**Team member 1 : Sankalpa Hota , PID : A69041322 - CLA**
**Team member 2 : Louis Lin , PID : A69043447 - Carry Skip**
**Team member 3 : Bing-You Yu, PID : A69041854 - Carry Select**

# Objective

The objective of this lab is to design a 4-tap signed magnitude FIR filter at the register-transfer-level (RTL) abstraction in SystemVerilog and synthesize it to a gate-level netlist. The FIR filter assumes all coefficients h[k]=1h[k] = 1h[k]=1. Various adder architectures are implemented to perform the multi-input addition required for the FIR computation. The differences in area, power, and performance are noted for each adder type.

# Tools used:

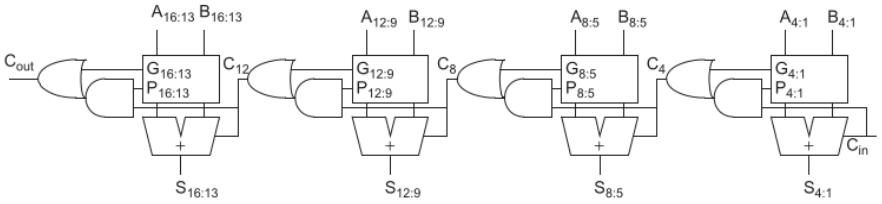
1) Iverilog for syntax and logic verification
2) Intel Quartus Prime Lite for Synthesis , Timing , Power and Area

# Designs

**1) Carry Lookahead Adder (CLA)**

The FIR filter was implemented using a multi-stage **Carry Lookahead Adder** for all intermediate summations.

- A 16-bit CLA was used for the first stage, followed by 17-bit and 18-bit stages to accommodate the carry propagation and pipeline registers.
- The lookahead logic generates block propagate and generate signals, allowing faster computation by reducing carry chain delay compared to conventional RCAs.
- Each stage was pipelined to improve timing performance for the 4-tap FIR addition.

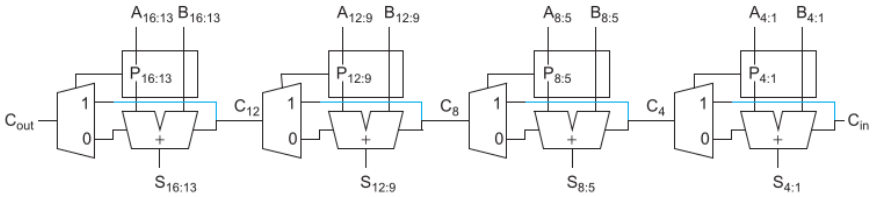



**2) Carry Skip Adder (CSKA)**

The **Carry Skip Adder** design divides the adder into blocks of 4 bits, with block propagate signals controlling whether the incoming carry can skip the entire block.

- The FIR filter addition is performed using 16-bit adders with 4-bit skip blocks.
- Intermediate sums are pipelined over four registers to match the FIR delay requirements.

- The carry-skip approach reduces worst-case delay by allowing carries to bypass blocks when all bits propagate.




**3) Carry Select Adder (CSLA)**

The FIR filter uses a **multi-input Carry Select Adder** for the first stage of addition.

- Each block precomputes sums for both possible carry-in values (0 and 1) in parallel.
- At block boundaries, the correct sum is selected using multiplexers based on the incoming carry.
- This structure reduces critical path delay compared to a pure ripple carry adder, as each block can compute sums independently before the carry is known.
- Pipeline registers are inserted at intermediate stages for timing alignment and to maintain synchronous FIR operation.

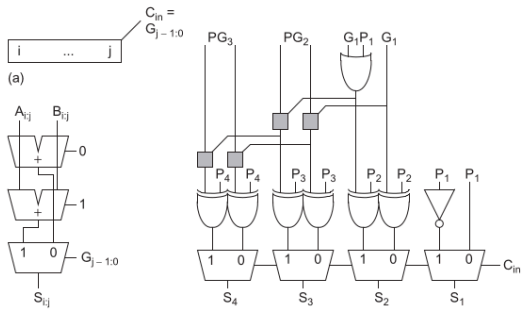



FIGURE 11.33 Carry-select implementation

**Time Complexity details :**

**Carry Lookahead Adder (CLA):** Time complexity is **O(log n)** due to parallel computation of carry generate and propagate signals across blocks.
**Carry Skip Adder (CSKA):** Time complexity is **O(√n)** in the worst case, as carry can skip blocks but may ripple through multiple blocks.
**Carry Select Adder (CSA):** Time complexity is **O(√n)**.

# Challenges during Design :

**1.Handling Signed Arithmetic and Carry Behavior**
Designing adders for signed inputs introduced multiple issues, especially when propagating carries through the MSB. During verification in Icarus Verilog (iverilog), several intermediate sums behaved incorrectly because sign-extension, MSB carry-out, or $signed() casting was mishandled. Ensuring that the CLA, CSA, and CSA-based FIR stages preserved 2's-complement semantics across pipeline registers required iterative debugging.

**2.Mismatch Between Simulation and Synthesis Tool Requirements**
Many Verilog/SystemVerilog constructs that compiled successfully in iverilog failed in Quartus Prime. Quartus is stricter about constant expressions, parameter binding, and part-select indexing. This resulted in errors such as "base is not a constant," incorrect parameter resolution (W not bound), and invalid bit-slices — all of which required restructuring the code to be fully synthesizable.

**3.Parameterization Issues and Width Explosion**
Creating parameterized adders (Ripple, CLA, Skip, Select) introduced width-alignment problems. Modules using W, W-1, W+1 in part-selects failed unless carefully enforced as compile-time constants. Intermediate sums in the FIR pipeline (e.g., sum1, sum_final) needed explicit width guards to avoid overflow or truncation. These subtle width mismatches caused unexplained RTL simulation mismatches before being corrected.

**4.Tool Constraints When Modeling Hierarchical Adders**
Carry Look-Ahead, Carry Skip, and Carry Select architectures involve internal generate/propagate networks, block-level carry chains, and mux logic. Translating these structures into clean synthesizable RTL without loops that Quartus rejects was challenging. Some constructs (e.g., variable part-selects inside loops or local functions using parameters) required hand-unrolling or rewriting into simpler fixed-width logic for Quartus to accept.

# Observation: Simulation on FPGA - Cyclone V

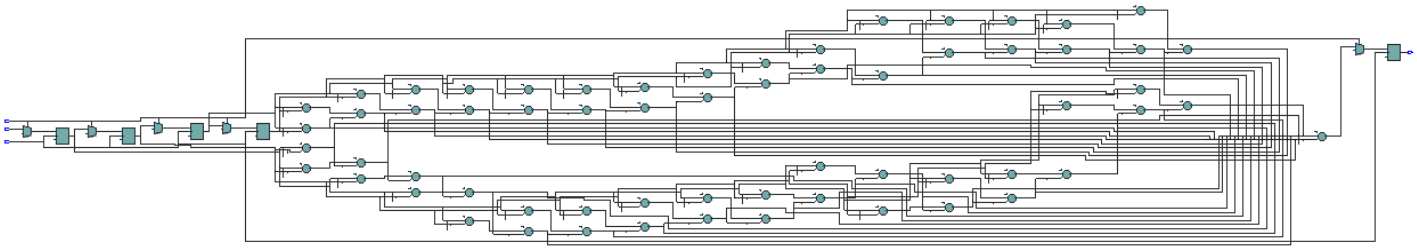
**Details -**
1)Intel Cyclone V data sheet = Area of 1 ALM = 250 μm^2
2)QP has 4 modes for timing delay : Slow ( 1 V , 1.1 V ) and Fast ( 1 V , 1.1 V) - Max timing violation we observed in Slow - 1.1 V mode at -40C.

| | Vanilla - Ripple | Carry Look Ahead | Carry Skip Adder | Carry Select Adder |
|---|---|---|---|---|
| **Worst negative slack (ps)** | **23 ( Slow 1100mv -40C model in QP)** | **4.5 ( Slow 1100mv -40C model in QP)** | **3.9 ( Slow 1100mv -40C model in QP)** | **5.5 ( Slow 1100mv -40C model in QP)** |
| **Total power consumed (mW)** | **420** | **416** | **426** | **421** |
| **Leakage** | **25** | **23** | **26** | **28** |

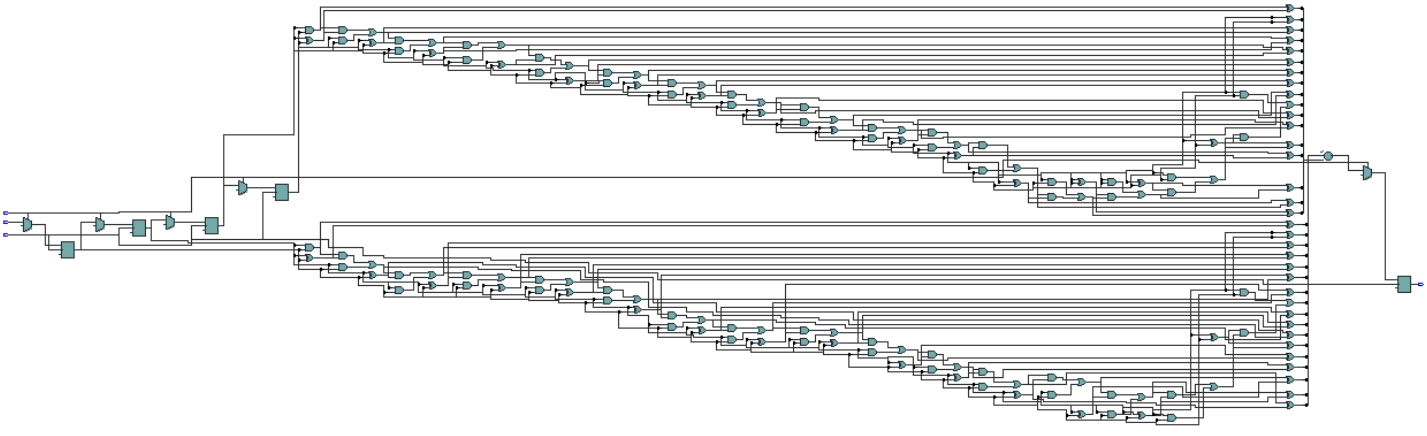
| Power(uW) | | | | |
|---|---|---|---|---|
| Area of combinational logic (um2 ) | 250 µm^2 | 250 µm^2 | 250 µm^2 | 250 µm^2 |
| no. of combinational Cells - ALMs | 155 | 49 | 30 | 39 |
| Total area (um2 ) | 38750 µm^2 | 12250 µm^2 | 7500µm^2 | 9750µm^2 |

# Synthesized RTLs: Using Quartus Prime

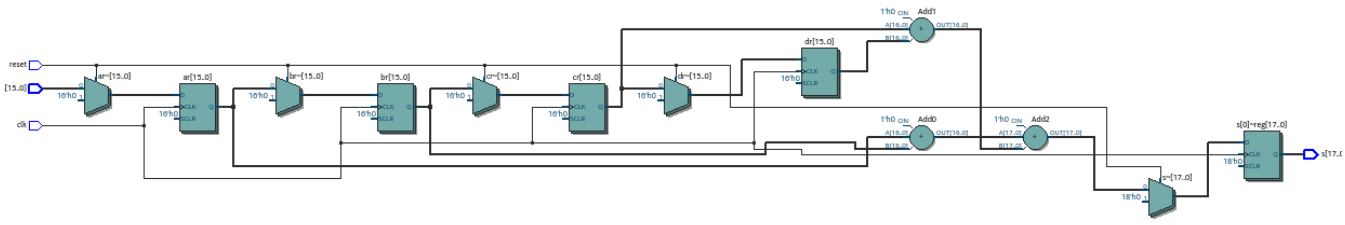
**Vanilla : Ripple Carry**



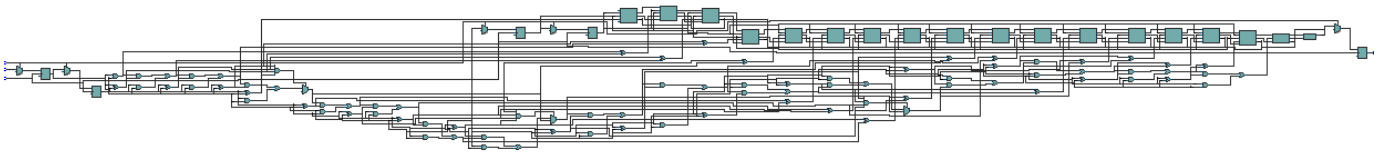
**CLA :**



**Carry Select : ( We do it for all bits at once) - Simple Architecture**



**Carry Skip :**



**Conclusion :**
Based on the results obtained from Quartus Prime using the Slow 1100 mV –40 °C timing model, all four adder architectures exhibit negative slack, confirming that none of the designs meet timing at the default clock frequency. Among the tested architectures, the **Carry Skip Adder shows the best timing behavior** with a Worst Negative Slack of 3.9 ps, followed closely by the Carry Look-Ahead Adder (4.5 ps). The Carry Select Adder, while more modular and structurally faster, reports a 5.5 ps negative slack. The baseline Ripple Carry Adder performs the worst with 23 ps of negative slack, clearly showing the penalty of linear carry propagation. From a pure timing perspective, the skip and look-ahead designs offer the most improvement over the vanilla ripple structure. For 16 bits **Skip adder** is the best Adder. As Root(n) < Log(n)<n for n = 16

In terms of implementation cost, **the Carry Skip Adder again stands out with only 30 ALMs, the smallest logic footprint among the four,** corresponding to a total physical area of 7500 μm². The Carry Look-Ahead and Carry Select adders also significantly reduce area compared to the ripple baseline, using 49 ALMs (12 250 μm²) and 39 ALMs (9750 μm²) respectively. All four designs show nearly identical combinational logic area (250 μm²) due to identical coefficient multipliers and FIR structure, so the differences arise mainly from the adder micro-architecture. Power results remain tightly grouped, ranging from 416–426 mW, with leakage power between 23–28 μW; these narrow differences indicate that the architectural changes influence timing and area far more than total power. Quartus Prime was used with default synthesis and timing settings, which allowed us to collect Fitter ALM count, physical area, timing slack, and power estimates via the integrated Power Analyzer and Timing Analyzer tools.