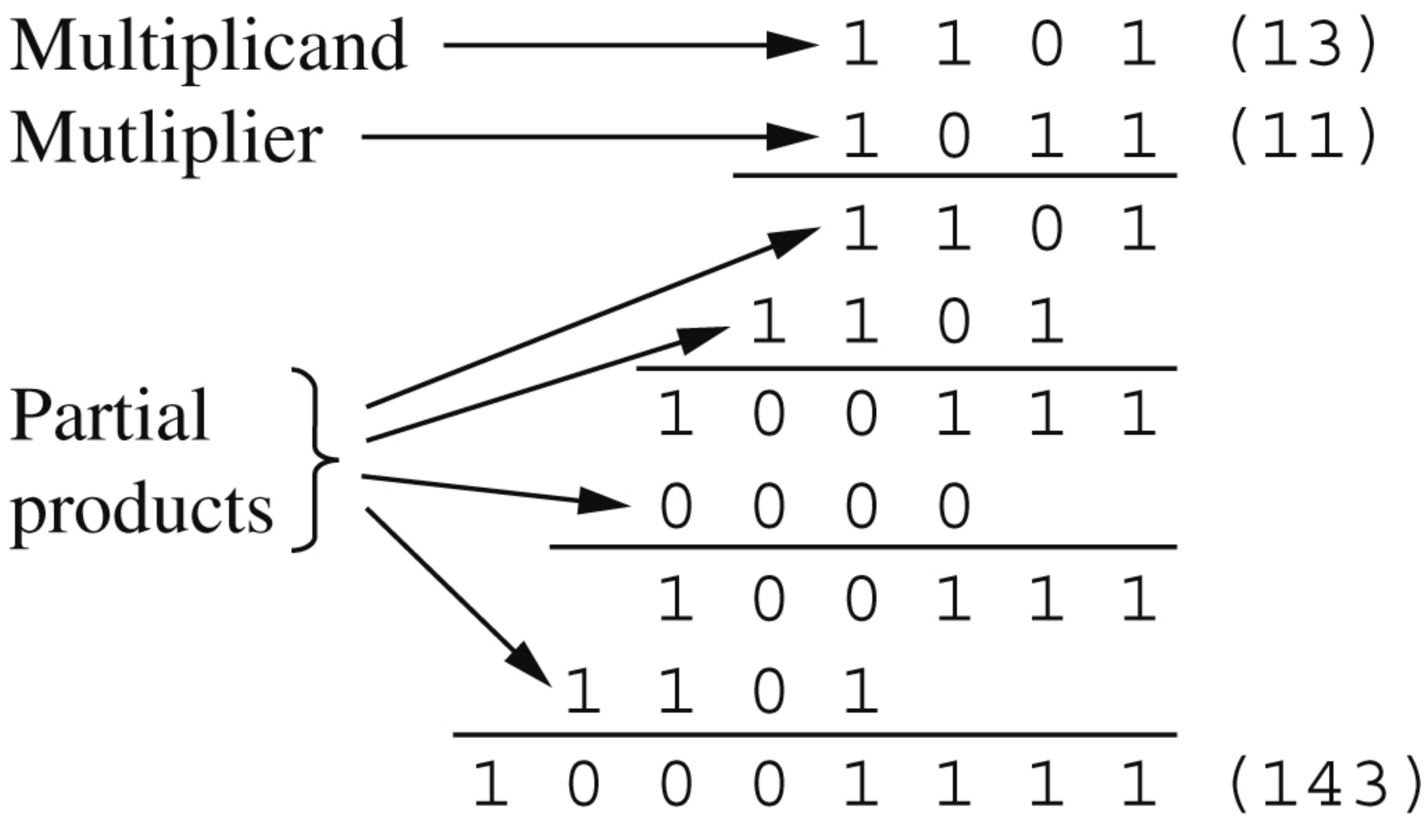
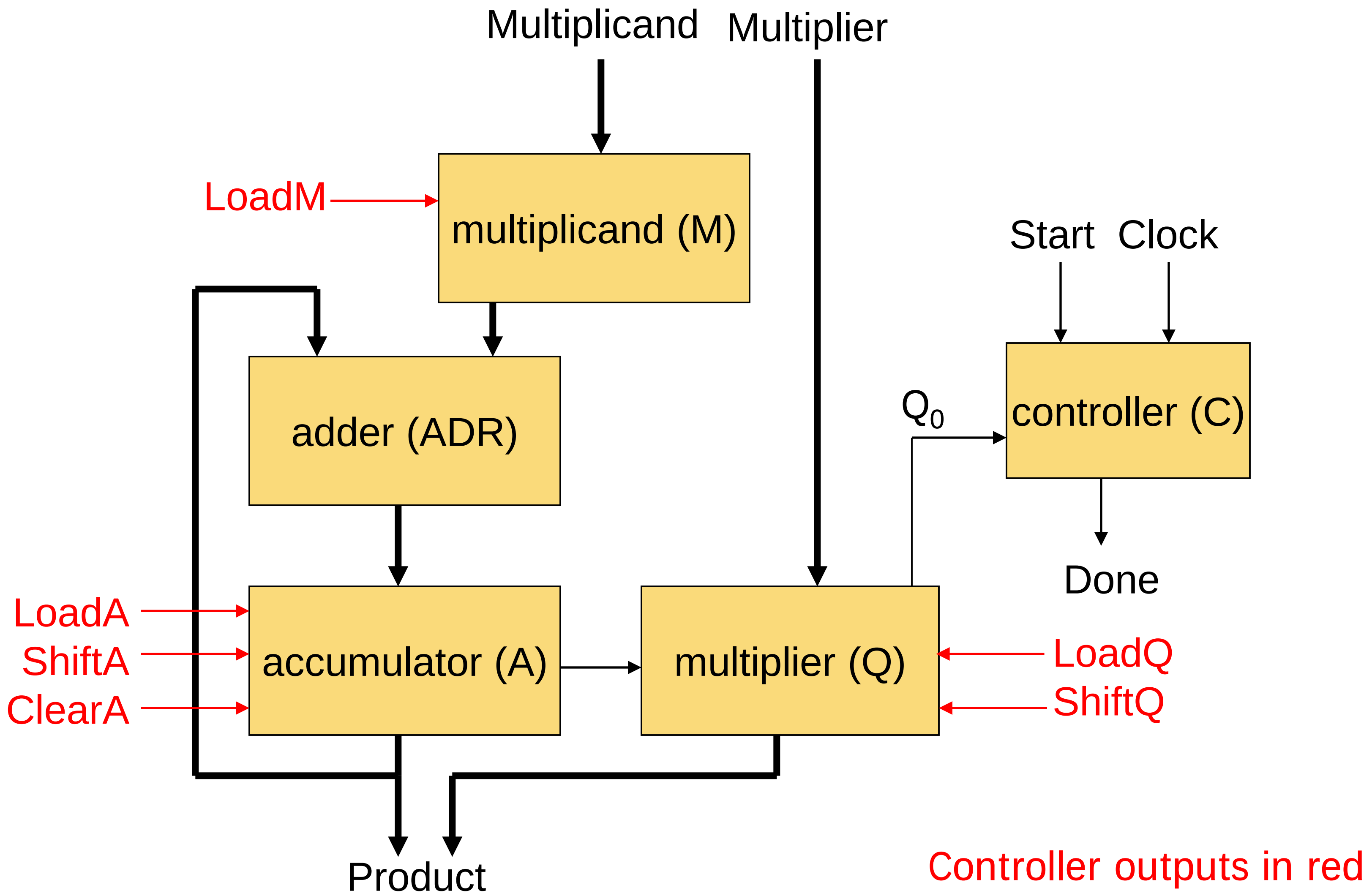
Verilog Modeling for Synthesis

Multiplier Design (Nelson model)

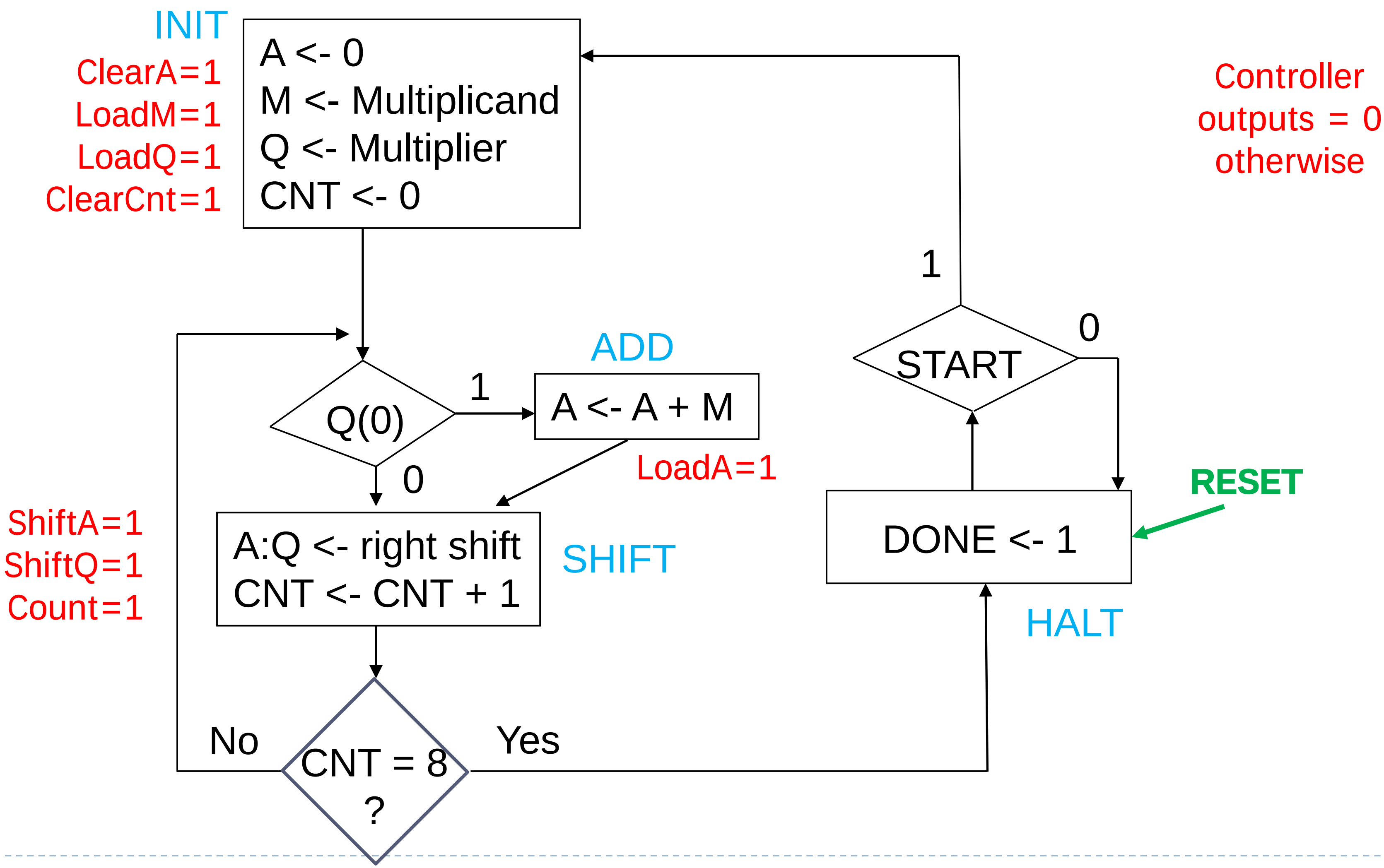
“Add and shift” binary multiplication



System Example: 8x8 multiplier



Multiply Algorithm



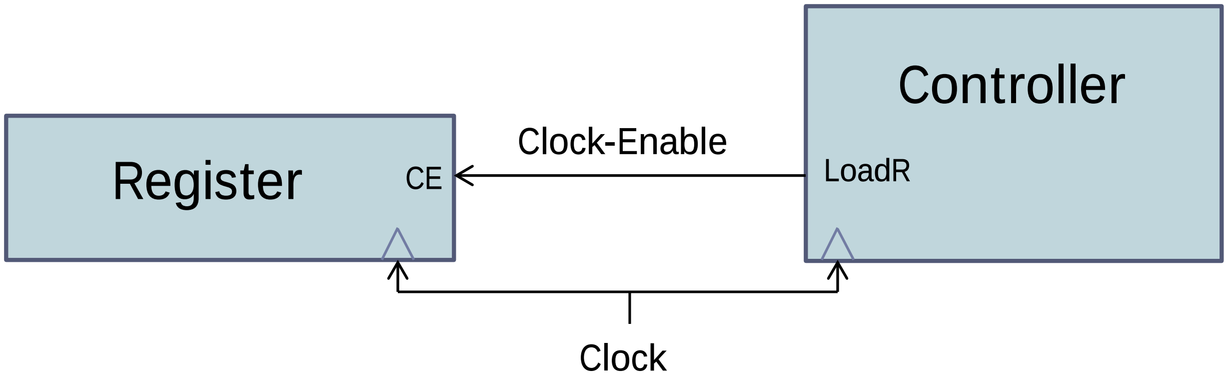
Example: 6 x 5 = 110 x 101

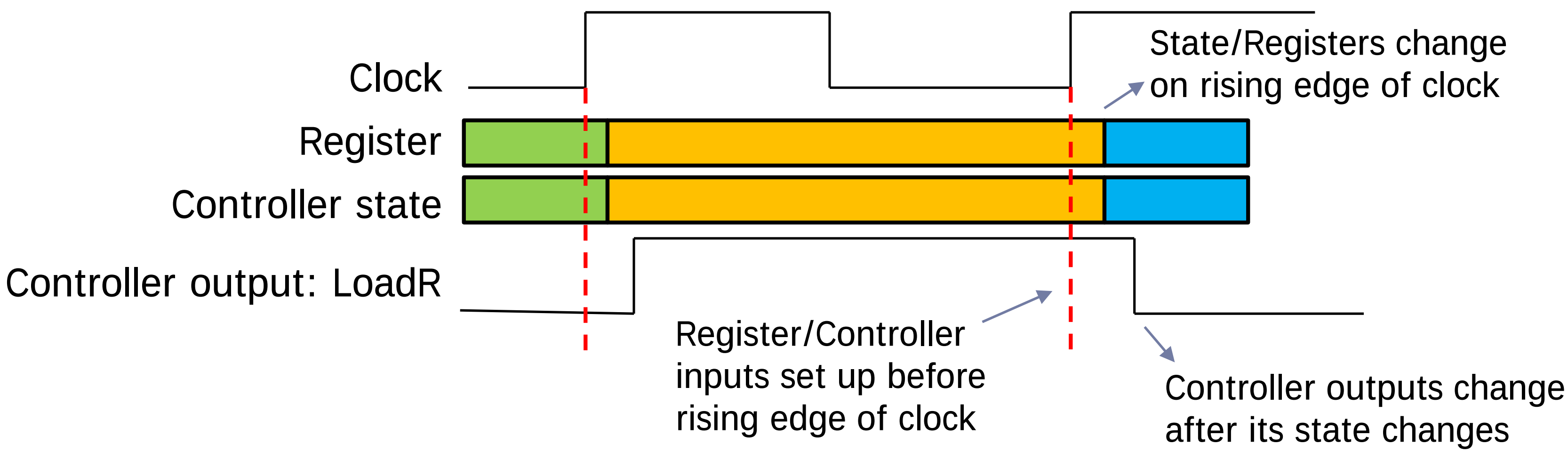
M	A	Q	CNT	State	
110	0000	101	0	INIT	Multiplicand->M, 0->A, Multiplier->Q, CNT=0
	+ 110			ADD	(Since Q0=1) A = A+M
	0110	101	0		
	0011	010	1	SHIFT	Shift A:Q, CNT+1=1 (CNT not 3 yet)
					(skip ADD, since Q0 = 0)
	0001	101	2	SHIFT	Shift A:Q, CNT+1=2 (CNT not 3 yet)
	+ 110			ADD	(Since Q0 = 1) A = A+M
	0111	101	2		
	0011	110	3	SHIFT	Shift A:Q, CNT+1=2 (CNT= 3)
	0011	110	3	HALT	Done = 1

P = 30

Timing considerations

Be aware of register/flip-flop setup and hold constraints





Multiplier – Top Level

```
module MultiplierTop #(parameter N = 4)   //N is data width
   (
    input Clock, Reset,                   //control inputs
    input [N-1:0] Multiplicand,           //N-bit data inputs
    input [N-1:0] Multiplier,
    output [2*N-1:0] Product,             //2N-bit product output
    output Halt                           //indicates product ready
    );
  wire [N-1:0] RegQ, RegM, Sum;           // Q and M register and adder outputs
  wire [N:0] RegA;                        // A register output
  wire Cout, Start, Add, Shift;           // Adder carry and controller outputs

  assign Product = {RegA[N-1:0],RegQ};   //product = A:Q
```

Continued on next slide

Multiplier – Top Level (continued)

```
ShiftReg #(N) M_Reg            // Multiplicand register: load at start
    (.Din(Multiplicand), .Dout(RegM), .ShiftIn(0), .Clock(Clock), .Clear(0), .Shift(0),
     .Load(Start));

ShiftReg #(N) Q_Reg            // Multiplier register: load at start, shift
    (.Din(Multiplier), .Dout(RegQ), .ShiftIn(RegA[0]), .Clock(Clock), .Clear(0), .Shift(Shift),
     .Load(Start));

ShiftReg #(N+1) A_Reg          // Accumulator register: clear, load, shift
    (.Din({Cout,Sum}), .Dout(RegA), .ShiftIn(0), .Clock(Clock), .Clear(Start), .Shift(Shift),
     .Load(Add));

AdderN #(N) Adder              // Accumulator register: clear, load, shift
    (.A(RegA[N-1:0]), .B(RegM), .Cin(0), .Cout(Cout), .Sum(Sum));

MultControl #(N) Ctrl          // Controller
    (.Clock(Clock), .Reset(Reset), .Q0(RegQ[0]), .Start(Start), .Add(Add), .Shift(Shift),
     .Halt(Halt));

endmodule
```

Multi-function N-bit register: clear, load, shift

```
module ShiftReg #(parameter N = 4)      // N is data width
 (
  input [N-1:0] Din,             //parallel inputs
  output reg [N-1:0] Dout,       //register outputs
  input ShiftIn,                 //shift input
  input Clock,                   //clock
  input Clear,                   //clear function select
  input Shift,                   //shift function select
  input Load                     //load function select
 );
always @(posedge Clock)          //rising-edge triggered
   begin
    if (Clear == 1)
          Dout <= 0;                         // clear the register
    else if (Load == 1)
          Dout <= Din;                       // load parallel inputs
    else if (Shift == 1)
          Dout <= {ShiftIn, Dout[N-1:1]};    // shift right
    else
          Dout <= Dout;                      // default is "hold"
   end
endmodule
```

N-bit adder (behavioral)

```
module AdderN #(parameter N = 4)
  (
  input [N-1:0] A, B,                    //N-bit adder inputs
  input Cin,                             //carry input
  output [N-1:0] Sum,                    //N-bit adder output
  output Cout,                           //carry output
  );
  wire temp;                             //temp bit 0

// Cout = bit N+1,  Sum = bits N:1, Cin+1 = carry into bit 1 if Cin=1
  assign {Cout,Sum,temp} = {1'b0, A, Cin} + {1'b0, B, 1'b1};
 endmodule
```

Multiplier Controller

```
module MultControl #(parameter N = 4)   //parameter for bit count
 (
   input Clock, Reset, Q0,              //control and testable inputs
   output Start, Add, Shift, Halt       //controller outputs (Moore machine)
  );

 reg [4:0] state;               //five states (one hot)
 //One-hot state assignments for five states
 parameter StartS=5'b00001, TestS=5'b00010, AddS=5'b00100, ShiftS=5'b01000,
           HaltS=5'b10000;

 reg [N-1:0] Count;    //iteration count
 wire C0;              // C0 = 1 if count < N

 // 2-bit counter for #iterations
 always @(posedge Clock)
    if (Start == 1) Count <= 0;                     // clear in Start state
    else if (Shift == 1) Count <= Count + 1;        // increment in Shift state

 assign C0 = (Count == N-1) ? 1 : 0;                //detect Nth iteration
```

Controller – State transition process

```
//State transitions
always @(posedge Clock, posedge Reset)   //detect positive edge of Clock or Reset
        if (Reset==1) state <= StartS;   //enter StartS state on Reset
        else
          case (state)    // next states
                StartS: state <= TestS;              // go to TestS
                TestS:  if (Q0) state <= AddS;       // go to AddS if Q0=1
                        else state <= ShiftS;        // go to ShiftS if Q0=0
                AddS:   state <= ShiftS;             // go to ShiftS
                ShiftS: if (C0) state <= HaltS;      // go to HaltS if Count = N
                        else state <= TestS;         // go to TestS if Count < N
                HaltS:  state <= HaltS;              // stay in HaltS
          endcase
  // Moore model outputs
  assign Start = state[0];    // Start=1 in state StartS, else 0
  assign Add = state[2];      // Add=1 in state AddS, else 0
  assign Shift = state[3];    // Shift=1 in state ShiftS, else 0
  assign Halt = state[4];     // Halt=1 in state HaltS, else 0
endmodule
```

Multiplier simulation "do file"

```
add wave Clock Reset Multiplicand Multiplier Product
add wave RegM RegQ RegA Sum Cout Start Add Shift Halt
add wave /Multiplier/Ctrl/state  /Multiplier/Ctrl/Count
force Clock 0 0, 1 5 -repeat 10
force Reset 0 0, 1 10, 0 20
force Multiplicand 16#F 0
force Multiplier 16#F 0
run 160
```

Computes 15 x 15 (4-bit inputs, 8-bit product)

Simulation results