Vectors and bounds checking

Arvind and Thomas Bourgeat 10/03/2018

Bluespec compiler does not do adequate bounds checking, therefore sometimes accepts codes, which show behaviors that may surprise the user.

A vector is declared as follows:
Vector#(n, t) xs;
where n is the size designated by an integer (constant or parameter), and t is the type of the elements in xs.

For example:
Vector#(10, Bit#(32)) xs;
produces a vector of size 10 whose elements are 32-bit wide.

A vector is accessed by writing xs[ idx ] where idx can be an integer or a bounded value like Bit#(k).

If idx is an integer, idx is defined necessarily at compile time. If idx is greater than n, the compiler is going to raise an error at compile time: “idx is greater than n”. (This error does not arise from the typechecking phase of the compilation)

If idx is a Bit#(k), there are two subcases. Either idx is a constant (or a parameter) known at compile time or it’s a dynamic value which is derived from a register value. If idx is a constant, like before, the compiler will complain in case of an out-of-bounds access. If idx is a dynamic value, then unexpected behaviors can arise. Consider the following code snippet:

Reg#(Bit#(5)) idx <- mkReg(25);
Vector#(10, Bit#(17)) xs;
xs[idx];

This snippet is accepted by the compiler even though xs[25] is not defined. Unfortunately, the compiler does not complain in this case and reports more or less an arbitrary value. This problem arises because the out-of-bound property for a dynamic index cannot be checked statically. In the above case, the compiler could have known statically that idx is constant and so xs[25] is not defined. Instead, it assumes that idx is computed dynamically and relies on the programmer to ensure bounds checking.

There are two ways to deal with out-of-bounds problems:
1. Use vectors of sizes $2^n$ and access them using `Bit#(n)` idx variables.
2. It is not always desirable to use vectors of size power of two. In that case we can explicitly check that the idx value falls within the bounds as follows:

```vhdl
Vector#(10, Bit#(17)) xs;
Reg#(Bit#(5)) idx <- mkReg(0);

//Replace each access xs[idx] with
if( idx => 10 ) begin
    $display("Out of bound access on xs");
    $finish;
end
xs[idx];
```

This solution will work in the simulator (bluesim) but won’t synthesize into hardware.

There are other tools, which, in many cases, can check statically if a dynamically generated idx is within bound in a given program. Unfortunately, such tools are “heavy duty” and we won’t be using them.