Contents

1 Matrix matrix multiplication

For your first foray into chisel you will design a matrix matrix multiplication unit. Matrix multiplication is fairly straight forward, however on hardware it's a little trickier than the standard for loops normally employed..

Important You will be working with skeleton code. Every component you implement has a corresponding skeleton source file found in src/main/scala/, so for Vector you're looking for src/main/scala/Vector.scala

1.1 Task 1 - Vector

The first component you should implement is a register bank for storing a vector.

In Vector.scala you will find the skeleton code for this component. Unlike the standard Chisel.Vec our custom vector has a read enable which means that the memory pointed to by idx will only be overWritten when writeEnable is true.

Implement the vector and test that it works by running `testOnly Ex0.VectorSpec` in your sbt console.

1.2 Task 2 - Matrix

The matrix works just like the vector only in two dimensions. The skeleton code and associated tests should make the purpose of this module obvious. Run the tests with `testOnly Ex0.MatrixSpec`

1.3 Task 3 - Dot Product

This component differs from the two previous in that it has no explicit control input, which might at first be rather confusing.

With only two inputs for data, how do we know when the dotproduct has been calculated? The answer to this is the `elements` argument, which tells the dot product calculator the size of the input vectors. Consequently, the resulting hardware can only (at least on its own) compute dotproducts for one size of vector, which is fine in our circuit.

To get a better understanding we can model this behavior in regular scala:

```
kcase kclass n+ncDotProdCalculatoro(nvectorLenk: k+ktInto,
↪  ntimeStepk: k+ktInto, naccumulatork: k+ktInto)
  kdef nupdateo(ninputAk: k+ktInto, ninputBk: k+ktInto)k:
  ↪  o(k+ktInto, k+ktBooleano, n+ncDotProdCalculatoro) k= o
    kval nproduct k= ninputA o* ninputB
    kifo(((ntimeStep o+ l+m+mi1o) o nvectorLeno) o== l+m+mi0o)
      o(naccumulator o+ nproducto, k+kctrueo,
      ↪  kthiso.ncopyo(ntimeStep k= l+m+mi0o, naccumulator k=
      ↪  l+m+mi0o))
    kelse
      o(naccumulator o+ nproducto, k+kcfalseo,
      ↪  kthiso.ncopyo(ntimeStep k= kthiso.ntimeStep o+
      ↪  l+m+mi1o, naccumulator k= naccumulator o+ nproducto))
  o
o
```

To see it in action run `testOnly Examples.SoftwareDotProdSpec` in your sbt console.

As with the previous tasks, the dot product calculator must pass the tests with `testOnly Ex0.DotProdSpec`

1.3.1 Timing

As shown in the timing diagram below, the dot product calculator should deliver the result as soon as possible. This means you will have to drive the output with the sum of the accumulator and the product of the two inputs. If you choose to drive the output only by the value of the accumulator your circuit will lag behind by one cycle, which while good for pipelining purposes is not good for passing the test purposes.

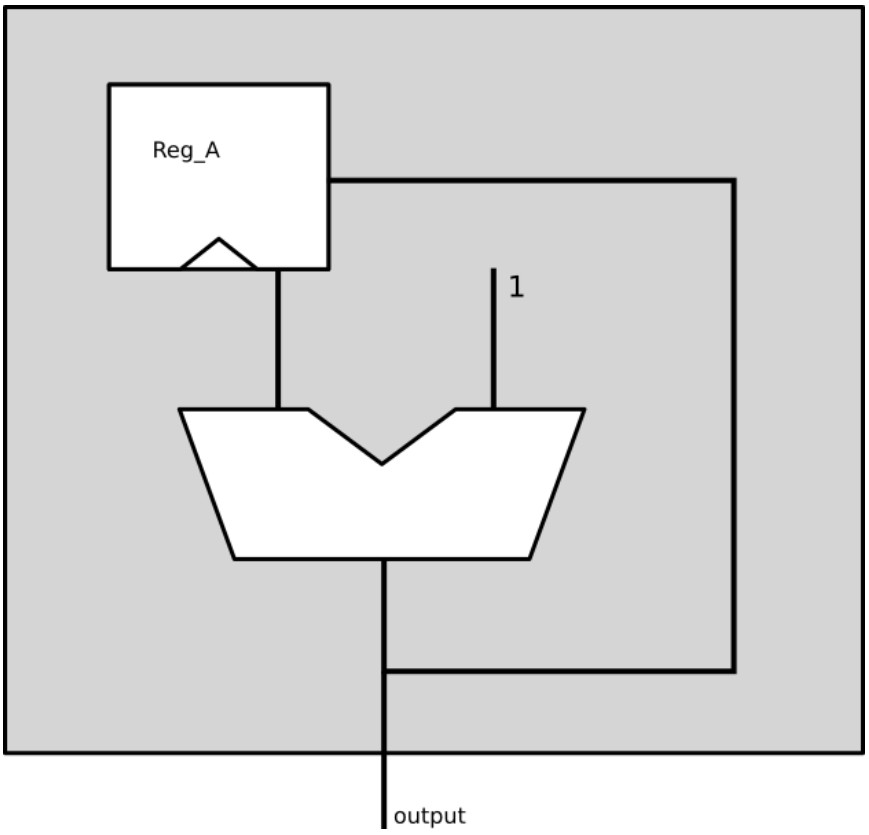


Figure 1: The expected output of the dot product calculator

1.3.2 Chisel counters and a short detour on scala documentation

Doing an action for a set amount of timesteps is a very common task in hardware design, so this functionality is included in chisel via the Counter class. In order to understand how this counter works you can google "chisel counter" and get the following `https://chisel.eecs.berkeley.edu/api/3.0.1/chisel3/util/Counter.html` as first result. However, this is for the counter Class, what you actually want is the counter Object: `https://chisel.eecs.berkeley.edu/api/3.0.1/chisel3/util/Counter\protect\T1\textdollar.html`

This can be very confusing when new to scala, but it is simply convention: When a class and an object share name this is just a convenience for keeping static methods, such as constructors, separated from the non-static methods.

In the Counter object (the second link) there is an apply method:

```
kdef napplyo(ncondk: k+ktBoolo, nnk: k+ktInto)k: o(k+ktUInto,
↪  k+ktBoolo)
```

The type signature tells you that the input is a regular scala integer, and a chisel boolean (scala booleans are of type `Boolean`, rather than `Bool`) and the output is a UInt and a chisel boolean. This means that upon instantiating a Counter with its apply method you only get the outputs from the counter, not the object itself. The result is a convenient method of making a counter, simply supply how many ticks it takes for the counter to roll over, as well as an input signal for enabling the clock, and receive a tuple with the signal for the counters value, as well as a boolean signal that toggles whenever the clock rolls over.

A special property of apply methods are that they can be called directly on the object. `Counter.apply(cCond, 10)` does the same as `Counter(cCond, 10)`. To call the class constructor, use the `new` keyword.

1.4 Task 4 - Matrix Matrix multiplication

With our matrix modules and dot product calculators we have every piece needed to implement the matrix multiplier.

When performing matrix multiplication on a computer transposing the second matrix can help us reduce complexity by quite a lot. To examplify, consider

```
    | 2,  5 |
A = | 7, 1 |
    | 0,  4 |
```

```
B = | 1,  1,  2 |
    | 0,  4,  0 |
```

It would be much simpler to just have two modules with the same dimensions, and we can do this by transposing B so we get

```
     | 2,  5 |
A  = | 7, 1 |
     | 0,  4 |

     | 1,  0 |
BT = | 1,  4 |
     | 2,  0 |
+endsrc text

Now we need to do is calculate the dot products for the final
↪  matrix:

+beginsrc text
if A*B = C then

     |  A[0] × BT[0],   A[0] × BT[1],   A[0] × BT[2] |
C  = |  A[1] × BT[0],   ...          ,   ...          |
     |  ...          ,   ...          ,   A[2] × BT[2] |

where
A[0] × BT[0] is the dot product of [2, 5] and [1, 0]
and
A[0] × BT[1] is the dot product of [2, 5] and [1, 4]
and so forth..
```

Because of this, the input for matrix B will be supplied transposed, thus you do not have to worry about this. For B the input would be [1, 0, 1, 4, 2, 0].

The skeleton code for the matrix multiplier is less detailed, with only one test. You're encouraged to write your own tests to make this easier.

1.4.1 Structuring your circuit

It is very easy to get bogged down with details in this exercise, so it's useful to take a few moments to plan ahead.

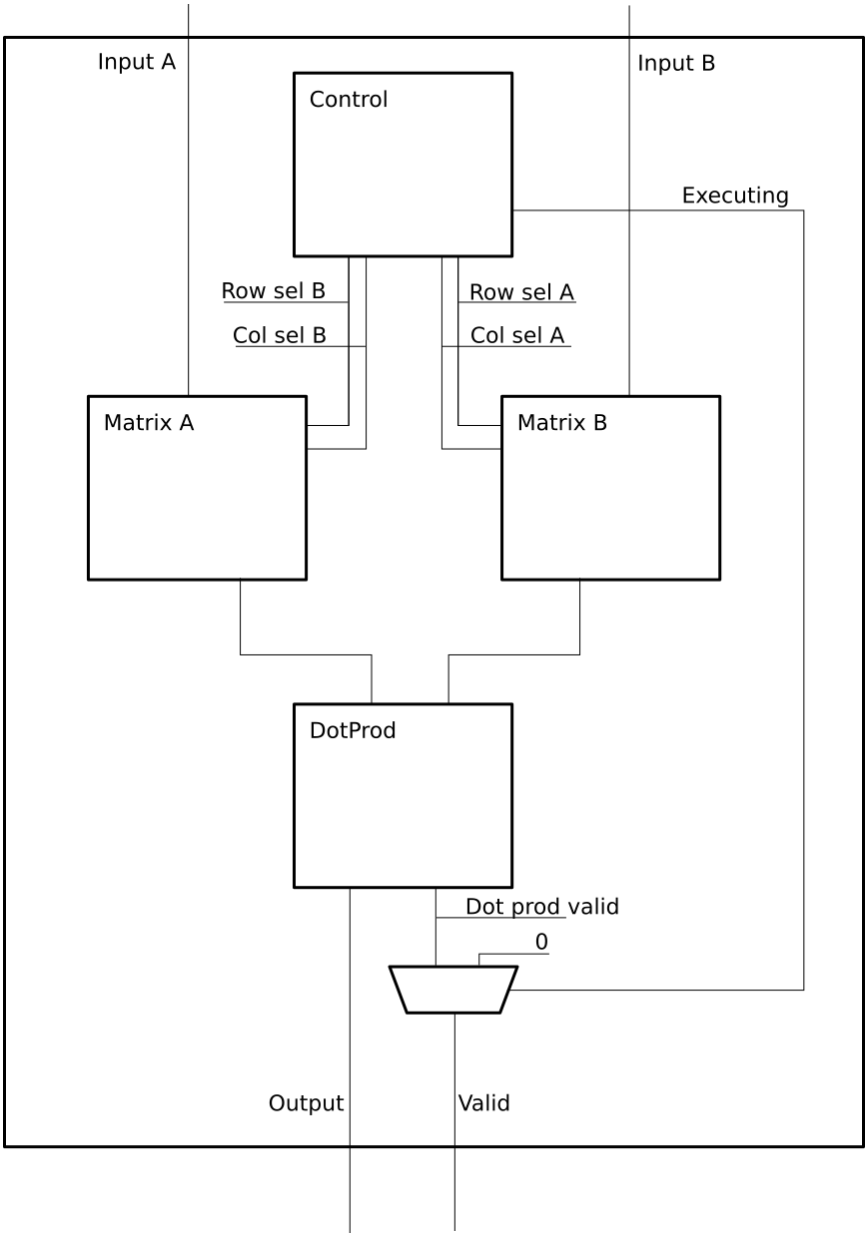
A natural way to break down the task is to split it into two phases: setup and execution. For setup you simply want to shuffle data from the input signals to your two matrix modules.

The next task is to actually perform the calculation. This is a little more complex, seeing as the read patterns are different from matrix A and B.

To make this simpler a good idea is to introduce a control module. This module should keep track of which state the multiplier is in, setup or execution, and provide the appropriate row and column select signals.

You may also choose to split the control module into an init controller and an execution controller if you see fit.

A suggested design is shown underneath:



1.4.2 Timing

The timing for your matrix multiplier is straight forward. For a 3x4 matrix it takes 12 cycles to input data (cycles 0 to 11), and execution should proceed on cycle 12. While you can technically start execution sooner than this the

tests expect you to not start executing before all data is loaded. As long as you start executing just as data has been loaded your dot prod design will take care of the rest.

1.4.3 Testing

In order to make testing easier, consider testing your row and column select signals first. The actual values stored in the matrixes are just noise, the important part is that you select the correct rows and columns at the correct times for the correct matrixes, and if you do this the rest is comparatively easy.

1.5 Bonus exercise - Introspection on code quality and design choices

This last exercise has no deliverable, but you should spend some time thinking about where you spent most of your efforts.

A common saying is "A few hours of work can save you from several minutes of planning", and this holds especially true for writing chisel!!