Gowin I3C SDR IP

User Guide

IPUG508-1.2E, 11/09/2018

Revision History

Date	Version	Description
01/19/2018	1.0E	Initial version published.
09/26/2018	1.1E	IP name modified.
11/09/2018	1.2E	The supported products modified.

Contents

List of Figures

List of Tables

1 About This Guide

1.1 Purpose

The purpose of this Gowin I3C SDR IP User Guide is to help users to quickly understand the features and usage of Gowin I3C SDR IP by providing an overview of the functions, signal definition, working principle, and GUI call, among other functionality.

1.2 Supported Products

The information in this guide applies to the following products:

- GW1N series of FPGA products: GW1N-2, GW1N-2B, GW1N-4, GW1N-4B, GW1N-6ES, GW1N-6, GW1N-9ES, and GW1N-9;
- GW1NR series of FPGA products: GW1NR-4, GW1NR-4B GW1NR-9ES, and GW1NR-9;
- GW1NS series of FPGA products: GW1NS-2, GW1NS-2C;
- GW2A series of FPGA products: GW2A-18, GW2A-55;
- GW2AR series of FPGA products: GW2AR-18.

1.3 Related Documents

The latest user guides are available on the Gowin website. Refer to the related documents at www.gowinsemi.com:

- GW1N series of FPGA Products Data Sheet
- GW1NR series of FPGA Products Data Sheet
- GW1NS series of FPGA Products Data Sheet
- GW2A series of FPGA Products Data Sheet
- GW2AR series of FPGA Products Data Sheet
- Gowin YunYuan Software User Guide

1.4 Abbreviations and Terminology

Table 1-1 Abbreviations and Terminology

Abbreviations and Terminology	Full Name/Meaning
FPGA	Field Programmable Gate Array
I3C Bus	Inter-Integrated Circuit Bus (I3C Serial Bus)

1.5 Support and Feedback

Gowin Semiconductor provides customers with comprehensive technical support. If you have any questions, comments, or suggestions, please feel free to contact us directly using the information presented below.

Website: www.gowinsemi.com
E-mail: support@gowinsemi.com
Tel: +86 755 8262 0391

2 Function

2.1 Overview

I3C bus is a two-pin serial bus that was released by the MIPI Alliance. It offers the key features of I2C and SPI, low pin count, low power consumption, high capacity, and novel performance. Furthermore, I3C bus is extensible and is compatible with I2C. As such, its use can effectively reduce the physical ports of the IC system, support low power consumption, increase data rate, and provide users with access to many of the other advantages of existing port protocols and, subsequently, the enhancements that support modern mobile handheld devices, intelligent driving, and IOT design.

Gowin I3C SDR IP follows MIPI alliance I3C SDR bus protocol and adopts register interfaces. Integrated with I3C SDR Master and I3C SDR Slave, it can instantiate I3C SDR Master or I3C SDR Slave and realize the communication between I3C SDR Master and I3C SDR/I2C Slave.

2.2 Features

2.2.1 Gowin I3C SDR Master

1. Compliant with MIPI I3C protocol.
2. Supports I3C address arbitration detection.
3. Supports single data rate (SDR) mode.
4. Offers a max. data transmission rate of up to 12.5 Mbps.
5. Can generate start/stop/repeated start/acknowledge.
6. Can detect start/stop/repeated start.
7. Supports the dynamic allocation of address via SETDASA or ENTDAA.
8. Supports data transmission and reception.
9. Supports in-band interrupts.
10. Supports hot-join.
11. Supports dynamic allocation of address when hot-join is employed.
12. Supports CCC command.
13. Supports dynamic adjustment of SCL frequency.

14. Compatible with I2C Slave.
15. Adopts the register interfaces.

2.2.2 Gowin I3C SDR Slave

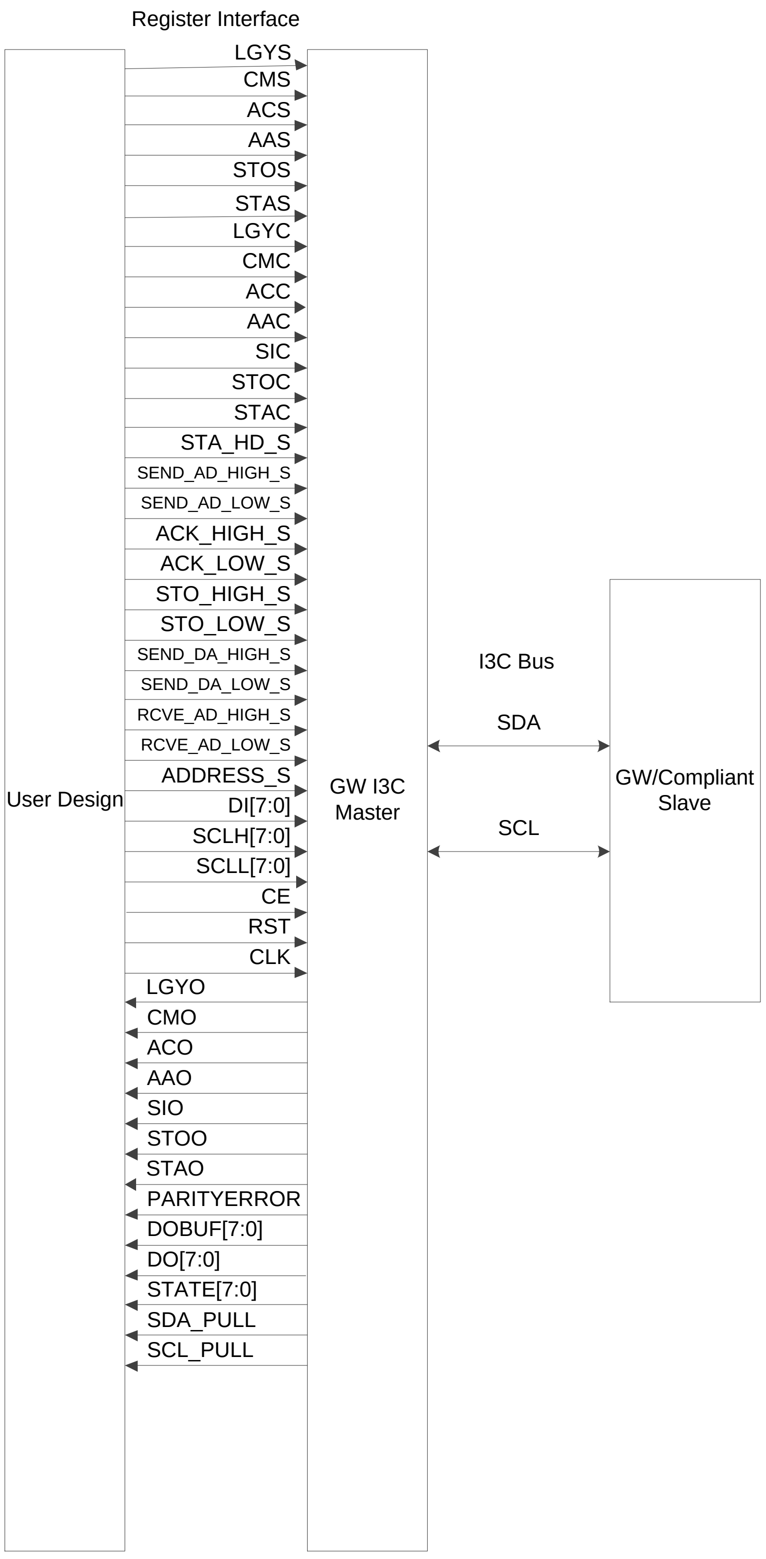
1. Compliant with MIPI I3C protocol.
2. Can generate start/acknowledge.
3. Can detect start/stop/repeated start/acknowledge.
4. Supports the dynamic allocation of address via SETDASA or ENTDAA.
5. Supports data transmission and reception.
6. Can send an IBI or hot-join request. If more than one slave sends the IBI or hot-join requests, the min. address obtains the arbitration.
7. Employs static address of slave configuration.
8. Adopts register interfaces.

3 Signal Definition

3.1 Gowin I3C SDR Master Signals

If Gowin I3C SDR IP is used for the I3C SDR Master, the I3C SDR Master can be controlled through register interfaces. The I3C SDR Master communicates with the peripheral devices using the I3C data SDA and the clock SCL. The communication diagram for this functionality is as per Figure 3-1.

Figure 3-1 I3C SDR Master



An overview of the I3C SDR Master signals is provided in Table 3-1.

Table 3-1 Definition of Gowin I3C SDR Master Signals

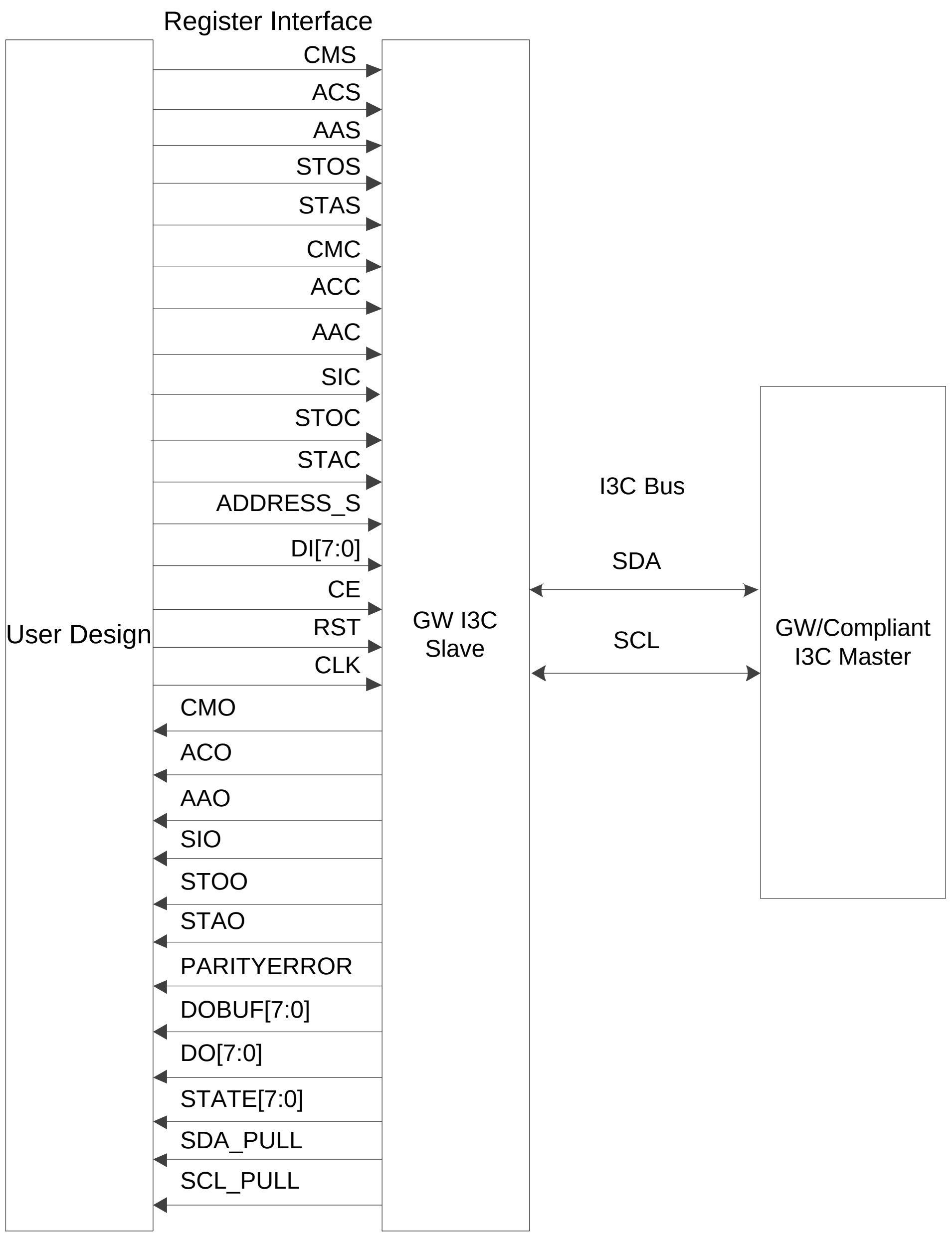
No.	Signal Name	I/O	Data Width	Description
1	LGYO	Output	1	Output the current communication object as the I2C command.
2	CMO	Output	1	Output the command of the device is in the Master mode.
3	ACO	Output	1	Continue to output when selecting whether to continue.
4	AAO	Output	1	Reply ACK when you need to reply ACK/NACK.
5	SIO	Output	1	Interrupt to output the identity bit.
6	STOO	Output	1	Output the STOP command.
7	STAO	Output	1	Output the START command.
8	PARITYERROR	Output	1	Output check when receiving data.
9	DOBUF	Output	[7:0]	Data output after caching.
10	DO	Output	[7:0]	Data output directly.
11	STATE	Output	[7:0]	Output the internal state.
12	SDA_PULL	Output	1	Controllable pull-up of the I3C serial data.
13	SCL_PULL	Output	1	Controllable pull-up of the I3C serial clock.
14	SDA	Inout	1	I3C serial data line.
15	SCL	Inout	1	I3C serial clock line.
16	LGYS	Input	1	The current communication object is the I2C setting signal.
17	CMS	Input	1	The device enters the master setting signal.
18	ACS	Input	1	Select the setting signal when determining whether to continue.
19	AAS	Input	1	Reply the ACK setting signal when a reply is required from the ACK/NACK.
20	STOS	Input	1	Input the STOP command.
21	STAS	Input	1	Input the START command.
22	LGYC	Input	1	The current communication object is the I2C.
23	CMC	Input	1	The reset signal that the device is in master.
24	ACC	Input	1	The reset signal that selects continue when selecting whether to continue.
25	AAC	Input	1	Reply the ACK reset signal when a reply is required from the ACK/NACK.
26	SIC	Input	1	Interrupt to identify the reset signal.
27	STOC	Input	1	The reset signal is in STOP state.
28	STAC	Input	1	The reset signal is in START state.
29	STA_HD_S	Input	1	Adjust the setting signal when generating START.
30	SEND_AD_HIG	Input	1	Adjust the setting signal of SCL at a high level

No.	Signal Name	I/O	Data Width	Description
	H_S			when the address is sent.
31	SEND_AD_LOW_S	Input	1	Adjust the setting signal of SCL at a low level when the address is sent.
32	ACK_HIGH_S	Input	1	Adjust the setting signal of SCL at a high level in ACK.
33	ACK_LOW_S	Input	1	Adjust the setting signal of SCL at a low level in ACK.
34	STO_HIGH_S	Input	1	Adjust the setting signal of SCL at a high level in ACK.
35	STO_LOW_S	Input	1	Adjust the setting signal of SCL at a low level in STOP.
36	SEND_DA_HIGH_S	Input	1	Adjust the setting signal of SCL at a high level when the data are sent.
37	SEND_DA_LOW_S	Input	1	Adjust the setting signal of SCL at a low level when the data are sent.
38	RCVE_DA_HIGH_S	Input	1	Adjust the setting signal of SCL at a high level when the data are received
39	RCVE_DA_LOW_S	Input	1	Adjust the setting signal of SCL at a low level when the data are received
40	ADDRESS_S	Input	1	The slave address setting interface.
41	DI	Input	[7:0]	Data Input.
42	SCLH	Input	[7:0]	Used to set the SCL cycles at a high level.
43	SCLL	Input	[7:0]	Used to set the SCL cycles at a low level.
44	CE	Input	1	Clock enable.
45	RST	Input	1	Reset signal.
46	CLK	Input	1	Clock signal.

3.2 Gowin I3C SDR Slave Signals

If the Gowin I3C SDR IP is used as the I3C SDR Slave, the I3C SDR Slave can be controlled through the register interfaces. The I3C SDR Slave communicates with the I3C SDR Master using the I3C data SDA and the SCL clock. The communication diagram for this process is as per Figure 3-2.

Figure 3-2 I3C SDR Slave



The I3C SDR Slave signals are shown in Table 3-2.

Table 3-2 Definition of the I3C SDR Slave Signals

No.	Signal Name	I/O	Data Width	Description
1	CMO	Output	1	Output the command of the device is in the into the master mode.
2	ACO	Output	1	Continue to output when selecting whether to continue.
3	AAO	Output	1	Reply ACK when a reply is required from ACK/NACK.
4	SIO	Output	1	Interrupt to output the identity bit.

No.	Signal Name	I/O	Data Width	Description
5	STOO	Output	1	Output the STOP command.
6	STAO	Output	1	Output the START command.
7	PARITYERROR	Output	1	Output check when receiving data.
8	DOBUF	Output	[7:0]	Data output after caching.
9	DO	Output	[7:0]	Data output directly.
10	STATE	Output	[7:0]	Output the internal state.
11	SDA_PULL	Output	1	Controllable pull-up the I3C serial data.
12	SCL_PULL	Output	1	Controllable pull-up the I3C serial clock.
13	SDA	Inout	1	I3C serial data line.
14	SCL	Inout	1	I3C serial clock line.
15	CMS	Input	1	The device enters the master setting signal.
16	ACS	Input	1	Select the setting signal when selecting whether to continue.
17	AAS	Input	1	Reply the ACK setting signal when a reply is required from ACK/NACK.
18	STOS	Input	1	Input the STOP command.
19	STAS	Input	1	Input the START command.
20	CMC	Input	1	The reset signal that the device is in Master.
21	ACC	Input	1	The reset signal that selects continue when selecting whether to continue.
22	AAC	Input	1	Reply the ACK reset signal when a reply from ACK/NACK is required.
23	SIC	Input	1	Interrupt to identify the reset signal.
24	STOC	Input	1	The reset signal is in STOP state.
25	STAC	Input	1	The reset signal is in START state.
26	ADDRESS_S	Input	1	The Slave address setting interface.
27	DI	Input	[7:0]	Data input.
28	CE	Input	1	Clock enable.
29	RST	Input	1	Reset signal.
30	CLK	Input	1	Clock signal.

4 GUI Parameters

4.1 Overview

The I3C SDR IP GUI interface is mainly used to set the time value by dynamically adjusting the SCL of the 3C Master and the static address of the I3C SDR Slave.

4.2 I3C SDR GUI Parameters

Table 4-1 I3C SDR Master GUI Parameters

No.	Name	Range	Default Value	Description
1	STA HD	20~254	20	Hold time at START.
2	SCL LOW	3~254	3	Low-level time of SCL.
3	SCL HIGH	2~254	2	High-level time of SCL.
4	SEND AD SCL HIGH	2~254	2	High-level time of SCL when the address is sent.
5	SEND AD SCL LOW	3~254	3	Low-level time of SCL when the address is sent.
6	ACK SCL HIGH	2~254	2	High-level time at Aclk.
7	ACK SCL LOW	3~254	3	Low-level time at Aclk.
8	STO SCL HIGH	2~254	2	High-level time at Aclk.
9	STO SCL LOW	3~254	3	Low-level time at Aclk.
10	SEND DA SCL HIGH	2~254	2	SCL time at a high level when the data are sent.
11	SEND DA SCL LOW	3~254	3	SCL time at a low level when the data are sent.
12	RCVE DA SCL HIGH	2~254	2	SCL time at a high level when the data are received.
13	RCVE DA SCL LOW	3~254	3	SCL time at a low level when the data are received.

Table 4-2 I3C SDR Slave GUI Parameters

No.	Name	Range	Default Value	Description
1	SLAVE STATIC ADDRESS	7’h00~7’h7F	7’h08	Slave static address.

5 Principle

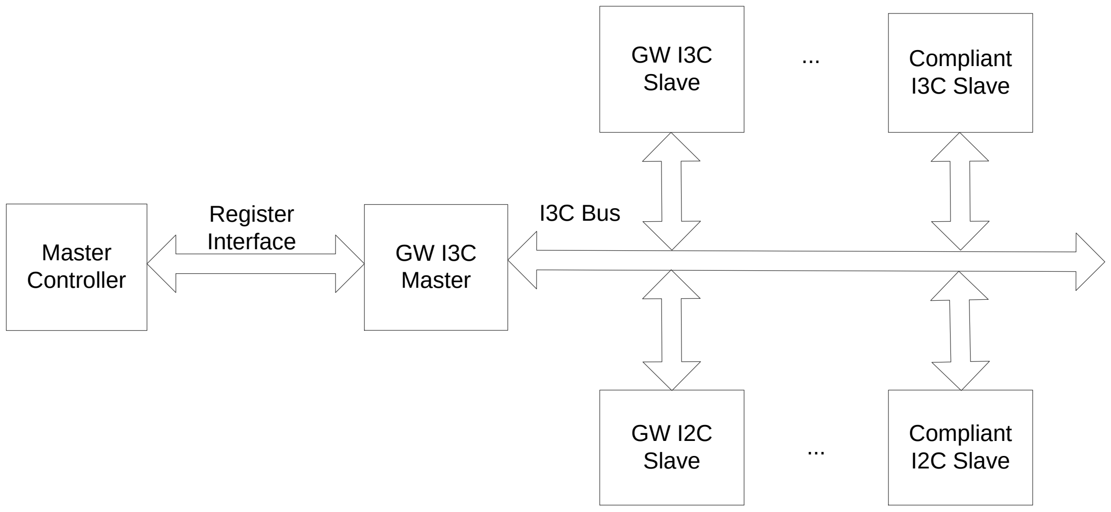
Gowin I3C SDR IP combines the I3C SDR Master with the I3C SDR Slave and is used for assigning dynamic addresses, sending/receiving data, and interrupting the application of the I3C SDR Master and I3C SDR Slave.

If the I3C SDR IP is used as an I3C SDR Master, multiple slaves can be mounted on the bus, including the I3C SDR/I2C Slave supported by the MIPI Alliance. START and STOP can be initiated and the address can be sent to the Slave; data can be sent to the I3C SDR Slave and received from the I3C SDR Slave; the dynamic address of the I3C SDR Slave can be allocated; and data can be sent to and from the 12C Slave. The I3C SDR IP can also be used to respond to IBI or hot-join requests from the I3C SDR Slave to allocate the dynamic address. If the I3C SDR IP is used as an I3C SDR Slave, the START can be initiated, the address can be sent and responded to, data can be sent and received from the I3C SDR Master and, and the dynamic address in the dynamic address assignment process can be assigned.

5.1 System Architecture

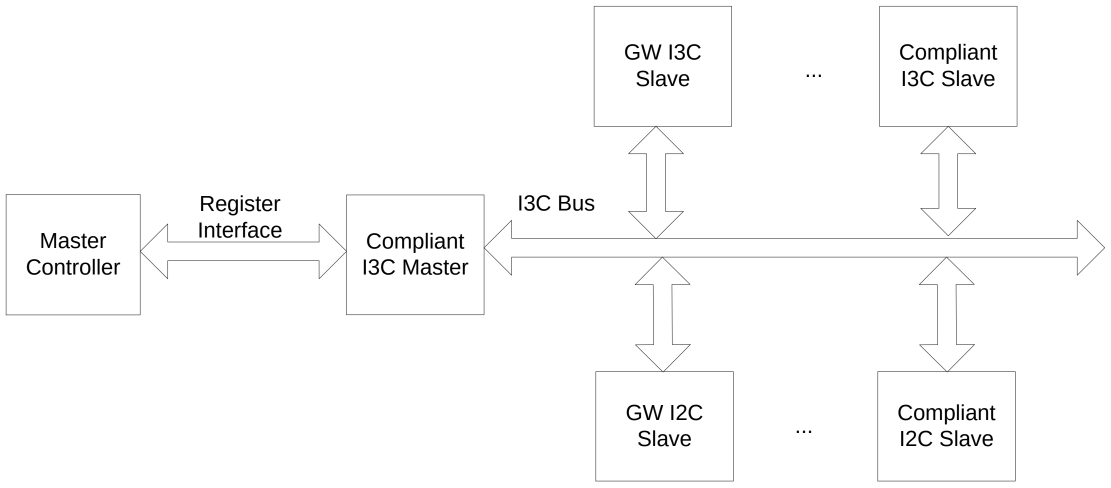
As shown in Figure 5-1, the master controller sends the commands and data to the Gowin I3C SDR Master through the register interface. The Gowin I3C SDR Master sends the information to the Gowin I3C SDR/I2C Slave or Compliant I3C SDR/I2C Slave using the I3C SDR bus, or sends the data from the Gowin I3C SDR/I2C Slave and Compliant I3C SDR/I2C Slave to the master controller using the Gowin I3C SDR Master.

Figure 5-1 System Architecture



As shown in Figure 5-2. The master controller sends the commands and data to a Compliant I3C SDR Master through the register interface. The Compliant I3C SDR Master sends the information to the Gowin I3C SDR/I2C Slave or Compliant I3C SDR/I2C Slave using the I3C bus, or transmits the data from the Gowin I3C SDR/I2C Slave and Compliant I3C SDR/I2C Slave to the master controller using the Gowin I3C SDR Master.

Figure 5-2 System Architecture



5.2 I3C SDR IP Status

The Gowin I3C SDR IP is used as the I3C SDR Master and I3C SDR Slave, as shown in the table below.

Table 5-1 I3C SDR Master Status

No.	Name	Description
1	S_IDLE	The I3C bus is available.
2	S_CM_STA	The master sends the START status.
3	S_CM_WAIT_AD	The master waits for the address provided by the user.

No.	Name	Description
4	S_CM_SEND_AD	The master sends the address from the machine.
5	S_CM_ADRS_ABTR	The master sends the address, the bus tests it.
6	S_CM_ADRS_ABTR_OK	The bus tests whether the address ends.
7	S_CM_WAIT_ACK_HF	The master waits for the ACK; the I3C bus has a mode switch.
8	S_CM_WAIT_ACK	The master waits for the ACK; the I3C bus has no mode switch.
9	S_CM_WAIT_NACK	The master receives the NACK.
10	S_CM_WAIT_W_DA	The master waits for the data provided by the user.
11	S_CM_W_STA	The master initiates the START after the write address is sent and the ACK is received.
12	S_CM_W_STO	The master initiates the STOP after the write address is sent.
13	S_CM_W_ACK_STO	The master initiates the STOP after the write address is sent and the ACK is received.
14	S_CM_W_SEND_DA	The master sends the write data.
15	S_CM_SEND_DA_OK	The master finishes sending the write data.
16	S_CM_READ_SL	The master reads the slave.
17	S_CM_READ_SL_OK	The master reads the slave, awaits the Tbit.
18	S_CM_READ_OK_SI	The master receives the SIO after reading the slave.
19	S_CM_SL_END	The master receives the Tbit after reading the slave.
20	S_CI2CM_W_SEND_DA	The master writes the data to I2C.
21	S_CI2CM_SEND_DA_OK	The master finishes writing the data to I2C.
22	S_CI2CM_WAIT_ACK_HF	The master waits for the I2C ACK; the I3C bus has a mode switch.
23	S_CI2CM_WAIT_NACK	The master receives the I2C NACK.
24	S_CI2CM_READ_SL	The master reads the I2C.
25	S_CI2CM_SEND_ACK	The master sends the ACK to the I2C slave.
26	S_CI2CM_SEND_ACK_OK	The master finishes sending the ACK to the I2C slave.
26	S_CM_LINE_STA	The master responds to the START from the slave.
28	S_CM_DAA_READ	The master ENTDAA reads 8 bytes.
29	S_CM_DAA_WRITE	The master ENTDAA writes a byte.
30	S_CM_DAA_SL_ACKDA	The dynamic address of the master is ENTDAA and the slave is ACK.
31	S_CM_DAA_SL_NACKDA	The dynamic address of the master is ENTDAA and the slave is NACK.

Table 5-2 I3C SDR Slave Status

No.	Name	Description
1	S_SL_LINE_STA	The slave receives the START of the bus.

No.	Name	Description
2	S_SL_STA	The slave initiates START.
3	S_SL_RCVE_AD	The slave receives the address.
4	S_SL_SEND_W_ACK	The slave ACK writes the address.
5	S_SL_W_DA	The slave receives the write data.
6	S_SL_W_DA_NINE	The slave receives 9 bit of write data.
7	S_SL_R_DA	The slave sends the read data.
8	S_SL_SEND_R_ACK	The slave ACK reads the address.
9	S_SL_R_DA_NINE	The slave sends the 9 bit of read data.
10	S_SL_SEND_AD	The slave sends the addresses.
11	S_SL_RCVE_ACK	The slave receives the addresses.
12	S_SL_CCC_W_DA	The slave receives the CCC data.
13	S_SL_DAA_READ	The slave sends 8 bit of the ENTDAA.
14	S_SL_DAA_WRITE	The slave receives 8 bit of the ENTDAA.
15	S_SL_DAA_ACK	The dynamic address of the slave is ACK.

5.3 Basic Operation Flow for I3C SDR Master Mounting Its Slave

Gowin I3C SDR IP supports the I3C SDR operation. The following sections describe the process by which it is possible to assign the dynamic address, read-write operation, IBI, and hot-join interrupt operation of I3C SDR.

5.3.1 I3C SDR IP Enters into Master State

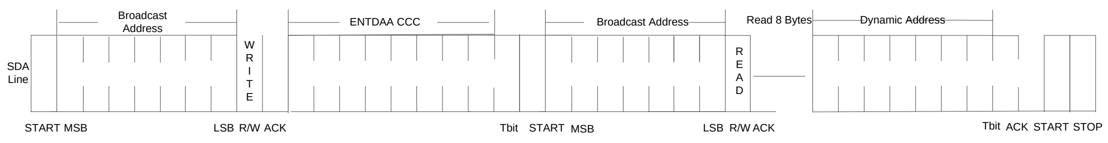
If the I3C SDR bus is available, the CMS can be set to send the device into the Current Master state.

5.3.2 I3C SDR Master Assigns Dynamic Address to I3C SDR Slave

After the device is set as the master, it can communicate with the slave. The dynamic address must be allocated first. The dynamic address allocation process includes the SETDASA and the ENTDAA modes.

SETDASA Mode

Figure 5-3 SETDASA Mode

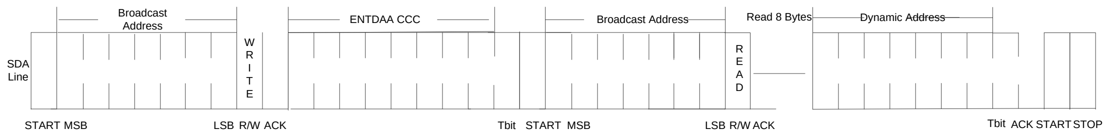


1. Set the STAS signal of the master. The master initiates START.
2. Set the DI value inputted from the master: The broadcast address (7 'h7E), and the Write operation (1' b0). Await slave ACK.

3. Set the AAS signal of the slave. ACK can be sent automatically when the slave receives its own address.
4. Set the DI value input via the master. Send SETDASA CCC (8 'h87).
5. Set the STAS signal of the master. The master initiates START.
6. Set the DI value inputted from the master, the static address (7 bit), and the write operation (1' b0) of the slave. Await slave ACK.
7. Set the AAS signal of the slave. ACK can be sent automatically when the slave receives its own address.
8. Set the DI value inputted from the master, the dynamic address (7 bit), and the 1' b0 of the slave.
9. Set the STOP or STAS signal of the master. The master initiates the mode and the STOP or START quits the SETDASA.

ENTDAA Mode

Figure 5-4 ENTDAA Mode



1. Set the STAS signal of the master. The master initiates START.
2. Set the DI value inputted from the master, the broadcast address (7 'h7E), and the Write operation (1' b0). Await the slave ACK.
3. Set the AAS signal of the slave. ACK can be sent automatically when the slave receives its own address.
4. Set the DI value inputted from the master. Send ENTDAA CCC (8 'h07).
5. Set the STAS signal from the master. The master initiates START.
6. Set the STAS signal inputted from the master, broadcast address (7 'h7E), and Read operation (1' b1). Await slave ACK.
7. Set the AAS signal of the slave. ACK can be sent automatically when the slave receives its own address.
8. The master reads 8 bytes continuously. The software resets the SIO signals to continually read after reading a byte, and the master sends the dynamic address (7 bit) and 1 'b0 to the slave after the eighth SIO is reset. Await slave ACK.
9. Set the AAS signal of the slave. ACK can be sent automatically when the slave receives its own address.
10. Set the STOS of the master. The master initiates STOP to exit the ENTDAA mode.

5.3.3 I3C SDR Master Writes Data to I3C SDR Slave

Figure 5-5 Write Data to the I3C SDR Slave

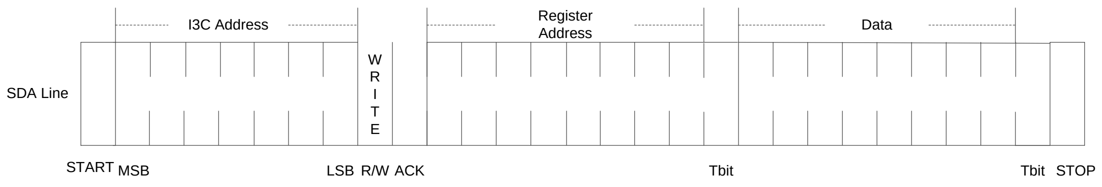
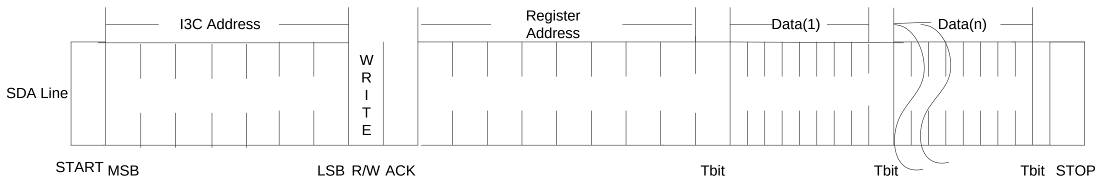


Figure 5-6 Write Multiple Data to the I3C SDR Slave



1. When the I3C bus is available, set the STAS signal of the master. The master initiates START.
2. Set the DI value inputted from the master, the broadcast address (7 'h7E), and the Write operation (1' b0). The master sends the slave address and writes. Await slave ACK.
3. Set the AAS signal of the slave. ACK can be sent automatically when the slave receives its own address.
4. Set the DI value inputted from the master: The register address, and the register address for the master sends the write operation.
5. Set the STAS signal of the master: Data (8 bit), i.e., write the data to the register.
6. The 5th step can be repeated. Write the multiple registers in sequence.
7. Set the STOS signal of the master. The master initiates the STOP.

5.3.4 I3C SDR Master Reads Data from I3C SDR Slave

Figure 5-7 Read Data From I3C SDR Slave

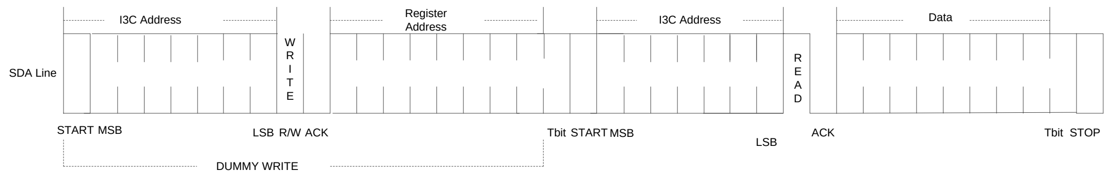
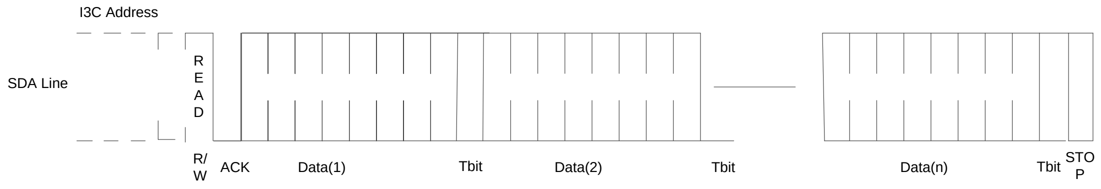


Figure 5-8 Write Multiple Data From I3C SDR Slave



1. Set the STAS signal of the master. The master initiates START.
2. Set the DI value inputted from the master, the broadcast address (7 'h7E), and the Write operation (1' b0). The master sends the slave

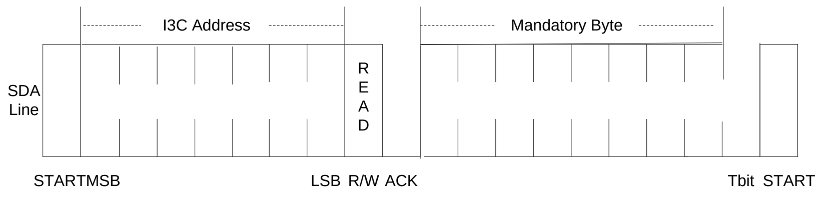
address and writes. Await slave ACK.

3. Set the AAS signal of the slave. ACK can be sent automatically when the slave receives its own address.
4. Set the DI value inputted from the master: the register address. The register address for the master sends the read operation.
5. Set the STAS signal of the master. The master initiates START.
6. Set the DI value inputted from the master: I3C SDR slave address (7 bit), and R operation (1 'b1). The master sends the slave address and reads. Await slave ACK.
7. Set the AAS signal of the slave. ACK can be sent automatically when the slave receives its own address.
8. Set the DI value inputted from the master: Data (8 bit). Determine whether to continue reading via the slave ACS. The master ACS determines whether the master continues to read.
9. The 8th step can be repeated. Write the multiple registers in sequence.

5.3.5 I3C SDR Slave Requires IBI

The I3C protocol supports internal interrupts. In the IDLE state, START permits the slave to initiate and the slave address is subsequently sent. Alternatively, when the master initiates START and the slave detects the START, the master and the slave simultaneously send the address. The smaller address obtains this arbitration.

Figure 5-9 Slave Requests the IBI

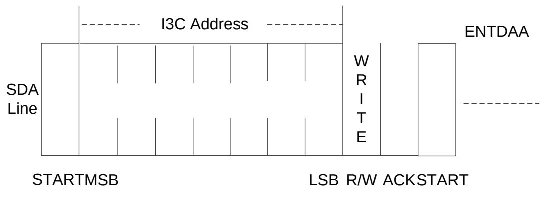


1. Set the STAS signal of the slave. The slave initiates START.
2. Set the DI value of the slave; send the slave address (7 bit) and read operation (1 'b1). Await master ACK.
3. Set the master AAS. The master sends ACK.
4. The master controls the bus to read the data. If the BCR[2] bit is 1, the master reads a byte data when reading.
5. The following operation is consistent with that of the master.

5.3.6 I3C SDR Requires Hot-join

The I3C SDR protocol supports Hot-join, which can be added to the I3C bus when the I3C bus is configured.

Figure 5-10 Slave Requires Hot-join



1. Set the STAS signal of the slave. The slave initiates START.
2. Set the DI value of the slave. Send the slave address (b0000_010) and write operation (1 'b0). Await Master ACK.
3. Set the master AAS. The master sends ACK.
4. The master enters the ENTDAA mode to assign the dynamic address for the slave.
5. The following operation is consistent with that the of the master.

5.4 Basic Operation Flow for I3C SDR Master Mounting I2C Slave

Gowin I3C SDR IP is compatible with the I2C. The read-write operation of I2C is described below.

5.4.1 I3C SDR IP Enters into Master State

If the I3C bus is available, the device can enter into the Current Master state by placing CMS. Set the LGYS and the master communicates with the I2C slave.

5.4.2 I3C SDR Master Writes Data to I2C Slave

Figure 5-11 Write Data to the I2C Slave

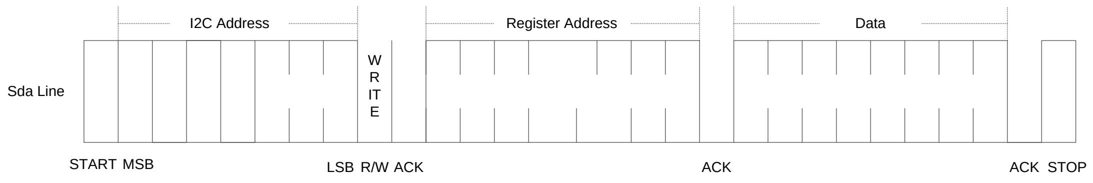
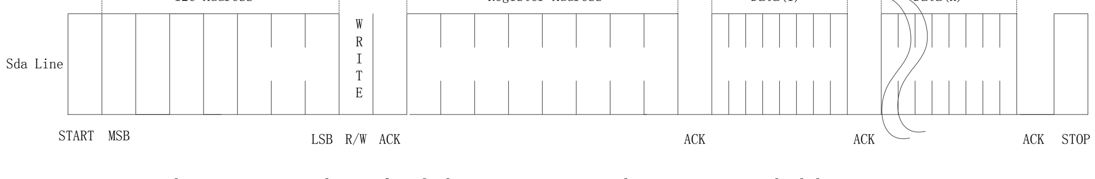


Figure 5-12 Write Multiple Data to the I2C Slave



1. Set the STAS signal of the master. The master initiates START.
2. Set the DI value of the slave. Send the slave address (7 bit) and write operation (1 'b0). Await slave ACK.
3. The slave sends the ACK.
4. Set the DI signal of the master. Send the register address. Await slave ACK.

5. The slave sends the ACK.
6. Set the DI signal of the master and send the data in the register (8 bit). Await slave ACK.
7. The slave sends the ACK.
8. Steps 6 and 7 can be repeated many times; i.e., write the multiple registers in sequence.
9. Set the STOS signal of the master. The master initiates the STOP.

5.4.3 I3C SDR Master Reads Data from I2C Slave

Figure 5-13 Read Data From the I2C Slave

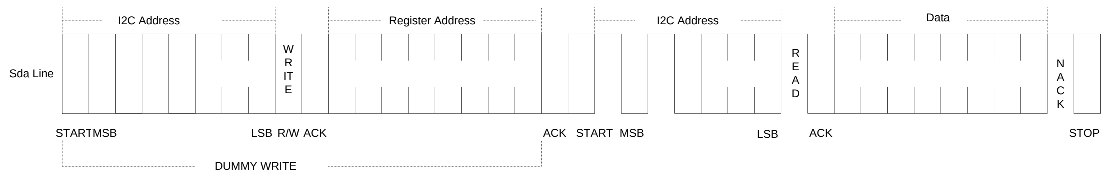
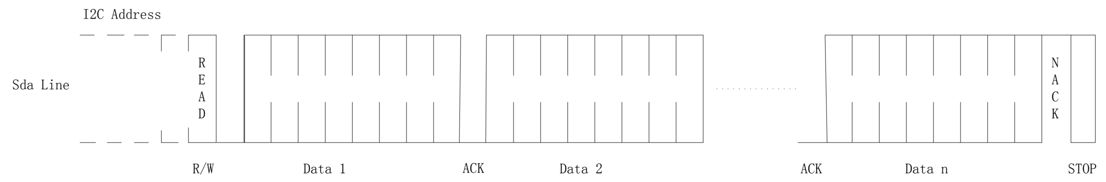


Figure 5-14 Write Multiple Data From the I2C Slave



1. Set the STAS signal of the master. The master initiates START.
2. Set the DI value of the slave and send the I2C address (7 bit) and write operation (1 'b0). Await slave ACK.
3. The slave sends the ACK.
4. Set the DI signal of the master, send the register address, await slave ACK.
5. The slave sends the ACK.
6. Set the STAS signal of the master. The master initiates START.
7. Set the DI value of the slave, send the I2C address (7 bit) and read operation (1 'b1), and await slave ACK.
8. The slave sends the ACK.
9. The slave sends the data (8 bit).
10. The master sends the ACK.
11. Steps 9 and 10 can be repeated many times. Write the multiple registers in sequence.

6 Examples and Explanation

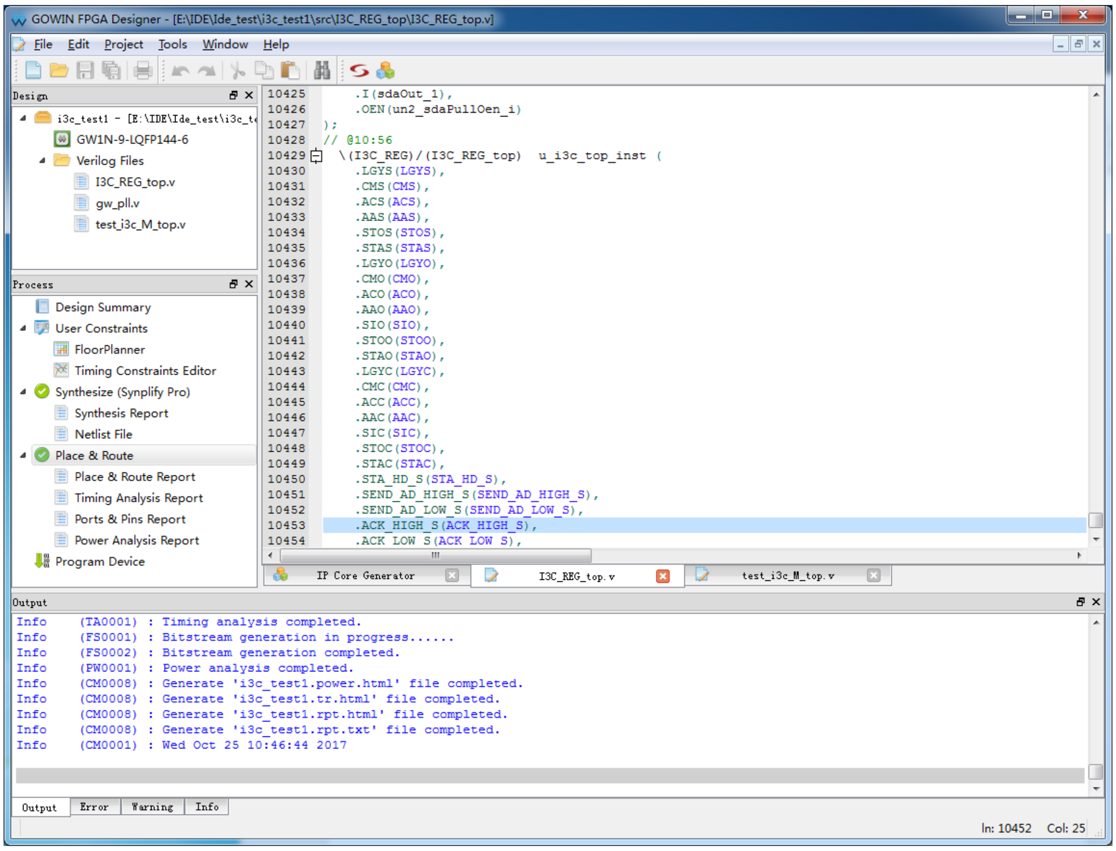
6.1 Open an Existing Project

After starting the Gowin YunYuan Software, Click “File> Open …” to open the “Open File” dialog box. To open an existing project, select the project file（*.gprj), as shown in Figure 6-1.

Note!

There are three ways to open an existing project. For the other two methods, please refer to Gowin YunYuan Software User Guide > 5 Operation > 5.2 Open an Existing Project.

Figure 6-1 Open an Existing Project



```
    .I(sdaOut_1),
    .OEN(un2_sdaPullOen_i)
);
// @10:56
  \(I3C_REG)/(I3C_REG_top)  u_i3c_top_inst (
    .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),
```

```
Info    (TA0001) : Timing analysis completed.
Info    (FS0001) : Bitstream generation in progress......
Info    (FS0002) : Bitstream generation completed.
Info    (PW0001) : Power analysis completed.
Info    (CM0008) : Generate 'i3c_test1.power.html' file completed.
Info    (CM0008) : Generate 'i3c_test1.tr.html' file completed.
Info    (CM0008) : Generate 'i3c_test1.rpt.html' file completed.
Info    (CM0008) : Generate 'i3c_test1.rpt.txt' file completed.
Info    (CM0001) : Wed Oct 25 10:46:44 2017
```

6.2 Gowin I3C SDR IP Calling

Gowin **I3C SDR** IP can be integrated in the Master and Slave, and the calling process is consistent. Users only need to modify the Module Name and File Name, as shown in Figure 6-3, to distinguish between the Master and the Slave. The corresponding operation steps are as follows:

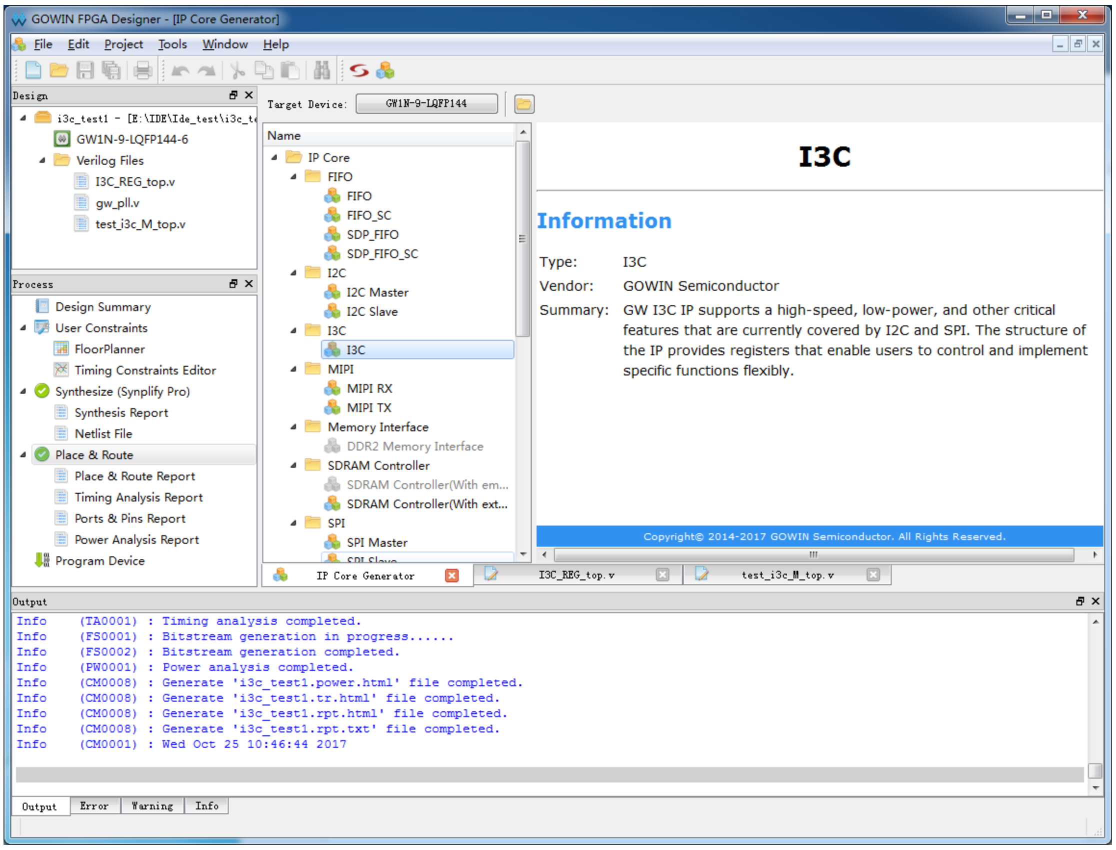
Click “Tool > IP Core Generator” on the menu. The window shown in Figure 6-2 appears.

Select the required devices; for example, GW1N-9-LQFP144. Double-click "I3C SDR" in "Name.” A pop-up window will be displayed, as shown in Figure 6-3. The default options are displayed. Click "OK" to generate the I3C_REG_top Module.

Note!

The communication between the Gowin **I3C SDR** IP and the I2C/SPI/UART Slave is similar and can be realized through the Gowin YunYuan software. However, the example provided below is limited to the **I3C SDR** communication.

Figure 6-2 IP Core Generator Page



I3C

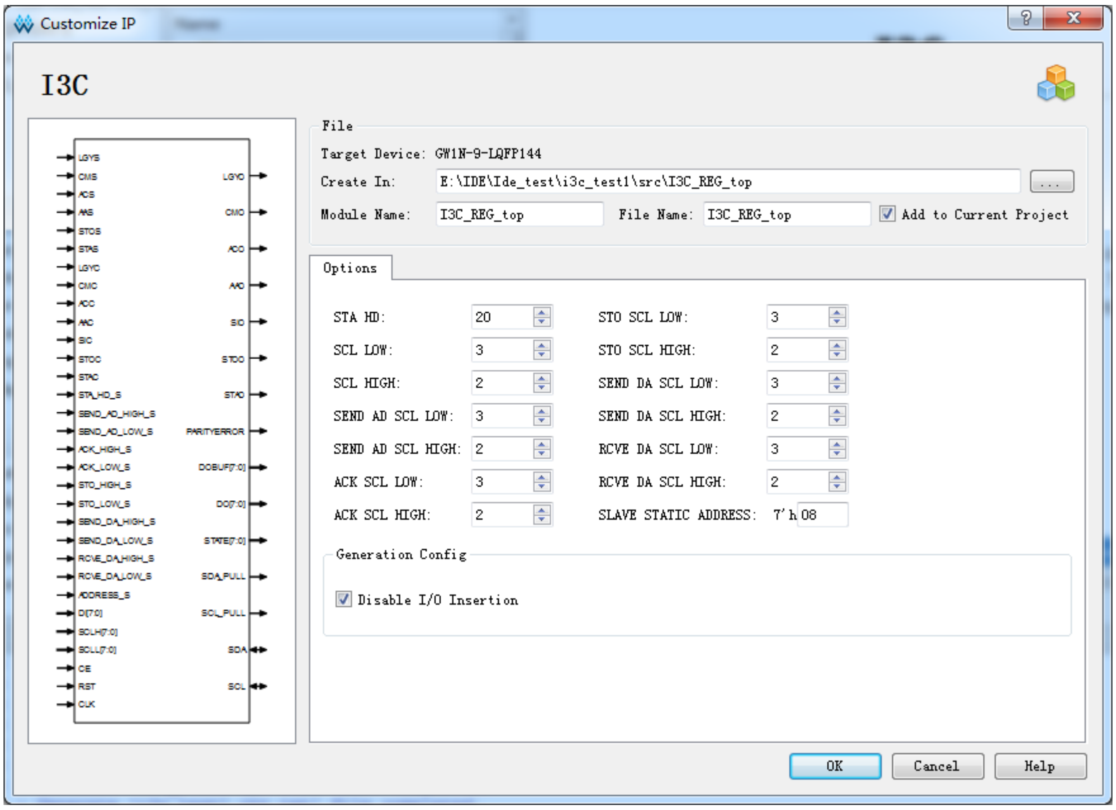
Information

Type:	I3C
Vendor:	GOWIN Semiconductor
Summary:	GW I3C IP supports a high-speed, low-power, and other critical features that are currently covered by I2C and SPI. The structure of the IP provides registers that enable users to control and implement specific functions flexibly.

Copyright© 2014-2017 GOWIN Semiconductor. All Rights Reserved.

```
Info    (TA0001) : Timing analysis completed.
Info    (FS0001) : Bitstream generation in progress......
Info    (FS0002) : Bitstream generation completed.
Info    (PW0001) : Power analysis completed.
Info    (CM0008) : Generate 'i3c_test1.power.html' file completed.
Info    (CM0008) : Generate 'i3c_test1.tr.html' file completed.
Info    (CM0008) : Generate 'i3c_test1.rpt.html' file completed.
Info    (CM0008) : Generate 'i3c_test1.rpt.txt' file completed.
Info    (CM0001) : Wed Oct 25 10:46:44 2017
```

Figure 6-3 Gowin I3C SDR IP Page



File

Target Device: GW1N-9-LQFP144

Create In: E:\IDE\Ide_test\i3c_test1\src\I3C_REG_top

Module Name: I3C_REG_top File Name: I3C_REG_top ☑ Add to Current Project

Options

STA HD:	20	STO SCL LOW:	3
SCL LOW:	3	STO SCL HIGH:	2
SCL HIGH:	2	SEND DA SCL LOW:	3
SEND AD SCL LOW:	3	SEND DA SCL HIGH:	2
SEND AD SCL HIGH:	2	RCVE DA SCL LOW:	3
ACK SCL LOW:	3	RCVE DA SCL HIGH:	2
ACK SCL HIGH:	2	SLAVE STATIC ADDRESS:	7'h 08

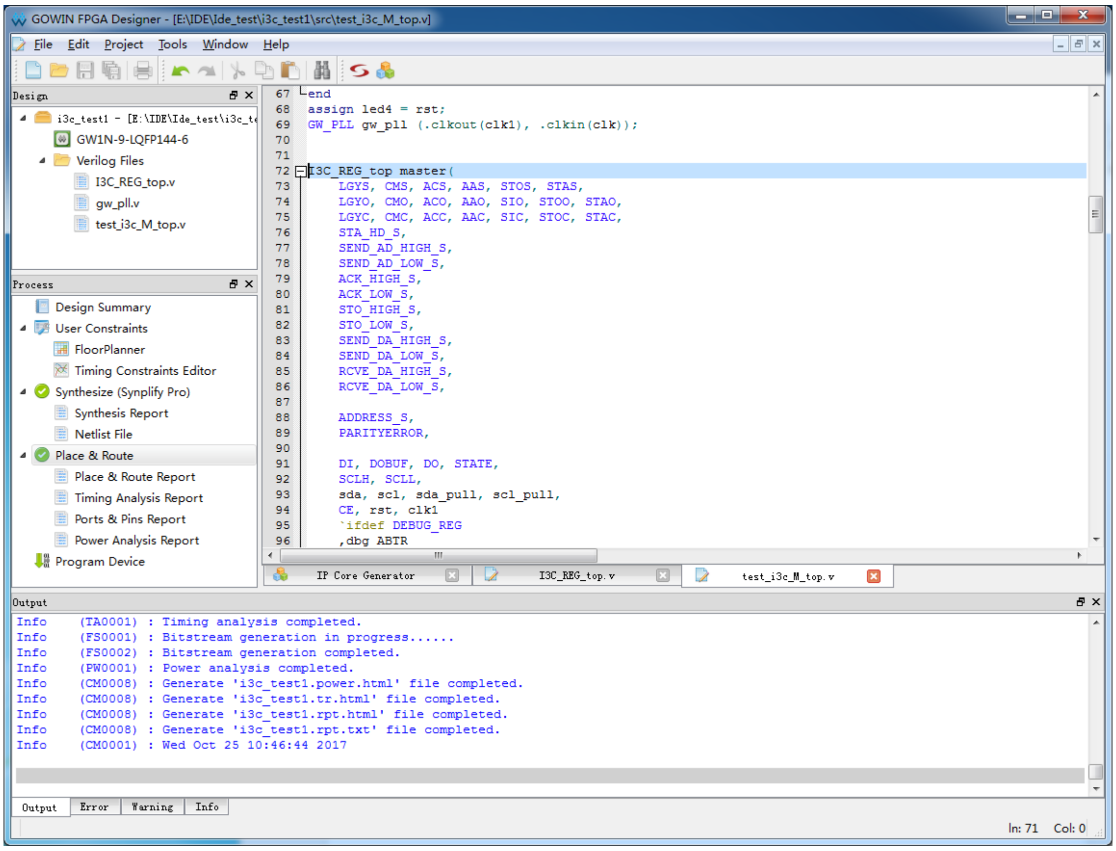
Generation Config

☑ Disable I/O Insertion

6.3 Gowin I3C SDR IP Instantiation

Instantiate I3C_REG_top in the project, as shown in Figure 6-4.

Figure 6-4 Gowin I3C SDR IP Instantiation



```
end
assign led4 = rst;
GW_PLL gw_pll (.clkout(clk1), .clkin(clk));


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

    DI, DOBUF, DO, STATE,
    SCLH, SCLL,
    sda, scl, sda_pull, scl_pull,
    CE, rst, clk1
    `ifdef DEBUG_REG
    ,dbg ABTR
```

```
Info     (TA0001) : Timing analysis completed.
Info     (FS0001) : Bitstream generation in progress......
Info     (FS0002) : Bitstream generation completed.
Info     (PW0001) : Power analysis completed.
Info     (CM0008) : Generate 'i3c_test1.power.html' file completed.
Info     (CM0008) : Generate 'i3c_test1.tr.html' file completed.
Info     (CM0008) : Generate 'i3c_test1.rpt.html' file completed.
Info     (CM0008) : Generate 'i3c_test1.rpt.txt' file completed.
Info     (CM0001) : Wed Oct 25 10:46:44 2017
```

6.4 Gowin I3C SDR IP Constraint

The SDA\SDA_PULL, SCL\SCL_PULL of Gowin **I3C SDR** IP needs to be constrained to the IOB of GW1N9K.

6.5 Bitstream File Generation

After adding the necessary constraints, begin to synthesize, P&R, and then generate the bitstream files. Load the bitstream files to the development board or the test board using the Gowin download cable. You can also observe the communication conditions between the master and the slave via the test interface.

7 Timing

7.1 Timing

I3C SDR IP timing includes the read timing, write timing, dynamic address allocation timing, IBI timing, Hot-join timing of the I3C SDR Master interface, and I3C SDR Slave interface.

Figure 7-1 Write Data Timing

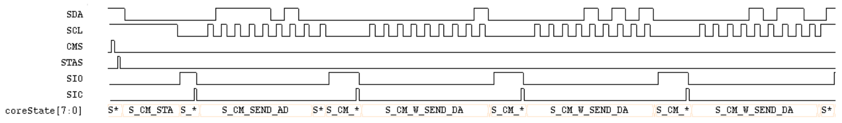

SDA
SCL
CMS
STAS
SIO
SIC
coreState[7:0]
S* S_CM_STA S_* S_CM_SEND_AD S* S_CM_* S_CM_W_SEND_DA S_CM_* S_CM_W_SEND_DA S_CM_* S_CM_W_SEND_DA S*

Figure 7-2 Read Data Timing

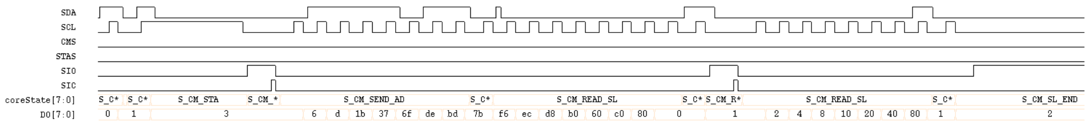

SDA
SCL
CMS
STAS
SIO
SIC
coreState[7:0]
D0[7:0]
S_C* S_C* S_CM_STA S_CM_* S_CM_SEND_AD S_C* S_CM_READ_SL S_C* S_CM_R* S_CM_READ_SL S_C* S_CM_SL_END
0 1 3 6 d 1b 37 6f de bd 7b f6 ec d8 b0 60 c0 80 0 1 2 4 8 10 20 40 80 1 2

Figure 7-3 SETDASA Timing

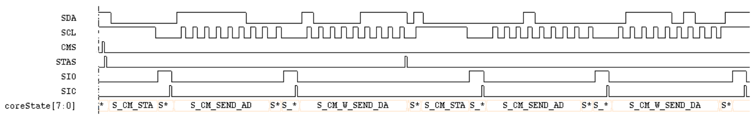

SDA
SCL
CMS
STAS
SIO
SIC
coreState[7:0]
S_CM_STA S* S_CM_SEND_AD S* S_* S_CM_W_SEND_DA S* S_CM_STA S_* S_CM_SEND_AD S* S_* S_CM_W_SEND_DA S*

Figure 7-4 ENTDAA Timing

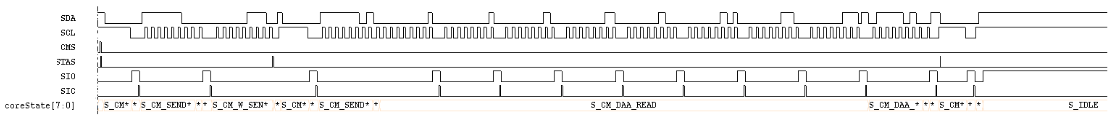

SDA
SCL
CMS
STAS
SIO
SIC
coreState[7:0]
S_CM* S_CM_SEND* S_CM_W_SEN* S_CM* S_CM_SEND* S_CM_DAA_READ S_CM_DAA_* S_CM* S_IDLE

Figure 7-5 IBI Timing

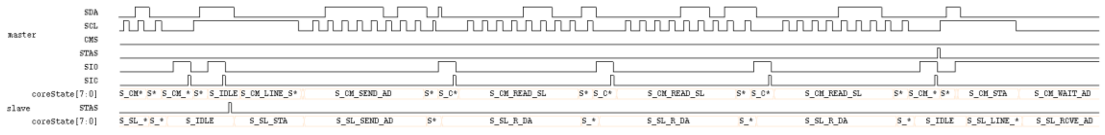
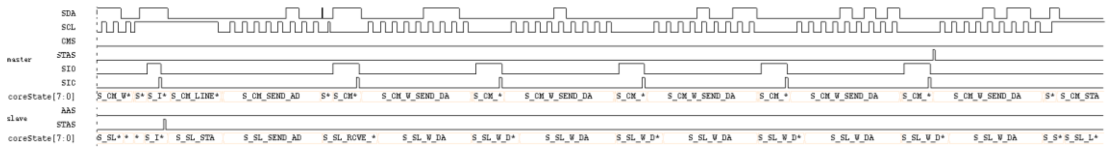


Figure 7-6 Hot-join Timing



8Reference Designs

8.1 Reference Designs

The reference design functionality is mainly used to verify whether the communication runs well between the I3C SDR Master and the I3C SDR Slave. Its main features include the master write (send Slave address and write operation bit 8 'haa, send register address 8' h55, and then write 128 data from 8 'h56 to Slave);

8.2 Top-Level Design of Master

The top-level design of the master top.v is shown in Appendix 9.1.

1. Top. V defines the number of pins, some of which are defined but not in use, and the purpose is to observe the signals that can be pulled to these reserved pins.
2. Top. V is mainly used for the clock division, jitter elimination, and write data.
   a). counter: submodule, used for clock division. Firstly, 50 MHz clock frequency is divided into 10 MHz via PLL, and then the clock frequency is divided into 10 KHz via the submodule "counter."
   b). deUstb: submodule, used for button jitter elimination.
   c). Write data module: The I3C SDR master outputs the pin STATE and the corresponding I3C SDR status is output. According to the state, the corresponding features are completed. If the STATE is S_CM_WAIT_AD, the I3C SDR master is in a wait to address state, and the master controller can send the address. If the STATE is S_CM_WAIT_W_DA, and the I3C SDR Master is in a wait to write data state, the master controller can write the data.
3. Introduction on top. V code.
   a). Clock division after PLL processing and counter. The 50 MHz clock frequency is divided into 10 KHz.

```
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;
```

b). Instantiate the **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)
);
```

c). Key jitter elimination

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

d). Write data

```
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;// send the slave address and write bit
                         SIC<=1; // set SIC, delete SIO
                    end
                    S_CM_WAIT_NACK:begin
                         STOS <=1;// set STOS
                         SIC<=1;// set SIC, delete SIO
                    end
                    S_CM_WAIT_W_DA:begin
                         if(&dataCnt)begin// write data end, set STOS
                              STOS <=1; // write data end, set STOS
                              dataCnt <= 0;
                         end
                         else begin
                              DI < = dataTobeSend; // send the register address.
                              DataTobeSend < = dataTobeSend + 1; // send data in turn.
                              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
```

e). Pull out the observed signal.

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

8.3 Top-Level Design of Slave

The top-level design of the slave, top.v (see Appendix 9.2), is similar to the top-level design of the master. However, the data processing module is different.

```
always @(posedge CLK)begin
    if(RST)begin
        slaveState <=0;
    end
    else begin
    if(SIO)begin
        case (STATE)
            S_SL_W_DA:begin//slave receiving 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 Board Level Connection

After inputting the necessary constraints, generate bitstream files via synthesizing and P&R. Load the bitstream files to the development board or the test board using the Gowin download cable. You can also observe the communication conditions via the test interface. This reference design implements communication between the I3C SDR Master and its Slave using double boards; i.e., set the **I3C SDR** Master on a GW1N-9K board and set the I3C SDR Slave on the other GW1N-9K board. Three lines are required to connect the boards: one for SCL, one for SDA, and one for GND. The constraint file can be referenced in the following format:

```
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 Result Observation

This reference design uses an oscilloscope to observe whether the SCL and the SDA are correct. In addition, if you want to observe the Bus signals, such as STATE, DI and DOBUF, the logic analyzer is recommended.

9 Appendix

9.1 Top-level Design of the Master

The codes at the top of the Master design are as follows:

```
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 responds to slave-initiated 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 receives 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 receiving write data
localparam    S_SL_W_DA_NINE          = 8'h24;//-----state slave receiving 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 8 bits
localparam    S_SL_DAA_WRITE           = 8'h2c;//-----state slave receiving 8 bits
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;

end module
```

9.2 Top-Level Design of Slave

The codes at the top-Level design of Slave are as follows:

```
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 receiving 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;
end module
```

PROGRAMMING FOR FUTURE