Application Report

SNLA267A–March 2019–Revised June 2019

How to Design a FPD-Link III System Using DS90UB953-Q1 and DS90UB954-Q1

Cole Macias and Mandeep Singh

ABSTRACT

FPD-Link III devices such as the DS90UB95x-Q1 can support cameras in Advanced Driver Assist Systems (ADAS) in the automotive industry. There are complications that can occur in system implementation due to the complexity of the technology, so it is important to ensure proper design. The purpose of this document is to systematically design a SERDES system using the DS90UB953-Q1 and DS90UB954-Q1.

Contents

1 Overview 3
1.1 System Level Functionality 3
2 Basic Design Rules 4
2.1 IDX and MODE Pin Verification 4
2.2 Successful I^{2}C Communication With 953 and 954 10
2.3 I2C Passthrough Verification 12
2.4 Basic Diagnostic and Error Registers 12
3 Designing the Link Between SER and DES 14
3.1 Back Channel Configuration 15
3.2 BIST 16
3.3 AEQ 20
3.4 CML Out 21
4 Designing Link Between SER and Image Sensor 21
4.1 Sensor Initialization Using SER GPIOs 21
4.2 CLKOUT 23
5 Designing Link Between DES and ISP 24
5.1 Frame Sync 24
5.2 Port Forwarding 29
5.3 Pattern Generation 29
6 Hardware Design 31
6.1 Basic I2C Connectors 31
6.2 AC Capacitor on FPD3 Link 32
6.3 Capacitance Used in Loop Filter 32
6.4 Critical Signal Routing 32
6.5 Time Domain Reflection 33
6.6 Return Loss and Insertion Loss 33
6.7 Power-over-Coax (PoC) 33
6.8 Voltage and Temperature Sensing 34
7 Appendix 34
7.1 Scripts 34
7.2 Acknowledgments 49

List of Figures

1 Simplified Block Diagram of DS90UB953/DS90UB954-Q1 System 3
2 Basic Design Rules Flowchart 4

Copyright © 2019, Texas Instruments Incorporated

3 Clocking System Diagram 6
4 Illustration of Synchronous Clocking 7
5 Illustration of Non-Synchronous Mode With External Oscillator 8
6 Illustration of Non-Synchronous Mode With Always on Clock (AON) 8
7 Illustration of Two Ports 11
8 Flowchart for SER and DES Link 15
9 BIST Script Flowchart 17
10 Image Sensor and SER Link Design 21
11 Flowchart for Initializing the Image Sensor Using GPIO’s 22
12 Clocking System Diagram 23
13 DES and ISP Link Flowchart 24
14 Steps for Controlling SER GPIOs Remotely and Locally 24
15 Block Diagram of Controlling SER GPIOs Over BC 25
16 Block Diagram of Controlling SER GPIOs Remotely 26
17 Block Diagram of Internally Generated Frame Sync 27
18 Block Diagram of Externally Generated Frame Sync 28
19 Hardware Design Flowchart 31
20 Simplified Block Diagram of DS90UB953-Q1/DS90UB954-Q1 System 32

List of Tables

1 Serial Control Bus Addresses for IDX on the 953 5
2 Serial Control Bus Addresses for IDX on the 954 5
3 DS90UB953-Q1 Strap Configuration Mode Select 6
4 DS90UB954-Q1 Strap Configuration Mode Select 6
5 Mode Clock Calculation Table 9
6 Mode Clock Settings With Descriptions of fo and f1 9
7 Bit Description of SER_ALIAS_ID Register With Example 11
8 Bit Description of FPD3_PORT_SEL Register 0x4C 12
9 Communication of I2C Devices Using Various Passthrough Settings 12
10 DS90UB954-Q1 Registers Used for Diagnostics and Checking Errors 13
11 DS90UB953-Q1 Registers Used for Diagnostics and Checking Errors 14
12 Settings for Bidirectional Configuration (BCC_CONFIG) Register 0x58 on 954 16
13 RESET_CTL Register Description on 953 17
14 DEVICE_STS Register Description on 954 18
15 GENERAL_STATUS Register Description for Lock on 953 18
16 DS90UB954-Q1 Registers Used in BIST Script 19
17 DS90UB953-Q1 Registers Used in BIST Script 20
18 Channel Monitor Loop-Through Output Configuration of the 954 21
19 Bit Description of GPIO_INPUT_CTRL Register 0x0E 22
20 Bit Description of LOCAL_GPIO_DATA Register 0x0D 22
21 Example Using GPIO_INPUT_CTRL Register 0x0E for Local SER GPIO Control 25
22 Example Using LOCAL_GPIO_DATA Register 0x0D for Local SER GPIO Control 25
23 Example Using LOCAL_GPIO_DATA Register 0x0D for Local SER GPIO Control 26
24 BC_GPIO_CTL0 Registers From the 954 Data Sheet 26
25 Example using BC_GPIO_CTL0 Register 0x6E for Remote SER GPIO Control 27
26 Registers Used When Configuring GPIOS and Frame Sync on the 953 and 954 28

Copyright © 2019, Texas Instruments Incorporated

Trademarks

All trademarks are the property of their respective owners.

1 Overview

1.1 System Level Functionality

The DS90UB954-Q1 FPD-Link III deserializer, in conjunction with an ADAS FPD-Link III serializer, supports the video transport needs with an ultra-high-speed forward channel and an embedded bidirectional control channel. After the DS90UB954-Q1 receives the data, the device outputs the data from a configurable MIPI CSI-2 port. The CSI-2 port may be configured as either a single CSI-2 output with four lanes up to 1.662 Gbps per lane, or as two 2-lane CSI-2 outputs that can send replicated data on both ports. The engineer can use a second differential clock for the second replicated output when the device is configured for dual CSI-2 outputs that can support one clock lane and one or two data lanes each. The DS90UB954-Q1 can support multiple data formats and different resolutions as provided by the sensor. Conversion between different data formats is not supported. The CSI-2 Tx module accommodates both image data and non-image data (including synchronization or embedded data packets).

The DS90UB953-Q1 serializes data from high-resolution image sensors or other sensors using the MIPI CSI-2 interface. The DS90UB953-Q1 serializer is optimized to interface with the DS90UB954-Q1 deserializer (dual hub) or the DS90UB960-Q1 deserializer (quad hub) as well as potential future deserializers. The interconnect between the serializer and the deserializer can be either a coaxial or shielded twisted-pair (STP) cable. The DS90UB953-Q1 was designed to support multi-sensor systems such as surround view, and as such has the ability to synchronize sensors through the DS90UB954-Q1 and DS90UB960-Q1 hub.

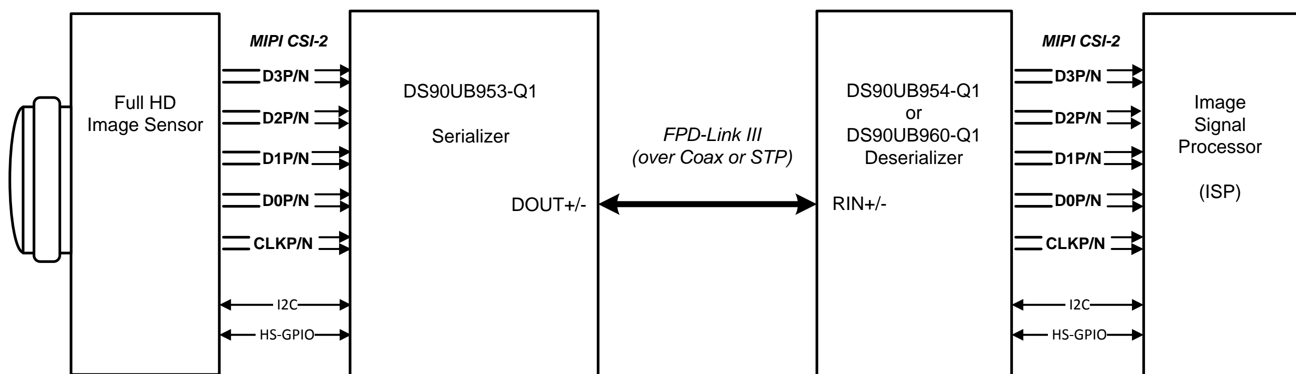

Figure 1. Simplified Block Diagram of DS90UB953/DS90UB954-Q1 System

Due to the complexity of this system, there are a variety of issues that can be avoided by proper design. The purpose of this guide is the help the user systematically design a system using the 953 and 954. The following chapters show the typical concepts and tests and how to implement them. This guide lists basic design rules as a checklist that should be followed to ensure proper initialization of the system. The items described in the suggested checklist are the building blocks of the system. Without them, the system will not work as expected.

Following the completion of the basic design rules section, this guide provides analysis of the three important links in the ADAS System using DS90UB953-Q1 and DS90UB954-Q1:

1. The link between the serializer (SER) and deserializer (DES)
2. The link between the serializer and imager sensor
3. The link between the deserializer and Image Signal Processor (ISP)

Important procedures and design considerations are highlighted and explained in the form of script examples, general steps, and flow charts.

Copyright © 2019, Texas Instruments Incorporated

At the end of this guide is a set of hardware design topics that are commonly discussed when designing a SERDES system.

2 Basic Design Rules

This chapter describes a list of key items that are fundamental when powering on the system. These items define the functionality of the entire system. To get the system up and running quickly, TI recommends to check these items before moving forward.

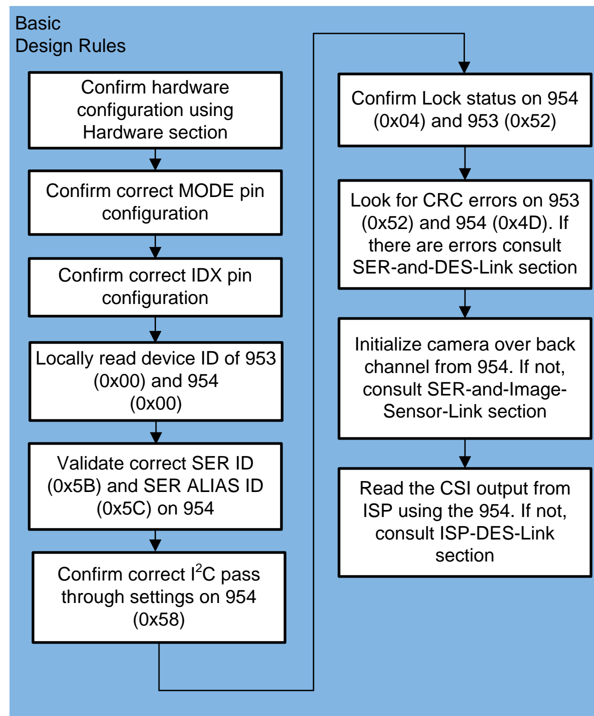

Figure 2. Basic Design Rules Flowchart

2.1 IDX and MODE Pin Verification

Each IDX and Mode pin contains a voltage divider to the respective IDX and Mode pins on the 953 and 954. The IDX and Mode pins read the voltage on the pin, and the internal comparators decide which IDX or Mode is assigned to each device. As a result, the required voltage supply and the ratio of the resistor divider are used to set the IDX and Mode pins.

1. **Ensure commands refer to correct I²C addresses by checking the IDX pin**.
   a. The IDX pin configures the control interface to one of many possible device addresses used in I2C communication. Usually for 1.8-V or 3.3-V referenced I2C I/O voltage, a pullup resistor and a pulldown resistor is used to set the appropriate voltage on the IDX input pin of both devices.
   b. The IDX resistor divider must be referred to Pin #25 on the 953 and Pin #35 on the 954. Tables that hold appropriate resistor values for setting IDX are shown below.
   c. For example the 953 can have an open pullup resistor, a 40.2-kΩ pulldown resistor, and an I2C supply of 1.8 V to achieve a device ID of 0x30. This is shown in Table 1.

Copyright © 2019, Texas Instruments Incorporated

Table 1. Serial Control Bus Addresses for IDX on the 953

IDX	V_{TARGET} VOLTAGE RANGE			V_{IDX} TARGET VOLTAGE	SUGGESTED STRAP RESISTORS (1% TOL)		I2C 8-BIT ADDRESS	I2C 7-BIT ADDRESS	I2C I/O VOLTAGE
	RATIO MIN	RATIO TYP	RATIO MAX	V_{VDD} = 1.8 V	R_{HIGH} (kΩ)	R_{LOW} (kΩ)			
1	0.000	0.00	0.131	0.000	Open	40.2	0x30	0x18	1.8 V
2	0.178	0.214	0.256	0.385	180	47.5	0x32	0x19	1.8 V
3	0.537	0.564	0.591	1.015	82.5	102	0x30	0x18	3.3 V
4	0.652	0.679	0.706	1.223	68.1	137	0x32	0x19	3.3 V

Table 2. Serial Control Bus Addresses for IDX on the 954

NO.	V_{IDX} VOLTAGE RANGE			V_{IDX} TARGET VOLTAGE	SUGGESTED STRAP RESISTORS (1% TOL)		PRIMARY ASSIGNED I²C ADDRESS	
	V_{MIN}	V_{TYP}	V_{MAX}	(V); VDD1P8 = 1.8 V	R_{HIGH} (KΩ)	R_{LOW} (KΩ)	7-BIT	8-BIT
0	0	0	0.131 × $V_{(VDD18)}$	0	OPEN	10.0	0x30	0x30
1	0.179 × $V_{(VDD18)}$	0.213 × $V_{(VDD18)}$	0.247 × $V_{(VDD18)}$	0.374	88.7	23.2	0x32	0x32
2	0.296 × $V_{(VDD18)}$	0.330 × $V_{(VDD18)}$	0.362 × $V_{(VDD18)}$	0.582	75.0	35.7	0x34	0x34
3	0.412 × $V_{(VDD18)}$	0.443 × $V_{(VDD18)}$	0.474 × $V_{(VDD18)}$	0.792	71.5	56.2	0x36	0x36
4	0.525 × $V_{(VDD18)}$	0.559 × $V_{(VDD18)}$	0.592 × $V_{(VDD18)}$	0.995	78.7	97.6	0x38	0x38
5	0.642 × $V_{(VDD18)}$	0.673 × $V_{(VDD18)}$	0.704 × $V_{(VDD18)}$	1.202	39.2	78.7	0x3A	0x3A
6	0.761 × $V_{(VDD18)}$	0.792 × $V_{(VDD18)}$	0.823 × $V_{(VDD18)}$	1.420	25.5	95.3	0x3C	0x3C
7	0.876 × $V_{(VDD18)}$	$V_{(VDD18)}$	$V_{(VDD18)}$	1.8	10.0	OPEN	0x3D	0x3D

2. **Ensure devices are in correct mode by checking the MODE**.
   a. As shown in Table 3, the DS90UB953-Q1 can operate in one of many different modes that define the clocking the 953. The default mode is selected by the bias voltage applied to the MODE pin (21) during power up. To set this voltage, a potential divider between VDD and GND is used to apply the appropriate bias. TI recommends that this potential divider should be referenced to the potential on the VDDD pin (25). After power up, the MODE can be read or changed through register access. On the 953, register 0x03 controls MODE_SEL.
   b. As shown in Table 4, the DS90UB954-Q1 can operate in many different modes that define the expected imager data format. Mode is defined on power up through a voltage divider to the Mode pin (37). While the 954 can be placed in different modes, the only compatible mode with the 953 is the CSI-2 Mode. After power up, the mode can be controlled by the first 2 bits of the PORT_CONFIG register with address of 0x6D.
   c. The most common deserializer mode configuration for a 954 and 953 system is to use a CSI-2 port and a coaxial cable between the devices. As a result, a pullup resistor of 78.7 kΩ, a pulldown resistor of 97.6 kΩ, and 1.8 V for VDD are used.

Copyright © 2019, Texas Instruments Incorporated

Table 3. DS90UB953-Q1 Strap Configuration Mode Select

MODE NO.	V_{TARGET} VOLTAGE RANGE			V_{TARGET} STRAP VOLTAGE	SUGGESTED STRAP RESISTORS (1% TOL)		DESCRIPTION
	RATIO MIN	RATIO TYP	RATIO MAX	(V); $V_{(VDD)}$ = 1.8 V	R_{HIGH} (kΩ)	R_{LOW} (kΩ)	
1	0.000	0.000	0.133	0.000	OPEN	10	CSI-2 Synchronous mode – FPD-Link III Clock reference derived from deserializer 2
2	0.288 × $V_{(VDD)}$	0.325 × $V_{(VDD)}$	0.367 × $V_{(VDD)}$	0.586	75	35.7	CSI-2 Non-synchronous CLK_IN – FPD-Link III Clock reference derived from external clock reference input CLK_IN pin
3	0.412 × $V_{(VDD)}$	0.443 × $V_{(VDD)}$	0.474 × $V_{(VDD)}$	0.792	71.5	56.2	CSI-2 Non-synchronous AON – FPD-Link III Clock reference derived from internal AON clock.

Table 4. DS90UB954-Q1 Strap Configuration Mode Select

MODE NO.	V_{TARGET} VOLTAGE RANGE			V_{TARGET} STRAP VOLTAGE	SUGGESTED STRAP RESISTORS (1% TOL)		RX MODE
	V_{MIN}	V_{TYP}TARGET	V_{MAX}	(V); VDD1P8 = 1.8 V	R_{HIGH} (kΩ)	R_{LOW} (kΩ)	
0	0	0	0.131 × VDD18	0	OPEN	10	CSI
1	0.525 × VDD18	0.559 × VDD18	0.592 × VDD18	0.995	78.7	97.6	CSI

2.1.1 REF Clock, CLK IN, AON and Frequency Selection

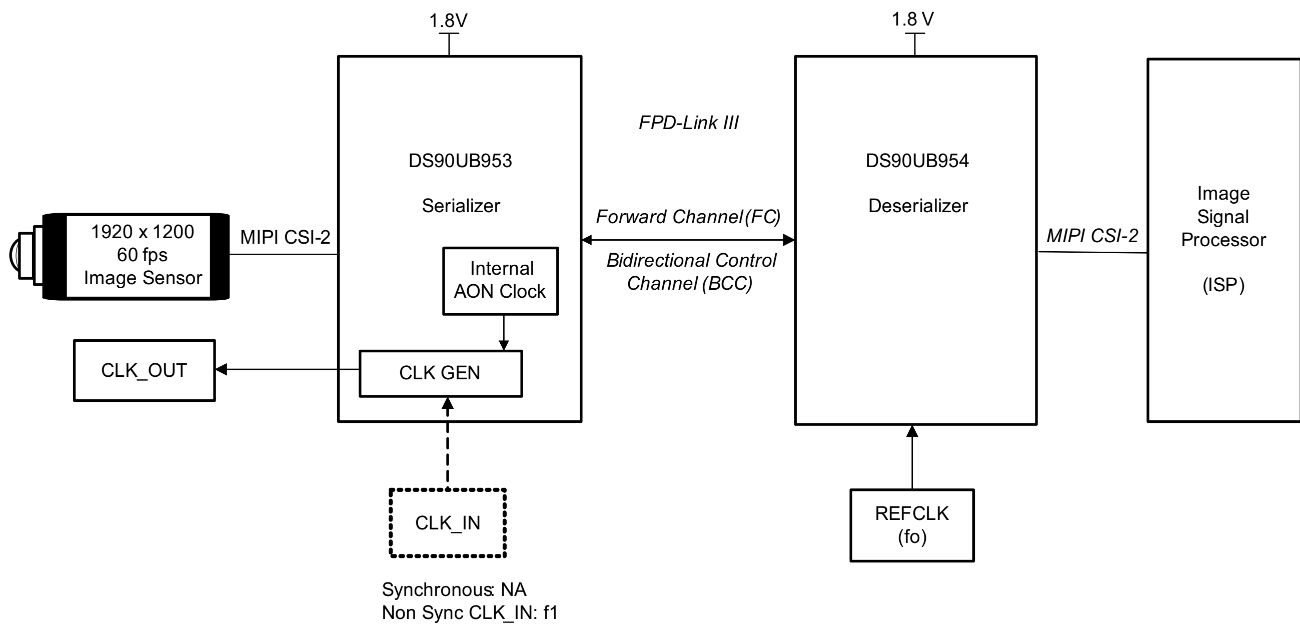

Figure 3. Clocking System Diagram

Copyright © 2019, Texas Instruments Incorporated

The DS90UB953-Q1 supports two different clocking schemes controlled by the MODE pin and MODE_SEL register, 0x03. They are the synchronous and non-synchronous CLK_IN.

2.1.1.1 Synchronous Mode

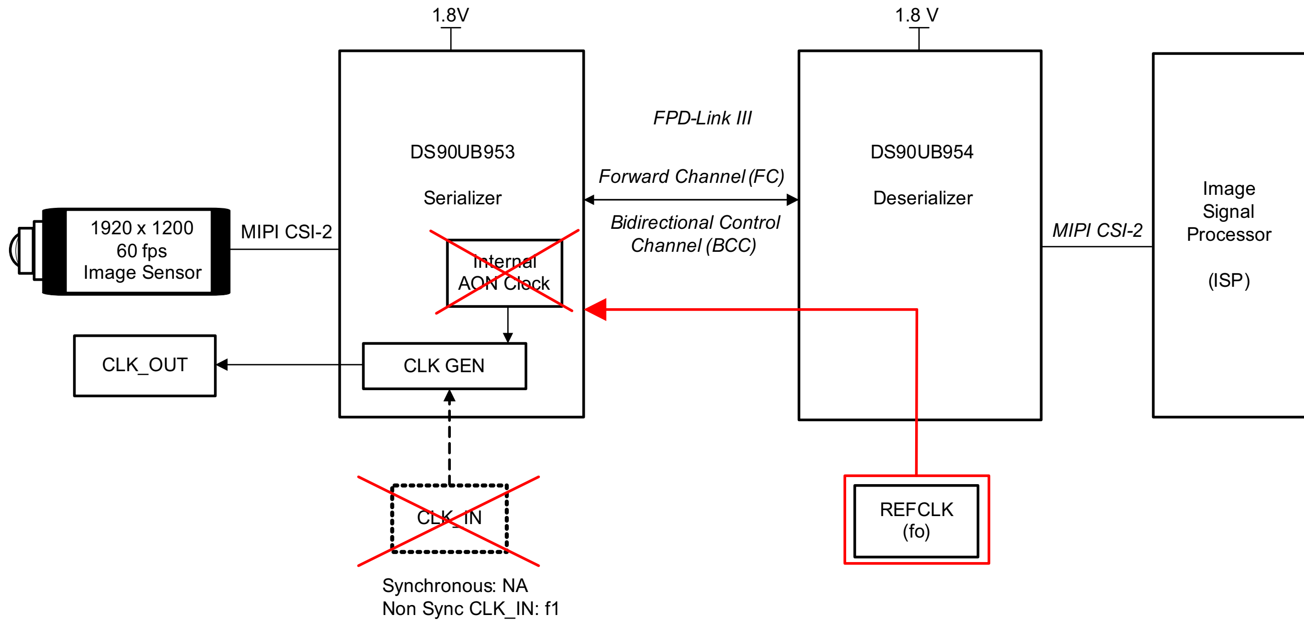

Figure 4. Illustration of Synchronous Clocking

The first mode, synchronous, is an internally generated clock. This clock reference is extracted from back channel on the bidirectional communications link, and the internal PLLs take the extracted signal to generate the required clocks. This allows multiple cameras within a system to operate in the same clock domain and allows more space on the PCB. As shown in Figure 4, REFCLK (f0) is input signal on the deserializer that is required for precise frequency operation. Refer to the REFCLK section in the 954 data sheet for more information. The RECLK specifications for the 953 states that REFCLK must range from 24 to 26 MHz, and this signal sends *two bits* across the back channel: high and low. As a result, multiply the REFCLK frequency by two when calculating the BC and FC rate. This is summarized in Table 5.

Copyright © 2019, Texas Instruments Incorporated

2.1.1.2 Non-Synchronous CLK_IN Mode

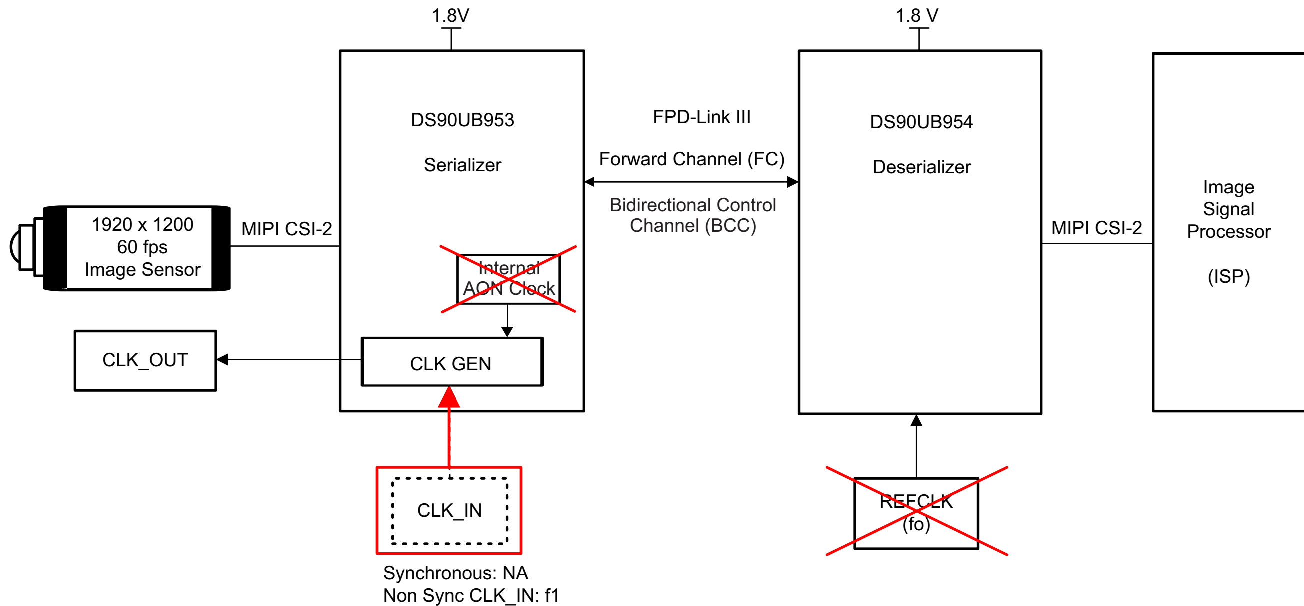

Figure 5. Illustration of Non-Synchronous Mode With External Oscillator

The second mode, non-synchronous CLK_IN, uses an external oscillator as a reference and generates the required clock for the FPD forward channel for that reference. Referring to Figure 5, the external clock must be fed into the CLK_IN pin (20) on the 953, running at a constant rate (f1) proportional to the REFCLK (f0), and a BC rate is then programmed to be less than or equal to 10 Mbps. Register 0x05, GENERAL_CFG, on the 953 holds parameters for the PLL clock control. Bits [6:4] of this register control the CLKIN divider. The CLKIN Divider can be divided by 1 or 2 by assigning these bits to 0b000 or 0b001, respectively. This division must be accounted for when calculating the FC rate which is shown in Table 5.

2.1.1.3 Non-Synchronous AON Mode

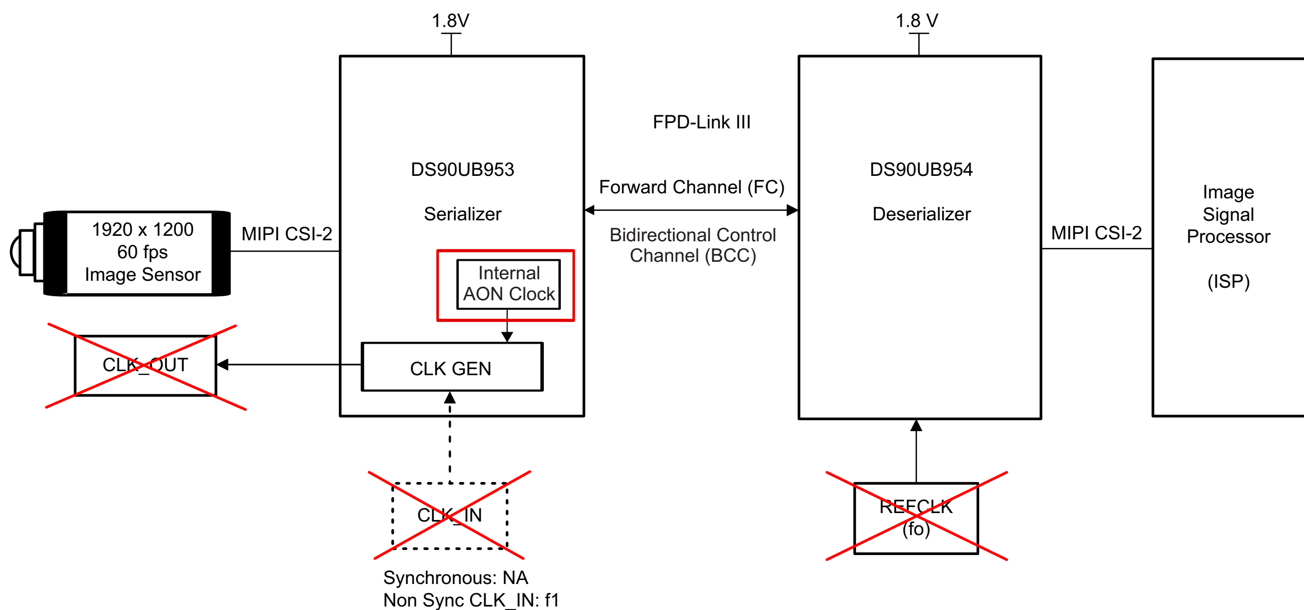

Figure 6. Illustration of Non-Synchronous Mode With Always on Clock (AON)

Copyright © 2019, Texas Instruments Incorporated

The third mode, non-synchronous internal clocking mode, the serializer uses the internal Always on Clock (AON) as the reference clock for the forward channel. The OSCCLK_SEL select must be asserted 0x05[3]=1 to enable maximum data rate when using internal clock mode, and the CLK_OUT function is disabled. A separate reference is provided to the image sensor or ISP. When in CSI-2 mode, the CSI-2 interface may be synchronous to this clock. The CSI-2 rate must be lower than the line rate. For example, with the internal clock of 24.2 MHz the FPD-Link III forward channel rate is 3.872 Gbps, the CSI-2 throughput must be ≤ 3.1 Gbps (See Table 5).

2.1.1.4 CSI Throughput

When calculating CSI throughput, it is important to account for the extra 8 bits in the 40-bit forward channel payload that are used for redundancy, parity, check sum, DC balancing, embedded clock, control, and error checking. A normal CSI payload is 32 bits, and adding these 8 bits will reduce the actual amount of data that is transferred. This is why 4.16 Gbps of data transfer is actually a maximum of 3.32 Gbps of CSI throughput.

Because there are four data lanes available, the maximum value of CSI throughput per lane is calculated as 832 Mbps after dividing the maximum possible CSI throughput by 4. Even if two lanes in use, the maximum throughput is still divided 4. As a result, the maximum CSI throughput is 832 Mbps, regardless of how many lanes are used. Note that this is a limitation of the serializer and not the deserializer. This information is found in Table 5.

2.1.1.5 Clocking and Frequency Selection Example

Using the correct resistor divider values at the MODE pin, the device will power up in synchronous mode. REFCLK (f0) is 25 MHz, which is the recommended value. Use Table 5 for reference. To calculate the CLK_OUT, see Section 4.2.

954 Back Channel Bit Rate:

$$\text{BC Rate} = 2(f_0) = 2(25\text{M}) = 50\text{ Mbps} \tag{1}$$

953 Forward Channel Bit Rate:

$$\text{FC Rate} = 2 \times 80(f_0) = 160(25\text{M}) = 4\text{ Gbps} \tag{2}$$

953 CSI Throughput:

$$\text{CSI Throughput} = 2 \times 80(f_0) \times (32/40) = 160(25\text{M}) \times (32/40) = 3.2\text{ Gbps} \tag{3}$$

CSI Throughput per Lane:

$$\frac{\text{CSI Throughput}}{\text{Lane}} = \frac{2 \times 80(f_0) \times (32/40)}{4} = \frac{160(25\text{M}) \times (32/40)}{4} = 800\text{Mbps} \tag{4}$$

Table 5. Mode Clock Calculation Table

MODE	953 CLK_IN (MHz)	954 REFCLK (MHz)	954 BC RATE (Mbps)	953 FORWARD (FC) RATE (Mbps)	953 CSI THROUGHPUT	MAX CSI THROUGHPUT	MAX CSI/LANE	CLK_OUT
Synchronous	NA	fo	2 × fo	fo × 160	≤ fo × 160 × 32/40	3.32 Gbps	832 Mbps	FC / HS_CLK_DIV) × (M/N)
Non Sync CLK_IN	f1 / CLKIN_DIV	NA	10 Mbps	f1 × 80	≤ f1 × 80 × 32/40	3.32 Gbps	832 Mbps	FC / HS_CLK_DIV) × (M/N)
	f2 / CLKIN_DIV			f2 × 40	≤ f2 × 40 × 32/40	3.32 Gbps	832 Mbps	FC / HS_CLK_DIV) × (M/N)
Non Sync AON	NA	NA	10 Mbps	f3 × 80	≤ f3 × 80 × 32/40	3.32 Gbps	832 Mbps	N/A

Table 6. Mode Clock Settings With Descriptions of fo and f1

	POSSIBLE RANGE (MHz)	DIVIDE
fo	24 to 26	N/A
f1	25 to 52	for CLKIN_DIV = 1

Copyright © 2019, Texas Instruments Incorporated

Table 6. Mode Clock Settings With Descriptions of fo and f1 (continued)

	POSSIBLE RANGE (MHz)	DIVIDE
f2	50 to 104	for CLKIN_DIV = 2
f3	48.4 to 51	for CLKIN_DIV = 1, OSCCLK_SEL = 1

2.2 Successful I²C Communication With 953 and 954

1. **Read Device ID of derserializer and serializer locally**.
   - On both the 953 and 954, register 0x00 contains the I2C_DEVICE_ID that is necessary for I^2C communication. Communicating locally does not require transactions across the bidirectional control channel (BCC), which means I^2C commands are made directly to the device.
   - Using the device ID that is established by the IDX pin, use a read command to access the value in register 0x00. If the correct value is returned then communication to the device was received. If the value returned was different than expected, see Section 2.1 about verifying the IDX pin. If the value returned was zero, then communication to the device may not be functioning properly.
   - The Python code used in Analog Launch Pad (ALP) for this operation is represented below where the 954_ID is the 8-bit address assigned by the IDX pin, 0x00 is the address where the read will occur, and 1 is number of bytes returned after the read command occurs:

     board.ReadI2C(954_ID,0x00,1)
2. **Verify that the deserializer and serializer are locked**.
   - The LOCK status serves the purpose of validating the link integrity of the connection between the SER and DES. When the LOCK status is high, the PLL in the DES is locked and validates the data and clock recovered from the serial input.

   Note that the deserializer and serilaizer may not be locked in the beginning stages of bringing up the system. As a result, continue to check the basic design rules and verify that they are correct.
   - On the 954, the DEVICE_STS can be found in register 0x04. Bit [2] holds the lock status of the device. In addition to the lock status, the device status holds many status flags regarding the reference clock, pass, power-up initialization, and check sum configuration. A healthy link between the SER and DES is indicated by the value 0xCF.
3. **Read the Device ID of serializer using the deserializer**.
   - After using I^2C locally and verifying a lock between devices, the next step is to send a transaction over the BCC and verify that it works. A basic way to do this is to read the SER ID using the DES which is found in register 0x5C on the 954.
   - To read the device ID of the SER from the DES, the Alias ID must be used for I^2C transactions. Simply reading register 0x00 of the SER Alias ID should return the value set by the IDX pin. For more information about aliasing, see Section 2.2.1.

2.2.1 Aliasing

Device Alias ID refers to the alternate 7-bit address assigned to either the serializer, deserializer, or remote slave. The Device Alias can help differentiate devices that have the same Device ID or physical I^2C address. TI recommends that the I^2C master always use the device alias to communicate with a remote I^2C slave.

For example, the DS90UB954-Q1 can support two serializers like the DS90UB953-Q1. If both serializers are 953s that house the same camera, the device IDs and default alias IDs for the corresponding devices will be the same. As a result, the best practice is to write a unique alias ID to each device. Note that these conventions only apply when the I^2C passthrough is enabled. Refer to Section 2.3 on I^2C passthrough for more information.

I^2C addresses are always 7 bits (binary). The majority of the registers on the DS90UB95x-Q1 associated with I^2C addresses uses bits [7:1] for the address, and bit 0 is either reserved or used for some other purpose. Therefore, while loading an address value to a specific register, it is always left-shifted by 1 bit. For example, 0x50 (101 0000) left-shifted by 1 bit is 0xA0 (1010 0000). This operation can be represented as, **0x50<<1** which is equal to **0xA0**.

Copyright © 2019, Texas Instruments Incorporated

Table 7. Bit Description of SER_ALIAS_ID Register With Example

ADDR.	0x5C[7:1]	0x5C[0]
Bits	101 0000	0
Dec.	Remote SER Alias ID	Automatically Acknowledges I2C writes to SER

If a defined alias ID does not follow this convention, problems can arise. For example, register 0x5C on the 954 holds the SER_ALIAS_ID. If bit [0] of the 8-bit address is set to 1, transactions using this alias ID will be automatically acknowledged. As a result, the controller (or master) sends the slave address and does not listen for a response from the slave when communicating with the serializer on the bus. All writes are attempted regardless of the forward channel lock state or status of the remote Serializer Acknowledge. This can be problematic when validating the link between the SER and DES.

2.2.2 Port Selection on 954

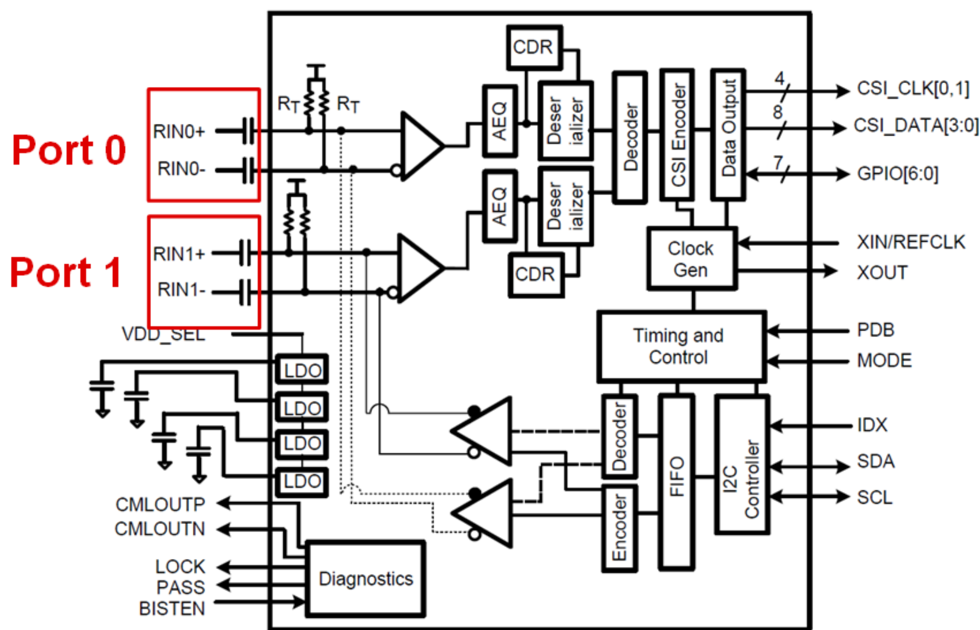

Figure 7. Illustration of Two Ports

The DS90UB954-Q1 has two ports (Port 0 and Port 1) that allow the user to interface two serializers—and subsequently, two image sensors—with one 954. As a result, the entire register space for Port 0 is similar but independent to Port 1. Therefore, the port must be accounted for when doing read or write commands to registers on the 954.

The FPD3_PORT_SEL register, with address of 0x4C, has the ability to control which port is read and which port has permission to write. Specifically, bit [4] controls which port is read where 0 indicates that read commands access port 0 and 1 indicates that read commands access port 1. Finally, bits [1] and [0] control write permissions for port 1 and port 0, respectively. Any combination of RX port registers can be written simultaneously. This is summarized in Table 8.

Broadcast mode refers to writing both to the port 0 and port 1 serializers simultaneously. This can achieved by defining both alias IDs to be the same and enabling writes to both ports.

Copyright © 2019, Texas Instruments Incorporated

Table 8. Bit Description of FPD3_PORT_SEL Register 0x4C

ADDR.	0x4C[7:5]	0x4C[4]	0x4C[3:2]	0x4C[1:0]
Bits	0X0	X	00	XX
Desc.	Holds physical port number and reserved bits	Determines which port is read.	Reserved	Enables writes for Port 1 and Port 0

2.3 I2C Passthrough Verification

The Bidirectional Control Channel Configuration register is located on the 954 and holds I2C passthrough and back channel controls. Respectively, bits [6] and [7] enable I2C passthrough and I2C passthrough all. I2C passthrough controls the way that I2C commands are mapped through the BCC. The BCC_CONFIG register has the address of 0x58.

I2C passthrough, bit [6] of 0x58, allows communication to slaves (SER) and remote slaves (image sensor) using their alias ID. As a result, it is not possible to communicate to the SER using its device ID defined by the IDX pin. See Section 2.2.1 for more information.

NOTE: TI recommends enabling this bit for normal operation of the 953 and 954 system.

I2C passthrough all, bit [7] of 0x58, allows communication to slave devices using their alias ID or device ID.

NOTE: Communication with remote slaves is not ensured while this bit is enabled. This setting should not be used except for debugging purposes or for certain situations where data must be sent in a critical timing window and an acknowledgment (ACK) is not required during the I2C communication.

A summary of these settings are shown in Section 3.2.3.

Table 9. Communication of I2C Devices Using Various Passthrough Settings

I2C PASSTHROUGH BIT 0x58[6]	I2C PASSTHROUGH ALL BIT 0x58[7]	COMMUNICATE WITH REMOTE SERIALIZER		ENSURED COMMUNICATION WITH REMOTE SLAVE WITH ALIAS AND ID DEFINED
		WITHOUT SER ALIAS DEFINED	WITH SER ALIAS DEFINED	
1	0	No	Yes	Yes
1	1	Yes	Yes	No
0	1	Yes	Yes	No
0	0	No	No	No

2.4 Basic Diagnostic and Error Registers

This section discusses the various errors and registers used in 953 and 954 systems. Various errors will be presented and discussed using basic definitions. The various registers pertaining to the errors are found in the tables below. More information regarding these registers can be found in the data sheet.

Parity Errors: Parity errors refer to errors that occur over the forward channel. These errors are caused by irregular changes to data, as it is recorded when it is entered in memory. Note that these errors only have to do with the link between the SER and DES (found in Section 3), which means they are independent of CSI errors. Check addresses 0x55, 0x56, and 0x4D on the 954.

Cyclic Redundancy Check (CRC) Errors: CRC errors refer to errors that occur over the back channel. These errors are caused by accidental changes to the data. The redundancy in the transmitted data is checked and flagged. Note that these errors only have to do with the link between the SER and DES (found in Section 3), which means they are independent of CSI errors. Check addresses 0x4D on the 954 and 0x55 and 0x56 on the 953.

Copyright © 2019, Texas Instruments Incorporated

BIST CRC Errors: BIST CRC Errors refer to errors generated during the Built-In Self Test (BIST) between the 954 and 953. Note that these errors only have to do with the link between the SER and DES (found in Section 3), which means they are independent of CSI errors. For more information about BIST, refer to Section 3.2. Check addresses.

CSI Errors: CSI errors refer to errors in CSI data packets. These can be Checksum, Length, or ECC errors. CSI errors can occur across any of the links, however, they usually occur when designing the link between the DES and ISP (found in Section 5) is designed.

CSI Checksum Errors: CSI Checksum errors refer to an error detected in the packet data portion of the CSI packet.

CSI Length Errors: CSI length errors refer to an error detected in expected packet length. Packet length errors occur if the data length field in the packet header does not match the actual data length for the packet.

CSI Error Correcting Code (ECC) Errors: CSI ECC errors refer to errors in CSI data packets that are 1 or 2 bits off from their correct value. Errors that are 1 bit, are automatically corrected while errors that are 2 bits are detected but not corrected.

FPD III Encoder Error: FPD III Encoder error refers to errors in the FPD-Link III encoding that has been detected by the receiver. These are also tied to the Link error count and Link error threshold.

Buffer Error: Buffer errors refer to the overflow condition that has occurred on the packet buffer FIFO.

Table 10. DS90UB954-Q1 Registers Used for Diagnostics and Checking Errors

DEVICE	REGISTER NUMBER	REGISTER NAME	REGISTER DESCRIPTION
954	0x04	DEVICE_STS	General flags of device status and communication between DES and SER: check sum config, power up initialization, refclk valid, pass, lock
954	0x05	PAR_ERR_THOLD_HI	Parity error threshold high byte that provides 8 most bits of threshold value. Flagged in RX_PORT_STS1
954	0x06	PAR_ERR_THOLD_LO	Parity error threshold low byte that provides 8 most bits of threshold value. Flagged in RX_PORT_STS1
954	0x4D	RX_PORT_STS1	Flags for various detected errors: BCC CRC error, Lock status Change, BCC sequencing error, Parity Error, Pass, and Lock
954	0x4E	RX_PORT_STS2	Flags for various detected errors: Line Length Unstable, Line Length changed, FPD3 Encoder error, packet buffer error, CSI Error, frequency stable, FPD3 CLK detect, Line count change
954	0x55	RX_PAR_ERR_HI	8 MSBs of FPD3 Parity Errors
954	0x56	RX_PAR_ERR_LO	8 LSBs of FPD3 Parity Errors
954	0x57	BIST_ERR_COUNT	Returns BIST error count
954	0x7A	CSI_RX_STS	Has general flags for CSI errors: Packet length, check sum, 2-bit ECC, 1-bit ECC
954	0x7B	CSI_ERR_COUNTER	Returns counts number of CSI packets received with errors
954	0xB9	LINK_ERROR_COUNT	Enables serial link data integrity error count, link error count threshold, and waiting for SFLITER to stabilize
954	0xD0	PORT_DEBUG	Indicates SER is in BIST mode. If not SER is not in BIST and bit [5] is high, could indicate error
954	0x24	INTERRUPT_STS	If interrupt enabled, flags when and where interrupts occurred: global interrupt, CSI Transmit port 0, RX port 1 and port 0

Copyright © 2019, Texas Instruments Incorporated

Table 10. DS90UB954-Q1 Registers Used for Diagnostics and Checking Errors (continued)

DEVICE	REGISTER NUMBER	REGISTER NAME	REGISTER DESCRIPTION
954	0x36-0x37	CSI_TX_ICR, CSI_TX_ISR	Detects CSI RX errors and enables interrupts if necessary
954	0xD8-0xDB	PORT_ICR_LO, PORT_ICR_HI, PORT_ISR_HI, and PORT_ISR_LO	Interrupts on various errors: see the 954 data sheet register table for more information

Table 11. DS90UB953-Q1 Registers Used for Diagnostics and Checking Errors

DEVICE	REGISTER NUMBER	REGISTER NAME	REGISTER DESCRIPTION
953	0x49	BC_CTRL	Back channel control used for clearing CRC and BIST errors and TX-RX link detect timer value
953	0x52	GENERAL_STS	General flags that indicate status of errors (BCC, BIST CRC, CRC, Link Lost) and communication between DES and SER
953	0x54	BIST_ERR_CN1	8 bits that count the CRC errors in BIST mode
953	0x55	CRC_ERR_CNT1	CRC Error count (LSB)
953	0x56	CRC_ERR_CNT2	CRC Error count (MSB)
953	0x5C	CSI_ERR_CNT	Counts number of CSI packets received with errors since the last read of the counter
953	0x5D	CSI_ERR_STATUS	Shows Line Length mismatch, Check sum error, ECC 2-bit error detected, ECC 1-bit Error Detect
953	0x5E	CSI_ERR_DLANE01	Shows errors for lanes 0 and 1: single bit error in sync, multi error in sync, control error in HS request Mode
953	0x5F	CSI_ERR_DLANE23	Shows errors for lanes 2 and 3: single bit error in sync, multi error in sync, control error in HS request Mode
953	0x60	CSI_ERR_CLK_LANE	CLK Lane: control error in HS request mode, Invalid LP state detected
953	0x79	BCC_STATUS	Error flags over BCC: master timeout, slave error, slave time out, and SER Response
953	0x77	ECC_ERR_SEL	Choose to force many different ECC errors

3 Designing the Link Between SER and DES

The link between the 953 and 954 is a core link that ensures proper communication between devices. Considering back channel configuration, BIST, AEQ, CML, and internal pattern generation will ensure that the link between the SER and DES is functional.

Copyright © 2019, Texas Instruments Incorporated

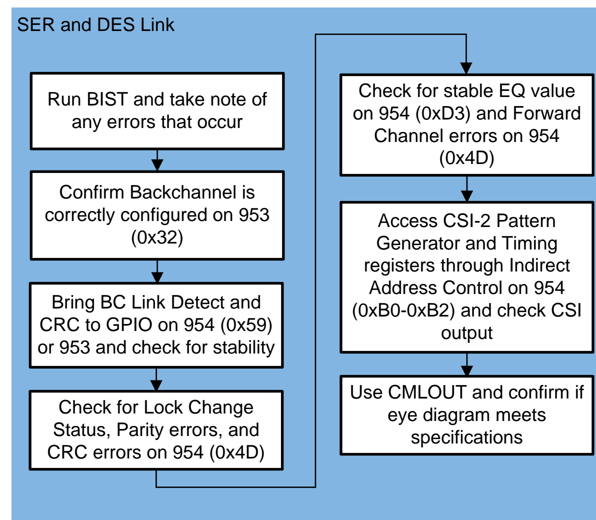

Figure 8. Flowchart for SER and DES Link

3.1 Back Channel Configuration

As shown in Section 2.1.1, the 953 is compatible with certain back channel frequencies. Furthermore, the 954 has the ability to adjust the back channel frequency to be compatible with the connected device. The process for changing the back channel configuration is shown below:

1. **On the 954, configure the back channel frequency select for 953 compatibility using the BCC Configuration register**.
   - The BCC_CONFIG register can be found at the 0x58 address. This register has the settings for I2C passthrough, which is explained in Section 2.3, as well as the BC frequency selector. By setting bits [2:0] to 0b110, the back channel will be set to the 953 default rate of 50 Mbps.
   - Note that DS90UB913A-Q1 and DS90UB933-Q1 have different defaults for back channel rates. As a result, they can be adjusted accordingly. See the 954 data sheet in the BCC_CONFIG register description for more information.

Copyright © 2019, Texas Instruments Incorporated

Table 12. Settings for Bidirectional Configuration (BCC_CONFIG) Register 0x58 on 954

PAGE	ADDR (HEX)	REGISTER NAME	BIT(S)	FIELD	TYPE	DEFAULT	DESCRIPTION
RX	0x58	BCC_CONFIG	7	I2C PASSTHROUGH ALL	RW	0	I2C Passthrough All Transactions 0: Disabled 1: Enabled
			6	I2C PASSTHROUGH	RW	0	I2C Passthrough to serializer if decode matches 0: Passthrough Disabled 1: Passthrough Enabled
			5	AUTO ACK ALL	RW	0	Automatically Acknowledge all I2C writes independent of the forward channel lock state or status of the remote Acknowledge 0: Disable 1: Enable
			4	BC_ALWAYS_ON	RW	1	Back channel enable 0: Back channel enable requires setting of either I2C_PASS_THROUGH and I2C_PASS_THROUGH_ALL. This bit may only be written through a local I^2C master. 1: Back channel is always enabled independent of I2C_PASS_THROUGH and I2C_PASS_THROUGH_ALL
			3	BC CRC GENERATOR ENABLE	RW	1	Back Channel CRC Generator Enable 0: Disable 1: Enable
			2:0	BC FREQ SELECT	RW, S	0x0	Back Channel Frequency Select. Default value set by strap condition upon asserting PDB = HIGH. 000: 2.5 Mbps (default for DS90UB933-Q1 or DS90UB913A-Q1 compatibility) 001- 011: Reserved 100: 10 Mbps (default for CSI Asynchronous back channel compatibility) 101: 25 Mbps 110: 50 Mbps (default for DS90UB953-Q1 CSI Synchronous back channel compatibility) 111: 100 Mbps Note that changing this setting will result in some errors on the back channel for a short period of time. If set over the control channel, the deserializer should first be programmed to Auto-Ack operation to avoid a control channel time-out due to lack of response from the serializer.

3.2 BIST

An optional At-Speed Built-In Self Test (BIST) feature supports testing of the high-speed serial link and the back channel without external data connections. This is useful in the prototype stage, equipment production, in-system test, and system diagnostics.

3.2.1 BIST Configuration and Status

The BIST mode is enabled by BIST configuration register 0xB3. The test may select either an external PCLK or the internal oscillator clock (OSC) frequency in the Serializer. In the absence of PCLK, the user can select the internal OSC frequency at the deserializer through the BIST configuration register.

Copyright © 2019, Texas Instruments Incorporated

When BIST is activated at the deserializer, a BIST enable signal is sent to the serializer through the Back Channel. The serializer outputs a continuous stream of a pseudo-random sequence and drives the link at speed. The deserializer detects the test pattern and monitors it for errors. The serializer also tracks errors indicated by the CRC fields in each back channel frame. While the lock indications are required to identify the beginning of proper data reception, for any link failures or data corruption, the best indication is the contents of the error counter in the BIST_ERR_COUNT register 0x57 for each RX port.

3.2.2 BIST Procedure

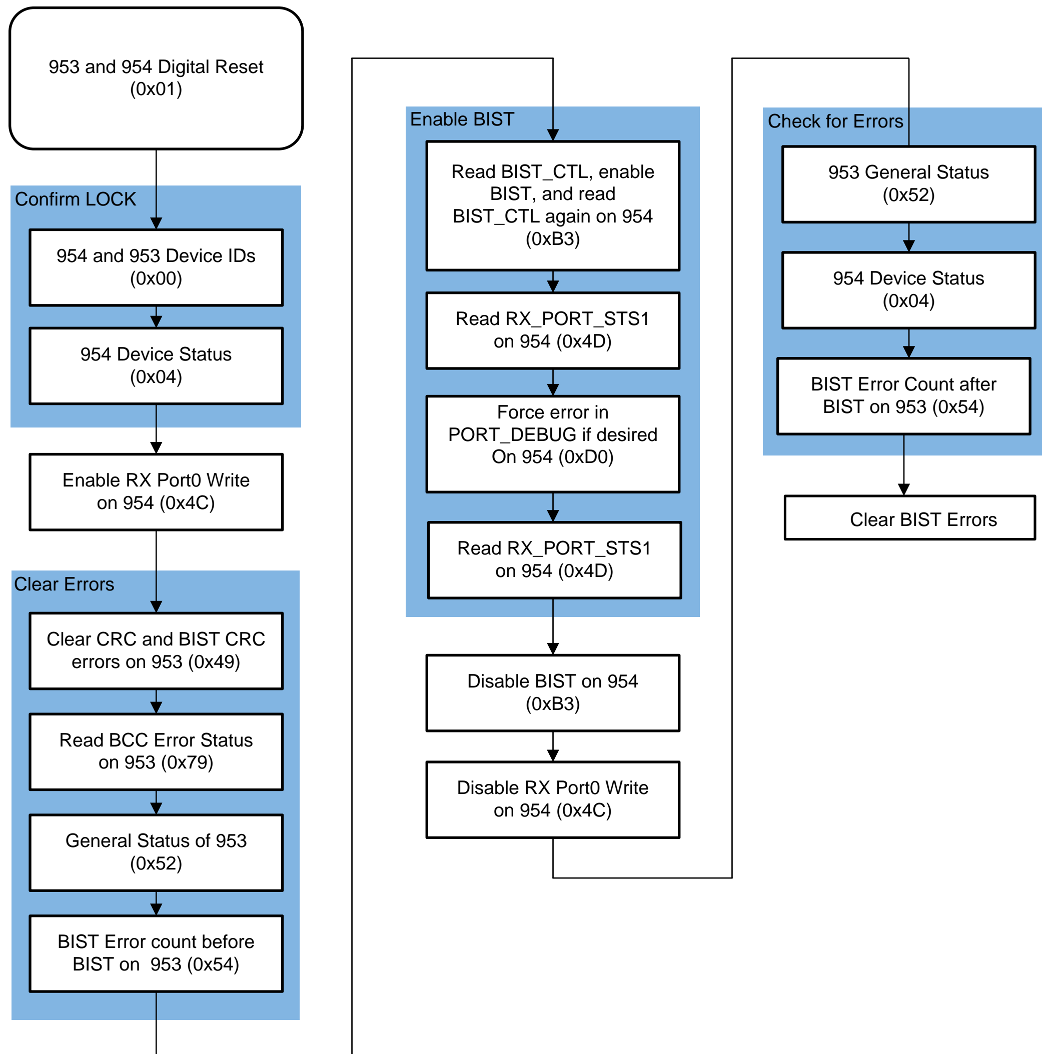

Figure 9. BIST Script Flowchart

The following steps will explain how BIST is conducted between the DS90UB953-Q1 and DS90UB954-Q1. After basic rules have been followed, BIST will determine the health of the link between the SER and DES only. Example code is listed in Section 7.1.1 and a list of registers used in the code is shown in Table 16 and Table 17.

Table 13. RESET_CTL Register Description on 953

ADDR.	0x01[7:3]	0x01[2]	0x01[1]	0x01[0]
Bits	XXX	X	0	1
Desc.	Reserved	Restart ROM Auto-Load	Digital reset including registers	Digital reset except registers

Copyright © 2019, Texas Instruments Incorporated

1. **Use the reset register to reset the entire digital block and set the serializer (SER) and deserializer (DES) to a known state.**
   - Register 0x01 is the RESET_CTL register on both the 953 and 954.
   - Note that bit [0] in REST_CTL does not reset the registers. Bit [1] controls the digital reset responsible for clearing registers. Generally, it is better to conduct a digital reset without clearing the registers to save time initial troubleshooting steps.

Table 14. DEVICE_STS Register Description on 954

ADDR.	0x04[7]	0x04[7]	0x04[7]	0x04[7]	0x04[7]	0x04[7]	0x04[1:0]
Bits	X	X	X	X	1	1	XX
Desc.	Configuration Checksum Passed	Power-up Initialization Complete	Reserved	REFCLK Valid	PASS Status	LOCK Status	Reserved

Table 15. GENERAL_STATUS Register Description for Lock on 953

ADDR.	0x52[7]	0x52[6]	0x52[5]	0x52[4]	0x52[3]	0x52[2]	0x52[1]	0x52[0]
Bits	X	1	X	X	X	1	X	1
Desc.	Reserved	LOCK Status	Reserved	BC Link lost	BIST Error detected	FC High-speed lock detected	BC error detected	BC Link detected

2. **Confirm that SER and DES are locked by accessing the respective devices IDs and DES device status.**
   - As mentioned before, verifying the correct SER and DES device IDs indicates that the correct DES device ID and SER alias ID is used when making I2C commands. The DEVICE_ID for both devices can be found in register 0x00 while the SER_ALIAS_ID can be found on the deserializer in register 0x5C.
   - The DES device status will indicate if the LOCK status is high. As mentioned before, the LOCK status serves the purpose of validating the link integrity of the connection between the SER and DES. When the LOCK status is high, the PLL in the DES is locked and validates the data and clock recovered from the serial input. As a result, the value should be 0xCF. DEVICE_STS on the 954 can be found in register 0x04.
3. **Enable the write permission for RX Port0.**
   - On the DES, this bit allows data to be written to RX port 0 registers. Any combination of RX port registers can be written simultaneously. This applies to all paged FPD3 Receiver port registers. Remember, configuring parameters that pertain to RX port 0 registers require a write command. If permission to write RX port 0 is not given, then register values will not be changed even with correct write commands. The FPD3_PORT_SEL can be found in register 0x4C.
4. **On the SER, clear any previous errors in the system before enabling BIST by clearing the CRC errors and BIST CRC errors in addition to reading the BCC status and SER general status.**
   - BIST CRC errors and CRC errors are cleared to ensure that the next BIST test only includes errors from the specified test. These errors are housed in the BC_CTRL register located at 0x49 on the 953. Value 0x28 selects the self-clearing bits to clear the BIST CRC and CRC errors.
   - Reading the bidirectional control channel (BCC) status has two functions. First, if there are any errors that occurred over the BCC, they may be flagged and categorized in this register. Second, reading this register may clear a flag that is raised in the SER general status register that can only be cleared when the BCC status register is read. The BCC_STATUS register is located at 0x79 on the 953.
   - The general status of the SER holds indicators for many different types of errors, including BIST CRC ERR and the RX LOCK DETECT. Verifying its value before BIST will show the changes that occur after BIST. A typical general status will indicate the value 0x45 showing that RX Lock Detect, HS PLL Lock, and Link Detect flags are high. GENERAL_STS can be found at register 0x52 on the 953.

Copyright © 2019, Texas Instruments Incorporated

5. **Read the BIST error counter before BIST.**
   - In addition to checking the indicator if an error occurred, it is important to read the number errors before the test. If one error is forced, we expect this value to be 0x01 after the BIST has completed. The BIST_ERR_CNT can be found in register 0x54 of the 953.
6. **On the DES, read the BIST control register, enable the BIST, force the singular, multiple, or no errors, and read the RX port status after the BIST starts and before the BIST ends.**
   - The BIST control register enables the BIST as well as various parameters important for BIST. Bit [0] of the BIST control represents BIST_EN. Reading it before and then after enabling BIST helps show that BIST has been enabled. By using the sleep command, BIST can run as long as the user desires. BIST_CTL can be located in register 0xB3 of the 954.
   - The RX port status, like the general status and BCC status on the 954, houses many useful error flags. These flags include: the BCC CRC error, lock status change, BCC sequence error, parity error, receiver pass indication, lock status, and locked to recovered clock status. Consider bit [4] which indicates the LOCK_STS_CHG. By reading this value right after the BIST is enabled, and right before the BIST is disabled, the user is able to tell if the lock status changed during BIST. It is important to read this register after BIST is started because enabling and disabling the BIST forces the devices to relock. RX_PORT_STS1 can be found in register 0x4D.
   - The port debug register is used for debugging various functions by enabling build in errors or tests. Bit [0] is self-clearing and controls the FORCE_ONE_BC_ERROR function. Bit [1], by comparison, controls FORCE_BC_ERRORS and must be cleared. This is helpful for testing the error detection of the system. The PORT_DEBUG register can be found at 0xD0 of the 954.
7. **Disable the BIST and write permissions of RX Port0.**
   - Disabling the BIST is done through the BIST control register, 0xB3, on the DES. BIST uses all functionality of the BCC and disabling the BIST reestablishes communication between the SER and DES.
   - This bit allows data to be written to RX port 0 registers. Any combination of RX port registers can be written simultaneously. This applies to all paged FPD3 Receiver port registers. Because only reads are required at this stage the BIST, it is good practice to disable the write permission. The FPD3_PORT_SEL can be found in register 0x4C.
8. **Check for errors on the SER general status, DES device status, and BIST error count.**
   - The general status of the 953, with register value 0x52, that was checked before BIST gave a value of 0x45. With the introduction of an error, a value of 0x4D is read from the register. This value indicates that the RX Lock Detect, HS PLL Lock, Link Detect, and BIST CRC Error flags are high.
   - The device status of the 954 was checked before BIST. Despite forcing an error in the system, the status should read still read 0xCF because the devices should still be LOCKED.
   - Before BIST was enabled, all possible BIST CRC errors were cleared and the BIST CRC error count was read. After forcing one error during BIST, one error should be displayed in the BIST_ERR_CNT. If the count is 0x01, that means that the devices were LOCKED, any errors that occurred were only produced during the BIST test, the error detection system is working as intended, and the link between the SER and DES is not producing any errors.

The LOCK and PASS statuses ensure that the serializer and deserializer are communicating effectively. By using the built-in self test (BIST), the LOCK between devices can be accurately evaluated. As a result, this is a fundamental step to ensure proper design of the board.

3.2.3 List of Registers Used in BIST Script

Table 16. DS90UB954-Q1 Registers Used in BIST Script

DEVICE	REGISTER NUMBER	REGISTER NAME	REGISTER DESCRIPTION
954	0x00	DEVICE_ID	I^{2}C device ID for DES. Used in I^{2}C commands
954	0x01	RESET_CTL	Responsible for digital resets and analog power down

Copyright © 2019, Texas Instruments Incorporated

Table 16. DS90UB954-Q1 Registers Used in BIST Script (continued)

DEVICE	REGISTER NUMBER	REGISTER NAME	REGISTER DESCRIPTION
954	0x5C	SER_ALIAS_ID	I^2C alias ID that is used to communicate to SER from DES
954	0x04	DEVICE_STS	General flags that indicate status of device and communication between DES and SER
954	0x4C	FPD3_PORT_SEL	RX port register
954	0xB3	BIST_CTL	Various parameters important for control of BIST
954	0x4D	RX_PORT_STS1	General flags that indicate various errors
954	0xD0	PORT_DEBUG	Port used for debugging various functions of DES
954	0x55	RX_PAR_ERR_HI	Shows 8 most significant bits of errors that occur on the forward channel
954	0x56	RX_PAR_ERR_LOW	Shows 8 least significant bits of errors that occur on forward channel

Table 17. DS90UB953-Q1 Registers Used in BIST Script

DEVICE	REGISTER NUMBER	REGISTER NAME	REGISTER DESCRIPTION
953	0x00	DEVICE_ID	I^2C device ID for SER. Used in I^2C commands
953	0x01	RESET_CTL	Responsible for digital resets and analog power down
953	0x49	BC_CTRL	Back channel control used for clearing errors and TX-RX link detect timer value
953	0x79	BCC_STATUS	Group of error flags that indicate various errors over BCC
953	0x52	GENERAL_STS	General flags that indicate status of errors and communication between DES and SER
953	0x54	BIST_ERR_CNT	8 bits that count the CRC errors in BIST mode

3.3 AEQ

On the 954, the receiver inputs provide an adaptive equalization filter to compensate for signal degradation from the interconnect components. To determine the maximum cable reach, consider the factors that affect signal integrity such as jitter, skew, ISI, crosstalk, and so forth. The equalization status and configuration are selected through AEQ registers 0xD2–0xD3. For more in depth AEQ control, see the AEQ register with an address of 0x42.

If these register values are continuously read and the values jump sporadically, then the AEQ is not properly compensating for factors that affect signal integrity. If the deserializer loses Lock, the adaptive equalizer will reset and perform the Lock algorithm again to reacquire the serial data stream being sent by the serializer.

In addition, the AEQ values may settle to a state that is overcompensating for the factors that affect signal integrity. If the AEQ values seem to be higher than expected in the AEQ_STATUS register (0xD3), set AEQ_RESTART (0xD2[3]) to 1, and let the AEQ values resettle. If the system continually settles to the wrong value on power up, then reset the 954 before port forwarding is enabled.

Copyright © 2019, Texas Instruments Incorporated

3.4 CML Out

The DS90UB954-Q1 includes an internal Channel Monitor Loop (CML)-through output on the CMLOUTP/N pins 38 and 39, respectively. A buffered loop-through output driver is provided on the CMLOUTP/N to observe jitter after equalization for each of the two RX receive channels. The CMLOUT monitors the post EQ stage, thus providing the recovered input of the deserializer signal. The measured serial data width on the CMLOUT loop-through is the total jitter including the internal driver, AEQ, back channel echo, and so forth. Each channel also has its own CMLOUT monitor and can be used for debug purposes. This CMLOUT is useful in identifying gross signal conditioning issues. Typically, these pins are routed to test points and not connected. For monitoring CMLOUT, be sure to terminate with 100-Ω differential load.

For more information regarding CMLOUT, refer to the Channel Monitor Loop-Through Output Driver section in the 594 data sheet. Example code fore enabling CMLOUT FPD3 RX Port 0 is given in Section 7.1.4 while the process is given in Table 18.

Table 18. Channel Monitor Loop-Through Output Configuration of the 954

	FPD-Link III RX Port 0	FPD-Link III RX Port 1
ENABLE MAIN LOOP-THROUGH DRIVER	0xB0 = 0x14; 0xB1 = 0x00; 0xB2 = 0x80 0xB1 = 0x03; 0xB2 = 0x28 0xB1 = 0x04; 0xB2 = 0x28	
SELECT CHANNEL MUX	0xB1 = 0x02; 0xB2 = 0x20	0xB1 = 0x02; 0xB2 = 0xA0
SELECT RX PORT	0xB0 = 0x04; 0xB1 = 0x0F; 0xB2 = 0x01 0xB1 = 0x10; 0xB2 = 0x02	0xB0 = 0x08; 0xB1 = 0x0F; 0xB2 = 0x01 0xB1 = 0x10; 0xB2 = 0x02
DISABLE MAIN LOOP-THROUGH DRIVER	0xB0 = 0x14; 0xB1 = 0x00; 0xB2 = 0x00 0xB1 = 0x03 ; 0xB2 = 0x08 0xB1 = 0x04; 0xB2 = 0x08	
DESELECT CHANNEL MUX	0xB1 = 0x02; 0xB2 = 0x20	0xB1 = 0x02; 0xB2 = 0x20
DESELECT RX PORT	0xB0 = 0x04; 0xB1 = 0x0F; 0xB2 = 0x00 0xB1 = 0x10; 0xB2 = 0x00	0xB0 = 0x08; 0xB1 = 0x0F; 0xB2 = 0x00 0xB1 = 0x10; 0xB2 = 0x00

4 Designing Link Between SER and Image Sensor

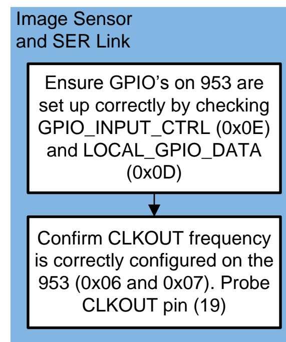

Figure 10. Image Sensor and SER Link Design

4.1 Sensor Initialization Using SER GPIOs

The following steps will generally explain how the initialization of the image sensor occurs from the DS90UB953-Q1. It is important that the link between the SER and DES is working. If the link between the SER and DES is not working, then you will not be able to use the DES to talk to the SER—and subsequently—the image sensor. Example code is listed in Section 7.1.2.

In the example script, the traces on the PCB connect GPIO0 to the imager reset, GPIO1 to the imager power down, GPIO2 to the SER WP, and GPIO3 to frame sync.

Copyright © 2019, Texas Instruments Incorporated

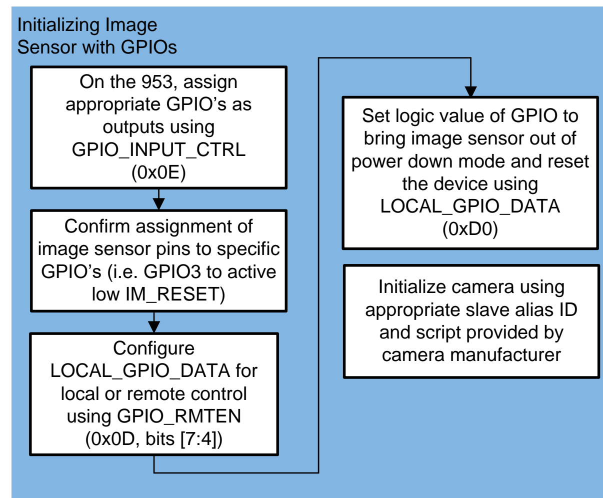

Figure 11. Flowchart for Initializing the Image Sensor Using GPIO's

1. **On the serializer enable the appropriate General-Purpose Input/Output (GPIO) pins for output mode.**
   - On the serializer, GPIO_INPUT_CTRL controls whether GPIO0-3 are configured as inputs or outputs. The first 4 bits [3:0] control whether they are used as inputs, and the last 4 bits [7:4] are used as outputs. Because all GPIOs in this example refer to settings that can be set, all GPIOs are set to outputs. This register has the value of 0x0E.
   - GPIO0 and GPIO1 on the 953 can be used for sensing the voltage applied to its input. See Section 6.8 for more information. As a result, GPIO2 and GPIO3 should be used before assigning other GPIO pins.

Table 19. Bit Description of GPIO_INPUT_CTRL Register 0x0E

ADDR.	0x0D[7:4]	0x0D[3:0]
Bits	XXXX	0000
Desc.	Controls if GPIO3-0 are outputs	Controls if GPIO3-0 are inputs

2. **Using the serializer, bring to the imager to a known state by bringing the camera out of power-down mode and resetting the camera. This can be accomplished by controlling the GPIO pins.**
   - On the 953, GPIO0-3 can be locally written to use the LOCAL_GPIO_DATA register. This register can be found at value 0x0D. The last 4 bits [7:4] (GPIO_RMTEN) force the selected GPIO pins to be remotely controlled by the deserializer, which is not required in this example. The concept of remotely controlling GPIOs is discussed in Section 5.1.1.
   - Furthermore, the first 4 bits [3:0] (GPIO_OUT_SRC) of the LOCAL_GPIO_DATA register control the logical output of the GPIO registers. Note that the corresponding GPIOs can only be changed when remote enable is disabled and the GPIOs are configured as outputs.
   - In the example script, both the imager reset and imager power-down controls are active low. By setting the GPIO1 (IM_PWDN_B) and GPIO2 (SER_WP) to high, the camera is brought out to power-down mode. Then, GPIO0 (IM_RESET) can be held low, which initializes the camera reset. Finally GPIO0 is brought high again to allow initialization of the camera.

Table 20. Bit Description of LOCAL_GPIO_DATA Register 0x0D

ADDR.	0x0E[7:4]	0x0E[3:0]
Bits	0000	XXXX
Desc.	Enables remote control of SER GPIO3-0	Controls logical outputs of GPIO3-0

Copyright © 2019, Texas Instruments Incorporated

3. **The camera is then initialized by programing the image sensor.**
   - For example, data can be written to a 32-bit address of the image sensor. This is usually provided by the camera manufacturer. Using ALP, the Python command can be written as:

     board.WriteI2C(Slave_Alias_ID, Address_pt1, [Address_pt2,Data])
     where:
     – Slave_Alias_ID is the I2C alias device ID of the image sensor
     – Address_pt1 and Address pt2 make up the address (that is, 0x1234 where Address_pt1 = 0x12 and Address_pt2 = 0x34)
     – Data is the data written to that address.

     This initialization is not provided in the example script.
   - When the camera has finished initializing, more current will be pulled from the power supply. This will indicate that the camera has successfully initialized. Consider the current limiting setting of the voltage supply. If the total current consumed after initialization is over the current limit of the supply, the voltages in the system will not be regulated correctly.

4.2 CLKOUT

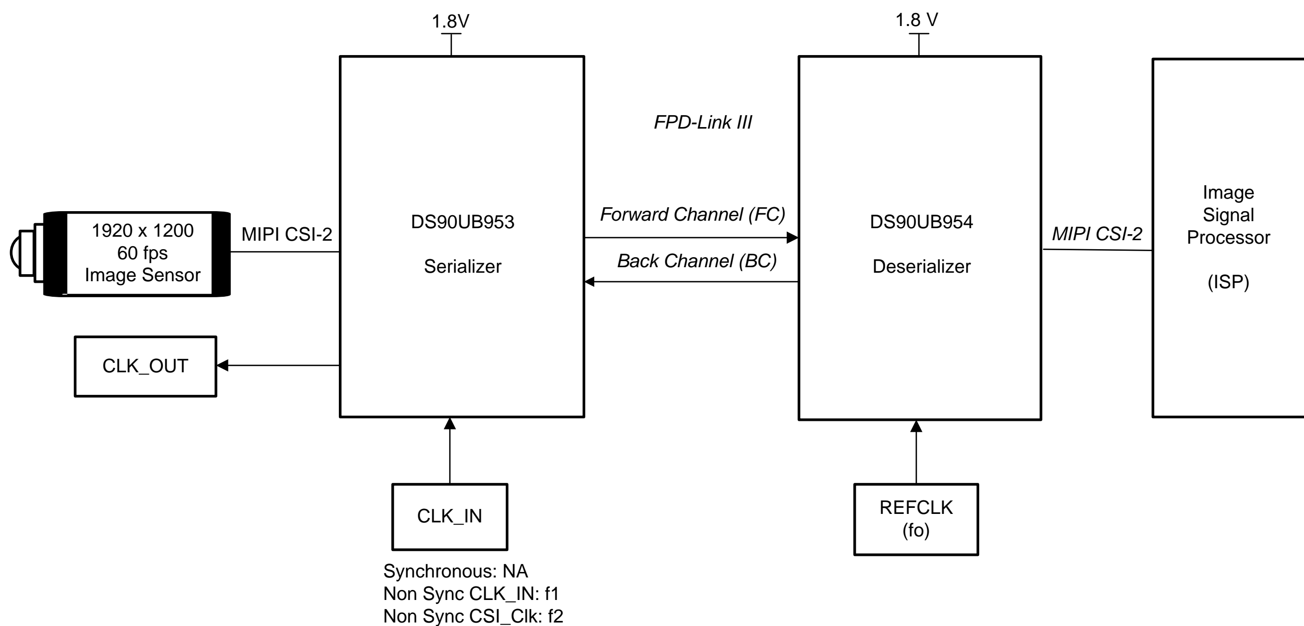

Figure 12. Clocking System Diagram

The 953 clock outline is intended as a reference clock for the image sensor. Note that the CLK_OUT/IDX pin (19) also assigns the I2C device ID on power up. See Section 2.1 for information. After power up, the clock out frequency is defined by Equation 5.

$$CLK_OUT = \frac{FC}{HS_CLK_DIV} \times \frac{M}{N}$$

where
- FC is the forward channel data rate,
- M, HS_CLK_DIV, and N are parameters set by the CLKOUT control registers 0x06 and 0x07 on the 953. (5)

For more information for calculating FC, see Section 2.1.1.

Copyright © 2019, Texas Instruments Incorporated

The PLL that generates CLK_OUT is a digital PLL that will have very low jitter if the ratio N/M is an integer. However, if N/M is not an integer, then the jitter on the signal will be approximately equal to HS_CLK_DIV/FC. As a result, if it is not possible to have an integer ratio of N/M, it is best to select a small value for HS_CLK_DIV.

If a particular frequency is required, for a system (for example, 37.125 MHz), then using values of M=0x09, N=0xF2, and HS_CLK_DIV=4 will result in an output frequency of 37.19 MHz and a frequency error of 0.175% with jitter of about 1 ns. Alternately, you could use M=0x01, N=0x1E, and HS_CLK_DIV=4 and get an output frequency of 37.037 MHz and a frequency error of 0.24% with much less jitter. A third alternative would be to use the M=0x01, N=0x1E, and HS_CLK_DIV=4, but rather than using a 25-MHz clock frequency for the DS90UB954-Q1 reference, use a frequency of 25.059 MHz for the DS90UB953-Q1 to get both a low jitter and low frequency error.

5 Designing Link Between DES and ISP

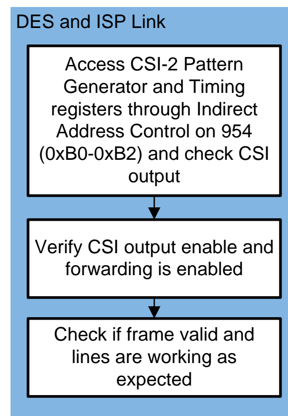

Figure 13. DES and ISP Link Flowchart

5.1 Frame Sync

Most multiple image-sensor systems require that the image data all arrive to the image signal processor (ISP) at the same time. The use of a frame synchronization signal (FrameSync) ensures that data for every frame is sent to every deserializer simultaneously. From there, the synchronized data can be forwarded to the ISP. The frame sync signal can be sent across the back channel from an internally generated signal from the DES or an externally generated signal gathered by the DES GPIOs.

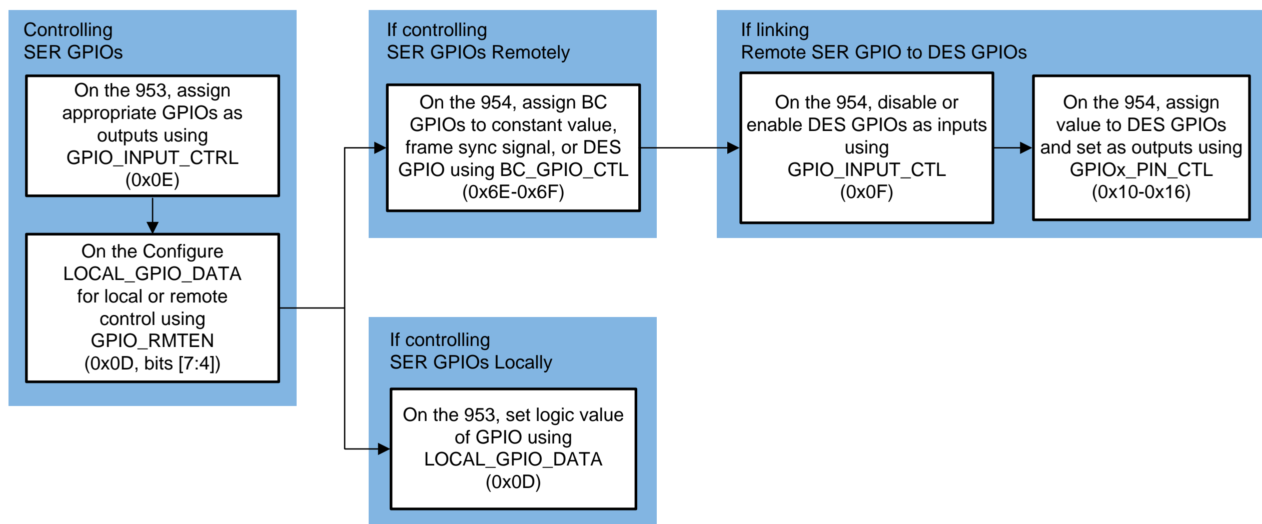

Figure 14. Steps for Controlling SER GPIOs Remotely and Locally

Copyright © 2019, Texas Instruments Incorporated

5.1.1 Using SER GPIOs From the DES

There are two ways to control the GPIOs of the SER using the deserializer: sending I2C commands across the back channel to the SER, or forwarding the SER GPIOs across the back channel using remote enable and controlling them from the DES. Because SER GPIOs are often connected to pins of the image sensor, it is important to understand how to control them.

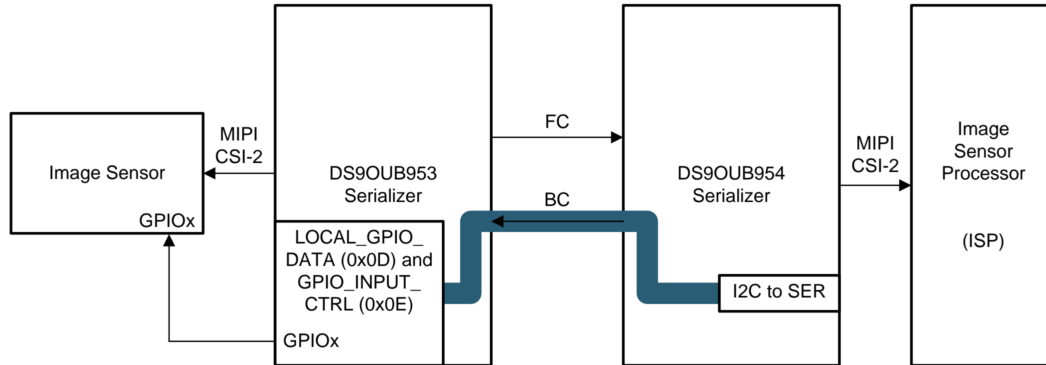

Figure 15. Block Diagram of Controlling SER GPIOs Over BC

As shown in Figure 15, the first method of controlling the serializer GPIOs is to send a write command over the back channel (BC) and control the appropriate registers on the 953. GPIO_INPUT_CTRL controls whether GPIO0-3 are configured as inputs or outputs. The first 4 bits [3:0] control whether they are used as inputs, and the last 4 bits [7:4] are used as outputs. This register has the address of 0x0E.

For example, the code below sends an I2C transaction over the BC using the 953 Alias ID, finds the GPIO_INPUT_CTRL register, and configures all four GPIOs on the 953 as outputs:

board.WriteI2C(953_Alias_ID,0x0E,0xF0)

Table 21. Example Using GPIO_INPUT_CTRL Register 0x0E for Local SER GPIO Control

ADDR.	0x0E[7:4]	0x0E[3:0]
Bits	1111	0000
Desc.	Controls if GPIO3-0 are outputs	Controls if GPIO3-0 are inputs

Furthermore, the LOCAL_GPIO_DATA (0x0D) register controls the SER GPIOs. The last 4 bits [7:4] of this register (GPIO_RMTEN) force the selected GPIO pins to be remotely controlled by the deserializer. These bits will not be used in this method. In addition, the first 4 bits [3:0] (GPIO_OUT_SRC) of the control the logical output of the GPIO registers. Note the corresponding GPIOs can only be changed when remote enable is disabled and the GPIOs are configured as outputs.

For example, the code below sends another I2C transaction over the BC using the 953 Alias ID, finds the LOCAL_GPIO_DATA register, and sets every SER GPIO to high. This is because the previous line of code set every GPIO to outputs—the last 4 bits for remote enable are 0, and the first 4 bits are set to logic 1:

board.WriteI2C(953_Alias_ID,0x0D,0x0F)

Table 22. Example Using LOCAL_GPIO_DATA Register 0x0D for Local SER GPIO Control

ADDR.	0x0D[7:4]	0x0D[3:0]
Bits	0000	1111
Desc.	Enables remote control of SER GPIO3-0	Controls logical outputs of GPIO3-0

Copyright © 2019, Texas Instruments Incorporated

An application of this process is described in Section 4.1.

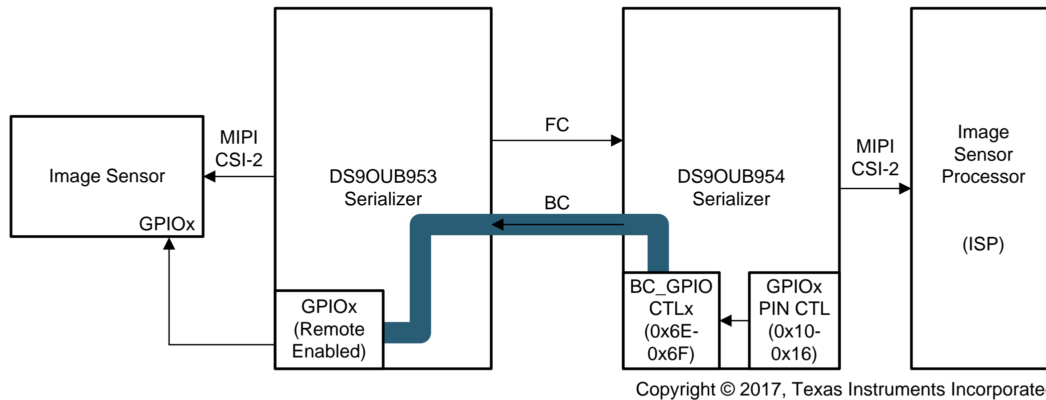

Figure 16. Block Diagram of Controlling SER GPIOs Remotely

As shown in Figure 16, the second method of controlling the serializer GPIOs is to remotely enable the SER GPIOs for access from the BC GPIO control of the DES. Then, the back channel GPIO control can be configured to mirror the DES GPIOs or different functions.

As mentioned before, the LOCAL_GPIO_DATA register controls the SER GPIOs. The last 4 bits [7:4] of this register (GPIO_RMTEN) force the selected GPIO pins to be remotely controlled by the deserializer. For example, the code below sends an I2C transaction over the BC using the 953 Alias ID, finds the GPIO_INPUT_CTRL register, and configures all four GPIOs on the 953 to be controlled by the DES:

board.WriteI2C(953_Alias_ID,0x0D,0xF0)

Table 23. Example Using LOCAL_GPIO_DATA Register 0x0D for Local SER GPIO Control

ADDR.	0x0D[7:4]	0x0D[3:0]
Bits	1111	0000
Desc.	Enables remote control of SER GPIO3-0	Controls logical outputs of GPIO3-0

Because the GPIOs are remote enabled, the GPIOs of the SER are forwarded to the DES, wait to be defined, and later send the data back to the SER. These forwarded GPIOs are stored on the DES in the form of BC GPIOs.

Table 24. BC_GPIO_CTL0 Registers From the 954 Data Sheet

ADDR (HEX)	REGISTER NAME	BIT(S)	FIELD	TYPE	DEFAULT	DESCRIPTION
0x6E	BC_GPIO_CTL0	7:4	BC_GPIO1_SEL	RW	0x8	Back channel GPIO1 Select: Determines the data sent on GPIO1 for the port back channel. 0xxx : Pin GPIOx where x is BC_GPIO1_SEL[2:0] 1000 : Constant value of 0 1001 : Constant value of 1 1010 : FrameSync signal 1011 - 1111 : Reserved
		3:0	BC_GPIO0_SEL	RW	0x8	Back channel GPIO0 Select: Determines the data sent on GPIO0 for the port back channel. 0xxx : Pin GPIOx where x is BC_GPIO0_SEL[2:0] 1000 : Constant value of 0 1001 : Constant value of 1 1010 : FrameSync signal 1011 - 1111 : Reserved

Copyright © 2019, Texas Instruments Incorporated

Because the GPIOs are remote enabled, the GPIOs of the SER are now forwarded to the DES, wait to be defined, and later send the data to the SER. These forwarded GPIOs are stored on the DES in the form of BC GPIOs. The BC_GPIO_CTL register, with an address of 0x6E and 0x6F, controls the state of the forwarded GPIOs. For example, the first 4 bits [3:0] of 0x6E control the BC_GPIO0, while the last 4 bits [7:4] of 0x6E control the BC_GPIO1. Configuring these 4 bits will select one of many different functions, such as sending constant values of 0 or 1, internally generating Frame Sync signals, or following the values of a DES GPIO.

If the BC GPIOs are not defined by one of the internally generated functions, they must be linked to one of the seven different GPIOs on the 954. Because GPIOs can function as inputs or outputs, a signal defined by the GPIOs of the 954—and subsequently the BC GPIO of the DES—will be visible on GPIO pins of the 953.

For example, the below code will define GPIO1 of the SER as 1, and link the GPIO0 of the DES to the GPIO0 of the SER. To reiterate, probing the GPIO1 pin of the SER will show 1, while probing the GPIO0 pin of the SER will be the same as probing the GPIO0 pin of DES. Note that this will not require an I2C transaction across the BC:

board.WriteI2C(954_ID,0x6E,0x90)

Table 25. Example using BC_GPIO_CTL0 Register 0x6E for Remote SER GPIO Control

ADDR.	0x6E[7:4]	0x6E[3:0]
Bits	1001	0000
Desc.	Determines data sent to SER GPIO1 (selected for constant value of 1)	Determines data sent to SER GPIO0 (selected to link to DES GPIO0)

After forwarding the BC GPIOs to one of the DES GPIOs, the DES GPIOs can be configured as an output using the GPIOx_PIN_CTL with addresses of 0x10-0x16. In addition, the GPIOx_PIN_CTL has controls to select the source of the output generated to the GPIO. Bits [4:2] select the output source, such as RX Port 0, Device Status, and so forth. Bits [7:5] select the output based on what source was selected, such as the RX Port Lock indicator. Bits [1] and [0] control the logical output and output enable bits, respectively.

For example, the below code will make the DES GPIO0 an output for an internally generated FrameSync signal that is forwarded to SER GPIO0:

board.WriteI2C(954_ID,0x0F,0x00)

board.WriteI2C(954_ID,0x10,0x91)

An application of this process is explained in Section 5.1.2.

5.1.2 Internal and External Frame Sync Configuration

TI recommends that frame sync is always sent from the deserializer using Internal or External Frame Sync. In multi-camera systems, it is important that every frame sync is sent to each camera at the same time. If the system uses the SER GPIOs to send a frame sync signal to the Image Sensor, the SER GPIOs must always be remote enabled whether the frame sync is internally or externally generated.

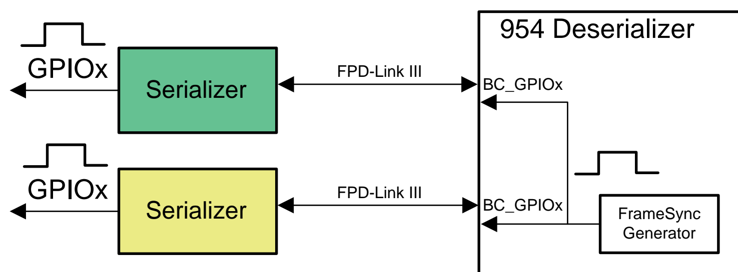

Figure 17. Block Diagram of Internally Generated Frame Sync

As shown in Figure 17, internally generated frame sync signals come from an internal block of the 954. Defining a GPIO for an internal frame sync generator can be accessed in any BC_GPIO_CTLx register (0x6E-0x6F).

Copyright © 2019, Texas Instruments Incorporated

Configuring the internal frame sync mode is done in the FS_CTL register with an address of 0x18. Enabling the internal FrameSync mode is done by setting the first bit of the FS_CTL register (0x18 [0]) to a value of 1, which is the FS_GEN_ENABLE control. The last four bits of the FS_CTL register, FS_MODE field (0x18 [7:4]), controls the clock source used for the FrameSync generation. The second bit in FS_CTL—the FS_GEN_MODE field (0x18 [1])—configures whether the duty cycle of the FrameSync is 50/50 or whether the high and low periods are controlled separately.

The FrameSync high and low periods are controlled by the FS_HIGH_TIME_x and FS_LOW_TIME_x (0x19–0x1C) registers. For more information regarding how to program high and low time frame sync, refer to the 954 data sheet. The resolution of the FrameSync generator clock (FS_CLK_PD) is derived from the back channel frame period. The frame period can be found in the first 3 bits of the BC_CONFIG register—the BC FREQ SELECT (0x58 [2:0]). For 50-Mbps back channel operation, the frame period is 600 ns (30 bits × 20 ns/bit).

An example on how to enable the Frame sync this way is shown in Section 5.1.1.

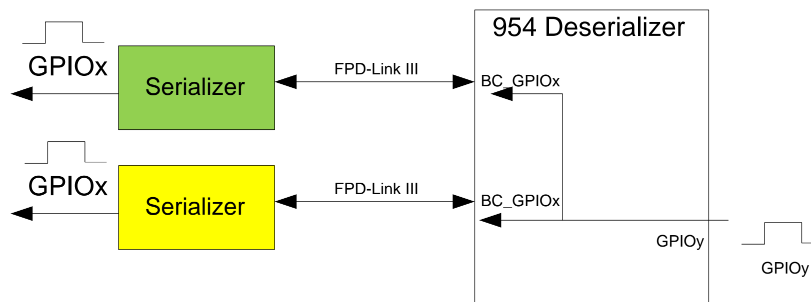

Figure 18. Block Diagram of Externally Generated Frame Sync

In External FrameSync mode, an external signal inputs to the DS90UB954-Q1 through one of the GPIO pins on the device. As shown in Figure 18, the external FrameSync signal may be propagated to one or more of the attached FPD3 serializers through a GPIO signal in the back channel. The expected skew timing for external FrameSynch mode is on the order of one back channel frame period.

Enabling the external FrameSync mode is done by setting the last 4 bits of the FS_CTL register (0x18 [7:4]) that indicates the FS_MODE control. The value should be between 0x8 (GPIO0 pin) to 0xF (GPIO7 pin). Set FS_GEN_ENABLE to 0 for this mode.

To send the FrameSync signal on a port's BC_GPIOx signal, the BC_GPIO_CTL0 (0x6E) or BC_GPIO_CTL1 (0x6F) register should be programmed for that port to select the FrameSync signal.

5.1.3 Tables for Using GPIOs and Frame Sync

Table 26. Registers Used When Configuring GPIOS and Frame Sync on the 953 and 954

DEVICE	REGISTER NAME	REGISTER NAME	REGISTER DESCRIPTION
953	0x0E	GPIO_INPUT_CTRL	Determines if SER GPIOs are inputs or outputs
953	0x0D	LOCAL_GPIO_DATA	Determines if SER GPIOs are remotely enabled and are the logical outputs of the GPIO
954	0x6E-0x6F	BC_GPIO_CTLx	Defines the BC GPIO on the DES
954	0x10-0x16	GPIO_PIN_CTLx	Determines if the GPIO is output and defines the logical output, output source, and output
954	0x4C	FPD3_PORT_SEL	RX port register
954	0x0F	GPIO_INPUT_CTL	Determines if GPIO0-7 are inputs
954	0x18	FS_CTL	Controls frame sync source, frame sync mode, generates a single pulse, and enables frame sync generation

Copyright © 2019, Texas Instruments Incorporated

Table 26. Registers Used When Configuring GPIOS and Frame Sync on the 953 and 954 (continued)

DEVICE	REGISTER NAME	REGISTER NAME	REGISTER DESCRIPTION
954	0x19-0x1A	FS_HIGH_TIMEx	Controls the high time of the frame sync signal
954	0x1B-0x1C	FS_LOW_TIMEx	Controls the low time of the frame sync signal
954	0x58	BC_CONFIG	Sets I2C Passthrough settings, auto ACK, BC enable, and BC frequency

5.2 Port Forwarding

Video stream forwarding is handled by the forwarding control in the DS90UB954-Q1 on FWD_CTL1 register 0x20. The forwarding control pulls data from the video buffers for each FPD3 RX port and forwards the data to one of the CSI-2 output interfaces. It also handles generating transitions between the LP and HS modes as well as sending synchronization frames. The forwarding control monitors each of the video buffers for packet and data availability.

Forwarding from input ports may be disabled using per-port controls. Each of the forwarding engines may be configured to pull data from either of the two video buffers, although both buffer may only be assigned to one CSI-2 Transmitter at a time unless in replicate mode. The two forwarding engines operate independently.

The CSI transmitter control can be found in the DS90UB954-Q1 on FWD_CTL2 register 0x21. The control holds the CSI replicate mode (0x21[7]) and the forwarding mode options. The two main forwarding options are synchronized and round-robin/best effort. Synchronized forwarding (0x21[3:2]) offers synchronization of all incoming data stored within the buffer. If packets arrive within a certain window, the forwarding control will attempt to synchronize the video buffer data. When using round-robin/best effort forwarding (0x21[0]), no attempt is made to synchronize the video traffic. When multiple sources have data available to forward, the data will be forwarded in a round-robin fashion.

For more information about CSI-2 Forwarding modes, including the difference between basic synchronized and line-interleaved forwarding, refer to the *CSI-2 Forwarding* section in the 954 data sheet.

An example script can be found in Section 7.1.3.

5.3 Pattern Generation

Pattern Generation is when the engineer uses an internal block of the 954 or 953 to generate CSI data. The generation of this data is used to check links in the system without directly using the CSI data gathered by the image sensor. As soon as the image sensor is used for checking CSI data, multiple links of the system are used, which makes identifying problems with a specific link problematic.

As a result, using the 954 pattern generation function and forwarding the data to the ISP will help identify whether or not there are problems with the CSI data traveling across the link between the DES and ISP. If the patterns generated match the output of the ISP, the link between the DES and ISP is healthy. The process can then be repeated with the pattern generation on the 953, which determines the health of the SER and DES link, and finally with the CSI data of the image sensor that determines the health of the SER and Image Sensor Link.

While this process is helpful in identifying which links are healthy, minimal insight is gained regarding the root problem of the link.

5.3.1 Accessing Indirect Registers

Pattern generation is access through the CSI-2 Pattern Generator and Timing Registers. Indirect registers hold many different registers—such as the CSI-2 Pattern Generation, Timing registers, and Analog controls—that are located at offsets of the main register space.

Copyright © 2019, Texas Instruments Incorporated

The indirect address mechanism involves setting the control register to select the desired block, setting the register offset address, and reading or writing the data register. In addition, an auto-increment function is provided in the control register to automatically increment the offset address following each read or write of the data register.

For writes, the process on both the 953 and the 954 is as follows:

1. **Write to the IND_ACC_CTL register (0xB0) to select the desired register block.**
2. **Write to the IND_ACC_ADDR register (0xB1) to set the register offset**.
3. **Write the data value to the IND_ACC_DATA register (0xB2)**.

If auto-increment is set in the IND_ACC_CTL register, repeating step 3 will write additional data bytes to subsequent register offset locations.

For reads, the process on both the 953 and the 954 is as follows:

1. **Write to the IND_ACC_CTL register (0xB0) to select the desired register block.**
2. **Write to the IND_ACC_ADDR register (0xB1) to set the register offset**.
3. **Read from the IND_ACC_DATA register (0xB2)**.

If auto-increment is set in the IND_ACC_CTL register, repeating step 3 will read additional data bytes from subsequent register offset locations.

5.3.2 Pattern Generation From DES to ISP and SER to DES

Pattern generator can be programmed to use different clock sources, the back channel recovered clock, the CSI2-clock, or the external CLK_IN pin. The output of the pattern generator is in 32-bit format. The output of the pattern generator and the output of the 32-bit CSI-2 parallel data are muxed before the data are transmitted to the TX data path module. Depending on the deserializer, the pattern generator output can be sent to a 40-bit mode or 28-bit mode data path.

The general flow for pattern generation is to enable CSI, CSI select, and pattern generation. Because the pattern generation simulates the CSI data, the pattern that appears must be configured before the rest of the parameters are defined. Because the data will come up as colored bars, the pattern configure register configures the number of color bars and the size of the block where fixed pattern is enabled.

The CSI data type must be defined through the Pattern Generator CSI DI register. While there are many different types of color spaces and various ways to define image data, such as RGB888 or YUV420 8-bit, the PGEN_CSI_DI register defaults to 0x24 indicating RGB888. This means a scale of red, green, and blue will define the colors that are used.

Because the pixels will be arranged in frames on the screen, the line size, bar size, active lines per frame, the total lines, line period, vertical back porch, and vertical front porch must be defined. Finally, the corresponding bytes are defined when the values for each color bar are referenced.

For more information regarding the register descriptions used during pattern generation, refer to the *Digital Page 0 Indirect Register* table in the 954 data sheet. Examples for pattern generation using the 954 and the 953 are found in Section 7.1.11 and Section 7.1.10, respectively.

Copyright © 2019, Texas Instruments Incorporated

6 Hardware Design

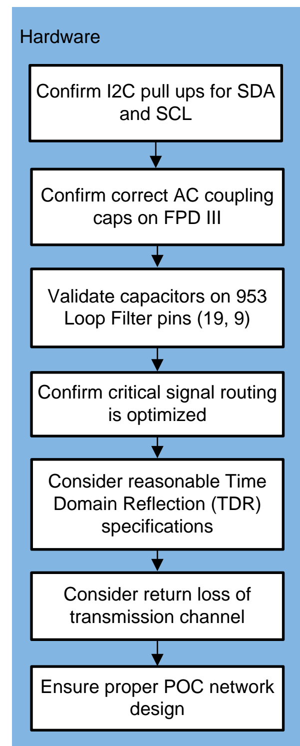

Figure 19. Hardware Design Flowchart

6.1 Basic I2C Connectors

The I^2C-compatible interface for each device consists of the clock (SCL) and data (SDA) pins. These two signals have open-drain I/Os. Both signals must be pulled up to VDD by an external resistor. A logic zero is transmitted by driving the output low. A logic one is transmitted by releasing the output and allowing it to be pulled up externally. The appropriate pullup resistor values will depend upon the total bus capacitance and operating speed.

Each serializer or deserializer determines its I^2C slave address from a resistor termination circuit attached to the IDX pin. This value is loaded into the I2C Device ID register at address 0x00. See the device data sheets for IDX configuration requirements and available I^2C addresses for each device.

6.1.1 I2C Pullups for SDA and SCL

The I2C SDA and SCL pins on the 954 and 953 are open-drain and require pullup resistors for operation. External 1-kΩ to 4.7-kΩ pullup resistors to the digital supply voltage—VDDIO on the 954 or VDDD on the 953—are recommended per I^2C interface standards. Note that I^2C communication to image sensors or microcontrollers may require different or the same digital supply voltages.

If the digital supply voltages are different between devices, I2C voltage level shifters are recommended to translate between any of the 1.8-V, 2.5-V, 3.3-V, and 5-V logical signal levels necessary for communication. For example, the TCA9406 is a 2-bit, bidirectional I2C and SMBus voltage-level translator with an output enable (OE) input. It is operational from 1.65 V to 3.6 V on the A-side—referenced to VCCA—and from 2.3 V to 5.5 V on the B-side—referenced to VCCB.

Copyright © 2019, Texas Instruments Incorporated

6.2 AC Capacitor on FPD3 Link

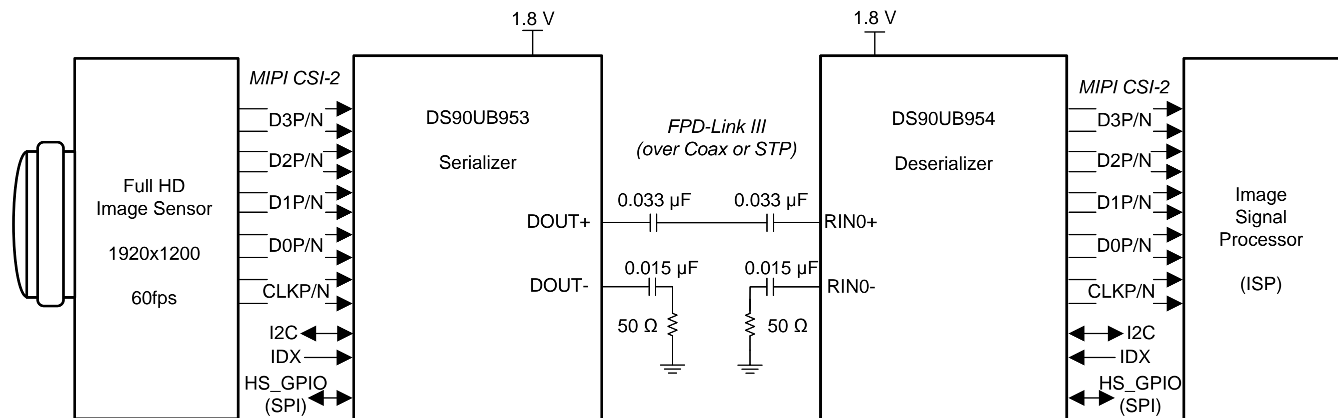

Figure 20. Simplified Block Diagram of DS90UB953-Q1/DS90UB954-Q1 System

The DS90UB953-Q1 FPD-Link III architecture is a 40-bit frame, high-speed synchronous back channel when communicating with 954 and 960. However, the mode that the device is operating in can change the data rate of the serial data output signal. For synchronous mode where the reference oscillator is provided by the deserializer, the serial data rate is 4 Gbps presented as a differential CML output on the DOUTP and DOUTN pins.

As shown in Figure 20, the signals at DOUTP and DOUTN must be AC-coupled with a series capacitor before the interconnect that leads to the deserializer. This AC-coupling capacitor will have a value of 0.033 µF when the device is running in 4-Gbps mode. At lower data rates, a larger coupling cap will be required. When connecting to a coax cable, the AC-coupling cap on the dummy side of the output (DOUTN) should also have an AC-coupling capacitor. This value should be half of the AC-coupling cap going to the cable (0.015 µF), and DOUTN should have a 50-Ω load on DOUTN. This load is needed because the AC and DC loads seen by DOUTP and DOUTN should be balanced. The capacitor on DOUTN should be half the value of the capacitor on DOUTP because the signal path must be balanced for the AC-coupling cap near the serializer and deserializer ends of the cable.

As mentioned previously, the AC capacitors will change if either the 953 or 954 are interfaced with lower data-rate parts such as 913A, 914A, 933, and 934. With a lower data-rate, the frequency band of interest is now lower in frequency and requires a larger capacitor.

6.3 Capacitance Used in Loop Filter

On the 953, there are two pins labeled LPF1 (9) and LPF2 (12), these are filter capacitors for two of the PLLs within the DS90UB953-Q1. LPF1 should have a 0.022-µF capacitor connected to the VDD_PLL pin (pin 11). The capacitor connected between LPF1 and VDDPLL should enclose as small of a loop as possible. LPF2 should have a 0.1-µF capacitor connecting the pin to GND. One of these PLLs generates the high-speed clock that is used for the serialization of the output, and the other is used for the PLL used in the CSI-2 receive port.

Noise coupled into these pins will degrade the performance of the PLLs in the DS90UB953-Q1, so the caps must be placed close to the pins that they are connected to, and the area of the loop enclosed must be minimized. Pin 10 is a bypass capacitor pin for the internal regulator for the PLL. The bypass capacitor can be mounted on the other side of the board so that the LPF1 capacitor can be placed very close to pins 9 and 11 with short connections.

6.4 Critical Signal Routing

Circuit board layout and stack-up for the FPD-Link III devices should be designed to provide a low-noise power feed to the device. It is also good layout practice to separate high-frequency or high-level inputs and outputs to minimize unwanted stray noise pickup, feedback, and interference.

Copyright © 2019, Texas Instruments Incorporated

The engineer can use thin dielectrics (2 to 4 mils) for power or ground sandwiches to improve power system performance. This arrangement provides plane capacitance for the PCB power system with low-inductance parasitics, which has proven to be especially effective at high frequencies, and makes the value and placement of external bypass capacitors less critical.

Some devices provide separate power and ground pins for different portions of the circuit. This is done to isolate switching noise effects between different sections of the circuit. Separate planes on the PCB are typically not required.

Pin Description tables typically provide guidance on which circuit blocks are connected to which power pin pairs. In some cases, an external filter may be used to provide clean power to sensitive circuits, such as PLLs. Use a four-layer board minimum with a power and ground plane.

Place the LVCMOS signals away from the differential lines to prevent coupling from the LVCMOS lines to the differential lines. Differential impedance of 100 Ω are typically recommended for STP interconnect and single-ended impedance of 50 Ω for coax interconnect. The closely coupled lines can help ensure that coupled noise will appear as common-mode, and thus is rejected by the receivers. The tightly coupled lines will also radiate less.

6.5 Time Domain Reflection

The FPD link signals coming from the DS90UB953-Q1 are carrying 4-Gbps data, and as a result, the traces must be controlled impedance transmission lines. If the traces are not controlled impedance, then the reflections generated are liable to cause problems with the LOCK.

Time domain reflection (TDR) measurements simply measure impedance running through a signal path. So long as the characteristic impedance is a good match to the 50-Ω coax cable, the trace length will not be a major factor. Connectors, components, and other items in the signal path can vary the impedance seen by the signal path.

6.6 Return Loss and Insertion Loss

When talking about loss, it is important to consider cables, the Power-over-Coax circuit on each side of the SERDES system, the connectors, the layout, and the AEQ settings.

Insertion Loss simply refers to the amount of loss that occurs when a signal is sent across the transmission line. We want the insertion loss to be as close to 0 dB as possible—which means minimal loss. The DS90UB954-Q1 has an adaptive equalizer at the input, which allows the device to compensate for various lengths of cable, provided the total attenuation stays within maximum allowable limits.

Return Loss is the loss of the signal power resulting from a discontinuity in the characteristic impedance of the transmission line. In other words, when signal is sent across the transmission channel, an amount of the signal will be reflected and returned back to the transmitter. It is most commonly encountered at the interface of one portion of the transmission line and another–such as the connector to a cable or board. Return Loss has been historically represented in dB. If the engineer wants to minimize the reflection of the signal, return loss must be as small as possible. This is usually represented by larger negative numbers in dB.

When discussing coax cables, it is important to note that a coax cable can be designed to be arbitrarily thin, and therefore low-cost and flexible. However, as the cable gets thinner, the losses increase. Inline Connectors also add loss to the signal. The frequency characteristics of inline connector loss differ from cables, and can often be considered as relatively flat across frequency. Connector manufacturers will specify the loss of their connectors, but typical FAKRA connectors have a loss of 0.1 to 0.2 dB per connector.

6.7 Power-over-Coax (PoC)

The Power-over-Coax (PoC) capability requires the use of circuitry following certain specifications connected to both ends of the cable. These specifications include impedance versus frequency characteristics and return loss. The PoC circuit must have low impedance at DC, but to not interfere with the data path, the characteristic impedance must be large over the band of the forward channel and back channel (10 MHz to 2 GHz) compared to the 50-Ω impedance of the coax line.

Copyright © 2019, Texas Instruments Incorporated

The critical consideration is the return loss profile after adding the PoC network. Limits on return loss make sure that the impedance of the PoC network is high enough that high-speed path signal integrity is maintained. The PoC network offers low impedance at DC to allow an efficient power transfer to the DC-DC powering the serializer and the camera, but also offers high impedance at the frequencies used for data transfer. Depending upon the current consumption of the camera and serializer board, there could be 200 to 500 mA of current going through the network. As a result, the engineer must check the inductor current limits before finalizing components to ensure that the PoC network components do not saturate at these current levels. A good starting point for the PoC Network to provide high impedance at the high frequencies is to use the recommended network in the EVM schematic.

For more information, see the *Power-over-Coax design guidelines for the DS90UB953-Q1* application note (SNLA272).

6.8 Voltage and Temperature Sensing

The DS90UB953-Q1 supports voltage measurement and temperature measurement. Temperature sensing will measure the internal temperature of the chip and GPI00 and GPIO1 pins are used for the voltage sensing. One of the usage of the voltage sensor is to monitor the PoC network on the board. Each GPIOs will have an individual setting for high and low voltage threshold. When the GPIOs voltage is outside the threshold range, an alarm bit can be set and sent to the deserializer. In a similar manner, the temperature sensor will trigger an alarm bit when the internal temperature of DS90UB953-Q1 is outside the range.

The measurement of the voltage and temperature will be performed in a round-robin manner. When the DS90UB953-Q1 reset state machine is in a normal mode, the sensor will start monitoring the voltage on GPIO0, GPIO1 pins, and internal temperature of the device. The monitoring will cycle every 1 ms per measurement, the reading will be stored in the register, and an alarm bit is set when the measured voltage or temperature is not within range set.

By default the sensor is disabled, and it can be enabled through a register setting. If one of the sensors is disabled, the round-robin will reduce to two sensor measurements and so forth. Registers regarding voltage and temperature sensing can be found in registers 0x15-0x1B.

NOTE: By default, the GPIO0/1 are used after power up for voltage sensing, while the GPIO2/3 are use for general-purpose signaling.

7 Appendix

7.1 Scripts

The scripts listed are BIST, Example Sensor Initialization, CSI Enable and Port Forwarding, Enabling CMLOUT FPD3 RX Port 0 on 954, Remote Enabled SER GPIO Toggle, Local SER GPIO Toggle, Internal FrameSync on 953 GPIO1, External FrameSync on 953 GPIO0, SER GPIOs as Inputs and Output to DES GPIO, Pattern Generation on the 953, Pattern Generation on the 954, Monitor Errors for Predetermined Time, 954 and 953 CSI Register Check, and Time Till Lock Script on 953.

7.1.1 BIST Script

```
##  BIST_953_954_WithForcedError.py
##
##  revision 1.1  June 13, 2017
##
########

######################
##
## This script file is used to check the link between the 953 and 954
##
## The Built In Self Test (BIST) generates a puesdo random sequence
## and checks for CRC errors between transmitted and received sequence.
##
```


Copyright © 2019, Texas Instruments Incorporated

```
## Rev. 1.1
## Added Parity Errors, restructured code
##
##
######################
print "\n\n"
import time


#    Define 954 and 953 Addresses
UB953 = 0x30                                                                    #953 SER Alias ID,
check 0x5C on 954 for confirmation
UB954 = 0x7A                                                                    #954 Device ID, check
0x00 on 954 for confirmation

#    Port selection, Passthrough, and aliasing
#board.WriteI2C(UB954, 0x4C, 0x01)                                               #Enable Port 0
writes
#board.WriteI2C(UB954, 0x58, 0x5E)                                               #I2C Pass through
Enabled and BC = 50Mbps
#board.WriteI2C(UB954, 0x5C, 0x18)                                               #953 Alias defined
as 0x18


#    Digital Reset except for registers
board.WriteI2C(UB953, 0x01, 0x01)                                                #Resets 953
time.sleep(0.5)
board.WriteI2C(UB954, 0x01, 0x01)                                                #Resets 954
time.sleep(0.5)

print ("Devices Reset")



#    Confirm Devices can communicate with each other
print ("954 Device ID (0x00):", hex(board.ReadI2C(UB954, 0x00, 1)))               #954 Device ID,
should be 0x7A, check 0x00 for confirmation
time.sleep(0.5)
print ("953 Device ID (0x00):", hex(board.ReadI2C(UB953, 0x00, 1)))               #953 Device ID,
should be 0x30, check 0x5B for confirmation

print ("------------------------------------------------------------------")
time.sleep(0.5)



#    Read Receiver Lock Status
print ("Receiver Lock Status (0x04):", hex(board.ReadI2C(UB954, 0x04, 0x1)))
    #0x04 is DEVICE_STS of 954
print ("Should read 0xCF")

print ("------------------------------------------------------------------")
time.sleep(1)



#    Enable write for Port0 of FPD3_PORT_SEL
board.WriteI2C(UB954, 0x4C, 0x01)                                                #0x4C is
FPD3_PORT_SEL



#    Clear Errors and Error Count
board.WriteI2C(UB953, 0x49, 0x28)                                                #0x49 is BC_CTRL.
0x28 selects BIST_CRC ERR CLR and CRC ERR CLR
```


Copyright © 2019, Texas Instruments Incorporated

```
print ("Read BCC Error Status (0x79):", hex(board.ReadI2C(UB953, 0x79, 1)))    #Clear possible
BCC_Error by Reading BCC Error Status
print ("Consult Register 0x79 on the SER for more information")

print ("------------------------------------------------------------------")

print ("Pre-
Error Link Status of 953 (0x52):", hex(board.ReadI2C(UB953, 0x52,1)))                    #0x52 is
GENERAL_STS of 953
print ("Should read 0x45 = RX Lock Detect, HS PLL Lock, Link Detect")

print ("------------------------------------------------------------------")

print ("BIST CRC Error count (0x54) on 953 before forced error.", hex(board.ReadI2C(UB953, 0x54,
1)))#0x54 is BIST ERR CNT

print ("------------------------------------------------------------------")

time.sleep(1)



#    Enabling BIST, Error, and Lock-Change Status
print ("Read BIST CTL register (0xB3) Before BIST ENABlED", hex(board.ReadI2C(UB954, 0xB3,
1)))        #0xB3 is BIST_CTL, bit 1 controls if enabled or not
print ("Should read 0x00 or 0x08\n")

board.WriteI2C(UB954, 0xB3, 0x01)                                            #Enable BIST using
BIST_CTL

print ("Read BIST CTL (0xB3) register After BIST ENABLED", hex(board.ReadI2C(UB954, 0xB3, 1)))
    #0xB3 is BIST_CTL
print ("Should read 0x01")

time.sleep(0.25)
print ("------------------------------------------------------------------")
print ("Read BIST Lock Status Change of 954 RIGHT AFTER BIST enabled (0x4D):",
hex(board.ReadI2C(UB954, 0x4D,1)))        #0x4D is RX_PORT_STS1 of 954
print ("Read to clear BIST enable Lock Status Change.")


board.WriteI2C(UB954, 0xD0, 0x01)                                            #Force 1 Error, 0xD0
is PORT_DEBUG register

#board.WriteI2C(UB954, 0xD0, 0x02)                                            #Force Continuous
errors, 0xD0 is PORT_DEBUG register
time.sleep(10)                                                                #Can run BIST for
as long as needed
#board.WriteI2C(UB954, 0xD0, 0x00)                                            #If forced
continuous errors, stop forcing errors

print ("Read Post-
BIST Lock Status Change of 954 RIGHT BEFORE BIST disabled (0x4D):", hex(board.ReadI2C(UB954,
0x4D,1)))        #0x4D is RX_PORT_STS1 of 954
print ("Should read 0x03, If lock status changed during BIST, will read 0x13")



#    Disable BIST and Port0
board.WriteI2C(UB954, 0xB3, 0x00)                                            #Disable BIST, using
BIST_CTL
board.WriteI2C(UB954, 0x4C, 0x00)                                            #0x4C is
FPD3_PORT_SEL

print ("------------------------------------------------------------------")
time.sleep(1)
```


Copyright © 2019, Texas Instruments Incorporated

```
#    Check if Error(s) occurred
print ("Post-
Error Link Status of 953 (0x52):", hex(board.ReadI2C(UB953, 0x52,1)))                    #0x52 is
GENERAL_STS of 953
print "Should read 0x4D = RX Lock Detect, HS PLL Lock, Link Detect, and BIST CRC Error"

print ("Receiver Lock Status (0x04):", hex(board.ReadI2C(UB954, 0x04, 0x1)))
    #0x04 is DEVICE_STS of 954
print ("Should read 0xCF")


print ("------------------------------------------------------------------")

print ("BIST CRC Error count (0x54) on 953.", hex(board.ReadI2C(UB953, 0x54, 1))) #0x54 is BIST
ERR CNT
print ("Parity Error count MSB (0x56) on 954.", hex(board.ReadI2C(UB954, 0x56, 1))) #0x56 is
number of Parity error 8 most significant bits
print ("Parity Error count LSB (0x55) on 954.", hex(board.ReadI2C(UB954, 0x55, 1))) #0x55 is
number of Parity error 8 least significant bits


print ("------------------------------------------------------------------")

#print ("953 Device ID (0x00):", hex(board.ReadI2C(UB953, 0x00, 1)))            #953 Device ID,
should be 0x30, check 0x5B for confirmation,
                                                                                #Usually use this
to see if 954 is stuck in BIST mode

#    Clear BIST Errors on 953
board.WriteI2C(UB953, 0x49, 0x28)                                            #0x49 is BC_CTRL.
0x28 selects BIST_CRC ERR CLR and CRC ERR CLR

print "\n\n"                                                                 #New line Printed
```

7.1.2 Example Sensor Initialization Script

```
##  SensorPowerOnAndReset.py
##
##  revision 1.0  6/13/17
##
########

import time

# Set up Variables
UB953 = 0x30        #could be alias or device ID depending if run locally or remotely


# Set GPIO2 and GPIO3 to outputs, where GPIO2 = RESET and GPIO3 = PWDN
board.WriteI2C(UB953,0x0E,0xC0)
# Set GPIO2 and GPIO3 to High - bring OVT10640 out of power down mode
board.WriteI2C(UB953,0x0D,0x0C)
time.sleep(0.1)
# Bring GPIO3 low to place 10640 in reset
board.WriteI2C(UB953,0x0D,0x08)
time.sleep(1)
# Bring GPIO3 high again to prepare 10640 for initialization
board.WriteI2C(UB953,0x0D,0x0C)
```


Copyright © 2019, Texas Instruments Incorporated

7.1.3 CSI Enable and Port Forwarding Script

```
##  CSI_EN_and_PortForwarding.py
##
##  revision 1.0 6/13/17
##
########


import time

devAddr = 0x7A

# Set CSI_EN and set to continuous clock mode
print  "CSI_EN"
board.WriteI2C(devAddr,0x33,0x03)
time.sleep(0.5)

# Set forwarding of RX port 0 to CSI output
print  "FWD_PORT0"
board.WriteI2C(devAddr,0x20,0x20)
time.sleep(0.5)
```

7.1.4 Enabling CMLOUT FPD3 RX Port 0 on 954

```
##  EnableCMLOUT_Port0_953.py
##
##  revision 1.0  6/13/17
##
########

import time


board.WriteReg(0xB0,0x14) # FPD3 RX Shared, page 0
board.WriteReg(0xB1,0x00) # Offset 0 (reg_0_sh)
board.WriteReg(0xB2,0x80) # Enable loop through driver
board.WriteReg(0xB1,0x02) #
board.WriteReg(0xB2,0x20) #
board.WriteReg(0xB1,0x03) #
board.WriteReg(0xB2,0x28) #
board.WriteReg(0xB1,0x04) #
board.WriteReg(0xB2,0x28) #
#
board.WriteReg(0xB0,0x18) #
board.WriteReg(0xB1,0x0F) #
board.WriteReg(0xB2,0x01) #
board.WriteReg(0xB1,0x10) #
board.WriteReg(0xB2,0x02) # Enable CML data output
```

7.1.5 Remote Enabled SER GPIO Toggle Script

```
##  RemoteEnabledSER_GPIOs_toggle.py
##
##  revision 1.0  6/13/17
##
########


#Makes 953 GPIO0-3 high and low by remote enabling them
```


Copyright © 2019, Texas Instruments Incorporated

```
print "\n"
import time


#    Define 954 and 953 Addresses
UB953 = 0x18                                        #953 SER Alias ID, check 0x5C on 954 for
confirmation
UB954 = 0x60                                        #954 Device ID, check 0x00 on 954 for
confirmation

#    Alias ID of SER
board.WriteI2C(UB954,0x5C,0x18)

#    Be sure to check Passthrough Settings!!! in 0x58, BCC CONFIG
#board.WriteI2C(UB954,0x58,0x58)

#    Port Select to enable writes to Port0
board.WriteI2C(UB954,0x4C,0x01)

#    Enable GPIO0-3 as outputs on 953
board.WriteI2C(UB953,0x0E,0xF0)

#    Remote Enable GPIO0-3 on 953
board.WriteI2C(UB953,0x0D,0xF0)


for y in range(0,300):
    board.WriteI2C(UB954,0x6E,0x88)        #Switch GPIO0 and GPIO1 to low or 0
    board.WriteI2C(UB954,0x6F,0x88)        #Switch GPIO2 and GPIO3 to low or 0
    time.sleep(0.0000024)

    board.WriteI2C(UB954,0x6E,0x99)        #Switch GPIO0 and GPIO1 to high or 1
    board.WriteI2C(UB954,0x6F,0x99)        #Switch GPIO2 and GPIO3 to high or 1
    time.sleep(0.0000024)
```

7.1.6 Local SER GPIO Toggle Script

```
##  LocalSER_GPIOs_toggle.py
##
##  revision 1.0  6/13/17
##
########


#Makes GPIO1 high and low by using sending and I2C Command across the BCC

print "\n"
import time


#    Define 954 and 953 Addresses
UB953 = 0x18                                        #953 SER Alias ID, check 0x5C on 954 for
confirmation
UB954 = 0x60                                        #954 Device ID, check 0x00 on 954 for
confirmation

#    Alias ID of SER
board.WriteI2C(UB954,0x5C,0x18)

#    Be sure to check Passthrough Settings!!! in 0x58, BCC CONFIG
#board.WriteI2C(UB954,0x58,0x58)

#    Port Select to enable writes to Port0
board.WriteI2C(UB954,0x4C,0x01)
```


Copyright © 2019, Texas Instruments Incorporated

```
#    Enable GPIO0-3 as outputs on 953
board.WriteI2C(UB953,0x0E,0xF0)


for y in range(0,300):
    board.WriteI2C(UB953,0x0D,0x00)        #Switch GPIO1 to low or 0
    time.sleep(0.0000024)


    board.WriteI2C(UB953,0x0D,0x02)        #Switch GPIO1 to high or 1
    time.sleep(0.0000024)
```

7.1.7 Internal FrameSync on 953 GPIO1

```
##   InternalFrameSync_GPIO1.py
##
##   revision 1.0   6/13/17
##
########


print "\n"
import time
#GPIO1 Internal Frame Sync on 953

#    Define 954 and 953 Addresses
UB953 = 0x18                    #953 SER Alias ID, check 0x5C on 954 for confirmation
UB954 = 0x60                    #954 Device ID, check 0x00 on 954 for confirmation

#    Alias ID of SER
board.WriteI2C(UB954,0x5C,0x18)

#    Port Select to enable writes
board.WriteI2C(UB954,0x4C,0x01)

#    Enable GPIO0-3 as outputs on 953
board.WriteI2C(UB953,0x0E,0xF0)

#    Enable Remote of GPIO0 and GPIO1 from DES
board.WriteI2C(UB953,0x0D,0xF0)

#    BC GPIO Control: Link BC GPIO0-1 (SER GPIO0 and GPIO1) to Frame Sync
board.WriteI2C(UB954,0x6E,0xAA)

#    FS CTL: Enable Frame Sync and set Internal Frame Sync for Port0
board.WriteI2C(UB954,0x18,0x01)

###### Create an aproximate 400kHz square wave for Frame Sync
#    High Time
board.WriteI2C(UB954,0x1A,0x01)

#    Low Time
board.WriteI2C(UB954,0x1C,0x01)
```

7.1.8 External FrameSync on 953 GPIO0

```
##   ExternalFrameSyncGPIO0.py
##
##   revision 1.0   6/13/17
##
########
```


Copyright © 2019, Texas Instruments Incorporated

```
print "\n"
import time
#GPIO0 External Frame Sync on 953
#Note: Script assumes Frame Sync Signal is applied to GPIO0 on DES

#    Define 954 and 953 Addresses
UB953 = 0x18                        #953 SER Alias ID, check 0x5C on 954 for confirmation
UB954 = 0x60                        #954 Device ID, check 0x00 on 954 for confirmation

#    Alias ID of SER
board.WriteI2C(UB954,0x5C,0x18)

#    Port Select to enable writes
board.WriteI2C(UB954,0x4C,0x01)

#    Enable GPIO0-3 as outputs on 953
board.WriteI2C(UB953,0x0E,0xF0)

#    Enable Remote of GPIO0-3 from DES
board.WriteI2C(UB953,0x0D,0xF0)

#    BC GPIO Control: Link BC GPIO0 and GPIO1 to DES GPIO0
board.WriteI2C(UB954,0x6E,0x00)

#    GPIO Input CTRL: Make all DES GPIOs Inputs
board.WriteI2C(UB954,0x0F,0x7F)

#    FS CTL: EN FS and state External Frame Sync comes from GPIO0
board.WriteI2C(UB954,0x18,0x81)
```

7.1.9 SER GPIOs as Inputs and Output to DES GPIO

```
##  Sensing953GPIOsTo954GPIO.py
##
##  revision 1.0 6/13/17
##
########


# Use 953 GPIO0 as input that can be fed to DES GPIO0

import time

UB954 = 0x60
UB953ID = 0x30
UB953 = 0x18

#    Set up Port0
board.WriteI2C(UB954, 0x4C, 0x01)    #Select Port 0

#    Set up Back Channel Config (0x58)
#board.WriteI2C(UB954,0x58,0x5E)        #50Mbps BC and Pass through on

#    Set up SER ID
#board.WriteI2C(UB954,0x5B,UB953ID)
#    Set up SER Alias ID
board.WriteI2C(UB954,0x5C,UB953)    #Set up SER alias


#    Set SER GPIO NOT as outputs
board.WriteI2C(UB953, 0x0D, 0x00)    #no remote enable, no outputs
#    Set SER GPIO0 as Input
board.WriteI2C(UB953, 0x0E, 0x01)    #Only GPIO0 is output
#    Set FC CTL for GPIO forwarding
```


Copyright © 2019, Texas Instruments Incorporated

```
board.WriteI2C(UB953, 0x33, 0x05)    #One GPIO forwarded and DCA CRC Enabled (by default)
                                    #Note, if you want to use forwarding for more GPIOs use
convention below
                                    #One GPIO = GPIO0 only
                                    #Two GPIOs = GPIO0 and GPIO1 only
                                    #Four GPIOs = GPIO0-3 sent

#    Make all DES GPIOs input disabled
board.WriteI2C(UB954, 0x0F, 0x00)    #GPIO0-6 Input Disabled
#    Make DES GPIO0 linked to RX0 and GPIO0
board.WriteI2C(UB954, 0x10, 0x01)    #0x10[4:2] = 000: RX Port 0
                                    #0x10[7:5] = 000: received GPIO0-3
                                    #0x10[1] = 0: Output value of GPIO0
                                    #0x10[0] = 1: GPIO0 Output Enabled
```

7.1.10 Pattern Generation on the 953 Script

```
##  PatternGenerationOn953.py
##
##  revision 1.0
##
########


import time

# 953 config
runTime = 3

board.devAddr = 0x30

# # CSI sel and CSI enable
# board.WriteReg(0x32, 0x01)
# time.sleep(0.5)
# board.WriteReg(0x33, 0x01)
# time.sleep(0.5)

# enable pat gen
board.WriteReg(0xB0, 0x00)
board.WriteReg(0xB1, 0x01)
board.WriteReg(0xB2, 0x01) #enable pattern generator

board.WriteReg(0xB1, 0x02)
board.WriteReg(0xB2, 0xB3) #fixed color pattern, 8 color bars, block size of 5

board.WriteReg(0xB1, 0x03)
board.WriteReg(0xB2, 0x24) #CSI Data Identifier (0x24 = RGB888, 0x2C = RAW12, 0x2B = RAW10)

board.WriteReg(0xB1, 0x04)
board.WriteReg(0xB2, 0x16) #line size (15:8)

board.WriteReg(0xB1, 0x05)
board.WriteReg(0xB2, 0x80) #line size (7:0)

board.WriteReg(0xB1, 0x06)
board.WriteReg(0xB2, 0x02) #bar size (15:8)

board.WriteReg(0xB1, 0x07)
board.WriteReg(0xB2, 0xd0) #bar size (7:0)

board.WriteReg(0xB1, 0x08)
board.WriteReg(0xB2, 0x04) #active lines per frame (15:8)

board.WriteReg(0xB1, 0x09)
board.WriteReg(0xB2, 0x38) #active lines per frame (7:0)
```


Copyright © 2019, Texas Instruments Incorporated

```
board.WriteReg(0xB1, 0x0a)
board.WriteReg(0xB2, 0x04) #total lines per frame (15:8)

board.WriteReg(0xB1, 0x0b)
board.WriteReg(0xB2, 0x65) #total lines per frame (7:0)

board.WriteReg(0xB1, 0x0c)
board.WriteReg(0xB2, 0x0B) #line period (15:8)

board.WriteReg(0xB1, 0x0d)
board.WriteReg(0xB2, 0x93) #line period (7:0)

board.WriteReg(0xB1, 0x0e)
board.WriteReg(0xB2, 0x21) #vertical back porch

board.WriteReg(0xB1, 0x0f)
board.WriteReg(0xB2, 0x0a) #vertical front porch

#
board.WriteReg(0xB1, 0x10)
board.WriteReg(0xB2, 0x00) #1st byte of fixed color

board.WriteReg(0xB1, 0x11)
board.WriteReg(0xB2, 0x00) #2nd byte of fixed color

board.WriteReg(0xB1, 0x12)
board.WriteReg(0xB2, 0xFF) #3rd byte of fixed color

# board.WriteReg(0xB1, 0x13)
# board.WriteReg(0xB2, 0xff) #4th byte of fixed color

# board.WriteReg(0xB1, 0x14)
# board.WriteReg(0xB2, 0xff) #5th byte of fixed color

# board.WriteReg(0xB1, 0x15)
# board.WriteReg(0xB2, 0x00) #6th byte of fixed color

# board.WriteReg(0xB1, 0x16)
# board.WriteReg(0xB2, 0x00) #7th byte of fixed color

# board.WriteReg(0xB1, 0x17)
# board.WriteReg(0xB2, 0x0f) #8th byte of fixed color

# board.WriteReg(0xB1, 0x18)
# board.WriteReg(0xB2, 0xf0) #9th byte of fixed color

# board.WriteReg(0xB1, 0x19)
# board.WriteReg(0xB2, 0x00) #10th byte of fixed color

# board.WriteReg(0xB1, 0x1A)
# board.WriteReg(0xB2, 0x00) #11th byte of fixed color

# board.WriteReg(0xB1, 0x1B)
# board.WriteReg(0xB1, 0x3f) #12th byte of fixed color

# board.WriteReg(0xB1, 0x1C)
# board.WriteReg(0xB2, 0xc0) #13th byte of fixed color

# board.WriteReg(0xB1, 0x1D)
# board.WriteReg(0xB2, 0x00) #14th byte of fixed color

# board.WriteReg(0xB1, 0x1E)
# board.WriteReg(0xB2, 0x00) #15th byte of fixed color

# board.WriteReg(0xB1, 0x1F)
# board.WriteReg(0xB2, 0x00) #16th byte of fixed color
```


Copyright © 2019, Texas Instruments Incorporated

7.1.11 Pattern Generation on the 954 Script

```
##  PatternGenerationOn954.py
##
##  revision 1.0
##
########



import time

board.devAddr = 0x7A

# CSI sel and CSI enable
board.WriteReg(0x32, 0x01)
time.sleep(0.5)
board.WriteReg(0x33, 0x01)
time.sleep(0.5)

# enable pat gen
board.WriteReg(0xB0, 0x00)
board.WriteReg(0xB1, 0x01)
board.WriteReg(0xB2, 0x01) #enable pattern generator

board.WriteReg(0xB1, 0x02)
board.WriteReg(0xB2, 0xB9) #fixed color pattern, 8 color bars, block size of 9

board.WriteReg(0xB1, 0x03)
board.WriteReg(0xB2, 0x2C) #CSI Data Identifier (0x24 = RGB888, 0x2C = 12-bit)

board.WriteReg(0xB1, 0x04)
board.WriteReg(0xB2, 0x16) #line size (15:8)

board.WriteReg(0xB1, 0x05)
board.WriteReg(0xB2, 0x80) #line size (7:0)

board.WriteReg(0xB1, 0x06)
board.WriteReg(0xB2, 0x02) #bar size (15:8)

board.WriteReg(0xB1, 0x07)
board.WriteReg(0xB2, 0xd0) #bar size (7:0)

board.WriteReg(0xB1, 0x08)
board.WriteReg(0xB2, 0x02) #active lines per frame (15:8)

board.WriteReg(0xB1, 0x09)
board.WriteReg(0xB2, 0xd0) #active lines per frame (7:0)

board.WriteReg(0xB1, 0x0a)
board.WriteReg(0xB2, 0x04) #total lines per frame (15:8)

board.WriteReg(0xB1, 0x0b)
board.WriteReg(0xB2, 0x1a) #total lines per frame (7:0)

board.WriteReg(0xB1, 0x0c)
board.WriteReg(0xB2, 0x0c) #line period (15:8)

board.WriteReg(0xB1, 0x0d)
board.WriteReg(0xB2, 0x67) #line period (7:0)

board.WriteReg(0xB1, 0x0e)
board.WriteReg(0xB2, 0x21) #vertical back porch

board.WriteReg(0xB1, 0x0f)
```


Copyright © 2019, Texas Instruments Incorporated

```
board.WriteReg(0xB2, 0x0a) #vertical front porch

#0xff0, 0x000, 0x000, 0xff0, 0x000, 0x000
board.WriteReg(0xB1, 0x10)
board.WriteReg(0xB2, 0xff) #1st byte of fixed color

board.WriteReg(0xB1, 0x11)
board.WriteReg(0xB2, 0x00) #2nd byte of fixed color

board.WriteReg(0xB1, 0x12)
board.WriteReg(0xB2, 0x00) #3rd byte of fixed color

board.WriteReg(0xB1, 0x13)
board.WriteReg(0xB2, 0x00) #4th byte of fixed color

board.WriteReg(0xB1, 0x14)
board.WriteReg(0xB2, 0xff) #5th byte of fixed color

board.WriteReg(0xB1, 0x15)
board.WriteReg(0xB2, 0x00) #6th byte of fixed color

board.WriteReg(0xB1, 0x16)
board.WriteReg(0xB2, 0x00) #7th byte of fixed color

board.WriteReg(0xB1, 0x17)
board.WriteReg(0xB2, 0x00) #8th byte of fixed color

board.WriteReg(0xB1, 0x18)
board.WriteReg(0xB2, 0x00) #9th byte of fixed color

# board.WriteReg(0xB1, 0x19)
# board.WriteReg(0xB2, 0x00) #10th byte of fixed color

# board.WriteReg(0xB1, 0x1A)
# board.WriteReg(0xB2, 0x00) #11th byte of fixed color

# board.WriteReg(0xB1, 0x1B)
# board.WriteReg(0xB1, 0x00) #12th byte of fixed color

# board.WriteReg(0xB1, 0x1C)
# board.WriteReg(0xB2, 0x00) #13th byte of fixed color

# board.WriteReg(0xB1, 0x1D)
# board.WriteReg(0xB2, 0x00) #14th byte of fixed color

# board.WriteReg(0xB1, 0x1E)
# board.WriteReg(0xB2, 0x00) #15th byte of fixed color

# board.WriteReg(0xB1, 0x1F)
# board.WriteReg(0xB2, 0x00) #16th byte of fixed color
```

7.1.12 Monitor Errors for Predetermined Time Script

```
##  953_954_Screen_CRC_Parity_CSI_InTime.py
##
##  revision 1.0  6/13/17
##
########

import time

devAddr_954 = 0x7A
devAddr_953 = 0x18
```


Copyright © 2019, Texas Instruments Incorporated

```
waitTime = 10        #Note this time is in seconds

print  "\n"
print  "devAddr_954 = 0x7A"
print  "devAddr_953 = 0x18"

#    Clear previous errors

board.ReadI2C(devAddr_954,0x4D)
time.sleep(0.1)
board.ReadI2C(devAddr_954,0x55)
time.sleep(0.1)
board.ReadI2C(devAddr_954,0x56)
time.sleep(0.1)
board.ReadI2C(devAddr_954,0x7A)
time.sleep(0.1)
board.ReadI2C(devAddr_954,0x7B)
time.sleep(0.1)

board.WriteI2C(devAddr_953,0x49,0x08)
time.sleep(0.1)
board.ReadI2C(devAddr_953,0x79)
time.sleep(0.1)
board.ReadI2C(devAddr_953,0x55)
time.sleep(0.1)
board.ReadI2C(devAddr_953,0x56)
time.sleep(0.1)
board.ReadI2C(devAddr_953,0x5C)
time.sleep(0.1)
board.ReadI2C(devAddr_953,0x5D)
time.sleep(0.1)

#    Program time to wait
print "Wait ", waitTime, " seconds"
time.sleep(waitTime)


#    Print Registers
print "BC Status:", hex(board.ReadI2C(devAddr_953,0x52))
time.sleep(0.1)
print "CRC Error HI:", hex(board.ReadI2C(devAddr_953,0x56))
time.sleep(0.1)
print "CRC Error LO:", hex(board.ReadI2C(devAddr_953,0x55))
time.sleep(0.1)

print "FC Status:", hex(board.ReadI2C(devAddr_954,0x4D))
time.sleep(0.1)
print "Parity Error HI:", hex(board.ReadI2C(devAddr_954,0x55))
time.sleep(0.1)
print "Parity Error LO:", hex(board.ReadI2C(devAddr_954,0x56))
time.sleep(0.1)

print "953 CSI Status:", hex(board.ReadI2C(devAddr_953,0x5D))
time.sleep(0.1)

print "954 CSI Status:", hex(board.ReadI2C(devAddr_954,0x7A))
time.sleep(0.1)

print "CSI Error = ", hex(board.ReadI2C(devAddr_953, 0x5C, 1))
time.sleep(0.1)
print "Packet Header data = ", hex(board.ReadI2C(devAddr_953, 0x61, 1))
time.sleep(0.1)
print "Packet Header Word Count 0 = ", hex(board.ReadI2C(devAddr_953, 0x62, 1))
time.sleep(0.1)
print "Packet Header Word Count 1 = ", hex(board.ReadI2C(devAddr_953, 0x63, 1))
time.sleep(0.1)
```


Copyright © 2019, Texas Instruments Incorporated

```
#    Clear Errors after test
board.ReadI2C(devAddr_954,0x4D)
time.sleep(0.1)
board.ReadI2C(devAddr_954,0x55)
time.sleep(0.1)
board.ReadI2C(devAddr_954,0x56)
time.sleep(0.1)
board.ReadI2C(devAddr_954,0x7A)
time.sleep(0.1)
board.ReadI2C(devAddr_954,0x7B)
time.sleep(0.1)

board.WriteI2C(devAddr_953,0x49,0x08)
time.sleep(0.1)
board.ReadI2C(devAddr_953,0x79)
time.sleep(0.1)
board.ReadI2C(devAddr_953,0x55)
time.sleep(0.1)
board.ReadI2C(devAddr_953,0x56)
time.sleep(0.1)
board.ReadI2C(devAddr_953,0x5C)
time.sleep(0.1)
board.ReadI2C(devAddr_953,0x5D)
time.sleep(0.1)
```

7.1.13 954 and 953 CSI Register Check Script

```
##  954_953_CSI_RegisterCheck.py
##
##  revision 1.0  6/13/17
##
########


print "\n\n"

SERalias = 0x18
DES = 0x7A

print("954 0x04: Device Status", hex(board.ReadI2C(DES, 0x04, 1)))
print("954 0x4D: RX Port Status", hex(board.ReadI2C(DES, 0x4D, 1)))
print("954 0x72: Map for Virtual Channel ID", hex(board.ReadI2C(DES, 0x72, 1)))
print("954 0x7A: CSI_RX_STS for Length and checksum errors", hex(board.ReadI2C(DES, 0x7A, 1)))
print("954 0x7B: CSI_ERR_COUNTER", hex(board.ReadI2C(DES, 0x7B, 1)))
print("954 0x7C: Port Config 2", hex(board.ReadI2C(DES, 0x7C, 1)))
print("954 0x33: CSI CTL", hex(board.ReadI2C(DES, 0x5F, 1)))
print("954 0x34: CSI STS", hex(board.ReadI2C(DES, 0x60, 1)))
print("954 0x4E: RX PORT STS2, Line Count Change, and CSI Error", hex(board.ReadI2C(DES, 0x4E,
1)))
print("954 0x55: RX Parity Error HI", hex(board.ReadI2C(DES, 0x55, 1)))
print("954 0x56: RX Parity Error LO", hex(board.ReadI2C(DES, 0x56, 1)))

print("------------------------------------------------")

print("953 0x52: General Status", hex(board.ReadI2C(SERalias, 0x52, 1)))
print("953 0x5C: CSI_ERR_CNT", hex(board.ReadI2C(SERalias, 0x5C, 1)))
print("953 0x5D: CSI Error Status, check Length and Checksum errors", hex(board.ReadI2C(SERalias,
0x5D, 1)))
print("953 0x5E: CSI Error Data lanes 0 and 1", hex(board.ReadI2C(SERalias, 0x5E, 1)))
print("953 0x5F: CSI Error Data Lanes 2 and 3", hex(board.ReadI2C(SERalias, 0x5F, 1)))
print("953 0x60: CSI Error Clock Lane", hex(board.ReadI2C(SERalias, 0x60, 1)))
print("953 0x61: CSI Packet Header Data", hex(board.ReadI2C(SERalias, 0x61, 1)))
print("953 0x62: CSI Packet Header Word Count LSB", hex(board.ReadI2C(SERalias, 0x62, 1)))
print("953 0x63: CSI Packet Header Word Count MSB", hex(board.ReadI2C(SERalias, 0x63, 1)))
print("953 0x64: CSI ECC, bit 7 does line length Change", hex(board.ReadI2C(SERalias, 0x64, 1)))
```


Copyright © 2019, Texas Instruments Incorporated

```
print("953 0x79: BCC Status: Types of BCC Errors", hex(board.ReadI2C(SERalias, 0x79, 1)))
print("953 0x56: CRC Error count: MSB", hex(board.ReadI2C(SERalias, 0x56, 1)))
print("953 0x55: CRC Error count: LSB", hex(board.ReadI2C(SERalias, 0x56, 1)))
```

7.1.14 Time Till Lock Script on 953

```
##  TimeTillLock_953.py
##
##  revision 1.0  6/13/17
##
########


#Note, run this script locally on 953 before powering on the 953. Then power on the 953 within 7
seconds
#Resolution of timing can be tweaked using the resolution variable (in number of seconds)

########################################################
#    Variables
import time

UB953ID = 0x30

resolution = 0.1

temp = 0x00
count = 0

pwd = 0x00
temp2 = 0

print "\n"

########################################################
#    This waits 7 seconds for 953 to be powered on until continuing with script

while ((pwd & 0x40) != 0x40):
    pwd = board.ReadI2C(UB953ID, 0x51, 1)
    time.sleep(resolution)
    temp2 = temp2 + resolution
    if temp2 > 7:
        pwd = 0x40
        print "Timed out"


#########################################################
#    Reset can be used to simulate powering up the system

#time.sleep(1.5)

#board.WriteI2C(UB953ID,0x01,0x01)    # Digital Reset without clearing registers
#board.WriteI2C(UB953ID,0x01,0x02)    # Digital Reset with clearing registers


##############################################################
#    Prints general status for 35 seconds or until lock occurs

while ((temp & 0x05) != 0x05):
    temp = board.ReadI2C(UB953ID,0x52,1)
    print "General Status = ", hex(temp)
    time.sleep(resolution)
    count = count + resolution
    if count > 35:
        temp = 0x05
        print "time out waiting for lock"
```


Copyright © 2019, Texas Instruments Incorporated

```
print "Lock established in approximately ", count, " seconds."

print "\n"
```

7.2 Acknowledgments

The authors, Cole Macias and Mandeep Singh, would like to thank Ramsin Ziazadeh, Justin Prayogo, Tattiana Davenport, Jason Blackman, George Alphonse, Liam Keese, Dave Lewis, Andy McLean, Zoe Nuyens, Davor Glisic, and Manickam Palaniappan for their technical contributions and guidance during the creation of this design guide.

Copyright © 2019, Texas Instruments Incorporated

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Original (March 2019) to A Revision **Page**

- Changed DS90UB935-Q1 to DS90UB953-Q1 in simplified block diagram graphic 3
- Fixed typo in flowchart graphic 22

Copyright © 2019, Texas Instruments Incorporated

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265

Copyright © 2022, Texas Instruments Incorporated