ECS 409: Computer Organization – Assignment 2

Adheesh Trivedi

2025-09-10

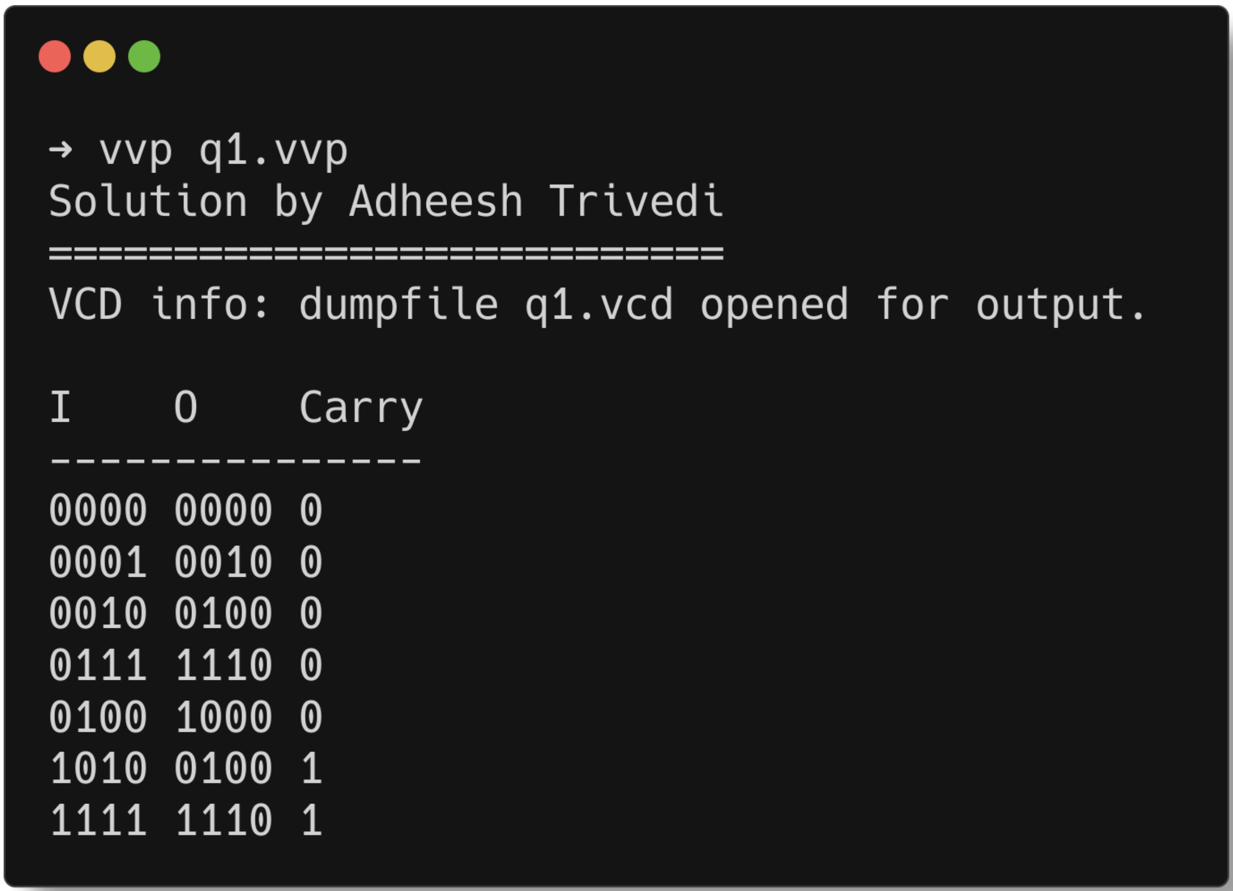
Problem 1: Multiply 4-bit by 2

Behavioral code multiplies input by 2 using shift.

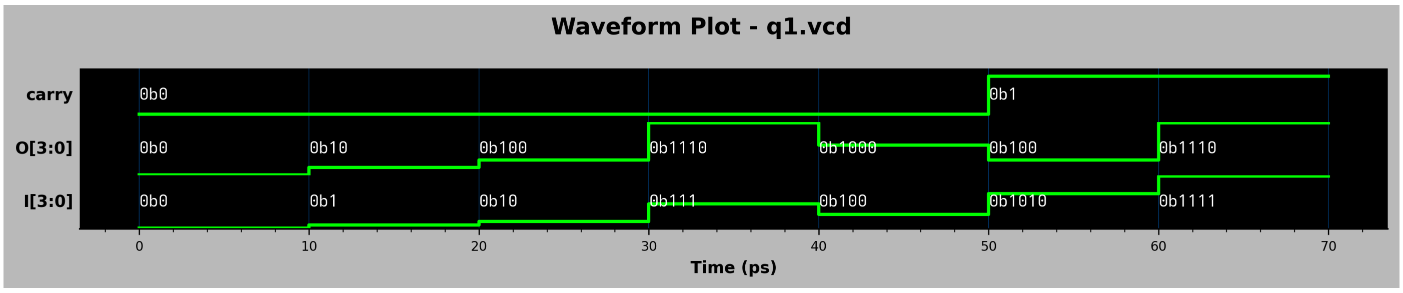
```verilog

module MULTBY2(input [3:0] I, output [3:0] O, output carry);

  assign O = I << 1;
  assign carry = I[3];

endmodule


`timescale 1ns / 1ps

module TEST();
  reg [3:0] I;
  wire [3:0] O;
  wire carry;

  MULTBY2 M(I, O, carry);

  initial begin
    $display("Solution by Adheesh Trivedi");
    $display("===========================");
    $dumpfile("q1.vcd");
    $dumpvars(0, TEST);

    $display("");
    $display("I    O    Carry");
    $display("---------------");
    $monitor("%b %b %b", I, O, carry);

    I = 4'b0000; #10;
    I = 4'b0001; #10;
    I = 4'b0010; #10;
    I = 4'b0111; #10;
    I = 4'b0100; #10;
    I = 4'b1010; #10;
    I = 4'b1111; #10;

  end
```

Multiplier code

Verilog code for 4-bit x2 multiplier



```
→ vvp q1.vvp
Solution by Adheesh Trivedi
===========================
VCD info: dumpfile q1.vcd opened for output.

I    O    Carry
---------------
0000 0000 0
0001 0010 0
0010 0100 0
0111 1110 0
0100 1000 0
1010 0100 1
1111 1110 1
```



Problem 2: 4-to-2 Encoder & 2-to-4 Decoder

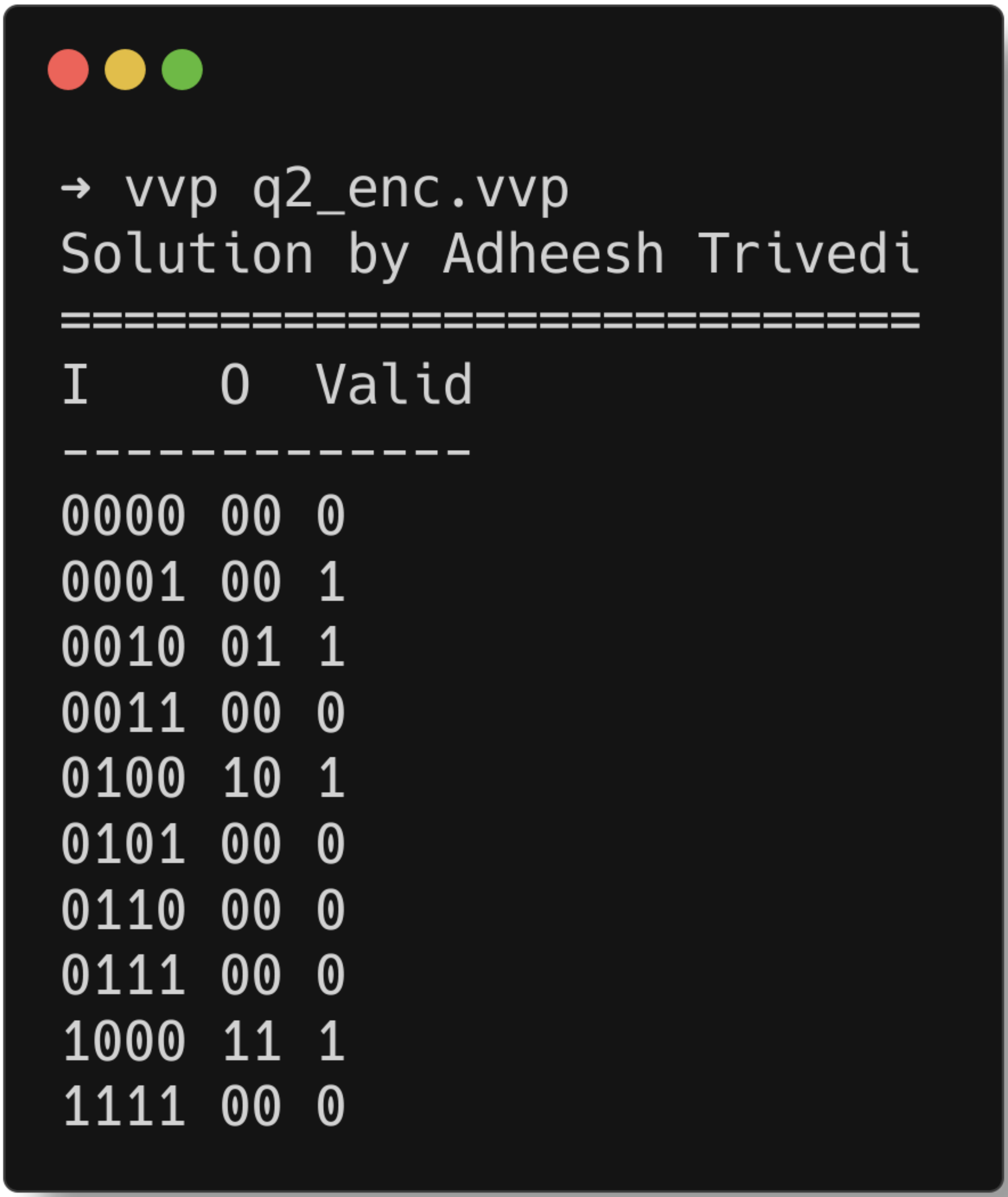
Encoder and decoder implemented behaviorally.

```
module ENC4TO2(
  input [3:0] I,
  output reg [1:0] O,
  output reg valid
);
  always @(I) begin
    case (I)
      4'b0001: begin O = 2'b00; valid = 1; end
      4'b0010: begin O = 2'b01; valid = 1; end
      4'b0100: begin O = 2'b10; valid = 1; end
      4'b1000: begin O = 2'b11; valid = 1; end
      default: begin O = 2'b00; valid = 0; end

    endcase
  end
endmodule


`timescale 1ns / 1ps

module TEST();
  reg [3:0] I;
  wire [1:0] O;
  wire valid;

  ENC4TO2 E(I, O, valid);

  initial begin
    $display("Solution by Adheesh Trivedi");
    $display("===========================");
    $display("I    O  Valid");
    $display("-------------");
    $monitor("%b %b %b", I, O, valid);

    I = 4'b0000; #10;
    I = 4'b0001; #10;
    I = 4'b0010; #10;
    I = 4'b0011; #10;
    I = 4'b0100; #10;
    I = 4'b0101; #10;
    I = 4'b0110; #10;
    I = 4'b0111; #10;
    I = 4'b1000; #10;
    I = 4'b1111; #10;

  end
```

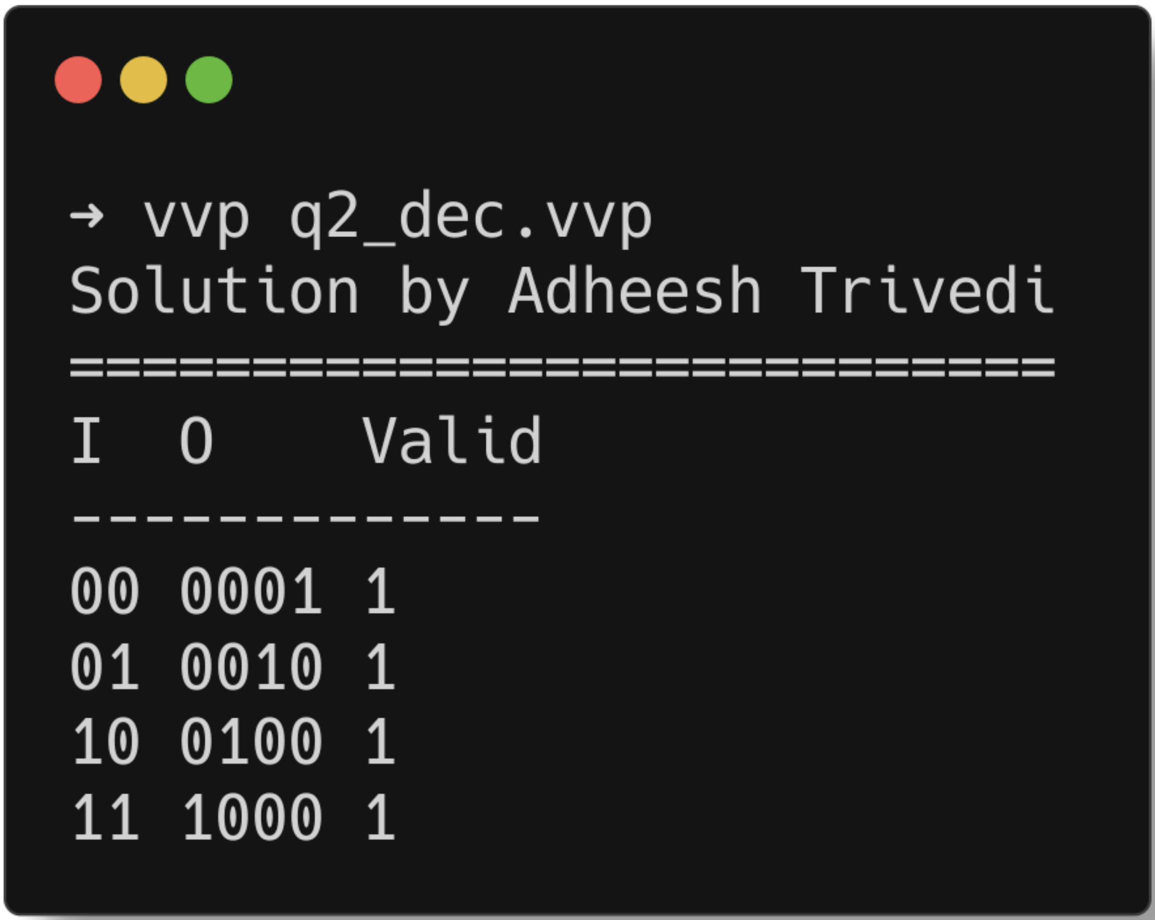
4-to-2 Encoder

```
module DEC2TO4(
  input [1:0] I,
  output reg [3:0] O,
  output reg valid
);
  always @(I) begin
    case (I)
      2'b00: begin O = 4'b0001; valid = 1; end
      2'b01: begin O = 4'b0010; valid = 1; end
      2'b10: begin O = 4'b0100; valid = 1; end
      2'b11: begin O = 4'b1000; valid = 1; end
      default: begin O = 4'b0000; valid = 0; end

    endcase
  end
endmodule


`timescale 1ns / 1ps

module TEST();
  reg [1:0] I;
  wire [3:0] O;
  wire valid;

  DEC2TO4 D(I, O, valid);

  initial begin
    $display("Solution by Adheesh Trivedi");
    $display("===========================");
    $display("I  O    Valid");
    $display("-------------");
    $monitor("%b %b %b", I, O, valid);

    I = 2'b00; #10;
    I = 2'b01; #10;
    I = 2'b10; #10;
    I = 2'b11; #10;

  end
endmodule
```

2-to-4 Decoder



```
→ vvp q2_enc.vvp
Solution by Adheesh Trivedi
===========================
I    O  Valid
-------------
0000 00 0
0001 00 1
0010 01 1
0011 00 0
0100 10 1
0101 00 0
0110 00 0
0111 00 0
1000 11 1
1111 00 0
```



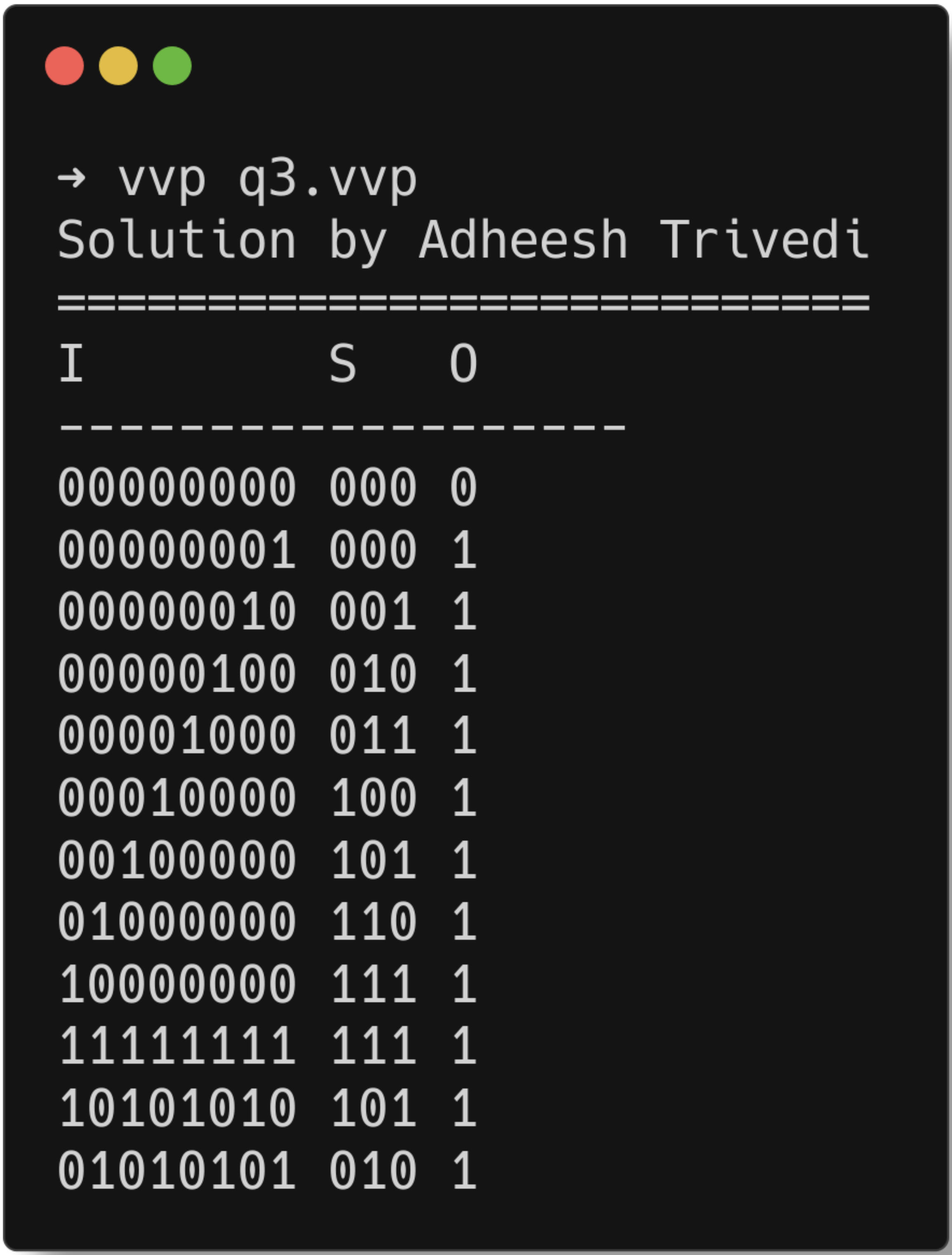
```
➜ vvp q2_dec.vvp
Solution by Adheesh Trivedi
===========================
I  O    Valid
-------------
00 0001 1
01 0010 1
10 0100 1
11 1000 1
```

Problem 3: 8-to-1 Multiplexer

Case statement selects one of 8 inputs.

```
module MUX8TO1(
  input [7:0] I,
  input [2:0] S,
  output reg O
);
  always @(I, S) begin
    case (S)
      3'b000: O = I[0];
      3'b001: O = I[1];
      3'b010: O = I[2];
      3'b011: O = I[3];
      3'b100: O = I[4];
      3'b101: O = I[5];
      3'b110: O = I[6];
      3'b111: O = I[7];
      default: O = 1'b0;

    endcase
  end
endmodule


`timescale 1ns / 1ps

module TEST();
  reg [7:0] I;
  reg [2:0] S;
  wire O;
```

8-to-1 MUX



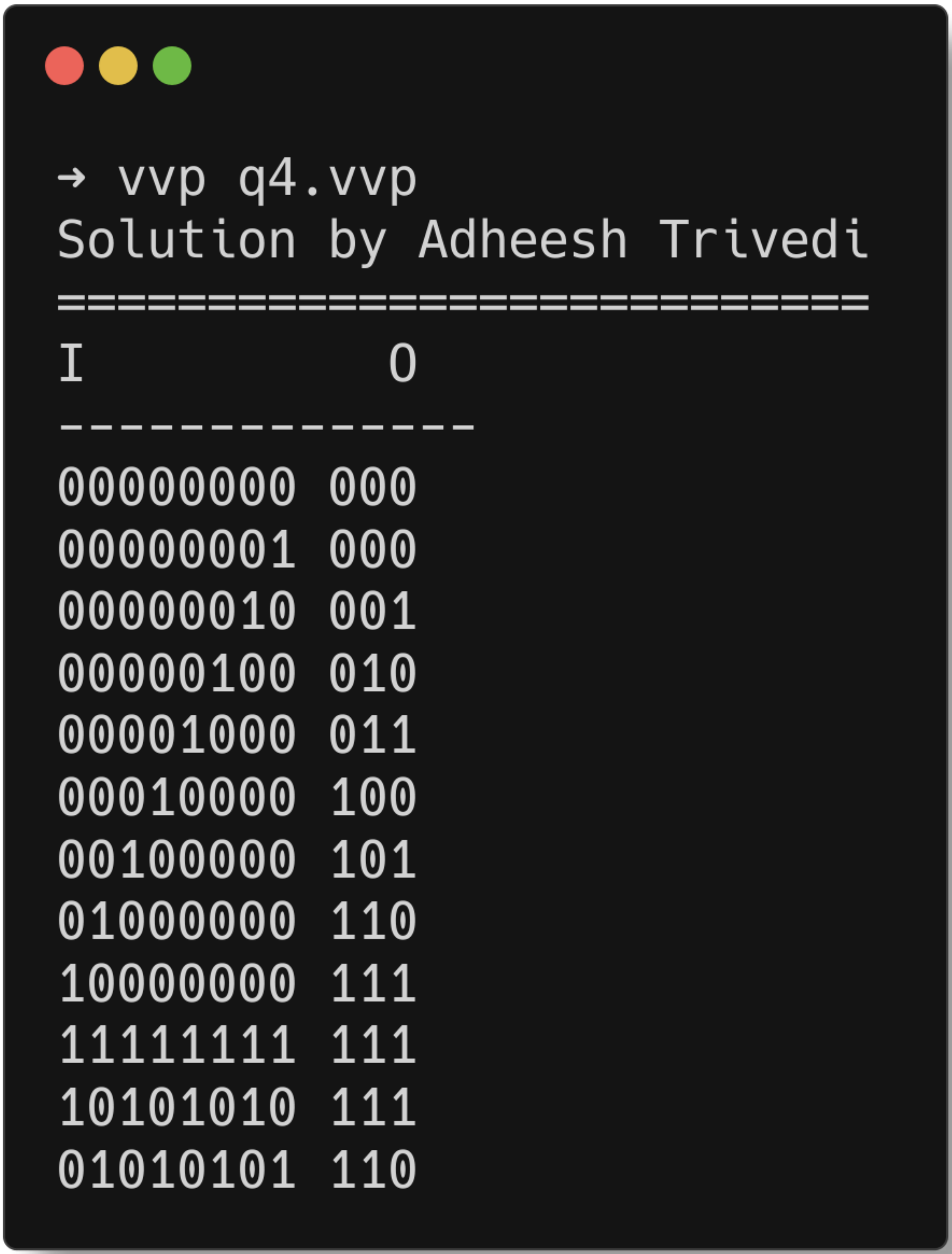
```
→ vvp q3.vvp
Solution by Adheesh Trivedi
===========================
I         S   O
-------------------
00000000 000 0
00000001 000 1
00000010 001 1
00000100 010 1
00001000 011 1
00010000 100 1
00100000 101 1
01000000 110 1
10000000 111 1
11111111 111 1
10101010 101 1
01010101 010 1
```

Problem 4: 8-to-3 Priority Encoder

If-else logic encodes highest active input.

```
module PRIOENC8TO3(
  input [7:0] I,
  output reg [2:0] O
);
  always @(I) begin
    if (I[7]) begin O = 3'b111; end
    else if (I[6]) begin O = 3'b110; end
    else if (I[5]) begin O = 3'b101; end
    else if (I[4]) begin O = 3'b100; end
    else if (I[3]) begin O = 3'b011; end
    else if (I[2]) begin O = 3'b010; end
    else if (I[1]) begin O = 3'b001; end
    else if (I[0]) begin O = 3'b000; end
    else begin O = 3'b000; end

  end
endmodule


`timescale 1ns / 1ps

module TEST();
  reg [7:0] I;
  wire [2:0] O;

  PRIOENC8TO3 P(I, O);

  initial begin
    $display("Solution by Adheesh Trivedi");
```

8-to-3 Priority Encoder



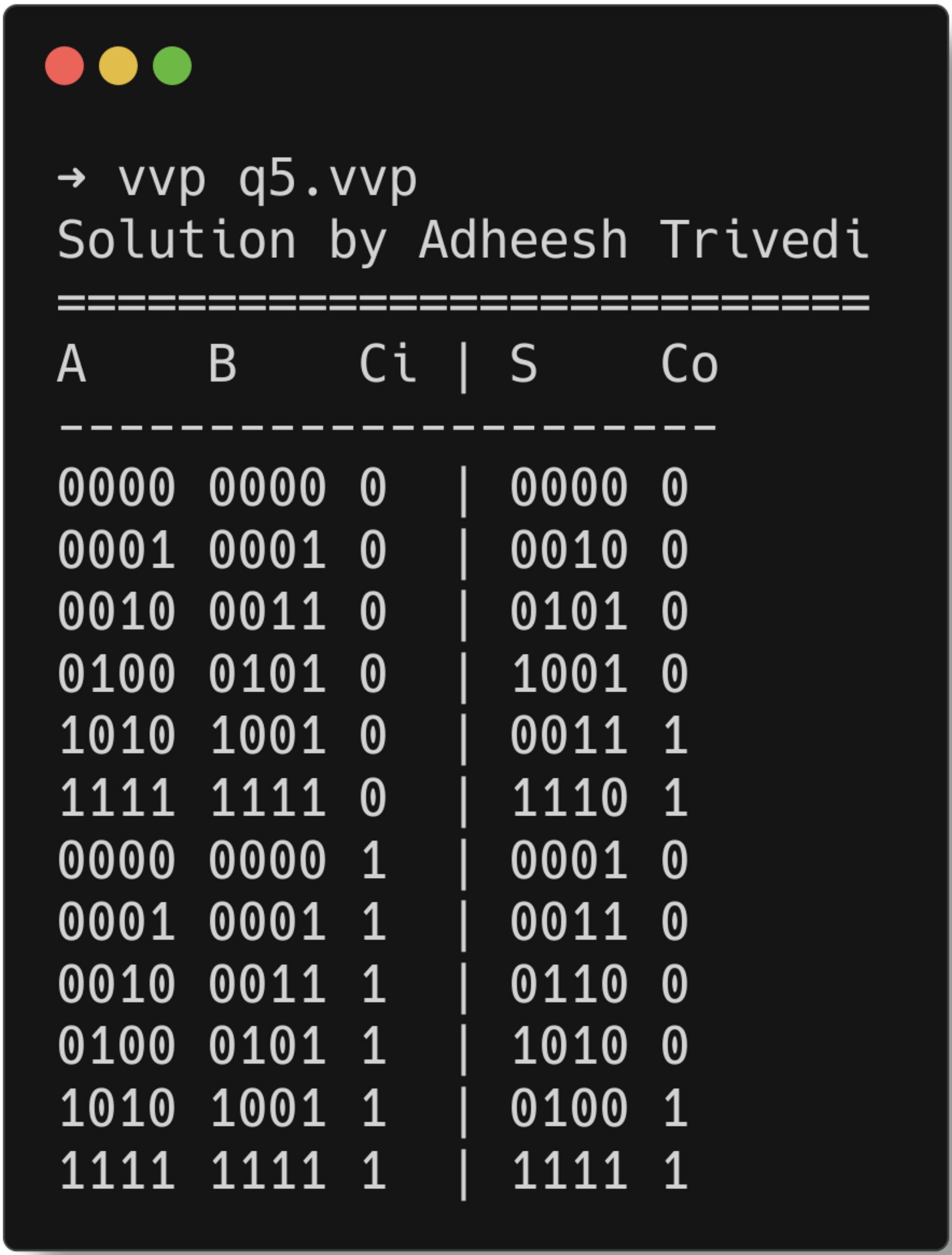
```
→ vvp q4.vvp
Solution by Adheesh Trivedi
============================
I          O
--------------
00000000 000
00000001 000
00000010 001
00000100 010
00001000 011
00010000 100
00100000 101
01000000 110
10000000 111
11111111 111
10101010 111
01010101 110
```

Problem 5: 4-bit Carry Look-Ahead Adder

Adder uses carry look-ahead logic for fast sum.

```
module CARRY_LOOKAHEAD4 (
  input [3:0] A,
  input [3:0] B,
  input Ci,
  output reg [3:0] S,
  output reg Co
);
  reg [3:0] P, G;
  reg [3:0] C;
  integer i;

  always @(*) begin
    G = A & B;
    P = A ^ B;
    C[0] = Ci;

    // Ci+1 = Gi | Pi & Ci
    for (i = 0; i < 3; i++) begin
      C[i+1] = G[i] | (P[i] * C[i]);
    end
    Co = G[3] | (P[3] * C[3]);
    S = P ^ C;

  end
endmodule


`timescale 1ns / 1ps

module TEST();
  reg [3:0] A, B;
  reg Ci;
  wire [3:0] S;
  wire Co;

  CARRY_LOOKAHEAD4 CLA(A, B, Ci, S, Co);

  initial begin
    $display("Solution by Adheesh Trivedi");
```

Carry Look-Ahead Adder



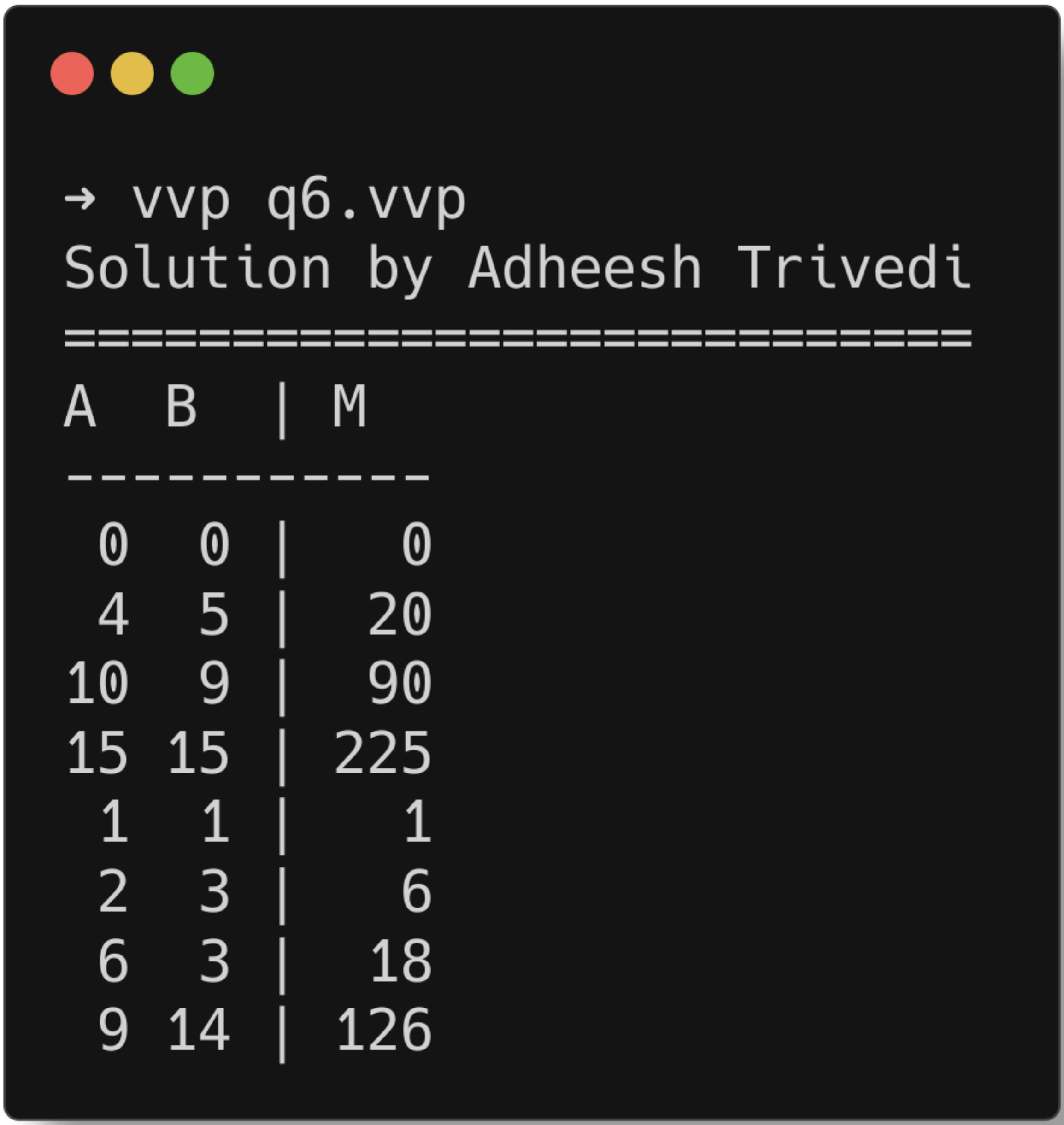
```
→ vvp q5.vvp
Solution by Adheesh Trivedi
===========================
A    B    Ci | S    Co
----------------------
0000 0000 0  | 0000 0
0001 0001 0  | 0010 0
0010 0011 0  | 0101 0
0100 0101 0  | 1001 0
1010 1001 0  | 0011 1
1111 1111 0  | 1110 1
0000 0000 1  | 0001 0
0001 0001 1  | 0011 0
0010 0011 1  | 0110 0
0100 0101 1  | 1010 0
1010 1001 1  | 0100 1
1111 1111 1  | 1111 1
```

Problem 6: 4-bit Wallace Tree Multiplier

Wallace tree structure for fast multiplication.

```
module HALF_ADDER (
  input A,
  input B,
  output S,
  output C
);
  assign S = A ^ B;
  assign C = A & B;

endmodule


module FULL_ADDER (
  input A,
  input B,
  input Ci,
  output S,
  output Co
);

  assign S = A ^ B ^ Ci;
  assign Co = A & B | ((A | B) & Ci);

endmodule

module WALLACE_TREE_MULT4 (
  input [3:0] A,
  input [3:0] B,
  output [7:0] M
);

  // Partial products
  wire [3:0] P0, P1, P2, P3;

  // Generate partial products
  assign P0 = A & {4{B[0]}};
  assign P1 = A & {4{B[1]}};
  assign P2 = A & {4{B[2]}};
  assign P3 = A & {4{B[3]}};

  // Intermediate wires for carries and sums
  wire c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11; // added c11
  wire s1, s2, s3, s4, s5, s6; // removed unused s7,s8,s9

  // First bit (no addition needed)
  assign M[0] = P0[0];

  // Second bit
  HALF_ADDER ha1 (P0[1], P1[0], M[1], c1);
```

Wallace Tree Multiplier



```
➜ vvp q6.vvp
Solution by Adheesh Trivedi
===========================
A  B  | M
-----------
 0  0 |   0
 4  5 |  20
10  9 |  90
15 15 | 225
 1  1 |   1
 2  3 |   6
 6  3 |  18
 9 14 | 126
```