LU decomposition module

To understand the overall logic on how to find the LU decomposition, let's understand the mathematical relations used in the verilog code and the FSM used.

$$A = \begin{bmatrix} 2 & 3 & 1 & 5 \\ 6 & 13 & 5 & 19 \\ 2 & 3 & 2 & 6 \\ 1 & 2 & 1 & 3 \end{bmatrix}.$$

2. Formulas for a 4×4 (Doolittle's Method)

For a 4×4 matrix A, you iterate $k = 0$ to 3. In each step:

1. **Compute row k of U:**

$$U[k,j] \;=\; A[k,j] \;-\; \sum_{m=0}^{k-1} L[k,m]\,U[m,j], \quad \text{for } j \geq k.$$

This parallels the step in Gaussian elimination where you subtract a combination of earlier rows from row k so the pivot remains.

2. **Compute column k of L:**

$$L[i,k] \;=\; \frac{\left(A[i,k] \;-\; \sum_{m=0}^{k-1} L[i,m]\,U[m,k]\right)}{U[k,k]}, \quad \text{for } i > k.$$

This corresponds to using row k as the pivot row to eliminate below it, storing the multipliers in L.

The code in your Verilog module effectively unrolls these formulas into a **state machine** that updates each element in turn.

Step by step calculations :

1. At Reset and Start

1. The **IDLE** state waits for `start=1`.
2. In the **LOAD** state, `A_in` (the flattened matrix) is loaded into the 1D array `A[0..15]`.
   - For row 0, columns 0..3: `A[0]=2, A[1]=3, A[2]=1, A[3]=5`
   - Row 1: `A[4]=6, A[5]=13, A[6]=5, A[7]=19`
   - Row 2: `A[8]=2, A[9]=3, A[10]=2, A[11]=6`
   - Row 3: `A[12]=1, A[13]=2, A[14]=1, A[15]=3`
3. **INIT** sets all `L[]` and `U[]` to zero, then puts 1 on the diagonal of `L`.

Thus:

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Now **k=0**, moving to U_START.

2. Pivot $k = 0$ – Compute $U[0, j]$ for $j = 0..3$

$$U(0,j) \;=\; A(0,j) \;-\; \sum_{m=0}^{k-1} L(0,m)\,U(m,j)$$

Because $k = 0$, there is no sum term ($m = 0..-1$ is empty).

- $U(0,0)$ = `A[0] - 0` = 2
- $U(0,1)$ = `A[1] - 0` = 3
- $U(0,2)$ = `A[2] - 0` = 1
- $U(0,3)$ = `A[3] - 0` = 5

Now:

$$U = \begin{bmatrix} 2 & 3 & 1 & 5 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Next, the code transitions to computing $L(i,0)$ for $i = 1..3$:

$$L(i,0) = \frac{A(i,0) - \sum_{m=0}^{k-1} L(i,m)\,U(m,0)}{U(0,0)}$$

Again, $\sum_{m=0}^{-1}(\ldots) = 0$ because $k = 0$.

1. $L(1,0) = \frac{A(4)-0}{U(0)} = \frac{6}{2} = 3$.
2. $L(2,0) = \frac{A(8)-0}{2} = \frac{2}{2} = 1$.
3. $L(3,0) = \frac{A(12)-0}{2} = \frac{1}{2} = 0.5$.

So now:

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0.5 & 0 & 0 & 1 \end{bmatrix}.$$

And **k=0** completes. The code sets k=1 and returns to U_START.

3. Pivot $k = 1$

3.1 Compute $U(1, j)$ for $j = 1..3$

$k \leq j \leq 3$

$$U(1, j) = A(1, j) - \sum_{m=0}^{0} (L(1, m) \times U(m, j)).$$

Here the sum is just for $m = 0$.

- For $j = 1$:
  - $\sum = L(1, 0) \times U(0, 1) = 3 \times 3 = 9$.
  - $U(1, 1) = A(5) - 9 = 13 - 9 = 4$.
- For $j = 2$:
  - $\sum = 3 \times U(0, 2) = 3 \times 1 = 3$.
  - $U(1, 2) = A(6) - 3 = 5 - 3 = 2$.
- For $j = 3$:
  - $\sum = 3 \times 5 = 15$.
  - $U(1, 3) = A(7) - 15 = 19 - 15 = 4$.

Now:

$$U = \begin{bmatrix} 2 & 3 & 1 & 5 \\ 0 & 4 & 2 & 4 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

**

3.2 Compute $L(i,1)$ for $i = 2..3$

$$L(i,1) = \frac{A(i,1) - \sum_{m=0}^{0}(L(i,m) \times U(m,1))}{U(1,1)}.$$

- For $i = 2$:
  - $\sum = L(2,0) \times U(0,1) = 1 \times 3 = 3$.
  - $L(2,1) = \frac{A(9)-3}{U(1,1)} = \frac{3-3}{4} = 0$.
- For $i = 3$:
  - $\sum = L(3,0) \times U(0,1) = (0.5) \times 3 = 1.5$.
  - $L(3,1) = \frac{2-1.5}{4} = \frac{0.5}{4} = 0.125$.

Hence:

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0.5 & 0.125 & 0 & 1 \end{bmatrix}.$$

Then **k=1** completes, we move to **k=2**.

6. Final L and U

1. **L:**

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0.5 & 0.125 & 0.25 & 1 \end{bmatrix}$$

2. **U:**

$$\begin{bmatrix} 2 & 3 & 1 & 5 \\ 0 & 4 & 2 & 4 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & -0.25 \end{bmatrix}$$

If you multiply $L \times U$, you get back A:

$$L \times U = \begin{bmatrix} 2 & 3 & 1 & 5 \\ 6 & 13 & 5 & 19 \\ 2 & 3 & 2 & 6 \\ 1 & 2 & 1 & 3 \end{bmatrix},$$

verifying correctness (again, ignoring integer-vs-floating differences).

2. Formulas for a 4×4 (Doolittle's Method)

For a 4×4 matrix A, you iterate $k = 0$ to 3. In each step:

1. **Compute row k of U:**

$$U[k, j] \;=\; A[k, j] \;-\; \sum_{m=0}^{k-1} L[k, m]\, U[m, j], \quad \text{for } j \geq k.$$

This parallels the step in Gaussian elimination where you subtract a combination of earlier rows from row k so the pivot remains.

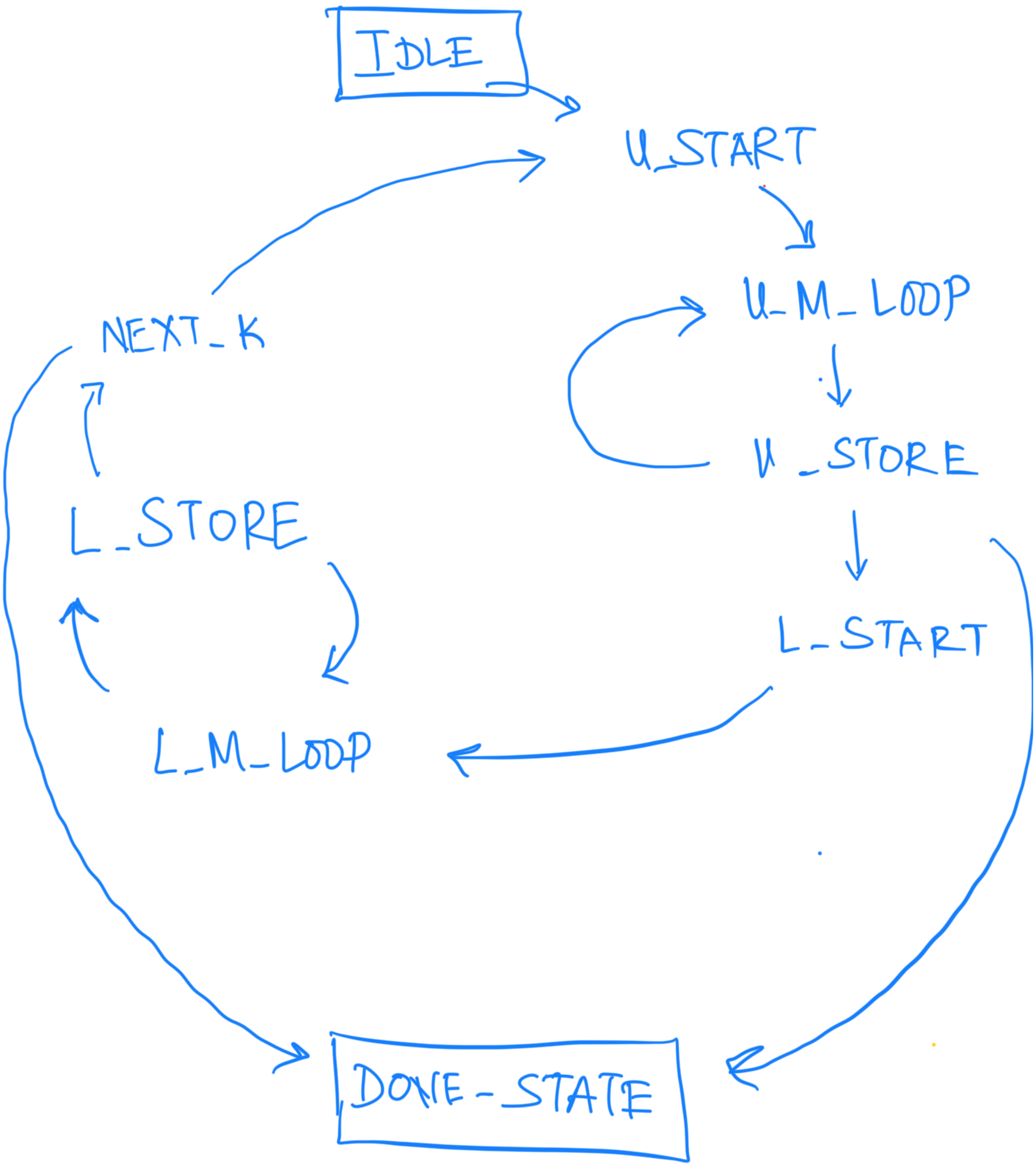
2. **Compute column k of L:**

$$L[i, k] \;=\; \frac{\left(A[i, k] \;-\; \sum_{m=0}^{k-1} L[i, m]\, U[m, k]\right)}{U[k, k]}, \quad \text{for } i > k.$$

This corresponds to using row k as the pivot row to eliminate below it, storing the multipliers in L.

The code in your Verilog module effectively unrolls these formulas into a **state machine** that updates each element in turn.

FSM



IDLE
U_START
U_M_LOOP
U_STORE
L_START
L_M_LOOP
L_STORE
NEXT_K
DONE-STATE

Summary Diagram of the FSM Flow

```
 +-------+
 | IDLE  | <--- Wait for start
 +-------+
     |
     v
 +-------+
 | LOAD  | <--- Load input matrix into A array
 +-------+
     |
     v
 +-------+
 | INIT  | <--- Initialize L and U arrays, set L diagonal=1, k=0
 +-------+
     |
     v
 +---------+
 | U_START | <--- Set j=k, clear sum, set m=0 for U computation
 +---------+
     |
     v
 +----------+
 | U_M_LOOP | <--- For m from 0 to k-1, accumulate sum
 +----------+
     |
     v
 +----------+
 | U_STORE  | <--- Compute U[k][j]=A[k][j]-sum; if more columns, loop; else proceed to L
 +----------+
     |
     v
 +----------+
 | L_START  | <--- Set i=k+1, clear sum and m for L computation
 +----------+
     |
     v
 +----------+
 | L_M_LOOP | <--- For m from 0 to k-1, accumulate sum for L
 +----------+
     |
     v
 +----------+
```

```
  | L_STORE  | <--- Compute L[i][k]=(A[i][k]-sum)/U[k][k]; if more rows, loop; else finish pivot
  +----------+
      |
      v
  +----------+
  | NEXT_K   | <--- Increment k; if k < 3, go to U_START; else DONE_STATE
  +----------+
      |
      v
  +----------+
  | DONE_STATE | <--- Pack L and U into outputs and assert done signal
```