**Instalar o driver para o chip de interface CH347 no Windows**

<https://www.wch.cn/downloads/CH343SER_EXE.html>

**Arquivos fonte para o projeto de comunicação UART no módulo Colorlight i9+**

Arquivo fonte “uart\_rx.v”

module uart\_rx #(parameter DATA\_BITS=8, STOP\_BITS=1, OVERSAMPLING=16)(

input clk, n\_rst, rx,

output ready\_out, valid\_out,

output [DATA\_BITS-1:0] data\_out

);

// Declaração dos estados simbólicos

localparam reg [1:0]

idle = 2'b00,

start = 2'b01,

data = 2'b10,

stop = 2'b11;

// Declaração dos sinais

reg ready\_reg, next\_ready;

reg valid\_reg, next\_valid;

reg [DATA\_BITS-1:0] data\_reg, next\_data;

reg [1:0] state, next\_state;

reg [$clog2((OVERSAMPLING\*2)-1)-1:0] clk\_cnt, next\_clk;

reg [2:0] bit\_cnt, next\_bit;

// Registradores da máquina de estados para o receptor UART

always @(posedge clk, negedge n\_rst) begin

if (~n\_rst) begin

state <= idle;

valid\_reg <= 1'b0;

ready\_reg <= 1'b0;

data\_reg <= 0;

clk\_cnt <= 0;

bit\_cnt <= 0;

end

else begin

state <= next\_state;

ready\_reg <= next\_ready;

valid\_reg <= next\_valid;

clk\_cnt <= next\_clk;

bit\_cnt <= next\_bit;

data\_reg <= next\_data;

end

end

// Lógica combinacional da máquina de estados para o receptor UART

always @(\*) begin

next\_state = state;

next\_ready = ready\_reg;

next\_valid = valid\_reg;

next\_clk = clk\_cnt;

next\_bit = bit\_cnt;

next\_data = data\_reg;

case (state)

idle: begin

next\_ready = 1'b1;

if (~rx) begin

next\_clk = 0;

next\_state = start;

end

end

start: begin

next\_ready = 1'b0;

if (clk\_cnt == (OVERSAMPLING/2)-1) begin

next\_clk = 0;

if (~rx) begin

next\_bit = 0;

next\_state = data;

end

else

next\_state = idle;

end

else

next\_clk = clk\_cnt + 1;

end

data: begin

if (clk\_cnt == OVERSAMPLING-1) begin

next\_clk = 0;

next\_data = {rx, data\_reg[DATA\_BITS-1:1]};

if (bit\_cnt == DATA\_BITS-1) begin

next\_valid = 1'b1;

next\_state = stop;

end

else

next\_bit = bit\_cnt + 1;

end

else

next\_clk = clk\_cnt + 1;

end

stop: begin

next\_valid = 1'b0;

if (clk\_cnt == (OVERSAMPLING\*STOP\_BITS)-1)

next\_state = idle;

else

next\_clk = clk\_cnt + 1;

end

endcase

end

// Direcionamento dos registradores para as saídas

assign data\_out = data\_reg;

assign valid\_out = valid\_reg;

assign ready\_out = ready\_reg;

endmodule

Arquivo fonte “uart\_tx.v”

module uart\_tx #(parameter DATA\_BITS=8, STOP\_BITS=1, OVERSAMPLING=16)(

input clk, n\_rst, uart\_en,

input [DATA\_BITS-1:0] data\_in,

output tx, ready\_out

);

// Declaração dos estados simbólicos

localparam reg [1:0]

idle = 2'b00,

start = 2'b01,

data = 2'b10,

stop = 2'b11;

// Declaração dos sinais

reg tx\_reg = 1'b1;

reg next\_tx;

reg ready\_reg, next\_ready;

reg [DATA\_BITS-1:0] data\_reg, next\_data;

reg [1:0] state, next\_state;

reg [$clog2((OVERSAMPLING\*2)-1)-1:0] clk\_cnt, next\_clk;

reg [2:0] bit\_cnt, next\_bit;

// Registradores da máquina de estados para o transmissor UART

always @(posedge clk, negedge n\_rst) begin

if (~n\_rst) begin

state <= idle;

tx\_reg <= 1'b1;

ready\_reg <= 1'b0;

data\_reg <= 0;

clk\_cnt <= 0;

bit\_cnt <= 0;

end

else begin

state <= next\_state;

tx\_reg <= next\_tx;

ready\_reg <= next\_ready;

clk\_cnt <= next\_clk;

bit\_cnt <= next\_bit;

data\_reg <= next\_data;

end

end

// Lógica combinacional da máquina de estados para o transmissor UART

always @(\*) begin

next\_state = state;

next\_tx = tx\_reg;

next\_ready = ready\_reg;

next\_clk = clk\_cnt;

next\_bit = bit\_cnt;

next\_data = data\_reg;

case (state)

idle: begin

next\_tx = 1'b1;

next\_ready = 1'b1;

if (uart\_en) begin

next\_data = data\_in;

next\_clk = 0;

next\_state = start;

end

end

start: begin

next\_ready = 1'b0;

next\_tx = 1'b0;

if (clk\_cnt == OVERSAMPLING-1) begin

next\_clk = 0;

next\_bit = 0;

next\_state = data;

end

else

next\_clk = clk\_cnt + 1;

end

data: begin

next\_tx = data\_reg[0];

if (clk\_cnt == OVERSAMPLING-1) begin

next\_clk = 0;

next\_data = data\_reg >> 1;

if (bit\_cnt == DATA\_BITS-1)

next\_state = stop;

else

next\_bit = bit\_cnt + 1;

end

else

next\_clk = clk\_cnt + 1;

end

stop: begin

next\_tx = 1'b1;

if (clk\_cnt == (OVERSAMPLING\*STOP\_BITS)-1)

next\_state = idle;

else

next\_clk = clk\_cnt + 1;

end

endcase

end

// Direcionamento dos registradores para as saídas

assign tx = tx\_reg;

assign ready\_out = ready\_reg;

endmodule

Arquivo fonte “blink.v”

module blink #(CLK\_IN=120000000, BLINK\_FREQ=1)(

input clk,

output led\_out

);

localparam DIV = (CLK\_IN/(BLINK\_FREQ\*2))-1;

reg [$clog2(DIV)-1:0] clk\_cnt = 0;

reg led\_reg = 1'b0;

always @(posedge clk)begin

if (clk\_cnt == DIV) begin

clk\_cnt <= 0;

led\_reg <= ~led\_reg;

end

else

clk\_cnt <= clk\_cnt + 1;

end

assign led\_out = led\_reg;

endmodule

Arquivo fonte “clock\_div.v”

module clock\_div #(parameter CLK\_IN=120000000, CLK\_OUT=60000000)(

input clk\_in,

output clk\_out

);

// Declaração do registrador para contagem de clock

localparam DIV = ((CLK\_IN/CLK\_OUT)/2)-1;

reg [$clog2(DIV):0] clk\_div = DIV;

// Divisor de frequência de clock (gera sinal clk\_reg)

reg [9:0] clk\_cnt = 0;

reg clk\_reg = 1'b0;

always @(posedge clk\_in) begin

if (clk\_cnt == clk\_div) begin

clk\_reg <= ~clk\_reg;

clk\_cnt <= 0;

end

else

clk\_cnt <= clk\_cnt + 1;

end

// Direcionamento do registrador para a saída

assign clk\_out = clk\_reg;

endmodule

Arquivo fonte “start\_n\_rst.v”

module start\_n\_rst(

input clk\_in,

output n\_rst\_out

);

reg [15:0] n\_rst\_reg = 16'h01;

always @(posedge clk\_in) begin

n\_rst\_reg <= n\_rst\_reg >> 1;

end

assign n\_rst\_out = ~n\_rst\_reg[0];

endmodule

Arquivo fonte “colorlight.xdc”

# System clock (xtal 25 MHz)

set\_property -dict {PACKAGE\_PIN K4 IOSTANDARD LVCMOS33} [get\_ports dev\_clk]; #IO\_L13P\_T2\_MRCC\_35 xtal\_25\_MHz

# Board LED D2

set\_property -dict {PACKAGE\_PIN A18 IOSTANDARD LVCMOS33} [get\_ports led]; #IO\_L17P\_T2\_16 LED\_D2

# UART

set\_property -dict {PACKAGE\_PIN R3 IOSTANDARD LVCMOS33} [get\_ports tx]; #IO\_L3P\_T0\_DQS\_34 P5\_16

set\_property -dict {PACKAGE\_PIN M3 IOSTANDARD LVCMOS33} [get\_ports rx]; #IO\_L16P\_T2\_35 P5\_15