GEORGIA INSTITUTE OF TECHNOLOGY

School of Electrical and Computer Engineering

ECE-6276-A DSP Hardware System Design (Spring 2025)

Lab 4: Distributed Arithmetic

Assigned Date: Wednesday, Apr 2, 2025

Due Date : 23:55, Friday, Apr 11, 2025

---

1 Introduction

In this lab, we will be designing **signed** *Distributed Arithmetic* (DA). DA is an important FPGA technology and is extensively used in computing the sum of products without using a multiplier:

$$y = \langle \boldsymbol{c}, \boldsymbol{x} \rangle = \sum_{n=0}^{N-1} c[n] \times x[n] \tag{1}$$

Convolution, correlation, matrix multiplication can all be formulated as such form of dot products. The computation of DA usually takes N cycles (given the bitwidth of inputs is N bit). A prerequisite for a DA design is that the filter coefficients $c[n]$ should be known a priori.

A $(B+1)$-bit signed number can be represented as:

$$x[n] = -2^B \times x_B[n] + \sum_{b=0}^{B-1} x_b[n] \times 2^b \quad \text{with} \quad x_b[n] \in \{0, 1\}, \tag{2}$$

where $x_b[n]$ denotes the b^{th} bit of $x[n]$, and $x_B[n]$ the sign bit. Combining Eq. 1 and 2 together, we have:

$$\begin{aligned}
y &= \sum_{n=0}^{N-1} c[n] \left(-2^B \times x_B[n] + \sum_{b=0}^{B-1} x_b[n] 2^b \right) \\
&= c[0] \left(-x_B[0]2^B + x_{B-1}[0]2^{B-1} + \cdots + x_0[0]2^0 \right) \\
&\quad + c[1] \left(-x_B[1]2^B + x_{B-1}[1]2^{B-1} + \cdots + x_0[1]2^0 \right) \\
&\quad \vdots \\
&\quad + c[N-1] \left(-x_B[N-1]2^B + x_{B-1}[N-1]2^{B-1} + \cdots + x_0[N-1]2^0 \right) \\
&= -(c[0]x_B[0] + c[1]x_B[1] + \cdots + c[N-1]x_B[N-1])2^B \\
&\quad + (c[0]x_{B-1}[0] + c[1]x_{B-1}[1] + \cdots + c[N-1]x_{B-1}[N-1])2^{B-1} \\
&\quad \vdots \\
&\quad + (c[0]x_0[0] + c[1]x_0[1] + \cdots + c[N-1]x_0[N-1])2^0
\end{aligned}$$

Therefore,

$$y = -2^B \left(\sum_{n=0}^{N-1} c[n] \times x_B[n] \right) + \sum_{b=0}^{B-1} 2^b \left(\sum_{n=0}^{N-1} c[n] \times x_b[n] \right) \tag{3}$$

Since $x_b[n] \in \{0, 1\},\ b \in [0 \dots B]$, the SOP $\sum_{n=0}^{N-1} c[n] \times x_b[n]$ is just the sum of either taking $c[n]$ or not. Therefore, we can make a LUT/ROM where each entry is a different combination of $c[0]$ to $c[N-1]$.

The data flow (Architecture) of DA looks like Figure 1, and the input, output ports are shown in Figure 2.

- `clk` ⇒ Clock signal. We use rising edge for flops.
- `rst` ⇒ Reset signal. Note that the sequential elements that are used in your design should have **synchronous** active high reset, i.e., if the reset is pulled high, the flops should be reset at the next rising edge of `clk`.
- `in_valid` ⇒ `in_valid` indicates the input data are valid. There are 4 input vectors for each valid set.
- **The computation to be done is:**

  ```
  data_out = data_in_0 * coef_0 + data_in_1 * coef_1
           + data_in_2 * coef_2 + data_in_3 * coef_3,
  ```

  where all `data_in_*` and `coef_*` are 4-bit signed numbers. The coefficients are as follows:

  ```
  coef_0 = 7, coef_1 = 3, coef_2 = -8, coef_3 = -5
  ```

- The `next_in` indicates whether the design is ready to take the next input data. If `next_in` is high, then it is ready. Since your design may take multiple cycles to finish the computation of one valid set of inputs, it is your responsibility to handle the `next_in`. From the view of the testbench, it will hold a valid set of input data when it sees `next_in` is logic low. When it sees `next_in` is logic high, at the next cycle it will update the input with a new set of data (either valid or invalid).
- `out_valid` ⇒ Valid signal for the output. When `out_valid = 1`, the outputs `data_out` are valid.
- For simplicity, there is no ready signal `next_out` on the output side of the design. That means there is no back pressure from the testbench. The testbench will only sample the valid output data and write them into the `output.txt` file.
- The controller will select which group of bits as address to look up the LUT.
- Note that Figure 1 may be incomplete. It is up to your design whether and where to add additional pipeline registers (Depending on the design, there may be a 1-cycle difference with the reference files).

The LUT/ROM can be implemented in various writing styles:

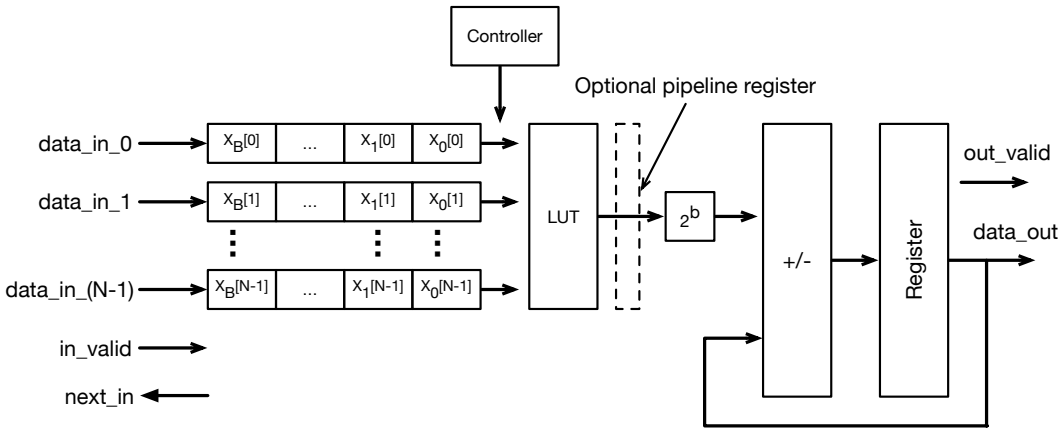

Figure 1: Data flow of a signed DA system

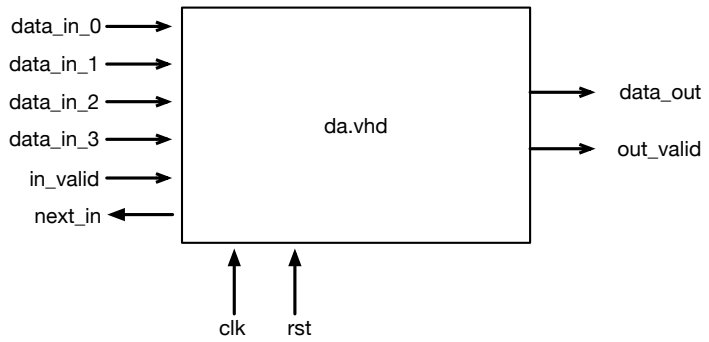

Figure 2: Input, output ports of the design

- Use a `case` statement.
- Use `constant rom: array (0 to x) of type := (xxxx, xxxx, xxxx, etc);`. In this case, an initial value is necessary.

The grading will take into consideration the functional correctness, utilization (area), timing, and power of the design. Three re-submissions are allowed for this lab.

2 Instructions

1. Before writing RTL code, think carefully how you are going to complete your data flow architecture and figure out the possible timing diagrams of your design.
2. Think of the question listed in Section 3 if you can update your design in an area-efficient way.
3. Complete your design in only *ONE* file: `da.vhd`. **NOTE: You are NOT allowed to use multiplication in this lab, which means you cannot use assignments like** `a <= b * c;`.
4. Run the simulation for `da.vhd` using the provided testbench `tb_da.vhd` to match your output file `output.txt` with the reference `output_ref.txt`.
5. Run the synthesis and implementation of your design `da.vhd` in Vivado using the provided constraint file `constraints.xdc`.

3 Deliverables

1. Create a PDF report containing:
   - Tables of
     (a) Resources (Only "Slice Logic", "IO and GT Specific", "Primitives") from `da_utilization_placed.rpt`
     (b) Power (Dynamic and Static) from `da_power_routed.rpt`
     (c) Worst Negative Slack ("Design Timing Summary") from `da_timing_summary_routed.rpt`
   - (20% of grade) Answer the question: Is there an area-efficient way to do the left-shift which is after the output of LUT?
     (a) (5%) If yes, draw the updated block diagram starting from the LUT on the left to the output `data_out` on the right.
     (b) (5%) Write down the mathematical recurrence relation induced from Eq. 3 that proves your thought above.
     (c) (10%) Why the original left-shift is not a good idea and why the updated one is area-efficient? (Answer should not exceed 4 sentences)

   The name of the PDF should be of the form `lab4_firstname_lastname.pdf`, e.g., `lab4_junyoung_hwang.pdf`.
2. (70% of grade) The *ONLY* completed design file `da.vhd`. The 70% grading will be based on:
   - (50%) Functional correctness of your design.
   - (20%) Sampling period and area of your design.
   - (extra 5% within a total of 100%) If implemented in an area-efficient left-shift way, extra 5% will be given.
3. The simulated output file `output.txt` in the "run" folder.
4. The simulated output file `output_cycle.txt` in the "run" folder.
5. (10% of grade) The implemented area (utilization), power, and timing reports, namely,
   - `da_utilization_placed.rpt`
   - `da_power_routed.rpt`
   - `da_timing_summary_routed.rpt`

Move all of these files into a folder called `lab4_firstname_lastname_gtID`. Zip the folder and upload the archive `lab4_firstname_lastname_gtID.zip` on Canvas (e.g., `lab4_junyoung_hwang_123456789.zip`). The first name and last name should be all in lower case. **Please strictly follow this naming convention. Otherwise, my script will not work, and you might get points deducted.**