT;
    --signals and constants
    CONSTANT data_width : INTEGER := 4;
    CONSTANT N : INTEGER := 9;
    CONSTANT HALF_PERIOD : TIME := 10 ns;
    CONSTANT PERIOD : TIME := 20 ns;
    SIGNAL sigQ : unsigned(data_width - 1 DOWNTO 0);
    SIGNAL sigclk : STD_LOGIC := '1';
    SIGNAL sigreset, sigincr, sigrollback, sigflag, sigflagback : STD_LOGIC;
BEGIN
    DUT : increment_control_unit GENERIC MAP(9, 4)
    PORT MAP(sigclk, sigreset, sigincr, sigrollback, sigflag, sigflagback, sigQ);
    --to cycle clk for the duration of the test
    sigclk <= NOT sigclk AFTER HALF_PERIOD;
    PROCESS IS
    BEGIN
        --reset
        sigreset <= '1';
        sigincr <= '0';
        sigrollback <= '0';
        WAIT FOR HALF_PERIOD;
        sigreset <= '0';
        --test incrementing all the way up and overflowing, twice
        sigincr <= '1';
        WAIT FOR 20 * PERIOD;
        sigincr <= '0';
        WAIT FOR PERIOD;
        --test decrementing all the way down and overflowing, twice
        sigrollback <= '1';
        WAIT FOR 20 * PERIOD;
        sigrollback <= '0';
        WAIT FOR PERIOD;
        --test resetting again
        sigreset <= '1';
        WAIT FOR PERIOD;
        sigreset <= '0';
        WAIT FOR 2*PERIOD;
        --test decrementing once and waiting
        sigrollback <= '1';
        WAIT FOR PERIOD;
        sigrollback <= '0';
        WAIT FOR 2 * PERIOD;
        --test incrementing once and waiting
        sigincr <= '1';
        WAIT FOR PERIOD;
        sigincr <= '0';
        WAIT FOR 2 * PERIOD;
        WAIT;
    END PROCESS;
END test;
```

Waveforms for the counter chain

Now that we have implemented three different registers in our circuit, the state diagram gets very complex, and showing the entire testbench on screen at once becomes unfeasible. As such, individual sections have been selected.

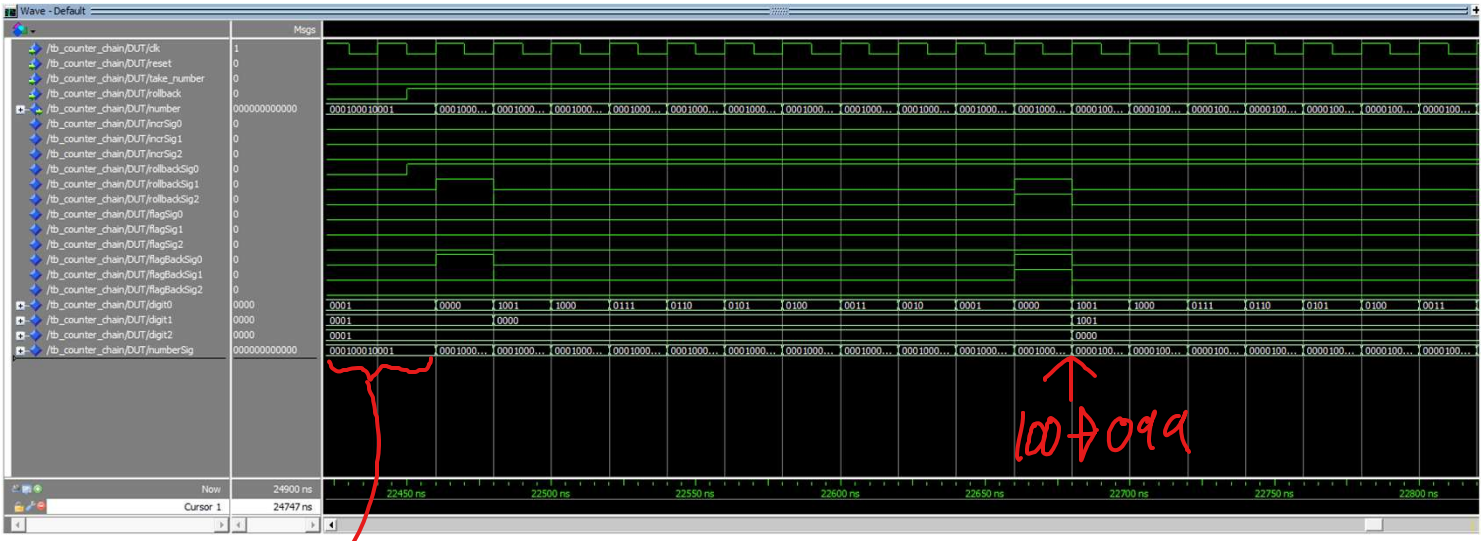



This first section demonstrates incrementing the ones place, as well as examples of flags being thrown and caught in order to increment the tens and hundreds place when lower places overflow.

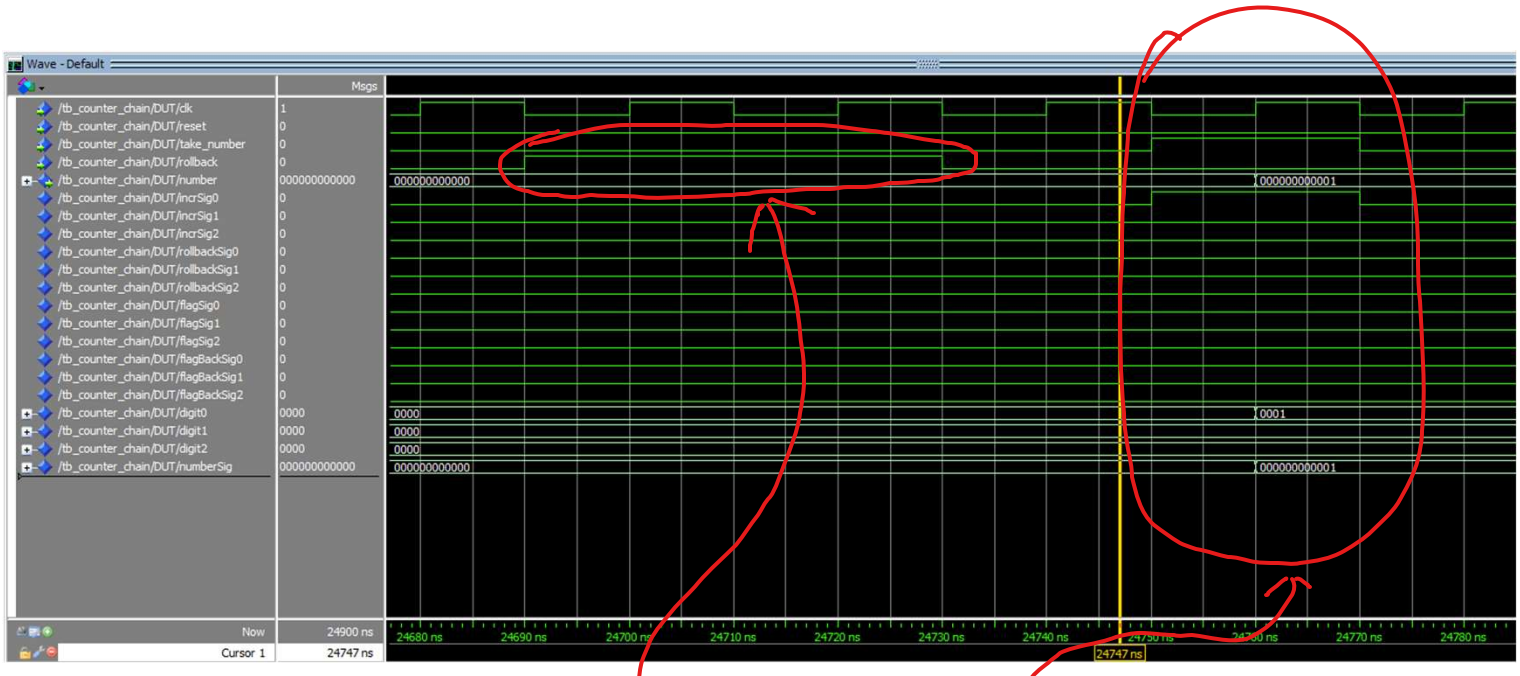



A demonstration of overflowing the hundreds place, and thus progressing smoothly from 999 to 000. It also then holds the value of 000 for a moment, and then begins incrementing back up again to prepare for decrementing tests.

Waveforms for the counter chain – part 2




Decrementing from 111, which allows me to demonstrate underflowing the hundreds place, the tens place, and the ones place in one tidy screenshot. Flags are thrown and caught accordingly.




Testing rollback when already at 0, and a single increment, both of which produce intended results.

# Written Report on Counter Chain

Objects are named from LSB to MSB, so ICU0 refers to the one's place and ICU2 refers to the tens place. IncrSig0 refers to the signal being input to the incr port on ICU0, and so on.

The one's digit should be incremented whenever take_number is true, of course, but it also has to increment when the hundreds place overflows, as that allows it to reset to 000 from 999. The tens digit is easier; increment whenever the ones digit overflows. The hundreds digit is just as simple.

Rollback functionality would be pretty straightforward, except for the detail that decrementing a counter that holds 000 must not rollback to 999, but instead remain at 000. If I were to design this system myself, I would have left the default behaviour in, allowing one to increment by a certain number, decrement by that same number, and remain at the original number. For instance, if the server incremented one too many times at 999, realized they made a mistake, and wanted to revert back to the previous position, they are unable to. However, mine is not to question the design parameters, so I implemented it to spec. The only way I could come up with to solve this was by checking the result for 000, which seems suboptimal; I'm sure there is a better way to do this.

That being said, the ones place decrements when rollback is true and the result isn't 000, the tens place decrements when the result isn't 000 and the ones place throws flag_back, meaning it underflowed to 9, and the hundreds place decrements when the tens place underflows, as long as the total result isn't 000.

VHDL code for the counter chain

Whitespace between lines removed to fit on one page

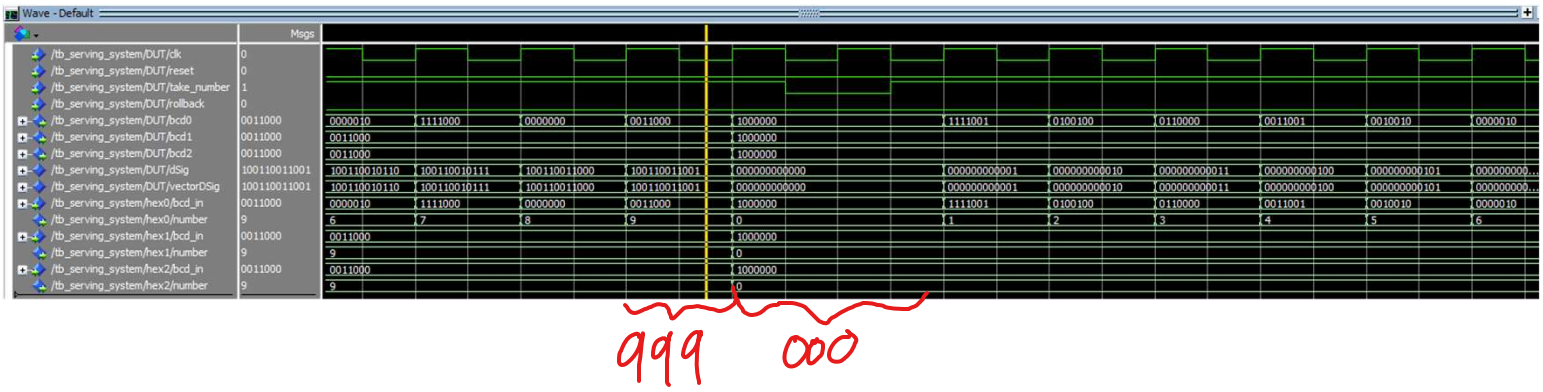
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
--testbenches have empty entity sections
ENTITY tb_counter_chain IS
END tb_counter_chain;
ARCHITECTURE test OF tb_counter_chain IS
COMPONENT counter_chain IS
GENERIC (
    radix : INTEGER := 9;
    data_width : INTEGER := 4);
PORT (
    clk, reset, take_number : IN STD_LOGIC;
    rollback : IN STD_LOGIC;
    number : OUT UNSIGNED((3 * data_width) - 1 DOWNTO 0));
END COMPONENT;
    --signals and constants
    CONSTANT data_width : INTEGER := 4;
    CONSTANT radix : INTEGER := 9;
    CONSTANT HALF_PERIOD : TIME := 10 ns;
    CONSTANT PERIOD : TIME := 20 ns;
    SIGNAL sigNumber : UNSIGNED((3 * data_width) - 1 DOWNTO 0);
    SIGNAL sigclk : STD_LOGIC := '1';
    SIGNAL sigreset, sigtake, sigrollback : STD_LOGIC;
BEGIN
    DUT : counter_chain GENERIC MAP(9, 4)
    PORT MAP(sigclk, sigreset, sigtake, sigrollback, signumber);
    --to cycle clk for the duration of the test
    sigclk <= NOT sigclk AFTER HALF_PERIOD;
    PROCESS IS
    BEGIN
        --reset
        sigreset <= '1';
        sigtake <= '0';
        sigrollback <= '0';
        WAIT FOR HALF_PERIOD;
        sigreset <= '0';
        --test incrementing all the way up and all possible overflowing
        sigtake <= '1';
        WAIT FOR 1000 * PERIOD;
        sigtake <= '0';
        WAIT FOR PERIOD;
        --test incrementing up to a number > 100 to prepare for decrementation
        sigtake <= '1';
        WAIT FOR 111 * PERIOD;
        sigtake <= '0';
        WAIT FOR 10*PERIOD;
        --test decrementing all the way down
        sigrollback <= '1';
        WAIT FOR 111 * PERIOD;
        sigrollback <= '0';
        WAIT FOR PERIOD;
        --test decrementing while at zero to ensure no overflow
        sigrollback <= '1';
        WAIT FOR 2 * PERIOD;
        sigrollback <= '0';
        WAIT FOR PERIOD;
        --test incrementing once at zero
        sigtake <= '1';
        WAIT FOR PERIOD;
        sigtake <= '0';
        WAIT FOR PERIOD;
        WAIT;
    END PROCESS;
END test;
```

Testbench code for the counter chain

Whitespace between lines removed to fit on one page

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
ENTITY counter_chain IS
    GENERIC ( radix : INTEGER := 9;
        data_width : INTEGER := 4);
    PORT ( clk, reset, take_number : IN STD_LOGIC;
        rollback : IN STD_LOGIC;
        number : OUT UNSIGNED((3 * data_width) - 1 DOWNTO 0));
END counter_chain;

ARCHITECTURE structure OF counter_chain IS
    --components
    COMPONENT increment_control_unit IS
        GENERIC (N : INTEGER := 9; data_width : INTEGER := 4);
        PORT ( clk, reset : IN STD_LOGIC;
            incr, rollback : IN STD_LOGIC; --generates next number, rollback decrements
            flag, flag_back : OUT STD_LOGIC;
            q : OUT UNSIGNED(data_width - 1 DOWNTO 0)); --output value
    END COMPONENT;

    --signals
    SIGNAL incrSig0 : STD_LOGIC;
    SIGNAL incrSig1 : STD_LOGIC;
    SIGNAL incrSig2 : STD_LOGIC;

    SIGNAL rollbackSig0 : STD_LOGIC;
    SIGNAL rollbackSig1 : STD_LOGIC;
    SIGNAL rollbackSig2 : STD_LOGIC;

    SIGNAL flagSig0 : STD_LOGIC;
    SIGNAL flagSig1 : STD_LOGIC;
    SIGNAL flagSig2 : STD_LOGIC;

    SIGNAL flagBackSig0 : STD_LOGIC;
    SIGNAL flagBackSig1 : STD_LOGIC;
    SIGNAL flagBackSig2 : STD_LOGIC;

    SIGNAL digit0 : unsigned(data_width - 1 DOWNTO 0);
    SIGNAL digit1 : unsigned(data_width - 1 DOWNTO 0);
    SIGNAL digit2 : unsigned(data_width - 1 DOWNTO 0);

    SIGNAL numberSig : UNSIGNED((3 * data_width) - 1 DOWNTO 0);
BEGIN
    numberSig <= (digit2 & digit1 & digit0); --concatenate, hundreds then tens then ones
    number <= numberSig;
    --instantiate components
    ICU0 : increment_control_unit GENERIC MAP(radix, data_width)
    PORT MAP( clk => clk, reset => reset, incr => incrSig0, rollback => rollbackSig0,
        flag => flagSig0, flag_back => flagBackSig0, q => digit0);
    ICU1 : increment_control_unit GENERIC MAP(radix, data_width)
    PORT MAP( clk => clk, reset => reset, incr => incrSig1, rollback => rollbackSig1,
        flag => flagSig1, flag_back => flagBackSig1, q => digit1);
    ICU2 : increment_control_unit GENERIC MAP(radix, data_width)
    PORT MAP( clk => clk, reset => reset, incr => incrSig2, rollback => rollbackSig2,
        flag => flagSig2, flag_back => flagBackSig2, q => digit2);

    incrSig0 <= (take_number or flagSig2);
    incrSig1 <= flagSig0;
    incrSig2 <= flagSig1;

    rollBackSig0 <= '1' when ((rollback = '1') and (numberSig /= 0)) else '0';
    rollBackSig1 <= '1' when ((flagBackSig0 = '1') and (numberSig /= 0)) else '0';
    rollBackSig2 <= '1' when ((flagBackSig1 = '1') and (numberSig /= 0)) else '0';
END structure;
```

Waveforms for the serving system




Minimal testing shown here, as the main logic has already been demonstrated in counter chain. The only difference here is mapping the number into seven segment displays, which has been accomplished here, as shown by the rollover from 999 to 000. Integer values are displayed correctly.

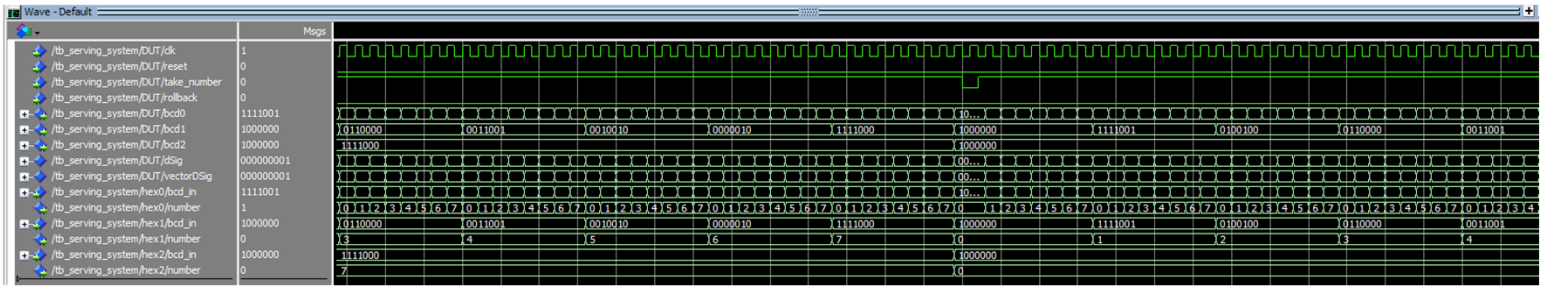
VHDL code for the serving system

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY serving_system IS
    GENERIC (
        radix : INTEGER := 9;
        data_width : INTEGER := 4);
    PORT (
        clk, reset, take_number : IN STD_LOGIC;
        rollback : IN STD_LOGIC;
        bcd0, bcd1, bcd2 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0));
END serving_system;

ARCHITECTURE structure OF serving_system IS

    --components
    COMPONENT SevenSeg IS
        PORT (
            D : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
            Y : OUT STD_LOGIC_VECTOR(6 DOWNTO 0));
    END COMPONENT;

    COMPONENT counter_chain IS
        GENERIC (
            radix : INTEGER := 9;
            data_width : INTEGER := 4);
        PORT (
            clk, reset, take_number : IN STD_LOGIC;
            rollback : IN STD_LOGIC;
            number : OUT UNSIGNED((3 * data_width) - 1 DOWNTO 0));
    END COMPONENT;

    --signals
    SIGNAL dSig : UNSIGNED((3 * data_width) - 1 DOWNTO 0);
    SIGNAL vectorDSig : STD_LOGIC_VECTOR ((3 * data_width) - 1 DOWNTO 0);

BEGIN

    vectorDSig <= STD_LOGIC_VECTOR(dSig);

    counter : counter_chain GENERIC MAP(radix, data_width)
    PORT MAP(clk, reset, take_number, rollback, dSig);

    seg0 : SevenSeg PORT MAP(vectorDSig((data_width - 1) DOWNTO 0), bcd0);
    seg1 : SevenSeg PORT MAP(vectorDSig((2 * data_width - 1) DOWNTO data_width), bcd1);
    seg2 : SevenSeg PORT MAP(vectorDSig((3 * data_width - 1) DOWNTO 2 * data_width), bcd2);
END structure;
```

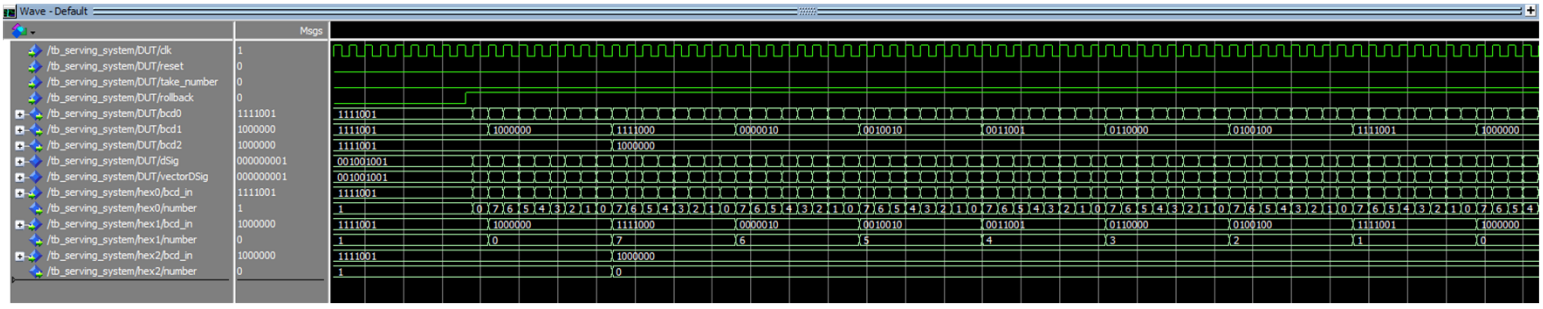
Testbench code for the serving system
Whitespace between lines removed to fit on one page

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
--testbenches have empty entity sections
ENTITY tb_serving_system IS
END tb_serving_system;
ARCHITECTURE test OF tb_serving_system IS
component serving_system IS
    GENERIC (radix : INTEGER := 9; data_width : INTEGER := 4);
    PORT (clk, reset, take_number : IN STD_LOGIC;
        rollback : IN STD_LOGIC;
        bcd0, bcd1, bcd2 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0));
END component;
COMPONENT hex_display IS
PORT (bcd_in : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
    number : OUT STD.STANDARD.INTEGER);
END COMPONENT;
    --signals and constants
    CONSTANT data_width : INTEGER := 4;
    CONSTANT radix : INTEGER := 9;
    CONSTANT HALF_PERIOD : TIME := 10 ns;
    CONSTANT PERIOD : TIME := 20 ns;
    SIGNAL sigclk : STD_LOGIC := '1';
    SIGNAL sigreset, sigtake, sigrollback : STD_LOGIC;
    signal sigBCD0, sigBCD1, sigBCD2 : std_logic_vector (6 downto 0);
    SIGNAL sevenSegOut0 : INTEGER := 0;
    SIGNAL sevenSegOut1 : INTEGER := 0;
    SIGNAL sevenSegOut2 : INTEGER := 0;
BEGIN
    DUT : serving_system GENERIC MAP(9, 4)
    PORT MAP(sigclk, sigreset, sigtake, sigrollback, sigBCD0, sigBCD1, sigBCD2);
    hex0 : hex_display PORT MAP (sigBCD0, sevenSegOut0);
    hex1 : hex_display PORT MAP (sigBCD1, sevenSegOut1);
    hex2 : hex_display PORT MAP (sigBCD2, sevenSegOut2);
    sigclk <= NOT sigclk AFTER HALF_PERIOD;
    PROCESS IS
    BEGIN
        sigreset <= '1';
        sigtake <= '0';
        sigrollback <= '0';
        WAIT FOR HALF_PERIOD;
        sigreset <= '0';
        sigtake <= '1';
        WAIT FOR 1000 * PERIOD;
        sigtake <= '0';
        WAIT FOR PERIOD;
        sigtake <= '1';
        WAIT FOR 111 * PERIOD;
        sigtake <= '0';
        WAIT FOR 10*PERIOD;
        sigrollback <= '1';
        WAIT FOR 111 * PERIOD;
        sigrollback <= '0';
        WAIT FOR PERIOD;
        sigrollback <= '1';
        WAIT FOR 2 * PERIOD;
        sigrollback <= '0';
        WAIT FOR PERIOD;
        sigtake <= '1';
        WAIT FOR PERIOD;
        sigtake <= '0';
        WAIT FOR PERIOD;
        WAIT;
    END PROCESS;
END test;
```

Bonus octal waveforms




Screenshot showing octal overflow from 777 to 000, along with other examples of incrementing in many aspects




Decrementing! All working as expected, even when underflowing. Not much to highlight here.

Bonus octal serving_system vhdl

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY serving_system IS
    GENERIC (
        radix : INTEGER := 9;
        data_width : INTEGER := 4);
    PORT (
        clk, reset, take_number : IN STD_LOGIC;
        rollback : IN STD_LOGIC;
        bcd0, bcd1, bcd2 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0));
END serving_system;

ARCHITECTURE structure OF serving_system IS

    --components
    COMPONENT SevenSeg IS
        PORT (
            D : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
            Y : OUT STD_LOGIC_VECTOR(6 DOWNTO 0));
    END COMPONENT;

    COMPONENT counter_chain IS
        GENERIC (
            radix : INTEGER := 9;
            data_width : INTEGER := 4);
        PORT (
            clk, reset, take_number : IN STD_LOGIC;
            rollback : IN STD_LOGIC;
            number : OUT UNSIGNED((3 * data_width) - 1 DOWNTO 0));
    END COMPONENT;

    --signals
    SIGNAL dSig : UNSIGNED((3 * data_width) - 1 DOWNTO 0);
    SIGNAL vectorDSig : STD_LOGIC_VECTOR ((3 * data_width) - 1 DOWNTO 0);

BEGIN

    vectorDSig <= STD_LOGIC_VECTOR(dSig);

    counter : counter_chain GENERIC MAP(radix, data_width)
    PORT MAP(clk, reset, take_number, rollback, dSig);

    seg0 : SevenSeg PORT MAP(D(3) => '0', D(2 downto 0) => vectorDSig((data_width - 1) DOWNTO
0), Y => bcd0);
    seg1 : SevenSeg PORT MAP(D(3) => '0', D(2 downto 0) => vectorDSig((2 * data_width - 1)
DOWNTO data_width), Y => bcd1);
    seg2 : SevenSeg PORT MAP(D(3) => '0', D(2 downto 0) => vectorDSig((3 * data_width - 1)
DOWNTO 2 * data_width), Y => bcd2);
END structure;
```

The main difference here is the modification of the seven-segment display port mapping, as I had to prepend a 0 to the beginning in order to make our 4-bit inputs work with a 3-bit number.

Bonus octal testbench

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
--testbenches have empty entity sections
ENTITY tb_serving_system IS
END tb_serving_system;
ARCHITECTURE test OF tb_serving_system IS
component serving_system IS GENERIC (radix : INTEGER := 9; data_width : INTEGER := 4);
    PORT (clk, reset, take_number : IN STD_LOGIC; rollback : IN STD_LOGIC;
        bcd0, bcd1, bcd2 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0));
END component;
COMPONENT hex_display IS
PORT (bcd_in : IN STD_LOGIC_VECTOR(6 DOWNTO 0); number : OUT STD.STANDARD.INTEGER);
END COMPONENT;
    --signals and constants
    CONSTANT data_width : INTEGER := 3;
    CONSTANT radix : INTEGER := 7;
    CONSTANT HALF_PERIOD : TIME := 10 ns;
    CONSTANT PERIOD : TIME := 20 ns;
    SIGNAL sigclk : STD_LOGIC := '1';
    SIGNAL sigreset, sigtake, sigrollback : STD_LOGIC;
    signal sigBCD0, sigBCD1, sigBCD2 : std_logic_vector (6 downto 0);
    SIGNAL sevenSegOut0 : INTEGER := 0;
    SIGNAL sevenSegOut1 : INTEGER := 0;
    SIGNAL sevenSegOut2 : INTEGER := 0;
BEGIN
    DUT : serving_system GENERIC MAP(7, 3)
    PORT MAP(sigclk, sigreset, sigtake, sigrollback, sigBCD0, sigBCD1, sigBCD2);
    hex0 : hex_display PORT MAP (sigBCD0, sevenSegOut0);
    hex1 : hex_display PORT MAP (sigBCD1, sevenSegOut1);
    hex2 : hex_display PORT MAP (sigBCD2, sevenSegOut2);
    sigclk <= NOT sigclk AFTER HALF_PERIOD;
    PROCESS IS
    BEGIN
        sigreset <= '1';
        sigtake <= '0';
        sigrollback <= '0';
        WAIT FOR HALF_PERIOD;
        sigreset <= '0';
        sigtake <= '1';
        WAIT FOR 8*8*8 * PERIOD;
        sigtake <= '0';
        WAIT FOR PERIOD;
        sigtake <= '1';
        WAIT FOR (1+8+8*8) * PERIOD;
        sigtake <= '0';
        WAIT FOR 10*PERIOD;
        sigrollback <= '1';
        WAIT FOR (1+8+8*8) * PERIOD;
        sigrollback <= '0';
        WAIT FOR PERIOD;
        sigrollback <= '1';
        WAIT FOR 2 * PERIOD;
        sigrollback <= '0';
        WAIT FOR PERIOD;
        sigtake <= '1';
        WAIT FOR PERIOD;
        sigtake <= '0';
        WAIT FOR PERIOD;
        WAIT;
    END PROCESS;
END test;
```

Differences here include the waiting periods, which are now based on powers of 8 instead of 10, and the fact that the DUT (serving_system) is instantiated with 7,3 as the radix and data width. Beyond that, all the files remained identical.