SystemVerilog & GCD

- GCD algorithm expects to use positive input values only
  - Negative input value(s) will make the algorithm to loop in a wrong way
  - Using absolute values would help
- Details depend on the used data types
  - *unsigned* – negative values are treated as very large positive numbers in comparisons
  - *signed* – negative values are treated as negative numbers in comparisons

- The following code examples combine unsigned & signed logic types
  - 24 & 27; 24 & -27; -24 & 27; -24 & -27 – GCD is 3 (based on absolute values)
- Waveforms cover all eight cases
  - The first one is interpreting signal as signed decimal values
  - The second one shows hexadecimal values (unsigned)

GCD – unsigned

```
// 1: unsigned & positive inputs
logic unsigned [15:0] x_u1, y_u1, r_u1;
initial begin
  x_u1=24;  y_u1=27;  @(posedge clk);
  while (x_u1!=y_u1) begin // Calculate
    if (x_u1<y_u1) y_u1=y_u1-x_u1;
    else           x_u1=x_u1-y_u1;
    @(posedge clk);
  end
  r_u1=x_u1; // Ready
end
```

```
// 2: unsigned & positive/negative inputs
logic unsigned [15:0] x_u2, y_u2, r_u2;
initial begin
  x_u2=24;  y_u2=-27;  @(posedge clk);
  while (x_u2!=y_u2) begin // Calculate
    if (x_u2<y_u2) y_u2=y_u2-x_u2;
    else           x_u2=x_u2-y_u2;
    @(posedge clk);
  end
  r_u2=x_u2; // Ready
end
```

```
// 3: unsigned & negative/positive inputs
logic unsigned [15:0] x_u3, y_u3, r_u3;
initial begin
  x_u3=-24;  y_u3=27;  @(posedge clk);
  while (x_u3!=y_u3) begin // Calculate
    if (x_u3<y_u3) y_u3=y_u3-x_u3;
    else           x_u3=x_u3-y_u3;
    @(posedge clk);
  end
  r_u3=x_u3; // Ready
end
```

```
// 4: unsigned & negative inputs
logic unsigned [15:0] x_u4, y_u4, r_u4;
initial begin
  x_u4=-24;  y_u4=-27;  @(posedge clk);
  while (x_u4!=y_u4) begin // Calculate
    if (x_u4<y_u4) y_u4=y_u4-x_u4;
    else           x_u4=x_u4-y_u4;
    @(posedge clk);
  end
  r_u4=x_u4; // Ready
end
```

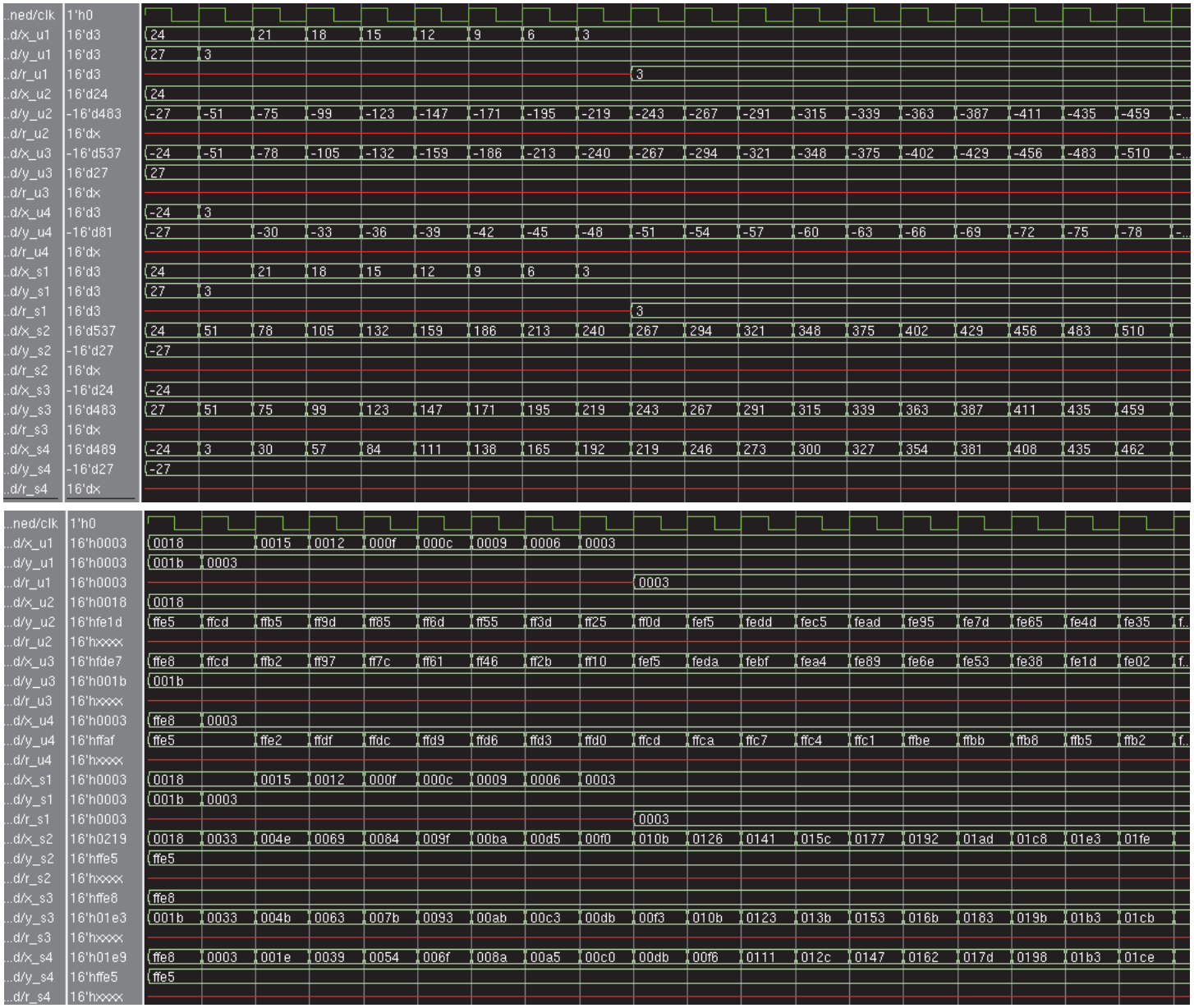
GCD – signed

```
// 5: signed & positive inputs
logic signed [15:0] x_s1, y_s1, r_s1;
initial begin
  x_s1=24;  y_s1=27;  @(posedge clk);
  while (x_s1!=y_s1) begin // Calculate
    if (x_s1<y_s1) y_s1=y_s1-x_s1;
    else           x_s1=x_s1-y_s1;
    @(posedge clk);
  end
  r_s1=x_s1; // Ready
end
```

```
// 6: signed & positive/negative inputs
logic signed [15:0] x_s2, y_s2, r_s2;
initial begin
  x_s2=24;  y_s2=-27;  @(posedge clk);
  while (x_s2!=y_s2) begin // Calculate
    if (x_s2<y_s2) y_s2=y_s2-x_s2;
    else           x_s2=x_s2-y_s2;
    @(posedge clk);
  end
  r_s2=x_s2; // Ready
end
```

```
// 7: signed & negative/positive inputs
logic signed [15:0] x_s3, y_s3, r_s3;
initial begin
  x_s3=-24;  y_s3=27;  @(posedge clk);
  while (x_s3!=y_s3) begin // Calculate
    if (x_s3<y_s3) y_s3=y_s3-x_s3;
    else           x_s3=x_s3-y_s3;
    @(posedge clk);
  end
  r_s3=x_s3; // Ready
end
```

```
// 8: signed & negative inputs
logic signed [15:0] x_s4, y_s4, r_s4;
initial begin
  x_s4=-24;  y_s4=-27;  @(posedge clk);
  while (x_s4!=y_s4) begin // Calculate
    if (x_s4<y_s4) y_s4=y_s4-x_s4;
    else           x_s4=x_s4-y_s4;
    @(posedge clk);
  end
  r_s4=x_s4; // Ready
end
```


decimal
hexadecimal