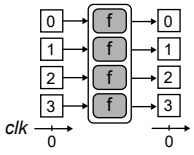
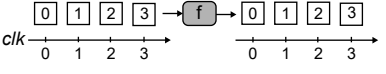
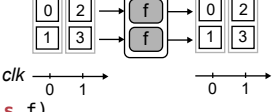
```
conv_math x =
  map (\y -> div (tuple y 3)) (reduce add x)

conv1d input =
  let shift_once = shift input
  let shift_twice = shift shift_once
  let window_tuple = map2 tuple_append
  (map2 tuple shift_once shift_twice) input
  let window = map tuple_to_seq
  (partition N 1 window_tuple)
  let result = map conv_math window
  unpartition result
```




```
map_s f :: SSeq 4 t -> SSeq 4 t'
throughput(map_s f) = 4 elt/clk
area(map_s f)       = 4 x area(f)
```




```
map_t f :: TSeq 4 0 t -> TSeq 4 0 t'
throughput(map_t f) = 1 elt/clk
area(map_t f)       = area(f)
```




```
map_t (map_s f)
  :: TSeq 2 0 (SSeq 2 t) -> TSeq 2 0 (SSeq 2 t')
throughput(map_t (map_s f)) = 2 elt/clk
area(map_t (map_s f))       = 2 x area(f)
```