Chip Design Course

Final Project Report

Date: 12/02/2026

Students:

Assaf Afriat

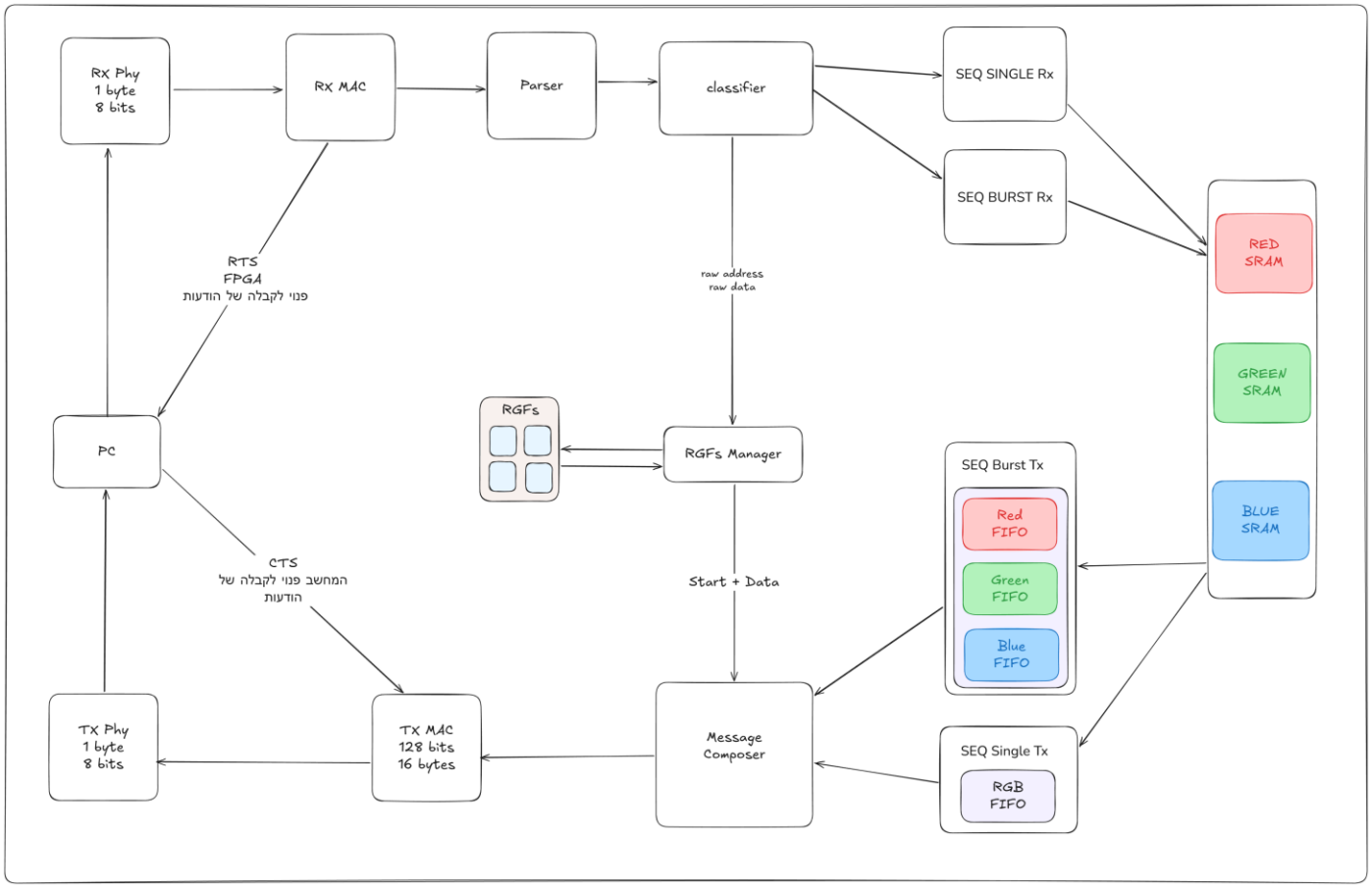
Peleg Shay

Yuval Kayat

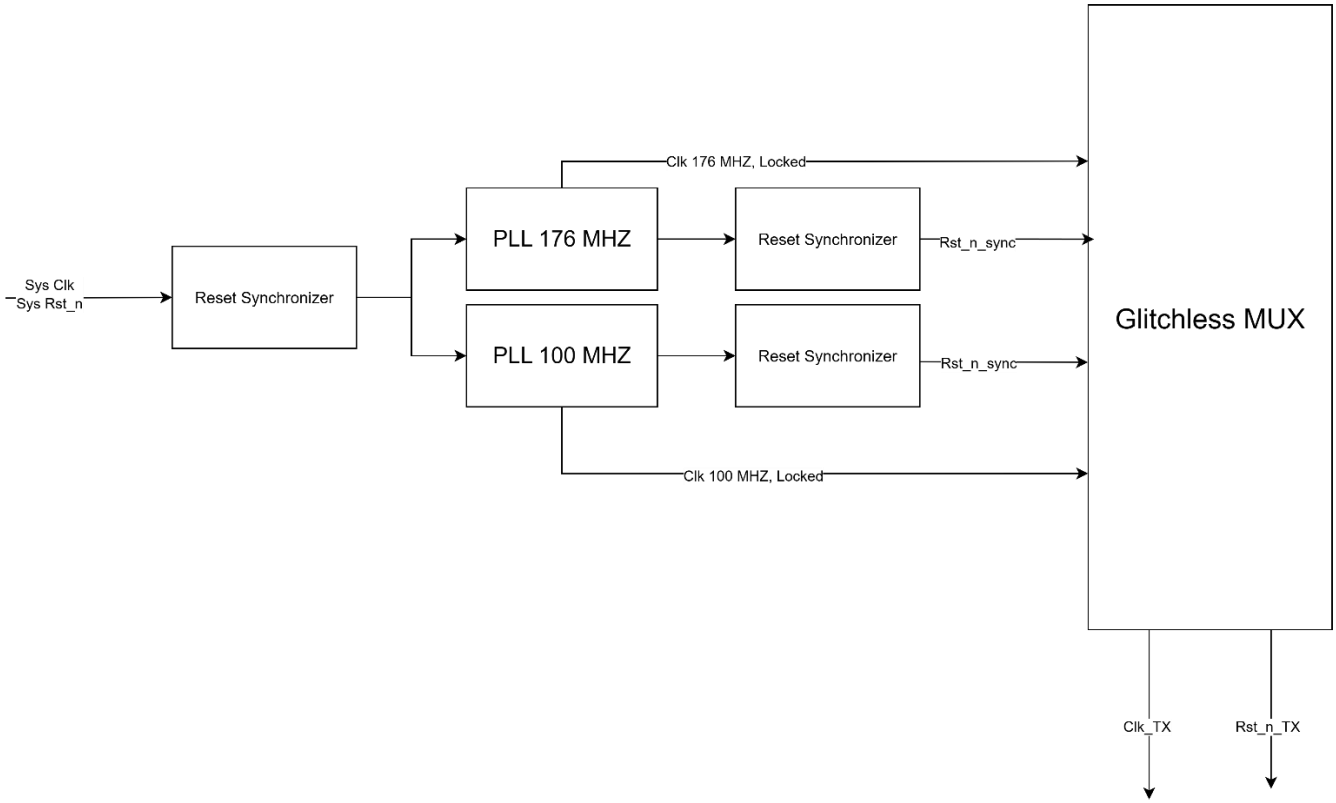
All screenshots are attached in Report folder

Contents

Full System Architecture

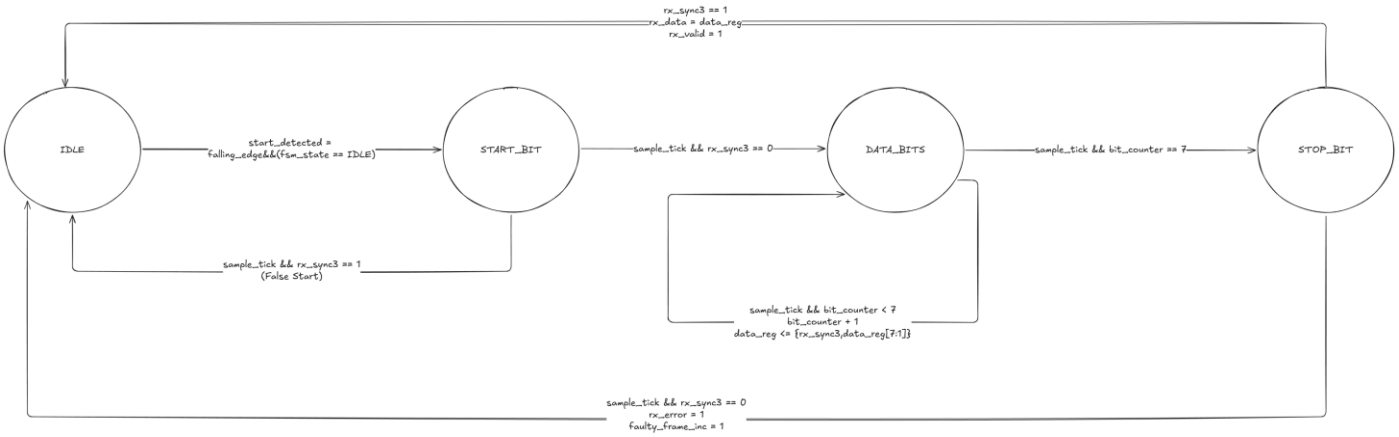


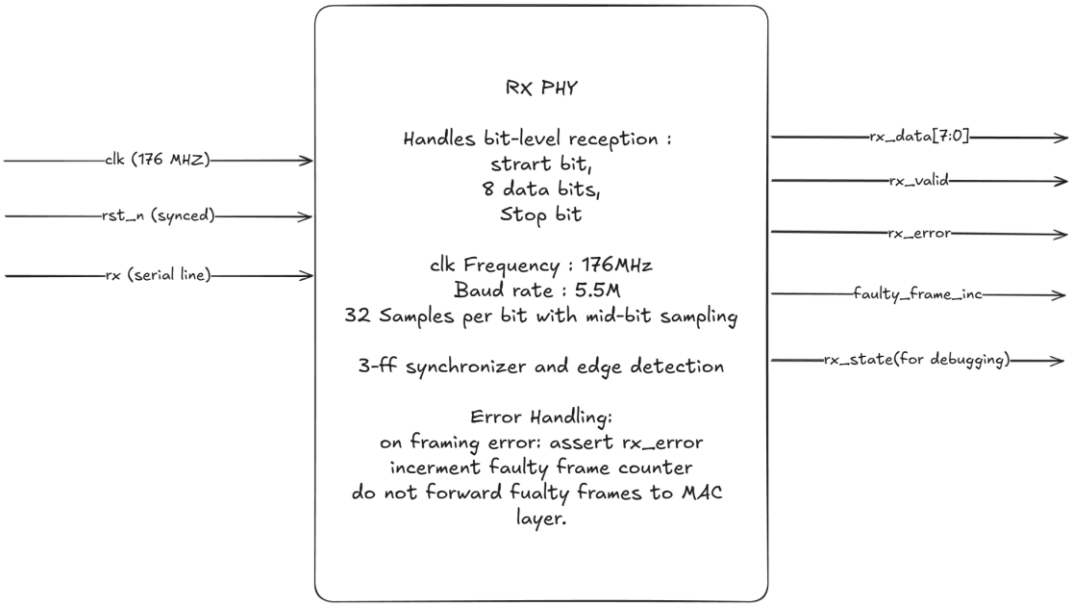
System Configuration



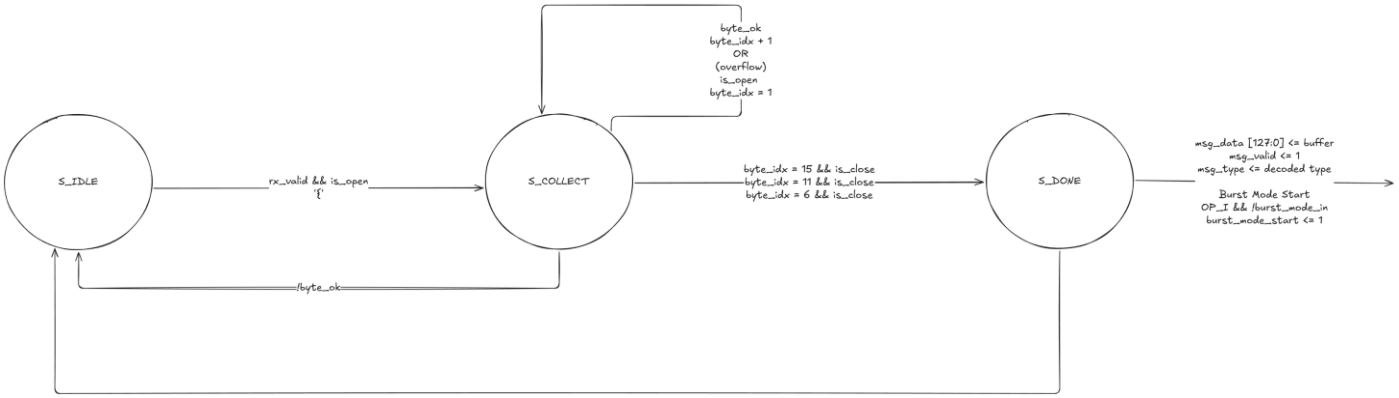
Sub-Systems Architecture

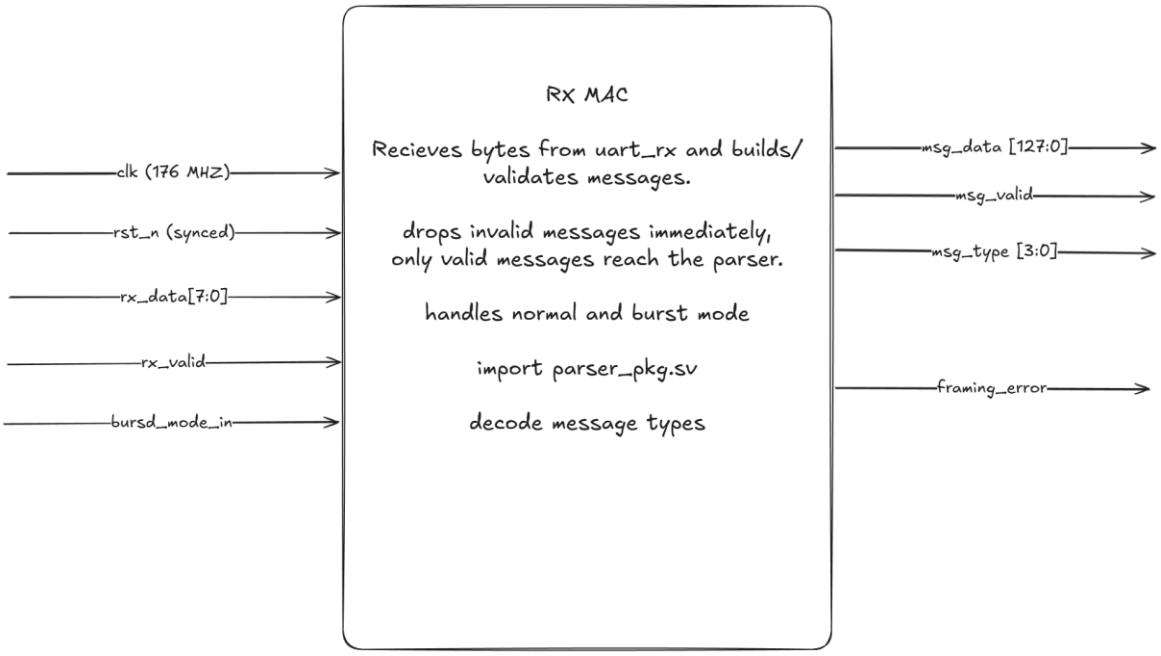
Rx PHY



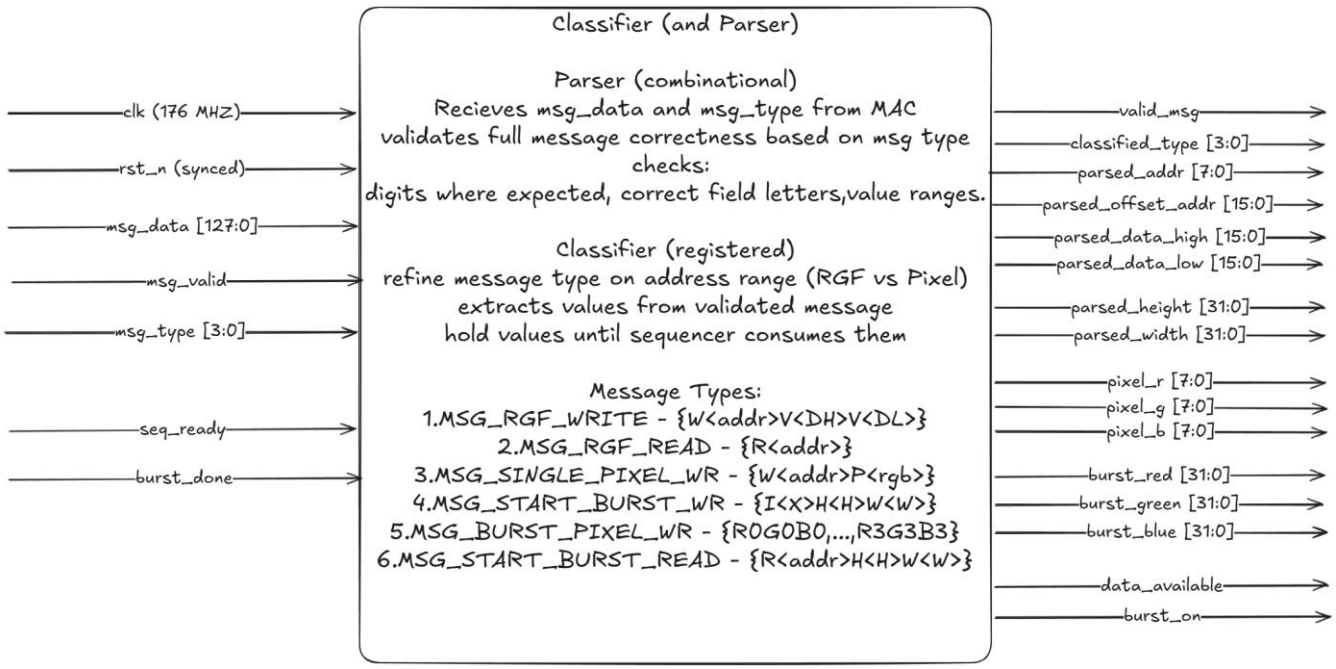


Rx MAC


byte_ok
byte_idx + 1
OR
(overflow)
is_open
byte_idx = 1
S_IDLE
rx_valid && is_open
'{'
S_COLLECT
byte_idx = 15 && is_close
byte_idx = 11 && is_close
byte_idx = 6 && is_close
S_DONE
msg_data [127:0] <= buffer
msg_valid <= 1
msg_type <= decoded type
Burst Mode Start
OP_I && !burst_mode_in
burst_mode_start <= 1
!byte_ok


Rx MAC
Recieves bytes from uart_rx and builds/ validates messages.
drops invalid messages immediately, only valid messages reach the parser.
handles normal and burst mode
import parser_pkg.sv
decode message types
clk (176 MHZ)
rst_n (synced)
rx_data[7:0]
rx_valid
bursd_mode_in
msg_data [127:0]
msg_valid
msg_type [3:0]
framing_error

Parser and Classifier



RGF Manger

RGFs Manager

inputs:
raw_address
raw_data
MSG_Type
rd_data_from_rgfs
cmpsr_busy

outputs:
RGF legs
offset address
wr_data
wr_en
rd_en

raw_address_to_cmpsr
data_out_to_cmpsr
start_req_to_cmpsr
rgf_read

Description:

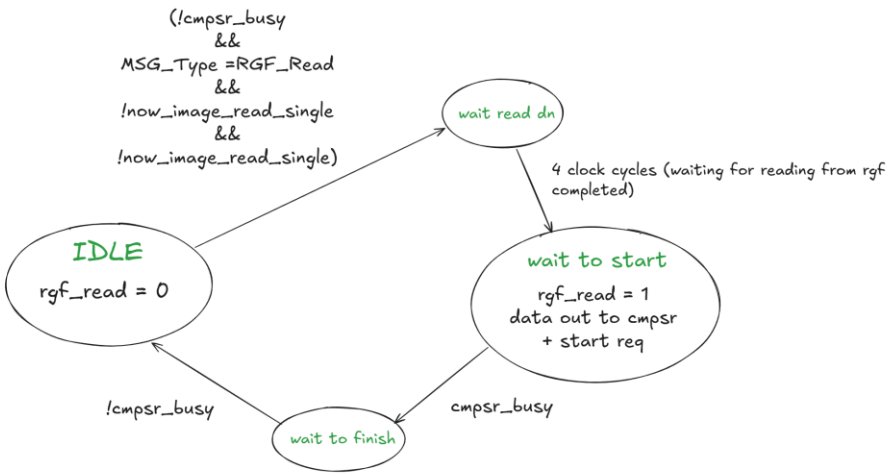
when write rgf request then wr_en
(write rgf requeset: MSG_Type = RGF_Write)

when read rgf request then rd_en
(read rgf requeset: MSG_Type = RGF_Read)

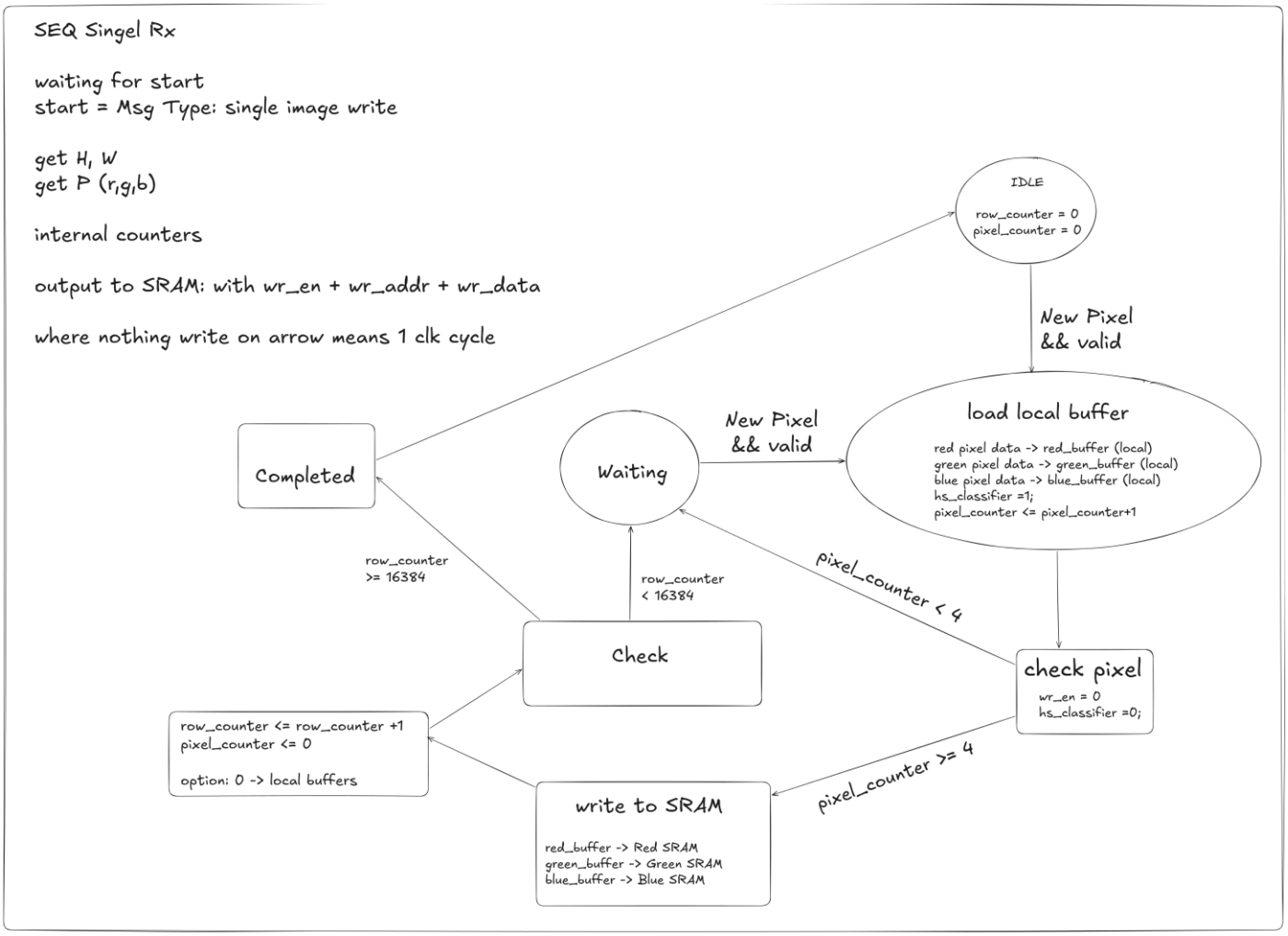
raw address to:
1. base address -> decoding legs
2. offset address -> RGF addr

for write: raw_data -> wr_data

for read: state machine
collecting all relevant data and send to
message composer withe start_req.



Rx Single Sequencer



Rx Sequencer Burst

SEQ Burst Rx

waiting for start
start = Msg Type: start burst image write

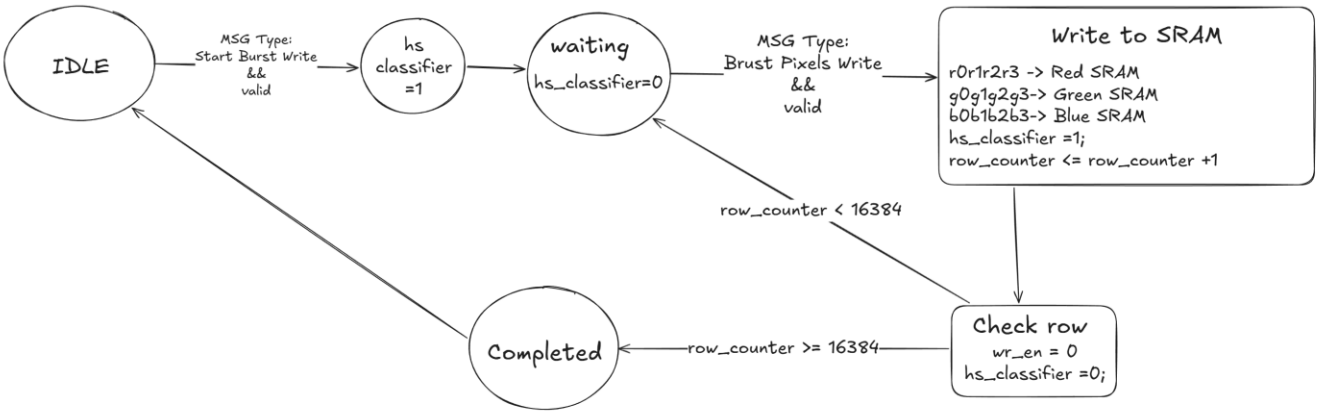
waiting for pixels
pixels = Msg Type: burst image write

get H, W
get 4 pixles (r0g0b0...r3g3b3)

internal counters

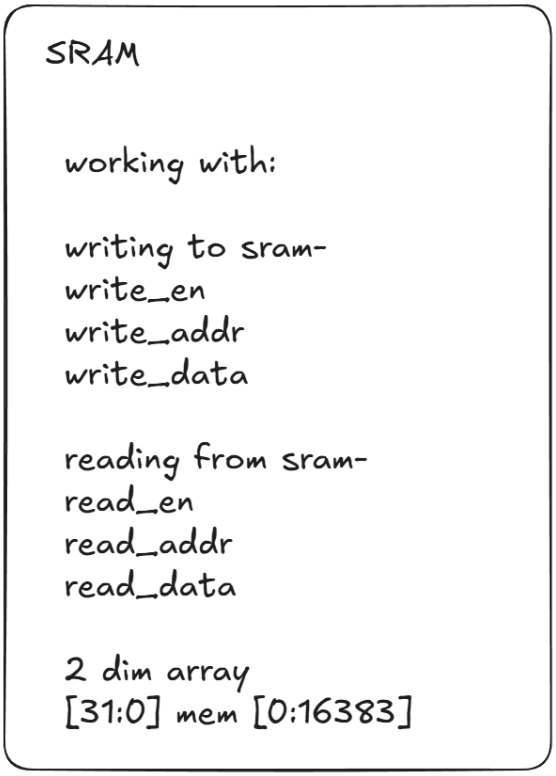
output to SRAM: with wr_en + wr_addr + wr_data

where nothing write on arrow means 1 clk cycle

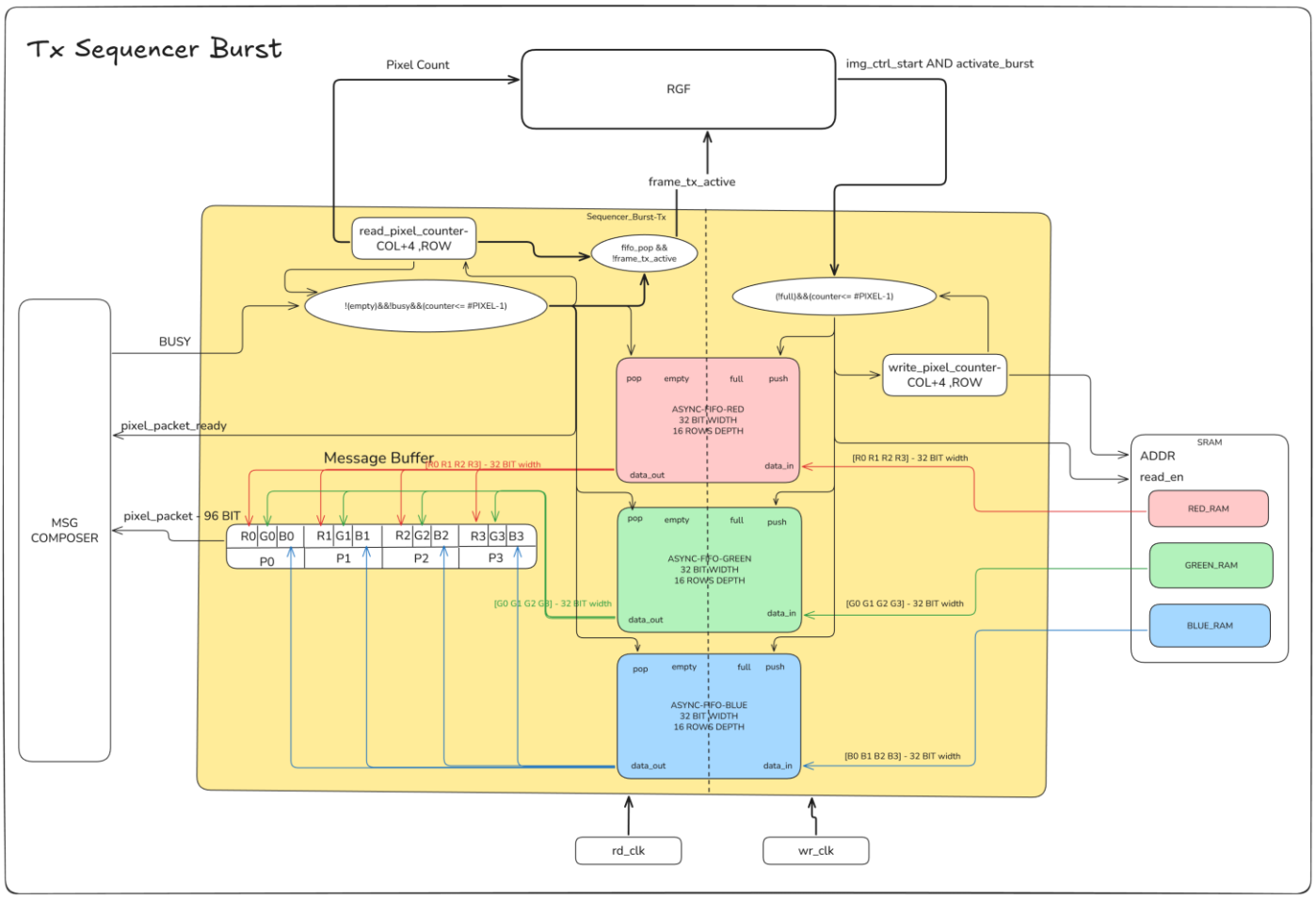


SRAM

*3 for Red, Green, Blue



Tx Sequencer Burst



Message Composer

Message Composer

1. buliding 3 different structre messages according to the "now" flag

{r0g0b0r1,g1b1r2g2,b2r3g3b3}

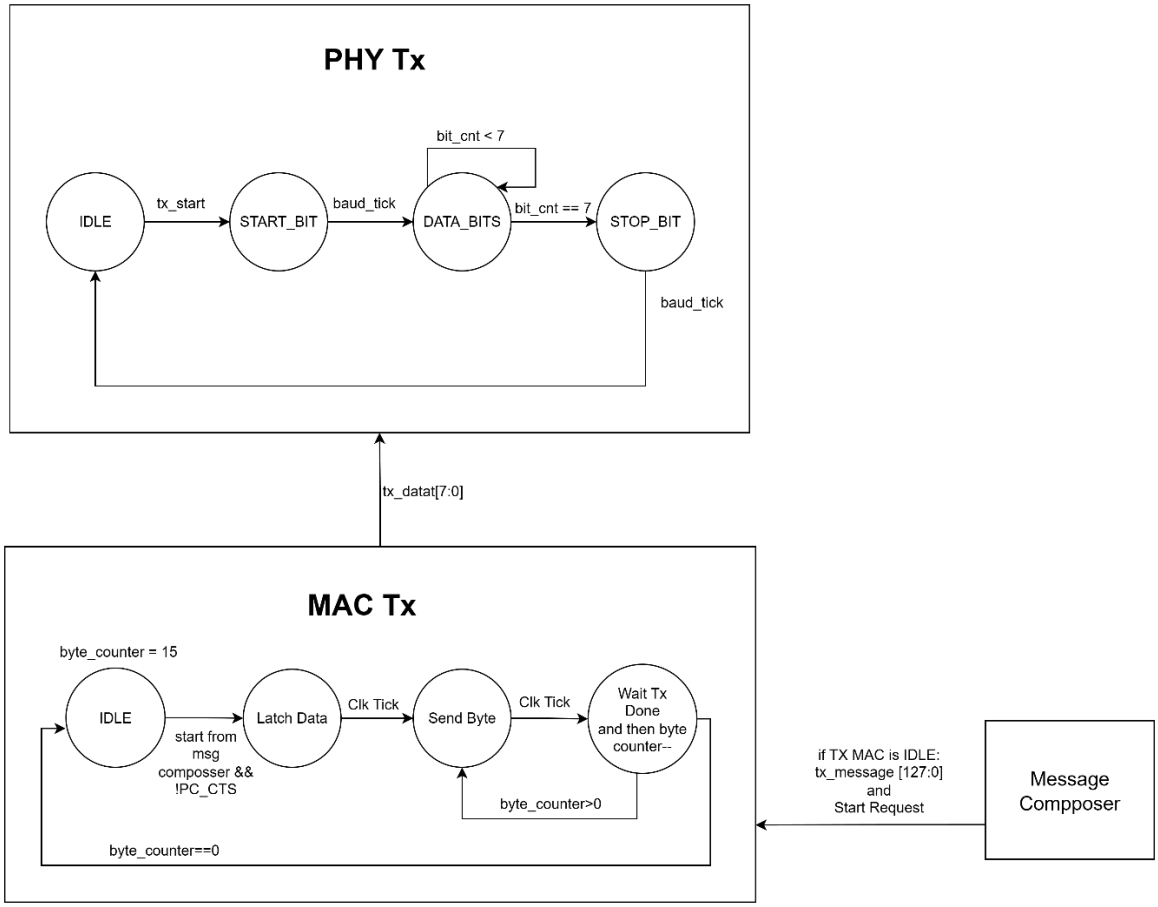
{Rxxx,Cxxx,Pr0g0b0}

{Raaa,VxDH1DH0,VxDL1DL0}

2. get 3 "cmpsr start req"
2.1. choose 1 and send out "tx mac start req"

3. get "tx mac busy"
3.1. push it forward as "msg cmpsr busy" to all 3

Tx PHY and Tx MAC



Modules IO Table

Rx PHY

Module: Rx PHY			
Parameters	Inputs	Outputs	Local Var
CLK_FREQ	clk	rx_data	fsm_state,next_fsm_state
BAUD_RATE	rst_n	rx_valid	clk_counter [COUNTER_WIDTH - 1 :0]
	rx	rx_error	bit_counter [2:0]
Local Parameters		faulty_frame_inc	data_reg [7:0]
CLKS_PER_BIT		rx_state	sample_tick
SAMPLE_POINT			rx_sync1,rx_sync2,rx_sync3
COUNTER_WIDTH			rx_prev
			falling_edge
			frame_error_flag
			start_detected

Rx MAC

Module: Rx MAC			
Parameters	Inputs	Outputs	Local Var
	clk	msg_data[127:0]	S_IDLE,S_COLLECT,S_DONE
	rst_n	msg_valid	buffer [127:0]
	rx_data [7:0]	msg_type	byte_idx [3:0]
Local Parameters	rx_valid		opcode [7:0]
NONE,REG_WRITE,REG_READ	burst_mode_in	frame_error	msg_ended_5,msg_ended_10
IMG_BURST_WRITE,IMG_BURST_READ,INVALID			is_open, is_close , is_comma
CHAR_OPEN, CHAR_CLOSE, CHAR_COMMA			is_valid_opcode, is_valid_field6, is_valid_field11
OP_W, OP_R, OP_I			byte_ok
FL6_V,FL6_P,FL6_C, FL6_H			
FL11_V, FL11_P, FL11_W			

Classifier

Module: Classifier			
Parameters	Inputs	Outputs	Local Var
	clk	valid_msg	byte0,byte1,byte2,byte3,byte4,byte5,byte6,byte7
	rst_n	classified_type[3:0]	byte8,byte9,byte10,byte11,byte12,byte13,byte14,byte15
	msg_data[127:0]	parsed_addr[7:0]	p_addr ,p_offset_addr, p_data_high, p_data_low, p_height, p_width
	msg_valid	parsed_offset_addr	p_pixel_r,p_pixel_g,p_pixel_b
	msg_type[3:0]	parsed_data_high[7:0]	p_pixel_r,p_pixel_g,p_pixel_b
	seq_ready	parsed_data_low[7:0]	p_burst_p0_r,p_burst_p1_r,p_burst_p2_r,p_burst_p3_r
	burst_done	parsed_height[7:0]	p_burst_p0_g,p_burst_p1_g,p_burst_p2_g,p_burst_p3_g
		parsed_width[7:0]	p_burst_p0_b,p_burst_p1_b,p_burst_p2_b,p_burst_p3_b
		pixel_r[7:0]	data_valid_reg
		pixel_g[7:0]	burst_on_reg
		pixel_b[7:0]	pixel_r_reg, pixel_g_reg, pixel_b_reg
		burst_red[31:0]	burst_red_reg, burst_green_reg, burst_blue_reg
		burst_green[31:0]	
		burst_blue[31:0]	
		data_available	
		burst_on	

RGF Manager

Module: RGFs Manager						
Parameters	Inputs	comment	Outputs	comment	Local Var	comment
	clk		rgf_leg_img		RGF_base_address	for decoding
	rst_n		rgf_leg_tx_fifo		current_state	
	raw_address		rgf_leg_led		next_state	
	raw_data		rgf_leg_sys			
	MSG_Type		rgf_leg_ctrl			
	read_data_from_rgfs		rgf_leg_pwm			
	cmpsr_busy		offset_address			
	now_image_read_single		wr_data			
	now_image_read_burst		wr_en			
			rd_en			
			raw_address_to_cmpsr			
			data_out_to_cmpsr			
			start_req_to_cmpsr			
			rgf_read	writing back to pc in process		

Rx Sequence Single

Module: Rx Sequencer Single						
Parameters	Inputs	comment	Outputs	comment	Local Var	comment
	clk	from RGF	red_wr_data		row_counter	
	rst_n	from RGF	green_wr_data		pixel_counter	
	IMG_WIDTH	from RGF	blue_wr_data		red_buffer	
	IMG_HIGHET	from RGF	sram_r_wr_en		green_buffer	
	MSG_TYPE		sram_r_wr_addr		blue_buffer	
	red_pixel	from msg	sram_g_wr_en		pixel_index	for load local buffer
	green_pixel	from msg	sram_g_wr_addr		current_state	
	blue_pixel	from msg	sram_b_wr_en		next_state	
	pixel_address	from msg	sram_b_wr_addr		New_Pixel_Write_Single	according MsgType
			rx_seq_single_busy	to all to know	max_rows	to finish writing
			rx_seq_single_dn	to all to know		

Rx Sequencer Burst

Module: Rx Sequencer Burst						
Parameters	Inputs	comment	Outputs	comment	Local Var	comment
	clk		red_wr_data		row_counter	
	rst_n	from Rx Msg	green_wr_data		current_state	
	MSG_TYPE		blue_wr_data		next_state	
	IMG_WIDTH	from Rx Msg	sram_r_wr_en		Start_IMG_Write_Burst	according MsgType
	IMG_HIGHET	from Rx Msg	sram_r_wr_addr		New_Pixels_Write_Brust	according MsgType
	burst_red		sram_g_wr_en		max_rows	to finish writing
	burst_green		sram_g_wr_addr			
	burst_blue		sram_b_wr_en			
			sram_b_wr_addr			
			rx_seq_burst_busy	to all to know		
			rx_seq_burst_dn	to all to know		

SRAM

Module: RGB SRAM							
Parameters	comment	Inputs	comment	Outputs	comment	Local Var	comment
DATA_WIDTH		clk		rd_data		addr_depth	
DATA_DEPTH		rst_n					
		wr_en					
		wr_addr					
		wr_data					
		rd_en					
		rd_addr					

Tx Sequencer Burst

Module: Sequencer_Burst-Tx			
Parameters	Inputs	Outputs	Local Var
RAM_ADDR_WIDTH	wr_clk	pixel_packet	
RAM_DATA_WIDTH	rd_clk	pixel_counter	
FIFO_WIDTH	rst_wr_n	red_RAM_addr	
FIFO_DEPTH	rst_rd_n	blue_RAM_addr	
PIXEL_COUNT	red_RAM_data	green_RAM_addr	
	blue_RAM_data	red_read_en	
	green_RAM_data	green_read_en	
	tx_busy	blue_read_en	
	img_complete	pixel_packet_ready	
	img_ctrl_start	frame_tx_active	
	activate_burst	burst_tx_seq_dn	

Async FIFO

Module: FIFO Async							
Parameters	comment	Inputs	comment	Outputs	comment	Local Var	comment
FIFO_DEPTH		wr_clk		empty		mem	
CELL_WIDTH		wr_rst_m		almost_empty		wr_pntr	
ALMST_EMPTY_LVL		rd_clk		almost_full		rd_pntr	
ALMST_FULL_LVL		rd_rst_n		full		wr_en	
		push		data_out	to Seq B	rd_en	
		data_in				cells_free_to_wr	
		pop				cells_writen	

Message Composer

Module: Message Compposer							
Parameters	comment	Inputs	comment	Outputs	comment	Local Var	comment
		clk		data_out_msg_cmps	to MAC_TX [127:0] {xxxx,xxxx,xxxx}	rgf_msg	assign to data out
		rst_n		MAC_TX_Start_Req	to MAC_TX	img_single_msg	
		now_rgf_read		cmpsr_busy	assign from tx_mac_busy, to all	img_burst_msg	
		rgf_raw_address	from rgf manager			base_address_rgf	
		rgf_data				offset_address_rgf	
		now_image_read_single					
		img_row_counter	from seq single tx				
		img_col_counter					
		pixel_cell					
		now_image_read_burst					
		red_burst_data	from seq burst tx				
		green_burst_data					
		blue_burst_data					
		to_cmpsr_start_req	from all				
		tx_mac_busy					

Tx MAC

Module: Tx Mac							
Parameters	comment	Inputs	comment	Outputs	comment	Local Var	comment
		clk		to_tx_phy_start		byte_counter	till 16
		rst_n		to_tx_phy_data [7:0]		current_state	fsm_state_t
		start_req_from_cmpsr		tx_mac_busy		next_state	fsm_state_t
		data_in_from_cmpsr [127:0]					
		tx_phy_busy					
		tx_phy_dn					
		PC_CTS					

Tx PHY

Module: Tx Phy							
Parameters	comment	Inputs	comment	Outputs	comment	Local Var	comment
		clk		tx_out		bit_counter	
		rst_n		tx_phy_busy		baud_tick	
		start_req		tx_phy_dn		current_state	
		data_in [7:0]				next_state	

RGFs Tables

IMG RGF

Reg Name	Offset Address	Type	Fields	Size	Deafult Value	Description
IMG Status	0x0000	RW	bit_field_0	10	10'h0	Image Height
			bit_field_1	10	10'h0	Image Width
			bit_field_2	1	1'h0	Image Ready in PC
IMG TX Monitor	0x0004	RO	bit_field_0	10	10'h0	Row cnt
			bit_field_1	10	10'h0	Column cnt
			bit_field_2	1	1'h0	Image transfer complete
			bit_field_3	1	1'h0	Image Ready in SRAM
IMG CTRL	0x0008	RW	bit_field_0	1	1'h0	Start image read

Seq TX IMG FIFO RGF

Reg Name	Offset Address	Type	Fields	Size	Deafult Value	Description
FIFO Sizes	0x0000	RW	bit_field_0	12	12'h400	FIFO Depth
			bit_field_1	6	6'h20	Singel cell width
Treshold LVL	0x0004	RW	bit_field_0	11	11'h40	Almost empty level
			bit_field_1	11	11'h40	Almost full level

LED RGF

Reg Name	Offset Address	Type	Fields	Size	Deafult Value	Description
LED_CTRL	0x0000	RW	bit_field_0	2	2'h0	Led Select
			bit_field_1	1	1'h0	LED16 on/of
			bit_field_2	1	1'h0	LED17 on/off
			bit_field_3	1	1'h0	LED16 CIE on/off
			bit_field_4	1	1'h0	LED17 CIE on/off
LED_Pattern_CFG	0x0004	RW	bit_field_0	2	2'h0	static (00), Breath (01), Rainbow (10)

PWM RGF

Reg Name	Offset Address	Type	Fields	Size	Deafult Value	Description
PWM_CFG	0x0000	RW	bit_field_0	16	16'h0000	PWM Time Slots
			bit_field_1	16	16'h0000	PWM Sweep time
Red_LUT	0x0004	RW	bit_field_0	13	13'h000A	Output Freq
			bit_field_1	2	2'h0	Magnitude
			bit_field_2	9	9'h000	Init Phase
Green_LUT	0x0008	RW	bit_field_0	13	13'h000A	Output Freq
			bit_field_1	2	2'h0	Magnitude
			bit_field_2	9	9'h000	Init Phase
Blue_LUT	0x000C	RW	bit_field_0	13	13'h000A	Output Freq
			bit_field_1	2	2'h0	Magnitude
			bit_field_2	9	9'h000	Init Phase

CNT RGF

Reg Name	Offset Address	Type	Fields	Size	Deafult Value	Description
UART RX Packets cnt	0x0000	RW	bit_field_0	32	32'h00000000	valid RX packet counter
UART TX Packets cnt	0x0004	RW	bit_field_0	32	32'h00000000	How many TX packets were transmitted
Color Messages cnt	0x0008	RW	bit_field_0	32	32'h00000000	How many of the valid received packets were color messages
Config Messages cnt	0x000C	RW	bit_field_0	32	32'h00000000	How many of the valid received packets were config messages

SYS RGF

Reg Name	Offset Address	Type	Fields	Size	Deafult Value	Description
CTRL	0x0000	RW	bit_field_0	1	1'h0	SW reset
			bit_field_1	1	1'h0	enable

FIFO Configuration and Threshold Definitions

Design Rationale

Following the design guideline to minimize FIFO resources when the downstream consumer (UART TX) requires a continuous data stream, we implemented a compact FIFO architecture. The primary objective was to prevent data starvation (underflow) at the UART interface while managing the high-speed burst writes from the ROM reader.

Configuration Parameters

1. **FIFO Depth:** 16.
    - *Justification:* A minimal depth is sufficient because the producer (Sequence A) operates at the system clock speed (100 MHz) and can refill the buffer significantly faster than the consumer (Sequence B) can drain it at the UART baud rate.
2. **FIFO Width:** 32 bits.
    - *Justification:* Selected to align with standard architectural word sizes, accommodating the 24-bit RGB pixel data with padding.

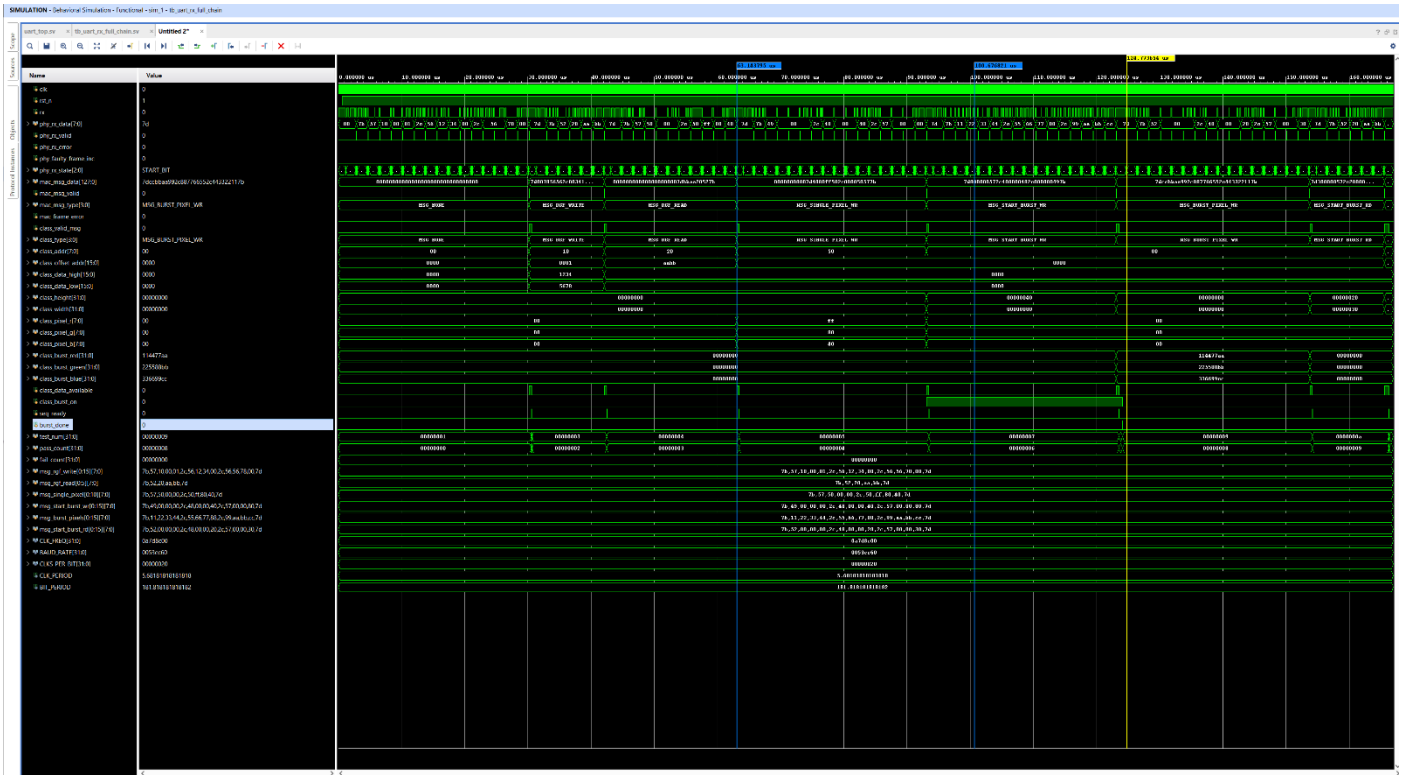
Threshold Levels

The Sequence A FSM utilizes hysteresis flow control, halting writes when the FIFO is "Almost Full" and resuming only when it reaches "Almost Empty."

- **Almost Full Level (12):** This threshold provides a safety margin of 4 values (16 - 12 = 4). Since Sequence A writes data in bursts of 4 values per loop, setting the flag at 12 ensures that an active burst can complete without causing a FIFO overflow.
- **Almost Empty Level (4):** This threshold creates a safety band to prevent the FIFO from reaching zero. By triggering a refill while 4 values remain, we ensure the UART MAC TX never idles due to missing data, maintaining continuous transmission.

Simulation

Simulation 1: Full Rx Path & Burst Mode Handshake

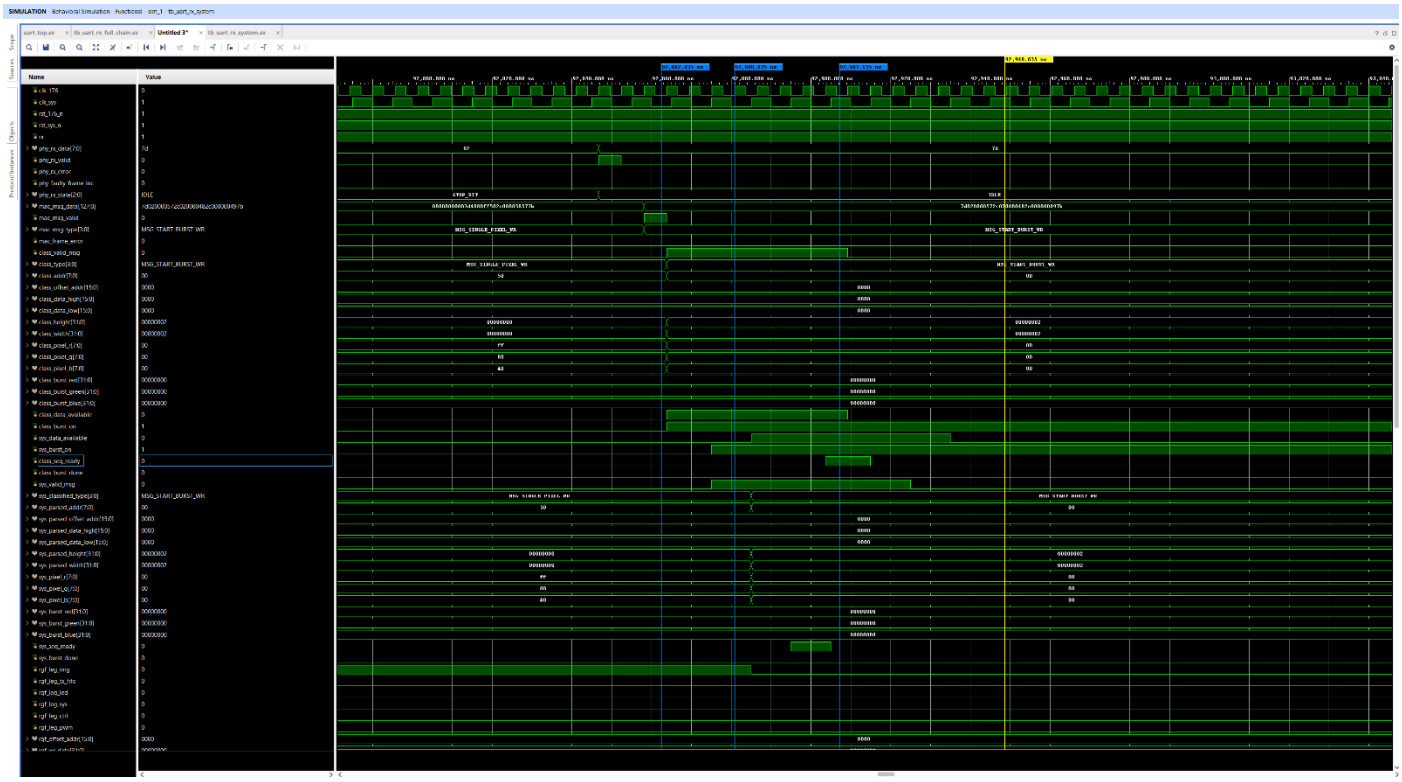


Objective: To verify the complete Receive (Rx) data path from the PHY layer up to the Classifier, focusing on the control logic and handshake mechanism between the Classifier and the Burst Sequencer during a high-speed image transfer.

Waveform Analysis:

- **First Blue Marker (Initialization Handshake):** The simulation begins by validating the interface between the **Classifier** and the **Sequencer**. The waveform shows the initial handshake where the Classifier validates the incoming message header and signals the Sequencer to prepare for operation.
- **Second Blue Marker (Burst Mode Activation):** This section demonstrates the system's entry into Burst Mode. Following the reception of a MSG_START_BURST_WR command, the BURST_ON signal is asserted (logic high). As observed, this signal remains latched high throughout the entire data transmission sequence, ensuring the system processes the stream of incoming pixel data continuously without requiring re-initialization for every byte.
- **Yellow Marker (Termination & Handshake Closure):** The yellow cursor highlights the completion of the burst transfer. The Sequencer generates a BURST_DONE pulse, indicating that the required number of pixels has been successfully written to memory. This pulse acts as a feedback signal to the Classifier, immediately triggering the de-assertion (drop) of the BURST_ON signal. This successfully completes the handshake loop, returning the system to an IDLE state ready for the next command.

Simulation 2: Clock Domain Crossing (CDC) & Multi-Clock Handshake

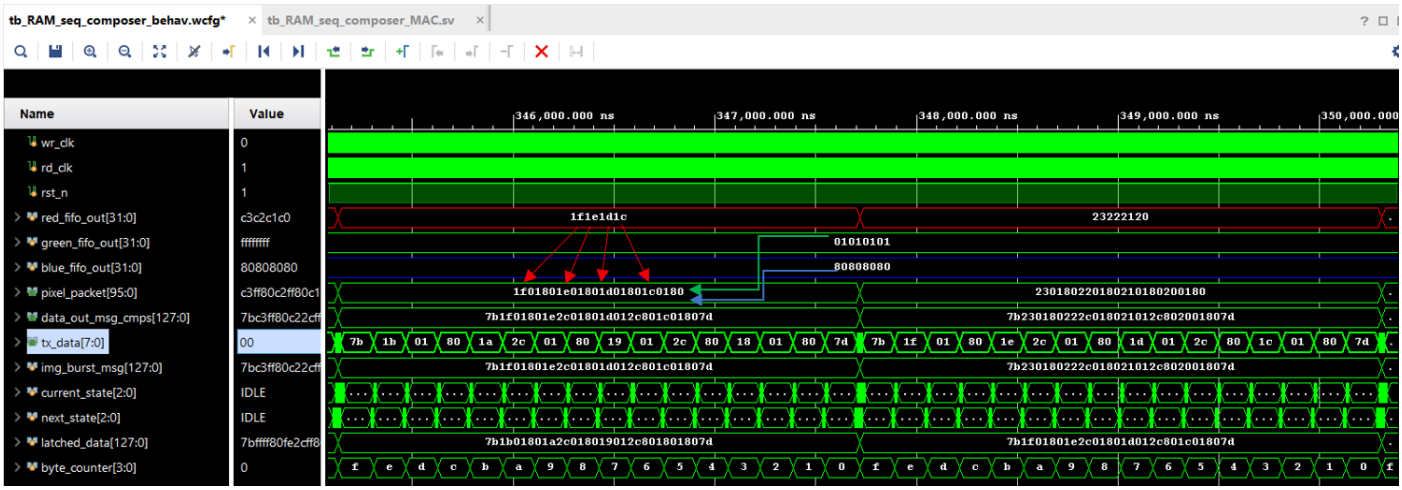


Objective: To verify the robustness of the **Clock Domain Crossing (CDC)** logic within the system. This simulation focuses on the msg_handler module (which wraps the Sequencers and RGF Manager), ensuring reliable signal synchronization between the **Fast Clock Domain (176 MHz)** used by the Rx PHY/Classifier and the **System Clock Domain** used by the Sequencers.

Waveform Analysis:

- **First Blue Marker (Source Domain Assertion):** The cursor highlights the initiation of a transaction in the Fast Clock domain. The signals class_data_available and class_burst_on are asserted (logic high) by the Classifier. This indicates that valid data has been parsed and is waiting to be consumed, but it has not yet crossed into the system clock domain.
- **Second Blue Marker (CDC to System Domain & Consumption):** This marker demonstrates the successful synchronization of control signals into the System Clock domain. The signals sys_data_available and sys_burst_on go high after passing through the CDC synchronizers.
    - **Handshake Confirmation:** Crucially, immediately after these system-domain signals are asserted and latched, the original signals in the Classifier (class_data_available, etc.) drop to low. This confirms that the handshake logic successfully communicated the "Data Consumed" status from the System domain back to the Fast domain, clearing the buffers.
- **Third Blue Marker (Feedback Path Synchronization):** The final cursor validates the reverse synchronization path (from Sequencer back to Classifier). The signal sys_seq_ready is asserted in the System domain (indicating the Sequencer is idle/ready). After a few clock cycles of synchronization delay, the corresponding class_seq_ready signal rises in the Fast domain. This confirms that status flags are correctly passing back across the clock boundary without glitches, allowing the Classifier to safely send the next packet.

Simulation 3: Tx Burst Path – From FIFO to UART PHY



Objective: To verify the **Tx Burst Data Path**, demonstrating the reconstruction of pixel data from parallel color FIFOs into a serialized UART message stream. This simulation tracks the data transformation from raw 32-bit FIFO outputs to the final 8-bit UART transmission.

Waveform Analysis:

- **Red/Green/Blue FIFO Out (Data Source):**

The waveforms at the top show the parallel data streams emerging from the asynchronous FIFOs. Each channel provides 32 bits of color data per clock cycle, representing the raw intensity values retrieved from the SRAM.

- **Pixel-Packet (Aggregation):** The Pixel-Packet signal demonstrates the internal aggregation logic within the Tx Sequencer. Here, the parallel FIFO outputs are concatenated into a single **96-bit** bus. This packet contains exactly four full pixels, arranged sequentially as [R_0G_0B_0, R_1G_1B_1, R_2G_2B_2, R_3G_3B_3].
- **Data_out_msg_cmps (Message Composition):** This signal validates the **Message Composer** logic. The 96-bit pixel packet is padded and framed into a **128-bit** UART-compliant message. The waveform shows the insertion of protocol delimiters (Start {, Commas ,, and Stop }), resulting in the format: {R_0G_0B_0R_1, G_1B_1R_2G_2, B_2R_3G_3B_3}.
- **Tx_data (Serialization):**

  The final signal, Tx_data, represents the interface to the Tx PHY. It shows the 128-bit message being broken down into individual 8-bit bytes, which are sent sequentially to the physical layer for transmission to the PC.

Linter Report and Schematics

```
RTL Linter Report

Table of Contents
-----------------
1. Summary

1. Summary
----------

+-----------+------------------+--------------+-----------+
| Rule ID   | Severity         | # Violations | # Waived |
+-----------+------------------+--------------+-----------+
| ASSIGN-5  | WARNING          | 21           | 0         |
| ASSIGN-6  | WARNING          | 47           | 0         |
| ASSIGN-7  | CRITICAL WARNING | 4            | 0         |
| ASSIGN-10 | WARNING          | 14           | 0         |
+-----------+------------------+--------------+-----------+
```

```
INFO: [Synth 37-85] Total of 86 linter message(s) generated.
INFO: [Synth 37-45] Linter Run Finished!
Synthesis Optimization Runtime : Time (s): cpu = 00:00:09 ; elapsed = 00:00:07 . Memory (MB): peak = 3135.816 ; gain = 260.152
INFO: [Common 17-83] Releasing license: Synthesis
69 Infos, 131 Warnings, 6 Critical Warnings and 0 Errors encountered.
synth_design completed successfully
synth_design: Time (s): cpu = 00:00:09 ; elapsed = 00:00:08 . Memory (MB): peak = 3135.816 ; gain = 262.250
```

Synthesis and Implementation Report

```
Report Cell Usage:
+------+---------+------+
|      |Cell     |Count |
+------+---------+------+
|1     |clk_wiz  |     2|
|3     |CARRY4   |    32|
|4     |DSP48E1  |     1|
|5     |LUT1     |    19|
|6     |LUT2     |   107|
|7     |LUT3     |    68|
|8     |LUT4     |   144|
|9     |LUT5     |   303|
|10    |LUT6     |  1015|
|11    |MUXF7    |   179|
|12    |MUXF8    |     8|
|13    |RAMB36E1 |    48|
|14    |FDCE     |  1603|
|15    |FDPE     |    21|
|16    |FDRE     |  1667|
|17    |IBUF     |     2|
|18    |OBUF     |    38|
+------+---------+------+
--------------------------------------------------------------------------------
Finished Writing Synthesis Report : Time (s): cpu = 00:00:25 ; elapsed = 00:00:26 . Memory (MB): peak = 1846.527 ; gain = 1136.164
--------------------------------------------------------------------------------
Synthesis finished with 0 errors, 66 critical warnings and 162 warnings.
Synthesis Optimization Runtime : Time (s): cpu = 00:00:19 ; elapsed = 00:00:24 . Memory (MB): peak = 1846.527 ; gain = 1055.480
Synthesis Optimization Complete : Time (s): cpu = 00:00:25 ; elapsed = 00:00:26 . Memory (MB): peak = 1846.527 ; gain = 1136.164
INFO: [Project 1-571] Translating synthesized netlist
Netlist sorting complete. Time (s): cpu = 00:00:00 ; elapsed = 00:00:00.034 . Memory (MB): peak = 1855.695 ; gain = 0.000
INFO: [Netlist 29-17] Analyzing 268 Unisim elements for replacement
INFO: [Netlist 29-28] Unisim Transformation completed in 0 CPU seconds
INFO: [Project 1-570] Preparing netlist for logic optimization
INFO: [Opt 31-140] Inserted 1 IBUFs to IO ports without IO buffers.
INFO: [Opt 31-138] Pushed 0 inverter(s) to 0 load pin(s).
Netlist sorting complete. Time (s): cpu = 00:00:00 ; elapsed = 00:00:00.001 . Memory (MB): peak = 1859.332 ; gain = 0.000
INFO: [Project 1-111] Unisim Transformation Summary:
No Unisim elements were transformed.

Synth Design complete | Checksum: d4928563
INFO: [Common 17-83] Releasing license: Synthesis
147 Infos, 187 Warnings, 68 Critical Warnings and 0 Errors encountered.
synth_design completed successfully
INFO: [Common 17-600] The following parameters have non-default value.
```

```
--------------------------------------------------------------------------------------------------------------------------------


Ending Logic Optimization Task | Checksum: 1eabe6ccd

Time (s): cpu = 00:00:00 ; elapsed = 00:00:00.257 . Memory (MB): peak = 1870.441 ; gain = 0.000

Starting Power Optimization Task
INFO: [Pwropt 34-132] Skipping clock gating for clocks with a period < 2.00 ns.
INFO: [Timing 38-35] Done setting XDC timing constraints.
Running Vector-less Activity Propagation...

Finished Running Vector-less Activity Propagation
INFO: [Pwropt 34-9] Applying IDT optimizations ...
INFO: [Pwropt 34-10] Applying ODC optimizations ...


Starting PowerOpt Patch Enables Task
INFO: [Pwropt 34-162] WRITE_MODE attribute of 0 BRAM(s) out of a total of 48 has been updated to save power. Run report_power_opt to get a complete listing of the BRAMs updated.
INFO: [Pwropt 34-201] Structural ODC has moved 0 WE to EN ports
Number of BRAM Ports augmented: 0 newly gated: 0 Total Ports: 96
Ending PowerOpt Patch Enables Task | Checksum: 1eabe6ccd

Time (s): cpu = 00:00:00 ; elapsed = 00:00:00.087 . Memory (MB): peak = 1943.516 ; gain = 0.000
Ending Power Optimization Task | Checksum: 1eabe6ccd

Time (s): cpu = 00:00:03 ; elapsed = 00:00:02 . Memory (MB): peak = 1943.516 ; gain = 73.074

Starting Final Cleanup Task
Ending Final Cleanup Task | Checksum: 1eabe6ccd

Time (s): cpu = 00:00:00 ; elapsed = 00:00:00.001 . Memory (MB): peak = 1943.516 ; gain = 0.000

Starting Netlist Obfuscation Task
Netlist sorting complete. Time (s): cpu = 00:00:00 ; elapsed = 00:00:00.001 . Memory (MB): peak = 1943.516 ; gain = 0.000
Ending Netlist Obfuscation Task | Checksum: 1eabe6ccd

Time (s): cpu = 00:00:00 ; elapsed = 00:00:00.002 . Memory (MB): peak = 1943.516 ; gain = 0.000
INFO: [Common 17-83] Releasing license: Implementation
44 Infos, 4 Warnings, 0 Critical Warnings and 0 Errors encountered.
opt_design completed successfully
INFO: [Common 17-600] The following parameters have non-default value.
tcl.statsThreshold
INFO: [Vivado 12-248281 Executing command : report_drc -file uart_top_drc_opted.rpt -pb uart_top_drc_opted.pb -rpx uart_top_drc_opted.rpx
```

Implementation of Asynchronous FIFO for Clock Domain Crossing

Objective In this lab iteration, the primary architectural modification was the replacement of the single-clock FIFO_sync module with a robust **Asynchronous FIFO (FIFO_async)**. This change decouples the "Producer" (Sequence A) from the "Consumer" (Sequence B), enabling them to operate on independent clock frequencies in future high-speed UART applications.

Design Methodology The asynchronous FIFO was implemented using robust Clock Domain Crossing (CDC) techniques to prevent metastability and data corruption:

1. **Gray Code Pointer Arithmetic:** The read and write pointers are converted from Binary to Gray code before crossing clock domains. This ensures that only one bit changes per increment, minimizing sampling errors during asynchronous capture.
2. **Multi-Stage Synchronization:** Pointers are passed through 2-stage flip-flop synchronizers in the destination domain before being used for flag generation.
3. **Local Flag Generation:** Status flags (Empty, Full) are generated within their respective local clock domains by comparing the local pointer against the synchronized pointer from the opposite domain.

Integration The FIFO_async module was instantiated in the top-level design (uart_top.sv) in place of the synchronous version. While the current hardware setup utilizes a single global 100 MHz clock, the wr_clk and rd_clk ports were individually mapped to the system clock, rendering the design forward-compatible with split-clock architectures without requiring further logic changes.

Python Verification Tools User Guide

Overview The system verification is performed using a set of Python scripts located in the image_testing/scripts directory. These scripts facilitate image transmission, image reception, and register file (RGF) read/write validation via the UART interface.

Prerequisites

- **Python:** Ensure Python is installed on the host PC.
- **Libraries:** Install the required serial communication library using the command: pip install pyserial
- **Hardware:** Connect the FPGA to the PC via USB.

1. Image Transmission (Burst Write)

- **Script Name:** send_burst_image.py
- **Description:** This script generates a 256x256 test image and transmits it to the FPGA using the high-speed Burst Write protocol. It utilizes the MSG_START_BURST_WR command followed by a stream of MSG_BURST_PIXEL_WR messages.
- **Execution Command:** python send_burst_image.py
- **Expected Output:** The console displays the transfer progress (percentage and pixels/sec). Upon completion, it confirms that all burst messages (typically 16,384 messages for a 256x256 image) have been sent.

2. Image Reception (Burst Read)

- **Script Name:** receive_burst_image.py
- **Description:** This script initiates a read request by sending MSG_START_BURST_RD to the FPGA. It captures the incoming pixel stream, checks for synchronization errors, and reconstructs the image.
- **Execution Command:** python receive_burst_image.py
- **Output Location:** The received image is saved as a .ppm file in the ../img_output/ directory.
- **Status Report:** The script provides a summary of total pixels received, the raw data rate, and any sync errors detected during the transfer.

3. Register Read/Write Test

- **Script Name:** test_rgf_readwrite.py
- **Description:** This script verifies the reliability of the RGF Manager. It performs a sequence of Write operations to various registers (e.g., LED Control, PWM Configuration) followed by Read operations to validate that the data was stored and retrieved correctly.
- **Execution Command:** python test_rgf_readwrite.py
- **Expected Output:** A detailed list of test cases is displayed, marking each register address with PASS or FAIL based on the data verification.

Configuration Note All scripts are pre-configured to use **COM5** at a Baud Rate of **5,500,000**. If your PC assigns a different COM port number:

1. Open the relevant .py file in a text editor.
2. Locate the line PORT = 'COM5'.
3. Update 'COM5' to your actual port (e.g., 'COM3').