1 TODOS

1. Need to add area/time estimates for each element and application

2 Type Notation

2.1 Combinational Element Types

The type signature of combinational elements is:

input 0 type $\rightarrow$ ... $\rightarrow$ input n-1 type $\rightarrow$ output type

Since these combinational elements process all inputs and output every clock cycle, they do not need ready-valid signals to indicate when they are ready for input or emitting valid outputs.

Each input or output has type T or T[p]. T is a base type, and it can contain nested types like arrays that are not relevant for the current operation. T[p] is an array of length p of T's. T[p][q] is an array of length p of arrays of length q of T's.

2.2 Sequential Element Types

The type signature of sequential elements is:

{input 0 num cycles, input 0 type per cycle} $\rightarrow$ ... $\rightarrow$
{input n-1 num cycles, input n-1 type per cycle} $\rightarrow$
{ouput num cycles, output type per cycle}

One of the inputs or outputs of a stream may have different types during different clock cycles in a stream. For example, a sequential reduce may emit invalid data for most of the stream's cycles and then emit the result on the final cycle of the stream. Instead of {cycles, type for all cycles}, this type is represented in the following way:

[type at cycle 0, type at cycle 1, ..., type at cycle n-1]

For short-hand where a type is the same for multiple cycles:

[type at cycle 0(0:n-2), type at cycle n-1]

The invalid type is represented as $\emptyset$.

These elements interfaces also must have clock inputs and may have ready-valid inputs and outputs. These ready-valid handshake ports indicate:

1. ready

   input: indicates to this sequential element that the next one in the pipeline has completed its prior input stream and is ready to receive more input.

   output: indicates to the previous sequential element in the pipeline that this one has completed its prior input stream and is ready to receive more input.

2. valid

   input: indicates to this sequential element that the previous one in the pipeline is emitting valid data that this sequential element can use as input.

   output: indicates to the next sequential element in the pipeline that this one is emitting valid data that the next one can use as input.

3 Basic Elements

These are elements that are not built using other Aetherling elements.

3.1 Combinational Elements

1. tuple :: S → T → (S,T)
2. lb p w :: T[p] → T[w+p-1]
3. overlap_partition p w :: T[w+p-1] → T[p][w]
4. partition p k :: T[k] → T[k/p][p]
5. flatten p k :: T[k/p][p] → T[k]
6. map p f :: S_1[p] → ... → S_{n-1}[p] → T[p]

   s.t. f :: S_1 → ... → S_n → T

7. reduce p f :: T[p] → T

   s.t. f :: (T,T)→ T

8. up k :: T → T[k]
9. down k :: T[k] → T
10. zip :: (S[k],T[k]) → (S,T)[k]
11. unzip :: (S,T)[k] → (S[k], T[k])
12. mem_read p :: () → T[p]
13. mem_write p :: T[p] → ()

3.2 Sequential Elements

1. serialize k :: [T[k], ∅(1:k-1)] → {k, T}
2. deserialize k :: {k, T} → [∅(0:k-2), T[k]]
3. reduce_seq p f :: [T[p], ∅(1:p-2)] → [∅(0:p-3), T]

   s.t. f :: (T,T)→ T

4. reduce_seq_stream p f :: {p, T} → [∅(0:p-2), T]

   s.t. f :: (T,T)→ T

4 Basic Applications

These are simple combinations of the basic elements.

4.1 Passthrough

1. mem_write 1 $ mem_read 1
2. mem_write t $ mem_write t

4.2 Array-Stream Conversions

1. mem_write 1 $ deserialize t $ mem_read t

   Note that mem_read fires onces every t'th clock cycle

2. mem_write 1 $ deserialize t $ serialize t $ mem_read 1

   Note mem_write fires every t'th clock cycle

4.3 Map

1. mem_write 1 $ map 1 f $ mem_read 1
2. mem_write t $ map t f $ mem_read t
3. mem_write t $ map t f2 $ map t f1 $ mem_read t

4.4 Reduce

1. mem_write 1 $ reduce t f $ mem_read t
2. mem_write 1 $ reduce t f $ deserialize t f $ mem_read 1

   Note everything after deserialize fires every t'th clock cycle

4.5 Array Dimension Conversions

1. mem_write $\frac{t}{p}$ $ partition p t $ mem_read t

   Note that the element type mem_write is writing is T[p], and it writes $\frac{t}{p}$ of them every clock.

2. mem_write t $ flatten t $ partition t $ mem_read t
3. mem_write 1 $ flatten t $ partition t $ deserialize t $ mem_read t

   Note everything after deserialize fires onces every t'th clock cycle

4. mem_write 1 $ down t $ up t $ mem_read 1

5 Composed Elements

These are elements that are composed from basic elements and other other composed elements.

5.1 Sequential Versions Of Basic Combinational Elements

These are operations that perform the same operations as the basic ones, but do it over multiple clock cycles.

1. map_seq p f :: $[S_1[\mathrm{p}], \emptyset(1\text{:p-1})] \to ... \to [S_{n-1}[\mathrm{p}], \emptyset(1\text{:p-1})] \to [\emptyset(0\text{:p-2}), \mathrm{T}[\mathrm{p}]]$

   s.t. f :: $S_1 \to ... \to S_n \to \mathrm{T}$

   This map takes all the inputs in on the first cycle of the stream and emits all the outputs on the final cycle of the stream.

   implementation: map_seq p f = deserialize p $ map 1 f $ serialize p

   note that in the above implementation, the type for serialize contains all the different input types to map

2. map_seq_stream p f :: $[S_1[\mathrm{p}], \emptyset(1\text{:p-1})] \to ... \to [S_{n-1}[\mathrm{p}], \emptyset(1\text{:p-1})] \to \{\mathrm{p}, \mathrm{T}\}$

   s.t. f :: $S_1 \to ... \to S_n \to \mathrm{T}$

   This map takes all the inputs in on the first cycle of the stream and emits one output on each cycle of the stream.

   implementation: map_seq_stream p f = map 1 f $ serialize p

3. up_seq k :: $[\mathrm{T}, \emptyset(1\text{:k-1})] \to \{\mathrm{k}, \mathrm{T}\}$

   implementation: up_seq k = serialize k

4. down k :: $\mathrm{T}[\mathrm{k}] \to \mathrm{T}$