**Rotary Encoder Explanation**

Rotary encoder signals look like the following, namely square waves out of phase by approximately 90°:

A

B

When the axle turns in one direction, the A signal will lead; when the axle turns in the other direction the B signal will lead. In order to describe hardware to read these signals, it is helpful to note the following pattern:

A

B

0

1

1

0

0

1

1

0

0

0

If A is leading, one has the following set of transitions:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| A | 0 | 1 | 1 | 0 | 0 | etc |
| B | 0 | 0 | 1 | 1 | 0 |

If B is leading, one can read the table backward for the transitions:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| A | 0 | 0 | 1 | 1 | 0 | etc |
| B | 0 | 1 | 1 | 0 | 0 |

The only allowed transitions are ones in the table plus the “no change” transitions. All other transitions are not allowed.

Therefore if one keeps track of the previous pair of signal values (A\_previous, B\_previous) and compares them to the current ones, one can determine the direction and whether one should enable a count change (up or down depending on direction).

The truth tables look like:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A\_previous | B\_previous | A | B | direction | count\_enable |
| 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | No change | 0 |
| 0 | 1 | 0 | 1 | No change | 0 |
| 1 | 0 | 1 | 0 | No change | 0 |
| 1 | 1 | 1 | 1 | No change | 0 |
| All others disallowed | | | |  |  |

One way to encode this table is as follows:

case ({a\_prev,b\_prev,a,b})

4'b0010: {direction,count\_enable} <={1'b1,1’b1};

4'b1011: {direction,count\_enable} <={1'b1,1’b1};

4'b1101: {direction,count\_enable} <={1'b1,1’b1};

4'b0100: {direction,count\_enable} <={1'b1,1’b1};

4'b0001: {direction,count\_enable} <={1'b0,1’b1};

4'b0111: {direction,count\_enable} <={1'b0,1’b1};

4'b1110: {direction,count\_enable} <={1'b0,1’b1};

4'b1000: {direction,count\_enable} <={1'b0,1’b1};

default: {direction,count\_enable} <= {direction,1’b0};

endcase

Using this approach, one could even include the count update:

always @(posedge clk or posedge reset)

begin

if (reset == 1'b1) begin

{dir\_reg, count\_reg} <= {1'b0, 32'b0};

end

else

case ({a\_prev,b\_prev,a,b})

4'b0010: {direction,count\_reg} <={1'b1,count\_reg+32'b1};

4'b1011: {direction,count\_reg} <={1'b1,count\_reg+32'b1};

4'b1101: {direction,count\_reg} <={1'b1,count\_reg+32'b1};

4'b0100: {direction,count\_reg} <={1'b1,count\_reg+32'b1};

4'b0001: {direction,count\_reg} <={1'b0,count\_reg-32'b1};

4'b0111: {direction,count\_reg} <={1'b0,count\_reg-32'b1};

4'b1110: {direction,count\_reg} <={1'b0,count\_reg-32'b1};

4'b1000: {direction,count\_reg} <={1'b0,count\_reg-32'b1};

default: {direction,count\_reg} <={direction,count\_reg};

endcase

end

However, one can write the logic in a succinct way, and this approach is sometimes employed.

For example, see the following [link](http://www.fpga4fun.com/QuadratureDecoder.html). Notice, as indicated in the red, that the direction is in fact equal to (B\_previous XOR A).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A\_previous | B\_previous | A | B | direction | count\_enable |
| 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | No change | 0 |
| 0 | 1 | 0 | 1 | No change | 0 |
| 1 | 0 | 1 | 0 | No change | 0 |
| 1 | 1 | 1 | 1 | No change | 0 |
| All others disallowed | | | |  |  |

The problem with this approach is that it violates the “no change” condition in general. So this succinct method seems to be useful only if one is not outputting the direction.

As for count\_enable, notice that count\_enable is 1 if there are an odd number of 1’s among the four values A\_previous, B\_previous, A, and B and count\_enable is 0 if there are even number of 1’s. This can be written succinctly as:

count\_enable = A\_previous XOR B\_previous XOR A XOR B

Below are two working versions. The first outputs the direction; the second uses the succinct logic but does not output the direction.

module quadrature\_encoder(

input clk,

input reset,

input a,

input b,

output dir,

output [31:0] count

);

reg a\_prev, b\_prev;

reg dir\_reg=1'b0;

reg [31:0] count\_reg=0;

//assumes synchronous inputs and synchronous reset

always @(posedge clk)

begin

a\_prev <= a;

b\_prev <= b;

end

always @(posedge clk or posedge reset)

begin

if (reset == 1'b1) begin

{dir\_reg, count\_reg} <= {1'b0, 32'b0};

end

else

case ({a\_prev,b\_prev,a,b})

4'b0010: {dir\_reg,count\_reg} <={1'b1,count\_reg+32'b1};

4'b1011: {dir\_reg,count\_reg} <={1'b1,count\_reg+32'b1};

4'b1101: {dir\_reg,count\_reg} <={1'b1,count\_reg+32'b1};

4'b0100: {dir\_reg,count\_reg} <={1'b1,count\_reg+32'b1};

4'b0001: {dir\_reg,count\_reg} <={1'b0,count\_reg-32'b1};

4'b0111: {dir\_reg,count\_reg} <={1'b0,count\_reg-32'b1};

4'b1110: {dir\_reg,count\_reg} <={1'b0,count\_reg-32'b1};

4'b1000: {dir\_reg,count\_reg} <={1'b0,count\_reg-32'b1};

default: {dir\_reg,count\_reg} <= {dir\_reg,count\_reg};

endcase

end

assign count=count\_reg;

assign dir = dir\_reg;

endmodule

module quad\_encoder(

input clk,

input reset,

input a,

input b,

output [31:0] count

);

reg a\_previous, b\_previous;

reg dir = 1'b0;

reg [31:0] count = 32'b0;

reg count\_enable = 1'b0;

always @(posedge clk)

begin

a\_previous <= a;

b\_previous <= b;

end

always @(posedge clk or posedge reset)

begin

if (reset == 1'b1)

begin

dir <= 1'b0;

count <= 32'b0;

count\_enable <= 0'b0;

end

else

count\_enable <= a\_previous ^ b\_previous ^ a ^ b;

dir <= b\_previous ^ a ;

if(count\_enable)

begin

if(dir)

begin

count <= count + 32'b1;

end

else

begin

count <= count - 32'b1;

end

end

end

endmodule

Finally, since the inputs A and B will in general not be synchronous, it is advisable to send them through a series of D-flip flops to prevent issues arising from metastability.

module quadrature\_encoder(

input clk,

input reset,

input a,

input b,

output dir,

output [31:0] count

);

reg [2:0] a\_prev;

reg [2:0] b\_prev;

reg dir\_reg=1'b0;

reg [31:0] count\_reg=0;

//assumes debounced inputs and synchronous reset

always @(posedge clk)

begin

a\_prev <= {a\_prev[1:0],a};

b\_prev <= {b\_prev[1:0],b};

end

always @(posedge clk or posedge reset)

begin

if (reset == 1'b1) begin

{dir\_reg, count\_reg} <= {1'b0, 32'b0};

end

else

case ({a\_prev[2],b\_prev[2],a\_prev[1],b\_prev[1]})

4'b0010: {dir\_reg,count\_reg} <={1'b1,count\_reg+32'b1};

4'b1011: {dir\_reg,count\_reg} <={1'b1,count\_reg+32'b1};

4'b1101: {dir\_reg,count\_reg} <={1'b1,count\_reg+32'b1};

4'b0100: {dir\_reg,count\_reg} <={1'b1,count\_reg+32'b1};

4'b0001: {dir\_reg,count\_reg} <={1'b0,count\_reg-32'b1};

4'b0111: {dir\_reg,count\_reg} <={1'b0,count\_reg-32'b1};

4'b1110: {dir\_reg,count\_reg} <={1'b0,count\_reg-32'b1};

4'b1000: {dir\_reg,count\_reg} <={1'b0,count\_reg-32'b1};

default: {dir\_reg,count\_reg} <= {dir\_reg,count\_reg};

endcase

end

assign count=count\_reg;

assign dir = dir\_reg;

endmodule

module quad\_encoder(

input clk,

input reset,

input a,

input b,

output [31:0] count

);

reg [2:0] a\_prev;

reg [2:0] b\_prev;

reg dir = 1'b0;

reg [31:0] count = 32'b0;

reg count\_enable = 1'b0;

always @(posedge clk)

begin

a\_prev <= {a\_prev[1:0],a};

b\_prev <= {b\_prev[1:0],b};

end

always @(posedge clk or posedge reset)

begin

if (reset == 1'b1)

begin

dir <= 1'b0;

count <= 32'b0;

count\_enable <= 0'b0;

end

else

count\_enable <= a\_prev[2] ^ b\_prev[2] ^ a\_prev[1] ^ b\_prev[1];

dir <= b\_prev[2] ^ a\_prev[1] ;

if(count\_enable)

begin

if(dir)

begin

count <= count + 32'b1;

end

else

begin

count <= count - 32'b1;

end

end

end

endmodule