Lab 06: Arithmetic Logic Unit

Arturo Salinas-Aguayo
CSE 2301: Principles and Practice of Digital Logic Design
Dr. Mohammad Khan, Section 003L-1248
Electrical and Computer Engineering Department

College of Engineering, University of Connecticut
Coded in LaTeX

Theory

Multiplication Circuits and Output Size Calculation

A binary multiplier is a combinational circuit that multiplies two binary numbers to produce a product. The circuit performs binary multiplication by breaking down the multiplication process into a series of addition operations, similar to how multiplication is done manually with partial products in decimal.

To create a binary multiplier for any input size:

- **Partial Product Generation:** Each bit of the multiplier (second operand) is ANDed with each bit of the multiplicand (first operand). For an m-bit multiplicand and an n-bit multiplier, this creates $m \times n$ partial products.
- **Summing Partial Products:** These partial products are then aligned according to their bit significance (like shifting in decimal multiplication) and summed to produce the final result. The addition process can be achieved using a series of adders, such as half-adders and full-adders, to accumulate each partial product.

Output Size Calculation:

The maximum size of the output in bits is crucial for ensuring that the circuit can handle any possible product without overflow. When multiplying two binary numbers:

- If you have an m-bit multiplicand and an n-bit multiplier, the resulting product will have a maximum of $m + n$ bits. This is because multiplying two numbers potentially doubles the value size, which requires more bits to represent accurately.
- For example:
  - A 2x2 binary multiplier (multiplying two 2-bit numbers) will produce a 4-bit product, as $2 + 2 = 4$
  - A 3x4 binary multiplier (multiplying a 3-bit number by a 4-bit number) will produce a 7-bit product, as $3 + 4 = 7$

Thus, to determine the output size of any binary multiplication circuit, you add the bit lengths of the two operands, ensuring enough bits to represent the highest possible result.

Use of 74153 Chips with Shared Selectors

The 74153 chip is a dual 4-to-1 multiplexer (MUX) IC, meaning it contains two independent 4:1 MUX circuits within a single chip. However, these multiplexers share the same set of selector inputs, meaning they operate in tandem, controlled by the same selector bits.

Benefit of Shared Selector Bits:

In our case, having shared selectors is beneficial because:

- **Synchronized Selection:** Shared selectors ensure that both multiplexers select their respective inputs simultaneously. This feature is useful when we need two related but separate signals to be processed in parallel, such as in addition circuits where similar operations are performed on corresponding bits of different operands.

- **Simplified Control Logic:** By using the same selector bits for both MUXes, the design complexity is reduced, as only one set of control signals is needed. This reduces the number of lines required for control inputs and makes the overall circuit more compact and less complex.

Modifications for Independent Selectors:

If we wanted each MUX to have different selectors (operate independently), we would need to modify the circuit as follows:

- **Add Separate Selector Lines:** Provide separate control inputs for each MUX, which would require additional control logic. This could involve adding extra multiplexers or decoders to ensure that each MUX has its own dedicated selector input.
- **Increase Chip Complexity:** This change would increase the circuit's complexity, requiring more wires and potentially additional ICs to handle the independent selectors, which could lead to higher power consumption and a larger physical layout.

In summary, shared selectors on the 74153 chip reduce circuit complexity and are advantageous in applications where synchronized selection is desired, making it a suitable choice for many standard combinational circuits.

Discussion

A 4x4 Bit Combinational Multiplier?

In my realized 4x4 bit multiplier circuit, two 4-bit binary numbers, $A = \{A3, A2, A1, A0\}$ and $B = \{B3, B2, B1, B0\}$, are multiplied using a combination of AND gates for generating partial products and 4-bit full adders for summing these products.

The maximum number this circuit can calculate is $15 \times 15 = 225$, which fits within 8 bits. This is because the bit size of the product equals the sum of the bit sizes of the multiplicand and multiplier.

Each bit of A is ANDed with each bit of B to generate 16 partial products. These partial products are fed into the **PPBus**, which is then connected to three 4-bit full adders to sum the partial products and produce the final 8-bit product $P = \{P7, P6, P5, P4, P3, P2, P1, P0\}$

The PPBus is organized as follows:

$$\text{pp}[0..15] = \{pp0, pp1, pp2, \ldots, pp15\}$$

where each $pp_{ij} = A_i \cdot B_j$.

The PPBus is a nickname for a data bus representing 15 bits of partial products. The partial products are generated by ANDing each bit of the multiplicand with each bit of the multiplier. The PPBus is then connected to the 4-bit full adders to sum the partial products and produce the final product. The final product P is given by:

$$P = \{P7, P6, P5, P4, P3, P2, P1, P0\}$$

The LSB of the first partial product flows directly to $P0$. The LSB of each adder's sum output contributes to a bit of the final product, while the higher bits are cascaded into the next

adder's inputs until all partial products are summed. The key to this major simplification is actually within the first full adder, by setting A3 to 0 a beautiful, cascading circuit can be made.

ALU Circuit Design Using 74153 and 74157 Multiplexers

1. 74157 Quad 2-1 MUX (Middle Multiplexer in Diagram)
This multiplexer is used for choosing between the AND and OR operations:

- **Inputs:**
  - $D0, C0, B0, A0$: Inputs from the AND operation outputs (Q_0 to Q_3).
  - $D1, C1, B1, A1$: Inputs from the OR operation outputs (R_0 to R_3).
- **Control:**
  - SEL: Control line that selects whether the AND or OR results are chosen based on the operation being performed.
- **Outputs:**
  - $Z0, Z1, Z2, Z3$: Outputs the result of either the AND or OR operations based on the SEL line.

2. Top 74153 Dual 4-1 MUX (Upper Multiplexer in Diagram)
This multiplexer is used for handling the addition outputs, which may have more than 4 bits:

- **Inputs:**
  - $A0, A1, A2, A3$: Inputs for one part of the addition outputs (lower bits).
  - $B0, B1, B2, B3$: Inputs for the other part of the addition outputs (upper bits, including the carry if present).
  - $C0, C1, C2, C3$ and $D0, D1, D2, D3$: Could be additional inputs for handling different scenarios or further operations.
- **Control:**
  - $S0, S1$: Control lines that choose which group of addition results to use.
- **Outputs:**
  - $Y0, Y1, Y2, Y3$: Selected outputs from the addition operation results.

3. Bottom 74153 Dual 4-1 MUX (Lower Multiplexer in Diagram)
This multiplexer is used for handling the 4x4 multiplication outputs, which can have up to 8 bits:

- **Inputs:**

  - $A0, A1, A2, A3$: Lower four bits of the multiplication result.
  - $B0, B1, B2, B3$: Upper four bits of the multiplication result.
  - $C0, C1, C2, C3$ and $D0, D1, D2, D3$: Additional inputs for potential future expansion or handling other conditions.

- **Control:**
  - $S0, S1$: Control lines to select the appropriate set of results from the multiplication operation.

- **Outputs:**
  - $Y0, Y1, Y2, Y3$: Outputs the selected 4-bit section from the multiplication result.

General Configuration

- **Control Logic:** The control lines ($SEL, S0, S1$) for each multiplexer need to be driven by logic that decodes the operation type. This could be based on opcode or function select lines within the ALU's control architecture.
- **Enable Lines:** All multiplexers should have their enable lines (EN for 74157, G for 74153) properly controlled to ensure that outputs are enabled only during valid operation cycles.

This configuration allows the ALU to dynamically select the appropriate operation results to output, handling various widths and ensuring that the results are directed through the correct paths.

Example 1 ***8-1 MUX Circuit***

To implement this function using an 8:1 multiplexer:

1. ***Identify Minterms:*** *The minterms for the function* $F = \sum m(0, 1, 6, 7)$ *represent binary combinations of* CBA*:*
   - $m(0) = 000$
   - $m(1) = 001$
   - $m(6) = 110$
   - $m(7) = 111$
2. ***Setup Multiplexer Inputs:*** *An 8:1 MUX uses three select inputs (here* C, B, A*) to choose from eight data inputs (*D_0 *through* D_7*). Assign each data input as follows:*
   - $D_0 = 1$ *(since* $m(0) = 000$*)*
   - $D_1 = 1$ *(since* $m(1) = 001$*)*
   - $D_6 = 1$ *(since* $m(6) = 110$*)*

- $D_7 = 1$ *(since* $m(7) = 111$*)*
- *All other inputs* (D_2, D_3, D_4, D_5) *are set to 0.*

3. ***Function Output:*** *With this setup, the 8:1 MUX selects the high (1) outputs for minterms 0, 1, 6, and 7, producing the desired function* F*.*

Example 2 ***4-1 MUX with XOR***

Imagine a 4-1 MUX that gets inputs from A_0 *through* A_3*, controlled by* S_0 *and* S_1 *(with* S_0 *as the least significant bit) and produces output* F*.* A_0 *and* A_1 *are grounded,* A_3 *is connected to* V_{CC}*, and* A_2 *is connected to an XOR gate that outputs* $\overline{X} \oplus Y$*. What is* F *when* $S_0 = 0$*,* $X = 0$*,* $S_1 = 1$*, and* $Y = 1$*?*

1. ***Determine the MUX Selection:*** *With* $S_1 = 1$ *and* $S_0 = 0$*, the select lines choose input* A_2*.*

2. ***Evaluate Input*** A_2***:*** *The input* A_2 *is connected to an XOR gate, which outputs* $\overline{X} \oplus Y$*. Substitute* $X = 0$ *and* $Y = 1$*:*

$$\overline{X} = 1 \quad \textit{(since } X = 0\textit{)}$$
$$\overline{X} \oplus Y = 1 \oplus 1 = 0$$

3. ***Output*** F***:*** *Since* $A_2 = 0$ *(from the XOR gate), the output* $F = 0$ *for this selection of inputs.*

Example 3 ***Binary Multiplication***

- $1011_2 \times 1001_2$ *Start by doing line by line multiplication like in decimal, then do binary addition as before:*

$$\begin{array}{cccccccccc}
 & & & & & & 1 & 0 & 1 & 1 \\
 & & & & & \times & 1 & 0 & 0 & 1 \\
\hline
 & 1 \times 1011_2 & & & & & 1 & 0 & 1 & 1 \\
 & 0 \times 1011_2 & & & & 0 & 0 & 0 & 0 & \\
 & 0 \times 1011_2 & & & 0 & 0 & 0 & 0 & & \\
+ & 1 \times 1011_2 & 1 & 0 & 1 & 1 & & & & \\
\hline
 & & 1 & 1 & 0 & 1 & 1 & 1 & 1 &
\end{array}$$

$$1011_2 \times 1001_2 = 1101111_2$$

- *In decimal,* $-15 \times 7 = 105$. *Because -15 is negative, we must first take the two's compliment in order to perform the multiplication.*

- *15 is* 1111_2*, so the two's complement is* 0001_2 *added to the flipped bits of 15 expanded to 8 bits.*

00001111_2	*(original number)*
11110000_2	*(flipped bits)*
$+00000001_2$	*(add 1)*
11110001_2	*(two's complement result)*

The two's complement of 00001111_2 *is:*

$$11110001_2$$

- *Multiply* 11110001_2 *by* 00000111_2 *following binary multiplication rules:*

				1	1	1	1	0	0	0	1
$\times$				0	0	0	0	0	1	1	1
$1\times$				1	1	1	1	0	0	0	1
$1\times$			1	1	1	1	0	0	0	1	
$1\times$		1	1	1	1	0	0	0	1		
$0\times$	0	0	0	0	0	0	0	0			
				1	0	0	1	0	1	1	1

There really is no need to keep extending the partial products to the left, as the result is already well over our 8-bits.

- *So,* $1111_2 \times 0111_2 = 10010111_2$*, which, converting back to two's complement:*

10010111_2	*(original number)*
01101000_2	*(flipped bits)*
$+00000001_2$	*(add 1)*
01101001_2	*(two's complement result)*

- *Since we know this is a negative value, we append the negative when we convert back to decimal to receive*

$$01101001_2$$

$$-105_{10}$$