# 60008 Custom Computing 2021 Past Paper Solution

*Note: this is not an official solution but I'm trying to make it as accurate as possible. If you find any typos or incorrect answers and explainations, please email me at xz1919@ic.ac.uk with the title <course_number>-<course_name>-<year>-past-paper-enquiries or directly message me through Whatsapp. If you need explaination or reviews on the provided answer, please contact me as well. :)*

---

1. Part A

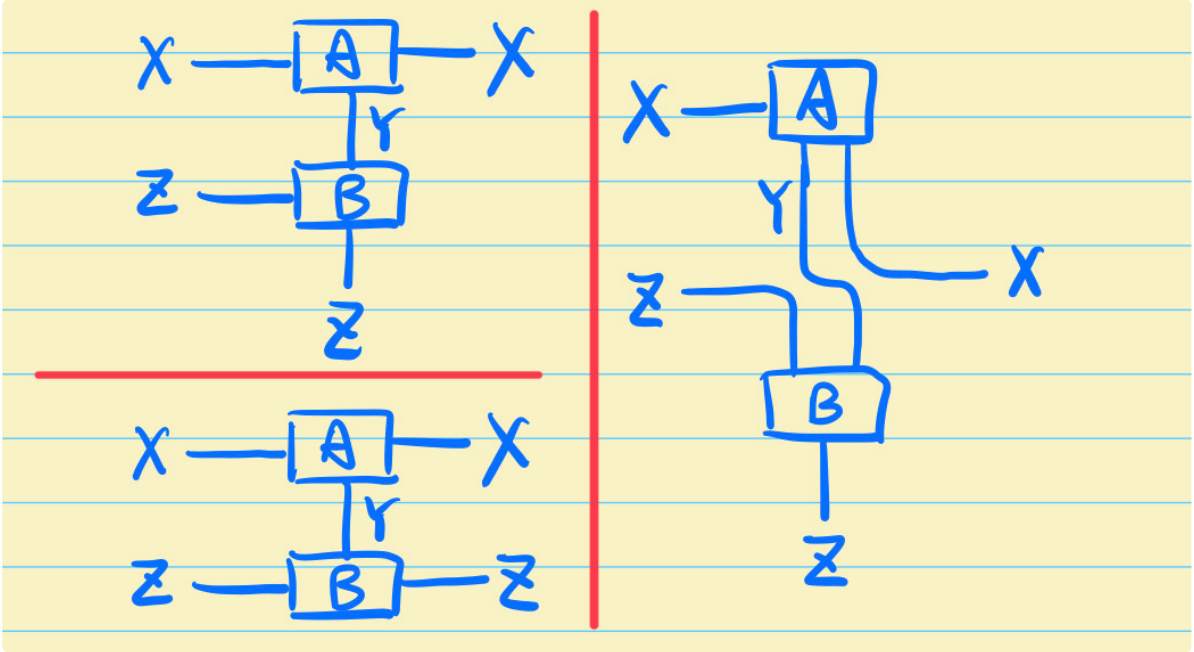
   a. Given A: X ~ <Y, X> and B: <Z, Y> ~ Z, the higher-order function above (↑) is given by:

   A ↑ B = snd A ; rsh ; fst B

   What is the type of A ↑ B? Provide a diagram of A ↑ B, showing that block A is above block B.

   The type of A ↑ B is <Z, X> ~ <Z, X>

   The diagram of A ↑ B is shown below, with block A above block B. All three diagrams work (e.g. you can choose to explicitly show rsh or not)

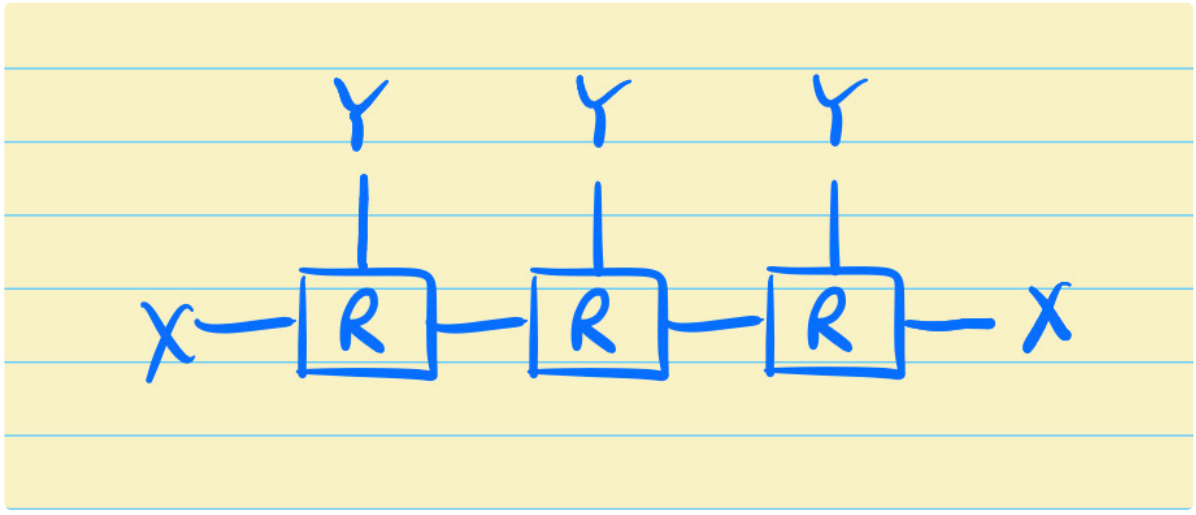



   b. Given R: <X, Y> ~ X, what are the type and the recursive definition of the left reduction combinator, $\text{rdl}_n$ R? Provide a diagram of rdln R when n=3.

   The type of $\text{rdl}_n$ R is <X, <Y>$_n$> ~ X and its recursive definition would be

```
rdl 1 R = snd [-]^~1; R
rdl n R = snd (apl (n-1))^~1; rsh; fst R; rdl (n-1) R
```

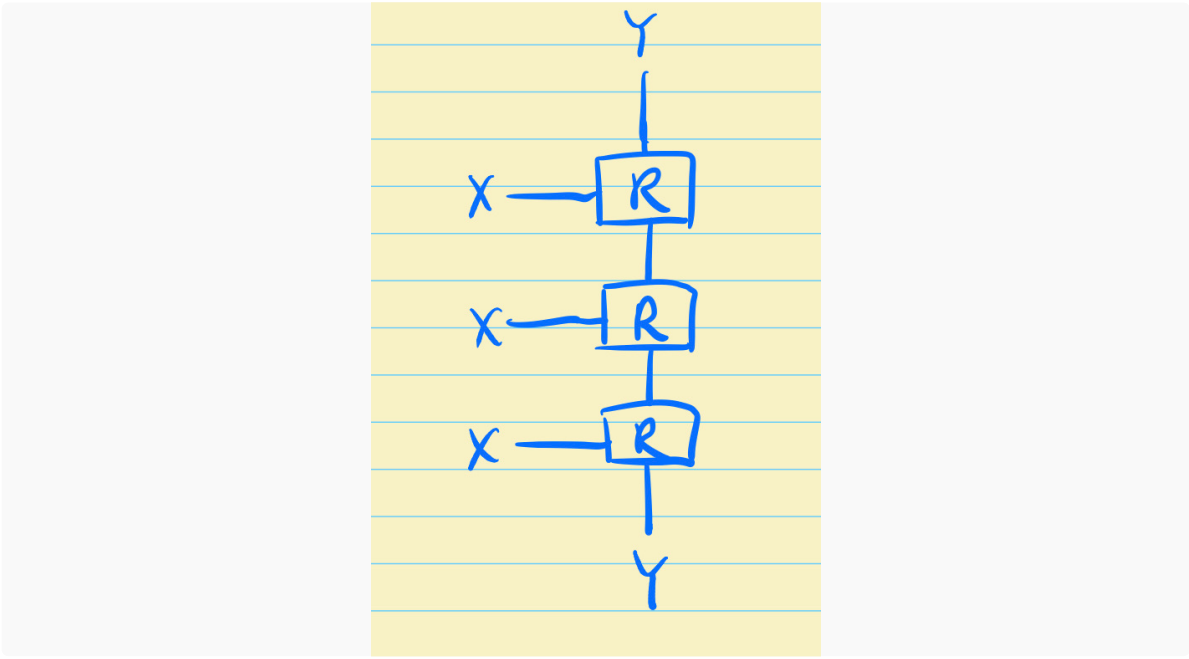
   The diagram of $\text{rdl}_n$ R is shown below



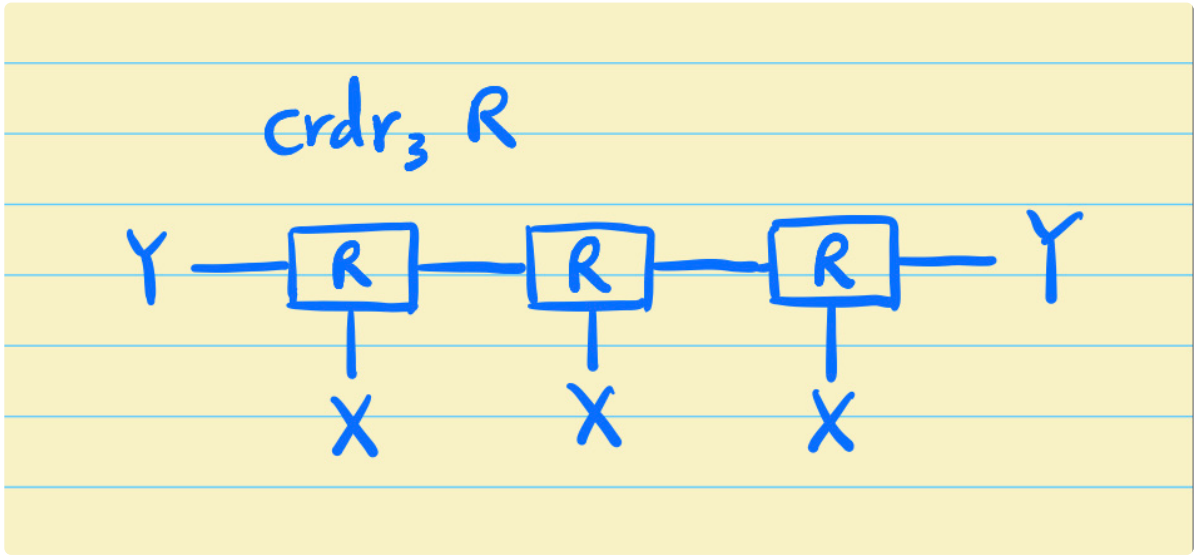

c. Given R: <X, Y> ~ Y, what are the type and the recursive definition of the right reduction combinator, $\text{rdr}_n$ R? Provide a diagram of rdrn R when n=3.

The type of $\text{rdr}_n$ R is $\langle\langle X\rangle_n, Y\rangle$ and its recursive definition would be

```
rdr 1 R = R
rdr n R = fst (apr (n-1))^~1; lsh; snd R; rdr (n-1) R
```




d. Given R: Y ~ <X, Y> and $\text{crdr}_n$ R = $(\text{rdr}_n\ (R^{-1}))^{-1}$ and the type of $\text{crdr}_n$ R is Y ~ $\langle\langle X\rangle_n, Y\rangle$, provide a diagram of crdrn R when n=3.

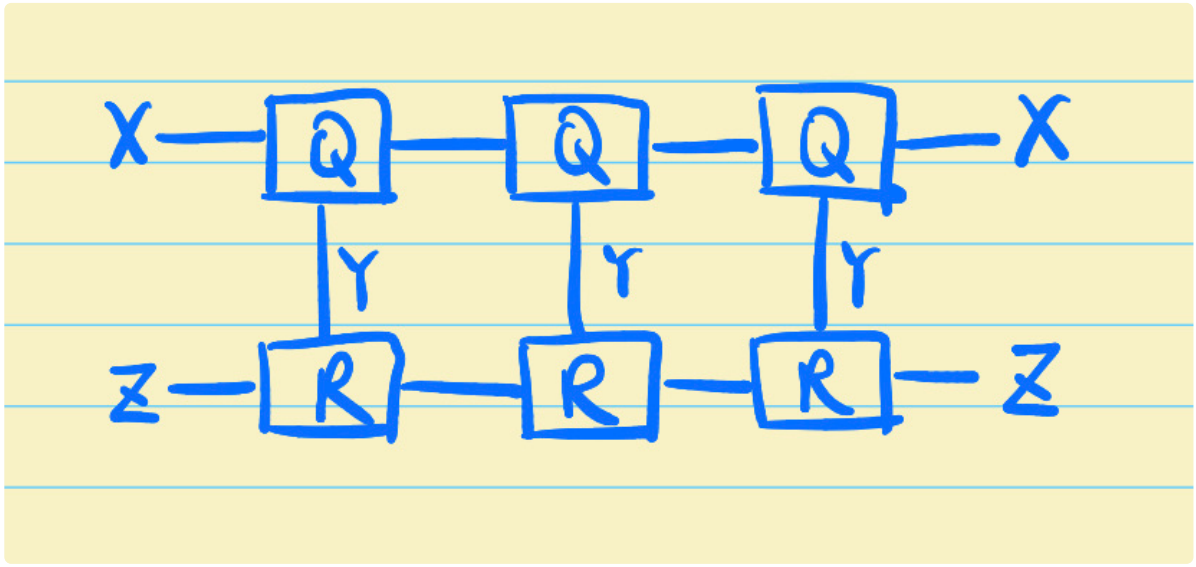



e. Given Q: X ~ <Y, X> and R: <Z, Y> ~ Z, the higher-order function laddern Q R is given by:

ladder$_n$ Q R = (crdr$_n$ Q) ↑ (rdl$_n$ R)

What is the type of ladder$_n$ Q R? Provide a diagram of ladder$_n$ Q R when n = 3.

The type of ladder$_n$ Q R is <Z, X> ~ <Z, X>. Below is a diagram when n = 3




f. Express ladder$_n$ Q R in terms of repeated series composition (no proof is needed). Provide an equation (no proof is needed) which can be used for pipelining ladder$_n$ Q R.

A repeated series composition version of the definition would be

```
ladder n Q R = (Q ↑ R)^n
```

An equation that can pipeline ladder$_n$ Q R would be

```
(Q ↑ R)^n; [D, D]^n = (Q ↑ R; [D, D])^n
```

We use the timeless property of ladder$_n$ Q R to push the delays between each unit

2. An FIR filter can be described by the following equation:

$$y_i = \sum_{j=0}^{j<W} x_{i-j} \cdot w_j$$

where W is the number of taps, x is the input stream, y is the output stream, and w is a sequence of W weights.

a. Implement the FIR filter as a Maxeler MaxJ kernel. Your design should, on every cycle, read one value from an input stream x and write one value to an output stream y. Assume that the weights $w_j$ are available in an array `weights[W]`. Assume that the type of all data elements is defined for you as `floatType`. Do not include hardware to prevent reading outside the stream. Omit the class declaration, imports and constructor declaration.

A possible implementation is shown below

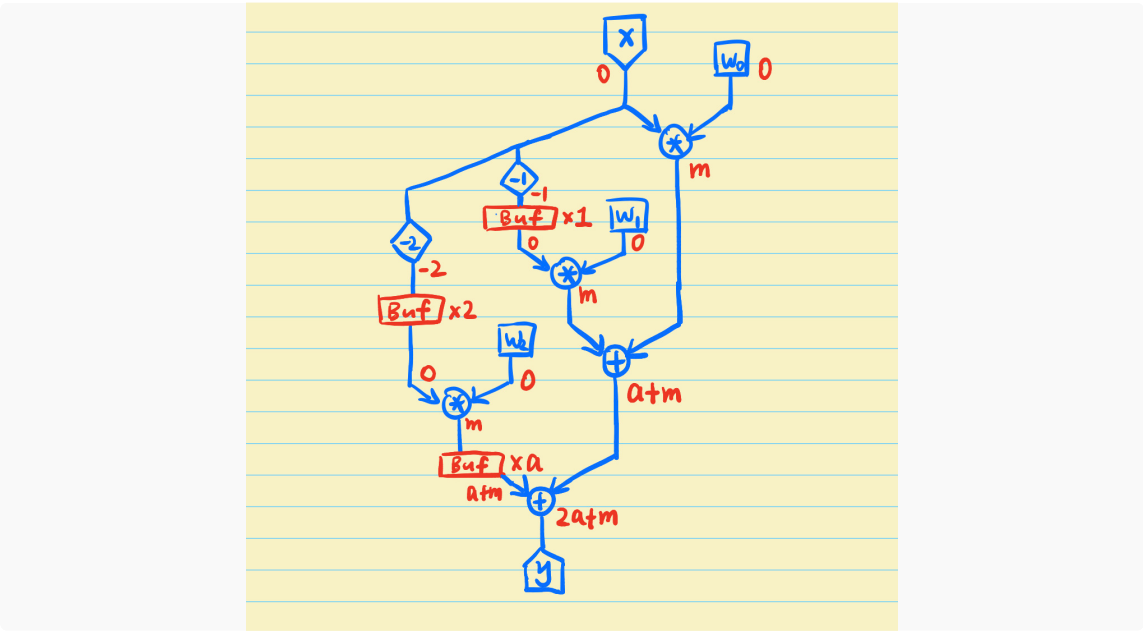
```
DFEVar x = io.input("x", floatType);
DFEVar y = constant.var(0.0, floatType);

for (int j = 0; j < W; j++) {
  y += stream.offset(x, -j) * weights[j]
}

io.output("y", y, floatType)
```

b. Briefly explain how the design in Part a could be modified to prevent reading outside the stream. You do not need to draw a diagram.

We can check the border cases by adding a counter to represent where we are in the stream (e.g. the value of i) and check if it's smaller then j. If it's smaller then we cap the value of j to the counter (e.g. the stopping criteria would be j < counter instead of j < W); if it's bigger then we just use the condition j < W

c. Question for 2c

i) Draw a diagram of the data flow graph of the kernel from Part a for W = 3, after the design has been scheduled by the ASAP algorithm. Make sure you label wires with their calculated latency according to the ASAP scheduling. Assume that adders have latency of a cycles, and multipliers have latency of m cycles.




Notice that the x1, x2, and xa besides the buffer are only indication of how much time/cycle the buffer should hold, not the number of buffers.

ii) Given filter width W, give an expression for the total number of buffers added by the ASAP algorithm.

From the diagram drawn in the previous question, we can deduce that for each j we need one buffer for the offset part and one buffer for the adders waiting multiplication to finish. Hence, the number of buffer would be $max(0, 2(W-1)-1)$. We need $max$ becauase when W = 1 we need 0 buffer.

d. Give an expression for the total runtime of the design in Part a, given that: input stream x is in a memory with bandwidth Bm bytes per second; output stream y is across the PCIe bus with bandwidth Bp bytes per second; cycle time is Tc; the size of floatType is 4 bytes; and the total number of items is N.

The total runtime of the design would be $max(T_{compute}, T_{memory}, T_{bus})$

We compute each term separately and sum up

Assuming that adders take $a$ cycles and multipliers take $m$ cycles. We know that there will be $W$ multiplications and $(W-1)$ additions. However, according to the diagram drawn in question c, the multiplication happens simultaneously. Thus, we can deduce that the total number of cycles needed for the entire computation is $a(W-1)+m$. We can get the total time of $T_{compute}$ by multiplying it with $T_c$:

$$T_{compute} = (a(W-1)+m) \cdot T_c$$

For the input stream, we are using on-chip memory unit and transfering N items each of size 4 bytes. Thus the memory transfering time is

$$T_{memory} = \frac{4N}{B_m}$$

For the output stream, same thing as $T_{memory}$

$$T_{bus} = \frac{4N}{B_p}$$

The total runtime is thus

$$T = max(T_{compute}, T_{memory}, T_{bus}) = max((a(W-1)+m) \cdot T_c, \frac{4N}{B_m}, \frac{4N}{B_p})$$

Notice this solution hasn't accounted for the edge cases where you need to fill the pipeline with several cycles at the beginning.

e. Sketch a design for the FIR filter using feedback. Your design should use a single adder and multiplier. Include control logic using counters to decide when to read and write to input and output streams. Omit hardware to prevent reading outside the stream.

Below is a diagram for this. The circuit in orange colour is the control logic.

The reason we put a mux to select from the weights array is because they are fixed constants. The mux here is a many-to-one multiplexer.