Gowin I3C SDR IP

用户指南

IPUG508-1.2,2018-09-26

版本信息

日期	版本	说明
2018/01/19	1.0	初始版本。
2018/09/26	1.1	更改 IP 名称。
2018/11/9	1.2	修改器件支持类型

目录

图目录

表目录

1 关于本手册

1.1 手册内容

Gowin I3C SDR IP 用户指南主要内容包括功能简介、信号定义、参数介绍、工作原理、GUI 调用等，旨在帮助用户快速了解高云半导体 Gowin I3C SDR IP 的特性及使用方法。

1.2 适用产品

本手册中描述的信息适用于以下产品：
GW1N 系列 FPGA 产品：

- GW1N 系列 FPGA 产品： GW1N-2，GW1N-2B，GW1N-4，GW1N-4B，GW1N-6ES，GW1N-6，GW1N-9ES，GW1N-9；
- GW1NR 系列 FPGA 产品：GW1NR-4，GW1NR-4B，GW1NR-9ES，GW1NR-9；
- GW1NS 系列 FPGA 产品：GW1NS-2，GW1NS-2C；
- GW2A 系列 FPGA 产品：GW2A-18，GW2A-55；
- GW2AR 系列 FPGA 产品：GW2AR-18。

1.3 相关文档

通过登录高云半导体网站 http://www.gowinsemi.com.cn/可以下载、查看以下相关文档：

- GW1N 系列 FPGA 产品数据手册
- GW1NR 系列 FPGA 产品数据手册
- GW1NS 系列 FPGA 产品数据手册
- GW2A 系列 FPGA 产品数据手册
- GW2AR 系列 FPGA 产品数据手册
- Gowin 云源软件用户指南

1.4 术语、缩略语

本手册中出现的相关术语、缩略语及相关释义如表 1-1 所示。

表 1-1 术语、缩略语

术语、缩略语	全称	含义
FPGA	Field Programmable Gate Array	现场可编程门阵列
I3C Bus	Inter-Integrated Circuit Bus	I3C 串行总线

1.5 技术支持与反馈

高云半导体提供全方位的技术支持，在使用过程中如有任何疑问或建议，可直接与公司联系：

网址：http://www.gowinsemi.com.cn/

E-mail：support@gowinsemi.com

Tel：+86 755 8262 0391

2 功能简介

2.1 概述

I3C 总线是一种由 MIPI 联盟发布的两线式串行总线，兼具了 I2C 和 SPI 的关键特性，具有低引线数、可扩展性、低功耗、更高的容量和新颖的性能，而且兼容 I2C，能有效的减少集成电路芯片系统的物理端口、支持低功耗、高数据速率和其他已有端口协议的优点，为支持现代移动手持设备、智能驾驶、IOT 设计添加了许多增强的特性。

Gowin I3C SDR IP 遵循 MIPI 联盟 I3C 总线协议，采用寄存器接口，集 I3C SDR Master 和 I3C SDR Slave 于一体，可实例化成 I3C SDR Master 或 I3C SDR Slave，实现 I3C SDR Master 与 I3C SDR Slave 或 I2C Slave 的通信。

2.2 特性

2.2.1 Gowin I3C SDR Master

1. 符合 MIPI I3C 协议；
2. 支持 I3C 地址仲裁检测；
3. 支持 Single Data Rate（SDR）通信模式；
4. 最高数据传输速率可达 12.5Mbps；
5. 产生起始、终止、重复起始和应答信息；
6. 支持起始、终止和重复起始检测；
7. 支持 SETDASA 或 ENTDAA 方式进行动态地址分配；
8. 支持接收/发送数据功能；
9. 支持线载中断（In-band Interrupts）；
10. 支持热接入（Hot-Join）；
11. 支持热接入时动态地址分配；
12. 支持 CCC's 命令；
13. 支持动态调整 SCL 频率；
14. 兼容 I2C Slave；
15. 采用寄存器接口。

2.2.2 Gowin I3C SDR Slave

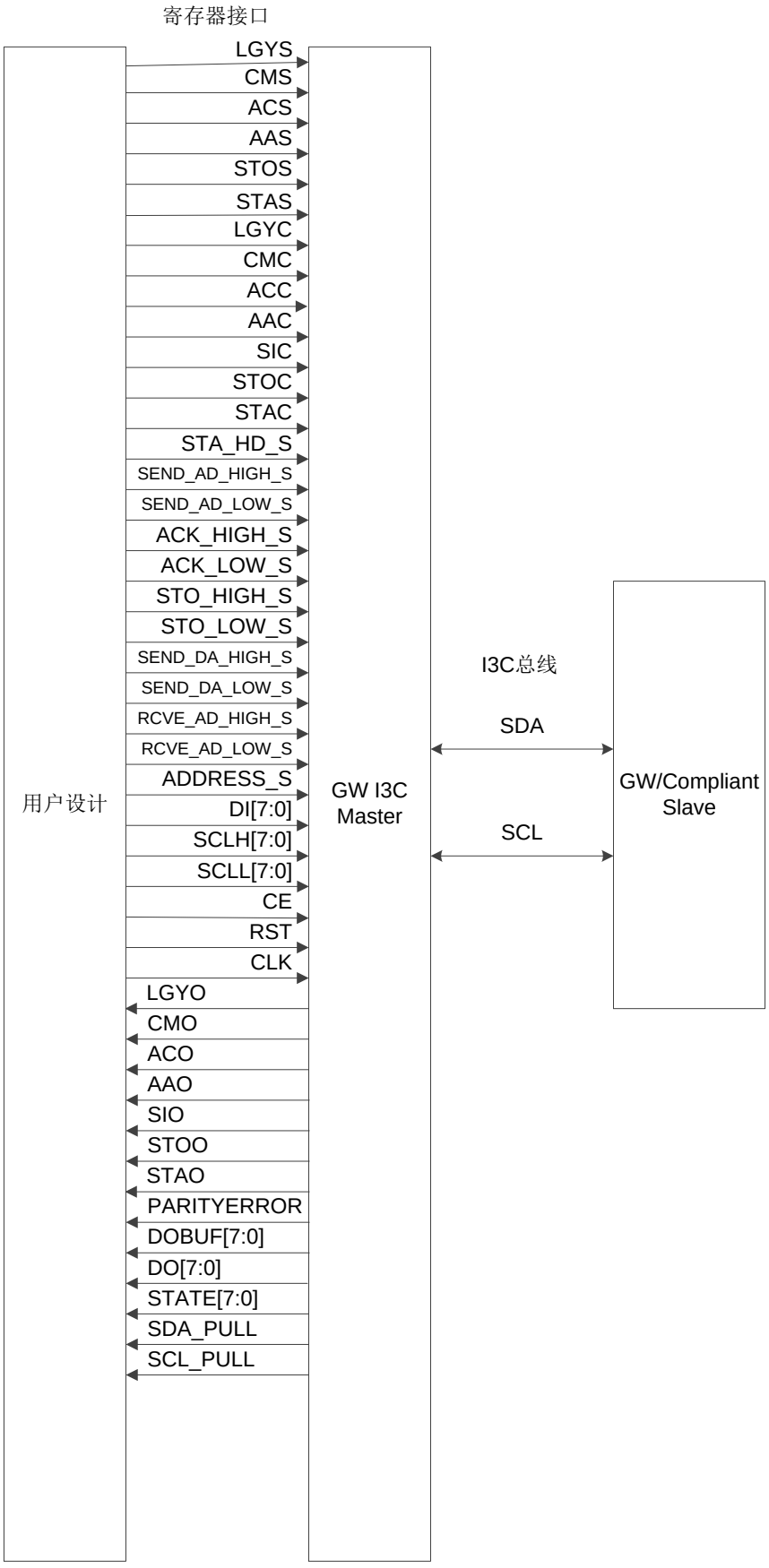
1. 符合 MIPI I3C 协议；
2. 产生起始和应答信息；
3. 支持起始、终止和重复起始检测；
4. 支持 SETDASA 或 ENTDAA 方式进行动态地址分配；
5. 接收/发送数据功能；
6. 发起 IBI 或 Hot-join 申请，若多个 Slave 发起 IBI 或 Hot-join 申请，地址最小的获得此次仲裁；
7. 可配置 Slave 静态地址；
8. 采用寄存器接口。

3 信号定义

3.1 Gowin I3C SDR Master 信号

Gowin I3C SDR IP 用作 I3C SDR Master 时，用户设计可以通过寄存器接口控制 I3C SDR Master，I3C SDR Master 通过 I3C 的数据 SDA 和时钟 SCL 这两条总线与外围设备 Slave 通信，其通信框图如下。

图 3-1 I3C SDR Master 通信框图



I3C SDR Master 的信号如表 3-1 所示。

表 3-1 I3C SDR Master 信号定义

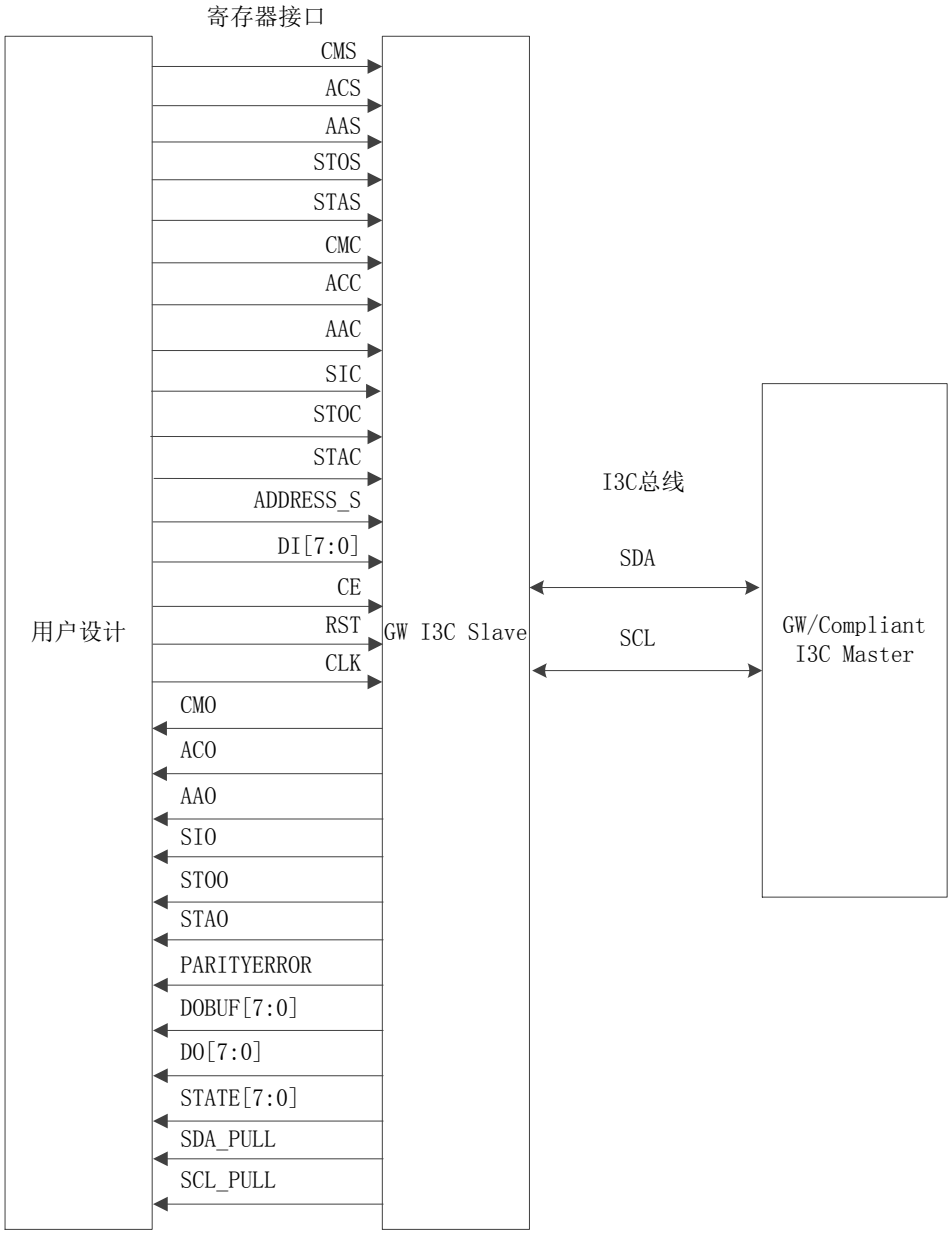
序号	信号名称	方向	位宽	描述
1	LGYO	Output	1	下达当前通讯对象为 I2C 命令的输出
2	CMO	Output	1	下达设备进入 Master 命令的输出
3	ACO	Output	1	在选择是否要继续时选择继续的输出
4	AAO	Output	1	在需要回复 ACK/NACK 时回复 ACK 的输出
5	SIO	Output	1	中断标识位的输出
6	STOO	Output	1	下达 STOP 命令的输出
7	STAO	Output	1	下达 START 命令的输出
8	PARITYERROR	Output	1	收数据时校验输出
9	DOBUF	Output	[7:0]	缓存后的数据输出
10	DO	Output	[7:0]	直接数据输出
11	STATE	Output	[7:0]	内部状态输出
12	SDA_PULL	Output	1	I3C 串行数据的可控上拉
13	SCL_PULL	Output	1	I3C 串行时钟的可控上拉
14	SDA	Inout	1	I3C 串行数据线
15	SCL	Inout	1	I3C 串行时钟线
16	LGYS	Input	1	当前通讯对象为 I2C 的置位信号
17	CMS	Input	1	设备进入 Master 的置位信号
18	ACS	Input	1	在选择是否要继续时选择继续的置位信号
19	AAS	Input	1	在需要回复 ACK/NACK 时回复 ACK 的置位信号
20	STOS	Input	1	下达 STOP 命令的置位信号
21	STAS	Input	1	下达 START 命令的置位信号
22	LGYC	Input	1	当前通讯对象为 I2C 的清零信号
23	CMC	Input	1	设备处于 Master 的清零信号
24	ACC	Input	1	在选择是否要继续时选择继续的清零信号
25	AAC	Input	1	在需要回复 ACK/NACK 时回复 ACK 的清零信号
26	SIC	Input	1	中断标识清零信号
27	STOC	Input	1	STOP 状态的清零信号
28	STAC	Input	1	START 状态的清零信号
29	STA_HD_S	Input	1	调整产生 START 时的保持时间的置位信号
30	SEND_AD_HIGH_S	Input	1	调整发送地址时 SCL 高电平时间的置位信号
31	SEND_AD_LOW_S	Input	1	调整发送地址时 SCL 低电平时间的置位信号
32	ACK_HIGH_S	Input	1	调整 ACK 时 SCL 高电平时间的置位信号
33	ACK_LOW_S	Input	1	调整 ACK 时 SCL 低电平时间的置位信号
34	STO_HIGH_S	Input	1	调整 STOP 时 SCL 高电平时间的置位信号
35	STO_LOW_S	Input	1	调整 STOP 时 SCL 低电平时间的置位信号
36	SEND_DA_HIGH_S	Input	1	调整发送数据时 SCL 高电平时间的置位信号
37	SEND_DA_LOW_S	Input	1	调整发送数据时 SCL 低电平时间的置位信号

序号	信号名称	方向	位宽	描述
38	RCVE_DA_HIGH_S	Input	1	调整接收数据时 SCL 高电平时间的置位信号
39	RCVE_DA_LOW_S	Input	1	调整接收数据时 SCL 低电平时间的置位信号
40	ADDRESS_S	Input	1	Slave 地址设置接口
41	DI	Input	[7:0]	数据输入
42	SCLH	Input	[7:0]	用于设置 SCL 高电平周期数
43	SCLL	Input	[7:0]	用于设置 SCL 低电平周期数
44	CE	Input	1	时钟使能信号
45	RST	Input	1	复位信号
46	CLK	Input	1	时钟信号

3.2 Gowin I3C SDR Slave 信号

Gowin I3C SDR IP 用作 I3C SDR Slave 时，用户设计可以通过寄存器接口控制 I3C SDR Slave，I3C SDR Slave 通过 I3C 的数据 SDA 和时钟 SCL 这两条总线与 I3C SDR Master 通信，其通信框图如图 3-2 所示。

图 3-2 I3C SDR Slave 通信框图



I3C SDR Slave 的信号如表 3-2 所示。

表 3-2 I3C SDR Slave 信号定义

序号	信号名称	方向	位宽	描述
1	CMO	Output	1	下达设备进入 Master 命令的输出
2	ACO	Output	1	在选择是否要继续时选择继续的输出
3	AAO	Output	1	在需要回复 ACK/NACK 时回复 ACK 的输出
4	SIO	Output	1	中断标识位的输出
5	STOO	Output	1	下达 STOP 命令的输出
6	STAO	Output	1	下达 START 命令的输出
7	PARITYERROR	Output	1	收数据时校验输出

序号	信号名称	方向	位宽	描述
8	DOBUF	Output	[7:0]	缓存后的数据输出
9	DO	Output	[7:0]	直接数据输出
10	STATE	Output	[7:0]	内部状态输出
11	SDA_PULL	Output	1	I3C 串行数据的可控上拉
12	SCL_PULL	Output	1	I3C 串行时钟的可控上拉
13	SDA	Inout	1	I3C 串行数据线
14	SCL	Inout	1	I3C 串行时钟线
15	CMS	Input	1	设备进入 Master 的置位信号
16	ACS	Input	1	在选择是否要继续时选择继续的置位信号
17	AAS	Input	1	在需要回复 ACK/NACK 时回复 ACK 的置位信号
18	STOS	Input	1	下达 STOP 命令的置位信号
19	STAS	Input	1	下达 START 命令的置位信号
20	CMC	Input	1	设备处于 Master 的清零信号
21	ACC	Input	1	在选择是否要继续时选择继续的清零信号
22	AAC	Input	1	在需要回复 ACK/NACK 时回复 ACK 的清零信号
23	SIC	Input	1	中断标识清零信号
24	STOC	Input	1	STOP 状态的清零信号
25	STAC	Input	1	START 状态的清零信号
26	ADDRESS_S	Input	1	Slave 地址设置接口
27	DI	Input	[7:0]	数据输入
28	CE	Input	1	时钟使能信号
29	RST	Input	1	复位信号
30	CLK	Input	1	时钟信号

4GUI 参数

4.1 概述

I3C IP GUI 界面参数主要用来设置动态调整 I3C SDR Master 的 SCL 的高低电平的时间值和 I3C SDR Slave 的静态地址。

4.2 I3C SDR GUI 参数

表 4-1 I3C SDR Master GUI 参数

序号	参数名称	范围	默认值	描述
1	STA HD	20~254	20	START 时 hold time
2	SCL LOW	3~254	3	SCL 低电平时间
3	SCL HIGH	2~254	2	SCL 高电平时间
4	SEND AD SCL HIGH	2~254	2	发送地址时 SCL 高电平时间
5	SEND AD SCL LOW	3~254	3	发送地址时 SCL 低电平时间
6	ACK SCL HIGH	2~254	2	ACK 时 SCL 高电平时间
7	ACK SCL LOW	3~254	3	ACK 时 SCL 低电平时间
8	STO SCL HIGH	2~254	2	STOP 时 SCL 高电平时间
9	STO SCL LOW	3~254	3	STOP 时 SCL 低电平时间
10	SEND DA SCL HIGH	2~254	2	发送数据时 SCL 高电平时间
11	SEND DA SCL LOW	3~254	3	发送数据时 SCL 低电平时间
12	RCVE DA SCL HIGH	2~254	2	接收数据时 SCL 高电平时间
13	RCVE DA SCL LOW	3~254	3	接收数据时 SCL 低电平时间

表 4-2 I3C SDR Slave GUI 参数

序号	参数名称	范围	默认值	描述
1	SLAVE STATIC ADDRESS	7’h00~7’h7F	7’h08	Slave 的静态地址

5 工作原理

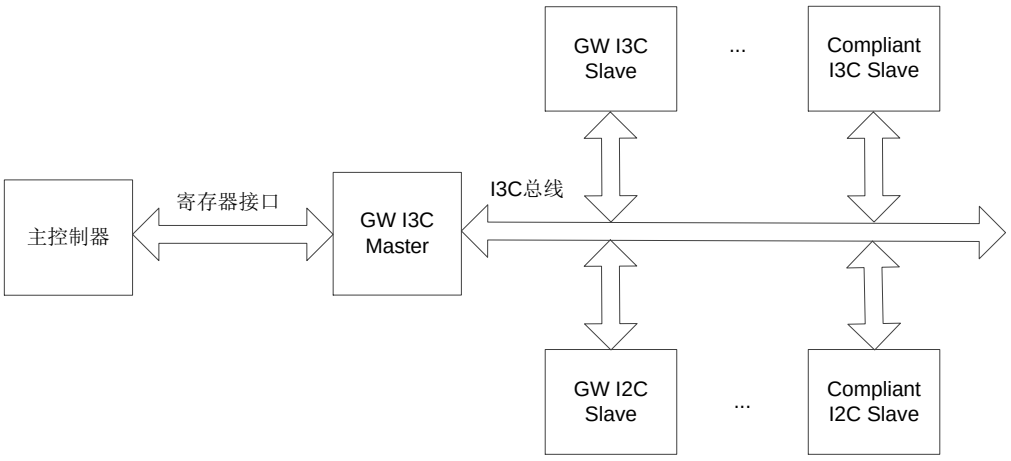
Gowin I3C SDR IP 集 I3C SDR Master 和 I3C SDR Slave 于一体，实现 I3C SDR Master 和 I3C SDR Slave 的动态地址分配、发送/接收数据、中断申请等功能。

若 I3C SDR IP 用作 I3C SDR Master，此时，总线上可以挂载多个 Slave，包括 MIPI 联盟支持的 I3C SDR/I2C Slave；可以发起 START、STOP、向 Slave 发送地址；向 I3C SDR Slave 发送数据，接收 I3C SDR Slave 发送的数据，响应 I3C SDR Slave 申请的 IBI 或 hot-join，进行 I3C SDR Slave 的动态地址分配；向 I2C Slave 发送数据，接收 I2C Slave 发送的数据。若 I3C SDR IP 用作 I3C SDR Slave，则可以发起 START、发送地址、向 I3C SDR Master 发送数据，接收 I3C SDR Master 发送的数据，响应地址，接受动态地址分配过程分配的动态地址。

5.1 系统框图

如图 5-1 所示，主控制器将指令或数据通过寄存器接口传给 Gowin I3C SDR Master，然后 Gowin I3C SDR Master 通过 I3C 总线下发给 Gowin I3C SDR/I2C Slave 或 Compliant I3C SDR/I2C Slave，或将 Gowin I3C SDR/I2C Slave 或 Compliant I3C SDR/I2C Slave 的数据通过 Gowin I3C SDR Master 上传给主控制器。

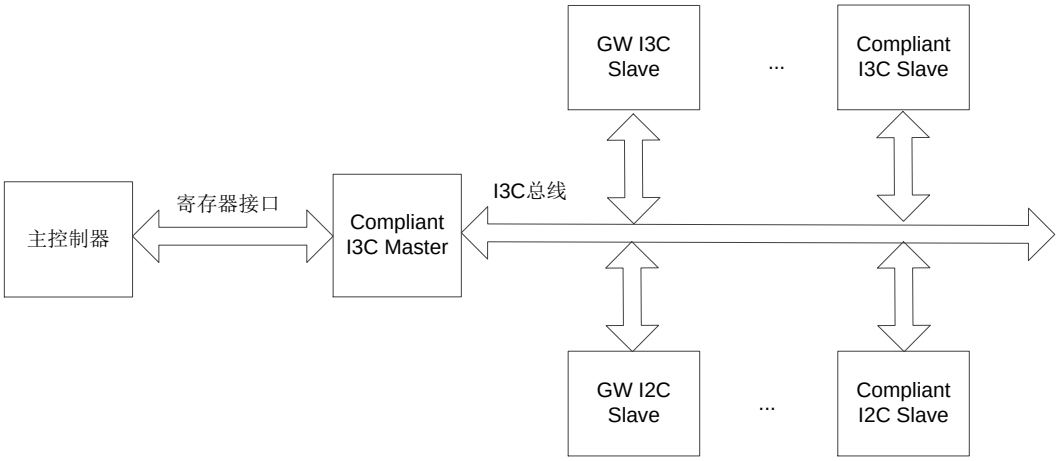
图 5-1 系统框图



如图 5-2 所示。主控制器将指令或数据通过寄存器接口传给 Compliant

I3C SDR Master，然后 Compliant I3C SDR Master 通过 I3C 总线下发给 Gowin I3C SDR/I2C Slave 或 Compliant I3C SDR/I2C Slave，或将 Gowin I3C SDR/I2C Slave 或 Compliant I3C SDR/I2C Slave 的数据通过 Compliant I3C SDR Master 上传给主控制器。

图 5-2 系统框图



5.2 I3C SDR IP 的状态

Gowin I3C SDR IP 用作 I3C SDR Master 和 I3C SDR Slave 时的状态如下表。

表 5-1 I3C SDR Master 的状态

序号	状态名称	描述
1	S_IDLE	I3C 总线处于空闲状态
2	S_CM_STA	主机发送 START 状态
3	S_CM_WAIT_AD	主机等待用户提供地址
4	S_CM_SEND_AD	主机向从机发送地址
5	S_CM_ADRS_ABTR	主机发送地址，总线仲裁检测
6	S_CM_ADRS_ABTR_OK	总线仲裁地址结束
7	S_CM_WAIT_ACK_HF	主机等待从机 ACK，I3C 总线有模式切换
8	S_CM_WAIT_ACK	主机等待从机 ACK，I3C 总线没有模式切换
9	S_CM_WAIT_NACK	主机接收 NACK
10	S_CM_WAIT_W_DA	主机等待用户提供写数据
11	S_CM_W_STA	主机发送写地址收到 ACK 后发起 START
12	S_CM_W_STO	主机发送写地址后发起 STOP
13	S_CM_W_ACK_STO	主机发送写地址收到 ACK 后发起 STOP
14	S_CM_W_SEND_DA	主机发送写数据
15	S_CM_SEND_DA_OK	主机发送完写数据
16	S_CM_READ_SL	主机读从机
17	S_CM_READ_SL_OK	主机读从机等待 Tbit
18	S_CM_READ_OK_SI	主机读从机后收到 SIO
19	S_CM_SL_END	主机读从机收到结束的 Tbit

序号	状态名称	描述
20	S_CI2CM_W_SEND_DA	主机向 I2C 从机写数据
21	S_CI2CM_SEND_DA_OK	主机向 I2C 从机写完数据
22	S_CI2CM_WAIT_ACK_HF	主机等待 I2C 从机 ACK，I3C 总线有模式切换
23	S_CI2CM_WAIT_NACK	主机收到 I2C 从机的 NACK
24	S_CI2CM_READ_SL	主机读 I2C 从机
25	S_CI2CM_SEND_ACK	主机发送 ACK 给 I2C 从机
26	S_CI2CM_SEND_ACK_OK	主机发送完 ACK 给 I2C 从机
26	S_CM_LINE_STA	主机响应从机发起的 START
28	S_CM_DAA_READ	主机 ENTDAA 读 8 字节
29	S_CM_DAA_WRITE	主机 ENTDAA 写一个字节
30	S_CM_DAA_SL_ACKDA	主机 ENTDAA，从机 ACK 动态地址
31	S_CM_DAA_SL_NACKDA	主机 ENTDAA，从机 NACK 动态地址

表 5-2 I3C SDR Slave 的状态

序号	状态名称	描述
1	S_SL_LINE_STA	从机收到总线的 START
2	S_SL_STA	从机发起 START
3	S_SL_RCVE_AD	从机接收地址
4	S_SL_SEND_W_ACK	从机 ACK 写地址
5	S_SL_W_DA	从机接收写数据
6	S_SL_W_DA_NINE	从机接收写数据第 9bit
7	S_SL_R_DA	从机发送读数据
8	S_SL_SEND_R_ACK	从机 ACK 读地址
9	S_SL_R_DA_NINE	从机发送读数据第 9bit
10	S_SL_SEND_AD	从机发送地址
11	S_SL_RCVE_ACK	从机接收 ACK 位
12	S_SL_CCC_W_DA	从机接收 CCC 数据
13	S_SL_DAA_READ	从机发送 ENTDAA 的 8 字节
14	S_SL_DAA_WRITE	从机接收 ENTDAA 的 8 字节
15	S_SL_DAA_ACK	从机 ACK 动态地址

5.3 I3C SDR 主机挂载 I3C SDR 从机基本操作流程

Gowin I3C SDR IP 支持 I3C 的操作，下面对 I3C 的动态地址分配、读写操作、IBI、Hot-join 中断操作进行介绍。

5.3.1 I3C SDR IP 进入主机状态

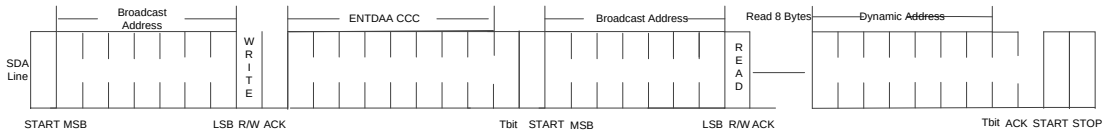
在 I3C 总线空闲状态下，通过置位 CMS，可使设备进入 Current Master 状态。

5.3.2 I3C SDR 主机给 I3C 从机分配动态地址

设备设置成主机后，可以与从机进行通信，首先需要进行动态地址分配，动态地址分配流程分为 SETDASA 和 ENTDAA 模式。

SETDASA 模式

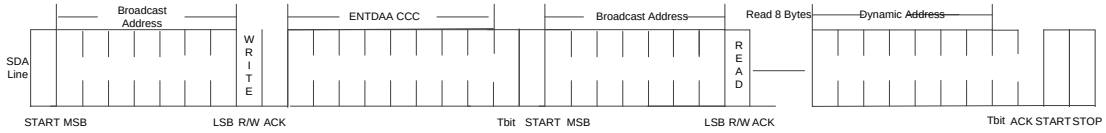
图 5-3 SETDASA 模式



1. 置位主机的 STAS 信号，主机发起 START；
2. 设置主机输入 DI 的值：广播地址（7’h7E）和 Write 操作（1’b0），等待从机 ACK；
3. 置位从机 AAS 信号，从机收到自身地址时可以自动发送 ACK；
4. 设置主机输入 DI 的值，发送 SETDASA CCC（8’h87）；
5. 置位主机的 STAS 信号，主机发起 START；
6. 设置主机输入 DI 的值：从机静态地址（7bit）和 Write 操作（1’b0），等待从机 ACK；
7. 置位从机 AAS 信号，从机收到自身地址时可以自动发送 ACK；
8. 设置主机输入 DI 的值：从机动态地址（7bit）和 1’b0；
9. 置位主机的 STOP 或 START 信号，主机发起 STOP 或 START 退出 SETDASA 模式。

ENTDAA 模式

图 5-4 ENTDAA 模式



1. 置位主机的 STAS 信号，主机发起 START；
2. 设置主机输入 DI 的值：广播地址（7’h7E）和 Write 操作（1’b0），等待从机 ACK；
3. 置位从机 AAS 信号，从机收到自身地址时可以自动发送 ACK；
4. 设置主机输入 DI 的值，发送 ENTDAA CCC（8’h07）；
5. 置位主机的 STAS 信号，主机发起 START；
6. 设置主机输入 DI 的值：广播地址（7’h7E）和 Read 操作（1’b1），等待从机 ACK；
7. 置位从机 AAS 信号，从机收到自身地址时可以自动发送 ACK；
8. 主机连续读取 8 个字节，每读完一个字节之后用户软件复位 SIO 信号继续读取，在复位第 8 个 SIO 后主机向从机发送动态地址（7bit）和 1’b0，等待从机 ACK；
9. 置位从机 AAS 信号，从机收到自身地址时可以自动发送 ACK；
10. 置位主机的 STOS，主机发起 STOP 退出 ENTDAA 模式。

5.3.3 I3C SDR 主机向 I3C 从机写数据

图 5-5 向 I3C 从机写一个数据

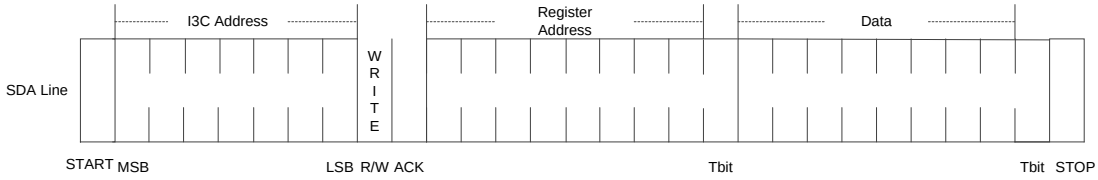
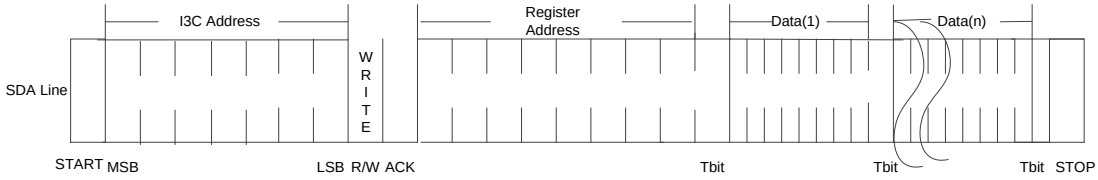


图 5-6 向 I3C 从机写多个数据



1. 在 I3C 总线空闲状态下，置位主机的 STAS 信号，主机发起 START；
2. 设置主机输入 DI 的值：I3C 从机地址（7bit）和 W 操作（1’b0），主机发送从机地址和写，等待从机 ACK；
3. 置位从机 AAS 信号，从机收到自身地址时可以自动发送 ACK；
4. 设置主机输入 DI 的值：寄存器地址，主机发送写操作的寄存器地址；
5. 设置主机输入 DI 的值：Data（8bit），即要向寄存器写入数据；
6. 可重复 5 步骤多次，即顺序写多个寄存器；
7. 置位主机的 STOS 信号，主机发起 STOP。

5.3.4 I3C 主机从 I3C SDR 从机读数据

图 5-7 从 I3C SDR 从机读一个数据

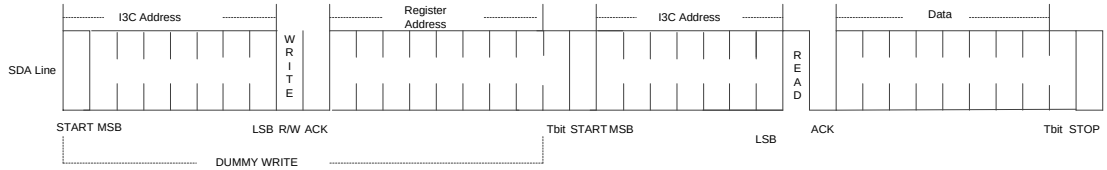
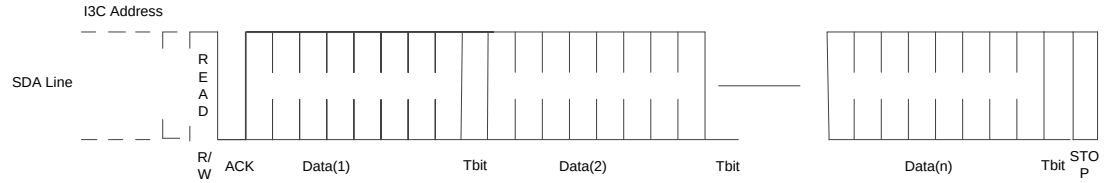


图 5-8 从 I3C SDR 从机读多个数据



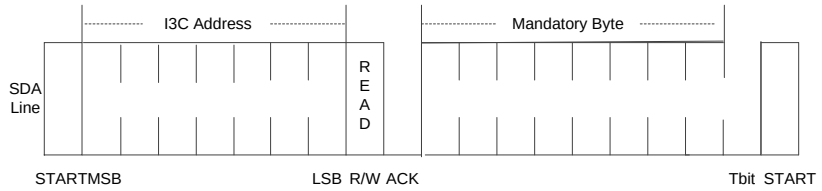
1. 置位主机的 STAS 信号，主机发起 START；
2. 设置主机输入 DI 的值：I3C SDR 从机地址（7bit）和 W 操作（1’b0），主机发送从机地址和写，等待从机 ACK；
3. 置位从机 AAS 信号，从机收到自身地址时可以自动发送 ACK；
4. 设置主机输入 DI 的值：寄存器地址，主机发送读操作的寄存器地址；
5. 置位主机的 STAS 信号，主机发起 START；
6. 设置主机输入 DI 的值：I3C SDR 从机地址（7bit）和 R 操作（1’b1），主机发送从机地址和读，等待从机 ACK；

7. 置位从机 AAS 信号，从机收到自身地址时可以自动发送 ACK；
8. 设置从机输入 DI 的值：Data（8bit），通过从机 ACS 决定从机是否继续读，主机 ACS 决定主机是否继续读；
9. 可重复 8 步骤多次，即顺序读多个寄存器。

5.3.5 I3C SDR 从机申请 IBI

I3C 协议支持带内中断，IDLE 状态下，允许从机发起 START，然后发送从机地址；或者当主机发起 START 时，从机检测到 START，主机和从机同时发送地址，地址小的获得此次仲裁。

图 5-9 从机申请 IBI

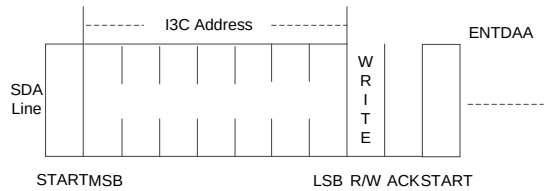


1. 置位从机的 STAS 信号，从机发起 START；
2. 设置从机的 DI 值，发送从机地址（7bit）和 Read 操作（1’b1），等待主机 ACK；
3. 置位主机 AAS，主机发送 ACK；
4. 主机获得总线控制权进行读操作，若 BCR[2]位为 1，Master 在读操作时强制读取一个字节的数据。
5. 接下来的操作同主机发起的操作一致。

5.3.6 I3C SDR 从机申请 Hot-join

I3C SDR 协议支持 Hot-join，当 I3C 总线配置好之后允许从机加入到 I3C 总线中。

图 5-10 从机申请 Hot-join



1. 置位从机的 STAS，从机发起 START；
2. 设置从机的 DI 值，发送预留的从机地址（7’b0000_010）和 Write 操作（1’b0），等待主机 ACK；
3. 置位主机 AAS，主机发送 ACK；
4. 主机进入 ENTDAA 模式为从机分配动态地址；
5. 接下来的操作同主机发起的操作一致。

5.4 I3C SDR 主机挂载 I2C 从机基本操作流程

Gowin I3C SDR IP 兼容 I2C 的操作，下面对 I2C 的读写操作进行介绍。

5.4.1 I3C SDR IP 进入主机状态

在 I3C 总线空闲状态下，通过置位 CMS，可使设备进入 Current Master

状态。置位 LGYS，主机与 I2C 从机进行通信。

5.4.2 I3C 主机向 I2C 从机写数据

图 5-11 向 I2C 从机写一个数据

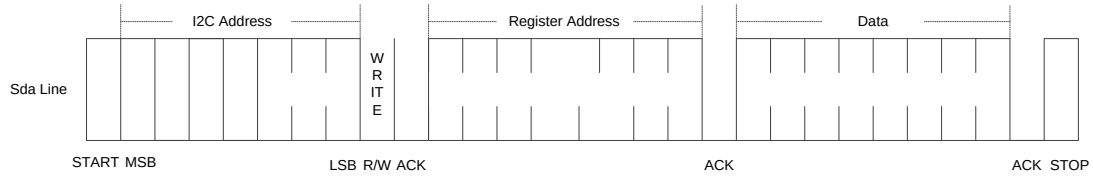
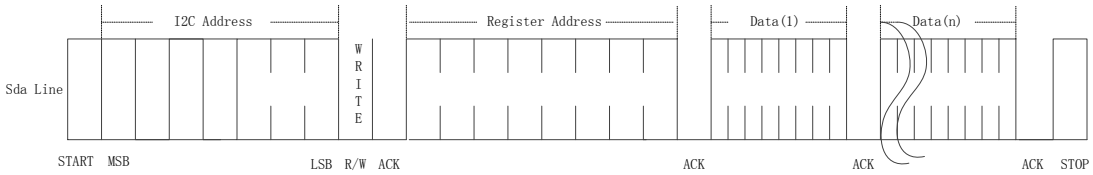


图 5-12 向 I2C 从机写多个数据



1. 置位主机的 STAS 信号，主机发起 START；
2. 设置主机的 DI 值，发送 I2C 从机地址（7bit）和 Write 操作(1’b0)，等待从机 ACK；
3. 从机发送 ACK；
4. 设置主机的 DI 值，发送寄存器地址，等待从机 ACK；
5. 从机发送 ACK；
6. 设置主机的 DI 值，发送要写入寄存器中的数据（8bit），等待从机 ACK；
7. 从机发送 ACK；
8. 可重复步骤 6 和 7 多次，发送多个数据，即顺序写入多个寄存器；
9. 置位主机的 STOS 信号，主机发起 STOP。

5.4.3 I3C SDR 主机从 I2C 从机读数据

图 5-13 从 I2C 从机读一个数据

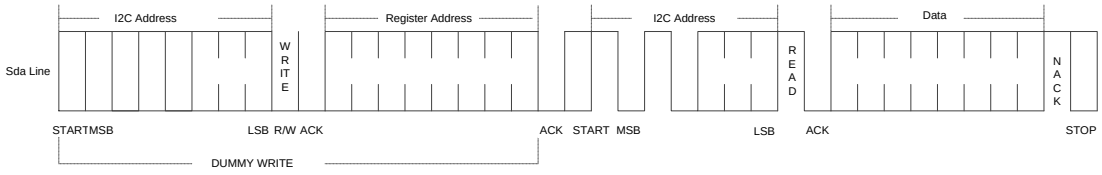
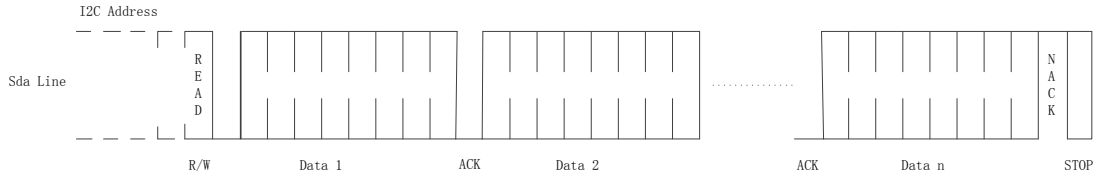


图 5-14 从 I2C 从机读多个数据



1. 置位主机的 STAS 信号，主机发起 START；
2. 设置主机的 DI 值，发送 I2C 地址（7bit）和 Write 操作(1’b0)，等待从机 ACK；

3. 从机发送 ACK；
4. 设置主机的 DI 值，发送寄存器地址，等待从机 ACK；
5. 从机发送 ACK；
6. 置位主机的 STAS，主机发起 START；
7. 设置主机的 DI 值，发送 I2C 地址（7bit）和 Read 操作(1’b1)，等待从机 ACK；
8. 从机发送 ACK；
9. 从机发送数据（8bit）；
10. 主机发送 ACK；
11. 可重复步骤 9 和 10 多次，顺序读取多个数据。

6 应用举例

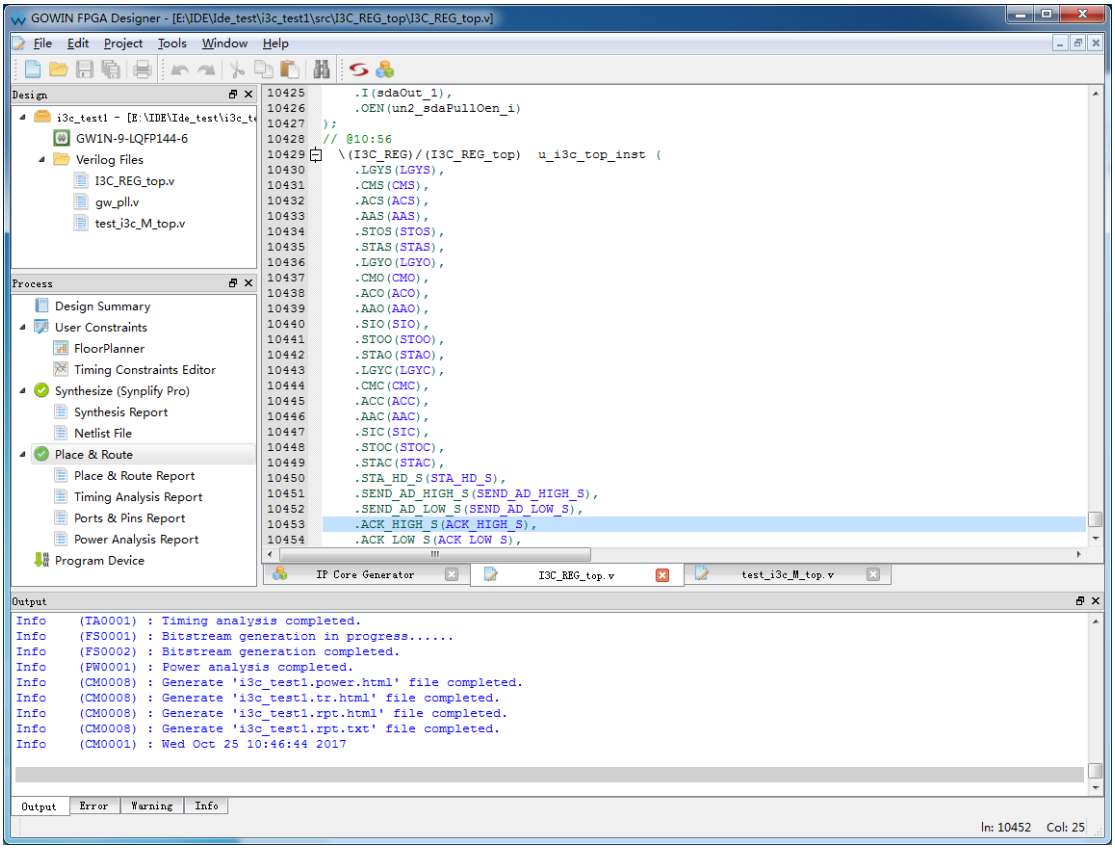
6.1 打开工程

启动 Gowin 云源软件后，单击“File> Open …”，打开“Open File”对话框，选择所需工程文件（*.gprj），打开工程，如图 6-1 所示。

注！

有三种方式打开工程，其它打开工程方式请参考 *Gowin 云源软件用户指南> 5 云源软件使用> 5.2 打开工程*。

图 6-1 打开工程



6.2 调用 Gowin I3C SDR IP

Gowin I3C SDR IP 集 Master 和 Slave 于一体，调用过程一致，只需修改图 6-3 中的 Module Name 和 File Name 来区分 Master 和 Slave，相关操作步骤如下：

在 Gowin 云源软件菜单栏中，单击“Tool > IP Core Generator”；弹出如图 6-2 所示界面；

选择所需器件，如 GW1N-9-LQFP144；在界面左侧“Name”窗口中双击“I3C”，弹出如图 6-3 所示界面；默认选项即可，单击“OK”，生成 I3C_REG_top Module。

注！

Gowin I3C SDR IP 与 I2C/SPI/UART Slave 的通信，与此类似，均可通过 Gowin 云源软件实现，此处仅以 I3C SDR 间的通信为例进行调用说明。

图 6-2 IP Core Generator 界面

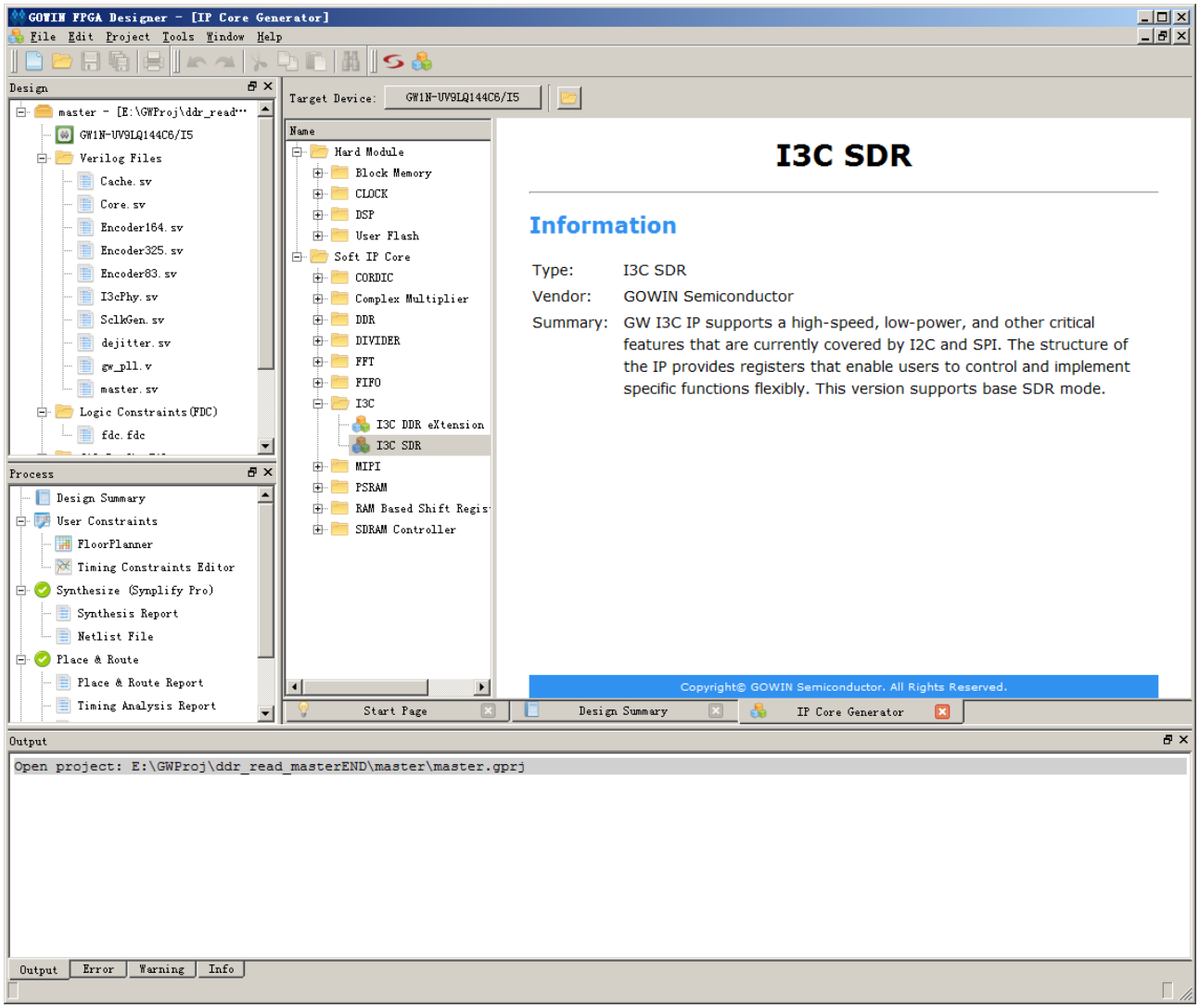
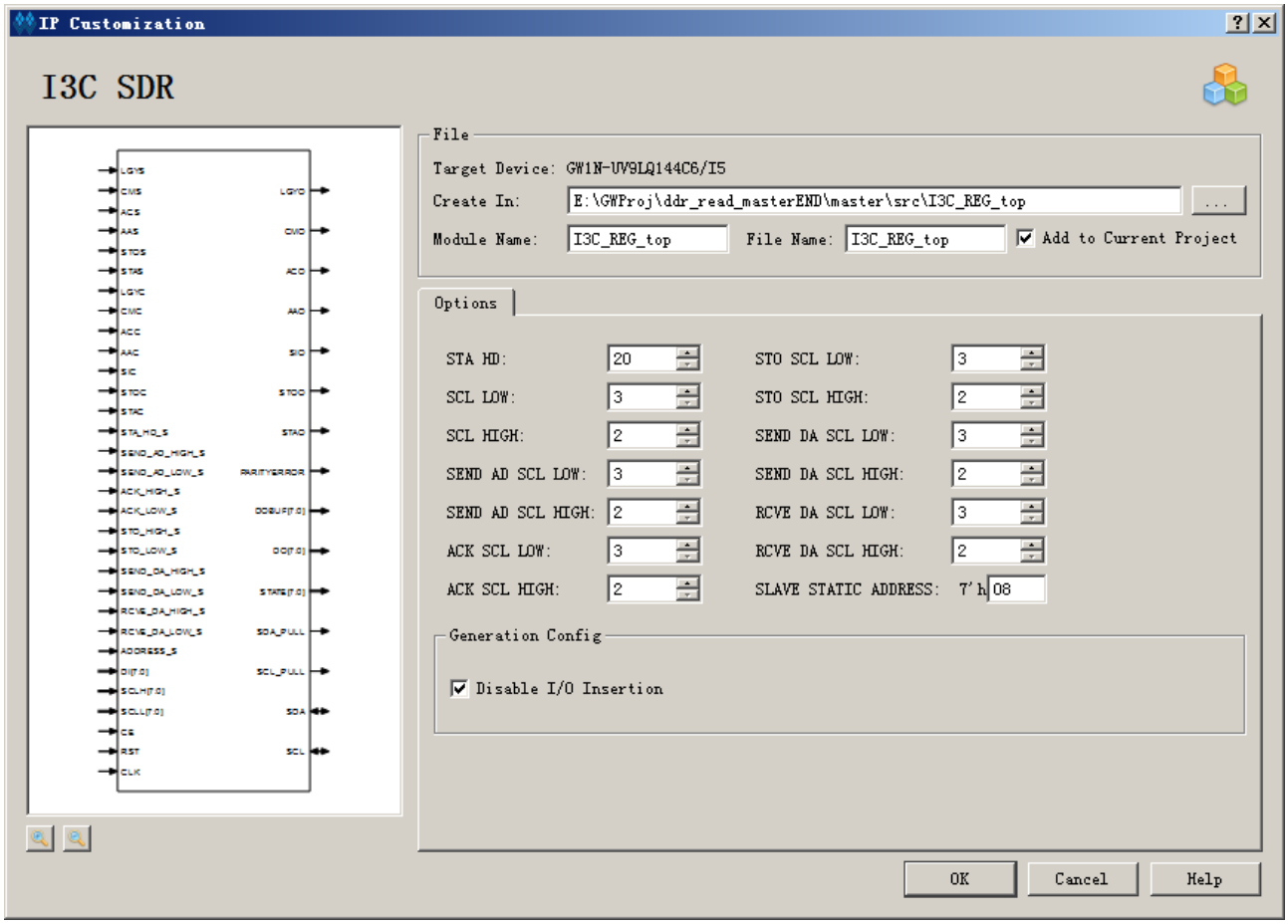


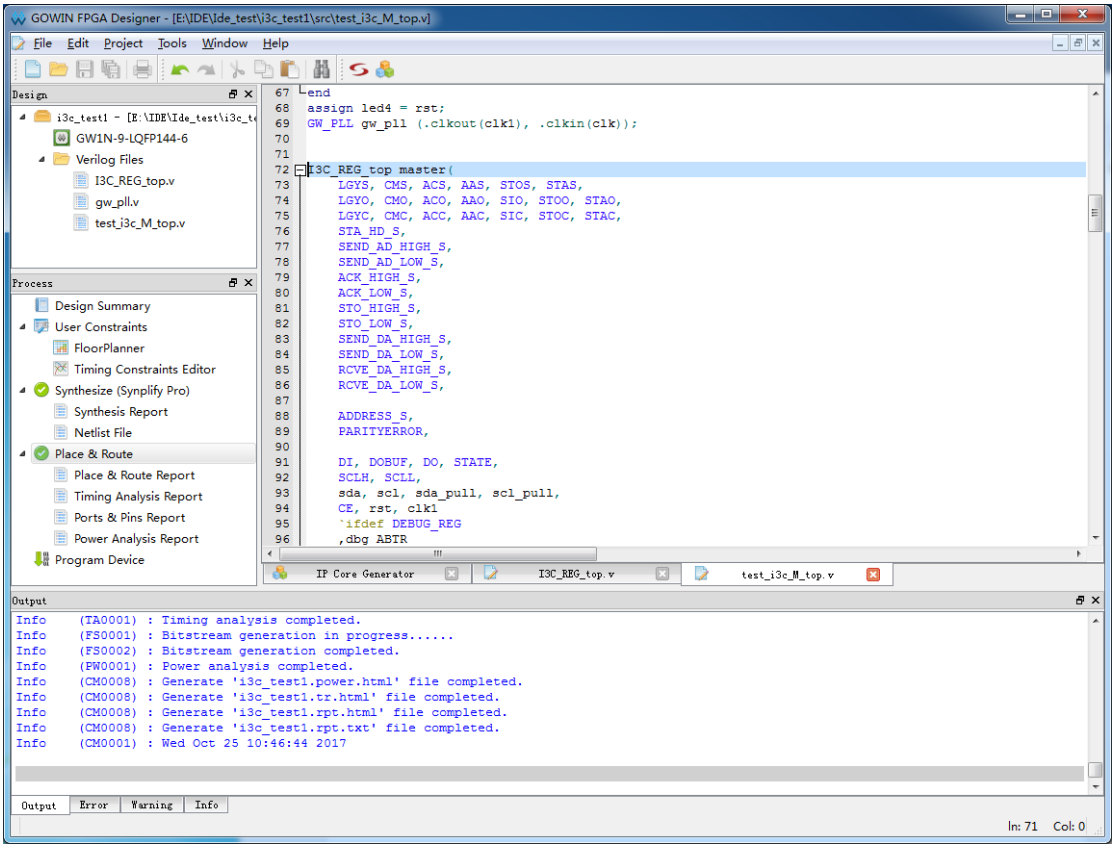
图 6-3 Gowin I3C IP 界面



6.3 例化 Gowin I3C SDR IP

在工程中例化 I3C_REG_top，如图 6-4 所示。

图 6-4 例化 Gowin I3C SDR IP



6.4 约束 Gowin I3C SDR IP

需要对 Gowin I3C SDR IP 的 SDA\SDA_PULL，SCL\SCL_PULL 进行约束，约束到 GW1N9K 的 IOB 上。

6.5 生成 bitstream 文件

进行必要的约束后，通过综合、布局布线、产生 bitstream 文件。通过 Gowin 下载线把 bitstream 文件下载到开发板或测试版，可通过测试接口观察主机和从机之间的通信情况。

7 时序

7.1 时序

I3C SDR IP 时序包括 I3C SDR Master 接口和 I3C SDR Slave 接口的读时序、写时序、动态地址分配时序、IBI 时序、Hot-join 时序。

图 7-1 写数据时序

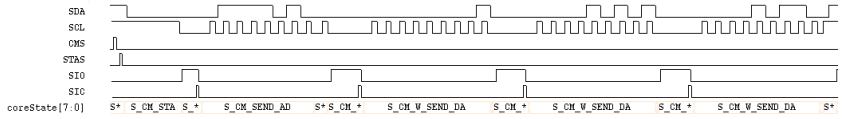


图 7-2 读数据时序

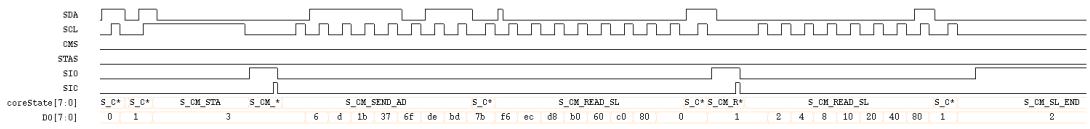


图 7-3 SETDASA 时序

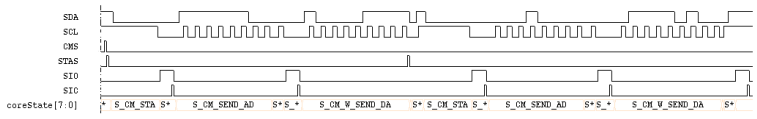


图 7-4 ENTDAA 时序

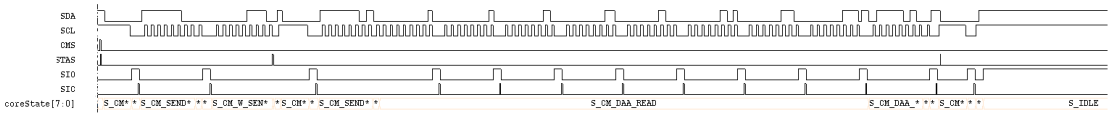


图 7-5 IBI 时序

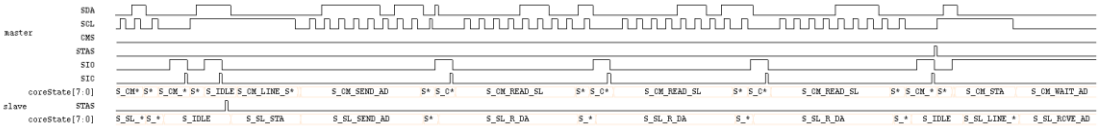
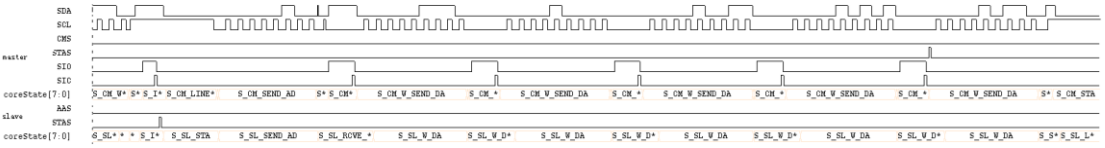


图 7-6 Hot-join 时序



8 参考设计

8.1 参考设计功能

本参考设计主要用于验证 I3C SDR 主机和 I3C SDR 从机的通信，主要功能包括：主机进行写操作（发送从机地址和写操作位 8'haa，发送寄存器地址 8'h55，然后给从机写从 8'h56 开始的 128 个数据）；

8.2 主机的顶层设计

主机的顶层设计 top.v 见附件 9.1，现对其进行简单说明：

1. top.v 定义了很多端口，其中有些端口虽定义但暂时未用，这样做的目的是，若想观察某些信号，可将其拉到这些预定端口上。
2. top.v 主要用于完成时钟分频、消抖、写数据。
   a). counter 子模块：时钟分频。首先 50MHz 的时钟频率经过 PLL 分频成 10MHz，然后经过 counter 子模块分频成 10KHz；
   b). deUstb 子模块：按键消抖；
   c). 写数据模块：I3C SDR 主机输出端口 STATE，会输出相应 I3C SDR 的状态。根据其状态，完成相应功能。如，STATE 为 S_CM_WAIT_AD，则 I3C SDR 主机处于等待地址状态，主控制器可发送地址；STATE 为 S_CM_WAIT_W_DA，则 I3C SDR 主机处于等待写入数据状态，主控制器可写数据。
3. 对 top.v 代码进行简单介绍。
   d). 时钟分频。经过 PLL 和 counter 模块处理后，时钟频率由 50MHz 分频为 10KHz。

```
GW_PLL myPll(
    .clkout(midClk), //output clkout 10M
    .clkin(clk_ext) //input clkin 50M
);
counter cnt1(lowOut,lowClk,lowOver,midClk,1'b1);
defparam cnt1.OVER = 1_000;
```

e). 实例化 I3C SDR Master

```
I3C_REG_top master(
    .LGYS(LGYS), .CMS(CMS), .ACS(ACS), .AAS(AAS), .STOS(STOS), .STAS(STAS),
```

```
    .LGYO(LGYO), .CMO(CMO), .ACO(ACO), .AAO(AAO), .SIO(SIO), .STOO(STOO), .STAO(STAO),
    .LGYC(LGYC), .CMC(CMC), .ACC(ACC), .AAC(AAC), .SIC(SIC), .STOC(STOC), .STAC(STAC),
    .STA_HD_S(STA_HD_S),
    .SEND_AD_HIGH_S(SEND_AD_HIGH_S),
    .SEND_AD_LOW_S(SEND_AD_LOW_S),
    .ACK_HIGH_S(ACK_HIGH_S),
    .ACK_LOW_S(ACK_LOW_S),
    .STO_HIGH_S(STO_HIGH_S),
    .STO_LOW_S(STO_LOW_S),
    .SEND_DA_HIGH_S(SEND_DA_HIGH_S),
    .SEND_DA_LOW_S(SEND_DA_LOW_S),
    .RCVE_DA_HIGH_S(RCVE_DA_HIGH_S),
    .RCVE_DA_LOW_S(RCVE_DA_LOW_S),
    .ADDRESS_S(ADDRESS_S),
    .PARITYERROR(PARITYERROR),
    .DI(DI), .DOBUF(DOBUF), .DO(DO), .STATE(STATE),
    .SCLH(SCLH), .SCLL(SCLL),
    .SDA(j10_1), .SCL(j10_5), .SDA_PULL(j10_2), .SCL_PULL(j10_6),
    .CE(CE), .RST(RST), .CLK(CLK)
);
```

f). 按键消抖

```
deUstb deKey1(key1,~key_1,CLK);
```

g). 写数据

```
always @(posedge CLK)begin
    if(RST)begin
        dataTobeSend <=8'h55;
        DI <=0;
        SIC <= 0;
        STOS<=0;
        pass<=0;
    end
    else begin
        if(SIO)begin
            case(STATE)
                S_CM_WAIT_AD:begin
                    DI<=8'haa;//发送从机地址和写操作位
                    SIC<=1; //置位SIC,清除SIO
                end
                S_CM_WAIT_NACK:begin
                    STOS <=1;//置位STOS
                    SIC<=1;//置位SIC,清除SIO
                end
```

```
                    S_CM_WAIT_W_DA:begin
                        if(&dataCnt)begin//写数据结束，置位STOS
                            STOS <=1; //写完数据后，置位STOS
                            dataCnt <= 0;
                        end
                        else begin
                            DI <= dataTobeSend;//发送寄存器地址
                            dataTobeSend <= dataTobeSend+1;//依次发送数据
                            dataCnt <= dataCnt+1;
                        end
                        SIC<=1;
                    end
                    S_IDLE:begin
                        pass <=1;
                        SIC<=1;
                    end
                    default: begin
                        DI<=8'h00;
                        SIC<=0;
                    end
                endcase
            end
            else begin
                DI<=8'h00;
                SIC<=0;
                STOS<=0;
            end
        end
end
```

h).

i). 拉出观察的信号

```
assign j9_4 = sda_line;//a1:0
assign j9_5 = scl_line;
assign j9_6 = STAS;
.....
```

8.3 从机的顶层设计

从机的顶层设计 top.v 见附件 9.2 与主机的顶层设计类似，只是数据处理模块不同。

```
always @(posedge CLK)begin
    if(RST)begin
        slaveState <=0;
    end
```

```
    else begin
    if(SIO)begin
        case (STATE)
            S_SL_W_DA:begin//slave recieving write data
                if(ramIndex ==dataNum)begin
                    writeBuf[ramIndex]   <= DO;
                    ramIndex <=0;
                end
                else begin
                    writeBuf[ramIndex]   <= DO;
                    ramIndex <= ramIndex+1;
                end
            end
        endcase
    end
    else begin
        case (STATE)
            S_IDLE:begin
                SIC<=1;
            end
            S_SL_RCVE_AD:begin//slave receiving address
                AAS <=1;
            end
            S_SL_SEND_W_ACK:begin//slave sending ack bit for a write address
                ACS <=1;
            end
            default:begin
                AAS<=0;
                ACS<=0;
                SIC<=0;
                DI<=0;
            end
        endcase
        end
    end
end
```

8.4 板级连接

进行必要的约束后，通过综合、布局布线、产生 bitstream 文件。通过 Gowin 下载线把 bitstream 文件下载到开发板或测试版，可通过测试接口观察主机和从机之间的通信情况。本参考设计采用双板方式完成 I3C SDR 主从机之间的通信，即一块 GW1N-9K 板上放 I3C SDR 主机，另外一块 GW1N-9K 板上放 I3C SDR 从机。因此板间连接需三根线：一根作为 SCL；一根作为 SDA；一根作为 GND。约束文件可参考以下格式：

```
IO_LOC "clk_ext" 6;
IO_LOC "clk_in" 56;
IO_LOC "key_1" 43;
IO_LOC "key_2" 44;
IO_LOC "key_3" 45;
IO_LOC "led_3" 47;
IO_LOC "led_4" 57;
IO_LOC "led_5" 60;
IO_LOC "led_6" 61;
IO_PORT "clk_ext"    SINGLE_RESISTOR=OFF;
IO_PORT "j10_1"    PULL_MODE=NONE;
IO_PORT "j10_5"    PULL_MODE=NONE;
IO_PORT "j10_2"    PULL_MODE=NONE;
IO_PORT "j10_6"    PULL_MODE=NONE;
```

8.5 结果观察

本参考设计采用示波器观察 SCL、SDA 是否正确。另外，若想观察 STATE、DI、DOBUF 等 Bus 信号，建议采用逻辑分析仪。

9 附件

9.1 主机顶层设计

主机顶层设计的代码如下：

```
module top(
input clk_ext,
input clk_in,

input key_1,
input key_2,
input key_3,
input key_4,

input sw_4,
input sw_5,
input sw_6,
input sw_7,

output led_3,
output led_4,
output led_5,
output led_6,

output j8_3,
output j8_4,
output j8_5,
output j8_6,
output j8_7,
output j8_8,
output j8_9,
output j8_10,
output j8_11,
output j8_12,
```

```
output j8_13,
output j8_14,
output j8_15,
output j8_16,
output j8_17,
output j8_18,
input j8_19,
input j8_20,
input j8_21,
input j8_22,
input j8_23,
input j8_24,
input j8_25,
input j8_26,
input j8_27,
input j8_28,
input j8_29,
input j8_30,
input j8_31,
input j8_32,
input j8_33,
input j8_34,
input j8_35,
input j8_36,
input j8_37,
input j8_38,

output j9_3,
output j9_4,
output j9_5,
output j9_6,
output j9_7,
output j9_8,
output j9_9,
output j9_10,
output j9_11,
output j9_12,
output j9_13,
output j9_14,
output j9_15,
output j9_16,
output j9_17,
output j9_18,
output j9_19,
```

```
output j9_20,
output j9_21,
output j9_22,
output j9_23,
output j9_24,
output j9_25,
output j9_26,
output j9_27,
output j9_28,
output j9_29,
output j9_30,
output j9_31,
output j9_32,
output j9_33,
output j9_34,
output j9_35,
output j9_36,
output j9_37,
output j9_38,

inout j10_1,
output j10_2,
inout j10_5,
output j10_6,
input j10_9,
input j10_10,
input j10_13,
input j10_14,
input j10_17,
input j10_18,

input j11_1,
input j11_2,
input j11_5,
input j11_6,
input j11_9,
input j11_10,
input j11_13,
input j11_14,
input j11_17,
input j11_18
);

wire midClk;
```

```
GW_PLL myPll(
        .clkout(midClk), //output clkout 10M
        .clkin(clk_ext) //input clkin 50M
    );
wire lowClk;                //10K
wire lowOver;
wire [31:0] lowOut;
counter cnt1(lowOut,lowClk,lowOver,midClk,1'b1);
defparam cnt1.OVER = 1_000;

wire veryLowClk;            //10Hz
wire veryLowOver;
wire [31:0] veryLowOut;
counter cnt2(veryLowOut,veryLowClk,veryLowOver,midClk,1'b1);
defparam cnt2.OVER = 1_000_000;

wire secondClk;             //1Hz
wire secondOver;
wire [31:0] secondOut;
counter cnt3(secondOut,secondClk,secondOver,midClk,1'b1);
defparam cnt3.OVER = 10_000_000;


localparam  S_IDLE              = 8'h00;//-----state idle
localparam  S_CM_STA            = 8'h01;//-----state current master send start
localparam  S_CM_WAIT_AD        = 8'h02;//-----state current master wait user for address
localparam  S_CM_SEND_AD        = 8'h03;//-----state current master send address to slave
localparam  S_CM_ADRS_ABTR      = 8'h04;//-----state current master send address but arbitrate
lost
localparam  S_CM_ADRS_ABTR_OK   = 8'h05;//-----state current master send address but arbitrate
lost
localparam  S_CM_WAIT_ACK_HF    = 8'h06;//-----state current master wait slave for ACK with
handoff
localparam  S_CM_WAIT_ACK       = 8'h07;//-----state current master wait slave for ACK without
handoff
localparam  S_CM_WAIT_NACK      = 8'h08;//-----state current master recv NACK
localparam  S_CM_WAIT_W_DA      = 8'h09;//-----state current master wait user for write data
localparam  S_CM_W_STA          = 8'h0a;//-----state current master send wAddress rcve ack but
start
localparam  S_CM_W_STO          = 8'h0b;//-----state current master send wAddress stop
localparam  S_CM_W_ACK_STO      = 8'h0c;//-----state current master send wAddress rcve ack but
stop
localparam  S_CM_W_SEND_DA      = 8'h0d;//-----state current master send writing data
```

```
localparam    S_CM_SEND_DA_OK           = 8'h0e;//-----state current master send data OK
localparam    S_CM_READ_SL      = 8'h0f;//-----state current master read slave
localparam    S_CM_READ_SL_OK       = 8'h10;//-----state current master read slave ok wait T bit
localparam    S_CM_READ_OK_SI       = 8'h11;//-----state current master read slave recv continue SI
localparam    S_CM_SL_END           = 8'h12;//-----state current master read slave recv end tbit

localparam    S_CI2CM_W_SEND_DA   = 8'h13;//-----state current I2C master write sending data
localparam    S_CI2CM_SEND_DA_OK = 8'h14;//-----state current I2C master send data OK
localparam    S_CI2CM_WAIT_ACK_HF= 8'h15;//-----state current I2C master wait slave for ACK with
handoff
localparam    S_CI2CM_WAIT_NACK    = 8'h16;//-----state current I2C master recv NACK
localparam    S_CI2CM_READ_SL       = 8'h17;//-----state current I2C master read slave
localparam    S_CI2CM_SEND_ACK      = 8'h18;//-----state current I2C master send ACK bit
localparam    S_CI2CM_SEND_ACK_OK      = 8'h19;//-----state current I2C master send ACK bit OK

localparam    S_CM_LINE_STA           = 8'h1a;//-----state current master reponses slave initated
START
localparam    S_CM_DAA_READ           = 8'h1b;//-----state current master enter dynamic address
assignment read 8 Bytes
localparam    S_CM_DAA_WRITE          = 8'h1c;//-----state current master enter dynamic address
assignment write 1 byte
localparam    S_CM_DAA_SL_ACKDA   = 8'h1d;//-----state current master enter dynamic address
assignment, slave ack the dynamic address
localparam    S_CM_DAA_SL_NACKDA= 8'h1e;//-----state current master enter dynamic address
assignment, slave nack the dynamic address

localparam    S_SL_LINE_STA       = 8'h1f;//-----state slave recive line start
localparam    S_SL_STA              = 8'h20;//-----state slave start
localparam    S_SL_RCVE_AD        = 8'h21;//-----state slave receiving address
localparam    S_SL_SEND_W_ACK         = 8'h22;//-----state slave sending ack bit for a write address
localparam    S_SL_W_DA               = 8'h23;//-----state slave recieving write data
localparam    S_SL_W_DA_NINE          = 8'h24;//-----state slave recieving the ninth bit
localparam    S_SL_R_DA               = 8'h25;//-----state slave sending read data
localparam    S_SL_SEND_R_ACK         = 8'h26;//-----state slave sending ack bit for a read address
localparam    S_SL_R_DA_NINE          = 8'h27;//-----state slave sending the ninth bit
localparam    S_SL_SEND_AD            = 8'h28;//-----state slave sending an address
localparam    S_SL_RCVE_ACK           = 8'h29;//-----state slave receiving the ack bit
localparam    S_SL_CCC_W_DA           = 8'h2a;//-----state slave receiving the Ccc data
localparam    S_SL_DAA_READ           = 8'h2b;//-----state slave sending 8 bytes x 8bits
localparam    S_SL_DAA_WRITE          = 8'h2c;//-----state slave receiving 8bits
localparam    S_SL_DAA_ACK        = 8'h2d;//-----state slave sending the dynamic address ack bit

    `ifdef DEBUG_REG
    wire dbg_ABTR;
```

```
    `endif
wire CMS;
wire STAS;
reg  LGYS, ACS, AAS, STOS;
wire      LGYO, CMO, ACO, AAO, SIO, STOO, STAO;
reg  LGYC, CMC, ACC, AAC, SIC, STOC, STAC;

reg  STA_HD_S,
     SEND_AD_HIGH_S,
     SEND_AD_LOW_S,
     ACK_HIGH_S,
     ACK_LOW_S,
     STO_HIGH_S,
     STO_LOW_S,
     SEND_DA_HIGH_S,
     SEND_DA_LOW_S,
     RCVE_DA_HIGH_S,
     RCVE_DA_LOW_S,
     ADDRESS_S;

wire PARITYERROR;
reg  [7:0] DI=0;
wire      [7:0] DOBUF, DO, STATE;
reg  [7:0] SCLH, SCLL;
wire      SDA, SCL, SDA_PULL, SCL_PULL;
wire      CE, RST, CLK;

I3C_REG_top master(
     .LGYS(LGYS), .CMS(CMS), .ACS(ACS), .AAS(AAS), .STOS(STOS), .STAS(STAS),
     .LGYO(LGYO), .CMO(CMO), .ACO(ACO), .AAO(AAO), .SIO(SIO), .STOO(STOO), .STAO(STAO),
     .LGYC(LGYC), .CMC(CMC), .ACC(ACC), .AAC(AAC), .SIC(SIC), .STOC(STOC), .STAC(STAC),

     .STA_HD_S(STA_HD_S),
     .SEND_AD_HIGH_S(SEND_AD_HIGH_S),
     .SEND_AD_LOW_S(SEND_AD_LOW_S),
     .ACK_HIGH_S(ACK_HIGH_S),
     .ACK_LOW_S(ACK_LOW_S),
     .STO_HIGH_S(STO_HIGH_S),
     .STO_LOW_S(STO_LOW_S),
     .SEND_DA_HIGH_S(SEND_DA_HIGH_S),
     .SEND_DA_LOW_S(SEND_DA_LOW_S),
     .RCVE_DA_HIGH_S(RCVE_DA_HIGH_S),
     .RCVE_DA_LOW_S(RCVE_DA_LOW_S),
```

```
    .ADDRESS_S(ADDRESS_S),
    .PARITYERROR(PARITYERROR),

    .DI(DI), .DOBUF(DOBUF), .DO(DO), .STATE(STATE),
    .SCLH(SCLH), .SCLL(SCLL),
    .SDA(j10_1), .SCL(j10_5), .SDA_PULL(j10_2), .SCL_PULL(j10_6),
    .CE(CE), .RST(RST), .CLK(CLK)
    `ifdef DEBUG_REG
    ,.dbg_ABTR(dbg_ABTR)
    `endif
);

wire key1,key2,key3,key4;
wire sw4,sw5,sw6,sw7;
deUstb deKey1(key1,~key_1,lowClk);
deUstb deKey2(key2,~key_2,lowClk);
deUstb deKey3(key3,~key_3,lowClk);
deUstb deKey4(key4,~key_4,lowClk);
deUstb deSw4(sw4,sw_4,lowClk);
deUstb deSw5(sw5,sw_5,lowClk);
deUstb deSw6(sw6,sw_6,lowClk);
deUstb deSw7(sw7,sw_7,lowClk);

reg [7:0] dataTobeSend = 8'h55;
reg [7:0] dataCnt =0;
reg pass =0;
assign CLK = lowClk;
assign RST = key1;
assign CMS = key2;
assign STAS =key3;
assign CE =1'b1;

always @(posedge CLK)begin
    if(RST)begin
        dataTobeSend <=8'h55;
        DI <=0;
        SIC <= 0;
        STOS<=0;            pass<=0;
    end
    else begin
        if(SIO)begin
            case(STATE)
                S_CM_WAIT_AD:begin
                    DI<=8'haa;
```

```
                    SIC<=1;
                end
                S_CM_WAIT_NACK:begin
                    STOS <=1;
                    SIC<=1;
                end
                S_CM_WAIT_W_DA:begin
                    if(&dataCnt)begin
                        STOS <=1;
                        dataCnt <= 0;
                    end
                    else begin
                        DI <= dataTobeSend;
                        dataTobeSend <= dataTobeSend+1;
                        dataCnt <= dataCnt+1;
                    end
                    SIC<=1;
                end                    S_IDLE:begin                    pass <=1;
SIC<=1;                     end
                default: begin
                    DI<=8'h00;
                    SIC<=0;
                end
            endcase
        end
        else begin
            DI<=8'h00;
            SIC<=0;
            STOS<=0;
        end
    end
end
wire sda_line = master.u_i3c_top_inst.SDA_LINE;
wire scl_line = master.u_i3c_top_inst.SCL_LINE;

assign j9_3 = CLK;
assign j9_4 = sda_line;
assign j9_5 = scl_line;
assign j9_6 = STAS;
assign j9_7 = STAO;
assign j9_8 = STOO;
assign j9_9 = SIO;
assign j9_10 = AAO;
assign j9_11 = ACO;
```

```
assign j9_12 = DI[0];
assign j9_13 = DI[1];
assign j9_14 = DI[2];
assign j9_15 = DI[3];
assign j9_16 = DI[4];
assign j9_17 = DI[5];
assign j9_18 = DI[6];
assign j9_19 = DI[7];
assign j9_20 = DOBUF[0];
assign j9_21 = DOBUF[1];
assign j9_22 = DOBUF[2];
assign j9_23 = DOBUF[3];
assign j9_24 = DOBUF[4];
assign j9_25 = DOBUF[5];
assign j9_26 = DOBUF[6];
assign j9_27 = DOBUF[7];
assign j9_28 = STATE[0];

assign j9_29 = STATE[1];
assign j9_30 = STATE[2];
assign j9_31 = STATE[3];
assign j9_32 = STATE[4];
assign j9_33 = STATE[5];
assign j9_34 = STATE[6];
assign j9_35 = STATE[7];

assign led_3 = ~(secondOut<100_000 & secondClk);
assign led_4 = ~CMO;
assign led_5 = ~SIO;
assign led_6 = ~pass;

endmodule
```

9.2 从机顶层设计

从机的顶层设计的代码如下：

```
module top(
input clk_ext,
input clk_in,

input key_1,
input key_2,
input key_3,
input key_4,
```

```
input sw_4,
input sw_5,
input sw_6,
input sw_7,

output led_3,
output led_4,
output led_5,
output led_6,

input j8_3,
input j8_4,
input j8_5,
input j8_6,
input j8_7,
input j8_8,
input j8_9,
input j8_10,
input j8_11,
input j8_12,
input j8_13,
input j8_14,
input j8_15,
input j8_16,
input j8_17,
input j8_18,
input j8_19,
input j8_20,
input j8_21,
input j8_22,
input j8_23,
input j8_24,
input j8_25,
input j8_26,
input j8_27,
input j8_28,
input j8_29,
input j8_30,
input j8_31,
input j8_32,
input j8_33,
input j8_34,
input j8_35,
```

```
input j8_36,
input j8_37,
input j8_38,

output j9_3,
output j9_4,
output j9_5,
output j9_6,
output j9_7,
output j9_8,
output j9_9,
output j9_10,
output j9_11,
output j9_12,
output j9_13,
output j9_14,
output j9_15,
output j9_16,
output j9_17,
output j9_18,
output j9_19,
output j9_20,
output j9_21,
output j9_22,
output j9_23,
output j9_24,
output j9_25,
output j9_26,
output j9_27,
output j9_28,
output j9_29,
output j9_30,
output j9_31,
output j9_32,
output j9_33,
output j9_34,
output j9_35,
output j9_36,
output j9_37,
output j9_38,

inout j10_1,
output j10_2,
inout j10_5,
```

```
output j10_6,
input j10_9,
input j10_10,
input j10_13,
input j10_14,
input j10_17,
input j10_18,

input j11_1,
input j11_2,
input j11_5,
input j11_6,
input j11_9,
input j11_10,
input j11_13,
input j11_14,
input j11_17,
input j11_18
);

wire midClk;

GW_PLL myPll(
        .clkout(midClk), //output clkout 10M
        .clkin(clk_ext) //input clkin 50M
    );
wire lowClk;                    //10K
wire lowOver;
wire [31:0] lowOut;
counter cnt1(lowOut,lowClk,lowOver,midClk,1'b1);
defparam cnt1.OVER = 1_000;

wire veryLowClk;            //10Hz
wire veryLowOver;
wire [31:0] veryLowOut;
counter cnt2(veryLowOut,veryLowClk,veryLowOver,midClk,1'b1);
defparam cnt2.OVER = 1_000_000;

wire secondClk;             //1Hz
wire secondOver;
wire [31:0] secondOut;
counter cnt3(secondOut,secondClk,secondOver,midClk,1'b1);
defparam cnt3.OVER = 10_000_000;
```

```
wire tenSecondClk;              //0.1Hz
wire tenSecondOver;
wire [31:0] tenSecondOut;
counter cnt4(tenSecondOut,tenSecondClk,tenSecondOver,midClk,secondOver);
defparam cnt4.OVER = 10;



    `ifdef DEBUG_REG
    wire dbg_ABTR;
    `endif
reg  LGYS, CMS, ACS, AAS, STOS, STAS;
wire     LGYO, CMO, ACO, AAO, SIO, STOO, STAO;
reg  LGYC, CMC, ACC, AAC, SIC, STOC, STAC;

reg  STA_HD_S,
    SEND_AD_HIGH_S,
    SEND_AD_LOW_S,
    ACK_HIGH_S,
    ACK_LOW_S,
    STO_HIGH_S,
    STO_LOW_S,
    SEND_DA_HIGH_S,
    SEND_DA_LOW_S,
    RCVE_DA_HIGH_S,
    RCVE_DA_LOW_S,
    ADDRESS_S;

wire PARITYERROR;
reg  [7:0] DI;
wire     [7:0] DOBUF, DO, STATE;
reg  [7:0] SCLH, SCLL;
wire     SDA, SCL, SDA_PULL, SCL_PULL;
wire     CE, RST, CLK;

I3C_REG_top slave(
    .LGYS(LGYS), .CMS(CMS), .ACS(ACS), .AAS(AAS), .STOS(STOS), .STAS(STAS),
    .LGYO(LGYO), .CMO(CMO), .ACO(ACO), .AAO(AAO), .SIO(SIO), .STOO(STOO), .STAO(STAO),
    .LGYC(LGYC), .CMC(CMC), .ACC(ACC), .AAC(AAC), .SIC(SIC), .STOC(STOC), .STAC(STAC),

    .STA_HD_S(STA_HD_S),
    .SEND_AD_HIGH_S(SEND_AD_HIGH_S),
    .SEND_AD_LOW_S(SEND_AD_LOW_S),
    .ACK_HIGH_S(ACK_HIGH_S),
    .ACK_LOW_S(ACK_LOW_S),
```

```
    .STO_HIGH_S(STO_HIGH_S),
    .STO_LOW_S(STO_LOW_S),
    .SEND_DA_HIGH_S(SEND_DA_HIGH_S),
    .SEND_DA_LOW_S(SEND_DA_LOW_S),
    .RCVE_DA_HIGH_S(RCVE_DA_HIGH_S),
    .RCVE_DA_LOW_S(RCVE_DA_LOW_S),

    .ADDRESS_S(ADDRESS_S),
    .PARITYERROR(PARITYERROR),

    .DI(DI), .DOBUF(DOBUF), .DO(DO), .STATE(STATE),
    .SCLH(SCLH), .SCLL(SCLL),
    .SDA(j10_1), .SCL(j10_5), .SDA_PULL(j10_2), .SCL_PULL(j10_6),
    .CE(CE), .RST(RST), .CLK(CLK)
    `ifdef DEBUG_REG
    ,.dbg_ABTR(dbg_ABTR)
    `endif
);
defparam slave.iregInterface.P_SLAVE_STATIC_ADDRESS=7'b1010101;
wire key1,key2,key3,key4;
wire sw4,sw5,sw6,sw7;

deUstb deKey1(key1,~key_1,lowClk);
deUstb deKey2(key2,~key_2,lowClk);
deUstb deKey3(key3,~key_3,lowClk);
deUstb deKey4(key4,~key_4,lowClk);
deUstb deSw4(sw4,sw_4,lowClk);
deUstb deSw5(sw5,sw_5,lowClk);
deUstb deSw6(sw6,sw_6,lowClk);
deUstb deSw7(sw7,sw_7,lowClk);

parameter i2c_s_addr = 7'b000_1000;//address for i2c slave
parameter wr_start_addr = 8'ha0;//write and read start address
reg [7:0] dataNum =10/* synthesis syn_keep=1 */;
reg [7:0] slaveState =0/* synthesis syn_keep=1 */;
reg [7:0] writeBuf [0:255]/* synthesis syn_keep=1 */;
reg [9:0] ramIndex =0/* synthesis syn_keep=1 */;
reg [7:0] w_cmp_buf [0:255]/* synthesis syn_keep=1 */;

assign CLK = lowClk;
assign RST = key1;
assign CE =1'b1;

always @(posedge CLK)begin
```

```
    if(RST)begin
        slaveState <=0;
    end
    else begin
    if(SIO)begin
        case (STATE)
            8'h23:begin//slave recieving write data
                if(slaveState ==4)begin
                    if(ramIndex ==dataNum)begin
                        writeBuf[ramIndex]   <= DO;
                        ramIndex <=0;
                        slaveState <= 5;
                    end
                    else begin
                        writeBuf[ramIndex]   <= DO;
                        ramIndex <= ramIndex+1;
                        slaveState <= 4;
                    end
                end
            end
        endcase
    end
    else begin
        case (STATE)
            8'h00:begin//idle
                if(key2==1)begin
                    slaveState   <= 2;
                end
                if(slaveState ==0)begin
                    //DI <= {i2c_s_addr,1'b0};
                   // ADDRESS_S <=1;
                    slaveState <= 1;
                end
                if(slaveState ==1)begin
                //      DI <= 8'h00;
               //       ADDRESS_S <=0;
                    slaveState <= 2;
                end
            end
            8'h21:begin//slave receiving address
                ramIndex <= 0;
                if(slaveState ==2)begin
                    AAS <=1;
                    slaveState <= 3;
```

```
            end
            if(slaveState ==3)begin
                AAS <=0;
                slaveState <= 4;
            end
            if(slaveState ==5)begin
                AAS <=1;
                slaveState <= 6;
            end
            if(slaveState ==6)begin
                AAS <=0;
                slaveState <= 10;
            end
            if(slaveState ==13)begin
                DI   <= writeBuf[ramIndex];
                ACS <=1;
                ramIndex <= ramIndex+1;
                AAS <=1;
                slaveState <= 14;
            end
            if(slaveState ==14)begin
                DI   <=0;
                ACS <=0;
                AAS <=0;
                slaveState <= 15;
            end
        end
        8'h22:begin//slave sending ack bit for a write address
            if(slaveState ==10)begin
                DI   <= writeBuf[ramIndex];
                ACS <=1;
                ramIndex <= ramIndex+1;
                slaveState <= 11;
            end
            if(slaveState ==11)begin
                DI <=0;
                ACS <=0;
                slaveState <= 13;
            end
        end
        8'h26:begin//slave sending ack bit for a read address
            if(slaveState ==15)begin
                DI   <= writeBuf[ramIndex];
                ACS <=1;
```

```
                        AAS <=1;
                        ramIndex <= ramIndex+1;
                        slaveState <= 16;
                    end
                    if(slaveState ==16)begin
                        AAS <=0;
                        ACS <=0;
                        slaveState <= 17;
                    end
                end
                8'h27:begin//slave sending the ninth bit
                    if(slaveState ==17)begin
                        if(ramIndex <=dataNum)begin
                            DI   <= writeBuf[ramIndex];
                            ACS <=1;
                            //AAS <= 1;
                            ramIndex <= ramIndex+1;
                            slaveState <= 18;
                        end
                    end
                end
                8'h25:begin//slave sending read data
                    if(slaveState ==18)begin
                        ACS <=0;
                        slaveState <= 17;
                    end
                end
            endcase
            end
        end
end

wire sda_line = slave.SDA_LINE;

wire scl_line = slave.SCL_LINE;

assign j9_3 = CLK;
assign j9_4 = sda_line;//A3_0
assign j9_5 = scl_line;
assign j9_6 = STAS;
assign j9_7 = STAO;
assign j9_8 = STOO;
assign j9_9 = SIO;
assign j9_10 = AAO;
```

```
assign j9_11 = ACO;//A3_7
assign j9_12 = DI[0];//A3_8
assign j9_13 = DI[1];
assign j9_14 = DI[2];
assign j9_15 = DI[3];
assign j9_16 = DI[4];
assign j9_17 = DI[5];
assign j9_18 = DI[6];
assign j9_19 = DI[7];
assign j9_20 = DOBUF[0];//A4_0
assign j9_21 = DOBUF[1];
assign j9_22 = DOBUF[2];
assign j9_23 = DOBUF[3];
assign j9_24 = DOBUF[4];
assign j9_25 = DOBUF[5];
assign j9_26 = DOBUF[6];
assign j9_27 = DOBUF[7];
assign j9_28 = STATE[0];//A4_8
assign j9_29 = STATE[1];
assign j9_30 = STATE[2];
assign j9_31 = STATE[3];
assign j9_32 = STATE[4];
assign j9_33 = STATE[5];
assign j9_34 = STATE[6];
assign j9_35 = STATE[7];

assign led_3 = ~(secondOut<100_000 & secondClk);
assign led_4 = ~CMO;
assign led_5 = ~STAO;
assign led_6 = ~SIO;
endmodule
```


GOWIN高云
智慧逻辑 定制未来