1 Notation

Single cycle, combinational logic has the form:

input 1 $\rightarrow$ ... $\rightarrow$ input n $\rightarrow$ output

Each input or output has type T or T[p]

T is a base type, 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.

Multiple cycle, sequential logic has the form: {c, T} is a stream of T's. There is one T per clock cycle for c cycles.

2 Basic Operations

2.1 Combinational Operations

These operations require only combinational logic. Therefore, they do not require a clock nor ready-valid signals. All their input and output types are over an implicit stream. This means that that they operate on one input every clock cycle and emit one output every clock cycle.

tuple :: $S \rightarrow T \rightarrow (S, T)$

lb p w :: $T[p] \rightarrow T[w + p - 1]$

overlap_partition p w :: $T[w + p - 1] \rightarrow T[p][w]$

partition p k :: $T[k] \rightarrow T[k/p][p]$

flatten p k :: $T[k/p][p] \rightarrow T[k]$

map p f :: $S[p] \rightarrow T[p]$ s.t. $f :: S \rightarrow T$

reduce p f :: $T[p] \rightarrow T$ s.t. $f :: (T, T) \rightarrow T$

up k :: $T \rightarrow T[k]$

down k :: $T[k] \rightarrow T$

zip :: $(S[k], T[k]) \to (S, T)[k]$

unzip :: $(S, T)[k] \to (S[k], T[k])$

mem_read p :: $() \to T[p]$

mem_write p :: $T[p] \to ()$

2.2 Sequential Operations

These operations require sequential logic. Therefore, they require a clock and ready-valid signals. Their input and output types are over explicit streams.

serialize k :: $\{1, T[k]\} \to \{k, T\}$

deserialize k :: $\{k, T\} \to \{1, T[k]\}$

2.2.1 Ready-Valid Connector Operations

Any sequential block of

3 Basic Applications

These are simple combinations of the

3.1 Passthrough

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

3.2 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

3.3 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

3.4 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

3.5 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

4 Advanced Applications

4.1

These are operations that are combinations of the basic ones. They are also applica

1. t