Full 4-Bit Ripple Carry Adder

1 Overview

Below is an explanation and a diagram (using text-based formatting) of a full 4-bit ripple carry adder. In this design, four full adder blocks are cascaded. Each full adder adds the corresponding bits of two 4-bit numbers (A and B) along with a carry input. The carry-out from each stage "ripples" to the next stage's carry-in.

1.1 Adder Stages

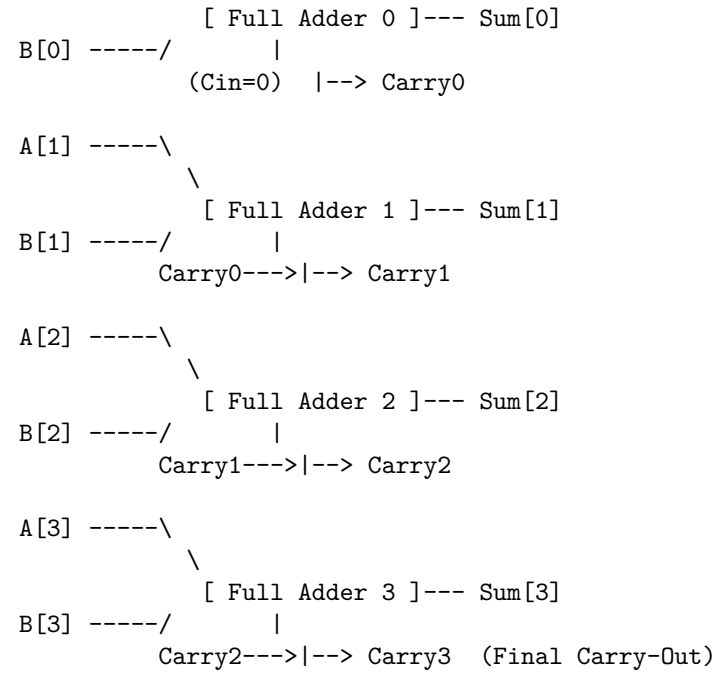
- **Stage 0 (LSB):**
  - **Inputs:** $A[0]$, $B[0]$
  - **Carry-in:** Fixed at 0 (since it's the LSB)
  - **Outputs:**
    - $\text{Sum}_0 = A[0] \oplus B[0]$
    - $\text{Carry}_0 = A[0] \cdot B[0]$
- **Stages 1 to 3 (Full Adders):** For each bit position i (where $i = 1, 2, 3$):
  - **Inputs:** $A[i]$, $B[i]$ and the carry-in from the previous stage (Carry$[i-1]$)
  - **Outputs:**
    - $\text{Sum}[i] = A[i] \oplus B[i] \oplus \text{Carry}[i-1]$
    - $\text{Carry}[i] = \big(A[i] \cdot B[i]\big) + \big(B[i] \cdot \text{Carry}[i-1]\big) + \big(A[i] \cdot \text{Carry}[i-1]\big)$

1.2 Schematic Diagram

Below is a schematic-style text diagram illustrating how the full adders are cascaded:

```
4-bit Ripple Carry Adder
-------------------------

  A[0] -----\
             \
```



1.3 Details of Each Full Adder Block

1. **Full Adder Logic Equations:**
   For stages 1–3 (and for stage 0 with the note that $Cin = 0$), the full adder uses these logic equations:

$$\begin{aligned} \text{Sum}[i] &= A[i] \oplus B[i] \oplus Cin, \\ \text{Carry-out} &= \big(A[i] \cdot B[i]\big) + \big(B[i] \cdot Cin\big) + \big(A[i] \cdot Cin\big). \end{aligned}$$

2. **LSB Special Case (Stage 0):**
   Since $Cin = 0$, the equations for Stage 0 simplify to:

$$\begin{aligned} \text{Sum}_0 &= A[0] \oplus B[0], \\ \text{Carry}_0 &= A[0] \cdot B[0]. \end{aligned}$$

1.4 Operation

- **Stage 0:**
  The adder for the least significant bit (LSB) computes the sum of $A[0]$ and $B[0]$ without needing a carry-in. If both $A[0]$ and $B[0]$ are 1, a carry is produced for the next stage.

- **Stages 1–3:**
  Each subsequent full adder takes its carry-in from the previous stage's

carry-out. This chain of carry propagation is why the circuit is called a "ripple carry adder" — the carry "ripples" from the LSB to the MSB.

- **Final Output:**
  The 4-bit sum is given by the outputs Sum[3 : 0]. An extra carry-out (Carry_3) is available if an overflow occurs (i.e., if the sum exceeds 4 bits).

This design meets the lab's requirements by cascading the basic full adder circuits (with the simplified LSB) to create a complete 4-bit ripple carry adder. You can implement this design in LTSpice by creating the individual full adder blocks (or using your previously designed LSB block for Stage 0) and then connecting them as shown above.

2 MermaidJS Diagrams

Below are nine separate MermaidJS diagrams — one for each section of the 4-bit ripple carry adder design process. You can paste each code snippet into a Mermaid-enabled environment (or an online Mermaid live editor) to view the corresponding visualization.

2.1 1. Block Diagram of the 4-Bit Ripple Carry Adder

```
flowchart TD
   %% Input Block
   subgraph Inputs
     A[A[3:0]]
     B[B[3:0]]
     Cin[Carry-In = 0]
   end

   %% LSB Adder Block (Stage 0)
   subgraph Stage0 [LSB Adder]
     LSB[LSB: A[0]  B[0]]
     C0[Carry-Out: A[0] •B[0]]
   end

   %% Full Adder Blocks (Stages 1-3)
   subgraph Stage1 [Full Adder 1]
     FA1[Full Adder]
   end
   subgraph Stage2 [Full Adder 2]
     FA2[Full Adder]
   end
   subgraph Stage3 [Full Adder 3]
     FA3[Full Adder]
   end
```

```
    %% Wiring between blocks
    A -->|Bit 0| LSB
    B -->|Bit 0| LSB
    Cin --> LSB

    LSB --> S0[SUM[0]]
    LSB --> C0

    C0 --> FA1
    FA1 --> S1[SUM[1]]
    FA1 --> C1[Carry-Out]

    C1 --> FA2
    FA2 --> S2[SUM[2]]
    FA2 --> C2[Carry-Out]

    C2 --> FA3
    FA3 --> S3[SUM[3]]
    FA3 --> Cout[Final Carry-Out]
```

2.2 2. Flowchart for the Design Process

```
flowchart TD
    A[Understand Requirements: LSB Block, Full Adder Blocks, Simplified
        LSB]
    B[Create Truth Tables for each block]
    C[Derive Logic Expressions: SUM = XOR; Carry-Out = AND (LSB) and Full
         Adder uses XOR with majority function]
    D[Implement LTSpice Circuits: Design & connect gates]
    E[Test & Simulate: Validate outputs vs. truth tables]
    F[Assemble 4-bit Adder: Chain blocks together]
    G[Final Testing: Run full adder simulation with sample inputs]

    A --> B
    B --> C
    C --> D
    D --> E
    E --> F
    F --> G
```

2.3 3. Data Flow Diagram (DFD)

```
flowchart TD
    %% Data Sources
    subgraph Sources
      A[A[3:0]]
      B[B[3:0]]
```

```
    Cin[Carry-In]
  end

  %% Processing Blocks
  subgraph Processing
    XOR[XOR Gates (for Sum Calculation)]
    AND_OR[AND/OR Gates (for Carry-Out Calculation)]
  end

  %% Outputs
  subgraph Outputs
    SUM[SUM[3:0]]
    Cout[Carry-Out]
  end

  A --> XOR
  B --> XOR
  XOR --> SUM

  Cin --> AND_OR
  AND_OR --> Cout
```

2.4 4. Truth Table for LSB Block (Carry-In = 0)

```
flowchart TD
    A0["A[0]=0, B[0]=0"]
    A1["A[0]=0, B[0]=1"]
    A2["A[0]=1, B[0]=0"]
    A3["A[0]=1, B[0]=1"]

    A0 --> T0["SUM[0]=0, Carry=0"]
    A1 --> T1["SUM[0]=1, Carry=0"]
    A2 --> T2["SUM[0]=1, Carry=0"]
    A3 --> T3["SUM[0]=0, Carry=1"]
```

2.5 5. Logic Circuit Diagram for LSB Adder

```
flowchart LR
    A[A[0]] --> XOR[XOR Gate]
    B[B[0]] --> XOR
    XOR --> SUM[SUM[0] = A  B]

    A --> AND[AND Gate]
    B --> AND
    AND --> Cout[CARRY_OUT = A •B]
```

2.6 6. Full Adder Circuit Diagram

```
flowchart TD
    %% Full Adder Inputs and Internal Gates
    subgraph Full_Adder
      A[A[i]]
      B[B[i]]
      Cin[Cin]

      XOR1[XOR Gate]
      XOR2[XOR Gate]
      SUM[SUM[i]]

      AND1[AND Gate]
      AND2[AND Gate]
      AND3[AND Gate]
      OR[OR Gate]
      Cout[CARRY_OUT]
    end

    %% Sum Calculation Path
    A --> XOR1
    B --> XOR1
    XOR1 --> XOR2
    Cin --> XOR2
    XOR2 --> SUM

    %% Carry-Out Calculation Paths
    A --> AND1
    B --> AND1

    B --> AND2
    Cin --> AND2

    A --> AND3
    Cin --> AND3

    AND1 --> OR
    AND2 --> OR
    AND3 --> OR
    OR --> Cout
```

2.7 7. Simplified LSB Block Diagram

```
flowchart LR
    A[A[0]] --> XOR[XOR Gate]
    B[B[0]] --> XOR
    XOR --> SUM[SUM[0] = A  B]
```

```
    A --> AND[AND Gate]
    B --> AND
    AND --> Cout[CARRY_OUT = A •B]
```

2.8 8. Integrated 4-Bit Ripple Carry Adder Diagram

```
flowchart TD
    %% Stage 0: Simplified LSB
    subgraph Stage0 [LSB Adder]
      LSB[LSB: A[0]  B[0]]
      C0[Carry-Out: A[0] •B[0]]
    end

    %% Stage 1: Full Adder 1
    subgraph Stage1 [Full Adder 1]
      FA1[Full Adder]
      S1[SUM[1]]
      C1[Carry-Out]
    end

    %% Stage 2: Full Adder 2
    subgraph Stage2 [Full Adder 2]
      FA2[Full Adder]
      S2[SUM[2]]
      C2[Carry-Out]
    end

    %% Stage 3: Full Adder 3
    subgraph Stage3 [Full Adder 3]
      FA3[Full Adder]
      S3[SUM[3]]
      C3[Final Carry-Out]
    end

    %% Inputs and Connections
    A[A[3:0]] -->|Bit0| LSB
    B[B[3:0]] -->|Bit0| LSB
    Cin[Carry-In = 0] --> LSB

    LSB --> S0[SUM[0]]
    LSB --> C0

    C0 --> FA1
    FA1 --> S1
    FA1 --> C1

    C1 --> FA2
```

```
    FA2 --> S2
    FA2 --> C2

    C2 --> FA3
    FA3 --> S3
    FA3 --> C3
```

2.9 9. Simulation Waveform Diagram

```
sequenceDiagram
    participant Test as Test Case
    participant Adder as 4-bit Adder

    Test->>Adder: Input 0000 + 0000
    Adder-->>Test: Output 00000 (No carry propagation)

    Test->>Adder: Input 0001 + 1111
    Adder-->>Test: Output 10000 (Carry propagates through all bits)

    Test->>Adder: Input 1010 + 0101
    Adder-->>Test: Output 11111 (Alternating bits addition)
```