

**A REPORT  
ON  
DESIGNING MIPI STANDARD RFFE INTERFACE IP  
FOR RF FRONT-END DEVICES**

**S1-24\_MELZG628T: Dissertation**

**by**

**Subramanya N N**

**2022ht80626**

**Dissertation carried out at:**

**Qualcomm India Pvt Ltd.**



**BIRLA INSTITUTE OF TECHNOLOGY &  
SCIENCE, PILANI, RAJASTHAN**

**November 2024**

# **DESIGNING MIPI STANDARD RFFE INTERFACE IP FOR RF FRONT-END DEVICES**

**Course No.: S1-24\_MELZG628T**

**Course Title: Dissertation**

**Dissertation Done by:**

**Student Name: Subramanya N N**

**BITS ID: 2022ht80626**

**Degree Program: MTech (Microelectronics)**

**Research Area: VLSI**

**Dissertation carried out at:  
Qualcomm India Pvt Ltd.**



**BIRLA INSTITUTE OF TECHNOLOGY &  
SCIENCE, PILANI, RAJASTHAN  
November 2024**

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI  
(RAJASTHAN)  
WILP Division**

**Organization:** Qualcomm.

**Location:** Bangalore

**Duration:** 2 years

**Date of Start:** 20/7/2024

**Date of Submission:** 11/11/2024

**Title of the Project:** DESIGNING MIPI STANDARD RFFE INTERFACE IP FOR RF  
FRONT-END DEVICES

**ID No./Name of the student:** 2022HT80626/ Subramanya N N

**Name (s) and Designation (s) of your Supervisor and Additional Examiner:**

**Supervisor:** Nakul Manjunath, Staff Engineer

**Additional Examiner:** Utpal Barman, Director Engineer

**Name of the Faculty mentor:** Sanjay Vidhyadharan

**Project Areas:** VLSI Architecture

**Signature of Student**

**Signature of your Supervisor**

**Date:** 11/11/2024

**Date:** 11/11/2024

# **1. Acknowledgement**

I would initially like to thank my organization Qualcomm for giving me such wonderful opportunity to pursue this project.

I would like to express my deepest gratitude to my supervisor Nakul Manjunath and Examiner Utpal Barman, whose guidance, support, and patience have been invaluable throughout the project. Their feedback and insights were crucial in shaping my research and helping me to navigate the challenges of writing a dissertation.

I would like to thank my guide Sanjay Vidhyadharan for his guidance, support, and evaluation. His feedback has played integral role in completion of the project.

I would also like to thank my colleagues, for their time, expertise, and valuable feedback. Their thoughtful critiques and suggestions have greatly improved the quality of this work.

I am grateful to the participants of this study, who generously gave their time and shared the experiences with me. Their contributions have been essential in advancing our understanding industry requirements.

I am humbled by the support and guidance of so many people, and I am grateful for the opportunity to have pursued this project.

## 2. Abstract

The Peripheral IP's are one of the crucial sub-systems in a System on Chip (SoC) which are responsible for communicating information such as Voltage rating, power rating and temperature values, etc. between baseband processor (Master) and Radio Frequency ICs in wireless communication systems. To facilitate the communication of crucial information, the Mobile Industry Processor Interface (MIPI) Standard introduced the RF Front-End protocol. The RF Front-End Control (RFFE) Interface IP is specifically introduced for the communication of RF front-end devices such as power amplifiers, low-noise amplifiers, filters, switches, antenna tuners, and sensors, which are essential components in wireless modems and mobile communication systems.

The RFFE is a two-wire serial interface designed to link one or more Radio Frequency Integrated Circuits (RFICs) in a mobile device to their corresponding Front-End Modules (FEMs) for control and monitoring purposes. It includes a bidirectional serial data signal (SDATA), which can be driven by either the Master or Slave depending on the command sequence, and a clock signal (SCLK) that is always driven by the Master.

The objectives of the project are as follows:

- Understand the MIPI standard RFFE protocol and its different command sequence between RFFE Master and Slave.
- Develop the architecture and requirements necessary to implement different command sequences supported by the RFFE interface IP for communication with RF front-end devices.
- Design a Finite State Machine (FSM) to define each phase of the command sequence, ensuring support for all types of register modes such as Register\_0 mode, Normal Register mode, Extended Register mode and Extended Register Long mode.
- Verify the design to make sure it does not break functionality of RFFE Protocol and adheres to MIPI standard.

### 3. Abbreviations

RF	Radio Frequency
RFFE	Radio Frequency Front End
FEM	Front-End Module
PA	Power Amplifier
LNA	Low Noise Amplifier
SPMI	System Power Management Interface
RFIC	Radio Frequency Integrated Circuit
MSB	Most Significant Bit
LSB	Least Significant Bit
SA	Slave Address
SSC	Sequence Start Condition
OE	Output Enable
FSM	Finite State Machine
SDATA	Serial Data Pin
SCLK	Serial Clock Pin
REG0	Register 0 Write Command
REG_WR	Register Write Command
REG_RD	Register Read Command
EXT_REG	Extended Register Command (Read/Write)
EXT_REG_WR	Extended Register Write Command
EXT_REG_RD	Extended Register Read Command
EXT_REG_LNG	Extended Register Long Command (Read/Write)
EXT_REG_WR_LNG	Extended Register Write Long Command
EXT_REG_RD_LNG	Extended Register Read Long Command

## 4. Table of contents

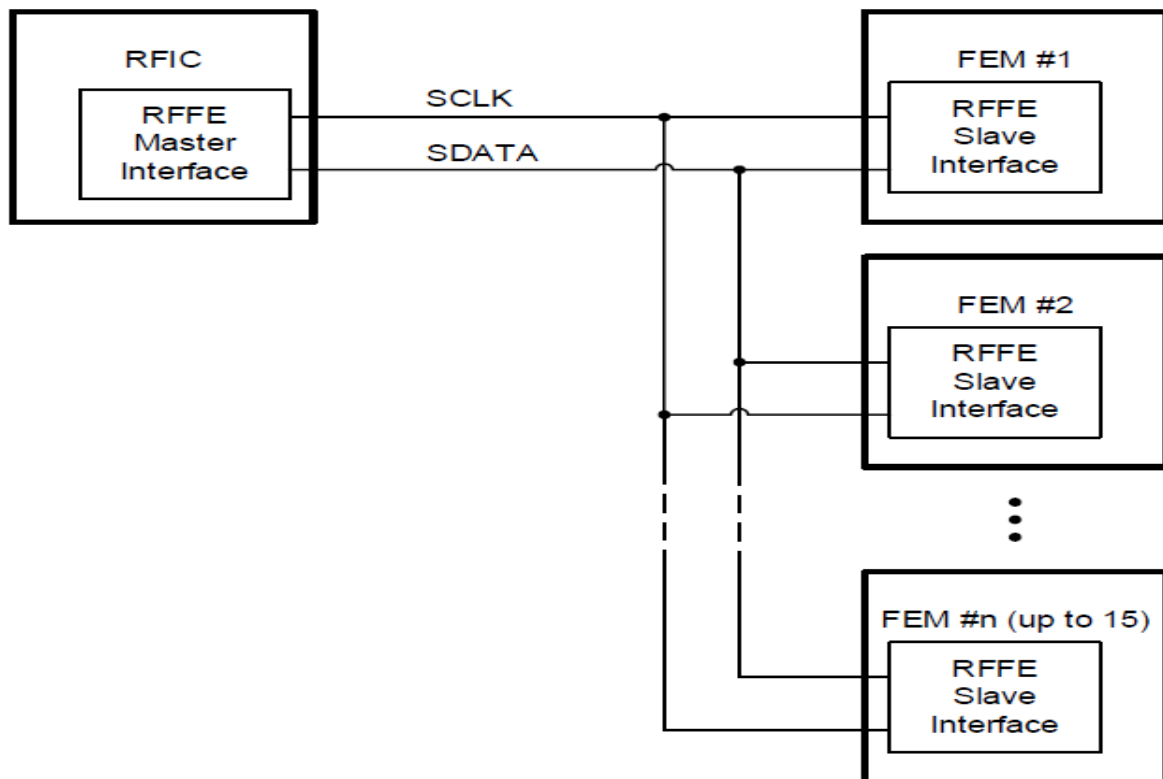
1. Acknowledgement .....	4
2. Abstract .....	5
3. Abbreviations .....	6
4. Table of contents .....	7
5. Overview of the RFFE Protocol .....	8
6. Importance of RFFE Interface in Wireless Communication Systems .....	9
7. Basic Structure of RFFE Protocol Commands .....	10
8. Power Saving Technique (Clock Gating) .....	13
9. Description of RFFE Command Sequences .....	15
10. RFFE Interface IP Controller Architecture .....	24
11. Observations and Simulations.....	30
12. Conclusion.....	38
13. Literature References .....	39
14. Timelines.....	40
15. Scanned Copy Signed Documents.....	41

## 5. Overview of the RFFE Protocol

The RFFE protocol was originally derived from a subset of the MIPI Alliance's System Power Management Interface (SPMI). It was introduced to provide a standardized method for controlling RF front-end devices. These devices include power amplifiers (PAs), low-noise amplifiers (LNAs), filters, switches, power management modules, antenna tuners, and sensors.

The RFFE is a two-wire serial interface designed to link one or more Radio Frequency Integrated Circuits (RFICs) in a mobile terminal to their corresponding Front-End Modules (FEMs) for communication purposes. It mainly consists of serial bidirectional data signal (SDATA) which will be driven by either Master or Slave based on write or read transaction and a clock signal (SCLK) which is always driven by Master. The bit ordering on the RFFE bus will be MSB first.

From the figure below, the single Master can access up to 15 slaves, hence in the command sequence we have four slave ID bits, which will be explained in detail in the further section. RFFE supports up to 16 bits of addressing with different command sequences which allows access address range from 1 Byte to 64 Kilobytes of slave device.



*Figure 1: RFFE Two-wire Bus Interface*



## 6. Importance of RFFE Interface in Wireless Communication Systems

In the fast-growing VLSI industry, the complexity of System-on-Chip (SoC) is exponentially increasing, more and more resources have been integrated on a single silicon die. As silicon technology advances further, there are several challenges and requirements in the design and operation. Below are the lists of key challenges and requirements in the present technology especially in modern wireless communication system.

- **Rising Complexity:** The evolution of wireless communication systems has led to greater complexity in RF front-end components, creating a need for a standardized protocol like RFFE which provides interoperability and simplifies integration.
- **Space Constraints:** The trend towards smaller, more compact devices has driven the need to reduce the number of control lines and pins. The RFFE protocol is a two-wire interface which significantly reduces the pin count
- **Power Efficiency:** Modern wireless communication systems mostly rely on battery power; hence power efficiency is a crucial factor. The RFFE protocol supports efficient control of RF components enabling them to switch between different power modes to conserve energy when full performance is not needed.
- **Performance:** As demand for higher data rates and performance increasing in wireless communication, a fast and reliable communication protocol between the baseband processor and RF front-end components became essential. The RFFE protocol provides quick adjustments to RF parameters, maintaining optimal performance.
- **Interoperability:** The RFFE protocol was developed as a standard by the MIPI Alliance. By adhering to a common protocol, manufacturers for RF front-end devices. Hence, we can ensure their components are compatible with other RFFE compliant devices, promoting interoperability.

Hence, RFFE interface addresses key challenges in modern wireless communication systems by providing a standardized, efficient, and interoperable interface for RF front-end components. It simplifies integration, reduces space constraints, enhances power efficiency, and ensures optimal performance.

## 7. Basic Structure of RFFE Protocol Commands

The command sequence mainly divided into four parts irrespective of any command modes.

- a. Sequence Start Condition (SSC).
- b. RFFE Frames such as Command Frame, Address Frame and Data frame depending on different command frame type.
- c. No Response Frame
- d. Bus Park Cycle.

### 1. Sequence Start Condition

The sequence start condition is a unique state identified by a rising edge of SDATA followed by a falling edge of SDATA, with SCLK remaining at logic '0'. The Master generates this condition by setting the signal to logic level '1' for one SCLK period and then to logic level '0' for the next SCLK period, while SCLK stays constant at logic level '0', as illustrated in the figure below.

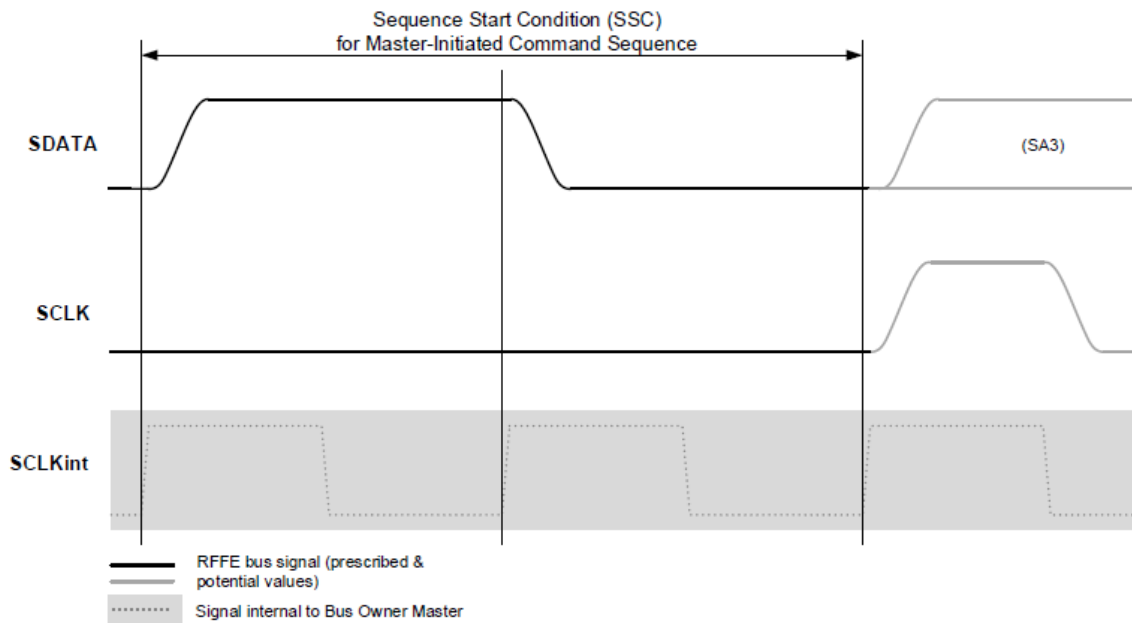


Figure 2: RFFE Sequence Start Condition

## 2. RFFE Frames

RFFE Frames are the basic building blocks for the frame fields in all RFFE command sequences. There are three basic types of frames in RFFE.

### 1. Command Frame

The command frame includes a four-bit Slave Address (SA), eight command bits, and a single parity bit, totaling 13 bits. The eight command bits can represent a command code, an address, or data, depending on the command sequence, which will be detailed in subsequent sections. The structure of the Command Frame is illustrated in the figure below.

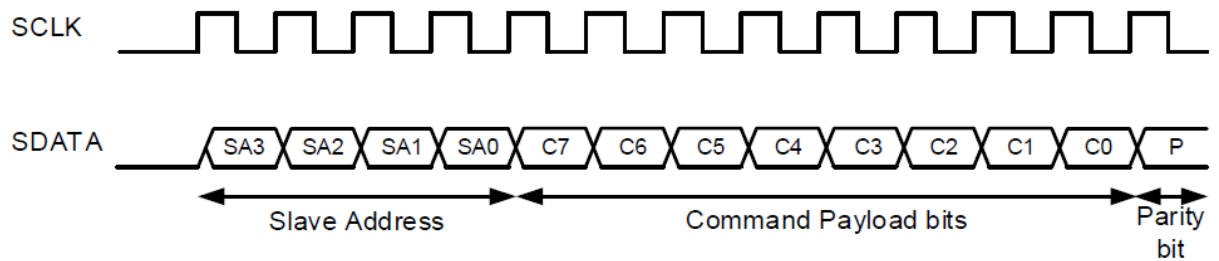


Figure 3: RFFE Command Frame

### 2. Data and Address Frames

A Data Frame (DF) and an Address Frame (AF) each consist of eight data or address bits along with a single parity bit. The structure of the Data or Address frame is illustrated in the figure below.

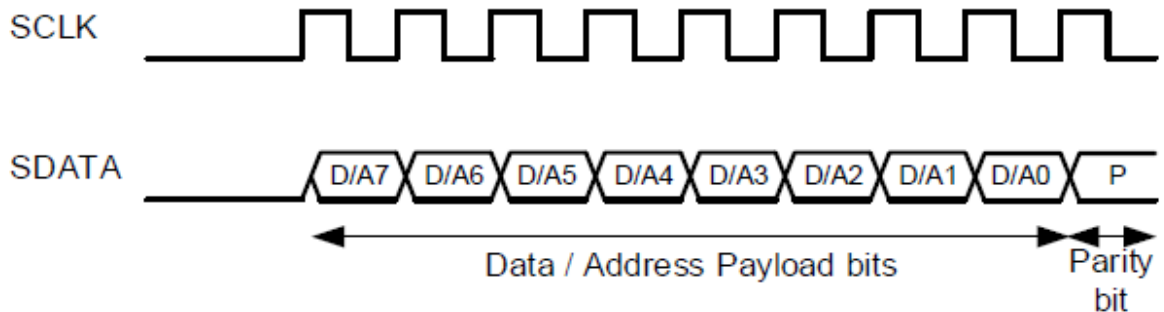


Figure 4: RFFE Data or Address Frame

### 3. No Response Frame

The No Response Frame (NRF) involves setting the SDATA pin to logic '0' for all bits, including the parity bit. As a variant of the Data Frame (DF), the NRF frame is 9 bits long. It can be driven intentionally by the Master or by the SDATA pull-down when the Master sends a read operation to a write-only slave device. The structure of the No Response Frame is illustrated in the figure below. This frame is rarely used in RFFE transactions.

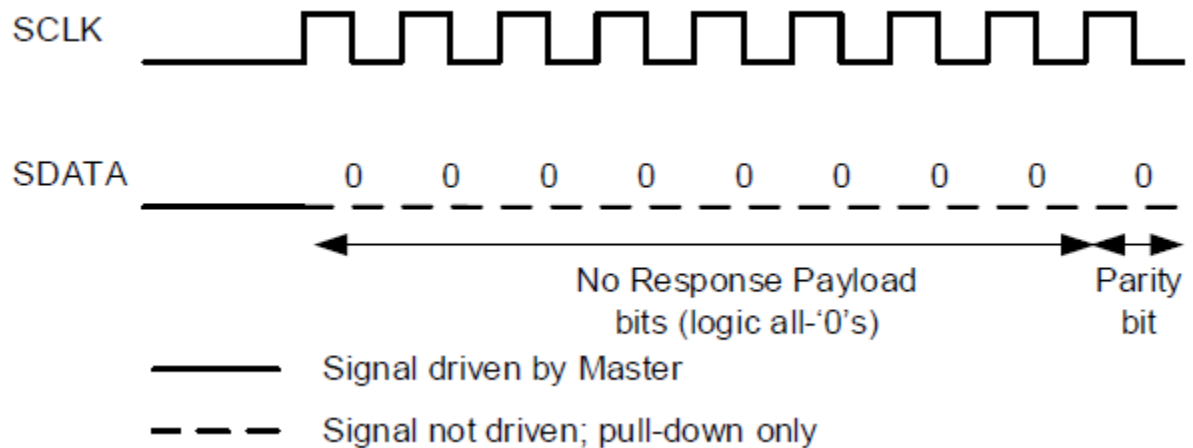


Figure 5: RFFE No Response Frame

### 4. Bus Park Cycle

The Bus Park Cycle (BPC) is initiated by the Master or Slave device that controls the SDATA line at the end of a command sequence. It can also occur when a device transfers control of the SDATA line to another device within a command sequence, such as during a read transaction.

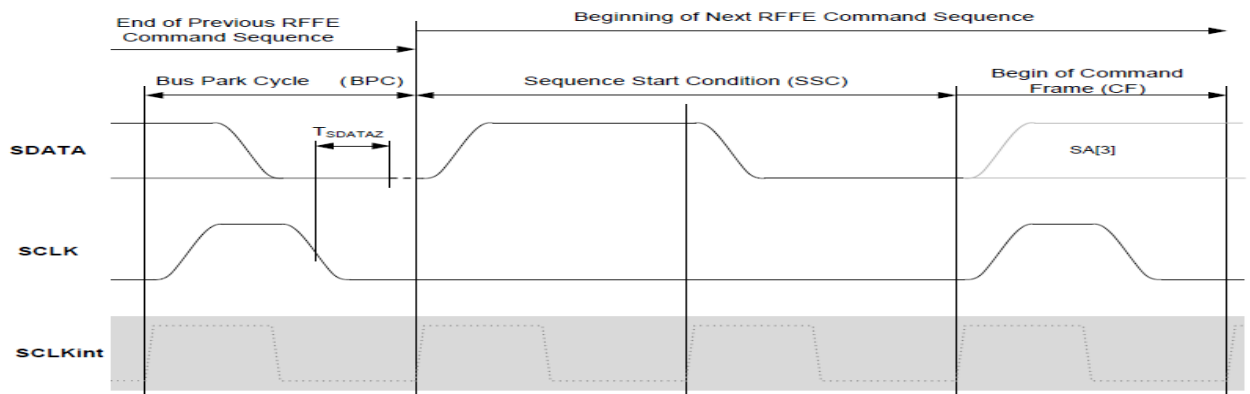


Figure 6: RFFE Bus Park Cycle

## 8. Power Saving Technique (Clock Gating)

In the fast-growing VLSI industry, there is a need for power reduction without sacrificing performance. Especially in wireless communication systems, the end products are mostly battery-operated, such as smartphones and laptops, where power consumption is a crucial factor. In this dissertation, we are implementing RTL clock gating, which is one of the main power-saving techniques since the clock accounts for most of the power dissipation.

In synchronous design, the system clock is connected to the clock pin of every flip-flop within the design. This configuration leads to three primary sources of power consumption:

1. Power used by combinatorial logic, which changes values with each clock edge.
2. Power used by flip-flops.
3. Power used by the clock buffer tree in the design.

Among these three components, the power consumption by combinatorial logic is the least significant. Conversely, the power consumption by flip-flops and the clock buffer tree is substantial. In this context, RTL clock gating plays a major role in reducing the power consumption due to flip-flops and the clock buffer tree.

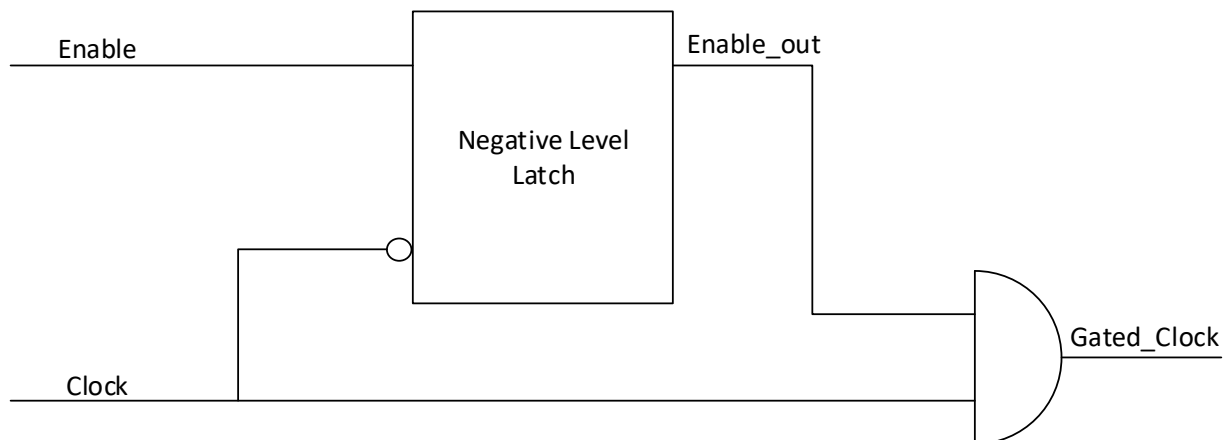
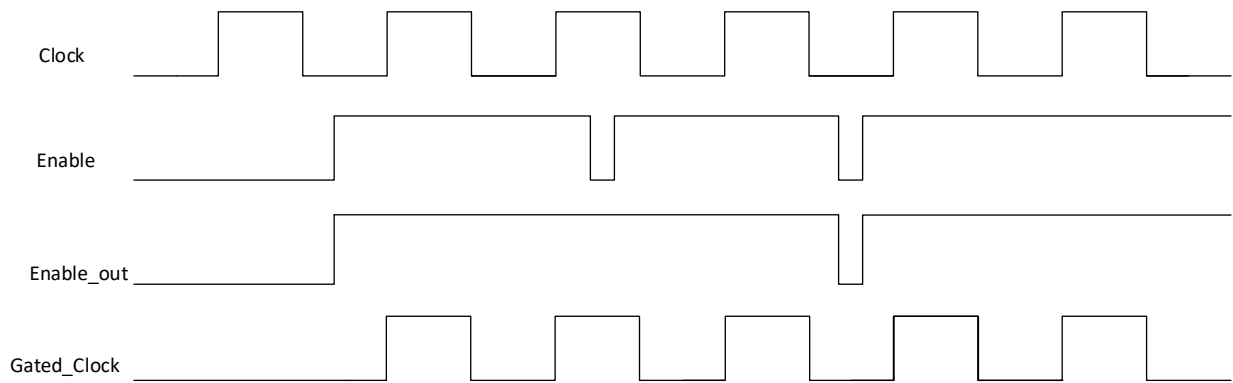


Figure 7: Latch Based Clock Gating Circuit

As illustrated in the figure, the latch-based clock gating circuit includes a negative level-sensitive. This latch retains the enable signal from the active edge of the clock until the inactive edge arrives, effectively eliminating glitches at the output of the AND gate.



*Figure 8: Timing Diagram of Latch Based Clock Gating Circuit*

From above timing diagram we can see that even though Enable signal toggles when input clock is HIGH, the output of Latch Enable\_out signal will not toggle thereby providing glitch-free clock (Gated\_Clock) at the output of AND gate.

## 9. Description of RFFE Command Sequences

RFFE Command sequences are mainly divided into 4 types based on data, address and byte count requirements such as Register 0 Write command sequence (REG0), Register Write/Read command sequence (REG\_WR/REG\_RD), Extended Register Write/Read Command Sequence (EXT\_REG\_WR/EXT\_REG\_RD), and Extended Register Write/Read Long Command Sequence (EXT\_REG\_WR\_LNG/EXT\_REG\_RD\_LNG).

### 1. Register 0 Write Command Format

Register 0 write command is the simplest command in RFFE protocol. It is the shortest and fastest to execute since it contains only Command frame and no Address/Data frames. As you can see from below figure where command always starts with SSC and command frame consists of slave address and 7 bits of data. This command sequence is primarily used to support RF Front-End devices (slaves) that may need repetitive command sequences to a fixed slave register while minimizing bandwidth consumption.

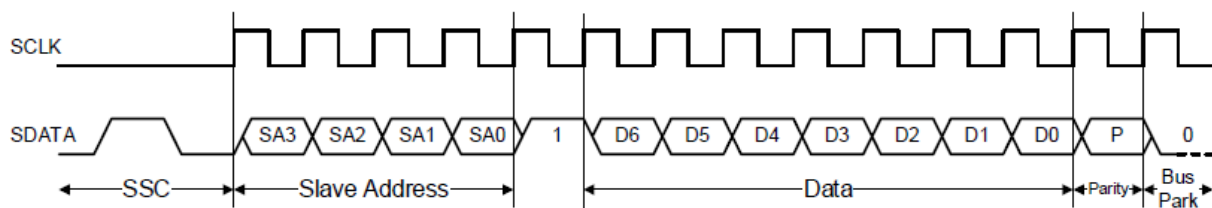


Figure 9: Register 0 Write Command Format

## 2. Register Write Command Format

Register write is one of the basic command sequences available in RFFE protocol. This command sequence is compact and provides user selectable addressing with address space of slave device from 0x00 – 0x1F (31) (5 bits of address support as shown in below figure). This is the shortest command sequence in RFFE that allows variable addressing with only a command frame and one data frame.

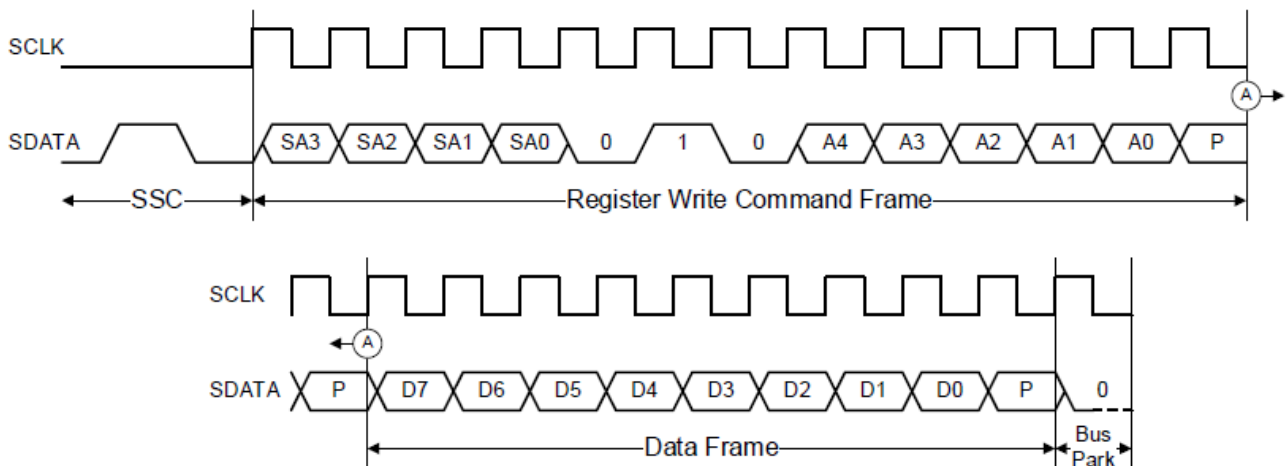


Figure 10: Register Write Command Format



### 3. Extended Register Write Command Format

The Extended Register Write command format allows write access to an extended address space of 8 bits, ranging from 0x00 to 0xFF (0-255) on a slave device. It also enables writing from one to sixteen bytes in a single command sequence. The four least significant bits (LSBs) of the command frame define the byte count (BC [3:0]), where 0x0 indicates one byte will be sent, and 0xF indicates sixteen bytes will be written to the slave device. As shown in the figure below, the Extended Register Command includes a separate address frame, and the byte count is part of the command frame itself. From the slave's perspective, if the incoming command sequence contains more than one byte, the address will automatically increment by one for each byte sent from the Master, without "wrap around" if the address reaches 0xFF. If the extended register address reaches 0xFF before the final data frame in the command sequence, the register at address 0xFF will be overwritten with the overflow data multiple times.

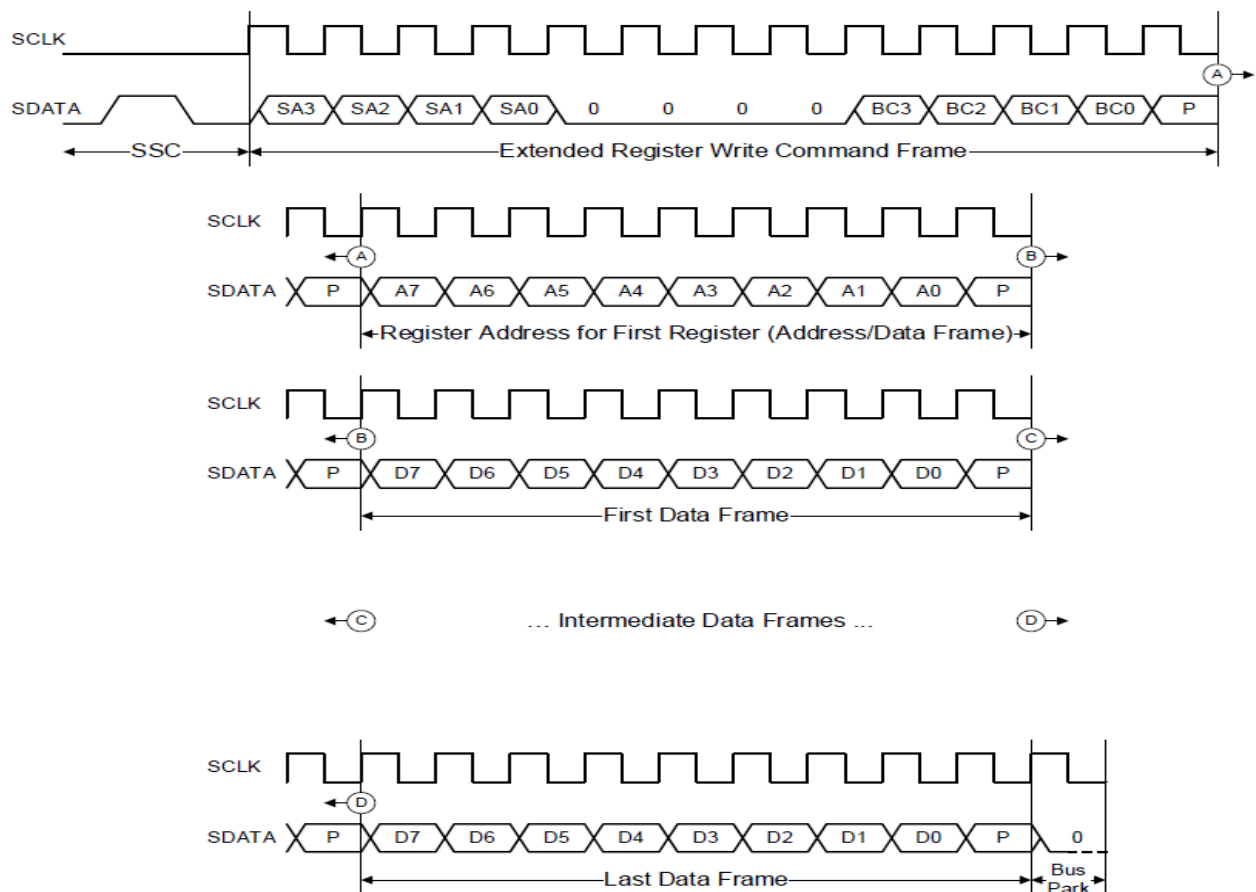


Figure 11: Extended Register Write Command Format

## 4. Extended Register Write Long Command Format

The Extended Register Write Long (EXT\_REG\_WR\_LNG) command sequence allows write access to a 16-bit extended address space, ranging from 0x0000 to 0xFFFF (0-65535) on a slave device. It supports writing from one to eight bytes in a single command sequence. The three least significant bits (LSBs) of the command frame define the byte count (BC [2:0]), where 3'b000 (decimal 0) indicates one byte will be sent, and 3'b111 (decimal 7) indicates eight bytes will be written to the slave device. As shown in the figure below, the Extended Register Long Command includes a separate address frame, with the byte count included in the command frame itself. From the slave's perspective, if the incoming command sequence contains more than one byte, the address will automatically increment by one for each byte sent from the Master, without "wrap around" if the address reaches 0xFFFF. If the extended register address reaches 0xFFFF before the final data frame in the command sequence, the register at address 0xFFFF will be overwritten with the overflow data multiple times.

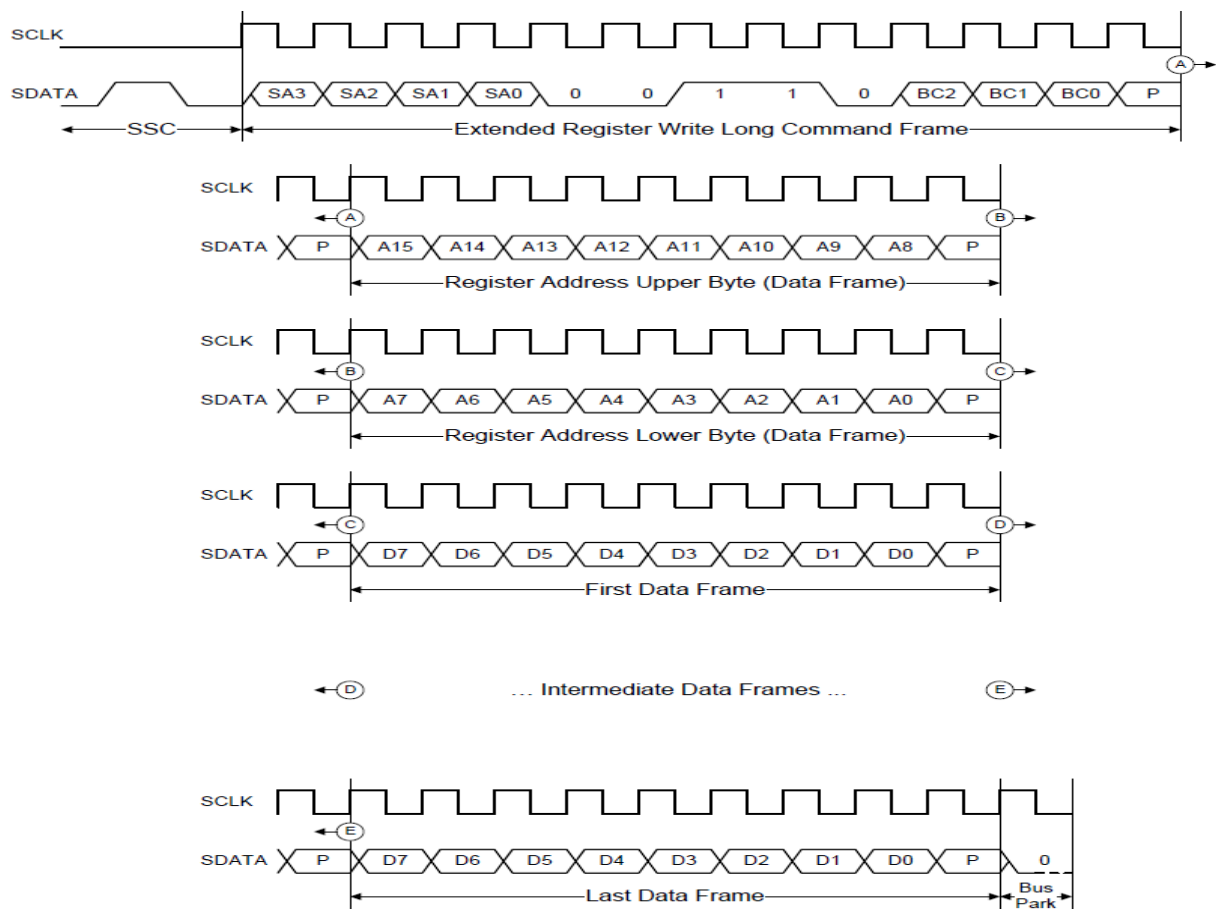


Figure 12: Extended Register Write Command Format

## 5. Register Read Command Format

Register read is one of the basic command sequences available in RFFE protocol. This command sequence is compact and provides user selectable addressing with address space of slave device from 0x00 – 0x1F (31) (5 bits of address support as shown in below figure). This is the shortest command sequence in RFFE that allows variable addressing with only a command frame and one data frame. Data frame which contains one byte of data which is read from slave at the specified address. Unlike the register write command sequence, this sequence starts with an SSC, followed by the read command frame and a master-initiated one clock cycle bus park cycle. After this BPC, the slave provides a data frame, and the sequence ends with a slave-initiated bus park cycle.

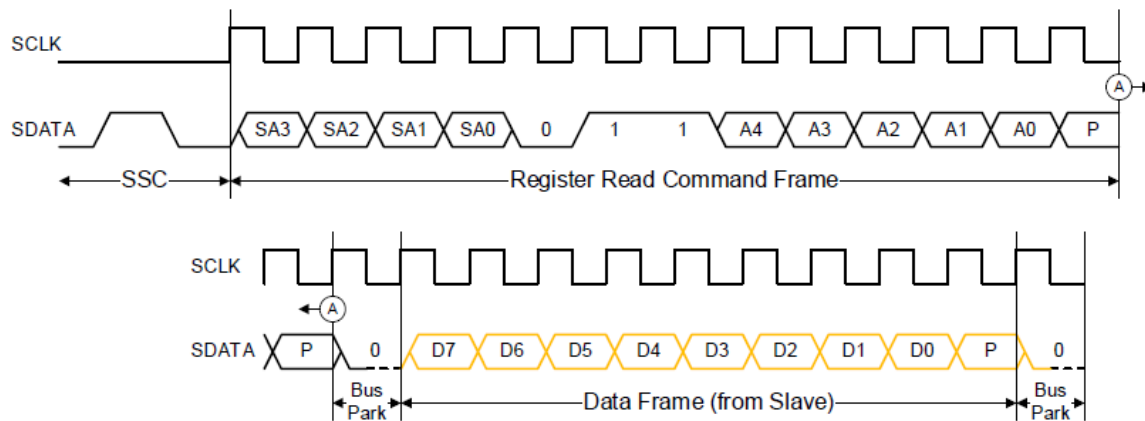


Figure 13: Register Read Command Format

## 6. Extended Register Read Command Format

The Extended Register Read command sequence allows read access to an extended address space of 8 bits, ranging from 0x00 to 0xFF (0-255) on a slave device. It also enables reading from one to sixteen bytes in a single command sequence. The four least significant bits (LSBs) of the command frame define the byte count (BC [3:0]), where 0b0000 indicates one byte will be read, and 0b1111 indicates sixteen bytes will be read from the slave device. As shown in the figure below, the Extended Register Command includes a separate address frame, and the byte count is part of the command frame itself.

The register address in the command sequence specifies the initial extended register to be read. If multiple bytes are read in a single command sequence, the slave's local extended register address automatically increments by one for each byte read, up to address 0xFF, starting from the address given in the address frame. If the extended register address reaches 0xFF before the final data frame in the command sequence, the register at address 0xFF is read repeatedly. If the master initiates an extended register read command sequence to an unsupported slave extended register address, the slave responds with a no response frame. Similarly, if the auto-incremented address points to an unsupported register, the slave sends a no response frame instead of a data frame. The command sequence then continues from the next extended register address.

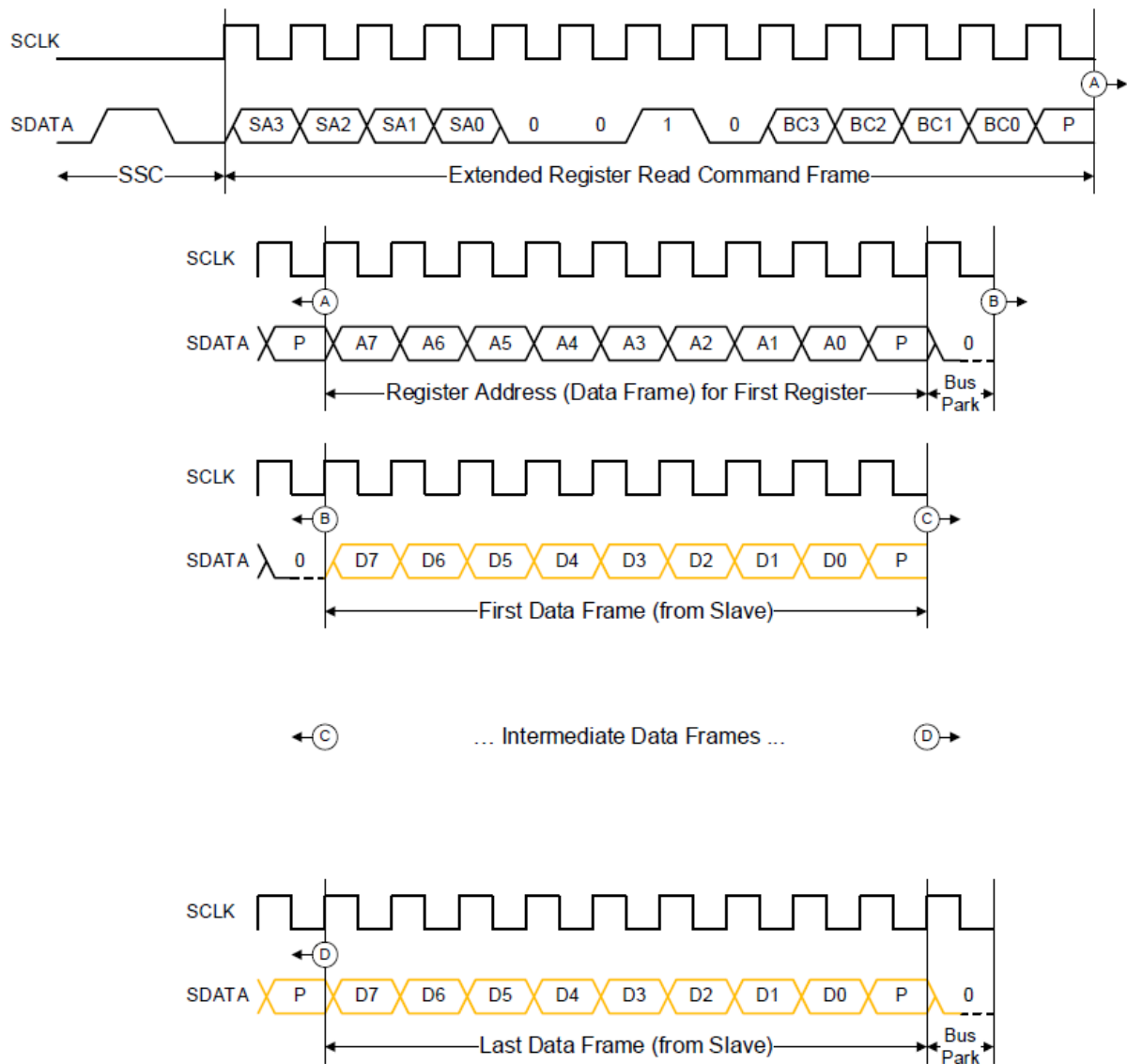


Figure 14: Extended Register Read Command Format

## 7. Extended Register Read Long Command Format

The Extended Register Read Long (EXT\_REG\_RD\_LNG) command sequence allows read access to a 16-bit extended address space, ranging from 0x0000 to 0xFFFF (0-65535) on a slave device. It supports reading from one to eight bytes in a single command sequence. The three least significant bits (LSBs) of the command frame define the byte count (BC [2:0]), where 0b000 indicates one byte will be read, and 0b111 indicates eight bytes will be read from the slave device. As shown in the figure below, the Extended Register Long Command includes a separate address frame, with the byte count included in the command frame itself.

The register address in the command sequence identifies the initial extended register to be read. If multiple bytes are read within a single command sequence, the slave's local extended register address automatically increments by one for each byte, up to address 0xFFFF, starting from the address specified in the address frame. If the extended register address reaches 0xFFFF before the final data frame in the command sequence, the register at address 0xFFFF is read repeatedly. If the master issues an extended register read long command sequence to an unsupported slave extended register address, the slave responds with a no response frame. Similarly, if the auto-incremented address points to an unsupported register, the slave sends a no response frame instead of a data frame. The command sequence then continues from the next extended register address.

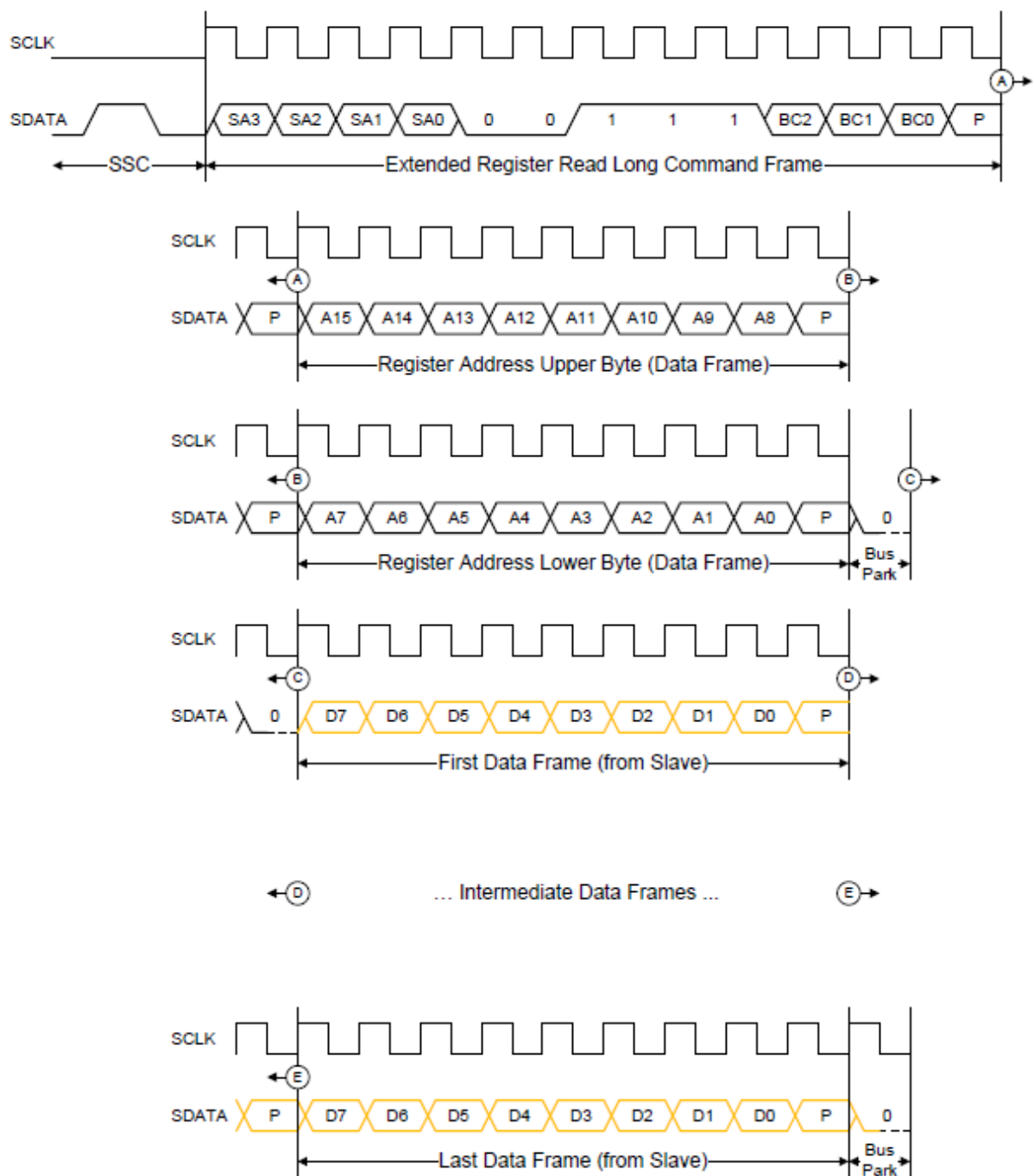


Figure 15: Extended Register Read Long Command Format

## 10. RFFE Interface IP Controller Architecture

The RFFE Interface IP controller is primarily divided into two modules: the RFFE Master Controller and the RFFE Slave Controller, as shown in the figure below.

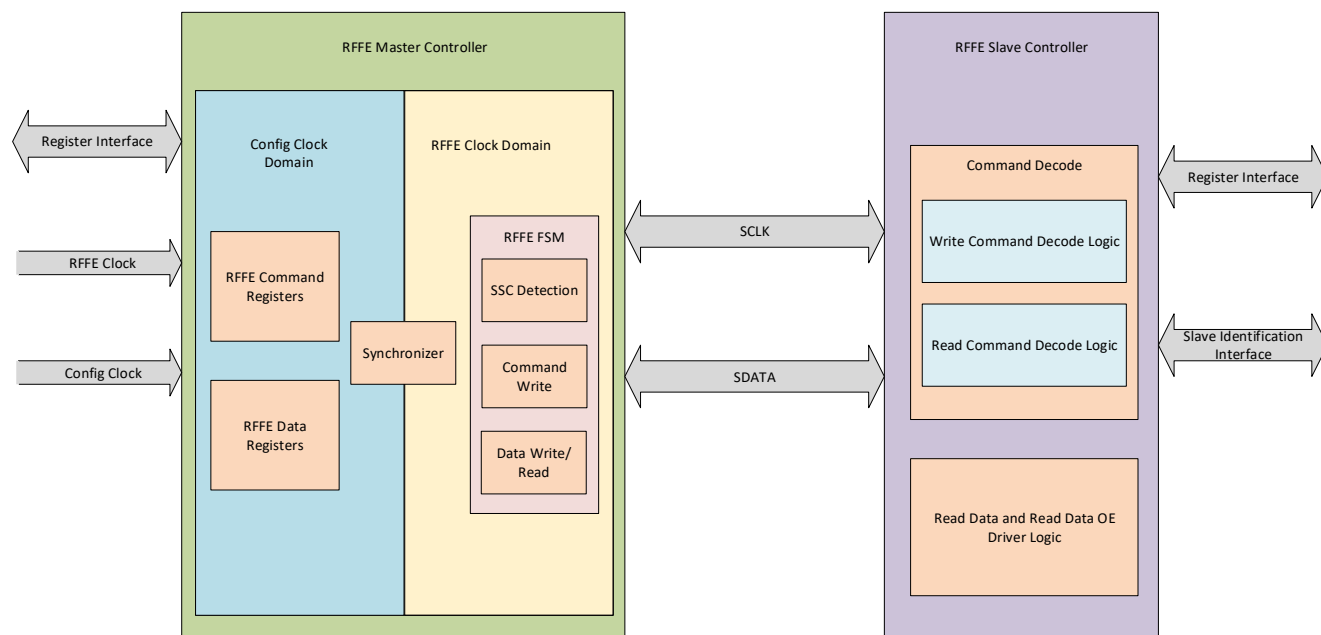


Figure 16: RFFE Interface Ip Controller

### 1. RFFE Master Controller

The RFFE Master controller receives a configuration clock to configure the registers and an RFFE clock for transactions on the SDATA and SCLK pads. It also has a register interface where an external module or software core can configure the registers to initiate transactions. Once a transaction is initiated via the register interface, the RFFE FSM triggers in the RFFE clock domain, which in turn drives the SCLK and SDATA pads. This process will be explained in detail in further sections.

The RFFE Master Controller is broadly divided into two domains based on the operating clocks, which are separated by a synchronizer. The synchronizer is primarily used for synchronizing the data signals and control signals from the configuration clock domain to the RFFE clock domain and vice versa.



### **a. Config Clock Domain**

As the name suggests, it is primarily used for configuring the RFFE registers to initiate transactions. The RFFE registers are mainly divided into command and data registers. The command registers are responsible for fields such as command type, address field, number of bytes to be sent, and mode type. The command types include Register 0 command (write only), Normal Register command, Extended Register command, and Extended Register Long command. The mode types are either write mode or read mode.

### **b. RFFE Clock Domain**

This domain mainly consists of a Finite State Machine (FSM), which gets triggered based on the incoming command. As shown in the figure below, once the incoming command from the config clock domain is detected, the FSM triggers and transitions from the IDLE state to the SSC\_HIGH state, generating logic '1' for one clock cycle. It then unconditionally transitions from SSC\_HIGH to SSC\_LOW, generating logic '0' for one clock cycle, as explained in the previous section (RFFE Frames). Once SSC generation is complete, the FSM unconditionally moves to the Command Phase based on the command type selected in the config register.

**Command Phase:** The state machine remains in this state for 13 clock cycles. The first 4 clocks are fixed for the Slave ID, and the remaining 9 bits are either Data (Register 0 command), command code and address (Register command), or command code and byte count (Extended Register or Extended Register Long command).

**Address Phase:** The state machine stays in this state for either 9 or 18 clock cycles, depending on whether it is an Extended Register or Extended Register Long command, respectively.

**Data Phase:** The state machine remains in this state for 9 to 144 clock cycles, based on the byte count in the case of Extended Register or Extended Register Long commands, respectively.

**Bus Park Phase:** The state machine stays in this state for 1 clock cycle, which occurs only during a read transaction when the bus is handed over from the Master to the Slave. Detailed information about the Bus Park Cycle is provided in the previous section.

**Done Phase:** The state machine stays in this state for 1 clock cycle, signifying the end of the transaction

### **c. Synchronizer**

The synchronizer is primarily used for synchronizing data signals and control signals between the configuration clock domain and the RFFE clock domain. It is crucial for ensuring reliable communication between different clock domains or between asynchronous signals and a synchronous system.

Synchronizers are mainly needed for:

- Preventing metastability, where the signal becomes unpredictable when crossing between two asynchronous clock domains.
- Ensuring reliable data transfer by preventing data corruption in the case of multi-bit bus crossing between two asynchronous clock domains.
- Providing system stability and reliability by reducing the chances of metastability.

In our design, we mainly use three types of synchronizers:

- Basic two flip-flop synchronizers to synchronize semi-static control signals between two asynchronous clock domains.
- Pulse synchronizers to synchronize pulse signals between clock domains without missing pulses, which can happen when a pulse signal crosses from a high-frequency to a low-frequency domain. This can be achieved using different types of handshaking mechanism synchronizers.
- Asynchronous FIFO to synchronize multi-bit data transfer between two asynchronous clock domains (binary to gray conversion technique).

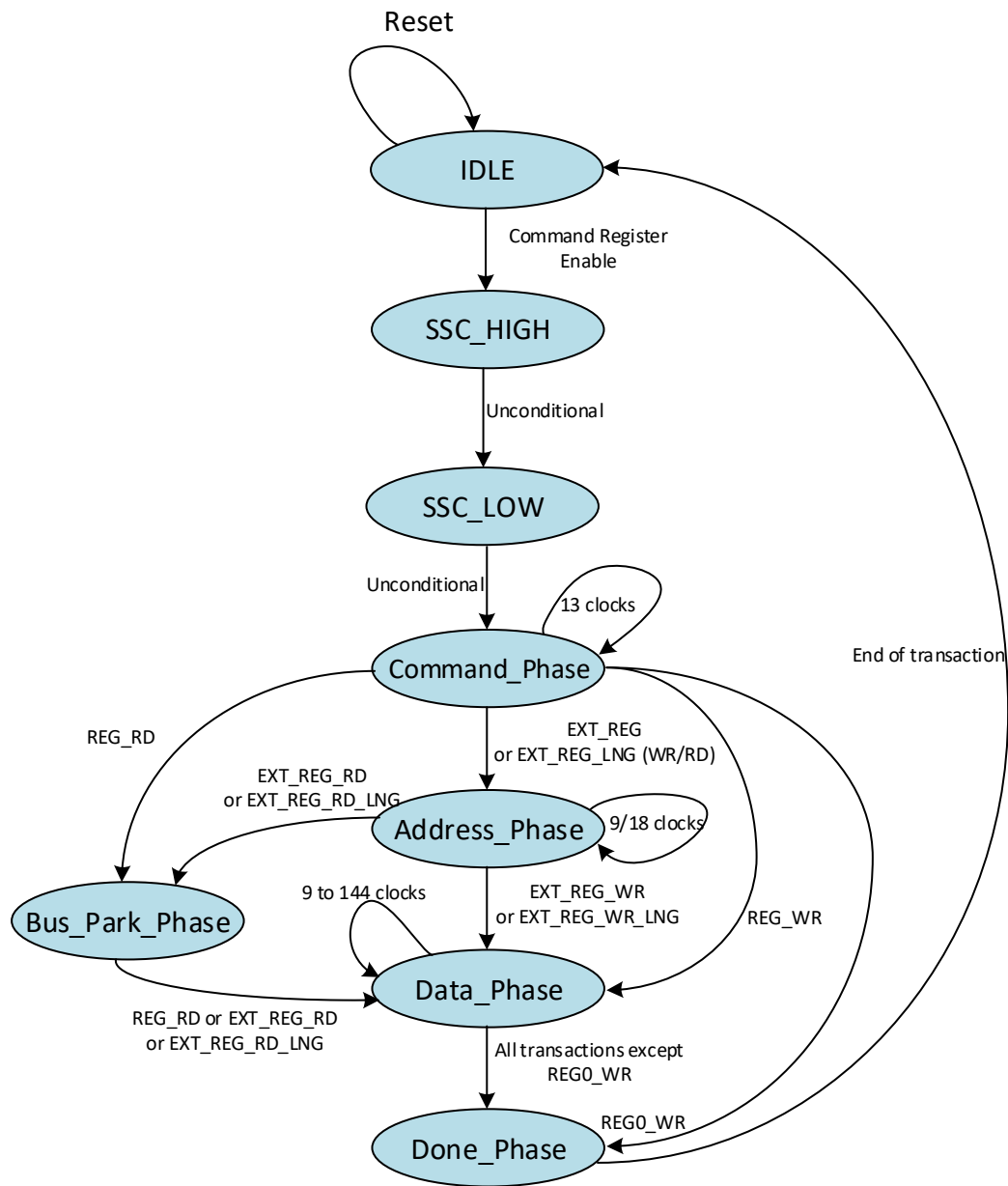


Figure 17: RFFE Master FSM

## 2. RFFE Slave Controller

The RFFE Slave controller is a simpler block compared to the RFFE Master controller. It mainly consists of two blocks: the command decode block and the read data driver logic. RFFE Slaves are usually RF Front-End devices that require simpler logic to decode incoming RFFE transactions from the Master and send back data for read transactions.

### 1. Command Decode:

This block is responsible for decoding the incoming transaction from the RFFE Master upon receiving SSC pulses. As shown in the figure below, the decode logic is mainly divided into four frames: the command frame, Address MSB frame, Address LSB frame, and Data frame.

**Command Frame:** This is one of the main frames, which decodes the slave it is addressed to base on the Slave ID (first 4 bits) and determines the nature of the command based on the remaining command code (9 bits). The specifics of each command sequence will be explained in detail in previous section.

**Address MSB Frame:** For the extended register long command, the address consists of 16 bits. Therefore, based on the command code in the command frame, we will extract the MSB (most significant byte) 8 bits of the address.

**Address LSB Frame:** This frame is used to extract the LSB (least significant byte) 8 bits of the address for either the extended register or extended register long command.

**Data Frame:** This frame is used to either extract the data bytes sent from the Master during a write transaction or to send data bytes to the Master during a read transaction. The command frame decodes whether the incoming transaction is a write or read operation based on the command code sent from the Master.

## 2. Read Data or OE driver logic:

Based on the command decode logic block, we can determine whether the incoming transaction is a write or read transaction. This block is enabled only for read transactions. OE or Output Enable, is an enable pin for the external IO pad, indicating that the slave will drive the IO pad during a read transaction. The number of bytes of read data to be sent to the Master is determined by the byte count field mentioned in the command frame for extended register read or extended register read long commands, and only one byte for a register read command.

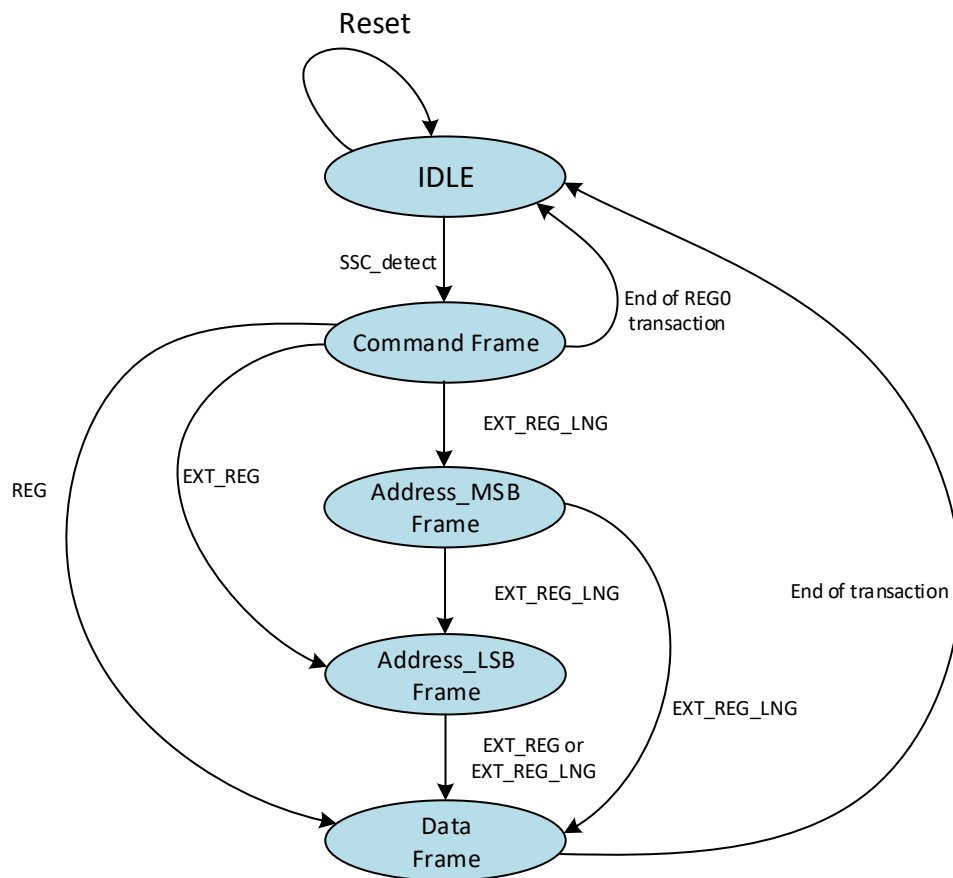


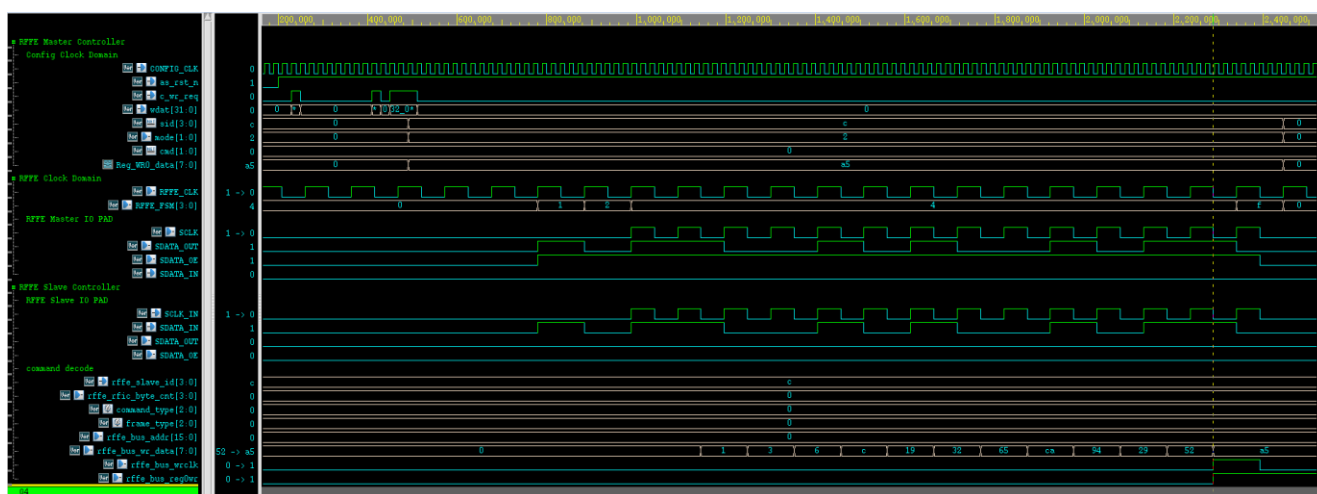
Figure 18: RFFE Slave Command decode logic

# 11. Observations and Simulations

## 1. Register 0 Write Command Format

The waveform below illustrates the configuration of the RFFE Master controller. It is set with a command register slave ID of 4'hC, mode as RFFE Write (2), command as REG0\_WR (0) (register 0 write), and Command Data as 8'hA5. This incoming command and data activate the RFFE FSM in the RFFE Clock domain. The FSM sequence for the Register 0 command is as follows: IDLE (0) → SSC\_HIGH (1) → SSC\_LOW (2) → Command\_phase (4) → Done\_phase (F) → IDLE (0). As shown in the waveform, the REG0 transaction contains only the command frame. Based on the FSM sequence, the corresponding clock (SCLK) and data pad (SDATA\_OUT) signals are sent to the slave. The oe pin, which is the output enable, is used as the enable pin for the IO pad logic at the chip level.

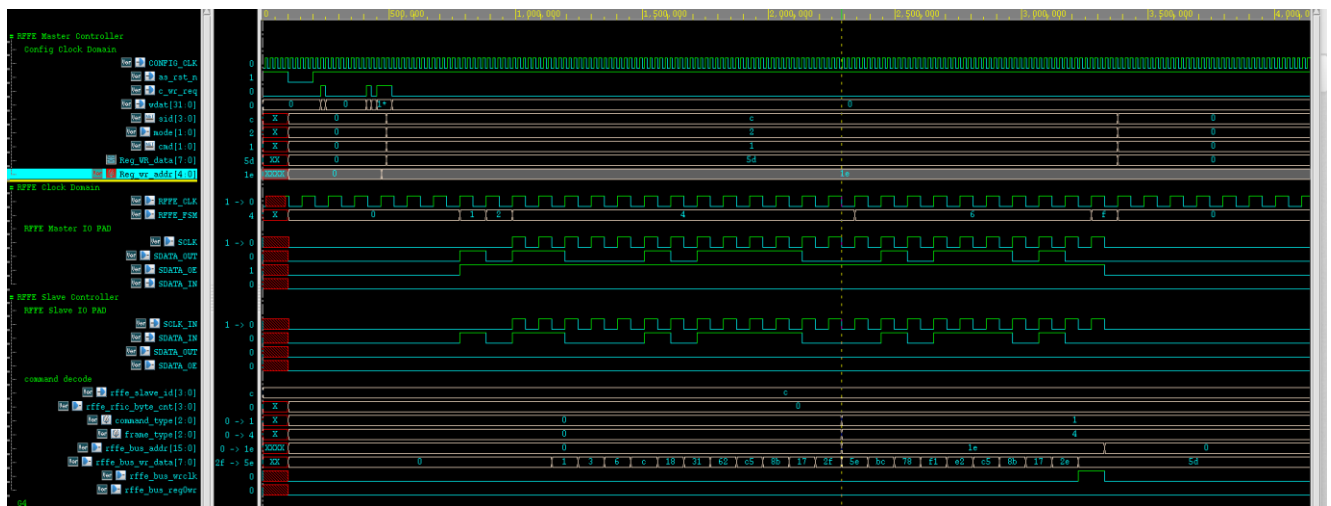
When the RFFE Slave controller detects SSC, it decodes the initial slave ID bits from the RFFE Master and compares them with its own slave ID (**rffe\_slave\_id**). If they match, the slave controller begins decoding the command. As shown in the waveform, the slave decodes the command and asserts the **rffe\_bus\_reg0wr** signal to logic 1, indicating a Register 0 command with **rffe\_bus\_wr\_data[7:0] = 8'hA5**, matching the data configured by the Master.



## 2. Register Write Command Sequence

The waveform below shows the configuration of the RFFE Master controller. It is set with a command register slave ID of 4'hC, mode as RFFE Write (2), command as REG\_WR (1) (register write), Address as 5'h1E, and Command Data as 8'h5D. This incoming command and data activate the RFFE FSM in the RFFE Clock domain. The FSM sequence for the Register Write command is as follows: IDLE (0) → SSC\_HIGH (1) → SSC\_LOW (2) → Command\_phase (4) → Data\_phase (6) → Done\_phase (F) → IDLE (0). As shown in the waveform, the REG\_WR transaction contains both the command frame and a one-byte data frame. Based on the FSM sequence, the corresponding clock (SCLK) and data pad (SDATA\_OUT) signals are sent to the slave. The oe pin, which is the output enable, is used as the enable pin for the IO pad logic at the chip level.

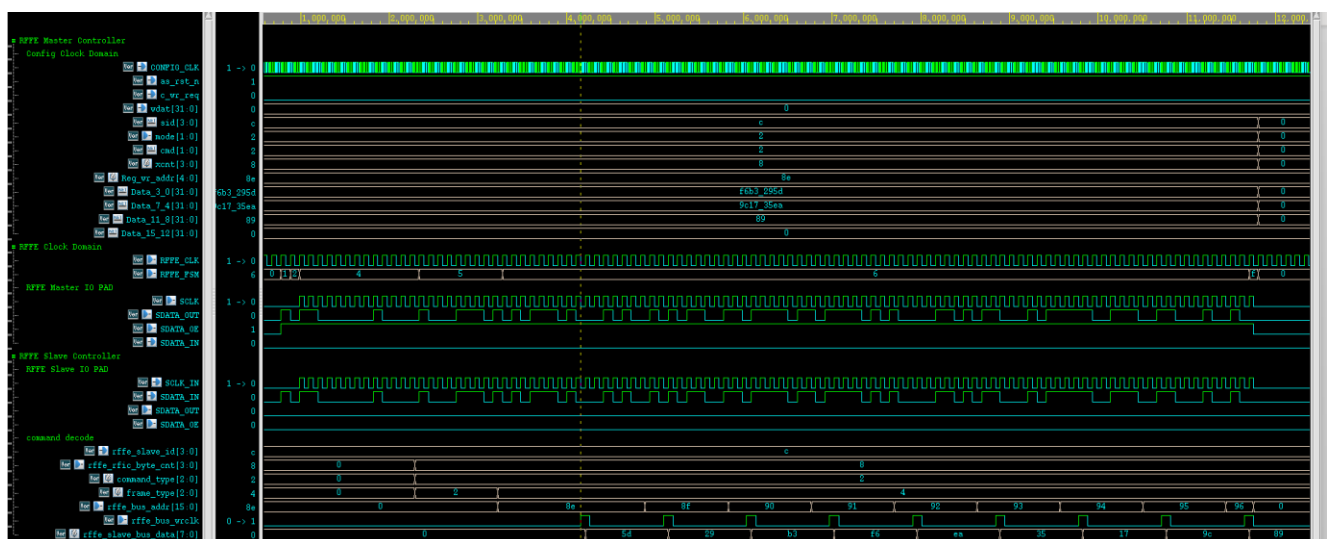
When the RFFE Slave controller detects SSC, it decodes the initial slave ID bits from the RFFE Master and compares them with its own slave ID (**rffe\_slave\_id**). If they match, the slave controller begins decoding the command. As shown in the waveform, the slave decodes the command and asserts the **rffe\_bus\_wrclk** signal to logic 1, with **rffe\_bus\_wr\_data[7:0] = 8'hA5** and **rffe\_bus\_addr[15:0] = 16'h001E**, matching the data and address sent by the Master.



### 3. Extended Register Write Command Sequence

The waveform below illustrates the configuration of the RFFE Master controller. It is set with a command register slave ID of 4'hC, mode as RFFE Write (2), command as EXT\_REG\_WR (2) (Extended Register Write), Address as 8'h8E, and Byte count as 8 (9 bytes). This incoming command and data activate the RFFE FSM in the RFFE Clock domain. The FSM sequence for the Register Write command is as follows: IDLE (0) → SSC\_HIGH (1) → SSC\_LOW (2) → Command\_phase (4) → Address\_phase (5) → Data\_phase (6) → Done\_phase (F) → IDLE (0). As shown in the waveform, the EXT\_REG\_WR transaction contains the command frame, a one-byte address frame, and a 9-byte data frame. Based on the FSM sequence, the corresponding clock (SCLK) and data pad (SDATA\_OUT) signals are sent to the slave. The oe pin, which is the output enable, is used as the enable pin for the IO pad logic at the chip level.

When the RFFE Slave controller detects SSC, it decodes the initial slave ID bits from the RFFE Master and compares them with its own slave ID (**rffe\_slave\_id**). If they match, the slave controller begins decoding the command. As shown in the waveform, the slave decodes the command and sets **rffe\_bus\_addr**[15:0] to 0x8E, the same as the Master-configured address, incrementing it 8 times for each byte of data sent. The **rffe\_bus\_wrclk** is asserted 9 times, signifying 9 bytes of data, with each data byte matching the Master-configured data, as shown in the waveform.

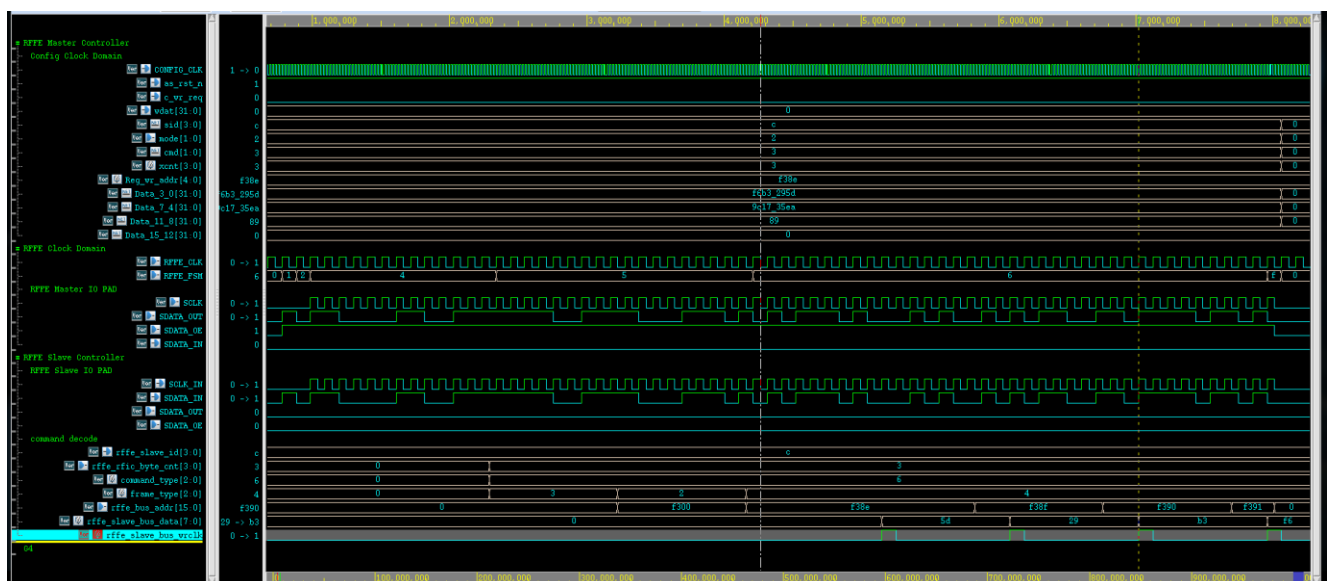




## 4. Extended Register Write Long Command Sequence

The waveform below illustrates the configuration of the RFFE Master controller. It is set with a command register slave ID of 4'hC, mode as RFFE Write (2), command as EXT\_REG\_WR\_LNG (3) (Extended Register Write), Address as 16'hF38E, and Byte count as 3 (4 bytes). This incoming command and data activate the RFFE FSM in the RFFE Clock domain. The FSM sequence for the Register Write command is as follows: IDLE (0) → SSC\_HIGH (1) → SSC\_LOW (2) → Command\_phase (4) → Address\_phase (5) → Data\_phase (6) → Done\_phase (F) → IDLE (0). As shown in the waveform, the EXT\_REG\_WR\_LNG transaction contains the command frame, a two-byte address frame, and a four-byte data frame. Based on the FSM sequence, the corresponding clock (SCLK) and data pad (SDATA\_OUT) signals are sent to the slave. The oe pin, which is the output enable, is used as the enable pin for the IO pad logic at the chip level.

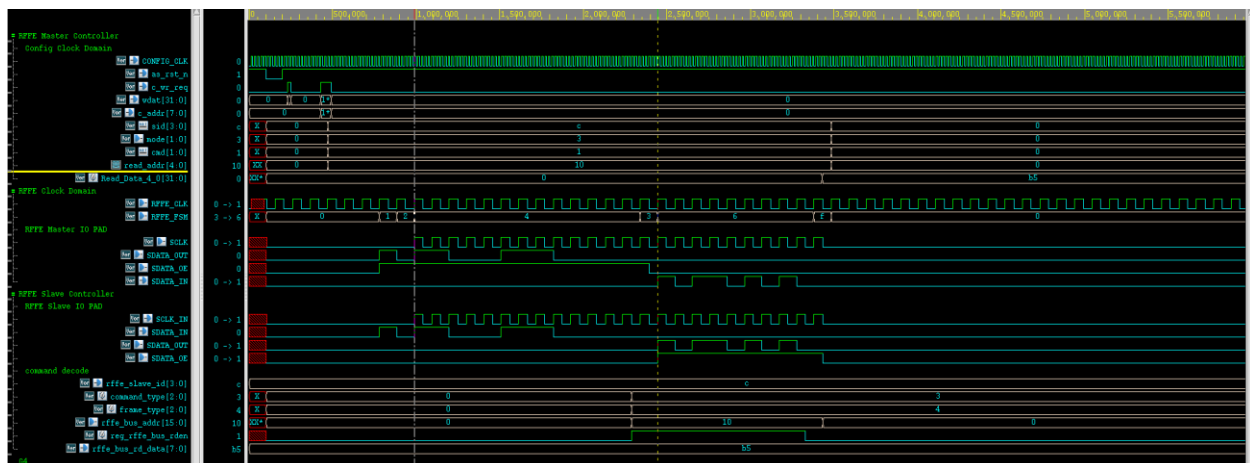
When the RFFE Slave controller detects SSC, it decodes the initial slave ID bits from the RFFE Master and compares them with its own slave ID (**rffe\_slave\_id**). If they match, the slave controller begins decoding the command. As shown in the waveform, the slave decodes the command and sets **rffe\_bus\_addr[15:0]** to 0xF38E, the same as the Master-configured address, incrementing it three times for each byte of data sent. The **rffe\_slave\_bus\_wrclk** is asserted four times, signifying four bytes of data, with each data byte matching the Master-configured data, as shown in the waveform.



## 5. Register Read Command Sequence

The waveform below illustrates the configuration of the RFFE Master controller. It is set with a command register slave ID of 4'hC, mode as RFFE Read (3), command as REG\_RD (1) (Register Read), Address as 5'h10. This incoming command and data activate the RFFE FSM in the RFFE Clock domain. The FSM sequence for the Register Read command is as follows: IDLE (0) → SSC\_HIGH (1) → SSC\_LOW (2) → Command\_phase (4) → Bus\_Park\_phase (3) → Data\_phase (6) → Done\_phase (F) → IDLE (0). As shown in the waveform, the REG\_RD transaction contains both the command frame and a one-byte data frame. Based on the FSM sequence, the corresponding clock (SCLK) and data pad (SDATA\_OUT) signals are sent to the slave. The oe pin, which is the output enable, is used as the enable pin for the IO pad logic at the chip level. For read transaction Master waits for read data from slave after Bus Park.

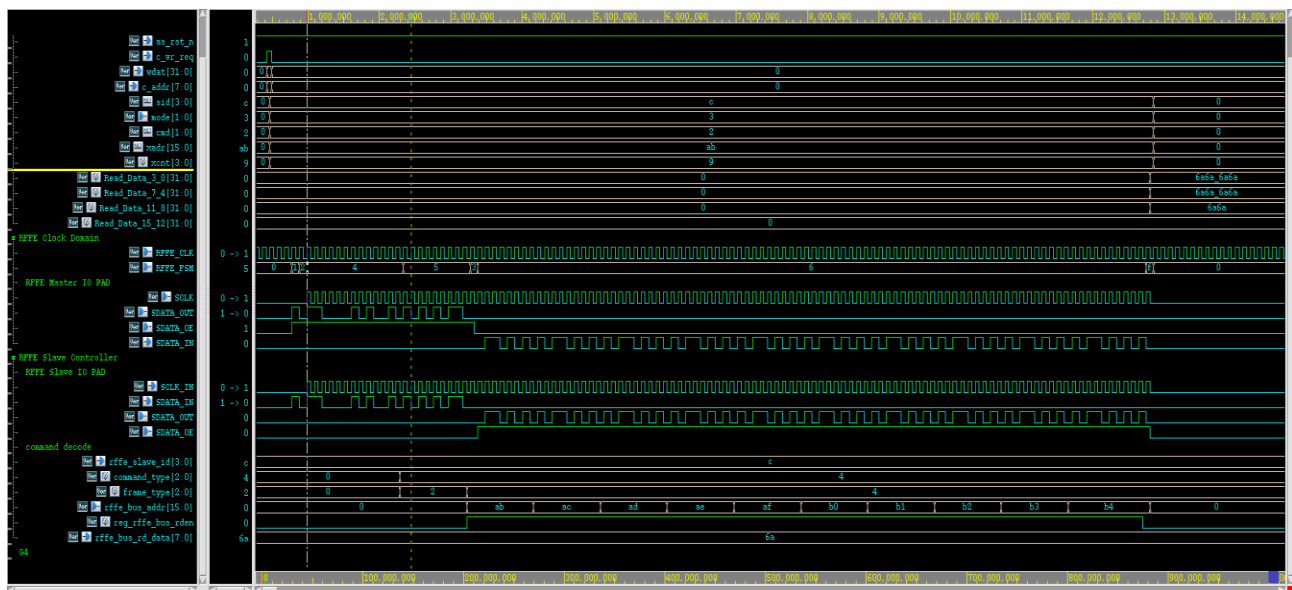
When the RFFE Slave controller detects SSC, it decodes the initial slave ID bits from the RFFE Master and compares them with its own slave ID (**rffe\_slave\_id**). If they match, the slave controller begins decoding the command. As shown in the waveform, the slave decodes the read command and sets **rffe\_bus\_addr**[15:0] to 0x10, once command frame is done after bus park, slave starts sending the data **rffe\_bus\_rd\_data**[7:0]= 8'hB5 when **reg\_rffe\_bus\_rd\_en** is asserted, which is decoded at the Master correctly as you can see in the below waveform.



## 6. Extended Register Read Command Sequence

The waveform below illustrates the configuration of the RFFE Master controller. It is set with a command register slave ID of 4'hC, mode as RFFE Read (2), command as EXT\_REG\_RD (2) (Extended Register Write), Address as 8'hAB, and Byte count as 9 (10 bytes). This incoming command and data activate the RFFE FSM in the RFFE Clock domain. The FSM sequence for the Extended Register Read command is as follows: IDLE (0) → SSC\_HIGH (1) → SSC\_LOW (2) → Command\_phase (4) → Address\_phase (5) → Bus\_Park\_Phase → Data\_phase (6) → Done\_phase (F) → IDLE (0). As shown in the waveform, the EXT\_REG\_RD transaction contains the command frame, a one-byte address frame, and a 9-byte data frame. Based on the FSM sequence, the corresponding clock (SCLK) and data pad (SDATA\_OUT) signals are sent to the slave. The oe pin, which is the output enable, is used as the enable pin for the IO pad logic at the chip level. For read transaction Master waits for read data from slave after Bus Park.

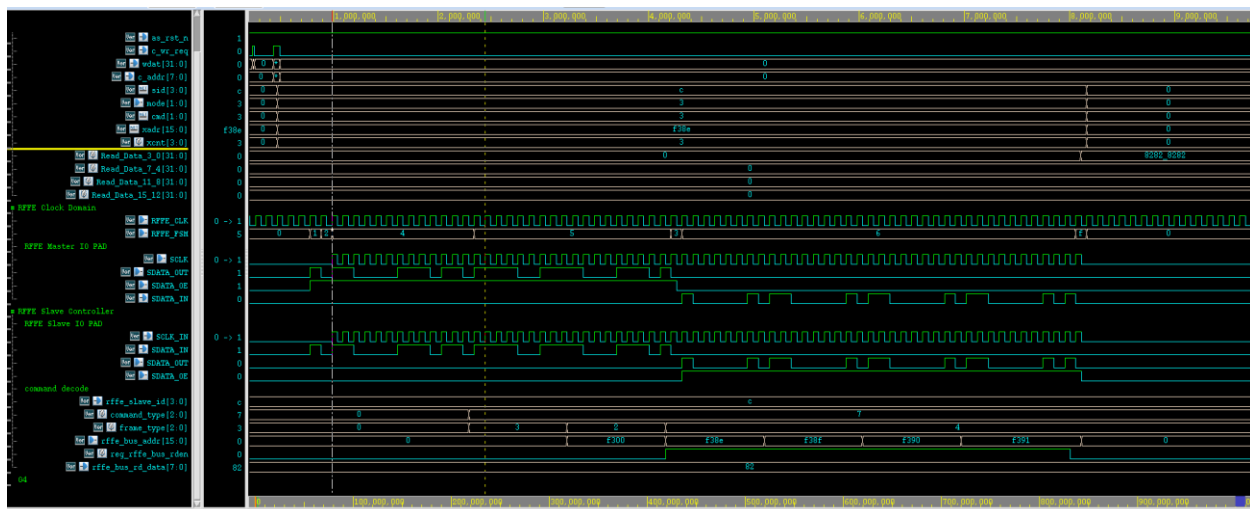
When the RFFE Slave controller detects SSC, it decodes the initial slave ID bits from the RFFE Master and compares them with its own slave ID (**rffe\_slave\_id**). If they match, the slave controller begins decoding the command. As shown in the waveform, the slave decodes the command and sets **rffe\_bus\_addr**[15:0] to 0xAB, same as the Master-configured address, incrementing it 10 times for each byte of data to be sent to Master and **reg\_rffe\_bus\_rd\_en** is asserted for 10 bytes to be sent from Slave.



## 7. Extended Register Read Long Command Sequence

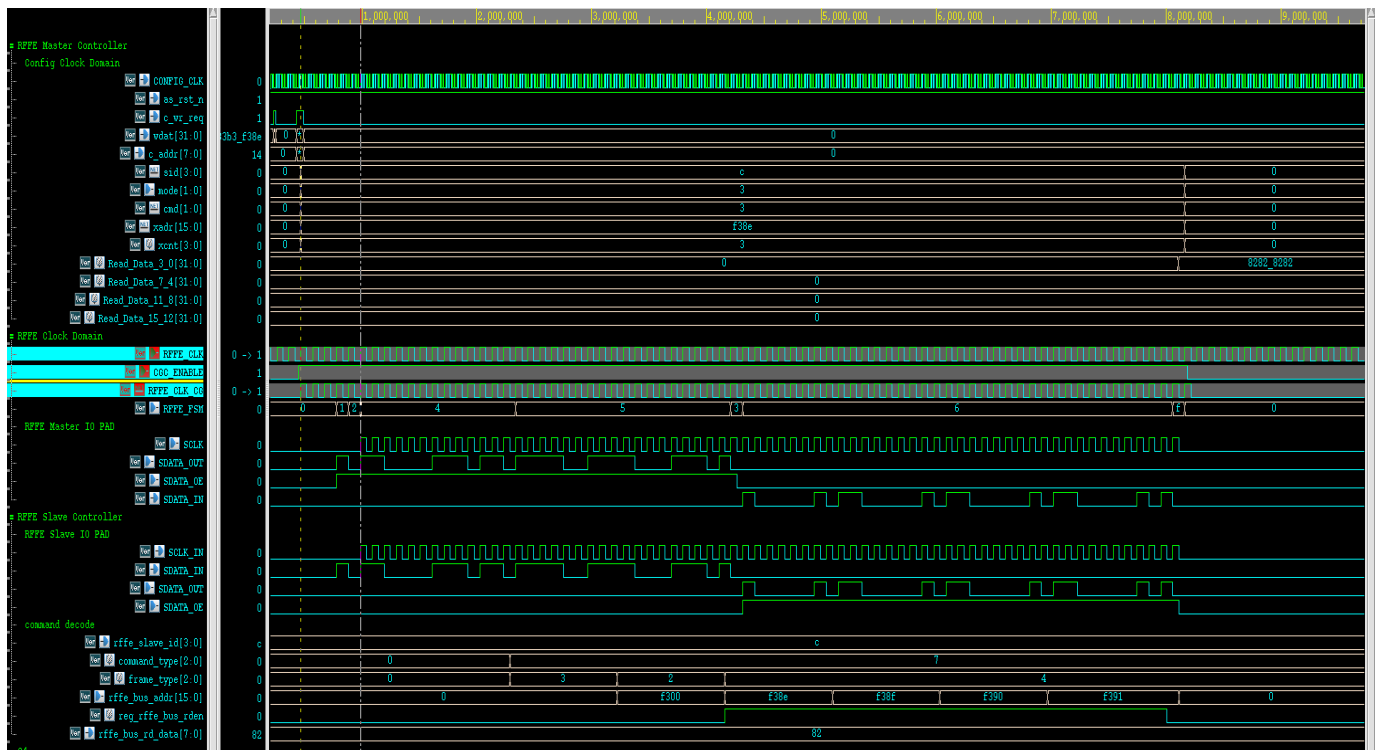
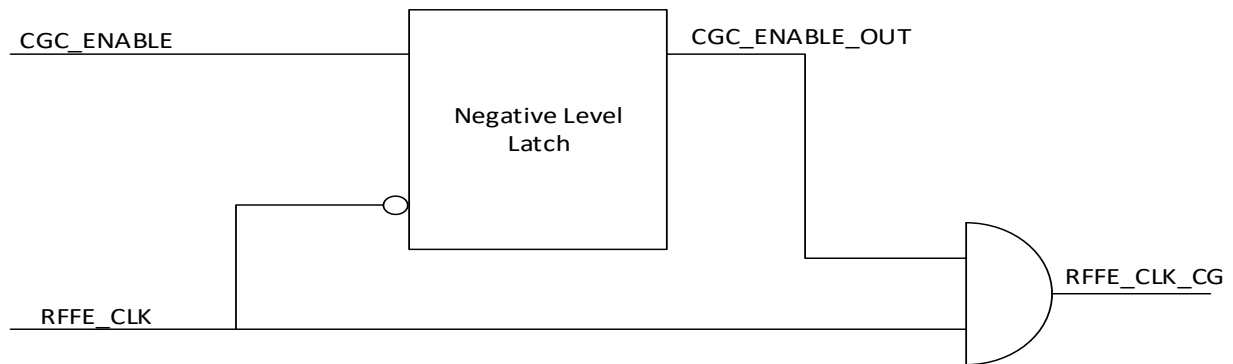
The waveform below illustrates the configuration of the RFFE Master controller. It is set with a command register slave ID of 4'hC, mode as RFFE Write (2), command as EXT\_REG\_RD\_LNG (3) (Extended Register Read), Address as 16'hF38E, and Byte count as 3 (4 bytes). This incoming command and data activate the RFFE FSM in the RFFE Clock domain. The FSM sequence for the Extended Register Read Long command is as follows: IDLE (0) → SSC\_HIGH (1) → SSC\_LOW (2) → Command\_phase (4) → Address\_phase (5) → Bus+park\_phase (3) → Data\_phase (6) → Done\_phase (F) → IDLE (0). As shown in the waveform, the EXT\_REG\_RD\_LNG transaction contains the command frame, a two-byte address frame, and a four-byte data frame. Based on the FSM sequence, the corresponding clock (SCLK) and data pad (SDATA\_OUT) signals are sent to the slave. The oe pin, which is the output enable, is used as the enable pin for the IO pad logic at the chip level.

When the RFFE Slave controller detects SSC, it decodes the initial slave ID bits from the RFFE Master and compares them with its own slave ID (**rffe\_slave\_id**). If they match, the slave controller begins decoding the command. As shown in the waveform, the slave decodes the command and sets **rffe\_bus\_addr[15:0]** to 0xF38E, the same as the Master-configured address, incrementing it three times for each byte of data to be send from Slave and **reg\_rffe\_bus\_rd\_en** is asserted for 4 bytes to be sent from Slave.



## 8. RFFE CLOCK GATING

From the waveform below, we can see that RFFE\_CLK\_CG is active only when CGC\_ENABLE is HIGH. We can see the de-assertion of CGC\_ENABLE happening during the active edge of the input clock RFFE\_CLK. Due to the negative level-sensitive latch-based clock gating, there is no glitch at the output RFFE\_CLK\_CG signal. The CGC\_ENABLE will be asserted when there is an incoming command configured from software and will remain high until the end of the transaction sent via the output data pad. Hence, the clock will toggle only when there is an active RFFE transaction happening on the RFFE data and clock pad, which will reduce the power consumption due to clock toggling.



## 12. Conclusion

The RFFE protocol, derived from the MIPI Alliance's SPMI, addresses the growing complexity and demands of modern wireless communication systems. It provides a standardized, two-wire serial interface for controlling RF front-end devices, such as power amplifiers, low-noise amplifiers, filters, switches, power management modules, antenna tuners, and sensors. This protocol simplifies integration, reduces space constraints, enhances power efficiency, and ensures optimal performance by supporting efficient control and quick adjustments to RF parameters.

The RFFE Interface IP controller is divided into the RFFE Master Controller and the RFFE Slave Controller. The Master Controller, operating in two clock domains, initiates transactions and drives the communication signals, while the Slave Controller responds to these commands. Synchronizers are crucial in this design to ensure reliable communication between different clock domains, preventing metastability and data corruption.

Overall, the RFFE protocol and its interface controllers provide a robust solution for the challenges in modern wireless communication, promoting interoperability and efficient operation of RF front-end components.

## **13. Literature References**

*[1]* MIPI Alliance Specification for RF Front-End Control Interface (RFFE SM), Version 3.0, MIPI Alliance, Inc., 1 December 2019

*[2]* Power reduction through RTL Clock Gating by Frank Emmett and Mark Biegel.

## 14. Timelines

SI	Phases	Start Date-End Date	Progress
1	Dissertation Outline	27 July 2024 – 03 Aug 2024	Completed
2	Architecture Planning & RTL Design Phase1 (RFFE Write command sequences)	04 Aug 2024 – 25 Aug 2024	Completed
3	Testing Phase1	26 Aug 2024 – 19 Sep 2024	Completed
3	Midsemester Preparation	20 Sep 2024 – 27 Sep 2024	Completed
2	Architecture Exploration & RTL Design Phase2 (RFFE Read command sequences and Clock gating)	28 Sep 2024 – 12 Oct 2024	Completed
3	Testing Phase2	13 Oct 2024 – 02 Nov 2024	Completed
4	Dissertation Review	03 Nov 2024 – 10 Nov 2024	Completed
5	Submission	11 Nov 2024 – 18 Nov 2024	Completed



## 15. Scanned Copy Signed Documents

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCEPILANI  
(RAJASTHAN)  
WILP Division**

**Organization:** Qualcomm. **Location:** Bangalore

**Duration:** 2 years **Date of Start:** 20/7/2024

**Date of Submission:** 11/11/2024

**Title of the Project:** DESIGNING MIPI STANDARD RFFE INTERFACE IP FOR RF FRONT-END DEVICES

**ID No./Name of the student:** 2022HT80626/ Subramanya N N

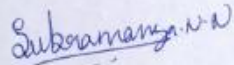
**Name (s) and Designation (s) of your Supervisor and Additional Examiner:**

**Supervisor:** Nakul Manjunath, Staff Engineer


**Additional Examiner:** Utpal Barman, Director Engineer

**Name of the Faculty mentor:** Sanjay Vidhyadharan

**Project Areas:** VLSI Architecture

  
**Signature of Student**

**Date:** 11/11/2024

  
**Signature of your Supervisor**

**Date:** 11/11/2024

### **Checklist of Items for the Final Dissertation / Project / Project Work Report**

This checklist is to be attached as the last page of the final report.

**This checklist is to be duly completed, verified and signed by the student.**

1.	<b>Is the final report neatly formatted with all the elements required for a technical Report?</b>	<b>Yes</b>
2.	Is the Cover page in proper format as given in Annexure A?	<b>Yes</b>
3.	Is the Title page (Inner cover page) in proper format?	<b>Yes</b>
4.	(a) Is the Certificate from the Supervisor in proper format? (b) Has it been signed by the Supervisor?	<b>Yes</b> <b>Yes</b>
5.	Is the Abstract included in the report properly written within one page? Have the technical keywords been specified properly?	<b>Yes</b> <b>Yes</b>
6.	Is the title of your report appropriate? <b>The title should be adequately descriptive, precise and must reflect scope of the actual work done.</b> Uncommon abbreviations / Acronyms should not be used in the title	<b>Yes</b>
7.	Have you included the List of abbreviations / Acronyms?	<b>Yes</b>
8.	Does the Report contain a summary of the literature survey?	<b>Yes</b>
9.	Does the Table of Contents include page numbers? (i). Are the Pages numbered properly? (Ch. 1 should start on Page # 1) (ii). Are the Figures numbered properly? (Figure Numbers and Figure Titles should be at the bottom of the figures) (iii). Are the Tables numbered properly? (Table Numbers and Table Titles should be at the top of the tables) (iv). Are the Captions for the Figures and Tables proper? (v). Are the Appendices numbered properly? Are their titles appropriate	<b>Yes</b> <b>Yes</b> <b>Yes</b> <b>Yes</b> <b>Yes</b> <b>Yes</b>
10.	Is the conclusion of the Report based on discussion of the work?	<b>Yes</b>
11.	Are References or Bibliography given at the end of the Report? Have the References been cited properly inside the text of the Report? Are all the references cited in the body of the report	<b>Yes</b> <b>Yes</b> <b>Yes</b>
12.	Is the report format and content according to the guidelines? The report should not be a mere printout of a PowerPoint Presentation, or a user manual. Source code of software need not be included in the report.	<b>Yes</b>

#### **Declaration by Student:**

I certify that I have properly verified all the items in this checklist and ensure that the report is in proper format as specified in the course handout.

Place: Bangalore

Date: 11/11/2024

Subramanya N N  
Signature of the Student

Name: SUBRAMANYA N N

ID No.: 2022ht80626

**BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI  
WORK INTEGRATED LEARNING PROGRAMMES (WILP) DIVISION**

**Final Evaluation Sheet**

**BITS ID No.: 2022ht80626**

**NAME OF THE STUDENT: Subramanya N N**

**EMAIL ADDRESS: 2022ht80626@wilp.bits-pilani.ac.in**

**NAME OF THE SUPERVISOR: Nakul Manjunath**

**PROJECT TITLE: Designing MIPI Standard RFFE Interface IP for RF Front-End Devices**

*Final Evaluation Please put a tick ( ✓ ) mark in the appropriate box)*

S. No.	Evaluation Component	Excellent	Good	Fair	Poor
1.	Final Project Report	✓			
2.	Final Seminar and Viva-Voce	✓			

S. No.	Evaluation Criteria	Excellent	Good	Fair	Poor
1	Technical/Professional Competence	✓			
2	Work Progress and Achievements	✓			
3	Documentation and expression	✓			
4	Initiative and Originality		✓		
5	Research & Innovation		✓		
6	Relevance to the work environment	✓			

Please ENCIRCLE the Recommended Final Grade: Excellent / Good / Fair / Poor

**Remarks of the Supervisor:** Subramanya has played a critical role in the development of MIPI RFFE IP, & has contributed to high quality IP with all the KPI's met.

	Supervisor	Additional Examiner
Name	Nakul Manjunath	Utpal Barman
Qualification	Master of Science and Technology	Master of Technology
Designation	Engineer, Staff	Director, Engineering
Employing Organization & Location	Qualcomm India Pvt Ltd, Bangalore	Qualcomm India Pvt Ltd, Bangalore
Phone Number	-	-
Mobile Number	+919739864419	+916360626835
Email Address	nakumanj@qti.qualcomm.com	utpalb@qti.qualcomm.com
Signature		
Place & Date	Bangalore, 11/11/2024	Bangalore, 11/11/2024