Multi-bit Extraction

We can extract multiple bits from a bit vector using a square bracket [] and an inclusive range (upper_index:lower_index) in Bluespec (e.g., x[32:10]). This operator differs from the Python’s one used for indexing arrays. The valid index range is N-1 to 0 for a N-bit variable.

Note that multiple bit extracting range can only be indexed starting from a more significant bit (higher index) to a less significant bit (lower index).

- x[5:1] is same as {x[5], x[4], x[3], x[2], x[1]}
- x[2:0] is same as {x[2], x[1], x[0]}
- x[1:1] is same as x[1]
- x[1:3] is illegal
- x[1:5] is illegal

Bluespec Integer Literals

Bluespec expresses integers in several manners. Bluespec Integer Literals could be sized or unsized and could be binary, decimal or hexadecimal.

Sized Literals
1. 5'b10011
2. 8'hAB
3. 12'd10

Unsized Literals
4. ’1
5. ’b1100_0110
6. ’hFFCC_0110
7. ’d70

The case statement

case ( expression_to_evaluate )
  case_item1, case_item2: expression1;
  case_item3: expression3;
  default: default_expression;
endcase
4:1 Multiplexer could be easily implemented using the case statement.

```markdown
function Bit#(1) four_to_one_mux1(Bit#(4) in, Bit#(2) sel);
    Bit#(1) ret = 0;
    case (sel)
        2'b00: ret = in[0];
        2'b01: ret = in[1];
        2'b10: ret = in[2];
        2'b11: ret = in[3];
    endcase
    return ret;
endfunction
```

```markdown
function Bit#(1) four_to_one_mux2(Bit#(4) in, Bit#(2) sel);
    return (case (sel)
        2'b00: in[0];
        2'b01: in[1];
        2'b10: in[2];
        2'b11: in[3];
        // default: not needed since we cover all case items
    endcase);
endfunction
```

**Problem 1.**

Write a Bluespec function checkParity, which takes as input a 5-bit vector and returns True if the input vector has odd number of 1’s, otherwise it returns a false.

```markdown
function Bit#(1) checkParity(Bit#(5) in);
endfunction
```
**Problem 2.**

Write Bluespec function `addParity`, which takes as input a 4 bit vector and returns a 5 bit vector which adds an odd-parity bit to the input in the most significant position.

Note that an odd-parity bit makes the number of 1’s in the resulting bit vector odd and an even-parity bit makes the number of 1’s in the resulting bit vector even.

```plaintext
function Bit#(5) addParity(Bit#(4) in);
    // checks if input has odd number of ones
    // if yes, add 0, if not, add 1 to make odd-parity
    Bit#(1) parity = (oddNumOnes == 1)? 0: 1;
    return {parity, in};
endfunction
```

**Problem 3.**

Implement a 4-bit ripple carry adder in Bluespec using the half adder (ha) and full adder (fa) functions that we discussed in the lecture. The function specification for this adder should look like:

```plaintext
function Bit#(4) addRecitation(Bit#(4) a, Bit#(4) b, Bit#(1) c_in);
endfunction
```

```plaintext
function Bit#(4) addRecitation(Bit#(4) a, Bit#(4) b, Bit#(1) c_in);
    Bit#(2) b0 = fa(a[0], b[0], c_in);
    Bit#(2) b1 = fa(a[1], b[1], b0[1]);
    Bit#(2) b2 = fa(a[2], b[2], b1[1]);
    let b3 = fa(a[3], b[3], b2[1]);  // we can use let
    return {b3[0], b2[0], b1[0], b0[0]};
endfunction
```