Lab Report: Hierarchical Verilog Code Generation Using AI Models

This report presents the results of automated Verilog code generation using hierarchical prompting techniques with two large language models: GPT-4o Mini and Claude Sonnet 4.5. The study focuses on generating two fundamental digital circuit components—multiplexers and comparators—using a bottom-up hierarchical approach. The generated modules were tested using testbenches to verify functional correctness.

Design Hierarchy

Multiplexer Design:

1. 2-to-1 Multiplexer (base module)
2. 4-to-1 Multiplexer (using 2-to-1 modules)
3. 8-to-1 Multiplexer (using 4-to-1 modules)

Comparator Design:

1. 2-bit Comparator (base module)
2. 4-bit Comparator (using 2-bit modules)
3. 8-bit Comparator (using 4-bit modules)
4. 16-bit Comparator (using 8-bit modules)

Results

Performance Metrics

The following tables present detailed performance metrics for each design hierarchy level generated by both AI models.

Table 1: GPT-4o Mini - Multiplexer Generation Metrics

Module	Total Time (s)	Generation Time (s)	Error Handling Time (s)	Compilation Errors	Status
2-to-1 MUX	1.725	1.711	0.014	0	Pass
4-to-1 MUX	5.699	5.685	0.015	0	Pass
8-to-1 MUX	38.301	38.263	0.037	2	Pass
Overall	**45.725**	**45.659**	**0.066**	**2**	**All Pass**

Code:

```
module mux2to1(input wire in1, input wire in2, input wire select, output wire out);
   assign out = select ? in2 : in1;
endmodule
```

```
module mux2to1(input wire in1, input wire in2, input wire select, output wire out);
   assign out = select ? in2 : in1;
endmodule
module mux4to1(input [1:0] sel, input [3:0] in, output reg out);
   wire out0, out1;

   // Instantiate two 2-to-1 multiplexers
   mux2to1 mux0(.in1(in[0]), .in2(in[1]), .select(sel[0]), .out(out0));
   mux2to1 mux1(.in1(in[2]), .in2(in[3]), .select(sel[0]), .out(out1));

   // Final selection using another 2-to-1 multiplexer
   always @(*) begin
      out = (sel[1] == 1'b0) ? out0 : out1;
   end
endmodule
```

```
module mux2to1(input wire in1, input wire in2, input wire select, output wire out);
   assign out = select ? in2 : in1;
endmodule
module mux4to1(input [1:0] sel, input [3:0] in, output wire out);
   wire mux0_out, mux1_out;

   // Instantiate two 2-to-1 multiplexers
   mux2to1 mux0(.in1(in[0]), .in2(in[1]), .select(sel[0]), .out(mux0_out));
   mux2to1 mux1(.in1(in[2]), .in2(in[3]), .select(sel[0]), .out(mux1_out));

   // Use another 2-to-1 multiplexer to select between the outputs of the first two
   mux2to1 mux2(.in1(mux0_out), .in2(mux1_out), .select(sel[1]), .out(out));
endmodule
module mux8to1(input [2:0] sel, input [7:0] in, output wire out);
   wire mux0_out, mux1_out;

   // Instantiate two 4-to-1 multiplexers
   mux4to1 mux0(.sel(sel[1:0]), .in(in[3:0]), .out(mux0_out));
   mux4to1 mux1(.sel(sel[1:0]), .in(in[7:4]), .out(mux1_out));

   // Use a 2-to-1 multiplexer to select between the outputs of the first two
   mux2to1 mux2(.in1(mux0_out), .in2(mux1_out), .select(sel[2]), .out(out));
endmodule
```

Table 2: GPT-4o Mini - Comparator Generation Metrics

Module	Total Time (s)	Generation Time (s)	Error Handling Time (s)	Compilation Errors	Status
2-bit Comparator	1.671	1.657	0.014	0	Pass
4-bit Comparator	4.614	4.593	0.021	0	Pass
8-bit Comparator	5.904	5.890	0.015	0	Pass
16-bit Comparator	14.610	14.583	0.027	1	Pass
Overall	**26.800**	**26.723**	**0.077**	**1**	**All Pass**

Code:

```
module comp2(input [1:0] A, input [1:0] B, output gt, output lt, output eq);
   assign gt = (A > B);
   assign lt = (A < B);
   assign eq = (A == B);
endmodule

module comp2(input [1:0] A, input [1:0] B, output gt, output lt, output eq);
   assign gt = (A > B);
   assign lt = (A < B);
   assign eq = (A == B);
endmodule
module comp4(input [3:0] A, input [3:0] B, output gt, output lt, output eq);
   wire gt0, lt0, eq0, gt1, lt1, eq1;

   comp2 lower_comp(A[1:0], B[1:0], gt0, lt0, eq0);
   comp2 upper_comp(A[3:2], B[3:2], gt1, lt1, eq1);

   assign gt = gt1 | (eq1 & gt0);
   assign lt = lt1 | (eq1 & lt0);
   assign eq = eq1 & eq0;
endmodule

module comp2(input [1:0] A, input [1:0] B, output gt, output lt, output eq);
   assign gt = (A > B);
   assign lt = (A < B);
   assign eq = (A == B);
endmodule
module comp4(input [3:0] A, input [3:0] B, output gt, output lt, output eq);
   wire gt0, lt0, eq0, gt1, lt1, eq1;

   comp2 lower_comp(A[1:0], B[1:0], gt0, lt0, eq0);
```

```
    comp2 upper_comp(A[3:2], B[3:2], gt1, lt1, eq1);

    assign gt = gt1 | (eq1 & gt0);
    assign lt = lt1 | (eq1 & lt0);
    assign eq = eq1 & eq0;
endmodule
module comp8(input [7:0] A, input [7:0] B, output gt, output lt, output eq);
    wire gt0, lt0, eq0, gt1, lt1, eq1;

    comp4 lower_comp(A[3:0], B[3:0], gt0, lt0, eq0);
    comp4 upper_comp(A[7:4], B[7:4], gt1, lt1, eq1);

    assign gt = gt1 | (eq1 & gt0);
    assign lt = lt1 | (eq1 & lt0);
    assign eq = eq1 & eq0;
endmodule


module comp2(input [1:0] A, input [1:0] B, output gt, output lt, output eq);
    assign gt = (A > B);
    assign lt = (A < B);
    assign eq = (A == B);
endmodule
module comp4(input [3:0] A, input [3:0] B, output gt, output lt, output eq);
    wire gt0, lt0, eq0, gt1, lt1, eq1;

    comp2 lower_comp(A[1:0], B[1:0], gt0, lt0, eq0);
    comp2 upper_comp(A[3:2], B[3:2], gt1, lt1, eq1);

    assign gt = gt1 | (eq1 & gt0);
    assign lt = lt1 | (eq1 & lt0);
    assign eq = eq1 & eq0;
endmodule
module comp8(input [7:0] A, input [7:0] B, output gt, output lt, output eq);
    wire gt0, lt0, eq0, gt1, lt1, eq1;

    comp4 lower_comp(A[3:0], B[3:0], gt0, lt0, eq0);
    comp4 upper_comp(A[7:4], B[7:4], gt1, lt1, eq1);

    assign gt = gt1 | (eq1 & gt0);
    assign lt = lt1 | (eq1 & lt0);
    assign eq = eq1 & eq0;
endmodule
module comp16(input [15:0] A, input [15:0] B, output gt, output lt, output eq);
    wire gt0, lt0, eq0, gt1, lt1, eq1;

    comp8 lower_comp(A[7:0], B[7:0], gt0, lt0, eq0);
    comp8 upper_comp(A[15:8], B[15:8], gt1, lt1, eq1);

    assign gt = gt1 | (eq1 & gt0);
    assign lt = lt1 | (eq1 & lt0);
```

```
   assign eq = eq1 & eq0;
endmodule
```

Error in last module first iter:

```
module comp2(input [1:0] A, input [1:0] B, output gt, output lt, output eq);
   assign gt = (A > B);
   assign lt = (A < B);
   assign eq = (A == B);
endmodule

module comp4(input [3:0] A, input [3:0] B, output gt, output lt, output eq);
   wire gt0, lt0, eq0, gt1, lt1, eq1;

   comp2 lower_comp(A[1:0], B[1:0], gt0, lt0, eq0);
   comp2 upper_comp(A[3:2], B[3:2], gt1, lt1, eq1);

   assign gt = gt1 | (eq1 & gt0);
   assign lt = lt1 | (eq1 & lt0);
   assign eq = eq1 & eq0;
endmodule

module comp8(input [7:0] A, input [7:0] B, output gt, output lt, output eq);
   wire gt0, lt0, eq0, gt1, lt1, eq1;

   comp4 lower_comp(A[3:0], B[3:0], gt0, lt0, eq0);
   comp4 upper_comp(A[7:4], B[7:4], gt1, lt1, eq1);

   assign gt = gt1 | (eq1 & gt0);
   assign lt = lt1 | (eq1 & lt0);
   assign eq = eq1 & eq0;
endmodule
```

user: The testbench failed to compile. Please fix the module. The output of iverilog is as follows:

```
./comp16tb.v:6: error: Unknown module type: comp16
2 error(s) during elaboration.
*** These modules were missing:
      comp16 referenced 1 times.
***
```

Table 3: Claude Sonnet 4.5 - Multiplexer Generation Metrics

Module	Total Time (s)	Generation Time (s)	Error Handling Time (s)	Compilation Errors	Status
2-to-1 MUX	2.881	2.862	0.019	0	Pass
4-to-1 MUX	3.787	3.773	0.014	0	Pass
8-to-1 MUX	9.617	9.593	0.024	1	Pass
Overall	**16.285**	**16.228**	**0.057**	**1**	**All Pass**

Table 4: Claude Sonnet 4.5 - Comparator Generation Metrics

Module	Total Time (s)	Generation Time (s)	Error Handling Time (s)	Compilation Errors	Status
2-bit Comparator	3.289	3.275	0.014	0	Pass
4-bit Comparator	4.840	4.824	0.016	0	Pass
8-bit Comparator	5.686	5.671	0.015	0	Pass
16-bit Comparator	15.625	15.592	0.033	1	Pass
Overall	**29.441**	**29.363**	**0.078**	**1**	**All Pass**

GPT-4o Mini Results

Multiplexer Generation

The hierarchical multiplexer generation proceeded through three stages: 2-to-1, 4-to-1, and 8-to-1 implementations. Each module successfully passed its testbench verification, as indicated by the "Test bench ran successfully" messages. The output shows:

- All three hierarchical levels were generated correctly
- Testbenches validated functional correctness at each stage
- The hierarchical approach successfully built complex structures from simpler ones

Comparator Generation

The comparator design scaled through four hierarchical levels (2-bit, 4-bit, 8-bit, and 16-bit). Results demonstrate:

- Successful generation of all four complexity levels
- Each comparator module passed testbench verification
- The model maintained design consistency across increasing bit-widths

Claude Sonnet 4.5 Results

Multiplexer Generation

Claude Sonnet 4.5 successfully generated the complete multiplexer hierarchy with all testbenches passing. The results mirror the success pattern of GPT-4o Mini, demonstrating reliable hierarchical design generation.

Comparator Generation

The comparator hierarchy was successfully generated through all four levels (2-bit through 16-bit), with each module passing its respective testbench validation.

Performance Comparison

Table 5: Overall Performance Summary

Metric	GPT-4o Mini MUX	Claude Sonnet 4.5 MUX	GPT-4o Mini CMP	Claude Sonnet 4.5 CMP
Total Time (s)	45.725	16.285	26.800	29.441
Generation Time (s)	45.659	16.228	26.723	29.363
Error Handling Time (s)	0.066	0.057	0.077	0.078
Compilation Errors	2	1	1	1
Modules Generated	3	3	4	4

Table 6: Module-by-Module Time Comparison

Module	GPT-4o Mini (s)	Claude Sonnet 4.5 (s)	Time Difference	Performance Ratio
Multiplexers				
2-to-1 MUX	1.725	2.881	+1.156	GPT 1.7x faster
4-to-1 MUX	5.699	3.787	-1.912	Claude 1.5x faster
8-to-1 MUX	38.301	9.617	-28.684	Claude 4.0x faster
MUX Subtotal	**45.725**	**16.285**	**-29.440**	**Claude 2.8x faster**
Comparators				
2-bit CMP	1.671	3.289	+1.618	GPT 2.0x faster
4-bit CMP	4.614	4.840	+0.226	Similar (~5% diff)
8-bit CMP	5.904	5.686	-0.218	Similar (~4% diff)
16-bit CMP	14.610	15.625	+1.015	Similar (~7% diff)
CMP Subtotal	**26.800**	**29.441**	**+2.641**	**GPT 1.1x faster**

Detailed Performance Analysis

Overall Time Analysis:

- **Combined total generation time**: Claude Sonnet 4.5 completed all tasks in 45.726 seconds compared to GPT-4o Mini's 72.525 seconds
- **Overall speedup**: Claude Sonnet 4.5 was **37% faster** across all module generations
- **Time saved**: Claude Sonnet 4.5 saved approximately 26.8 seconds of total generation time

Multiplexer Performance Breakdown:

1. **2-to-1 Multiplexer (Base Module)**:
    - GPT-4o Mini: 1.725s (faster)
    - Claude Sonnet 4.5: 2.881s
    - Analysis: For the simplest base module, GPT-4o Mini performed 67% faster. This suggests GPT-4o Mini has lower initialization overhead for simple designs.
2. **4-to-1 Multiplexer (First Hierarchy Level)**:
    - GPT-4o Mini: 5.699s
    - Claude Sonnet 4.5: 3.787s (faster)
    - Analysis: Claude begins to show advantages at this level, being 1.5x faster. The crossover point where Claude's efficiency emerges is between the 2-to-1 and 4-to-1 complexity levels.
3. **8-to-1 Multiplexer (Second Hierarchy Level)**:
    - GPT-4o Mini: 38.301s
    - Claude Sonnet 4.5: 9.617s (much faster)
    - Analysis: This represents a dramatic 4x performance advantage for Claude. The disproportionate increase in GPT-4o Mini's time (6.7x increase from 4-to-1 to 8-to-1) versus Claude's more modest increase (2.5x) suggests Claude handles hierarchical complexity more efficiently.

Comparator Performance Breakdown:

1. **2-bit Comparator (Base Module)**:
    - GPT-4o Mini: 1.671s (faster)
    - Claude Sonnet 4.5: 3.289s
    - Analysis: Similar to multiplexers, GPT-4o Mini is 2x faster for the base module, reinforcing the pattern of lower overhead for simple designs.
2. **4-bit Comparator**:
    - GPT-4o Mini: 4.614s
    - Claude Sonnet 4.5: 4.840s
    - Analysis: Performance converges at this level with only 5% difference. Both models handle this complexity similarly.
3. **8-bit Comparator**:
    - GPT-4o Mini: 5.904s
    - Claude Sonnet 4.5: 5.686s
    - Analysis: Nearly identical performance (4% difference). Unlike multiplexers, comparator scaling doesn't strongly favor either model.
4. **16-bit Comparator**:
    - GPT-4o Mini: 14.610s
    - Claude Sonnet 4.5: 15.625s
    - Analysis: GPT-4o Mini maintains slight advantage (7% faster). The scaling behavior for both models is more linear compared to multiplexers.

Conclusion

This lab successfully demonstrated that modern large language models (GPT-4o Mini and Claude Sonnet 4.5) can reliably generate hierarchical Verilog code for fundamental digital circuits. The hierarchical prompting approach proved effective in creating scalable, modular designs that passed all functional verification tests.

Although the reason for the error in the last module generation was seen in every example. So it is likely a mistake in code somewhere where it's not taking the previous module in for the last module generation.