# Course "Formal Methods"
# Lab Test

Roberto Sebastiani
DISI, Università di Trento, Italy

July $09^{th}$, 2015

976985749

[COPY WITH SOLUTIONS]

Implement a 5-bit counter that alternates counting all odd numbers from 31 to 1 *(e.g. 31, 29, 27, ..., 3, 1)* and counting all even numbers from 30 to 0 *(e.g. 30, 28, 26, 2, 0)*. Use a variable "`out`" to represent the output of the counter. Use five Boolean variables "`b0`", "`b1`", "`b2`", "`b3`", "`b4`" to represent the bits of the counter, from the least-significative to the most-significative ones. Initially, all bits are set to TRUE. The transition relation is described as follows:

- "`b0`" changes value only when all other bits are FALSE
- "`b1`" changes value at each transition
- "`b2`" changes value only when "`b1`" is FALSE
- "`b3`" changes value only when both "`b1`" and "`b2`" are FALSE
- "`b4`" changes value only when "`b1`", "`b2`" and "`b3`" are all FALSE

Model the 5-bit counter, express the following properties, and check with nuXmv that all properties are verified.

- it is necessarily always the case that, if out is 1, then at the next step the value of the counter is 30
- it is necessarily always the case that if out = 31 then in 5 iterations out will evaluate to 21
- it is always the case that b1 changes value at each iteration
- it is always the case that, if b1, b2 and b3 are all FALSE, then the next value of b4 is !b4
- infinitely often out is 0
- if out=30 then eventually in the future out=20

Solution:

```
MODULE main
VAR
  b0: boolean; b1: boolean; b2: boolean; b3: boolean; b4: boolean;

DEFINE
  out := toint(b0) + 2*toint(b1) + 4*toint(b2) + 8*toint(b3) + 16*toint(b4);

ASSIGN
  init(b0) := TRUE; init(b1) := TRUE; init(b2) := TRUE;
  init(b3) := TRUE; init(b4) := TRUE;

  next(b0) := case
    !b1 & !b2 & !b3 & !b4 : !b0;
    TRUE : b0;
  esac;
  next(b1) := !b1;
  next(b2) := case
    !b1 : !b2;
    TRUE : b2;
  esac;
  next(b3) := case
    !b1 & !b2 : !b3;
    TRUE : b3;
  esac;
  next(b4) := case
    !b1 & !b2 & !b3 : !b4;
    TRUE : b4;
  esac;

-- it is necessarily always the case that when out is 1 then at the next
-- step the value of the counter is 30
CTLSPEC AG (out=1 -> AX(out=30))

-- it is necessarily always the case that if out = 31 then in 5
-- iterations out will evaluate to 21
CTLSPEC AG (out=31 -> AX(AX(AX(AX(AX(out=21))))))

-- it is always the case that b1 changes value at each iteration
CTLSPEC AG (b1 <-> AX(!b1))

-- it is always the case that, if b1, b2 and b3 are all FALSE,
-- then the next value of b4 is !b4
CTLSPEC AG (!b1&!b2&!b3 -> (b4 <-> AX(!b4)))
```

```
-- infinitely often out is 0
CTLSPEC AG AF (out = 0)

-- if out=30 then eventually in the future out=20
CTLSPEC (out=30 -> AF (out=20))
```

# 1 Spin

Write a Promela program defining a process $sum(n, c)$ which recursively computes the sum of the first $n$ positive integer numbers $sum(n) = n + (n-1) + (n-2) + ... + 1$. More in detail, the process should compute $sum(n)$ as $n + sum(n-1)$ for $(n >= 1)$ and return the computed value to the parent process via a message on channel c. In the init function, compute the sum of 100 and verify that it is equal to 5050 with an assertion check.

Solution:

```
proctype sum(int n; chan c) {
    int result;
    chan child = [1] of { int };
    if
        :: (n <= 1) -> c!1
        :: (n >= 2) ->
            run sum(n-1, child);
            child?result;
            c!n+result
    fi
}

init {
    int result;
    chan child = [1] of { int };
    run sum(100, child);
    child?result;
    printf("result: %d\n", result);
    assert(result == 5050);
}
```

| b4 | b3 | b2 | b1 | b0 | out |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 31 |
| 1 | 1 | 1 | 0 | 1 | 29 |
| 1 | 1 | 0 | 1 | 1 | 27 |
| 1 | 1 | 0 | 0 | 1 | 25 |
| 1 | 0 | 1 | 1 | 1 | 23 |
| 1 | 0 | 1 | 0 | 1 | 21 |
| 1 | 0 | 0 | 1 | 1 | 19 |
| 1 | 0 | 0 | 0 | 1 | 17 |
| 0 | 1 | 1 | 1 | 1 | 15 |
| 0 | 1 | 1 | 0 | 1 | 13 |
| 0 | 1 | 0 | 1 | 1 | 11 |
| 0 | 1 | 0 | 0 | 1 | 09 |
| 0 | 0 | 1 | 1 | 1 | 07 |
| 0 | 0 | 1 | 0 | 1 | 05 |
| 0 | 0 | 0 | 1 | 1 | 03 |
| 0 | 0 | 0 | 0 | 1 | 01 |
| 1 | 1 | 1 | 1 | 0 | 30 |
| 1 | 1 | 1 | 0 | 0 | 28 |
| 1 | 1 | 0 | 1 | 0 | 26 |
| 1 | 1 | 0 | 0 | 0 | 24 |
| 1 | 0 | 1 | 1 | 0 | 22 |
| 1 | 0 | 1 | 0 | 0 | 20 |
| 1 | 0 | 0 | 1 | 0 | 18 |
| 1 | 0 | 0 | 0 | 0 | 16 |
| 0 | 1 | 1 | 1 | 0 | 14 |
| 0 | 1 | 1 | 0 | 0 | 12 |
| 0 | 1 | 0 | 1 | 0 | 10 |
| 0 | 1 | 0 | 0 | 0 | 08 |
| 0 | 0 | 1 | 1 | 0 | 06 |
| 0 | 0 | 1 | 0 | 0 | 04 |
| 0 | 0 | 0 | 1 | 0 | 02 |
| 0 | 0 | 0 | 0 | 0 | 00 |

Figure 1: bits evolution at each transition