Verilog BCD-CSA Projects Part 6

Project Overview

This project contains five Verilog modules implementing various digital design concepts including BCD (Binary Coded Decimal) operations, carry-save addition, carry look-ahead addition, and seven-segment display conversion. Each module comes with comprehensive testbenches for verification.

Project Structure

```
verilog-bcd-csa-projects-part6/
├── verilog/
│   ├── bcd_sevensegment_conv.v
│   ├── carry_look_ahead_adder.v
│   ├── bcd_addition.v
│   ├── bcd_x3_conv.v
│   └── carry_save_adder.v
├── testbench/
│   ├── tb_bcd_sevensegment_conv.v
│   ├── tb_carry_look_ahead_adder.v
│   ├── tb_bcd_addition.v
│   ├── tb_bcd_x3_conv.v
│   └── tb_carry_save_adder.v
├── assets/
└── README.md
```

Module Descriptions

1. BCD to Seven-Segment Converter (`bcd_sevensegment_conv.v`)

Purpose: Converts 4-bit BCD input to seven-segment display outputs for digits 0-9.

Module Declaration:

```verilog
module bcd_sevensegment_conv(D,a,b,c,d,e,f,g);
```

Ports:

- `input [3:0] D`: 4-bit BCD input
- `output reg a,b,c,d,e,f,g`: Seven-segment display outputs

Functionality:

- Uses a case statement to map BCD values (0-9) to seven-segment patterns
- Each segment is controlled by individual output bits
- Invalid BCD values (10-15) result in all segments off
- Active high output configuration

Seven-Segment Mapping:

- `0`: `1111110` (displays "0")
- `1`: `0110000` (displays "1")
- `2`: `1101101` (displays "2")
- `3`: `1111001` (displays "3")
- `4`: `0110011` (displays "4")
- `5`: `1011011` (displays "5")
- `6`: `1011111` (displays "6")
- `7`: `1110000` (displays "7")
- `8`: `1111111` (displays "8")
- `9`: `1111011` (displays "9")

2. Carry Look-Ahead Adder (`carry_look_ahead_adder.v`)

Purpose: Implements a 4-bit carry look-ahead adder for faster addition compared to ripple carry adders.

Module Declaration:

```verilog
module carry_look_ahead_adder(
    input [3:0] A, B,
    input Cin,
    output [3:0] S,
    output Cout,
    output [4:0] out
);
```

Ports:

- `input [3:0] A, B`: 4-bit operands

- `input Cin`: Carry input
- `output [3:0] S`: 4-bit sum output
- `output Cout`: Carry output
- `output [4:0] out`: Combined carry and sum output

Key Features:

- Generate (G) and Propagate (P) signal computation
- Parallel carry computation for improved speed
- Reduces propagation delay compared to ripple carry
- Suitable for high-speed arithmetic operations

Algorithm:

1. Generate signals: `G[i] = A[i] & B[i]`
2. Propagate signals: `P[i] = A[i] | B[i]`
3. Carry computation: `C[i+1] = G[i] | (P[i] & C[i])`
4. Sum computation: `S[i] = A[i] ^ B[i] ^ C[i]`

3. BCD Addition (`bcd_addition.v`)

Purpose: Performs BCD addition with automatic correction for results exceeding 9.

Module Declaration:

```verilog
module bcd_addition(a,b,cin,sum,cout);
```

Ports:

- `input [3:0] a,b`: 4-bit BCD operands
- `input cin`: Carry input
- `output [3:0] sum`: 4-bit BCD sum
- `output cout`: Carry output

BCD Addition Algorithm:

1. Perform binary addition: `sum_temp = a + b + cin`

2. If result > 9: add 6 for BCD correction and set carry
3. If result ≤ 9: use result as-is with no carry

BCD Correction Logic:

- Binary addition can produce results 0-19 for single BCD digits
- Results 10-15 need +6 correction to skip invalid BCD codes (A-F)
- Results 16-19 naturally map to correct BCD with carry

4. BCD to Excess-3 Converter (`bcd_x3_conv.v`)

Purpose: Converts 4-bit BCD input to Excess-3 (XS-3) code.

Module Declaration:

```verilog
module bcd_x3_conv(b,x);
```

Ports:

- `input [3:0] b`: 4-bit BCD input
- `output [3:0] x`: 4-bit Excess-3 output

Excess-3 Code:

- Self-complementing code where XS-3 of digit = BCD + 3
- Useful in arithmetic operations and error detection
- Mapping: 0→3, 1→4, 2→5, ..., 9→12

Logic Implementation:

- `x[3]`: MSB logic based on input patterns
- `x[2]`: Second bit with complement and combination logic
- `x[1]`: Third bit using XOR-based patterns
- `x[0]`: LSB as simple complement of input LSB

5. Carry Save Adder (`carry_save_adder.v`)

Purpose: Implements a 4-bit carry-save adder for adding three numbers simultaneously.

Module Declaration:

```verilog
module carry_save_adder(a,b,c,sum,cout,out);
```

Ports:

- `input [3:0] a,b,c`: Three 4-bit operands
- `output [4:0] sum`: 5-bit sum output
- `output cout`: Final carry output
- `output [5:0] out`: Combined result output

Architecture:

1. **Full Adder Module**: Basic building block for bit-wise addition
2. **First Stage**: Parallel full adders generate sum and carry arrays
3. **Second Stage**: Ripple carry addition of sum and carry arrays
4. **Final Result**: 6-bit output accommodating maximum sum

Advantages:

- Reduces critical path delay for multi-operand addition
- Parallel processing in first stage
- Efficient for DSP and arithmetic-heavy applications

Testbench Descriptions

1. BCD Seven-Segment Testbench (`tb_bcd_sevensegment_conv.v`)

Test Cases:

- `0000` (0): Tests zero display
- `1000` (8): Tests digit 8
- `0111` (7): Tests digit 7
- `1001` (9): Tests digit 9
- `1100` (12): Tests invalid BCD input

Verification Points:

- Correct seven-segment patterns for valid BCD

- All-off pattern for invalid BCD inputs
- Timing and signal integrity

2. Carry Look-Ahead Adder Testbench (`tb_carry_look_ahead_adder.v`)

Test Cases:

- `8 + 8 + 1 = 17`: Tests carry generation
- `12 + 2 + 1 = 15`: Tests maximum 4-bit result
- `4 + 7 + 0 = 11`: Tests mid-range values
- `15 + 12 + 1 = 28`: Tests overflow condition
- `15 + 15 + 1 = 31`: Tests maximum possible sum

Verification Points:

- Correct sum calculation
- Proper carry propagation
- Speed advantage over ripple carry (timing analysis)

3. BCD Addition Testbench (`tb_bcd_addition.v`)

Test Cases:

- `1 + 9 + 0 = 10`: Tests BCD correction (carry generation)
- `9 + 9 + 1 = 19`: Tests maximum BCD sum with carry
- `1 + 5 + 1 = 7`: Tests normal BCD addition
- `7 + 9 + 1 = 17`: Tests correction with carry
- `7 + 7 + 0 = 14`: Tests correction without carry

Verification Points:

- Correct BCD sum output
- Proper carry flag generation
- BCD correction logic validation

4. BCD to Excess-3 Testbench (`tb_bcd_x3_conv.v`)

Test Cases:

- All BCD digits 0-9 tested systematically

- Verifies complete conversion table
- Tests all possible input combinations

Expected Results:

- `0→3`, `1→4`, `2→5`, `3→6`, `4→7`, `5→8`, `6→9`, `7→10`, `8→11`, `9→12`

5. Carry Save Adder Testbench (`tb_carry_save_adder.v`)

Test Cases:

- `2 + 9 + 5 = 16`: Tests moderate three-number sum
- `10 + 15 + 13 = 38`: Tests with invalid BCD inputs
- `15 + 15 + 14 = 44`: Tests near-maximum sum
- `12 + 9 + 4 = 25`: Tests mixed input patterns

Verification Points:

- Correct three-operand addition
- Proper carry propagation through both stages
- Output format verification

Usage Instructions

Simulation Setup

1. **Compile all source files** in the `verilog/` directory
2. **Compile testbench files** in the `testbench/` directory
3. **Run simulations** using your preferred Verilog simulator (ModelSim, Vivado, etc.)

Example Simulation Commands (ModelSim)

```bash
# Compile source files
vlog verilog/*.v

# Compile and run specific testbench
vlog testbench/tb_bcd_sevensegment_conv.v
vsim tb7
run -all

# For other testbenches, replace module names accordingly
```

Synthesis Considerations

- All modules are synthesizable for FPGA/ASIC implementation
- Consider timing constraints for carry look-ahead adder
- Seven-segment converter outputs may need level shifting for actual displays
- Add appropriate clock domains if integrating into larger systems

Performance Analysis

Timing Characteristics

1. **BCD Seven-Segment**: Combinational logic, single LUT delay
2. **Carry Look-Ahead**: Faster than ripple carry, ~2-3 gate delays
3. **BCD Addition**: Includes correction logic, moderate delay
4. **BCD to XS-3**: Pure combinational, minimal delay
5. **Carry Save Adder**: Two-stage pipeline, optimized for throughput

Resource Utilization

- **LUTs**: Primarily combinational logic implementation
- **Registers**: Minimal usage (only in BCD addition for intermediate storage)
- **Routing**: Moderate complexity due to parallel operations

Applications

Digital Systems Integration

1. **Calculator Implementations**: BCD arithmetic modules
2. **Display Controllers**: Seven-segment conversion

3. **DSP Applications**: Carry-save adders for multiply-accumulate
4. **Arithmetic Units**: High-speed addition with carry look-ahead
5. **Code Converters**: BCD to Excess-3 for error detection

Educational Value

- **Digital Design Concepts**: Combinational and basic sequential logic
- **Number Systems**: BCD, binary, and Excess-3 representations
- **Arithmetic Algorithms**: Different addition techniques and optimizations
- **Testing Methodology**: Comprehensive testbench development
- **Hardware Description**: Verilog coding best practices

Troubleshooting

Common Issues

1. **Simulation Mismatches**: Check testbench timing and signal assignments
2. **Synthesis Warnings**: Verify all signals are properly driven
3. **Timing Violations**: Consider pipeline stages for high-frequency operation
4. **Display Issues**: Verify seven-segment polarity (active high/low)

Debug Recommendations

- Use waveform viewers to analyze signal transitions
- Add intermediate signal monitors in complex modules
- Verify testbench stimulus covers all functional cases
- Check for race conditions in combinational logic

Conclusion

This project provides a comprehensive foundation for understanding digital arithmetic and display systems. The modular design allows for easy integration into larger systems while maintaining clear interfaces and thorough testing coverage. Each module demonstrates different aspects of digital design, from basic combinational logic to optimized arithmetic architectures.

The included testbenches provide confidence in functionality and serve as examples for developing robust verification environments. This collection serves both educational purposes and practical implementation needs in embedded systems and digital signal processing applications.