LSU EE 4755 Homework 4 Solution Due: 6 November 2023

Collaboration Rules

Each student is expected to complete his or her own assignment. It is okay to work with other students and to ask questions in order to get ideas on how to solve the problems or how to overcome some obstacle (be it a question of Verilog syntax, how a part of the problem might be solved, etc.) It is also acceptable to seek out digital design resources for help on Verilog, digital design, etc. It is okay to make use of AI LLM tools such as ChatGPT to answer these questions. Just don't trust the answers. (Do not assume LLM output is correct. Treat LLM output the same way one might treat legal advice given by a lawyer character in a movie: it may sound impressive, but it can range from sage advice to utter nonsense.)

After availing oneself to these resources **each student is expected to be able to complete the assignment alone**. Test questions will be based on homework questions and **the assumed time needed to complete the question will be for a student who had solved the homework assignment on which it was based**.

Helpful Examples

See the simple model slides for material on computing cost and delay, and also for a list of some sample problems. Also see 2022 Homework 3.

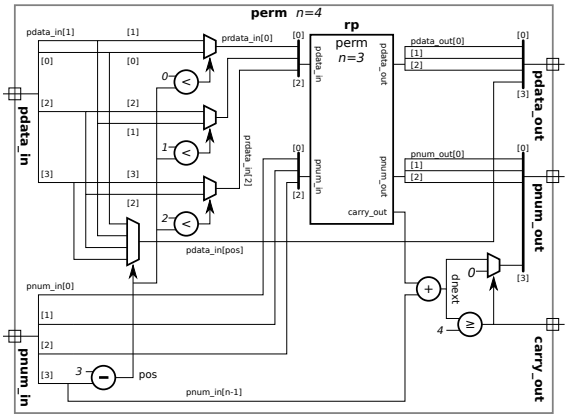
Permutation Module

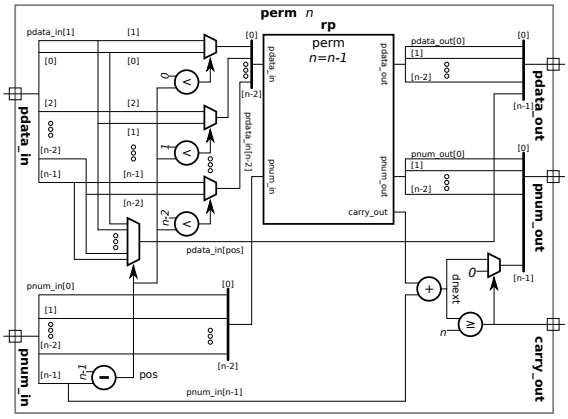
This assignment is based on the solution to Homework 3, the recursive permutation module `perm`, and the solution to Midterm Exam Problem 1, the inferred hardware for the permutation module. See Homework 3 for details on what the permutation module does. Appearing below is the Homework 3 solution with some comments removed. For the unabridged version visit `https://www.ece.lsu.edu/koppel/v/2023/hw03-sol.v.html`.

```
module perm
  #( int w = 8, n = 20, wd = $clog2(n) )
   ( output uwire [w-1:0] pdata_out[n],    output uwire [wd-1:0] pnum_out[n],
     output uwire carry_out,
     input uwire [w-1:0] pdata_in[n],       input uwire [wd-1:0] pnum_in[n] );

   if ( n == 1 ) begin

      assign pdata_out[0] = pdata_in[0];
      assign carry_out = 1;
      assign pnum_out[0] = 0;

   end else begin

      //  Set pos to the position of the element to be moved.
      uwire [wd-1:0] pos = n - 1 - pnum_in[n-1];

      //  Copy the element at position pos to position n-1 in the output.
      assign pdata_out[n-1] = pdata_in[pos];

      //  Prepare an array of n-1 elements and set to ..
      //  .. the elements of pdata_in except for the element at pos.
      uwire [w-1:0] prdata_in[n-1];
      for ( genvar i=0; i<n-1; i++ )
        assign prdata_in[i] =  i < pos  ?  pdata_in[i]  :  pdata_in[i+1];

      // Recursively instantiate perm.
      uwire co;
      perm #(w,n-1,wd) rp( pdata_out[0:n-2], pnum_out[0:n-2], co,
                           prdata_in, pnum_in[0:n-2] );

      //  Compute a tentative next value of digit n-1.
      uwire [wd-1:0] dnext = pnum_in[n-1] + co;

      //  Determine whether there is a carry.
      assign carry_out = dnext >= n;

      //  Set the next value of digit n-1 based on whether there is a carry.
      assign pnum_out[n-1] = carry_out ? 0 : dnext;

   end

endmodule
```

Permutation Module Inferred Hardware

Midterm Exam Problem 1 asked for the inferred hardware for the `perm` module instantiated with `n=4`. The solution appears below on the left. For this assignment the inferred hardware for a non-specific value of `n` will be needed, that is shown on the right.





There's no need to squint, the diagrams appear again in larger size at the end of this assignment. Also, SVG source for these modules are at `https://www.ece.lsu.edu/koppel/v/2023/mt-p1-sol.svg` and `https://www.ece.lsu.edu/koppel/v/2023/hw04-perm-gen.svg`.

Problem 1: Compute the cost and delay of the following arithmetic hardware from the `perm` module. Assume that ripple units are used for addition, subtraction, and comparison.

(*a*) Compute the cost and delay of the hardware computing `pos = n - 1 - pnum_in[n-1]` in terms of w_d, the value of parameter `wd`. Optimize for constants, including `n`.

☑ Cost of hardware in terms of w_d. ☑ Delay of hardware in terms of w_d.

☑ Optimize for constants, don't confuse elaboration-time computation with computation hardware.

The hardware is a subtractor with constant input `n-1` and non-constant input `pnum_in[n-1]`. The exact cost of an adder would depend on the value of `n-1`, for example if `n-1=0` the cost would be zero. But for a subtractor we set the carry in to 1 and so with a constant input the cost is the cost of w_d BHAs. So the cost is $4w_d\,\mathrm{u_c}$ (see the midterm exam solution for details). (The cost can be reduced to $3w_d\,\mathrm{u_c}$ by splitting the XOR gate in each BHA.)

The delay is one unit per bit (because the delay from `ci` to `co` of a BHA is just one gate delay), for a total delay of $w_d\,\mathrm{u_t}$.

(*b*) Compute the cost and delay of the hardware computing `dnext = pnum_in[n-1] + co` in terms of w_d, the value of parameter `wd`. Optimize for constants and for the size of `co`. Assume in this problem that `pnum_in` and `co` arrive at $t = 0$.

☑ Cost of hardware in terms of w_d. ☑ Delay of hardware in terms of w_d.

☑ Optimize considering the size of `co`. ☑ Optimize for constants, don't confuse elaboration-time computation with computation hardware.

The `dnext` value is computed by adding a 1-bit value, `co` to `pnum_in[n-1]`. So this is equivalent to an adder with a constant input, 0, with the carry-in connected to `co`. The cost then will be $4w_d\,\mathrm{u_c}$ (or $3w_d\,\mathrm{u_c}$) and the delay $w_d\,\mathrm{u_t}$.

(*c*) Compute the cost and delay of the hardware described by these lines:

```
uwire [wd-1:0] dnext = pnum_in[n-1] + co;
assign carry_out = dnext >= n;
```

Assume in this problem that `co` and `pnum_in` arrive at $t = 0$. The cost, of course, includes the cost of computing `dnext` in the previous part. The delay must be computed taking both lines into account.

☑ Cost of hardware in terms of w_d. ☑ Delay of `co` in terms of w_d.

☑ Optimize considering the size of `co`. ☑ Optimize for constants, don't confuse elaboration-time computation with computation hardware.

The cost of the hardware to compute `carry_out` is the cost of the hardware to compute `dnext`, $4w_d\,\mathrm{u_c}$, plus the cost of the comparison module. A comparison module can be constructed from a subtractor with the difference bits eliminated. For two non-constant w-bit inputs the cost would be $4w\,\mathrm{u_c}$, but in this case one input is constant dropping the cost to just $w_d\,\mathrm{u_c}$. The total cost is $[4w_d + w_d]\,\mathrm{u_c}$. As with the subtractor, the carry chain delay is one gate per bit so the delay of the comparison built using a ripple circuit is $w_d\,\mathrm{u_t}$. Because the adder and the ripple circuit are cascadable the total delay is $[2 + w_d]\,\mathrm{u_t}$, where the $2\,\mathrm{u_t}$ is the time for the adder to compute the first bit of the sum.

There are more problems on the next page.

Problem 2: In this problem consider the multiplexors with inputs connecting to `pdata_in`. (In the diagram they are the multiplexors on the upper-left including the 2-input muxes the n-input mux.) Call these the pdata multiplexors. In the solutions to the parts below use w for the value of parameter `w` and w_d for the value of parameter w_d.

(a) Compute the cost of the pdata multiplexors for a module instantiated at size $n = N$ including only the hardware in the `n=N` instantiation, not in the recursive instantiations. The answer should be in terms of N and w. *Hint: this is easy.*

☑ Cost of the pdata multiplexors at one level in terms of N, w, and (if needed) w_d.

In all of the multiplexors the inputs are w bits each. There are $N-1$ 2-input multiplexors and one N-input mux. The cost of a 2-input mux is $3w\,\mathrm{u_c}$, and there are $N-1$ of them so their cost is $3w(N-1)\,\mathrm{u_c}$. The cost of an N-input, w-bit mux is $3w(N-1)\,\mathrm{u_c}$, which interestingly is the same as the total cost of the 2-input multiplexors. The $\boxed{\text{total cost is } 6w(N-1)\,\mathrm{u_c}}$.

(b) **This is important. Expect to expend brain energy. Don't skip.** Compute the total cost of the pdata multiplexors for an instantiation at size $n = N$ including the recursive instantiations all the way down. The answer should be in terms of N and w.

☑ Cost of the pdata multiplexors including recursive instantiations in terms of N, w, and (if needed) w_d.

The cost at level n, based on the previous part (but using lower-case n) is $6w(n-1)\,\mathrm{u_c}$. The cost of the $n = 1$ instantiation is zero because all that module does is connect its inputs to its outputs. So the total cost of instantiations from N to 2, which we'll call $C(N)$, is $\sum_{n=2}^{N} 6w(n-1)\,\mathrm{u_c}$. Proceeding step by step for the benefit of those who are rusty, even on one of the more storied finite sums

$$\begin{aligned} C(N) &= \sum_{n=2}^{N} 6w(n-1)\,\mathrm{u_c} \\ &= 6w\,\mathrm{u_c} \sum_{n=2}^{N} (n-1) \\ &= 6w\,\mathrm{u_c} \sum_{n=1}^{N-1} n \\ &= 6w\,\mathrm{u_c} \frac{N(N-1)}{2} \\ &= 3wN(N-1)\,\mathrm{u_c} \end{aligned}$$

For those scanning for boxes, the total $\boxed{\text{cost is } 3wN(N-1)\,\mathrm{u_c}}$.

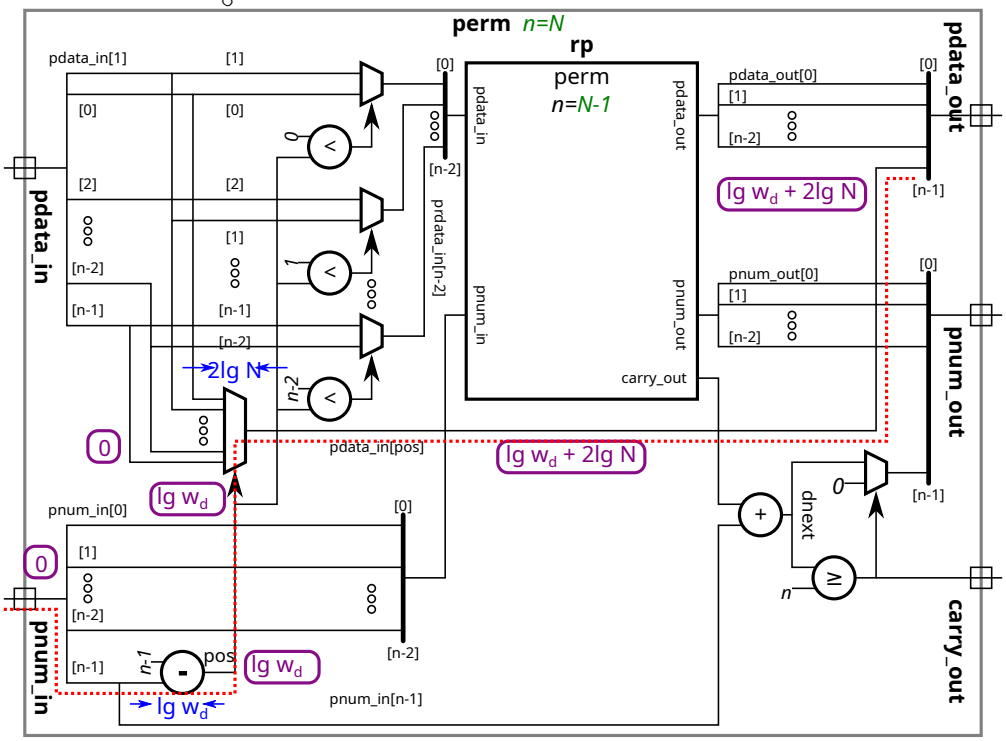
Problem 3: In this problem compute delays for `pdata_out` and `pnum_out`. In the solutions use d for the value of parameter `wd`. **This is also important and even more interesting. Expect to expend brain energy. Don't skip.**

(a) Assume that the delay of the subtractors computing `pos` is $\lg w_d$, where w_d is the value of parameter `wd`. (Note that $\lg w_d$ is not an answer to Problem 1.) Further, suppose the delay of the less-than units providing a select signal to the 2-input pdata multiplexors is zero. Using these assumptions compute the delay of the first and last elements of `pdata_out` for an instantiation at `n=N` and show the critical path. The delay should be in terms of N and w_d. To solve this problem it might be helpful to draw two instantiation levels to help find the critical path.

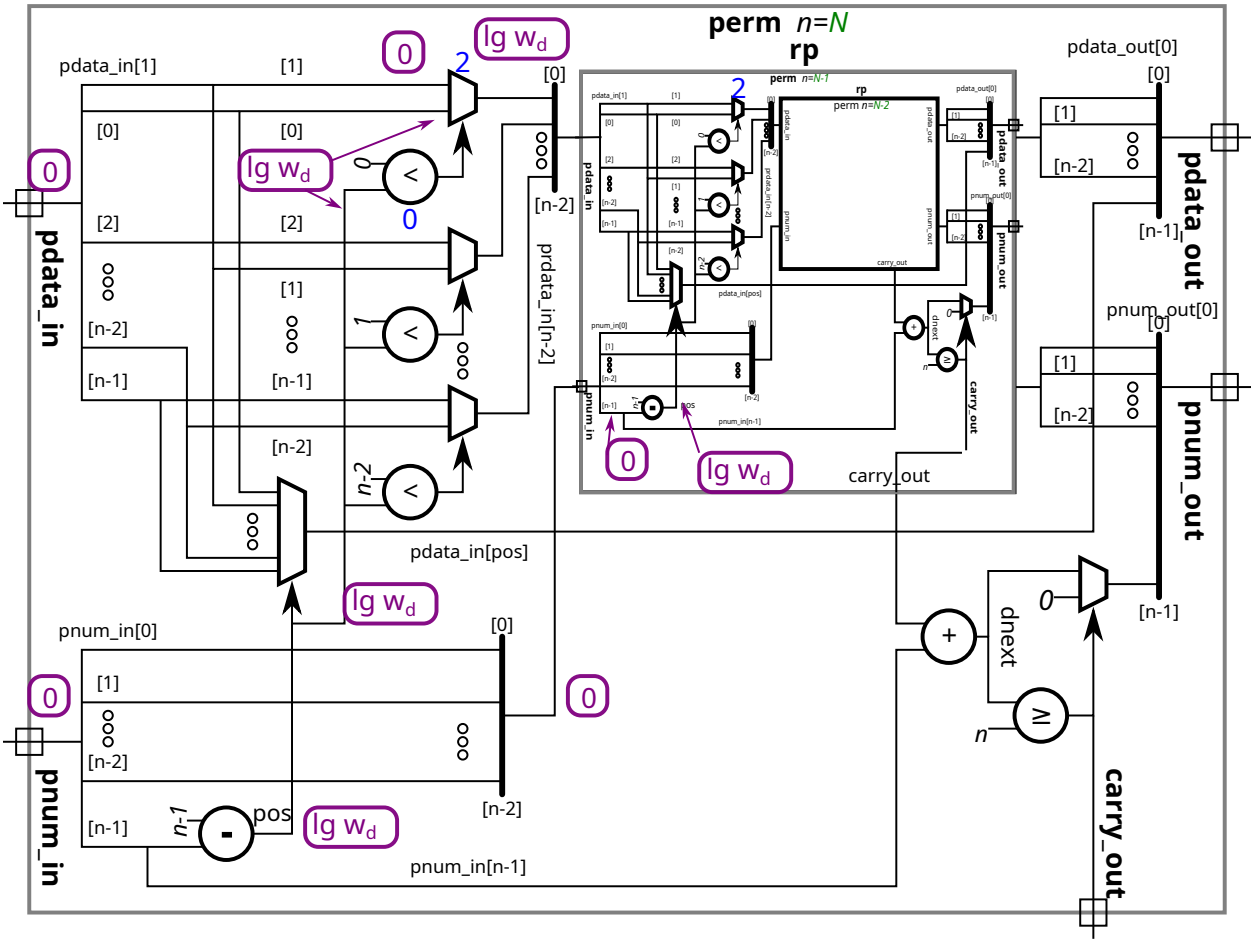
☑ Delay of `pdata_out[0]` in terms of N and w_d accounting for recursive instantiations. ☑ Show critical path.

☑ Delay of `pdata_out[N-1]` in terms of N and w_d accounting for recursive instantiations. ☑ Show critical path.

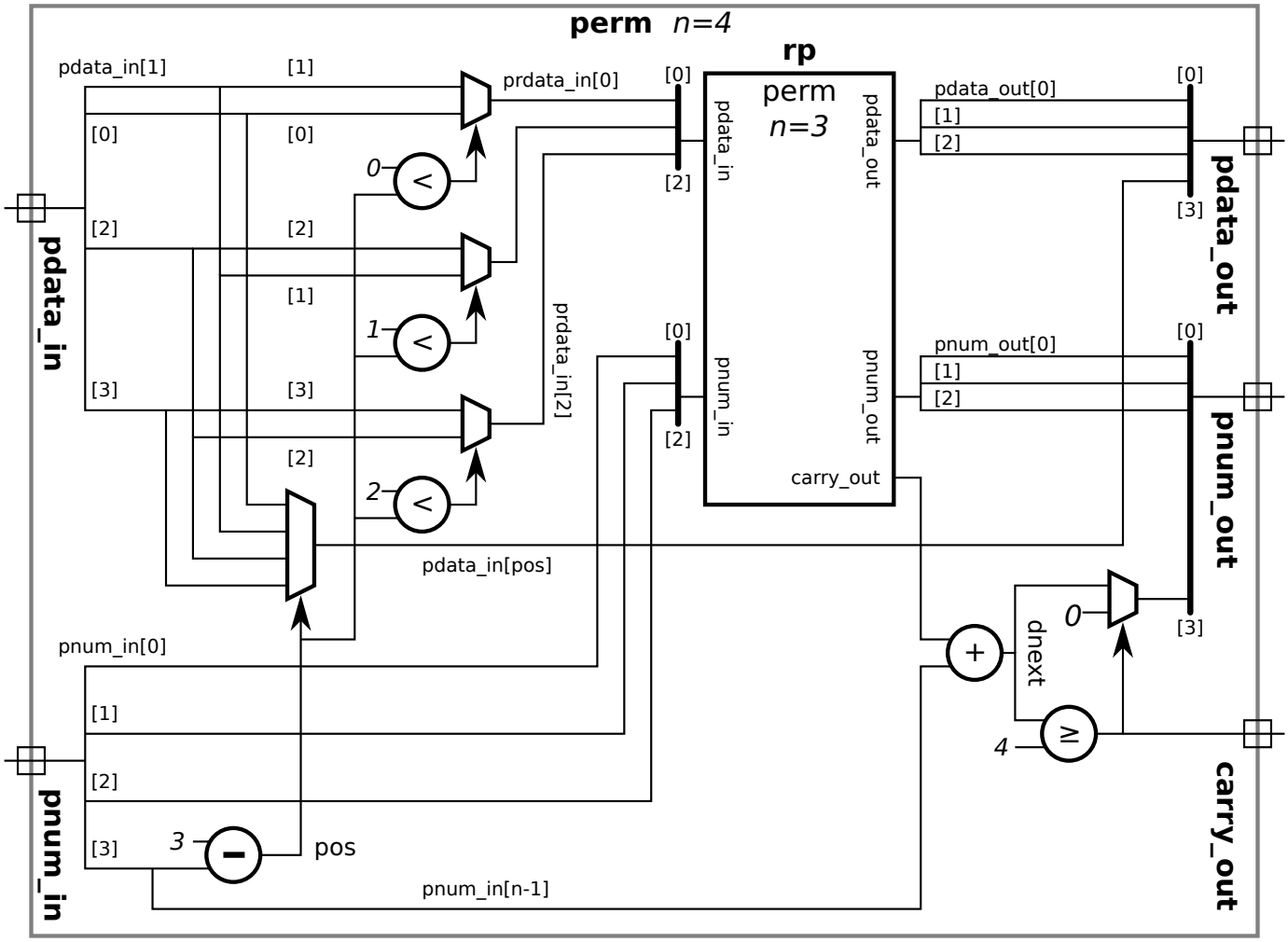
The easier of these to solve is `pdata_out[N-1]` because its value is computed without using data from a recursive instance. As everyone reading this should know or at least learn now and not forget, the delay of an N-input multiplexor is $2\lceil \lg N \rceil \, \mathrm{u_t}$. For this problem we were to assume that the subtractor computing `pos` has a delay of $\lg w_d$. We can safely assume that the inputs to the $n = N$ instance arrive at $t = 0$, and so `pos` (the output of the subtractor) arrives at $t = \lg w_d$. Therefore the delay of `pdata_out[N-1]` is $\lceil \lg(w_d) + 2\lceil \lg N \rceil \rceil \, \mathrm{u_t}$. The delays, arrival times, and critical path are shown in the diagram below.



To compute the delay of output `pdata_out[0]` we need to find the path it will take through the recursive instantiations. The illustration below shows the top-level instantiation, for `n=N` and one level down, for `n=N-1`. To understand the solution it is important that you pay attention to the arrival times of signals, shown in circled purple numbers and expressions. For the top-level instantiation the arrival time of all inputs is at $t = 0$. **But**, for the $n = N-1$ instantiation notice that some signals arrive at $t = 0$, such as `pnum_in`, while `pdata_in` arrives later, at $t = \lceil \lg(w_d) + 2 \rceil \, \mathrm{u_t}$. (The time unit, $\mathrm{u_t}$, is not shown on the illustrations.) The fact that `pnum_in` to the $n = N - 1$ instantiation arrives at $t = 0$ means that the select signals to the 2-input multiplexors arrives at $t = \lg w_d$ at *all* instantiations. The `pdata_in` inputs are different. At $n = N$ they arrive at $t = 0$, while for $n = N - 1$ they arrive at $\lceil \lg(w_d) + 2 \rceil \, \mathrm{u_t}$. In the $n = N - 1$ instance consider the 2-input multiplexors. Whereas at $n = N$ the data inputs arrived before the select signal, at $n = N - 1$ the data inputs arrive *after* the select signal. That means that the arrival time at the outputs of the 2-input multiplexors at $n = N - 1$ is at $\lceil \lg(w_d) + 2 + 2 \rceil \, \mathrm{u_t} = \lceil \lg(w_d) + 4 \rceil \, \mathrm{u_t}$. Each further level down adds just 2 units of delay. At level n input `pdata_in[n-1]` does not go through the recursive instantiation. But input `pdata_in[0]` goes all the way down to $n = 1$, and at each level before $n = 1$ another 2 units are added. (The delay, remember, at $n = 1$ is zero.) Therefore the total delay down to $n = 1$ is $\lceil \lg(w_d) + 2(N - 1) \rceil \, \mathrm{u_t}$. The `pdata_out` output of the recursive instance connects directly to the `pdata_out` of the containing instance, and so no further delay is added. Therefore $\boxed{\text{the total delay is } \lceil \lg(w_d) + 2(N - 1) \rceil \, \mathrm{u_t}}$.



SVG source for the module below is at
`https://www.ece.lsu.edu/koppel/v/2023/mt-p1-sol.svg`.



SVG source for the module below is at
`https://www.ece.lsu.edu/koppel/v/2023/hw04-perm-gen.svg`.