

Digital Design and Verification of I3C protocol

Omar Mohamed Rizk, Salah-Eldin Attia Sayed, Youssef Soliman Mohammed,
Abd-El Rahman Mohammed Kamal, Marina Mounir Fouad

Electronics and Communications Engineering Department Faculty of Engineering, Helwan
University

July 2025



4th Year of Electronics and Communication Engineering Department

Design and Verification of I3C protocol

Names:

- Marina Mounir Fouad Shamandi
- Salah Eldin Attia Sayed
- Youssef Soliman Mohammed Fathallah
- Omar Mohamed Rizk Ibrahim
- Abd El-Rahman Mohamed Kamal El-Din

Supervised By

Dr. Mohamed EL-Dakroury

Eng. Abdulkareem Mohamed

Eng: Ahmed Abdelsalam

Sponsored By

ST Microelectronics

Declaration

We hereby certify that this project submitted as part of our partial fulfillment of BSc in Electronics and Communications Engineering is entirely our own work, that we have exercised reasonable care to ensure its originality and does not to the best of our knowledge breach any copyrighted materials and have not been taken from the work of others and to the extent that such work has been cited and acknowledged within the text of our work. Signed

Acknowledgment

All praise for ALLAH, our only lord and ruler. If it weren't for ALLAH's help and mercy, we wouldn't be able to make it this far into the Project.

We thank our supervisor, Dr. Mohamed EL-Dakroury, for encouraging us throughout the year and for his support and guidance.

Thanks to our mentors in STMicroelectronics Inc, Eng.Abdulkareem Mohamed, Eng. Ahmed Abd-Elsalam for their dedication and guidance. Also, we would like to thank Dr. Ahmed Salah for providing us with the required Hardware kit for completing our project.

Finally, we would like to express our gratitude to our parents and close friends. Without their understanding and encouragement in the past few years, it would have been impossible for us to complete our studies.

Abstract

This thesis presents the design and verification of the MIPI I3C protocol for STMicroelectronics microcontrollers (MCUs), focusing on a complete functional implementation and validation of the protocol. The I3C protocol is a modern serial communication standard that integrates the advantages of I²C and SPI, offering higher speed, improved power efficiency, and dynamic address assignment, making it well-suited for embedded systems and MCU-based applications.

The project involves the design, integration, and verification of an I3C Controller that facilitates efficient communication between I3C Targets and Controllers, ensuring data integrity, arbitration handling, and error detection. The design is implemented in Verilog HDL and verified through an extensive testbench environment, incorporating APB-based register access, private messaging, asynchronous FIFOs, and frame management. Functional validation is conducted using simulation and FPGA-based prototyping, ensuring compliance with STMicroelectronics' MCU requirements.

Verification is performed using a comprehensive testbench, including APB register read/write operations, private message transactions, and real-time data exchanges.

The implementation is evaluated based on protocol compliance, timing performance, and resource utilization, ensuring the design meets the specifications of STMicroelectronics' MCU communication frameworks.

The results of this thesis demonstrate a fully functional I3C Controller, successfully Integrated and verified for MCU-based applications, providing a reliable and efficient communication solution. This work contributes to the advancement of I3C Protocol implementations and lays the foundation for future enhancements and optimizations in embedded system designs.

Contents

Chapter 1: Introduction	19
1.1 Communication Protocols.....	19
1.2 From I ² C to I3C.....	19
1.3 Motivation for MIPI I3C	20
1.4 MIPI I3C Main System.....	22
1.5 Introduction to Push-Pull Based Communication	23
1.5.1 Open Drain Circuit.....	23
1.5.2 Push-Pull Circuit	24
1.5.3 Keeper Circuits.....	26
Chapter 2: MIPI I3C Protocol.....	27
2.1 MIPI I3C Message Types.....	27
2.2 MIPI I3C Communication Modes.....	28
2.2.1 I3C Single Data Rate (SDR Mode)	28
2.2.2 I3C High Data Rate (HDR Mode).....	28
2.3 Bus Configurations	29
2.3.1 Device Roles.....	29
2.3.2 Point to Point Communication	33
2.3.3 MIPI I3C Device Characteristics.....	33
2.4 Bus Conditions	35
2.5 Bus Communication	36
2.5.1 I3C Message Elements	36
2.5.2 SDR Message Types	38
2.5.3 Role of I3C Target.....	39
2.5.4 I3C Address Header.....	40
2.6 SDR Data Word	45
2.6.1 Handoff From Address ACK to SDR Controller Write Data	45
2.6.2 Ninth Bit of SDR Controller Written Data As Parity	46
2.6.3 Ninth Bit of SDR Target Returned (Read) Data as End-of-Data.....	47
Chapter 3: Main Features	48
3.1 Dynamic Address Assignment	48
3.1.1 Data Given From The Host	48
3.1.2 Address Assignment Rules.....	49
3.1.3 Dynamic address assignment procedure.....	49
3.1.4 ENTDAAs Frame Format	51
3.1.5 Dynamic Address Assignment Error Case and Handling	51
3.2 Hot-Join Mechanism	52
3.2.1 Hot-Join Flow Procedure.....	52
3.2.2 Hot-Join Frame Format.....	53
3.2.3 Hot-Join Target Requirements.....	53
3.3 In-Band Interrupt.....	55
3.3.1 Priority level.....	55
3.3.2 Interrupt request	55
3.3.3 Mandatory Data Byte (MBD).....	56
3.3.4 IN-Band Interrupt Frame Format.....	56
3.4 Common Command Codes (CCCs).....	57
3.4.1 Enter Activity State 0-3 (ENTAS0-ENTAS3).....	57
3.4.2 Get Device Status (GETSTATUS)	58
3.4.3 Target Reset Action (RSTACT)	59

3.5 I²C Legacy Communication	63
3.5.1 I ² C Legacy Frame Format.....	63
63	
Chapter 4: STM I3C System Architecture	64
4.1 APB Interface Block	65
4.2 Registers.....	65
4.2.1 I3C message control register (I3C_CR).....	65
4.2.2 I3C message control register [alternate] (I3C_CR)	69
4.2.3 I3C configuration register (I3C_CFGR)	70
4.2.4 I3C receives data byte register (I3C_RDR)	75
4.2.5 I3C transmit data byte register (I3C_TDR)	76
4.2.6 I3C event register (I3C_EVR)	76
4.2.7 I3C own device characteristics register (I3C_DEVR0)	81
4.3 I3C Kernel	83
4.4 I3C Bus Interface	84
Chapter 5: STM I3C Controller Design	85
5.1 SCL Generation	85
5.2 Frame Counter	86
5.3 Timer	88
5.4 SDA Interface	90
5.5 FSM Block	94
5.5.2 Block I/Os.....	Error! Bookmark not defined.
5.5.3 FSM.....	Error! Bookmark not defined.
5.6 I ² C Legacy Mode.....	Error! Bookmark not defined.
5.7 Dynamic Address Assignment.....	Error! Bookmark not defined.
5.8 Hot-Join Management	Error! Bookmark not defined.
5.9 In-Band Interrupt Management.....	Error! Bookmark not defined.
5.10 Controller Role Request Management.....	Error! Bookmark not defined.
5.11 Timer	Error! Bookmark not defined.
5.12 I3C Engine.....	Error! Bookmark not defined.
Chapter 5.....	95
6 MIPI I3C Controller Implementation.....	95
6.5 FPGA Board Features	95
6.6 Timing Constraints.....	96
6.7 Vivado Xilinx – AMD FPGA Flow	100
6.8 FPGA Debugging	107
6.9 How to choose the perfect FPGA for your Design?	110
References.....	113

List of Figures

FIGURE 1 TYPICAL IOT DEVICE ARCHITECTURE	20
FIGURE 2 I3C MODES VS I ² C EFFECTIVE ENERGY RANGE PER 1KB (MJ).....	21
FIGURE 3 I3C MODES VS I2C EFFECTIVE BITRATES FOR 12.5 MHZ CLOCK (MBPS)	21
FIGURE 4 MIPI I3C SYSTEM DIAGRAM.....	22
FIGURE 5 OPEN-DRAIN CIRCUIT	23
FIGURE 6 PUSH DOWN CASE IN OPEN DRAIN CIRCUITS.....	23
FIGURE 7 PULLING UP CASE IN OPEN DRAIN CIRCUITS	24
FIGURE 8 PUSH-PULL CIRCUIT.....	24
FIGURE 9 PUSH DOWN CASE IN PUSH-PULL CIRCUITS.....	24
FIGURE 10 PULLING DOWN CASE IN PUSH-PULL CIRCUITS.....	25
FIGURE 11 PUSH-PULL DISCHARGING PROBLEM	25
FIGURE 12 KEEPER CIRCUIT	26
FIGURE 13 MIPI I3C BUS BLOCK DIAGRAM.....	31
FIGURE 14 SHARED PERIPHERAL DEVICE	31
FIGURE 15 I3C SECONDARY CONTROLLER	32
FIGURE 16 I3C BRIDGING AND ROUTING DEVICES	32
FIGURE 17 POINT TO POINT CONNECTION.....	33
FIGURE 18 MIPI I3C BUS CONDITIONS	35
FIGURE 19 I2C MESSAGE	36
FIGURE 20 START CONDITION	36
FIGURE 21 STOP CONDITION	36
FIGURE 22 T-BIT WHEN TARGET ENDS READ AND CONTROLLER GENERATES STOP.....	37
FIGURE 23 T- BIT WHEN TARGET AND CONTROLLER AGREE TO CONTINUE READ MESSAGE	37
FIGURE 24 I3C MESSAGE TYPES.....	38
FIGURE 25 IN-BAND INTERRUPT REQUEST FRAME	41
FIGURE 26 CONTROLLER ROLE REQUEST FRAME	41
FIGURE 27 HOT-JOIN REQUEST	41
FIGURE 28 PRIVATE WRITE BY I3C CONTROLLER VS IBI REQUEST BY I3C TARGET	42
FIGURE 29 PRIVATE READ BY I3C CONTROLLER VS CONTROLLER ROLE REQUEST BY I3C TARGE.....	42
FIGURE 30 PRIVATE WRITE BY I3C CONTROLLER VS CONTROLLER ROLE REQUEST BY I3C TARGET	43
FIGURE 31 PRIVATE READ BY I3C CONTROLLER VS IBI REQUEST BY I3C TARGET	43
FIGURE 32 TRANSITION FROM ADDRESS ACK TO MANDATORY BYTE DURING IBI	45
FIGURE 33 PARITY BIT USING X OR	46
FIGURE 34 DYNAMIC ADDRESSES AVAILABLE FOR USE.....	49
FIGURE 35 BROADCAST WRITE FOR DAA.....	49
FIGURE 36 ENTDAACCC	49
FIGURE 37 BROADCAST READ FOR TARGET DATA.....	50
FIGURE 38 TARGET'S 8 BYTES DATA.....	50
FIGURE 39 DYNAMIC ADDRESS ASSIGNED WITH ODD PARITY.....	50
FIGURE 40 I3C BROADCAST ENTDAACCC— SENT BY THE CONTROLLER	51
FIGURE 41 HOT-JOIN REQUEST TRANSFER, AS CONTROLLER/TARGET	53
FIGURE 42 IBI SEQUENCE WITH MANDATORY DATA BYTE	55
FIGURE 43 MDB FIELD FORMAT	56
FIGURE 44 IBI TRANSFER, AS CONTROLLER/TARGET.....	56
FIGURE 45 ENTASx BROADCAST FORMAT	57
FIGURE 46 ENTAS DIRECT FORMAT	58
FIGURE 47 GETSTATUS FORMAT 1.....	58
FIGURE 48 GETSTATUS FORMAT 2 USING PRECR	59
FIGURE 49 I3C BROADCAST CCC WRITE (EXCEPTED ENTDAAC, RSTACT)	60
FIGURE 50 "I3C DIRECT CCC WRITE — 1ST PART / MULTIPLE 2ND PARTS (EXCEPT RSTACT)"	60
FIGURE 51 FIGURE 52 "I3C DIRECT CCC READ — 1ST PART / MULTIPLE 2ND PARTS (EXCEPT RSTACT)"	61

FIGURE 53 LEGACY I2C WRITE MESSAGES - AS CONTROLLER..... 63

FIGURE 54 LEGACY I2C READ MESSAGES - AS CONTROLLER 63

FIGURE 55 I3C CONTROLLER ARCHITECTURE TOP MODULE 64

List of Tables

TABLE 1 COMPARISON BETWEEN I2C AND MIPI I3C	21
TABLE 2 DEVICE ROLES ON I3C BUS.....	29
TABLE 3 BUS CHARACTERISTICS REGISTERS	34
TABLE 4 DEVICE CHARACTERISTICS REGISTERS.....	34
TABLE 5 START AND STOP TIMING PARAMETERS	36
TABLE 6 I3C TARGET ADDRESS RESTRICTIONS	44
TABLE 7 HOT-JOIN STANDARD VS PASSIVE MEHODS	54
TABLE 8 ENTER ACTIVITY STATE 0-3 COMMAND CODES	57
TABLE 9 MINIMUM BUS ACTIVITY INTERVAL.....	57
TABLE 10 GETSTATUS MSB-LSB.....	58
TABLE 11 GETSTATUS MSB-LSB.....	59
TABLE 12 LIST OF SUPPORTED I3C CCCs, AS CONTROLLER/TARGET	61
TABLE 13 LIST OF SUPPORTED I3C CCCs, AS CONTROLLER/TARGET (CONTINUED)	62

List of Abbreviations

MIPI	Mobile Industry Processor Interface
I3C	Improved Inter-Integrated Circuit
STM	STMicroelectronics
MCU	Microcontroller Unit
SDR	Single Data Rate
HDR	High Data Rate
SDA	Serial Data
SCL	Serial Clock
HDR-DDR	High Data Rate Dual Data Rate Mode
HDR-TSL	High Data Rate Ternary Symbol Legacy
HDR-TSP	High Data Rate Ternary Symbol Pure
HDR-BT	High Data Rate Bulk Transport
DMA	Direct Memory Access
D2D	Device to Device Communication
IOT	Internet of Things
DAA	Dynamic Address Assignment
HJ	Hot-Join
IBI	In-Band Interrupt
CRR	Controller Role Request
CRH	Controller Role Handoff
TrDA	Data Rising Time
Vol	Output low voltage level
CCC	Common Command Codes
DCR	Device Characteristics Registers
BCR	Bus Characteristics Registers
LVR	Legacy Characteristics Registers
TCBS	Clock Before Stop
LUT	Look Up Tables
FSCL	SCL Frequency
tCAS	Clock After Start
tBUF	Bus Free Clock
tAVAL	Bus Available Clock
tIDLE	Bus Idle Clock
APB	Advanced Peripheral Bus

Chapter 1: Introduction

1.1 Communication Protocols

Communication protocols have been a main part of the Electronics and Communications Industry, especially protocols that focus on the Inter-Integrated Circuit communication, like communication between two blocks in a processor, or the communication between the processor itself and other blocks in the system, or even the communication between the whole system and other peripherals like the communication between the processor and the sensor.

Communication Protocols act as the traffic light that governs the roads, preventing any congestions, accidents or failure to reach destinations, here our traffic light is the protocol itself which is being controlled by the **controller** or in the old naming the “Master”, our road is the **bus**, and our cars are the **messages** that need to be delivered between the controller and the destinations which are our **targets** or in the old naming “Slaves”, also a communication could occur between each destination and the others without the need of the main controller in the name of D2D (Device to Device Communication) which new protocols like MIPI I3C cover.

There are many communication protocols that specialize in the Inter-Integrated Circuit communications like: SPI, UART, I²C and now I3C, each protocol excels in a set of jobs and is needed when the application requires a specific feature, like the use of SPI when devices count is a priority or the use of I²C when low cost is required.

1.2 From I²C to I3C

I²C is a great communication protocol when it comes to a processor that needs to relate to a set of devices, as mentioned above, its low cost made it a go-to in many scenarios, especially in embedded systems and IOT. To keep up with the increasing revolutionary requirements of the industry, I²C new versions were released, which their low- cost ability to communicate worked perfectly with the required jobs, it worked good enough that there was no need for a major update, only newer versions of the same I²C were developed.

The changing in requirements has made I²C so limited and problematic, with the increasing use of sensors, especially **Wireless Devices**, hence a new upgrade in the communication protocols had to be done, and a new protocol with a huge update to the I²C is developed to solve the problems that none of the old protocols had the ability to solve including Bandwidth and Pin count with respect to increasing the number of sensors.

1.3 Motivation for MIPI I3C

I3C was developed to face sensor integration problems by providing fast , low cost , low power and managed two wire digital interface with a backward compatibility with legacy I²C devices. I3C Main Concerns were the use of low Energy for Data and control Transfers while reducing numbers of physical pins used by the interface.

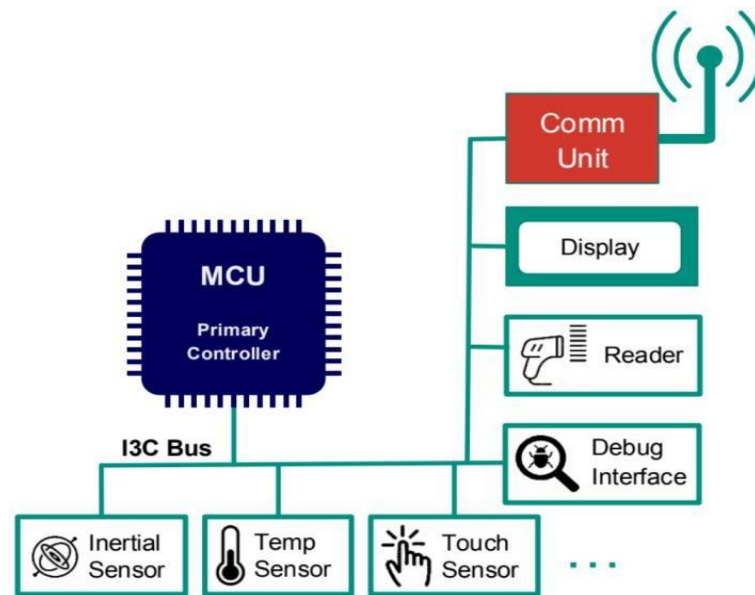


Figure 1 Typical IoT Device Architecture

I3C Main Purposes:

- Improve I²C Features.
- Multi drop between host processors and peripheral Devices.
- Providing an easier design for System Design Engineers.

This last point has been a great addition to I3C's set of features as it saved time, cost and effort that used to be needed to implement interfaces between each device and the protocol, but thanks to the I3C target BCR and DCR which we will discuss shortly, it has become an easy task for System Engineers to integrate or work with MIPI I3C.

So, if we want to preview a comparison between these protocols, The I²C and The MIPI I3C:

Table 1 Comparison Between I2C and MIPI I3C

Feature	I3C	I2C
Speed & Efficiency	High-Speed & Power-Efficient	Less Speed & Power-Efficient
Address Assignment	Dynamic Address Assignment	Static Address Assignment
Maximum Clock Speed	SDR Mode: 12.5 MHz HDR-DDR Mode: 25 MHz HDR-TSL Mode: 33 MHz	Standard Mode: 100 KHz Fast Mode: 400 KHz Fast Mode+: 1 MHz High-Speed Mode: 3.4 MHz
Driver Type	Push-Pull & Open Drain	Open Drain Only
Effective Data Rate	33.3 Mbps	3 Mbps

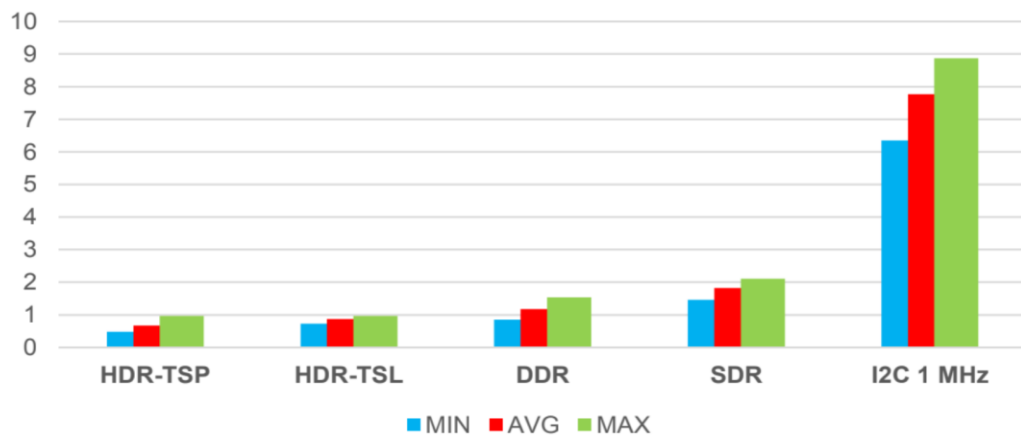


Figure 2 I3C Modes Vs PC Effective Energy Range per 1kB (μJ)

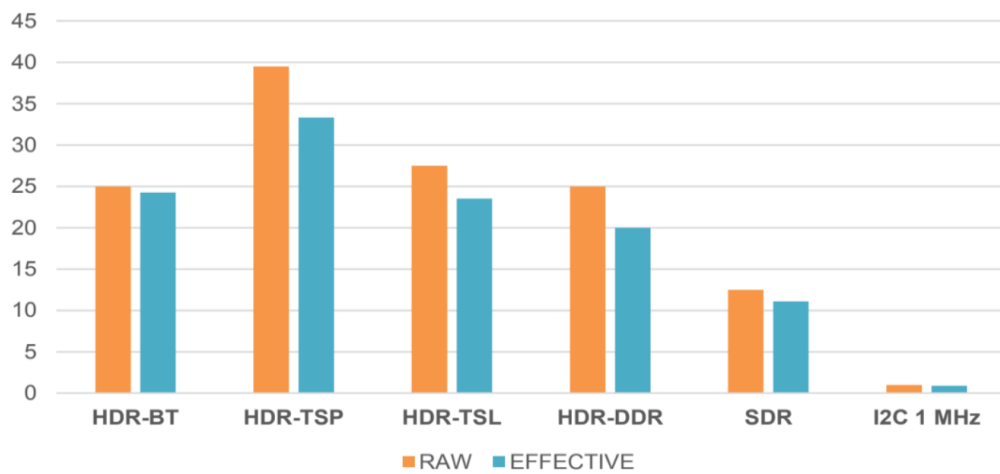


Figure 3 I3C Modes Vs I2C Effective Bitrates for 12.5 MHz Clock (Mbps)

1.4 MIPI I3C Main System

The Generic I3C System would be:

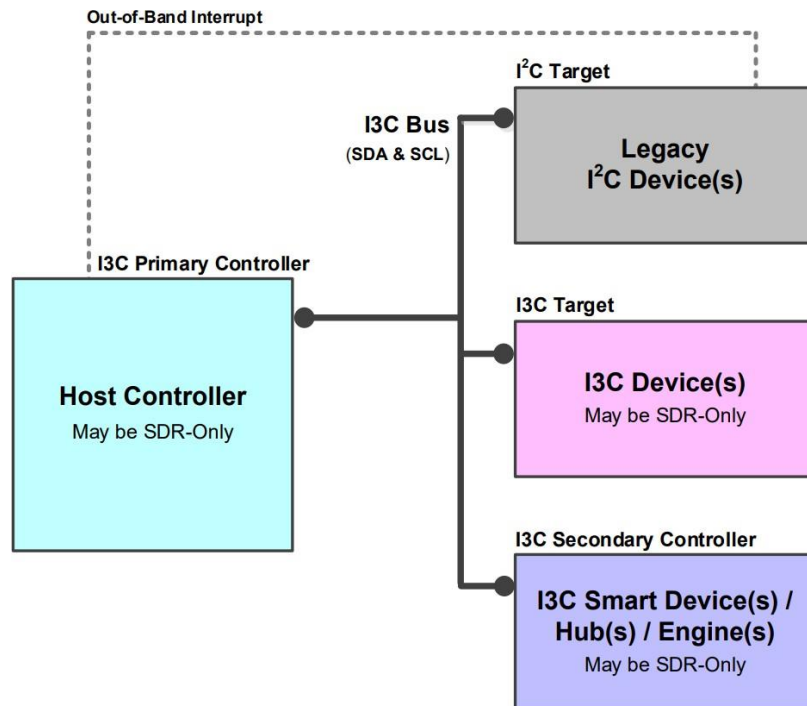


Figure 4 MIPI I3C System Diagram

From the System Diagram, we can notice that The I3C System mainly consists of three pillars, The Controller, The Target, and The Bus itself, as each one has a specific role:

- **The Controller:** It controls the bus and governs communication with the target.
- **The Target:** It is the destination that receives the data and checks for any errors.
- **The Bus:** It carries the data between the controller and the target, and it consists of two main parts:
 - **SDA (Serial Data):** A bi-directional data pin that carries data between devices.
 - **SCL (Serial Clock):** can either be a clock pin or a bi-directional data pin in certain HDR-Modes.

1.5 Introduction to Push-Pull Based Communication

Many Communication protocols have worked with the well-known Open-Drain Circuit such as I²C, as Open-Drain enabled point-to-point (2-ports) based communication good enough that no other circuit was needed, but technology needed faster communication, which is done with the help of Push-Pull circuits.

1.5.1 Open Drain Circuit

Here, the SDA and SCL lines are always pulled up to the VDD through the Pull-Up resistor that may be connected in any of these ways:

1. As a Passive Resistor connected to VDD
2. As a Passive Resistor Connected to a Current Source
3. In any other approach that ensures that the SDA rises within the allowed Data Rising Time (t_{rDA}), Targets with output low voltage level (V_{ol}) can drive SDA low within the allowed (t_{rDA})

Open-Drain Circuits work as follows:

- **Pushing Down**

When NMOS turns on, SDA and SCL are Pulled Low

Hence, Quick transition occurs from high to low as NMOS pulls charge from any bus capacitance from SDA, SCL:

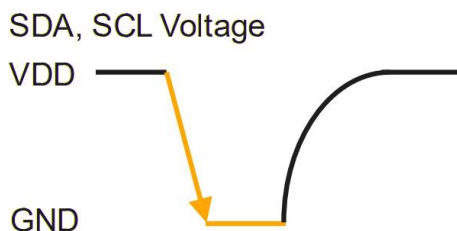


Figure 6 Push Down Case in Open Drain Circuits

The speed of the transition is determined by NMOS drive strength and the bus Capacitance on SDA and SCL.

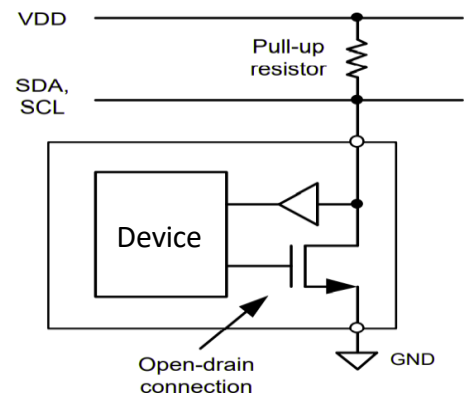
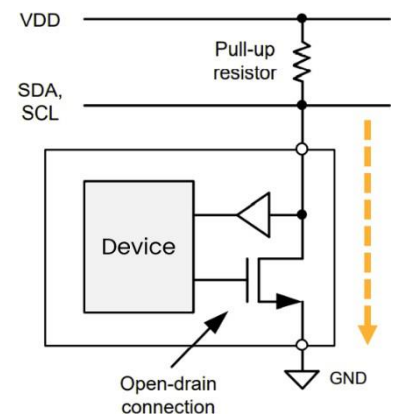


Figure 5 Open-Drain Circuit



- **Pulling Up**

When NMOS turns OFF, SDA or SCL is released and returns high through the pullup resistor.

Hence, exponential rise occurs:

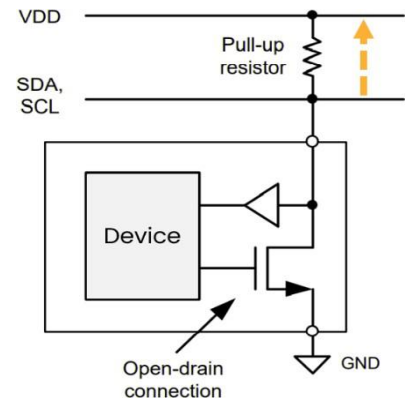
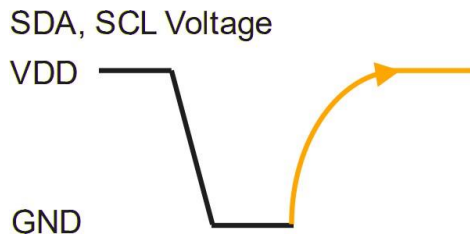


Figure 7 Pulling Up Case in Open Drain Circuits

The exponential transition is determined by capacitance on SDA or SCL and Pull-Up resistor size.

1.5.2 Push-Pull Circuit

Here, the SDA and SCL lines are either pulled up to VDD or pushed down to GND.

Push-Pull Circuits work as follows:

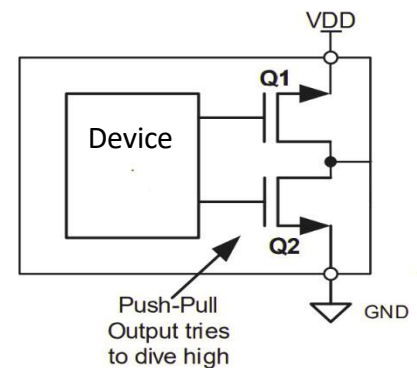


Figure 8 Push-Pull Circuit

- **Pushing Down**

When NMOS is ON and PMOS is OFF:

- Lines are discharged to GND
- SDA and SCL will be 0

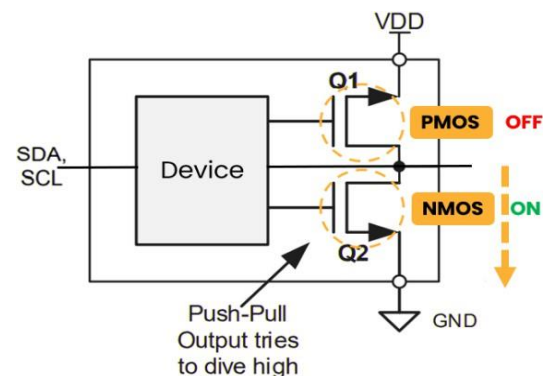
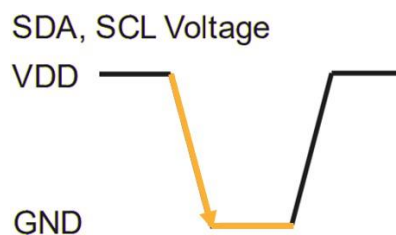


Figure 9 Push Down Case in Push-Pull Circuits

A Fast discharge occurs as current runs through NMOS acting as a short circuit to GND.

• Pulling Up

When NMOS is OFF and PMOS is ON:

- Lines are charged from VDD
- SDA and SCL will be 1

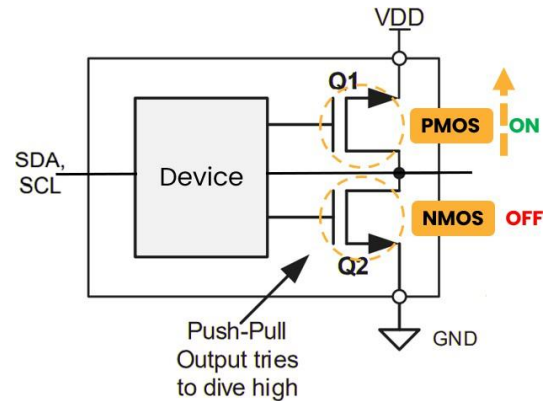
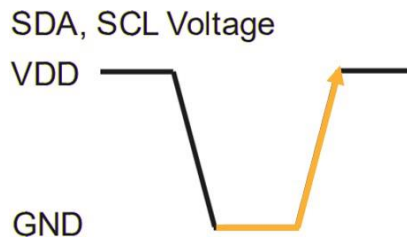


Figure 10 Pulling Down Case in Push-Pull Circuits

Fast charging occurs as current runs through PMOS acting as a short circuit to the Line.

Here, depending on the working behavior of Push-Pull circuits, more than one device can be connected to one line with the ability to call all these devices at once, which wouldn't have happened with Open-Drain circuits where we can connect more than one device on the Bus but only targeting one device at once.

But a problem may occur in this case if one of these connected devices has NMOS ON while the other devices have it OFF with the line being connected to VDD, which will then cause the line to discharge to GND and hence Bus contention output at indeterminate state occurs which may damage this device:

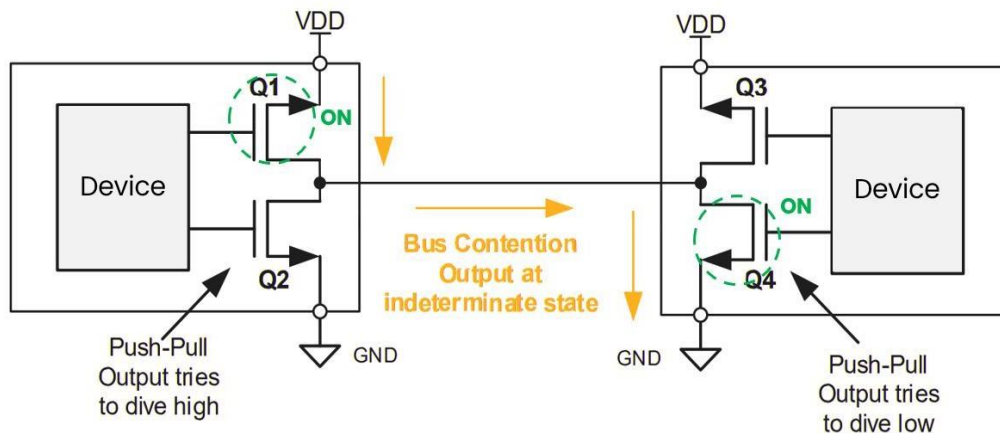


Figure 11 Push-Pull Discharging Problem

This will be solved by using Keeper Circuits, which can be installed by different methods:

- By using keeper Circuits form the controller of the I3C
- By using a separate Keeper Circuit on the bus

Using the keeper Circuit of the I3C controller may be an overhead task for designers, hence the second approach is usually the go to.

1.5.3 Keeper Circuits

The Problem here is when the output is floating, which occurs most of the time in push- pull circuits when the output isn't driven.

The basis keeper circuits could be designed as follows:

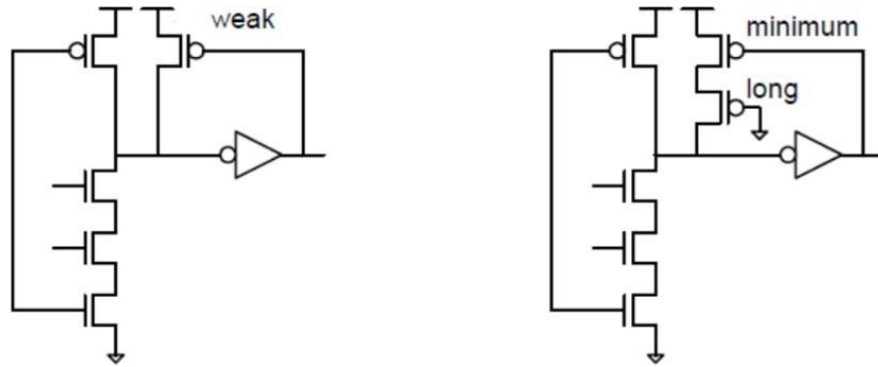


Figure 12 Keeper Circuit

This circuit will help keep the bus pulled to VDD when it's floating (not driven) and when there is a push request, it will win the fight as the keeper has weak transistor (small w/l)

A Domino Circuit or a NORA Circuit can also be used to solve push-pull discharging problems.

So, the innovation in the MIPI I3C circuitry was the integration of Push-Pull circuits with other additions to the Hardware of the MIPI I3C, the next chapter will go through the Innovation of MIPI I3C Protocol Features and Algorithms including different messaging techniques and words, with a new In-band integrated commands (CCC) support.

Chapter 2: MIPI I3C Protocol

MIPI I3C Protocol is a major upgrade to the I²C Protocol, both sharing the same built as a Block Diagram with Controller and Targets communicating over 2 lines SDA for Serial Data and SCL for Serial Clock except for the I²C having addition separate wires for each required interrupt signal, which is solved in MIPI I3C in the feature of In-Band-Interrupt as illustrated in Chapter 2 section 2.9, MIPI I3C having only 2 wires made it easy to be compatible to interface with other peripherals and systems easily, with a backward compatibility to communicate with Legacy I²C devices that could co-exist on the same bus, also MIPI I3C has introduced Dynamic Addressing which is an Upgrade to the old Static Addressing while maintaining the ability to Statically Address the I²C Legacy devices that could co-exist on the same bus, with other features as Sharing Controller ownership in what is known as Controller Role Handoff where the device that has the controller role called “Active Controller”, the ability to receive directly form recently powered on devices that wasn’t on since the start of messaging in what is called Hot-Join, having Two modes of Communication each supporting different speeds, the first is Single Data Rate (SDR) and the other is High Data Rate (HDR).

In this Chapter we will go through MIPI I3C Protocol Specifications, including its features, focusing on SDR Mode Communication, but first, an introduction to MIPI I3C two types of messaging and I3C Modes of communication.

2.1 MIPI I3C Message Types

MIPI I3C has two types of messaging:

- 1. Direct Messaging:**

Active Controller directly messaging one Target or vice versa. (Using the address of the required target to call)

- 2. Broadcast Messaging:**

Active Controller broadcasts a message for all the available targets on the bus excluding the legacy I²C device which doesn’t support this messaging type. (using a special address that all targets can receive from (7’h7E))

2.2 MIPI I3C Communication Modes

2.2.1 I3C Single Data Rate (SDR Mode)

This mode is like the I²C modes of communication but with the enhanced messaging of the previous section, the SDR Mode speed is **12.5 MHZ**.

I3C Bus is always initialized and configured in SDR Mode, as a result most of the essential features are covered in SDR Mode.

SDR Mode is the default mode for MIPI I3C and is used to enter other modes, or sub modes, or states, also is used for built in functions as CCCs, IBI, or for transition from I²C to I3C by assignment of Dynamic addressing.

2.2.2 I3C High Data Rate (HDR Mode)

This mode provides high speed communication, having 4 main types:

- **Dual Data Rate (HDR-DDR) (25 MHZ):**
Not so different from I²C Fast mode, but it runs at twice the speed of SDR
- **Ternary Symbol Legacy (HDR-TSL) (33 MHZ):**
Significantly different from the I²C, with higher data rates plus ternary coding for buses with a mix of I²C and I3C devices
- **Ternary Symbol Pure Bus (HDR-TSP) (up to 33.4 MHZ):**
For I3C pure buses only
- **Bulk Transport (HDR-BT):**
Gives the highest possible speed, using both data wire (SDA) and clock wire (SCL) to transmit data in model leveraging Dual Data Rate over Single, or Double, or Quad- lane configurations

The project mainly focuses on SDR Mode Communication, the next sections go through MIPI I3C SDR Mode Specifications.

2.3 Bus Configurations

2.3.1 Device Roles

MIPI I3C Bus can be configured as the link between all the devices existing on it, these devices can have different roles as follows:

Table 2 Device Roles on I3C Bus

Device Type	Device Role	Description
I3C Controller	I3C Primary Controller	Initially configures the I3C Bus, has HDR Support
	SDR-Only Primary Controller	Initially configures the I3C Bus, no HDR Support
	I3C Secondary Controller	Can control the bus but currently operating as Target
	SDR-only Secondary Controller	Can control the bus but currently operating as Target, no HDR Support
I3C Target	I3C Target	Ordinary I3C Target, no controller capability
	SDR Only Target	Ordinary I3C Target, no controller capability, no HDR Support
	PC Target	Ordinary I3C Target, no controller capability

As shown, Devices on the bus can have different roles, they can either be a Controller or a Target.

- **Controller Roles:**

The I3C controller device can function as either a Primary Controller or a Secondary Controller. The Primary Controller is responsible for controlling and configuring the bus initially, while the Secondary Controller can act as both an I3C Controller and Target. The Secondary Controller operates as a target until a controller role handoff occurs using the I3C CCC. The handoff passes the active controller role from the Primary Controller to the Secondary Controller. The Primary and Secondary Controllers can both operate only at SDR and do not support HDR Modes. The I3C bus is always configured in SDR Mode. Hence, we can define Controller devices as:

- **Primary Controller:**

It's the controller device that initially configure the I3C Bus and serves as the first active controller, only one device on the bus can be primary controller, supports both SDR mode and at least one HDR Mode, while the SDR only Primary Controller supports only SDR.

- **Secondary Controller:**

It's any device on the bus other than the active controller that have I3C Controller Capability, more than one device can be secondary controller, it behaves as a target until it requests the Active Controller role, supports both SDR Mode and at least one HDR Mode, while SDR only Secondary Controllers supports only SDR.

- **Target Roles:**

The target devices on the I3C Bus can either be I3C only targets which are the ordinary devices that can interface with I3C and have no Controller role capabilities, or SDR-only targets which can only communicate in SDR Mode and will ignore any messages in other HDR Modes , I3C Bus can also has I²C Target devices which are legacy devices that could only understand I²C- based messages, these devices cannot have I3C Controller or Target capabilities.

- **Targets:**

The old common known as Slave, which is the device that is always listening to bus for relevant CCC sent by the active controller and respond to it, Always supports SDR Mode and may at least support one HDR Mode unless it was a SDR only I3C Target which supports only SDR, it doesn't generate the clock in the SDR Mode, it should at least support one of the dynamic address methods, it also can request IBI, generate Hot-join, request to become Active controller if it supports the I3C Controller Capability.

Based on the I3C device roles, a generic MIPI I3C Bus block Diagram can be as follows:

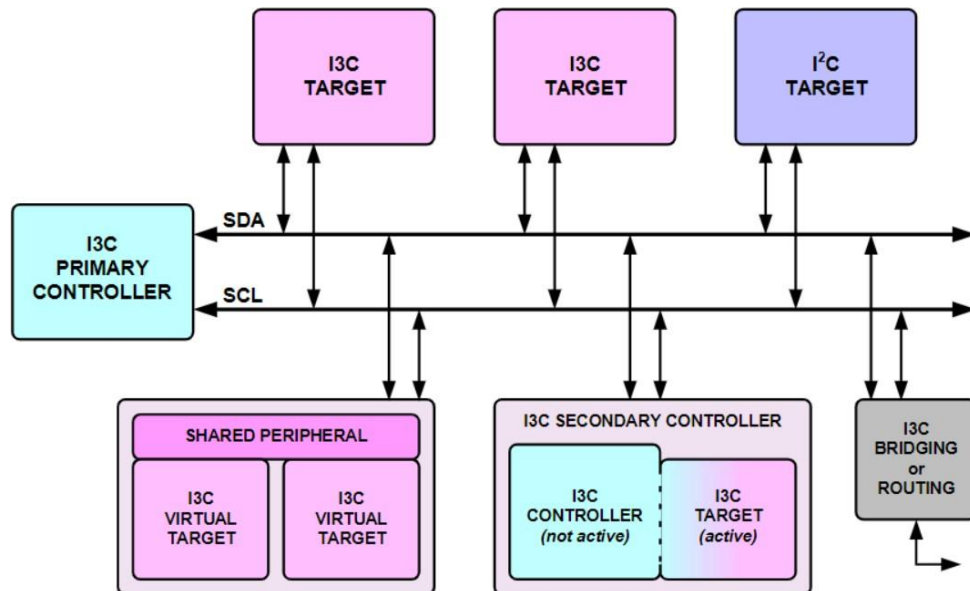


Figure 13 MIPI I3C Bus Block Diagram

From the block diagram, target devices can be either a one target device or a set of targets device called “**Composite Devices**”, some of them may have virtual targets, these composite devices can be classified into these types:

1. Shared Peripheral and virtual targets:

- Shall maintain separate configuration for each Virtual Target, including the assigned Dynamic Address.
- Initiates the Hot-Join Request during Bus Initialization or whenever necessary.
- Handles the signaling and framing for HDR Modes (if supported)
- Participate in Broadcast and Direct CCC flows, both in SDR and HDR Modes.

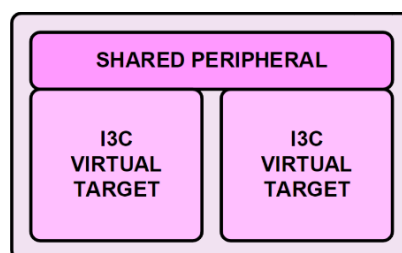


Figure 14 Shared Peripheral Device

For virtual targets:

Each Virtual target has its own dynamic address which Receives data for Private Write transfers addressed to its Dynamic Address and provides data to the shared Peripheral logic for responding to Private Read transfers and raising In-Band Interrupt requests.

2. I3C Secondary Controller:

It has the ability to act as both the target (when it's Controller Capability is not active) and as a controller (when its controller capability is active)

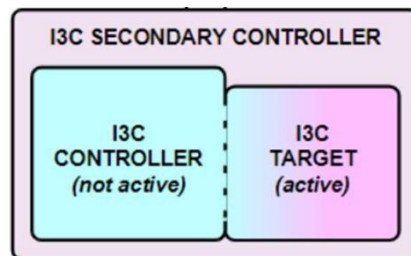


Figure 15 I3C Secondary Controller

3. Bridges and Routing devices:

Bridges Connect the I3C Bus to different protocol Bus(SPI OR I2C) while Routing devices Connect the I3C Bus to another I3C Bus

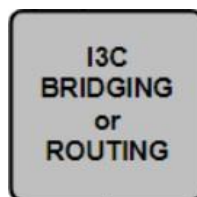


Figure 16 I3C Bridging and Routing Devices

2.3.2 Point to Point Communication

Another use case for I3C Bus is Point to Point Connection as:

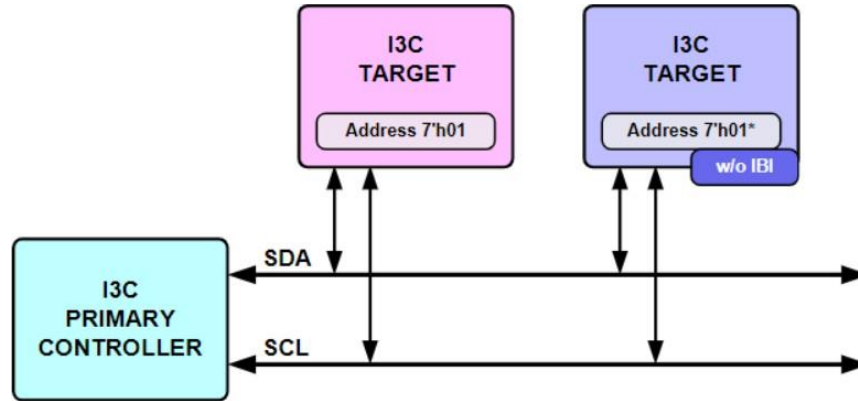


Figure 17 Point to Point Connection

Where controller assigns the same dynamic address to one or more I3C Devices only one of which supports Read Transactions and In Band Interrupts

2.3.3 MIPI I3C Device Characteristics

All the mentioned I3C Devices have ton of features, it's not necessary for a device to enable all its features, as the enabled features will appear in Characteristics Registers associated with the device and its roles, these features could be:

2.3.3.1 Bus Characteristics Registers (BCR)

It's a read only register that includes I3C's compliant device's roles and capabilities to be used in Dynamic Address Assignment and Common Command Codes.

Table 3 Bus Characteristics Registers

Bit	Name	Description	Notes
BCR[7]	Device Role[1]	2'b00: I3C Target 2'b01: I3C Controller-capable 2'b10: Reserved for future definition by MIPI Alliance I3C WG	1
BCR[6]	Device Role[0]	2'b11: Reserved for future definition by MIPI Alliance I3C WG	
BCR[5]	Advanced Capabilities	1: Supports optional advanced capabilities. Use GETCAPS CCC (Section 5.1.9.3.19) to determine which ones. 0: Does not support optional advanced capabilities	2
BCR[4]	Virtual Target Support	0: Is not a Virtual Target and does not expose other downstream Device(s) 1: Is a Virtual Target, or exposes other downstream Device(s)	3
BCR[3]	Offline Capable	0: Device will always respond to I3C Bus commands 1: Device will not always respond to I3C Bus commands	4
BCR[2]	IBI Payload	0: No data bytes follow the accepted IBI 1: One data byte (MDB) shall follow the accepted IBI, and additional data bytes may follow; see also the Set/Get Maximum Read Length CCC (Section 5.1.9.3.6). Data byte continuation is indicated by T-Bit per Section 5.1.2.3.4 . See also Section 5.1.8 on use of IBI Payloads for Timing Control.	—
BCR[1]	IBI Request Capable	0: Not Capable 1: Capable	—
BCR[0]	Max Data Speed Limitation	0: No Limitation 1: Limitation	5

So, BCR will be an 8 bits registers, that would be encoded in a way that for each bit's value it reflects on Device role, Capabilities, and speed limitations.

2.3.3.2 Device Characteristics Registers (DCR)

It's a read only register that includes I3C's compliant device's type to be used in Dynamic Address Assignment and Common Command Codes.

Table 4 Device Characteristics Registers

Bit	Name	Description
DCR[7]	Device ID[7]	255 available codes for describing the type of sensor, or Device. Examples: Accelerometer, gyroscope, composite Devices. Default value is 8'b0: Generic Device
DCR[6]	Device ID[6]	
DCR[5]	Device ID[5]	
DCR[4]	Device ID[4]	
DCR[3]	Device ID[3]	
DCR[2]	Device ID[2]	
DCR[1]	Device ID[1]	
DCR[0]	Device ID[0]	

So, DCR will be 8 bits register, that would be encoded by the connected device Vendors, describing the device type, for example:

2.4 Bus Conditions

MIPI I3C Bus will have many states, but for the features (Hot-Join, IBI, Controller role request) of the I3C to be allowed on the bus, it should be in a specific condition for each feature to be valid, these conditions come after detecting a stop pattern as follows:

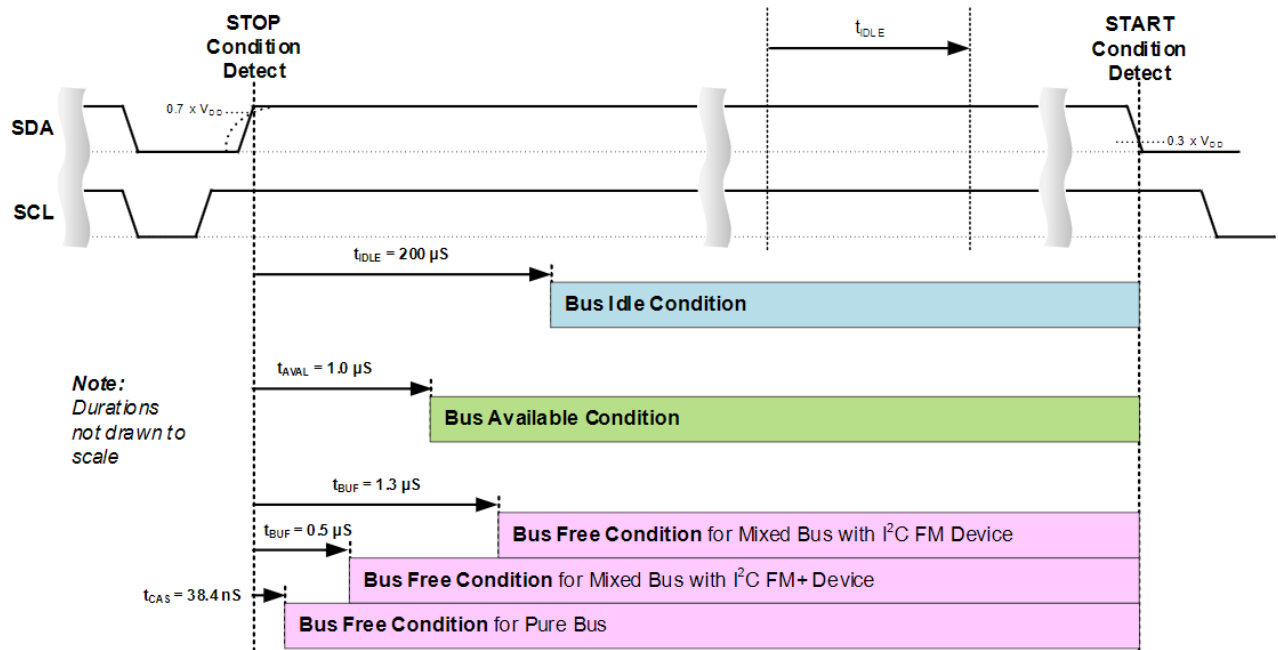


Figure 18 MIPI I3C Bus Conditions

1. Bus Free Condition

After a specific duration from the stop pattern being detected, the bus will enter Bus Free Condition depending on the devices on the bus as:

- For Pure Bus (I3C Devices only): the duration is t_{CAS} (38.4 ns)
- For Mixed Bus of I3C and FM+ I²C Devices: the duration is t_{BUF} (0.5 us)
- For Mixed Bus of I3C and FM I²C Devices: the duration is t_{BUF} (1.3 us)

2. Bus Available Condition

A Target may only issue a START Request (for an In-Band Interrupt, or for a Controller Role Request) after a Bus Available Condition, which I3C bus enter after a duration of t_{AVAL} (1.0 us) from the stop pattern being detected.

3. Bus Idle Condition

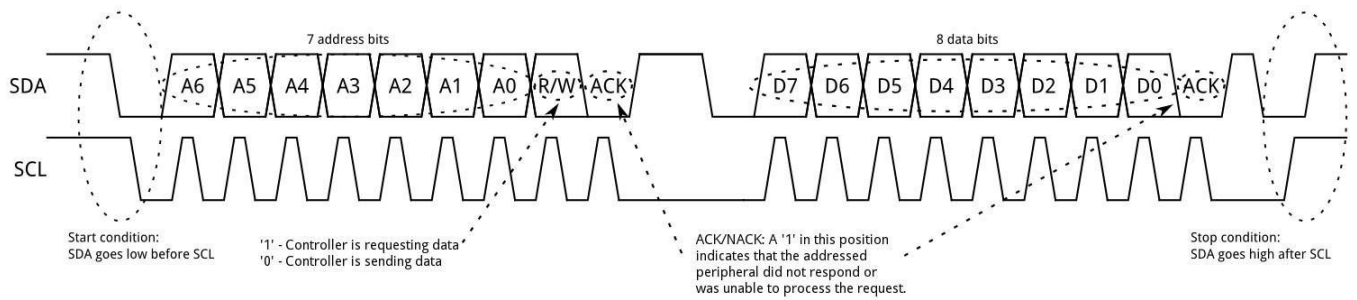
The I3C Bus Idle Condition is defined in order to help ensure Bus stability during Hot-Join events, and is defined as a period during which the SDA and SCL lines both sustain High level for a duration of at least t_{IDLE} (200 us) from stop pattern detected

2.5 Bus Communication

2.5.1 I3C Message Elements

The primary protocol and mode of I3C is SDR (Single Data Rate) Mode which is based on the I²C standard protocol with notable variations.

The I²C message illustrated below consists of START condition, 7-bit ADDRESS, 1-bit R/W, 1-bit address ACK/NACK, Single/Multiple frame/s of 8-bit DATA with 1-bit data ACK/NACK and ends



with STOP condition.

Figure 19 I2C Message

The I3C START (S) and STOP (P) have the same signaling as I²C but with different time parameters summarized in the following table.

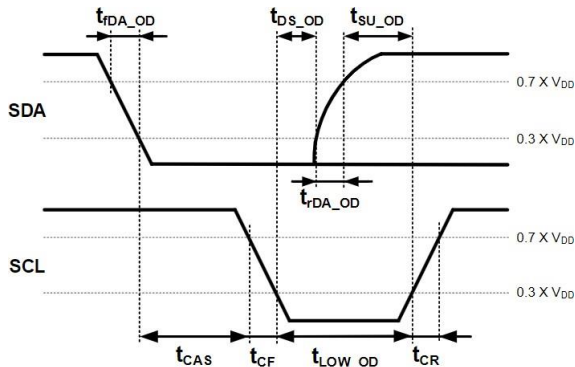


Figure 20 START Condition

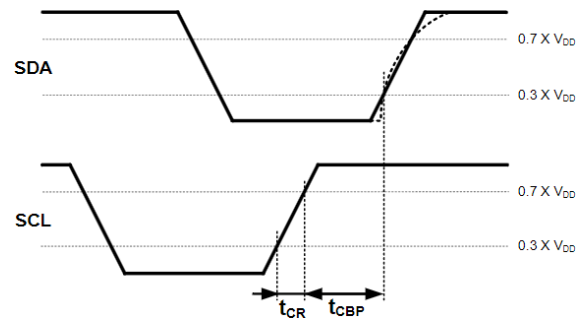


Figure 21 STOP Condition

Table 5 START and STOP Timing Parameters

t_{CAS}	Clock After START = 38.4 ns to 50 ms (depending on ENTASX)
t_{SU_OD}	Setup Time of SDA Signal = 3 ns
t_{LOW_OD}	Low Period of SCL Clock = 200 ns
t_{CBP}	Clock Before STOP = $t_{CAS(min)}/2 = 19.2$ ns

The previous time parameters reaffirm the fact of supporting the communication with I²C legacy devices where the I3C Controller still has to ensure that all targets are able to see the START condition with the Open-Drain drive.

The I3C Repeated START (Sr) and STOP is tolerated any time that SCL is High while the Controller controls SDA or SDA is Open-Drain. This is unlike I²C, which wants STOP or Repeated START only after a NACK of an address, or after ACK/NACK of data.

Although I3C SDR also prefers STOP and Repeated START to be used only in those situations, and after an I3C Broadcast Address is ACKed, it does not disallow them in any other location. However, appearance of a STOP or Repeated START in the middle of an Address or in the middle of data is interpreted to cancel that Address or data.

The I3C Address Header has the same signaling as I²C Address Header that consists of 7-bit Address and R/W bit to determine whether the previous Target address is for Reading operation '1' or Writing operation '0'. The difference comes in timing of both addresses as the I3C Address has the ability of optimization for some cases of un-arbitrable addresses.

Data Words for I3C differs from I²C's in the 9th bit. This bit followed by 8-bit Data Word is called T-bit that represent the Parity bit for Writing Data and the Transition bit for Reading Data. Transition bit indicates whether the I3C Target wants to abort the messaging, or it is the decision of I3C Controller based on the number of Data Frames defined in the beginning of the message.

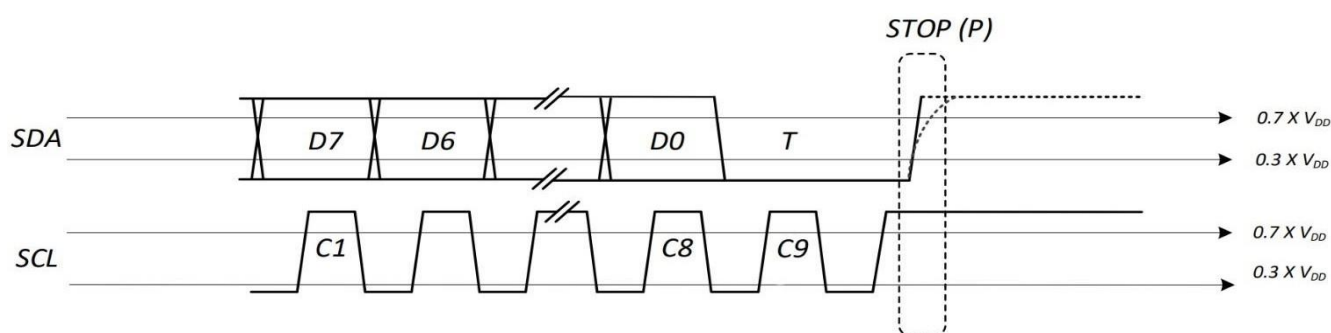


Figure 22 T-bit When Target Ends Read and Controller Generates STOP

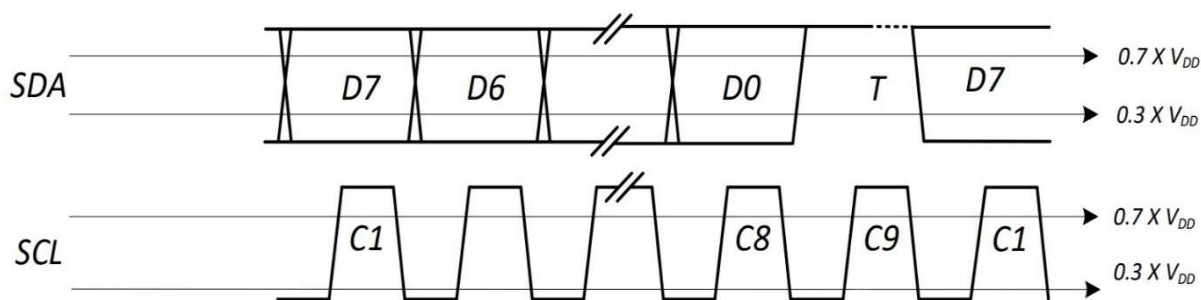


Figure 23 T-bit When Target and Controller Agree to Continue Read Message

In I3C, the SCL line is only driven by the Controller. Normally this drive is Push-Pull, but it can also be Open Drain.

An I3C Message is defined as everything from the initial START (or Repeated START) to the next Repeated START or STOP.

2.5.2 SDR Message Types

An I3C Message is an SDR Message if:

- The Address in the Address Header is 7'h7E (the **I3C Broadcast Address**). All I3C Targets shall match the Address value 7'h7E. No I²C Target will match Address 7'h7E, because that value is reserved and unused in I²C.
- The Address in the Address Header matches Target's **Dynamic Address**. All I3C Targets shall match their own Dynamic Address. (It is permitted to then NACK the Header if needed.)

All I3C Targets shall ignore all Messages with Addresses other than 7'h7E or the I3C Controller Assigned Address and shall await either the Repeated START or the STOP. I3C Targets shall not transmit on the Bus in response to a non-matching Address.

Legacy I²C Targets will ignore any Message not addressed to them and will await the next START or STOP. Legacy I²C Targets may also not see some or all the I3C Messages and Modes due to the speed of SCL signaling.

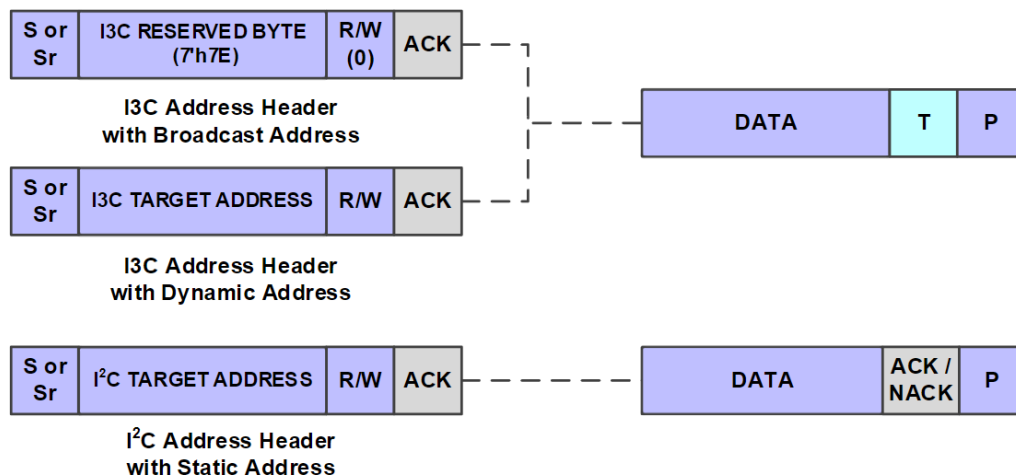


Figure 24 I3C Message Types

2.5.3 Role of I3C Target

An I3C Target is a functional role that is embodied or presented by an I3C Device. Such a Device may be a standalone physical Device that presents only one Target, or it may be a composite Device that presents multiple Virtual Targets using shared Peripheral logic. In either case, each Target (which may be a Virtual Target) presented on the I3C Bus may be assigned its own unique Dynamic Address, and the Controller can address it independently for transfers.

Many such I3C Device implementations are possible that might embody or present one or more active I3C Target roles. For composite I3C Devices that present multiple Virtual Targets, the shared Peripheral logic might handle many of the underlying functions that would normally be handled by internal logic for a single, standalone I3C Device that only presented a single Target on the I3C Bus. For simplicity, the remainder of this section treats both options equally, referring to “Target” in a generic sense. The I3C Target does not have to know whether it is on a Legacy I²C Bus or an I3C Bus. If it has an I²C Static Address, then it may participate using that Address up until it is assigned a Dynamic Address. Once assigned a Dynamic Address, unless asked to Reset, it shall only operate as an I3C Target.

An I3C Capable Target may act as an I²C Device before it gets its Dynamic Address (DA) assigned. However, the Target shall ACK the START with Address 7'h7E. (The only exception would be if the Target is choosing to remain an I²C-only Device on a given Bus or use, in which case it would leave its 50 ns Spike Filter enabled.)

2.5.3.1 Before Receiving a Dynamic Address

The Target shall process all required Broadcast CCCs, including:

- **ENTDAA (0x07) – Enter Dynamic Address Assignment**
- **RSTDAA (0x06) – Reset Dynamic Address Assignment**
- **ENEC (0x00) – Enable Target Event Command**
- **DISEC (0x01) – Disable Target Event Command**

The Target may choose to understand and process the SETAASA CCC and shall disregard all Directed CCC commands but shall properly recognize the ends of Directed CCCs (i.e., either Repeated START followed by 7'h7E, or STOP).

When no Dynamic Address has been assigned yet, the Target may either support or ignore non- Required and Conditionally Required Broadcast CCCs. The I3C Target shall ignore TE0 type errors related to incorrect Addresses only.

2.5.3.2 After Receiving a Dynamic Address

The Address Header Matching procedure is initiated after the START or Repeated START signal for any of the specified addresses, such as the I3C Broadcast Address (7h'7E / RnW 0). Upon receiving a matching message, the target is obligated to acknowledge and process the message at least through the first byte of data, unless the target has chosen to remain an I²C-only device on a given bus with the 50 ns Spike Filter enabled.

If a 7'h7E Broadcast Message contains a byte of data, the message is a CCC. If the CCC is required, the target must remain ready to respond even if it is processing previously sent messages (such as GETSTATUS) or has not yet received a Dynamic Address that includes the ENTHDRn CCCs. In the case of a Mode Change CCC, it may result in the Target entering one of the following modes: Dynamic Address Assignment (DAA) Mode, High Data Rate (HDR) Mode, or Target's assigned Dynamic Address or assigned Group Addresses.

If the Target has no dynamic address, it shall participate in the Dynamic Address Assignment (DAA) mode and wait for the STOP condition if it already has an assigned dynamic address. If the Target supports the HDR mode, it shall enter the HDR process and detect the HDR Exit Pattern if the HDR mode is not supported.

Finally, when the Target is assigned a dynamic or group address, it has the option to either acknowledge and process the message as I3C SDR or ignore any bits up until the next STOP or Repeated START by NACKing the message.

2.5.3.3 I3C Target Acting as an I²C Target with Static Address

- **On a Legacy I²C Bus:** I3C Target can deal safely with all messaging as it is already coming from a Legacy I²C Controller. This will be done acting as standard I²C Target using an I²C Fm/Fm+ Spike Filter of 50 ns (or more) if it has one.
- **On an I3C Bus:** I3C Target's I²C Fm/Fm+ Spike Filter of 50 ns shall disable once it sees a Message from an I3C Controller (ACK then disable), the first I3C Address Header emitted with Fm/Fm+ timing parameters after Bus initialization (i.e., a START followed by 7'h7E and a RnW bit of 0).

2.5.4 I3C Address Header

The Address Header following a START is an Arbitrable Address Header, while the Address Header following a repeated START shall not Arbitrated. This means the START and at least the first Address bit and ACK/NACK are issued on SDA using Open Drain Bus drive, similar to I²C. However, some of the Arbitrable Address Header may be driven on SDA using Push-Pull and higher speed which is considered an Arbitration Optimization. Using the I3C Arbitrable Address Header, I3C Targets may transmit any of three requests to the I3C Controller:

A. An In-Band Interrupt

This is equivalent to toggling a wire to get the Controller's attention. The In-Band Interrupt request shall be made using Target's Dynamic Address with a RnW bit of 1.



Figure 25 In-Band Interrupt Request Frame

B. A Controller Role Request

An I3C Device shall not make such a request unless it is marked as a Controller-capable Device in its BCR register. The Controller Role Request may be made by a Secondary Controller wishing to gain the Controller Role from the Active Controller, or by the Primary Controller after it has relinquished the Controller Role and now wishes to regain it from the Active Controller. The Controller Role Request shall include the Device's Dynamic Address with a RnW bit of 0.



Figure 26 Controller Role Request Frame

C. A Hot-Join Request

An I3C Target shall only make such a request when becoming available after the I3C Bus is operational. The Hot-Join Request shall be made using the reserved Hot-Join Address (i.e., 7'h02).



Figure 27 Hot-Join Request

The I3C Targets are required to make requests to the I3C Controller in only two Bus conditions. Firstly, in the **Bus Free Condition**, a START is issued on the Bus. The Target may then transmit its Dynamic Address or the Hot-Join Address (7'h02) following the START by adhering to the I3C Address Arbitration rules. Secondly, in the **Bus Available Condition**, the Target may issue a START by pulling the SDA Low. The Controller shall then pull SCL Low within tCAS and also pull SDA Low (overlapping the Target pulling it Low). Once the Controller has pulled SCL Low, the Target shall control the SDA line in Open Drain mode (i.e., either pull Low or release High) and finally issue its Address in the normal way.

2.5.4.1 Consequence of Starting a Frame with Target Address

The I3C Controller normally should start a Frame with 7'h7E (for all I3C Messages) or an I²C static address (when sending only to a legacy I²C target). However, in both cases, the address may undergo arbitration. Hence, the Controller must monitor the line to identify any In-Band Interrupt request, Controller Role Request (such as a Secondary Controller requesting to become the Active Controller), or Hot-Join Request. If no such request is detected, the Controller can proceed normally. However, if any of these requests are detected, the Controller should ACK or NACK the request and proceed accordingly. If the Controller intends to start an I3C Message with an I3C Dynamic Address, special provisions must be made as the same I3C Target may initiate an IBI or a Controller Role Request, which may result in one of three things happening:

1. The Addresses match, but the difference is caught on the RnW bit, and the Controller was initiating a Private Write (i.e., RnW = 0) while the Target was attempting to send an IBI request (i.e., RnW = 1). In this case, the Controller wins (i.e., an IBI with RnW = 1 loses), and the Target shall ACK or NACK the Private Write. The Target shall defer its IBI and may retry at a later opportunity.

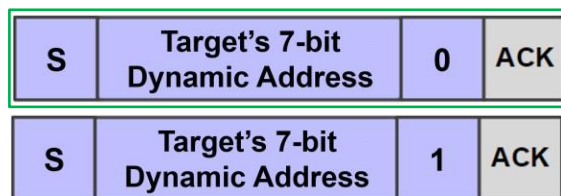


Figure 28 Private Write by I3C Controller VS IBI Request by I3C Target

2. The Addresses match, but the difference is caught on the RnW bit, and the Controller was initiating a Private Read (i.e., RnW = 1) while the Target was attempting to send a Controller Role Request (i.e., RnW = 0). In this case, the Target wins (i.e., a Private Read with RnW = 1 loses), and the Controller must ACK or NACK the Controller Role Request. The Controller shall defer its Private Read and may retry at a later opportunity.

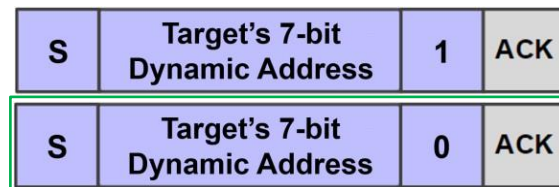


Figure 29 Private Read by I3C Controller VS Controller Role Request by I3C Target

3. The Addresses match and the RnW bits also match, and so neither Controller nor Target will ACK since both are expecting the other side to provide ACK. As a result, each side might think it had "won" arbitration, but neither side would continue, as each would subsequently see that the other did not provide ACK.

If RnW = 0 (i.e., if the Controller was initiating a Private Write while the Target was attempting to send a Controller Role Request), then the Target shall defer its Controller Role Request and may retry at a later opportunity.

If RnW = 1 (i.e., if the Controller was initiating a Private Read while the Target was attempting to send an IBI request), then the Target shall defer its IBI request, and may retry at a later opportunity.

S	Target's 7-bit Dynamic Address	0	ACK
S	Target's 7-bit Dynamic Address	0	ACK

Figure 30 Private Write by I3C Controller VS Controller Role Request by I3C Target

S	Target's 7-bit Dynamic Address	1	ACK
S	Target's 7-bit Dynamic Address	1	ACK

Figure 31 Private Read by I3C Controller VS IBI Request by I3C Target

For either value of RnW: Due to the NACK, the Controller shall defer the Private Write or Private Read and should typically transmit the Target Address again after a Repeated START (i.e., the next one or anyone prior to a STOP in the Frame). Since the Address Header follows a Repeated START and is not arbitrated, the Controller will always win.

2.5.4.2 I3C Target Address Restrictions

The I3C Controller may choose Dynamic Addresses from a set of values as follows:

Table 6 I3C Target Address Restrictions

Target Dynamic Address		Restriction	Description
Binary	Hex		
000 0000	7'h00	Shall not use	I3C Reserved
000 0001	7'h01	Shall not use	I3C Reserved: For use with SETDASA CCC in special Point-to-Point Communication
000 0010	7'h02	Shall not use	I3C Reserved: Hot-Join Address
000 0011	7'h03	Optional	Marked 'Reserved' by PC
000 0100	7'h04	Conditional	Available for use only if no Legacy I ² C Devices supporting I ² C "High-Speed Mode" are present on the Bus
000 0101	7'h05		
000 0110	7'h06		
000 0111	7'h07		
000 1000	7'h08 – 7'h3D	Available for use	54 Addresses
011 1101			
011 1110	7'h3E	Shall not use	I3C Reserved: Broadcast Address single bit error detect
011 1111	7'h3F – 7'h5D	Available for use	31 Addresses
101 1101			
101 1110	7'h5E	Shall not use	I3C Reserved: Broadcast Address single bit error detect
101 1111	7'h5F – 7'h6D	Available for use	15 Addresses
110 1101			
110 1110	7'h6E	Shall not use	I3C Reserved: Broadcast Address single bit error detect
110 1111	7'h6F – 7'h75	Available for use	7 Addresses
111 0101			
111 0110	7'h76	Shall not use	I3C Reserved: Broadcast Address single bit error detect
111 0111	7'h77	Available for use	1 Address
111 1000	7'h78	Conditional	Available for use only if no Legacy I ² C Devices are present on the Bus
111 1001	7'h79		
111 1010	7'h7A	Shall not use	I3C Reserved: Broadcast Address single bit error detect
111 1011	7'h7B	Conditional	Available for use only if no Legacy I ² C Devices are present on the Bus
111 1100	7'h7C	Shall not use	I3C Reserved: Broadcast Address single bit error detect
111 1101	7'h7D	Conditional	Available for use only if no Legacy I ² C Devices supporting I ² C "Device ID Mode" are on the Bus
111 1110	7'h7E	Shall not use	I3C Reserved: Broadcast Address single bit error detect
111 1111	7'h7F	Shall not use	I3C Reserved: Broadcast Address single bit error detect

2.6 SDR Data Word

In I3C SDR, the Data Words match I2C only in the sense that they are both 9 bits long, I3C SDR Data Words differ from I2C in three ways:

1. Handoff from Address ACK to SDR Controller Write Data.
2. Ninth Bit of SDR Controller Written Data as Parity.
3. Ninth Bit of SDR Target Returned (Read) Data as End-of-Data.

2.6.1 Handoff From Address ACK to SDR Controller Write Data

The end of any Address Header (whether Arbitrated or not) is an ACK or NACK by the one or more addressed Targets, using Open Drain on SDA. When the Address Header results in an ACK, and the Message is SDR Write from Controller, the SDA line has to be turned from Open Drain to Push-Pull for the first data bit. To do that safely, I3C SDR specifies how the handoff is to occur.

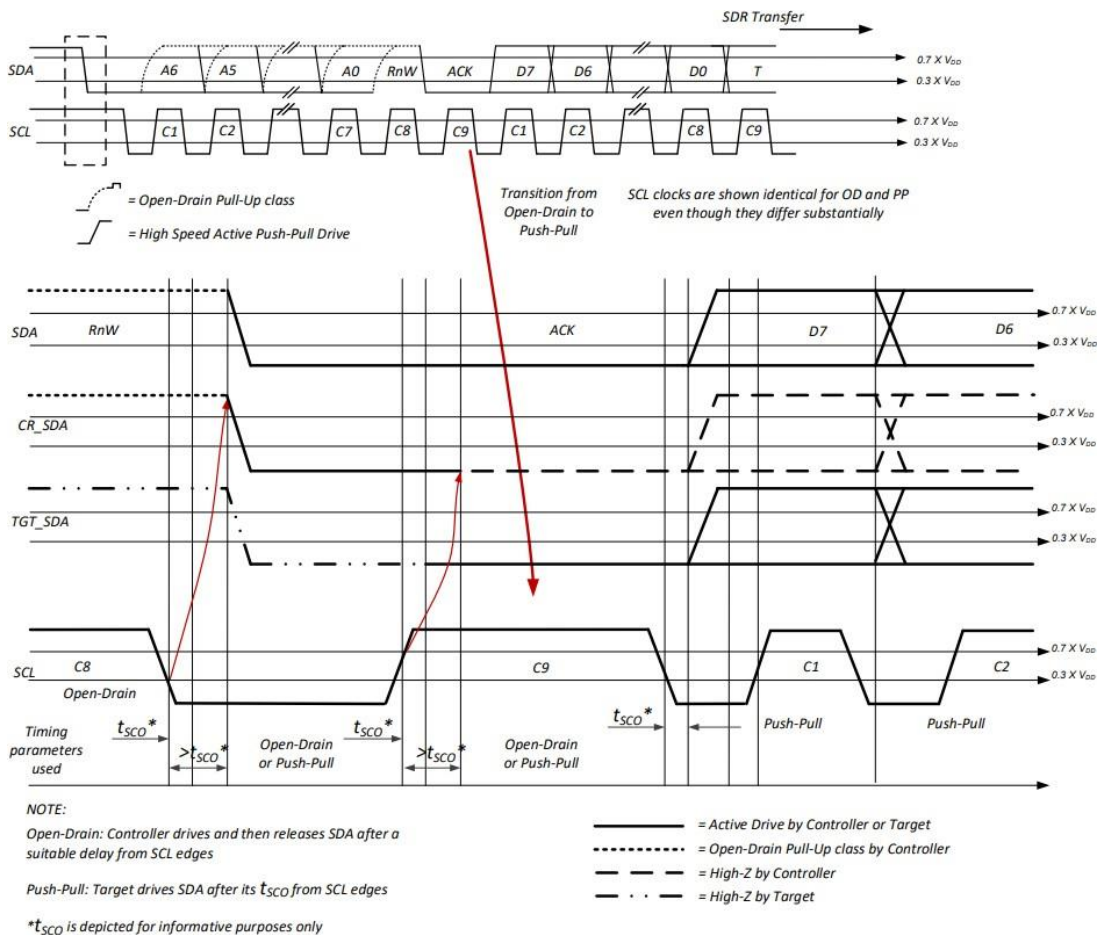


Figure 32 Transition from Address ACK to Mandatory Byte During IBI

1. The I3C Target shall hold the SDA line Low during the ACK (while SCL is Low).
 - This shall be an Open Drain SCL Low period.
2. After the I3C Target sees the rising edge of SCL, it releases the SDA line to High-Z.
 - The I3C Target shall release the SDA line using normal (Push-Pull) timing (release the SDA line as soon as it sees SCL rising).
3. After the rising edge of SCL, the I3C Controller shall drive the SDA line Low.
 - As a result, both Controller and Target will be driving the SDA line Low for a short overlap (which is safe).
4. On the falling edge of SCL the I3C Controller shall begin driving data on the SDA line, using Push-Pull drive. When the Address Header results in a NACK, the Controller may choose to:

either:

 1. Continue the transaction, by generating a Repeated START.

or:

 2. Relinquish the Bus, by generating a STOP.

2.6.2 Ninth Bit of SDR Controller Written Data As Parity

In I2C, the ninth Data bit written by the Controller is an ACK by the Target. By contrast, in I3C the ninth Data bit written by the Controller is the Parity of the preceding eight Data bits. Therefore, in I3C the Target shall not drive the SDA line for Data written by the Controller in SDR. In SDR terms, the ninth bit of Write data is referred to as the T-Bit (for ‘Transition’).

T (Parity) bit writes shall always be kept valid through the SCL High period. In the case of a T-Bit representing the last data byte, the write is therefore kept valid through the SCL High period, and the next SCL Low can then be used to either change the SDA, or not change the SDA, in preparation for the Repeated START or STOP that follows.

The ninth data bit of each SDR Data Word written by the I3C Controller (also referred to as the T-Bit) is a Parity bit, calculated using odd parity. Parity can help in detecting noise- caused errors on the line. The value of this Parity bit shall be the XOR of the 8 Data bits with 1.

- **A:** Data[7:0]
- **B:** 1
- **out:** odd Parity

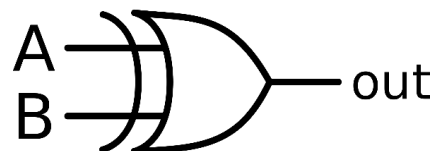


Figure 33 Parity Bit Using X OR

2.6.3 Ninth Bit of SDR Target Returned (Read) Data as End-of-Data

In I2C, the ninth Data bit from Target to Controller is an ACK by the Controller. By contrast, in I3C this bit allows the Target to end a Read and allows the Controller to Abort a Read. In SDR terms, the ninth bit of Read data is referred to as the T-Bit (for ‘Transition’)

In I2C, Read from Target has the issue that only the Controller ends the Read, so the Target has no ability to control the amount of data it returns. In I3C SDR, by contrast, the Target controls the number of data Words it returns; but it also allows the I3C Controller to abort the Read prematurely when necessary. This mechanism is controlled purely by the ninth (T) Data bit of each SDR Data Word returned by the I3C Target.

The ninth bit is returned by the Target in one of two ways:

1) The I3C Target returns the ninth bit as 0 (SDA Low) to end the Message:

- The Target shall set SDA Low on the falling edge of SCL.
- On the following rising edge of SCL, the Target shall set SDA to High-Z.
- The I3C Controller shall drive SDA Low on the rising edge of SCL, thereby overlapping with the Target.
- The I3C Controller then shall issue either a STOP, or a Repeated START.

2) The I3C Target returns the ninth bit as 1 (SDA High) to continue the Message (and permit the Controller to abort the Message):

- The Target shall set SDA High on the falling edge of SCL.
- On the following rising edge of SCL, the Target shall set SDA to High-Z.

Thereby Parking the Bus for the SCL High period:

- i. If the I3C Controller is able to continue the reply from the Target, then it shall do nothing.
- ii. If the I3C Controller wants to abort the Message, then it shall drive SDA Low after the rising edge of SCL.

The I3C Target returns the ninth bit as 1 (SDA High) to continue the Message (and permit the Controller to abort the Message):

- The Target shall monitor the SDA on the falling edge of SCL:
 - If SDA is High, then the Target shall continue with the next data value.
 - If SDA is Low (i.e., if there has been a Repeated START), then the Message has been aborted, and the Target shall not drive SDA after that.

Chapter 3: Main Features

3.1 Dynamic Address Assignment

Addressing in I3C protocol is dynamic, unlike I2C. which means that I3C targets will have addresses assigned to them dynamically after they've already joined the bus.

We will discuss the operation, limitations, and sequence of Dynamic address assignment procedure that occurs during the bus initialization.

The Primary Controller acts as the authority for the initial configuration of the Bus and all Devices, including any Legacy I2C Devices. Since it acts as the Bus's first Active Controller, during Bus initialization. it also performs the Dynamic Address Assignment procedure as part of configuring all I3C Devices that require a Dynamic Address.

a Controller must be responsible for performing a Dynamic Address Assignment

procedure, in order to provide a unique Dynamic Address to each I3C Device (i.e., with Target or Secondary Controller role) that is connected to the Bus.

Once a Target or Secondary Controller receives a Dynamic Address, that Dynamic Address shall be used in all subsequent transactions on the I3C Bus.

The Controller controls the Dynamic Address Assignment process. This process includes an Address Arbitration procedure like I2C's. The I3C Arbitration procedure differs from I2C by using the values of the 48-bit Provisioned ID and the Device's I3C Characteristic Registers (that is, BCR and DCR), concatenated. The Device on the I3C Bus with the lowest concatenated value wins each Arbitration round in turn, and the Controller assigns a unique Dynamic Address to each winning Device.

3.1.1 Data Given From The Host

Before the primary controller starts the dynamic address assignment procedure, information about the bus from the host should be given. This information includes:

- a) Number of I3C targets on the bus, in addition to number of I3C targets that have I2C static addresses.
- b) Number of I2C targets on the bus.
- c) Static addresses of both I3C targets and I2C targets.

3.1.2 Address Assignment Rules

There are some addresses restricted from usage in dynamic address assignment due to different reasons, mainly due to I3C reserved addresses, or a single bit error in similar addresses that will lead to one of the I3C reserved addresses.

- 7'h (00 , 01 , 02 , 7E , 7F) are all I3C reserved addresses.
- 7'h (3E , 5E , 6E , 76 , 7A , 7C , 7F) are all restricted because if a single bit error occurs it will be seen as an I3C reserved address.

Address assignment follows the incremental rule beginning from the address **7'h08**. Meaning the 1st winning device will receive the target address **7'h08**, the 2nd winning device will receive the following address 7'h09 and so on. The last address to be used is **7'h3D**.



Figure 34 Dynamic Addresses Available for Use

3.1.3 Dynamic address assignment procedure

The Dynamic Address Assignment process is fully performed in Open Drain mode, For Open Drain the Controller shall drive the SCL line with clocks at the appropriate Open Drain speed for the Devices present on the I3C Bus.

- 1) The active controller starts the procedure by initiating a start, then sending the Broadcast 7'h7E (Write), to inform the devices on the bus that the controller will hold the bus and send data.



Figure 35 Broadcast Write For DAA

- 2) After receiving the acknowledgement from any device on the bus, the I3C controller initiates the **ENTDAA CCC (0x07)** accompanied with the parity bit, which informs the devices on the bus about beginning the dynamic address assignment procedure.

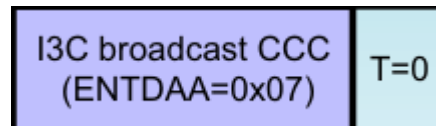


Figure 36 ENTDAA CCC

- 3) The active controller initiates a repeated start, to send again the Broadcast 7'h7E (read) but with a read bit this time, that informs the targets that the arbitration process will start and that they should send their data according to the arbitration rules.

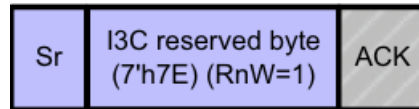


Figure 37 Broadcast Read For Target Data

- 4) The Active Controller shall drive only the SCL line. and releasing the SDA line to a High-Z state, allowing SDA to go to High level via the Bus Pull-Up resistor. To allow the devices on the bus to start the arbitration process and driving the SDA.

Every I3C Device that is eligible to participate and responds to the I3C Broadcast Address sent in The last step shall drive the SDA line with its own 48-bit Provisioned ID (using Big Endian bit order), until it loses Dynamic Address Arbitration.

The 48-bit Provisioned ID shall be transferred continuously, starting with the most significant bit (bit[47]), with no delimitation or ACK/NACK pulse.

After that the I3C Device that did not yet lose the Arbitration shall then transfer its Bus Characteristics Register (BCR) and Device Characteristic Register (DCR) until it eventually loses Dynamic Address Arbitration.

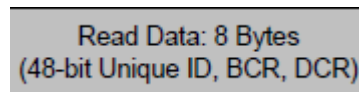


Figure 38 Target's 8 Bytes Data

The device whose concatenated Provisioned ID, BCR, and DCR have the lowest value will win the arbitration round, due to the nature of arbitration.

- 5) The active controller regains control over the SDA line, in order to send the assigned address to the winning device, the address is 7-bit wide chosen according to the addressing rules specified in the last section. The 8th bit is a parity bit of the serialized address. The target then shall ACK the received address if the parity check is correct.

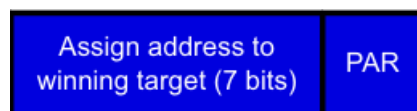


Figure 39 Dynamic Address Assigned with Odd Parity

- 6) In this step, the I3C controller has managed to assign a dynamic address to the 1st winning device, so it shall repeat the procedure again from step 3, sending the Broadcast command 7'h7E with (read) bit. To continue with the other devices on the bus (that have lost the arbitration)
If the controller receives ACK to the broadcast, this means that there is targets that yet to be assigned an address. While if the controller received NACK to the broadcast, this means that all the targets on the bus have already been assigned an address and it should end the dynamic address process.
- 7) If the controller received NACK to the broadcast in the last step, it would end the dynamic address procedure buy initiating a STOP on the bus (SDA Line).

3.1.4 ENTDAAC Frame Format

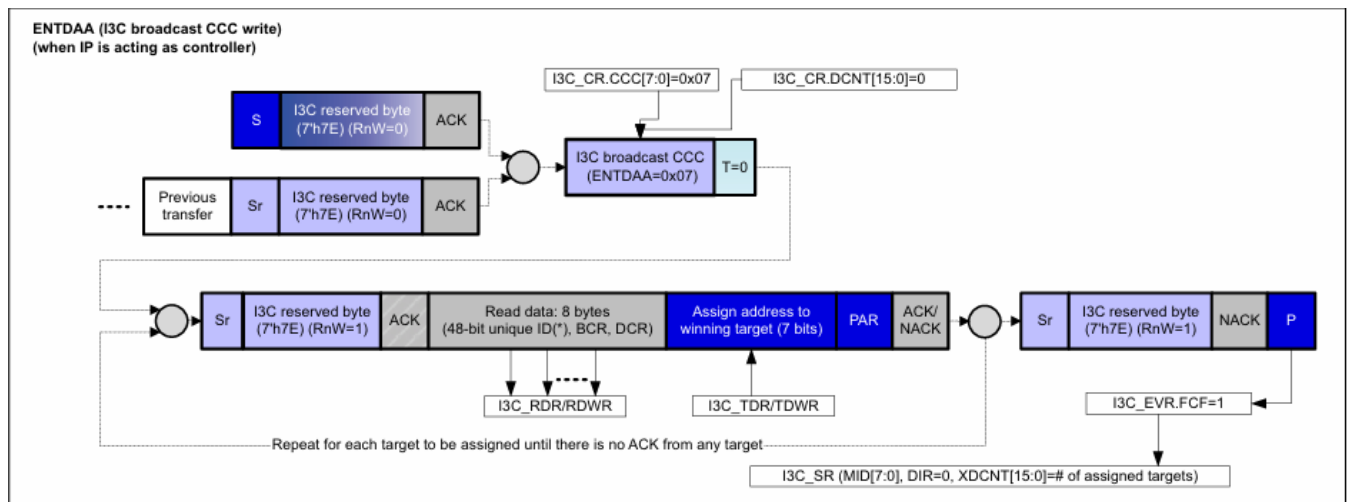


Figure 40 I3C broadcast ENTDAAC CCC– Sent by the Controller

3.1.5 Dynamic Address Assignment Error Case and Handling

As mentioned in step 5 in the dynamic address assignment procedure, the controller assigns the address to the winning device accompanied by the parity bit and waits for ACK/NACK from the target.

- ✓ If the Target ACKS, the address then the parity is correct, and address is assigned.
- If the target NACKS the address then the parity is not correct, and the target should not take this address as the assigned dynamic address.

In the last case, the controller repeats the flow again, the last winning device will win the arbitration again due to the nature of arbitration. Then the controller will assign the same address to the target.

The error case occurs if the same target NACKS the same address twice. In this case the controller should terminate the dynamic address assignment and notify the error handling mechanisms. Then redo the dynamic address assignment procedure again, but, only for the devices that did not yet receive their dynamic address.

3.2 Hot-Join Mechanism

The I3C protocol allows devices to join the bus after it has already configured via the hot join mechanism. The hot-join mechanism is that a hot joining device informs the controller that it has joined the bus after the configuration, so it needs to receive a dynamic address in order to be able to participate in bus transactions.

Hot-Join may be used for:

- I3C Devices mounted on the same board, but de-powered until needed. Such Devices shall not violate electrical limits for Targets when de-powered (or while transitioning).
- I3C Devices mounted on a module/board that is physically inserted after the I3C Bus has already been configured. This specification does not attempt to address how that physical insertion is handled; however, such insertion shall not disrupt the SCL and SDA lines, including respecting all electrical limits.

3.2.1 Hot-Join Flow Procedure

- 1) For a hot joining target to inform the controller of joining the bus, it must wait for the suitable time (after a start to participate in arbitration) to send an IBI request, with the reserved address for hot-join **7'h02** with a **Write** bit.

Unlike a typical IBI request, hot-join request doesn't have a data payload, or a mandatory data byte. This address basically requesting a dynamic address assignment procedure.

- 2) The active controller shall either ACK the request indicating that it's seen it and at some point, it will begin the dynamic address assignment procedure again or NACK the request, which is rejecting the hot join for now and the device must try again later.
 - In the case of ACK. The target knows that the controller accepted the request, and it must wait for the dynamic address procedure to start, without sending any more requests.
 - But if the controller NACKs the request, the hot-joining device has the right to re-initiate it again in the eligible time.
- 3) The active controller should eventually send the CCC ENTDAAs, for starting the dynamic address assignment procedure, only targets that have not yet received a dynamic address will participate, including hot-joining devices.

Controller ACK in a Hot-Join Request does not imply that the Dynamic Address Assignment will necessarily start immediately. The Controller may wait for a potentially prolonged period before issuing the ENTDAAs CCC.

3.2.2 Hot-Join Frame Format

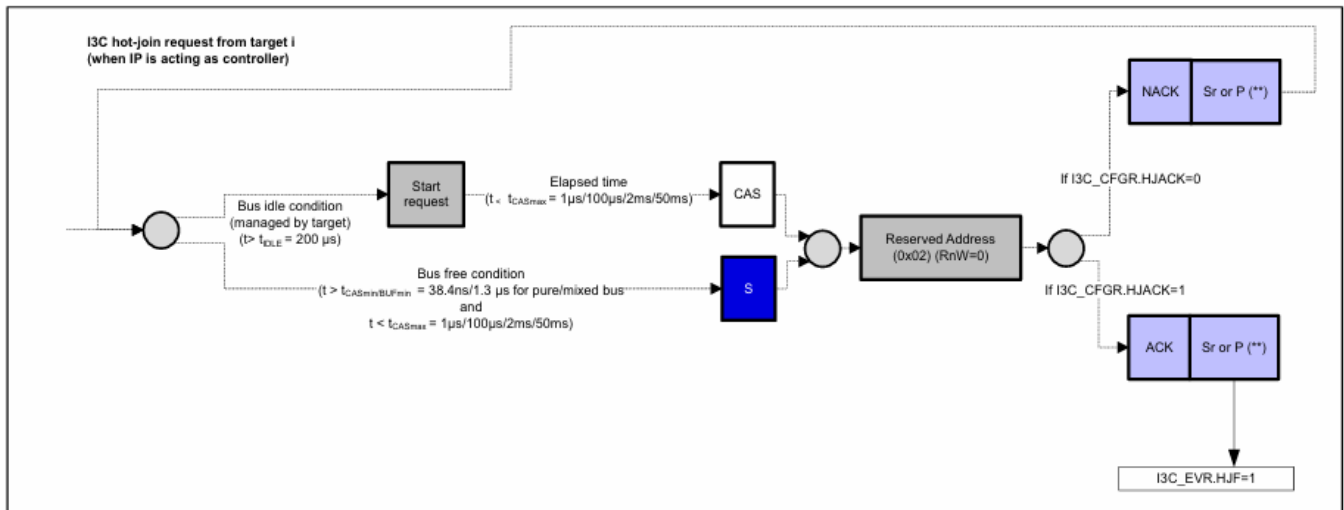


Figure 41 Hot-join request transfer, as controller/target

3.2.3 Hot-Join Target Requirements

- 1 The first time the Hot-Joining Target connects to the I3C Bus, the Target shall wait for the appropriate opportunity to send the Hot-Join Request, based on its eligibility, a Hot- Joining Target using the standard method shall become eligible after waiting for a period of at least **Bus Idle Condition (tIDLE)** and ensuring that the Bus remains Idle (i.e., that SDA and SCL are both High).
- 2 The Hot-Joining Target shall, when eligible, send the Hot-Join Request on the I3C Bus, following a START.
The Target shall send the Hot-Join Request, by either waiting for a START, or requesting a START by pulling SDA Low (for the standard method only). After the START, the Target shall drive the Address of **7'h02/W** into the Arbitrable Address Header.
- 3 The Hot-Joining Target shall conditionally continue sending the Hot-Join Request on each subsequent START (i.e., at the next appropriate opportunity) until and unless the Active Controller provides an ACK for the Hot-Join Request.
- 4 Once the Hot-Joining Target sees the Active Controller ACK or NACK the Hot-Join Request, it shall follow all normative requirements for the Role of an I3C Target (**i.e., one that has not yet received its Dynamic Address**).

The passive Hot-Join method works the same basic way, except with a modification to the conditions of eligibility for a Hot-Joining Target. A standard Hot-Joining Target must first wait for at least the Bus Idle Condition, and then it may pull SDA Low, or it may wait for a START. A passive Hot-Joining Target will first wait for an SDR Frame ending in STOP, and then wait for the next START, which must be issued by either the Active Controller or any other Target that can request a START.

Table 7 Hot-Join Standard Vs Passive Mehods

Hot-Join Request	
Standard Method	Passive Method
<ul style="list-style-type: none"> • Wait for bus idle (both SCL and SDA are high for more than tidle). • Target pulls SDA = 0 or wait for START. • Wait for Controller SCL=0. • Send 7'h02/W. • Wait for ACK/NACK. 	<ul style="list-style-type: none"> • Target doesn't have a timer. • Must wait for a START request b another target, or the controller after the end of SDR frame. • After START it sends 7'h02/W • Has higher delay than standard method

3.3 In-Band Interrupt

3.3.1 Priority level

It's the level that controls the order in which in band interrupt requests are processed. During dynamic address assignment, the Controller assigns lower addresses to Targets with higher priority that arbitrated sooner so they also can request IBI or control requests sooner. So, targets with lower addresses have higher priorities.

3.3.2 Interrupt request

I3C target shall wait for a start condition or issue a start if the bus is in the bus available condition by pulling SDA line low and waiting for the controller to pull SCL low. After the start condition it can put its address on the line and participate in the arbitration process. After the start condition the target drives SDA line with its address followed by RnW bit and it should be one at the interrupt request. The controller processes the IBI requests using priority level order and the target that lost arbitration may issue another IBI request in the next bus available condition. When the controller receives interrupt request it can do one of three actions:

- Refuse IBI request without disabling interrupts:
Controller can do so by sending NACK to the target after receiving the Interrupt request and target can try again after START or Bus available condition.
- Refuse IBI request without disabling interrupts:
Controller can do so by sending NACK to the target and sends a repeated start, then it sets the DISINT bit in DISEC CCC "Disable target events" so the target will not send the IBI request again until the target events is enabled again .
- Accept the IBI request:
Controller accepts the IBI request by sending ACK bit after receiving the request. Controller's action after the IBI depends on the Target's BCR [2] bit, If the BCR[2]=0 Controller has to read the mandatory data byte MDB which contains some information about the event that happened and the size of the data to be received from the target, After the MDB, target can send additional IBI data bytes and the controller can accept these bytes or terminate it. Mandatory data byte is sent using push pull and here is a figure representing the frame of the IBI sequence.

Open Drain	Open Drain	Open Drain	Hand Off	Push-Pull	Drive High or Low, and then High-Z	Push-Pull
S	Target_addr_as_IBI/R	Controller_ACK	SCL High	Target_byte	T	Sr

Figure 42 IBI sequence with mandatory data byte

If the BCR[2] = 0 in the target, then it doesn't have MDB, and the controller may take any valid action to terminate the frame after providing the ACK bit using STOP condition.

3.3.3 Mandatory Data Byte (MBD)

The mandatory data byte is the data that follows the dynamic address when a device sends an IBI request. Availability of the mandatory data byte is determined by the BCR[2] register in the bus characteristic register. The target takes over the line after the IBI ACK and Controller can't decline this byte and must wait for T-bit to terminate any subsequent data. The MDB gives the controller additional information about the event that has happened and what information the target wants to send. The MDB is divided into two fields:

- **Interrupt Group Identifier:** the three most significant bits MDB[7:5].
- **Specific Group Identifier:** the five least significant bits MDB[4:0].

Interrupt Group Identifier Field			Specific Interrupt Identifier Field				
MDB[7]	MDB[6]	MDB[5]	MDB[4]	MDB[3]	MDB[2]	MDB[1]	MDB[0]

Figure 43 MDB Field Format

All the values of the MDB Identifier values and their description are specified in MIPI Specification for I3C Basic Version 1.1.1 09-Jun-2021 section 5.1.6.2.1.

3.3.4 IN-Band Interrupt Frame Format

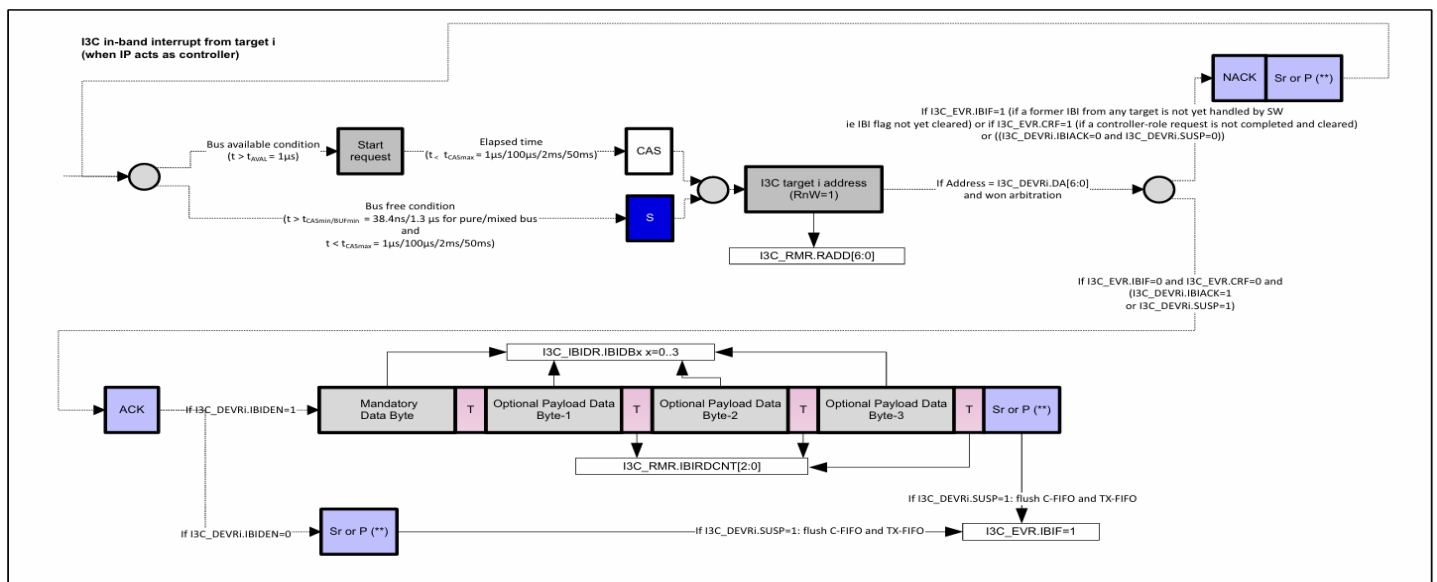


Figure 44 IBI transfer, as controller/target

3.4 Common Command Codes (CCCs)

Common Command codes (CCCs) are standardized set of commands that act like a contract between the controller and the targets used to communicate with each other. Most of the commands have the same frame structure except a few according to STMicroelectronics standard.

They can be classified as follows:

- 1- Broadcast CCCs: those CCCs are addressed to all I3C targets on the bus, and they're always written CCCs.
- 2- Direct CCCs: those CCCs are addressed to a specific target on the bus, and they can be Read, write, and Read/write CCCs.

3.4.1 Enter Activity State 0-3 (ENTAS0-ENTAS3)

Those 4 CCCs can be Broadcast Write or Direct Write. They're used to inform one or all targets that the active controller won't be active for a specific time based on which ENTAS is sent. The command codes for Enter Activity State 0-3 can be summarized in the next table:

Table 8 Enter Activity State 0-3 Command Codes

Command	Support	Command Codes		Purpose
		Broadcast	Direct	
ENTAS0	Conditional	0x02	0x82	Enter Activity State 0
ENTAS1	Optional	0x03	0x83	Enter Activity State 1
ENTAS2	Optional	0x04	0x84	Enter Activity State 2
ENTAS3	Optional	0x05	0x85	Enter Activity State 3

Minimum Bus activity interval for each Activity state can be summarized in the next table:

Table 9 Minimum Bus activity interval

CCC	Activity State	Minimum Bus Activity Interval
ENTAS0	Activity State 0	1 µs: Latency-free operation
ENTAS1	Activity State 1	100 µs
ENTAS2	Activity State 2	2 ms
ENTAS3	Activity State 3	50 ms: Lowest-activity operation

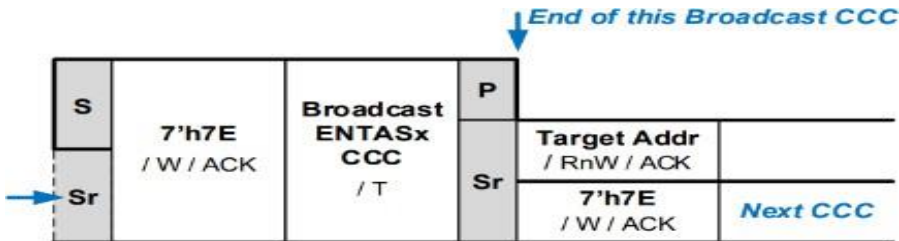


Figure 45 ENTASx Broadcast Format

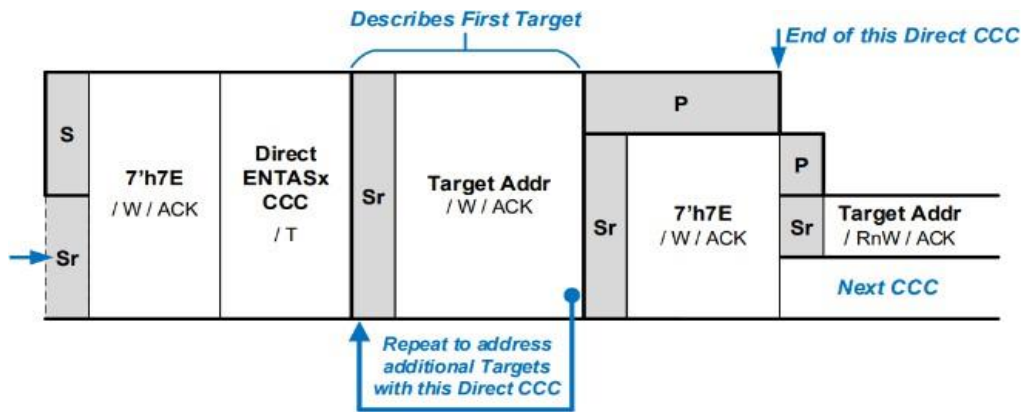


Figure 46 ENTAS Direct Format

3.4.2 Get Device Status (GETSTATUS)

It's a direct CCC used to get request from a particular target to return its current status and its command code is 0x90.

It has 2 formats:

- 1- GETSTATUS format 1 returns the two bytes consisting of MSB and LSB.

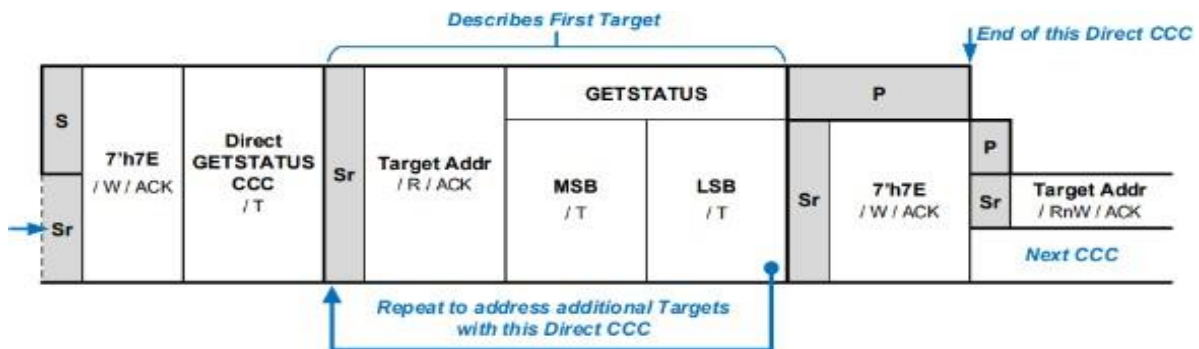


Figure 47 GETSTATUS Format 1

Table 10 GETSTATUS MSB-LSB

Byte	Bits	Field	Description
MSB	15:8	Vendor Reserved	Reserved for vendor-specific meaning.
LSB	7:6	Activity Mode	Contains the two-bit ID of the Target Device's current Activity Mode (i.e., its readiness to support data read of sensor or related information). A Controller-capable Device (i.e., Secondary Controller) must use this field to indicate when it is unable to participate in any of the steps to prepare for Handoff of the Controller Role (see Section 5.1.7.1). If such a Device is currently unable to participate, then it shall return a value of 2'b11. Any other value shall indicate that the Device may be ready to participate. For all other Target Devices, the meanings of the four possible values are not defined by this Specification; instead, they depend upon a private contract between the Controller and each Target.
	5	Protocol Error	1'b1: The Target detected a protocol error since the last Status read. The Target might or might not be able to check for such errors. Note that this value self-clears upon every successful completion of a Controller read of the Target's Status.
	4	Reserved	Reserved for future definition by MIPI Alliance I3C WG.
	3:0	Pending Interrupt	Contains the interrupt number of any pending interrupt, or 0 if no interrupts are pending. This encoding allows for up to 15 numbered interrupts. If more than one interrupt is set, then the highest priority interrupt shall be returned.

- 2- GETSTATUS format 2 with a defining byte returns a variable number of Bytes depending on the defining byte used. For example, using PRECR defining byte which is used to allow the active controller to query the secondary controller about its current state whether it entered a deep sleep mode or still processing data.

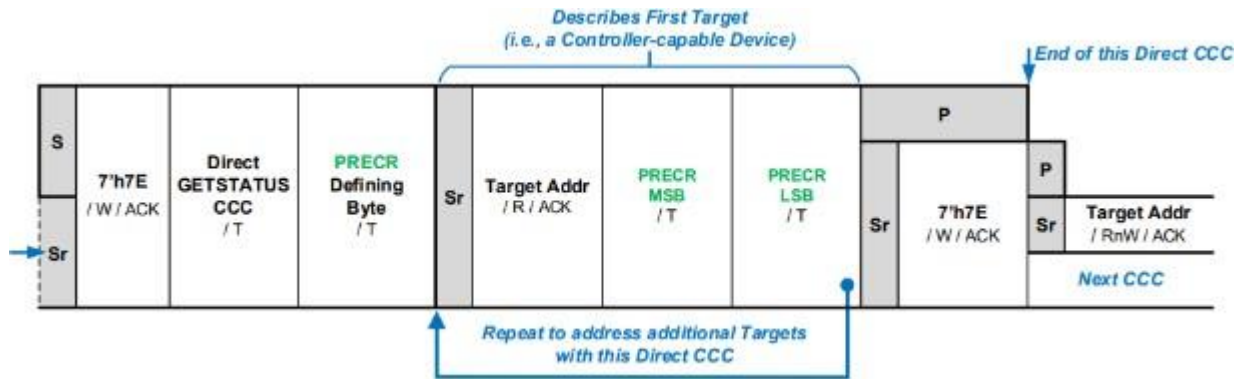


Figure 48 GETSTATUS Format 2 using PRECR

Table 11 GETSTATUS MSB-LSB

Byte	Bits	Field	Description
MSB	15:8	Vendor Reserved	Reserved for vendor-specific meaning.
	7:2	Reserved	Reserved for future definition by MIPI Alliance I3C WG.
LSB	1	Handoff Delay NACK	<p>This bit indicates whether this Device is currently processing any DEFTGTS CCC (and DEFGRPA CCC, if supported) Broadcasts that may have been sent by the Active Controller. It is expected that this Device might take significant time to process the data in such Broadcasts.</p> <p>1'b1: The Active Controller may wait to send GETACCCR CCC to this Device, or should at least be aware that any attempts to send GETACCCR CCC to this Device will be met with a NACK response, until this bit is read again later with a value of 1'b0, to indicate that the Device has finished processing the data in these Broadcasts and has updated its internal state.</p> <p>1'b0: This Device is not currently processing Broadcast data, or has finished processing this data; in either case, it can safely accept the Controller Role.</p>
	0	Deep Sleep Detected	<p>This bit indicates whether this Device has entered a "deep sleep" state, in which it might have missed any DEFTGTS CCC (and DEFGRPA CCC, if supported) Broadcasts sent by the Active Controller. Consequently, this Device's internal state of known Target Devices (and known Group Addresses, if applicable) should be considered outdated.</p> <p>1'b1: The Active Controller is obligated to send a fresh Broadcast of DEFTGTS CCC (and DEFGRPA CCC, if applicable and supported) to update this Device's internal state before sending GETACCCR CCC to this Device.</p> <p>1'b0: This Device has not entered a "deep sleep" state, or is not capable of doing so.</p>

3.4.3 Target Reset Action (RSTACT)

This Broadcast, Direct Read, and Direct Write CCC is used to configure the next Target Reset action and may be used to retrieve a Target's reset recovery timing. The RSTACT CCC is used in conjunction with the Target Reset Pattern, i.e., the reset action previously configured in a Target via the RSTACT CCC is triggered when the immediately following Target Reset Pattern is received.

- For the Broadcast and Direct Write formats, the Defining Byte indicates which Target Reset action (including taking no action) is to be configured (values 0x00 through 0x7F).
 - Defining Bytes 0x00 and 0x01 are required: A Target shall support these operations and shall ACK its Target Address if issued as a Direct CCC.
 - Support for other Defining Bytes (values 0x02 through 0x7F) is optional and depends on other conditions or support for other capabilities. If a Target does not support such an operation, then it shall NACK its Target Address for such a Defining Byte, as well as any related Defining Bytes defined for the Direct Read format (values 0x82 through 0xFF) if issued as a Direct CCC.

- For the Direct Read format, the Defining Byte may also indicate the Controller's desire to read back the Target's reset recovering timing or other parameters for the operation (values 0x81 through 0xFF).
 - If the CCC is NACKed and the related reset operation is supported, then the Controller should assume the default reset return times of 1 ms to reset the Peripheral (i.e., the reset operation for Defining Byte 0x01) and 1 second to reset the whole Target Device (i.e., the reset operation for Defining Byte 0x02).

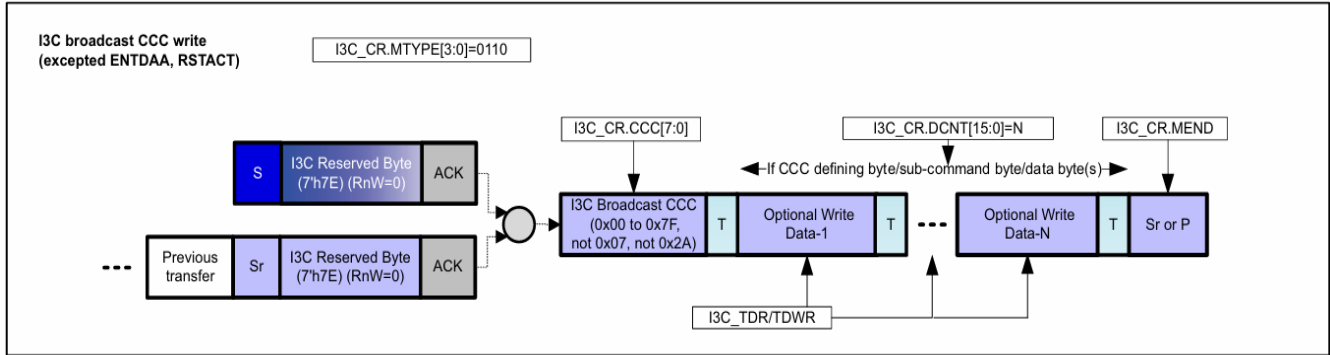


Figure 49 I3C broadcast CCC write (excepted ENTDA, RSTACT)

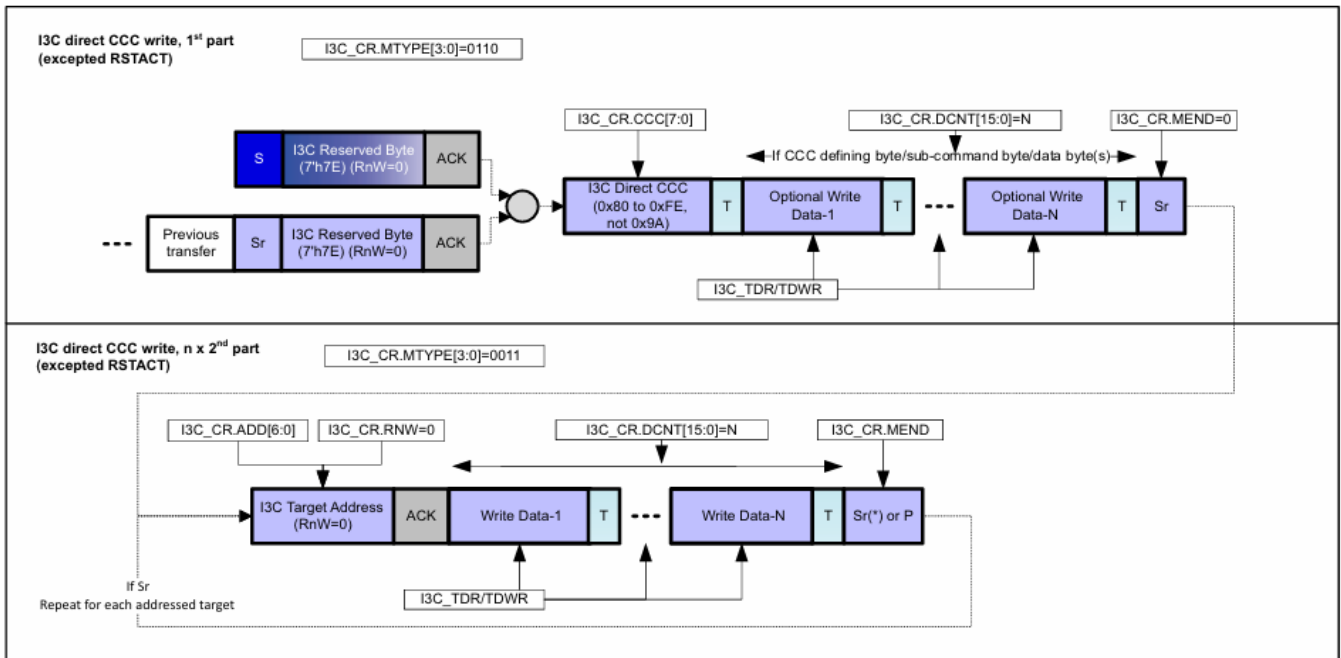


Figure 50 "I3C Direct CCC Write – 1st Part / Multiple 2nd Parts (Except RSTACT)"

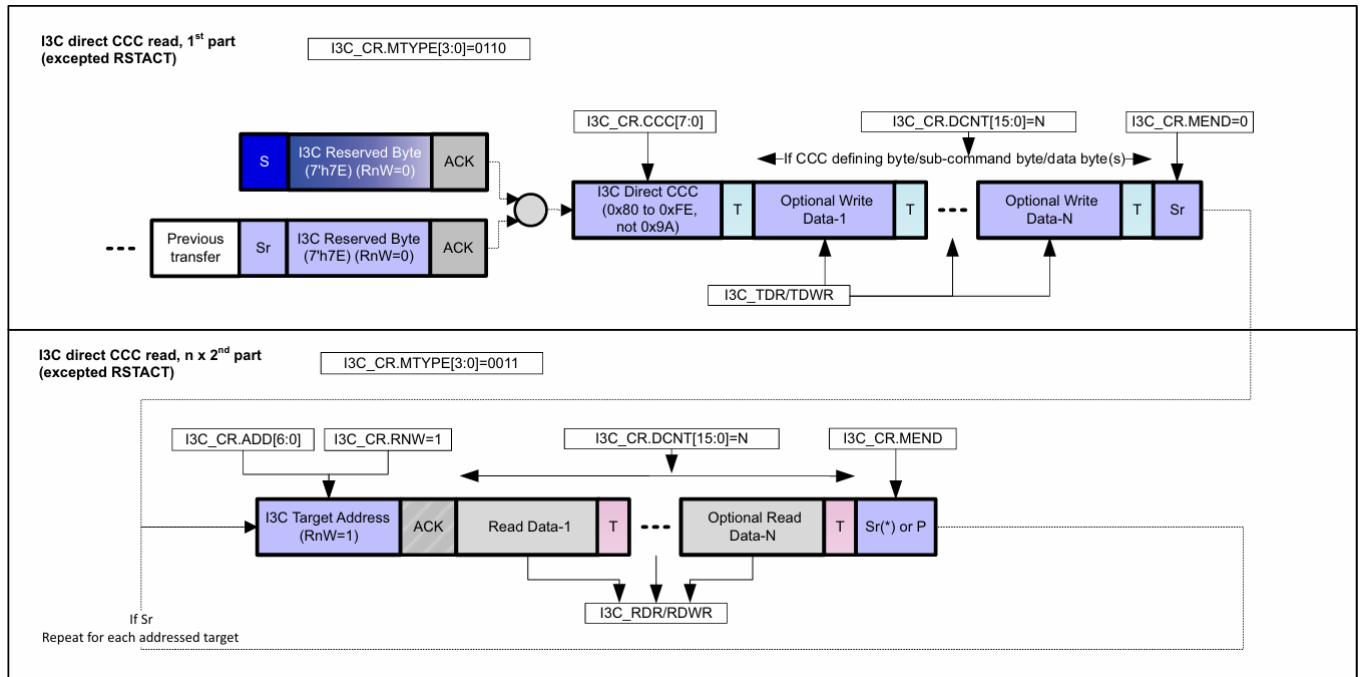


Figure 51 "I3C Direct CCC Read – 1st Part / Multiple 2nd Parts (Except RSTACT)"

Table 12 List of supported I3C CCCs, as controller/target

CCC name	CCC value	Read /write	With/without defining byte With/without sub-command byte	With/without optional data byte(s)	Use as controller	Use as target, raised I3C_EVR event	When target: specific action
Broadcast CCCs							
ENEC	0x00	Write	No defining/sub-command byte	With one data byte (enable target events byte)	X	X, INTUPDF	Update and enable I3C_DEVR0: HJEN, CREN, IBIEN if any
DISEC	0x01			With one data byte (disable target events byte)	X	X, INTUPDF	
ENTAS _x x = 0...3	0x02			No data byte	X	X, ASUPDF	
RSTDAA	0x06			-	X	X, DAUPDF	
ENTDAA	0x07			-	X	X, DAUPDF	
SETMWL	0x09			With two data byte	X	X, MWLUPDF	
SETMRL	0x0A			With 2 or 3 data bytes	X	X, MRLUPDF	Update I3C_MAXRLR
SETAASA	0x29		No defining/sub-command byte		X		
RSTACT	0x2A		With defining byte (0x00, 0x01 or 0x02)	No data byte	X	X, RSTF after detected reset pattern	Update I3C_DEVR0: RSTACT[1:0] and set RSTVAL = 1

Table 13 List of supported I3C CCCs, as controller/target (continued)

CCC name	CCC value	Read /write	With/without defining byte With/without sub-command byte	With/without optional data byte(s)	Use as controller	Use as target, raised I3C_EVR event	When target: specific action	
Direct CCCs							Action if ACK (if I3C target Address = I3C_DEVR0.DA[6:0] and I3C_DEVR0.DAVAL = 1) (else NACK)	
ENEC	0x80	Write	No defining/sub-command byte	With one data byte (enable target events byte)	X	X, INTUPDF	Update and enable I3C_DEVR0: HJEN, CREN, IBIEN if any	
DISEC	0x81			With one data byte (disable target events byte)	X	X, INTUPDF	Update and disable I3C_DEVR0: HJEN, CREN, IBIEN if any	
ENTASx x = 0...3	0x82			No data byte	X	X, ASUPDF	Update I3C_DEVR0.AS[1:0]	
SETDASA	0x87			No data byte	X	X, DAUPDF	-	
SETNEWDA	0x88			With one data byte	X	X, DAUPDF	Update I3C_DEVR0: DA[6:0] (and set DAVAL = 1)	
SETMWL	0x89			With two data bytes	X	X, MWLUPDF	Update I3C_MAXWLR	
SETMRL	0x8A			With two or three data bytes	X	X, MRLUPDF	Update I3C_MAXRLR	
GETMWL	0x8B			Read	No defining/sub-command byte	With two data bytes	X	X, GETF
GETMRL	0x8C	With two or three data bytes	X			X, GETF	Return data bytes from I3C_MAXRLR[15:0] and if I3C_BCR.BCR2 = 1 return third byte from I3C_MAXRLR.IBIP[2:0]. Refer to Section49.16.18	
GETPID	0x8D	With six data bytes	X			X, GETF	Return data bytes from I3C_EPIDR. Refer to Section49.16.28	
GETBCR	0x8E	With one data byte	X			X, GETF	Return data byte from I3C_BCR[7:0]. Refer to Section49.16.23.	
GETDCR	0x8F		X			X, GETF	Return I3C_DCR[7:0]. Refer to Section49.16.24.	
GETSTATUS	0x90	With or without defining byte (TGTSTAT, PRECR)	With two data bytes (format 1 or format 2 with PRECR)			X	X, STAF if format 1 X, GETF if format 2	Return 2 data bytes, as detailed in Section49.9.9.
GETMXDS	0x94	With or without defining byte (WRRDTURN, CRHDLY)	With two data bytes (format 1) or 5 data bytes (format 2 or format 3 with WRRDTURN) or 1 data byte (format 3 with CRHDLY)			X	X, GETF	Return data byte(s) from I3C_GETMXDSR. Refer to Section49.16.27.
D2DXFER	0x97	Write	With defining byte			With defining byte	X	-
SETXTIME	0x98		With sub-command byte	With sub command byte	X	-	-	
GETXTIME	0x99	Read	No defining/sub-command byte	No defining/sub-command byte	X	-	-	
RSTACT	0x9A	Read/ Write	With defining byte (0x00, 0x01, or 0x02)	With defining byte (0x00, 0x01, or 0x02)	X	X, RSTF if detected reset pattern	Read: return data byte from RSTACT[1:0] in the I3C_DEVR0 register. Write: update I3C_DEVR0: RSTACT[1:0] and set RSTVAL = 1	

3.5 I²C Legacy Communication

I3C is designed to be backward compatible with I²C, allowing seamless communication with legacy I²C devices while benefiting from enhanced performance and features. In mixed bus environments, an I3C controller can operate in I²C-compatible mode, where it communicates with legacy I²C targets using standard I²C clock stretching, Start and Stop conditions and acknowledge (ACK/NACK) mechanisms. Additionally, I3C devices recognize and respond to traditional I²C transactions, including 7-bit and 10-bit addressing, ensuring interoperability. While I²C devices operate at their standard speeds (100 kHz, 400 kHz, or 1 MHz), I3C targets can utilize dynamic clocking and in-band interrupt (IBI) capabilities when interacting with other I3C devices, optimizing bus efficiency.

This dual compatibility allows for a smooth transition from I²C to I3C while maintaining support for existing I²C peripherals.

The figure below presents both a legacy I2C and typical read register-based device transfer (write register address followed by data reads), and a legacy I2C typical write register-based device transfer (write register address followed by data writes).

3.5.1 I²C Legacy Frame Format

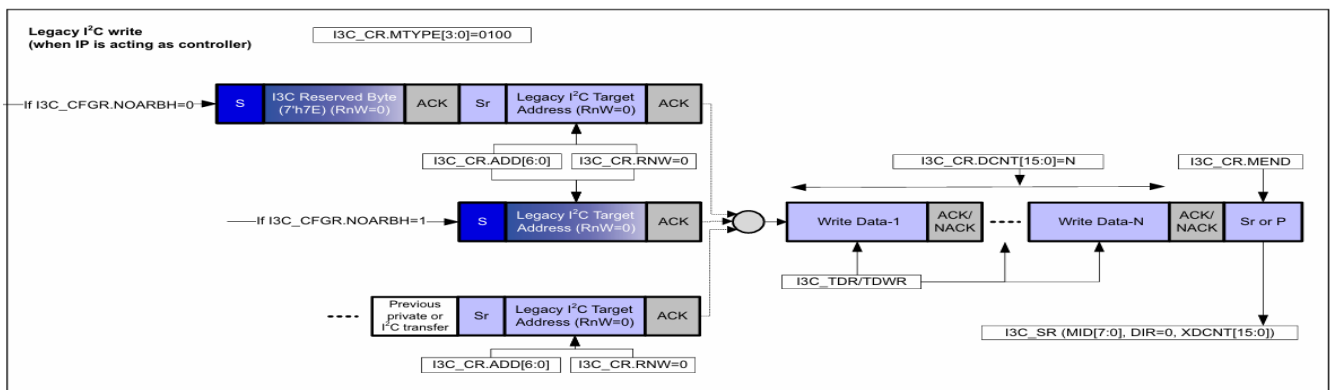


Figure 52 Legacy I2C write messages - as controller

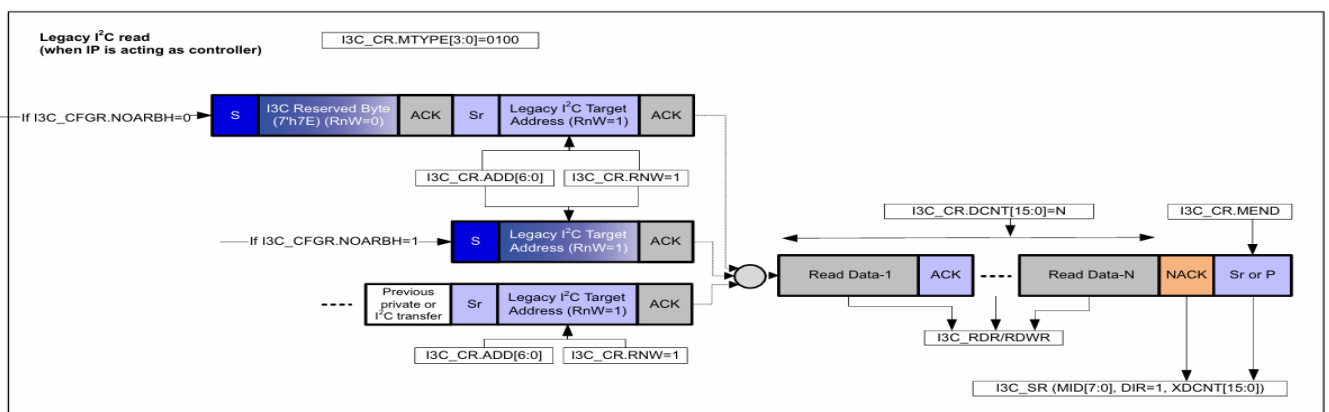
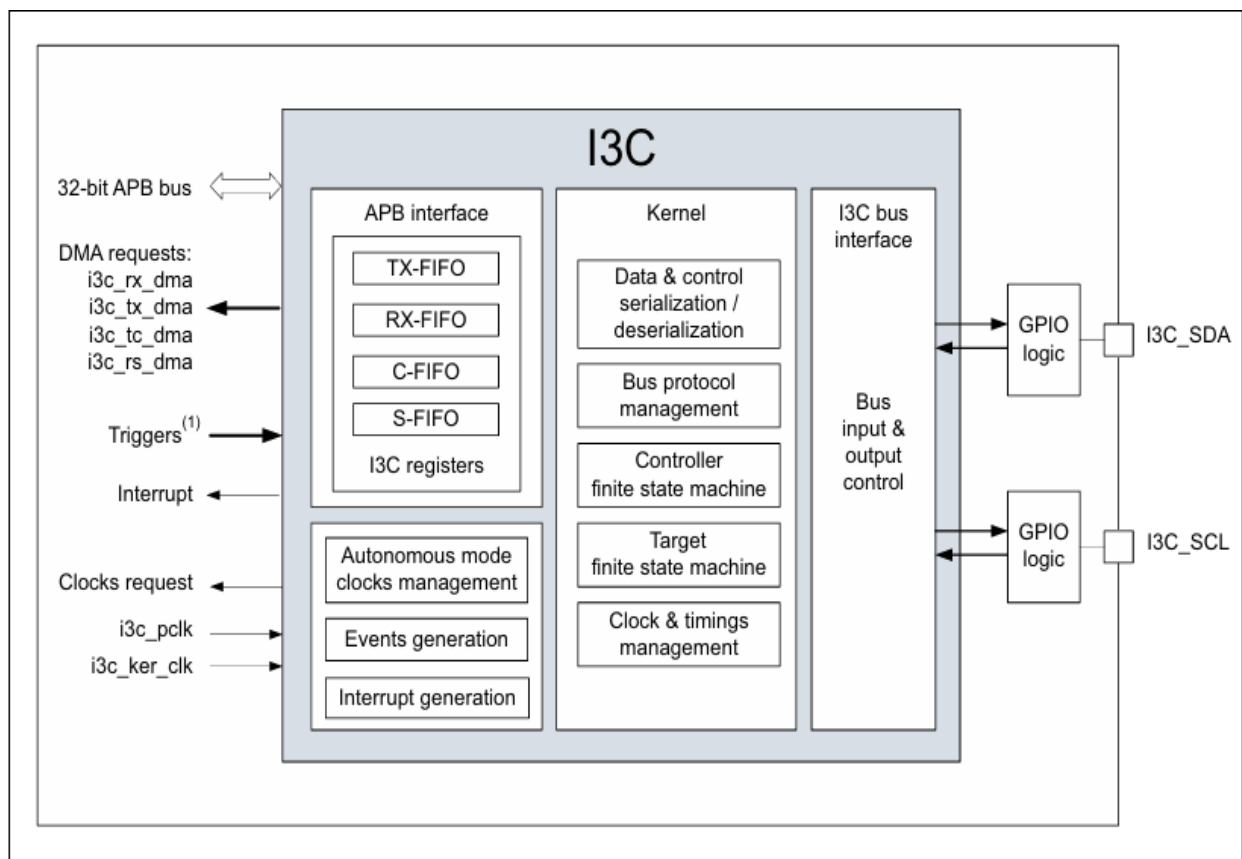


Figure 53 Legacy I2C read messages - as controller

Chapter 4: STM I3C System Architecture

This chapter details the hardware architecture of the custom-designed MIPI I3C Controller. The design was approached with a modular philosophy to ensure clear separation of concerns, facilitate verification, and produce a synthesizable IP core suitable for integration into a larger System-on-Chip (SoC). The primary interface for system-level integration is a 32-bit AMBA APB slave port, while the protocol-level logic is managed by a dedicated kernel.

Figure 54 I3C Block Diagram



The overall architecture, illustrated in Figure 4.1, is partitioned into three primary functional units: the **APB Interface**, the **I3C Kernel**, and the **I3C Bus Interface**. The design operates on two main clock domains: **i3c_pclk** for the APB bus and system register access, and **i3c_ker_clk** for the core protocol logic, which are decoupled using asynchronous fifos. The following sections will provide a detailed description of each component.

4.1 APB Interface Block

The APB Interface block serves as the bridge between the I3C controller and the host processor or system bus. It is responsible for all configuration, control, and data-transfer operations initiated by the system. It adheres to the AMBA APB protocol specification, ensuring standard-compliant integration. This block contains the following sub-modules:

- **I3C Registers:** This is the primary control and status plane for the controller. A dedicated address space is mapped to a set of registers accessible via APB read/write transactions. You should include a detailed Register Map table here. These registers include:
 - **Control Registers:** To start/stop I3C transactions, configure the controller mode (e.g., Controller, Target), and issue specific commands.
 - **Configuration Registers:** To set the I3C bus speed by programming clock dividers, define the controller's own address, etc.
 - **Status Registers:** To provide real-time feedback to the processor, indicating FIFO levels (empty, full, thresholds), transfer completion, error conditions, and interrupt sources.

4.2 Registers

The I3C registers must be accessed with a 32-bit word-aligned address.

4.2.1 I3C message control register (I3C_CR)

Address offset: 0x000

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MEND	MTYPE[3:0]					Res.	Res.	Res.	ADD[6:0]						RNW
w	w	w	w	w				w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DCNT[15:0]															
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

This register must be used to control the message to emit on the I3C bus:

- when I3C acts as controller (bit[30] = MTYPE[3] = 0): if there is no CCC code to be emitted bits[29:27] = MTYPE[2:0] differ from 110; else the alternate register description Section 49.16.2 must be considered.

- when I3C acts as target (bit[30] = MTYPE[3] = 1).

When I3C acts as controller:

- If the control FIFO (C-FIFO) is not full (CFNFF = 1 in the I3C_EVR register), writing into

this register means pushing a new control word into the C-FIFO; either by software, or automatically by DMA, as defined by CDMAEN in the I3C_CFGR register.

- If C-FIFO is empty and a restart must be emitted with a new control word, the I3C hardware asserts the control FIFO error underrun flag (COVR = 1 in the I3C_SER

register). If enabled by $ERRIE = 1$ in the $I3C_IER$ register, an interrupt is generated.

- After the last message of the frame is completed (a message with $MEND = 1$ in the $I3C_CR$ register), the I3C hardware asserts the frame completed flag ($FCF = 1$ in the $I3C_EVR$ register) and the corresponding interrupt, if enabled.

When I3C acts as target, this register is used in register mode:

- Software writes into this register to initiate a command (IBI, controller-role or hot-join request) on the I3C bus.
 - C-FIFO is disabled, and there is no DMA mode neither for control words.

Bit 31 MEND: Message end type/last message of a frame (when the I3C acts as controller)

0: this message from controller is followed by a repeated start (Sr), before another message

must be emitted

1: this message from controller ends with a stop (P), being the last message of a frame

Bits 30:27 MTYPE[3:0]: Message type (whatever I3C acts as controller/target)

Condition: when I3C acts as I3C controller

0000: SCL clock is forced to stop until a next control word is executed

Bits[26:0] are ignored. On a CE1 error detection ($ERRF = 1$ in the $I3C_EVR$ register and $CODERR[3:0] =$

0001 in the $I3C_SER$ register) where a start/restart/stop is prevented from being generated, the software

must use this message type for SCL “stuck at” recovery. Refer to Table 540.

0001: header message

Bits[26:0] are ignored. If the addressed target is not responding with an ACK to a private/direct message, as

an escalation stage after a failed GETSTATUS tentative, the software must program this with $EXITPTRN = 1$

in the $I3C_CFGR$ register, so that an HDR exit pattern is emitted on the bus, whatever the header is ACK-ed

or NACK-ed (to avoid the target to consider that the I3C bus is in HDR mode). Refer to Table 540 and MIPI

specification about escalation handling.

0010: private message (refer to Figure 667)

Bits[23:17] ($ADD[6:0]$) are the emitted 7-bit dynamic address.

Bit[16] (RNW) is the emitted RnW bit.

Bits[15:0] ($DCNT[15:0]$) are the number of programmed data bytes.

The transferred private message is:—

$\{S / S + 0b111_1110 + RnW = 0 + Sr/Sr+*\} + 7\text{-bit DynAddr} + RnW + (8\text{-bit Data} + T)* + Sr/P$.—

After an S (start), depending upon bit NOARBH in the $I3C_CFGR$ register, the arbitrable header

($0b111_1110 + RnW = 0$) is inserted or not.

Sr+*: after an Sr (repeated start), the hardware automatically inserts ($0b111_1110 + RnW = 0$) if

needed, if it follows a previous message without ending by a P (stop).

0011: direct message (second part of an I3C SDR direct CCC command) (refer to Figure 660)

Bits[23:17] (ADD[6:0]) are the emitted 7-bit dynamic address.

Bit[16] (RNW) is the emitted RnW bit.

Bits[15:0] (DCNT[15:0]) are the number of programmed data bytes.

The transferred direct message is: $Sr + 7\text{-bit DynAddr} + RnW + (8\text{-bit Data} + T)^* + Sr/P$

0100: legacy I2C message (refer to Figure 669)

Bits[23:17] (ADD[6:0]) are the emitted 7-bit static address.

Bit[16] (RNW) is the emitted RnW bit.

Bits[15:0] (DCNT[15:0]) are the number of programmed data bytes.

The transferred legacy I2C message is: –

$\{S / S + 0b111_1110 + RnW = 0 + Sr/Sr+*\} + 7\text{-bit StaAddr} + RnW + (8\text{-bit data} + T)^* + Sr/P$.—

After an S, depending on NOARBH, the arbitrable header ($0b111_1110 + RnW = 0$) is inserted or not.

Sr+*: after an Sr (repeated start), the hardware automatically inserts ($0b111_1110 + RnW = 0$) if

needed (if it follows a previous message without ending by a P (stop)).

Others: reserved

Condition: when I3C acts as I3C target

1000: hot-join request (W) (refer to Figure 671)

The transferred hot-join request is $\{S + \} 0b000_0010 \text{ addr} + RnW = 0$.

Writing the control word initiates the hot-join request if target is allowed to do so (HJEN = 1 in the

I3C_DEVR0 register), either actively after a bus idle condition via the hardware issuing a start request (SDA

low) and waiting for the controller to activate SCL clock, or passively if the controller initiates a concurrent

message.

1001: controller-role request (W) (refer to Figure 672)

The transferred controller-role request is $\{S + \} DA[6:0] + RnW = 0$ (DA in the I3C_DEVR0 register)

Writing the control word initiates the controller-role request if target is allowed to do so (CREN = 1 and

DAVAL = 1 in the I3C_DEVR0 register), either actively after a bus idle condition via the hardware issuing a

start request (SDA low) and waiting for the controller to activate SCL clock, or passively if the controller

initiates a concurrent message.

1010: IBI (in-band interrupt) request (R) (refer to Figure 670)

Bits[15:0] (DCNT[15:0]) are the number of the IBI data payload (including the first MDB), if any.

The transferred IBI request is $\{S + \} DA[6:0] + RnW = 1 + \text{optional IBI data payload}$.

Writing the control word initiates the IBI request if target is allowed to do so (IBIEN = 1 and DAVAL = 1 in the I3C_DEVR0.register), either actively after a bus idle condition via the hardware issuing a start request (SDA low) and waiting for the controller to activate SCL clock, or passively if the controller initiates a concurrent message.

When acknowledged from controller, the transmitted IBI payload data (optional, depending upon BCR2 in the I3C_BCR register) is defined by DCNT[15:0] in the I3C_CR register and I3C_IBIDR, and must be consistently programmed vs. the IBI payload data size defined by IBIP[2:0] in the I3C_IBIDR register.

Others: reserved

Bits 26:24 Reserved, must be kept at reset value.

Bits 23:17 ADD[6:0]: 7-bit I3C dynamic / I2C static target address (when I3C acts as controller)

When I3C acts as controller, this field is used if MTYPE[3:0] = 0010 (private message), or MTYPE[3:0] = 0011 (direct message), or MTYPE[3:0] = 0100 (legacy I2C message)

Bit 16 RNW: Read / non-write message (when I3C acts as controller)

When I3C acts as controller, this field is used if MTYPE[3:0] = 0010 (private message), or MTYPE[3:0] = 0011 (direct message), or MTYPE[3:0] = 0100 (legacy I2C message), to emit the RnW bit on the I3C bus.

0: write message

1: read message

Bits 15:0 DCNT[15:0]: Count of data to transfer during a read or write message, in bytes (whatever I3C acts as controller/target)

When I3C acts as controller, this field is used if MTYPE[3:0] = 0010 (private message), or MTYPE[3:0] = 0011 (direct message), or MTYPE[3:0] = 0100 (legacy I2C message), to set the number of exchanged data bytes on the bus. In case of a private or legacy I2C read/write message, this field must be non-null.

When I3C acts as target, this field is used if MTYPE[3:0] = 1010 (IBI request) and if any IBI

data payload (data to be transmitted if BCR2 = 1 in the I3C_BCR register), to set the number

of bytes of the IBI data payload (1, 2, 3, or 4).

Linear encoding up to 64 Kbytes - 1

0x0000: no data to transfer

0x0001: 1 byte

0x0002: 2 bytes

...

0xFFFF: 64 Kbytes - 1 byte

4.2.2 I3C message control register [alternate] (I3C_CR)

Address offset: 0x000

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MEND	MTYPE[3:0]					Res.	Res.	Res.	CCC[7:0]						
w	w	w	w	w				w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DCNT[15:0]															
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

This write register description must be used to control the message for when the controller has to emit a CCC (whatever is the type of the CCC: for a CCC broadcast, a CCC direct, or a CCC Enter HDR).

This is the alternate description of register I3C_CR, for when MTYPE[3:0] = 0110. Else refer to Section 49.16.1.

If the control FIFO (also known as C-FIFO) is not full (CFNFF = 1 in the I3C_EVR register), writing into this register means pushing a new control word into the C-FIFO; either by the software or automatically by DMA, as defined by the CDMAEN bit in the I3C_CFGR register.

When the last message of the frame is completed (a message with MEND = 1 in the I3C_CR register), the I3C hardware asserts the frame completed flag (FCF = 1 in the I3C_EVR register) and the corresponding interrupt, if enabled.

Bit 31 MEND: Message end type/last message of a frame (when I3C acts as controller)

0: this message from controller is followed by a repeated start (Sr), before another message must be emitted

1: the message from the controller ends with a stop (P), being the last message of a frame

Bits 30:27 MTYPE[3:0]: Message type (when I3C acts as controller)

Condition: when I3C acts as I3C controller

0110: broadcast/direct CCC command (refer to Table 539, Figure 660, Figure 661, Figure 662)

Bits[23:16] (CCC[7:0]) are the emitted 8-bit CCC code

Bits[15:0] (DCNT[15:0]) are the number of the CCC defining bytes, or CCC sub-command bytes, or CCC data bytes.

If Bit[23] = CCC[7] = 1: this is the first part of an I3C SDR direct CCC command

The transferred direct CCC command (first part) message is: –

{S / S + 0b111_1110 + RnW = 0 / Sr+*} + (direct CCC + T) + (8-bit Data + T)* + Sr—

After an S (start), depending upon NOARBH in the I3C_CFGR register, the arbitrable header (0b111_1110 + RnW = 0) is inserted or not.

Sr+*: after an Sr (repeated start), the hardware automatically inserts (0b111_1110 + R/W).

If Bit[23] = CCC[7] = 0: this is an I3C SDR broadcast CCC command (including specific ENTDAAs, refer to Figure 661)

The transferred broadcast CCC command message is: –

{S / S + 0b111_1110 + RnW = 0 / Sr+*} + (broadcast CCC + T) + (8-bit Data + T) * + Sr/P—

After an S (start), depending on NOARBH, the arbitable header (0b111_1110 + RnW = 0) is inserted or not.

Sr+*: after an Sr (repeated start), the hardware automatically inserts (0b111_1110 + R/W).

Others: reserved

Bits 26:24 Reserved, must be kept at reset value.

Bits 23:16 CCC[7:0]: 8-bit CCC code (when I3C acts as controller)

If bit[23] = CCC[7] = 1, this is the first part of an I3C SDR direct CCC command.

If bit[23] = CCC[7] = 0, this is an I3C SDR broadcast CCC command (including ENTDAAs).

Bits 15:0 DCNT[15:0]: Count of related data to the CCC command to transfer as CCC defining bytes, or

CCC sub-command bytes, or CCC data bytes, in bytes

Linear encoding up to 64 Kbytes - 1.

0x0000: no data to transfer.

Note: Value mandatory when emitting ENTDAAs broadcast CCC (refer to Figure 661).

0x0001: 1 byte

Note: Value mandatory when emitting RSTACT direct/broadcast CCC (refer to Figure 662).

0x0002: 2 bytes

...

0xFFFF: 64 Kbytes - 1 byte

4.2.3 I3C configuration register (I3C_CFGR)

Address offset: 0x004

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	TSFSET	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CFLUSH	CDMA EN	TMODE	SMODE	SFLUSH	SDMA EN
	w									w	rw	rw	rw	w	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TX THRES	TX FLUSH	TX DMAEN	Res.	RX THRES	RX FLUSH	RX DMAEN	HJACK	Res.	HK SDAEN	EXIT PTRN	RST PTRN	NO ARBH	CRINIT	EN
	rw	w	rw		rw	w	rw	rw		rw	rw	rw	rw	rw	rw

This register is used to configure:

- features that apply when the I3C acts as controller or target: RX-FIFO and TX-FIFO management (RXDMAEN, RXTHRES, RXFLUSH, TXDMAEN, TXTHRES, TXFLUSH), I3C peripheral role (CRINIT)
- dedicated features when the I3C acts as a controller: frame-based control-word triggering (TSFSET), FIFOs management (TMODE, SMODE, SFLUSH, SDMAEN, CDMAEN), and miscellaneous ones (HJACK, HKSDAEN, EXITPTRN, RSTPATRN, NOARBH)

The configuration fields CRINIT, HKSDAEN can be modified only when EN = 0. This condition is respected if they are modified at the same time when EN is set to 1 (it is not necessary to set EN later on, with another write operation).

Bit 31 Reserved, must be kept at reset value.

Bit 30 TSFSET : Frame transfer set (software trigger) (when I3C acts as controller)

This bit can only be written. When I3C acts as I3C controller:

0: no action

1: setting this bit initiates a frame transfer by causing the hardware to assert the flag CFNFF in the I3C_EVR register (C-FIFO not full and a control word is needed)

Note: If this bit is not set, the other alternative for the software to initiate a frame transfer is to directly write the first control word register (I3C_CR) while C-FIFO is empty (CFEF = 1 in the I3C_EVR register). Then, if the first written control word is not tagged as a message end (MEND = 0 in the I3C_CR register), it causes the hardware to assert CFNFF.

Bits 29:24 Reserved, must be kept at reset value.

Bit 23 Reserved, must be kept at reset value.

Bit 22 Reserved, must be kept at reset value.

Bit 21 CFLUSH: C-FIFO flush (when I3C acts as controller)

This bit can only be written.

0: no action

1: flush C-FIFO

Bit 20 CDMAEN: C-FIFO DMA request enable (when I3C acts as controller)

When I3C acts as controller:

0: DMA mode is disabled for C-FIFO- Software writes and pushes control word(s) into C-FIFO (writes I3C_CR register), as needed for a given frame - A next control word transfer can be written by software either via polling on the flag CFNFF = 1 in the I3C_EVR register, or via interrupt notification (enabled by CFNFIE = 1 in the I3C_IER register).

1: DMA mode is enabled for C-FIFO- DMA writes and pushes control word(s) into C-FIFO (writes I3C_CR register), as needed for a given frame.- A next control word transfer is automatically written by the programmed hardware (via the asserted C-FIFO DMA request from the I3C and the programmed DMA channel).

Bit 19 TMODE: Transmit mode (when I3C acts as controller)

When I3C acts as controller, this bit is used for the C-FIFO and TX-FIFO management vs. the

emitted frame on the I3C bus.

0: C-FIFO and TX-FIFO are not preloaded before starting to emit a frame transfer.

A frame transfer starts as soon as the first control word is present in C-FIFO.

1: C-FIFO and TX-FIFO are first preloaded (also TX-FIFO if needed, depending on the frame format) before starting to emit a frame transfer. Refer to Section 49.10.2 for more details.

Bit 18 SMODE: S-FIFO enable / status receive mode (when I3C acts as controller)

When I3C acts as controller, this bit is used to enable the FIFO for the status (S-FIFO) of the exchanged message on the I3C bus.

When I3C acts as target, this bit must be cleared.

0: S-FIFO is disabled - Status register (I3C_SR) is used without FIFO mechanism.- There is no SCL stalling if a new status register content is not read.- Status register must be read before being overwritten by the hardware.- Must have SDMAEN = 0 in the I3C_CFGR register.

1: S-FIFO is enabled. - Each message status must be read.- There is SCL stalling when the S-FIFO is full and a next message status must be read.- S-FIFO overrun error is reported after the maximum SCL clock stalling time.

Bit 17 SFLUSH: S-FIFO flush (when I3C acts as controller)

This bit can be written and used only when I3C acts as controller.

0: no action

1: flush S-FIFO

Bit 16 SDMAEN: S-FIFO DMA request enable (when I3C acts as controller)

This bit must be cleared if SMODE = 0 in the I3C_CFGR register (S-FIFO is disabled). In other words, DMA mode cannot be used if S-FIFO is disabled. Then the status register I3C_SR can be read or not.

This bit can be set or cleared if SMODE = 1 (S-FIFO is enabled). In other words, status

register I3C_SR must be read for each message, either by software, or via an allocated DMA channel.

0: DMA mode is disabled for reading status register I3C_SR- SMODE = 0: software can read the I3C_SR register after a completed frame (FCF = 1 in the I3C_EVR register) or an error (ERRF = 1 in the I3C_EVR register). Via polling on these register flags or via interrupt notification (enabled by FCIE = 1 and ERRIE = 1 in the I3C_IER register).- SMODE = 1: software must read and pop a status word from S-FIFO (read I3C_SR register) after each asserted flag SFNEF = 1. Via polling on this register flag or via interrupt notification (enabled by SFNEIE = 1 in the I3C_IER register).

1: DMA mode is enabled for reading status register I3C_SR- Must have SMODE = 1 in the I3C_CFGR register (S-FIFO enabled)- DMA reads and pops status word(s) from S-FIFO (it reads I3C_SR register)- Status word(s) are automatically read by the programmed hardware (via the asserted S-FIFO DMA request from the I3C and the programmed DMA channel).

Bit 15 Reserved, must be kept at reset value.

Bit 14 TXTHRES: TX-FIFO threshold (whatever I3C acts as controller/target)

This threshold defines, compared to the TX-FIFO level, when the TXFNFF flag is set in the I3C_EVR register (and consequently if TXDMAEN = 1 when is asserted a

DMA TX request).

0: 1-byte threshold

TXFNFF is set when 1 byte must be written in TX-FIFO (in I3C_TDR).

1: 1-word / 4-byte threshold TXFNFF is set when 1 word / 4 bytes must be written in TX-FIFO (in the I3C_TDWR register). If the a number of the last transmitted data is not a multiple of 4 bytes (XDCNT[1:0] = 00 in the I3C_SR register), only the relevant 1, 2, or 3 valid LSB bytes of the last word are taken into account by the hardware, and sent on the I3C bus.

Bit 13 TXFLUSH: TX-FIFO flush (whatever I3C acts as controller/target)

This bit can only be written.

When the I3C acts as target, this bit can be used to flush the TX-FIFO on a private read if the controller has aborted the data read (driven low the T bit), and there is/are remaining data in the TX-FIFO (ABT = 1, and XDCNT[15:0] in the I3C_SR register < TGTDCNT[15:0] in the I3C_TGTTDR register).

0: no action

1: flush TX-FIFO

Bit 12 TXDMAEN: TX-FIFO DMA request enable (whatever I3C acts as controller/target)

0: DMA mode is disabled for TX-FIFO- Software writes and pushes a data byte/word into TX-FIFO (writes I3C_TDR or I3C_TDWR register), to be transmitted over the I3C bus. - A next data byte/word must be written by the software either via polling on the flag TXFNFF = 1 or via interrupt notification (enabled by TXFNIE = 1).

1: DMA mode is enabled for TX-FIFO- DMA writes and pushes data byte(s)/word(s) into TX-FIFO (writes I3C_TDR or I3C_TDWR register). - A next data byte/word transfer is automatically pushed by the programmed hardware (via the asserted TX-FIFO DMA request from the I3C and the programmed DMA channel).

Bit 11 Reserved, must be kept at reset value.

Bit 10 RXTHRES: RX-FIFO threshold (whatever I3C acts as controller/target)

This threshold defines, compared to the RX-FIFO level, when the RXFNEF flag in the I3C_EVR register is set (and consequently if RXDMAEN = 1 when is asserted a DMA RX request).

0: 1-byte threshold

RXFNEF is set when 1 byte must be read in RX-FIFO (in the I3C_RDR register).

1: 1-word/4-bytes threshold

RXFNEF is set when 1 word / 4 bytes is/are to be read in RX-FIFO (in I3C_RDWR).

In the case of a number of last received data being not a multiple of 4 bytes, only the relevant 1, 2 or 3 valid LSB bytes of the last word are to be considered by the software. The number of effective received data bytes is reported by XDCNT[15:0] in the I3C_SR register.

Bit 9 RXFLUSH: RX-FIFO flush (whatever I3C acts as controller/target)

This bit can only be written.

0: no action

1: flush RX-FIFO

Bit 8 RXDMAEN: RX-FIFO DMA request enable (whatever I3C acts as controller/target)

0: DMA mode is disabled for RX-FIFO- Software reads and pops a data byte/word from RX-FIFO (it reads I3C_RDR or I3C_RDWR register).- A next data byte/word must be read by the software either via polling flag RXFNEF = 1 in the I3C_EVR register, or via interrupt notification (enabled by RXFNEIE = 1 in the I3C_IER register).

1: DMA mode is enabled for RX-FIFO- DMA reads and pops data byte(s)/word(s) from RX-FIFO (reads I3C_RDR or I3C_RDWR register).- A next data byte/word is automatically read by the programmed hardware (via the asserted RX-FIFO DMA request from the I3C and the programmed DMA channel).

Bit 7 HJACK: Hot-join request acknowledge (when I3C acts as a controller)

0: hot-join request is not acknowledged

After the NACK, the controller continues as initially programmed (the hot-joining target is aware of the NACK and must emit another hot-join request later on).

1: hot-join request is acknowledged

After the ACK, the controller continues as initially programmed. The software is notified by the HJ interrupt (flag HJF is set in the I3C_EVR register), and must initiate the ENTDA sequence later on, potentially preventing other hot-join requests with a disable target events command (DISEC, with DISHJ = 1).

Bit 6 Reserved, must be kept at reset value.

Bit 5 HKSDAEN: High-keeper enable on SDA line (when I3C acts as a controller)

0: High-keeper is disabled

1: High-keeper is enabled, and the weak pull-up is effective on the T bit, instead of the open drain class pull-up.

Note: This bit can be modified only when EN = 0 in the I3C_CFGR register.

Bit 4 EXITPTRN: HDR exit pattern enable (when I3C acts as a controller)

This bit can be modified only when there is no on-going frame.

0: HDR exit pattern is not sent after the issued message header (MTYPE[3:0] = 0001 in the I3C_CR register). This is used to send the header, to test ownership of the bus when there is a suspicion of a problem after controller-role hand-off (new controller did not assert its controller-role by accessing the previous one in less than the delay defined by the activity state).

1: HDR exit pattern is sent after the issued message header (MTYPE[3:0] = 0001).

This is used on a controller error detection and escalation handling, in case of a not responding target to a private message or a direct read CCC.

The HDR exit pattern is sent whatever the message header {S/Sr + 0x7E addr + W} is ACKed or NACK-ed..

Bit 3 RSTPTRN: HDR reset pattern enable (when I3C acts as a controller)

This bit can be modified only when there is no on-going frame.

0: standard stop emitted at the end of a frame

1: HDR reset pattern is inserted before the stop of any emitted frame that includes a RSTACT CCC command

Bit 2 NOARBH: No arbitrable header after a start (when I3C acts as a controller)

This bit can be modified only when there is no on-going frame.

0: An arbitrable header (0b111_1110 + RnW = 0) is emitted after a start and before a legacy I2C message or an I3C SDR private read/write message (default).

1: No arbitrable header- The target address is emitted directly after a start in case of a legacy I2C message or an I3C SDR private read/write message. - This is a more performing option (when the emission of the 0x7E arbitrable header is useless), but must be used only when the controller is sure that the addressed target device cannot emit concurrently an IBI or a controller-role request (to insure no misinterpretation and no potential conflict between the address emitted by the controller in open-drain mode and the same address a target device can emit after a start, for IBI or MR).

Bit 1 CRINIT: Initial controller/target role

This bit can be modified only when EN = 0 in the I3C_CFGR register.

0: target role Once enabled by setting EN = 1, the peripheral initially acts as a target. I3C does not drive SCL line and does not enable SDA pull-up, until it eventually acquires the controller role.

1: controller role Once enabled by setting EN = 1, the peripheral initially acts as a controller. It has the I3C controller role, so drives SCL line and enables SDA pull-up, until it eventually offers the controller role to an I3C secondary controller.

Bit 0 EN: I3C enable (whatever I3C acts as controller/target)

0: I3C is disabled

- Except registers, the peripheral is under reset (partial reset).

- Before clearing EN, when I3C acts as a controller, all the possible target requests must be disabled using DISEC CCC.

- When I3C acts as a target, software must not disable the I3C, unless a partial reset is needed.

1: I3C is enabled

In this state, some register fields cannot be modified (like CRINIT, HKSDAEN for the I3C_CFGR).

4.2.4 I3C receives data byte register (I3C_RDR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RDB0[7:0]							
								r	r	r	r	r	r	r	r

Bits 31:8 Reserved: must be kept at reset value.

Bits 7:0 RDB0[7:0]: 8-bit received data on I3C bus.

Byte-Based Read (RXTHRES = 0)

- Reads one byte at a time (stored in LSB of a 32-bit word).
- If RXDMAEN = 1, DMA manages to read automatically.
- If RXDMAEN = 0, software must check RXFNEF flag before reading.
- If FIFO is full, an overrun error (DOVRF) is triggered.

4.2.5 I3C transmit data byte register (I3C_TDR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TDB0[7:0]							
								w	w	w	w	w	w	w	w

Bits 31:8 Reserved: must be kept at reset value.

Bits 7:0 TDB0[7:0]: 8-bit data to transmit on I3C bus.

Byte-Based Writing (TXTHRES = 0) Transmits one byte at a time.

- TXDMAEN = 1: Data transfer is managed automatically via DMA.
- TXDMAEN = 0: Software must wait for TXFNFF = 1 before writing.
- If TX-FIFO is empty and the data is delayed, a data underrun error (DOVR) occurs.

4.2.6 I3C event register (I3C_EVR)

Address offset: 0x050

Reset value: 0x0000 0003

This is a read register, used for reporting event flags.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
GRPF	DEFF	INT UPDF	AS UPDF	RSTF	MRL UPDF	MWL UPDF	DA UPDF	STAF	GETF	WKPF	Res.	HJF	CR UPDF	CRF	IBI ENDF
r	r	r	r	r	r	r	r	r	r	r		r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IBIF	Res.	Res.	Res.	ERRF	RXTGT ENDF	FCF	Res.	RX LASTF	TX LASTF	RX FNEF	TX FNFF	SFNEF	CFNFF	TXFEF	CFEF
r				r	r	r		r	r	r	r	r	r	r	r

Bit 31 GRPF: Group addressing flag (when the I3C acts as target)

When the I3C acts as target (and is typically controller-capable), this flag is asserted by hardware to indicate that the broadcast DEFGRPA CCC (define list of group addresses) has been received. Then, software can store the received data for when getting controller role. The flag is cleared when software writes 1 into the corresponding CGRPF bit in the I3C_CR register.

Bit 30 DEFF: DEFTGTS flag (when the I3C acts as target)

When the I3C acts as target (and is typically controller capable), this flag is asserted by hardware to indicate that the broadcast DEFTGTS CCC (define list of targets) has been received. Then, software can store the received data for when getting the controller role.

The flag is cleared when software writes 1 into the corresponding CDEFF bit in the I3C_CEV register.

Bit 29 INTUPDF: Interrupt/controller-role/hot-join update flag (when the I3C acts as target)

When the I3C acts as target, this flag is asserted by hardware to indicate that the direct or broadcast ENEC/DISEC CCC (enable/disable target events) has been received, where a target event is either an interrupt/IBI request, a controller-role request, or an hot-join request.

Then, software must read respectively IBIEN, CREN, or HJEN in the I3C_DEVR0 register.

The flag is cleared when software writes 1 into the corresponding CINTUPDF bit in the I3C_CEV register.

Bit 28 ASUPDF: Activity state update flag (when the I3C acts as target)

When the I3C acts as target, this flag is asserted by hardware to indicate that the direct or broadcast ENTASx CCC (with x = 0...3) has been received. Then, software must read AS[1:0] in the I3C_DEVR0 register.

The flag is cleared when software writes 1 into the corresponding CASUPDF bit in the I3C_CEV register.

Bit 27 RSTF: Reset pattern flag (when the I3C acts as target)

When I3C acts as target, this flag is asserted by hardware to indicate that a reset pattern has been detected (14 SDA transitions while SCL is low, followed by repeated start, then stop).

Then, when not in Stop mode, software must read RSTACT[1:0] and RSTVAL in the I3C_DEVR0 register, to know the required reset level.

- If RSTVAL = 1: when the RSTF is asserted (and/or the corresponding interrupt if enabled), RSTACT[1:0] in the I3C_DEVR0 register dictates the reset action to be performed by the software, if any.
- If RSTVAL = 0: when the RSTF is asserted (and/or the corresponding interrupt if enabled), the software must issue an I3C reset after a first detected reset pattern, and a system reset on the second one.

When in Stop mode, the corresponding interrupt can be used to wake up the device.

The flag is cleared when software writes 1 into the corresponding CRSTF bit in the I3C_CEV register.

Bit 26 MRLUPDF: Maximum read length update flag (when the I3C acts as target)

When the I3C acts as target, this flag is asserted by hardware to indicate that a direct SETMRL CCC (set max read length) has been received. Then, software must read MRL[15:0] in the I3C_MAXRLR register to get the maximum read length value.

The flag is cleared when software writes 1 into the corresponding CMRLUPDF bit in the I3C_CEV register.

Bit 25 MWLUPDF: Maximum write length update flag (when the I3C acts as target)

When the I3C acts as target, this flag is asserted by hardware to indicate that a direct SETMWL CCC (set max write length) has been received. Then, software must read MWL[15:0] in the I3C_MAXRLR register to get the maximum write length value.

The flag is cleared when software writes 1 into the corresponding CMWLUPDF bit in the I3C_CEVr register.

Bit 24 DAUPDF: Dynamic address update flag (when the I3C acts as target)

When the I3C acts as target, this flag is asserted by hardware to indicate that a dynamic address update has been received via any of the broadcast ENTDA, RSTDA and direct SETNEWDAA CCC. Then, software must read DA[6:0] and DAVAL in the I3C_DEVR0 register to get the dynamic address update.

The flag is cleared when software writes 1 into the corresponding CDAUPDF bit in the I3C_CEVr register.

Bit 23 STAF: Get status flag (when the I3C acts as target)

When the I3C acts as target, this flag is asserted by hardware to indicate that a direct GETSTATUS CCC of format 1 (without defining byte or with defining byte TGTSTAT) has been received.

The flag is cleared when software writes 1 into the corresponding CSTAF bit in the I3C_CEVr register.

Bit 22 GETF: Get flag (when the I3C acts as target)

When the I3C acts as target, this flag is asserted by hardware to indicate that any direct CCC of get type (GET*** CCC) except the GETSTATUS of format 1 (but including GETSTATUS of format 2) has been received.

The flag is cleared when software writes 1 into the corresponding CGETF bit in the I3C_CEVr register.

Bit 21 WKPF: Wake-up/missed start flag (when the I3C acts as target)

When the I3C acts as target, this flag is asserted by hardware to indicate that a start has been detected (an SDA falling edge followed by an SCL falling edge) but on the next SCL falling edge, the I3C kernel clock is (still) gated. Thus an I3C bus transaction may have been lost by the target.

The corresponding interrupt can be used to wake up the device from a low power (Sleep or Stop) mode.

The flag is cleared when software writes 1 into the corresponding CWKPF bit in the I3C_CEVr register.

Bit 20 Reserved, must be kept at reset value.

Bit 19 HJF: Hot-join flag (when the I3C acts as controller)

When the I3C acts as controller, this flag is asserted by hardware to indicate that an hot join request has been received.

The flag is cleared when software writes 1 into the corresponding CHJF bit in the I3C_CEVr register.

Bit 18 CRUPDF: Controller-role update flag (when the I3C acts as target)

When the I3C acts as target, this flag is asserted by hardware to indicate that it has now gained the controller role after the completed controller-role hand-off procedure.

The flag is cleared when software writes 1 into the corresponding CCRUPDF bit in the I3C_CEVr register.

Bit 17 CRF: Controller-role request flag (when the I3C acts as controller)

When the I3C acts as controller, this flag is asserted by hardware to indicate that a controller role request has been acknowledged and completed (by hardware). The software must then

issue a GETACCCR CCC (get accept controller role) for the controller-role hand-off procedure.

The flag is cleared when software writes 1 into the corresponding CCRF bit in the I3C_CEV register.

Bit 16 IBIENDF: IBI end flag (when the I3C acts as target)

When the I3C acts as target, this flag is asserted by hardware to indicate that an IBI transfer has been received and completed (IBI acknowledged and IBI data bytes read by controller if any).

The flag is cleared when software writes 1 into the corresponding CIBIENDF bit in the I3C_CEV register.

Bit 15 IBIF: IBI flag (when the I3C acts as controller)

When the I3C acts as controller, this flag is asserted by hardware to indicate that an IBI request has been received.

The flag is cleared when software writes 1 into the corresponding CIBIF bit in the I3C_CEV register.

Bits 14:12 Reserved, must be kept at reset value.

Bit 11 ERRF: Flag (whatever the I3C acts as controller/target)

This flag is asserted by hardware to indicate that an error occurred. Then, software must read I3C_SER to get the error type.

The flag is cleared when software writes 1 into the corresponding CERRF bit in the I3C_CEV register.

Bit 10 RXTGTENDF: Target-initiated read end flag (when the I3C acts as controller)

When the I3C acts as controller, and only if the S-FIFO is disabled (SMODE = 0 in the I3C_CFGR register), this flag is asserted by hardware to indicate that the target has prematurely ended a read transfer. Then, software must read the status register I3C_SR to check information related to the last message and get the number of received data bytes on the prematurely read transfer (XDCNT in the I3C_SR register).

The flag is cleared when software writes 1 into the corresponding CRXTGTENDF bit in the I3C_CEV register.

Bit 9 FCF: Frame complete flag (whatever the I3C acts as controller/target)

When the I3C acts as controller, this flag is asserted by hardware to indicate that a frame has been (normally) completed on the I3C bus, for example, when a stop is issued.

When the I3C acts as target, this flag is asserted by hardware to indicate that a message addressed to/by this target has been (normally) completed on the I3C bus, for example, when a next stop or repeated start is then issued by the controller.

The flag is cleared when software writes 1 into the corresponding CFCF bit in the I3C_CEV register.

Bit 8 Reserved, must be kept at reset value.

Bit 7 RXLASTF: Last read data byte/word flag (when the I3C acts as controller)

When the I3C acts as controller, this flag is asserted by hardware to indicate that the last data byte/word (depending upon RXTHRES in the I3C_CFGR register) of a message must be read from the RX-FIFO. The flag is de-asserted by hardware when the last data byte/word of a

message is read.

Bit 6 TXLASTF: Last written data byte/word flag (whatever the I3C acts as controller/target)

This flag is asserted by hardware to indicate that the last data byte/word (depending upon TXTHRES in the I3C_CFGR register) of a message must be written to the TX-FIFO. The flag is de-asserted by hardware when the last data byte/word of a message is written.

Bit 5 RXFNEF: RX-FIFO not empty flag (whatever the I3C acts as controller/target)

This flag is asserted/de-asserted by hardware to indicate that a data byte must/must not be read from the RX-FIFO.

Note: The software must wait for RXFNEF = 1 (by polling or via the enabled interrupt) before reading from RX-FIFO (reading from I3C_RDR or I3C_RDWR, depending upon RXTHRES).

Bit 4 TXFNFF: TX-FIFO not full flag (whatever the I3C acts as controller/target)

This flag is asserted/de-asserted by hardware to indicate that a data byte/word must/must not be written to the TX-FIFO.

Note: The software must wait for TXFNFF = 1 (by polling or via the enabled interrupt) before writing to TX-FIFO (writing to I3C_TDR or I3C_TDWR, depending upon TXTHRES).

Note: When the I3C acts as target, if the software intends to use the TXFNFF flag for writing into I3C_TDR/I3C_TDWR, it must have configured and set the TX-FIFO preload (write PRELOAD in the I3C_TGTTDR register).

Bit 3 SFNEF: S-FIFO not empty flag (when the I3C acts as controller)

When the I3C acts as controller, if the S-FIFO is enabled (SMODE = 1 in the I3C_CFGR register), this flag is asserted by hardware to indicate that a status word must be read from the S-FIFO. The flag is de-asserted by hardware to indicate that a status word is not to be read from the S-FIFO.

Bit 2 CFNFF: C-FIFO not full flag (when the I3C acts as controller)

When the I3C acts as controller, this flag is asserted by hardware to indicate that a control word must be written to the C-FIFO. The flag is de-asserted by hardware to indicate that a control word is not to be written to the C-FIFO.

Note: The software must wait for CFNFF = 1 (by polling or via the enabled interrupt) before writing to C-FIFO (writing to I3C_CR).

Bit 1 TXFEF: TX-FIFO empty flag (whatever the I3C acts as controller/target)

This flag is asserted by hardware to indicate that the TX-FIFO is empty.

This flag is de-asserted by hardware to indicate that the TX-FIFO is not empty.

Bit 0 CFEF: C-FIFO empty flag (whatever the I3C acts as controller)

This flag is asserted by hardware to indicate that the C-FIFO is empty when controller, and that the I3C_CR register contains no control word (none IBI/CR/HJ request) when target.

This flag is de-asserted by hardware to indicate that the C-FIFO is not empty when controller, and that the I3C_CR register contains one control word (a pending IBI/CR/HJ request) when target.

Note: When the I3C acts as controller, if the C-FIFO and TX-FIFO preload is configured (TMODE = 1 in the I3C_CFGR register), the software must wait for TXFEF = 1 and CFEF = 1 before starting a new frame transfer.

4.2.7 I3C own device characteristics register (I3C_DEVR0)

Address offset: 0x060

Reset value: 0x00000000

When the I3C peripheral acts as target, this register is used to write or read its own device characteristics.

When the I3C peripheral acts as controller, the field DA[6:0] is used to write and store its own dynamic address

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	RSTVAL	RSTACT[1:0]		AS[1:0]		HJEN	Res.	CREN	IBIEN
							r	r	r	r	r	rw		rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DA[6:0]							DAVAL
								rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:25 Reserved, must be kept at reset value.

Bit 24 RSTVAL: Reset action is valid (when the I3C acts as target)

This bit is asserted by hardware to indicate that the RSTACT[1:0] field has been updated on the reception of a broadcast or direct write RSTACT CCC (target reset action) and is valid.

This bit is cleared by hardware when the target receives a frame start.

When the device is not in Stop mode:

- If RSTVAL = 1: when RSTF in the I3C_EVR register is asserted (and/or the corresponding interrupt if enabled), RSTACT[1:0] in the I3C_DEVR0 register dictates the reset action to be performed by the software, if any.
- If RSTVAL = 0: when RSTF is asserted (and/or the corresponding interrupt if enabled), the software must issue an I3C reset after a first detected reset pattern, and a system reset on the second one.

When in Stop mode, the corresponding interrupt can be used to wake up the device.

Bits 23:22 RSTACT[1:0]: Reset action/level on received reset pattern

(when the I3C acts as target)

This read field is used by hardware on the reception of a direct read RSTACT CCC in order to return the corresponding data byte on the I3C bus.

This read field is updated by hardware on the reception of a broadcast or direct write RSTACT CCC (target reset action).

Only the defining bytes 0x00, 0x01 and 0x02 are mapped, and RSTACT[1:0] = Defining Byte[1:0].

00: no reset action

01: first level of reset: the application software must either:

a) partially reset the peripheral, by a write and clear of the enable bit of the I3C configuration register (write EN = 0). This resets the I3C bus interface and the I3C kernel sub-parts, without modifying the content of the I3C APB registers (except the EN bit).

b) fully reset the peripheral, including all its registers, via a write and set of the I3C reset

control bit of the RCC (reset and clock controller) register.

10: second level of reset: the application software must issue a warm reset, also known as a system reset.

This (see Section 11: Reset and clock control (RCC)) has the same impact as a pin reset (NRST = 0):

- the software writes and sets the SYSRESETREQ control bit of the AITR register, when the device is controlled by a Cortex®-M.
- the software writes and sets SYSRST = 1 in the RCC_GRSTCSETR register, when the device is controlled by a Cortex®-A.

11: no reset action

Bits 21:20AS[1:0]: Activity state (when the I3C acts as target)

This read field is updated by hardware on the reception of a ENTASx CCC (enter activity state, with x = 0-3):

00: activity state 0

01: activity state 1

10: activity state 2

11: activity state 3

Bit 19HJEN: Hot-join request enable (when the I3C acts as target)

This bit is initially written by software when EN = 0, and is updated by hardware on the reception of DISEC CCC with DISHJ= 1 (cleared) and the reception of ENEC CCC with ENHJ= 1 (set). This bit can only be written by software when EN = 0 in the I3C_CFGR register.

0: hot-join request disabled

1: hot-join request enabled

Bit 18 Reserved, must be kept at reset value.

Bit 17CREN: Controller-role request enable (when the I3C acts as target)

This bit is initially written by software when EN = 0, and is updated by hardware on the reception of DISEC CCC with DISCR = 1 (cleared) and the reception of ENEC CCC with ENCR=1 (set). This bit can only be written by software when EN = 0 in the I3C_CFGR register.

0: controller-role request disabled

1: controller-role request enabled

Bit 16IBIEN: IBI request enable (when the I3C acts as target)

This bit is initially written by software when EN = 0, and is updated by hardware on the reception of DISEC CCC with DISINT = 1 (cleared) and the reception of ENEC CCC with ENINT = 1 (set). This bit can only be written by software when EN = 0 in the I3C_CFGR register.

0: IBI request disabled

1: IBI request enabled

Bits 15:8 Reserved, must be kept at reset value.

Bits 7:1 DA[6:0]: 7-bit dynamic address

When the I3C acts as controller, this field can be written by software, for defining its own

dynamic address.

When the I3C acts as target, this field is updated by hardware on the reception of either the broadcast ENTDAACCC or the direct SETNEWDA CCC.

Bit 0 DAVAL: Dynamic address is valid (when the I3C acts as target)

When the I3C acts as controller, this bit can be written by software, for validating its own dynamic address, for example before a controller-role hand-off.

When the I3C acts as target, this bit is asserted by hardware on the acknowledge of the broadcast ENTDAACCC or the direct SETNEWDA CCC, and this field is cleared by hardware on the acknowledge of the broadcast RSTDAA CCC.

- **TX-FIFO (Transmit FIFO):** A memory buffer used to store data that the host processor intends to transmit over the I3C bus. Its primary function is to decouple the fast, bursty writes from the processor over the APB bus from the slower, serial transmission by the I3C Kernel. This prevents the processor from having to wait for each byte to be sent.
- **RX-FIFO (Receive FIFO):** A memory buffer that stores data received from the I3C bus by the Kernel. This allows the Kernel to receive data at the I3C bus speed and store it, while the host processor can read it out in bursts at its own pace. **Crucially, both the TX and RX fifos are asynchronous (or dual-clock) fifos that perform the vital function of Clock Domain Crossing (CDC) between the i3c_pclk and i3c_ker_clk domains.**
- **C-FIFO (Command FIFO):** This advanced feature allows the host to queue a sequence of I3C commands (e.g., Write 3 bytes, then Read 8 bytes). The I3C Kernel can then execute this sequence autonomously, reducing the processor's management overhead and interrupt frequency.
- **S-FIFO (Status FIFO):** Paired with the C-FIFO, this buffer stores the completion status of each command executed from the C-FIFO. After a sequence is complete, the processor can read this FIFO to check the outcome of each individual operation.

4.3 I3C Kernel

The Kernel is the core intelligence of the I3C controller. It operates in the **i3c_ker_clk** domain and is responsible for all aspects of protocol execution on the I3C bus itself.

- **Controller & Target Finite State Machines (fsms):** These are the heart of the Kernel.
- The **Controller FSM** implements the logic for when the device is acting as the bus master. It sequences through the protocol states required to generate START conditions, transmit addresses and commands, manage data transfers, and generate STOP conditions.
- The **Target FSM** implements the logic for when the device is acting as a slave. It monitors the bus for its address, responds to commands from the controller, and manages In-Band Interrupt (IBI) requests.
(You should include a simplified state diagram for at least the Controller FSM here).
- **Data & Control Serialization/Deserialization:** This logic block acts as the interface between the parallel data world (inside the fifos and registers) and the serial world of the I3C bus. When transmitting, it takes a byte from the TX-FIFO and shifts it out bit-by-bit. When receiving, it shifts in bits from the bus, assembles them into a byte, and pushes the byte into

the RX-FIFO.

- **Bus Protocol Management:** This higher-level logic oversees the rules of the I3C protocol. It includes functionality for bus arbitration (if multiple controllers are present), the handling of Common Command Codes (cccs) for tasks like Dynamic Address Assignment, and the management of Hot-Join events.
- **Clock & Timings Management:** This critical block is responsible for generating the I3C clock (**I3C_SCL**) from the faster **i3c_ker_clk**. It uses programmable dividers (configured via the APB registers) to produce various standard I3C clock speeds. It also ensures that all protocol timing parameters, such as setup and hold times for the data line (**I3C_SDA**) relative to the clock, are strictly met.

4.4 I3C Bus Interface

This block forms the physical connection between the digital logic of the kernel and the physical pads of the integrated circuit.

- **Bus Input & Output Control:** This module manages the bi-directional nature of the **I3C_SDA** line and the output-driving of the **I3C_SCL** line. For I3C, this requires logic to control open-drain output drivers (to pull the lines low) and input buffers (to sense the state of the lines).
- **GPIO Logic:** This block indicates that the physical **I3C_SDA** and **I3C_SCL** pins are connected through General Purpose Input/Output pads. This logic is responsible for configuring these pads to operate in the correct mode for I3C (e.g., open-drain output enable, input buffer enable, slew rate control) as directed by the Bus Input & Output Control module.

Chapter 5: STM I3C Controller Design

This is the crucial Block Diagram for the I3C protocol

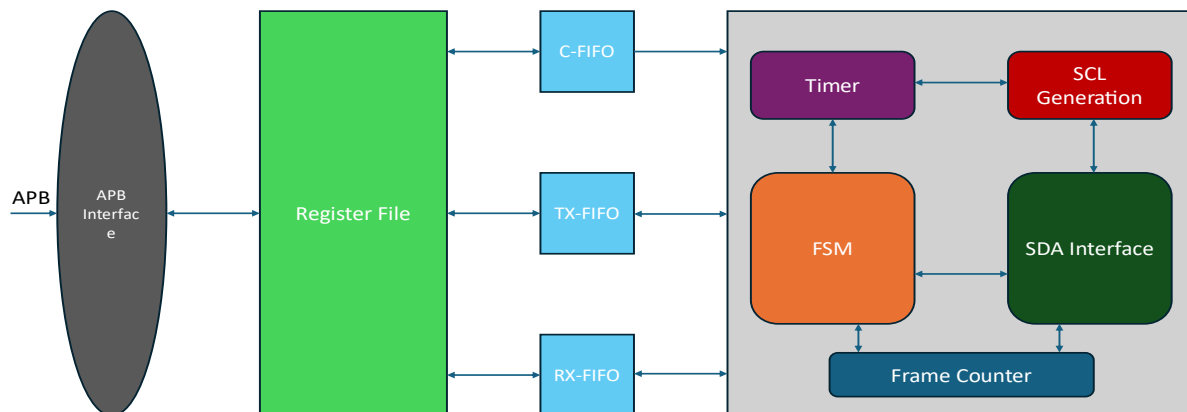


Figure 62 Communication Mechanism Between Blocks

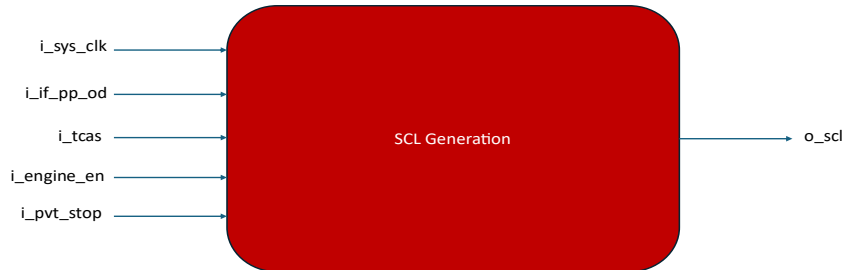
5.1 SCL Generation

5.1.1 Functionality and Implementation

The SCL generation module is responsible for generating the serial clock (SCL) signal on the SCL line at two different data rates: Push-Pull (12.5 MHz) and Open-Drain (400 KHz). The module takes a 50 MHz clock as input from the clock divider. The module consists of two main sub-blocks: a finite state machine (FSM) and a counter. The FSM has two states, LOW and HIGH, which control whether the SCL line is held low or high. The internal signal "switch" is used in conjunction with another counter to toggle the state of the SCL signal. The counter value depends on the desired frequency, counting to 125 or 2 cycles for the Open-Drain or Push-Pull data rates, respectively.

The module also supports the ability to disable the clock using the " i_pvt_stop" input signal. When this signal is active, the FSM remains in HIGH state, keeping the clock signal constant and the SCL line continuously high.

5.1.2 SCL Block



5.1.3 Block I/Os

Signal	Direction	Description
i_sys_clk	input	50 MHz clock input
i_engine_en	input	Enable signal to initiate the operation
i_if_pp_od	input	Used to select the type of SCL signal generation: 1 or Push-Pull mode, and 0 for Open-Drain mode
i_tcas	input	Clock after start timing (used for START bit timing)
i_pvt_stop	input	Indication for stopping the operation and drive SCL high
o_scl	output	Generated SCL signal

5.2 Frame Counter

5.2.1 Functionality and Implementation

The number of frames required for data transmission in either the SDR mode or the I2C mode is determined by the host. The host specifies the maximum number of frames to be sent or received by writing this value into the register file. In our system, we use a signal called "i_fcnt_no_frms," which is an 8-bit wide signal directly output from the register file. This signal serves the purpose of determining the number of frames to be transmitted or received. The block has one output "o_fcnt_last_frame" that goes high when the last frame is being transmitted informing the I3C Engine or the other blocks to stop transmitting or receiving the data frames.

5.2.2 Block Diagram

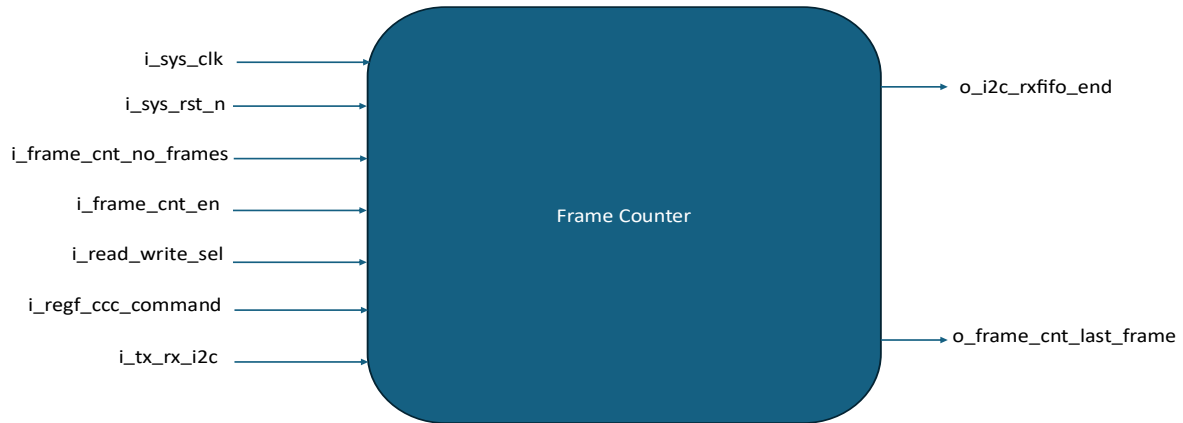


Figure 64 Frame Counter Block Diagram

5.2.3 Block I/Os

Signal	Direction	Description
i_sys_clk	input	50 MHz clock input
i_sys_rst_n	input	Asynchronous active-low reset signal
i_frame_cnt_no_frames	input	Number of data bytes chose by software
i_frame_cnt_en	input	Enable signal for frame counting
i_read_write_sel	input	Selection between read and write operations
i_tx_rx_i2c	input	Signal to count data bytes in I2C
i_regf_ccc_command	input	Count number of data for some commands
o_i2c_rxfifo_end	output	Indication for last frame in I2C
o_frame_cnt_last_frame	output	Indication for last frame

5.3 Timer

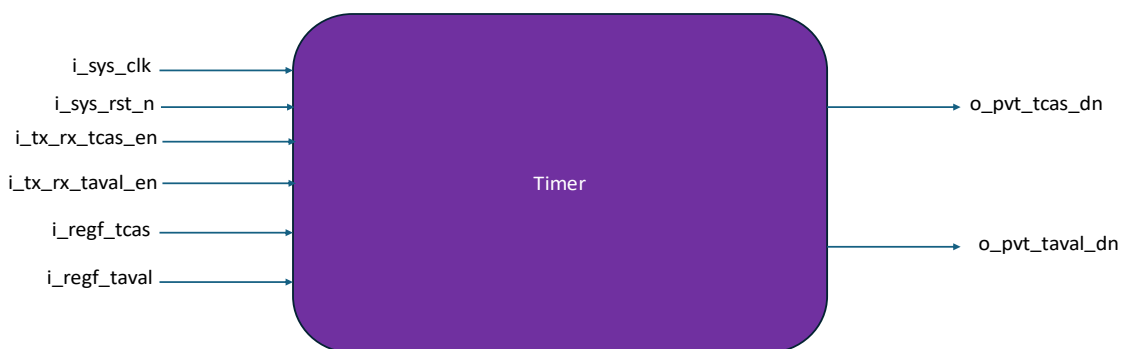
5.3.1 Functionality and Implementation

To ensure compliance with the I3C protocol's strict timing specifications, a dedicated hardware timer module was implemented. This module is responsible for generating two critical timing delays mandated by the standard: `t_cas` (Clock After START) and `t_aval` (Bus Available Condition). The module is designed to be controlled directly by the I3C Kernel's main Finite State Machine (FSM), offloading the complexity of stateful counting and allowing the FSM to operate more efficiently.

The timer module is a synchronous digital circuit that contains two independent, parallel timing units within a single module: one for the `t_cas` parameter and one for the `t_aval` parameter. This parallel architecture allows both timing functions to be conceptually separate and simplifies the control logic, at the minor expense of requiring two separate counter registers.

This implementation provides a robust and simple-to-control timing utility. By delegating the counting task to this module, the I3C Kernel's FSM design is significantly simplified, as it only needs to manage the level-sensitive enable signals rather than initiating and monitoring a pulse-based handshake.

5.3.2 Block Diagram



5.3.3 Block I/Os

Signal	Direction	Description
i_sys_clk	input	50 MHz clock input
i_sys_rst_n	input	Asynchronous active-low reset signal
i_tx_rx_tcas_en	input	Enable signal from the I3C Kernel, The t_cas timer is active for the entire duration that this signal is held high.
i_tx_rx_taval_en	input	Enable signal for the t_aval timer
i_regf_tcas	input	A 7-bit value loaded from the I3C register file, defining the target count for the t_cas delay. This makes the delay programmable by the host processor
i_regf_taval	input	An 8-bit value from the register file defining the target count for the t_aval delay
o_pvt_tcas_dn	output	A signal that pulses high for a single clock cycle to indicate that the t_cas timer has completed its count
o_pvt_taval_dn	output	A single-cycle pulse indicating the completion of the t_aval timer

5.4 SDA Interface

5.4.1 Functionality and Implementation

This block contains the TX and RX of the kernel. It has 9 operating modes that are defined using the "i_pvt_tx_rx_mode" signal from FSM block.

Table 28 Controller Tx Operating Modes

<i>i_pvt_tx_rx_mode</i> Value	Operating Mode
0000	START_BIT
0001	RE_START_BIT
0010	SERIALIZING
0011	DESERIALIZING
0100	ZERO_ACK_TX
0101	ONE_NACK_TX
0111	ACK_NACK_RX
1000	STOP
1001	DETECT_START

- **START_BIT:** Generates the start bit by waiting till the SCL is high and then pulling the sda low for time "tcas" that is output from the timer block.
- **SERIALIZING:** It's the core of the SDA interface and this mode serializes the 8-bit data packets and addresses from the register file and sends them on the SDA line.
- **DESERIALIZING:** It's the core of the SDA interface and this mode deserializes the 8-bit data packets and addresses from SDA line and sends them to the register file.
- **STOP:** It sends stop bit by waiting for SCL line to be high and then pulls SDA line to high.
- **ONE_NACK_TX:** Used by the controller wants to terminate the request when the target sends its address by leaving the bus on the high impedance state.
- **ZERO_ACK_TX:** Used when the controller wants to send the ACK bit after accepting the request.
- **REPEATED_START:** Like the start bit, but the repeated start usually comes between two frames without a STOP bit.
- **ACK_NACK_RX:** The controller receives ACK or NACK from the target.
- **DETECT_START:** The controller detects a start from the target in IBI and hot-join.

5.4.2 Block I/Os

Signal	Direction	Description
i_sys_clk	input	50 MHz clock input
i_sys_rst_n	input	Asynchronous active-low reset signal
i_scl	input	Serial clock signal
i_tx_rx_en	input	Enable signal from the FSM block
i_t_bit	input	Ninth bit sent with data byte
i_regf_wr_rd	input	Used to select the Read / Write operation: 1 for Read and 0 for Write
i_regf_cr_init	input	Used to select between: 1 for Controller and 0 for Target
i_pvt_size_mode	input	Size selection for serializing / deserializing state: 00 for Address, 01 for Data and 10 for CCC address/Data
i_pvt_tx_rx_mode	input	Mode for selection of functions
i_sda_data	input	SDA bus input
i_address_7e	input	Indication signal to send 7e address
i_ctrl_first_time	input	To differentiate between sending first address(7e) or target address
i_ctrl_daa	input	Enable Signal for daa operation
i_ctrl_ccc	input	Enable Signal for CCC operation
i_ctrl_i2c	input	Enable Signal for I2C operation
i_ctrl_i2c_data_r	input	Signal to differentiate between I3C and i2C data
i_regf_ccc_cmd	input	8-bit CCC command from register file
i_daa_7e_read	input	Signal for read operation in DAA to read ID,BCR and DCR of the target
i_regf_data	input	Frame Data read from the register file
i_regf_address	input	Target Address
i_ser_en	input	Serializer enable signal

i_taval_done	input	Done signal from Timer block that indicates of completing bus available condition
i_tcas_done	input	Done signal from Timer block that indicates of completing clock after start time
i_devri_tgt_address	input	Target address stored in register file in DAA
o_scl_pp_od	output	Used to select the type of SCL signal generation: 1 for Push-Pull mode, and 0 for Open-Drain mode
o_sda_data	output	Serialized data output on the line
	output	
o_pvt_mode_done	output	Indicator for Pvt_Msg block about mode done
o_pvt_ack_nack	output	Ack output for the Pvt_Msg block
sda_oen	output	Enable signal for SDA tri-state buffer
o_regf_address	output	Received Address to be stored in a register file
o_reg_ibidr	output	
o_reg_ibidcnt	output	Number of data bytes in IBI
o_is_ibi_req	output	
o_start_detect	output	Start detection signal in IBI and hot-join sent to FSM block
o_taval_idle	output	Enable signal for timer block to start the bus available condition
o_tcas	output	Enable signal for timer block to start the clock after start time
o_last_frame_ibi	output	Indication signal for last data byte in IBI
o_address_match	output	Indication signal for matching address in IBI and hot-join to be sent to FSM block
o_is_hj_req	output	
o_regf_data	output	Received Data to be stored in a register file

4.1.1 Register File

4.1.1.1 Functionality and Implementation

Simple register file with depth 1024. We communicate with the host via this register file, so we write for the host in specific defined addresses, and we read from the host the data that it writes in the register file and the host is modeled as a testbench. The addresses of the register file are chosen based on the message we want to convey, and they are defined as follows:

5.5 FSM Block

5.5.1 Functionality and Implementation

- The FSM block is responsible for controlling the flow of all operations read or write messages and makes the frame. It controls the SDA interface block of transmitting and receiving frames. It sends and receives the data in both data rates, push-pull, and open-drain. It does so by controlling all the universal blocks.
- This FSM contains 10 states that are being controlled using the "current_state" and "next_state" signals

current_state Value	STATE
0000	IDLE
0001	TCAS
0010	START_CONDITION
0011	ADDRESS
0100	ACK_WAITING
0101	SERIALIZING_DATA
0110	COMMAND
0111	REPEATED_START
1000	RECIEVE_ADDRESS
1001	STOP

- **IDLE:** Idle state for controller.
- **TCAS:** Clock after start state, the state after bus available condition is achieved.
- **START_CONDITION:** To send start bit on the SDA line by driving it low.
- **ADDRESS:** Serializing the target's address stored in register file.
- **ACK_WAITING:** Wait for ACK/NACK from target state and to send ACK/NACK as a response to target's request.
- **SERIALIZING_DATA:** Serializing/Deserializing data state.
- **COMMAND:** To send an 8-bit command in CCC operations.
- **REPEATED_START:** Same as start condition but comes between two frames without stop bit.
- **STOP:** To send a stop bit by driving SDA line high when SCL is high.

Chapter 6

6 MIPI I3C Controller Implementation

6.5 FPGA Board Features

The chosen FPGA is XILINX ARTIX-7 xc7a35tcpg236-1 from Digilent and Xilinx:

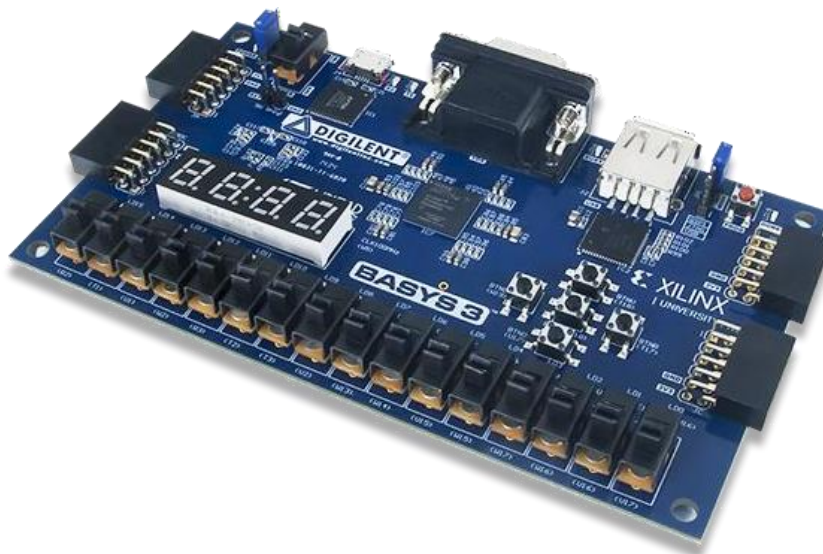


Figure 142 XILINX ARTIX-7 FPGA

It supports the following features:

Table 46 Artix-7 FPGA Specifications

System Clock	Up to 450 MHZ
Logic cells	33,280 in 5200 slices
Block RAM	1800 Kbits
DSP Slices	90
Pmod I/Os	3 connectors (each supports 8 I/Os)
Switches	16
Buttons	5
User LED	16
7-Seg	4-Digit
VGA	12-bit
USB	HID Host
I/O Voltage	3.3 V

6.6 Timing Constraints

In general, there are a variety of constraints that need to be met for the design to be validated, as:

- Timing Constraints
- Modeling the world external to the Block
- Optimization goals and Timing Exceptions

But for the design requirements and the fact that it will go through FPGA Prototyping requirements only, meaning no ASIC based constraints would be needed, hence, only Timing Constraints will be our interest.

Timing Constraints can generally cover:

- Primary clock Generation
- Generated Clock
- Transition Delay
- Clock Uncertainty
- CDC Constraints
- Input/Output Delay
- External Feedback Delay
- Logical Exclusive Clock Groups
- Physical Exclusive Clock Groups

For our FPGA Requirements, we will only have an interest in Primary Clock, Generated Clock, Input/Output Delay and Clock Uncertainty.

There are no multi-clock domains in our I3C Design as the System Clock only feeds the clock divider which in-turns feeds all the blocks, so there is no need to for CDC constraints, and the other constraints like Transition Delay and External Feedback are mainly considered when dealing with ASICs.

- Primary Clock and Generated Clock

```
## Clock Environment
create_clock -period 10 -name i_sdr_clk [get_ports i_sdr_clk]
create_generated_clock -name sys_clk_50mhz -divide_by 2 -source \
[get_ports i_sdr_clk] [get_pins u_clk_divider/o_clk_out_reg/Q]
```

Figure 143 clock and generated clock generation constraints

- Input/Output Delay Constraints

```
## Input/Output Timing
set_input_delay -clock sys_clk_50mhz -min -add_delay 1 [get_ports i_sdr_rst_n]
set_input_delay -clock sys_clk_50mhz -max -add_delay 3 [get_ports i_sdr_rst_n]
set_input_delay -clock sys_clk_50mhz -min -add_delay 1 [get_ports i_controller_en]
set_input_delay -clock sys_clk_50mhz -max -add_delay 3 [get_ports i_controller_en]
set_input_delay -clock sys_clk_50mhz -min -add_delay 1 [get_ports i_i3c_i2c_sel]
set_input_delay -clock sys_clk_50mhz -max -add_delay 3 [get_ports i_i3c_i2c_sel]

set_output_delay -clock sys_clk_50mhz -min -add_delay 1 [get_ports sda]
set_output_delay -clock sys_clk_50mhz -max -add_delay 3 [get_ports sda]
set_output_delay -clock sys_clk_50mhz -min -add_delay 1 [get_ports scl]
set_output_delay -clock sys_clk_50mhz -max -add_delay 3 [get_ports scl]
set_output_delay -clock sys_clk_50mhz -min -add_delay 1 [get_ports o_sdr_rx_valid]
set_output_delay -clock sys_clk_50mhz -max -add_delay 3 [get_ports o_sdr_rx_valid]
set_output_delay -clock sys_clk_50mhz -min -add_delay 1 [get_ports o_ctrl_done]
set_output_delay -clock sys_clk_50mhz -max -add_delay 3 [get_ports o_ctrl_done]
```

Figure 144 Input/Output Delay Constraints

- Clock Uncertainty

```
## Setup/Hold Timing
set_clock_uncertainty -setup 1 [get_clocks i_sdr_clk]
set_clock_uncertainty -setup 3 [get_clocks sys_clk_50mhz]
```

Figure 145 Clock Uncertainty Constraints

- FPGA Environment Setting

```
## FPGA Environment Setting
set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCC0 [current_design]
```

Figure 146 FPGA Environment Setting

- Pin Mapping

```
## Port Mapping
set_property PACKAGE_PIN W5 [get_ports i_sdr_clk]
set_property IOSTANDARD LVCMOS33 [get_ports i_sdr_clk]

set_property PACKAGE_PIN W16 [get_ports {i_sdr_rst_n}]
set_property IOSTANDARD LVCMOS33 [get_ports {i_sdr_rst_n}]

set_property PACKAGE_PIN V16 [get_ports {i_controller_en}]
set_property IOSTANDARD LVCMOS33 [get_ports {i_controller_en}]

set_property PACKAGE_PIN V17 [get_ports {i_i3c_i2c_sel}]
set_property IOSTANDARD LVCMOS33 [get_ports {i_i3c_i2c_sel}]

set_property PACKAGE_PIN E19 [get_ports {o_sdr_rx_valid}]
set_property IOSTANDARD LVCMOS33 [get_ports {o_sdr_rx_valid}]

set_property PACKAGE_PIN U19 [get_ports {o_ctrl_done}]
set_property IOSTANDARD LVCMOS33 [get_ports {o_ctrl_done}]

set_property PACKAGE_PIN U16 [get_ports {o_reset_debug}]
set_property IOSTANDARD LVCMOS33 [get_ports {o_reset_debug}]

set_property PACKAGE_PIN V19 [get_ports {o_en_debug}]
set_property IOSTANDARD LVCMOS33 [get_ports {o_en_debug}]

set_property PACKAGE_PIN A14 [get_ports {scl}]
set_property IOSTANDARD LVCMOS33 [get_ports {scl}]
set_property PULLUP true [get_ports scl]

set_property PACKAGE_PIN A16 [get_ports {sda}]
set_property IOSTANDARD LVCMOS33 [get_ports {sda}]
set_property PULLUP true [get_ports sda]
```

Figure 147 Pin Mapping

Time Budget:

We agreed for a clock budget as follows:

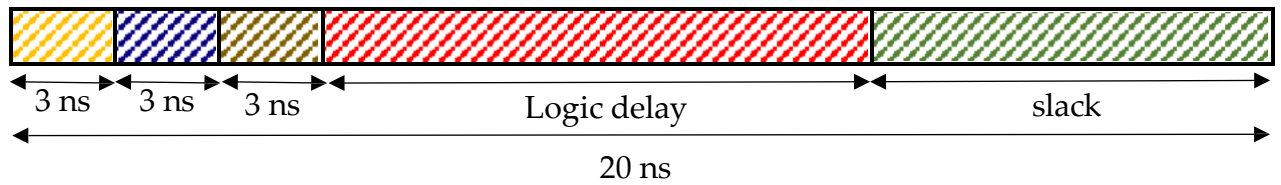


Figure 148 Design Time Budget

Normally, clock uncertainty especially setup delay would be 15 ~ 30 % from clock period, same for input and output delays, so here we have:

- **Setup delay** = 3 ns
- **Input delay** = 3 ns (max value = 3 ns while min value = 1 ns)
- **Output delay** = 3 ns (max value = 3 ns while min value = 1 ns)
- Logic will have a free delay of **11 ns** before we get negative slack

Also, for pin mapping, pins on the FPGA Kit will be as follows:

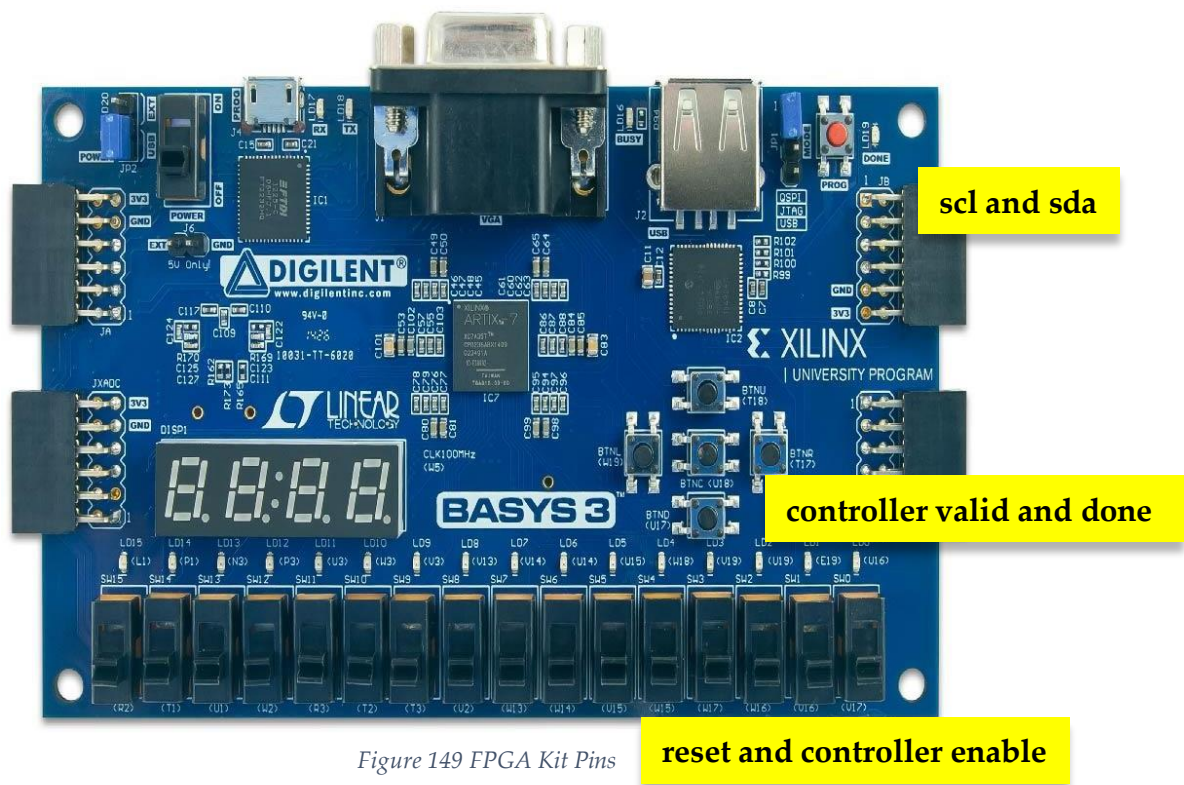


Figure 149 FPGA Kit Pins

6.7 Vivado Xilinx – AMD FPGA Flow

6.7.1 Elaboration

Resulted schematic from Elaboration:

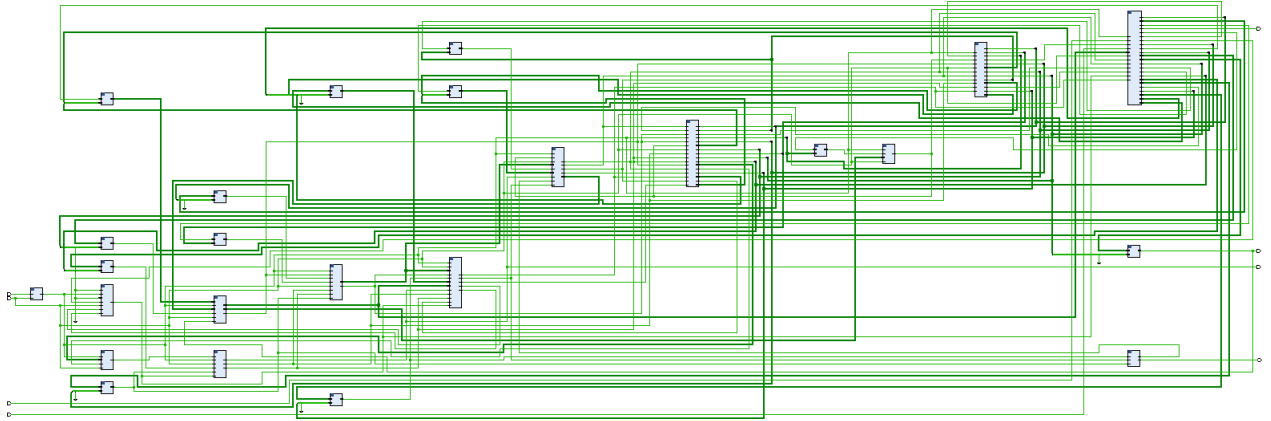


Figure 150 Elaboration output schematic

If we wanted to look in depth of a certain block after elaboration, we will find that our design translated to gates, muxs and Flip-Flops, for example, controller-tx block after elaboration:

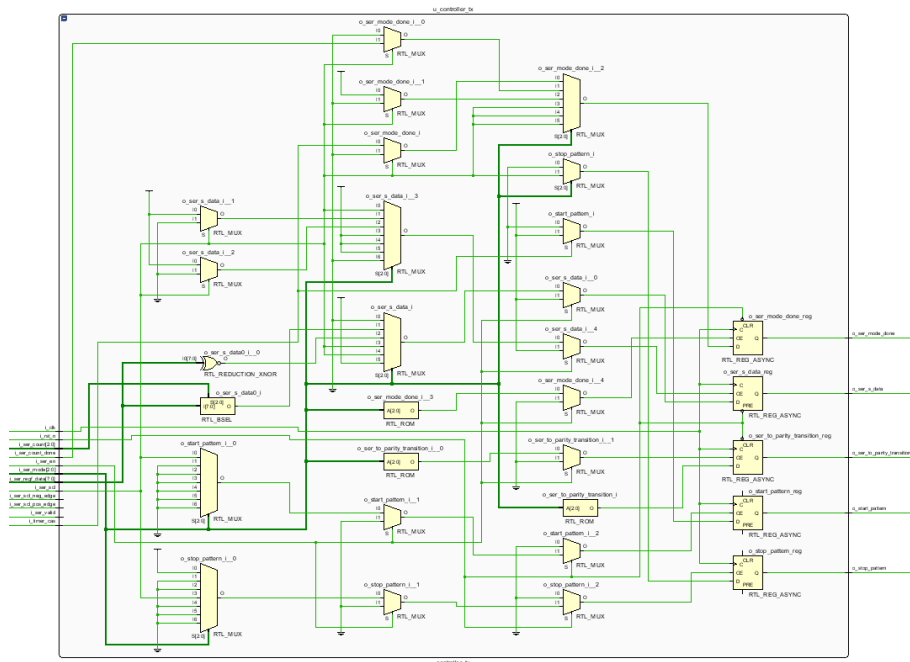


Figure 151 controller_tx after elaboration

6.7.2 Synthesis

Resulted schematic from Synthesis:

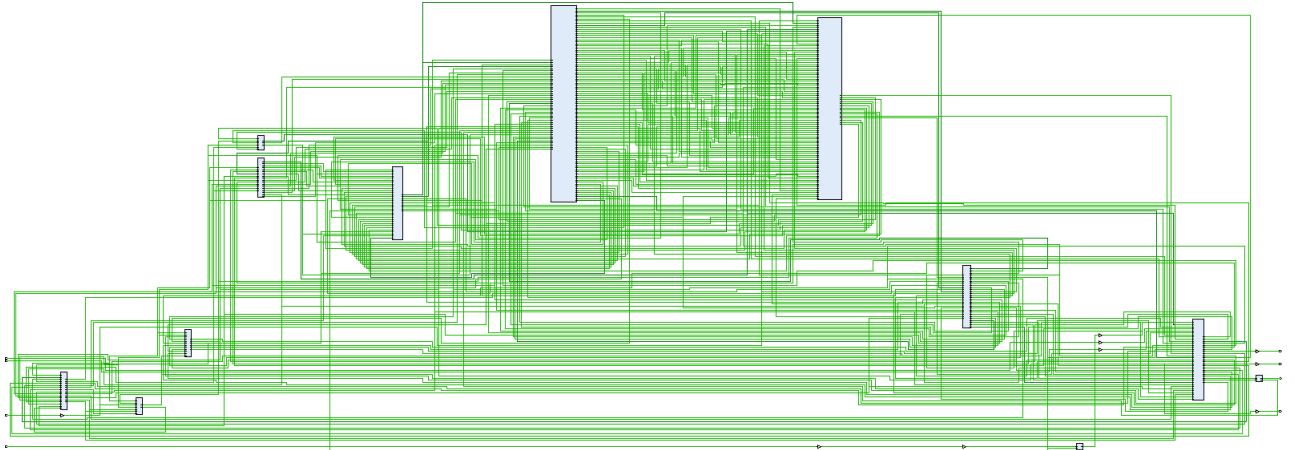


Figure 152 synthesis output schematic

Also, If we wanted to look in depth of a certain block after synthesis, we will find that our design translated to LUTs for example, controller-tx block after synthesis:

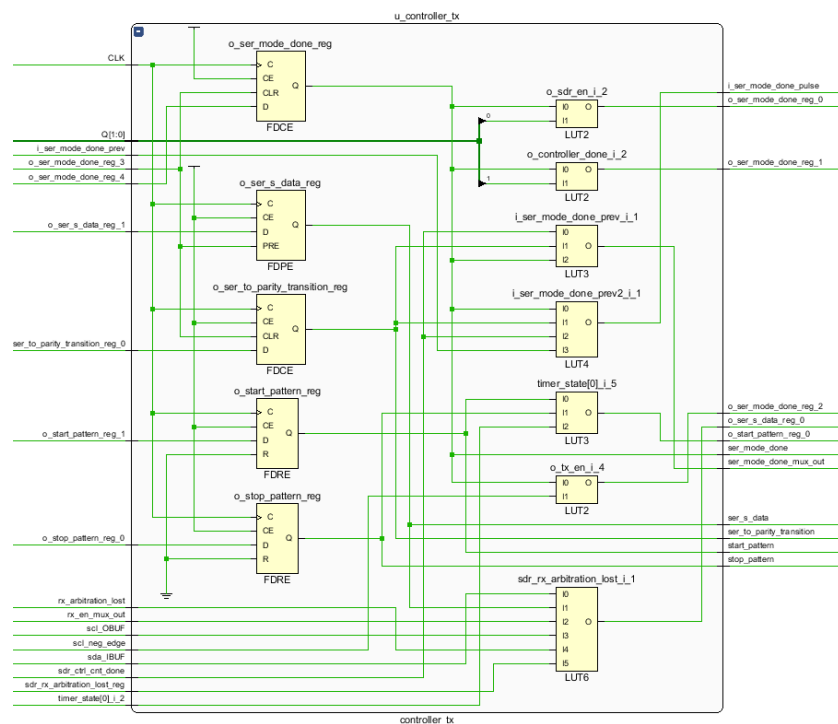


Figure 153 controller_tx after synthesis

For timing after synthesis, we get the following:

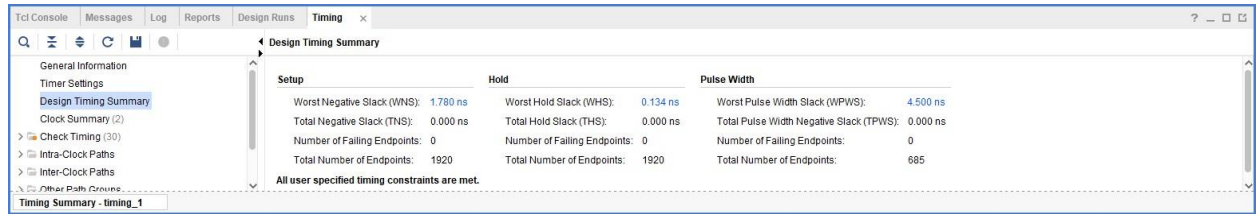


Figure 154 timing result after synthesis

We can notice a positive slack of **1.780 ns**, that resulted after multiple iterations of solving timing violations that resulted due to our restricted old Time budget, where we assumed setup time to be 4 ns and input/output delay of 4 ns which resulted in the following:

Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): -0.220 ns	Worst Hold Slack (WHS): 0.134 ns	Worst Pulse Width Slack (WPWS): 4.500 ns
Total Negative Slack (TNS): -0.220 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 1	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 1920	Total Number of Endpoints: 1920	Total Number of Endpoints: 685

Timing constraints are not met.

Figure 155 negative slack violation

And the resulted utilization pf resources after synthesis:

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	F8 Muxes (8150)	Bonded IOB (106)	BUFGCTRL (32)
i3c_controller_top	491	688	62	30	8	2
u_bits_counter (bits_counter)	16	4	0	0	0	0
u_clk_divider (clk_divider)	1	1	0	0	0	0
u_controller_rx (controller_rx)	3	19	0	0	0	0
u_controller_tx (controller_tx)	5	5	0	0	0	0
u_frame_counter (frame_counter)	3	5	0	0	0	0
u_i2c_legacy_mode (i2c_legacy_mode)	124	23	1	0	0	0
u_i3c_engine (i3c_engine)	63	28	0	0	0	0
u_i3c_timer (i3c_timer_fsm)	32	28	0	0	0	0
u_reg_file (reg_file)	162	529	61	30	0	0
u_scl_generation (scl_generation)	14	12	0	0	0	0
u_sdr_mode (sdr_mode)	68	34	0	0	0	0

Figure 156 utilization after synthesis

Resulted Clock Network and Interaction:

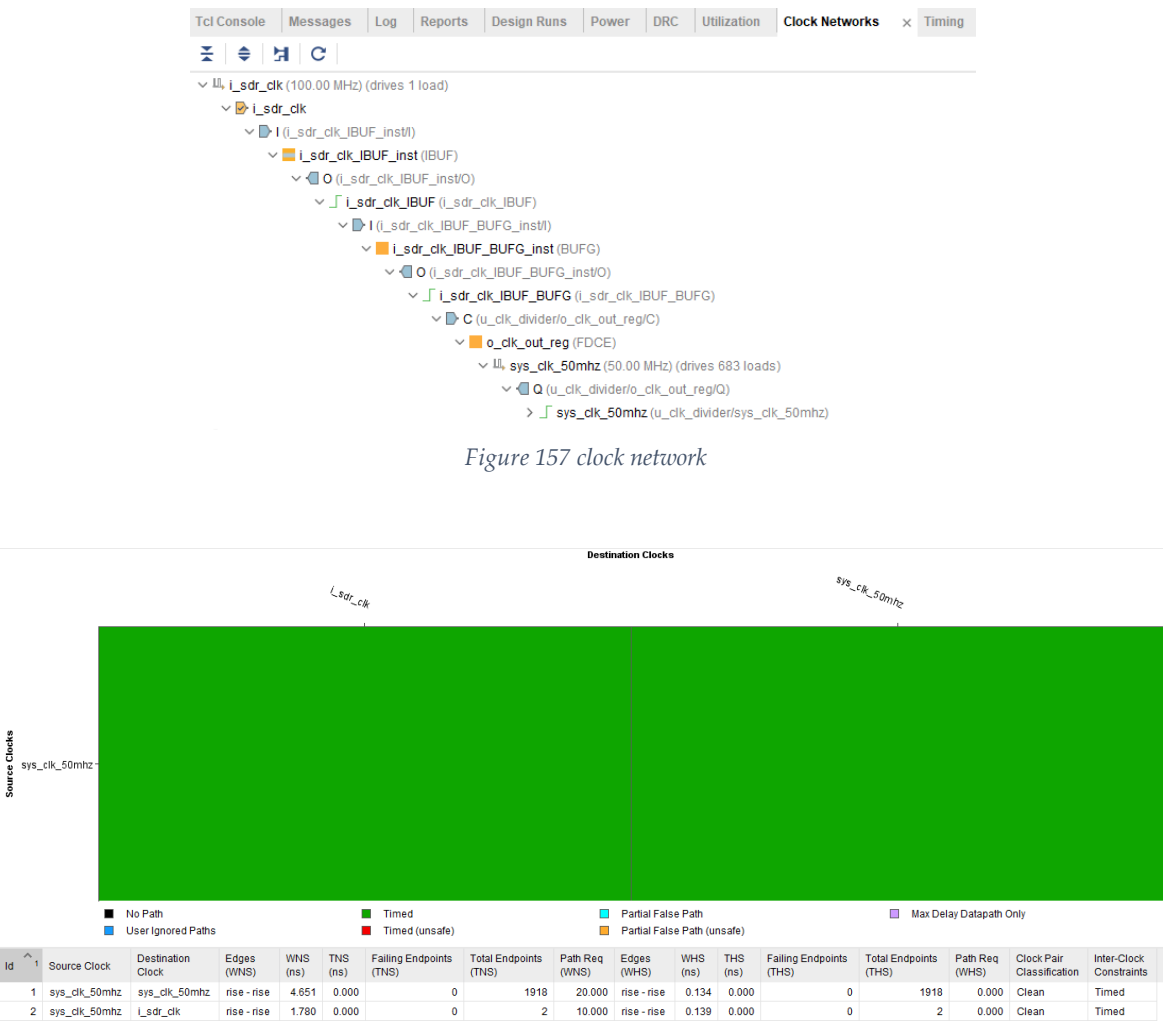
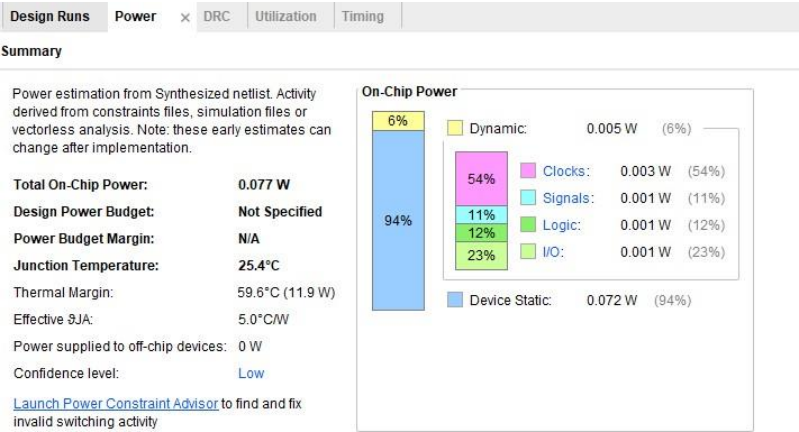


Figure 158 clock interaction

And for system power usage estimation after synthesis:



We can notice that dynamic power is 6% of the total power, which is valid.

6.7.3 Implementation

Resulted Implementation of our design on the device:

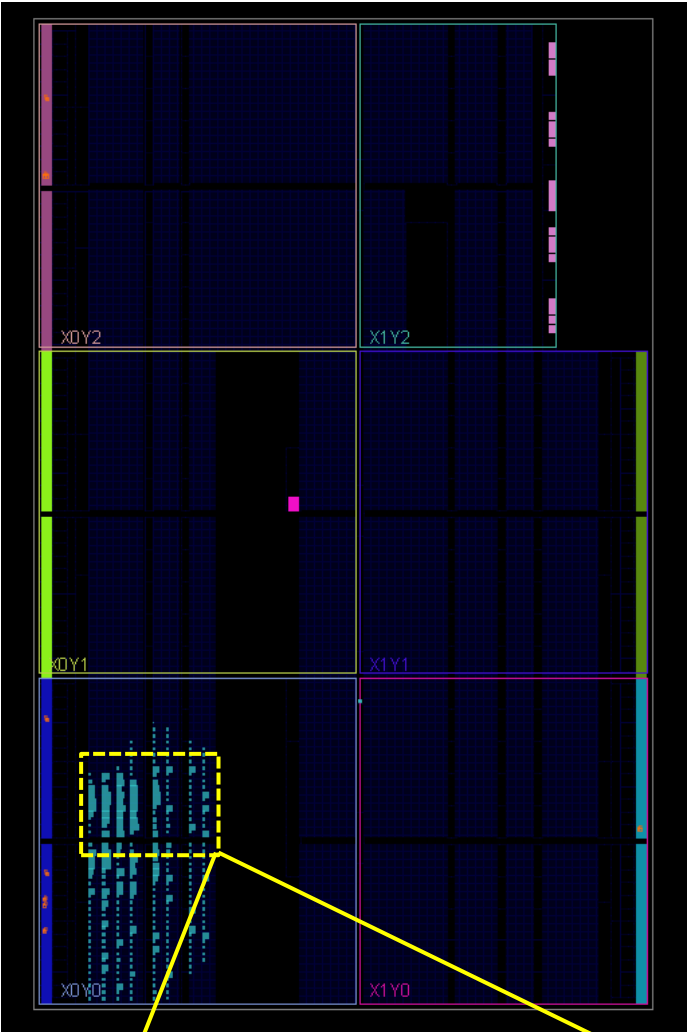


Figure 159 implementation results

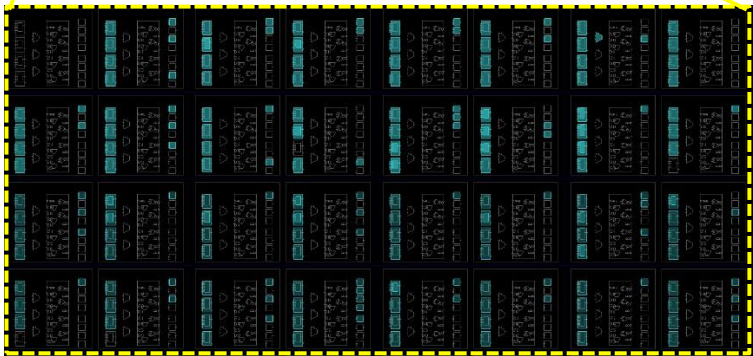


Figure 160 zoom on LUTs of the design

For timing after implementation, we get the following:

Design Timing Summary			
Setup		Hold	Pulse Width
Worst Negative Slack (WNS): 0.606 ns		Worst Hold Slack (WHS): 0.110 ns	Worst Pulse Width Slack (WPWS): 4.500 ns
Total Negative Slack (TNS): 0.000 ns		Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 0		Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 1922		Total Number of Endpoints: 1922	Total Number of Endpoints: 686
All user specified timing constraints are met.			

Figure 161 timing after implementation

Utilization after implementation:

Name	^1	Slice LUTs (20800)	Bonded IOB (106)	BUFGCTRL (32)	Slice Registers (41600)	F7 Muxes (16300)	F8 Muxes (8150)	Slice (8150)	LUT as Logic (20800)
▼ i3c_controller_top		489	8	2	689	62	30	590	489
u_bits_counter (bits_counter)		16	0	0		0	0	11	16
u_clk_divider (clk_divider)		1	0	0		0	0	1	1
u_controller_rx (controller_rx)		3	0	0		0	0	14	3
u_controller_tx (controller_tx)		5	0	0		0	0	9	5
u_frame_counter (frame_counter)		3	0	0		0	0	3	3
u_i2c_legacy_mode (i2c_legacy_mode)		124	0	0		1	0	59	124
u_i3c_engine (i3c_engine)		63	0	0		0	0	24	63
u_i3c_timer (i3c_timer_fsm)		32	0	0		0	0	16	32
u_reg_file (reg_file)		161	0	0		61	30	522	161
u_scl_generation (scl_generation)		14	0	0		0	0	6	14
u_sdr_mode (sdr_mode)		68	0	0		0	0	30	68

Figure 162 utilization after implementation

Notice that **Artix-7** FPGA has optimized Area usage which appear when we compare utilization after synthesis *Figure 99* and utilization after implementation *Figure 105*

6.7.4 Bitstream Generation

After finalizing the flow above, the next station of the flow would be FPGA Program and debugging section, by clicking Generate Bitstream we get a .bit file that is a hex file of the designed system:

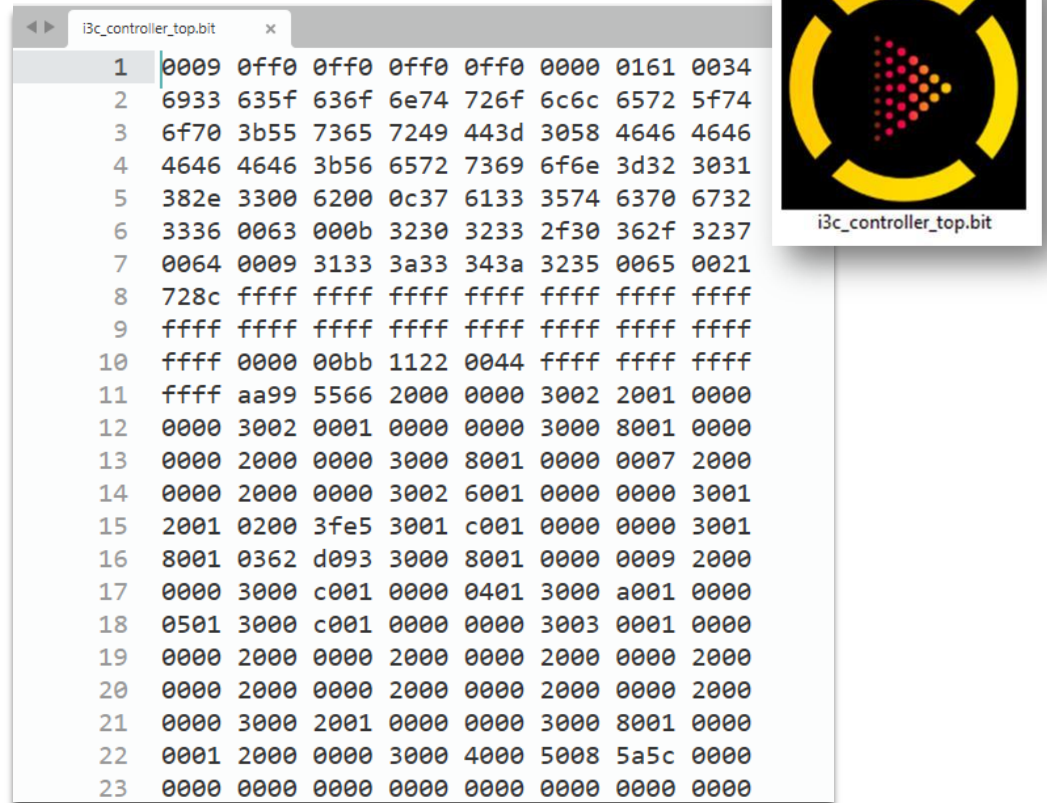


Figure 163 bitstream file

6.8 FPGA Debugging

After uploading the bitstream file on the FPGA, debugging the design on the FPGA Hardware is required and here, Integrated Logic Analyzer is one of the best approaches that assures a Function-Like simulation but with signals running on a real-time hardware.

– First, Designing the Integrated Logic Analyzer (ILA):

From IP Catalogue, we choose Debugging and Verification and then choose ILA:

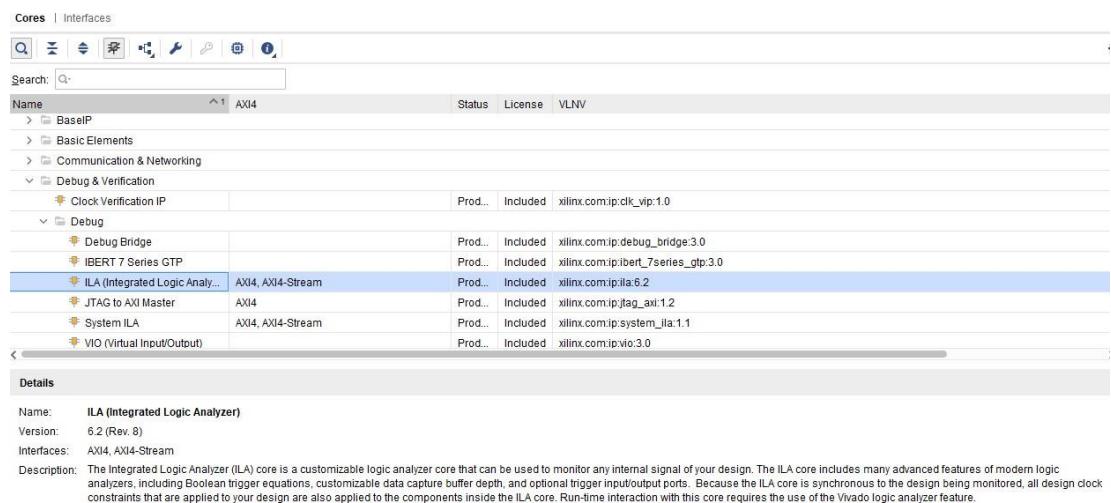


Figure 164 ILA from IP Catalogue

Then we design our ILA as follows:

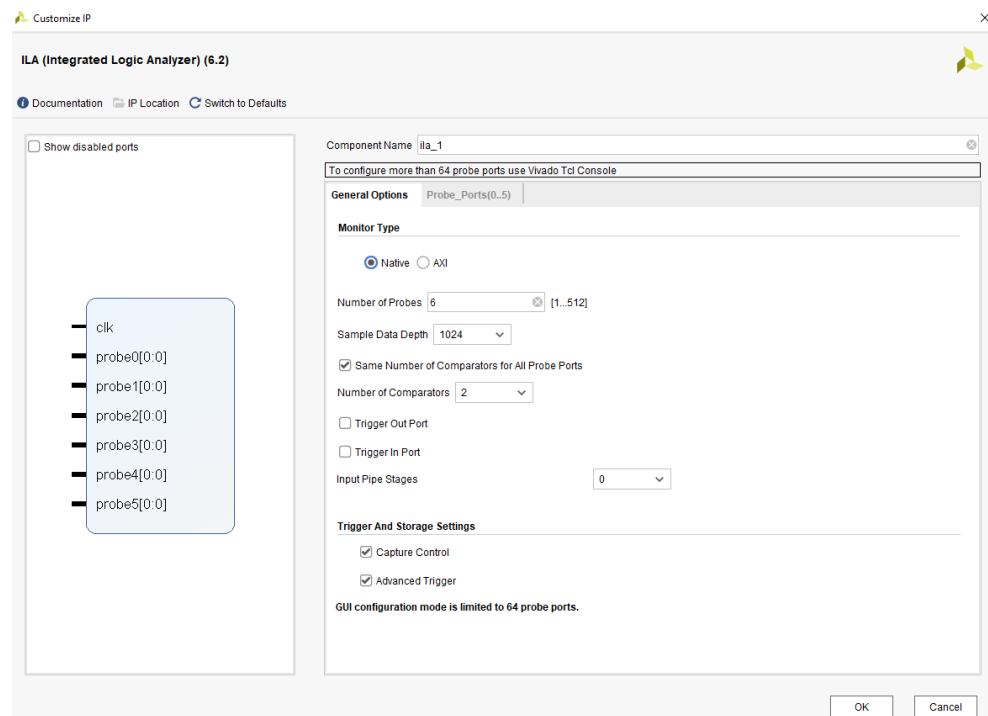
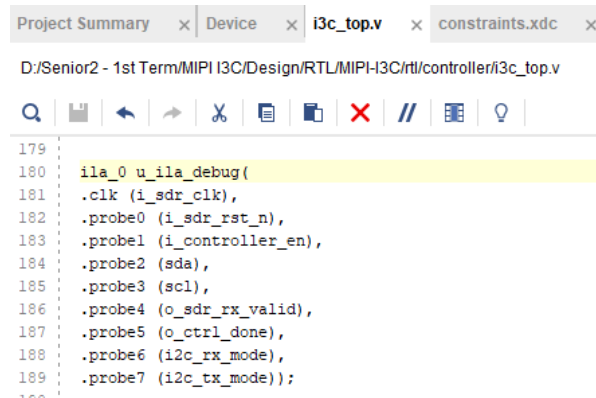


Figure 165 ILA Design

This will result in a module of the ILA that we can instantiate in our top module and connect the signals we would like to monitor to the probes inputs of the ILA:



```

179
180   ila_0 u_ila_debug(
181     .clk (i_sdr_clk),
182     .probe0 (i_sdr_rst_n),
183     .probe1 (i_controller_en),
184     .probe2 (sda),
185     .probe3 (scl),
186     .probe4 (o_sdr_rx_valid),
187     .probe5 (o_ctrl_done),
188     .probe6 (i2c_rx_mode),
189     .probe7 (i2c_tx_mode));

```

Figure 166 ILA Instance

Here we would like to debug our design by monitoring these signals:

- **sda and scl:** as they are our main bus, and all the data will be on them
 - **reset and enable:** to program the ILA to trigger when bot change to monitor bus starting conditions
 - **rx and tx mode signals:** to monitor the state of the bus driving blocks which are our controller_tx and controller_rx
- **Second, Re-running the FPGA Flow including the ILA:**

The Logic Analyzer will be integrated in our system as follows:

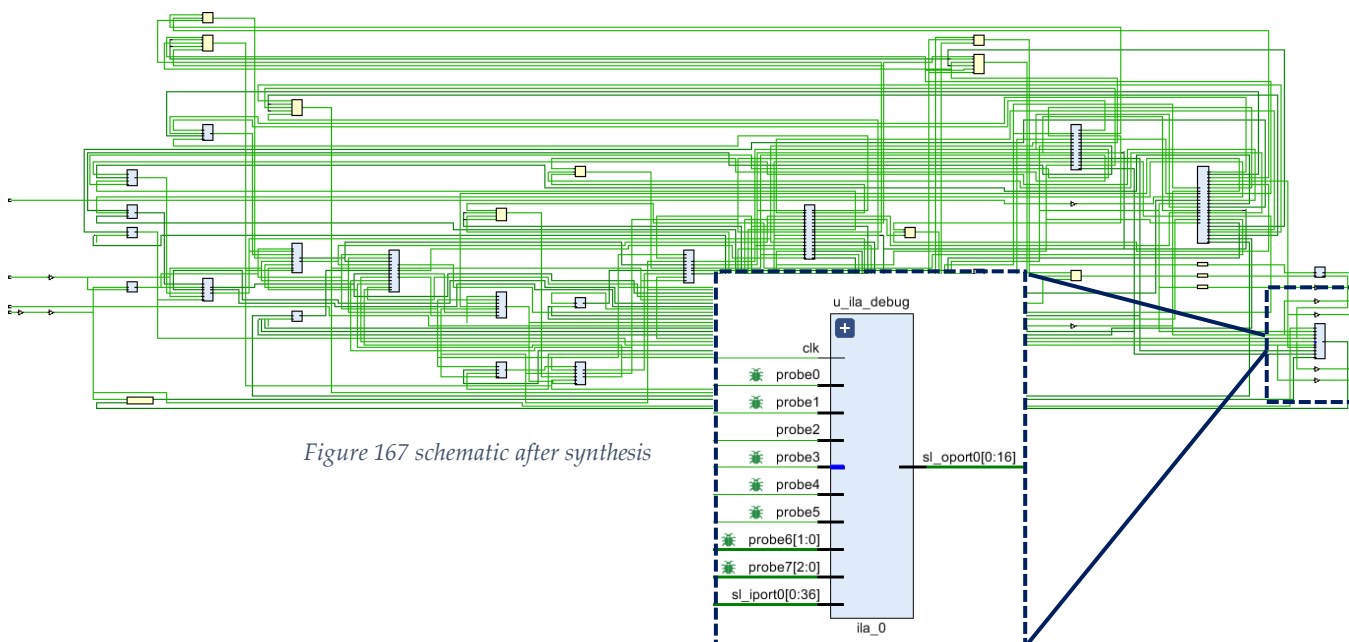


Figure 167 schematic after synthesis

Figure 168 Integrated Logic Analyzer

Resulted timing after implementation with ILA:

Design Timing Summary			
Setup		Hold	Pulse Width
Worst Negative Slack (WNS):		Worst Hold Slack (WHS):	Worst Pulse Width Slack (WPWS):
1.133 ns		0.024 ns	3.750 ns
Total Negative Slack (TNS):		Total Hold Slack (THS):	Total Pulse Width Negative Slack (TPWS):
0.000 ns		0.000 ns	0.000 ns
Number of Failing Endpoints:		Number of Failing Endpoints:	Number of Failing Endpoints:
0		0	0
Total Number of Endpoints:		Total Number of Endpoints:	Total Number of Endpoints:
11955		11939	6105
All user specified timing constraints are met.			

Figure 169 timing after implementation with ILA

Notice the huge increase in the total number of endpoints due to the integration of the ILA.

Hence, we can connect our FPGA and upload our new bitstream file including the ILA, and a use case we can use the ILA is checking the bus initialization with the start condition timing examination as follows:

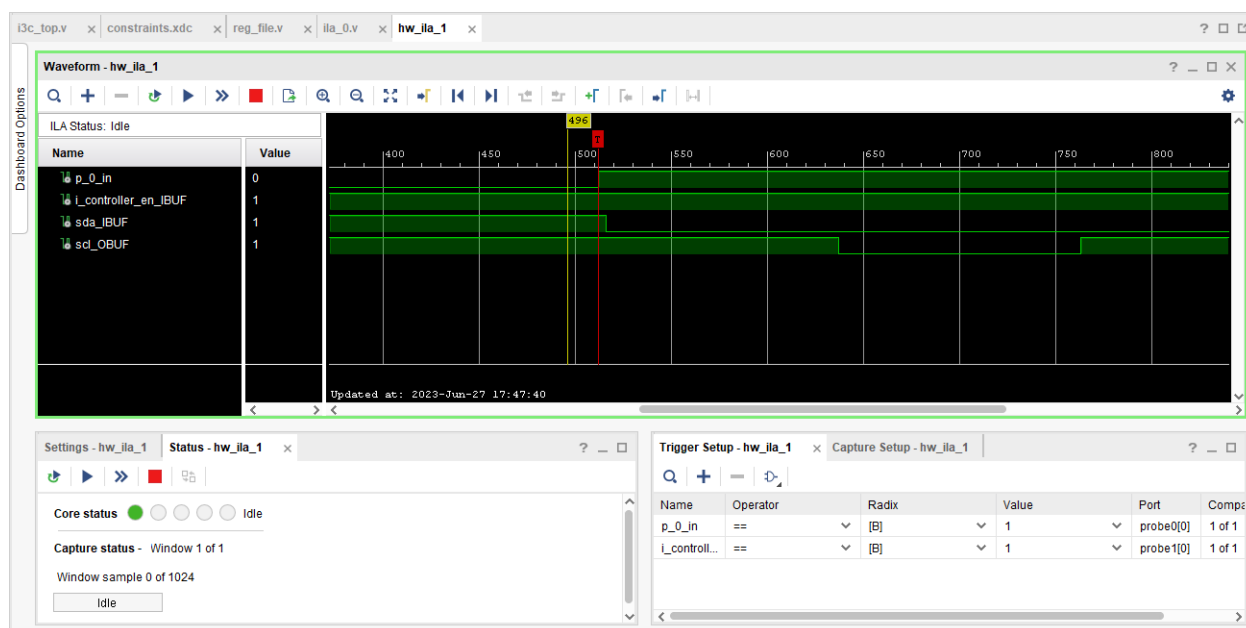


Figure 170 start condition on FPGA using ILA

Our trigger setup here is when the controller_en signal equals 1 and the reset signal equals 1, which reflects the start of the I3C Flow.

The Red Trigger cursor rises when both the reset signal and the enable signal are 1, announcing the start of the bus to be initialized, and the ILA shows a capture of the waveform after this condition, where the start condition occur as the scl signal falls to 0 while sda is high and then the sda signal falls to 0 after t_{CAS} from the scl falling time.

6.9 How to choose the perfect FPGA for your Design?

Before we stick to our ARTIX-7 FPGA, we took a little trip in FPGA world that different vendors offer and also different models form the same vendor, we worked with the following FPGAs in our design and got the same results **functionally**, but they are far different in **optimization and constraining** aspects:

1. **XILINX SPARTAN-7 XC7S15:** from Xilinx AMD

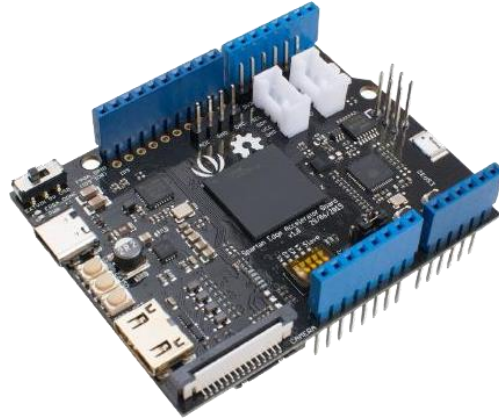


Figure 171 XILINX SPARTAN-7 XC7S15 FPGA

2. **ALTERA CYCLONE IV FPGA:** from Intel Altera

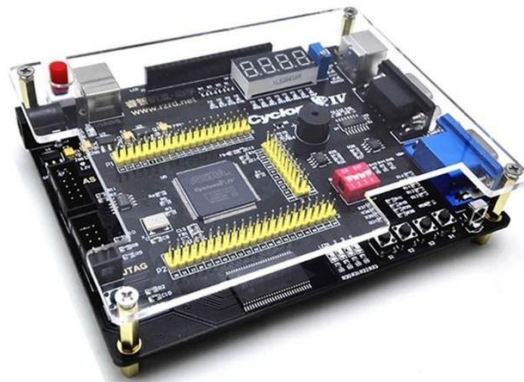


Figure 172 ALTERA CYCLONE IV FPGA

3. **XILINX ARTIX-7 xc7a35tcpg236-1:** from Xilinx AMD and Digilent

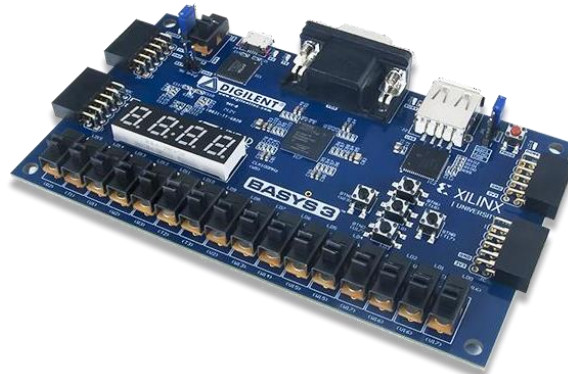


Figure 173 XILINX ARTIX-7 xc7a35tcpg236-1

After using the three FPGAs with our design, we found the following results:

- **For Timing Closure strictness:** using the same timing constraints for the three.

Table 47 Timing Closure for the Three FPGAs

<div>Spartan-7 FPGA</div>	<div><div>It was moderate for our assumed time budget:</div><div>we based our time budget to be mainly strict, just for our design to be compatible with any timing environment whatever how strict it gets, and Spartan-7 showed great timing results with a positive slack of 0.571 ns:</div></div> <div><table><thead><tr><th>Setup</th><th>Hold</th><th>Pulse Width</th></tr></thead><tbody><tr><td>Worst Negative Slack (WNS): 0.571 ns</td><td>Worst Hold Slack (WHS): 0.175 ns</td><td>Worst Pulse Width Slack (WPWS): 4.500 ns</td></tr><tr><td>Total Negative Slack (TNS): 0.000 ns</td><td>Total Hold Slack (THS): 0.000 ns</td><td>Total Pulse Width Negative Slack (TPWS): 0.000 ns</td></tr><tr><td>Number of Failing Endpoints: 0</td><td>Number of Failing Endpoints: 0</td><td>Number of Failing Endpoints: 0</td></tr><tr><td>Total Number of Endpoints: 1922</td><td>Total Number of Endpoints: 1922</td><td>Total Number of Endpoints: 686</td></tr></tbody></table><div>All user specified timing constraints are met.</div></div>	Setup	Hold	Pulse Width	Worst Negative Slack (WNS): 0.571 ns	Worst Hold Slack (WHS): 0.175 ns	Worst Pulse Width Slack (WPWS): 4.500 ns	Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Total Number of Endpoints: 1922	Total Number of Endpoints: 1922	Total Number of Endpoints: 686
Setup	Hold	Pulse Width														
Worst Negative Slack (WNS): 0.571 ns	Worst Hold Slack (WHS): 0.175 ns	Worst Pulse Width Slack (WPWS): 4.500 ns														
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns														
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0														
Total Number of Endpoints: 1922	Total Number of Endpoints: 1922	Total Number of Endpoints: 686														
<div>Altera Cyclone IV</div>	<div><div>The loosest timing closure of them all:</div><div>With the same constraints that we used for the other FPGAs; timing closure was met with a 6.7 ns positive slack:</div><div><div>Timing Closure Recommendations</div><div>Summary</div><div>This design does not contain any failing setup paths. The worst-case slack is 6.695 ns.</div><div>Top Failing Paths</div><div>No paths fail setup timing.</div></div></div>															
<div>Artix-7 FPGA</div>	<div><div>The strictest timing closure of them all:</div><div>As we used the same constraints that we used for Spartan-7 FPGA but resulted in a negative slack of 0.220 ns:</div></div> <div><div>Design Timing Summary</div><table><thead><tr><th>Setup</th><th>Hold</th><th>Pulse Width</th></tr></thead><tbody><tr><td>Worst Negative Slack (WNS): -0.220 ns</td><td>Worst Hold Slack (WHS): 0.134 ns</td><td>Worst Pulse Width Slack (WPWS): 4.500 ns</td></tr><tr><td>Total Negative Slack (TNS): -0.220 ns</td><td>Total Hold Slack (THS): 0.000 ns</td><td>Total Pulse Width Negative Slack (TPWS): 0.000 ns</td></tr><tr><td>Number of Failing Endpoints: 1</td><td>Number of Failing Endpoints: 0</td><td>Number of Failing Endpoints: 0</td></tr><tr><td>Total Number of Endpoints: 1920</td><td>Total Number of Endpoints: 1920</td><td>Total Number of Endpoints: 685</td></tr></tbody></table><div>Timing constraints are not met.</div></div>	Setup	Hold	Pulse Width	Worst Negative Slack (WNS): -0.220 ns	Worst Hold Slack (WHS): 0.134 ns	Worst Pulse Width Slack (WPWS): 4.500 ns	Total Negative Slack (TNS): -0.220 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns	Number of Failing Endpoints: 1	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Total Number of Endpoints: 1920	Total Number of Endpoints: 1920	Total Number of Endpoints: 685
Setup	Hold	Pulse Width														
Worst Negative Slack (WNS): -0.220 ns	Worst Hold Slack (WHS): 0.134 ns	Worst Pulse Width Slack (WPWS): 4.500 ns														
Total Negative Slack (TNS): -0.220 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns														
Number of Failing Endpoints: 1	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0														
Total Number of Endpoints: 1920	Total Number of Endpoints: 1920	Total Number of Endpoints: 685														

- **For Area Optimization:** using no area constraints, leaving optimization up to each FPGA

Table 48 Area optimization of the three FPGAs

Spartan-7
FPGA

It was moderate, with a difference of only one LUT from Artix-7:

490 used LUTs, 689 Slice Registers and 92 total Muxs and 599 Slices are used

Name	Slice LUTs (8000)	Bonded IOB (100)	BUFGCTRL (16)	Slice Registers (16000)	F7 Muxes (4000)	F8 Muxes (2000)	Slice (2000)	LUT as Logic (8000)
i3c_controller_top	490	8	2	689	62	30	599	490
u_bits_counter (bits_counter)	16	0	0		0	0	8	16
u_clk_divider (clk_divider)	1	0	0		0	0	1	1
u_controller_rx (controller_rx)	3	0	0		0	0	15	3
u_controller_tx (controller_tx)	5	0	0		0	0	8	5
u_frame_counter (frame_counter)	3	0	0		0	0	3	3
u_i2c_legacy_mode (i2c_legacy_mode)	124	0	0		1	0	56	124
u_i3c_engine (i3c_engine)	62	0	0		0	0	25	62
u_i3c_timer (i3c_timer_fsm)	32	0	0		0	0	16	32
u_reg_file (reg_file)	162	0	0		61	30	520	162
u_scl_generation (scl_generation)	14	0	0		0	0	8	14
u_sdr_mode (sdr_mode)	68	0	0		0	0	32	68

Altera
Cyclone IV

Nearly no area optimization occurred:

After Fitting and utilization, the results are far free of any optimization compared to the two Xilinx’s FPGAs, as 1775 LUTs with a total of 1203 registers were used

Fitter Resource Usage Summary

<<Filter>>

	Resource	Usage
1	Total logic elements	1,775 / 6,272 (28 %)
1	-- Combinational with no register	572
2	-- Register only	461
3	-- Combinational with a register	742
2		
3	Logic element usage by number of LUT inputs	
1	-- 4 input functions	868
2	-- 3 input functions	219
3	-- <=2 input functions	227
4	-- Register only	461

5

▼ Logic elements by mode

1	-- normal mode	1206
2	-- arithmetic mode	108
6		
7	▼ Total registers*	1,203 / 6,684 (18 %)
1	-- Dedicated logic registers	1,203 / 6,272 (19 %)
2	-- I/O registers	0 / 412 (0 %)
8		
9	Total LABs: partially or completely used	140 / 392 (36 %)
10	Virtual pins	0
11	▼ I/O pins	9 / 92 (10 %)
1	-- Clock pins	3 / 3 (100 %)
2	-- Dedicated input pins	3 / 9 (33 %)

Artix-7
FPGA

The strictest Area optimization (in numbers) with no difference from Spartan-7 FPGA: The resulted utilization of FPGA resources has proved that there are 489 used LUTs, 689 Slice Registers and 92 total Muxs and 590 Slices

Name	Slice LUTs (20800)	Bonded IOB (106)	BUFGCTRL (32)	Slice Registers (41600)	F7 Muxes (16300)	F8 Muxes (8150)	Slice (8150)	LUT as Logic (20800)
i3c_controller_top	489	8	2	689	62	30	590	489
u_bits_counter (bits_counter)	16	0	0		0	0	11	16
u_clk_divider (clk_divider)	1	0	0		0	0	1	1
u_controller_rx (controller_rx)	3	0	0		0	0	14	3
u_controller_tx (controller_tx)	5	0	0		0	0	9	5
u_frame_counter (frame_counter)	3	0	0		0	0	3	3
u_i2c_legacy_mode (i2c_legacy_mode)	124	0	0		1	0	59	124
u_i3c_engine (i3c_engine)	63	0	0		0	0	24	63
u_i3c_timer (i3c_timer_fsm)	32	0	0		0	0	16	32
u_reg_file (reg_file)	161	0	0		61	30	522	161
u_scl_generation (scl_generation)	14	0	0		0	0	6	14
u_sdr_mode (sdr_mode)	68	0	0		0	0	30	68

From these results, Artix-7 FPGA is the strictest in Timing and Area Closures, hence we used it to assure the strictest test and prototyping were met to make sure that our IP is ready to be used in strict timing or area environment.

References

- [1] MIPI Alliance. (2021). MIPI I3C Basic Specifications v1.1.1.
- [2] MIPI Alliance. (2022). I3C White Paper: Achieving Power Efficiency in IoT Devices.
- [3] STM. (2022). STM32H523/33xx, STM32H562/63xx, and STM32H573xx Arm®-based 32-bit MCUs.
- [4] Digilent. (2016). Basys 3™ FPGA Board Reference Manual.