COMP 408: Verifying a Systolic Array Element

This exercise is intended to be a large-scale review of the various aspects of SMV-based verification that we have covered so far. As it is a more complicated example than those you have seen before, I'm giving you three weeks in which to complete it. However, I'm also setting intermediate deadlines for various stages of the project. These deadlines are indicated next to each portion of the assignment below.

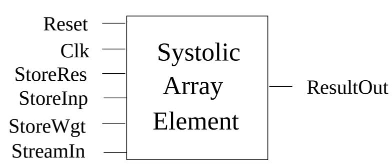
For this exercise, you will implement and verify a portion of a systolic array. The desired circuit processes a stream of numeric input values $x_1, \ldots, x_n$. These input values have to be multiplied with a list of weights, $w_1, \ldots, w_k$. The output stream consists of a sequence of values $y_1, y_2, \ldots, y_{n+1-k}$, where each y_i is computed as per the equation:

$$y_i = w_1 x_i + w_2 x_{i+1} + \ldots + w_k x_{i+k-1}$$

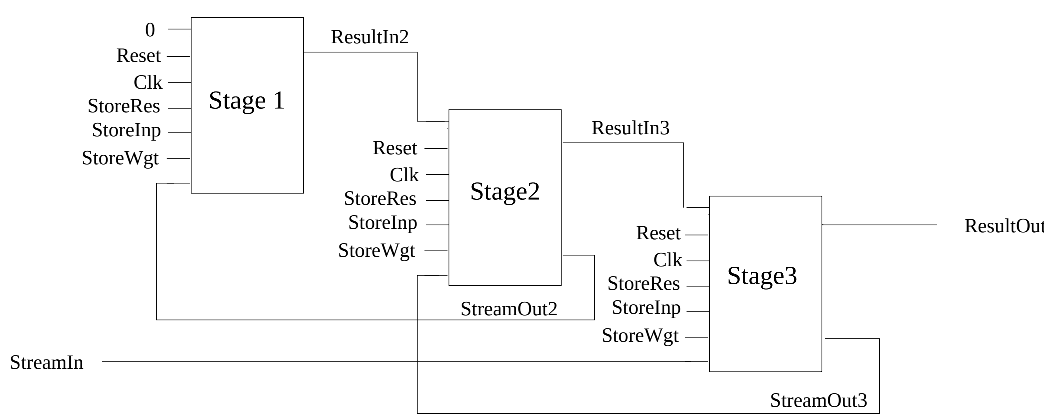
For example, if $k = 3$ and $n = 5$, the output sequence consists of three values, defined as follows:

$$\begin{aligned} y_1 &= w_1 x_1 + w_2 x_2 + w_3 x_3 \\ y_2 &= w_1 x_2 + w_2 x_3 + w_3 x_4 \\ y_3 &= w_1 x_3 + w_2 x_4 + w_3 x_5 \end{aligned}$$

The interface to the circuit is given by the diagram below. The weights and the inputs are fed into the circuit serially on StreamIn. The output sequence is shifted out serially on ResultOut. Reset is used to clear the current weights and values as stored in the circuit. StoreWgt should be true when the values on StreamIn are intended to be weights. StoreInp should be true whenever the current value on StreamIn needs to be passed along to the next stage (i.e., for loading weights or processing inputs). StoreRes should be true when the output sequence should be produced.



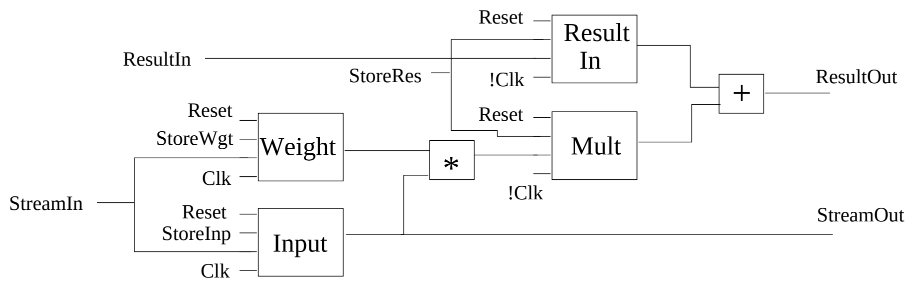
You will implement a version of such a circuit with three weights. At an abstract level, the circuit consists of three stages (one corresponding to each weight):



Each stage multiplies its stored weight with its current input, adds the result to the result passed in from the previous stage, and passes this sum out as the ResultIn to the next stage. Each stage also passes the input value it last processed on to the previous stage on its StreamOut wire. In order for this circuit to work properly, the values on StreamIn can only change on every fourth step. In all of these diagrams, input signals with the same names correspond to the same variables from the environment.

Before you go any further, you should convince yourself that the y_i's are computed as per the above equations under this configuration of stages.

Internally, each stage has the following structure, which implements the multiplication and addition described above:



The large squares correspond to registers that store certain values relevant to the stage. The **Weight** register stores the weight, the Input register stores the input value currently being processed, the ResultIn register stores the partial result passed into the stage, and the Mult register stores the product computed in the stage. Each register takes four inputs: Reset, an enabling signal (such as StoreWgt), a data input, and a clock signal. The registers should set their values to 0 whenever Reset is true. If the value passed to the clock signal is true and the enabling signal is true, the value on the data input should be stored in the register. Otherwise, the registers hold their previous values.

For this exercise, I want you to do the following:

1. Write an SMV definition of the three-stage systolic array element described through the above diagrams. (Note: do not try to convert this into a state machine as you did for the single pulser example. Simply use variables for the various registers in the stage architecture.) Assume StreamIn has range 0..3.

   This portion is due by class on Wednesday, February 5. You do not need to verify anything for this deadline. However, please make sure that SMV accepts your definition (you can run SMV on a file containing no SPECs — this will cause SMV to parse and typecheck the file, as well as indicate how many BDD nodes are needed to represent it).

2. Write an environment model for this element. Your model needs to generate the input signals shown in the diagram describing the interface. The environment should repeatedly reset the system, feed in weights, then feed in the inputs (being sure to wait for the output to be produced before resetting for the next iteration). To keep things simple, your model can have Reset issued only at the start of a new computation (i.e., just before a new set of weights is shifted in). Your model should define signal Clk to simply toggle its value on each step.

3. Write the SPEC or SPECs that would verify the behavior of this systolic array element.

   These two portions are due by class on Wednesday, February 12. Again, you do not need to verify the SPEC for this week. Just write down the SPEC (or SPECs) that you intend to verify (with a description of what each one is intended to test). Make sure that your SMV file from the first portion is still accepted by SMV after you add the environment model.

4. Attempt to verify your SPECs. If you are unable to verify an entire SPEC, see if you can verify some simplified or modified version. In deciding what to simplify, keep in mind that your goal should be

to convince yourself that your design works. As each simplified version verifies, try to extend it to something more complicated. Keep in mind that you may be able to verify a SPEC by tinkering with the variable ordering.

In writing up this part, you do not need to provide error track analyses. I would, however, like you to write up the simplified versions that you tested; give the simplified SPEC and a short description of why you decided to try that particular simplification. Also indicate the size of the BDD used to verify each simplification. Discuss what made the original SPEC difficult to verify; i.e, was the design itself too large for model-checking, or were there other factors that led to the large BDDs? If you played with variable orderings at all, talk about what you tried and whether or not it had any effect. Also discuss how much confidence you have in your design based on what you managed to verify. In short, I'd like you to write a small report on your experiences verifying this circuit (consider this a dry run for a similar report you will have to write for your course project).

If you did get your entire original SPECs to verify, make a new version of the design that makes StreamIn have range 0..15. Can you also verify your SPECs under this version?

This final portion is due by class on Wednesday, February 19.