分类号_____ 密级_____

UDC 注 1 _____

南京理工大学

NANJING UNIVERSITY OF SCIENCE & TECHNOLOGY

结 课 论 文

基于 Vivado 实现的 Biss-C 接口协议仿真

（题名和副题名）

刘裕

（作者姓名）

指导教师姓名 李洪涛 **教 授**

学 位 类 别 工学硕士

学 科 名 称 FPGA 原理与应用

研 究 方 向 通信与信息系统

论文提交时间 **2017.12.14**

注 1：注明《国际十进分类法 UDC》的类号。

结 课 论 文

基于 Vivado 实现的 Biss-C 接口协议仿真

作　　者：刘裕

指导教师：李洪涛　教 授

南 京 理 工 大 学

2017 年 12 月

Dissertation

A research of Biss-C Interface

By

Yu. Liu

Supervised by Prof. Hongtao Li

Nanjing University of Science & Technology

December, 2017

声　明

本学位论文是我在导师的指导下取得的研究成果，尽我所知，在本学位论文中，除了加以标注和致谢的部分外，不包含其他人已经发表或公布过的研究成果，也不包含我为获得任何教育机构的学位或学历而使用过的材料。与我一同工作的同事对本学位论文做出的贡献均已在论文中作了明确的说明。

研究生签名：刘裕　　2017 年 12 月 14 日

学位论文使用授权声明

南京理工大学有权保存本学位论文的电子和纸质文档，可以借阅或上网公布本学位论文的部分或全部内容，可以向有关部门或机构送交并授权其保存、借阅或上网公布本学位论文的部分或全部内容。对于保密论文，按保密的有关规定和程序处理。

研究生签名：刘裕　　2017 年 12 月 14 日

摘　　要

本文研究了 Biss-C 单向传输协议，根据协议编写 verilog 程序并通过 Vivado 仿真验证协议。

首先，研究了 Biss-C 单向模式的协议原理。BiSS-C 通信协议是由主设备 Master 与从设备 Client 通过串行传输方式建立，周期性读取数据的通信协议。其次重点研究 BiSS-C 协议中数据传输周期的状态机，并对其周期中的逻辑进行研究仿真。

关键词：Biss-C，FPGA，Vivado，状态机

Abstract

In this paper, biss-c one-way transmission protocol is studied. Verilog program is written according to the protocol and the protocol is verified through the Vivado simulation.

First, the principle of biss-c one-way mode is studied.Biss-c communication protocol (biss-c) is a communication protocol that is established by the Master device Master and from the device Client through serial transmission and periodically reads the data.Secondly, the state machine of data transmission cycle in biss-c protocol is studied, and the logic of its cycle is studied and simulated.

Key word: Biss-C , FPGA , Vivado , FSM

目　录

图表目录

1 绪论

BiSS 是一个给传感器和执行器的开放数字接口。该接口提供双向快速通信，使用简单硬件实现 (跟工业标准 SSI 接口兼容)。BiSS 适合实时数据采集，而且带安全功能例如 CRC 校验保护数据传送。

1.1 Biss 发展历史

长期以来，编码器制造商靠模拟信号或简单数字增量信号接口传送位置信息。随着时间的流逝和编码器技术的发展，带新处理方法和高集成度，让编码器生成高分辨率位置数据，而且增加了先进的功能例如命令，寄存器通信等等。因此需要实现一个新的接口，带能适合此高端设备的特性。SSI 是一个过时的工业标准，无法满足制造商的速度和功能要求。少数制造商推出了自己的协议，但是封闭性很强，不同牌子产品存在不兼容性。

2002 年 iC-Haus 推出 BiSS 开放接口。BiSS 目的是提供给传感器和执行器的双向快速通信标准，而且保留与 SSI (Synchronous Serial Interface)标准接口硬件兼容。在超过 10 年的时间里 BiSS 使用不断的发展，目前超过 300 家授权设备制造商，包括工业领先如 巴鲁夫 (Balluff)，堡盟 (Baumer)，丹纳赫 (Danaher Motion)，库伯勒 (Kübler)，亨士乐 (Hengstler)，Höhner，倍加福 (Pepperl+Fuchs)，雷尼绍 (Renishaw)，施耐德电气 (Schneider Electric)，安川电机 (Yaskawa)，禹衡光学 (Yuheng Optics) 和其它在使用 BiSS 接口。

1.2 Biss 特点和优点

BiSS 接口是专用实时传感器和执行器接口，和通用现场总线不同。BiSS 接口主要特性如下：

1.2.1 开放 (Open source)

与市场上的私有的接口不同，BiSS 是完全开放。不同制造商能开发同样接口产品，提高不同牌子的兼容性。此优点不仅造福了能直接使用高性能多功能接口的制造商，而且也造福了用户有多产品选择和不靠一个制造商的价格垄断。

1.2.2 使用两个单向信号提供快速串行同步通信

BiSS 是能达到高速度传送的串行接口，长距离用标准 RS422 收发器速度一般高于 10 MHz，短距离用 LVDS 连接一般高于 100 MHz。

1.2.3 单向或双向

BiSS 按产品要求提供不同操作模式。简单单向传送用 BiSS-C Unidirectional，永久双向模式 (连续工作，无需模式转换) 用 BiSS-C 加两条线实现单通道传感器和执行器通信。

1.2.4 线延迟补偿

工业应用要求通过长线在多噪音环境的可靠通信。BiSS 接口带线延迟补偿，经过长距离后还能达到高速度，因此适合此应用。

1.2.5 安全功能

BiSS 接口满足安全应用要求，带安全功能，如：从传感器到控制器的错误和报警信息和传送数据的 16 位 CRC 校验。

1.2.6 组网功能

部分应用需要多传感器数据同步采集，例如多轴运动控制。BiSS 接口提供组网功能，将所有传感器串联到控制器的一个数据通道，方便控制器采集全部数据。

1.2.7 设备描述

BiSS 接口带设备电子识别，连接传感器到控制器后，控制器就自动鉴定连接的传感器参数 (Plug-and-Play)。通过 EDS (Electronic DataSheet, 信息存在传感器内存)，XML 文件或 BiSS Profile 里存的标准描述数据，设备的所有参数能直接传送到控制器，控制器读取设备描述后能自动修改其自身参数以配合该设备。BiSS 授权设备制造商接收独特的身份识别 (Manufacturer ID)，以便任何控制器可识别此产品制造商。

1.3 BiSS 协议帧结构

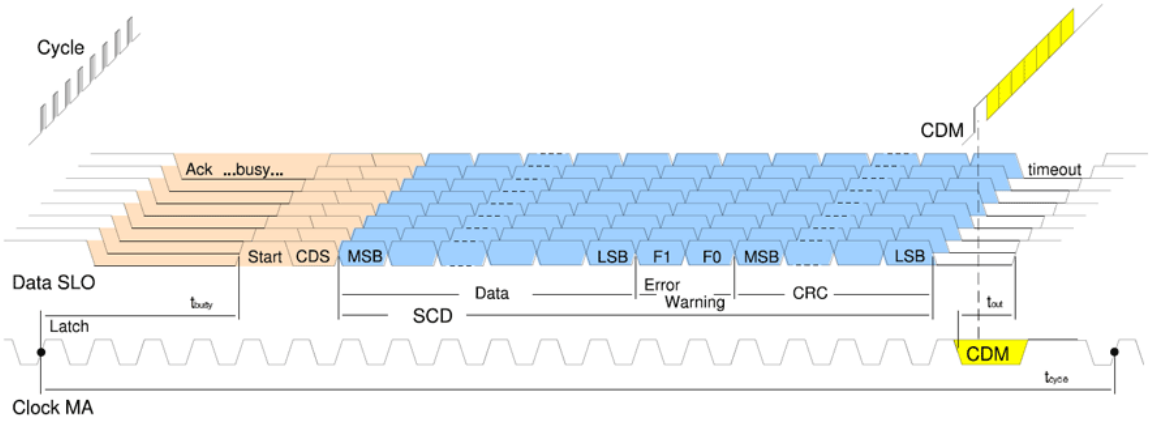

图 1.1 BiSS 协议帧结构示意图

BiSS 协议帧结构如图 1.1 所示，在 BiSS 通信下，采集数据的控制器 (控制操作)叫 BiSS Master (主机)，提供数据的传感器叫 BiSS Slave (从机)。执行器也算是 BiSS Slave，但是接收 BiSS Master 传送的数据。

2 BiSS-C 单向传输协议的研究

2.1 BiSS-C 协议定义

BiSS C 模式（单向）是一种用于从光栅采集位置数据的快速同步串行接口。

它是一种主-从接口。主接口控制位置获取时序和数据传输速度，而光栅为从接口。接口由两个单向差分线耦组成。

- “MA”将位置采集请求和时序信息（时钟）从主接口传输到光栅
- “SLO”将位置数据从光栅传输到与 MA 同步的主接口。

传输数据格式如图 2.1.1 所示。

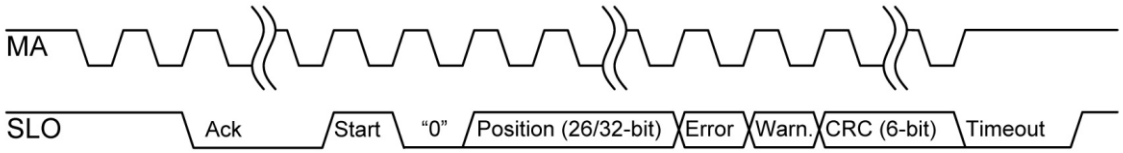

图 2.1.1 BiSS-C 数据格式示意图

2.2 典型的请求循环进程

BiSS-C 传输中存在一种典型的请求循环进程，将从机的数据通过串口方式获取，其典型的请求循环进程如下所述：

1. 当空闲时，主接口使 MA 线保持高电平。光栅通过使 SLO 线保持高电平显示它已准备就绪。

2. 主接口通过开始在 MA 上传输时钟脉冲来请求位置采集。

3. 光栅通过将 MA 的第二上升沿的 SLO 线设为低电平做出响应。

4. 完成 “Ack” 周期后，光栅将数据传输到与时钟同步的主接口，如上图所示。

5. 当所有数据都传送完毕，主接口停下时钟，将 MA 线设为高电平。

6. 如果光栅尚未准备进行下一个请求周期，它会将 SLO 线设为低电平（超时周期）。

7. 当光栅准备进行下一请求周期时，它通过将 SLO 线设为高电平的方式提示主接口。

2.3 数据描述

数据结构依次由 Ack、起始、“0”、位置（26 或 32 位）、误差（1 位）、警告（1 位）、位置数据 CRC（6 位）组成。下面依次进行说明。

2.3.1 Ack

这是读数头计算绝对位置的时间段。 参见下面的时序信息表。

2.3.2 起始和 “0”（每个为 1 位）

光栅传输起始位，发信号给主接口开始传输数据。

起始位始终为高电平，“0” 位始终为低电平。

2.3.3 位置（26 或 32 位）

绝对位置数据为二进制格式，首先发送给 MSB。对于圆光栅，每转正好有 2^n 个脉冲，之后脉冲数溢出绕回到零。[忽略位置数据的最小有效位可能获得较低的分辨率。]

2.3.4 误差（1 位）

误差位低电平有效：“1” 表示传输的位置信息已被读数头的内置安全校验算法校验，结果正确；“0”表示内部检查失败，位置信息不可信。对于有温度感应的 RESOLUTE 系统，如果温度超过产品最高指标，错误位也设为 “0”。

2.3.5 警告（1 位）

警告位低电平有效：“0” 表示应对光栅尺（及/或读数窗口）进行清洁。

请注意，警告位并不表示位置数据可信。只有错误位才能用作此目的。

2.3.6 位置数据 CRC（6 位）

位置、错误及警告数据的 CRC 多项式为：x6 + x1 + x0。它先被传输为 MSB，然后转换。 起始位和 “0” 位从 CRC 计算中忽略。

2.3.7 超时

RESOLUTE 光栅每 40 µs 可以采集一个新的位置读数（最大请求循环率 25 kHz)。因此 40 µs 必须是在一个请求循环开始和另一个请求循环开始之间的时间。但是，数据传输有可能在 40 µs 过去之前完成。在这种情况下，光栅通过将 SLO 线保持低电平直至 40 µs 过后的方式向主接口发信号。这是超时期间。

2.3.7 重置光栅

主接口可能通过停止时钟及将 MA 线设置为高电平的方式,在请求循环过程中随时重置光栅。光栅会用两到三个时钟循环对此进行检测，然后会立即开始超时期间。超时操作与上述方法相同。

3 BiSS-C 设计实现与仿真

3.1 设计思路

由前章节分析可知项目为时序项目，因此应设计为时序电路。

首先将 FPGA 输入的 100MHz 时钟分频为项目的要求时钟 10MHz，之后设计状态机，最后根据每个状态设计相应的程序将串行信号接收处理。

3.1.1 模块总接口设计

BiSS 模块分为输入与输出，相关接口如表 3.1.1 所示：

表 3.1.1 模块信号接口表

接口	I/O	接口解释
I_reset_n	输入	系统重置
I_sys_clk	输入	系统时钟
I_biss_slo	输入	BiSS SLO
O_biss_ma	输出	BiSS MA
O_angle_ena	输出	数据使能
O_angle_data	输出	位置数据
O_angle_error	输出	误差
O_angle_warning	输出	警告
O_angle_crc	输出	位置数据 CRC

因此顶层设计图如图 3.1.1 所示：

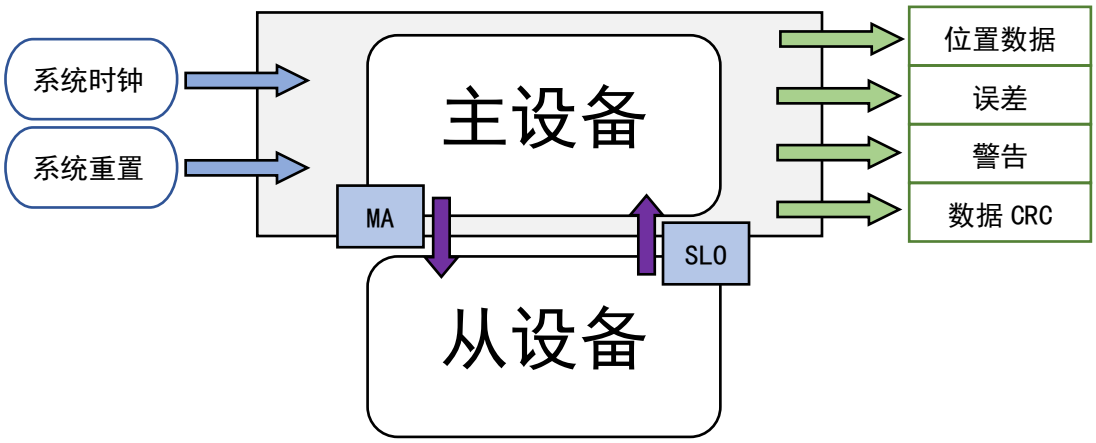

图 3.1.1 BiSS 顶层设计示意图

3.1.2 状态机设计

根据系统状态，大致可以分为空闲状态，等待状态，应答状态以及工作状态。状态机详情由表 3.1.2 所示。

表 3.1.2 状态机状态表

状态标识符	状态值	状态解释
P_STATE_IDLE	4'b0001	空闲状态
P_STATE_WAIT	4'b0010	等待状态
P_STATE_ACK	4'b0100	应答状态
P_STATE_WORK	4'b1000	工作状态

3.1.3 实现细节

为了增强整个项目的时钟驱动能力，选用 BUFG 模块将寄存器转化为连线资源。另外增加数据使能位，标识数据已接收完毕。由于接收串行信号，因此采用计数器配合 case()的方式实现数据寄存器按位赋值的功能。本次位置数据采用 26bit 来记录。

本项目使用 Vivado 2017.1 软件，并使用本软件自带的程序进行仿真。

3.2 仿真结果

仿真文件规定先重置系统，之后按照 BiSS-C 数据结构变换 I_biss_slo 信号。单次接收周期仿真如图 3.2.1 所示。

图 3.2.1 BiSS-C 数据接收时序仿真图

由仿真图可以看出系统重置后，系统在初始化阶段，状态机状态为空闲状态，且由于 I_biss_slo 信号由高电平变为低电平，所以进入了等待状态；当 I_biss_slo 信号由低电平变为高电平后，状态机改变状态为应答状态(ACK)，之后便开始数据接收。

对于仿真接收数据部分，由仿真图 3.2.2 可以看到位置数据由 26 个周期依次从高位到低位接收，之后接收误差、警告及 6 bit 位置数据 CRC，共 34 bit，与设计预期相符。

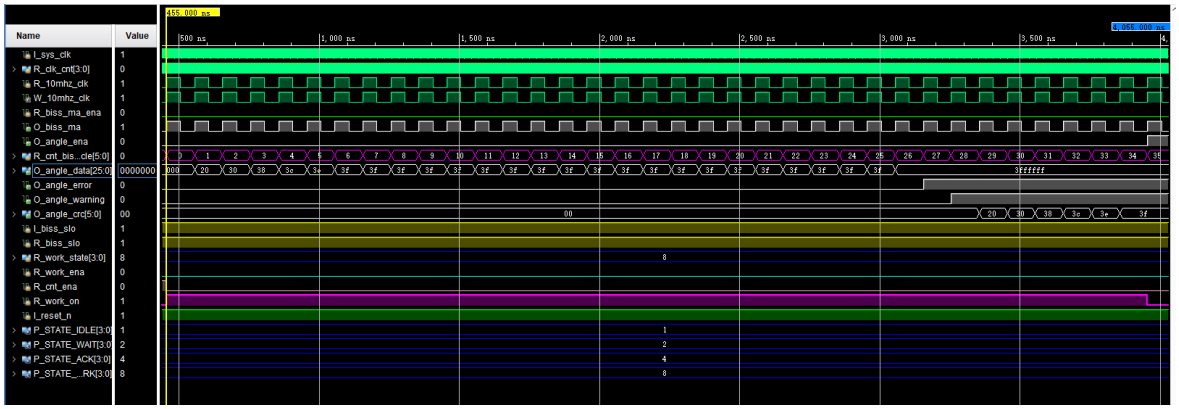

图 3.2.2 BiSS-C 位置数据接收时序局部仿真图

为了证明多周期的可重现性，做了带有两个周期的第二次仿真，并对 I_biss_slo 做了一定的变换，仿真结果如图 3.2.3 所示。

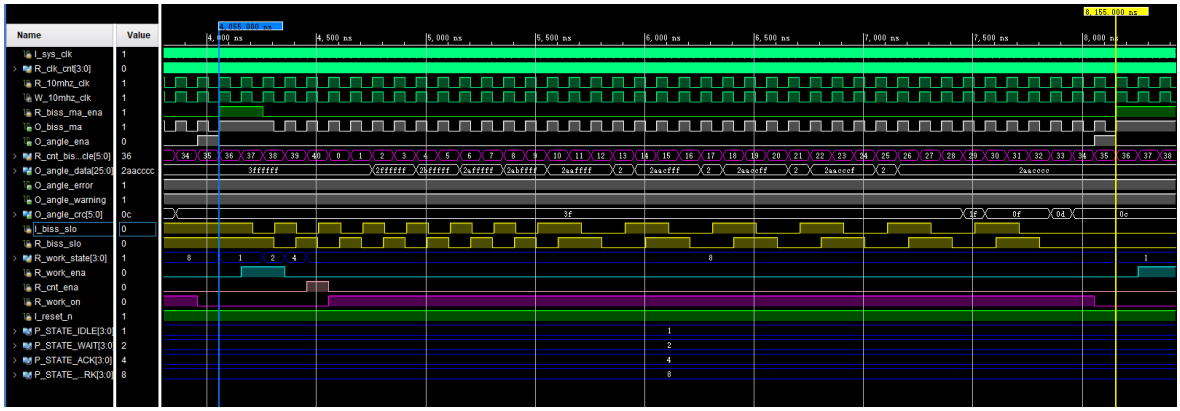

图 3.2.3 BiSS-C 第二周期数据接收时序局部仿真图

仿真结果依旧符合 BiSS-C 的协议数据时序图。

4 结论

本次仿真结果与协议给出的数据时序图一致，因此本次设计的 Verilog 程序有效且正确。

BiSS-C 协议的周期从 SLO 的准备信号开始，经过等待，应答，数据传输及超时阶段完成一个整周期。其中等待及应答阶段都需要时序逻辑判断，来完成状态保留。数据传输阶段要根据协议规定好接手位数来确保数据接收阶段的有效性。

致　　谢

值此论文完稿之际，感激之情油然而生。

感谢老师带给大家精彩的课堂，并传授打击十分实用的设计思想和常用的设计技巧。同时感谢教研室师兄师姐的帮助以及提点，在我做项目的时候帮助我上手软件，以及解决新手遇到的各种困难。最后感谢父母为我的投资，能让我在研究生期间学习到有用的知识。

参考文献

[1] BiSS_C_protocol_C6en.pdf (http://www.biss-interface.com/)

[2] biss_encoder_9709-9007-03-b(zh).pdf

(http://www.renishaw.com.cn/zh/biss-c-protocol-support--39687)

[3] iC-Haus China http://www.ichauschina.com/biss.htm

[4] 维基百科 https://en.wikipedia.org/wiki/BiSS_interface

附　　录

1. 代码：

```
`timescale 1ns/100ps
module    biss_mode(
      //====system reset====
      //I_reset_n            ,
      //====system clock====
      //I_sys_clk            ,     //100MHz working clock
      //====biss interface====
      O_biss_ma            ,
      //I_biss_slo           ,
      //====angle signal output====
      O_angle_ena          ,
      O_angle_error      ,
      O_angle_warning ,
      O_angle_crc          ,
      O_angle_data
      );
//-----------------------------------External Signal Definitions---------------------------------------
//====input====
//input                I_reset_n                ;
//input                I_sys_clk                ;
//input                I_biss_slo               ;
//====output====
output               O_biss_ma                ;
output               O_angle_ena              ;
output               O_angle_error            ;
output               O_angle_warning          ;
output[5:0]          O_angle_crc              ;
output[25:0]         O_angle_data             ;
//-----------------------------------------------Reg Definitions-------------------------------------------------
//====parameter====
parameter            P_STATE_IDLE       =     4'b0001;
parameter            P_STATE_WAIT       =     4'b0010;
parameter            P_STATE_ACK        =     4'b0100;
parameter            P_STATE_WORK       =     4'b1000;
//====output register define====
wire                 O_biss_ma                ;
reg                  O_angle_warning          ;
reg                  O_angle_error            ;
reg                  O_angle_ena              ;
```

```
reg[25:0]           O_angle_data            ;     // 26 bit data
reg[5:0]            O_angle_crc             ;     // 6    bit crc data
//====internal register define====
reg                   R_10mhz_clk               ;
reg[3:0]            R_clk_cnt               ;
wire                  W_10mhz_clk               ;
reg[5:0]            R_cnt_biss_cycle      ;
reg[8:0]            R_cnt_100us             ;
reg[3:0]            R_work_state            ;
reg                   R_work_ena              ;
reg                   R_cnt_ena               ;
reg                   R_work_on               ;
reg                   R_biss_ma_ena           ;
reg                   R_biss_slo                    ;
//=====test====
reg                 I_reset_n               ;
reg                 I_sys_clk               ;
reg                 I_biss_slo              ;
initial
begin
    I_reset_n = 0;
    I_sys_clk = 0;
    I_biss_slo = 1'b1;
    #10         I_reset_n = 1'b1;
    #100      I_biss_slo = 1'b0;
    #100        I_biss_slo = 1'b1;
    #1000    I_biss_slo = 1'b0;
    #200    I_biss_slo = 1'b1;
    #200        I_biss_slo = 1'b0;
    #200        I_biss_slo = 1'b1;
end
always #5 I_sys_clk <= ~I_sys_clk;
//-----------------------------------------------Main Body of Code-----------------------------------------------
//-------------------------clock---------------------------
// 10 conter 100MHz --> 10MHz
always @(posedge I_sys_clk or negedge I_reset_n)
begin
    if (~I_reset_n)
    begin
        R_clk_cnt      <= 4'd0;
    end
    else
    begin
        if(R_clk_cnt == 4'd4)
```

```
            begin
                R_clk_cnt      <= 4'd0;
            end
            else
            begin
                R_clk_cnt <= R_clk_cnt + 4'd1;
            end
        end
end
// Generate 5MHz Clock R_10mhz_clk
always @(posedge I_sys_clk or negedge I_reset_n)
begin
        if (~I_reset_n)
        begin
            R_10mhz_clk    <= 1'b0;
        end
        else
        begin
            if(R_clk_cnt == 4'd4)
            begin
                R_10mhz_clk    <= ~R_10mhz_clk;
            end
        end
end
// Buffer change R_10mhz_clk to    W_10mhz_clk
BUFG BUFG_u1 (
        .O(W_10mhz_clk), // 1-bit output: Clock output
        .I(R_10mhz_clk)    // 1-bit input: Clock input
        );
//------------------------clock---------------------------
//-----------------------------------10MHz Clock domain--------------------------------
always @ (negedge W_10mhz_clk or negedge I_reset_n)
begin
        if(~I_reset_n)
        begin
            R_biss_slo <= 1'b0;
        end
        else
        begin
            R_biss_slo <= I_biss_slo; // Get the signal from I_biss_slo
        end
end
// R_work_ena enable
always @(posedge W_10mhz_clk or negedge I_reset_n)
```

```
begin
    if (~I_reset_n)
    begin
        R_work_ena    <= 1'b0;
    end
    else
    begin
        if (R_work_state == P_STATE_IDLE)
        begin
            R_work_ena    <= 1'b1;       // to activate the prosses
        end
        else if(~R_biss_ma_ena)
        begin
            R_work_ena    <= 1'b0;
        end
    end
end
// State machine
always @(posedge W_10mhz_clk or negedge I_reset_n)
begin
    if (~I_reset_n)
    begin
        R_work_state      <=   P_STATE_IDLE;
        R_biss_ma_ena     <=   1'b1;                //low to work
        R_cnt_ena         <=   1'b0;
    end
    else
    begin
        case(R_work_state)
        P_STATE_IDLE:
        begin
            if(R_work_ena && R_biss_slo)
            begin
                R_work_state      <=   P_STATE_WAIT;
                R_biss_ma_ena     <=   1'b0;
            end
            else
            begin
                R_work_state      <=   P_STATE_IDLE;
                R_biss_ma_ena     <=   1'b1;
            end
        end
        P_STATE_WAIT:
        begin
```

```
            if(R_biss_slo)
            begin
                R_work_state      <=   P_STATE_WAIT;
            end
            else
            begin
                R_work_state      <=   P_STATE_ACK;
            end
        end
        P_STATE_ACK:
        begin
            if(R_biss_slo)
            begin
                R_work_state      <=   P_STATE_WORK;
                R_cnt_ena         <=   1'b1;
            end
            else
            begin
                R_work_state      <=   P_STATE_ACK;
                R_cnt_ena         <=   1'b0;
            end
        end
        P_STATE_WORK:
        begin
            if(R_work_on || R_cnt_ena)
            begin
                R_work_state      <=   P_STATE_WORK;
                R_cnt_ena         <=   1'b0;
            end
            else
            begin
                R_work_state      <=   P_STATE_IDLE;
                R_cnt_ena         <=   1'b0;
                R_biss_ma_ena     <=   1'b1;
            end
        end
        default:
        begin
            R_work_state      <=   P_STATE_IDLE;
            R_biss_ma_ena     <=   1'b1;
            R_cnt_ena         <=   1'b0;
        end
        endcase
    end
```

```
end
// R_cnt_biss_cycle    11111 6'h3f
always @(posedge W_10mhz_clk or negedge I_reset_n)
begin
    if (~I_reset_n)
    begin
        R_cnt_biss_cycle    <= 6'h3f;
    end
    else
    begin
        if(R_cnt_ena)
        begin
            R_cnt_biss_cycle        <= 6'h0;
        end
        else if(~(&R_cnt_biss_cycle))
        begin
            R_cnt_biss_cycle        <= R_cnt_biss_cycle + 6'b1;
        end
    end
end

always @(posedge W_10mhz_clk or negedge I_reset_n)
begin
    if (~I_reset_n)
    begin
        R_work_on      <= 1'b0;
    end
    else
    begin
        if(R_cnt_ena)
        begin
            R_work_on      <= 1'b1;
        end
        else if(R_cnt_biss_cycle == 6'd34)
        begin
            R_work_on      <=    1'b0;
        end
    end
end

always @(posedge W_10mhz_clk or negedge I_reset_n)
begin
    if (~I_reset_n)
    begin
```

```
        O_angle_data      <= 18'b0;
        O_angle_crc       <= 6'b0;
    end
    else
    begin
        case(R_cnt_biss_cycle)
        6'h00:   O_angle_data[25]      <= R_biss_slo;
        6'h01:   O_angle_data[24]      <= R_biss_slo;
        6'h02:   O_angle_data[23]      <= R_biss_slo;
        6'h03:   O_angle_data[22]      <= R_biss_slo;
        6'h04:   O_angle_data[21]      <= R_biss_slo;
        6'h05:   O_angle_data[20]      <= R_biss_slo;
        6'h06:   O_angle_data[19]      <= R_biss_slo;
        6'h07:   O_angle_data[18]      <= R_biss_slo;
        6'h08:   O_angle_data[17]      <= R_biss_slo;
        6'h09:   O_angle_data[16]      <= R_biss_slo;
        6'h0a:   O_angle_data[15]      <= R_biss_slo;
        6'h0b:   O_angle_data[14]      <= R_biss_slo;
        6'h0c:   O_angle_data[13]      <= R_biss_slo;
        6'h0d:   O_angle_data[12]      <= R_biss_slo;
        6'h0e:   O_angle_data[11]      <= R_biss_slo;
        6'h0f:   O_angle_data[10]      <= R_biss_slo;
        6'h10:   O_angle_data[9]       <= R_biss_slo;
        6'h11:   O_angle_data[8]       <= R_biss_slo;
        6'h12:   O_angle_data[7]       <= R_biss_slo;
        6'h13:   O_angle_data[6]       <= R_biss_slo;
        6'h14:   O_angle_data[5]       <= R_biss_slo;
        6'h15:   O_angle_data[4]       <= R_biss_slo;
        6'h16:   O_angle_data[3]       <= R_biss_slo;
        6'h17:   O_angle_data[2]       <= R_biss_slo;
        6'h18:   O_angle_data[1]       <= R_biss_slo;
        6'h19:   O_angle_data[0]       <= R_biss_slo;
        // CRC Data
        6'h1c:   O_angle_crc[5]        <= R_biss_slo;
        6'h1d:   O_angle_crc[4]        <= R_biss_slo;
        6'h1e:   O_angle_crc[3]        <= R_biss_slo;
        6'h1f:   O_angle_crc[2]        <= R_biss_slo;
        6'h20:   O_angle_crc[1]        <= R_biss_slo;
        6'h21:   O_angle_crc[0]        <= R_biss_slo;
        default:;
        endcase
    end
end
```

```
always @(posedge W_10mhz_clk or negedge I_reset_n)
begin
    if (~I_reset_n)
    begin
        O_angle_ena <= 1'b0;
    end
    else
    begin
        O_angle_ena <= (R_cnt_biss_cycle == 6'd34);
    end
end

always @(posedge W_10mhz_clk or negedge I_reset_n)
begin
    if (~I_reset_n)
    begin
        O_angle_error   <= 1'b0;
        O_angle_warning <= 1'b0;
    end
    else
    begin
        if(R_cnt_biss_cycle == 6'h1a)
        begin
            O_angle_error   <= R_biss_slo;
        end
        else if(R_cnt_biss_cycle == 6'h1b)
        begin
            O_angle_warning <= R_biss_slo;
        end
    end
end

assign  O_biss_ma = R_biss_ma_ena || R_10mhz_clk;

endmodule
//====END====
```